# Evaluation of Complex Surveillance Systems for Emergent Vulnerability

Christopher Thornton Ori Cohen Jörg Denzinger Jeffrey E. Boyd
University of Calgary
Calgary, Alberta, Canada
chris.thornton@shaw.ca ocohen@ucalgary.ca denzinge@cpsc.ucalgary.ca jboyd@ucalgary.ca

## Abstract

*The current paradigm for testing tracking and surveillance systems is to identify representative metrics for system components, then optimize the performance of that metric against test data. The assumption is that optimization of individual components will optimize the surveillance system as a whole. However, while optimizing components is a necessary step to improve systems, it is not sufficient to address vulnerabilities that emerge in a large system with many components. A large surveillance system will have many cameras and other sensors. In some cases, to cover more area, the cameras and sensors may be mobile. Coverage is unlikely to be complete in all areas at all times, so sensor allocation will follow some policy. The combination of sensors, sensor properties, mobility and policy can result in a system that is vulnerable in ways that are difficult to predict. We present a method to model and predict emergent vulnerabilities in a complex surveillance system. To demonstrate the method, we apply it to a downscaled physical surveillance system that uses multiple stationary and mobile camera platforms to monitor and defend against intrusions. Our method finds two vulnerabilities in the system in simulation, one of which we demonstrate with the physical system.*

## 1. Introduction

With detailed knowledge of the operation of a system, methods of testing appear to be obvious. Such is the case with tracking and surveillance system. We know that success in tracking depends on factors including accuracy of data over varying conditions, dynamic modelling of targets, data association, missing data, occluded targets, and spurious targets. A natural way to test a tracking system is then to acquire data that confounds these factors in some way and then measure performance by some metric such as track accuracy, or erroneously added and dropped tracks. This type of testing is an essential part of success in tracking and surveillance.

We take a step beyond this type of testing and consider surveillance systems that are composed of numerous components of varying complexity and quality. Such systems can be extraordinarily expensive [15] and as examples show, can fail in spite of the investment [1] in the underlying technology. Because of costs and unpredictable outcomes, it is essential to test systems, not just components, before deployment.

It is known that unexpected behaviors can emerge from large systems of simple components. A classic example is Reynolds' flock simulations [19]. In these flock simulations, a collection of simple components can result in interesting emergent behaviors [14]. Similarly, we expect interesting behaviors to emerge from the complexity of a large surveillance system. Factors contributing to this include the following.

1. **Variety:** Although video cameras are the chief sensor, other sensors including infrared imagers, motion detectors, and optical beams can be part of a surveillance system.

2. **System size:** Surveillance systems have a large component/sensor count as the surveilled area gets larger.

3. **Dynamic elements:** Video cameras often have variable pan, tilt, and zoom. Also, in some situations, sensors can be mounted on mobile platforms. Therefore, the location and properties can change over time.

4. **Policy:** Dynamic elements require a policy to guide their deployment. This policy will be encoded into the software that aims a pan-tilt-zoom (PTZ) camera, or drives the robotic platform carrying the sensors.

For a surveillance system, an *interesting* behavior can be a vulnerability, and we refer to this as *emergent vulnerability*.

To illustrate, consider a trivial example with a single PTZ camera. The camera and associated software and processing can track all it sees in its field of view. An intruder walks into the field of view, activating a policy for the camera to follow with its PTZ capability. This amounts to allowing the

intruder to steer the field of the camera, enabling a second intruder to pass by undetected. While this vulnerability is easy to predict, complex systems can easily surpass human foresight and we must turn to other methods.

One option is to consider analyses emerging from the field of computational geometry, which provides theoretical results regarding sensor placement to protect a given region [13, 5, 12, 4]. Such analysis can show theoretically what sensor deployment is required to protect a region by monitoring its borders. However, to make the analysis possible, it is necessary to restrict the scope of the system in sensor variety, complexity, dynamic behavior, and policy. Consequently, the conclusions do not transfer well to real situations.

We describe a novel system that uses a *learning multiagent system* to probe a surveillance system for emergent vulnerabilities. Multiagent systems capture the innate complexity of a surveillance system, and the combination of machine learning with multi agents gives us a powerful tool for testing. Other such systems have been shown to reveal emergent weaknesses in other systems as complex as a computer game and a search and rescue simulation system [11, 2].

This work stems from our interest in harbor and border surveillance and security. We consider a red-team-versus-blue-team exercise in which the blue team defends a harbor and a red team attempts to penetrate the harbour surveillance to cause damage before the blue team can intercede. We test the blue-team system by simulating it in as much detail as possible. The red team is a multiagent system that has a set of actions it can perform to attack. Through repeated trials, the red team probes the blue team to learn blue-team vulnerabilities. The repetitive probing dictates that the process must be simulated in software. With suitable simulation, we can predict emergent vulnerabilities in a real system, as we demonstrate in this paper.

There are two important assumptions required for our testing to succeed. First, our learning and simulation systems must have complete knowledge of the blue team. While this may not always be the case in the real world, it is a safe assumption – if we find no vulnerability when our opponent knows everything, we should be safer when they know even less. Second, the red team has a set of actions available to it when testing. If a real-world opponent has actions available that are not considered in testing, then a real-world vulnerability may be overlooked. Thus, the testing works best when we assume an opponent that is both omniscient and omnipotent.

## 2. Background

### 2.1. Tracking System Performance

Most tracking systems are built around the well-known Kalman and extended Kalman filters [8], and variants such as the particle filter [10]. These algorithms produce trajectories (state estimates over time) of a target from a temporal sequence of measurements. This is sufficient for the simplistic scenario where we have measurements for only a single target. Tracking multiple targets requires data association, the mapping of measurements to trajectories [3].

Most challenges in tracking in surveillance systems can be reduced to either acquiring accurate measurements for trajectories and/or data association. Thus most testing is built around metrics and data sets that focus on these issues. A system that can acquire consistently accurate data is bound to be better for surveillance, as is a system that can correctly associate data to trajectories. For example, this can be seen in the datasets for PETS 2009 [6]. The datasets contain examples of crowded scenes. Successful systems will accurately identify salient objects, such as people, i.e., they will measure well. Successful systems must also keep track of the people, i.e., they will correctly associate measurements for an individual to a single, contiguous trajectory.

The emergent vulnerabilities likely to stem from large scale deployment with various policies is largely unexplored.

### 2.2. Multiagent Systems

Multiagent systems have as one of their goals the study of interactions between entities (called *agents*) and creating emergent properties, either by appropriate (human) design of interactions or by learning by the agents, and is still a key research topic [17]. Surprisingly, there have been few works to develop concepts for testing multiagent systems for emergent vulnerabilities. Some exceptions are Kidney and Denzinger [11], Flanagan et al. [7], and Atalla and Denzinger [2]. These use techniques aimed at creating emergent behavior, namely learning of cooperative behavior for a group of agents, to reveal emergent vulnerabilities in another group of agents – fighting fire with fire. Our following approach is also based on this general idea, but goes further by incorporating realistic sensor models and agent motion. In so doing, we find vulnerabilities due to target detection, tracking and data association, motion control and path planning policies that can be duplicated in a real physical system.

## 3. System-Level Testing of Tracking and Surveillance

### 3.1. Surveillance as a Multiagent System

Consider a multiagent system where a single *agent*, *Ag* is defined by the tuple, (*Sit*, *Act*, *Dat*, $f_{ag}$), where the elements of the tuple are described as follows.

- *Sit* is the agent's *situation* as perceived by the robot. I.e., it is the agent's view of the world through its sensors.
- *Act* is a set of actions that the agent can perform. In the context of physical agents, *Act* is a set of things the agent can do with its actuators.
- *Dat* is the agents internal data or state.
- $f_{ag} : Sit \times Dat \rightarrow Act$ is a *decision function* mapping the agents perception of the world and its internal state to actions – it is how the agent *decides* what it will do.

A multiagent *system* as a tuple $(\{Ag\}, Env)$, where $\{Ag\}$ is the set of agents, and *Env* is the environment in which those agents interact.

We test surveillance systems embedded in a hypothetical military exercise in which a blue team conducts surveillance on a region to defend against an intruding red team. Figure 1 shows the embedding. The surveillance system itself is is a set of $m$ blue agents. A team of $n$ red agents attacks the blue agents. All of this occurs in an environment, *Env*. An *events* agent, $Ag_{\text{events}}$ acts independently to trigger changes in the environment. In a physical system, this might correspond to a change from daylight to darkness, or a change in weather. Although we do not use this for the results presented in this paper, it is possible to include agents representing neutral parties, $\{Ag_{\text{other},i}\}$. For example, in harbor surveillance, this could be legitimate harbour traffic, or in border surveillance in remote areas, other agents might be animals that would show up on infrared sensors.

### 3.2. System Testing Method

The blue-team agents embody the surveillance system we want to test. This includes sensor properties, mobile platform capabilities, and deployment policies, all of which are captured by the (*Sit*, *Act*, *Dat*, $f_{ag}$) tuples for the blue agents.

A *machine learner* tests the policies represented by the blue agents for weaknesses by learning behaviours for the red agents to achieve a goal state in which they have defeated the surveillance. In effect, the red team learns decision functions (the $f_{Ag}$) that will beat blue (or more precisely, $\{Ag_{red}\}$ learns parameters for its decision functions).

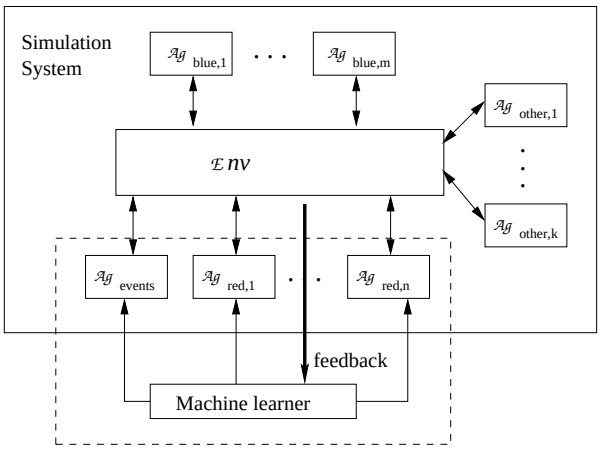



Figure 1. Blue-team-versus-read-team multiagent system simulation: the machine learning module learns strategies that allow the red agents to defeat blue.

The events agents must also be part of the learning process so that red can exploit specific environmental conditions. For example, if blue's sensors do not work well in rain, red can wait for rain until it mounts its attack. Note that this does not mean the red team can change the environment, but that they can use environment changes to advantage when they occur.

The *learner* uses feedback from simulations to learn the decision functions. Typically, there are a large number of behaviours for the red and event agents so that it is not possible to systematically try out all possibilities. Instead, similar to human decision makers, a learner evaluates the results of simulation runs, adjust the behaviours of the agents and repeats the cycle until a weakness is found or the maximum number of iterations allowed is exceeded.

### 3.3. The Machine Learner

Our machine learner uses multi-objective particle swarm optimization [18]. Each particle represents a possible solution to a problem. Particles move through the solution space with a combined tendency to move towards better solutions (as determined by some metric), and a tendency of the swarm to move together. When a particle achieves a goal state in the solution space, the algorithm terminates. Randomization of the particle initial states leads to different possible solutions.

In our surveillance problem, the machine learner must learn how to move the agents in $\{Ag_{red}\}$ in a spatial environment. The solution space consists of sequences of high-level waypoints and speeds. Using standard path finding techniques [9], the high-level waypoints are translated into a series of low-level waypoints that take into account obstacles in the environment. The learner then runs the simulation, to determine several *evaluation numbers* for each se-

quence/swarm particle through various measures (e.g., the number of surviving red agents, and the progress of agents toward goals). Using the method outlined in [7], these evaluation numbers along with their associated swarm particles are placed into an *ordering structure*. From the *ordering structure*, the learner creates new values for the sequences/swarm particles. Our instantiation of the particle swarm defines a particle as a tuple consisting of the high-level waypoints, a *particle velocity*, and the set of *best states* from the particle history. In each iteration of the swarm, after all the particles in the swarm have been evaluated, the particle moves in two steps. First the particle velocity is applied. Second, the particle is *pulled* into the direction of other particles in the swarm that are considered better based on the *ordering structures*. Thus, the particles move together, searching the solution space about a region of improving options.

The algorithm stops when a particle has reached a goal state representing a possible solution. Multiple solutions can be found by re-initializing the swarm with randomized starting points.

## 4. Demonstration

### 4.1. Overview

While our testing technique is built on simulation, it is worthwhile to demonstrate its validity with a physical system. To that end, we created a model surveillance system from several small robots with cameras in a laboratory environment. We model the same system in software and test for vulnerabilities using the multiagent/learning system described in the previous section. Finally, we verify that the learned vulnerabilities do indeed exist by demonstration with the physical system.

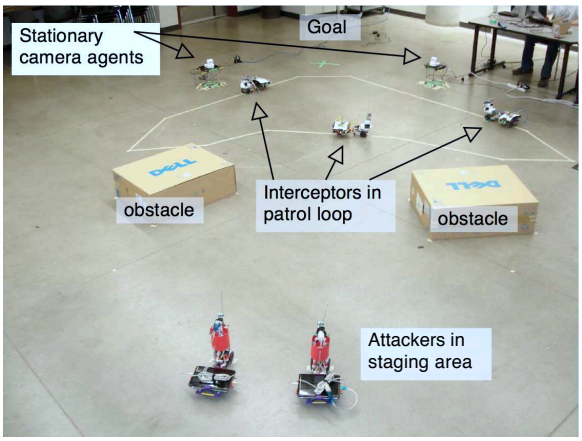
The model surveillance area is a flat floor, indoors, about six meters square. We place some *terrain features* (boxes) on the floor to make the surveillance task more interesting (Figure 2(a)). The area is equipped with fixed cricket motes [16] for positioning. We have two types of agents:

1. small mobile platforms(or *unmanned vehicle*) with differential steering, carrying a netbook computer, cricket motes, and sensors as required (Figure 2(b)), and

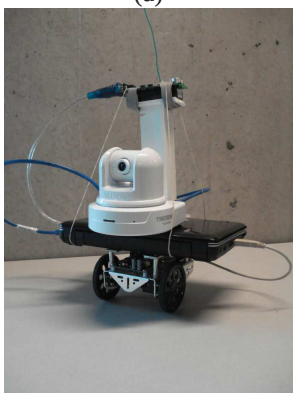
2. stationary platforms with pan-tilt (PT) cameras and a netbook for processing and communication.

Blue uses a mixture of mobile and stationary agents while the red team uses mobile agents exclusively.

Figure 3 shows the surveillance region and agent tasks schematically. Two red agents start in the lower right-hand corner of the region and attempt to get one of them to their goal in the upper left-hand corner. The blue team uses five agents to defend their region. Two agents are stationary




(a)



(b)

Figure 2. Elements of the physical system: (a) the testing area, and (b) one of the physical agents. The tape marking the patrolling pattern on the floor is not used by the agents and is for debugging only.

with PT cameras. They coordinate their motions to have the two cameras (for triangulation) cycle their coverage over the shaded surveillance region. Three mobile interceptor agents have cameras (the cameras look in the direction of travel only). They are deployed by the policy described in the following sub-section. We say that an interceptor has successfully stopped a red agent when it has 9000 pixels identified on-target (corresponding to about $3\%$ of the image area, or about $0.5m$ from the target). While this distance is arbitrary in our simulation, it mimics a *lock* on a target that is sufficient to assure the destruction of an intruder.

After the physical implementation of the blue team, we port the code to the simulation environment so that the blue team runs as it would but with simulated sensor input and modelled physical dynamics. Then we use our testing system to learn emergent vulnerabilities that the red team can use to defeat blue, and verify with the physical system.

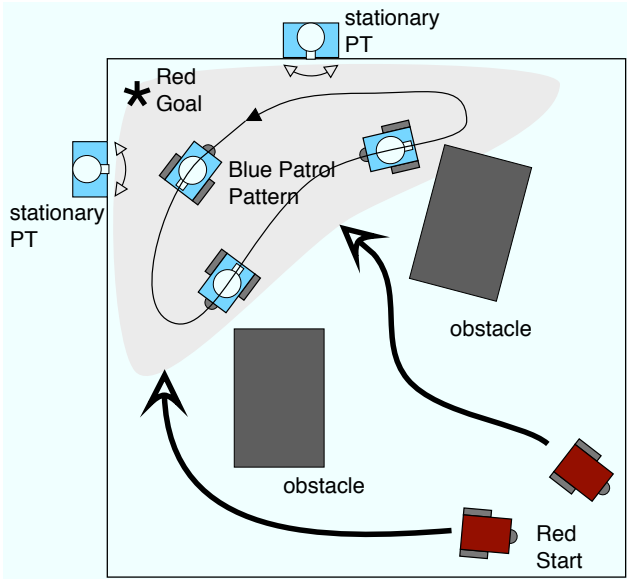



Figure 3. Schematic diagram of the surveillance exercise shown in Figure 2(a). The blue agents (stationary PT cameras and patrolling interceptors) defend the upper left-hand corner of the region. The mobile red agents start in the lower right-hand corner and attempt to reach their goal in the upper left before blue can intercept.

### 4.2. Blue Team

Our blue team is not a *state-of-the-art* surveillance system. It is a compromise between cost and complexity, and the need for a demonstration system with representative functions including: motion, data acquisition, tracking and data association, and policy.

**Motion:** Mobile agents get position data from the cricket motes, which provide feedback to the motion control, allowing the agents to drive to assigned way-points. We estimate an agent's speed and heading from differences in position data, low-pass filtered to compensate for the instabilities of the cricket mote measurements.

**Data Acquisition:** All tracking is done visually via the two stationary agents with PT cameras situated at the periphery of the surveillance area. For simplicity, we assume that the red team will oblige and wear a large red marker, so chroma-keying is sufficient to detect intruding objects. The agent reports all large objects to a base station that ultimately does the tracking. The report includes the agents position, and the bearing of the red objects computed from the position in the image, and the pan setting on the camera.

**Tracking and Data Association:** A base station collects data from the stationary blue-team agents to track the red objects. The tracking is done by Kalman filters with:

- four-dimensional states (two-dimensional position in the surveillance region and corresponding velocities),
- a constant velocity dynamic model.

The base station converts the vehicle position and target bearing data to a measurement that is a two-dimensional estimate of the target position. Since it is impossible to resolve a two-dimensional position from a single bearing, the base station assumes the target is at a range of $0.5m$ in front of the camera. The measurement covariance represents the uncertainty in range. Therefore, the tracking system cannot correctly resolve the target position until it has associated data from other cameras. The data association uses a *best hypothesis* that matches measurements to tracks, optimizing for the quality of match (based on the innovation covariance), and requiring unique measurement-to-target correspondences. The measurement rate is $4Hz$, one round of measurements from all cameras four times per second. Any unassociated measurement causes the tracking system to initiate a new track. Tracks that are not updated for longer than $8s$ due to a lack of associated measurements are dropped.

**Policy:** It is not possible for us to use actual policy for real-world systems from official sources because these are generally not disclosed to the public. Therefore, we contrive our own policies.

Our *contrived* mobile agent policy has three modes of behavior. First, under normal conditions, the blue team drives through a programmed set of way-points while scanning for red objects, i.e., a patrol pattern. As long as no red objects are found, this behavior is maintained. Second, when the tracking system (with data from the stationary agents) detects a red object the nearest blue-team mobile agent (interceptor) drives directly towards the location indicated by the tracker. Third, when the patroller sees a red intruder with its own forward-facing camera, it enters a pursuit mode and steers towards the intruder.

The policy of the stationary cameras is to work together so that the region of overlap of the fields of view (the region where triangulation in track is possible) cycles around the surveillance region. When they find a red object, the central control initiates an intercept action by the interceptors.

### 4.3. Red Team

For verification, we break the red-team strategies into two categories: naïve and learned. The purpose of the naïve strategies is to show that the blue team is a plausible surveillance system. If the blue team could be defeated naïvely, then learning emergent vulnerability is irrelevant. The learned strategies are the result of the multiagent systems testing to reveal an emergent vulnerability as described in Section 3.

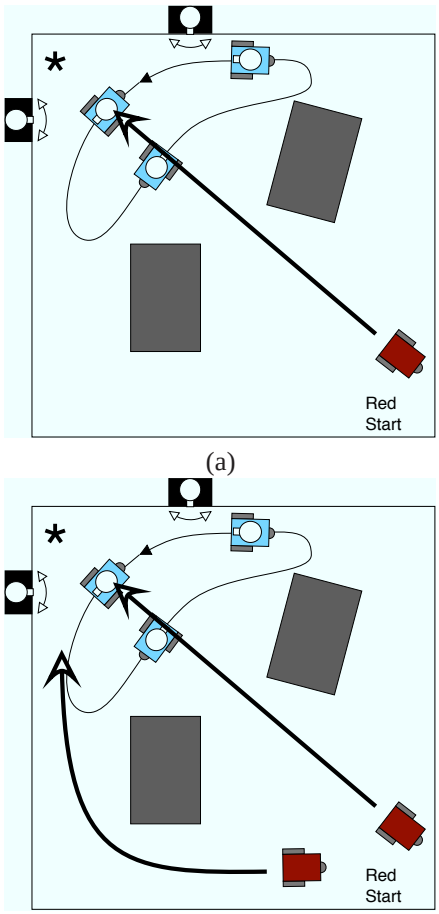



Figure 4. Naïve red team strategies: (a) with a single attacker, and (b) with two attackers.

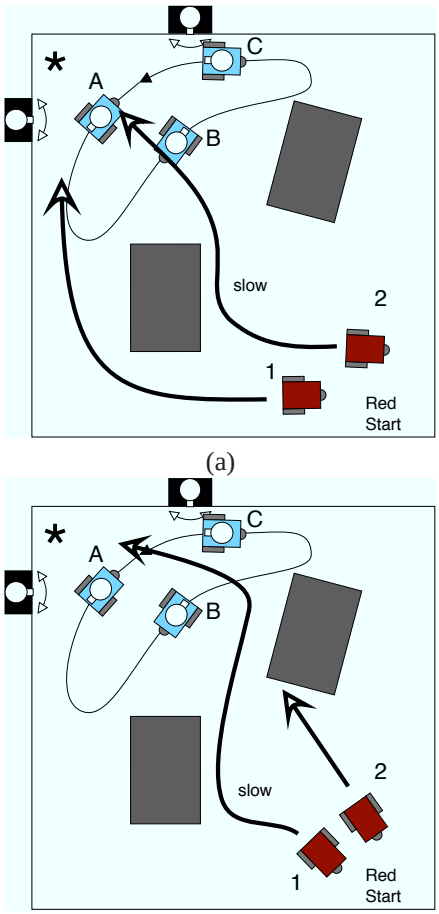



Figure 5. Learned red-team strategies: (a) strategy I and (b) strategy II.

**Naïve Strategy – Single Attacker** Figure 4(a) shows a naïve attack by a single red intruder. The attacker starts from the staging area and drives directly towards its goal.

**Naïve Strategy – Two Attackers** Figure 4(b) shows a naïve attack by two coordinated red intruders. The attacker on the left moves to the left of the obstacle and then to the goal, while the right attacker moves directly towards the goal. The intention is that one of the two attackers will draw the attention of the tracking system and mobile interceptors, while the other will make it to its goal.

**Learned Strategy I:** Figure 5(a) shows the first learned strategy. In this strategy (discovered by the learner), attacker 1 moves to the right of the obstacle, then moves towards the goal. Attacker 2 lingers in the area to the right of the left-most obstacle, slowly moving toward the center of the test area. At about the same time attacker 1 is intercepted (the learner anticipates this), attacker 2 turns towards the goal and drives to it at its maximum speed.

**Learned Strategy II:** Figure 5(b) shows the second learned strategy. Attacker 2 drives directly towards the corner of the right-most obstacle. Meanwhile, attacker 1 moves first to its left, then slowly moves across the surveillance area before turning towards the goal.

### 4.4. Results

**Naïve Strategies:** We ran the two described naïve attacks (and others) with the physical system. In all cases, the blue team defended successfully. We found that we either had sufficient interceptors to spot intruders, serendipitously initiating the pursuit mode, or the tracking by stationary agents would lead an interceptor to the correct position, thus initiating pursuit when the intruder is spotted.

It is important to note that our goal in this research is not to build the best blue team possible. Our blue team needs only to present a plausible surveillance and interception system that uses elements typically found in real surveillance systems. The naïve attacks verify that our blue team is plausible. However, if blue has flaws, so much the better, because we want to know that we can find them.

**Learned Strategy I:** Figure 6(a) shows the result of

learned strategy I. Attacker 1 follows its route to the left of the obstacle and turns toward the goal. Interceptor A sees attacker 1 and pursues, eventually meeting the requirement to remove the attacker from the scene. With attacker 1 removed, interceptor A is then free to return to its patrol, or assist another interceptor.

While attacker 1 takes these actions, attacker 2 quickly moves behind the obstacle, then moves slowly toward the center of the area. As it makes this movement, it is not seen by interceptors B and C (A is already occupied). The stationary camera at the top of the area sees attacker 2, and initiates a track. However, because the other stationary camera cannot see the attacker, the tracker cannot correctly triangulate the attacker's position, and the system cannot dispatch an interceptor to the correct location.

Eventually attacker 2 emerges to where both stationary cameras can see it. It is quickly identified and tracked correctly by the system, and interceptor B is dispatched. Also, interceptor A sees the attacker as it returns to its patrol pattern, and A and B both start to pursue attacker 2. However, due the decoy action of attacker 1, and the timing to get interceptor B in a distant position of the patrol pattern, attacker 2 reaches the goal before an interceptor can catch it.

We were able to duplicate this strategy with the physical system and see it succeed. The decoy behavior of attacker 1 is essential. We repeated the attack, but without the decoy. In that case interceptor A easily catches the attacker.

**Learned Strategy II:** Figure 6(b) shows the result of learned strategy II. Here, the *purpose* of attacker 2's curious path becomes clear. The blue team quickly finds attacker 2 and dispatches interceptor B. Interceptor B stops the attacker, but ends up in close proximity to the obstacle and is unable to maneuver away - it hits a dead end, at least in the context of its path planning algorithm - and is thus removed from participation.

As this happens, attacker 1 moves toward the center of the area, drawing interceptor A in pursuit. A fails in its pursuit as it nears the *stuck* interceptor A. This happens due to the collision-avoidance policy used by the interceptors - A waits for B, B is stuck, so neither moves. Interceptor C continues on its patrol, not seeing the attacker with its forward-facing camera, leaving the attacker an unimpeded path to its goal.

We did not duplicate this strategy in our physical system because the attackers and interceptors cooperate for collision avoidance (for obvious pragmatic reasons), and all the agents would get stuck by the obstacle. Given the time to rewrite some software to free the attacker from respecting the wide berth and cooperation implemented in the collision avoidance policy, we could have duplicated this attack too.

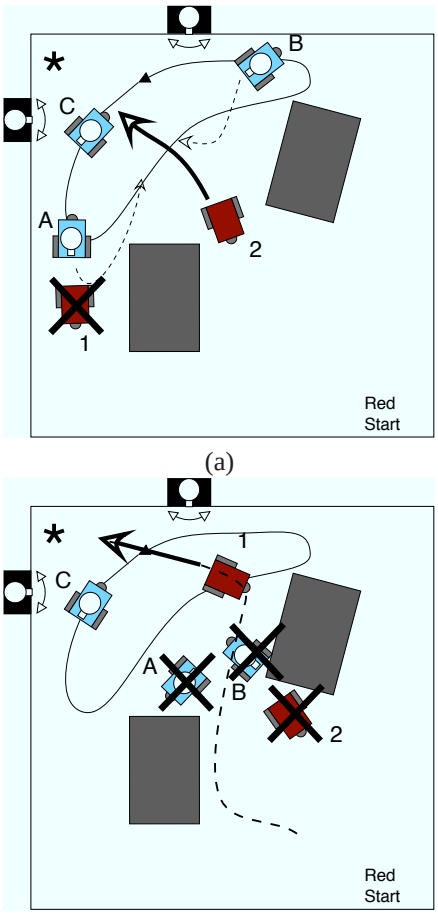



Figure 6. Outcomes from the learned red-team strategies.

## 5. Discussion

The two learned attacks revealed vulnerabilities in our blue-team's surveillance and intercept system. Fundamentally, all components of the blue team did their jobs correctly including path finding, target detection, tracking, and collision avoidance. Furthermore, the overall deployment was sufficient to defeat naïve attacks. Therefore, we conclude that the vulnerabilities were emergent, due to the interactions among components, not within the components themselves.

The learner succeeded in using *timing* to find a vulnerability. The naïve strategy with two attackers is superficially similar to the first learned strategy, but whereas the naïve strategy moves attackers at full speed throughout, the learned strategy slows down one attacker in a key position until there is a large gap in blue's defences. This would be impossible to test with a single component.

Exploitation of deployment policy is evident in the learned strategies. A successful attack only required one agent to reach the goal, so other agents could be sacrificed.

Policies to pursue under certain conditions then allowed an attacker to force the blue agents into a predictable action. The second learned attack exploited a collision avoidance policy to succeed. This attack also used a pursuit policy that ignored the possibility of a dead end to eliminate a defender from further action.

The interaction of vision components with other aspects of the system were evident in the successful attacks. At its simplest, the learner could know how far an attacker should remain from an interceptor. More interestingly, in the first learned attack, the successful attacker is in the view of one of the stationary cameras in the tracking system during the entire attack. Nevertheless, the tracker could not triangulate from a single camera and the attacker remained free of pursuit until it was much closer to the goal. We believe it would be possible to find attacks that deliberately try to confound the tracker's data association, although we did not find any examples in our trials.

Beyond the obvious application in testing for vulnerabilities, our approach could be used to tune parameters in a surveillance system. Given that any component has a number of parameters that affects its behavior, testing could reveal parameter selections that yield a vulnerable system.

Would it be possible to find these learned attacks without the assistance of the learner? Possibly, but we think not. While the first learned attack might have been seen with a bit of experimentation, the second learned attack seems unintuitive, exploiting a subtle *feature* of the agent's control system.

## 6. Conclusions

Over $1 billion has been spent on the *virtual fence* along the US-Mexico border [1]. It is obvious that security and surveillance systems can be large, complex, and expensive. This combination of cost and complexity mean new methods of testing are needed. We have presented one such method.

## References


[1] R. C. Archibold. Budget cut for fence on u.s.-mexico border. Retrieved August 30, 2010, from http://www.nytimes.com/-2010/03/17/us/17fence.html, March 2010.

[2] M. Atalla and J. Denzinger. Improving testing of multi-unit computer players for unwanted behavior using coordination macros. In *CIG'09: Proceedings of the 5th International Conference on Computational Intelligence and Games*, pages 355–362, Milano, Italy, 2009.

[3] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*. Academic Press, Boston, Massachusetts, 1988.

[4] B. Bhattacharya, M. Burmester, Y. Hu, E. Kranakis, Q. Shi, and A. Wiese. Optimal movement of mobile sensors for barrier coverage of a planar region. *Theoretical Computer Science*, 410:5515–5528, 2009.

[5] A. Chen, S. Kumar, and T. H. Lai. Designing localized algorithms for barrier coverage. In *MobiCom '07: International Conference on Mobile Computing and Networking*, pages 63–74, Montréal, Québec, Canada, 2007.

[6] J. Ferryman and A. Shahrokni. Pets2009: dataset and challenge. In *IEEE Workshop on Performance Evaluation of Tracking and Surveillance 2009*, pages 1–6, Snowbird, UT, 2009.

[7] T. Flanagan, C. Thornton, and J. Denzinger. Testing harbour patrol and interception policies using particle-swarm-based learning of behaviour. In *Proc. CISDA-09*, pages 1–8, Ottawa, Canada, 2009.

[8] A. Gelb, editor. *Applied Optimal Estimation*. MIT Press, Cambridge, Massachusetts, 1974.

[9] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968.

[10] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[11] J. Kidney and J. Denzinger. Testing the limits of emergent behavior in mas using learning of cooperative behavior. In *Proceeding of the 2006 conference on ECAI 2006*, pages 260–264, Riva del Garda, Italy, August 2006.

[12] E. Kranakis, D. Krizanc, L. Narayanan, and K. Xu. Inapproximability of the perimeter defense problem. In *21st Canadian Conference on Computational Geometry*, Vancouver, British Columbia, August 2009.

[13] S. Kumar, T. H. Lai, and A. Arora. Barrier coverage with wireless sensors. In *MobiCom '05: International Conference on Mobile Computing and Networking*, pages 284–298, Cologne, Germany, 2005.

[14] H. Kwong and C. Jacob. Evolutionary exploration of dynamic swarm behaviour. In *Congress on Evolutionary Computation*, volume 1, pages 367–374, Canberra, Australia, December 2003.

[15] E. Lipton. U.s. project to secure borders will begin in arizona desert. Retrieved August 30, 2010, from http://www.nytimes.com/2006/09/22/us/22border.html, September 2006.

[16] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *MobiCom '00: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 32–43, Boston, Massachusetts, United States, August 2000.

[17] P.Stone and M. Veloso. Multiagent systems: a survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[18] M. Reyes-Sierra and C. C. Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *Int. Jour. Comp. Int. Res.*, 2(3):287–308, 2006.

[19] C. Reynolds. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4), July 1991.