Causal Reasoning about Attacks in SCADA Networks

Wenyu Ren*, Timothy Yardley* and Klara Nahrstedt*
* University of Illinois Urbana-Champaign, Urbana, Illinois, USA
Email: {wren3, 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.**

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.

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 described our 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 report, 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 Figure 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 is able to reveal the attack scenarios from the alerts and produces a high-level view of the security state of the protected SCADA network.

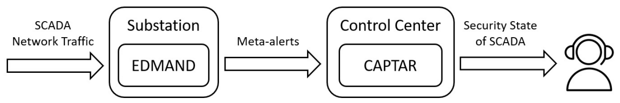


Fig. 1: Locations of EDMAND and CAPTAR

The remainder of this report is organized as follows: Section II reviews the related work. Section III introduces the basic concept of Bayesian network, Bayesian inference, and belief propagation. These concepts are utilized in the anomaly reasoning in this report. Two canonical models which are used to build our causal polytrees are also introduced in Section III. Section IV gives an overview of the design of CAPTAR. Section V shows the performance evaluation of CAPTAR and Section VI concludes the report. Since this report uses many difference notations, some of the most important notations are listed in Table I for quick reference.

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 of alert correlation, 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

Notation	Description
X	A node in the Bayesian network representing a random variable.
U	A parent of X in the Bayesian network.
Y	A child of X in the Bayesian network.
$\boldsymbol{e}_X^+$	Evidence contained in the sub-tree rooted at X.
$\boldsymbol{e}_X^-$	Evidence contained in the rest of the Bayesian network other than $\boldsymbol{e}_X^+$.
$\pi_X(u)$	Causal support provided by parent U to X.
$\lambda_Y(x)$	Diagnostic support provided by child Y to X.
$BEL(x)$	Belief at node X.
$\widetilde{X}$	Auxiliary child node of X to simulate evidence of matched meta-alerts.
$\lambda_{\widetilde{X}}(x)$	Diagnostic support provided by $\widetilde{X}$ to X.
I	Inhibitory mechanism in "noisy-OR" model.
E	Enabling mechanism in "noisy-AND" model.
q	Probability that the inhibitory or enabling mechanism is active.
$CS(a)$	Confidence score of meta-alert a.
AT	Attack template.
$\boldsymbol{ATS}$	Attack template set.
$\boldsymbol{S}_{AT}$	Set of consequence nodes of attack template AT.
AU	Alert unit.
w	Weight in the alert unit.
A	Alert type in the alert unit.
CS_{total}	Total confidence score of all matched meta-alerts of a node.
$BEL_{max}(AT)$	Maximum probability of existence of all consequence nodes in AT.
Cor_{max}	Maximum correlation score of a meta-alert in an attack template.
$\boldsymbol{ATS}_{match}$	Attack template set containing alert matching results.
X_{cor}	Exact match node with the highest correlation score for a meta-alert.
$\boldsymbol{X}_{pot}$	Set of potential nodes a meta-alert could match to.
K	Maximum number of attack templates to keep for each kind of attack.
M	Number of meta-alerts in the meta-alert database.
N	Maximum number of nodes in any attack template.
L	Number of attack templates in the attack template database.

TABLE I: Table of notation

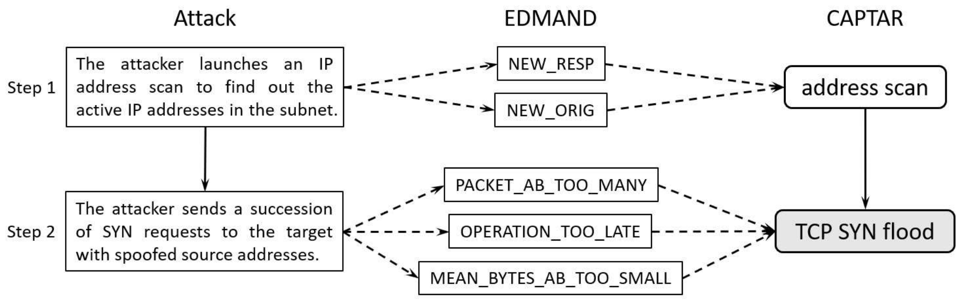
evolves towards a "worse" direction during attacks. There are also works [8], [9] that define prerequisites and consequences of each attack step and construct chains or graphs based on the matching of prerequisites and consequences. Bayesian networks are also utilized by many papers [8], [10]–[13] to 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

An example workflow of EDMAND and CAPTAR is shown in Figure 2. After an attacker launches an attack, each step of the attack could result in one or more anomalies in the network traffic. These anomalies trigger meta-alerts in EDMAND. CAPTAR receives the meta-alerts from EDMAND and tries to infer the attack steps that triggered them by mapping meta-alerts to attack steps. Potential attack steps are structured as nodes in causal polytrees whose nature are Bayesian networks. Bayesian inference is performed on those causal polytrees to reason about the existence of attacks. In this section, we first



Attack
EDMAND
CAPTAR
Step 1
The attacker launches an IP address scan to find out the active IP addresses in the subnet.
NEW_RESP
NEW_ORIG
address scan
Step 2
The attacker sends a succession of SYN requests to the target with spoofed source addresses.
PACKET_AB_TOO_MANY
OPERATION_TOO_LATE
MEAN_BYTES_AB_TOO_SMALL
TCP SYN flood

Fig. 2: An example workflow of EDMAND and CAPTAR

introduce the basic concept of Bayesian network. Then we describe inference in Bayesian network followed by the belief propagation algorithm to conduct Bayesian inference. Finally, we present two canonical models we use to build our causal trees: "noisy-OR" and"noisy-AND" models.

A. Bayesian Network

Before going into exactly what a Bayesian network is, it is first useful to review two concepts in probability theory. The first concept is the chain rule of probability. It says that if we have a set of n random variables, $X_1, X_2, \ldots, X_n$, then the joint probability distribution $P(X_1, X_2, \ldots, X_n)$ can be written as a product of n conditional probabilities:

$$\begin{aligned} &P(X_1, X_2, \ldots, X_n) = \\ &P(X_n|X_{n-1}, \ldots, X_2, X_1) \cdots P(X_2|X_1)P(X_1). \end{aligned} \tag{1}$$

The second concept is the conditional independence. We say that two random variables, A and B, are conditionally independent given another random variable C if $P(A|B,C) = P(A|C)$. In other words, once we know C, learning B would not change our belief in A.

After recalling the chain rule of probability and the conditional independence, we can introduce the basics of Bayesian network. A Bayesian network is a directed acyclic graph (DAG) in which the nodes represent variables, the edges signify the existence of direct causal influences between the linked variables, and the strengths of these influences are expressed by conditional probabilities. Figure 3 illustrates a simple yet typical Bayesian network. It describes relationships among the seasons of the year (X_1), whether rain falls (X_2), whether the sprinkler is on (X_3), whether the pavement would get wet (X_4), and whether the pavement would be slippery (X_5). All variables in this figure are binary (taking a value of either true or false) except for the root variable X_1, which can take one of four values: spring, summer, fall, or winter.

Each edge in the figure represents a direct causal influence from the head of the edge to the tail. In Figure 3, X_4 has a directed edge pointing to X_5. This is because the fact that the pavement is wet has a direct causal influence on whether the pavement is slippery. On the contrary, the absence of a direct edge between two nodes implies conditional independence. For example, the absence of a direct edge between X_1 and X_5 captures the understanding that the influence of seasonal

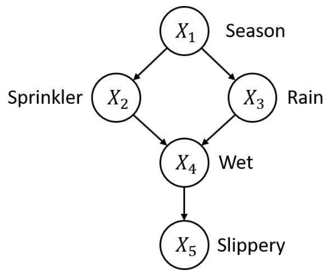


Fig. 3: Bayesian network example

variations on the slipperiness of the pavement is mediated by other conditions (e.g., the wetness of the pavement).

Each node in the Bayesian network is associated with a probability function that takes (as input) a particular set of values for the node's parent variables, and gives (as output) the probability of the variable represented by the node. The most common form of this probability function is a conditional probability table (CPT). CPT is a table defined for a set of discrete and mutually dependent random variables to display conditional probabilities of a single variable with respect to the others. An example CPT of X_4 in Figure 3 is shown in Table II. It gives the conditional probabilities of $P(X_4|X_2, X_3)$.

X_2	X_3	$X_4 = $ T	$X_4 = $ F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

TABLE II: Conditional probability table of X_4

An important property of Bayesian networks is the local (parental) Markov condition, which states that every node in a Bayesian network is conditionally independent of all its non-descendants given its parent. In the above example, we have $P(X_5|X_1, X_2, X_3, X_4) = P(X_5|X_4)$ since Slippery is conditionally independent of its non-descendants, Season, Sprinkler, and Rain, given its parent Wet. This property allows us to simplify the joint distribution, obtained using the chain rule, to a simpler form. Assume a Bayesian network has n nodes $X_1, \ldots, X_n$ in total. The joint distribution can be simplified as

$$P(X_1,\ldots,X_n) = \prod_{i=1}^{n} P(X_i|X_1,\ldots,X_{i-1}) = \prod_{i=1}^{n} P(X_i|Parents(X_i)), \tag{2}$$

where $Parents(X_i)$ is the set of direct parents of X_i. In the above example, we are able to rewrite the joint distribution as

$$P(X_1,X_2,X_3,X_4,X_5) = P(X_1)P(X_2|X_1)P(X_3|X_1)P(X_4|X_2,X_3)P(X_5|X_4). \tag{3}$$

This property significantly reduces the amount of required computation in large Bayesian networks since each node usually has fewer parents compared with the overall size of the network.

B. Bayesian Inference

There are two kinds of inference over a Bayesian network. The first is to evaluate the joint probability of a specific assignment of values for all or a subset of variables in the network. For all variables, we simply factorize the joint probability using Equation 2 and calculate the product using provided conditional probabilities. For a subset of variables, we marginalize over the variables not in the subset by summing up probabilities over them and get the marginal probability of the subset of variables we are interested in.

The second and more interesting inference is to evaluate $P(\boldsymbol{x}|\boldsymbol{e})$, that is, the probability of some particular assignment of a subset of variables ($\boldsymbol{x}$) given assignments of other variables (evidence $\boldsymbol{e}$). In the scenario we mentioned in Section III-A, one example of this kind of inference could be to evaluate $P(X_2 = T, X_4 = T, X_5 = T|X_1 = spring)$. In this case, $\{X_2 = T, X_4 = T, X_5 = T\}$ is our $\boldsymbol{x}$ and $\{X_1 = spring\}$ is our $\boldsymbol{e}$. According to the definition of conditional probability, we have $P(\boldsymbol{x}|\boldsymbol{e}) = P(\boldsymbol{x},\boldsymbol{e})/P(\boldsymbol{e}) = \alpha P(\boldsymbol{x},\boldsymbol{e})$, where $\alpha = 1/P(\boldsymbol{e})$ is a normalizing constant rendering $\sum_{\boldsymbol{x}} P(\boldsymbol{x}|\boldsymbol{e}) = 1$. Let $\boldsymbol{Z}$ represent the set of variables in the network that is not in $\boldsymbol{x}$ and $\boldsymbol{e}$, and $\boldsymbol{z}$ represents any particular value assignment of $\boldsymbol{Z}$. To get $P(\boldsymbol{x},\boldsymbol{e})$, the marginal probability of $\{\boldsymbol{x},\boldsymbol{e}\}$ over $\boldsymbol{Z}$ needs to be calculated. Therefore, we have

$$P(\boldsymbol{x}|\boldsymbol{e}) = \alpha \sum_{\forall \boldsymbol{z}\in\boldsymbol{Z}} P(\boldsymbol{x},\boldsymbol{e},\boldsymbol{z}). \tag{4}$$

In the example, we can calculate $P(X_2 = T, X_4 = T, X_5 = T|X_1 = spring)$ as

$$\begin{aligned}
&P(X_2 = T, X_4 = T, X_5 = T|X_1 = spring)\\
&= \alpha \sum_{X_3} P(X_1 = spring)P(X_2 = T|X_1 = spring)\\
&\quad P(X_3|X_1 = spring)P(X_4 = T|X_2 = T, X_3)\\
&\quad P(X_5 = T|X_4 = T)\\
&= \alpha P(X_1 = spring)P(X_2 = T|X_1 = spring)\\
&\quad P(X_3 = T|X_1 = spring)P(X_4 = T|X_2 = T, X_3 = T)\\
&\quad P(X_5 = T|X_4 = T) + \alpha P(X_1 = spring)\\
&\quad P(X_2 = T|X_1 = spring)P(X_3 = F|X_1 = spring)\\
&\quad P(X_4 = T|X_2 = T, X_3 = F)P(X_5 = T|X_4 = T)
\end{aligned} \tag{5}$$

C. 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.

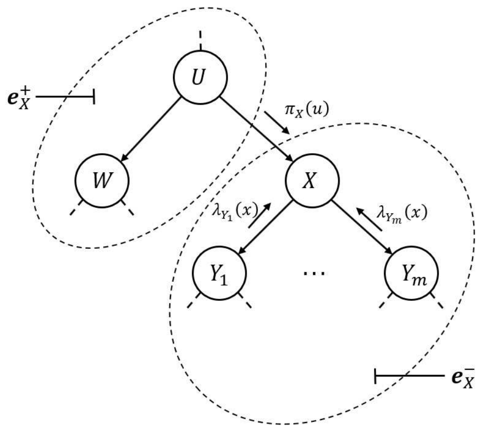


Fig. 4: Fragment of a causal tree, showing different kinds of evidence and support of a node X

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 Figure 4. 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: $\lambda_{Y_j}(x) = P(e^-_{Y_j}|x)$, 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.

We utilize the tree-structured Bayesian network for our anomaly reasoning. Each node represents an attack step in the entire attack plan and it has two states of exist (1) and not exist (0). A direct edge from node U to node X means that attack step U is a direct prior step of attack step X and needs to be launched before X. In this way, we are able to reason about the probability of existence of the attack by calculating the belief of each attack step.

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 $\lambda_{Y_1}(x)$, $\lambda_{Y_2}(x)$, $\ldots$, $\lambda_{Y_m}(x)$ from each of its children as shown in Figure 4.

$$BEL(x) = \alpha\lambda(x)\pi(x), \tag{6}$$

where

$$\lambda(x) = \prod_j \lambda_{Y_j}(x), \tag{7}$$

$$\pi(x) = \sum_u P(x|u)\pi_X(u), \tag{8}$$

and α is a normalizing constant rendering $\sum_x BEL(x) = 1$.
Step 2 — Bottom-up propagation: As shown in Figure 6, node X computes a new message $\lambda_X(u)$ based on its CPT and λ messages received from its children. Then X sends $\lambda_X(u)$ to its parent U.

$$\lambda_X(u) = \sum_x \lambda(x)P(x|u), \tag{9}$$

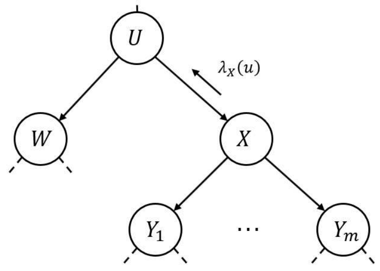


Fig. 5: Bottom-up propagation

Step 3 — Top-down propagation: As shown in Figure 5, node X computes new π messages and sends them to its children. The new $\pi_{Y_j}(x)$ message for its j-th child Y_j is calculated as

$$\pi_{Y_j}(x) = \alpha\pi(x)\prod_{k\neq j}\lambda_{Y_k}(x). \tag{10}$$

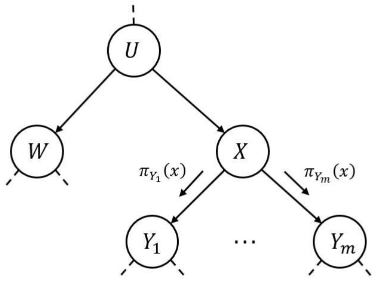


Fig. 6: Top-down propagation

Boundary conditions are established as follows:
1) *Root nodes:* If X is a node with no parents, we set $\pi(x)$ equal to the prior probability $P(x)$.
2) *Anticipatory nodes:* If X is a childless node that has not been instantiated, we set $\lambda(x) = 1$ for $x \in \{0, 1\}$
3) *Evidence nodes:* In our anomaly reasoning, evidence for a node X is obtained when meta-alerts are matched to the node. We will discuss the matching mechanism in Section IV-D. When evidence is obtained for X, we add a dummy auxiliary child node $\widetilde{X}$ to X as shown in Figure 7 and simulate the evidence by letting $\widetilde{X}$ provide a diagnostic support message $\lambda_{\widetilde{X}}(x)$ to X. We will describe our way to calculate $\lambda_{\widetilde{X}}(x)$ in Section IV-B. This auxiliary node $\widetilde{X}$ is not updated during the belief propagation using the 3 steps mentioned. It only changes the way X calculates its own $\lambda(x)$. Therefore, if evidence of X is obtained, Equation 7 needs to be rewritten as

$$\lambda(x) = \lambda_{\widetilde{X}}(x)\prod_j \lambda_{Y_j}(x). \tag{11}$$

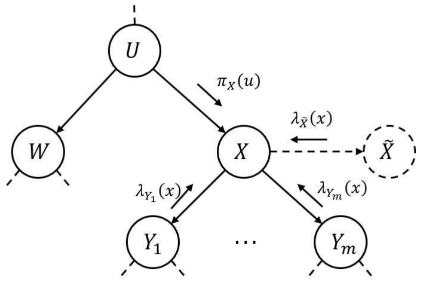


Fig. 7: Auxiliary child $\widetilde{X}$ of X representing evidence received by X

D. The "noisy-OR" and"noisy-AND" Models

In Section III-C, 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. A polytree is a directed acyclic graph whose underlying undirected graph is a tree. An example polytree is shown in Figure 8. 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.

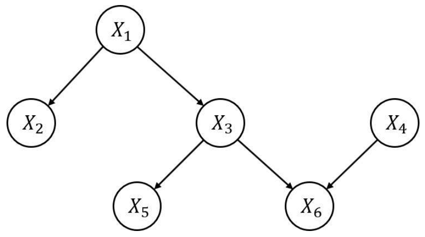


Fig. 8: Polytree example

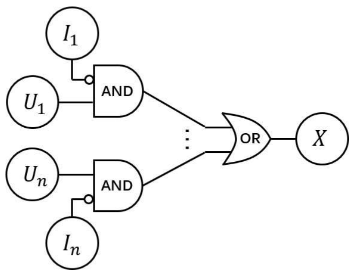


Fig. 9: The noisy OR-gate

1) The "noisy-OR" Model: The "noisy-OR" model [15] is based on the noisy OR-gate structure shown in Figure 9. Each node represents an event (attack step in our anomaly reasoning) with binary state 0 or 1. For a node X with n parents $\boldsymbol{U} = \{U_1, U_2, \dots, U_n\}$, 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 $I_1, \dots, I_n$ 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,

$$P(X = 1|U_i = 1, U_k = 0 \; k \neq i) = 1 - q_i. \tag{12}$$

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

$$\boldsymbol{u} = (u_1, u_2, \dots, u_n) \quad u_i \in \{0, 1\} \tag{13}$$

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

$$P(x|\boldsymbol{u}) = \begin{cases} \prod_{i \in T_u} q_i & \text{if} \quad x = 0 \\ 1 - \prod_{i \in T_u} q_i & \text{if} \quad x = 1. \end{cases} \tag{14}$$

Having the link matrix $P(x|\boldsymbol{u})$, we can follow similar belief propagation algorithm described in Section III-C. Assume X has m children, Y_1, Y_2, ..., Y_m. As it is shown in Figure 10, the local belief updating at X can be also executed by three steps in any order.

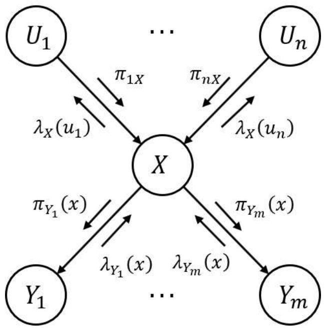


Fig. 10: Belief propagation in causal polytree

Belief Propagation Algorithm in "Noisy-OR" Model

Step 1 — Belief updating: Node X updates its belief measure based on the π_{1X}, ..., π_{nX} from its parents and the $\lambda_{Y_1}(x)$, ..., $\lambda_{Y_m}(x)$ from its children:

$$BEL(x) = \begin{cases} \alpha \lambda_0 \prod_i (1 - c_i \pi_{iX}) & \text{if} \quad x = 0 \\ \alpha \lambda_1 \left[1 - \prod_i (1 - c_i \pi_{iX})\right] & \text{if} \quad x = 1, \end{cases} \tag{15}$$

where

$$\lambda(x) = \prod_j \lambda_{Y_j}(x) = \begin{cases} \lambda_0 & \text{if} \quad x = 0 \\ \lambda_1 & \text{if} \quad x = 1 \end{cases}, \tag{16}$$

$$\pi_{iX} = P(u_i = 1), \tag{17}$$

and α is a normalizing constant rendering $\sum_x BEL(x) = 1$.

Step 2 — Bottom-up propagation: Node X computes new $\lambda_X(u_i)$ messages and sends them to its parents $\boldsymbol{U}$. The new $\lambda_X(u_i)$ message for its i-th parent U_i is calculates as

$$\lambda_X(u_i) = \begin{cases} \beta \left[\lambda_0 q_i \Pi_i' + \lambda_1 (1 - q_i \Pi_i')\right] & \text{if} \quad u_i = 1 \\ \beta \left[\lambda_0 \Pi_i' + \lambda_1 (1 - \Pi_i')\right] & \text{if} \quad u_i = 0, \end{cases} \tag{18}$$

where

$$\Pi_i' = \prod_{k \neq i} (1 - c_k \pi_{kX}), \tag{19}$$

and β is a normalizing constant.

Step 3 — Top-down propagation: Node X computes new $\pi_{Y_j}(x)$ messages and sends them to its children. The new

$\pi_{Y_j}(x)$ message for its j-th child Y_j is calculated as

$$\pi_{Y_j}(x) = \alpha \frac{BEL(x)}{\lambda_{Y_j}(x)}. \tag{20}$$

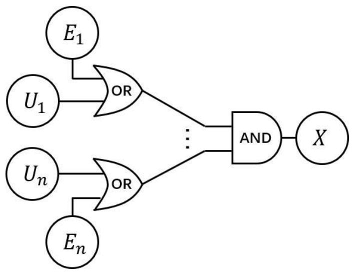


Fig. 11: The noisy AND-gate

2) The "noisy-AND" Model: The "noisy-AND" model [15] is based on the noisy AND-gate structure shown in Figure 11. The value of a node X with n parents $\boldsymbol{U} = \{U_1, U_2, \ldots, U_n\}$ can be seen as the output of a logical AND-gate. Each input to the AND-gate is the output of an OR-gate representing the conjunction of U_i and the its specific enabling mechanism E_i. We use q_i to represent the probability that the i-th enabler is active. Assume all inputs are 1 except $U_i = 0$. X will be 0 if and only if the enabler E_i associated with U_i remains inactive. That is,

$$P(X = 1 | U_i = 0, U_k = 1 \; k \neq i) = q_i. \tag{21}$$

Let $c_i = 1 - q_i$ and use $F_u = \{i : U_i = 0\}$ to represent the subset of parents that are 0. The link matrix $P(x|\boldsymbol{u})$ can be written as

$$P(x|\boldsymbol{u}) = \begin{cases} 1 - \prod_{i \in F_u} q_i & \text{if} \quad x = 0 \\ \prod_{i \in F_u} q_i & \text{if} \quad x = 1. \end{cases} \tag{22}$$

Assume X has m children, Y_1, Y_2, ..., Y_m. The three steps of local belief updating at X are listed as follows.

Belief Propagation Algorithm in "Noisy-AND" Model

Step 1 — Belief updating: Node X updates its belief measure based on the π_{1X}, ..., π_{nX} from its parents and the $\lambda_{Y_1}(x)$, ..., $\lambda_{Y_m}(x)$ from its children:

$$BEL(x) = \begin{cases} \alpha\lambda_0 \Big\{ 1 - \prod_i \big[1 - c_i(1 - \pi_{iX})\big] \Big\} & \text{if} \quad x = 0 \\ \alpha\lambda_1 \prod_i \big[1 - c_i(1 - \pi_{iX})\big] & \text{if} \quad x = 1, \end{cases} \tag{23}$$

where

$$\lambda(x) = \prod_j \lambda_{Y_j}(x) = \begin{cases} \lambda_0 & \text{if} \quad x = 0 \\ \lambda_1 & \text{if} \quad x = 1 \end{cases}, \tag{24}$$

$$\pi_{iX} = P(u_i = 1), \tag{25}$$

and α is a normalizing constant rendering $\sum_x BEL(x) = 1$.
Step 2 — Bottom-up propagation: Node X computes new $\lambda_X(u_i)$ messages and sends them to its parents $\boldsymbol{U}$. The new $\lambda_X(u_i)$ message for its i-th parent U_i is calculates as

$$\lambda_X(u_i) = \begin{cases} \beta\big[\lambda_0(1 - \Pi'_i) + \lambda_1 \Pi'_i\big] & \text{if} \quad u_i = 1 \\ \beta\big[\lambda_0(1 - q_i\Pi'_i) + \lambda_1 q_i \Pi'_i\big] & \text{if} \quad u_i = 0, \end{cases} \tag{26}$$

where

$$\Pi'_i = \prod_{k \neq i} \big[1 - c_k(1 - \pi_{kX})\big], \tag{27}$$

and β is a normalizing constant.
Step 3 — Top-down propagation: Node X computes new π messages and send them to its children. The new $\pi_{Y_j}(x)$ message for its j-th child Y_j is calculates as

$$\pi_{Y_j}(x) = \alpha \frac{BEL(x)}{\lambda_{Y_j}(x)}. \tag{28}$$

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 Figure 12. CAPTAR consists of 4 components: (1)*Meta-alert Database*, (2)*Attack Template Database*, (3)*Alert Correlator*, (4)*Causal Reasoning Engine*.

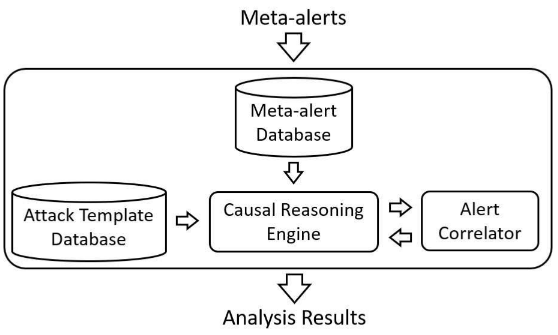


Fig. 12: CAPTAR architecture

The meta-alert database is used to store the meta-alerts from EDMAND. These meta-alerts serve as evidence to our causal reasoning of anomalies. The attack template database stores the potential attack templates which are causal poly-trees created by domain experts. These attack templates are Bayesian networks mentioned in Section III-A and contain the "noisy-OR" and "noisy-AND" models mentioned in Section III-D. They represent the prior domain knowledge we have on potential attack plans and are used as the underlying Bayesian networks for the belief propagation mentioned in Section III-C. One important step to reason about the anomalies is to match meta-alerts to nodes (attack steps) in our attack templates. And to do that, we need to evaluate whether one meta-alert is correlated with other meta-alerts. 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 [1] and sent to the control center where CAPTAR resides. Each meta-alert is the aggregation of similar alerts. Each meta-alert has several fields which are listed in Table III. 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 (implemented by MongoDB). 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. A complete list of the alert types is shown in Table IV. 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. It is the maximum of the confidence scores of all the aggregated alerts for this meta-alert. As we mentioned in [1], the confidence score (CS) for alert is calculated by $CS = MA \times AS$, where MA is the model accuracy and AS is the anomaly score. As stated in Section III-C, if meta-alerts are matched to a node X in our causal polytree, an auxiliary child node $\widetilde{X}$ is added to X. The confidence scores of the matched meta-alerts are used to calculate the diagnostic support message $\lambda_{\widetilde{X}}(x)$ that $\widetilde{X}$ provides to X. The way to calculate $\lambda_{\widetilde{X}}(x)$ will be introduced in Section IV-B.

Alert Field	Description
alert id	a unique id for retrieving a meta-alert from the database
alert type	a name that describes the meta-alert (see Table IV)
index field	a set of auxiliary information that helps to describe the meta-alert
timestamp	a start time and an end time for the meta-alert
confidence score	the confidence that the meta-alert represents an anomaly
statistical fields	more detailed information about the last alert aggregated
anomaly data	
count	number of alerts aggregated by the meta-alert

TABLE III: Meta-alert fields and description

Index	Alert Type
0	PACKET_IAT
1	PACKET_BYTES
2	NEW_ORIG
3	NEW_RESP
4	NEW_PROTOCOL
5	NEW_SERVICE
6	PACKET_AB_TOO_MANY
7	PACKET_AB_TOO_FEW
8	PACKET_BA_TOO_MANY
9	PACKET_BA_TOO_FEW
10	MEAN_BYTES_AB_TOO_LARGE
11	MEAN_BYTES_AB_TOO_SMALL
12	MEAN_BYTES_BA_TOO_LARGE
13	MEAN_BYTES_BA_TOO_SMALL
14	OPERATION_TOO_LATE
15	OPERATION_TOO_EARLY
16	OPERATION_MISSING
17	INVALID_FUNCTION_CODE
18	RESPONSE_FROM_ORIG
19	REQUEST_FROM_RESP
20	NEW_OPERATION
21	BINARY_FAULT
22	ANALOG_TOO_LARGE
23	ANALOG_TOO_SMALL

TABLE IV: Alert type

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 ATs to represent them. Attack templates represent and store the prior domain knowledge we have for attacks. When an attack is launched, the triggered meta-alerts from EDMAND are matched to the corresponding attack template and the belief propagation mentioned in Section III-C is conducted on it. An example attack template for the data integrity attack is shown in Figure 13. 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 in Section III-D. The prior probability at each node, the probabilities q_is 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 Figure 13). 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.

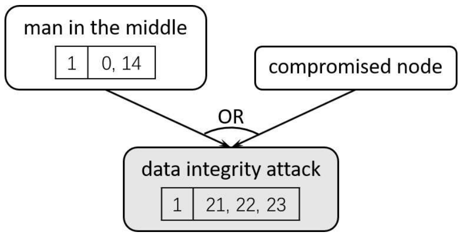


Fig. 13: An example of an attack template (causal polytree) using the "noisy-OR" model

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 mentioned in Section IV-A. Multiple meta-alerts can match to one alert type of an attack step. As we mentioned in Section III-C, these alerts are treated 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 V. 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, $AU_i = \{w_i, A_{i1}, A_{i2}, \ldots, A_{in_i}\}$. As we mentioned in Section IV-A, our current prototype of EDMAND can generate 24 types of alerts. $A_{i1}, A_{i2}, \ldots, A_{in_i}$ are represented by the alert type indexes in Table IV for simplicity reason. 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 13, 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.

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	$A_{k1}, A_{k2}, \ldots, A_{kn_k}$

TABLE V: Alert unit table for each attack step

As we mentioned in Section III-C, if meta-alerts are matched to a node X and stored in its alert unit table, an auxiliary child node $\widetilde{X}$ is added to X. The confidence scores of the matched meta-alerts are used to calculate the diagnostic support message $\lambda_{\widetilde{X}}(x)$ that $\widetilde{X}$ provides to X. To calculate $\lambda_{\widetilde{X}}(x)$, we utilize the confidence score of each meta-alert mentioned in Section IV-A. 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 matching mechanism will be described in Section IV-D). 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

$$CS(A_{ij}) = \begin{cases} \dfrac{\prod_{l=1}^{m_{ij}} CS(a_{ijl})}{\prod_{l=1}^{m_{ij}} CS(a_{ijl}) + \prod_{l=1}^{m_{ij}} (1 - CS(a_{ijl}))} & m_{ij} > 0 \\ P_{miss} & m_{ij} = 0, \end{cases} \tag{29}$$

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

$$CS(AU_i) = \max_{j=1}^{n_i} CS(A_{ij}). \tag{30}$$

The final total confidence score of the attack step CS_{total} is calculated by

$$CS_{total} = \sum_{i=1}^{k} w_i CS(AU_i). \tag{31}$$

The diagnostic support $\lambda_{\widetilde{X}}(x)$ provided by all the matched alerts to the attack step X is written as

$$\lambda_{\widetilde{X}}(x) = \begin{cases} 1 - CS_{total} & \text{if} \quad x = 0 \\ CS_{total} & \text{if} \quad x = 1 \end{cases}. \tag{32}$$

Attack templates are created by domain experts and stored in the attack template database before we start the anomaly reasoning. At the beginning of the reasoning, the causal reasoning engine will fetch copies of the original attack templates and create an attack template set $\boldsymbol{ATS}$. Then the engine

[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.

conducts alert matching as well as the belief propagation mentioned in Section III-C on them. Each attack template AT in $\boldsymbol{ATS}$ originates from one attack template in the database. And multiple attack templates in $\boldsymbol{ATS}$ could correspond to the same attack (same attack template in the database). For each attack step X in an attack template AT, $BEL_X(1)$ represents the probability of existence of this attack step. The way to calculate $BEL_X(1)$ is introduced in Section III. 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

$$BEL_{max}(AT) = \max_{X \in \boldsymbol{S}_{AT}} BEL_X(1). \tag{33}$$

C. Alert Correlator

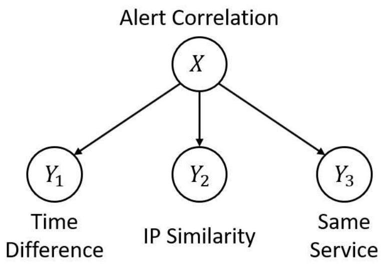


Fig. 14: Alert correlation model

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) 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 Figure 14 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).

- *Time difference:* The state of Y_1 depends on the closeness in the time axis of the two meta-alerts and we use T_{diff} to represent that. As we described in Section IV-A, each meta-alert from EDMAND has a start time and an end time. If the two meta-alerts overlap, we assign 0 to T_{diff}. Otherwise, we calculate T_{diff} as the difference between the end time of the earlier meta-alert and the start time of the latter one. Y_1 has four corresponding states according to T_{diff}:

$$Y_1 = \begin{cases} 0 & \text{if} \quad T_{diff} \leq 60\text{seconds} \\ 1 & \text{if} \quad 60\text{seconds} < T_{diff} \leq 1\text{hour} \\ 2 & \text{if} \quad 1\text{hour} < T_{diff} \leq 1\text{day} \\ 3 & \text{if} \quad T_{diff} > 1\text{day} \end{cases}. \tag{34}$$

- *IP similarity:* The state of Y_2 depends on the similarity of IP addresses related to the two meta-alerts. Each meta-alert could have one or two related IP addresses. Content-level alerts have one measure source IP while transport and operation level alerts have two IPs for originator and responder. For every pair of IP addresses (IP_a, IP_b), where IP_a relates to one input meta-alert and IP_b relates to the other, we calculate the similarity of them as follows:

$$SIM(IP_a, IP_b) = \begin{cases} 3 & IP_a \text{ and } IP_b \text{ are exactly the same} \\ 2 & \begin{array}{l} IP_a \text{ and } IP_b \text{ are not the same but} \\ \text{within the same 8-bit block} \end{array} \\ 1 & \begin{array}{l} IP_a \text{ and } IP_b \text{ are not within the same} \\ \text{8-bit block but within the same 16-bit block} \end{array} \\ 0 & \begin{array}{l} IP_a \text{ and } IP_b \text{ are not within the same} \\ \text{16-bit block} \end{array} \end{cases} \cdot \tag{35}$$

The maximum similarity of all such IP pairs is selected as the state of Y_2. Therefore, Y_2 has four states of $\{0, 1, 2, 3\}$ and $Y_2 = \max_{(IP_a, IP_b)} SIM(IP_a, IP_b)$.

- *Same service:* Y_3 evaluates whether the two meta-alerts share the same service (i.e., the same industrial control protocol). There are two states of Y_3: "yes" (1) and "no" (0). A "no" is also specified if any of the input meta-alerts does not have a related service.

Let x ($x \in \{0, 1\}$) represent the state of the root node in Figure 14. 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. As we stated in Section III-C, 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. Since X is a root node with no parent, we set $pi(x)$ to be the prior probability $P(x)$ according to the boundary condition mentioned in Section III-C. $P(x)$ 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. And since the state of Y_j is already observed as $\widehat{y}_j$, we have

$$\lambda(y_j) = \begin{cases} 1 & \text{if} \quad y_j = \widehat{y}_j \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

According to the bottom-up propagation step in the belief propagation, the diagnostic support provided by Y_k to X is $\lambda_{Y_j}(x) = \sum_{y_j} \lambda(y_j) P(y_j|x) = P(\widehat{y}_j|x)$. Therefore, the belief at root X can be calculated as

$$BEL(x) = \alpha\pi(x)\prod_{j=1}^{3}\lambda_{Y_j}(x) = \alpha P(x)\prod_{j=1}^{3}P(\widehat{y}_j|x), \quad (37)$$

where α 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:

$$\text{CORRELATE}(a, b) = \begin{cases} BEL(1) & \text{if} \quad BEL(1) > 0.5 \\ -1 & \text{otherwise,} \end{cases} \quad (38)$$

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 ATs 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 meta-alerts used in the alert matching are retrieved from the meta-alert database and the alert correlator is also used to correlate meta-alerts during the matching process. The belief propagation is introduced in Section III.

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} (presented in Section IV-B) of X. The diagnostic support $\lambda_{\widehat{x}}(x)$ 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 matches 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 $\boldsymbol{ATS}_{new}$. 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 $BEL_{max}(AT)$ (defined in Section IV-B) of every attack template AT in the new attack template set. If it finds $BEL_{max}(AT) > \theta_{BEL}$ for any AT, it will output that attack template AT for operator's further analysis. Here θ_{BEL} is a predefined threshold and we use $\theta_{BEL} = 0.8$ for our CAPTAR prototype.

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 $\lambda_{\widehat{x}}(x)$ 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
```

Before we introduce more details of the MATCHALERT procedure, there are one concept and another procedure we need to describe first. The concept is called *happens before* and the procedure's name is FINDCORRELATION. *Happens before* is a relationship between two meta-alerts. We say meta-alert a *happens before* meta-alert b if the start time of a is at least T_{hb} earlier than the start time of b, where $T_{hb} = 10s$ is a predefined threshold. The procedure FINDCORRELATION is shown in Algorithm 2. It takes a meta-alert a and a node X in the attack template as inputs and outputs a correlation score Cor_{max}. The objective of this procedure is to find whether the given node, its parents and children have any matched alert that correlates with the given alert. The procedure does so by iterating through every matched alert b of X, parents of X and children of X. For each b, it calls the alert correlator and uses the CORRELATE procedure to correlate a and b. The maximum result from CORRELATE(a, b) is stored in Cor_{max}. If any correlation is found, Cor_{max} contains the highest correlation score. Otherwise, $Cor_{max} = 0$. There are two exceptions while correlating alerts from parents and children. For any matched alert b of X's parents, there is a conflict if *a happens before b*. a is to be matched to X and X's parents are attack steps that should lead to X. If there is an attack, the attack steps represented by X's parents should be launched before X. That means a could not *happens before b*. Therefore, a should not be matched to X and the procedure outputs -1 in this case. For any matched alert b of X's children, the procedure outputs -1 if *b happens before a* for similar reasons.

Algorithm 2 Find Correlation Procedure

```
Input:
    $a$ - meta-alert to find correlation with
    $X$ - a node in the attack template whose alert unit table contains the alert type of $a$
Output:
    $Cor_{max}$ - maximum correlation
procedure FINDCORRELATION($a, X$)
    $Cor_{max} \leftarrow 0$
    for each matched alert $b$ of $X$ do
        $Cor_{max} \leftarrow \max(Cor_{max}, \text{CORRELATE}(a, b))$
    end for
    for each parent $U$ of $X$ do
        for each matched alert $b$ of $U$ do
            if $a$ happens before $b$ then
                return $-1$
            end if
            $Cor_{max} \leftarrow \max(Cor_{max}, \text{CORRELATE}(a, b))$
        end for
    end for
    for each child $Y$ of $X$ do
        for each matched alert $b$ of $Y$ do
            if $b$ happens before $a$ then
                return $-1$
            end if
            $Cor_{max} \leftarrow \max(Cor_{max}, \text{CORRELATE}(a, b))$
        end for
    end for
    return $Cor_{max}$
end procedure
```

After describing the *happens before* concept and the FINDCORRELATION procedure, we can start looking at the MATCHALERT procedure which is shown in Algorithm 3. It takes a meta-alert a and an attack template AT as inputs and outputs a set $\boldsymbol{ATS}_{match}$ containing attack templates generated after matching. The objective of this procedure is to try to match meta-alert a to the attack template AT. It iterates over every node X in AT whose alert unit table contains alert type of a. It calls the procedure FINDCORRELATION to correlate a with X. If the result is greater than 0, it means a finds correlation in X. If the result is 0, it means a finds no correlation in X but it can be matched to X. We add X to a potential node set $\boldsymbol{X}_{pot}$. If the result is less than 0, it means a could not be matched to X due to conflicts. If a finds correlation in any node in AT, this means we have good reason to believe a is triggered by the attack represented by the current attack template AT. Therefore, we match a to the node X_{cor} with the highest correlation score and start a belief propagation from X_{cor}. In this case, the output will be a set containing only the updated AT with a matched. If a finds no correlation in AT but $\boldsymbol{X}_{pot}$ is not empty, this means there is no proof that a is triggered by AT but there are attack steps in AT that could potentially trigger a and the attack steps are included in $\boldsymbol{X}_{pot}$. Therefore, the procedure iterates over $\boldsymbol{X}_{pot}$ explores every possibility. For every node X in $\boldsymbol{X}_{pot}$, it creates a new copy AT_{match} of AT. Note that this copy contains not only the nodes of AT but also all already matched alerts of AT. It then matches a to X's counterpart in AT_{match} and starts a belief propagation in AT_{match} from that node. By doing this, the procedure takes every potential match of a in AT into consideration and the final output will contain the original AT as well as all updated copies of it. Finally, if there is no node in AT that a could match to, the output will just contain the original AT.

Algorithm 3 Match Alert Procedure

```
Input:
    $a$ - meta-alert to be matched
    $AT$ - attack template
Output:
    $\boldsymbol{ATS}_{match}$ - attack template set after matching
procedure MATCHALERT($a, AT$)
    $\boldsymbol{ATS}_{match} \leftarrow \{AT\}$, $X_{cor} \leftarrow None$, $\boldsymbol{X}_{pot} \leftarrow \emptyset$, $Cor_{max} \leftarrow 0$
    for each node $X$ in $AT$ whose alert unit table contains alert type of $a$ do
        $Cor \leftarrow$ FINDCORRELATION($a, X$)
        if $Cor > 0$ then
            if $Cor > Cor_{max}$ then
                $Cor_{max} \leftarrow Cor$, $X_{cor} \leftarrow X$
            end if
        else if $Cor = 0$ then
            add $X$ to $\boldsymbol{X}_{pot}$
        end if
    end for
    if $X_{cor}$ is not $None$ then
        match $a$ to $X_{cor}$ and start the belief propagation of $AT$ from $X_{cor}$
    else
        for each node $X$ in $\boldsymbol{X}_{pot}$ do
            $AT_{match} \leftarrow$ copy of $AT$
            match $a$ to $X$ in $AT_{match}$ and start a belief propagation of $AT_{match}$ from $X$
            add $AT_{match}$ to $\boldsymbol{ATS}_{match}$
        end for
    end if
    return $\boldsymbol{ATS}_{match}$
end procedure
```

In the description of the MATCHALERT procedure, we mentioned that the 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.

1) Example Run for the Anomaly Reasoning Algorithm: We use an example to better illustrate the anomaly reasoning algorithm. Consider the attack template AT in Figure 15. Let us assume this is the only attack template in the database. At the beginning of the anomaly reasoning, the causal reasoning engine fetches AT from the database and creates the attack template set $\boldsymbol{ATS} = \{AT\}$ as shown in Figure 16. Now the attacker first launched a man-in-the-middle attack. CAPTAR receives an `OPERATION_TOO_LATE` meta-alert a_1 from EDMAND. This meta-alert a_1 is first stored in the meta-alert database and then fed into the causal reasoning engine. Upon receiving this meta-alert a_1, the engine calls the anomaly reasoning algorithm. It finds that a_1 is a new meta-alert, so the procedure MATCHALERT is called to match a_1 to the only attack template AT in $\boldsymbol{ATS}$. The MATCHALERT procedure finds that node X_1 is the only node whose alert unit tables contains alert type `OPERATION_TOO_LATE`. Therefore, it calls the procedure FINDCORRELATION with a_1 and X_1 as inputs. The procedure FINDCORRELATION tries to find any meta-alert in X_1 and X_3 that correlates with meta-alert a_1. However, since X_1 and X_3 have no matched alert yet, the procedure finds no correlation and returns 0 in this case. Since no correlation of a_1 in the attack template AT is found and X_1 is the only node that a_1 can match to, a copy $\widehat{AT}$ of the attack template AT is created, and the meta-alert a_1 is matched to $\widehat{X_1}$ in the copy $\widehat{AT}$. A belief propagation is performed on $\widehat{AT}$. The MATCHALERT procedure returns both AT and $\widehat{AT}$. Finally, both AT and $\widehat{AT}$ are added to $\boldsymbol{ATS}_{new}$ to replace $\boldsymbol{ATS}$.

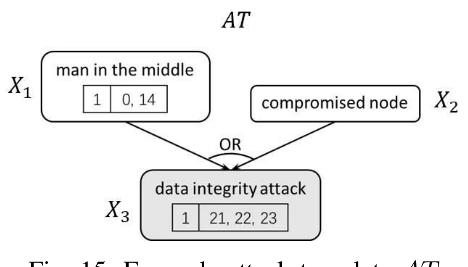


Fig. 15: Example attack template AT

Now the attacker intercepts and tampers with some binary data. CAPTAR receives a meta-alert a_2 with alert type of `BINARY_FAULT` from EDMAND as shown in Figure 17. a_2 is first stored in the meta-alert database and then forwarded to the causal reasoning engine. The anomaly reasoning algorithm again finds that a_2 is a new meta-alert, so the procedure MATCHALERT is called to match a_2 to both AT and $\widehat{AT}$. Matching a_2 to AT is similar to matching a_1 to AT and there is no correlation of a_2 in the attack template AT. Matching a_2 to $\widehat{AT}$ is a bit different. The procedure finds that node $\widehat{X_3}$ is the only node whose alert unit tables contains alert type `BINARY_FAULT`. Therefore, the procedure FINDCORRELATION is called with a_2 and $\widehat{X_3}$ as inputs. Since $\widehat{X_1}$ is the parent of $\widehat{X_3}$ and a_1 is a matched alert to $\widehat{X_1}$. The procedure sends both a_2 and a_1 to the alert correlator and finds that they are correlated. Therefore, it returns the correlation score of a_1 and a_2. Since the FINDCORRELATION procedure finds one correlation of a_2 in the attack template $\widehat{AT}$, a_2 is matched to $\widehat{X_3}$ and a belief propagation is performed on $\widehat{AT}$. After this run, the attack template set $\boldsymbol{ATS}$ contains the original AT and updated $\widehat{AT}$.

Later, EDMAND sends another updated `BINARY_FAULT` meta-alert $\widehat{a_2}$ to CAPTAR as shown in Figure 18. The anomaly reasoning algorithm finds that $\widehat{a_2}$ is an update to an existing meta-alert a_2. Also, it finds that $\widehat{AT}$ in $\boldsymbol{ATS}$ contains a_2. Therefore, it replaces a_2 with $\widehat{a_2}$ in $\widehat{AT}$ and starts new belief propagations in $\widehat{AT}$. After this run, the causal reasoning engine finds that $BEL_{max}(\widehat{AT})$ exceeds the predefined threshold. Therefore, the engine outputs $\widehat{AT}$ to the operator. The operator can see from $\widehat{AT}$ that there is a data integrity attack going on and the attacker first launched a man in the middle attack (MITM) to achieve that. The two matched alerts a_1 and $\widehat{a_2}$ can also be used for more detailed analysis by the operator.

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 described in [1]. The baseline traffic is 14 days of simulated DNP3 traffic of one control center communicating with 10 remote terminal units (RTUs). More details of the baseline traffic can be found in [1]. 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 Figure 19. 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.
- *Data integrity attack*: The attack template for data integrity attack is shown in Figure 20. 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.

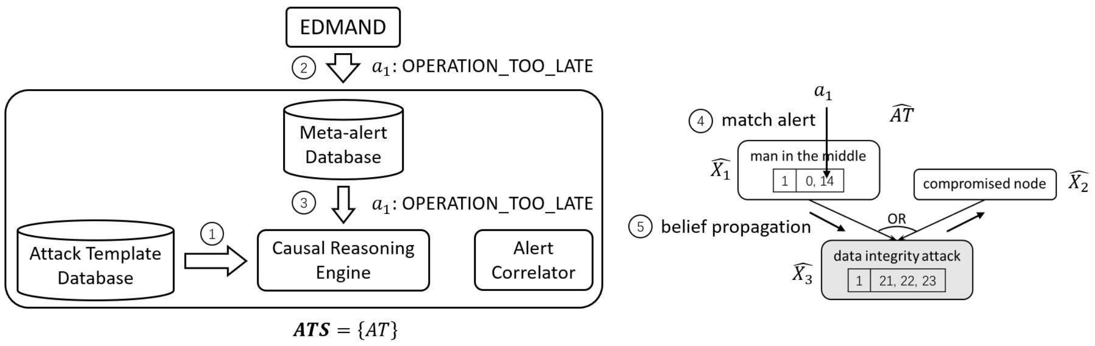


Fig. 16: Algorithm run upon receiving meta-alert a_1

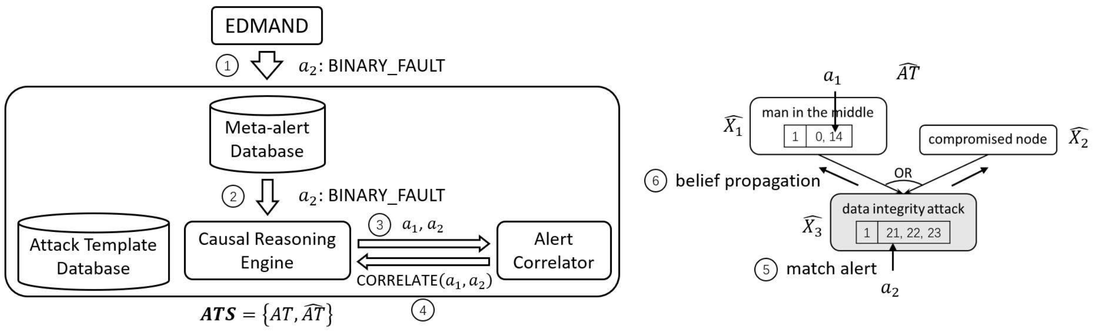


Fig. 17: Algorithm run upon receiving meta-alert a_2

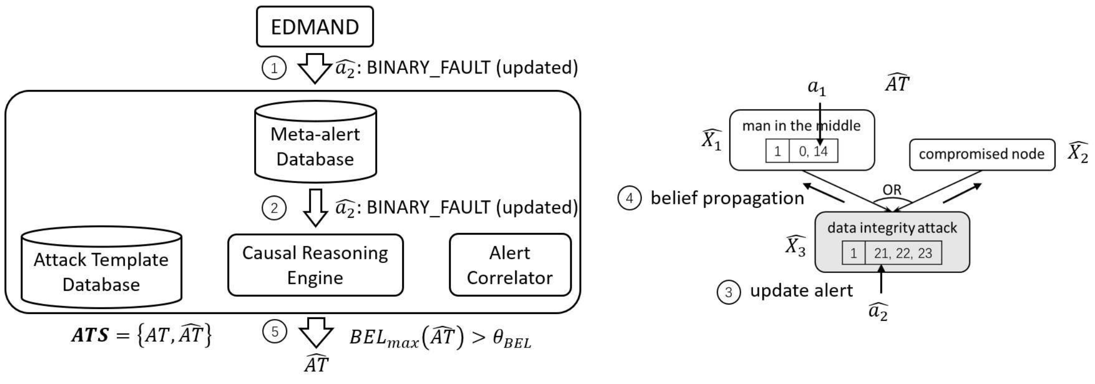


Fig. 18: Algorithm run upon receiving meta-alert $\widehat{a_2}$

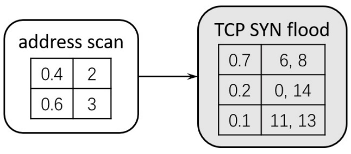


Fig. 19: TCP SYN flood

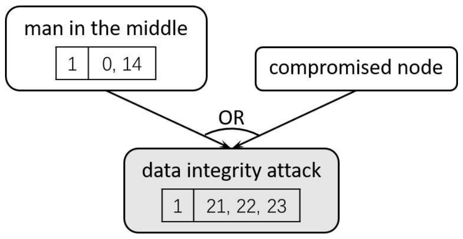


Fig. 20: Data integrity attack

- *Command injection*: The attack template for command injection is shown in Figure 21. 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.

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

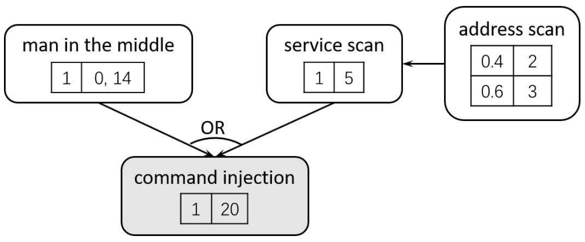


Fig. 21: Command injection

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.

We now briefly calculate the time complexity of the anomaly reasoning algorithm. We start by estimating the time complexity of the FINDCORRELATION procedure. Let us assume M to be the number of meta-alerts in the database. In the worst case, the FINDCORRELATION needs to correlate the input meta-alert with all other meta-alerts. Since the time complexity of correlating a pair of meta-alerts is constant, the FINDCORRELATION procedure has a $O(M)$ time complexity. Let us assume the maximum number of nodes in any attack template is N and L is the number of attack templates in the database. In the MATCHALERT procedure, the first ‘for’ loop needs to go over every node in the template in the worst case, which has a time complexity of $O(MN)$. The belief propagation is $O(N)$, and N_{pot} has N nodes in the worst case. So the rest of the procedure has a time complexity of $O(N^2)$. The total time complexity of MATCHALERT is therefore $O(MN + N^2)$. In the anomaly reasoning algorithm, the maximum attack template number is KL. It can be easily derived that the time complexity of the algorithm is $O(KLN(M+N))$. 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 reasonable. And notice that the frequency CAPTAR runs the anomaly reasoning algorithm is decided by the frequency that EDMAND sends meta-alerts. As mentioned in [1], EDMAND sends meta-alerts in a periodic manner only if there are updates to those meta-alerts in the latest period. So the sending rate of meta-alerts by EDMAND is also limited. 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 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. For each attack scenario, we run CAPTAR on the entire traffic set including the corresponding attack and calculate the average and standard deviation in millisecond of the time overheads for FINDCORRELATION, belief propagation, and the anomaly reasoning algorithm. We also record the sample number, which is the number of time FINDCORRELATION, belief propagation, and the anomaly reasoning algorithm have been performed. The results are shown in Table VI. 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 the FINDCORRELATION procedure and the anomaly reasoning algorithm varies a lot across different attack scenarios. This is because the time overheads of FINDCORRELATION and the anomaly reasoning algorithm depend on the number of meta-alert M as we described previously. 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 them. Another fact is that all the time overheads have relatively high standard deviation. This is mainly due to the change to meta-alert number in the meta-alert database during the attack. As the attack continues, the number of meta-alerts in the database increases, and so do the time overheads for FINDCORRELATION, belief propagation, and the anomaly reasoning algorithm.

VI. CONCLUSION

In this report, 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.

DISCLAIMER

Attack	FINDCORRELATION			Belief Propagation			Anomaly Reasoning		
	avg	std	num	avg	std	num	avg	std	num
TCP SYN flood	7.60	4.58	64	0.21	0.12	64	41.39	28.90	122
Data integrity attack	0.48	0.37	4	0.10	0.04	4	19.76	48.72	12
Command injection	2.65	1.61	25	0.14	0.02	25	13.95	34.52	40

TABLE VI: Average(avg in ms), standard deviation(std in ms), and sample number(num) of time overhead for FINDCORRELATION, belief propagation, and the anomaly reasoning algorithm

specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

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.