A Service Dependency Model for Cost-Sensitive Intrusion Response

Nizar Kheir[1,2], Nora Cuppens-Boulahia[1],
Frédéric Cuppens[1], and Hervé Debar[3]

[1] Télécom Bretagne, 2 rue de la Chataigneraie, 35512 Cesson Sévigné, France
{nora.cuppens,frederic.cuppens}@telecom-bretagne.eu
[2] France Télécom R&D, 42 Rue des Coutures, 14066 Caen, France
nizar.kheir@orange-ftgroup.com
[3] Télécom SudParis, 9 rue Charles Fourier, 91011 Evry, France
herve.debar@telecom-sudparis.eu

Abstract. Recent advances in intrusion detection and prevention have brought promising solutions to enhance IT security. Despite these efforts, the battle with cyber attackers has reached a deadlock. While attackers always try to unveil new vulnerabilities, security experts are bounded to keep their softwares compliant with the latest updates. Intrusion response systems are thus relegated to a second rank because no one trusts them to modify system configuration during runtime.

Current response cost evaluation techniques do not cover all impact aspects, favoring availability over confidentiality and integrity. They do not profit from the findings in intrusion prevention which led to powerful models including vulnerability graphs, exploit graphs, etc. This paper bridges the gap between these models and service dependency models that are used for response evaluation. It proposes a new service dependency representation that enables intrusion and response impact evaluation. The outcome is a service dependency model and a complete methodology to use this model in order to evaluate intrusion and response costs. The latter covers response collateral damages and positive response effects as they reduce intrusion costs.

1 Introduction

The dot-com bubble that occurred in the late nineteen nineties has changed the nature of market places, maybe for ever. These have become very dynamic, pushing IT industries to propose innovative software tools in order to attract new customers. IT industries started suffering increasing challenges, among which are shorter product life cycles, rapid outsourcing and price erosion caused by a fierce worldwide competition [19]. In order to withstand to these challenges, products' security had been relegated to a secondary priority that is handled as an add-on property [18]. Security flaws are more likely to be tolerated in newly released products. IT industries become inclined to reveal new updates and patches on a regular basis in order to strengthen the security of their products. The opposite coin facet for this reality is a vicious race between security experts and cyber

D. Gritzalis, B. Preneel, and M. Theoharidou (Eds.): ESORICS 2010, LNCS 6345, pp. 626–642, 2010.

attackers. By the time a new security flaw is discovered, thousands of cyber attacks are being reported before a suitable patch is released. For instance, an article from the techworld magazine discusses the effect of one security flaw for the Microsoft Internet Explorer browser. After just a couple of weeks this security flaw is disclosed, and before the next scheduled security updates are revealed, more than thirty thousand daily attacks have been reported [17]. In such a teased environment, security experts cannot only rely on the security of their products when they are constantly updated with new patches. Indeed, they must be equipped with powerful prevention and monitoring tools that help to prevent security breaches when they occur, and detect these before they succeed.

Setting and keeping security equipments require increasing investments that constitute a heavy burden on the shoulders of small and medium companies. Nonetheless, these equipments are often bypassed by skilled attackers, but also script-kiddies, proving to be insufficient when used as a single line of defense. As a consequence, intrusion response systems have started to play a growing role in today's security architectures [24]. Response systems are unable to prevent the threat before it occurs, but they react as it occurs in order to prevent its direct effects. They modify configuration setups in order to contain an attack or prevent its success. Unfortunately, trivial responses that use static mappings between intrusive patterns and available responses do not provide a reliable solution. Attackers rapidly learn about those mappings and adapt their attacks in consequence. This has promoted the need for more advanced response systems that implement more sophisticated strategies, including security policies [8].

Although response systems have reached a high level of sophistication [24], security experts are still reluctant to use those systems due to the potential of damages they may provoke. In fact, current systems are increasingly growing in complexity. They experience growing trends towards providing more interactive services to support every user's need in terms of quality of service. Service providers are thus constrained to use granular and interdependent service architectures which yield a better agility for service configuration. However, the aftermath of one security breach could be drastic because impacts rapidly spread through service dependencies [10]. Besides, cyber attacks are becoming more sophisticated. Internet provides an exceptional facility to conduct collaborative attacks or to use the excessively available botnets [20]. As attacks are getting extremely complicated, they require accurate and severe responses in order to be contained or blocked. The decision to take these responses is more likely to be taken by a security expert in order to avoid boomerang effects as the self-inflicted denial of service. We believe that the lack of a comprehensive approach to represent service dependencies is a major reason for not using those dependencies to support decisions for intrusion response. This is a major limitation because service dependencies provide a well-suited platform to compare intrusion responses and to select cost-sensitive responses [12][25]. One major contribution of this paper is thus to develop a new service dependency representation that is used to evaluate and compare intrusion and response impacts in order to select optimal responses, i.e. responses that inflict less impact to the system.

When inspecting the literature of the domain, we find that dependencies in their broadest sense have been longly used for intrusion response. We will thus precise what makes this contribution so different. In fact, dependencies are mostly structured into dependency graphs that embed either *logical* or *functional* dependencies. In the former category, we cite the examples of privilege graphs, attack graphs and vulnerability graphs. Privilege graphs [7][6] trace the attacker privilege escalations in target systems. Dependencies point-out privileges that enable an attacker to achieve a security objective. Attack graphs [23][3] specify causal relations between potential attacker actions. Finally, vulnerability graphs [2][11] describe the access that is required for an attacker in order to carry out an attack. These approaches offer to assess intrusion costs by statically assigning quantitative cost metrics to nodes in these graphs. Intrusion costs are evaluated as the aggregation of elementary costs for the already achieved steps in the graph. Meanwhile, and unless they rely on expert knowledge, no formal approach to evaluate elementary costs is yet provided. This is due to the fact that critical services and their dependencies are not represented in these graphs. Moreover, service dependencies are dynamic and may be modified by intrusion responses, which modifies the static elementary costs in these graphs. Another limitation for using these graphs without further extensions is the inability to assess response costs. In fact, only *positive* response costs may be evaluated, i.e. intrusion costs that are prevented by a response. To the best of our knowledge, no formal approach to evaluate response collateral damages can be applied.

On the other hand, functional dependencies are dependencies between system resources [4][10][12][13][26]. They represent the need for a dependent resource to use an antecedent resource in order to be fully operational. Functional dependency graphs propagate impacts as applied in system dependability management. Although they are more appropriate to evaluate intrusion and response impacts, these graphs suffer multiple limitations. In fact, intrusion impacts are often propagated downwards in these graphs, i.e. from an antecedent resource towards a dependent resource. They are also used to evaluate only response *negative* impacts, i.e. the response collateral damages. However, they are less likely to be used in order to evaluate response positive impacts.

This paper is motivated by the limitations of both existing approaches. It rather bridges the gap between them, by representing both security objectives in logical dependency graphs and resource dynamic dependencies in functional dependency graphs. It provides a new service dependency representation and implements intrusions and responses using the same semantics as for service dependencies. A simulation platform is defined, which simulates intrusion impacts, response impacts and the combined impacts for intrusion and response. Measures are further aggregated in order to select the most cost-effective response. This paper is structured as follows. Section 2 defines a new response index in order to compare and select intrusion responses. Section 3 implements the attributes that constitute this index using service dependencies. Section 4 presents the simulation platform that is used to compute those attributes. Section 5 demonstrates the use of this platform using a real-world example and section 6 concludes.

2 Return-On-Response-Investment index

The fundamental question an intrusion response system must answer is whether the self-inflicted response cost is reasonably tolerated when reacting against an intrusion attempt. In fact, this question challenges every aspect of IT security, that is whether a security investment is justified to avoid a security threat. Finance books provide multiple indicators that answer this question, among which the Return-On-Investment (ROI) index [21]. The ROI index, written as the ratio of net benefit to costs, compares multiple investment alternatives. It is used during risk analysis and to decide about investments in intrusion prevention [1]. The yet Return-On-Security-Investment (ROSI) index is used to promote investments in preventive IT security measures. It is defined in [1] as:

$$\texttt{ROSI} = \frac{(\textit{Expected Losses} - \textit{Residual Losses}) - \textit{Investment Costs}}{\textit{Investment Costs}}$$

Among multiple security investments, the security expert picks-up the one that satisfies a *maximal positive* ROSI index. Based on these facts, this paper proposes a new decision making process for intrusion response that is rather based on a financial comparison of response alternatives. We consider that a system often specifies some security objectives that are directly expressed in terms of monetary losses when they are not satisfied. Intrusions and responses inflict some costs when they affect these security objectives. These costs are classified into three components: response goodness (RG), response collateral damages (CD) and response operational costs (OC) [25]. RG measures the response ability to reduce the costs inflicted by the intrusion attempt. By analogy to the ROSI index, RG is compared to the prevented losses due to security investments. CD is the cost that is added by a newly enacted response, and that is not related to intrusion costs. It is inherent to the response mechanism as the latter affects some security objectives. OC is independent from the security objectives of the system. It includes response setup and deployment costs, such as manpower and over provisioning. By analogy to the ROSI index, investment costs are compared to the aggregation of CD and OC. We adapt the ROSI index to the response process, and thus we obtain the Return-On-Response-Investment (RORI) index, as follows.

$$\texttt{RORI} = \frac{\texttt{RG} - (\texttt{CD} + \texttt{OC})}{\texttt{CD} + \texttt{OC}}$$

To specify the response goodness index, we define the following cost metrics. $\texttt{IC}_b$ represents expected intrusion impacts when no response is enacted. $\texttt{IC}_a$ represents expected intrusion impacts after some response is enacted. $\texttt{IC}_a$ is difficult to evaluate because it is almost impossible to discern intrusion and response costs when they are applied simultaneously. We thus propose the index RC to represent the combined impact for both intrusion and response. Based on these metrics, we develop the expression of the RORI index as follows.

$$\texttt{RORI} = \frac{(\texttt{IC}_b - \texttt{IC}_a) - (\texttt{CD} + \texttt{OC})}{\texttt{CD} + \texttt{OC}} = \frac{\texttt{IC}_b - (\texttt{IC}_a + \texttt{CD}) - \texttt{OC}}{\texttt{CD} + \texttt{OC}} = \frac{[\texttt{IC}_b - \texttt{RC}] - \texttt{OC}}{\texttt{CD} + \texttt{OC}}$$

The OC metric is not related to the system security objectives. It is statically defined as part of a risk analysis plan prior to system runtime. The three remaining metrics, i.e. IC_b, RC and CD, are evaluated online as soon as new intrusions are detected and new candidate responses are proposed. These metrics, for the same intrusion and response combinations, depend on current service configuration. In following sections, we propose a complete methodology to evaluate those metrics using service dependencies. The ultimate goal is to select the candidate response set that provides a maximal positive RORI index, if any.

3 Service Dependency Framework

3.1 Specification of System Security Objectives

We denote security objectives as the set of security guidelines that must be satisfied. These guidelines are specified within security policies that characterize users and their access permissions. Access permissions are sometimes explicitly granted to users; they constitute explicit privileges, e.g. all employees are granted personal laptops. IT systems also implement service architectures where users interact with the system in order to obtain additional privileges. Users are thus less likely to be granted explicit privileges, but only credentials that enable them to interact with the system, e.g. PKI certificates. Users belong to equivalence classes $\mathcal{C}u_i\rfloor_{i=1}^{n}$ where they enjoy the same access permissions. An equivalence class is compared to a role where all users have the same privileges. A user may thus belong to more than one equivalence class. We use equivalence classes and user privileges in order to define security objectives, as follows: The IT system must guarantee the secure user access to the privileges that are relevant within his/her equivalence class. The secure access covers privilege availability, i.e. user ability to acquire this privilege, but also privilege misuse, i.e. to be acquired by an unintended user. Failing to do so implies some security objectives to be unsatisfied. The resulting impacts are manifested as availability impacts, or confidentiality and integrity impacts following a privilege misuse. Before we evaluate those impacts, we first formalize the conditions under which they occur.

We assign a privilege p to an equivalence class $\mathcal{C}u_i$ using the predicate $assign$. Assigning a privilege to an equivalence class means that users of this class require access to this privilege. We express this statement as follows: $assign(p, \mathcal{C}u_i) \Leftrightarrow \forall u \in \mathcal{C}u_i, requires(u, p)$. We note that assigning a privilege to an equivalence class does not mean users are explicitly granted this privilege. It only means that a security objective is unsatisfied when users cannot acquire this privilege. We model the granting of a privilege to a user with the predicate $granted$, which implies the following statement: $assign(p, \mathcal{C}u_i) \not\Rightarrow \forall u \in \mathcal{C}u_i, granted(p, u)$. It follows up that revoking a privilege to a user within an equivalence class provokes the failure of a security objective only when this privilege is assigned to this class. We propose the $revoked$ predicate to represent this statement. It is defined as: $revoked(p, \mathcal{C}u_i) \Leftrightarrow \exists u \in \mathcal{C}u_i, assign(p, \mathcal{C}u_i), \neg granted(p, u)$.

Definition: we define an availability failure every combination of one privilege p and one equivalence class $\mathcal{C}u_i$ that satisfies the predicate $revoked(p, \mathcal{C}u_i)$.

We also define the condition for a privilege misuse, that is a privilege to be granted to an unintended user. We use the predicate $infected$ that is defined as: $infected(p, \mathcal{C}u_i) \Leftrightarrow \exists u \notin \mathcal{C}u_i : assign(p, \mathcal{C}u_i), \neg requires(u, p), granted\ (p, u)$. The outcome of a privilege infection is the granting of inappropriate permissions to a user, which may provoke integrity or confidentiality impacts, according to the permissions that are associated to this privilege.

Definition: we define a confidentiality or integrity failure every combination of one privilege p and one equivalence class $\mathcal{C}u_i$ that satisfies the predicate $infected(p, \mathcal{C}u_i)$.

We propose to use privilege infection and revocation in order to quantify costs for intrusion and response, i.e. to evaluate the metrics $\mathtt{IC}_b$, RC and CD that constitute the RORI index. Meanwhile, intrusions and responses target either users or system services. They infect and/or revoke privileges on either user-side or system-side. Impacts further propagate through service dependencies until they affect end-users. Precisely one needs to know how users interact with system services, and how dependencies influence the impact propagation process.

3.2 Privilege Sharing and Service Dependencies

Trust relationship: As far as users are only granted credentials, they interact with the system services in order to acquire the privileges they are assigned. Privileges are initially held by services and shared with users in counterpart to *trusted* credentials and privileges. We introduce trust relationships as part of an authorization scheme by which we specify the way privileges are shared between users and services. Trust relationships do not only apply to user credentials, but also to privileges. In fact, some services may evolve in a trusted environment, e.g. a shared repository service accessible via an Intranet connection. As a consequence, users who have the privilege of 'being connected to the Intranet' are granted the permission to 'upload data to this service'. We define a trust relationship tr using the predicate $trust(tr)$. tr has two attributes: (1) The *trustee* specifies a privilege or credential $priv_2$ trusted by the service $subj_1$ that implements tr (i.e. $implement(subj_1, tr)$). (2) The *grantee* specifies a privilege $priv_1$ that is granted by the service $subj_1$ when the trusted privilege or credential is used by a subject $subj_2$. We formalize the notion of trust as follows.

$$granted(priv_1, subj_2) \leftarrow trust(tr), implement(subj_1, tr), grantee(tr, priv_1), \\ trustee(tr, priv_2), granted(priv_2, subj_2)$$

Trust relationships are implemented in order to set and configure service dependencies. They enable access control as they restrain access to an antecedent service to the only service that is granted the trusted credentials and privileges.

Service dependencies are made explicit by a request to an antecedent service. The Role-based Trust-management (RT) framework in [16] represents the request concept as a delegation process by which the requester delegates some privileges to its request. The RT framework applies to role management and delegation.

For instance, that some subject Ea requests an authorization which belongs to the role Rb from Eb with its capacity of being empowered in the role Ra is represented as: $Ea \xrightarrow{\texttt{Ea as Ra}} Eb.Rb$. Service dependencies comply to the same request specification. A dependent service uses credentials and privileges to satisfy trust relationships implemented by an antecedent service. The role concept in the `RT` framework is treated as a collection of permissions [22], which makes it compatible with the privilege concept for service dependencies. We use the dot notation '.' to represent the fact that a subject $subj$ is granted a privilege or credential $priv$. It is defined as: $subj.priv \Leftrightarrow granted(priv, subj)$. We introduce a service dependency with the statement: $dep \xrightarrow{dep.priv_1} ant.priv_2$. It states that the dependent subject dep uses the privilege $priv_1$ in order to support its request through which it requires the privilege $priv_2$ from the antecedent subject ant.

Example: A web server has its root directories hosted by a network file system (`NFS`) service. Access to `NFS` service is controlled using the `/etc/exports` file where `IP` addresses are registered. The request statement is modeled as:

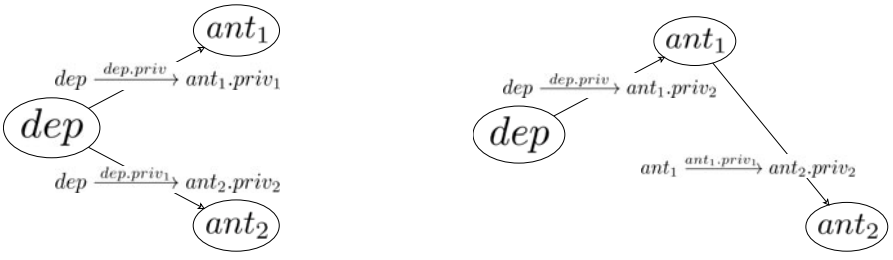
$$\texttt{Web} \xrightarrow{\texttt{Web IP in /etc/exports}} \texttt{NFS.permission(access, Root directories)}$$

Dependency satisfaction constraint: A dependency is satisfied after the dependent service obtains the required privileges. The outcome is expressed as the fact that the dependent service shares some privileges with the antecedent service. We infer, using the definitions of dependency and trust, the condition for a dependency to be satisfied. A dependency is satisfied if, and only if, the dependent service uses the credentials and privileges that are trusted by the antecedent service. The antecedent service trusts a privilege if it implements trust relationships that map between this privilege and the privilege requested by the dependent service. We formalize these concepts as follows.

$$(dep \xrightarrow{dep.priv_1,..,dep.priv_n} ant.priv_o \rightarrow granted(priv_o, dep)) \Leftrightarrow (\forall tr : (implement(ant, tr) \wedge grantee(tr, priv_o)), \exists i(trustee(tr, priv_i)))$$

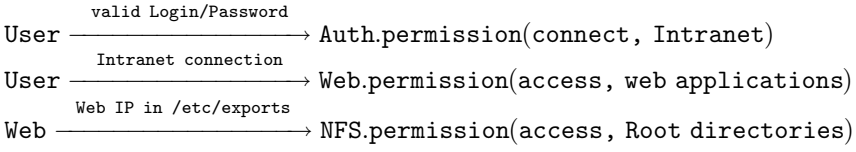
Dependency compositions occur when multiple dependencies contribute to providing the same privilege. Dependency compositions include two types of elementary patterns, which are logical and functional compositions. We use the same request statement to express these patterns, as in figure 1. In a logical composition (Fig. 1a), the dependent service cannot satisfy its second dependency for the service ant_2 until the former dependency for ant_1 has been satisfied. In a functional composition (Fig. 1b), the dependent service satisfies its unique dependency when the antecedent service ant_1 satisfies its own dependency for service ant_2. Dependency compositions constitute elementary patterns through which intrusion and response impacts propagate as discussed in section 3.3.

Example: In the example of the web and `NFS` services, the web service provides applications for `Intranet` users. We have two composition patterns. In fact, only authenticated users access the web service; the latter cannot answer requests unless it accesses the `NFS` service. We model this example as follows:



(a) Logical composition **(b)** Functional composition

Fig. 1. Dependency composition



3.3 Intrusions, Responses and Impact Propagations

This section uses service privileges in order to introduce the impact of intrusions and responses. It relies on existing methods to represent IT attacks, namely attack graphs. It rather bridges the gap between attack graphs and service dependency models using the dependency representation described in section 3.2.

Intrusion costs: It is shown in [5] that the impact of an attack on a target component can be fully characterized using pre/post-condition statements. Attack pre-conditions define the state of the target system prior to an attack success. Post-conditions define the system state after an attack succeeds. From a service dependency perspective, that means an attacker should have enough privileges to access a vulnerability and thus to conduct his attack. The outcome of this attack is that some privileges are either *revoked* to the target component or *infected* by the attacker. The National Vulnerability Database[1] provides a similar classification of vulnerabilities. It associates to a vulnerability (1) an access vector that indicates the requirements that enable exploiting a vulnerability and (2) an impact vector that indicates the post-attack effects on the target service.

We introduce a vulnerability v using the predicate $vulnerability(v, subj)$ where $subj$ is the vulnerable subject. v is assigned three properties: $infects(v, priv)$, $revokes(v, priv)$ and $access(v, priv)$. The first property specifies privileges $priv$ that are infected when v is exploited by an attacker. The second property specifies privileges that are revoked to the vulnerable service when v is exploited. The third property specifies attacker privileges that are required to access the vulnerability v. We use the predicate $vulnerability$ to propose a privilege-based definition of attacks. We use the same request statement as for service dependencies. Meanwhile,

[1] http://nvd.nist.gov/

the attack success criteria are different since the attacker uses an illicit access path. We define an attack and its success criteria as follows:

Listing 1.1. Attack success criteria

$$(att \xrightarrow{att.priv_1,..,att.priv_n} tgt.priv_o \rightarrow granted(priv_o, att)) \Leftrightarrow$$
$$(\exists v : (vulnerability(v, tgt), infects(v, priv_o)), \neg(access(v, q), \forall i(q \neq priv_i)))$$
$$(att \xrightarrow{att.priv_1,..,att.priv_n} tgt.priv_o \rightarrow revoked(priv_o, tgt)) \Leftrightarrow$$
$$(\exists v : (vulnerability(v, tgt), revokes(v, priv_o)), \neg(access(v, q), \forall i(q \neq priv_i)))$$

Privileges that are granted to an attacker are also infected privileges, we may thus infer the statement $infected(priv_o, tgt)$. An intrusion impact further propagates through service dependencies because the attacker abuses of trust relationships implemented to satisfy those dependencies. Impacts either propagate upwards or downwards as illustrated by the following propagation patterns.

Listing 1.2. Impact propagation patterns

$$infected(priv_o, tgt) \wedge \exists(ant, priv_a) : tgt \xrightarrow{tgt.priv_o} ant.priv_a \Rightarrow infected(priv_a, tgt)$$
$$infected(priv_o, tgt) \wedge \exists(dep, priv_a) : dep \xrightarrow{dep.priv_a} tgt.priv_o \Rightarrow infected(priv_o, dep)$$
$$revoked(priv_o, tgt) \wedge \exists(dep, priv_a) : dep \xrightarrow{dep.priv_a} tgt.priv_o \Rightarrow revoked(priv_o, dep)$$
$$revoked(priv_o, tgt) \wedge \exists(ant, priv_a) : tgt \xrightarrow{tgt.priv_o} ant.priv_a \Rightarrow revoked(priv_a, tgt)$$

We add intrusions to the model and we propagate their impacts using the propagation patterns in listing 1.2 in order to evaluate intrusion costs. These costs are a direct consequence to the intrusion impacts in terms of *revoked* and *infected* privileges, as in section 3.1. We may use existing approaches to convert service failures into costs, as in [15][25]. We evaluate the expected intrusion costs when no response is enacted, i.e. the $\texttt{IC}_b$ metric in the `RORI` expression. We still need to evaluate the `RC` and `CD` metrics, which requires to represent intrusion responses.

Response representation: We model intrusion responses using the same approach that we used to model intrusions. We shall point out two differences between intrusions and responses, and how we handle them in our model. A response is first a decision that is deliberately taken by the system. The latter degrades some security objectives in order to react against an ongoing threat. We thus dispose of the attacker notation (Att) in the response representation. Besides, a response does not infect privileges as intrusions do, at least for conventional responses we consider in this paper (e.g. quarantine host, set firewall rule, block port, stop service, block account). They only render a service more vulnerable to an attack. Privilege infections that may occur if this service is further attacked are not a direct consequence to this response. An attack is still required in order for these infections to take place. We thus dispose of privilege infections (i.e. $infects$ predicate) in the response representation.

A response grants and/or revokes some privileges to a target subject. It is expected to prevent the attack or to contain its impacts. An intrusion is prevented when the response revokes some privileges used by the attacker in order to access the target vulnerability. Intrusion impacts ($\texttt{IC}_a$) are thus reduced to null. Intrusion prevention analysis is not a particularity to our model, it is already handled using attack graphs and anti-correlation techniques [5]. Meanwhile, the containment of attack impacts is difficult to handle using only attack graphs. In fact, we separate between attack containment and attack impact containment. In the former, we shall prevent the attacker from using the privileges he acquired in order to conduct a new attack step. Attack containment is possible using techniques based on attack graphs and does not require excessive knowledge about service dependencies [9]. Meanwhile, attack impact containment requires interleaving with service dependencies in order to prevent the attacker from realizing any benefit when he/she uses the infected privileges. It prevents impact propagations through service dependencies as presented in the previous paragraph.

We introduce the predicate $response(resp, tgt)$ to model the enforcement of a response $resp$ on a target resource tgt. We represent the impact of response on the target resource as: $response(resp, tgt) \Rightarrow \exists priv : granted(priv, tgt) \vee revoked(priv, tgt)$. We note that this definition applies to one elementary response. A comprehensive response scenario against an ongoing attack includes multiple elementary responses that are modeled each using the predicate $response$. These responses interfere with intrusion impact propagations, by either increasing or -*hopefully* - decreasing those impacts. We shall evaluate response impacts in order to infer the RC and CD metrics.

Intrusion prevention is modeled using the following statement:

$$att \xrightarrow{att.priv_1,..,att.priv_n} tgt.priv_o \wedge \exists i(revoked(priv_i, att)) \Rightarrow \neg(infected(priv_o, tgt) \vee revoked(priv_o, tgt))$$

This response expression is a direct consequence to the attack success criteria in listing 1.1. These may no longer be satisfied because the attacker is revoked from privileges that he needs in order to access the vulnerable service. A targeted response like this does not impact the system security objectives because it only affects the malicious user. Although it often constitutes an ideal case, it could be impossible because of multiple reasons. In fact, an attacker may be unknown (e.g. IP spoofing), which constrains the system to select target-centric responses. An intrusion scenario may also be detected at mid-point to its ultimate goal. The known attacker would be only a stepping stone to the real remote attacker. Attacker-centric responses would thus apply to some system component and not to the real attacker. Besides, the attacker-centric response may be excluded because the system may not have enough capability to do so. Therefore, and by the time a privilege is revoked to some system component, impacts may further propagate as used for intrusion impacts.

Intrusion impact containment does not deal with the direct causes of an intrusion, but limits its impacts when they occur. Intrusion impacts are actually

prevented from propagating through service dependency paths. Responses interleave with these paths in order to stop impact propagations. This paragraph discusses propagation patterns in listing 1.2 and shows how responses interleave with these patterns. We use the example of the web-NFS dependency in section 3.2 in order to illustrate each of these patterns.

First propagation pattern represents upward infection propagation. Infected privileges for a dependent service are misused in order to infect privileges to an antecedent service. Impact propagation is contained by denying access to the antecedent service ($revoked(priv_o, tgt)$) or by quarantining (i.e. revoking) the threatened privileges for the antecedent service ($revoked(priv_a, ant)$). The counterpart of this response (CD metric) is to initiate new impact propagations that are described by the third and fourth propagation patterns in listing 1.2.

Example: An attacker conducts a buffer overflow against the web server, which enables him to execute arbitrary code using the web server permissions, including its IP address (`infected (web IP in /etc/exports, web)`). Upward propagation affects the NFS server, i.e. `infected(access root directories, web)` (please refer to the first pattern in listing 1.2). Responses revoke access to root directories for the NFS server (`revoked(access root directories, NFS)`) or deny web access to the NFS server (`revoked(web IP in /etc/exports, web)`).

Second propagation pattern represents downward infection propagation. Infected privileges for an antecedent service remain infected when they are shared with its dependent services. In fact, infected privileges are granted to the attacker. He actually uses these privileges wherever the service configuration (i.e. dependency) enables to do so. Downward propagation is prevented, as for upward propagation, by disabling the threatened dependency. It either requires to revoke infected privileges to the target resource, ($revoked(priv_o, tgt)$) or to deny access to those privileges for other dependent services ($revoked(priv_a, dep)$).

Example: An attacker targets the NFS server, and thus directly provoking the infection `infected(access root directories, NFS)`. Root directories remain infected when they are shared through the web-NFS dependency (please refer to the second pattern in listing 1.2). The denial of web access to the NFS service (`revoked(web IP in /etc/exports, web)`) keeps the root directories infected, but hampers the use of infected directories by the web service. Downward propagation is also hampered by denying access to the root directories for the NFS service, i.e. `revoked(access root directories, NFS)`.

Third propagation pattern mimics availability impact propagations that occur in case of functional dependency compositions. A privilege that is revoked to a service is also revoked to all its dependent services. Interleaving with availability propagations includes the ability to implement disjunctive dependencies. Impact propagation may be prevented if the following condition is satisfied. It expresses the ability for a dependent service to use more than only one antecedent service in order to obtain its required privileges.

$$revoked(priv_o, tgt) \wedge \exists(dep, ant, priv_a, priv_b) : (dep \xrightarrow{dep.priv_a} tgt.priv_o) \wedge$$
$$(dep \xrightarrow{dep.priv_b} ant.priv_o) \wedge granted(priv_o, ant) \Rightarrow \neg revoked(priv_o, dep)$$

Example: we discuss the example of a `DoS` attack against the `NFS` service. It revokes access to the root directories for the `NFS` service, i.e. `revoked(access root directories, NFS)`. This privilege may no longer be shared with the web service, i.e. `revoked(access root directories, web)`. Impact propagation may be hampered in case of another web-`NFS` dependency providing load balancing with the failed `NFS` dependency.

Fourth propagation pattern mimics availability impact propagations that occur in case of logical dependency compositions. A privilege that is revoked to a service may no longer be used by this service in order to support dependencies for other services. We also refer to dependency disjunction in order to illustrate the condition for a response to prevent this pattern. It is written as:

$$revoked(priv_o, tgt) \wedge \exists(ant_1, ant_2, priv_a, priv_1) : (tgt \xrightarrow{tgt.priv_o} ant_1.priv_a) \wedge$$
$$(tgt \xrightarrow{tgt.priv_1} ant_2.priv_a) \wedge granted(priv_1, tgt) \Rightarrow \neg revoked(priv_a, tgt)$$

The example for this inference rule is similar to the one of the third rule, but denying the web instead of the `NFS` service. It thus revokes the `IP` connection to the `NFS` service, i.e. `revoked(web IP in /etc/exports, web)`.

The evaluation of response collateral damages, i.e. the `CD` metric, consists of adding only the response to the model and to exclude the intrusion. Response collateral damages are not only restrained to availability impacts, that is privilege revocations. It may also provoke privilege infection when yet some granted privilege enables the propagation of an infection that was previously intercepted. This is automatically depicted by the inference process using the rules in listing 1.2. Response collateral damages are thus closely related to the current system state and may not be statically defined beforehand. On the other hand, the evaluation of the `RC` metric, that is the combined impact of intrusion and response, requires adding both intrusion and response to the model. The resulting cost after all impacts have been propagated corresponds to the metric `RC`. The use of inference rules that are based on first order logic statements guarantees the convergence of the propagation process within a polynomial time.

The response evaluation process presented in this section is used to assist intrusion response systems by comparing candidate responses. It is implemented in a dynamic environment that requires interleaving with the dependency model. An appropriate implementation of this model must be thus provided. We suggest using Colored Petri Nets (`CPN`) for this purpose.

4 Simulation Platform

4.1 Using Colored Petri Nets

Although we may use a datalog inference process to implement our model, we discarded this alternative for the following reasons. The use of our dependency

model as part of a cost-sensitive response mechanism requires interleaving with the inference process. This is, to the best of our knowledge, difficult to integrate in a datalog engine. Furthermore, the size of datalog inference engines may often be unacceptable. The complexity of these engines makes them inappropriate for systems including a large number of resources and dependencies.

On the other hand, CPNs [14] provide appropriate features to implement our model. They are extensions of petri nets where tokens transit between model places. We use CPN tokens to represent privileges in our model. A service dependency is modeled as a CPN transition that is enabled when some conditions are met, i.e. enough privileges for the dependent service to support its dependency. We also use CPN places to represent user equivalence classes. They are initially marked with default user privileges. User places thus interact with system services through well-defined interfaces. Attackers are actually modeled in a different way. They are not assigned explicit places because their behavior is considered as unpredictable. Any infected privilege (i.e. token) would be thus considered as an attacker property. Last but not least, a CPN simulator enables the iterative simulation and interleaving with the simulation process. It may also constrain the simulation to run on a transition basis. System costs are obtained after the CPN reaches a state where no more transitions are activated.

We transform the request statement that represents a service dependency into the CPN transition in figure 2. This transition shares tokens (i.e. requested privileges) between the dependent and the antecedent services represented as CPN places. This transition is constrained by the existence of specific tokens (i.e. dependency requirements) in the destination place (i.e. dependent service). The privileges used by the dependent service to support its request are implemented as a transition activation constraint. This transition satisfies the properties of the request statement for service dependencies. It is not activated unless the destination place contains the privileges that are required to support the dependency (i.e. $priv_i$). It also shares the privilege $priv_o$ between the dependent and antecedent resources. The transition in figure 2 also satisfies the impact propagation patterns. The token $priv_o$ is infected in the destination place when all $priv_i$ tokens are infected or when $priv_o$ was already infected at the source place (we add a boolean attribute to a token definition, it is set to true when this token is infected). We thus implement the first and second statements in listing 1.2. On the other hand, the transition is only activated when the source place includes the $priv_o$ token and the destination place includes the $priv_i$ tokens. We thus implement the third and fourth statements in listing 1.2.

4.2 Simulation Process

We implement the simulation platform using the CPN tools simulator[2]. The overall architecture we use is illustrated in figure 3. The service dependency model, expressed as a CPN skeleton (without initial marking) is a static input to the CPN simulator. It reliably describes the services that constitute the modeled

[2] http://wiki.daimi.au.dk/cpntools/

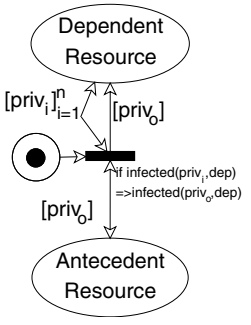


Fig. 2. CPN dependency

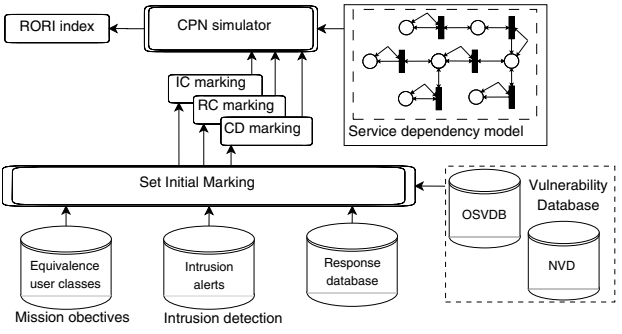


Fig. 3. Dependency simulation framework

environment and its dependencies. It is obtained by transforming dependency statements into a CPN model, as shown in the previous paragraph. The initial marking of the CPN model is dynamic. It characterizes the dynamic system state, including currently applied responses and the existing, but contained, intrusions. Responses either grant or revoke privileges to their target components. This is represented as token assignment or token extraction from appropriate places. A token is either explicitly infected by an intrusion when determining the initial marking of the CPN model (infection boolean attribute set to true), or further infected through CPN simulation (please refer to figure 2). By the end of the simulation, all infected tokens (the boolean attribute set to true) and revoked tokens (by comparing the final marking of the CPN model to the equivalence classes) are used to evaluate the RORI index attributes. As illustrated in figure 3, the CPN model is simulated three times for each candidate response set. The first simulation is executed by introducing only the intrusion attempt in order to obtain the IC_b metric. The second simulation is executed by introducing only the response in order to obtain the CD metric. Finally, the third simulation is executed by introducing both intrusion and response in order to obtain the RC metric. Measures are combined within the RORI index. This operation is iterated for all candidate response sets that are proposed by an external response system. We finally choose the response set that provides a maximal positive RORI index.

5 Case Study

We demonstrate the use of our proposal through the simplified example of an enterprise email service, which is illustrated in figure 4. It uses IMAP and the native exchange mailing protocols, i.e. outlook and outlook web access. Intranet users access the email service using a courier-Imap server or the outlook web access (OWA). The courier-Imap server uses IMAP extension for the exchange server. Extranet users connect to the email service through web access to an apache2 server connected to a DMZ. The latter connects to the IMAP server or to the OWA server using the mod-proxy extension for apache2 server.

In our simple example, we pick-up two user classes, which are `Intranet` and `Extranet` email users. `Extranet` users have the privilege of being able to connect to the web server, as well as one credential, which is a valid email account. `Intranet` users have the privilege of being connected to the `Intranet`, and the same credential as for `Extranet` users. Both user classes require access to mailboxes hosted by the exchange server. The security objectives in this example are to guarantee the mailbox availability and to prevent unintended mailbox access. The email platform also includes four elementary services, which are the web, `IMAP`, `OWA` and Exchange services. Service dependencies are summarized in the resulting `CPN` model in figure 5. The default initial marking (i.e. with no security threats) is illustrated within parenthesis inside `CPN` places.

We examine the following attack scenario. In a first step, an attacker exploits a vulnerability to the web application that enables to execute arbitrary code on the web server. The attacker uses the web server as a stepping stone in order to access the `IMAP` server through the `DMZ-Intranet` firewall. He exploits a flaw for the authentication front-end of the `courier-Imap` server which enables the attacker to connect to the `IMAP` server. The attacker finally conducts a buffer overflow attack to have a root shell on the `IMAP` server. The first attack step infects the `Dm` token (please refer to figure 5), that is to have direct access to the `DMZ`. The infection does not propagate through the dependency between the web and the mail delivery services because the `web-Intranet` transition also requires the `Vm` token to be infected (i.e. a valid user account) in order to propagate the infection. The second attack step infects the `Int` token, that is the connection to the `IMAP` service. The infection does not propagate elsewhere (please check the `CPN` model). The last attack step infects the token `Ex`, i.e. the `IMAP` account to the exchange server. The infection propagates through the `IMAP`-Exchange transition because both `Ex` and `Int` tokens are infected. The mailbox access (`Mb`) token is thus infected (first propagation pattern in listing 1.2). One security objective has failed, that is the misuse of user mailboxes ($infected(Mb, Exchange)$).

Two responses are possible: The first blocks access to the web application. It revokes the `Cw` token for `Extranet` users since the attacker is unknown. The second denies access to the Exchange server for the vulnerable `IMAP` service, i.e. to revoke the token `Ex` to the `IMAP` place. By revoking the `Cw` token, extranet users cannot access their emails because the appropriate transitions are disabled

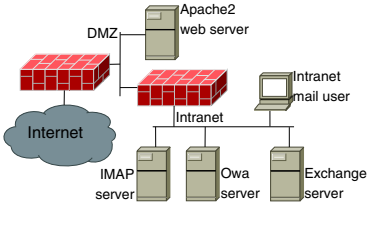


Fig. 4. Email case study

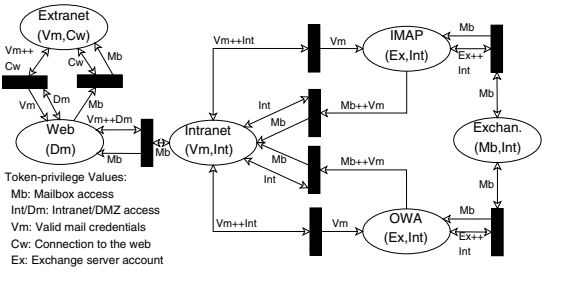


Fig. 5. `CPN` dependency representation

(fourth propagation pattern in listing 1.2). Meanwhile, `Intranet` users are not affected (please check the `CPN` model). The second response prevents email access only through the `IMAP` protocol while not affecting other protocols. The second response has less collateral damages since it does not prevent email access for external users (disjunctive dependency that prevents the third propagation pattern in listing 1.2). Both responses prevent the infection to propagate to user mailboxes. Based on the `RORI` index values provided by the `CPN` simulations discussed above, the second response is selected when detecting the third attack.

6 Conclusion

We implemented in this paper the Return-On-Response-Investment as an adaptation to the `ROI` index. The `RORI` index, in contrast to most existing response evaluation functions, does not use informal cost metrics that rely on expert knowledge. Rather, it is accompanied with a complete methodology to evaluate the metrics that contribute to this index. To the best of our knowledge, the `RORI` index is the first to consider not only response collateral damages, but also response effects on intrusion. The `RORI` index is implemented along with a comprehensive service dependency model that enables to track intrusion and response impacts in the target system. By doing so, we introduce intrusions and responses in the model and analyze the interference between their impacts. The `RORI` index supports privilege infections, that express confidentiality and integrity impacts, but also privilege revocations that express availability impacts. It outruns most existing response evaluation mechanisms that rely on dependability management techniques and therefore only apply to availability failures.

Future work will investigate how time may be added to the `RORI` index. The cost of a privilege infection or revocation may not be static. A response may have a higher `RORI` index when an attack is detected, but later for this index to be degraded in favor of other responses. We may also extend the `RORI` index to tune impacts as the attacker gets closer to critical mission objectives. An intrusion impact will not be restrained to the direct effects in terms of privilege infection and revocation. It will also consider the impact of new attack steps that are made possible by the current intrusion if no response is enacted.

References

1. Aceituno, V.: Return on security investment. ISSA Jounal 1, 16–19 (2006)
2. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proc. 9th ACM Conf. on Computer and Communications Security, pp. 217–224 (2002)
3. Artz, M.L.: A Network Security Planning Architecture. Ph.D. thesis, Cambrdige: Massachusetts Institute of Technology (May 2002)
4. Balepin, I., Maltsev, S., Rowe, J., Levitt, K.: Using specification-based intrusion detection for automated response. In: Vigna, G., Krügel, C., Jonsson, E. (eds.) RAID 2003. LNCS, vol. 2820, pp. 136–154. Springer, Heidelberg (2003)
5. Cuppens, F., Autrel, F., Yacine Bouzida, J.G., Gombault, S., Sans, T.: Anti-correlation as a criterion to select appropriate counter-measures in an intrusion detection framework. Annals of Telecommunications 61, 197–217 (2006)

6. Dacier, M., Deswarte, Y., Kaaniche, M.: Quantitative assessment of operational security: models and tools. Tech. Rep. 96493, LAAS (May 1996)
7. Dacier, M., Deswartes, Y.: Privilege graph: An extension to the typed access matrix model. In: European Symp. on Research in Computer Security, pp. 319–334 (1994)
8. Debar, H., Thomas, Y., Cuppens, F., Cuppens-Boulahia, N.: Enabling automated threat response through the use of a dynamic security policy. Journal in Computer Virology 3, 195–210 (2007)
9. Foo, B., Wu, Y.S., Mao, Y.C., Bagchi, S., Spafford, E.: Adepts: Adaptive intrusion response using attack graphs in an e-commerce environment. In: Proc. Intr'l Conf. DSN, pp. 508–517 (2005)
10. Jahnke, M., Thul, C., Martini, P.: Graph based metrics for intrusion response measures in computer networks. In: 32nd IEEE Conf. Local Computer Networks (2007)
11. Jajodia, S., Noel, S.: Topological vulnerability analysis: A powerful new approach for network attack prevention, detection, and response. Algorithms, Architectures and Information Systems Security 1, 285–305 (2007)
12. Kheir, N., Debar, H., Cuppens, F., Cuppens-Boulahia, N., Viinikka, J.: A service dependency modeling framework for policy-based response enforcement. In: Flegel, U., Bruschi, D. (eds.) DIMVA 2009. LNCS, vol. 5587, pp. 174–193. Springer, Heidelberg (2009)
13. Kheir, N., Debar, H., Cuppens-Boulahia, N., Cuppens, F., Viinikka, J.: Cost assessment for intrusion response using dependency graphs. In: Proc. IFIP Intrn'l Conf. N2S (2009)
14. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured petri nets. Intr'l Journal Software Tools for Technology Transfer, 98–132 (1998)
15. Lee, W., Fan, W., Miller, M., Stolfo, S.J., Zadok, E.: Toward cost-sensitive modeling for intrusion detection and response. Journal of Computer Security 10, 5–22 (2002)
16. Li, N., Mitchell, J., Winsborough, W.: Design of a role-based trust-management framework. In: Proc. IEEE Symp. on Security and Privacy, p. 114 (2002)
17. McMillan, R.: Internet explorer vulnerable to hackers, warn experts. microsoft and avg warn of danger. TechWorld magazine (March 2010)
18. Mead, N.R., McGraw, G.: A portal for software security. In: IEEE Security & Privacy, pp. 75–79 (2005)
19. Microsoft: Why microsoft dynamics for high-tech and electronics manufacturers? Microsoft Dynamics CRM
20. Rajab, M.A., Zarfoss, J., Monrose, F., Terzis, A.: A multifaceted approach to understanding the botnet phenomenon. In: Proc. 6th ACM Conf. Internet measurement, pp. 41–52 (2006)
21. Ross, S., Westerfield, R., Jordan, B.: Fundamentals of Corporate Finance Standard Edition. McGraw-Hill/Irwin (2005)
22. Sandhu, R.S., Coynek, E.J., Feinsteink, H.L., Youmank, C.E.: Role-based access control models. IEEE Computer 29, 38–47 (1996)
23. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: IEEE Symp. Security & Privacy (2002)
24. Stakhanova, N., Basu, S., Wong, J.: A taxonomy of intrusion response systems. Intr'l Journal of Information and Computer Security 1, 169–184 (2007)
25. Strasburg, C., Stakhanova, N., Basu, S., Wong, J.S.: Intrusion response cost assessment methodology. In: Proc. ACM Symp. ASIACCS, pp. 388–391 (2009)
26. Toth, T., Kruegel, C.: Evaluating the impact of automated intrusion response mechanisms. In: Proc. 18th Annual Conf. ACSAC (2002)