# Multi-step Attack Modelling and Simulation (MsAMS) Framework based on Mobile Ambients

Virginia N. L. Franqueira[1] and Raul H. C. Lopes[2] and Pascal van Eck[1]

[1] University of Twente
Enschede, The Netherlands
{franqueirav,p.a.t.vaneck}@ewi.utwente.nl
[2] Brunel University
London, England
raul.lopes@brunel.ac.uk

**Abstract.** Attackers take advantage of any security breach to penetrate an organisation perimeter and exploit hosts as stepping stones to reach valuable assets, deeper in the network. The exploitation of hosts is possible not only when vulnerabilities in commercial off-the-shelf (COTS) software components are present, but also e.g. when an attacker acquires a key (e.g. a password) on one host which allows him to exploit further hosts on the network. Finding attacks involving the latter case requires the ability to represent dynamic models.
In this paper we present MsAMS (Multi-step Attack Modelling and Simulation), an implemented framework, based on Mobile Ambients, to discover attacks in networks. The idea of ambients fits naturally into this domain and has the advantage of providing flexibility for modelling. Additionally, the concept of mobility allows the simulation of attackers exploiting opportunities derived either from the exploitation of vulnerable as well as from the exploitation of non-vulnerable hosts, through the acquisition of keys.

**Keywords:** Network Attack, Mobile Ambients, Attack Simulation, Vulnerability Assessment, Attack Graph

## 1 Introduction

One single hole in the network perimeter is enough to allow an attacker to penetrate the network and exploit hosts as stepping stones to reach valuable assets. Defenders need to tune into the mindset of attackers [1] to track those possible stepping stones and manage the trade-off between impact of attacks and cost of countermeasures. This task is rather challenging due to the complexity and size of current networks, and due to the variety of opportunities and strategies used by attackers. For example, attackers do not necessarily only take advantage of vulnerabilities in commercial off-the-shelf (COTS) software components but can also take advantage of keys (e.g. passwords or private session keys) acquired

dynamically while the simulated attack is taking place. Such keys greatly increase an attacker's spectrum of possibilities since he gains the ability to exploit safe, i.e. not vulnerable, hosts as well. However, static models, such as Attack Graphs [2,3,4,5,6,7,8,9], are unable to represent this dynamic aspect.

We address this issue by proposing MsAMS, a framework for modelling and simulation of network attacks, the design of which draws heavily on Cardelli's work on *Mobile Ambients* [10,11] and formal biology [12], and on Milner's work on bigraphs [13]. In this framework, a network is viewed as a so-called *Ambient* containing other ambients, such as subnets, hosts, and firewalls, on a tree structure. As further explained in the paper, an *Ambient* defines a hyperedge [14] which represents the idea that a communication performed over it is seen by each ambient in that hyperedge, thus no link between sibling hosts (belonging to a same ambient) is required. Besides, an ambient's boundary may contain rules which allow its reaction with other ambients, resulting in changes on the rules of the ambients involved. After the modelling is complete, MsAMS simulates an attacker (also an ambient) dynamically finding an attack path allowed by the modelled ambients and their embedded rules. The consequence of this is that the graph generated has as many nodes as the number of ambients modelled. Thus, it is not subject to the problem of state explosion [15].

We see this framework very much as a *working tool* whose users are security practitioners, such as network administrators. It allows them to gain knowledge about their network. It also permits zooming in on some parts of the network they want to investigate at the level of Access Control Lists, and zooming out to a more abstract level of network as a whole. MsAMS is flexible enough to allow the modelling of a network in different ways and with more or less details, at the discretion of the person using it. MsAMS does not require complex sets of pre- and postconditions to model the composition of vulnerabilities in consecutive attack steps, as it happens with other approaches [4,16,17]. Modelling the input requires (i) the network configuration, (ii) vulnerabilities in COTS present in the network which can be obtained automatically from vulnerability scanning tools such as Nessus [18], and (iii) their attributes, which can be obtained from vulnerability databases such as the NVD [19].

The paper is organised as follows. We provide an overview about network modelling and simulation of attacks with the MsAMS framework in Sections 4 and 5, respectively. A running example, presented in Section 2, is used to support the explanations. In Section 6, we introduce the concept of exposure and describe acquisition of keys, using an insider attack example. An overview of the heuristic algorithms implemented to find attacks is provided in Section 7, and results are reported in Section 8. Finally, in Section 9, we conclude and list future work.

## 2 Running example

We use the network illustrated in Figure 1 from Ingols et al. [2] as the basis for introducing core concepts and to give an overview about modelling and simulation with MsAMS.

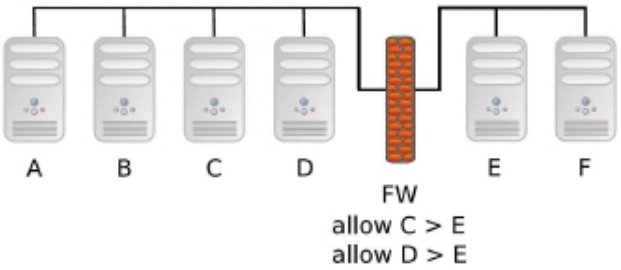
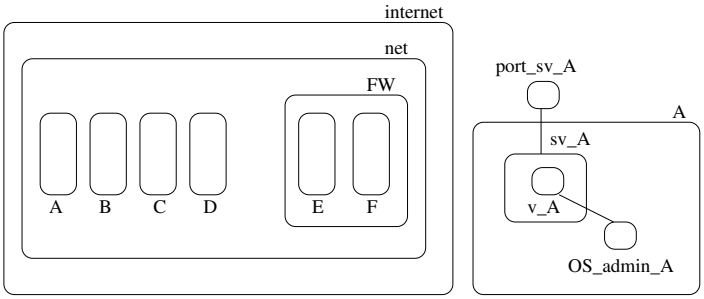



**Fig. 1.** An example network from [2]




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

**Fig. 2.** Modelling the example network as *Ambient*s

In this example-network the attacker is 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 Figure 2(a). In the figure, we see an ambient *internet* containing the ambient *net*. The ambient *net* contains 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 $E, F$, i.e. hosts $E$ and $F$. Figure 2(b) provides a zoom view of ambient $A$. This ambient contains an ambient representing a listening service $sv\_A$, containing itself an ambient representing a vulnerability $v\_A$ on that service. The exploitation of $v\_A$ results in the acquisition of privilege admin ($OS\_admin\_A$) over host $A$'s Operating System. Furthermore, there is an ambient $port\_sv\_A$ providing access to service $sv\_A$ from any ambient within the network *net*.

Notice that we presented above *one* intuitive way of modelling the example network. However, this is not at all the only option since MsAMS is flexible. For example, the firewall could be modelled as well as an ambient which interfaces with two ambients representing subnets $(A, B, C, D)$ and $(E, F)$.

## 3 Related work

The MsAMS framework has basically the same objective as an Attack Graph, both show "the ways an attacker can compromise a network or host" [2]. Therefore, we consider the latter as our main source of related work.

Ou et al. [20] use Datalog rules in the MulVAL tool for modelling the input required for the generation of Attack Graphs. A Prolog reasoning engine then captures the relationship among those with further rules and generates the graph. Their framework, like ours, has a formal approach for input specification and for reasoning. Recent work from Sawilla and Ou [21, Page 1-3] also uses the same running example as we do, illustrated on Figure 1, and the graph produced by MulVAL is shown in page 3. Thus, we were to compare theirs and ours approach more closely. Their graph has 50 nodes while ours has 33 nodes [3]. This difference may be explained by the presence of redundant nodes in the former graph, such as the ones related to firewall rules ($C > E$ and $D > E$). Firewall rules, modelled as "hacl" clauses in MulVAL, appear twelve times in the graph. In our approach the *Ambient* firewall appears once in the graph and contains two filtering rules at its boundary.

Ingols, Lippmann et al. [2] have the same approach, in NetSPA tool, as we have for the modelling of vulnerabilities, based on an access-to-effect paradigm (also adopted by others, e.g. [7]). Besides, they use a matrix ($n \times n$, where $n$ is the number of hosts) to capture reachability among hosts, and apply collapsing techniques to reduce its size. Thus, intra-subnet reachability is reduced to one row in their matrix. Reachability is more naturally modelled when using *Ambient*s, since all ambients inside a same ambient can communicate among themselves, and filtering rules can be embedded on ambients' boundaries. Additionally, their multi-prerequisite graph accommodates the concept of credential for what we call *key*. However, no example using credential is provided in their paper. Nevertheless, because their model does not allow the dynamic acquisition of a credential (as ours), they may have to rely on the assumption that an attacker *has* already a credential and will be able to access non-vulnerable hosts which requires this credential as pre-requisite. Thus, we cannot speculate further.

Jajodia, Noel et al. [4] rely on detailed pre- and postcondition rules to compose attack steps. The specification of these conditions enables TVA tool to find attacks composed both by vulnerable and non-vulnerable hosts (as illustrated in example on page 258, where host *ned* is not vulnerable). However, this is achieved as a consequence of the postcondition enabled by the exploitation of the preceding vulnerability, requiring a detailed analysis of dependencies among vulnerabilities. We have a different perspective and use a more simplistic approach, based on access-to-effect, to model vulnerabilities, but nevertheless consider the possibility of an attacker acquiring keys which enable the compromise of non-vulnerable hosts.

[3] eight nodes illustrated in Figure 2(a) and additional four nodes for each host, as illustrated for host $A$ in Figure 2(b), plus one attacking node

Sheyner and Wing [9] provide a toolkit based on symbolic model checker. Like other model checker-based approaches [22], state explosion [15] is an issue. These graphs represent a path for every single possible combination of attack steps, thus its complexity becomes exponential. Our approach does not suffer from this problem since the graph only represents nodes (i.e. *Ambient*s) and links which occur in practice, and the composition of attack steps relies on the matching of rules embedded on the ambients' boundaries. Sheyner also models trust relations among hosts. It means that the key for a host $h_1$ gives the attacker access to host $h_2$ as well, if these two hosts trust each other. MsAMS permits the specification of these relations at the level of users, thus a user $u_1$ may be trusted by $h_2$ while $u_2$ may not.

Ha, Chinchani et al. [23,24] propose a type of graph which allows not only the modelling but also the simulation of an attacker searching through the graph. Nodes are associated with tokens and edges associated with minimum and maximum costs. Like it happens in MsAMS, nodes can be of many kinds e.g. firewall, vulnerability, service, accounts. During the simulation process, the attacker acquires tokens, and if he has a token he incurs on minimum cost to traverse an edge, otherwise on maximum cost. This approach has many characteristics similar to ours, e.g. it allows modelling insiders which "hold" tokens (i.e. keys) at the beginning of an attack, but it does not allow the representation of Access Control Lists, and the possibility of modelling nodes which release key while others not.

Gorodetski and Kotenko [25] have a grammar-based approach for the simulation of attacks in networks. A family of grammars for each attacker intention has to be specified which, when specialised by substitution, generates attack paths. Among other characterists, their approach differ from ours because they do not have the objective of *discovering* attacks given a network model.

## 4 Modelling with the MsAMS framework

We have seen in Section 2 the network topology of the running example represented in terms of ambients. In essence *Ambient*s are environments where any computing activity will happen. They are abstractions introduced to represent hosts, services, vulnerabilities, networks, users and even keys. Each ambient can possibly react with other ambients in its neighbourhood, depending on its and theirs capabitities.

**Definition 1** *An* Ambient *Amb is a tuple* $(Amb\prime, R)$*, where* $Amb\prime$ *is a list of* Ambient*s contained in Amb, and R is a list of static rules defining the dynamic behaviour of ambient Amb.*

**Definition 2** *Each ambient Amb has an unique name N such that* $N = name(Amb)$.

The ambients of the network example, illustrated in Figure 2, are specified in MsAMS as follows.

```
1  internet: ["net"] []
2  net: ["A" "B" "C" "D" "FW"] [Repeat (Accept "internet")]
3  FW: ["E" "F"] [Repeat (AllowIn "C" "E"), Repeat (AllowIn "D" "E")]
3  port_sv_A: [] [Repeat (Accept "net")]
4  A: ["sv_A" "OS_admin_A"] []
5  sv_A: ["v_A"] [Repeat (Accept "port_sv_A")]
7  OS_admin_A: [] [Repeat (Accept "v_A")]
8  v_A: [] []
```

The primitive *Repeat* allows the execution of the rule which follows, on a infinite loop.

The primitive *Enter* allows one ambient to move into another ambient, given that this last ambient accepts it via an *Accept*.

The *AllowIn* primitive allows one ambient to define which ambient is allowed to cross its boundaries towards another ambient. Although this rule can be modelled using *Enter* and *Accept*, it has been explicitly included for convenience when modelling firewalls.

We now look into more details the specification of ambients provided above. Line 1 specifies the ambient *internet*. It contains the ambients *net* and no action rules.

Line 2 specifies the ambient *net*. It contains five other ambients and is continuously *accepting* that any ambient within the ambient *internet* enters its perimeter. Note that *net* could restrict this access via an external firewall for example, as it happens in practice.

Line 3 specifies the ambient $FW$ which contains two other ambients. It restricts the broadcasting of messages from hosts A-D towards its children ambients, *allowing* only ambients $C$ and $D$ to communicate with ambient $E$. *AllowIn AmbientSource AmbientDestination* may restrict communication between any pair of ambients. For example, if we had defined that ambients $A$ to $D$ were inside ambient "dmz", we could have firewall rules of the type: $AllowIn\ dmz\ F$ or $AllowIn\ ssh - tcp - 22\ E$.

Line 4 specifies an empty ambient $port\_sv\_A$. It accepts communication coming from ambient *internet*.

Line 5 specifies ambient $A$ containing two other ambients, a listening service $sv\_A$ and a privileged account $OS\_admin\_A$, but no action rules.

Line 6 specifies ambient $sv\_A$, containing a vulnerability represented by the ambient $v\_A$. The latter is continuously *accepting* ambient $port\_sv\_A$.

Line 7 specifies an empty ambient $OS\_admin\_A$. This ambient is continuously *accepting* ambient $v\_A$.

Line 8 specifies an empty ambient $v\_A$. On the one hand, since this vulnerability is remotely exploitable, it does not require an attacker to be authenticated as a user of the host to enable its exploitation. Thus, it is exploitable by anyone entering its parent ambient $sv\_A$. On the other hand, since this vulnerability has the effect admin, it contains a link with $OS\_admin\_A$ and it accepts it.

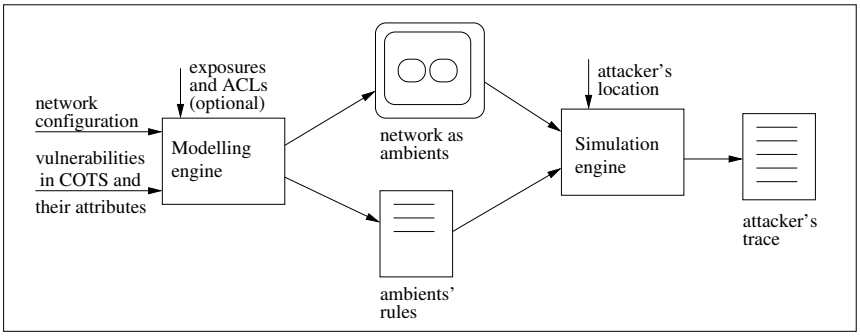



**Fig. 3.** Architecture of the MsAMS framework

### 4.1 Matching Enter/Accept

In Cardelli's Bioware Languages [12], an *Ambient* (called *membrane* in that paper) can possibly contain other *Ambient*s inside it. In that sense, an *Ambient* has a set of children and a parent. Cardelli defines two conditions that must **both** be satisfied for an *Ambient* $X$ to move into an *Ambient* $Y$:

- $X$ is a sibling of $Y$ or $X$ and $Y$ are children of sibling parents.
- $X$ has an action request entry into $Y$, which we denote *Enter* $Y$, and $Y$ allows the entry movement with an action *Accept* $X$.

We replace in MsAMS framework the first condition with:
*$X$ is a sibling of $Y$ or $X$ and $Y$ are children of sibling parents, or an edge exists connecting $X$ and $Y$.*

## 5 Simulations with the MsAMS framework

The architecture of the MsAMS framework and where the simulation engine fits is illustrated in Figure 3.

The simulation engine receives as input (i) a hypergraph defining the network topology as ambients (similar to Figure 2), (ii) a set of static rules determining *Capabilities* and *Action*s, as shown in Section 4. These can be used by the ambient and can be performed at the boundary that the ambient defines. The set of actions implicitly defines a non-deterministic choice of action to perform, and (iii) a set of computing *Agent*s, i.e. one or more attacker ambients (i.e. the location of attackers). The engine then performs two basic tasks using heuristics, as described in Section 7. First, it assigns automatically value to ambients and cost to links of the hypergraph. Second, it computes possible steps an attacker can perform on the ranked hypergraph. Each of these steps can either be accepted, if the attacker's actions and the ambients' action match, as described in Section 4.1, or rejected if the actions do not match. This match is achieved via reduction rules [10]. A match means that the attack can actually perform the

step, and this is recorded. In the end, we have the attacker's complete trace until a target, i.e. a host of high value, is reached. This trace is a possible multi-step attack on the modelled network. An attacker trace for the running example (see Figure 1) is illustrated next.

```
Enter "net"
Enter "port_sv_A"
Enter "sv_A"
Enter "v_A"
Enter "OS_admin_A"
Enter "port_sv_D"
Enter "sv_D"
Enter "v_D"
Enter "OS_admin_D"
Enter "port_sv_E"
Enter "sv_E"
Enter "v_E"
Enter "OS_admin_E"
```

This trace shows the possible attack $ADE$. Other possibilities which can be obtained by executing the simulation several times are $ACE$, $ABCE$, and $ABDE$.

## 6 Acquisition of keys

In this section we use an insider attack example, more appropriate for illustrating the acquisition of keys in MsAMS. However, first we introduce the concept of exposure.

### 6.1 Exposures

We use exposures to represent stealthy ways to acquire keys. An attacker can get remote or local access to a host by means of vulnerabilities but, most of the times, he does not automatically obtain keys (e.g passwords) for that host. Thus, an exposure is an abstraction to model the disponibility of keys. It corresponds to the real situation of passwords saved locally, social engineering or even passwords acquired using key logging mechanisms. However, exposures can also be used for modelling Public Key Infrastructure (PKI), as shown in Section 6.2.

In this context, an exposure is an empty *Ambient* that is forever repeating the action of releasing a key. To illustrate this concept let's consider there is a host protected by a key (ambient $k$). The host contains an exposure which releases this key. If a (malicious) agent, by simulation, happens to have the capability described next, then a match between *AcquireKey* and *ReleaseKey* is possible.

```
host: [exposure] []
exposure: [] [Repeat (ReleaseKey "k")]
alice: [] [Repeat (AcquireKey "k")]
```

This match represents the acquisition of a key. It occurs by means of reduction rules [10], when agents are computed. After the computation, alice and exposure ambients have their capabilities updated, allowing alice to enter the host and behave like its legitimate user.

```
host: [] [Repeat (Accept "alice")]
alice: [] [Repeat (Enter "host")]
```

After being captured, a key is forever kept in a set of *Keys* which can be used by the ambient, in this case by ambient *alice*. This list of keys can be seen as an abstraction of a "bag of keys" which an attacker can accumulate along a multi-step attack. We assume that the monotonicity property [26] holds (as many other researchers [2,3,4,7] do) and, therefore, once acquired a key is never lost, i.e. the attacker does not need to backtrack to re-acquire a key.

### 6.2 An example of insider attack

Figure 4 shows an example of key acquisition using a Public Key Infrastructure (*pki*). This example has been adapted from [24].

The environment is a *Bank* with a set of teller hosts, represented by *teller0* and *teller1*, a *manager* host, and a database, *DB*. There is a key to access the manager host, *manager.key*. The *DB* ambient has a user ambient, *dbUser*, that users can access through *In/Out* sequences. Primitives "In" and "Out" estabish communication between two ambients, representing for example the ability to read and write. There is also a file system, *dbFS*, and an administrator account, *DBA*. Outside the bank, a *PKI* ambient gives a ticket for a *DB* session. Only the manager has access to the *pkiKey*.

The general rule for each ambient is that an ambient is a hyperedge connecting each ambient inside it. Notice that a list of actions denotes a nondeterministic choice.

The actions in the example are:

```
1  Bank: ["teller0" "teller1" "managerVuln" dbUserKey" dbaKey"
          "manager.Key" "manager" "DB"] []
2  DB: ["dbFS" "dbUser" "DBA"] []
3  PKI: ["pkiKey" "exp.dbaKey"] []
4  dbFS: [Repeat (Accept "DBA"),
       (Repeat (Prefix (In "dbUser") (Out "dbUser")]
5  dbUser: [Repeat (Accept "dbUserKey")]
6  dbUserKey: [] []
7  DBA: [Repeat (Accept "managerVuln"), Repeat (Accept "dbaKey")]
8  managerVuln: [] []
9  dbaKey: [Repeat (Accept "Bank")]
10 manager: [Repeat (Accept "managerKey")]
11 pkiKey: [Repeat (Accept "manager")]
12 exp.dbaKey: [Repeat (ReleaseKey "dbaKey")]
```

Notice that the *DBA* can be entered through a vulnerability or by having a *Enter dbaKey* capability. This capability can be acquired by:

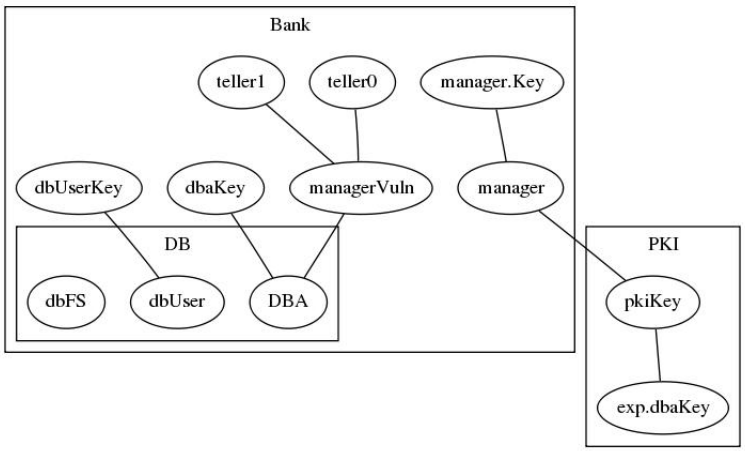



**Fig. 4.** Modelling the insider attack example (adapted from [24]) as *Ambient*s

– using a capability *AcquireKeyFrom "expdbaKey"*, or
– giving some ambient the capability *Enter "dbaKey"* from the start.

## 7 Heuristics used to find attacks

In a previous paper [27], we introduced the idea of representing attacks using a language based on CSP [28]. In that paper, we assumed that a network model was a graph with values assigned to nodes and costs assigned to edges. Searching for an attack was treated as an optimisation problem.

In the present approach we still assume that values are assigned to ambients and we still have a cost measure for moves from ambient to ambient. However, we have developed algorithms for assigning values and costs. Borrowing from social network analysis [29] and page ranking we assign value to an ambient proportional to the value assigned to other ambients that point to it. An ambient $A$ points to an ambient $B$ when a move from the former into the latter is possible.

An algorithm based on pageRank [30] computes a score for each ambient in a network model. That score, *authority*, can be considered as the value for the ambients. Starts with a square matrix where cell $(i, j)$ contains a zero when there is no link from $i$ to $j$. This algorithm runs a fixed number of iterations, computing a matrix multiplication in each, giving a $O(n^2)$ time algorithm, where $n$ is the number of nodes.

The search for an attack uses the values assigned to all ambients in the network model, computed using pageRank, and an initial set of suspicious ambients, the ambients where the attacker would be located. The algorithm can also be given a set of ambients that could be considered as hints in the sense that

we expect that an attack would use them. This can be useful when a network administrator wants to know if the network is subject to a specific attack, as e.g. the massive RealPlayer exploit via SQL injection, reported by the press in January 2008 [31].

The search algorithm computes in each step the set of all possible moves, ordering them by a priority scheme, that is akin to the cost value used in [27]. To compute the value of a move, the pageRank algorithm is extended with the concept of hub, borrowed from [32]. A hub is an ambient that can be source of a move. An ambient $A$ has high hub score when it is possible to move from $A$ to ambients with high value. In a local network, the filesystem where every user has her home directory should have a high asset (authority) rank. Assuming that each user gets access to the home directory using NFS, an ambient modelling the NFS would have both high authority (accessed by all users) and high hub score, because the NFS uses the filesystem to satisfy each user request. The HITS algorithm assigns both authority and hub score to a set of ambients, given their neighbourhood, and an initial set of values for them. It also runs a fixed number of iterations that can be computed in $O(m^2)$ each, where $m$ is the number of ambients in iteration. The search algorithm selects a fixed subset of the *hubbiest* e proceeds in depth-first search, giving priority to moves into high scoring authorities or hubs. More details will be left out due to lack of space.

## 8 Performance and scalability of MsAMS framework

The time for computing an attack is dominated by the computation of assets' ranks. This is perforformed by an algorithm based on the pageRank [30] algorithm. A naïve implementation in Haskell can take $O(n^3)$. An efficient implementation in C of the matrix multiplication performed in each cycle can take advantage of the sparse matrix that is generated to produce a $O(n)$. Our implementation precomputes the matrix in $O(n^2)$ and then applies ranking algorithm in $O(n^2)$. We do not run, at least in the present version, ranking algorithms, up to convergence. They executed for a fixed number of steps. However, experiments have shown [30] that fixed number of iterations is adequate in practice.

Figure 5 shows the time for experiments with a variation on our running example, where the network consisted of the hosts shown in Figure 1 and up to 8192 nodes evenly split between the subnet behing the firewall and the subnet outside the firewall. All experiments assumed the atacker positioned initially inside host $A$.

It is fair to notice that when we double the number of nodes from 4096 to 8192 the time multiplies by 6, when we would expect it to multiply by 4, given that we claim that our algorithm is quadratic. We attribute that to the garbage collection of Haskell due to the heavy use we make of lists for collecting intermediate results, but we admit that more extensive testing could identify more important performance bottlenecks in our algorithms. We know that our implementation of the ranking algorithms could be greatly improved by using

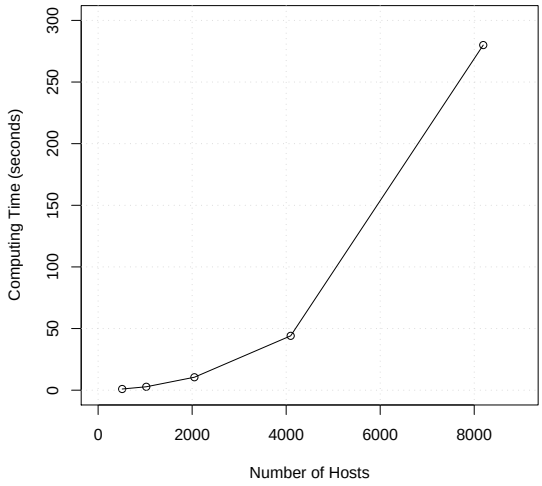



**Fig. 5.** Performance of the MsAMS framework

effient matrix multiplication libraries and we know that the heavy use of lists might be replaced with local updates in vectors or matrices.

## 9 Conclusion and further work

We presented in this paper the MsAMS framework based on Mobile Ambients theory. It allows the modelling of a network in an intuitive way as *Ambient*s, and the specification of static rules defining the dynamic behaviour of each ambient. Then, an engine based on heuristics, simulates attackers' steps to find attacks which are possible, given the network modelled. The dynamic aspect of MsAMS allows the simulation of an attacker with the ability to compose attack steps either from the exploitation of vulnerable and non-vulnerable hosts.

The scalability of the approach has been demonstrated by experimental tests with up to 8192 hosts. We are aware that further testing is needed and, therefore, this topic is our priority for future work. We believe that our implementation, even in Haskell, can be greatly improved. In addition, we are planning for an implementation in C of both the Asset Ranking and the Hub Ranking algorithms.

One advantage of the framework is its flexibility. It is up to its users to decide which level of details is needed. He/she can decompose the network at his discretion and model fine grained entities and their relationships, including ACLs (Access Control Lists), or abstract to a higher level and only model the minimum which still allows him to find possible attacks on the network modelled.

Maybe he can adopt the latter option first as a way to prioritise a sub-graph of interest, and then adopt the former to zoom in this sub-graph.

Usability is an aspect that needs to be improved and, therefore, is listed for future work. We plan to build an user interface to allow the graphical manipulation of *Ambient*s, when modelling, and the visualisation of possible attacks discovered by the framework, when simulating.


### Acknowledgements

This research is supported by the research program Sentinels (www.sentinels.nl). Sentinels is being financed by Technology Foundation STW, the Netherlands Organisation for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.


## References


1. Schneier, B.: The Ethics of Vulnerability Research. Information Security Magazine (2008) http://www.schneier.com/essay-211.html.
2. Ingols, K., Lippmann, R., Piwowarski, K.: 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) 121–130
3. Ammann, P., Pamula, J., Street, J., Ritchey, R.: A host-based approach to network attack chaining analysis. In: ACSAC '05: Proc. of the 21st Annual Computer Security Applications Conference, Washington, DC, USA, IEEE Computer Society (2005) 72–84
4. Jajodia, S., Noel, S., O'Berry, B.: Topological Analysis of Network Attack Vulnerability. In: Managing Cyber Threats: Issues, Approaches and Challenges. Springer-Verlag, Germany (2005)
5. Noel, S., Jajodia, S.: 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) 109–118 http://doi.acm.org/10.1145/1029208.1029225.
6. Swiler, L.P., Phillips, C., Ellis, D., Chakerian, S.: Computer-attack graph generation tool. In: DISCEX II'01: DARPA Information Survivability Conference and Exposition Conference and Exposition. Volume 2., Washington, DC, USA, IEEE Computer Society (2001) 307–321
7. Li, W., Vaughn, R.B., Dandass, Y.S.: An approach to model network exploitations using exploitation graphs. Simulation **82**(8) (2006) 523–541
8. Ou, X., Boyer, W.F., McQueen, M.A.: 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) 336–345 people.cis.ksu.edu/~xou/publications/ccs06.pdf.
9. Sheyner, O., Wing, J.: Tools for Generating and Analyzing Attack Graphs. In: In Proc. of Workshop on Formal Methods for Components and Objects. LNCS 3188, Germany, Springer-Verlag (2004) 344–371
10. Cardelli, L., Gordon, A.D.: Mobile Ambients. In: Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98. Volume 1378 of LNCS., Berlin Germany, Springer-Verlag (1998)

11. Cardelli, L., Gordon, A.D.: Types for Mobile Ambients. In: Symposium on Principles of Programming Languages. (1999) 79–92
12. Cardelli, L.: Bioware Languages. In: Computer Systems: Theory, Technology, and Applications. Monographs in Computer Science. Springer, New York (2004) 59–65
13. Milner, R.: Pure bigraphs. Technical Report UCAM-CL-TR-614, University of Cambridge (2005)
14. Jukna, S.: Extremal Combinatorics. Springer (2000)
15. Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and software verification: Model-checking techniques and tools. Springer-Verlag, Berlin (2001)
16. Cuppens, F., Ortalo, R.: Lambda: A language to model a database for detection of attacks. In: RAID'00: Proc. of the Third Int. Workshop on Recent Advances in Intrusion Detection, London, UK, Springer-Verlag (2000) 197–216
17. Templeton, S.J., Levitt, K.: A requires/provides model for computer attacks. In: NSPW'00: Proc. of the 2000 Workshop on New Security Paradigms, New York, NY, USA, ACM (2000) 31–38
18. Nessus: (Tenable network security: The Nessus Security Scanner) www.nessus.org.
19. NVD: (National vulnerability database v2) http://nvd.nist.gov/.
20. Ou, X., Govindavajhala, S., Appel, A.W.: Mulval: a logic-based network security analyzer. In: SSYM'05: Proc. of the 14th Conf. on USENIX Security Symposium, Berkeley, CA, USA, USENIX Association (2005) www.cs.princeton.edu/~appel/papers/mulval.pdf.
21. Sawilla, R., Ou, X.: Googling Attack Graphs. Technical Report TM-2007-205, Defense Research and Development Canada (2007) http://www.ottawa.drdc-rddc.gc.ca/html/tm_2007_205_e.html.
22. Ritchey, R.W., Ammann, P.: 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) 156–165
23. Ha, D., Upadhyaya, S., Ngo, H.Q., Pramanik, S., Chinchani, R., Mathew, S.: Insider threat analysis using information-centric modeling. In: Advances in Digital Forensics III. IFIP International Federation for Information Processing. Springer, Boston (2007) 55–73
24. Chinchani, R., Iyer, A., Ngo, H.Q., Upadhyaya, S.: Towards a Theory of Insider Threat Assessment. In: DSN 2005: Int. Conference on Dependable Systems and Networks, IEEE Publishing (2005) 108–117 http://ieeexplore.ieee.org/iel5/9904/31476/01467785.pdf.
25. Gorodetski, V., Kotenko, I.: Attacks against computer network: Formal grammar-based framework and simulation tool. In A.Wespi, G.Vigna, L.Deri, eds.: RAID 2002: Proc. of the Fifth Int. Symposium on Recent Advances in Intrusion Detection. Volume 2516 of LNCS., Springer (2002) 219–238
26. Ammann, P., Wijesekera, D., Kaushik, S.: 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) 217–224
27. Franqueira, V.N.L., Lopes, R.H.C.: Vulnerability Assessment by Learning Attack Specifications in Graphs. In: IAS'07: Proc. of the 3rd Int. Symposium on Information Assurance and Security). (2007)
28. Hoare, C.A.R.: Communicating Sequential Processes. Second edn. Prentice Hall International (2004) online version at http://www.usingcsp.com/cspbook.pdf.
29. Seeley, J.R.: The net of reciprocal influence: A problem in treating sociometric data. Canadian Jounal of Psychology (3) (1949)

30. Langville, A.N., Meyer, C.D.: Google's PageRank and Beyond: The Science of Search Engine Rankings. Princeton Universty Press (2006)
31. Keizer, G.: Mass hack infects tens of thousands of sites. (Computerworld, January 07, 2008) http://www.computerworld.com/action/article.do?command=viewArticleBasic&taxonomyId=16&articleId=9055858&intsrc=hm_topic.
32. Kleinberg, J.M.: Authoritative Sources in a Hyperlinked Environment. In: In Proc. Ninth Ann. ACM-SIAM Symp. Discrete Algorithms, New York, ACM Press (1998) 668–677