

May 27, 2021

**Cybersecurity Considerations for Blockchain**

**Final Report for Task1**

1. Introduction

The objective of this research project is to investigate to what extent blockchain technology, or hyperledger fabric, helps or hinders cybersecurity. This applies to cases where a system has already been compromised, and in cases where the system may be vulnerable to cyberattacks but not yet hacked.

This research project consists of two parallel research tasks:

* Task 1: Use of blockchain technology to store cyber activities and logs into immutable records to protect them from deletion, tampering, or surreptitious modifications.
* Task 2: Create a risk model to assess vulnerability to hypothetical attacks against Hyperledger-based solution networks.

This report will cover Task 1 activities and deliverables. A separate report by another team will cover Task 2.

1. Task 1 Description

Task 1 in divided into the following subtasks:

* Subtask 1.1: Identify target IT/IoT devices for activity monitoring. Develop a client (an application program) for the target devices to capture their static states and feature sets, such as manufacturer, hardware features, software features, version number, in addition to dynamic cyber activities captured during cyber operations and the handshaking sequence. Metrics of monitored static and dynamic parameters to be generated and refreshed in real-time by the client on the selected devices.
* Subtask 1.2: Select a suitable server on the cloud, and develop a Hyperledger Fabric (HLF) infrastructure on the selected server. Create the ledger needed to store the static and dynamic parameters collected by the clients.
* Subtask 1.3: Transfer and log the metrics generated in Subtask 1 into Digital Twins stored on the HLF generated in subtask 2. Log/Organize/compile the metrics using smart contracts and Blockchain technology.
* Subtask 1.4: Provide specific Guidelines for using blockchain in cybersecurity solutions for the mathematical modeling effort conducted by a different team.
* Subtask 1.5: Provide support to the mathematical team in performing cybersecurity testing after they complete the mathematical model.
* Subtask 1.6: Prepare a report describing the work conducted, deliverables, with specific recommendations.

1. Task 1 Design Requirements

A methodical approach was followed in pursuing Task 1, where a set of requirements were carefully formulated to meet the project objectives, followed by design and development to meet these requirements. This section will document the requirements.

**Cybersecurity Monitoring Client (CMC)**

The first step is to develop a *Cybersecurity Monitoring Client (CMC)* on a selected IoT device type to monitor and log the following information:

* **Static information,** which is collected from device itself, such as manufacturer, hardware features, software features, version number, etc.,
* **Dynamic information,** which isgenerated during IP-based communication handshaking sequences, as defined by the respective communication protocols used by the device to connect with the network, such as security key lengths, encryption and signature algorithms, refresh periods, initialization vectors and similar cryptographic-relevant parameters.

These device parameters/attributes will be logged and stored with cryptographic access. This document will define the parameters to be captured on the device itself, and on the adjacent devices interacting with the IoT in a defined local cluster.

The IoT *Cybersecurity Monitoring Client (CMC)* shall acquire the following parameters:

1. **Static Parameters**: Upon device boot/startup sequence, the client shall make function calls to the OS to collect the following parameters:
2. Device MAC address & serial number
3. Manufacturer
4. OS type and version number
5. Memory size
6. If the device IP address was hard-coded, then this static address should also be collected by the client. If the IP address is dynamically assigned to the device, then the IP address will be determined using DHCP, as described in the next section.
7. Other static system parameters are to be considered

These parameters should be stored on the device and on the remote Hyperledger Server.

1. **IP layer Addressing Parameters**: An IP device will typically run the DHCP protocol after a powerup sequence, and will exchanges the DHCP messages (Discover, Offer, Request, and Ack) with the local/remote Gateway. The IoT CMC shall capture the following parameters from the device (after the completion of the DHCP dialog):
2. Device dynamic IP address and subnet mask
3. First Hop IP address and subnet mask
4. Name and IP address of the DNS server
5. Other IP layer parameters in the DHCP messages are to be considered

These parameters should be stored on the device and on the remote Hyperledger Server.

1. **IP Layer Error Reporting Protocol**: This is the standard ICMP protocol used by IP devices to discover other IP devices on the Internet. The IoT CMC shall capture the following parameters:
2. Echo request & Reply (ping)
3. Destination unreachable
4. TTL Expired (routing loop, or destination is too far)
5. Traceroute information

These parameters should be stored on the device and on the remote Hyperledger Server.

1. **ARP Cache and ARP Table:** On a single physical network, individual hosts are known to other devices in the network by their physical hardware address (MAC address). Since higher-level protocols address destination hosts using IP addresses, a translation table between Physical MAC addresses and Network IP addresses is created by all the devices on the local network using the ARP protocol. This information is stored in the dynamic ARP Cache, and in the ARP Table (the ATP Table which also contains manually entered IP-MAC mapping). The ARP Cache has the potential of being used by cyber attackers to discover other nodes on the LAN. It is therefore important to consider both ARP Cache and ARP Table.

The IoT CMC shall capture the following:

1. The ARP Cache content
2. The ARP Table content

Both the ARP Cache and ARP Table should be captured periodically (periodicity to be determined), and stored on the device and on the remote Hyperledger Server.

1. **The Domain Name System (DNS)** is the distributed network of servers that acts as a directory, cataloging domain names and their corresponding Internet Protocol (IP) addresses. On the other hand, a DNS cache is a local storage on the end-user side that contains the records of a computer’s query history, including recent website visits. Each IP device has a name resolver routine, which knows the name of a local DNS server. The Resolver sends a DNS request to the DNS server to get the desired destination IP address from the destination name (this is called DNS lookup). The IP device keeps a DNS Cache in its memory, and each entry in the Cache is associated with a Time-to-Live (TTL) value, which will expire over time. When the entry is valid (not expired), DNS caching expedites the DNS lookup process to more quickly resolve a domain name to an IP address when the OS has visited a web page before. If the entry is expired, the OS on the device will query the DNS server to update/refresh the Cache.

DNS poisoning, or DNS spoofing, refers to the cybersecurity threat in which hackers corrupt DNS resource records. By changing the IP addresses associated with particular domain names, hackers can hijack a web session and send computers requesting a particular site to the wrong web server.

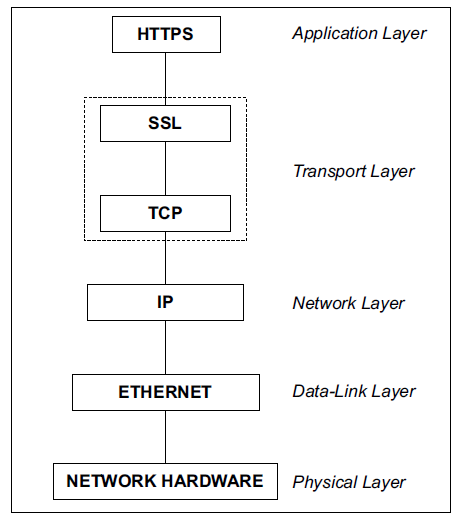
These alternate pages may expose users to advertisements, prompt them to install malware, or succeed at stealing private data (like Social Security numbers or financial information) if they pass as the correct website and convince users to enter sensitive data. Routinely clearing DNS caches both narrows the window of opportunity for DNS poisoning and wipes any corrupted records. However, a cyberattack may occur before the DNS cache is cleared.

The IoT CMC shall capture the DNS Cache periodically (periodicity to be determined). The DNS Cache is to be stored on the device, and on the remote Hyperledger Server.

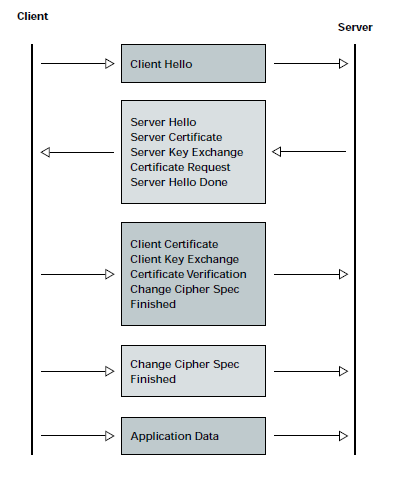
1. **TCP layer Connectivity Parameters**: TCP sessions are created and released all the time between IP devices communicating over the Internet. The IoT CMC shall capture the following parameters:
2. Source IP address and Port
3. Destination IP address and Port

These parameters should be stored on the device and on the remote Hyperledger Server.

1. **SSL layer Connectivity Parameters**. Secure Sockets Layer (SSL) was originally developed by Netscape Communications to allow secure access of a browser to a Web server. SSL is the accepted standard for Web security. The main role of SSL is to provide security for Web traffic. Security includes confidentiality, message integrity, and authentication. SSL achieves these elements of security through the use of cryptography, digital signatures, and certificates. The SSL reference model in relation to TCP/IP protocol stack is as shown below.



Every time a device attempts to establish a secure connection/session with a web server, handshaking for session initialization takes place between the client and the server to exchange and agree upon the keys to be used for cryptography, digital signatures, and certificates. The handshaking is shown below.



The key parameters in this handshaking are the following:

1. Server IP address
2. Server’s digital certificate
3. Agreed upon Cryptographic algorithm (between the client and the Server)
4. Server's public Key

These parameters should be stored on the IoT device and on the remote Hyperledger Server.

1. **Cookies:** Servers deposit cookies on the client device when the server is accessed. The cookies should be captured periodically (periodicity to be determined), and stored on the device and on the remote Hyperledger Server.
2. **Browsing History**: Browsers keep history of web activities on the local device. This history should be captured periodically (periodicity to be determined), and stored on the device and on the remote Hyperledger Server.

**Hyperledger Fabric (HLF)**

The VizLore Chain Rider platform has been selected as the Blockchain-as-a-service (BaaS) platform for the Hyperledger fabric. It shall be used to create and populate the hyperledger fabric with the parameters extracted from the IoT devices. This will create permanent record as Digital Twins of the IoT status and cyber activities. The Chain Rider REST API shall be used between the IoT monitoring client and the Server on the cloud.

1. Selected IT/IoT Device

An IoT device used for smart access and proximity functions is selected as a typical edge device. The device has the following technical specifications:

1. Hardware platform

* Model: Raspberry Pi 4
* Architecture: 32 Bit

1. Operating System & Programming Environment

* OS: Raspbian (Unix-like family, Raspberry Pi OS optimized for the Raspberry Pi line of compact single-board computers with ARM CPUs)
* Type: Debian

1. Basic Functions of the Device

This device imitates several full-stack devices of IT infrastructure such as servers, computers, desktop etc. Although the device is used to portray several devices of IT infrastructures, Raspberry Pi are used in robotics, weather monitoring, task automation. Since these devices are modular, inexpensive and have open design, the presence of these device is ubiquitous.

1. IP address

IP Address is allocated dynamically using the DHCP protocol. We can assign static IP address to devices as well.

1. Cybersecurity Monitoring Client (CMC) Functions and Design

An application, referred to as a Cybersecurity Monitoring Client, or CMC, running in the background of the user’s IT/IoT device is designed and implemented on the selected device, as follows:

Position in architecture

The cybersecurity monitoring client (CMC) resides within the device, monitoring all the network communications in and out of the device. Every network packet the device receives or sends are sniffed by the monitoring client to process necessary information to be able to retrack communication metadata if needed.

Functions

The CMC is able to monitor and store the static and dynamic parameters on the device while it is operating for a short amount of time. In addition, it is able push this information to a Hyperledger fabric residing in the cloud, over REST messages. Since these devices at hand have limited storage space, we flush the locally stored data on a regular interval.

Interfaces

The device supports CLI (Command Line Interface) communication. The device uses REST API to transfer the static information collected right after the boot sequence.

CMC Permission

CMC is able to sniff the packets coming in and going out of the device. Hence the client will need root (administrator) level access to be able to function.

Boot Sequence and Network Connection

In case the device shuts down, the device boot-up sequence will complete only if the device is connected to active network connection. As it is important not to lose any communication to-and-from of the device, the device will wait for a network connection to complete its boot sequence.

Static Data

These parameters are acquired by the device once the CMC is up and running. This static information will be intact until the next reboot / boot sequence.

ARP Cache and ARP Table

The selected device, like all IP capable devices, maintains ARP cache and ARP Table (IP to MAC mapping) of all the devices that the device had communicated with previously. There might be cases where the contents of ARP cache have been poisoned or corrupted, in case the attacker has specifically targeted a certain device. This can be only detected by comparing it with the networks ARP Table. Hence it is necessary to capture, store, and transmit to the Hyperledger fabric both the ARP Cache and the ARP Table.

Caching DNS Data

The selected device doesn’t cache DNS data, unlike Windows systems, so it was imperative that the DNS records be sniffed on the fly and stored on a file, and then read and flushed periodically.

Single Thread vs. Multi-Thread

The cybersecurity monitoring client (CMC) is developed in multi-tasking environment, with modular design in mind; hence, the CMC will work efficiently on the devices capable of having a large thread pool.

**Capturing Static Data**

The Static parameters of the IT/IoT were captured as follows:

* Device MAC address:

Used “Interfaces MAC” packages to obtain MAC Address of devices. The current configuration of the device is enabled only on ‘eth0’ interface of the device as the device has only one network card that supports only ethernet connection. The CMC currently is enabled to get static information for one of its active network connections.

* Serial Number, Manufacturer, Hardware, Memory size:

The file “/proc/cpuinfo” contains relevant information such as Serial Number, Manufacturer, Hardware, etc. There is also a mapping from Model to Manufacturer stored as a json file to get corresponding Manufacturer info. This file with mapping from Manufacturer code to Manufacturer name is taken from various online resources.

(\*Note:Various UNIX based devices with recent updates have changed the formatting of /proc/cpuinfo file, hence this feature might need further work to obtain these results consistently).

* OS Info and version number

Upon reading the contents of “/etc/os-release” file, we obtain information pertaining to the operating system, such as Name, Hostname, OS Release and Architecture.

* Static IP (if assigned):

Categorize the IP by the contents of “/etc/network/interfaces”. If the current IP is assigned dynamically, there won’t be any entry for Static IP address.

**Capturing Dynamic Data**

The Dynamic parameters of the IT/IoT were captured as follows:

Capturing IP layer Addressing Parameters:

The parameters such as IP Address, Broadcast IP, Gateway IP and DNS Nameservers need not be captured on the go, but can be retrieved back from the device, as follows:

* For IP Layer Parameters: Used “netifaces” package to handle IP Layer addressing. “netifaces.ifaddresses” method and “netifaces.gateways()” provide necessary interface and gateway addresses.
* Captured Dynamic Device IP, Broadcast IP, Gateway / First Hop IP Addresses along with their respective subnet mask, information about DNS Nameservers.
* The necessary DNS information is obtained from “/etc/resolv.conf” file. This file contains all required information such as “Name”, “Alias”, “Record Type” etc., to uniquely identify DNS information.

Note: We assume in this project that the device is connected on a single network interface. Even if the device has multiple interfaces, we strictly assume that the device is interacting only on a single Interface.

Capturing IP Layer Error Reporting Protocol:

These parameters are captured using Scapy. Scapy facilitates the interaction with packets sent and received along with the ability to capture these packets to analyze it further. Used “ICMP” class of Scapy to dissect the ICMP Layer of the packet. Retrieved the IP information of the machine requesting the details and the response, along with the ICMP Type and ICMP code to understand if the packets correspond to Echo (Request / Reply), Destination Unreachable, TTL Expired.

After dissecting ICMP packets, we collected information such as Source IP, Destination IP, ICMP packet type and ICMP packet code. These four parameters are useful to understand the type of communication going on.

Capturing ARP Cache and ARP Table:

Capturing the ARP cache is done by making use of the file named “/proc/net/arp”, since ARP cache is not just enough to conclude if there was an attempt to commit ARP poisoning, we need to take into consideration the ARP Table. ARP Table is obtained by crafting ARP request packets for the specific sub-network in which the device is present. Scapy can be used to craft these messages and capture the response to the corresponding request.

For every record of ARP Cache, we collect the following information. We collect IP Address, Hardware Type, Flags, Hardware Address and Mask.

Capturing DNS Cache:

As opposed to windows, UNIX-based systems don’t store the DNS Cache periodically. In-order to achieve the DNS query and responses, we will need to capture live queries and associate them with their responses. Hence, we will not be able to capture DNS Cache periodically, rather on the go. The DNS queries and responses are captured on the fly and stored in a file and then read periodically after a given interval. This file is then flushed and starts to populate once again. We used “DNS” module of Scapy to dissect DNS packets, which supports IPv4 only.

DNS record has various different sub-sections namely, QD (Query definition), NS (Name server), AN (Answers), AR (Additional record). Every subsection has data organised in similar way. In this task, we only retrieve information for three sub-sections, QD, NS, and AN. We use QD.count(), NS.count() etc., to find the number of records within a sub-section.

For QD subsection we store Query Name (qName) and Query Type (qType). For AN subsection we collect Requested Name (rrName), Response Data (rData), Time to live (ttl) and Answer Type (type). Within NS, we extract Requested Name (rrName), Response Name (rName), Type (type), Serial number (serial), number of retries (retry) and Expiration Time (expiry).

Capturing TCP Parameters:

These packets are also captured using Scapy and filtered out using its filtering parameters. Used “IP” and “TCP” Class of Scapy to gain information about the TCP Segment of the packet. From these Packets, mainly four parameters are extracted, “SourceIP”, “SourcePort”, “DestinationIP”, “DestinationPort”

Capturing SSL Parameters:

Since SSL/TLS packets are built upon IP Packets, we also need to store the SourceIP and Destination IP. Used “TLS” layer of Scapy to disassemble the packet to retrieve information such as Server IP address, Server’s digital certificate, Agreed upon Cryptographic algorithm (between the client and the Server) and Server's public Key.

Capturing Cookies:

Cookie history of a browser is generally stored as an SQL file. We need to determine the browsers on the device. We look for possibility of finding a browser path and traverse it to find ‘cookies.sqlite’ or ‘Cookies’ file. Used SQL Commands to retrieve and store the data, based on the cookie timings so that we can capture the cookies periodically.

Capturing Browser History:

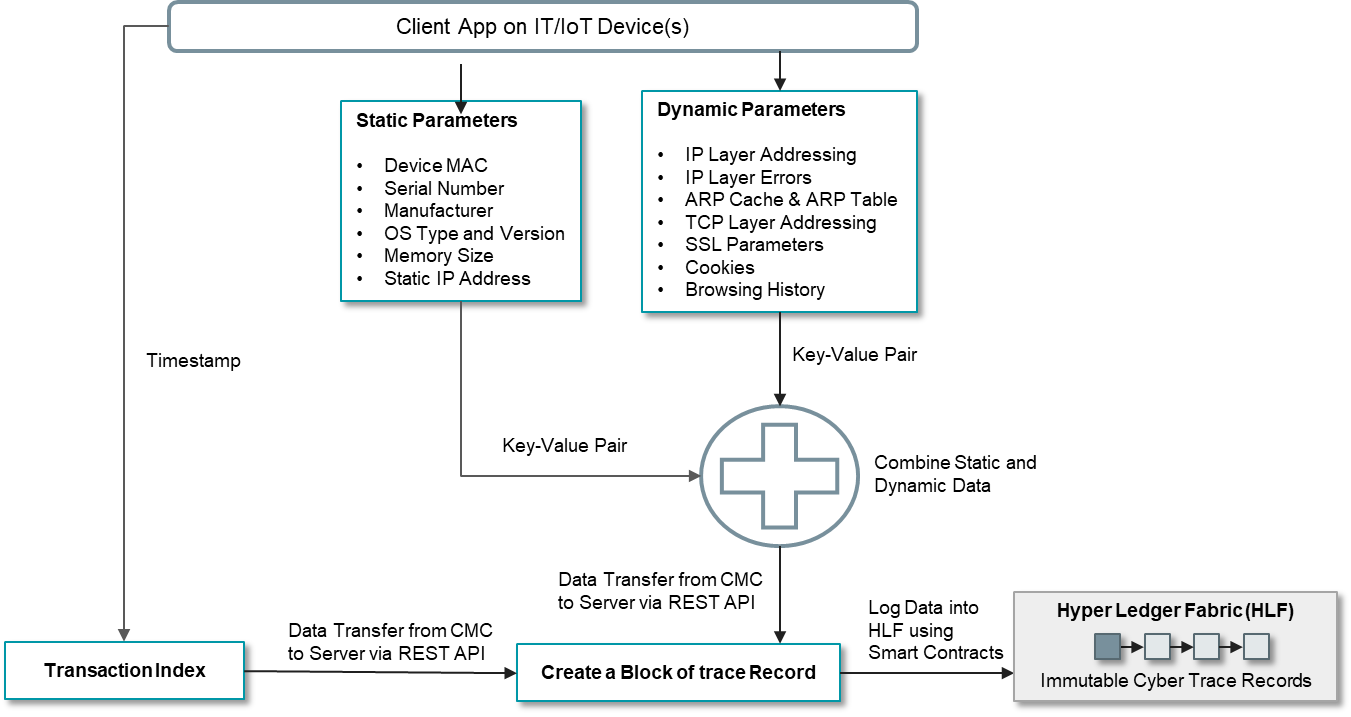
Browser history is generally stored as an SQL file, like the cookies. Hence, we use the similar technique used to retrieve cookie files. Used SQL Commands to retrieve and store the data, based on the visited time so that we can capture the browsing history periodically.

Note: As mentioned earlier, we need root/administrator access to capture most of the dynamic information. All the necessary dynamic information that we intend to extract, if residing within a network packet, needs root access to collect and read such information.

We used “Scapy”, an open-source packet manipulation tool to extract information from network packets. Scapy is capable of decoding packets of a wide variety of protocols.

1. Digital Twin

The static and dynamic information is collected from the devices/network based on a regular interval, transferred to the server using REST API, and stored on the server to create a Digital Twin representation of the IT/IoT devices. To make it easily accessible and readable, this digital twin information is stored in JSON format. The digital twin construction flow is shown below.



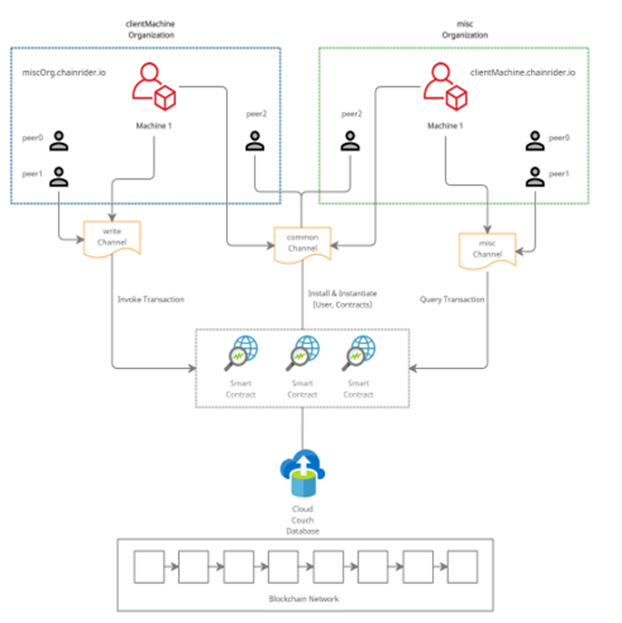
***Figure 1 – Digital Twin Construction Flow***

The digital twin containing the static and dynamic information of devices is collected from a script. Furthermore, every information is timestamped.

While querying back the response from the smart contract to get back the data for a given timestamp, we can request the response to be in JSON format. The script updates the digital twin information in regular interval on the Hyperledger fabric.

1. Hyperledger Fabric Architecture

A three-peered double organizational permissioned blockchain used in this project, as shown in Figure 1 below.

**

***Figure 2 – Hyperledger Fabric (HLF) Architecture***

The Hyperledger Fabric consists of two organisations namely clientMachine and miscOrganization. ClientMachine is used to send in the data from the devices to the Hyperledger Fabric, and miscOrganization is used for other miscellaneous tasks such as reading the data etc.

A smart contract has been employed into Hyperledger Fabric, which is able to record a transaction onto the blockchain and also read the records based on given uniquely identifiable key (timestamp). We have used CouchDB to store the digital information of a device.

1. Pushing Data from Device CMC to the HLF on the Server

REST API is used to push / log the static and dynamic parameters captured by the CMC into the HLF on the Server on the cloud. The following steps were executed:

**Step 1:** Generated a “permissioned blockchain” on the ChainRider platform, and deployed it on the cloud.

**Design Description:**

Deployed a 2-Organization three-peered permissioned blockchain to cloud through the Chainrider.io platform.

**Step 2:**  Developed Smart Contracts / Using Contracts from the ChainRider platform Marketplace library.

**Design Description:**

Once the network is deployed, we developed custom smart contracts to communicate with the blockchain.

**Stage 3:** Communication through REST API

**Design Description:**

For the CMC to interact with the permissioned blockchain fabric on the server, we used the Rest API module provided by ChainRider platform, where:

* Every information within a transaction has a timestamp associated with it.
* Every transaction/information sent within a certain interval has a timestamp at which the transaction was created and sent to the blockchain.

Therefore, the data collected from the IT/IoT devices (2 devices were used) are basically time-stamped, and logged in immutable trace records on the hyper ledger fabric, specifically,

* Records are tagged with device-specific IDs and with timestamps of transactions.
* Digital twins of respective IoT devices are logged into Hyperledger Fabric network in periodic intervals, programmable as needed.
* Structure of digital twins is such that the data about a device with a particular IP address can be traced, thus providing immutable traces to track normal cyber activities and cyberattack activities.
* Two organization with three peers and
* Every transaction on the hyperledger network is tagged with its time of creation.
* Every communication in and out of the device is labeled with time of communication (packet arrival) and enough data to recreate the whole entirety of the IoT device for analysis.

1. Additional Hyperledger Data

The Mathematical team requested additional hyperledger fabric specific parameters to be also provided, so a special tool called “Caliper” (Benchmarking Software) on the local network to extract hyperledger fabric specific parameters.

Used Caliper, a benchmarking software to capture the requested performance parameters of the hyperledger fabric, and collected the following hyperledger fabric specific parameters from the machine in the cloud:

1. Mean/Average Time of transactions
2. Throughput
3. Transaction latency
4. CPU Utilization
5. LedgerWrite Time
6. Concluding Remarks for Task 1

The objective of this Task 1 effort is to develop a light-weight cyber monitoring client (CMC) on a typical IT/IoT device to capture key static and dynamic parameters, and transfer the collected data to a Hyperledger fabric (HLF) on a server on the cloud. Additional hyperledger-specific data was requested by the Task 2 team, and that request was met by using the Caliper tools.

The collective data is to be used by a mathematical model, developed by Task 2, for the purpose of assessing cyber risks in blockchain-based systems.

1. Future Work

In this project, for Task 1, our focus has been on capturing the IT/IoT device data (static and dynamic), creating the corresponding digital twin on the server, and logging the data in immutable records on a blockchain network so permanent trace records are preserved for analysis.

While we were conducting the research and development of this project, the US Department of Commerce, National Institute of Standard and Technology (NIST), issued a draft NIST Special Publication 800-213, on December 2020, titled: “*IoT Device Cybersecurity Guidance for the Federal Government, Establishing IoT Device Cybersecurity Requirements*”.

In comparison with our project, it is noticed that the NIST took a functional view of the IT device cybersecurity issue. Specifically, the NIST requirements focused on the device ability to perform certain functions, including: ability to monitor some specific device actions (based on the device type), or monitor & control downloading applications, or ability to execute cryptographic mechanisms, or ability to purge/clean/delete some data, ability to lock device, ability to create or disable an account, ability to establish remote access, ability to identify organization, ability to monitor communications and events, ability to generate alerts, ability to track user logging, ability to synchronize with verified time source,  ability to separate IoT device processes into separate execution domains, ability to enforce traffic flow policies, ability to manage memory address space assigned to processes.

It is recommended that future work combines the functional-based requirements with the data-centric requirements we provided in the project to create a more complete set of functional requirements, immutable digital twin data, and analysis tools to monitor IoT device cyber operations, detect cyberattacks in progress, and also predict the potential of cyberattacks based on the level of risks. We stand ready to proceed with the next phase of this work.