Protecting Cyber-Physical System Testbeds from Red-Teaming/Blue-Teaming Experiments Gone Awry

Md Rakibul Hasan Talukder(✉), Md Al Amin, and Indrajit Ray

Colorado State University, Fort Collins, CO 80523, USA
{rakibul.talukder,md.al_amin,indrajit.ray}@colostate.edu

Abstract. Many cyber-physical systems (CPS) are critical infrastructure. Security attacks on these critical systems can have catastrophic consequences, putting human lives at risk. Consequently, it is very important to pace CPS systems to red-teaming/blue teaming exercises to understand vulnerabilities and the progression/impact of cyber attacks on them. Since it is not always prudent to conduct such security exercises on live CPS, researchers use CPS testbeds to conduct security-related experiments. Often, such testbeds are very expensive. Since attack scripts used in red-teaming/blue-teaming exercises are, in the strictest sense of the term, malicious in nature, there is a need to protect the testbed itself from these attack experiments that have the potential to go awry. Moreover, when multiple experiments are conducted on the same testbed, there is a need to maintain isolation among these experiments so that no experiment can accidentally or maliciously affect/compromise others. In this work, we describe a novel security architecture and framework to ensure protection of security-related experiments on a CPS testbed and at the same time support secure communication services among simultaneously running experiments based on well-formulated access control policies.

Keywords: CPS · Testbed · Security experiment · Authorization · Isolation · Tuple space

1 Introduction

A Cyber-Physical System (CPS) consists of many individual units or systems and often is a critical infrastructure. Some example of such CPSs are power plants and distribution grids, gas transmission systems, traffic control systems, water treatment and supply systems, transportation systems, and others [2,10]. A single security vulnerability in CPS can lead to catastrophic consequences, which ultimately can cause considerable financial and business loss, human lives, suffering, and others [7,8]. Thus, it is paramount that CPSs are free from security

C. Su et al. (Eds.): ISPEC 2022, LNCS 13620, pp. 140–157, 2022.
https://doi.org/10.1007/978-3-031-21280-2_8

vulnerabilities. However, it is challenging to test a functioning CPS to identify security issues or weaknesses. This is because a live CPS cannot afford even a single error or mistake introduced while performing the testing; there is a real possibility of the testing process damaging the CPS. Consider, for example, a security experiment that needs the introduction of spoofed sensor measurements in the control network of a CPS to simulate an attack. Allowing this experiment to be conducted on a live system will cause it to malfunction. Unfortunately, the CPS can not be stopped or interrupted for this purpose. Moreover, such security testing needs to be done periodically since security threats are continuously evolving.

CPS testbeds play an important role for the security analysis of the CPS [14]. The testbed environment emulates the actual behaviors of the different CPS components and simulates their interactions. This allows researchers and engineers to identify security issues in the designed systems before deploying them in the real world and continuously update the knowledge with evolving threats. Since the testbed mimics existing systems' behaviors, it is essential to ensure that different experiments launched in the testbed environment reflect real-world behavior as closely as possible. This raises significant challenges in the design of the CPS testbed itself.

The CPS testbed provides an environment to conduct experiments to study the behaviors of the concerned systems. Researchers [6,11–13], have identified several requirements for testbeds to conduct CPS centric experiments. Fidelity, repeatability, scalability, adaptability, cost-effectiveness, measurement precision, diversity, and safe experiment execution are the major requirements. These requirements form the minimum set to conduct the CPS experiments effectively. However, these do not fully satisfy the requirements for cybersecurity-related experiments, those requirements are not enough to ensure the security of the security-oriented testbeds. Researchers also have to ensure that one experiment can not get data from another experiment without authorization (intentionally or unintentionally) [9]. Also, shared hardware resources can be an attack vector or data leakage platform.

For security centric experiments, it is important to ensure that one experimental process cannot go outside of its run-time memory. If this is allowed to happen, then it can cause unintended program execution or memory corruption of other infected experiments' memory and processes. This would ultimately produce erroneous results. Therefore, we must deploy individual experiments in isolated environments to confine operations and data inside the allocated working memory area. It may require different components of a sizeable cyber-physical system to be deployed as multiple experiments, and these experiments may need to share specific data among them. An experiment can utilize another experiment's data to complete a specific task. Although different units depend on each other to reflect the whole system's activities, ensuring safe communication to exchange information among the isolated components is necessary. It is impossible to provide a communication mechanism among isolated nodes using inter-process communication (IPC).

In this paper, we propose a novel communication design leveraging the tuple space model to provide inter-experiment data sharing where experiment nodes are deployed in isolated manner. The idea of classic tuple space is based on the Linda Programming Model [5]. Our proposed framework ensures the reliability of the whole system incorporating the mechanisms of isolation of the nodes, safe communication among experiments, and an authorization module with well formed access control policies. Specific contributions of out work are noted in the following.

- Identifying design requirements for security experiment-oriented testbed.
- Proposing a novel design of inter-experiment secure communication leveraging the concept of tuple space.
- Modeling access control system to allocate system resources for the experiments and approve inter- communication requests.

2 Threat Model Relevant for the CPS Testbed

Before delving into our contributions, we would like to explain the threats that a security-oriented testbed must guard against. Experiments perform various attacks on different simulated physical models of different devices in the testbed. If an attack goes beyond any experiment run-time environment, it causes severe damage to another experiment, like modifying configuration files, input data sets, output results, and others. The testbed threats can mainly be classified into two groups: *(i) outsider threats* and *(i) insider threats* and are discussed in the following.

2.1 Outsider Threats

A testbed is vulnerable to threats that are initiated from outside of it. Malicious actors from the outside world can exploit the testbed to compromise it and gain confidential information from it. Outside attackers can compromise the testbed by exploiting vulnerabilities in the testbed's hardware and software resources. In this work, we assume the testbed is secured from outside threats.

2.2 Insider Threats

Insider threats are originated within the testbed itself. Multiple experiment nodes share testbed resources like hardware, software, attack library module, network I/O, and other essential resources while deployed on the testbed and executed at the same time. Without any preventive mechanism, an experiment can access information from another experiment by unauthorized means and can push malicious data into another experiment's memory space. We classify testbed insider threats into three groups: *(i) confidentiality threats*, *(ii) integrity threats*, and *(iii) availability threats*.

Confidentiality Threats: Critical Cyber Physical Systems (CCPS) have many critical units to provide services. Since the testbed simulates the actual behaviors of those systems, it can hold notable data about those critical units and proprietary information about the system organization. If a malicious entity gains access of an experiment node, it may also obtain significant architectural information or system vulnerabilities. This reveal of system knowledge may help the malicious entity to plan a more sophisticated attack in the testbed. Moreover, malicious organizations can use that information to gain business profits and defeat competitors in business market competition.

Integrity Threats: An erroneous input data or process can introduce faulty results or add bias to the experiment results. An experiment can intentionally or unintentionally violate the data integrity of other experiments, if attack and data isolation are not maintained properly. As experiments share the same testbed resources, if a set of attacks goes beyond a component's memory scope, then an erroneous evaluation may take place.

Availability Threats: A malicious experiment can hold the computing resources intentionally for indefinite time period to make the service/resources unavailable for other experiments. In this scenario, other experiments remain in the waiting queue for log time because of resource shortage. Our current scope of proposed method does not deal with this kind of threat.

3 Overview of Our CPS Testbed Structure

As most CCPSs are distributed and networked, a testbed should imitate the functionalities of a distributed networked system. Otherwise, the threats related to any distributed networked system can not be explored while performing security experiments in the testbed. This is why, in order to provide a platform for message-passing capabilities among the nodes, a network event simulator is required. This simulator is also responsible for managing and distributing resources among the experiment nodes. A bunch of experimental nodes together build the whole structure of a cyber-physical system. Figure 1 illustrates different components of a testbed that we require. The major components are *(i) testbed controller*, *(ii) experiment server*, *(iii) virtualization of experiment nodes*, *(iv) experiment controller layer*, and *(v) experiment nodes.*

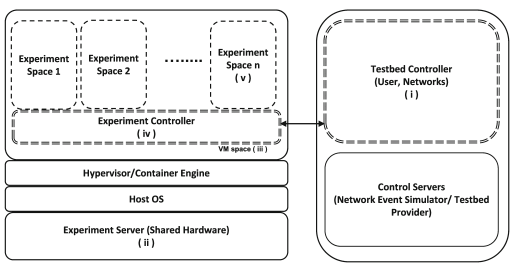

Fig. 1. Testbed structure.

The testbed controller (i) is responsible for managing resources for experiment nodes (v). It also provides services to allocate network address spaces for the experiment nodes as required by the experiment requirements. The controller has overall knowledge of the availability of resources, user space, node visualization, shell management, etc. It must be installed on a machine other than the experiment servers, allowing us to use resources in a request-response manner. This separation prevents experiment nodes from manipulating the control server's memory space. The system on which this controller is installed is known as the testbed control server.

Experiment servers(ii) provide the computing platform for experiment nodes. A single or multiple machines combined can provide the processing capability for experiment nodes. The number of experiments and connectivity among them is generally greater than the actual physical resources. Therefore, an abstraction of virtual machines (iii) and networking topology are supported based on containerization or hypervisor. One control node mapped to each owner of the experiment(s) will reside in the experiment control layer(iv). It is responsible for maintaining a gateway to the experiment space containing multiple nodes. A user, the experiment owner, can instantiate multiple nodes for the experiment. An experiment cannot have more than one owner. Experiment nodes are allocated resources from the experiment server(s). A hypervisor will run on the hardware of the experiment server to provide virtualization to the nodes.

4 Cybersecurity Centric Experiment Support in Testbed

There are some essential requirements for the testbed which play vital roles in providing a secure, efficient, and effective simulation environment and preventing improper experiment results. The most common requirements, identified by researchers [6,11–13], are *fidelity, repeatability, scalability, flexibility, cost-effectiveness, measurement accuracy, diversity, and safe execution of experiments*. In the following, we identify some essential cyber security-oriented testbed requirements in addition to the above-mentioned requirements. We also discuss their impact and influence on the testbed environment for conducting cybersecurity experiments in a safe manner.

Nodes and Experiments Isolation: To prevent unintended or malicious data exchange across experiments, an isolation mechanism is required to protect the data and process of an experiment. Assuming that each experiment has a single owner, this grants control over the nodes on which the experiment is executed. No inter-experiment network communication is allowed unless an authorized mechanism supports one. Moreover, parallel execution of processes (component functionality) on a single node may increase the risk of unintended influence. However, initializing an isolated container or VM per component can provide an extra layer of control and separation from the other components' functionality.

Secured Inter-Experiment Communication: Each experiment deployed on the testbed may represent a single component of an extensive system. Because

separate units rely on one another to represent the activities of the entire system, it is necessary to ensure secure communication between the experiments. But before one experiment collaborates with another, a coordinated approval is necessary to make the collaboration secure. Moreover, the testbed must ensure the communication messages' confidentiality, integrity, and availability. Violation of these three aspects of security can compromise sensitive information or produce incorrect results. Since experiment outputs influence the real CCPS design, it would be vulnerable when erroneous results are considered to design and deploy the real system.

Attack Libraries: Users need to launch multiple attacks on different experiments based on the deployed system components in the testbed. Support of built-in attack libraries provides accessible interfaces to perform attack experiments. Distribution and execution of attack libraries can be done in three ways: *(i) scripts provided and run by the testbed*, *(ii) scripts pulled from third-party organizations*, and *(iii) scripts developed by the owner*. The best practice is that users don't need to write scripts or access third-party sources. Before adding attack scripts to the testbed, they must be tested and evaluated correctly to ensure their intended outputs. Also, the testbed must maintain attack script integrity once they are included in the testbed.

Monitoring Module and Attack Analytic: An experiment owner can spawn a dedicated monitoring node with predefined objectives. The monitoring module performs tasks to collect and analyze experiment activities. An attack analytic, a part of the monitoring module, generates insights for further actions with visualization and relevant reports from the collected data.

Experiment Checkpoints: Sometimes it is necessary to roll back experiments to a certain executed state so that users can review experimental decisions and reconfigure the setup. Experiment checkpoints are necessary to recover from wrong states caused by cybersecurity experiments [4]. They also provide a quick recovery option so that experiments become fault-tolerant.

Attack Confinement: Various security analyses are carried out across multiple experiments based on the deployed system components at different nodes. But security attack experiments may attempt to expose the component's security flaws. Attack scripts are executed with predetermined actions to observe their consequences on a predefined perimeter of the system. It is essential to protect the testbed to confine an attack and its effect to prevent intentional or unintentional damage to another experiment. So an attack can not go from one experiment run-time environment to another without proper authorization.

Experiment Data Confinement: Each experiment node has a different set of data, like configuration and design files, experiment results, attack scripts, etc., containing sensitive information about the system organization. Any data leakage event may compromise the sensitive information, which can cause many unwanted consequences. Moreover, data integrity is also essential to ensure that experiments generate correct results, which are considered while making design decisions in a real CCPS system to protect it from known and unknown security

threats. However, data integrity may also be tampered with while data is shared and processed across experiments. Hence, the testbed needs to avoid these data leakage events where isolation can provide data confinement services.

5 Experiment Communication Model

Experiments are deployed as a combination of isolated nodes in the testbed. An isolated node of an experiment cannot communicate or share any data with other experiment nodes using the inter-process communication (IPC). But there are some scenarios where experiments need to exchange information to complete tasks. In this section, we identify and explain two types of communication required for our proposed security framework of communication in the testbed environment. They are: *(i) coordination communication* and *(ii) collaboration communication.*

Suppose, two experiments are active at the same time on the testbed and one experiment needs to access some data that another experiment contains. Any kind of communication that involves two nodes from different experiment is supported by the combined execution of coordination and collaboration communication. The coordination communication model (illustrated in Fig. 2) is occurred first to ensure the availability of resources, approval of the communication from etc. The actual intended communication or data sharing will not start before the coordination completes and the approval is received. Collaboration communication (illustrated in Fig. 3) starts when coordination is complete and nodes have received approval in the completed coordination process.

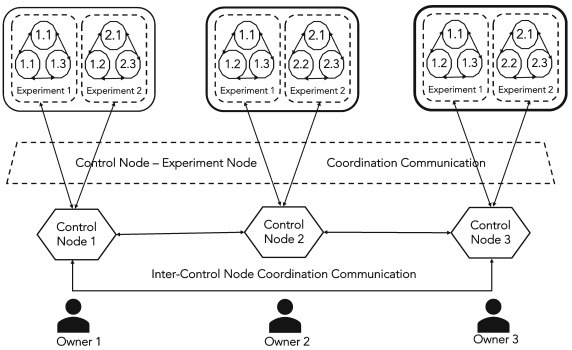

Fig. 2. Coordination communication.

Coordination Communication: A control node performs the main role in this communication model, where it sends predefined control messages to/from other experiment nodes. A control node acts as a coordinator on behalf of a user to initiate, manage, and terminate experiments in the testbed environment. A control node can send/receive coordination information (data access

request, broadcast request, node summary, etc.) to/from the experiments it controls or the other control nodes. The control node also expects feedback if it sends messages to other control nodes or experiments. Figure 2 depicts the coordination communication model where two types of coordination communication may occur: (i) control-to-control, (ii) control-to/from-experiment. In coordination communication, no experiment data is shared. Coordination communication should occur first if two experiments need to collaborate (share data).

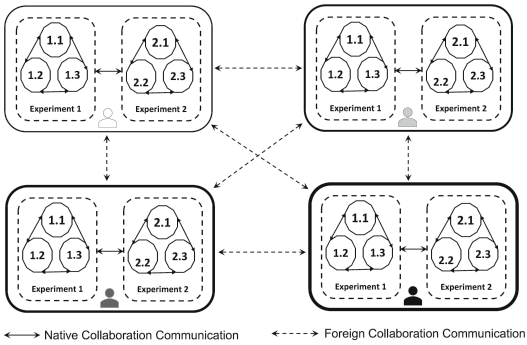

Fig. 3. Collaboration communication.

Collaboration Communication: The actual data transfer, labelled as collaboration of experiments, happens between two experiment nodes in this type of communication. No collaboration occurs without executing the coordination communication beforehand. In the collaboration communication, no control node is involved. The solid bi-directional arrows in Fig. 3 indicate this native collaboration where experiments under the same control collaborate (data transfer) with each other. In foreign collaboration communication (depicted in the Fig. 3 with dotted arrows), experiments from one control can communicate with experiments from another control node.

6 Overview of Experiment Execution on Testbed

In this work, we provide a system design for the testbed to provide control over communication among the experiments. Our proposed approach for inter-experiment communication leverages the technology of tuple space in an isolated environment. We assume that the testbed already has the facility of providing virtualization and isolation of nodes.

Experiment Initialization: When an owner wants to create experiments, a control node (allocating an isolated node) is instantiated first. A secure communication channel between a user and a control node is established to exchange necessary information. A control node is responsible for managing multiple experiment nodes for its owner. It passes a resources and privileges allocation request

containing the necessary node configuration from the owner to the authorization module. After getting the approval, multiple isolated nodes are allocated with a defined networking topology among them. For each initialized node, a tuple space is initialized, which is required for future coordination or collaboration communication. Thus, an experiment is initialized in the testbed.

Inter-Experiment Secure Communication: When two nodes from different experiments (owned by the same or different user) need to communicate, secure communication via tuple space is used. A tuple space is a service that provides a dedicated memory region in the isolated node's local memory space to store data or remove data when required. Only a mapped node itself and the tuple space manager (TSM) can access its memory region and perform an action on it. All the operations with the tuple space by TSM or the node itself are assumed to be secured. In this mechanism, two experiment nodes do not directly communicate with each other; rather, TSM (assumed as trusted) passes information from one tuple space to another.

Authorization of Communication: It is reasonable to assume that the communication between all the pairs of nodes from different experiments is not allowed directly. Whether communication between a pair of nodes is allowed or not is defined by the access control policy. When one node needs to access information held by another one from a different experiment space, a communication request is sent to the authorization module via the control node. The approval decision is sent back to the control node after evaluating the predefined access control policy. The process of approving resource allocation undergoes the same approval procedure.

7 Inter-Experiment Secure Communications

Three types of communication channels are notably used in the testbed to fulfill the requirements of communication nature. Nodes from the same experiments communicate with each other via standard or specialized networking protocols without prior approval (called intra-experiment communication, illustrated as a dotted line in Fig. 4). Communication is performed using the IP address or hostname of the nodes.

The secured communication between the authorization module and any control node (illustrated as a solid arrow in Fig. 4) is another type of channel. The third type of communication is between two tuple spaces. Each node (experiment or control) is mapped with a tuple space (illustrated as a rectangular purple region attached to each node). The details of tuple space communication managed by a Tuple Space Manager (TSM) are described in Sect. 8. Figure 4 illustrates the step-by-step process of inter-experiment communication involving different channels and related components. The steps are explained in the following:

Send Data Access Request (DAR): The data requester node sends the request information to the control node of its own experiment space (we call it the requester control node) first, leveraging tuple space communication (step 1

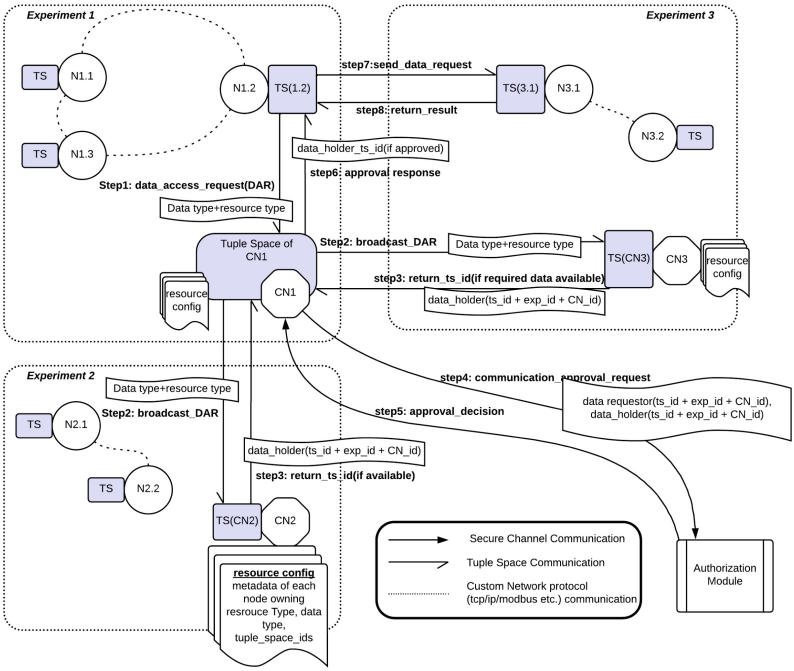

Fig. 4. Inter-experiment communication flow diagram.

from Fig. 4). The data requester does not know which node has the specific type of data that the requester wants. It only informs its own control node about the requirements of the specific data type. And in return, it only expects the identity of the data-holder tuple space so that it can start communication with it. No IP address or machine address of any node is disclosed in the whole communication process.

Broadcast DAR to Other Control Nodes: After receiving the DAR, the control node looks for the identity information of the requested data type from the past communication history that is stored in the *resource config*. A resource config file is maintained at every control node to provide necessary information about tuple space (TS) identities of experiment nodes it controls, experiment identities, mapping information between nodes and experiments, type of data each node holds, past communication history, TS identities of other control nodes, etc. This resource config file is updated from time to time if any event occurs at its own experiment spaces or other control nodes so that the config information remains consistent. Resource config files can also be updated when any previous approval decision is changed. If any approved active tuple space is found from the history as a data holder, no further approval is necessary. The TS id of the data holder is returned to the requester (go to step 6 as illustrated in Fig. 4). But if there is no history of prior approval of the same DAR, then a

broadcast to all the other control nodes takes place (step 2 from Fig. 4). It relays the same message of DAR in this broadcasting phase.

Return Data/Resource Availability Information: After getting a broadcast DAR from any other control node, the receiving control node will look for the availability of the data type in its own resource config file. If there is any experiment node that holds the requested type of data, the corresponding control node will find that information in its resource config file and return the TS id of the data holder along with other information (experiment id, control node id, etc.) back to the requester control node. If a receiving control node finds no data availability of the requested type, 'NOT_AVAILABLE' is sent back (step 3 from Fig. 4).

Send Approval Request to the Authorization Module: Now the requester control node has the information (node TS id, experiment id, etc.) about the data requester and data holder. Using the secured communication channel already established between the control node and the authorization module, an approval request is sent to the authorization module to assess whether the requested data from the data holder can be accessed or not. (step 4 from Fig. 4).

Return Approval Decision: After receiving an approval request from a control node, the authorization module intends to check if the request complies with the access control policies. Details of the authorization module and access control can be found in Sect. 9. The approval decision is notified to the tuple space manager to update the *approved_communication_list*. Finally, the approval decision is sent back to the requester control node (step 5 from Fig. 4).

Return TS Id to Requester: Before passing the approval decision to the data requester node, the control node stores this information in the resource config file for future use. If the DAR is approved, the TS id of the data holder node is included in the approval response message. If the DAR is denied or data is not available, DENIED or NOT_AVAILABLE is included in the approval response message. No further communication takes place; this flow terminates here.

Send Data Request to Data Holder: If the DAR is approved, the awaited communication via tuple space takes place now. First, a data request message is passed from the requester to the holder via tuple space manager. This message includes the holder's TS id and data type.

Return Result to Data Requester: In response to the data request sent by the requester, the requested result is passed from the holder to the requester.

8 Testbed Tuple Space Design for Isolated Experiments

The tuple space model provides a mechanism which allows experiment nodes placed in an isolated environment share data without using any direct communication channel. In the following, we discuss the tuple space manager, tuple space operations, and tuple space transactions.

8.1 Tuple Space Manager-TSM

The TSM is a secured and trusted entity that performs data transfer operations from source tuple space to destination tuple space. A secured entity protects data from being modified or revealed to illegitimate subjects. Also, it does not analyze the tuple space content (called tuple) to disclose data to other entities or to learn more about data for itself. It needs to read a tuple from the sender tuple space and add it to the receiver tuple space. There is only one global TSM in the proposed security-oriented testbed. The TSM maintains an *approved_communication_list* that gets updated by the authorization module from time to time. The list is essential to crosscheck the authorization status of the incoming request to prevent malicious transactions.

8.2 Tuple Space Operation

The proposed framework supports three basic tuple space operations: *(i) write*, *(ii) take*, and *(iii) read* and described in the following.

Write(tuple): This operation provides the functionality to add a tuple within the mapped tuple space. It does not modify the tuple space contents. The sender node and the tuple space manager can execute this operation. The sender node uses this to add the tuple into the tuple space it owns. The TSM executes this operation by adding the tuple in the receiver node's tuple space.

Take(template tuple): This operation is called to execute an associative search for a tuple that matches the template. Once found, the tuple is deleted from the space and then returned to the tuple space owner's run-time memory. Only tuple space owners can use this function to remove tuples from their tuple space. The Tuple Space Manager cannot execute this operation because TSM does not have any rights to remove tuples from any tuple space.

Read(tuple): To read from the sender's tuple space, the TSM executes the read operation. This operation gets a tuple back from sender's tuple space to TSM's own memory space without removing from the source tuple space.

8.3 Tuple Space Transaction

This section provides the illustration (depicted in Fig. 5) of a tuple space transaction for communication between two isolated nodes. The tuple space manager is the main medium for passing contents(tuple) from sender tuple space to receiver tuple space. The TSM has access to all the tuple spaces whereas the nodes can access their respective ones.

There are two phases in communication: the request phase and the response phase. In the request phase, the sender node sends a message to the receiver node. In the response phase, the receiver node returns a message to the sender node. The returned response can be completely new information or serve as an acknowledgment for the tuple that has just been received. After sending the response message, the communication flow is terminated. Both request and

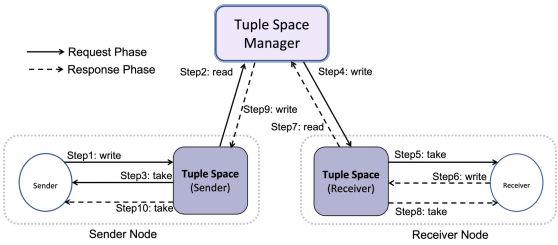

Fig. 5. Tuple space transaction.

response phase execute the same transaction process to share information which are depicted in the Fig. 5. If a node acts as sender in request phase then it is receiver in the response phase and vice versa.

9 Authorization Module (AM)

The authorization Module's entire structure and functionalities are explored and illustrated in this section, including the necessary examples. The resources and privilege allocation method are discussed to construct an isolated space for each experiment.

9.1 Experiment Resources and Operations

We need to define the resources required and the operations performed by the experiments. The authorization module acts as a reference monitor to control the operations of resources requested by the nodes. An experiment needs multiple hardware and software resources with different privileges to perform operations to complete the scheduled tasks. The control node is responsible for identifying and releasing both resources before an experiment. Some of the resources required by the experimental nodes to complete experiments are network topology configurations, internal memory, disk space, simulated physical models, attack libraries, PLC program control code, proprietary information, snapshots, loggers, status logs, tracers, experimental results, and others.

The experiments in this model perform three operations: *(i) read operation*, *(i) write operation*, and *(i) execute operation*. The *read operation* accesses various files and resources, such as configuration files, input data, programs, and so on. Experiments use *write operation* to write output results, modify configuration files, adjust input parameters, and so on. Finally, the *execute operation* allows experiments to run various attack scripts, simulated physical models, and other processes.

9.2 Resources and Privileges Allocation

A control node (CN) determines an experiment's required resources and privileges. After selecting the needed resources and rights, CN securely sends the

list to the authorization module. In Algorithm 1, the details of resource and privilege allocation processes are given. Both the control node and the authorization module ensure that no experiment is given root permission or excessive resources and rights than required. In addition, the control node can terminate an experiment node after the simulation is completed.

Algorithm 1: Resources and privileges allocation

Input : A list of resources (R) and privileges (P), and experiment id m.
Output: *IsolatedNode* for the experiment m under control node i.

Initialization:
$R \leftarrow \{R_1, R_2, R_3, ...R_r\}$,
$P \leftarrow \{P_1, P_2, P_3, ...P_p\}$;
$CN_i \rightarrow AM$;
AM checks resources' availability and verify resources and privileges legitimacy for the CN;
if *resource available* **then**
    AM executes required OS command with R and P;
    $IsolatedNode_{i,m}$ is deployed;
    AM returns a success message to CN with contained id;
else
    AM returns an error message to CN;
end if

9.3 Relationship Between Control Node, Experiments, and Subjects

The distinction between control nodes, experiments, and subjects is fundamental to the authorization module of this work. The relationship between control node, experiments, and subjects is depicted in Fig. 6. Usually, each node can perform three operations in the testbed, but in data sharing, only read operation is permitted. The other two operations, write and execute, are not allowed for data sharing.

Control Node: Every user has one control node, which acts as a single identity in the testbed environment. An authorized organization has only one control node but multiple control nodes. Violation of this requirement is often the cause of security violations in the proposed system.

Experiments: Each control node may have several experiments associated with it. On the other hand, one experiment must not be mapped to more than one control unit. Therefore, each experiment associated with the control node gains different resources and privileges based on its functionalities.

Subjects: A subject is a program in the system being executed. An experiment can generally spawn several subjects, but each subject is associated with only one experiment. A subject runs with all the privileges of its associated experiment.

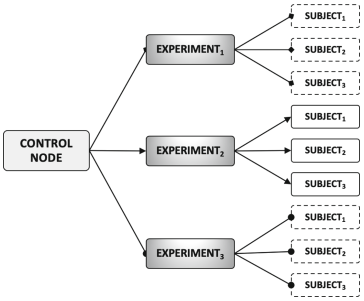

Fig. 6. Relationship between control node, experiments, and subjects.

9.4 Object Classification

To maintain data security and privacy, we classify objects(data) into three main categories. The object classes are the conflict of interest class, agreement class, and open class.

Conflict of Interest Class: The member objects of this type of class are in a conflict of interest and cannot be shared among their owner.

Agreement Class: Objects labelled with this class are in an agreement and can be shared among the agreement signing parties.

Open Class: Object data tagged with open class is for all. Any control node can send requests to get the data.

9.5 Control Node Identification

Each control node is labeled as a member of the conflict of interest class and the agreement class. The authorization module considers the control node's identity when making a data-sharing decision where the sender and receiver are from different control nodes. Experiments or subjects are not considered. Because experiment nodes from the same control node can share information without regard to security or policy constraints. We avoid experiments or subjects while the access control module approves data sharing to prevent this situation. When a control node is in a conflict of interest class or not in the agreement class, it will not receive data from that class's members.

9.6 Security Policy

In this section, we present some access control policies to ensure inter-experiment communication and testbed security and privacy. There are mainly two, native owner and foreign owner, communication scenarios.

Native Owner: In this case, we describe data sharing among various experiments when they are all from the same control node. Every node does not need to access data from every node. Experiment nodes must ensure data confidentiality. There are two principles when sharing data among experiments or subjects. Both principles are noted in the following.

Principle 1: *An experiment or a subject can not read data from other experiments or subjects if it violates the confidentiality of the data.*

Principle 2: *An experiment or a subject can read data from other experiments or subjects if they do not violate Principle 1.*

Foreign Owner: When there is a data-sharing request where the requester and data holder are from different control nodes, in this case, data sharing can be done if there is no conflict of interest among the experiments. If there is a conflict of interest, the authorization module must not approve data sharing. The conflict of interest issue is raised when the requester and the data holder are from the same conflict of interest class. Data sharing is also possible if there is an agreement between the requester and the data holder. Any control node can access the open class data. A control node puts in a request on behalf of its experiments and subjects. The authorization module depends on the object class and requester control node identity to make the decision. In the following, there are three principles for each type of object.

Principle 3: *A control node can read an object if they are not in the same conflict of interest class.*

Principle 4: *A control node can read an object if they are in the same agreement class.*

Principle 5: *A control node can read any object if objects are in the open class.*

10 Related Works

The Linux Policy Machine is proposed by [3] as the centralized reference monitor to provide secure inter-component communication in an isolated environment. In addition, they introduce tuple space to facilitate communication mechanisms for the isolated components. They may require regulated and secure access to system resources and the ability to collaborate and coordinate with one another. [9] propose an architecture, ISAAC, for performing security experiments on a testbed for smart grid systems. It's a cross-domain, re-configurable, and distributed framework that simulates data from power generation in operations. It allows researchers to develop, test, evaluate, and validate holistic cyber-physical security techniques for cyber-physical systems and the Smart Grid. [1] utilize simulation data from the Internet Scale Event and Attack Generation Environment (ISEAGE) to characterize the system architecture of a security testbed for PowerCyber developed at Iowa State University. [12] present EPIC framework that can accurately assess the effects of cyber-attacks on the cyber and physical dimensions of networked critical infrastructures (NCIs), such as power plants.

11 Conclusion and Future Directions

This paper identifies some requirements for cyber security-oriented testbeds to ensure the security of the testbed while carrying out various security experiments. Our designed communication mechanism using tuple space in context

with isolated experiments provides desired protection against threats. We also recognize the significance of data sharing among organizations while avoiding conflicts of interest and protecting proprietary information. Our designed authorization module and access control policy prevent unauthorized access to data and communication.

In future, we will consider the TSM not trusted. Cryptographic algorithms or zero-trust-based solutions may come into use in that scenario. We will also detect availability threats based on testbed resource consumption and run time.

Acknowledgements. This work was supported in part through funding from the US Department of Energy under CID #DE-NE0008986, the US National Science Foundation under grant #1822118, the industry partners AMI, NIST, Cyber Risk Research, Statnett, New Push and ARL of the NSF IUCRC Center for Cybersecurity Analytics and Automation, and the Colorado State University. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the DOE, the NSF, the industry partners, the University, or any other federal agencies.

References

1. Ashok, A., Hahn, A., Govindarasu, M.: A cyber-physical security testbed for smart grid: system architecture and studies. In: Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research, pp. 1–1 (2011)
2. Banerjee, A., Venkatasubramanian, K.K., Mukherjee, T., Gupta, S.K.S.: Ensuring safety, security, and sustainability of mission-critical cyber-physical systems. Proc. IEEE **100**(1), 283–299 (2011)
3. Belyaev, K., Ray, I.: Component-oriented access control for deployment of application services in containerized environments. In: Foresti, S., Persiano, G. (eds.) CANS 2016. LNCS, vol. 10052, pp. 383–399. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48965-0_23
4. Burtsev, A., Radhakrishnan, P., Hibler, M., Lepreau, J.: Transparent checkpoints of closed distributed systems in emulab. In: Proceedings of the 4th ACM European Conference on Computer Systems, pp. 173–186 (2009)
5. Carriero, N., Gelernter, D.: Linda in context. Commun. ACM **32**(4), 444–458 (1989)
6. Holm, H., Karresand, M., Vidström, A., Westring, E.: A survey of industrial control system testbeds. In: Buchegger, S., Dam, M. (eds.) Nordic Conference on Secure IT Systems, LNSC, vol. 9417, pp. 11–26. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26502-5_2
7. Kim, S., Heo, G., Zio, E., Shin, J., Song, J.G.: Cyber attack taxonomy for digital environment in nuclear power plants. Nuclear Eng. Technol. **52**(5), 995–1001 (2020)
8. Line, M.B., Tøndel, I.A., Jaatun, M.G.: Cyber security challenges in smart grids. In: 2011 2nd IEEE Pes International Conference and Exhibition on Innovative Smart Grid Technologies, pp. 1–8. IEEE (2011)
9. Oyewumi, I.A., et al.: ISAAC: the idaho cps smart grid cybersecurity testbed. In: 2019 IEEE Texas Power and Energy Conference (TPEC), pp. 1–6. IEEE (2019)
10. Shi, J., Wan, J., Yan, H., Suo, H.: A survey of cyber-physical systems. In: 2011 International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1–6. IEEE (2011)

11. Siaterlis, C., Garcia, A.P., Genge, B.: On the use of emulab testbeds for scientifically rigorous experiments. IEEE Commun. Surv. Tutorials **15**(2), 929–942 (2012)
12. Siaterlis, C., Genge, B., Hohenadel, M.: Epic: a testbed for scientifically rigorous cyber-physical security experimentation. IEEE Trans. Emerging Top. Comput. **1**(2), 319–330 (2013)
13. Smadi, A.A., Ajao, B.T., Johnson, B.K., Lei, H., Chakhchoukh, Y., Al-Haija, Q.A.: A comprehensive survey on cyber-physical smart grid testbed architectures: requirements and challenges. Electronics **10**(9), 1043 (2021)
14. Sridhar, S., Hahn, A., Govindarasu, M.: Cyber-physical system security for the electric power grid. Proc. IEEE **100**(1), 210–224 (2011)