Remediating Logical Attack Paths Using Information System Simulated Topologies

François-Xavier Aguessy, Lucie Gaspard, Olivier Bettan, Vania Conan

Remediating Logical Attack Paths Using Information System Simulated Topologies

Category: Specialized

François-Xavier Aguessy[1,2], Lucie Gaspard[1], Olivier Bettan[1], and Vania Conan[1]
francois-xavier.aguessy@thalesgroup.com

[1] Thales Group, 4 avenue des Louvresses, 92622 Gennevilliers, France
[2] Telecom SudParis, 9 rue Charles Fourier, 91011 Evry, France

Abstract. With the increase of attacks and Information Systems getting ever more complex, security operators need tools to help them protecting critical assets. An attack graph is a model to assess the level of security of an Information System, but it can be used to compute actions that mitigate the modeled threats. In this paper we present a method to remediate the most relevant attack paths extracted from a logical attack graph. In order to help an operator to choose between several remediation candidates, we rank them according to a cost of remediation combining operational and impact costs. We implement this method using MulVAL attack graphs and several publicly available sets of data.

Keywords: logical attack paths, remediation candidates, MulVAL attack graph, simulated topology, remediation costs, remediation database.

1 Introduction

Due to the increase in the number and complexity of attacks, any Information System (IS) is vulnerable. Accurately assessing the risk is necessary but can be difficult. An attack graph is a risk analysis model regrouping all the paths an attacker may follow. It is composed of nodes, representing the actions possible for an attacker. Nodes are linked together with edges, representing dependencies between these actions. The main attack paths can be extracted from this graph in accordance to the IS priorities. In this paper we do not take into account the probability of occurrence of the attack path nor the damages it can do on the IS, but rather focus on the computation of remediations to an attack path.

A remediation aims at protecting the IS against an attack path by preventing its fulfillment. The nodes of an attack path not having any incoming edge are the starting points of attacks and are called *preconditions*. It is possible to offer remediation actions to apply on preconditions to mitigate the vulnerabilities and secure the targeted assets. In this paper, the definition of remediation encompass a patch, a firewall rule or an Intrusion Prevention System rule. Other more complex remediations such as access control or security policy management will

not be investigated here. The computation of filtering remediations has required the use of an accurate simulation of the IS network topology. To be usable in an operational environment, remediation candidates have to be ranked according to their operational and impact costs depending on the monitored IS. The assessment of the impact cost requires both the simulation of network flows and a functional model describing the normal behavior of the IS, in order to determine which nominal services may be disturbed by the deployment of a remediation.

The main contributions of this paper are (1) the design of a remediation process correcting the relevant attack paths rather than the whole attack graph generally too complex, (2) the method to remediate these paths based on the correction of attack preconditions, ranking remediation candidates according to a cost function considering both operational and impact costs, (3) the build of a generic remediation database assigning vulnerabilities with their remediations.

This paper is organized as follows: in Section 2, we briefly describe the state of the art. Section 3 formally defines the attack path and its main components. Section 4 explains how to compute remediations for an attack path. Section 5 describes the ranking of remediation candidates according to their cost. Section 6 details an implementation of this method with the MulVAL attack graph engine we use for the experiments of Section 7. Finally, in Section 8 we compare our model with the related work, before concluding on our work.

2 State of the art

2.1 Topological and functional models of the Information System

An efficient security analysis needs an accurate topological model of the system studied. Recent developments have shown that models of an IS can be created automatically from network scans [16] [10]. It can also be created by importing the configuration of network devices as it is done in some commercial solutions [28]. The model should be as accurate as possible to be exploitable. Nevertheless, its completeness is not guaranteed by the available technology.

The functional model of an IS is complementary to the topological one. It contains the dependencies between components of the IS and can be used to improve the automation of the deployment of remediations. In [33], Toth and Kruegel present a dependency model that allow to determine the impact of remediations on the whole system. In [14], Kheir et al. propose a framework modeling dependencies which handles both confidentiality, integrity and availability.

2.2 Attack graphs and attack paths

Attack graphs are a model regrouping all the steps an attacker may follow in an IS during an attack. It has been first introduced by Phillips and Swiler in [25]. It has been widely used ever since, thus many heterogeneous models are now behind the name *attack graph*. Generally, vertices (also called nodes in the literature) represent opportunities in an IS or actions that can be done by an attacker,

and edges (also called arcs in the literature) represent the dependency relations between the opportunities and/or actions. An attack graph can be built using information about the potential exploits that can be carried out on a network and using existing vulnerabilities databases [19] [20]. A summary of the state of the art on the early papers about attack graphs (from 2002 to 2005) has been done by Lippmann and Ingols in [17]; a more recent by Kordy et al. in [15].

MulVAL, The Multi-host, Multi-stage Vulnerability Analysis Language tool is an open-source attack graph engine created by Ou et al. [24]. It uses Datalog, a logic programming language, in order to generate an attack graph in which nodes are related to each other with logical relations (OR or AND). The other two main attack graph engines are commercial products: Cauldron [11] (originally presented by Jajodia et al. in [12]) and Artz's NetSPA [3].

An attack graph contains targets and multiple paths to reach them. Such attack paths can be extracted from the graph using different algorithms, according to the needs. Swiler et al. describe in [31] shortest paths algorithms used to find the most likely or lowest cost attack paths. In [26], Sawilla and Ou present a generalization of Google's Page-Rank algorithm to identify the most critical attack nodes. An attack path extraction and scoring function has also been presented by Bettan et al. in PoSecCo, a FP7 European Research project [4].

2.3 Selecting remediations

Remediations to an attack can be regrouped in three types: corrective, active and passive. The first one is the correction of the exploitable vulnerability. This is generally implemented by patch management software. The technology is quite mature (several tools exist, vendors regularly propose patches for their software) but it suffers from limitations detailed by Cavusoglu et al. in [5]. The most important is that patch deployment still requires human intervention: each patch must be tested on all platforms to prevent conflicts or regressions before being applied. The active remediations regroup those that prevent the exploitation of a vulnerability that still exists after the deployment. This is for example the case of simple filtering by a firewall or an Intrusion Prevention System (IPS). An IPS blocks flows that has been flagged as abnormal thanks to a signature or due to its statistical behavior [30]. More generally, an Intrusion Response System (IRS) is a system that provides other types of responses to a detection. This is a currently active research topic and many papers treat this subject, as summarized by Shameli-Sendi et al. in [27]. Finally, the passive remediations regroup the detection of the exploitation of a vulnerability and its report. This is a last resort but is widely used, as it can help security operators to know what happened in their IS. A system that only provides passive responses (alerts, reports, logs...) is generally called Intrusion Detection System (IDS) [34].

Some papers propose techniques to compute or select remediations using attack graphs. In [6], Cuppens et al. describe how the LAMBDA language, that has been used to model attacks, can also be used to model counter-measures and using the concept of anti-correlation allow to compute the appropriate counter-measures for an attack. In [35], Wang et al. base their analysis on the initial

conditions of an attack graph to compute all hardening options for a network. This approach has been improved by Albanese et al. in [2] with a near-optimal algorithm more efficient and cost-sensitive. In [23], Noel and Jajodia describe several methods to prioritize the deployment of patches depending on an attack graph. In [9], Ingols et al. explain how they represent three types of counter-measures (Firewall rules, IPSs and proxy firewall) in the NetSPA attack graph. Attack graphs are also used by Noel and Jajodia [22] to compute the optimal locations to deploy IDSs in an IS: they allow to minimize the cost of sensors while keeping a complete coverage of potential attack paths.

2.4 Ranking remediations

There are sometimes several remediations that could be deployed for a detection. Thus, they should be ranked to select the most appropriate one. Generally, this evaluation contains an estimation of the impact of a remediation and thus rely on a functional model as presented above. The ranking method presented by Toth and Kruegel in [33] first uses an algorithm that evaluates the impact of the remediations. Then, it advises to select the remediation with minimal negative effect on legitimate users. In [14], Kheir et al. present how to compute an index called RORI which represents a return on investment of a remediation. It is based on a dependency graph where are propagated the levels of Confidentiality, Integrity and Availability of assets. This index can then be used to rank remediations. An other parameter that can be taken into account when selecting a remediation is the impact of such counter-measure against the success likelihood of the attack. This kind of approach has been presented by Kanoun et al. in [13], where they implement a model based on dynamic Markov Models to assess the success likelihood of attacks. This model allow to select the most appropriate counter-measures to prevent an attacker from reaching its objectives.

Selecting a remediation among many brings challenges to overcome because it needs the knowledge of many parameters (costs parameters, functional and topological models) and how to combine them. It also requires assumptions regarding the coverage of the remediations on the attack.

3 Attack paths and preconditions

3.1 Attack path representation

Definition 1. *A **logical attack graph** G is a directed AND-OR graph represented by $G(V, A)$ where:*

- *V is a set of vertices that describe logical facts. Each vertex could be an* AND *(respectively an* OR*) vertex, meaning that this vertex needs the* conjunction *(resp. the* disjunction*) of its incoming arcs to be true.*
- *A is a set of directed arcs that represent a logical dependency from the child vertex to the parent one.*

In an attack graph as defined above, it is possible to choose attack targets. These are the vertices describing important final steps for an attacker. Based on these targets can be built attack paths using a bottom-up approach from the target to the upper preconditions of the attack graph. They can be ranked according to their impact and probability of occurrence, but this is a full-fledged subject that has been, for example, described in [4] or [26].

Definition 2. *An **attack path** is an acyclic and* logically valid *subgraph of an attack graph with one* target *and several* preconditions*.*

Definition 3. *The **target** of an attack path is the vertex whose outdegree is 0,* $\deg^+(v) = 0$ *(no outgoing arcs).*

Definition 4. *A **precondition** in an attack path is a vertex whose indegree is 0,* $\deg^-(v) = 0$ *(no incoming arcs).*

Definition 5. *A subgraph S of an attack graph G is **logically valid** if S contains at least one vertex and for each vertex $v \in S$, $v \in G$ and if* $\deg^-_G(v) > 0$ *(more than one incoming arc in G):*

- *if v is an AND, all the parents and incoming arcs of v in G are in S,*
- *if v is an OR, at least one parent of v and its incoming arc in G is in S.*

An attack path may have several intermediate goals but has only one main goal: the target of the attack path. It contains one, several or all the possible paths in the attack graph allowing to compromise this target.

3.2 Remediations can be applied only on preconditions

Proposing remediations to an attack path is searching the means to prevent the attacks and protect its target. An attack path is a logical graph: a fact is true if and only if the conjunction or disjunction of its parents is also true. As a precondition does not have any parent, it is not deducted from any other vertex. So, they are the first conditions from which all other vertices are deducted and thus the only vertices where can be applied remediations. This is the basic assumption on which we will build our remediation method.

Sufficient preconditions The attack path contains one target that should be protected by the remediations. As the attack path is an AND-OR graph, it is possible to compute all conjunctions of to-be-remediated preconditions, sufficient to protect the target. This logical expression SP can be represented with a set of disjunctions containing conjunctions as following:

$$SP = \bigvee_i SP_i = \bigvee_i \bigwedge_j SP_{i,j} \tag{1}$$

where $\bigvee$ is logical OR, $\bigwedge$ is logical AND, SP_i is a conjunction of preconditions sufficient to protect the target (i indexing the conjunction of preconditions) and $SP_{i,j}$ is a precondition to remediate (j indexing the preconditions).

Fig. 1. Recursive algorithm computing the conjunctions of sufficient preconditions

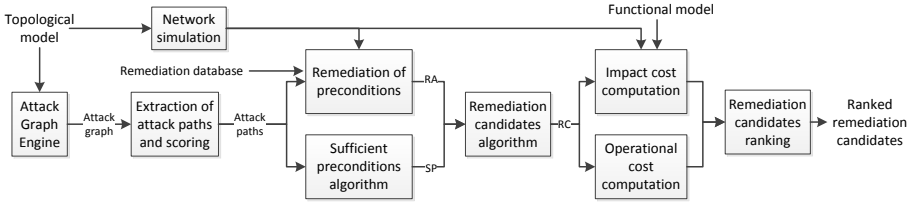
```
1: function computeSP(Vertex v)        ▷ Returns the list SP to delete the vertex v
2:     if v is a precondition then                                   ▷ v has no parent
3:         return [[v]]                                  ▷ it is the only precondition
4:     else if v is an AND then
5:         res ←[[]]
6:         for each parent p of v do
7:             res ← res;computeSP(p)
8:         end for
9:         return res                                  ▷ ⋁_{p∈{parents of v}} computeSP(p)
10:    else if v is an OR then
11:        res ←computeSP(first parent of v)
12:        for each other parents p of v do
13:            res ←conjunctionOfSets(res,computeSP(p))
14:        end for
15:        return res                                  ▷ ⋀_{p∈{parents of v}} computeSP(p)
16:    end if
17: end function
18: function conjunctionOfSets(A, B)   ▷ Makes the conjunction of sets A and B
19:    result ← [[]]          ▷ A and B are Or/And sets: A = ⋁_i A_i, A_i = ⋀_k A_{i,k}
20:    for i = 1 to size(B) do
21:        buildingResult ← A
22:        for j = 1 to size(buildingResult) do
23:            buildingResult[j] ← buildingResult[j]; B[i]
24:        end for
25:        result ← result; buildingResult
26:    end for
27:    return result                          ▷ (⋁_i A_i) ∧ (⋁_j B_j) = ⋁_{i,j}(A_i ∧ B_j)
28: end function
```

In fact, computing SP is identical to find all the conjunctions of preconditions sufficient to delete the target vertex according to the AND/OR formalism.

Definition 6. *A conjunction of precondition SP_i is **sufficient** to delete the target t of an attack path AP if the deletion of each precondition $SP_{i,j} \in SP_i$ and its propagation in AP implies that henceforth $t \notin AP$.*

The recursive algorithm that computes SP can be found in Figure 1. It should be called on the target of the attack path and will go up recursively along the arcs. All conjunctions of preconditions computed with this algorithm allow to prevent the access to the target, if remediated. Of course all preconditions can not be remediated. If this is the case in a conjunction, the entire set of preconditions will not be usable to successfully protect the target of the attack path.

Fig. 2. Remediation process based on attack paths



4 Remediation of an attack path

A diagram summarizing the remediation process can be found in Figure 2.

4.1 Remediate a precondition

A precondition contains a logical fact describing what can be used by an attacker. We detail real preconditions and their remediations in Subsection 6.3, but will first describe two general methodologies that can be applied to preconditions.

Simple remediations to preconditions The first case appearing when remediating a precondition is a simple remediation that can be applied to negate this precondition. This usually corresponds to the first type of remediation presented in the state of the art. You need a database that makes the link between the precondition (eg: the vulnerability) and how you can remediate it (eg: a patch).

Remediations using the network topology Some remediations require more advanced techniques. This is the case of those which try to prevent the *exploitation* of a vulnerability. It corresponds to the second type of remediation of the state of the art. Computing such remediations requires an accurate knowledge of the flows exchanged on the network and thus need to simulate the network topology.

4.2 Remediation candidates for an attack path

Each remediation potentially contains several elementary actions. More formally, for each attack path, we have succeeded to compute:

- A disjunction of conjunctions of **sufficient preconditions** to remediate, in order to protect the target of the attack path: SP
- A disjunction of conjunctions of **remediation actions** sufficient to prevent a precondition p:

$$RA(p) = \bigvee_i RA_i(p) = \bigvee_i \bigwedge_j RA_{i,j}(p) \tag{2}$$

  where $RA_i(p)$ is a conjunction of remediation actions allowing to prevent the precondition p (i indexing each conjunction) and $RA_{i,j}(p)$ is a remediation action (j indexing each action of the conjunction) each remediation action $RA_{i,j}$ is constituted of the tuple (action to apply, machine to deploy it).

Fig. 3. Algorithm computing the remediation candidates

```
1: function computeCandidates(SP, RA)   ▷ Returns all remediation candidates protecting an attack path.
2:     res ←[[]]
3:     for SP_i in SP do
4:         res ← res ; computeRemedsToPreconds(SP_i, 1, RA)
5:     end for
6:     return res
7: end function
8: function computeRemedsToPreconds(SP_i, j, RA)   ▷ Returns all conjunctions of actions allowing to remediate the preconditions of SP_i starting from j.
9:     SP_{i,j} ← SP_i[j]                          ▷ j^th precondition to remediate
10:    RA_j ← RA[SP_{i,j}]                         ▷ Remediations of j^th precond
11:    if empty(RA_j) then                         ▷ j^th precondition can not be remediated
12:        return [[]]
13:    else
14:        if j=size(SP_i) then                    ▷ Terminaison of recursion
15:            return RA_j
16:        else
17:            RA_{j+1..n} ←computeRemedsToPreconds(SP_i, j + 1, RA)
18:            return conjunctionOfSets(RA_j, RA_{j+1..n})
19:        end if
20:    end if
21: end function
```

We need to combine SP and RA to compute a disjunction of **remediation candidates** containing the actions that allow to protect the target:

$$RC = \bigvee_i RC_i = \bigvee_i \bigwedge_j RC_{i,j} \tag{3}$$

where RC_i is a remediation candidate (i indexing the number of candidates) and $RC_{i,j}$ is a remediation action (j indexing the number of actions in the candidate).

An algorithm computing such remediation candidates is shown in Figure 3.

5 Costs of remediations

The last essential point for our remediation method is to estimate the cost of a candidate. This will help an operator to choose between several candidates remediating the same attack path. We have identified two principal sources of cost that must be considered: the operational and the impact costs.

5.1 Operational cost

The first important cost for an operator deploying a remediation is the operational cost (OC). It represents the difficulty to implement the remediation on

the assets and to maintain it. For each remediation action $RC_{i,j}$ that should be applied, we identified four categories in which this cost can be split.

1. **Remediation cost (RC)**: This is the cost of the input necessary to apply the remediation, for example, the price of a patch or of a signature.
2. **Deployment costs (DC)**: This is the cost representing the workload to apply the remediation on the concerned machine.
3. **Test costs (TC)**: This is the cost to test that all important features of the Information System are still working as expected, after the deployment.
4. **Maintenance costs (MC)**: This is the cost per year of the maintenance induced by the remediation. It reflects for example the increase of CPU, memory, storage and treatment of logs that will be induced.

These elements can be expressed in a monetary unit and we detail in Subsection 6.5 how to compute them. The operational cost of a remediation action is the sum of all these elements as shown in Equation 4.

$$OC(RC_{i,j}) = RC + DC + TC + MC \tag{4}$$

To simplify the estimation of the operational costs of a candidate containing several remediation actions, we made the assumption that these actions are independent. This assumption has been introduced and justified by Gonzalez-Granadillo et al. in [8] as Axiom 1. This is moreover the most common case, as the remediation actions are usually deployed on different machines or are of different types. With this assumption, the operational cost of a candidate is the sum of the operational costs of its actions, as shown in Equation 5.

$$OC(RC_i) = \sum_j OC(RC_{i,j}) \tag{5}$$

5.2 Impact cost

We propose here a basic impact cost (IC) function that measures the loss due to the unavailability of services after the deployment of a remediation. This function uses a list of (1) dependencies of business applications toward services, (2) services toward network accesses and (3) interdependencies between services. In addition to this are added cost values related to the temporary and permanent unavailability (UC) of business applications.

Thanks to those parameters, we can compute the cost of unavailability of all business applications before (on the real system) and after (on a simulated system) deploying a candidate RC_i, by checking recursively that the services dependencies are verified as shown in Equation 6.

$$IC(RC_i) = \sum_{ba \in businessApp} isImpactedBy(RC_i, ba) * UC(ba) \tag{6}$$

The impact cost is certainly the most important part to take into account when deploying a remediation but it is perhaps also the most difficult to quantify,

as it is hard to estimate the cost if a business application is unavailable and to know which applications will be disrupted by a remediation candidate. It is therefore very important to provide security operators with indications about such a cost, to help them choose at best the remediation to deploy.

5.3 Ranking remediation candidates

These costs allow us to attach a global cost C to a candidate as shown below:

$$C(RC_i) = OC(RC_i) + IC(RC_i) \tag{7}$$

The candidate cost function can be considered as a ranking function taking as input an unsorted set of remediation candidates and that outputs a set of the same candidates sorted according to their cost. This allow a security operator to select one of the candidates that has the lowest cost.

As the remediation candidates cost function is only used to compare candidates with each other, even if the cost parameters are not assigned exactly, it does not change significantly the order between them. Thus, the details of the cost models parameters are not required, but only need to represent a tendency, in order to conserve the ranking between candidates. This assumption has also already been justified by Gonzalez-Granadillo et al. in [8].

6 Implementation

6.1 Simulation of the network topology

As we need a simulated network topology representing the target IS, we created a network simulator that accurately reproduces simple network behaviors of hosts: we are able to simulate exchanges between hosts, calculate routes, test if a packet can pass firewall rules, etc. We designed a pivot file in which we put the topological information needed by the simulator. We also created connectors to automatically build such a file. The first connector we built was a python server that gathered the topological information collected by agents deployed on Linux machines into the pivot format. The second connector was built for the European Research Project PoSecCo [1], where we had an ontology containing the network topology. We thus implemented a connector that was querying in the ontology for the information needed.

6.2 Generation of attack paths

We use the open source attack graph engine MulVAL [24]. It requires three types of inputs: topological, filtering and vulnerabilities information. We combine our simulated topology with a vulnerability scan (of Nessus [32]) by merging the information about services and their vulnerabilities extracted from the scanner report into our topology. MulVAL inputs are stored in a file using the Datalog language. It outputs an XML file containing the logical attack graph computed thanks to its engine from which the attack paths are extracted.

Table 1. Main MulVAL preconditions and their remediations

Preconditions	Description	Possible remediations
hacl(src, dst, port, portocol)	The host *src* has access to *dst* on *port* using *protocol*	Deploy a firewall rule
vulExists(host, vulID, program)	*program* on *host* has a vulnerability *vulID*	Apply a patch or deploy a Snort rule
networkServiceInfo(host, program, protocol, port, user)	*program* on *host* launched as *user* open *port* using *protocol*	Stop this network service
hasAccount(user, host, account)	*user* has *account* on *host*	Disable this account

6.3 Preconditions in MulVAL and their remediations

The main preconditions proposed by MulVAL to model attacks and their remediations can be seen in Table 1. We will focus here only on three relevant types of remediation for an enterprise: applying a patch, deploying a rule on an IPS (preventing *vulExists()*) and deploying a firewall rule (preventing *hacl()*).

Application of a patch In order to propose the right patch to a vulnerability, we use the parameter in the fact of the precondition *vulExists* containing the identifier of a vulnerability, generally a CVE (Common Vulnerabilities and Exposures) [20]. We use this identifier to look for known patches in the remediation database we describe in Subsection 6.4.

Deployment of a firewall rule To compute the firewall rule that should be deployed, we use all the parameters of the fact *hacl(src, dst, port, protocol)*. This precondition explains the network access the attacker needs for his attack. So, it should be negated by the rule to deploy, which should have the following form:

DROP FROM src TO dst:port USING protocol

It can be generated according to the type of firewall aimed. For example, we propose an automatic generation of iptables [18] firewall rules.

The last problem we need to deal with for the firewall rules proposal is where it should be deployed. We use here the topology simulation presented in Section 4.1 to determine the route followed by packets between *src* and *dest:port*. We then deduce on which machine the firewall rule can be deployed.

Deployment of an IPS rule The last type or remediation we will detail is the deployment of IPS rules for Snort [29] which prevent the exploitation of a vulnerability. For each *vulExists* related to an *hacl*, we can know (1) The Snort rules that may exist to prevent the exploitation of the vulnerability by searching its identifier in the remediation database presented in Subsection 6.4, and (2) the network routes that may be used by the attacker to exploit this vulnerability, by using the simulation of the network and a deduction process similar to the calculation of the firewall rules. On each route, we must have an IPS host where we can deploy the rule. Otherwise, the remediation is not possible. The rules we propose here must be used with Snort in inline mode and they begin with the *drop* keyword, meaning that we use it as an IPS.

6.4 Filling the remediation database

One challenge of the proposition of remediation is the ability to build automatically a remediation database. We will describe here how we overcome it.

Database model We use a relational model stored in a SQLite file. We choose to use a model similar to the one used in the National Vulnerability Database [21] to represent vulnerabilities. Then, we added two tables corresponding to the types of remediations. In order to have a N-to-N association with the vulnerabilities, we also add a join table for each kind of remediation.

Patches We used the NVD [21] to find the links toward patches that correct vulnerabilities. Among the attributes related to a CVE, a *reference* can point to a website describing how to patch the vulnerability. So, we parse the dumps of the NVD, extract the links toward patches and store it in the database. Around 20% of the CVE have a "PATCH" reference attached.

Snort rules In the standard format of a Snort rule, there is an option "reference" which often contains a CVE. In the freely available database of rules provided by Sourcefire [29], nearly 50% of the rules are related to a CVE.

6.5 Providing the costs parameters

Operational cost Operational costs depends highly on the company and on the remediation. So, we choose in our prototype to assign parameters per types of remediation. Generally the difference of operational cost between remediations of the same type is low, but it may be also possible to add the cost parameters into the remediation database, in order to be able to attach to each remediation specific operational cost parameters.

Impact cost The description of dependencies used for impact cost is also totally dependent on the IS and has to be provided by the security operator. To describe these dependencies, we use an XML file in which the dependencies relations are described according to a dependency graph.

7 Experiments and results

In order to validate the whole remediation method described in this paper, we applied it on several test topologies. We implemented it on the use-cases of the European Research Project PoSeCo [4]. But before detailing this test-bed and our experiments on it, we will present a simpler scenario implementing the main concepts. We will end with a discussion about the complexity of our approach.

7.1 Simple experiment scenario

Network topology and attack scenario The first scenario we implement rely on a topology that we deployed on virtual machines. It contains 5 Linux-based hosts: a web server, a database server, an administration machine inside

a LAN, a firewall that protects the LAN and the servers from the Internet, and the attacker's machine that is on the Internet. We configure the firewall in such a way that the web server is the only service that is accessible from the Internet and the LAN. The web server needs the database server to work properly and has a full access to it. The enterprise has two business applications using the IT: an Extranet which is rarely used and an Intranet which is used for all employees and is thus much more critical. The database server contains also some confidential information that the company wants to protect.

When the web server is exploited, the attacker can access the database server and with an other exploit can try to gain access to all the data it contains. This is the attack path that will be described in the rest of this scenario.

Generation of the attack path and proposition of remediations To collect the topological information, we use the python agents described in Subsection 6.1. We generate MulVAL inputs and launch the attack graph engine, then extract the attack paths and select the one presented above. It chains the exploitation of two vulnerabilities: the first one CVE-2004-1315 is on the web server, the second one, CVE-2012-3951, is on the database server.

We use our prototype to visualize the attack path to correct and the remediation candidates. We present here the four most relevant candidates, ranked by cost. We also explain for each candidate why its cost is low, medium or high.

1. The first candidate is the deployment on the firewall of the Snort rule sid:12610 that allows to block the exploitation of the first vulnerability. This remediation has a lot of advantages, because it doesn't interrupt any genuine service, is not too much expensive (deployment can be nearly fully automated), and blocks successfully the attack. This candidate has no impact cost, a low operational cost and thus a very low global cost, that is why this is the first one to be proposed.
2. The second one is the proposal of a patch to the first vulnerability. As the first one, it doesn't have any impact on normal service, but has much more operational cost, because deploying a patch need human intervention. So, this candidate has a low global cost and thus is the second one to be proposed.
3. The third one is a firewall rule that blocks all the traffic from the Internet to the web server on http port. It has a low operational cost, because it can be automatized, but has a medium impact because it cuts the access from the Internet to the web server, even if it keeps all the accesses from the LAN. So this candidate has a medium global cost and thus is the third to be proposed.
4. The last one is a firewall rule that blocks all the traffic to the web server on http port. It has also a low operational cost, but has a huge impact because it cuts also all the accesses for the employees of the LAN to the web server. So this candidate has a high global cost and thus is the last one to be proposed.

7.2 Results on PoSecCo's testbed

We will now present the results of this method applied on the testbed of the FP7 European Research Project PoSecCo [1].

The main use case in which the PoSecCo prototypes have been tested has two main business services: a broadcaster Internet distribution and a corporate streaming service. These services have several security requirements and run on a testbed that has been deployed during the project, on which prototypes have been tested. It contains around twenty machines (some are representing server farms) and eight routers. All the topological information needed for our prototype are collected from an ontology and the attack paths extracted are ranked according to their impact on security requirements.

On the twenty machines and eight routers, there are more than a thousand vulnerabilities in total. It was chosen for this project that there will be one attack path per target, gathering the relevant ways to compromise it. Thus, after establishing a list of five hosts to protect in priority, the operator has five attack paths to assess and correct. These attack paths contain between thirty and hundreds of nodes, the possible remediations are computed in a few seconds. Due to project limitations, only two types of remediations are proposed: patches and firewall rules. For each attack path, many candidates are proposed (up to a hundred), and are ranked according to their operational and impact cost. The first candidates (lower cost) offer the best compromise between efficiency and cost and should be the best option for a security operator.

During the project, the prototype implementing this approach was presented to end-users that compared their risk analysis and its remediation, in anticipation of a change in the testbed topology, with and without the prototype. The end-users, independent of the project, concluded that the scenario using this methodology was much more efficient: it reduces the analysis from four hours to twenty two minutes and could reduce the number of people needed for this task from between three and six to only one. The result of this evaluation can be found in PoSecCo's Deliverable 1.7, in scenario SP06 [7].

7.3 Complexity

What must be well understood before talking about the complexity of our algorithms is that in this paper, we propose remediations to attack *paths* and not to a whole attack *graph*, we thus have smaller complexity issues. Indeed, an attack graph is usually a large graph whereas, an attack path is smaller, because it focuses only on the very impactive or the most likely ways to access a target.

The complexity of the algorithm computing the candidates is not very impactive, because 1) it is linear in the number of conjunctions of precondition and 2) the number of remediations for one precondition is generally low. The algorithm computing SP depends highly on the structure of the input attack path, especially on the number of parents of each vertex. In the best case (each vertex has only one parent) this algorithm is linear in the number of vertices. In the worst case (each vertex has several parents), the complexity is exponential in the number of parents of OR vertices. This is the factor that most influence the complexity. We made simulations on non-realistic graphs with different varying parameters (number of parents, OR nodes, AND nodes, preconditions...) to validate these results. Nevertheless, in practice on several real use cases, we found

that the number of parents for OR vertices in attack paths is generally low: an average of 1.7 per OR vertex in attack paths produced by MulVAL. This can be simply explained knowing that, in an attack path, as explained above, we only have few different possibilities to compromise a target, alternatives creating disjunctions in the attack path. It implies that this methodology generally scales well, if the attack paths treated are properly generated.

8 Related Work

The papers which describe the closest approach to our work are [35] and [2]. In [35], Wang et al. base their analysis on the preconditions of an attack graph to compute ways to prevent attacks. But even when they evocate the cost to choose one remediation solution rather than another, they do not present a cost function to sort candidates as we did in this paper. In [2], Albanese et al. extend [35] mainly by adding a cost model, similar to the one we present here, and by improving the complexity of the algorithm to compute candidates. But what distinguish our approach from both these ones is that we do not compute remediations to an attack *graph* but to attack *paths*, meaning that our algorithms are working with smaller inputs. We are convinced that it is much more sound and efficient to correct only the paths that are significant rather than reasoning on the global attack graph. This was assessed on realistic use-cases.

What is also original in our approach is that our remediation computation is generic. In [23] and [22], Noel and Jajodia propose various types of remediations to predefined types of attacks modeled by attack graphs. The method we present here is more generic. Indeed, the expressiveness of logical attack graphs allows the modeling of every attack described with AND/OR conditions and our method applies to all of them, without the limitations identified in the related work. Dealing with new attacks only imply to define remediation for potential new kinds of preconditions. These remediations can be simple or may require network topology simulation, as the ones presented in this paper.

Furthermore, several databases that contain remediations exist. However, each database is dedicated to a type of remediation. For example, the NVD contains information about patches [21] and Snort databases contain only Snort rules [29]. In this paper, we design a remediation database and fill it using several online available sets of data. This database contains different kinds of remediations and can be extended to provide new types of remediations.

9 Conclusion

We present in this paper a method describing how to compute remediations for scored attack paths extracted from an attack graph. Attack graphs have been widely used for assessing the security level of an Information System, we chose instead to use them in order to propose solutions to enhance this security level, by computing remediations preventing attack paths in an Information System. Using scored attack paths extracted from an attack graph allows us to remediate

only the very likely or impacting paths that lead to main assets which is much more efficient than remediating the global attack graph.

We have stated that the only vertices on which we compute remediations, within a logical attack path, are the preconditions. We have implemented algorithms to cluster these nodes into conjunctions of sufficient preconditions to be remediated, in order to protect the target of an attack path. Then, after explaining how to compute remediation actions to prevent a precondition, we detailed their combination with the sufficient conjunctions of preconditions to determine the candidates. As the operator has to choose one remediation among several candidates providing the same remediation objective, we assign to each remediation a global cost combining operational and impact costs. To calculate topological remediations to certain preconditions and to assess the effects of remediations on the system, we have designed a simulated network topology.

Limitations of the logical model used for modeling attack graphs were detailed during this study since this model is deterministic and not dynamic. Thus, it has to be extended into a quantitative model to represent dynamic attacks and to model them more accurately. However, this logical model has the advantage to be efficiently generated and processed and is well suited to model potential attacks. Future work will study the computation of more complex remediations with the development of a more accurate cost function. A prerequisite will be a better knowledge of the IS through new mining techniques.

References

1. Posecco, `http://www.posecco.eu`
2. Albanese, M., Jajodia, S., Noel, S.: Time-efficient and cost-effective network hardening using attack graphs. In: Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on. pp. 1–12. IEEE (2012)
3. Artz, M.L.: NetSPA: A Network Security Planning Architecture. Ph.D. thesis, Massachusetts Institute of Technology (2002)
4. Bettan, O., Ponta, S., Musaraj, K., Casalino, M.: D4.8 - prototype: Standardized audit interface. Tech. rep., PoSecCo European Project from the 7th Framework (project no. 257129) (2012)
5. Cavusoglu, H., Cavusoglu, H., Zhang, J.: Security patch management: Share the burden or share the damage? Management Science 54(4), 657–670 (Apr 2008)
6. Cuppens, F., Autrel, F., Bouzida, Y., Garcia, J., Gombault, S., Sans, T.: Anti-correlation as a criterion to select appropriate counter-measures in an intrusion detection framework. In: Annales des télécommunications. vol. 61, pp. 197–217. Springer (2006)
7. Demetz, L., Maier, R., Manhart, M., Plate, H., Fitz, M.: D1.7 - final project evaluation. Tech. rep., PoSecCo European Project from the 7th Framework (project no. 257129) (2013)
8. Granadillo, G.G., Jacob, G., Debar, H., Coppolino, L.: Combination approach to select optimal countermeasures based on the rori index. In: Second International Conference on Innovative Computing Technology. pp. 38–45. IEEE (2012)
9. Ingols, K., Chu, M., Lippmann, R., Webster, S., Boyer, S.: Modeling modern network attacks and countermeasures using attack graphs. In: Annual Computer Security Applications Conference. pp. 117–126. IEEE (2009)

10. Jajodia, S., Noel, S.: Advanced cyber attack modeling analysis and visualization. Tech. rep., DTIC Document (2010)
11. Jajodia, S., Noel, S., Kalapa, P., Albanese, M., Williams, J.: Cauldron mission-centric cyber situational awareness with defense in depth. In: Military Communications Conference. pp. 1339–1344 (2011)
12. Jajodia, S., Noel, S., O'Berry, B.: Topological analysis of network attack vulnerability. Managing Cyber Threats pp. 247–266 (2005)
13. Kanoun, W., Dubus, S., Papillon, S., Cuppens-Boulahia, N., Cuppens, F.: Towards dynamic risk management: Success likelihood of ongoing attacks. Bell Labs Technical Journal 17(3), 61–78 (2012)
14. Kheir, N., Debar, H., Cuppens-Boulahia, N., Cuppens, F., Viinikka, J.: Cost evaluation for intrusion response using dependency graphs. In: Network and Service Security. pp. 1–6. IEEE (2009)
15. Kordy, B., Piètre-Cambacédès, L., Schweitzer, P.: Dag-based attack and defense modeling: Don't miss the forest for the attack trees. CoRR (Mar 2013)
16. Lagadec, P.: Visualisation et analyse de risque dynamique pour la cyber-défense. SSTIC (2010)
17. Lippmann, R.P., Ingols, K.W.: An annotated review of past papers on attack graphs. Tech. rep., DTIC Document (2005)
18. Netfilter: iptables, `http://www.netfilter.org/projects/iptables/index.html`
19. NIST: Capec, common attack pattern enumeration and classification, `http://capec.mitre.org/`
20. NIST: Cve, common vulnerabilities and exposures, `https://cve.mitre.org/`
21. NIST: Nvd, national vulnerability database, `https://nvd.nist.gov/`
22. Noel, S., Jajodia, S.: Optimal ids sensor placement and alert prioritization using attack graphs - springer. Journal of Network and Systems Management (2008)
23. Noel, S., Jajodia, S.: Proactive intrusion prevention and response via attack graphs. Tech. rep., Addison-Wesley Professional (2009)
24. Ou, X., Govindavajhala, S., Appel, A.W.: Mulval: A logic-based network security analyzer. In: Proceedings of the 14th conference on USENIX Security Symposium-Volume 14. pp. 8–8. USENIX Association (2005)
25. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: the 1998 workshop. pp. 71–79. ACM Press, New York, New York, USA (1998)
26. Sawilla, R.E., Ou, X.: Identifying critical attack assets in dependency attack graphs. Springer (2008)
27. Shameli-Sendi, A., Ezzati-Jivan, N., Jabbarifar, M.: Intrusion response systems: survey and taxonomy. SIGMOD (2012)
28. Skybox security, i.: Skybox, `http://www.skyboxsecurity.com/`
29. Sourcefire: Snort, `http://www.snort.org/`
30. Stakhanova, N., Basu, S., Wong, J.: A taxonomy of intrusion response systems. International Journal of Information and Computer Security 1(1), 169–184 (2007)
31. Swiler, L.P., Phillips, C., Ellis, D., Chakerian, S.: Computer-attack graph generation tool. In: DARPA Information Survivability Conference and Exposition. pp. 307–321. IEEE (2001)
32. Tenable: Nessus, `http://www.tenable.com/products/nessus`
33. Toth, T., Kruegel, C.: Evaluating the impact of automated intrusion response mechanisms. In: CSAC. pp. 301–310. IEEE (2002)
34. Tucker, C.J., Furnell, S.M., Ghita, BV, Brooke, P.J.: A new taxonomy for comparing intrusion detection systems. Internet Research 17(1), 88–98 (2007)
35. Wang, L., Noel, S., Jajodia, S.: Minimum-cost network hardening using attack graphs. Computer Communications 29(18), 3812–3824 (2006)