Agent-Based Network Protection Against Malicious Code

Martin Rehák[1], Michal Pěchouček[2], Jan Tožička[2], Magda Prokopová[1], David Medvigy[2], and Jiří Novotný[3]

[1] Center for Applied Cybernetics, Faculty of Electrical Engineering
[2] Department of Cybernetics, Faculty of Electrical Engineering,
Czech Technical University in Prague Technická 2, 166 27 Prague, Czech Republic
{mrehak,pechouc,prokopova,tozicka}@labe.felk.cvut.cz
[3] Institute of Computer Science, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
novotny@ics.muni.cz

Abstract. This paper presents an agent-based approach to Network Intrusion Prevention on corporate networks, emphasizing the protection from fast-spreading mobile malicious code outbreaks (e.g. worms) and related threats. Agents are not only used as a system-integration platform, but we use modern agent approaches to trust modeling and distributed task allocation to efficiently detect and also counter the attack by automatically created and deployed filters. The ability of the system to react autonomously, without direct human supervision, is crucial in countering the fast-spreading worms, that employ efficient scanning strategies to immediately spread farther once they infect a single host in the network.[1]

1 Introduction

This paper presents an application of classic agent technology techniques: trust modeling, meta-agents, computational reflection and distributed task allocation to protect a local or campus network against the spread of malicious code, e.g. worms [1]. The worms are a specific type of malicious code, that spreads across the computer networks, and uses known vulnerabilities of widely deployed software to infect the hosts, possibly perform malicious actions and spread further. The spread of the worm is influenced by its scanning strategy. Scanning strategy uses the current IP address as an input, and determines the addresses to propagate to, including the order in which the propagation is attempted. Strategies are typically defined in terms of various types of subnets [2] – for example, a Slammer worm uses a random scanning over the whole IPv4 address space, while the Code Red II scans local subnet with higher probability.

[1] This material is based upon work supported by the European Research Office of the US Army under Contract No. N62558-07-C-0001 and N62558-07-C-0007. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the European Research Office of the US Army. Also supported by Czech Ministry of Education grants 1M0567 and 6840770038.

H.-D. Burkhard et al. (Eds.): CEEMAS 2007, LNAI 4696, pp. 22–31, 2007.

Historically, there were many attempts to use agents in intrusion detection systems (IDS) [3]. However, these attempts have never matured into widely-deployed commercial solutions, and we argue that the computational power and techniques were not advanced enough to succeed. In our work, we aim to present a system design that efficiently protects the small and medium-sized networks against the worm attacks with minimum administrative overhead. Therefore, the performance and autonomy of the solution are crucial from the operational point of view.

Our detection approach, presented in Section 3 exploits the weak spot of worm propagation, the randomness of scanning strategy that determines the IP addresses targeted by the worm for future infection attempts. The randomness makes the spread less efficient, due to the exploration of non-existing hosts, and generates significant network traffic that can be detected and matched with the alarms raised by protected hosts (see Section 2). The matching is performed by *IDS Agents*, that use their *extended trust models* [4] to assess the trustfulness/maliciousness of individual network flows. The conclusions of the models are used by IDS agents to generate filtering policies, and to convert these policies into the bytecode (JAVA) by means of computational reflection. Filters are then deployed on network devices to limit the future spread of the worm by means of collective reflection process, based on established distributed task allocation methods [5].

2 System Architecture and Observation

The system, as presented in Fig 1, is designed to fulfill three principal functionalities: to **observe** the network traffic and host behavior, to detect malicious traffic and to react by deploying efficient filters on network devices to prevent the spread of the threat. This functionality is implemented by following elements:

• **Observation** is realized by two types of sensors: - **Host sensors** are deployed on selected *protected hosts* in the network and raise alarms when an exceptional behavior suggests a possibility of attack or intrusion attempt. **Network sensors** are located on network devices and capture the information about the network flows, together with the traffic statistics. The aggregated observations are then processed by IDS agents.

• **Detection** of malicious traffic is the primary responsibility of **IDS agents**, who process the information from the sensors using their trust model [4] in order to correctly correlate the alarms received from hosts with the current network flows. Their

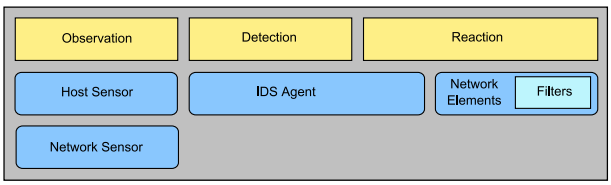

Fig. 1. System Components Overview

role is to correctly identify the malicious traffic and to generate a description of the malicious flow(s) for filter creation in reaction phase.

• **Reaction** phase is triggered upon attack detection by IDS agents. The agents generate one or more filters matching the attack and pass these filters to the network devices. These reflective [6] devices are able to autonomously adapt to changing environment. In our case, the adaptation is ensured by insertion of generated filters into their code and associated delegated operations.

When deployed, the system components use classic agent methods (e.g. directories or matchmakers) to discover each other and to maintain the system operational in the dynamic environment. The essential communication protocol in the observation and detection phase is `SUBSCRIBE` [2], which allows efficient information gathering by the IDS Agents.

The observation of attacks by the Host sensors is based on an assumption of randomized worm spread. As all currently known worms exploit a vulnerability specific for a single system type, we base our defence on an assumption that the deployed hosts are composed of wide variety of system configurations, and that most of these systems ar able to detect an unsuccessful (or even successful) attack, aimed at the vulnerability of other system. The detection of possible attack triggers an alarm that is sent to the IDS agents. We currently assume only simple, binary alarms without any additional information regarding the attack description. This maximizes the range of software/devices able to provide the alarm, and also makes the misuse of the mechanism more difficult, as the potential attacker can not directly specify the ”suspicious” traffic.

At the same time, network sensors capture the information specified in Table 1 about each individual network flow (unidirectional connection component identified by srcIP, dstIP, srcPrt, dstPrt and Protocol) and also provide this information to the same IDS agents that perform the detection process.

Table 1. Characteristics of the flow: Identity and Context (based on [7])

Feature	**Description**	**Feature**	**Description**
Connection Identity		***Connection Context***	
srcIP	Source IP Address.	count-src	Number of flows from the unique sources toward the same destination.
destIP	Destination IP	count-dest	Number of flows to unique destinations from the same source.
srcPort	Source Port	count-serv-dest	Number of flows to the same destination IP using the same source port.
destPort	Destination Port	count-serv-src	Number of flows from the same IP to the same port.
Payload Signature	First 256 bytes of the flow content (application headers)		

[2] www.fipa.org

3 Attack Detection

IDS agents use an extended *trust model* [4,8] to correlate the alarm received from the hosts with the observed traffic and to identify the malicious flows. The trust model inside each IDS agent processes the inputs that are provided by both host and network sensors in near-real time: sensors: (*i*) *network flows* information that contains the TCP/IP headers and first 256 bytes of the flow content [9], (*ii*) *statistics* of the existing network flows, listed in the context part of Table 1, and (*iii*) the *alarms* raised by Host sensors.

The trust model correlates the occurrence of alarms received from the hosts with the features of the relevant captured flows and the statistics associated with a particular flow. The features and the statistics form an *Identity-Context* feature space [4]. As the number of flows in an average network can be significant, it is important to process, represent, and maintain the data efficiently, and to keep the model synchronized with the most recent alerts and traffic. Therefore, instead of associating the information with individual flows in the *Identity-Context* space, the flows are aggregated into clusters. For each cluster, the IDS maintain its trustfulness, expressed as a fuzzy number [10]. The learning of trustfulness in our model is iterative, and each iteration consists of two stages. In the first stage, IDS agents generate and update the clusters using the Leader-Follower[3] algorithm [11], while in the second stage IDS agents update the trustworthiness that is associated with the centroids of the clusters that are adjacent to the observation representation in the Identity-Context space. For sake of efficiency, our implementation actually performs both stages in the same time, as we can see in Alg. 1.

Algorithm 1. Processing of the new observation.

```
Input: flow,situat,trustObs
closest ← nil;
mindist ← ∞;
id ← identityCx(flow,situat)
foreach rc ∈ rclist do
    dist ← distance(rc,id)
    if dist < mindist then
        closest ← rc
    wei = weight(dist)
    if wei > threshold then
        rc.updateTrust(trustObs,wei)
end
if mindist > clustsize then
    rclist.append(id)
    id.updateTrust(trustObs,wei)
else
    closest.updatePosition(id)
```

When IDS agent **observes** a flow, it extracts the identity features (see Tab. 1), then retrieves the associated statistics to determine the position of this vector in the

[3] Our current implementation uses LF clustering as it is efficient in terms of computational and memory cost, and allows on-line processing. Any other clustering algorithm can also be used instead, the reference points can even be placed arbitrarily [8].

Identity-Context space (*id* in Alg. 1) . Then, it computes the distance of the observed flow vector to each of the existing centroids ($rc \in rclist$), and updates the trustfulness of these centroids with a weight *wei* that decreases with growing distance. If the closest centroid is farther away than a predetermined threshold *clustsize*, a new cluster is defined around the new observation. When the model is **queried** regarding the trustfulness of a specific flow vector, it aggregates the trustfulness associated with the centroids, with a weight that decreases with the distance from the centroid to the flow representation in the Identity-Context space. Once it has determined the trustfulness, in a form of a quasi-triangular fuzzy number, it calculates inferences with a high trust and low trust fuzzy intervals. These intervals represent the knowledge about the normal level of trustfulness in the environment, and allow the IDS agent to infer whether the flow is trusted or not under current conditions [10].

The output of the detection phase is a selection of malicious traffic, specified as a cluster (or a group of clusters) in the identity-context space. This description, together with the examples of typical cluster members is then used to generate a traffic filter that only use the identity part of the features from Table 1, which can be matched with actual flows.

4 Reaction to Attack

As the detection uses the actual infections to detect the malicious traffic, we must assume that at least some of the vulnerable hosts in the network were already infected by the detected threat. Therefore, the filter generated by the detection upon attack discovery must not only protect the network interfaces (firewalls, VPN access,...), but shall be deployed at least once on each path between any two hosts on the network. If the filter can not be deployed in some network parts, these parts shall be considered as a single host by the protection mechanism and treated as such.

As there may be multiple filters deployed on the network in the same time, and a big part of network devices may be unable to perform the complete filtering of the passing traffic on their own, we use the distributed task allocation methods to find optimal delegation of filtering tasks between the network devices.

The delegation of filtering is based on negotiation among two or more nodes. As a result of the negotiation one node would be dedicated to provide filtering services to the other nodes. Such extension can increase the number of deployed filters, but increases also the communication traffic (as the flows must be tunneled to filtering devices). In case of delegation, outlined in Extended Filter Deployment Algorithm (EFDA, see Alg. 2), the agent uses Extended Contract Net Protocol (ECNP) [5] to ask trusted agents (determined by policy or a trust model) to perform filtering on its behalf, instead of local filter deployment.

One of the main features of Extended Contract Net Protocol compared to simple Contract Net Protocol [12] is that the participants can propose to fulfill only part of the task. The initiator agent starts negotiation by sending task specification to all trusted agents and is waiting for proposals with specified costs. In case participant agent decides to propose for given task it reserves all resources necessary for fulfilling task or part of it and sends proposal to the initiator. At this point participant agent can't cancel its

Algorithm 2. EFDA Algorithm

$\Theta =$ all trusted agents.
$\Phi =$ set of my filters.
$\Psi = \emptyset$.
repeat
  Send Call-For-Proposals to all agents Θ.
  Wait for all proposals Π.
  Choose the winner A:
    the agent that proposes minimal average price for deploying filters F.
  $\Psi = \Psi \cup \{A\}$
  $\Phi = \Phi \setminus F$
  Send Temporary Accept to agent A.
  Send Reject to agents $\Theta \setminus \{A\}$.
until $\Phi = \emptyset$ *or* $\Pi = \emptyset$;
Send Accept to all agents Ψ.

offer by itself and is waiting for the initiator answer. After initiator receives proposals covering parts of the task, the agent with best proposal is given temporary accept. After temporary accept is received, the agent is committed to the task and is waiting for final accept to start performing the task. Initiator sends rejects to all agents but the accepted one and thus they can cancel reservation of their resources.

New negotiation turn is taken for the remaining part of the task and again all trusted agents receive the task specification. After whole task being distributed among participant agents or getting to the point where no more agents can propose to fulfill any part of the task, the initiator agent accepts a consistent subset of proposals that were previously temporarily accepted. At this point both initiator and participant agents are fully committed.

In our domain the task is represented by all of the filters agent is willing to delegate. Participant agents after receiving the task specification are willing to cooperate and create proposal for as many filters as possible. The cost for each filter combines the distance of the agents (initiator and participant - this distance corresponds to the growth of the traffic in the network) and computational costs/price for deploying the filter. When task contains filter that has already been deployed on participant node the price for deploying is zero and thus the proposed filter cost is based only on distance. According to the strategy we want to follow the importance of distance or computational cost in the final price for the task can be changed. The higher the importance of distance is the less filters can be accommodated in the network, but on the other hand the growth of additional network traffic is less significant.

Our task is to deploy as many filters in the network as possible, but at the same time we are trying to keep the additional network traffic as low as possible. Therefore the delegation of filters starts at the point when no more filters can be deployed on local network device.

Using Filter Delegation. Figure 2 shows simple situation of filter delegation. $Switch\ A$ didn't have enough free computational capacity to deploy all filters available, so the EFDA assigned one of additional filters to $Switch\ B$ located slightly off the way. Therefore every-time traffic is sent from $Host\ 1$ to $Host\ 2$ it is forwarded by $Switch\ A$

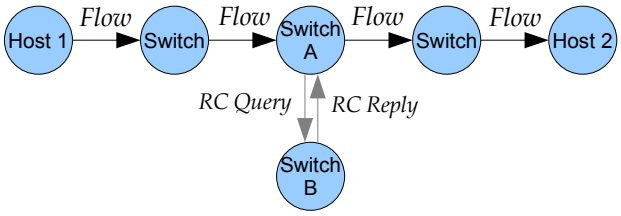

Fig. 2. Using filter delegation $RC\,Query$ is created for traffic from $Host\,1$ to $Host\,2$. $RC\,Query$ contains original flow, which is either returned if determined as not malicious or dropped in other case.

to $Switch\ B$ that contains remaining filters. Special filter called *filter-for* deployed on $Switch\ A$ encloses the traffic in new flow and marked as $RC\ Query$ to be easily recognizable by switch it is addressed to. After receiving $RC\ Query$ device performs filtering and in the case that flow is considered to be safe it is sent back enclosed in $RC\ Reply$ flow otherwise it is dropped. Network devices don't implement cache to store flow content while waiting for $RC\ Reply$ and thus the whole flow needs to be forwarded although it means increase in network traffic. Forwarding all the traffic would cause excessive increase in network traffic and therefore each filter-for contains port number of traffic it relates to and forwarding is performed only for traffic containing this port number.

5 Experimental Results

As a testbed for our experiments we used an agent-based network simulation called Anet (Agent network). Several tests have been performed to evaluate the above-presented strategies and to identify their limitations. Scenario used for our experiments reported below contains 170 hosts of two different types. This means that each worm can infect approximately half of the population and the other half rises alarm upon receiving vulnerable flow. The propagation strategy that the worms are using to spread through the network is essential for our method - we have implemented a common strategy (similar to the one employed by Code Red II or Blaster worms) where worm is choosing IP address pseudo-randomly. There is 50% chance that the IP address will be from the same C-subnet as it's source address, and a 50% chance it will be from the same B-subnet. This strategy actually favors propagation on our simulated network, and is therefore a worst-case scenario – many worms scan local subnet with higher priority and than the whole IPv4 space.

In experimental scenario, we launch a sequence of worms into the system. Each worm starts spreading as soon as the last one has finished the spread, plus the time to terminate the reaction of the protective system. We observe the percentage of host that gets infected by worm as well as the percentage of malicious worm flows that are deleted by our filters before reaching their destination. These two parameters are related: more flows we filter, less hosts shall get infected, as we can clearly see in

Table 2. Percentage of infected hosts/filtered flows in experimental runs on identical network. Differences are due to the scanning strategy influence.

Experiment	**First worm**		**Second worm**	
Experiment	**% Filtered Flows**	**% Infected Hosts**	**% Filtered Flows**	**% Infected Hosts**
1	86%	09%	19%	93%
2	93%	07%	00%	98%
3	50%	22%	00%	00%
4	65%	51%	33%	54%
5	88%	01%	44%	86%
6	93%	01%	00%	85%
7	90%	03%	93%	28%
8	46%	39%	11%	68%
9	60%	07%	00%	51%
10	90%	13%	03%	30%
Avg	**76%**	**15%**	**23%**	**59%**

the results. On the other hand, the inherent randomness of the worm spread causes important variations in results presented in Table 2.

In the results, we can see that the system in its current stage of development is able to handle a single virus outbreak, and to spread its spread rather efficiently. The results clearly show that the system reacts only once the attack is reported by non-vulnerable hosts – it is not designed to prevent the outbreak, but to contain it and to limit its impact.

The results for second and all subsequent worms are rather similar – while the system is able to reduce the impact of the attack, its influence is limited and about 60% of vulnerable hosts are infected on average (compared to 15%) in case of the first worm. We attribute this degradation of performance to two factors - the detection module data is influenced by the activity of the first worm, and the capacity of network devices for filter deployment is already partially used.

6 Related Work

There are two fundamental approaches to network intrusion detection: *anomaly detection* and *signature detection*. Most commercial and existing open-source products fall into the *signature-detection* category: these systems match the packets in the network with the predefined set of rules [13]. While this system can be valuable in detecting known attack patterns, it provides no protection against novel threats that are not represented in its knowledge base. Furthermore, as the rules are typically sparse to achieve efficiency, the signature-based IDS have a non-negligible false positives ratio. Another major disadvantage of the existing systems is a high computational cost of reasonably complete set of rules – currently, complete SNORT database is not typically entirely deployed due to the computational power limitations.

The *anomaly detection* approaches in NIDS are typically based on classification/ aggregation of sessions or flows (unidirectional components of sessions) into classes and deciding whether a particular class contains malicious or legitimate flows. While

these approaches are able to detect novel attacks, they suffer from comparably high false-positive (or false negative depending on the model tuning) rate. Most research in the NIDS area currently delivers a combination of signature and anomaly detection and aims to improve the effectiveness and efficiency of intrusion detection. In the following paragraphs we are going to present a brief overview of selected (N)IDS systems [3]. Two of the recent representative systems are MINDS and SABER.

MINDS system [7] is an IDS system incorporating signature and anomaly detection suitable for host and network protection. As an input for traffic analysis it uses flow information which is aggregated in 10 minutes intervals. It extracts time-window and connection-window based features, from the traffic, to detect both fast and stealth attacks. The signature detection is used for protection against known threats. The anomaly detection the the principle of local outlier factor by which an unusual traffic is discovered. The human operator then decides which anomalous traffic is actually malicious and can use the captured information do define the signatures for subsequent traffic filtering.

SABER [14] combines both detection (anomaly and signature) and protection mechanisms. It incorporates a DoS-resistant network topology that shields valuable resources from the malicious traffic, and supports a business process migration to the safe location upon the attack. Real-time host-based IDS monitors program's use of Windows Registry and detects the actions of malicious software. The analogous approach has been used for file-based anomaly detection in Unix environment, when the IDS monitors a set of information about each file access. SABER is completed by autonomic software patching mechanism [15] which tries to automatically patch vulnerable software by identifying and correcting buffer overflows in the code, using the information from specialized honeypots operated alongside production machines.

7 Conclusions and Future Work

In our work, we present a multi-agent approach to intrusion detection and response, which relies on a combination of established IDS techniques (traffic observation, preprocessing and filtering) with efficient multi-agent approaches – trust modeling and distributed task allocation.

Our approach was evaluated in a high-level simulation of a campus-like network, featuring approx. 180 hosts and 30 network devices. Hosts were split in two groups, roughly of the same size, each with a different set of vulnerabilities. We have assumed that the network is protected by Firewall/NAT from direct attacks, and that the infections propagate from a single host (infected host connected by VPN, connected infected laptop, etc.). In this setting, the system is able to significantly reduce the spread of simulated worm, reducing the infection rate to approx. 20% of hosts infected before the attacks are reported, detected and the filter is crated and deployed on network devices.

In our current work, we concentrate on handling of multiple intrusions, prioritization of filtering and other issues where a rigorous application of multi-agent principles can tackle the real network-security problems.

References

1. Moore, D., Shannon, C., Voelker, G.M., Savage, S.: Internet quarantine: Requirements for containing self-propagating code. In: INFOCOM (2003)
2. Stallings, W.: Data and computer communications, 5th edn. Prentice-Hall, Inc., Englewood Cliffs (1997)
3. Axelsson, S.: Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ. (2000)
4. Rehak, M., Pechoucek, M.: Trust modeling with context representation and generalized identities. In: Cooperative Information Agents XI. LNAI/LNCS, vol. 4676, Springer, Heidelberg (2007)
5. Fischer, K., Muller, J.P., Pischel, M., Schier, D.: A model for cooperative transportation scheduling. In: Proceedings of the First International Conference on Multiagent Systems, pp. 109–116. AAAI Press / MIT Press, Menlo park, California (1995)
6. Maes, P.: Computational reflection. Technical report 87-2, Free University of Brussels, AI Lab (1987)
7. Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P.N., Kumar, V., Srivastava, J., Dokas, P.: Minds - minnesota intrusion detection system. In: Next Generation Data Mining, MIT Press, Cambridge (2004)
8. Rehak, M., Gregor, M., Pechoucek, M., Bradshaw, J.M.: Representing context for multiagent trust modeling. In: IAT'06. IEEE/WIC/ACM Intl. Conf. on Intelligent Agent Technology, USA, pp. 737–746. IEEE Computer Society, Los Alamitos (2006)
9. Haffner, P., Sen, S., Spatscheck, O., Wang, D.: Acas: automated construction of application signatures. In: MineNet '05. Proceeding of the 2005 ACM SIGCOMM workshop on Mining network data, pp. 197–202. ACM Press, New York (2005)
10. Rehák, M., Foltýn, L., Pěchouček, M., Benda, P.: Trust model for open ubiquitous agent systems. In: Intelligent Agent Technology, 2005. IEEE/WIC/ACM International Conference, vol. PR2416, IEEE, Los Alamitos (2005)
11. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd edn. John Wiley & Sons, New York (2001)
12. Smith, R.G.: The contract net protocol: High level communication and control in a distributed problem solver. IEEE Transactions on Computers C-29, 1104–1113 (1980)
13. SNORT intrusion prevention system (2007) (accessed, January 2007), `http://www.snort.org/`
14. Keromytis, A.D., Parekh, J., Gross, P.N., Kaiser, G., Misra, V., Nieh, J., Rubenstein, D., Stolfo, S.: A holistic approach to service survivability. In: Proceedings of the 2003 ACM Workshop on Survivable and Self-Regenerative Systems (SSRS), pp. 11–22. ACM Press, New York (2003)
15. Sidiroglou, S., Keromytis, A.D.: Countering network worms through automatic patch generation. IEEE Security & Privacy 3, 41–49 (2005)