# CAPTAR: Causal-Polytree-based Anomaly Reasoning for SCADA Networks

Wenyu Ren*, Tuo Yu*, Timothy Yardley*, Klara Nahrstedt*
* University of Illinois Urbana-Champaign, Urbana, Illinois, USA
Email: {wren3, tuoyu2, yardley, klara}@illinois.edu

***Abstract*—The Supervisory Control and Data Acquisition (SCADA) system is the most commonly used industrial control system but is subject to a wide range of serious threats. Intrusion detection systems are deployed to promote the security of SCADA systems, but they continuously generate tremendous number of alerts without further comprehending them. There is a need for an efficient system to correlate alerts and discover attack strategies to provide explainable situational awareness to SCADA operators. In this paper, we present a causal-polytree-based anomaly reasoning framework for SCADA networks, named CAPTAR. CAPTAR takes the meta-alerts from our previous anomaly detection framework EDMAND, correlates the them using a naive Bayes classifier, and matches them to predefined causal polytrees. Utilizing Bayesian inference on the causal polytrees, CAPTAR can produces a high-level view of the security state of the protected SCADA network. Experiments on a prototype of CAPTAR proves its anomaly reasoning ability and its capabilities of satisfying the real-time reasoning requirement.**

***Index Terms*—Smart Grid, SCADA, causal analysis, anomaly reasoning**

## I. Introduction

Nowadays, large-scale distributed critical infrastructure systems such as power grids and refineries increasingly rely on digital industrial control systems (ICSs) for real-time monitoring, data collection, and control. The Supervisory Control and Data Acquisition (SCADA) system is the most commonly used ICS. Critical as they are, SCADA systems are subject to a wide range of serious threats [1]. Therefore, securing SCADA systems against various threats and vulnerabilities has become a major challenge.

To promote the security of SCADA systems, intrusion detection systems (IDSs) are increasingly deployed by SCADA operators. As the name suggests, the main objective of IDSs is to monitor the system, detect suspicious activities caused by intrusion attempts, and report alerts to the system operators. Although IDSs play an undeniable role in the protection of SCADA systems, they still suffer from some defects. The biggest issue with traditional IDSs is that they continuously generate tremendous number of alerts without further comprehending them. Drowned in an ocean of unstructured alerts mixed with false positives, SCADA operators are almost blind to see any useful information. Due to the high volume and low quality of the alerts, it becomes a nearly impossible task for the operators to figure out the complete pictures of the attacks and take appropriate actions in a timely manner.

978-1-5386-8099-5/19/$31.00 ©2019 IEEE

To address the aforementioned problem of traditional IDSs and provide the SCADA operators with *explainable situational awareness*, there is a need for an efficient system to aggregate redundant alerts from IDSs, correlate them in an intelligent manner, and discover attack strategies based on domain knowledge as well as causal reasoning. In a previous work [1], we present an edge-based multi-level anomaly detection framework for SCADA, named EDMAND. EDMAND resides at the edges of the SCADA network and detects anomalies in multiple levels of the network. The triggered alerts are aggregated, prioritized, and sent to the control center. In this paper, we present a causal-polytree-based anomaly reasoning framework for SCADA networks, named CAPTAR. CAPTAR resides in the control center of the SCADA network and takes the meta-alerts from EDMAND as input (shown in Fig. 1). CAPTAR correlates the alerts using a naive Bayes classifier and matches them to predefined causal polytrees. Utilizing Bayesian inference on the causal polytrees, CAPTAR can reveal the attack scenarios from the alerts and produces a high-level view of the security state of the protected SCADA network.

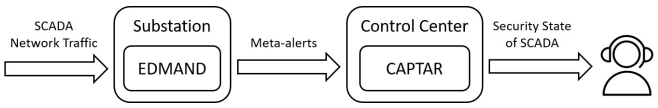


Fig. 1: Locations of EDMAND and CAPTAR

The remainder of this paper is organized as follows: Section II reviews the related work. Section III introduces the basic concept Bayesian inference by belief propagation and two canonical models which are used to build our causal polytrees. Section IV gives an overview of the design of CAPTAR. Section V shows the performance evaluation of CAPTAR and Section VI concludes the paper.

## II. Related Work

Various techniques have been used to measure the similarity of common features of alerts to correlate them [2]–[5]. However, alert correlation alone can only measure the correlation strength between alerts and are not sufficient to recognize the whole picture of the attack. To fill the gap, many works have been proposed in the area of attack plan recognition. Some works [6], [7] keep the state of the system and assume that the state evolves towards a "worse" direction during attacks. There are also works [8], [9] that define prerequisites and consequences of each attack step and construct

chains or graphs based on the matching of prerequisites and consequences. Bayesian networks are also utilized by many papers [8], [10]–[13] to correlate alerts or to represent and infer the causal relationship between attack steps. The closest previous work [14] to ours is the integration of alert aggregation, prioritization, correlation, and attack plan recognition. Three alert correlation methods are proposed: probabilistic-based, causal discovery-based, and temporal based methods. The attack plan recognition step also uses causal polytrees to represent attack plans.

CAPTAR mainly differentiates from all previous works in two aspects. First, the alerts received by CAPTAR are meta-alerts generated by EDMAND, which is our edge-based multi-level anomaly detection framework for SCADA. EDMAND applies network-based rather than host-based detection and it mainly takes the anomaly-based approach instead of the signature-based approach. The alerts from EDMAND do not directly relate to each attack step in the attack plan but instead relate to various network behaviors triggered by each attack step. Therefore, mapping between alerts from EDMAND and underlying attack steps is necessary for our anomaly reasoning. Second, we define the concept of confidence score for each alert in EDMAND. In CAPTAR, the confidence scores of meta-alerts are utilized to calculate the diagnostic support for each node in the causal polytrees during the belief propagation. This allows each alert to carry more belief information instead of only a binary state (exist/not exist).

## III. Preliminaries

Before describing the design of CAPTAR, we first introduce the belief propagation algorithm to conduct Bayesian inference. Then, we present two canonical models we use to build our causal trees: "noisy-OR" and"noisy-AND" models.

### A. Belief Propagation

Belief propagation via message passing [15] is an algorithm to conduct inference on Bayesian networks. To make it clearer, we first illustrate the belief propagation rules in a general tree-structured Bayesian network where a node might have several children and one parent. In the next subsection, we will introduce the two canonical models which generalize our causal trees to polytrees.

We illustrate the belief propagation by specifying the activities of a typical node $X$ having $m$ children, $Y_1$, $Y_2$, ..., $Y_m$, and a parent $U$ as shown in Fig. 2. The belief in the various values of $X$ depends on two distinct sets of evidence: evidence from the sub-tree rooted at $X$, and the evidence from the rest of the tree. In general, let us define $\boldsymbol{e}_X^-$ as the evidence contained in the tree rooted at $X$ and define $\boldsymbol{e}_X^+$ as the evidence contained in the rest of the network. $\boldsymbol{e}_{Y_j}^-$ therefore represents the evidence from the sub-tree rooted at $Y_j$ where $j \in \{1, \ldots, m\}$. $x \in \{0, 1\}$ is a particular value of $X$ and $u \in \{0, 1\}$ is a particular value of $U$. The belief distribution of variable $X$ can be calculated based on the following three kinds of parameters:

1) *Causal* Support: $\pi_X(u) = P(u|\boldsymbol{e}_X^+)$, contributed by parent of $X$.
2) *Diagnostic* Support: $\lambda_{Y_j}(x) = P(\boldsymbol{e}_{Y_j}^-|x)$, contributed by the $Y_j$ which is the $j$-th child of $X$ where $j \in \{1, \ldots, m\}$.
3) Conditional Probability Table (CPT): $P(x|u)$ that relates the variable $X$ to its direct parent $U$. Each entry $P(x|u)$ in the table defines the probability of value $x$ of node $X$ given certain value $u$ of node $U$.

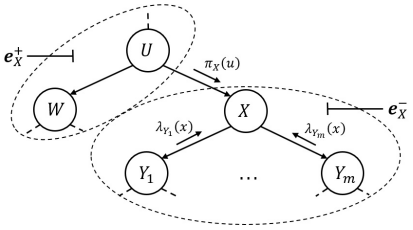


Fig. 2: Fragment of a causal tree, showing different kinds of evidence and support of a node $X$

The belief propagation algorithm runs whenever new evidence is found in the tree. The propagation starts from the node which receives the new evidence and the new belief propagates along the edges of the tree until all nodes get updated. The local belief updating at each node $X$ can be executed by three steps in any order.

**Belief Propagation Algorithm**

**Step 1 — Belief updating:** Node $X$ updates its belief measure based on the $\pi_X(u)$ message from its parent and the messages $\lambda_{Y_1}(x)$, $\lambda_{Y_2}(x)$, ..., $\lambda_{Y_m}(x)$ from each of its children as shown in Fig. 2.

$$BEL(x) = \alpha\lambda(x)\pi(x), \tag{1}$$

where $\lambda(x) = \prod_j \lambda_{Y_j}(x)$, $\pi(x) = \sum_u P(x|u)\pi_X(u)$, and $\alpha$ is a normalizing constant rendering $\sum_x BEL(x) = 1$.

**Step 2 — Bottom-up propagation:** Node $X$ computes a new message $\lambda_X(u)$ based on its CPT and $\lambda$ messages received from its children. Then $X$ sends $\lambda_X(u)$ to its parent $U$.

$$\lambda_X(u) = \sum_x \lambda(x)P(x|u), \tag{2}$$

**Step 3 — Top-down propagation:** Node $X$ computes new $\pi$ messages and sends them to its children. The new $\pi_{Y_j}(x)$ message for its $j$-th child $Y_j$ is calculated as

$$\pi_{Y_j}(x) = \alpha\pi(x) \prod_{k \neq j} \lambda_{Y_k}(x). \tag{3}$$

Boundary conditions and more details about the derivation of the algorithm can be found in [15].

### B. The "noisy-OR" and"noisy-AND" Models

In Section III-A, we illustrate the belief propagation algorithm in a general tree-structured Bayesian network where a node has at most one parent. However, this structure lacks the ability to represent nodes that might have multiple causes (i.e., node may have multiple parents). In this subsection, we introduce two canonical models which allow us to generalize our causal trees to causal polytrees. The difference between a polytree and a normal tree is that a node could have multiple parents in a polytree. The two canonical models

contain structures similar to logical OR-gate and AND-gate with noises and are thus called "noisy-OR" and "noisy-AND" models. The characteristics of these two typical structures enable us to conduct the belief updating more efficiently in polytrees.

The "noisy-OR" model [15] is based on the noisy OR-gate structure shown in Fig. 3. Each node represents an event with binary state 0 or 1. For a node $X$ with $n$ parents $\boldsymbol{U} = \{U_1, U_2, \ldots, U_n\}$, its value can be seen as the output of a logical OR-gate. Each input to the OR-gate is the output of an AND-gate representing the conjunction of $U_i$ and the negation of its specific inhibitory mechanism $I_i$. The inhibitors $I_1, \ldots, I_n$ represent exceptions or abnormalities that interfere with the normal relationship between $\boldsymbol{U}$ and $X$. We use $q_i$ to represent the probability that the $i$-th inhibitor is active. Assume all inputs are 0 except $U_i = 1$. $X$ will only be 1 if and only if the inhibitor $I_i$ associated with $U_i$ remains inactive. That is, $P(X = 1|U_i = 1, U_k = 0\ k \neq i) = 1 - q_i$. Therefore, $c_i = 1 - q_i$ represents the degree to which the single cause $U_i = 1$ can endorse the consequent event $X = 1$. Let $\boldsymbol{u} = (u_1, u_2, \ldots, u_n) \quad u_i \in \{0, 1\}$ represent any assignment of values to parent set $\boldsymbol{U}$. Note that both $\boldsymbol{u}$ and $\boldsymbol{U}$ are vectors since $X$ could have multiple parents. Let $T_u = \{i : U_i = 1\}$ represent the subset of parents that are 1. In the "noisy-OR" model, a link matrix $P(x|\boldsymbol{u})$ is used to relate $X$ to its parent set $\boldsymbol{U}$ and can be written as

$$P(x|\boldsymbol{u}) = \begin{cases} \prod_{i \in T_u} q_i & x = 0 \\ 1 - \prod_{i \in T_u} q_i & x = 1. \end{cases} \tag{4}$$

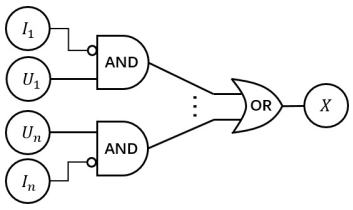


Fig. 3: The noisy OR-gate

Having the link matrix $P(x|\boldsymbol{u})$, we can follow similar belief propagation algorithm described in Section III-A. The belief propagation algorithm in "noisy-OR" model also has three steps and the details can be found in [16].

The "noisy-AND" model [15] is based on the noisy AND-gate structure and has very similar properties to the "noisy-OR" model. More details about the characteristics of the "noisy-AND" model and the belief propagation algorithm in the model are included in [16].

## IV. Design Overview

As we mentioned in Section I, CAPTAR resides in the control center of the SCADA network and its inputs are meta-alerts sent by EDMAND at the edge of the network. In this section, we present a design overview of CAPTAR. The main architecture of CAPTAR is shown in Fig. 4. CAPTAR consists of 4 components: (1)*Meta-alert Database*, (2)*Attack Template Database*, (3)*Alert Correlator*, (4)*Causal Reasoning Engine*.

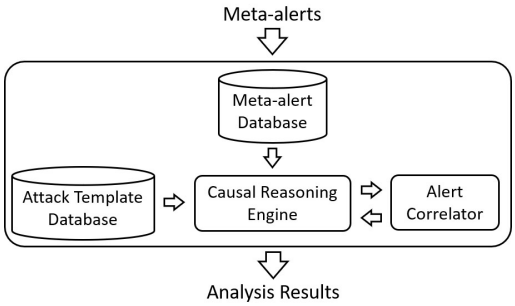


Fig. 4: CAPTAR architecture

The meta-alert database is used to store the meta-alerts from EDMAND which serve as evidence to our causal reasoning of anomalies. The attack template database stores the potential attack templates which are causal polytrees created by domain experts. The alert correlator takes two meta-alerts as inputs and outputs a correlation score which is used to decide whether the two input meta-alerts are correlated or not. The core component of CAPTAR is the causal reasoning engine which interacts with all other three components. When the causal reasoning engine is started, it fetches copies of the attack templates in the database and conducts alert matching as well as belief propagation on them. The meta-alerts are retrieved from the meta-alert database and the alert matching is done using the alert correlator. Whenever the belief of an attack is high enough, the engine outputs the causal polytree corresponding to that attack with matched alerts. The operator can further analyze the believes and matched alerts in the causal polytree to understand each step of the attack. In the following subsections, we will introduce the meta-alert, the attack template, the alert correlator, and the causal reasoning engine in more detail.

### A. Meta-alert

Meta-alerts are generated by EDMAND and sent to the control center where CAPTAR resides. Each meta-alert is the aggregation of similar alerts and has several fields. The fields that will be used in CAPTAR are alert id, alert type, index field, timestamp, confidence score. Alert id is a string that is unique for each meta-alert. The received meta-alerts from EDMAND will be first stored in the meta-alert database and the alert id serves as the key to locate and retrieve the meta-alert from the database. Alert type is a name that briefly describes the meta-alert. The current prototype of EDMAND generates 24 types of alerts from the transport, operation, and content levels and a complete list can be found in [16]. For simplicity reason, we assign an alert type index to each alert type and we will use the index to represent the corresponding alert type. Index field of the meta-alert contains additional information that helps to describe the meta-alert, such as IP addresses, protocol, service, etc. This field is later used by the alert correlator to correlate meta-alerts. Timestamp field simply contains a pair of timestamps (start time, end time). They are the timestamps of the earliest and the latest alerts that have been aggregated to the meta-alert. Confidence score field in the meta-alert represents the confidence that the meta-

alert is an anomaly indeed [1]. If meta-alerts are matched to a node in our causal polytree to provide evidence, the confidence scores of the matched meta-alerts are used to calculate the strength of the evidence.

### B. Attack Template

As we mentioned in Section III, we utilize causal polytrees to reason about anomalies in SCADA networks. We call these special causal polytrees attack templates and use $AT$s to represent them. Attack templates are created by domain experts and stored in the attack template database. When an attack is launched, the triggered meta-alerts from EDMAND are matched to the corresponding attack template and the belief propagation is conducted on it. An example attack template for the data integrity attack is shown in Fig. 5. Each node $X$ in an attack template $AT$ is an attack step with zero, one, or multiple parents and children. Each parent represents a prior cause attack step that can lead to the current one and each child represents a posterior consequence attack step that the current one can lead to. If there are multiple parents, they follow either the "noisy-OR" or the "noisy-AND" model. The prior probability at each node, the probabilities $q_i$s of the inhibitory or enabling mechanisms in "noisy-OR" and "noisy-AND" models are all specified by domain experts (e.g. power grid/SCADA security administrator) when the attack template is created. Also, each attack template $AT$ contains one or more sink nodes (shaded node in Fig. 5). Denote the set of sink nodes as $\boldsymbol{S}_{AT}$. Nodes in $\boldsymbol{S}_{AT}$ represent the final targets of the entire attack and we call them consequence nodes. Each consequence node has domain knowledge associated with such as attack consequence, severity, and potential countermeasure.

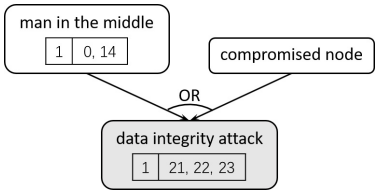


Fig. 5: Attack template of the data integrity attack

Each attack step (each node) has two binary states: not exist (0) and exist (1). However, the attack steps cannot be observed directly. We can only infer the existence of each attack step by the alerts it triggers in EDMAND. Each attack step could trigger meta-alerts that belong to multiple[1] alert types. Multiple meta-alerts can match to one alert type of an attack step and serve as evidence to each attack step node. We create a structure, called alert unit table, to store the matched meta-alerts at each attack step. An example of the alert unit table is shown in Table I. Each row in the table is an alert unit ($AU$), which represents one proportion of evidence. Let us assume there are $k$ alert units in the table. Each alert unit $AU_i$ consists of a weight $w_i$ and a list of alert types $A_{i1}, A_{i2}, \ldots, A_{in_i}$, where $n_i$ is the number of alert types in $AU_i$. Therefore, $AU_i = \{w_i, A_{i1}, A_{i2}, \ldots, A_{in_i}\}$. $w_i$ represents how much the observation of one or more of the following alert types $A_{i1}, A_{i2}, \ldots, A_{in_i}$ can prove the existence of the attack step and $\sum_i w_i = 1$. Alert types in the same alert unit express the same aspect of the attack step. Each alert type $A_{ij}$ in the alert unit table can contain multiple meta-alerts from EDMAND of the same corresponding alert type. For example, in the "data integrity attack" attack step in Figure 5, the alert unit table contains one alert unit $\{1, 21, 22, 23\}$. Since there is just one alert unit, its weight is 1. The three alert types are 21, 22, and 23, which represent `BINARY_FAULT`, `ANALOG_TOO_LARGE`, and `ANALOG_TOO_SMALL`. These three types of content-level meta-alerts all represent the actual tampering of the measurement data and are therefore included in the same alert unit.

TABLE I: Alert unit table for each attack step

| Alert Unit | Weight | Alert Types |
|---|---|---|
| $AU_1$ | $w_1$ | $A_{11}, A_{12}, \ldots, A_{1n_1}$ |
| $AU_2$ | $w_2$ | $A_{21}, A_{22}, \ldots, A_{2n_2}$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $AU_k$ | $w_k$ | $A_{k1}, A_{k2}, \ldots, A_{kn_k}$ |

When meta-alerts are matched to a node $X$ and stored in its alert unit table, we add a dummy auxiliary child node $\widetilde{X}$ to $X$ as shown in Fig. 6 and simulate the evidence from those meta-alerts by letting $\widetilde{X}$ provide a diagnostic support message $\lambda_{\widetilde{X}}(x)$ to $X$. The confidence scores of the matched meta-alerts are used to calculate $\lambda_{\widetilde{X}}(x)$. For each alert type $A_{ij}$ in the alert unit table, we assume there are $m_{ij}$ meta-alerts $a_{ij1}, a_{ij2}, \ldots, a_{ijm_{ij}}$ matched to it. The confidence scores of them are $CS(a_{ij1}), CS(a_{ij2}), \ldots, CS(a_{ijm_{ij}})$. Let $CS(A_{ij})$ be the confidence score of the alert type $A_{ij}$ and it is calculated as

$$CS(A_{ij}) = \begin{cases} \dfrac{\prod_{l=1}^{m_{ij}} CS(a_{ijl})}{\prod_{l=1}^{m_{ij}} CS(a_{ijl}) + \prod_{l=1}^{m_{ij}} (1 - CS(a_{ijl}))} & m_{ij} > 0, \\ P_{miss} & m_{ij} = 0 \end{cases} \quad (5)$$

where $P_{miss}$ is a probability of missing meta-alerts and can be predefined by experience or calculated if training data is available. After we have confidence score calculated for every alert type in one alert unit $AU_i$, we can write the confidence score of the alert unit $CS(AU_i)$ as $CS(AU_i) = \max_{j=1}^{n_i} CS(A_{ij})$. The final total confidence score of the attack step $CS_{total}$ is calculated by $CS_{total} = \sum_{i=1}^{k} w_i CS(AU_i)$. The diagnostic support $\lambda_{\widetilde{X}}(x)$ provided by all the matched alerts to the attack step $X$ is written as

$$\lambda_{\widetilde{X}}(x) = \begin{cases} 1 - CS_{total} & x = 0 \\ CS_{total} & x = 1 \end{cases}. \quad (6)$$

### C. Alert Correlator

CAPTAR's anomaly reasoning consists of meta-alert matching and belief propagation. Meta-alert matching is the process of matching meta-alerts to attack steps (in attack templates)

[1]It is also possible that one attack step triggers no alerts in EDMAND. In this case, we can only infer the existence of this attack step by the existence of its parents and children.

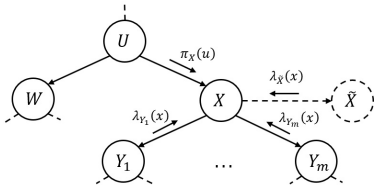


Fig. 6: Auxiliary child $\widetilde{X}$ of $X$ representing evidence received by $X$

that trigger them. And the most important step of alert matching is to decide whether two meta-alerts are correlated or not. Therefore, the alert correlator is designed for this purpose. The alert correlator is a naive Bayes classifier whose graphical representation is a Bayesian network in Fig. 7 with one root node $X$ and three leaf nodes $Y_1$, $Y_2$, and $Y_3$. The root node $X$ represents the hypothesis that "the two input meta-alerts are correlated" and has two states: "yes" (1) and "no" (0). Each leaf node $Y_j$ ($j \in \{1, 2, 3\}$) stands for one type of observable evidence that helps to evaluate the hypothesis and has several discrete states. Depending on whether two meta-alerts are correlated or not, the distribution of states at the evidence nodes will be different. Therefore, based on the observed states at the evidence nodes, one can infer the probability that two meta-alerts are correlated. We consider three kinds of observable evidence while correlating two meta-alerts: *time difference* ($Y_1$), *IP similarity* ($Y_2$), and whether they share the *same service* ($Y_3$). More details of the assigning of states for each leaf node

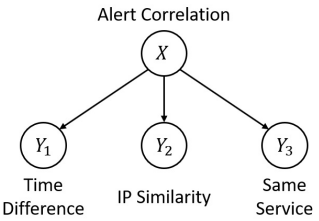


Fig. 7: Alert correlation model

Let $x$ ($x \in \{0, 1\}$) represent the state of the root node in Figure 7. Let $y_j$ ($j \in \{1, 2, 3\}$) represent the state at each leaf node $Y_j$ and $\widehat{y}_j$ represent the already observed state. There is a conditional probability table (CPT) at each leaf node $Y_j$ which relates $Y_j$ to $X$. Each entry $P(y_j|x)$ in the table defines the probability of state $y_j$ of node $Y_j$ given certain state $x$ of node $X$. The prior probability $P(x)$ varies depending on the alert types of the two input meta-alerts. There is a predefined prior probability for each pair of alert types based on domain knowledge. According to the believe propagation, the belief at root $X$ can be calculated as

$$BEL(x) = \alpha\pi(x)\prod_{j=1}^{3}\lambda_{Y_j}(x) = \alpha P(x)\prod_{j=1}^{3}P(\widehat{y}_j|x), \quad (7)$$

where $\alpha$ is a normalizing factor rendering $\sum_x BEL(x) = 1$. We say two meta-alerts are correlated if $BEL(1) > 0.5$ for $X$. Let $a$ and $b$ be the two input meta-alerts for the alert correlator. We define the CORRELATE procedure of the alert correlator as follows:

$$\text{CORRELATE}(a, b) = \begin{cases} BEL(1) & BEL(1) > 0.5 \\ -1 & \text{otherwise,} \end{cases} \quad (8)$$

### D. Causal Reasoning Engine

The causal reasoning engine is the core component of CAPTAR and it interacts with all other three components. When the causal reasoning engine starts, it fetches copies of attack templates $AT$s from the attack template database and creates an attack template set $ATS$. Then it runs an anomaly reasoning algorithm to perform alert matching and belief propagation on the attack templates in the attack template set.

The anomaly reasoning algorithm is shown in Algorithm 1. The ANALYZEALERT procedure in this algorithm is called whenever CAPTAR receives a new meta-alert or an update to an existing alert. The procedure takes the meta-alert $a$ and the current attack template set $\boldsymbol{ATS}$ in the causal reasoning engine as inputs. The output is a new attack template set $\boldsymbol{ATS}_{new}$ with the meta-alert $a$ matched to some of the attack templates inside and belief propagation performed. The procedure has two cases. If $a$ is an update to an existing meta-alert, then some attack templates in $\boldsymbol{ATS}$ might already have $a$ matched. For each $AT$ of those attack templates, the algorithm gets the node $X$ in $AT$ that $a$ is matched to. Since the meta-alert is updated, the procedure recalculates the total confidence score $CS_{total}$ of $X$. The diagnostic support $\lambda_{\widetilde{x}}(x)$ from all the matched alerts is also recalculated. Since the evidence contained at $X$ changes, a belief propagation in $AT$ from node $X$ is initiated. In this case, the $\boldsymbol{ATS}$ with the updated attack templates are directly assigned to $\boldsymbol{ATS}_{new}$ for output. If $a$ is a newly detected meta-alert, the algorithm iterates over the entire set $\boldsymbol{ATS}$. For each attack template $AT$ in $\boldsymbol{ATS}$, it uses the alert correlator to match the meta-alert $a$ to nodes in $AT$ and performs a belief propagation if there is a successful match. This process is included in the procedure called MATCHALERT. This procedure takes $a$ and $AT$ as inputs and outputs a set of attack templates $\boldsymbol{ATS}_{match}$. The attack templates in $\boldsymbol{ATS}_{match}$ are copies of $AT$ with $a$ matched and belief propagation performed. Since it is possible that $a$ can match to multiple nodes in $AT$, $\boldsymbol{ATS}_{match}$ could contain multiple copies. If $a$ cannot be matched to $AT$, $\boldsymbol{ATS}_{match}$ will just contain the original $AT$. After we get $\boldsymbol{ATS}_{match}$ from MATCHALERT$(a, AT)$, the attack templates in $\boldsymbol{ATS}_{match}$ are all added to $\boldsymbol{ATS}_{new}$. Due to the limit of space, more details of the MATCHALERT is not included here and can be found in [16].

For each attack step $X$ in an attack template $AT$, $BEL_X(1)$ represents the probability of existence of this attack step. Since consequence nodes in $\boldsymbol{S}_{AT}$ stand for final targets of the entire attack represented by $AT$, the maximum probability of existence of all consequence nodes in $AT$, denoted by $BEL_{max}(AT)$, can represent the inferred success possibility of the attack and is calculated as $BEL_{max}(AT) = \max_{X \in \boldsymbol{S}_{AT}} BEL_X(1)$. After each run of the algorithm, namely each call of procedureANALYZEALERT, the attack template set $\boldsymbol{ATS}$ in the causal reasoning engine is replaced by $\boldsymbol{ATS}_{new}$. The engine then checks $BEL_{max}(AT)$ of every attack template $AT$ in the new attack template set. If it finds

Algorithm 1: Anomaly Reasoning Algorithm

**Input:**
$a$ - meta-alert to be analyzed
$\boldsymbol{ATS}$ - attack template set
**Output:**
$\boldsymbol{ATS}_{new}$ - new attack template set
**procedure** ANALYZEALERT($a, \boldsymbol{ATS}$)
$\boldsymbol{ATS}_{new} \leftarrow \emptyset$
**if** $a$ is an update of an existing meta-alert **then**
**for** each $AT$ in $\boldsymbol{ATS}$ that has $a$ as a matched alert **do**
recalculate $CS_{total}$ and $\lambda_{\tilde{x}}(x)$ of the matched node $X$
start a new belief propagation in $AT$ from node $X$
**end for**
$\boldsymbol{ATS}_{new} \leftarrow \boldsymbol{ATS}$
**else**
**for** each $AT$ in $\boldsymbol{ATS}$ **do**
$\boldsymbol{ATS}_{match} \leftarrow$ MATCHALERT($a, AT$)
add $\boldsymbol{ATS}_{match}$ to $\boldsymbol{ATS}_{new}$
**end for**
**end if**
**return** $\boldsymbol{ATS}_{new}$
**end procedure**

$BEL_{max}(AT) > \theta_{BEL}$ for any $AT$, it will output that attack template $AT$ for operator's further analysis. Here $\theta_{BEL}$ is a predefined threshold and we use $\theta_{BEL} = 0.8$ for our CAPTAR prototype.

During the alert matching process, the MATCHALERT procedure will explore every potential match of $a$ and create multiple copies of the original attack template $AT$ if no exact match can be found. This will increase the number of attack templates in the attack template set $\boldsymbol{ATS}$. To prevent the number of attack templates from exploding, we set a maximum limit $K$ for the number of attack templates to keep for each kind of attack. Attack templates with lower $BEL_{max}(AT)$ will be dropped when the number exceeds the limit. Also, attack templates will also be dropped from the set if they have not been updated for a long time.

The attack templates, output by the causal reasoning engine, represent attacks of high probability of existence in the SCADA network. The operators can not only understand the origin of the attacks by examining the belief of each attack step and the corresponding alerts, but also evaluate the attack consequences and take countermeasures by utilizing the domain knowledge contained in the consequence nodes.

## V. PERFORMANCE EVALUATION

In this section, we evaluate the anomaly reasoning ability of CAPTAR via three simulated attack scenarios. We implement a prototype of CAPTAR and reuse our prototype of EDMAND. The baseline traffic is 14 days of simulated DNP3 traffic of one control center communicating with 10 remote terminal units (RTUs). We create three attack templates representing three common attacks in SCADA networks: TCP SYN flood, data integrity attack, and command injection.

- *TCP SYN flood*: The attack template for TCP SYN flood is shown in Fig. 8. The attacker starts by an IP address scan to find out the active IP addresses in the subnet. Then the TCP SYN flood is conducted by sending a succession of SYN requests to the target with spoofed source addresses.

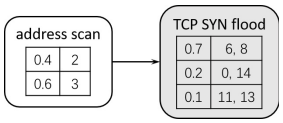


Fig. 8: Attack template of the TCP SYN flood

- *Data integrity attack*: The attack template for data integrity attack is shown in Fig. 5. The attacker first either launches a man-in-the-middle attack or compromises some field devices. The measurement data sent back to the control center are then tampered to mislead the control system.
- *Command injection*: The attack template for command injection is shown in Fig. 9. The attacker first either launches a man-in-the-middle attack or conducts an IP address scan followed by a service scan. Malicious control commands are then injected into the packets to attack the substations.

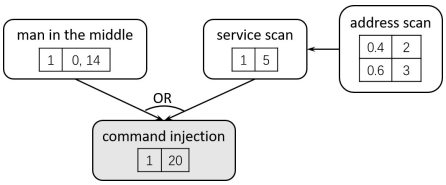


Fig. 9: Attack template of the command injection

In our evaluation, we launch the above three attacks in our simulated SCADA network. CAPTAR together with EDMAND are able to identify and differentiate all three attacks. Moreover, the output of CAPTAR gives the operator a better idea of the likelihood of each attack step even if there is no direct alert representation of the step. For example, the attack step of "compromised node" in the data integrity attack has no detectable alert by EDMAND (for now). However, CAPTAR can still infer the high chance of existence of a compromised node if it sees the existence of the "data integrity attack" consequence node and the absence of the "man in the middle" node. Notice that the expressiveness of attack templates can be improved by increasing the number of meta-alert types that can be triggered by EDMAND. CAPTAR can also reason about alerts not from EDMAND as long as they are preprocessed to follow the same format.

Let us assume $M$ to be the number of meta-alerts in the database, $N$ to be the maximum number of nodes in any attack template, $L$ to be the number of attack templates in the database, and $K$ to be the maximum limit for the number

of attack templates to keep in $\boldsymbol{ATS}$ for each kind of attack. It can be derived that the time complexity of the algorithm is $O(KLN(M+N))$ in the worst case. Usually, we have $M \gg N$, so the anomaly reasoning algorithm has an estimated time complexity of $O(KLMN)$ in the worst case. $K$ and $N$ are usually less than 10. $L$ should be several dozens. $M$ is also limited to dozens or hundreds due to the alert aggregation and removing of stale meta-alerts from the database. Therefore, the total time complexity of the algorithm is pretty reasonable. And note that the frequency CAPTAR runs the anomaly reasoning algorithm is decided by the frequency that EDMAND sends meta-alerts which is limited [1]. Therefore, CAPTAR is able to satisfy the real-time anomaly reasoning need for those meta-alerts.

To give a better understanding of the time overhead of CAPTAR, we measure the average time to run the FINDCORRELATION procedure, the belief propagation, and the anomaly reasoning algorithm for the three attack scenarios on a Ubuntu 16.04 desktop with 12 Intel Xeon 3.60GHz CPUs and 16GB memory. The results are shown in Table II. We can see that the time overheads are definitely small enough to satisfy the real-time reasoning requirement of the meta-alerts. Note that the average time to run FINDCORRELATION and the anomaly reasoning algorithm varies a lot across different attack scenarios. This is because the time overheads of the procedure and the algorithm depend on the number of meta-alert $M$. And those three attack scenarios generate 104(TCP SYN flood), 7(data integrity attack), and 26(command injection) meta-alerts respectively. This results in the different time overheads of FINDCORRELATION and the anomaly reasoning algorithm for the three attacks.

TABLE II: Average time overhead for FINDCORRELATION, belief propagation, and the anomaly reasoning algorithm (T="TCP SYN flood" D="data integrity attack" C="command injection")

| Attack | FINDCORRELATION | Belief Propagation | Anomaly Reasoning |
|---|---|---|---|
| T | 7.60ms | 0.21ms | 41.39ms |
| D | 0.48ms | 0.10ms | 19.76ms |
| C | 2.65ms | 0.14ms | 13.95ms |

## VI. CONCLUSION

In this paper, we propose a causal-polytree-based anomaly reasoning framework for SCADA networks, named CAPTAR. CAPTAR takes the meta-alerts from EDMAND and performs alert correlation and attack plan recognition. Experiments using a prototype of CAPTAR and simulated traffic show that CAPTAR is able to detect and differentiate various attack scenarios in a real-time manner. The generated reasoning results can provide the operators with a high-level view of the security state of the protected SCADA network.


## ACKNOWLEDGMENT

This material is based upon work supported by the Department of Energy under Award Number DE-OE0000780.




## REFERENCES


[1] W. Ren, T. Yardley, and K. Nahrstedt, "EDMAND: Edge-Based Multi-Level Anomaly Detection for SCADA Networks," in *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. IEEE, 2018, pp. 1–7.

[2] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2001, pp. 54–68.

[3] F. Cuppens, "Managing alerts in a multi-intrusion detection environment," in *acsac*. IEEE, 2001, p. 0022.

[4] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, no. 1-2, pp. 105–136, 2002.

[5] A. Siraj and R. B. Vaughn, "Multi-level alert clustering for intrusion detection sensor data," in *NAFIPS 2005-2005 Annual Meeting of the North American Fuzzy Information Processing Society*. IEEE, 2005, pp. 748–753.

[6] S. Zhang, J. Li, X. Chen, and L. Fan, "Building network attack graph for alert causal correlation," *Computers & security*, vol. 27, no. 5-6, pp. 188–196, 2008.

[7] L. Briesemeister, S. Cheung, U. Lindqvist, and A. Valdes, "Detection, correlation, and visualization of attacks against critical infrastructure systems," in *2010 Eighth International Conference on Privacy, Security and Trust*. IEEE, 2010, pp. 15–22.

[8] Y. Zhai, P. Ning, P. Iyer, and D. S. Reeves, "Reasoning about complementary intrusion evidence," in *20th Annual Computer Security Applications Conference*. IEEE, 2004, pp. 39–48.

[9] Z. Zali, M. R. Hashemi, and H. Saidi, "Real-time attack scenario detection via intrusion detection alert correlation," in *2012 9th International ISC Conference on Information Security and Cryptology*. IEEE, 2012, pp. 95–102.

[10] A. Valdes and K. Skinner, "Adaptive, model-based monitoring for cyber attack detection," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2000, pp. 80–93.

[11] F. Xuewei, W. Dongxia, H. Minhuan, and S. Xiaoxia, "An approach of discovering causal knowledge for alert correlating based on data mining," in *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 2014, pp. 57–62.

[12] F. Kavousi and B. Akbari, "A Bayesian network-based approach for learning attack strategies from intrusion alerts," *Security and Communication Networks*, vol. 7, no. 5, pp. 833–853, 2014.

[13] A. A. Ramaki, M. Amini, and R. E. Atani, "RTECA: Real time episode correlation algorithm for multi-step attack scenarios detection," *computers & security*, vol. 49, pp. 206–219, 2015.

[14] X. Qin, "A probabilistic-based framework for infosec alert correlation," Ph.D. dissertation, Georgia Institute of Technology, 2005.

[15] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[16] W. Ren, T. Yardley, and K. Nahrstedt. (2019, April) Causal reasoning about attacks in SCADA networks. [Online]. Available: http://hdl.handle.net/2142/103425