Ex-SDF: An Extended Service Dependency Framework for Intrusion Impact Assessment

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,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. Information systems are increasingly dependent on highly distributed architectures that include multiple dependencies. Even basic attacks like script-kiddies have drastic effects on target systems as they easily spread through existing dependencies. Unless intrusion effects are accurately assessed, response systems will still be blinded when selecting optimal responses. In fact, using only response costs as a basis to select responses is still meaningless if not compared to intrusion costs. While conventional responses provoke mostly availability impacts, intrusions affect confidentiality, integrity and availability.

This paper develops an approach to assess intrusion impacts on IT systems. It uses service dependencies as frames for propagating impacts. It goes beyond existing methods which mostly use dependability analysis techniques. It explores service privileges as being the main targets for attackers, and the tunable parameters for intrusion response. The approach presented in this paper is implemented as a simulation-based framework and demonstrated for the example of a vehicle reservation service.

1 Introduction

Despite the efforts to improve their security, IT systems continue to provide large incentives to attackers because of the benefits that can be realized by compromising those systems. Even the most robust systems can be brought to their knees within a short time by the so-called script-kiddies, which are easily accessible through the Internet. In such circumstances, automated response has proven to be essential, and especially cost-sensitive response [4,18]. As in [14], informing the response process starts by assessing intrusion impacts.

While attack graphs trace dependencies between elementary steps constituting an exploit plan, each step is only assigned an abstract cost, without providing formal evaluation methods [16,13]. It has been shown that service dependencies enable to reason about intrusion impacts [1,6,10,19]. However, service dependency models, by using tree- [19] or graph- [1,6,10] based structures, are unable

K. Rannenberg, V. Varadharajan, and C. Weber (Eds.): SEC 2010, IFIP AICT 330, pp. 148–160, 2010.

to catch the way intrusion impacts spread in the system. They only evaluate availability impacts, but not confidentiality nor integrity.

In service oriented architectures, an intruder uses the privileges obtained on the target service in order to increase his benefits [3]. An intrusion impact propagates through some dependencies to the target service, but not all dependencies. Existing tree or graph-based service dependency models do not represent conditional impact propagations because they do not implement attacker privileges. This paper formally represents service dependencies. It enables the inline evaluation of intrusion costs using both the privileges realized on the target service and its dependencies. The notion of privilege enables the distinction of availability impacts from those of confidentiality and integrity. In the former, the attacker revokes some privileges to legitimate users. In the latter, the attacker acquires illicit privileges, and thus to have fraudulent access to some assets.

In [9], a service dependency framework is presented. It provides a platform which models a service workflow and the topology of its dependencies. Dependencies are modeled as connections which share data access between inter-dependent services. The framework in [9] is used for exploring possible applications of an intrusion response, but it does not include attributes for assessing response impacts. In fact, it does not include the privileges which apply to the data shared through service dependencies. This paper extends the framework in [9]. It introduces the notion of privilege, and thus extends the representation of services as black boxes. Services are modeled as components which are granted privileges over some assets. An intrusion impact covers the privileges realized on the target service and the services which use the target service. We note a main difference between the purpose of this paper, which is to evaluate the impact of elementary intrusions, and the prediction of attacker objectives as used in exploit graphs [8]. Exploit graphs often assign static damage costs to each elementary node within these graphs [17]. These costs are statically set, informally defined and do not consider the dynamic aspect of service dependencies.

This paper will be structured as follows: Section 2 extends the service definition in [9] by adding the notion of privilege. Section 3 defines service dependencies, models attacks and specifies the conditions for attack impact propagation using first-order logic predicates. Section 4 implements the service dependency model, and describes a simulation platform using Colored-Petri Nets. Section 5 demonstrates the use of our approach using a web service example. Finally section 6 summarizes related work and section 7 concludes.

2 Privilege: An Extension to the Service Model

This paper uses the service representation in [9] where a service is modeled as an *entity* which handles access to some *assets*. A service implements interfaces which share access to these assets with users. An access is studied in [9] as a whole, and the authorizations granted on a given asset do not figure in the model. This is a major limitation when dealing with intrusion impacts.

This paper extends the definition in [9] by adding the following specifications: (1) the privileges granted to the service, (2) the credentials accredited to this

service and (3) the trust it has regarding other privileges or credentials. A service is defined as $S = (\mathcal{P}r_S,\ \mathcal{T}r_S,\ \mathcal{C}r_S)$, where $\mathcal{P}r_S$, $\mathcal{T}r_S$ and $\mathcal{C}r_S$ respectively represent the privileges, trust relationships and credentials implemented by service S. These implementations specify permissions granted to a service and configure the way it interacts with other services through service dependencies. This section defines the notions of privilege, credential and trust. Further we use these definitions to propose a representation of service dependencies.

2.1 Privilege Specification and Assignment

We define an authorization as a *logical right* that applies to some assets. An authorization may be granted to a subject. We introduce the notion of privilege to model the *grant* of a permission to a subject. A privilege is specified as follows:

$$permission(Subj, Act, Obj) \leftarrow privilege(Priv), subject(Priv, Subj), \\ authorization(Priv, Auth), action(Auth, Act), object(Auth, Obj).$$

A privilege specifies a subject and an appropriate authorization. The latter includes an action which applies to an object. We represent a privilege $Priv$ detained by a subject $Subj$ with the notation $Subj.Priv$. It is interpreted as: $Subj.Priv \Leftrightarrow privilege(Priv), subject(Priv, Subj)$. We use privileges in order to define security objectives in terms of confidentiality (C), integrity (I) and availability (A). We argue that the assignment of CIA cost vectors to critical assets does not provide enough expressiveness. As discussed in [6], A is not managed the same way as for C and I. C and I are appropriate to the asset to which they apply, but A is related to the asset and the entity which uses this asset.

We specify security objectives in terms of C and I as cost metrics assigned to their appropriate assets. They are defined as square cost vectors (C_i, I_i) which apply to asset a_i. The metric C_i (resp. I_i) is assigned a higher value when the compromise of the C (resp. I) of a_i provokes higher losses to the system. The resulting cost for illicitly acquiring an authorization α which applies to an asset a_i is evaluated to $max(c_\alpha \times C_i, i_\alpha \times I_i)$. c_α (resp. i_α) is a boolean variable that is set to null when α does not disclose (resp. alter) the C (resp. I) of a_i. We specify security objectives in terms of A by assigning cost scalars to privileges rather than objects. Some privilege $S.Priv$ is critical if the unavailability of $Priv$ to the subject S (i.e. user) provokes high system losses. While C and I impacts are evaluated according to the authorizations illicitly acquired by an attacker, A impacts are evaluated according to the privileges which are revoked to their appropriate users. We thus have more granularity to evaluate availability failure costs because some privileges may be denied to certain, but not all, users.

2.2 Privilege Sharing: Credential and Trust

We define a credential as an 'entitlement to privilege' [12]. It is not coupled to an object, but to an entity that trusts this credential and shares in counterpart some privilege. A credential enables an entity which is not assigned some

privilege, to share this privilege with other entities. We introduce credentials with the predicate *credential*, as follows: $credential(Cr) \Leftrightarrow \exists(Subj_1, Subj_2) : owner(Cr, Subj_1), authority(Cr, Subj_2)$. It means the credential Cr is granted to subject $Subj_1$, and trusted by subject $Subj_2$. We use the notation $Subj.Cr$ to represent a credential Cr owned by a subject $Subj$, and we write $Subj.Cr \Leftrightarrow credential(Cr), owner(Cr, Subj)$. We define trust as an association of a privilege to be shared in counterpart to some credentials and/or privileges. Trust relationships are implemented as part of an authorization scheme by which we may specify the way privileges may be shared between the different subjects of a system. We introduce the predicate *trust* as follows:

$$trust(tr) \Leftrightarrow \exists(subj, inp, out) : subject(tr, subj), grantee(tr, out), \\ privilege(out), trustee(tr, inp), credential(inp) \vee privilege(inp).$$

In other terms, the subject *subj* implements a trust relationship by which it shares the privilege *out* in counterpart to the trusted credential or privilege *inp*. The satisfaction of a trust relationship results in additional authorizations granted to the trusted subject (i.e. the subject which has the trusted credentials or privileges). The satisfaction of a trust relationship is formalized as follows:

$$Subj_2.out \leftarrow trust(tr), subject(tr, subj), trustee(tr, inp), grantee(tr, out), \\ subject(out, subj), [privilege(inp), subject(inp, Subj_2)] \vee \\ [credential(inp), owner(inp, Subj_2), authority(inp, subj)]$$

Trust relationships configure access through service dependencies. The satisfaction of a dependency is constrained by the implementation of appropriate trust relationships. These are often bypassed by intruders, as described in section 3.

3 Dependencies and Impact Propagation

Service dependencies express the need for dependent services to access antecedent services. A dependent service requires some privileges not assigned to this service. It accesses an antecedent service in order to obtain the required privileges.

We represent service dependencies using the RT^D framework in [12]. RT^D introduces the concept of request which is represented by a delegation credential that delegates from the requester to the request. For example, that *Ea* requests an authorization which belongs to the role *Rb* from *Eb* with its capacity of being empowered in the role *Ra* is represented by: $Ea \xrightarrow{\texttt{Ea as Ra}} Eb.Rb$. We use the same delegation concept, but replacing roles with privileges. This is motivated by the fact that the role concept in role-based management languages is treated as a collection of authorizations [15], which makes it compatible with the privilege concept for service dependencies.

We represent the dependency for a service A towards a service B as follows: $A \xrightarrow{\texttt{(A.Cr, A.Pr)}} B.\mathcal{R}$. It states that service A, in its faculty of having the credential (Cr) and/or the privilege (Pr), requests the privilege $\mathcal{R}$ from the antecedent

service B. The dependent service, after it satisfies its dependency, acquires additional privileges granted by the antecedent service. The satisfaction of the dependency implies the sharing of the privilege $\mathcal{R}$ between the dependent and the antecedent services.

We use the definitions of a service dependency and trust in order to specifiy the condition for a dependency to be satisfied. It is written as:

$$(A \xrightarrow{\texttt{A.Cr,A.Pr}} B.\mathcal{R} \Rightarrow A.\mathcal{R}) \Leftrightarrow \forall tr : (trust(tr), subject(tr, B), grantee(tr, \mathcal{R})), \\ [(trustee(tr, Cr), owner(Cr, A)) \vee (trustee(tr, Pr), subject(Pr, A))].$$

Dependencies are satisfied only when dependent services use credentials and privileges which apply to trust relationships implemented by antecedent services.

3.1 Modeling Attacks in the Framework

As discussed in section 2.1, a privilege is affected when it is illicitly acquired by an attacker or denied to legitimate users. We introduce an intrusion as a way by which an intruder alters privilege assignments, either by denying access to legitimate users or providing illegitimate access to the intruder. We define infected privileges as privileges that are illegally acquired by an attacker, and revoked privileges as those which are illegally revoked to a target service.

We use exploited vulnerabilities to represent attack impacts on target services. We define a vulnerability using the same pre/post-condition model as in [2]. We introduce for this purpose the following attributes: (1) *target* to represent the vulnerable service, (2) *access* to represent the vulnerability access vector (i.e. the privileges which must be satisfied by the attacker before he could access the vulnerability), (3) *infects* to represent privileges for the target service that are infected by the intruder and (4) *revokes* to represent privileges which are revoked to the target service after the vulnerability is exploited. We model an attack using the same request statement as for service dependencies. An attacker, with his faculty of having some privileges (i.e. vulnerability access vector), exploits a vulnerability on a target service in order to increase his benefits and thus to acquire additional privileges and/or deny other privileges to the target service. The attack success condition is extended in order to include information about the exploited vulnerability. We introduce attack impact as follows:

$$[Att \xrightarrow{Att.Pr} B.\mathcal{R} \Rightarrow Att.\mathcal{R}] \Leftrightarrow [\exists v : vulnerability(v), \\ target(v, B), infects(v, \mathcal{R}), access(v, Pr), subject(Pr, Att)].$$

$$[Att \xrightarrow{Att.Pr} B.\mathcal{R} \Rightarrow \neg(B.\mathcal{R})] \Leftrightarrow [\exists v : vulnerability(v), \\ target(v, B), revokes(v, \mathcal{R}), access(v, Pr), subject(Pr, Att)].$$

We introduce the predicate $infected(B.\mathcal{R})$ to represent the outcome of the first attack, and the predicate $revoked(B.\mathcal{R})$ to represent the outcome of the second. We may also explore the correlation of attacks by comparing the outcome of one

attack to the access vector of the second, as in [2]. Attack correlation combines elementary impacts with the level of expertise required for succeeding an attack and the prediction of intrusion objectives in order to foresee additional impacts. This is a subject of interest which must be detailed in a future extension to this study. This paper thus evaluates impacts of elementary (i.e. separated) attacks.

3.2 Attack Impact Propagation

The impact of an attack propagates when components other than the target component are affected by this attack. The attacker acquires (resp. revokes) privileges granted to components other than his target component. He bypasses the trust relations already configured for service dependencies, by using the privileges he already acquired, in order to increase his gain (i.e. system loss). We infer, using the definitions of attacks, dependencies and trust relationships, the conditions for attack impact propagation which are summarized in listing 1.1.

Listing 1.1. Attack impact: propagation of infections and revocations

$$\textit{Statement } 1: \ \textit{infected}\ (A.\mathcal{R}) \wedge \exists\ (B, \mathcal{Q}) : A \xrightarrow{A.\mathcal{R}} B.\mathcal{Q} \Rightarrow \textit{infected}\ (A.\mathcal{Q})$$
$$\textit{Statement } 2: \ \textit{revoked}\ (B.\mathcal{R}) \wedge \exists\ (A, \mathcal{Q}) : A \xrightarrow{A.\mathcal{Q}} B.\mathcal{R} \Rightarrow \textit{revoked}\ (A.\mathcal{R})$$
$$\textit{Statement } 3: \ \textit{revoked}\ (A.\mathcal{R}) \wedge \exists\ (B, \mathcal{Q}) : A \xrightarrow{A.\mathcal{R}} B.\mathcal{Q} \Rightarrow \textit{revoked}\ (A.\mathcal{Q})$$
$$\textit{Statement } 4: \ \textit{infected}\ (B.\mathcal{R}) \wedge \exists\ (A, \mathcal{Q}) : A \xrightarrow{A.\mathcal{Q}} B.\mathcal{R} \Rightarrow \textit{infected}\ (A.\mathcal{R})$$

Statement 1 characterizes an opportunistic attacker who accesses an antecedent service after his attack against a dependent service. The attacker illicitly acquires from the dependent service some credentials and/or privileges which are trusted by the antecedent service. The attacker benefits are thus extended to include all the privileges granted by the antecedent service. Statement 2 illustrates availability propagation. The revocation of some privileges to an antecedent service makes them unavailable for its dependent services. In statement 3, some credentials and/or privileges that satisfy some access to an antecedent service are revoked to the target service. As a consequence, the privileges shared by the antecedent service are also revoked to this service, and subsequently to all its users. Statement 4 characterizes an undisciplined attacker who uses the infected privileges in order to access any dependency and thus to increase his gain.

Impacts iteratively propagate through service dependencies. The resulting impact corresponds to all infected ($\forall(u, Pr) : \textit{infected}(u.Pr)$) and revoked ($\forall(u, Pr) : \textit{revoked}(u.Pr)$) privileges. Our model also evaluates the conjunction of multiple attacks. By separately infecting more privileges, more dependencies could be infected, and so more damages could be inflicted to the system. In the following section, we develop a simulation platform which implements this model. It simulates impacts inline by mapping attacks onto our model and propagating impacts as in listing 1.1 in order to assess intrusion damages.

4 Implementation as Colored Petri Nets

4.1 Use of Colored Petri Nets (CPNs)

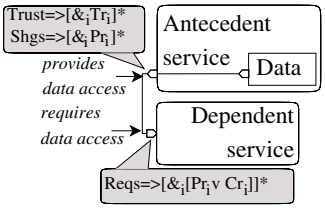
A straightforward approach to implement our model is to use a datalog inference engine. Meanwhile, we discarded this approach for two reasons. Firstly, the tracking of impact propagations may require interleaving with the inference steps, which is, to the best of our knowledge, hard to integrate in a datalog engine. Secondly, the size and complexity of datalog inference engines restrain their use for large systems. They do not guarantee the process termination in presence of cyclic dependencies. On the other hand, our model includes several characteristics. Firstly, it uses privileges which travel through specific dependencies. The satisfaction of a dependency adds new privileges to some dependent component. Secondly, we include attackers who modify the system status by infecting or revoking privileges. They use those privileges to extend the impact of their attacks, and thus to modify the system state. Thirdly, attack impacts propagate iteratively. Attacker gain is revaluated in a recursive way each time he acquires new privileges. A simulation framework thus better fits to our model.

`CPNs` [11] are an appropriate tool to implement our model. Firstly, `CPNs` enable token classification into different types. We model privileges as tokens and services as places. A dependency transfers some tokens from an antecedent place to a dependent place. We use the concept of token exchange to model privilege exchange in our model. Secondly, users and attackers are differently represented in our model. Users may access only public service interfaces. They are assigned explicit places which are initialized with default user privileges. Attackers use any interface and may spoof any service or user place. They are not assigned explicit places. We introduce the concept of infected tokens to represent attacker gain. Thirdly, the use of `CPNs` enable the iterative simulation of our model. A single transition is activated at a given time. It initiates a system state transition. The new state is characterized with new privileges acquired by a dependent component (either a user, service or attacker). Attacker benefits are obtained after the `CPN` reaches a state where no more transitions are enabled.

4.2 Building the Simulation Platform

`AADL` Extensions and `CPN` Conversion. Service privileges extend the dependency model in [9], which is implemented using the architecture analysis and design language[1] (`AADL`). `AADL` is standardized by the society of automotive engineers and used to describe real-time systems. A dependency is modeled in [9] with the white components in Fig.1. We add the properties in gray brackets to this dependency. A dependency is represented with a connection through which a dependent service *requires* data access from an antecedent service. We extend the data access concept by adding three properties to represent the trust relationship. The first property (*Reqs*) applies to the *requires* interface of the dependent

[1] `http://www.aadl.info/`




```
system ⟨DpS⟩ --Dependent service--
  ⟨Iface⟩:requires data access ⟨D_Type⟩
                {Reqs => access
                    [&i[Pri ∨ Cri]]*;};
system ⟨AtS⟩ --Antecedent service--
  ⟨Iface⟩:provides data access ⟨D_Type⟩{
    Trust => access [&iTri]*;
    Shgs => access [&iPri]*;};
system dependency_Model
 connections -- Toplogy description --
  data access ⟨AtS⟩.⟨Iface⟩->⟨DpS⟩.⟨Iface⟩;
```

Fig. 1. Dependency representation in AADL

service. It specifies the credentials and/or privileges delegated by this service. The two last properties apply to the *provides* interface of the antecedent service. The *Trust* property specifies the trust relationships which must be satisfied by the dependent service. The *Shgs* (i.e. privilege sharings) property specifies the privileges shared with the dependent service after the dependency is satisfied. The listing in Fig.1 illustrates the resulting AADL textual representation.

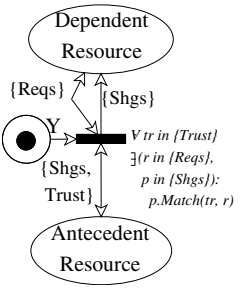


Fig. 2. CPN dep

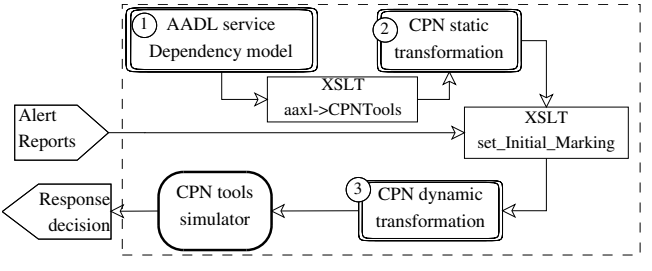


Fig. 3. Framework architecture

We transform the AADL dependency into the CPN transition in Fig.2. This transition is activated only when the trust relationships implemented by the antecedent service apply to the credentials and/or privileges delegated by the dependent service. This is implemented through the activation constraint added to the CPN transition. The dependency, when enabled, shares the privileges 'Shgs' between the dependent and the antecedent services. Replacing all AADL dependencies provides a CPN representation of the dependency model.

Attacks, Infection and Propagation. The simulation of the CPN model illustrates the normal system behavior in absence of threats. Intrusions modify the state of the system by modifying the initial marking of the CPN model. The simulation of the modified (i.e. dynamic) CPN model traces the system behavior in the presence of users and attackers. We model privilege revocation by omitting the revoked tokens from the initial marking of the CPN model. Transitions that use these tokens are disabled, which consequently implements statements 2 and 3 in listing 1.1. We also define tokens as data structures to which we add

a boolean variable. It is set to true when the appropriate privilege is infected by an intrusion. The infection propagates through a service dependency when the privileges and/or credentials delegated by the dependent service are all infected. The resulting effect is the infection of the privileges yet shared with the dependent service. We add to each CPN transition an action which is executed by the CPNTools simulator [11] after the dependency is enabled. It updates the infection status of the 'Shgs' tokens, and thus implements statements 1 and 4.

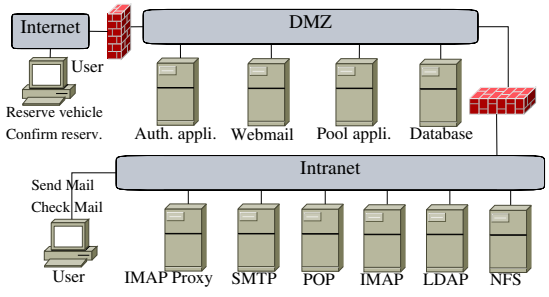
The framework in Fig.3 summarizes the impact assessment process. It uses the extended AADL model and intrusion alerts. We rely on existing correlation techniques [5] and these are out of the scope of this paper. The XSLTs transform the AADL model into a dynamic CPN model. The dynamic CPN model is simulated using the CPNTools simulator. The use of datalog formalism guarantees the simulation to converge in a polynomial time. The final marking of the CPN model characterizes the attack impact. All infected tokens (boolean attribute set to true) are aggregated to obtain attacker privileges. Tokens no longer held by user places are aggregated to obtain availability impacts. The final measures are used by the response module in order to decide about response selection.

5 Use-Case and Simulation Results

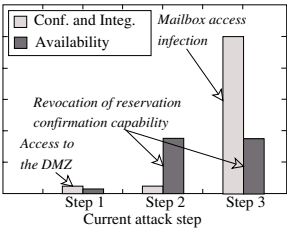
This section implements our proposal using the example of a vehicle reservation service. It describes the service platform and shows its appropriate CPN representation. Later it proposes some attack samples and demonstrates the use of our approach in order to assess the impact of these attacks.

5.1 Use-Case Architecture: Modeling Normal Behavior

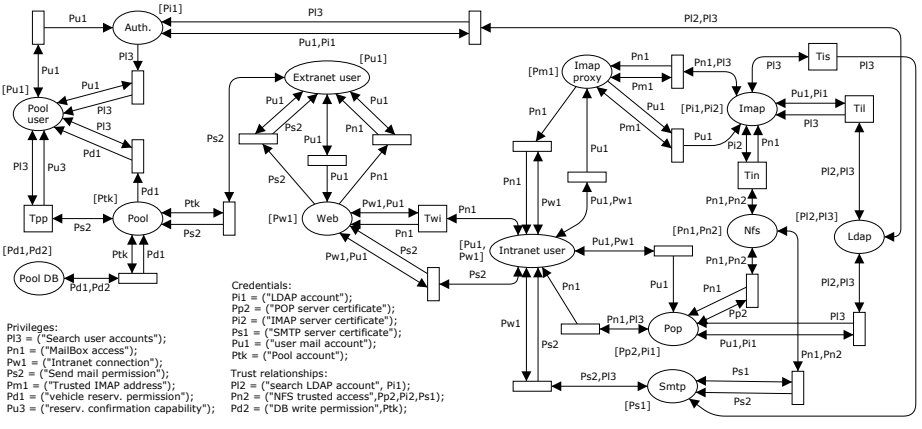
The vehicle reservation service is accessible for registered users who authenticate through a front-end application. User authentication is centralized using an Intranet directory service. The Pool application uses a database which contains information about the stock of vehicles. It also sends confirmation emails using a webmail application. The use case also includes intranet and extranet email users. Intranet users use POP and IMAP protocols, and extranet users use the webmail application. The webmail service is configured as in [9]. The SMTP service is configured for IMAP authentication before SMTP access. Figure 4a shows the appropriate network configuration and user requirements. The CPN provided by the XSLT is shown in Fig.4c (please refer to [9] for details about the AADL model). Privileges, Credentials and trust relations are also summarized in Fig.4c. Due to space limitations, we demonstrate the implementation of only the IMAP-LDAP dependency (Til). Other examples are easily verifiable in Fig.4c. LDAP authentication requires the IMAP service to have acquired valid user credentials (Pu_1). The IMAP service also needs a valid LDAP account (Pi_1) to search user accounts. This account must be trusted by the LDAP service, and granted the suitable authorizations. This is specified in the *slapd.conf* file for the LDAP service and modeled by the token Pl_2. This dependency shares the search right on user



(a) Use-case network architecture



(b) Simulation measures



(c) Use-case `CPN` representation

Fig. 4. Use-case presentation

accounts (Pl_3) between the two services. It enables the `IMAP` service to authenticate users. The simulation of the `CPN` model shows the privilege sharing through service dependencies until users are granted appropriate privileges.

5.2 Attack Injection and Impact Assessment

We demonstrate the use of our framework using the following attack scenario. The first attack is a `BoF` which enables an attacker to execute arbitrary code on the web server. The second attack uses the web server as a stepping stone to forge a crafted command to crash the courier-authlib, providing restricted shell on the `IMAP` server. The third attack enables the attacker, through a `BoF`, to have root shell on the `IMAP` server. The diagram in Fig.4b illustrates the inline tracking of intrusion impacts. The first attack enables the attacker to acquire the privilege Pw_1. The impact does not propagate elsewhere because the transition Twi for the Webmail-`IMAP` dependency also requires the credential Pu_1 to be infected in order to propagate the infection. The second attack causes the courier-authlib to

crash. It denies access to the credential Pi_1 for the `IMAP` server. The transition Til for the `IMAP-LDAP` dependency is disabled, and consequently the transition Tis for the `IMAP-SMTP` dependency. The transition disabling propagates until the final user transition Tpp is disabled, thus denying the privilege Pu_3 for pool users. The impact of this attack is considerable since Pu_3 corresponds to a high security objective (Pool users can no longer confirm their reservations). The third attack enables the attacker to access the `IMAP` server certificate for the `NFS` server. Pi_2 is infected, which causes the infection to propagate through the transition Tin for the `IMAP-NFS` dependency. The privilege Pn_1 is thus infected. It is a critical security objective because the attacker accesses user mailboxes.

6 Related Work

Impact assessment techniques for intrusion response may be classified into reactive and pre-emptive approaches. Studies in the first category use service dependency trees or graphs to propagate impacts. For instance, [1] models host-based systems as dependency graphs where nodes are components and edges are dependencies. This approach lacks the formalism required to model different intrusive states for a component. The failure of a component thus provokes the failure of all its dependent components. A similar approach in [19] applies to network intrusion response. It uses dependency trees and defines a capability function to propagate impacts. This approach does not compare response impacts to those of intrusions. It does not model scenarios where the best response option is not to respond. [6] extends the approach in [19] by using dependency graphs. It uses dependability analysis techniques, and thus only considers availability impacts. [10] also uses dependency graphs. It weighs dependencies by matrices rather than scalars, and thus models the interference between different impact propagations. Meanwhile, it lacks the ability to constrain impact propagations to some intrusive states for an antecedent component. The approach in this paper provides a solution to this problem using privileges, thus propagating impacts only when infected privileges enable such propagations. Pre-emptive response draws dependencies between elementary exploits [8,17] or vulnerabilities [3,7] rather than services. It provides techniques to evaluate attacker intentions and to adjust cost measures accordingly. It assigns costs metrics to each node in the exploit or vulnerability graphs, but only relies on expert knowledge to implement those metrics because these graphs do not model service operational dependencies.

7 Conclusion

This paper provided a systematic approach to assess intrusion impacts. It formally defines service dependencies which are further used as frames to propagate intrusion impacts. It overcomes the limitations of using dependability analysis techniques by adding privileges to the dependency model. It proves that adding privileges to the dependency model enables the evaluation of confidentiality and

integrity impact propagations in addition to availability. Privileges also constrain the occurrence of some impact propagations to the satisfaction of multiple conditions such as attacker privileges and service configurations. The approach in this paper is implemented as a simulation framework using colored petri nets.

Future work will extend the approach in this paper by adding the reaction dimension. A response either revokes or assigns some privileges. It is analyzed according to its effects on attackers (hampering infection propagations) and legitimate users (privilege revocations). Attack correlation is also possible using our model. An intrusion impact would not be restrained to the direct effects in terms of privilege infection and revocation. It will be analyzed according to the new attack steps which are made possible by the current intrusion success.

References

1. Balepin, I., Maltsev, S., Rowe, J., Levitt, K.: Using specification-based intrusion detection for automated response. In: Vigna, G., Jonsson, E., Krugel, C. (eds.) RAID 2003. LNCS, vol. 2820, pp. 136–154. Springer, Heidelberg (2003)
2. 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 Telecom 61, 197–217 (2006)
3. Dacier, M., Deswarte, Y., Kaâniche, M.: Quantitative assessment of operational security: Models and tools. LAAS Research Report 96493 (1996)
4. 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 (2007)
5. Debar, H., Wespi, A.: Aggregation and correlation of intrusion-detection alerts. In: Lee, W., Mé, L., Wespi, A. (eds.) RAID 2001. LNCS, vol. 2212, pp. 85–103. Springer, Heidelberg (2001)
6. Jahnke, M., Thul, C., Martini, P.: Graph based metrics for intrusion response measures in computer networks. In: 32nd IEEE Conf. LCN (2007)
7. Jajodia, S., Noel, S.: Topological vulnerability analysis: A powerful new approach for network attack prevention, detection, and response. In: Algorithms, Architectures, and Information Systems Security (2007)
8. Kanoun, W., Cuppens-Boulahia, N., Cuppens, F., Dubus, S., Martin, A.: Success likelihood of ongoing attacks for intrusion detection and response systems. In: Inter'l Conf. on Computational Science and Engineering (2009)
9. 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. 176–195. Springer, Heidelberg (2009)
10. Kheir, N., Debar, H., Cuppens-Boulahia, N., Cuppens, F., Viinikka, J.: Cost assessment for intrusion response using dependency graphs. In: Proc. IFIP Inter'l Conf. on Network and Service Security (2009)
11. Kristensen, L.M., Christensen, S., Jensen, K.: Practitioner's guide to colored petri nets. Inter'l Journal on Software Tools for Technology Transfer (1998)
12. Li, N., Mitchell, J., Winsborough, W.: Design of a role-based trust-management framework. In: Proc. IEEE Symp. Security and Privacy, p. 114 (2002)
13. Noel, S., Jajodia, S., O'Berry, B., Jacobs, M.: Efficient minimum-cost network hardening via exploit dependency graphs. In: Proc. 19th Conf. ACSAC (2003)

14. Papadaki, M., Furnell, S.: Informing the decision process in an automated intrusion response system. Information security Tech. Report, pp. 150–161 (2005)
15. Sandhu, R.S., Coynek, E.J., Feinsteink, H.L., Youmank, C.E.: Role-based access control models. IEEE Computer 29, 38–47 (1996)
16. Sheyner, O., Wing, J.: Tools for generating and analyzing attack graphs. In: Proc. Wkshp on Formal Methods for Components and Objects (2004)
17. Stakhanova, N., Basu, S., Wong, J.: A cost-sensitive model for preemptive intrusion response systems. In: Proc. 21st Inter'l Conf. AINA, pp. 428–435
18. Stakhanova, N., Basu, S., Wong, J.: A taxonomy of intrusion response systems. Inter'l Journal information and computer security, 169–184 (2007)
19. Toth, T., Kruegel, C.: Evaluating the impact of automated intrusion response mechanisms. In: Proc. 18th Conf. ACSAC (2002)