Ant-Based Cyber Security

Jereme N. Haack, Glenn A. Fink, Wendy M. Maiden,
A. David McKinnon

Pacific Northwest National Laboratory
Richland WA, USA
Jereme.Haack@pnl.gov

Steven J. Templeton
University of California, Davis, Dept. of Computer Science
Davis, CA, USA
templets@omsoft.com

Errin W. Fulp
Wake Forest University, Computer Science Department
Winston-Salem, NC, USA
fulp@wfu.edu

Abstract—**We describe a swarming-agent-based, mixed-initiative approach to infrastructure defense where teams of humans and software agents defend cooperating organizations in tandem by sharing insights and solutions without violating proprietary boundaries. The system places human administrators at the appropriate level: where they provide system guidance while lower-level agents carry out tasks humans are unable to perform quickly enough to mitigate today's security threats. Cooperative Infrastructure Defense, or CID, uses our ant-based approach to enable dialogue between humans and agents to foster a collaborative problem-solving environment, to increase human situational awareness and to influence using visualization and shared control. We discuss theoretical implementation characteristics along with results from recent proof-of-concept implementations.**

Keywords—agents; security; digital ants; digital pheromone

I. INTRODUCTION

Accomplishing concerted, infrastructure-wide cyber-defensive action that spans organizational boundaries is difficult. Infrastructures have unique cyber-security needs that cannot be adequately addressed by individual enterprise solutions, and the complicated relationships in infrastructures make it possible for defensive actions by one organization to affect the others adversely [1]. Both legal [2],[3] and practical issues require that the consequences of change be managed through coordination across organizations in a near-real-time manner.

Cyber adversaries, on the other hand, are not hindered by central coordination—they can rapidly and concertedly disrupt multi-organizational computer infrastructures. Therefore, enclave defenders need to be able to coordinate the activities of their defenses without violating the sensitivities of cooperating organizations. For example, if one organization is an Internet service provider, it might be convenient to be able to push security updates to reliant organizations to keep all of them up to date. These organizations, however, would wish to retain control for themselves, because new updates may break mission-critical legacy applications. Humans must not become a bottleneck, but automated defenses must strictly observe access policies that differ across cooperating organizations. The mixed-initiative approach provides a basis for shared control among humans at different sites and for humans and software agents at different sites to collaborate.

Current cyber-defense systems involve humans at multiple levels, but people are often far down in the control structure, requiring them to make too many time-critical decisions. Information flow between humans is slow and frequently asynchronous. In a crisis, humans may be unable to cooperate because of cultural, language, legal, proprietary, availability, or other obstacles. Such systems cannot adapt to rapid cyber threats. Effective cyber defense requires a framework that simultaneously capitalizes on the adaptability of humans and the speed of machines. Humans must have a correct balance of decision making and delegation to maximize their effectiveness and to acknowledge their legal responsibility for the actions of their automated systems [3].

This paper presents the Cooperative Infrastructure Defense (CID). CID implements the mixed-initiative hierarchical framework of humans and agents called DigitalAnts™ for cyber security. While DigitalAnts™ can apply to many domains, CID is its application to cyber security and the topic of this work.

CID is designed to rapidly and automatically adapt to new cyber attacks while enabling humans to supervise the system at an appropriate level. We interpose a hierarchy of rational software agents between the swarm and the human supervisors to provide a channel for system guidance and feedback. Each type of agent is rational (in that it has logical reasons for its actions), but the types of rationality include human thought, logic-based programming, and supervised, semi-supervised, and unsupervised learning. As explained later, employing a mix of these diverse kinds of rationalities improves system performance. Particularly, apparent false positives at the lowest level of the hierarchy produce feedback loops that help the system collect information and autonomously differentiate potential problems from

previously unknown safe states. In CID, software agents share the decision-making power, handling most of the real-time portion autonomously but enabling human involvement at all levels. The human supervisor does not directly control the system; rather, humans exert supervisory influence, sharing the initiative for action with their software agents. CID is designed to be a scalable, dynamic, and robust framework for securing increasingly complex computational infrastructures. CID makes humans an intrinsic part of the solution by engaging them without requiring them to directly control the agents.

In our research, we have created simulations of the entire framework, prototypes of the UI, and prototypes of the mobile sensor agents. Our initial prototype, built in an agent simulation framework, was demonstrated at the VizSec 2008 [4] conference. We have implemented the framework in Java using the JADE agent framework [5] on a network of virtual machines and on a set of Emulab nodes in the DETER network (http://www.isi.deterlab.net/). Currently we are conducting DigitalAnts™ research for a variety of application domains in addition to CID.

This paper starts by giving an overview of the system including the types of agents, their roles, and how trust between them is managed. Section III discusses a prototype implementation of the system giving more concrete details of system functionality. Section IV contains directions for future research. Our conclusion follows in Section V along with acknowledgements in Section VI.

II. SYSTEM OVERVIEW

In CID, humans and various types of software agents share the responsibilities of securing a cyber infrastructure consisting of enclaves that belong to member organizations. Figure 1 shows how one human can supervise a multi-enclave system with a few enclave-level agents, a host-level agent at each machine or group of similar machines, and a large swarm of simple mobile agents.

Using agents—especially ones modeled on biological organisms—can lead to overly anthropomorphizing concepts in the system. When biological terms such as "spawn" or "die" are used in this paper, it is meant as shorthand for "one agent creating another agent" and "terminating agent execution". These terms are not meant to imply that anything more complex or that true "artificial life" is taking place.

Our terminology is as follows:

- Humans function as *Supervisors*. They provide guidance to and receive feedback from one or more enclaves. They take action only when the lower-level agents encounter a problem that requires human involvement. Supervisors may take initiative as desired to inspect and guide any element of the system. However, direct human control of the system is discouraged, because such involvement would adversely impact its natural adaptive abilities.
- Enclave-level agents, called *Sergeants*, are responsible for the security state of an entire enclave. Sergeants may make service agreements with Sergeants of other enclaves. Sergeants dialogue with humans to gain guidance for running the system

according to business drivers and human security policies. Sergeants create and enforce executable policies for the entire enclave.
- Host-level agents, called *Sentinels*, protect and configure a single host or a collection of similarly configured hosts such as a cluster or storage network. Sentinels interact with human supervisors only when they need clarification about how to classify ambiguous evidence from the swarm.
- Swarming agents, called *Sensors*, roam from machine to machine within their enclave searching for problems and reporting to the appropriate Sentinel. Sensors are diverse; their classifiers are each uniquely derived from the set of known problem indicators. Sensors use stigmergic messages called digital pheromone [6] to communicate.

The concept of Supervisors and agents of the CID framework operating within a hierarchical structure is supported by the research of Parunak [7], Smieja [8], and Selfridge [9], who each suggested hierarchical arrangements of heterogeneous agents. Interposing logic-based rational agents between the humans and the swarm provides a basis for communication, interaction, and shared initiative. The hierarchical arrangement gives humans a single point of influence that allows multiple points of effect. Figure 2 shows the relationships among CID actors, and the subsequent sections describe the roles of the actors. It may be useful to illustrate CID in terms of autonomic computing [10] since CID is a system of autonomous agents that manages and protects an enclave of computers. We have drawn comparisons to autonomics where useful, but CID concentrates on maintaining a concerted cyber defense without considering the other functions of autonomics.
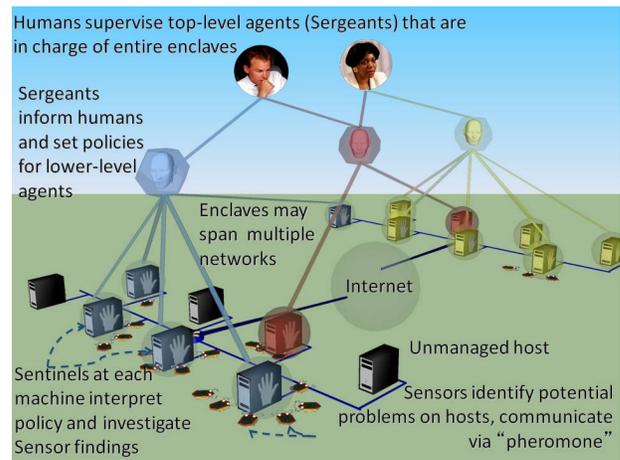

Figure 1. CID is a hierarchical framework of human supervisors, enclave-level rational agents, and swarming agents. A single human may supervise multiple enclaves via the agent hierarchy.

A. Supervisors

At the top layer of Figure 1 and Figure 2 are human supervisors who may direct one or more enclaves. Supervisors may belong to one or more interdependent organizations within the infrastructure, while the cyber assets

in every enclave all belong to a single organization by definition. A Supervisor might also be a member of a regulatory organization or law enforcement agency and only monitor the equipment in the enclaves. Human supervisors translate business policy into guidance via natural-language and graphical controls for top-level agents called Sergeants.
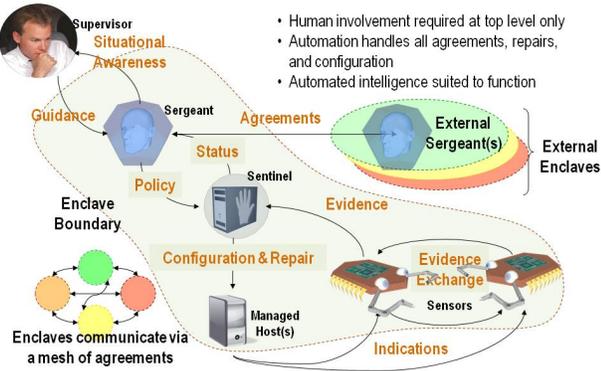

Figure 2. Cooperation among humans and agents in CID.

B. Seargents

Each enclave has a top-level agent called a Sergeant. In autonomic computing terms, the Sergeant corresponds to the orchestrating Autonomic Manager. Sergeants provide situational awareness to the Supervisor, via an information visualization interface. Sergeants translate the guidance from the human Supervisor into actionable policy across all the machines within an enclave. Sergeants will employ supervised learning algorithms so that interactions with them become more efficient over time. Sergeants are "heavy-weight" rational agents that make decisions based on logic. One possible implementation we have considered for Sergeants is Belief-Desire-Intention logic [11]. The Sergeant presents the activities of lower-level agents to the human Supervisor and functions as an interface to influence system operation. Supervisors use Sergeants to enact environmental settings and policies that govern the general operation of the lower-level agents without controlling the lower-level agents directly. Sergeants define the "geography" (a two-dimensional representation of the network) for the enclave. The geography concept will be further defined under the System Architecture subsection. Sergeants also broker agreements between CID enclaves on behalf of the Supervisors. To ensure that their actions are properly attributable, Sergeants must have a separate digital identification from the Supervisor to which they report. Since they negotiate on behalf of humans, they may incur liability for their owning organization. Thus, there must be a mechanism to describe the types and degrees of authority the Supervisor has delegated to the Sergeant. This authorization could be quantified, for example, in terms of maximum dollars that can be spent or types of service contracts that can be negotiated.

C. Sentinel

Sentinels are mid-level rational software agents that, with their managed host(s), function as the autonomic elements in an autonomic computing system. In autonomics parlance, the Sentinel is the Autonomic Manager, and the host corresponds to the Managed Element. Each Sentinel is responsible for a single machine or group of machines that are similarly configured. For example, a Sentinel might be responsible for a single server, a router, a storage area network, a group of web servers, or even a set of managed user workstations. Sentinels implement the policy they receive from the Sergeant.

Sentinels also interface with the lowest-level agents: the swarming Sensor agents that gather information on potential problems found on the hosts. Sentinels provide the local geography to Sensors and provide mobility by negotiating with their destination. Sentinels also provide rewards and spawning capabilities to Sensors. The Sentinels combine evidence from the Sensors with their own historical data, shared knowledge from other Sentinels, guidance from Sergeants and Supervisors (interpreted by the Sergeant), and contextual host information to determine whether a problem exists and to devise potential solutions. Mechanisms similar to this are used in survivability architecture [12].

Sentinels give feedback on the utility of Sensor findings in the form of "rewards" to the Sensors that visit their nodes. We use the analogy of foraging ants to illustrate the effect of this feedback on the system. By rewarding visiting Sensors, the Sentinel will cause them to deposit digital pheromone message trails that will attract more Sensors to visit. A variety of visiting Sensors will provide more information on the potential problems experienced by the Sentinel and provide data to help diagnose and fix the problems.

D. Sensor

Sensors are lightweight, swarming, mobile software agents that move from machine to machine and collect evidence of potential problems. Technically, CID does not fully implement the DigitalAnts™ notion of true mobile agents because of security concerns. Instead, CID moves Sensor state from one Sentinel to another while the actual Sensor code resides on the Sentinel itself. The resulting system resembles a remote procedure call system plus message passing. Sensors are modeled after behaviors of social insects and they employ a form of ant-colony algorithms and swarm intelligence [13]. The Sensors' logic is as simple as possible; their power is in their numbers and their diversity. Sensors wander across the geography superimposed on the enclave, randomly adjusting their current heading similar to the movement of real ants. Unlike a true random walk, the new heading chosen is a perturbation on their existing heading rather than a truly random new heading. Each Sensor uses a learning classifier [14] to match a particular set of conditions in the hosts they visit. There are two broad categories of Sensors: Markovian (memoryless) and differential. Markovian Sensors look for static conditions that may either define signatures of known problems or well-known anomalous conditions. Differential Sensors look for differences in conditions between hosts in recent memory

and their current host. For example, there may be an unusual rate of network connections, a large number of open files, many "zombie" processes (those hung just before termination), or unusually high processor utilization levels. Evidences used by differential Sensors are not specific signatures of any type of intrusion—they are general indicators that one might use to determine whether anything unusual was happening on a machine.

Sensors communicate with each other stigmergically via trails of digital pheromone [6] messages. Decentralized, pheromone-based systems have been demonstrated to simply and effectively solve highly constrained problems where logic-based, optimizing approaches prove intractable [15]. Pheromone-based techniques have been shown to be robust and therefore appropriate for dynamic applications, such as network routing [16].

Sergeants and Sentinels select successful Sensors (those that are most frequently rewarded for useful findings) as templates for spawning new Sensors. New Sensor types may also be created at any time by the Supervisor. The interactions between Sentinels and Sensors provide highly flexible classification and unsupervised learning. The resulting classification resembles the random forest [17] approach, except the members of the forest are not tree-structured classifiers but learning agents. Also, the voting mechanism is different because there is no central consensus; all decisions are made locally and influenced by transient pheromone pathways.

E. Managing Trust

Trust management serves to protect the system from malicious behavior by insiders and entities that have penetrated network defenses. In previous research [18],[19] we examined the trust relationships, evidence, and decisions in CID and found that by monitoring the trustworthiness of the Sentinels rather than the swarming Sensors, the trust management problem became much more scalable and still served to protect the swarm. We then proposed the DualTrust conceptual trust model [19],[20] for managing trust. By addressing the Sentinel's bi-directional (vertical and horizontal) primary relationships in the CID architecture, DualTrust is designed to monitor the trustworthiness of the Sentinels, protect the Sensor swarm in a scalable manner, and provide global trust awareness for the Sergeant.

In addition, in future work, trust delegation techniques and authorization credentials will be used to safely delegate the types and degrees of authority the Supervisor delegates to the Sergeant, and trust negotiation techniques will be used to negotiate agreements between Sergeants of different enclaves.

III. PROTOTYPE IMPLEMENTATIONS

During the development of CID one simulation and two prototype systems have been developed. The Netlogo (http://ccl.northwestern.edu/netlogo/) simulation was initially developed at Pacific Northwest National Laboratory to help guide the design of CID and understand its behavior given a large set of nodes. This later became the prototype UI for the system. Next, Pacific Northwest National Laboratory and Wake Forest University faculty and students developed the first prototype of a true implementation system using the JADE framework. This implementation provided insight to the approach's ability to protect a small enclave of computers when exposed to a computer worm. The latest version of CID was developed by University of California, Davis faculty and students as a lightweight implementation designed for limited resource use and to have minimal impact on the monitored system. This test bed implementation has been successfully tested on a collection of networked computers on the University of California, Davis campus and on the DETER [21],[22] and ProtoGENI [23] test beds (both as part of the NSF GENI project [24]). DETER and GENI allow experimenters to request collections of hosts (referred to as slices) for use in network research. One goal of our CID work is to investigate supporting security services for GENI. The first phase of this work includes adding security monitoring to slices by using DigitalAnts™. It will be important for the monitoring system to have negligible impact on GENI experimenters' research whenever timing is an important factor. The following sections discuss the Netlogo UI and the most recent testbed implementation of CID.

A. Netlogo Supervisor Interface

The Supervisor interacts with the Sergeant through an information visualization and graphical UI that gives the Supervisor situational awareness of state of the enclave. Utilizing the Netlogo agent simulation toolkit, we have implemented variations of the Sergeant interface. These are conceptual interfaces, suitable for research purposes. The eventual UI will require usability engineering to achieve the desired level of human interaction effectiveness. Using slider bars, the user can adjust the parameters of the environment to influence the behavior of the agents involved. As mentioned earlier, the Supervisor uses this interface to adjust the activation and crowding tolerance target levels.

In the simulation UI shown in Figure 3, the Sergeant provides enclave-wide situational awareness to the Supervisor and enables interaction with various agents. Each square in the visualization represents the status of a Sentinel that reports to the Sergeant. The color of the squares could indicate activity levels, security conditions, or any encoding the Supervisor prefers. In the simulation, we use the red, green, and blue components of the color to represent file system, memory, and CPU activity levels respectively. Systems performing similar tasks typically have similar colors. In Figure 3, workstations appear reddish, web servers are shades of blue, and file servers appear gray. A color change would indicate a change in behavior and function and could indicate a problem.
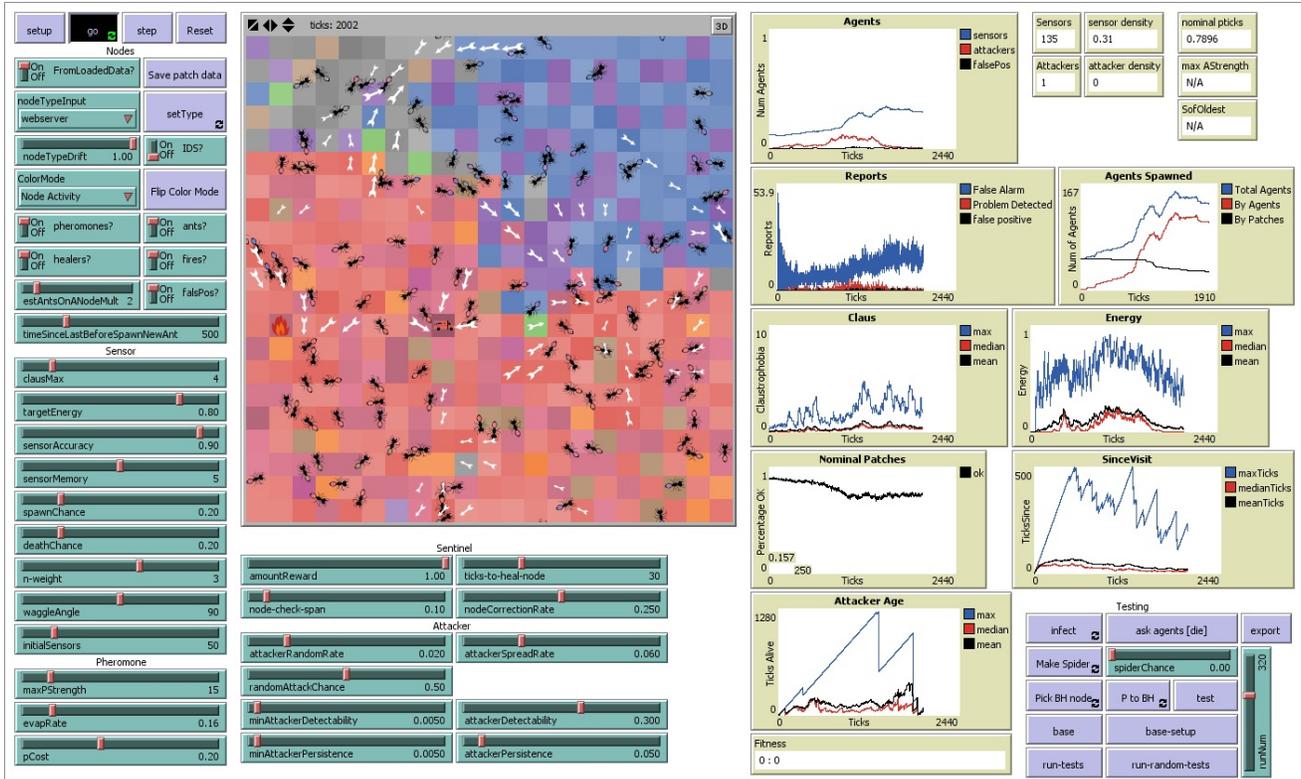
Figure 3. Prototype of Sergeant UI

B. Test Bed Implemetantion

The goal of our test bed is to create a lightweight framework useful for experimentation and as a prototype for future implementations. All code was written using the Python language to allow rapid prototyping and a development path to C++. This implementation, while true to the spirit of the DigitalAnts™ concept, for performance and security reasons digresses somewhat from the general model. This implementation is described below.

A Sentinel process runs on each monitored host. The key function of the Sentinel in the test bed implementation is to manage Sensor movement. This involves executing the Sensors' specified classifiers, updating its state and routing Sensors to neighboring Sentinels. Sensors are not full mobile agents but are implemented as network messages containing a classifier type and other state information. Sentinels are also responsible for Sensor state changes, Sensor creation and destruction, Sensor evaluation, and communication with the Sergeant. In the test bed implementation, the Sergeant process runs on a dedicated host separate from the Sentinels, and its primary function is to initialize and monitor the Sentinels. Sergeants create new sensor types, report alerts and other information from the Sentinels. Sensors are implemented as single packets. Sensor movement is simply the passing of Sensor packets between neighboring Sentinels. Communication occurs only between neighboring Sentinels, and between Sentinels and their Sergeant. Details of our implementation are described below.

1) Sensors: In the test bed implementation Sensors are called "Ants". These are not true agents carrying mobile code. Ants are simply messages that carry an Ant's identity, type and state. Ant data fields are described in Table 1. Ants are implemented as IPv4 network packets. Because lightweight design was important, we investigated both TCP and UDP protocols. Although undetected packet loss may occur with UDP, if the drop rate is acceptably low locally and globally, the overhead of TCP can be avoided without adverse effects. This may be important in some implementations with particularly limited resources. Because Ants are transmitted as single packets, the entire state must be very small. While efficient, this limits the length of allowed parameters and the possibility of carrying mobile code.

Configurable system parameters for Ant creation and termination rates can be adjusted to tradeoff the amount of network bandwidth consumed vs. the detection effectiveness.

2) Sentinels: Sentinels maintain local state, manage sensor functions and Ants, and communicate with their Sergeant. Their functions include:

- Evaporate Pheromone: the pheromone trail is not permanent, but dissipates over time. This prevents Ants from taking action (i.e., following trails) based on old information. The Sentinel monitors the age of the pheromone and removes it after a set interval.

Field	Description	Use
id	unique identifier for the ant.	Used to determine if a pheromone was left by itself.
sensor_ type	the evidence type the ant is seeking.	This tells the Sentinel what sensor function to execute.
sensor_ parameters	parameters for a particular sensor type.	Allows for variants of the same sensor, e.g. thresholds, filenames, character sequences, etc.
state	foraging, following, dropping, idle.	Determines an ant's actions.
age	how long the ant has been traveling.	After a period of time ants will die (i.e. be removed).
direction	the direction vector for the ant.	This is used to determine the next node for the ant when the ant is not following a pheromone trail.
prior node	the host the ant was received from.	Used to direct ants along pheromone trail.
time_ dropping	how long the ant has been dropping pheromone.	After a period of time an ant will stop dropping and wander idle
time_ idle	how long the ant has been idle.	After a period of idle wandering ant's will return to foraging.
where_ found	the location the evidence was found.	Used in experiments for alternative ways for pheromone to direct ants to a target.

- Receive Ant: Ant packets are read from a network queue and processed sequentially.

- Kill Ant: based on crowding and an Ant's age, the Ant may die. Both of these are implemented probabilistically. The age of the Ant is tracked and updated in the Ant's state. Once it reaches a set threshold, it may die, with increasing likelihood with increased age. Similarly, depending on the crowding factor (i.e., how many Ants have been seen at this node in a preceding window of time) the likelihood of the Ant's death increases. Depending on the Ant's state (foraging, following, dropping, idle), the Sentinel will carry out different actions. In general, this means checking for conditions that would change the Ant's state, updating any local (i.e., host) state, and determining the Ant's next destination.

- Foraging: The primary Ant state is "foraging". When the Sentinel receives a foraging Ant, it executes the sensor function specified for the Ant's type. If this result indicates something of significant interest exists, the Ant's state will be changed to "dropping". If instead the node managed by the Sentinel is part of a pheromone trail, the Sentinel will, with high probability, change the Ant's state to "following".

- Following: This state is similar to foraging. The Sentinel executes the Ant's sensor function, and may change state to dropping. The difference is that rather than determine the Ant's next location using its direction parameter, the Ant is directed along the pheromone trail toward the location where another Ant found evidence. If the pheromone trail ends, the Ant's state is changed back to foraging. In either of the above cases, if the Ant enters the dropping state, a new random direction for the Ant is selected.

- Dropping: When the Sentinel receives dropping Ant, its sensor function is not executed. Instead, the Sentinel updates its state to indicate that it is part of the pheromone trail and from which neighbor node the dropping Ant came (i.e., toward the evidence). Because, pheromone is dropped for a limited number of steps, the Sentinel will decrement the drop counter of the Ant. If it reaches zero, the Ant's state is changed to idle.

- Idle: An idle Ant simply moves about the mesh; no sensor functions are executed, nor does it follow any pheromone trails it encounters. This causes the Ant to move away from where it last found something it returns to the foraging state and again looks for evidence. Idle Ants remain in this state for a fixed number of steps. The Sentinel decrements the Ant's idle counter. When it reaches zero the Ant's state is changed back to foraging.

- Create Ant: new Ants are created in response to a low crowding factor and a high utility of a particular Ant type. This allows us to maintain a sufficient number of Ants for threat detection, and to bias this towards Ants that are currently helpful in detecting a threat.

- Send Ant: connect to the node for the Ant's next destination and send the Ant packet with its updated state information.

- Receive_Sergent_Message: accepts messages from the Sergeant and takes the appropriate action. Examples include: report status, and add new sensor type.

3) Sensor Functions: Sensor functions refer to the code executed on a Sentinel to collect the information requested by an Ant. These can be either some current state for a summary of recent activity, such as "is port 31373/tcp open?" or "how many page faults have occurred in the last minute?" respectively. An Ant's type determines the particular sensor function the Sentinel is to execute. Any associated parameters may be used to modify the particular sensor function. In this way a sensor function for "is a process named X running?" can become "is a process named cmd.exe running?" The results of this, modified by the Sentinel's determination if this is a false-positive (i.e.,

not significant for the Sentinel under current circumstances), determine if the Ant begins to leave a pheromone trail.

Sensor functions are cached in the Sentinels. An initial set is predefined upon Sentinel initialization. When a Sentinel receives an Ant type of an unknown sensor function, the Sentinel will request the function definition from its Sergeant. This enhances flexibility, eliminates the need to support mobile code, simplifies Ant structure, and increases security.

4) Sergeants: During system initialization, Sergeants identify the Sentinels they monitor and determines each Sentinel's neighbors. The following section on geography describes how neighbors are assigned. At runtime, Sergeants reply to sensor function definitions requests, and collect activity reports from the Sentinels. This information is used to track alerts, Ant motion and state More advanced features such as a Sergeant UI and the ability to create new sensor types have not yet been implemented. However, via a related support utility, a user may create Ants with new parameter values using an existing sensor type.

5) Geography and Direction: This implementation constructs a geography based on a two-dimension square mesh with edge wrapping. That is each node has four neighbors (up, down, right, left); edge nodes' neighbors are at the opposite parallel position. Nodes in the mesh represent monitored hosts. The Sergeant creates a rectangular mesh as close to square as possible and assigns neighbors from a list of hosts to be monitored.

Direction in this mesh is based on four directions: zero for up, 90 for left, 180 for down, and 270 for left. Each Ant's direction is an integer from 0 to 359. If the Ant's direction exactly matched a neighbor's position the Ant would tend move to that neighbor. Otherwise, we pick from its neighbors weighted by how close the Ant matches their position. If the Ant's direction was 45, there is an equal chance the Ant will move up or left. To add a controllable amount of randomness to the Ant's wandering, the Ant's effective direction is modified by a random value using a uniform distribution. This will cause Ant's path to diverge slightly from its set direction.

The hosts used in our initial work are representative of computers connected on the same subnet or with ready inter-host communication. We are also actively investigating the use of DigitalAnts™ in other settings such as wireless sensor networks. Our initial model is based on wireless mesh networks similar to those seen with the ZigBee [25] and WirelessHART (IEC 62591) [26] protocols. WirelessHART is a common protocol in industrial control systems. ZigBee is a widely adopted IEEE 802.15.4-based wireless protocol used in sensor networks, building automation and Smart Meters currently being deployed worldwide. Unlike our original implementation where an artificial geography must be imposed on the host being monitored, for these networks a real geography exists. Devices are capable of communicating directly with only devices near enough (and unobstructed) to have sufficiently strong and noise free radio communications. Unlike the original design, here a device will have a varying number of neighbors, both in number and availability over time.

Because the mesh may change and Sentinels do not know the location of their neighbors, rather than have the Ants wander using a direction, we use a random walk of the mesh. To prevent Ants from wandering the same paths, the Ants leave a pheromone trail indicating where they have been. Ants will preferentially not take paths with a strong self-pheromone trail. Also, unlike the original implementation, the Sergeant is not required for initialization of neighbor relations; instead, the geography is self-organizing based on connectivity.

We are implementing this via simulation on the DETER test bed. To initialize our simulation, each node will randomly generate (or be assigned) an (x,y) position within a set range. These coordinates are sent to a central simulator process that tracks node position, inter-device distances, and sends messages to nodes that are their neighbors (relative to a set distance). This allows for dynamic changes in future testing.

In these systems, because of the low transmission power, most nodes cannot directly communicate with the Sergeant, but must route packets through the mesh to a node that has connectivity. Each of the above mentioned protocols has methods for routing packets through the mesh. Because the Sergeant may be associated with a different type of device with a much higher transmit power, in some configurations it may be possible to send messages to Sentinels directly. However, except for gains from a more sensitive antenna and receive circuitry, it is likely that Sentinels may not send messages to the Sergeant. For the purpose of our initial tests, we assume that messages between Sentinels and Sergeant require routing through the mesh.

Through our implementation effort, we have concluded that implementation details must vary to best fit the particular domain. This must take into account processing resources, networking characteristics and threat profile.

In our prototype, we took no effort to protect the code from malicious users. Future GENI implementation plans include integrating it into the virtualization process (e.g., hypervisor) and deploying it to monitor infrastructure hosts, not just slice hosts. Authentication of Ants to Sentinels and other system communications must be implemented.

C. Preliminary Results

The JADE prototype system has been a valuable resource for gaining an initial understanding of system performance, such as threat detection time and system efficiency, under varying conditions. For example, an important system aspect to consider is the impact of the Sensor population size on threat detection, where is it expected that larger populations will reduce the detection time. Consider the number of Sensors in the system as a density, where 100% density represents an equal number of Sensors per type as there are computers to protect, while a 50% Sensor density represents half as many Sensors per type as computers. Using the prototype system and a simple grid of computers to protect, a density of 75% Sensors was found to perform as well as a traditional IDS modeled as one resident static agent per

computer [27]. Adding Sensors beyond a 75% density level for this experiment yielded diminishing marginal returns, reducing the detection time variance.

Another important factor to system performance is the use of pheromone. If pheromone is not used then the Sensors randomly select their next destination. For example consider a 75% Sensor density with and without pheromone as compared to a traditional IDS configuration (again, static agents; one per computer). Preliminary results showed that the impact of pheromone was significant, resulting in a 34% reduction in detection time as compared to no pheromone. Therefore, the 75% Sensor density performed comparably to a traditional IDS deployment in terms of detection time, but it also required 11% fewer CPU cycles than the static IDS approach [27]. Although these initial results are informative more experiments are needed to further understand system performance, deployment, and management.

IV. FUTURE RESEARCH

We have focused our research mainly on the Sentinel and Sensor behavior in our implementations, thoroughly mapping out the behavior of the Sensors and their interactions with the Sentinels. We have created a demonstration system of the Sensor-Sentinel interactions in JADE and have created vertical prototypes of interactions at the Supervisor, Sergeant, and Sentinel levels. We intend to conduct usability studies on our prototype Sergeant interface to determine how best to enable the Supervisor's situational awareness. The next step would be to develop a functionally complete interface between Supervisors and Sergeants that translates human guidance into executable policy for the Sentinels. Further, we wish to experiment with our Sensor-Sentinel implementations on larger network topologies in order to characterize the scalability of our framework. We also intend to test the framework against real-world attacks and learn to generalize knowledge gained from solutions to those attacks. Similar problems could be solved with similar approaches and this would help generalize the system and enable adaptive responses to new attacks.

V. CONCLUSION

The CID framework enables security event monitoring for distributed systems with multiple overlapping and decentralized administrative domains in a way that traditional NIDS do not. Additionally, the system allows for:

- Specifying and simulating a framework suitable for multi-organizational cyber security.
- Developing feedback mechanisms that keep Sensor populations under control with minimal resource usage.
- Specifying a mixed-initiative approach that is highly scalable and difficult to disrupt.
- Deployment into a variety of interconnected systems, not just across IP networks.

Our experiments have shown that our framework of shared system control using a hierarchy of agents and the adaptive capabilities of swarm intelligence, has greatly enhanced the security of key, critical, interconnected infrastructures.

REFERENCES

[1] D. Frincke, A. Wespi, and D. Zamboni, "From intrusion detection to self-protection," Comput. Networks, vol. 51, 2007, pp. 1233–1238.

[2] E.A.R. Dahiyat, "Intelligent agents and intentionality: Should we begin to think outside the box?," Comput. Law Secur. Rep., vol. 22, 2006, pp. 472–480.

[3] M.B. Scher, "On doing 'being reasonable'," ;login:, vol. 31, 2006, pp. 40–47.

[4] J. Haack., G. Fink, E. Fulp, and W. Maiden, "Cooperative Infrastructure Defense," presented at the Workshop on Visualization for Computer Security (VizSec), 2008, www.vizsec.org/workshop2008/fink.pdf.

[5] F. Bellifemine, G. Caire, D. Greenwood, Developing Multi-Agent Systems with JADE, Wiley & Assoc, 2007.

[6] S. Brueckner, Return From the Ant: Synthetic Ecosystems For Manufacturing Control, PhD Dissertation, Berlin: Humboldt University, Department of Computer Science, 2000.

[7] H.V.D. Parunak, P. Nielsen, S. Brueckner, and R. Alonso, "Hybrid multi-agent systems: Integrating swarming and BDI agents," in ESOA 2006, LNAI, vol. 4335, S.A. Brueckner., S. Hassas, M. Jelasity, and D. Yamins, Eds. Heidelberg: Springer, 2007, pp. 1–14.

[8] F. Smieja, "The pandemonium system of reflective agents," IEEE Transactions on Neural Networks, vol. 7, 1996, pp. 97–106.

[9] O.G. Selfridge, "Pandemonium: a paradigm for learning," In Neurocomputing: Foundations of Research, J.A.D. Anderson and, E. Rosenfeld, Eds. Cambridge, MA: MIT Press, 1988, pp. 115–122.

[10] J.O. Kephart and D.M. Chess, "The vision of autonomic computing," Computer, vol. 36, 2003, pp. 41–50.

[11] A.S. Rao and M.P. Georgeff, "BDI agents: From theory to practice," First International Conference on Multi-Agent Systems (ICMAS-95), AAAI Press, 1995, pp. 312–319.

[12] J. Knight et al., "The Willow architecture: Comprehensive survivability for large-scale distributed applications," Distributed Applications: Intrusion Tolerance Workshop, Dependable Systems and Networks (DSN 2002), 2002.

[13] M. Dorigo and L.M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," IEEE Trans. Evolutionary Computation, vol. 1, 1997, pp. 53–66.

[14] J.H. Holland, et al., "What is a learning classifier system?," Learning Classifier Systems: From Foundations to Applications (LCS 99), vol. 1813, P.L. Lanzi, W. Stolzmann, and S.W. Wilson, Eds., Heidelberg : Springer-Verlag, 2000, pp. 3–32.

[15] D. Di Caro and M. Dorigo, "AntNet: Distributed stigmergetic control for communications networks," in J. Artificial Intelligence Research, vol. 9, 1998, pp. 317–365.

[16] E. Bonabeau, et al., "Routing in telecommunications networks with 'smart' ant-like agents," Working Papers 98-01-003, Santa Fe Institute, 1998.

[17] L. Breiman, "Random forests," Mach. Learning, vol. 45, 2001, pp. 5–32.

[18] W.M. Maiden, J.N. Haack, G.A. Fink, A.D. McKinnon, and E.W. Fulp, "Trust management in swarm-based autonomic computing systems," Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing (UIC-ATC 09), 2009, pp. 46–53.

[19] W.M. Maiden, DualTrust: A Trust Management Model for Swarm-Based Autonomic Computing Systems, Master's Thesis, Pullman, WA: Washington State University, 2010, http://www.dissertations.wsu.edu/Thesis/Spring2010/W_Maiden_604 1310.pdf.

[20] W.M. Maiden, I. Dionysiou, D.A. Frincke, G.A. Fink,and D.E. Bakken, "DualTrust: A distributed trust model for swarm-based autonomic computing systems," Data Privacy Management and Autonomous Spontaneous Security (DPM/SETOP 2010), LNCS 6514, J. Garcia-Alfaro et al., Eds. Springer-Verlag, 2011, in press.

[21] T. Benzel, et al., "Experience with DETER: A testbed for security research," Proceedings of Tridentcom (International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities), March 2006.

[22] T. Benzel, et al., "Design, deployment, and use of the DETER testbed", Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test (DETER 2007), Berkeley, CA: USENIX Association, 2007.

[23] ProtoGENI, http://www.protogeni.net/trac/protogeni.

[24] "GENI: Exploring Networks of the Future", http://www.geni.net.

[25] ZigBee Alliance, http://www.zigbee.org/en/index.asp.

[26] "IEC 62591: Industrial communication networks – WirelessHart Communication Network and Communication Profile", International Electrotechnical Commission (IEC), http://en.wikipedia.org/wiki/WirelessHART.

[27] B.C. Williams, A Comparison of Static to Biologically Modeled Intrusion Detection Systems, Master's Thesis, Wake Forest University, 2010, http://wakespace.lib.wfu.edu/jspui/handle/10339/14740.