J-PARC MLF BL11 でのスクリーンショットによる測定状況確認システム

総合科学研究機構（CROSS）中性子科学センター，
日本原子力研究開発機構(JAEA) J-PARC センター[A]
岡崎 伸生，服部 高典[A]

Monitoring System of Experiments with Screenshots at BL11 in J-PARC MLF

Neutron Science and Technology Center, Comprehensive Research Organization for Science and Society (CROSS),
J-PARC Center, Japan Atomic Energy Agency (JAEA)[A]
Nobuo Okazaki, Takanori Hattori[A]

e-mail: n_okazaki@cross.or.jp
Published online: 22 December 2023

要約

大強度陽子加速器施設(J-PARC)の物質・生命科学実験施設(MLF)に設置されているビームライン BL11 PLANET では、ユーザーが J-PARC エリア外から実験状況を確認することができなかった。この問題を解決するために、装置制御用計算機(PC)のスクリーンショットを外部端末から取得できる仕組み(*RemoShot*)を導入した。この仕組みはスクリーンショット制御サーバー、および MLF 側 PC とクラウドから構成されており、一連の動作は以下のようになる。1) ユーザーは装置制御用 PC 上でスクリーンショット制御サーバーを起動する、2) スクリーンショットのリクエストを受けて、スクリーンショット制御サーバーが MLF 側 PC(測定制御用 PC)に Secure Shell(SSH)認証でアクセスし、スクリーンショットを取得、3) 取得したスクリーンショットをクラウドストレージに格納、4) ユーザーは認証後、クラウドのスクリーンショットを確認、というように動作する。*RemoShot* を用いることによって、ユーザーはスマートフォンなどを使い、遠隔地からでも実験状況を確認できるようになり、測定エラーへの対応、想定外のビーム停止によるシーケンスの遅延に対し、いち早く実験計画の見直しが行えるようになった。あわせて、セキュリティや運用コストに関する考察を行った。

Abstract

There has been no method for users to monitor the current status of the experiments at BL11(PLANET) from outside. To solve this, a monitoring system, named *RemoShot*, has been implemented to capture remotely the screenshots of the control PC. The *RemoShot* consists of a screenshot control server, a control PC in the beamline, and cloud storage. It operates in the following sequence: 1) Start a screenshot control server on the PC at beamline, 2) The screenshot control server accesses the MLF-side PC (measurement control PC) via SSH authentication and captures a screenshot, 3) the captured screenshot is stored in cloud storage, and 4) after authentication, users can see the screenshot in the cloud. This system allows users to check the status of their experiments from anywhere using smartphones and other devices, and to quickly revise the experimental plan in case of measurement issuers or sequence delays due to unexpected beam off. We also discussed security and operating costs of this system.

Keywords

BL11 PLANET、リモート監視、クラウド, Remote monitoring、Cloud computing

1. 緒言

茨城県東海村に位置する大強度陽子加速器施設(J-PARC*[1]) 物質・生命科学実験施設(MLF*[2])では、ユーザーから遠隔地から実験等を行える環境が望まれており、各ビームライン(BL)においてはそれぞれのニーズに合わせた遠隔化が様々な形で行われている。ここでいう遠隔地とは J-PARC イントラネットワーク(JLAN)に接続されていないネットワーク環境を意味する。MLF において、遠隔地から実験状況を確認するためには実験制御ソフトウェアフレームワーク IROHA2 による仕組み[1]が広く使われているが、今回の開発対象である超高圧中性子回折装置 BL11 PLANET[2]には IROHA2 が導入されていない。しかし、BL11 のユーザーからも実験の進行状況を遠隔地から確認できるような仕組みが欲しいとの要望があり、遠隔地から制御用 PC のスクリーンショット*[3]を閲覧するシステム(本システム)を開発することになった。

一般的に、イントラネットワークはインターネットからの直接接続は厳しく制限されている。J-PARC のイントラネットワークである JLAN も同様である。その制限内において、JLAN に外部から接続する仕組み(SSL-VPN)が利用可能である。ユーザーは装置責任者を通して申請することで利用可能であるが、接続手順が煩雑であることや帯域の制限、ユーザーの環境(海外からは接続できないことがある)、などの問題があり、広くユーザーに提供できる状況にない。

このような現状を考慮して、本システムは装置制御 PC のスクリーンショットをクラウド上にアップロードし、ユーザーはクラウド上に保存された装置制御 PC のスクリーンショットを Web ブラウザで閲覧することで実験の状況を確認するという方法を採ることにした(図 1)。JLAN に接続せずにクラウド上のイメージを閲覧するという方法を使った場合、以下のメリットが考えられる。1) JLAN に直接接続しないため、不正なアクセスがあった場合でもクラウド上に被害が限定される。2) 通信帯域が不足するなどの事態を避けることができる。3) 国際情勢等に影響されにくい。などといったことが挙げられる。一方、以下の点では注意が必要である。1) 世界中のどこからでもアクセスが可能であるために、認証等によりユーザー制限を確実に行う必要がある。2) 不正アクセスをされた場合に情報が流出する恐れがある。3) ユーザー管理方法によっては当該ビームタイム以外のユーザーがスクリーンショットにアクセスできてしまう。以上の状況を踏まえて、BL11 において外部から測定制御 PC のスクリーンショットを Web ブラウザ経由で確認可能なシステム(*RemoShot*)を構築した。

*[1] Japan Proton Accelerator Research Complex
*[2] Material and Life science Experimental Facility
*[3] ここでは、対象 PC の画面全体を画像として取り込むことを指す

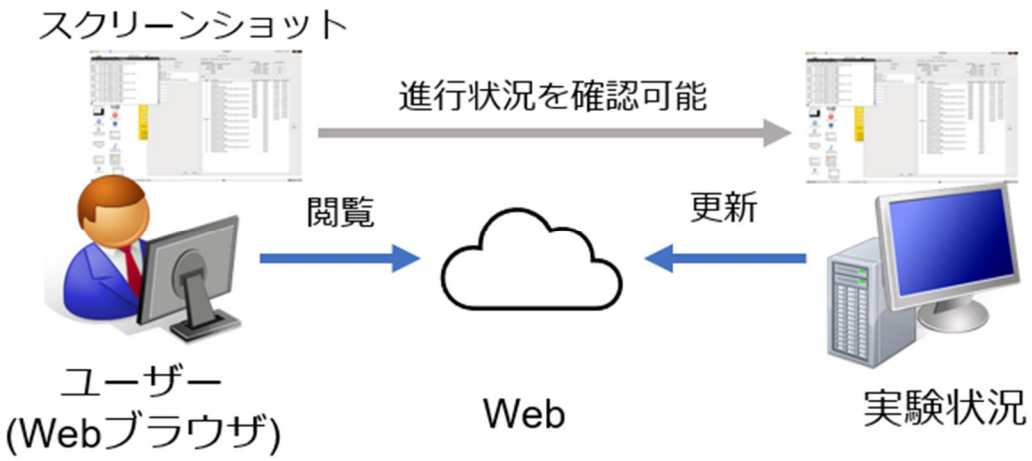

図 1 スクリーンショットで実験状況を確認する

2. 全体構成

本システムは、1) 監視対象 PC、2) クラウドによる構成、3) 制御ノード、4) 認証ノードの 4 つのサブシステムで構成されている(図 2)。

2.1 監視対象 PC

スクリーンショットによる監視を行う対象の PC。BL11 の場合は実験装置の制御コマンドシーケンスを管理、実行する Experimental Scheduler(ES)[3]が起動している PC が対象となる。監視対象 PC 上で後述する認証ノードを起動する。

2.2 クラウド構成

パブリッククラウドである Amazon Web Services(AWS)[4]を利用し構成されている。使用しているサービスおよび機能は表 1 の通り。クラウド上で本システムを構成しているサービスは他のネットワークから独立しており、これらのサービスのみで閉じた構成となっている。

2.3 制御ノード

セッション認証やスクリーンショット取得、取得した画像のアップロードなどを行う。信頼できる内部ネットワークに配置される常時稼働サーバー。メッセージキューである Amazon Simple Queue Service (SQS)を介したメッセージ転送により動作を行う。AWS の認証情報を保持するため、システム管理者のみがアクセス可能な信頼できるネットワークに配置する。クラウド、後述の認証ノード、監視対象とする測定制御 PC へのアクセスが可能なネットワークに配置する。稼働中のシステムは CROSS 内ネットワークに配置した。

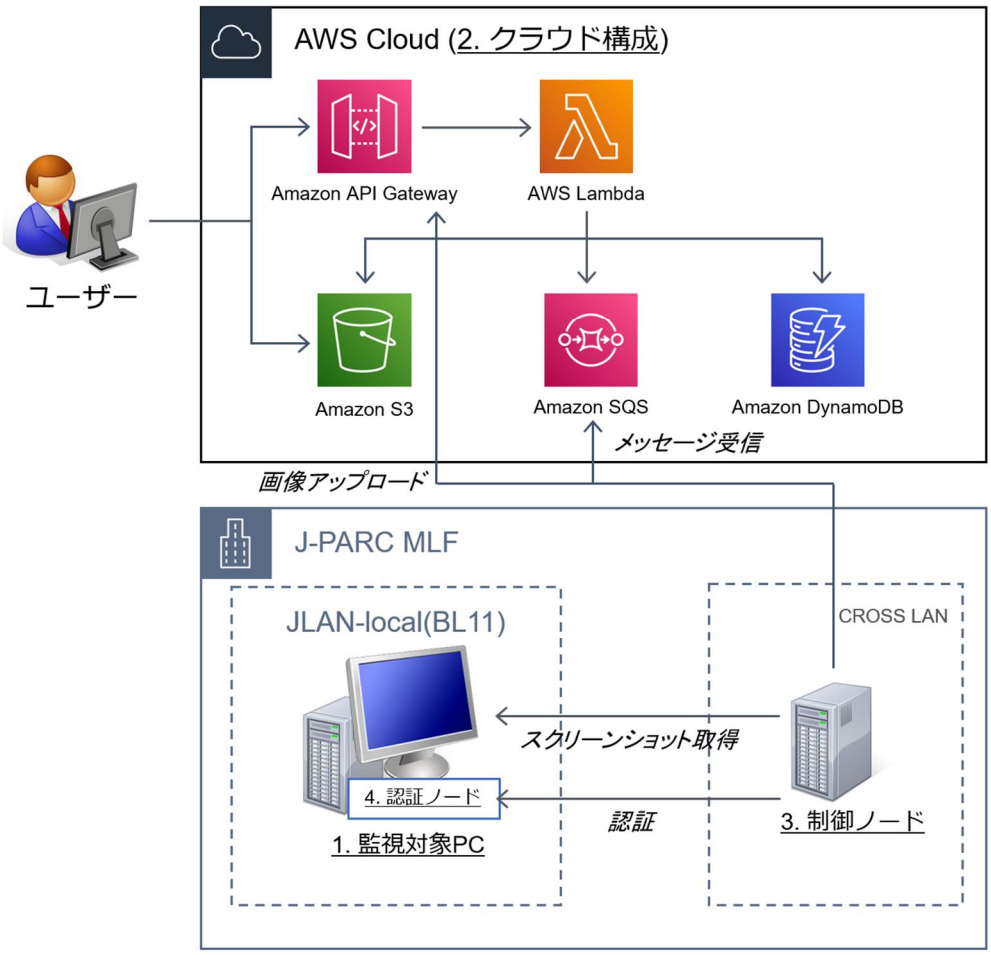

図 2 構成図

表 1 AWS で利用しているサービス

サービス名	機能
AWS Lambda[5]	構成しているサービスを制御する
Amazon API Gateway[6]	Web アクセスを受け入れる(Web アクセスエンドポイント)
Amazon S3[7]	スクリーンショットを格納するストレージ
Amazon SQS[8]	シンプルなタスクキュー
Amazon DynamoDB[9]	認証データ等を格納する

2.4 認証ノード

ユーザーの識別用セッションを確認するためのサーバー。先に述べた監視対象 PC にスクリプトを配置し、スクリーンショット機能を利用する期間起動する。制御ノードはセッション確認を認証ノードに対して行い、認証が通ったときのみスクリーンショットを取得する。監視対象 PC は管理者以外のユーザーが使用するため、セキュリティリスクとなる認証情報は一切持たない。

3. 動作フロー

ユーザーがスクリーンショットを閲覧するまでの流れは以下のようになる。

1. ユーザーは BL の監視対象 PC 上で認証ノードをコマンドラインで起動し、アクセス用 URL を得る。この URL は認証ノードを起動するたびに変更される(URL はセッションキーを含むため機密情報として扱う)。この URL はスマートフォンで取得しやすいように QR コードとしても出力される

2. ユーザーはBL外からスクリーンショットを閲覧するためにWebブラウザを用いてアクセス用URLを開く(インターネット経由)
3. Webページが開かれると、API Gateway→Lambda→SQSという経路で、制御ノードにスクリーンショット取得コマンドが通知される
4. 制御ノードがセッション情報を元に認証ノードで認証確認を行う。問い合わせが成功したら制御ノードはSSHを経由し、監視対象PCのスクリーンショットを取得する。取得されたイメージはAPI Gateway→Lambda経由でS3にアップロードされる
5. ユーザーが開いているWebページは取得成功の応答を得た後、S3からスクリーンショットを取得し、表示される
6. 定期的(設定によるが1-5分間隔)にスクリーンショットが更新される
7. ユーザーが停止するか、規定の時間(通常は1週間)が経つと認証ノードが終了し、アクセス用URLが破棄される。このURLでのスクリーンショットの取得が終了する

4. 詳細な動作

4.1 認証ノードの起動

認証ノードが起動し、セッションの登録を行う。ベースURLを基準にエンドポイント*4を決定し、クラウド構成中のAPI Gateway経由でWeb APIにアクセスする。以下の流れでセッションを確立する(図3)。

1. ユーザーが認証ノードスクリプトを起動する
2. 認証ノードはセッション開始要求をWeb APIに送る。この際に検証用のリクエストトークン*5(REQ_TOKEN)を送る。リクエストボディに{req: REQ_TOKEN}をJSON*6形式

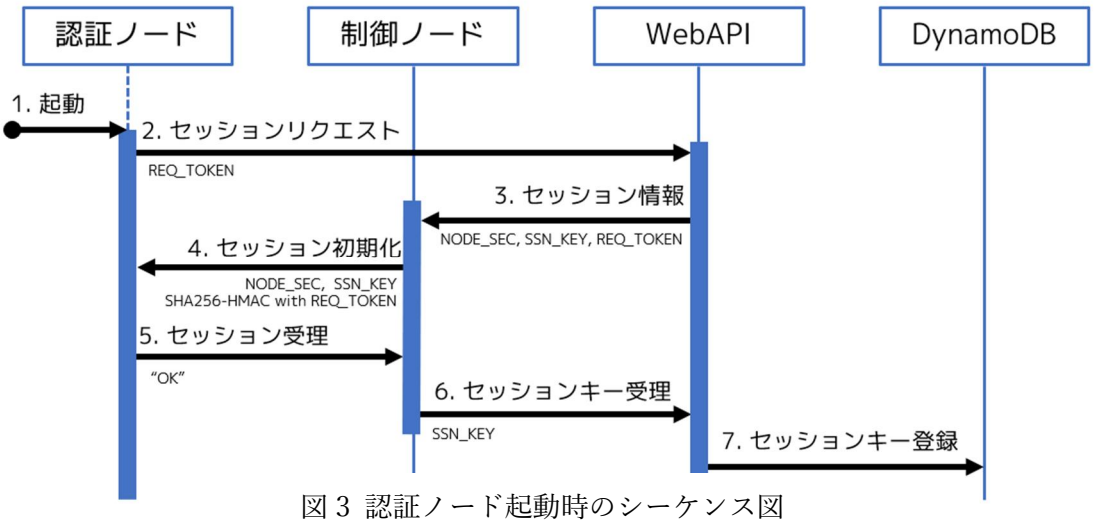

図3 認証ノード起動時のシーケンス図

でセットし送る。REQ_TOKENはBase64*7でエンコードされた64バイトのランダムなバイト列である

*4 Web APIのアクセス先URL

*5 ここではある長さの文字列

*6 JavaScript Object Notation。JavaScriptのオブジェクト表記法がベースとなっている

*7 バイナリを64種類の印字可能文字列に変換する方式

3. Web API はセッション情報を発行する。監視対象 PC ごとの固定値であるノードシークレット(NODE_SEC)、セッション維持用のセッションキー(SSN_KEY)を生成し、認証ノードから送られてきた REQ_TOKEN を制御ノードに通知する。NODE_SEC は Base64 でエンコードされた 32 バイトのランダムなバイト列で、SSN_KEY は Base32*[8]でエンコードした 5 バイトのランダムなバイト列である
4. 制御ノードは認証ノードに接続し、セッションを初期化する。認証ノードに対して、NODE_SEC、SSN_KEY を通知するが、このとき検証のため REQ_TOKEN を用いて、SHA256-HMAC*[9]により署名する
5. 認証ノードは送られてきた SHA256-HMAC 署名を検証し、問題なければ文字列"OK"を制御ノードに返す。合わせてユーザーに SSN_KEY からアクセス用 URL を生成し、提示する
6. 制御ノードは、認証ノードの検証に成功したら SSN_KEY を Web API に登録する
7. Web API は制御ノードから送られてきた SSN_KEY を DynamoDB に登録する

これにより新たにセッションが発行され、古い SSN_KEY は破棄される。また、ここで表示される URL は起動した認証ノードが終了するまで使用可能であり、認証情報と同等の意味を持つため、使用するユーザー以外には公開してはならない。

4.2 Web ブラウザを経由した画像の閲覧

上に示した手順で得られたアクセス用 URL はインターネットからアクセス可能であり、Web ブラウザで開くことでスクリーンショットの取得をリクエストし、閲覧することができる。スクリーンショットの最小取得間隔(秒)は設定により変更可能である(通常 60～300 秒)。

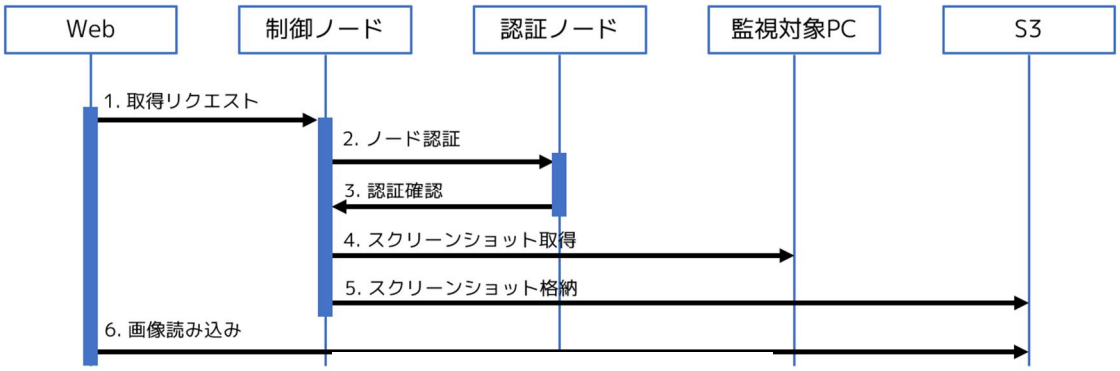

図 4 画像閲覧時のシーケンス図

1. アクセス用 URL をブラウザで開くと制御ノードにスクリーンショット取得リクエストが発行される。直接制御ノードには接続できないため、実際のリクエストは API Gateway を通じて Lambda から SQS にメッセージが送信され、SQS を経由して制御ノードにメッセージが届く
2. 制御ノードは認証ノードに対して、送られてきたセッション情報を確認する
3. セッション情報が正しければ確認が成功する
4. 制御ノードは監視対象 PC のスクリーンショットを取得する(SSH 経由)
5. 制御ノードは取得した画像を S3 にアップロードする
6. Web ブラウザで S3 上の画像が表示される

*[8] Base64 と同様だが 32 種類の文字を用いて表現した形式

*[9] Hash-based Message Authentication Code。ハッシュベースのメッセージ認証符号であり、ハッシュ値を SHA256(256 ビット Secure Hash Algorithm)で求める

4.3 ソフトウェア構成

本システムは以下の環境で構築した。

- CentOS 7.9
- Docker 20.10.12
- Python 3.6.15

本番環境は Docker コンテナとして稼働している。Docker のベースイメージとして alpine:3.15.4 を使用した。

5. 実行方法

5.1 制御ノードの起動

信頼できるサーバー上で Docker コンテナとして起動する。制御ノードは常時起動しておく。

```
$ docker run -d ¥
  --restart always ¥
  --name remoteshot-ctrl ¥
  -v $(pwd)/dot.ssh:/home/remoteshot/.ssh ¥
  -e RS_SECRET_KEY=EXAMPLESECRETKEY ¥
  remoteshot https://example.com
```

5.2 認証ノードの起動

監視対象 PC で認証ノードを起動する。これは、監視が必要な時に、ユーザー自身で起動する。起動している時は https://example.com/view/nodename/SSN_KEY にアクセス可能。

```
$ authnode.py -d https://example.com
2023-04-12 18:54:21 INFO Start Authentication Server on localhost:31508...
2023-04-12 18:54:22 INFO Node initialized.
URL     : https://example.com/view/nodename/SSN_KEY
NOTICE: This node will be finished automatically at 2023-04-19 18:54:21
```

5.3 Web ブラウザでの閲覧

認証ノード起動時に得られる URL に Web ブラウザでアクセスすることで、スクリーンショットを閲覧することができる(図 5)。

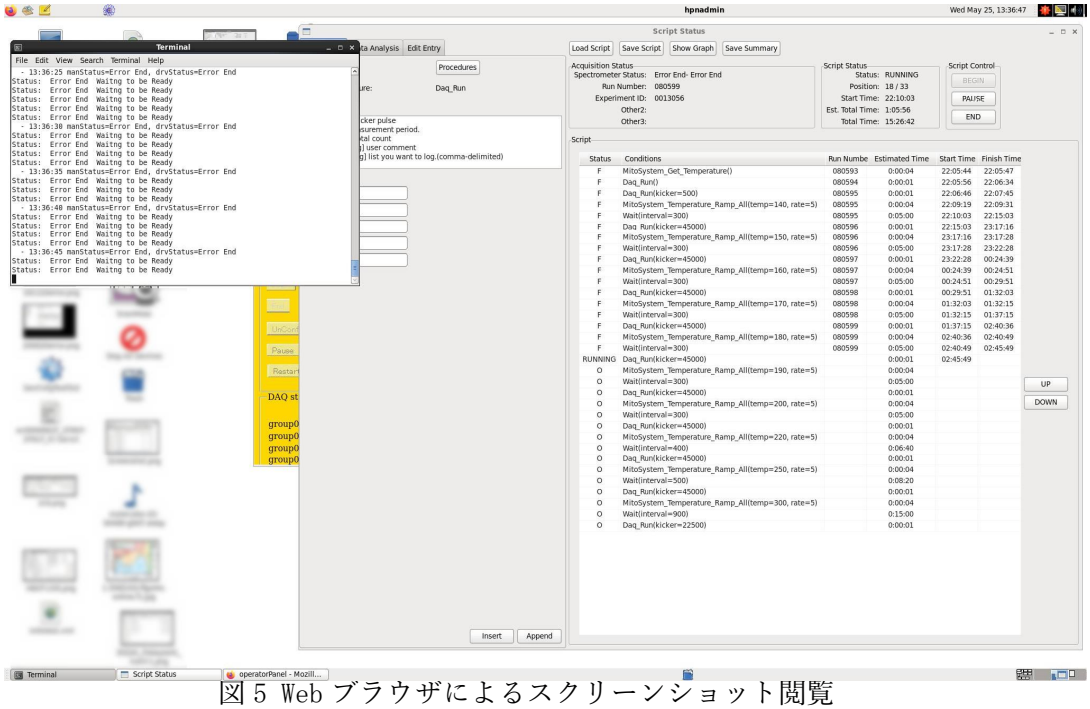

図 5 Web ブラウザによるスクリーンショット閲覧

5.4 認証ノードの終了

監視が不要になったときには認証ノードを終了する。

```
$ authnode.py -a
Node finished.
```

6. 考察

6.1 セキュリティ

本システムで考えられるセキュリティ上のリスクは以下のようなものが考えられる。1) セッションキーの総当たり攻撃。2) 認証ノードの偽装。3) 制御ノードの乗っ取り。これらについて実現可能性の検証を行う。

6.2 セッションキーの総当たり攻撃(ブルートフォース攻撃)

セッションキーは 5 バイト(=40 ビット)で表現されているため、$2^{40} = 1.1 \times 10^{12} =$ 約 1 兆通りのセッションキーが存在する。このキーに対して総当たり攻撃を行うことを考える。実行サービスである Lambda は最大 1,000 セッション[10]を同時に処理することができる(仕様上の上限)。要求から応答までの時間を 1 秒とすると、1 秒につき 1,000 通りのキーを試すことができる。従って、最大負荷で試行した場合、すべてのキーを試すためには、最大で 1 兆/1,000 = 10 億秒 = 約 31 年かかる。セッションの最長期限は 1〜2 週間(設定による)のため、総当たり攻撃にかかる時間よりも十分に短い。今後、短時間で複数回以上無効なセッションキーが入力された場合はセッションを停止するなどの対策を追加することでさらなるセキュリティの向上を目指すことが可能である。

6.3 認証ノードの偽装

認証ノードを偽装することができれば、スクリーンショット取得の際にセッション(SSN_KEY)のチェックをスキップすることができる。その場合、秘密情報である SSN_KEY を知ることなくスクリーンショットを取得することができる可能性がある。それを実行するためには、REQ_TOKEN を Web API 経由で制御ノードに届け、制御ノードから REQ_TOKEN で署名された情報を受け入れる必要がある。認証ノ

ードは監視対象 PC 上で実現することが想定されているため、認証ノードを偽装する必要があるのは監視対象 PC にアクセスできない場合なので、監視対象 PC 以外の端末で実行することになる。認証ノードを偽装し、制御ノードに”OK”を返すためには制御ノードからセッション情報を受け取る必要があるが、制御ノードからのセッション初期化(4)はノード名によりアクセス先が固定されており、監視対象 PC 以外で受けることはできない。このことにより認証ノードの偽装を防ぐことができる。

6.4 制御ノードの乗っ取り

制御ノードはユーザーが利用できるネットワークとは別のネットワークに配置され、ユーザーはアクセスすることができない(図 2)。また、インターネットからの接続もできないため、ネットワーク経由による攻撃は原理上不可能であるため、乗っ取りはできないとみなしてよい。

6.5 運用ついて

本システムは、実験状況を監視するという機能のため、継続的な運用が必要となる。MLF サイト内で運用するシステムとクラウド環境に分けて考察する。

6.6 MLF サイト内での運用

MLF サイト内(オンプレミス)で運用するシステムは監視対象 PC および制御ノードが該当する。監視対象 PC は BL の装置と一体で運用されている。制御ノードは独立して稼働しているため、運用・保守が必要となる。

6.7 クラウド環境

クラウドに構築したシステムは、従量制により課金される(AWS の場合)。本システムのクラウド部分はいわゆるサーバーレスで構築されており、課金されるのはプログラム等が動作した時間のみであるため、一般的にサーバーを起動しておくよりも安価かつ保守が最低限で済むという特徴がある。本システムの場合は、API Gateway、Lambda、SQS は数万回以上の呼び出しが想定されているため、MLF の規模の場合、料金はほぼかからないと考えてよい(1 円/月以下)。DynamoDB や S3 では保持しておくデータ容量が存在するが、KB オーダーであるため、数円～数十円と予想される。使用状況によるが、動作に関わる課金は数十円以下/月と考えている。

参考文献

[1] J-PARC MLF IROHA2 ポータルサイト https://mlfinfo.jp/groups/comp/ja/iroha2.html
[2] J-PARC MLF BL11 PLANET https://mlfinfo.jp/ja/bl11/
[3] J-PARC/MLF 計算環境ソフトウェアフレームワーク IROHA の現状 https://jopss.jaea.go.jp/search/servlet/search?5029891
[4] Amazon Web Services https://aws.amazon.com/jp/
[5] AWS Lambda https://aws.amazon.com/jp/lambda/
[6] Amazon API Gateway https://aws.amazon.com/jp/api-gateway/
[7] Amazon S3 https://aws.amazon.com/jp/s3/
[8] Amazon SQS https://aws.amazon.com/jp/sqs/
[9] Amazon DynamoDB https://aws.amazon.com/jp/dynamodb/
[10] Lambda 関数のスケーリング https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/lambda-concurrency.html