An Evolutionary Approach for Learning Attack Specifications in Network Graphs

Virginia N. L. Franqueira
University of Twente
Enschede, The Netherlands
franqueirav@ewi.utwente.nl

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

Pascal van Eck
University of Twente
Enschede, The Netherlands
p.a.t.vaneck@ewi.utwente.nl

Abstract

This paper presents an evolutionary algorithm that learns attack scenarios, called attack specifications, from a network graph. This learning process aims to find attack specifications that minimise cost and maximise the value that an attacker gets from a successful attack. The attack specifications that the algorithm learns are represented using an approach based on Hoare's CSP (Communicating Sequential Processes). This new approach is able to represent several elements found in attacks, for example synchronisation. These attack specifications can be used by network administrators to find vulnerable scenarios, composed from the basic constructs Sequence, Parallel and Choice, that lead to valuable assets in the network.

Keywords: AI in Security and Information Assurance, Evolutionary Computing

1 Introduction

This paper presents an evolutionary approach for learning **attack specifications** for network attacks involving synchronisation, coordination, concurrency, distribution, choice, and sequencing of attack steps. To the best of our knowledge, this is the first time an approach permits attack specifications with such a level of expressiveness to be generated by machine.

The idea of attack specifications comes from the specification of parallel programming. It is based on the notion of communicating processes (CSP: Communicating Sequential Processes by Hare [11, 12]). We view an attack step as a process with input and output ports which allow its composition with other attack steps. The resulting attack specification, output of the algorithm, is a possible attack scenario in a network topology represented as a graph. In our **network graph**, nodes represent computers and arcs represent communication channels. A node can have value associated, representing the benefit gained by the attacker in compromising that node, and sets of nodes can have **added value**, greater than the sum of individual values. An arc has a **cost** representing the level of protection of the communication channel, i.e. it represents the difficulty incurred by the attacker in traversing the arc.

Applied to attacks, synchronisation happens in two levels. The first level of synchronisation is one which assures that attack steps are not disjoint effort but rather a synchronised composition of attack steps to achieve attackers' goals. The second level of synchronisation is one which is required in some attacks. For example, a Denial of Service attack only happens if the there is a synchronism in overloading a resource, service or communication channel. In this case, there is what we call a *point of synchronisation* in the overloading process which requires, as input, the convergence of outputs from previous attack steps.

Coordinated attacks [8, 5] form a class of attacks which involve a collusion of attackers and one or several targets. Additionally, it can also involve the coordination of resources to launch an attack. Thus, coordination is directly related to synchronisation achieved by composition. It is also related to distribution, when resources and/or actors are scattered.

Choice happens in attacks when the attacker has several possibilities, for example, by acquiring access to a collection of computers, and chooses one option that better fits his goals. In this case we assume the attacker (representing the environment) makes a deterministic selection.

Sequencing is a basic concept for modelling attack scenarios in steps. It allows the breakdown of attacks in terms of organised actions.

1.1 Contribution of the paper

The contribution of this paper is twofold. First, the approach we describe allows **vulnerability assessment** of a network, represented as a graph. We believe that the output attack scenarios returned by the algorithm can be useful for a system administrator to gain awareness of potential means that attackers have to reach valuable targets.

Second, the **evolutionary algorithm** on its own is a contribution since it deals with multiple solutions but, unlike traditional Genetic Algorithm and Ant Colony Optimisation for example, the number of solutions under consideration is not limited to the size of the population or the number of ants. The algorithm relies on the analogy of the evolution of species that allows a population to grow until its individuals start to compete for resources when unfit individuals start to die. Most important, the algorithm potentially applies to other optimisation problems, like the Travel Salesman Problem, as well, provided a representation of the solution, and a fitness function.

1.2 Organisation of the paper

The paper is organised as follows. First, related work is reviewed in Section 2. Second, we present our evolutionary approach in Section 3 in terms of solution representation (called **CSP**), edition operations over CSPs and the algorithm itself. Third, we evaluate our approach in Section 4, using a motivating example. Finally, we draw conclusions and point to future work in Section 5.

2 Related work

Vulnerability assessment and attack modelling has been extensively researched using techniques such as trees and graphs. However, to our knowledge, none of these approaches permit the modelling of synchronism between attack steps, required for representing for example distributed attacks, such as DoS.

Fault Tree, Attack Tree and Event Tree (e.g. [10]) refine a tree root representing an attack goal or known vulnerability. These trees cannot represent cycles, cannot represent order between nodes and cannot model parallelism.

Attack graphs represent attack steps as nodes, and step transitions as arcs. Phillips and Swiler [19] uses near-optimal shortest path applied to attack graphs to assess the vulnerability of targets. Although their approach permits the modelling of attack step cycles, it does not permit the modelling of simultaneous steps performed by an attacker, as the authors pointed out themselves.

Sheyner et al. [21] generate attack graph using a symbolic model checker. This process requires as input a model of the network represented as a finite state machine and a safety property to be satisfied by the model. Attack graphs are generated from all possible counter-examples, if the property is not satisfied by the model. The approach does not consider any form of parallel attacks.

Dacier et al. [4] propose privilege graphs where nodes are possible attack initiators and possible targets, and arcs are vulnerabilities which allow the acquisition of privileges for a node-initiator towards a node-target. They transform a privilege graph to Petri-net and then derive a state graph, which is used for vulnerability assessment in terms of MTTF (Mean Time To Failure), i.e. mean time for an attacker to reach a target. The assessment is performed in terms of attack step and does not permit the composition of attack scenarios.

Chinchani et al. [3, 2] present challenge graphs. Nodes are entities which provide information or capabilities, represented by keys. Arcs are channels of interaction, which have key-challenges and costs attached. The set of vertices which reaches the set of target with minimum cost indicates vulnerable attack scenarios. Their graph can model colluding attackers, i.e. more than one attacker aiming the same target. However, it cannot model synchronised attack steps.

Gorman et al. [7] uses a graph approach to represent internet autonomous systems (i.e. key internet nodes), as nodes, and their connections, as arcs. They analyse security in terms of statistical parameters and infection propagation, using different attack and defence strategies. Their assessment of vulnerability is based on investment with defence. Gregg et al [9] measure attack effectiveness using Probability of DoS computed as a function of timeout settings, number of connections allowed, and rate of attack requests. Like Gorman et al., they draw conclusions in a quantitative way only, without interest on attack scenarios.

Petri nets allow the modelling of the two levels of synchronisation mentioned in Section 1 the first is achieved by net soundness, and the second, by combining structures like AND-split and AND-join [23, 14] in a net. Petri Nets have the same level of expressiveness as our approach, since they can represent all the elements we mentioned as required for modelling attacks. However, we suspect that the effort to specify complex attack scenario involving many steps is higher in Petri Nets compared to our approach due in part to a substantial increase in net length.

CSP (Communicating Sequential Processes) by Hoare [11, 12] and CCS (Calculus of Communicating Systems) by Milner [16], have been traditionally used to model concurrent processes. However, recently there has been an increasing interest in using them, and Pi-calculus[17] as well, for modelling for example cryptographic protocols using the Dolev-Yao [6] methodology with CCS [15], and other protocols, like non-repudiation between parties, using CSP [20].

3 Our evolutionary approach

We use a system of pools and credits to simulate the evolutionary process of species, where species are allowed to grow until their individuals start to compete for resources to survive. Our approach has the following characteristics which represent a benefit over traditional local search optimisation methods, such as Genetic Algorithm, Simulated Annealing and Ant Colony Optimisation, for example: (i) the size of the solution population is flexible and depends only on the number of credits which are consumed in the reproduction process, (ii) a solution (i.e. a CSP) is not discharged after it has reproduced, increasing the chances of producing future good quality offsprings, and (iii) individuals have second chances of survival, if they have proofed themselves worthy, i.e. CSPs receive recharges of credits if they have value. Thus, the algorithm uses three CSP pools, named **Speculation Pool** (SP), **Attack Pool** (AP) and **Dying Pool** (DP). Figure 1 shows the life-cycle of a CSP based on those pools.

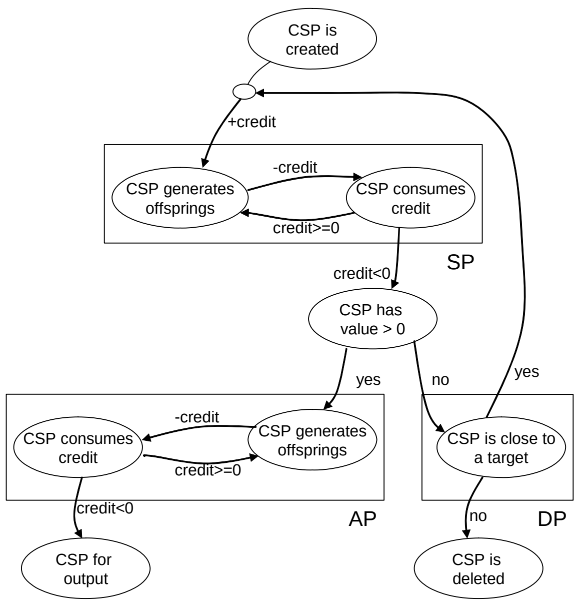

Figure 1. Life-cycle of a CSP

After a CSP is created, it receives credits and is allowed to mature in the SP. Each time a CSP is edited, its amount of credit either decreases, if the generated offspring has fitness worse than its father, or remains unchanged, in the opposite case. This decrease is proportional to the complexity of the generated offspring. Thus, if in order to generate the offspring, it is necessary to traverse the father CSP structure then the complexity is given by the number of arcs in the father CSP. Otherwise, the complexity is one. The CSP remains in the SP until its credits have finished. When this happens, there are two possibilities. Either the CSP has reached a target (i.e. CSP has value different than zero), and in this case it is moved to the AP, or it has not reached a target, and in this case it is moved to the DP. In the AP, the CSP is allowed more credit to improve its complexity. By the end of this AP credits, the CSP is sent to output. In the DP, the CSP is checked to see whether it is within a distance (according to a threshold parameter) to a target. If "yes", its SP credits are restored and it is sent back to the SP. If "no", the CSP is deleted. A CSP is only allowed to return to SP from DP once.

3.1 Solution representation

A solution for the evolutionary algorithm is represented as a CSP attack specification, called CSP. Our CSP is based on Hoare's CSP [11, 12]. Thus, an attack CSP is a **composition of arcs** from a network graph, similar to a composition of processes. Thus, an arc is regarded as a process in our CSP.

Definition 1 (A network graph.) *A network graph is a tuple:*

$$G = (N, A, I, \alpha, \beta), \quad (1)$$

*where N is a set of nodes, representing computers, and A is a set of arcs, representing communication channels, I is an initial set of nodes the organisation has under suspicion or wants to investigate, $\alpha : A \to \mathbb{N}$ is a function that assigns cost to arcs, and $\beta : 2^N \to \mathbb{N}$ (where 2^N is the set of all sub-sets of N) a function that assigns value to sets of nodes. We call this β function **Added Value**.*

Definition 2 (CSP.) *A CSP is: (i) an arc $\in A$ from a network graph, represented as $Arc(a, b)$, or (ii) a sequential composition of a pair of CSPs, $Seq[CSP_1, CSP_2]$, or (iii) a parallel composition of a pair of CSPs, $Par[CSP_1, CSP_2]$, or (iv) a choice composition of a pair of CSPs, $Choice[CSP_1, CSP_2]$.*

Definition 3 (CSP Head Set and Tail Set.) *The head of an $Arc(a, b)$ is a and its tail is b. The head set of a CSP_1 is the subset of nodes h in CSP_1 such that no node in h is tail of any arc in CSP_1. The tail set of a CSP_1 is the subset of nodes t in CSP_1 such that no node in t is head of any arc in CSP_1.*

Definition 4 (Sequential composition.) *A sequential composition $Seq[CSP_1, CSP_2]$ can happen when the tail set of CSP_1 is a subset of the head set of CSP_2. This composition means that CSP_1 happens and, when completed, CSP_2 follows.*

Definition 5 (Parallel composition.) *A parallel composition $Par[CSP_1, CSP_2]$ can happen between any pair of CSPs. It means that CSP_1 and CSP_2 start simultaneously.*

Definition 6 (Choice composition.) *A choice composition $Choice[CSP_1, CSP_2]$ can happen between any pair of CSPs. It means that CSP_1 or CSP_2, but not both, is selected deterministically by the attacker, i.e. by the environment.*

Definition 7 (Target.) *A target is a set of nodes from a CSP_1's tail set that has added value greater than a given threshold.*

Definition 8 (Attack.) *An attack is a CSP_1 that starts on a node $\in I$ and ends on a target.*

Definition 9 (CSP Value.) *The value of a CSP is the added value of its tail set. If the CSP tail set has no added value, its value is zero.*

Definition 10 (CSP Cost.) *The cost of a CSP is the sum of costs of its arcs.*

Definition 11 (CSP Fitness.) *The fitness F of a CSP is:*

$$F = CSPValue - CSPCost \tag{2}$$

3.2 Edition operations

Edition operations are selected based on a probabilistic distribution function, among a set of five possible operations.

The possible edition operations are described next. Editions always generate new offsprings, i.e. the original CSP remains as it is.

1. **New atomic CSP**: A new CSP, consisting of one arc, can be created according to two different options. The first option is: the created $Arc(a, b)$ can have node a chosen from the initial set I. In this case, the CSP will grow towards a target. The second option is: the created $Arc(a, b)$ has node $a \notin I$ and node b with no added value. Thus, the CSP will grow both towards a node $\in I$ and towards a target.

2. **Arc extension**: This edition involves two steps. The first step is the selection of a CSP to be extended. This selection happens according to one of the following criterion: (i) CSP with smallest cost, (ii) CSP with highest value, (iii) CSP with highest fitness, or (iv) random. The second step is the selection of an arc to be added to the CSP. Among all the arcs of the graph, and depending on the type of CSP (i.e. if it grows forwards, backwards or both), an arc that can be composed with the CSP head set or tail set is selected following the four criteria described before.

3. **Seq composition**: This edition involves a pair of CSPs: CSP_1 and CSP_2. First, as described on the previous edition, CSP_1 is selected. Second, a list of candidate for CSP_2 is generated using the following criterion: the tail set of CSP_1 needs to be a subset of CSP_2 head set. Third, one CSP is selected randomly from the list of candidates. Finally,the sequential composition $Seq[CSP_1, CSP_2]$ is generated with selected CSP_1 and CSP_2.

4. **Par composition**: This edition involves a pair of CSPs: CSP_1 and CSP_2. In this case, both CSPs are selected as described in edition "Arc extension". The only restriction imposed in this case is: if both CSPs have value greater than zero (Definition 9), then both tail sets need to be disjoint. A the parallel composition $Par[CSP_1, CSP_2]$ is generated with selected CSP_1 and CSP_2.

5. **New Par CSP**: This edition creates a CSP_1 which is the Par composition of several arcs selected from the target nodes. This selection can be either: random, by best value (i.e. the highest value is selected), or by preference for best value (i.e. the higher values have more chance to be selected).

6. **Parallel join**: This edition extends an existing CSP_1 with a new CSP_2. A CSP_2 with head set equal to CSP_1 tail set is created, if CSP_1 grows forwards. The sequential composition $Seq[CSP_1, CSP_2]$ is generated in this case. A CSP_2 with tail set equal to CSP_1 head set is created, if CSP_1 grows backwards (or both forwards and backwards). The sequential composition $Seq[CSP_2, CSP_1]$ is generated in this case.

All the editions enumerated above apply to CSPs in SP. However, only the "Arc extension" edition apply to CSPs in AP.

3.3 The algorithm

The evolutionary algorithm consists of two main phases: the reproduction phase and the retirement phase. In the former, editions occur creating new generations of CSPs. In the latter, CSPs are selected to be deleted, for output and for a new stage of reproduction. An algorithm iteration, called **cycle**, also has credits (provided as parameter) which increases each time a CSP is edited. A cycle can generate several possible attack specifications as output (from the AP). Figure 2 presents the main algorithm, where cycle_credits sums credits consumed in the reproduction phase and MAX_credits is a parameter. Figure 3 presents the algorithm for the reproduction phase, where nSP and nAP are parameters, and Figure 4 presents the algorithm for the retirement phase.

```
SP = {}, AP = {}, DP = {}
FOR n cycles with MAX_credits each
  reproduction phase
  IF cycle_credits > MAX_credits
      retirement phase
```

Figure 2. Main algorithm

```
FOR nSP editions
    FOR each CSP in SP
        select edition
        perform edition
        update credit

FOR nAP editions
    FOR each CSP in AP
      perform edition (arc extension)
      update credit
```

Figure 3. Reproduction phase algorithm

```
FOR each CSP in SP with credit<0
    IF CSP Value > 0
       restore credit
       move to AP
    ELSE move to DP

FOR each CSP in AP with credit<0
    output CSP

FOR each CSP in DP
    IF tail set close to target (threshold)
       restore credit
       move to SP
    ELSE delete CSP
```

Figure 4. Retirement phase algorithm

4 Analysis of our Evolutionary Approach

4.1 Motivating example: Denial of Services by E-mail Worm.

This example of a Denial of Services (DoS) attack by an e-mail worm was collected and adapted from Chinchani et al. [3, Section 4.2]. Figure 5 shows a graph representation of the attack in four stages.

In the first stage, an insider (node i denoting the insider computer) sends an e-mail to a coworker (node $n52$) containing an attachment, for example requesting review of an attached document. When the coworker opens the attachment, his computer gets contaminated, causing the original e-mail (worm) to be replicated and sent to e-mails contained in his address book. Thus, the e-mail worm from node $n52$ contaminates node $n61$ connected to a different mail server inside the Local Area Network (LAN), in stage 2. The same process happens on stage 3, where node $n61$ contaminates nodes $n62$ and $n63$. In stage 4, we see the DoS taking place with nodes $n61$, $n62$ and $n63$ flooding the mail server's capacity (e.g. bandwidth) (node $ms2$) with e-mails arriving within a short period of time, i.e. synchronised e-mails.

4.2 CSP representation of the example

Figure 6 shows the CSP attack specification which correspond to the DoS example.

The DoS on the target happens because arcs $Arc(n61, ms2)$, $Arc(n62, ms2)$ and $Arc(n63, ms2)$ are triggered together due to the semantics of the parallel composition. Thus, this attack involves one point of synchronisation in its last stage. This point is represented in the CSP specification by the last Par inside a Seq. It means that $Par[Arc(n61, ms2), Arc(n62, ms2), Arc(n63, ms2)]$ will only start after the previous Par has completed, and that $Arc(n61, ms2)$, $Arc(n62, ms2)$, and $Arc(n63, ms2)$ will start simultaneously.

4.3 Learning the example scenario

We consider a default network topology, adapted from Suehring [22] and illustrated in Figure 7, to construct a network graph to be used as input to the evolutionary algorithm. The network topology represents an organisational network that has a router which interfaces internal and external traffic, and is connected to four firewalls. Firewalls 1 and 2 interface with LANs 1 and 2 respectively, and firewalls 3 and 4 interface with servers 1 and 2 respectively.

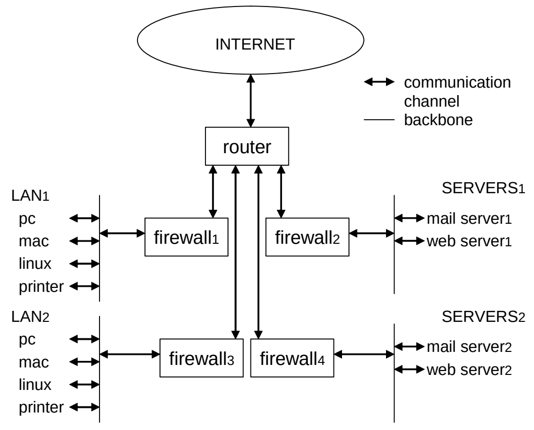

Figure 7. Default network topology adapted from Suehring [22]

We have implemented the algorithm in Haskell and performed a number of tests. We found that the algorithm is able to generate the example CSP, using the following input.

- An input network graph with up to 2000 nodes: we used a graph with 3 LANs (LAN1, LAN2 and LAN3). Nodes in LAN1 (nodes 1-) are connected to mail server

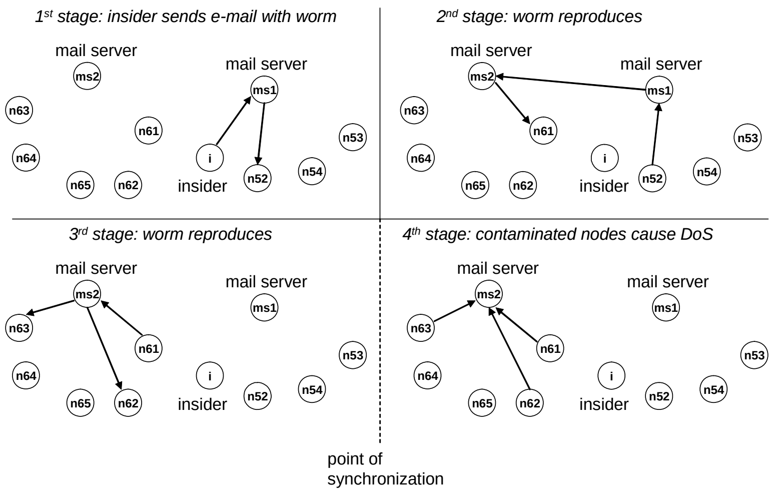
1st stage: insider sends e-mail with worm
2nd stage: worm reproduces
3rd stage: worm reproduces
4th stage: contaminated nodes cause DoS
mail server
ms1
ms2
n52
n53
n54
n61
n62
n63
n64
n65
i
insider
point of synchronization

Figure 5. Denial of Services by E-mail Worm

```
CSP = Seq[Seq[Arc(i,ms1),Arc(ms1,n52)],                                     (stage 1)
          Seq[Arc(n52,ms1),Arc(ms1,ms2),Arc(ms2,n61)],                      (stage 2)
          Par[Seq[Arc(n61,ms2),Arc(ms2,n62)],Seq[Arc(n61,ms2),Arc(ms2,n63)]], (stage 3)
          Par[Arc(n61,ms2),Arc(n62,ms2),Arc(n63,ms2)]]                      (stage 4)
```

Figure 6. CSP representing the attack scenario shown in Figure 5

represented by node 91, nodes in LAN2 (nodes 2-) are connected to mail server 92 and nodes in LAN3 (nodes 3-) are connected to mail server 93.

- The cost of each type of communication channel: this cost represents the difficulty the attacker will have to traverse the channel, thus it represents its level of protection provided. For example a ssh communication is more secure than a smtp connection and, consequently, the ssh cost should be higher than the smtp cost. In this case, the cost of the smtp connection for the example is set to 10.
- The initial set: our initial set is node 12 in LAN1.
- The added value of targets: our target is node 93. To simulate three attacking nodes (representing the limit of simultaneous connections that mail server 93 can handle) to node 93, we set the added value to $(1000.0, 92, 92, 92)$.

Figure 8 shows an output sample from the algorithm. The attacking node 12 contaminates node 21, located in another LAN. Node 21 has node 39 in its address book, although node 11 did not. Thus, the worm is propagated from node 39 to nodes 311, 312 and 313, yet in another LAN. These last nodes mount the DoS attack on the mail server, node 93. This CSP was produced from a network of 20 nodes with 1000 cycles. Its final attributes were: (i) head set = [12], (ii) tail set = [93], (iii) cost = 130 (13 arcs of 10), (iv) value = 1000, and (v) fitness = 870.

Although the output reproduced in Figure 8 has not many nodes involved, similar but more complex DoS scenarios have been reproduced with cycles ranging from 1000 for a network with 50 or 100 nodes, to 5000 for networks of up 2000 nodes. The algorithm found DoS-like attacks within a maximum of 30 minutes when using a 2000 nodes network in a Pentium 4/512MB RAM/2.8 GHz machine running Linux Ubuntu. These networks were all randomly generated and nodes were distributed among one to five LANs.

Proposition 1 *The complexity of the algorithm is $O(C * n^2)$, where n is the number of nodes in the graph and C is the number of cycles. (A newly created CSP is discarded if it has more than n^2 arcs.)*

Proof. Each cycle has a $(nSP + nAP + nDP)$ editions demanding time $O(n^2)$, given limit on the number of arcs of a CSP, and the fact that nSP, nAP, nDP are small constants compared to the size of the graph. The number of cycles in the main loop is in general equal or greater than n, which makes the algorithm run with a cubic upper bound. However, it must be observed that in general CSPs have

```
CSP = Seq[Seq[Arc(12,91),Arc(91,92],Arc(92,21)],                          (stage 1)
          Seq[Arc(21,92),Arc(92,93),Arc(93,39),Arc(39,93)],               (stage 2)
          Par[Arc(93,311),Arc(93,312),Arc(93,313)],                        (stage 3)
          Par[Arc(311,93),Arc(312,93),Arc(313,93)]]                        (stage 4)
```

Figure 8. Output sample produced by the algorithm

size much smaller than n because their size is limited by the credit they receive.

5. Conclusions and Future Work

We have presented an evolutionary-based algorithm which learns attack specifications representing attack scenarios from a network graph. We took the approach of validating the algorithm by modelling a known attack that is especially hard to represent because it requires a *point of synchronisation*. Thus, we used a Denial of Services by Email Worm attack as a motivating example. The algorithm was able to learn this type of attack from networks up to 2000 nodes. We believe that this type of tool can be valuable for administrators to acknowledge potential attack scenarios towards valuable assets.

Furthermore, we were also able to reproduce, with 700 cycles, a Distributed DoS (DDoS) attack[18], using a network with 150 nodes. For the execution of this attack, an attacker commands a set of contaminated machines, called masters, which listen for connections from the attacker on non-standard service port numbers. Another set of contaminated machines (called zombies), each one with its IP address registered with a master, listen for its master command to attack. Thus, this attack requires two *points of synchronisation*, one for the triggering of masters and one for the triggering of zombies. As a result, the zombies perform a DoS, for example, by launching a simultaneous packet flooding attack against the target. Figure 9 illustrates this generic DDoS attack.

We will focus next on further validating the algorithm for other known attacks which also involve synchronisation, coordination, concurrency, distribution, choice, and sequencing of attack steps.

In principle, the algorithm also applies for the assessment of vulnerabilities which involve a more complex structure, such as the ones within the domain of Access Control. In this case, we need to use a RBAC-based graph [13], which has several types of nodes and different relations between them, represented as arcs. Therefore, in this case, an hypergraph is needed, and we envision two ways to achieve this: (i) use the current algorithm and transform an hypergraph into a graph, by applying a conversion method described by Berge [1, Section 17.4], or (ii) incorporate the hypergraph into the algorithm. In this last case, instead of

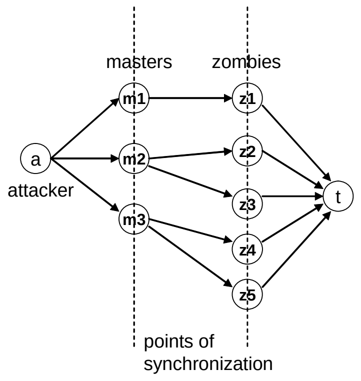

Figure 9. A generic Distributed Denial of Services attack

composing CSPs using head set and tail set, as we do now, we would need to compose them using further information.

In terms of the algorithm itself, we have presented an evolutionary-based approach that represents an improvement compared to traditional local search optimisation heuristics. In our approach, the search space is explored by many alternative solutions at the same time, like it happens with Genetic Algorithm for example, but the parent solutions remain active in the system of pools. Thus, a same parent solution has several opportunities to improve by generating more than one offspring.

References

[1] C. Berge. *Graphs and Hypergraphs*, volume 6 of *North-Holland Mathematical Library*. American Elsevier Pub. Co, second edition, 1975.

[2] R. Chinchani, A. Iyer, H. Q. Ngo, and S. Upadhyaya. Towards a Theory of Insider Threat Assessment. In *DSN 2005: Int. Conference on Dependable Systems and Networks*, pages 108–117. IEEE Publishing, July 2005. `http://ieeexplore.ieee.org/iel5/9904/31476/01467785.pdf`.

[3] R. Chinchani, A. Iyer, H. N. Q., and S. Upadhyaya. A Target-Centric Formal Model For Insider Threat and More. Technical Report 2004-16, University of Buffalo, US, October 2004.

[4] M. Dacier, Y. Deswarte, and M. Kaaniche. Models and Tools for Quantitative Assessment of Operational Security. In *IFIP SEC'96*, pages 177–186, May 1996.

[5] P. Defibaugh-Chavez, S. Mukkamala, and A. H. Sung. Efficacy of Coordinated Distributed Multiple Attacks (A Proactive Approach to Cyber Defense). In *AINA 2006: 20th Int. Conf. on Advanced Information Networking and Applications*, pages 10–14. IEEE Computer Society, April 2006. `http://doi.ieeecomputersociety.org/10.1109/AINA.2006.161`.

[6] D. Dolev and A. C. Yao. On the Security ofPublic Key Protocols. In *Proc. of the IEEE 22nd Annual Symposium on Foundations of Computer Science*, pages 350–357, 1981.

[7] S. P. Gorman, R. G. Kulkarni, L. A. Schintler, and R. R. Stough. A Network Based Simulation Approach to Cybersecurity Policy. `http://policy.gmu.edu/imp/research.html`. George Mason University, School of Public Policy.

[8] J. Green, D. Marchette, S. Northcutt, and B. Ralph. Analysis Techniques for Detecting Coordinated Attacks and Probes. In *Proc. of the Workshop on Intrusion Detection and Network Monitoring*, pages 1–9, Berkeley, CA, USA, 1999. USENIX Association.

[9] D. M. Gregg, W. J. Blackert, D. V. Heinbuch, and D. Furnanage. Assessing and Quantifying Denial of Service Attacks. *MILCOM'01: Military Communications Conference*, 1:76–80, 2001. `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=985767`.

[10] G. Helmer, J. Wonga, M. Slagell, V. Honavar, L. Miller, and R. Lutz. A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System. *Requirements Engineering*, 7(4):207–220, November 2002.

[11] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978. `http://doi.acm.org/10.1145/359576.359585`.

[12] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, second edition, June 2004. online version at `http://www.usingcsp.com/cspbook.pdf`.

[13] M. Koch, L. V. Mancini, and F. Parisi-Presicce. A GraphB-based Formalism for RBAC. *ACM Trans. Inf. Syst. Secur.*, 5(3):332–365, 2002. `http://doi.acm.org/10.1145/545186.545191`.

[14] S. Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, Department of Computer Sciences, 1995. `citeseer.ist.psu.edu/kumar95classification.html`.

[15] W. Mao. A Structured Operational Modelling of the Dolev-Yao Threat Model. In *Security Protocols 2002*, volume 2845/2003 of *LNCS*, pages 34–46, Berlin Heidelberg, 2004. Springer.

[16] R. Milner. *Calculus of Communicating Systems*. Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany, 1980.

[17] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999. `http://www.amazon.ca/exec/obidos/redirect?tag=citeulike04-20{\&}path=AS%IN/0521658691`.

[18] J. Mirkovic and P. Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *SIGCOMM Computer Communications Review*, 34(2):39–53, 2004. `http://doi.acm.org/10.1145/997150.997156`.

[19] C. Phillips and L. P. Swiler. A Graph-Based System for Network-Vulnerability Analysis. In *NSPW '98: Proc. 1998 workshop on New Security Paradigms*, pages 71–79, New York, NY, USA, 1998. ACM Press.

[20] S. Schneider. Formal Analysis of a Non-Repudiation Protocol. In *CSFW '98: Proc. of the 11th IEEE Workshop on Computer Security Foundations*, page 54, Washington, DC, USA, 1998. IEEE Computer Society.

[21] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated Generation and Analysis of Attack Graphs. In *SP'02: Proc. 2002 IEEE Symposium on Security and Privacy*, pages 273–284, Washington, DC, USA, 2002. IEEE Computer Society. `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1004377`.

[22] S. Suehring and R. L. Ziegler. *Linux Firewalls*. Novell Press, US, third edition, 2005.

[23] W. van der Aalst and K. van Hee. *Workflow Management Models, Methods, and Systems*. Cooperative Information Systems. The MIT Press, Cambridge, Massachusetts, 2002.