World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:6, 2007

Identifying Attack Code through an Ontology-Based Multiagent Tool: FROID

Salvador Mandujano

***Abstract*—**This paper describes the design and results of FROID, an outbound intrusion detection system built with agent technology and supported by an attacker-centric ontology. The prototype features a misuse-based detection mechanism that identifies remote attack tools in execution. Misuse signatures composed of attributes selected through entropy analysis of outgoing traffic streams and process runtime data are derived from execution variants of attack programs. The core of the architecture is a mesh of self-contained detection cells organized non-hierarchically that group agents in a functional fashion. The experiments show performance gains when the ontology is enabled as well as an increase in accuracy achieved when correlation cells combine detection evidence received from independent detection cells.

***Keywords*—**Outbound intrusion detection, knowledge management, multiagent systems, ontology.

I. INTRODUCTION

PREVIOUS research in the area of intrusion detection has considered the use of software agents for building security monitoring tools [1]. A number of prototypes based on autonomous agents have been developed but no ontological constructs have been actually implemented in order to allow agents to collect and share information in a semantically rich format that enables more intelligent conducts. Similarly, a variety of programming languages have been used for these implementations but, unlike other agent-based applications in areas such as negotiation and knowledge sharing, just a few intrusion detection tools have taken advantage of actual agent frameworks and libraries to build such systems [2].

The potential benefits of using knowledge representation techniques in the area of computer security have been already acknowledged [3]. A smooth, maintainable integration with other technologies and a reduction to the amount of data to be transferred through message passing are two potential advantages of the approach that this paper addresses.

We built the *FROID* (First Resource for Outbound Intrusion Detection) system in order to explore the possibilities that an ontology-based intrusion detection system can offer. Being social behavior one of the key characteristics of multiagent systems, a common interpretation of the environment is necessary to endow intrusion detection agents with inference and knowledge sharing capabilities that help them speed up data filtering and decision making. An attacker-centric ontology [4] is the underlying layer of the FROID multiagent architecture whose purpose is to identify automated remote-attack tools in execution.

The system follows the *outbound intrusion detection* paradigm [5], a collective approach to security monitoring that aims at protecting a society of nodes by guaranteeing that each member monitors its own outbound traffic for signs of malicious activity. By distributing the monitoring workload among all member nodes, rather than positioning the detection mechanism at intermediate choke points, the society as a whole achieves security monitoring by making sure that no local processes running on any of the nodes attempt to launch an attack towards others.

FROID was built using the FIPA-compatible JADE agent framework and OWL was used to define the ontology that represents the components of the environment agents inhabit. The design of the tool departed from the characterization of automated remote-attack tools by means of two independent, yet complementary signature types (i.e., network traffic and process execution signatures) as well as pattern matching data structure based on the internals of *Snort* [6].

II. BACKGROUND

A. *Outbound Intrusion Detection*

Outbound Intrusion Detection (*OID*) is a variant to the problem of intrusion detection. Its objectives differ from those of traditional intrusion detection systems but both approaches share fundamental aspects such as their monitoring nature and the type of security threats they handle. OID focuses, not on protecting local resources from being compromised, but on preventing those resources from being used to compromise other systems. It aims at containing the impact of a security-relevant incident by monitoring outbound activity at the host level so that the system cannot be utilized as an attack launcher or intrusion relay.

B. *Ontologies and Semantics*

For humans, knowing the relationships among entities in a certain context allows us to infer information not explicitly communicated. Ontologies are knowledge representation models that can be used to enable that kind of behavior in software [1]. They capture the description of an environment so that a unique interpretation can be shared among all

Manuscript received May 20, 2005. This work was supported in part by the Information Security Research Grant of ITESM Monterrey (Cátedra de Seguridad Informática del ITESM Campus Monterrey.)

Salvador Mandujano, Center for Intelligent Systems, Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM), Monterrey, NL 64849, Mexico (e-mail: smv@itesm.mx).

interested parties. Ontologies go beyond lexicographic representation and querying by allowing programs to perform logic operations on the data they handle.

One of the limitations that agent-based intrusion detection software has is the lack of autonomy of its agents. Recent work in the area of knowledge representation has proposed the use of ontologies for security-related problems [7]. We leverage that idea and experimentally evaluate the potential of using a semantic layer for the problem of outbound intrusion detection. The ontology the system uses is based on the perspective on the attacker. This is assumed to be able to probe other systems while logged on into a local host.

OWL [8] is an XML/RDF-based language that evolved from DAML+OIL and became one of the richest ontology languages available. To define a class derived from a superclass, for instance, FROID utilizes the following syntax:

```
<owl:Class rdf:ID="ProcessSignature">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Signature"/>
  </rdfs:subClassof>
</owl:Class>
```

All agent-communicated messages are based on the ontology. For the sender it is not necessary to indicate that *ProcessSignature* is a *Signature*. The knowledge stored in the ontology, which is shared among all agents, is accessed by the recipient upon getting the message in order to run the interpretation. This is how message contents can be optimized and data exchanges reduced.

C. Trends in Cybercrime

Over the last years, there have been some trends in information security that have changed the priority of certain threats hereby generating new requirements. One of them is the need to develop solutions that deal with the increasing number of attack tools available on the Web. This type of software can be easily downloaded and executed by many users allowing them to perform system fingerprinting, scanning and actual vulnerability exploitation [8].

Not much expertise is required for running some applications and this has contributed to an increase in the number of users who explore the capabilities of this software. Along the same lines goes the increasing number of attacks launched remotely that target systems connected to the Internet. The count of internal attacks has been decreasing whereas security reports indicate external attacks continue on the rise due to increased connectivity around the globe [8].

III. Design Considerations

The objective of the system is to use an ontology-based multiagent architecture to identify remote-attack program signatures (for a detailed ontology description see [4]).

A. Characterization of remote attack tools

Many different aspects may be considered when characterizing programs in execution. We have used two primary aspects: (1) the peculiarities of the network traffic generated by the tools, and (2) the resources accessed by its process(es) when in execution. The former is an approach widely used by network-based intrusion detection tools. The difference is that an OID system has access to details such as how packets are being generated (e.g., how source IP addresses and/or port numbers are being locally modified, or how the sender is forcing fragmentation to bypass a firewall). This information is not typically accessible to traditional intrusion detection systems, which are typically placed on the victim's side.

Analyzing the specific resources accessed by an attack tool also helps identify its execution. This is because execution details such as the data and text segments uploaded into primary memory, the libraries and other files opened by the program, as well as the sequence of system calls requested to the kernel can be, as a whole, distinctive enough as to identify a program from the rest.

B. Test Set and Signature Definition

In order to identify the fields to include in the signatures, we exercised a group of 50 remote attack programs downloaded from the Web. The selected programs exploit vulnerabilities on different services through TCP/IP connections and all run on Linux (kernel 2.1 was used for the testing). We further limited our tool collection to programs that: use socket connectivity, target Windows, Linux or CISCO platforms, were written in C, C++ or PERL (our research indicates these languages cover the majority of attack tools), and those whose source code is freely available.

The programs were exercised with five different parameter sets. With every parameter set, each tool was executed multiple times for consistency (see section IV). Network traffic is, in general, not well behaved, but all our trails were consistent. Both, the outgoing network traffic streams (the first 100 packets per execution) as well as */proc* profiles were captured. We then applied entropy analysis to the data in order to identify the most relevant fields.

For the purpose of avoiding biased results, we used three entropy analysis rules commonly utilized for growing decision trees. These are:

(1) $Entropy(s_i) = \sum_{v \in Values(s_i)}^{k} - p(v) \log_k p(v)$

(2) $InfoGain(s_i) = Entropy(s_i) - \sum_{v \in Values(s_i)}^{k} \frac{|s_{i,v}|}{|s_i|} Entropy(s_{i,v})$

(3) $GINI(s_i) = 1 - \sum_{v \in Values(s_i)}^{k} p(v)^2$

The amount of information each field conveys was quantified and a ranking was created. From the results we found out that the first 16 packets of each stream did suffice to identify with 93% confidence the originating program.

Every tool can be executed with different parameter combinations. Through experimentation we found that it is difficult to define a single signature per program to enclose all the different execution paths the program takes as determined by the parameters supplied by the user.

Executions from the same parameter combination, however, can be easily grouped together. In the example of Fig. 1, three programs are executed three times each with two parameter combinations. Three attributes compose the sample signature. *Program 3*, for example, is represented by rings. Even when the three executions corresponding to the same single parameter combination are clustered together, the other parameter combinations generate totally different clusters that stand separate from each other. With more programs and many more attributes, generating a single signature per program becomes more difficult. Based on these findings, FROID identifies independent tool executions depending on the parameter combination used.

Signatures are composed of high-entropy attributes that help define crisp clusters. The network signature fields are:

Network signature = {source port, source address, destination port, destination address, fragmentation, window size, {$flags_1$, $flags_2$, …, $flags_k$}, {$hash_1$, $hash_2$, …, $hash_k$}}

Where k represents the number of packet considered for detection; $flags_i$ are the *ACK, SYN, FIN* and *RST* flags of each TCP packet; and $hash_i$ represents the MD5 hash of the packet's payload. Process signatures contain:

Process signature = {program name, text segment checksum, number of files opened, {$file_1$, $file_2$, …, $file_k$}, number of dynamic libraries opened, {$dlibrary_1$, $dlibrary_2$, … , $dlibrary_k$}, memory allocated at upload, file hash}

Different modifications to the execution realm may lead to inaccuracies during detection, however, we focus on the majority of the cases [9] in which users will download and immediately execute the tool.

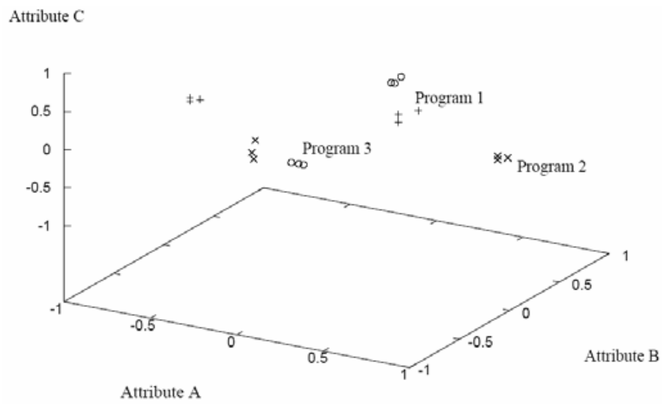

Fig. 1 Clustering program signatures

C. Multiagent Architecture

Multiple agent-based intrusion detection implementations were reviewed. In general, they are composed of three agent types: *sensors* in charge of collecting information, *processors* in charge of operating on that information, and *action* agents responsible for reporting the findings (most tools do not have reaction capabilities built in).

We used these types of agents as the foundation of the architecture and built a non-hierarchical system that groups agents into *cells*. Cells have the purpose of providing an environment whose integrity can be easily verified. Being the malicious host a challenge to agent software, in particular to mobile agent applications, having an execution subenvironment in software makes security evaluation easy.

The system was built in Java using the JADE agent generation framework, the JENA ontology-processing library, and the OWL XMP/RDF-based ontology definition language. Cells are implemented as JADE containers by means of Java virtual machines. The basic agent types are: sensors (*S*), correlators (*C*), and reactors (*R*). These are combined into five different cell types. Some cells perform signature generation and some do signature detection:

- *Traffic Signature Generation cells (TSGC).* Generate traffic signatures and are composed of S and C agents.
- *Traffic Signature Detection cells (TSDC).* Detect traffic signatures and are composed of S, C, and R agents.
- *Process Signature Generation cell (PSGC).* Generate process signatures and are composed of S and C agents.
- *Process Signature Detection cell (PSDC).* Detect process signatures and are composed of S, C, and R agents.
- *Correlation cell (CC).* Receive input from PSDC and TSDC cells in order to identify signatures more accurately. Are composed of S, C and R agents.

Detection cells are fully independent and self-contained and perform detection tasks by themselves. Correlation cells are only used to provide higher detection accuracy by incorporating input from process and network traffic cells.

D. Signature Matching

Signature matching is an important component of intrusion detection technologies. In particular, it is necessary to develop efficient data structures and algorithms that allow the detection mechanism to process all the evidence. Snort is a tool used for identifying anomalous network patterns. Part of its success is the simplicity of the signatures it handles and the flexibility this provides [6]. We extended the basic data structure of Snort with additional dimensions required to capture our signatures (Fig. 2). It is important to mention that we support a permanently sorted data structure that places high-entropy attributes at the front of the lists.

IV. Operational Outline

Two data streams are the input to the architecture: outbound network traffic and process execution data. Signature generation cells read these streams and extract information at the time the attack tool is executed during training. They then share the signatures with local monitoring agents and alternatively upload signature updates into a signature repository. Traffic monitoring cells permanently screen TCP network packets looking for a match with the existing signatures. At the same time, process monitoring cells do the same by monitoring */proc* profiles (the detection trigger is the request of an open socket system call to the kernel). These two monitoring cells may independently identify an attack tool. An additional correlation cell receives these detection hypotheses can more precisely identify the attack program that is being executed.

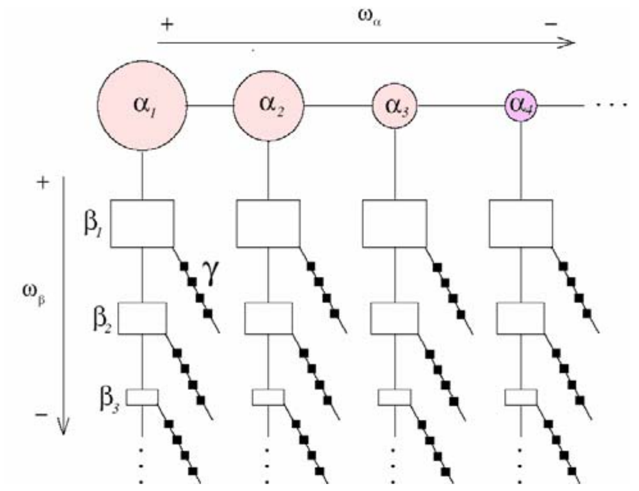

Fig. 2 Snort-based, 3D data structure

V. Experimantal Results

The objective of the experimentation with FROID was two-fold: to evaluate performance and to evaluate accuracy. The system was exercised with 50 attack tools, each of them executed with five different parameter variations. Each variation was executed five times for a total of 1,250 samples.

1. *Performance:* Ontology-disabled. A network monitoring cell takes 2.2 sec in average to generate an hypothesis using programmatic structures. In this scenario, messages exchanged between agents contain all the information needed for interpretation (i.e., no ontology use). A process monitoring cell takes 2.8 sec to produce a hypothesis. When correlation agents are enabled, the generation of a hypothesis takes 2.6 sec in average. This is most likely due to (1) the time it takes for the detection cells to send input to the correlator, in addition to (2) the time it takes to run through the inference rules that allow the correlator to produce its output.

2. *Performance*: Ontology-enabled. All agents have a copy of the OID ontology. The size of the messages is reduced around 56% *and* the interpretation task takes less time. A process monitoring cell will take 2.0 sec in average to generate a hypothesis using the ontology, whereas a network monitoring cell takes around 1.7 sec. When correlation is enabled, detection time goes up to 2.7 sec. This is due to the additional interpretation of the ontology structures necessary to derive the correlator´s hypothesis.

3. *Accuracy*. In the two scenarios used before, detection accuracy is not very different – as it was expected. With ontology disabled and enabled, process cells correctly identify the attack tool in execution with probability 0.81 and 0.85, respectively. Traffic cells are less precise due to packet misses and payload variations. They detect the correct attack tool with probability 0.78 and 0.76, respectively.

Integrating independent detection input into a correlator, significantly increases detection accuracy. When correlation cells are enabled to combine traffic and process detection hypotheses, the average detection accuracy from 1,250 executions is 0.91, which is considerably higher than the detection performed by the most accurate detection cell (0.85, process detection cells with ontology enabled).

VI. Conclusion

The integration of an ontology into an outbound intrusion detection tool built with agent software shows performance gains due to reduced message size and accelerated knowledge interpretation supported by inference-support structures uploaded into the agents' RAM (less lexicographical processing, more inference capabilities.) By combining inputs from independent detection engines that access different evidence sources a more accurate and robust detection mechanism can be built through the use of correlators that prioritize information pieces by their entropy value.

Acknowledgment

We thank Dr. Juan Arturo Nolazco-Flores (Department of Computer Science, ITESM Monterrey) and Dr. Arturo Galván-Rodríguez (Center for Intelligent Systems, ITESM Monterrey) for their invaluable support to this project.

References

[1] X. Guan, Y. Yang and J. You. G. O. Young, "POM - A mobile agent security model against malicious hosts", *Proceedings of the 4th International Conference on High-Performance Computing in the Asia-Pacific Region*, vol. 2, pp. 1165-1168, May 2000.
[2] D. Lange and M. Oshima, "Programming and deploying Java mobile agents with Aglets", Addison-Wesley Press, Menlo Park, CA, 1998.
[3] V. Raskin, C. Helpenmann, K. Triezenberg, and S. Nirenburg, "Ontology in information security: a useful theoretical foundation and methodological tool", *New Security Paradigms Workshop*, ACM Press, pp. 53-59, Cloudcroft, NM, 2001.
[4] S. Mandujano, A. Galván, J. A. Nolazco, "An Ontology-based Multiagent Architecture for Outbound Intrusion Detection", *3rd ACS/IEEE International Conference on Computer Systems and Applications, AICCSA ´05,* vol. 1, pp. 120-128, Cairo, Egypt, January 2005.
[5] S. Mandujano and A. Galván, "Outbound Intrusion Detection", *Proceedings of the International Computer, Communications and Control Technologies, CCCT 04*, vol. 1, pp. 68-73, Austin, TX, Nov. 2004.
[6] C.J. Coit, S. Staniford, and J. McAlerney, "Towards Faster String Matching for Intrusion Detection or Exceeding the Speed of Snort", *DARPA Information Survivability Conference and Exposition (DISCEX II),* vo1. 1, pp. 132-139, Anaheim, CA, June, 2001.
[7] J. Undercoffer, A. Joshi,, T. Finin, and John Pinkston, "A target centric ontology for intrusion detection: using DAML+OIL to classify intrusive behaviors", *Knowledge Engineering Review,* Cambridge University Press, pp. 23-29, January, 2004.
[8] P. Schneider, P. Hayes, I. Horrocks, F. Van-Harmelen, "Web Ontology Language (OWL): abstract syntax and semantics", working draft, W3C web consortium, November, 2002.
[9] P. Rapalus et al., "CSI/FBI Computer Crime & Security Survey 2004", *Computer Security Institute and Federal Bureau of Investigations*, April, 2004.