A Mobile Ambients-based Approach for Network Attack Modelling and Simulation

Virginia N. L. Franqueira*, Pascal van Eck, Roel Wieringa
University of Twente
Enschede, The Netherlands
Email: {franqueirav, p.a.t.vaneck, r.j.wieringa}@ewi.utwente.nl

Raul H. C. Lopes
Brunel University
London, England
Email: raul.lopes@brunel.ac.uk

***Abstract*—Attack Graphs are an important support for assessment and subsequent improvement of network security. They reveal possible paths an attacker can take to break through security perimeters and traverse a network to reach valuable assets deep inside the network. Although scalability is no longer the main issue, Attack Graphs still have some problems that make them less useful in practice. First, Attack Graphs remain difficult to relate to the network topology. Second, Attack Graphs traditionally only consider the exploitation of vulnerable hosts. Third, Attack Graphs do not rely on automatic identification of potential attack targets. We address these gaps in our MsAMS (Multi-step Attack Modelling and Simulation) tool, based on Mobile Ambients. The tool not only allows the modelling of more static aspects of the network, such as the network topology, but also the dynamics of network attacks. In addition to Mobile Ambients, we use the PageRank algorithm to determine targets and hub scores produced by the HITS (Hypertext Induced Topic Search) algorithm to guide the simulation of an attacker searching for targets.**

***Index Terms*—Network Security, Vulnerability Assessment, Attack Modelling, PageRank, HITS.**

I. Introduction

A computer network is an ever changing environment. New business agreements trigger changes in firewall rules. New network functionalities trigger the configuration of new servers, new network services, and new users increasing the chance of introducing mis-configurations in the network. Additionally, patches are not always available and, even when they are, it may not be cost-effective to patch all vulnerabilities present in a network. Hence, a network is hardly free from opportunities for attackers, and needs to be assessed constantly.

Attackers take advantage of reachable vulnerabilities in COTS (Commercial-Of-The-Shelf) and open source software components and of exposures [1] as stepping stones to penetrate a network. Each step opens further opportunities by exposing more hosts, and attackers can progress this way until targets are reached. Early Model Checker approaches [2], [3] suffered from severe scalability problems due to the state explosion problem [4]. Since then, Attack Graphs algorithms have evolved to exploit-based approaches which take advantage of dependencies between vulnerabilities, later simplified by the access-to-effect paradigm [5], [6]. Exploit-based approaches are supported by the assumption of monotonicity [7] which means that once a resource is acquired by an attacker it is never released. Several customised Attack Graph algorithms by researchers [5]–[11] and commercial initiatives [12] have been proposed, some scaling to thousands of hosts [5], [10]. However, although scalability is no longer the *main* issue, there are three other areas where improvement is still needed:

1) Attack graphs are still difficult to understand by people since they do not fully represent the network topology needed to relate attack paths identified in the graph to the network itself, and to support decisions about countermeasures. Approaches to this problem rely on Aggregation [13], [14] or Clusterization [15] of graph nodes, but these approaches still suffer from the problem that firewalls are only used for calculation of reachability and not clearly represented in the graph. Therefore, if several firewalls are traversed by an attacker it may be difficult, e.g., to identify which ones should be changed.
2) Steps in an attack graph are typically generated by matching post- and preconditions of subsequent attack steps [5]–[11] but (i) acquisition, movement or replication of resources cannot always be represented in terms of pre/postcondition pairs, and (ii) pre/postcondition pairs are memoryless whereas attackers may gain access due to resources acquired more than one step ago. There is a need for attack dynamics.
3) Some algorithms to generate attack graphs consider all possible attack paths. Hence, it is if every node in the network would be a potential target [5]. Other algorithms require the explicit indication of targets, either by naming a specific target [10], [12] or by assuming that asset values are given [16]. In line with this last approach, we assume, like other researchers (e.g. [17]), that attackers are rational and search for assets which represent some value. However, business valuation of assets is a complex, time-consuming process. Therefore, for large networks, asset values are usually not available.

From these observations we derive a list of requirements we would like to address in our proposed solution:

R_1 The network topology should be fully represented in the

*Supported by the research program Sentinels (www.sentinels.nl), under contract 06679

[1] A vulnerability is a mistake in software which hackers can use directly to access protected data, while an exposure provides information or capabilities that can function as stepping-stones for direct access to protected data [1].

attack graph.

R_2 The algorithm should allow for attack dynamics.

R_3 The algorithm should make reasonable automatic estimation of which network nodes are targets.

A. Contribution

We address these current deficiencies in Attack Graphs by proposing MsAMS, a tool for modelling and simulation of network attacks, the design of which draws heavily on Cardelli's work on *Mobile Ambients* [18], [19] and formal biology [20], and on Milner's work on bigraphs [21]. Specifically, we address R_1 and R_2 by applying the concept of Mobile Ambients to the domain of networks, and R_3 with Google's PageRank algorithm [22].

We have chosen *Ambient*s because they allow the representation of a network as a graph of nested nodes. They also allow the representation of any type of resources, such as firewalls, routers, hosts, services, vulnerabilities, privileges, users, attackers, and credentials. This way, we are able to fully represent the topology of a network since hierarchy and grouping is intrinsic to Mobile Ambients. Ambients have capabilities which allow them to move. Furthermore, ambients can interact with other ambients depending on their capabilities. These two features allow the representation of attack dynamics without compromising scalability. Finally, by replacing asset value by asset connectivity we are able to define automatically a set of targets without relying on valuations of all assets in a network, which is not readily available, as we observed above. PageRank algorithm returns high authority scores for graph nodes with many inlinks. If a network node with high number of inlinks is compromised it may affect a high number of other nodes which depend on it. Therefore, based on this rationale, we assume that high authority nodes are network nodes to be protected, i.e. they are targets.

In this paper, we extend [23] in many ways. We review the modelling of the running example, introduce a new example, and provide more details on how we achieve requirements R_1, R_2 and R_3, such as, how we capture network locality and connectivity (Section VI), how virtual links are processed (Section VIII), and how we use ranking algorithms (Section IX).

II. Overview of MsAMS (Multi-step Attack Modelling and Simulation)

MsAMS is a tool which requires as input (i) the network configuration, including filtering rules, (ii) vulnerabilities in COTS present in the network, which can be obtained automatically from vulnerability scanning tools, (iii) their attributes, which can be obtained from vulnerability databases such as the National Vulnerability Database (NVD) [24], and (iv) the location of the attacker (e.g. inside or outside the network). Additionally, and at the discretion of the network administrator, Access Control Lists (ACLs) from services can also be used, to assess potential attacks which exploit credential theft and trust relationships. These input allow the tool to build an ambient-based model of the network. After the model is complete, MsAMS simulates an attacker (also an *Ambient*) dynamically acquiring resources and searching for attack paths allowed by the modelled ambients and their embedded rules. Therefore, MsAMS produces attack traces which represent possible multi-step attack paths, as output.

III. Conceptual Model of a Network

We borrow from the concept of Mobile Ambients, which is a calculus that allows us to define places (i.e. ambients) where computation happens and to express movement of processes [18]. We view a network as an *Ambient* which contains other *Ambient*s i.e hosts, subnets and firewalls, which recursively may contain other *Ambient*s. Therefore, a subnet is an ambient which contains several other ambients representing hosts; a firewall is also an ambient which protects ambients by filtering communication between ambients outside its boundaries and protected ambients contained within its boundaries. A host contains interfaces which allow interactions with other hosts, internal or external to the network. Interfaces may be ports allowing access to services, or application interfaces, such as login to the Operating System (OS) or web browsers; these interfaces may contain vulnerabilities. According to previous study of the NVD [25] only an insignificant percentage of vulnerabilities require credentials, hence, we assume that vulnerabilities represent an opportunity for attackers to enter a host without the need for credentials (e.g. password or private session key). However, some interfaces require themselves credentials, e.g., SSH service and OS login.

We use a simple vulnerability model based on access required for its exploitation and effect resulting from its successful exploitation. Thus, the access can be either of the type "network" which means the vulnerability can be exploited remotely, opposed to the type "local" which means the vulnerability can only be exploited if the attacker is authenticated, via an interface, on the host. The effect of a vulnerability can be of the type Privilege Gained (i.e. "user" or "admin" privilege over the OS) or Impact. In this paper, we restrict ourselves to the use of vulnerabilities which result in privilege acquisition.

We use exposures to represent stealthy ways to acquire credentials. An attacker can get remote or local access to a host by means of vulnerabilities but, most of the time, he does not automatically obtain credentials for that host. Thus, an exposure is an abstraction to model the availability of credentials by means e.g. of social engineering, clear-text passwords saved locally, or via key stroke mechanisms. A credential obtained from an exposure in one host may allow an attacker to further access non-vulnerable hosts in the network.

IV. Abstracting a Network as Ambients

As defined by Cardelli [18]–[20] an ambient has a name, a list of ambients contained within it, and a list of processes running in it. Therefore, an ambient could contain non-ambient processes and sub-ambients. We simplify this approach for the domain of network attacks by considering that an ambient only contains sub-ambients and each (sub-)ambient may contain

a list of processes running on its boundaries which execute actions. A process may execute (i) movement actions, (ii) communication actions, (iii) resource-acquisition actions, and (iv) replication action. We consider that each action executed on an ambient provides the ambient with a capability, which happens at the level of ambient, not at the level of process. Thus, although in Cardelli's work only movement actions are regulated by capabilities, we take that all actions are regulated by capabilities, and actions are always inter-ambients. Besides, there are action-rules which define how the execution of actions should happen; by default all actions are executed in parallel and only once, otherwise when specified, they can be executed sequentially (called "paths" in [19]) and repeatedly. We define *Ambient* as follows.

Definition 1: An ambient named Amb is defined as **Amb:[AmbientList][ActRuleList]**, where $AmbientList$ is a list of ambients, and $ActRuleList$ is a list of action-rules executed in parallel at the boundaries of Amb.

Definition 2: An action-rule $ActRule$ is an expression of the following form.

1) **Repeat** Act: this action-rule repeats action Act indefinitely.
2) **Seq** Act_i Act_j: this action-rule performs Act_i followed by Act_j.

Definition 3: An action Act is an expression of the following form.

1) Movement Actions
   a) **Enter** Amb_i: an ambient Amb with this capability is able to enter in ambient Amb_i; potentially acquiring access to ambients contained in Amb_i.
   b) **Accept** Amb_i: an ambient Amb with this capability is able to accept the entry of Amb_i in its boundaries; potentially allowing Amb_i to acquire access to ambients contained in it.
   c) **AllowIn** Amb_i Amb_j: an ambient Amb with this capability allows that Amb_i moves through its boundaries to gain access to Amb_j.
2) Communication Actions
   a) **Out** Amb_i: an ambient Amb with this capability is able to send messages/requests to ambient Amb_i.
   b) **In** Amb_i: an ambient Amb with this capability is able to respond to messages/requests from ambient Amb_i.
3) Resource-Acquisition Actions
   a) **ReleaseCred** Amb_i: an ambient Amb with this capability is able to release the credential represented by ambient Amb_i.
   b) **AcquireCred** Amb_i: an ambient Amb with this capability is able to request acquisition of the credential represented by ambient Amb_i.
4) Replication Action
   a) **Replicate**: an ambient Amb with this capability is able to produce one replica of itself, generating another ambient $Amb\prime$ identical to ambient Amb.

Note that Cardelli's primitive capabilities, "in" (corresponds to an Enter Amb), "open", "exit", and later "accept" [20] can be used to derive composed capabilities, such as "allow in", "acquire" and "release" [18]. Thus, *AllowIn* is derived from "in" which causes active ambients to move, plus "open" which dissolves Amb from the outside revealing its content. We adapted Cardelli's "acquire" and "release" which in his work is derived from "open" to the domain of network attacks. As we will see on Section XI, our "AcquireCred" and "ReleaseCred" is a composition of "Enter" and "Accept". Note also that we did not identify the need for capabilities "open" and "exit" yet, that is why it was not incorporated into MsAMS. Thus, we assume that exit of an ambient is by default permitted, and that ambients' boundaries are never dissolved.

A. Matching Capabilities

The actions which potentially allow movement, communication and resource-acquisition (described above) will only happen if a match between *Capabilities* occur. Similar to ambients applied to biology [20], these actions require synchronisation between two ambients, in our case, determined by a common ambient name. This synchronisation is achieved by means of reduction rules between: (i) Enter and Accept, (ii) Out and In, (iii) ReleaseCred and AcquireCred, and (iv) Enter and AllowIn. Fig. 1 illustrates an Enter/Accept reduction rule resulting in ambient m successfully entering inside ambient n. We use our notation and a graphical notation inspired by BioAmbients [20].

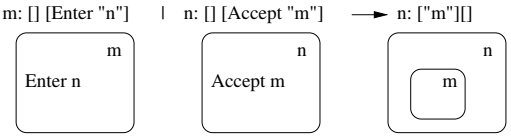


Fig. 1. Enter/Accept reduction rule which allows ambient move

V. Running Example

We use the network illustrated in Fig. 2 from Ingols et al. [5] as the basis for introducing core concepts and the method used by the MsAMS tool.

In this example network the attacker is initially located on host A and wants to reach either host E or F. The firewall only allows traffic from host C or D to host E. Additionally, all hosts have a single open port with a vulnerable service running. Each vulnerability is remotely exploitable and allows the attacker to gain privileged access to the host.

The example network can be represented in terms of *Ambient* as illustrated in Fig. 3(a).

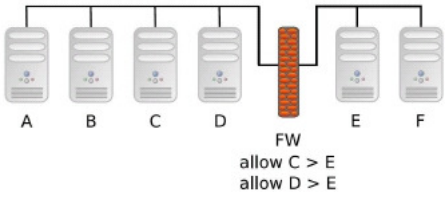
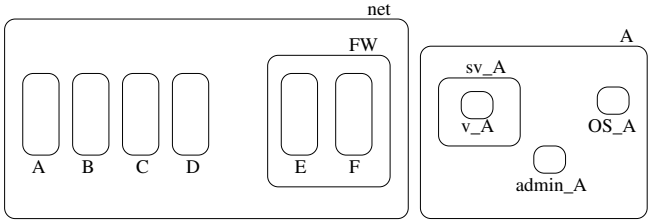


Fig. 2. An example network

The figure shows ambient net, containing five ambients A, B, C, D, FW, which represent hosts A to D and firewall FW. The firewall is viewed as a *membrane* protecting ambients, i.e. hosts E and F. Fig. 3(b) provides a zoom view of host A, which contains an ambient representing a listening service sv_A, which in turn contains an ambient representing a vulnerability v_A on that service. Additionally, ambient A contains (i) an ambient $admin\_A$ representing privilege of root (unix-based hosts) or administrator (windows-based hosts), and (ii) an ambient OS_A representing the host OS or kernel. The choice of entities to represent depends on what is relevant to model. For example, in this case v_A is a vulnerability of the type remote-to-admin, that is why $admin\_A$ is relevant.



(a) The example network (b) Zoom in host A

Fig. 3. Modelling the example network as *Ambient*s

VI. Capturing Network Locality and Connectivity

This section describes how we fulfil requirement R_1.

The topology of the example network illustrated as *Ambient*s in Fig. 3 is now represented in a tree structure, shown in Fig. 4. It defines the locality of ambients in Milner's terminology [21], henceforth called *Locality Tree*. Children nodes of ambients $B-D$ and F, although not fully represented in the figure, are similar to A.

The connectivity of the network defines a hypergraph $H = \{N, E\}$, the *Connectivity Hypergraph*, where N is the set of nodes:

$N = \{net, A, B, C, D, E, F, FW, sv\_A, admin\_A, ...\}$

and E is the set of hyperedges:

$E = \{e_1 = \{net, A, B, C, D, FW\}, e_2 = \{E, F\},$

$e_3 = \{A, sv\_A, admin\_A, OS\_A\}, e_4 = \{v\_A\}, ...\}$

Hyperedges referring to hosts B, C, D, E and F have been omitted because they are similar to e_3 and e_4.

Note that the nesting of nodes, i.e. of *Ambient*s, is captured via the Locality Tree, while each hyperedge represents fully-connected environments.

VII. Capturing Network Dynamics

This section describes how we fulfil requirement R_2.

We have seen in Section V the network topology of the example network, i.e the ambients locality, and the hypergraph corresponding to the connectivity of the network. So far, we have addressed mostly the static aspect of the network. Now we specify the *Ambient*s with their action-rules which determine the dynamic behaviour of the ambients, how they can interact. The $ActRuleList$, as defined in Section IV, is a list

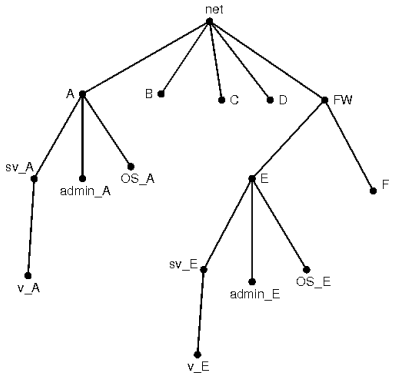


Fig. 4. Locality Tree for example network shown in Fig. 3

of action-rules executed as parallel compositions. Therefore, an $ActRuleList$ of the type [Repeat Act_i, Repeat Act_j] means repeat Act_i indefinitely and repeat Act_j indefinitely.

```
1  net: ["A" "B" "C" "D" "FW"] []
2  FW: ["E" "F"] [Repeat (AllowIn "C" "sv_E"),
       Repeat (AllowIn "D" "sv_E")]
3  A: ["sv_A" "admin_A" "OS_A"]
      [Repeat (AllowIn "net" "sv_A")]
4  sv_A: ["v_A"] [Repeat (Accept "net"),
                  Repeat (Out "OS_A")]
5  v_A: [] [Repeat (Accept "sv_A")]
6  admin_A: [] [Repeat (Accept "v_A"),
                Repeat (Enter "OS_A")]
7  OS_A: [] [Repeat (Accept "admin_A"),
             Repeat (In "sv_A")]

similar rules as 3-7 apply to ambients B-F
```

Rule 1 defines that ambient net contains ambients A, B, C, D and FW, but no action-rules.

The *Capability AllowIn* used in the second rule captures the firewall rules, restricting the traffic of messages from outside to inside its boundaries. In the example, the firewall allows only that ambients coming from hosts C and D access the service in host E, i.e. sv_E.

Rule 3 defines that host A allows any traffic from net to its service sv_A. The capability AllowIn in this case performs the role of a port which gives access to its service. We can also think that it represents a host-based firewall governing traffic allowed into and out of the host.

As we have seen, host A contains a listening service sv_A which contains a vulnerability v_A. This service accepts ambients from net into its boundaries meaning that the service accepts requests from the net, and possibly from the internet if we had represented it here. Service requests give the opportunity of exploiting v_A. That's why v_A accepts sv_A, meaning that once in sv_A, vulnerability v_A becomes available, as specified in rules 4 and 5.

Rule 4 also defines that service sv_A can make requests to OS_A, which represents, e.g., the kernel of a Linux system and all services it can provide for someone with admin privileges over A. Thus, on the one hand service sv_A can make requests, represented by its capability *Out "OS_A"* and, on the other hand, OS_A can answer service requests coming from service sv_A, represented by capability *In "sv_A"* in

rule 7.

The ambient $admin\_A$ represents the privilege of admin (root or administrator) acquired over the host. The meaning of this privilege is evident by the fact that an ambient in $admin\_A$, e.g. an attacker, can *Enter "OS_A"* (rule 6) and OS_A accepts it (rule 7), allowing the attacker to take full advantage of host A OS.

All the other hosts have similar rules as 3-7 specified for host A, including hosts E and F. Hence, all hosts have one *AllowIn "net"* on their service action rules. These rules characterise the network behaviour, i.e. all network traffic, for the example network. Note that the action-rules for ambient FW come from the firewall rules, and can be retrieved automatically. Other ambients follow some patterns which can be duplicated (automatically). For example, all services containing the same type of vulnerability (e.g. remote-to-admin) are defined the same way, and this information can be retrieved from scanning tools and the NVD. Hence, in practise, the majority of the ambients can be specified automatically, and the network administrator has only to specify a few critical servers manually.

VIII. Processing Virtual Links

So far, we have seen, in Section VI, how we capture a network topology via (i) locality tree, and how we capture network connectivity via (ii) connectivity hypergraph. Besides, we have also seen how the dynamics of the network are specified in terms of (iii) ambients action rules in Section VII. In this section, we describe how we capture what we call *Virtual Links* from (i) and (iii), introducing the concept of *Least Common Ancestor*. Note that the computation of these links allows us to build a matrix of links, as described in Section IX.

Definition 4: There's a directed **Virtual Link** from Amb_i to Amb_j when: (i) Amb_j has an Accept "Amb_k" where Amb_k is an ancestor of Amb_i, and (ii) there is an *Allow path* letting Amb_i into Amb_k.

An *Allow path* letting Amb_i into Amb_k is a path that would allow Amb_i to exit to the Least Common Ancestor of Amb_i and Amb_k, and let it enter through successive firewalls into Amb_k. Note that, as mentioned in Section IV, "exit" to an ambient is by default permitted, and currently not incorporated into MsAMS.

Definition 5: **Least Common Ancestor** of two ambients $lca(Amb_i, Amb_k)$ is the first ancestor that ambients Amb_i and Amb_k have in common on the Locality Tree.

For example, according to Fig. 4, we have: $lca(sv\_E, F) \Rightarrow FW$ and $lca(v\_A, admin\_E) \Rightarrow net$.

A virtual link between ambients X and Y is created if X can actually move into Y. Let's take as an example the firewall FW. Although sv_E accepts traffic from net, meaning that potentially an ambient coming from $A - D$ can reach sv_E, the firewall restricts this possibility to ambients coming from C or D. Hence, we have in fact two virtual (directed) links $C \rightarrow sv\_E$ and $D \rightarrow sv\_E$. The following algorithm processes Virtual Links of a modelled network, according to this rationale.

```
for each Y
  for each ActRule in Y: Accept "X" or In "X"
    follow the path from Y to lca(X,Y),
    and test if X is allowed in through
    each node in the path
```

IX. Computing Ranks using the Matrix of Network Links

This section shows how we fulfil requirement R_3.

So far, we can represent an attacker as an *Ambient* that can travel through the network according to action-rules. However, the attacker up to now moves at random, only bound by permitted moves. In this section, we describe how we determine targets automatically and how we calculate authority and hub scores. This way, we are able to incorporate rationality to attackers moves by guiding their search toward valuable assets (i.e. targets) with preference to lower cost moves (i.e. high hub scores) when more than one move is possible. More in detail, we borrow from Link Analysis Ranking [2] for two tasks which support the simulation of attackers, described in Section X:

1) We use Google's PageRank algorithm [22] to identify a set of target nodes. Large authority scores returned by the algorithm represent network nodes with large number of inlinks, i.e. nodes that will affect a large number of other nodes if compromised. We assume these nodes should be protected, and thus, represent targets for attackers. Note that our virtual links, described in the previous section, are directed links. Hence, the notion of inlinks and outlinks apply to them as it happens with webpages.
2) We use HITS (Hypertext Induced Topic Search) algorithm [26], basis of Teoma search engine, to compute scores used for searching for attack paths. HITS relies on the assumption that a webpage with many inlinks has a high authority score and a webpage with many outlinks has a high hub score. Besides, each page is an authority and a hub to a certain extent, and as a consequence, each page has both scores. It is further assumed that "Good authorities are pointed to by good hubs and good hubs point to good authorities" [27, Page 115]. We take advantage of HITS scores to simulate a rational attacker giving preference for hubbiest steps, whenever alternative moves are available.

From the virtual links obtained as shown in the previous section, we create an adjacency matrix L where L_{ij} is one, if there is a link from ambient i to ambient j, and zero, otherwise. This is a $n \times n$, where n is the number of ambients modelled, which is sparse since we only represent links which are enabled via capabilities and locality, and do not represent links resulting from connectivity.

[2]This field of research deals with the prioritization of search results using the link structure of web pages.

A. *PageRank for Computing Targets Automatically.*

The PageRank value (PR) of an ambient Amb_i is proportional to the sum of PR values of its inlinking ambients Amb_j. The $\vec{PR}$ is obtained efficiently via power method [27, Chapter 4] applied to the matrix G, as shown in (1).

$$\vec{PR}^{(k+1)T} = \vec{PR}^{(k)T} G, \quad where\ G = \alpha H + (\alpha\vec{a} + (1-\alpha)\vec{e})1/n\vec{e^T} \tag{1}$$

Thus, matrix G is computed by means of the sparse matrix $|n| \times |n|$ of links H where H_{ij} is $\frac{1}{|Amb_i|}$ if there is a link from ambient i to ambient j and zero, otherwise. Note that matrix H has the same structure as matrix L (as seen above), but non-zero values are different; in L non-zero elements are ones, while in H non-zero elements are probabilities. The parameter $\alpha \in [0, 1]$ is the damping factor, which conveys the idea of *random walk*. The damping factor α for an ambient-based graph still represents this notion. Thus, α is the probability that the attacker will follow one of the outlinks from the present node, $1 - \alpha$ being the probability that the attacker abandons or starts the attack over again to follow another previously unexplored path.

Vector $\vec{a}$ contains one if ambient i is a dangling node, i.e. if it contains no outlinks, and zero otherwise. It corrects dangling ambients (nodes) by given $\frac{1}{|n|}$ equal probability that any ambient is selected from it. Vector $\vec{e}$ is a column vector of ones, $\vec{e^T}$ is the transpose of vector e, n is the number of ambients while PR^T is a row vector containing the PageRank scores, after convergence.

The PageRank vector for the running example illustrated in Fig. 3 is obtained after 13 iterations ($\alpha = 0.6$). From this vector, we select t ambients with the higher scores for the target set. Note that the top two scores ($t = 2$) are OS_E (0.05820134) and $admin\_E$ (0.06054866), which correspond to the intuition of asset value since the compromise of host E turns impossible the communication from net to the hosts protected by the firewall.

B. *HITS for Computation of Hubs.*

Authority $\vec{x}^k$ and hub $\vec{y}^k$ scores, used to simulate a rational attacker, are calculated using (2) and (3), respectively, where IN is the set of inlinks of ambient Amb_i, OUT is the set of outlinks of Amb_i, and k is the iteration counter.

$$\vec{x}^k(Amb_i) = \sum_{Amb_j \in IN_{Amb_i}} \vec{y}^{(k-1)}(Amb_j) \tag{2}$$

$$\vec{y}^k(Amb_i) = \sum_{Amb_j \in OUT_{Amb_i}} \vec{x}^k(Amb_j) \tag{3}$$

The summations (2) and (3) are also resolved by power method [27, Chapter 11] applied to the matrix resulting from the multiplication of matrix L and its transpose L_T: $L^T L$ (called authority matrix) or LL^T (called hub matrix). Authority scores are obtained resolving (4) and hub scores are obtained resolving (5), where L is the matrix of zeros and ones containing the virtual links between every pair of ambients Amb_i and Amb_j, as described in Section VIII.

$$\vec{x}^k = \vec{y}^{k-1} L^T L \tag{4}$$

$$\vec{y}^k = \vec{x}^{k-1} L L^T \tag{5}$$

X. Simulation of Attackers

The simulation engine is in reality the execution of computing agents, i.e. ambients which *actively* move through the Locality Tree according to the Matrix of Network Links L, defined in Section VIII, and the Connectivity Hypergraph, defined in Section VI. These agents *search* for valuable assets (i.e. high PageRank scores) giving preference to lower cost moves (i.e. high hub scores).

An attacker *Ambient* can be assigned as a computing *Agent*, more precisely "they [agents] are confined to ambients" as defined by Cardelli [18]. A computation is run by executing in parallel a list of actions defined by the *Ambient* of each computing *Agent*. Thus, at each step, a computing *Agent* executes one *Action* (non-deterministic choice) defined by its action-rule list. Each of these steps can either be accepted, if the attacker (ambient) actions and the other ambients actions match, as described in Section IV-A, or rejected if the actions do not match. A match means that the attacker can actually perform the step, and this is recorded by the simulation engine as a move from the attacker. In the end, the engine provides the attacker complete trace up to a target. This trace is a possible multi-step attack on the modelled network. An attacker trace for the running example (see Fig. 3) is illustrated next.

```
Enter "sv_D"
Enter "sv_E" (through FW:[AllowIn "D" "sv_E"]
              through E:[AllowIn "net" "sv_E"]
              through sv_E:[Accept "net"])
Enter "v_E"
Enter "admin_E"
```

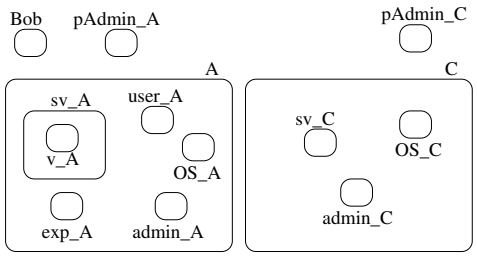
This trace shows the possible attack ADE. Note that the trace indicates if a firewall is traversed to facilitate relating the output path with the actual network path. Note also that vulnerabilities V_A and v_D were not exploited because the attacker had more incentive to look for vulnerability v_E which leads to $admin\_E$.

XI. Modified Running Example: Requesting/Acquiring Credentials

This section aims to show how the acquisition of credentials by an attacker happens. For this purpose, we now consider the running example, illustrated in Fig. 2, with hosts A and C modified. A has a vulnerability remote-to-user and has an exposure which reveals the admin password ($pAdmin\_A$) of host A for any ambient which enters it. Host C is no longer vulnerable, and it has a service running, let's say SSH, used to administer the host remotely. Administrator Bob is able to do so from host A. Fig. 5(a) and 5(b) illustrate these changes.

The modified host A is specified as follows.

```
1 A: ["sv_A" "user_A" "admin_A" "OS_A"
```



(a) Zoom in host A (b) Zoom in host C

Fig. 5. Modified running example shown in Fig. 2

```
      "exp_A"]
      [Repeat (AllowIn "net" "sv_A")]
2 sv_A: ["v_A"] [Repeat (Accept "net"),
                 Repeat (Out "OS_A")]
3 v_A: [] [Repeat (Accept "sv_A")]
4 exp_A: [] [Repeat (Accept "user_A"),
        Repeat (ReleaseCred "pAdmin_A")]
5 user_A: [] [Repeat (Accept "v_A"),
              Repeat (Out "OS_A"]
6 admin_A: [] [Repeat (Accept "pAdmin_A"),
               Repeat (Enter "OS_A")]
7 OS_A: [] [Repeat (In "user_A"),
            Repeat (In "sv_A"),
            Repeat (Accept "admin_A")]
8 Bob: [] [Repeat (Enter "pAdmin_A")]
9 pAdmin_A: [] [Repeat (Accept "Bob")]
```

And the modified host C as follows.

```
1 C: ["sv_A" "OS_C" "admin_C"]
     [Repeat (AllowIn "net" "sv_C")]
2 sv_C: [] [Repeat (Accept "pAdmin_C"),
            Repeat (Accept "pAdmin_A"),
            Repeat (Out "OS_C")]
3 admin_C: [] [Repeat (Accept "sv_C"),
               Repeat (Enter "OS_C")]
4 OS_C: [] [Repeat (Accept "admin_C"),
            Repeat (In "sv_C")]
5 pAdmin_C: [] []
```

The SSH service in host C (sv_C) now requires passwords represented by ambients $pAdmin\_C$ and $pAdmin\_A$ (rule 2), while before (as shown in Section VII) it accepted any ambient within ambient net.

If a computing agent, ambient $attacker$, happens to issue (at simulation time) an action rule $AcquireCred$ "$pAdmin\_A$", then a reduction rule between *ReleaseCred* and *AcquireCred* occurs, as illustrated in Fig. 6.

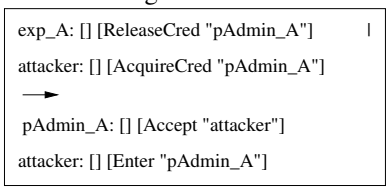


Fig. 6. Reduction rule between actions $ReleaseCred$ and $AcquireCred$

Note that the acquisition of *Action* Enter "$pAdmin\_A$" by the ambient $attacker$ allows the attacker to use credential $pAdmin\_A$ for the remaining of the computation, i.e. until the engine stops when a target is reached. This is an advantage over approaches which rely on pre/postconditions.

XII. Experimental Results

The time for computing an attack is dominated by the computation of assets' ranks and hub scores. This is performed by an algorithm based on the PageRank algorithm [27], and the query-independent HITS algorithm [26]. A naïve implementation of either PageRank or HITS can take $O(n^3)$, demanding a $O(n^2)$ matrix multiplication in each cycle. A more efficient implementation, however, takes into account the fact that the adjacency matrix is sparse and that the matrix multiplication performed in each cycle can be executed in $O(n)$. Assuming that n is the number of ambients represented, our implementation precomputes the matrix in $O(n^2)$, and then applies ranking algorithm in time that ranges from $O(kn)$ to $O(kn^2)$, depending on the density of the adjacency matrix and on k, the number of iterations necessary for convergence of the power method applied to the computation of either PageRank or HITS. It is important to notice that even for a matrix with billions of nodes the PageRank algorithm tends to converge in less than a hundred iteration. In our tests it converged in less than 60 cycles for a test with 8000 nodes. In a previous implementation [28] we used a full matrix multiplication and fixed k, obtaining running times of $O(n^3)$ when using more than 8000 nodes. Currently, we have an implementation in Haskell using a sparse matrix multiplication and a matrix akin to the Google matrix [27]. The whole process of both ranking (with HITS and PageRank) and searching for an attack executes in less than 30 seconds for a network with more than 8000 nodes.

We modified the running example illustrated in Fig. 3 for our experiments. Thus, we used the following configuration of nodes to the left and right of the firewall, respectively: (4,512), (8,1024), (16,2048), (32,4096), and (64,8192). That choice generates a dense adjacency sub-matrix for the part of the model representing the right side of the firewall. All experiments assumed the attacker positioned initially inside host A. Fig. 7 shows the computing time for these experiments, performed on machine with Intel Core 2 Duo T5250, 1.5GHz processor, 2GB RAM.

We express the network models input of our tool in a dedicated language that has also been implemented in Haskell. The 8256 nodes' network used in the experiments, e.g., is described in this language with just 46 lines. It takes 7.18 seconds to compile those lines into the internal representation used by PageRank and HITS algorithms.

XIII. Conclusion and Future Work

We presented MsAMS (Multi-step Attack Modelling and Simulation), a tool which implements Mobile Ambients applied to the domain of network attacks, and two Link Analysis Ranking algorithms: PageRank and HITS. MsAMS satisfies the three requirements identified in Section I, since (i) it allows capturing the exact topology of the network, fulfilling R_1, as seen in Section VI, (ii) it allows representing attack dynamics, fulfilling R_2, as seen in Sections VII, X and XI, and (iii) it determines network targets automatically, fulfilling R_3, as seen in Section IX. This is achieved without compromising

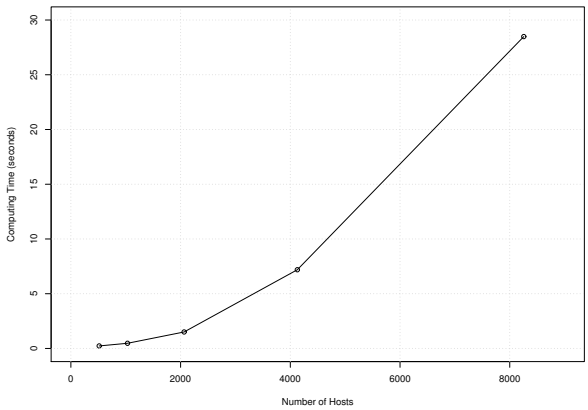


Fig. 7. Performance of the MsAMS tool

performance, as seen in Section XII. Besides, the approach is flexible since the level of details modelled is left at the discretion of the network administrator; he can focus on one specific aspect or on the entire network.

We have many plans for future work. Among them, we would like to have a graphical interface to show input and output in terms of ambients, and would like to experiment with weighted ranking algorithms (e.g. [29]). This way we could cover the case of hosts with high business value but low number of incoming links. Finally, our choice of attackers actions is currently non-deterministic. However, stochastic choice [20], based on risk indicators, would be even better.

REFERENCES

[1] R. A. Martin, "Managing Vulnerabilities in Networked Systems," *IEEE Computer Society Computer Magazine*, vol. 34, no. 11, pp. 32–38, November 2001.

[2] O. Sheyner and J. Wing, "Tools for Generating and Analyzing Attack Graphs," in *In Proc. of Workshop on Formal Methods for Components and Objects*, ser. LNCS 3188. Germany: Springer-Verlag, 2004, pp. 344–371.

[3] R. W. Ritchey and P. Ammann, "Using Model Checking to Analyze Network Vulnerabilities," in *SP'00: Proc. of the 2000 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 156–165.

[4] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen, *Systems and software verification: Model-checking techniques and tools*. Berlin: Springer-Verlag, 2001.

[5] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical attack graph generation for network defense," in *ACSAC '06: Proc. of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 121–130.

[6] W. Li, R. B. Vaughn, and Y. S. Dandass, "An approach to model network exploitations using exploitation graphs," *Simulation*, vol. 82, no. 8, pp. 523–541, 2006.

[7] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2002, pp. 217–224.

[8] S. Jajodia, S. Noel, and B. O'Berry, "Topological Analysis of Network Attack Vulnerability," in *Managing Cyber Threats: Issues, Approaches and Challenges*. Germany: Springer-Verlag, 2005.

[9] L. P. Swiler, C. Phillips, D. Ellis, and S. Chakerian, "Computer-attack graph generation tool," in *DISCEX II'01: DARPA Information Survivability Conference and Exposition Conference and Exposition*, vol. 2. Washington, DC, USA: IEEE Computer Society, June 2001, pp. 307–321.

[10] X. Ou, W. F. Boyer, and M. A. McQueen, "A Scalable Approach to Attack Graph Generation," in *CCS '06: Proc. of the 13th ACM Conf. on Computer and Communications Security*. New York, NY, USA: ACM, 2006, pp. 336–345, people.cis.ksu.edu/~xou/publications/ccs06.pdf.

[11] J. Dawkins and J. Hale, "A Systematic Approach to Multi-Stage Network Attack Analysis," in *IWIA '04: Proc. of the 2nd IEEE Int. Information Assurance Workshop*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 48–56.

[12] "Skybox security inc." 2008, http://www.skyboxsecurity.com/, accessed 16 Sept 2008.

[13] S. Noel and S. Jajodia, "Managing attack graph complexity through visual hierarchical aggregation," in *VizSEC/DMSEC '04: Proc. of the 2004 ACM workshop on Visualization and data mining for computer security*. New York, NY, USA: ACM, 2004, pp. 109–118, http://doi.acm.org/10.1145/1029208.1029225.

[14] L. Williams, R. Lippmann, and K. Ingols, "An interactive attack graph cascade and reachability display," in *VizSEC'07: Proc. of the Workshop on Visualization for Computer Security*. Springer-Verlag, October 2007, pp. 221–235.

[15] S. Noel and S. Jajodia, "Understanding Complex Network Attack Graphs through Clustered Adjacency Matrices," in *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 160–169.

[16] V. N. L. Franqueira and R. H. C. Lopes, "Vulnerability Assessment by Learning Attack Specifications in Graphs," in *IAS'07: Proc. of the 3rd Int. Symposium on Information Assurance and Security)*, August 2007, pp. 161–164.

[17] M. Cremonini and D. Nizovtsev, "Understanding and Influencing Attackers Decisions: Implications for Security Investment Strategies," in *WEIS06: 5th Workshop on the Economics of Information Security*, June 2006, http://weis2006.econinfosec.org/docs/3.pdf.

[18] L. Cardelli and A. D. Gordon, "Mobile Ambients," in *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS'98*, ser. LNCS, vol. 1378. Berlin Germany: Springer-Verlag, 1998, pp. 140–155.

[19] L. Cardelli, "Mobility and security," in *Proc. of the NATO Advanced Study Institute on Foundations of Secure Computation*, ser. NATO Science Series, F. L. Bauer and R. Steinbrggen, Eds. Marktoberdorf, Germany: IOS Press, 27 July - 8 August 2000, pp. 3–37, lecture notes for Marktoberdorf Summer School 1999.

[20] A. Regev, E. M. Panina, W. Silverman, L.Cardelli, and E. Shapiro, "BioAmbients: An abstraction for biological compartments," *Theoretical Computer Science, Special Issue on Computational Methods in Systems Biology*, vol. 325, no. 1, pp. 141–167, September 2004.

[21] R. Milner, "Pure bigraphs," University of Cambridge, Tech. Rep. UCAM-CL-TR-614, January 2005.

[22] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Comput. Netw. ISDN Syst.*, vol. 30, no. 1-7, pp. 107–117, 1998.

[23] V. N. L. Franqueira, R. H. C. Lopes, and P. van Eck, "Multi-step Attack Modelling and Simulation (MsAMS) Framework based on Mobile Ambients," in *SAC'2009: Proc. of the 24th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM Press, March 2009, accepted for publication.

[24] NVD, "National vulnerability database v2," http://nvd.nist.gov/. Visited 10-July-2008.

[25] V. N. L. Franqueira and M. van Keulen, "Analysis of the NIST database towards the composition of vulnerabilities in attack scenarios," Centre for Telematics and Information Technology (CTIT), University of Twente, Enschede, The Netherlands, Tech. Rep. TR-CTIT-08-08, Feb. 2008.

[26] J. M. Kleinberg, "Authoritative Sources in a Hyperlinked Environment," in *In Proc. Ninth Ann. ACM-SIAM Symp. Discrete Algorithms*. New York: ACM Press, 1998, pp. 668–677.

[27] A. N. Langville and C. D. Meyer, *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton Universty Press, 2006.

[28] V. N. L. Franqueira, R. H. C. Lopes, and P. van Eck, "Multi-step Attack Modelling and Simulation (MsAMS) Framework based on Mobile Ambients," Centre for Telematics and Information Technology (CTIT), University of Twente, Enschede, The Netherlands, Tech. Rep. TR-CTIT-08-44, Jun. 2008.

[29] J. A. Tomlin, "A new paradigm for ranking pages on the world wide web," in *WWW '03: Proc. of the 12th Int. Conf. on World Wide Web*. New York, NY, USA: ACM, 2003, pp. 350–355.