Abstract—We propose a hierarchical cooperative response framework for containment of value-changing attacks in large-scale hierarchical critical infrastructures. We define a notion of attack container, which is a logical entity that captures the behavior of a group of nodes and aims to contain the damage of the attack. This entity is a basic entity that is used for distributed attack detection and cooperative response in our framework. The simulation results show that our scheme can mitigate and contain large-scale attacks very fast.

I. INTRODUCTION

Critical Infrastructure is defined as "systems and assets, whether physical or virtual, so vital to the United States that their incapacity or destruction would have a debilitating impact on security, national economic security, national public health or safety, or any combination of those matters."[1]. Industry sectors such as power grid, telecommunication, transportation and financial systems are considered critical infrastructures, which offer a wide variety of important services that our society relies on. It is a fact that these critical services are supported by large-scale computer information systems. Although in supporting roles, these systems are so important that losses of them lead to the reduction or even disruption of the critical infrastructure services.

Unfortunately, these computer information systems are vulnerable to cyber-attacks as they move from isolated systems with propriety protocols to systems with COTS components. Many threats and vulnerabilities have been reported[2]. Even worse, many critical infrastructures are interdependent. For example, transportation management systems will be disrupted if there is a wide loss of power. Another example could be a loss of the communication service leading to the disruption of other information systems such as financial systems.

Our view of the computing critical infrastructure systems is large-scale in nature and organized in multi-tier and hierarchical structure1. We are particularly interested in the effect of large-scale attacks through virus and worm propagation of events in sensor networks where the leave nodes (e.g. sensing digital devices) of the hierarchy get infected via external attackers (e.g. through the maintenance of digital measurement devices). Particularly, we will investigate an attack container framework that will quickly detect, response to and contain such attacks.

In the prior work, distributed detection has been an active research areas. These techniques, such as those in [6][7][8] are particularly interesting since they provide solutions for a quick(est) event detection. However, they are parametric methods which require known probability distribution in advance. Other non-parametric methods are useful as well even though they usually address some specific changes such as mean-changing or variance-changing detection [9][10].

Regarding worm and virus propagation and prevention, there have been many results as well. In [11][12], epidemiology is proposed for viral infection process. Some other researchers have studied the immunization defense against virus propagation [4].

The notion of “cooperative response” also has been considered. In [13], Nojiri considers an abstract model of “friends” protocol where nodes somehow could detect that they are compromised and warn their friends of the presence of the worm. Virtually, there will be a race between alert propagation and worm propagation through the network. However, how to perform local detection and response are not given in the paper. In [14], the author investigates “friends” protocol under a hierarchical structure. However, they only consider coordination between children and parent and do not consider any peer-to-peer cooperation. In [15], the notion of cooperation mechanism is generalized by two cooperative mechanisms: “implicit signaling” in which malicious packets are marked to alert other hosts and “explicit signaling” which are alerts exchanged among hosts.

Our approach to fast containment of viral or worm attacks in large-scale hierarchical computing critical infrastructure is to develop an attack container framework. This framework includes distributed monitoring and detection, cooperative response and mitigation. Distributed monitoring and detection

*This material is based upon work supported by the National Science Foundation under Grant CNS-0524695. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1This structure can be found in many critical infrastructure such as banking and financial networks or energy-control networks [3][4][5].
allow the system to quickly detect abnormal and critical events happening in the field. Each node in the network plays a monitoring role in the environment. We use a non-parametric Cumulative Sum (CUSUM) algorithm for quick detection of abrupt measurement changes in the environment. Furthermore, we can also quantify the severity of the change. The ability to detect and quantify events quickly enables a cooperative response strategy to large-scale attacks. Based on the attack severity, nodes act locally and inform others to mitigate and contain the attacks before they can spread out to the whole network. Due to the complexity and stochastic nature of the systems, which is difficult to capture by analytical models, we take the simulation approach. Our simulation results show that our framework can contain the large-scale attacks effectively.

The rest of the paper is organized as follows. First, we give our system models and assumption in section II. Then we present our attack container framework in section III and the detail of attack container in IV. We will give the detail of our cooperative response protocol in section V. Section VI show our simulation setup and results. Finally, we conclude the paper in section VII.

II. SYSTEM MODELS & ASSUMPTIONS
A. Network Model

We model the underlying communication network of critical infrastructure as a tree-like structure as shown in Figure 1. Nodes are connected with their parent, siblings and children. In this model, when we talk about “neighbors” of a node A, we mean nodes that are in the same level and directly connected with A.

Leave nodes in the tree are sensing nodes (called as sensors). Sensors are digital devices attached to physical measurement devices that capture the measurement value in digital form and send this value to the higher level intermediate digital nodes for control\(^2\). Other nodes in the network are called intermediate nodes who receive and process data from their children, alert their neighbors and parent if necessary. They can also issue commands to their children to react to special events.

Depending on the critical infrastructure, links between these nodes can be either wired or wireless. For example, in the Power networks, Intelligent Electronic Devices - IEDs (i.e. sensors) communicate with Remote Terminal Units - RTUs (i.e. intermediate nodes in second layer) by serial communication such as RS-232. RTUs can communicate with its parent, Control Center by dedicated phone lines, radios or Wide Area Networks [5][16][17].

For notation convenience, we denote children\((x)\), degree\((y)\) as the set of children of node \(x\) and the degree of node \(y\).

B. Data Model

Data reported by sensors could be temperature, voltage or water level. However, we assume that each correct data measurement reading has a range \((min, max)\).

We model the reported data by sensors as random variables \(X_i\) whose mean is fixed under the normal condition. Furthermore, we assume the mean are identical for all sensors. It means we assume a homogenous environment in which sensors have identical statistical behaviors. We denote \(\mu_i, \sigma_i\) as the mean and variance of \(X_i\) i.e. \(E[X_i], Var[X_i]\) and a range of \((\mu_{min}, \mu_{max})\) and \((\sigma_{min}, \sigma_{max})\). These ranges can be calculated if we know the characteristics of the data.

Data is sampled and reported periodically. Each node in the network expects to receive data from children periodically and is expected to send data periodically. The period of data reports may be different at different layers of the hierarchy. For example, deadlines for reporting data from IEDs to RTUs are 2-4 seconds [5]. Deadlines for upper layers may be longer in the order of minutes.

C. Trust Assumptions

We assume all nodes except sensors are trusted. Specifically, the operating systems are trusted and the software running on these nodes is tampering-resistant. This can be achieved with the support of hardware such as eXecute Only Memory architecture (XOM) [18] or Trusted Platform Module (TPM) [19]. These techniques basically can prevent modifications on executed software. However, they cannot prevent Denial of Service attacks.

We also assume that all nodes except sensors will trust each other. Since intermediate nodes are already trusted, this assumption can be achieved by using secure communication mechanism (authentication, encryption).

However, we assume weak security assumptions on sensor nodes (e.g. no trusted software). These nodes can be compromised and be used for some specific purposes. They can be infected by viruses via external updates or they can just simply fail and expose random behaviors.

D. Threats

1) External factors: Although the critical infrastructure networks are usually isolated from open shared networks, they get connected to external machines. (e.g. vendors do software updates on sensing devices \(^3\)). These external updates introduce threats for injecting flash worms into the system.

\(^2\)Here we consider sensors that are different from motes. Power-consumption is not an issue.

\(^3\)Updating software from vendors can also be a threat. Since sensors are proprietary devices, they are usually updated and maintained by vendors. If the maintainer’s machine is infected, it could infect the sensors as well. [2]
2) Internal factors: Once a sensor is infected or compromised, it is fairly easy to infect neighbor nodes because they can directly talk to each other due to a given protocol distribution. This behavior can cause a dramatic spread of worm in the network.

E. Attack/Failure Model

Once a sensor is infected or compromised through some of the threats listed above, it will expose abnormal behaviors on readings. The first type of attacks, value-changing attacks, would be changing readings maliciously. Specifically, it could report random readings due to infection or it could shift the mean of reading values arbitrarily. We call the former attack as random-changing attack and the latter as mean-changing attack. The consequence of this type of attacks, either intentionally or unintentionally, is reporting of malicious values possibly leading to false alarms of the critical infrastructure systems, inappropriate decisions of the operators or preferably catastrophic failures of the critical infrastructure systems.

The second type of attacks would be changing reading patterns to cause DoS (Denial of Service) or WoS (Withdrawal of Service) where the readings can be flooded or delayed arbitrarily. Although our framework enables the detection of both types of attacks, we only consider the first type in this paper.

F. Infection model

The infection process caused by external factors has a constant rate $\alpha$ since it is independent with the number of infected nodes and the underlying network topology. We model this infection process as follows. The infection process happens as if there were an agent that keeps infecting sensors one-by-one randomly and uniformly. This is a reasonable model because that agent (external vendor), intentionally or accidentally, just “wants” to randomly choose one sensor to update/inject and let the virus itself propagate through the internal network.

In contrasts, the infection process rate caused by internal factors depends on the number of infected nodes and the underlying network topology. To model this spreading behavior, we adopt the epidemic models proposed by Kermack-McKendrick [20] in which a node has three states: “susceptible”, “infectious” and “removed”. At the beginning, a fraction of nodes is in “infectious” state while the rest of nodes is in “susceptible” state. Once node is “removed”, it will not be infected again. This is mainly because once a compromised sensor is detected, it will be isolated until it is correctly patched and becomes immune. Therefore, a node has either the state transition “susceptible $\rightarrow$ infectious $\rightarrow$ removed” or stays in “susceptible” forever.

Since a sensor node can communicate with multiple neighbors by multicast, we assume an infected sensor can choose any subset of its neighbors to infect.

III. ATTACK CONTAINER FRAMEWORK

The goal of our system is monitoring, detecting and isolating infected sensors as soon as possible, under communication and false alarm rate constraints.

As mentioned in section II-E, we only consider value-changing attacks. In the subsequent sections, first we will show our approach to achieve the goals. Then, we give the details of our attack container framework including protocols and algorithms for monitoring, detection and response.

A. Attack Container

Attack Container is a logical entity, defined by a group, that keeps track of the behavior of nodes in the group. It is represented by a data structure and corresponding operations (see section IV). The container data could be built from readings of sensors or others’ attack container data. A monitoring node builds its container from sensors’ readings. The root of a tree can build the container by aggregating its children containers and peers containers (see Figure 2). A more detail of attack container construction and its operations will be given in section IV.

Attack Container is a basic entity that will be used in our distributed monitoring, detection and cooperative response in our framework, which we will show next.

B. Approach

To deal with the value-changing infection attack in a large-scale system, we use a distributed monitoring and detection approach and a cooperative response strategy.

- Distributed Monitoring and Detection: Each intermediate node plays a monitoring role in the hierarchy. It maintains an attack container for its sub-tree, which is updated and aggregated on receiving either readings or attack containers from the children. It performs a non-parametric CUSUM on the attack container to detect value-changing attacks. In this manner, the monitoring and detection is distributed among nodes which enables the fast detection capability.

- Cooperative Response Strategy: Once an intermediate node detects a potential value-changing attack in its subtree, it evaluates the severity and informs its neighbors and parent nodes. If the severity is critical, it can solely decide without waiting information from its parents and peers.

The rate at which an intermediate node updates its neighbors and parent on its subtree condition depends on the severity of the event. However, the maximum rate is limited appropriately to prevent a flooding of exchanged messages among intermediate nodes during emergency situation.

The coordination among nodes forms a cooperative response strategy which enables early warning or a containment for a large-scale attack.

IV. OPERATIONS OF ATTACK CONTAINER

In our framework, an attack container is guided by a pair of two metrics, abnormality and severity (see Figure 2 for illustration) which are defined as follows.
A. Severity
A severity indicates how severe the attack happening in the group in terms of sensor readings. Severity metric must be aggregatable i.e. it is possible to compute a severity of a node from other severities. To be precise, we define the severity of a sensor and intermediate nodes as follows.

1) Severity of a sensor: A severity $S(i)$ of a sensor $i$ is measured by the deviation of its mean and variance values under attack from the mean and variance values in normal condition. The reasons for using both mean and variance value is that the mean can show the trend while the variance captures random behaviors of the data.

To make the severity aggregatable, we normalize its to the range of $[0..1]$ as follows.

Let $\bar{\mu}_i, \mu_i$ be the deviated mean, normal mean of $X_i$, the values reported by sensor $i$.

$$S(i) = (\bar{\mu}_i - \mu_i) / (\mu_{max} - \mu_{min})$$

With the assumption that $\bar{\mu}_i$ can only be greater than $\mu_i$, $S(i)$ can take only values from $[0..1]$.

The variance can be taken into account of the severity similarly.

2) Severity of an intermediate node: A severity of an intermediate node $k$ is an average of severity of its children. Formally,

$$S(k) = \sum_{j \in children(k)} S(j)/|children(k)|$$ (1)

Apparently, $S(k)$ also takes only values from $[0..1]$.

It is now becoming obvious that severity could be aggregatable by the above definition.

B. Abnormality
An abnormality $A(i)$ of an intermediate node $i$ is the fraction of abnormal children in the subtree over the total number of its children. Formally,

$$A(i) = \sum_{j \in children(i)} A(j)/|children(k)|$$ (2)

It is fairly easy to see that the abnormality metric is in the range $[0..1]$ and is aggregatable.

C. Aggregation Operation
The aggregation operation of an attack container is defined as in Equation 1 and Equation 2.

D. Value-Changing Detection and Cumulative Sum Monitoring Box (CMB)
The core of the container attack is the monitoring operation, which is done by Cumulative Sum Monitoring Box (CMB). A CMB is used to monitor the changes of a data stream. It processes the data stream and generates alerts based on two thresholds: abnormal threshold and critical threshold, which are set during the setup phase. The CMB could return “normal” if no change in data stream is detected, “abnormal” if the change exceeds abnormal threshold and “critical” if the change exceeds critical threshold. The illustration of CMB is shown in Figure 3.

CMB uses non-parametric Cumulative SUM change-detection algorithm as its value-changing detection algorithm. Essentially, the Cumulative Sum Change-Point keeps the positive part of the log-likelihood ratio and triggers an alarm if the cumulation exceeds the threshold. The threshold is set according to the required false alarm rate. The non-parametric version extends the parametric method by estimating the changes based on historical observation. Due to limit of space,
we will not discuss further here. The detail of non-parametric algorithm could be found in [10] (Chapter 4) and [9].

According to the space requirement of the CMB and CUSUM algorithm, a CMB needs to maintain two thresholds and internal counter. This requires little space of memory. The non-parametric CUSUM change-detection is also very lightweight in terms of computation. These two characteristics are important for the use of CMB in large-scale networks.

For notation convenience, we denote $CMB(S)$ as the current state of the data stream $S$.

V. Cooperative Response Protocol

In this section, we describe in detail the protocol to perform distributed monitoring and detection and cooperative response strategy.

To enable the cooperative response, each intermediate node $i$ keeps two attack containers: one for its children and the other one for its peers. We denote them as $C_{children}(i)$ and $C_{peers}(i)$, respectively.

For each attack container (i.e. either $C_{children}(i)$ or $C_{peers}(i)$), the two abnormality $A(t)$ and severity $S(t)$ streams of the container will be given to two CMB boxes $CMB(A(t))$ and $CMB(S(t))$ for monitoring and detection. Therefore, in total, each node $i$ has four CMB boxes for monitoring and detection.

On receiving any attack container data or readings from children or peers, intermediate node $i$ aggregates and updates its attack containers $C_{children}(i)$ and $C_{peers}(i)$ accordingly. The behavior of node $i$ depends on the states returned from the four CMB boxes, which are defined as follows.

- If all CMB boxes indicate “NORMAL”, node $i$ just operates normally. It sends its attack container to its parent and peers periodically at a default reporting rate. If the reporting rates are higher than the default ones due to some previous adjustment, it reduces the rates additively.
- If any box indicates “ABNORMAL” and the others indicates “NORMAL”, the node’s state transits to “ABNORMAL” state. It doubles the reporting rates to the parent and peers. The reason for this behavior is that the “ABNORMAL” state only indicates there is a minor unusual behavior of children and peers. The node only needs to be more active in reporting its condition.
- If any box indicates “CRITICAL”, the node’s state transit to “CRITICAL”. It is mainly because there is either a large number of sensors are abnormal or the the reported readings are very unusual. This is likely to be a starting point for a large-scale attack.

On this transition, the node first sends “ALERT” message to its children which basically tells them to block any external access to avoid getting infected. Then, it also sends an “ALERT” message attached with its attack container to its parent and peers.

It is important to emphasize that when an intermediate node receives an “ALERT” message, it will not be alerted immediately. A node only alerts when it observes a significant number of alerted peers or children, which is captured by the attack containers of its children and peers.

VI. Simulation Study

We have implemented an event-based packet-level simulator in C++ for the evaluation of our scheme. For each experiment, we repeat it 10 times with different seeds and report the average value as the final results. We use the number of infected nodes as the metric for the evaluation.

A. Simulation Setup

We evaluate our scheme in a hierarchical network with four levels. Nodes at each level have the same number of children and number of peers. Links at each level will also be assigned different delay. The parameters are shown in Table I.

B. Scenarios

We evaluate our scheme under three scenarios A, B and C.

In scenario A, the external vendor/attacker will pick randomly a sensor to update/inject independent in which group it resides. After the inject/update, he will wait for $d_{\text{external}}$ time units before picking another sensor.

In scenario B, the external vendor will randomly pick a group and then select a sensor in that group to inject/update. After the inject/update, he will wait for $d_{\text{external}}$ time units before picking another group that is not injected.

In scenario C, the external vendor will pick randomly a subset of sensors to update/inject simultaneously. After each inject/update, he will wait for $d_{\text{external}}$ time units before picking another random subset.

In each scenario, the simulation will run until all sensors are either in “ALERTED” or “INFECTED” state.

C. Results

Figure 4 and 5 show the number of infected nodes and alerted nodes under the system with (and without) our scheme in each scenario. The x-axes represents the delay of external vendor $d_{\text{external}}$ and the y-axes is the number of infected nodes when the simulation stops at $t=500$. As we can see that, our scheme can reduce a significant number of infected nodes under three scenarios.

VII. Conclusion

We just introduce the concept of attack container and propose an attack container framework for large-scale attack mitigation in hierarchical infrastructure. The concept of attack container is important in the sense that it provides a uniform view for each node about the behavior of its group as well as other peer’s groups. This characteristic enables the distributed detection and cooperative response capability.

We also give a protocol for cooperative response based on our proposed framework. The simulation results clearly show that our scheme can mitigate and contain large-scale attacks under various scenarios.

4In this case, the vendor/attacker could pick multiple sensors in the same group.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Children</td>
<td>[5,10,15]</td>
<td>#children for each node at each level (fanout degree)</td>
</tr>
<tr>
<td>#Sensors</td>
<td>750</td>
<td>#sensors in the network</td>
</tr>
<tr>
<td>#Nodes</td>
<td>805</td>
<td>total #nodes in the network, including sensors and intermediate nodes</td>
</tr>
<tr>
<td>Children link weights</td>
<td>[2.0, 1.0, 0.5]</td>
<td>delays of link from parent to children at each level</td>
</tr>
<tr>
<td>Peers link weights</td>
<td>[0.0, 2.0, 1.0, 0.5]</td>
<td>delays of peer-to-peer link at each level</td>
</tr>
<tr>
<td>Simulation time</td>
<td>500</td>
<td>simulation time</td>
</tr>
<tr>
<td>Peer degree</td>
<td>50%</td>
<td>50% of nodes in the same level are connected peers</td>
</tr>
<tr>
<td>Default reporting rate</td>
<td>2 msgs / 1 time unit</td>
<td>Default rate at which nodes report attack containers</td>
</tr>
<tr>
<td>Virus infection delay</td>
<td>1 time unit</td>
<td>Delay a virus spends at infected sensor before infecting neighbor sensors (incubation time)</td>
</tr>
</tbody>
</table>

**Table I**

SIMULATION PARAMETERS

Fig. 4. Number of infected sensors vs. delay of external vendor ($d_{external}$) (time unit)

(a) Scenario A  
(b) Scenario B  
(c) Scenario C

Fig. 5. Number of alerted sensors vs. delay of external vendor ($d_{external}$) (time unit)

(a) Scenario A  
(b) Scenario B  
(c) Scenario C

**REFERENCES**

[1] “United and strengthening america by providing appropriate tools required to intercept and obstruct terrorism (usa patriot) act, p.l. 107-56, title x, section 1016.”


