自動巡回ツール

MBSD＿Web＿CSS

作成報告書

**【目次】**

**1.前談　P4～P6**

1.1　どのような機能を目指したか P4

1.2 　目標・目的 P4

1.3 　参考させてもらったもの　 P5~6

**２.クローラーの機能　P7~P11**

2.1　クローラーにより実行可能なこと P7

2.2 抽出可能な診断対象 P7~8

2.3 規模の抽出 P8

2.4 作成した自動巡回ツール P8~10

2.5 アピールポイント P10~11

**3.クローラーの動作条件 P12**

3.1 動作条件 P12

3.2 実行ブラウザの推奨環境 P12

**4.実行例 P13**

4.1 実行例 P13

**5.関数 P14~P19**

5.1関数について

5.1.1　MBSDmain\_HTML.py P14

5.1.2　GetHTML.py P14

5.1.3　GetTAG.py P14

5.1.4　INPUT.py P15

5.1.5　MBSD＿SEARCH.py P15

5.1.6　mbsd\_title.py P15

5.1.7　url\_Getter.py P16  
5.1.8　url\_Host.py P16

5.1.9　url\_Parameter.py P17

5.1.10　is\_image\_url.py P17

5.1.11　url\_Visited.py P18

5.1.12　url\_Domain\_Select\_Getter.py P18~19

5.1.13　url\_Crawler .py P19

5.1.14　danger\_url.py P19

**6.考察・まとめ　P20~P21**

6.1　望月　紋乃丞 P20

6.2　望月　健翔 P20

6.3　法村　寿史 P21

6.4　大石　翔亜 P21

**1.前談**

　1.1どのような機能を目指したか

　今回作成したツールは、大会を主催しているMBSDから提示されて　　　　　　　　　いる競技概要に則り自動巡回ツールを作成した。

　ツールに搭載する為に考えた機能としては、複数あるので個々に挙　　　　　　　　　　　　　　　げる。

　・診断対象先であるWebサイトの入力フォーム数

　・Webサイト内に存在するURLの検出及び出力

　・診断対象のパラメータの出力

　・Webサイトのコンテンツ内に含まれているMBSD｛××××｝の　　　　　　　　　　　　キーワードの出力

　・診断URLにページタイトルが含まれる場合に出力をする

　これら機能に加えて実行時にできるだけ処理時間を短くし実行結果　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　をわかりやすく表示させる機能を持ち合わせた自動巡回ツールの作　成を目指した。

　1.2目標・目的

　Webサイトを網羅的に検査をしできるだけ脆弱性を検出でき、実際　　　　　　　　　　　　　　に運用が出来るような自動巡回ツールを作ることを目標として作成　を開始した。

　1.3参考にさせてもらったもの

Pythonプログラミング VTuber サプー

<https://www.youtube.com/@pythonvtuber9917/videos>

【Pythonプログラミング入門】メモ化で高速化！cacheデコレータを使ってみよう！〜初心者向け〜

<https://youtu.be/lRaSMlHY3aY?feature=shared>

【Pythonプログラミング入門】自作モジュールの使い方を解説！〜VTuberと学習〜 【初心者向け】

<https://youtu.be/X3uBMY3JQqM?feature=shared>

HTMLでチェックボックスを表示する方法！基本的な作り方と使い方を解説　byウェブカツ #初心者 - Qiita

<https://qiita.com/kazukichi/items/1af73244df0e67137531>

【バッチファイル】ログの出力方法と日時取得／ログファイル名に日時取得 #bat – Qiita

<https://qiita.com/pekosyu/items/a2d416f9f2afe9c40066>

ローカルにpythonのCGI環境を構築　ほどよく解説しながらスピード重視で説明するよ

<https://jimaru.blog/programming/python/local-cgi-python/>

HTMLでformタグのaction属性に複数の送信先を指定する方法を現役エンジニアが解説【初心者向け】

<https://magazine.techacademy.jp/magazine/32105>

Webサーバーで動くPythonアプリ 【Windows 版】

<https://irohaplat.com/windows-python-http-server-calculator-application/>

CSSのコピペだけ！おしゃれな見出しのデザイン例まとめ6

<https://saruwakakun.com/html-css/reference/h-design>

図解！PythonでWEB スクレイピングを始めよう！(サンプルコード付きチュートリアル)

<https://ai-inter1.com/python-webscraping/#st-toc-h-2>

【HTML】フォーム作成の基本！formとinputの使い方

<https://creive.me/archives/13526/>

スクレイピング練習場（ベータ）

<https://scraping-training.vercel.app/>

図解！PythonのRequestsを徹底解説！(インストール・使い方)

<https://ai-inter1.com/python-requests/>

【コピペで完成】HTML・CSSのみで作る棒グラフのデザイン2選【アニメーション対応】 (pote-chil.com)

<https://pote-chil.com/html-maker/bar-chart>

table・tr・th・tdタグ | HTMLでの正しい表の構造と使い方・作り方を徹底解説 | Webのいろは (webnoiroha.net)

<https://www.webnoiroha.net/html-table/>

2.**クローラーの機能**

2.1 クローラーにより実行可能なこと

・検査対象のWebサイトの脆弱性の検査

・Webサイトの規模の調査

・ローカル環境下によるツールの実行

・URL・パラメータ・ページタイトル・性能審査用のキーワードを表にして出力

・ドメインの指定

･診断対象個数のそれぞれの合計割合をグラフで表示

・検査するURLの数の制限

2.2抽出可能な診断対象

・HTML内に含まれる「input type=”text”」(入力フォーム)

検出可能な脆弱性：クロスサイトスクリプティング(XSS)

・HTML内に含まれる「input type=”file”」

検出可能な脆弱性：不正ファイルのアップロード

・HTML内に含まれる「input type=”password”」

検出可能な脆弱性：パスワードセキュリティ

・HTML内に含まれる「input type=”hidden”」

検出可能な脆弱性：隠しフィールドに保存されているデータの悪用

・HTM内に含まれる「form」  
 検出可能な脆弱性：クロスサイトリクエストフォージェリ(CSRF)

・HTML内に含まれる「textarea」

検出可能な脆弱性：クロスサイトスクリプティング(XSS)

・Webサイトを構築するHTMLに含まれるMBSD{××××}の内容

検出可能：MBSD{××××}に含まれるキーワード

・URL内に含まれるパラメータ

検出可能な脆弱性：クエリ文字インジェクション

実行例(1)

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

2.3規模の抽出

①検査するURLの中の同じドメインのURLを抽出する。

②①で抽出した同じドメインのURL中のURLにも検査をして検査元と同じドメイン名　　　　　　　　　　　　　　　　　　　　　　　　　　　　のURLが無いか探す。

③検出済みでない同じドメインのURLがでてきたら①と②を繰り返し同じドメインの　　　URLが無くなるまで処理をする。

検査したURLの個数

検査にかけたURLの中に同じドメインのURLが何個存在するか検出する。　　　(①②③の合計数)

　※大規模すぎると時間がかかるので検査するURLの数は制限できる。

　　　　※画像だけのURLなどは削除している

　　　　※アクセス先が同じでもURLが1文字でも同じ場合はカウントされる

　※脆弱性がないURLは省かれます

2.4作成した自動巡回ツール

名称：MBSD＿Web＿CSS

ディレクトリの構造

MBSD＿Web＿CSS

├── cgi-bin

│ └── team\_motiduki

│ ├── MBSDmain\_HTML.py

│ ├── MBSDmain.py

│ └── MBSD\_Tools

│ ├──\_\_init\_\_.py

│ ├──danger\_url.py

│ ├──GetHTML.py

│ ├──GetTAG.py

│ ├──HTML\_CONTENT.py

│ ├──INPUT.py

│ ├──is\_image\_url.py

│ ├──MBSD\_SEARCH.py

│ ├──mbsd\_title.py

│ ├──url\_Crawler.py

│ ├──url\_Domain\_Select\_Getter.py

│ ├──url\_Getter.py

│ ├──url\_Host.py

│ ├──url\_Parameter.py

│ ├──url\_Visited.py

│ └──\_pycache\_

│ ├──\_\_init\_\_.cpython-39.pyc

│ ├──\_\_init\_\_.cpython-311.pyc

│ ├──danger\_url.cpython-39.pyc  
│ ├──danger\_url.cpython-311.pyc

│ ├──GetHTML.cpython-39.pyc

│ ├──GetHTML.cpython-311.pyc

│ ├──GetTAG.cpython-39.pyc

│ ├──GetTAG.cpython-311.pyc

│ ├──HTML\_CONTENT.cpython-39.pyc

│ ├──HTML\_CONTENT.cpython-311.pyc

│ ├──INPUT.cpython-39.pyc

│ ├──INPUT.cpython-311.pyc

│ ├──is\_image\_url.cpython-39.pyc  
│ ├──is\_image\_url.cpython-311.pyc

│ ├──MBSD\_SEARCH.cpython-39.pyc

│ ├──MBSD\_SEARCH.cpython-311.pyc

│ ├──mbsd\_title.cpython-39.pyc

│ ├──mbsd\_title.cpython-311.pyc

│ ├──url\_Crawler.cpython-39.pyc  
│ ├──url\_Crawler.cpython-311.pyc

│ ├──url\_Domain\_Select\_Getter.cpython-39.pyc

│ ├──url\_Domain\_Select\_Getter.cpython-311.pyc

│ ├──url\_Getter.cpython-39.pyc

│ ├──url\_Getter.cpython-311.pyc

│ ├──url\_Host.cpython-39.pyc

│ ├──url\_Host.cpython-311.pyc

│ ├──url\_Parameter.cpython-39.pyc

│ ├──url\_Parameter.cpython-311.pyc

│ ├──url\_Visited.cpython-39.pyc

│ └──url\_Visited.cpython-311.pyc

├── css

│ └──answer.css

│

├──answer.html

│

├──hello.css

│

├──hello.html  
│

├──pythonin.bat  
│

└──サーバースタート.bat

2.5 アピールポイント

・ローカルサーバへの接続・パスの移動・検出Webサイトの立上げを全てbat処理により

　自動化しました

・検索Webサイトのデザインを見やすいようにしました

・結果表示を棒グラフを用いて表すことでどの脆弱性が多いのか分かるようにしました

・HTMLを取得する関数をメモ化することで実行時間の短縮をしました。

・設定で検出するURLの数を制限できるようにしました。

・beautifulsoupを使わずタグの解析をhtml文書を元に行うことで拡張性を高くしました　　(例:ある商品名がそのURLの中に存在するかなどの処理の追加が安易にできる)

・プログラムは関数を使いパッケージ化を行うことで重複した処理を少なくしました

・画像URLの検出は省きwebページだけの検出にしました

・抽出には正規表現を使いURLの検出の範囲を広くしました

・URLのhtml文書を取得する処理を少なくするために取得する前になるべく絞るよう

　しました

3.**クローラーの動作条件**

3.1動作条件

・python バージョン

　python3 version 3.9.12（動作確認済）

・必要なパッケージ

　urllib3 2.0.6（動作確認済）

・実行推奨環境

　 　エディション Windows 11 Pro

　 　バージョン　 22H2

　OS ビルド 　　　22621.2428

　 　エクスペリエンス Windows Feature Experience Pack 1000.22674.1000.0

**3.2 実行ブラウザの推奨環境**

　 ・Microsoft Edge

バージョン 119.0.2151.44 (公式ビルド) (64 ビット) （動作確認済）

・Google Chrome

バージョン: 119.0.6045.124（Official Build)(64 ビット）（動作確認済）

・Firefox

バージョン:116.0.3(64ビット) （動作確認済）

4.**実行例**

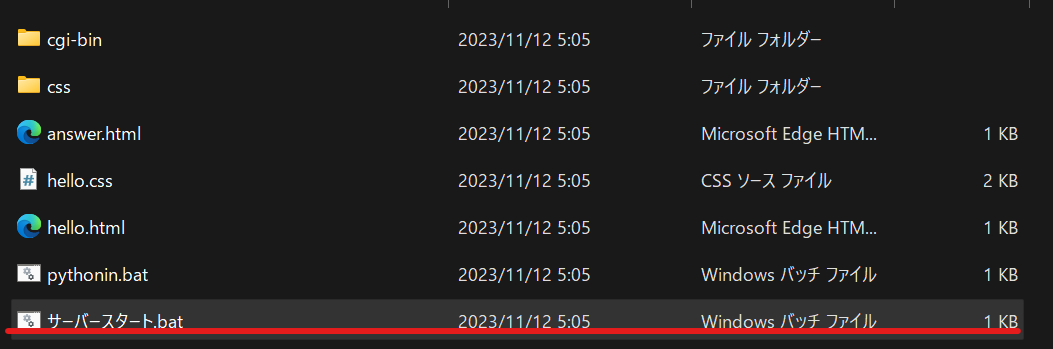
4.1 実行例

・ファイル内に含まれているサーバスタートbatを実行する。

※対応したpip,urllibのインストール,サーバの立ち上げが実行されます

※python自体がインストールされていない場合はインストールしてください。

※pythonのバージョンがあっていない場合はpythonin.batを実行しバージョンを合わせてください



・入力フォームにURLを入力する

今回は、<https://scraping-training.vercel.app/>を使って実行した

※チェックを入れることで検出するURLは制限をすることができます。

(url\_CrawlerのCRAWLING\_RESTRICTIONの値を変えることで制限の幅を変更できます)



・出力結果を確認する



5.**関数**

　5.1関数について

　作成に当たって準備した関数

**5.1.1　MBSDmain\_HTML.py**

検査するURLを受けとりそのURLを元に解析をして結果を表示する

動作概要

① 検査URLをフォームから受け取る

② 結果を表示するHTMLファイルを取得

③ 検査URLの検査を開始

④ 検査結果をHTMLのテーブルテンプレートとして保存する

⑤ 検査URLの中の脆弱性がある同じドメインのURLの検査を開始

⑥ 同様にテーブルテンプレートを保存

⑦ ドメインURLの数だけ⑤、⑥を繰り返す

⑧ ②のHTMLファイルを保存したテーブルを表示するように書き換え

⑨ 書き換えたHTMLを出力する

**5.1.2　GetHTML.py**

①GET＿HTML

引数 : url(文字列型)

戻り値 : html(文字列型)

与えられたurlを元にHTMLを取得

utf-8でhtmlをデコードしエラーが出た場合はShift-JISでデコード

デコードができなかった場合はからのhtmlの文字列を返す

メモ化をしており引数が同じの二回目以降の処理は短縮される

**5.1.3　GetTAG.py**

①GET\_TAG

引数 : html(文字列型)

戻り値 : taglist(文字列型配列)

与えられたhtmlを元に存在するタグを抽出。

タグを一つ一つ分けて配列に格納

**5.1.4　INPUT.py**

　①GET＿INPUT

　引数 : 文字列型　html…html文が文字列として格納されている。

　戻り値 ：リスト tagVul…タグの種類

リスト tagcount…検出したタグの個数

　変数htmlを引数として関数を実行すると、html文の中にあるタグの種類の個数を

　tagcountに格納して返す。

　検出するタグは、<input><form><textarea>の３つと、<input>の属性である

　text,file,password,hiddenを対象にして検出している。

　リストtagVulは添え字が0から始まり、

　<input>,text,file,password,hidden,<form>,<textarea>の順に格納されている。

　また、リストtagcountの個数の格納順はリストtagVulと共通である。

**5.1.5　MBSD＿SEARCH.py**

　①MBSD\_SEARCH

　引数 : html(文字列型)

　戻り値 : result(文字列型配列)

　与えられたhtmlの中からMBSD{XXXX}のXXXXの内容があるか探す。

　あった場合は配列に格納してその結果を返す。

**5.1.6　mbsd\_title.py**

　①HTML\_TITLE

　引数 : html(文字列型)

　戻り値 : title(文字列型配列)

　与えられたhtmlからタイトルを抽出しその結果を配列で返す。

**5.1.7　url\_Getter.py**

①getURL(url,html):

　引数:url(文字列型),html(html文書を文字列で取得したもの:文字列型)

　戻り値: urls(文字列型配列)

　　アクセス先に含まれる別のURLを重複がないように取り出す。(複雑すぎるパラメー

　　タを含む場合はそのURLは取り出せない)

　　この関数は<a>タグ内のhrefのリンクとhtml文書を解析してURLらしきものを全

　　て検出できるようにしました。この解析処理を高速に終わらせるために正規表現を用

　　いました。

　②del\_equal\_url(urls):

　引数:urls(文字列型配列)

　戻り値: urls(文字列型配列)

　　同じURLが含まれていた場合に削除します

　　mark\_visited(url),is\_visited(url),reset\_visited\_urls()の関数を用いて同じURLを削除す

　　る機構を作りました。使う関数ごとに差別化する(保存する領域を差別化する)ために

　　del\_equal\_url(urls)内では同じ動作をする別の関数を用いています。

**5.1.8　url\_Host.py**

　①getURL\_host(url):

　引数:url(文字列型)

　戻り値: host(文字列型)

　ホスト名を取得します

**5.1.9　url\_Parameter.py**

① getURL\_para(url):

引数:url(文字列型)

戻り値:passstr(文字列型)

　URLを取得した際のパラメータが'&'などの文字列が'&amp;'のようにエスケープされる

　とパラメータが正常に取得できないので正常に取得できるように工夫しました。

②getURL\_path\_para(url):

引数:url(文字列型)

戻り値:path\_parameters(文字列型)又はなし

　URLに含まれるパスパラメータのみを取得する。パスパラメータがない場合は戻り値を

　返されません。

**5.1.10　is\_image\_url.py**

　①is\_image\_url(url):

　引数:url(文字列型)

　戻り値:Ture、False

　URLに画像ファイル、.js、.json、.cssを含むURLがある 時にTrue,ない時にFalseが

　戻り値として返す正規表現を用いてURL(文字列型)に対する検索をかけました。より高

　速に解析できると考えます。

　②image\_del(urls):

　引数:urls(文字列型配列)

戻り値:urls(文字列型配列)

　URLにis\_image\_url(url)を適用、TrueのときにそのURLを削除しurlsの上書き保存し

　ます。

**5.1.11　url\_Visited.py**

　①mark\_visited(url):

　引数:url(文字列型)

　戻り値:なし

URLを保存します(マークする)。

　②is\_visited(url):

　引数:url(文字列型)

　戻り値:Ture、False

　url(文字列型)が保存されている(mark\_visited(url)が既に実行されている)ときにTrue,そ

　れ以外はFalseが戻り値として返します。

　③reset\_visited\_urls():

　引数:なし

　戻り値:なし

　mark\_visited(url)で保存されたデータを全て削除します。

**5.1.12　url\_Domain\_Select\_Getter.py**

　①is\_getURL\_selected\_by\_domain(source\_url, comparison\_url):

　引数:source\_url(文字列型),comparison\_urll(文字列型)

　戻り値:Ture、False

引数として指定されたURL(文字列型)2つに対して、ドメイン名が同じの場合True,ド

イン名が異なる時にFalseが戻り値として返します。

②getURL\_selected\_by\_domain(url,html):

引数:url(文字列型),html(html文書を文字列で取得したもの:文字列型)

戻り値:domain\_urls(文字列型配列)

引数として指定されたURL(文字列型)とhtml(html文書を文字列で取得したもの)に対し

てドメイン名で絞りだした後にimage\_del(urls)を用いて画像URLなどを削除します。そ

の結果を戻り値として返します。

**5.1.13　url\_Crawler .py**

　①url\_crawling(url,html):

　引数:url(文字列型),html(html文書を文字列で取得したもの:文字列型)

　CRAWLING\_RESTRICTIONの値を変えることで制限を変えることができる。

　値を増やすと検出するURLの個数が少なくなる。(初期値3)

　例：CRAWLING\_RESTRICTIONの値が３の時、開始URLを含んで３個までのURLを取得してそこまでのURLのクローリングを実行し検査する

　戻り値: crawling\_urls(文字列型配列)

　引数として指定されたurl(文字列型)とhtml(html文書を文字列で取得したもの)に対し

　てクローリングを実行し、結果を戻り値として返します

**5.1.14　danger\_url.py**

　①danger\_url.py(html):

　引数:html(html文書を文字列で取得したもの:文字列型)

　戻り値:ht\_mbsd(文字列型配列),ht\_title(文字列型配列),tagVul(文字列型配列),

　tagcount(整数列型配列),flag(整数型)

　引数として指定されたhtml(html文書を文字列で取得したもの)に対してMBSD{xxxx}の

　xxxxの文字をht\_mbsd、タイトルをht\_title、脆弱性タグリスト名をtagVul、それに対応

　するタグの個数をtagcountに格納して返します。

　flagは脆弱性が0の時に1になって返します。(ht\_mbsd,tagcount全てが空値か0の時)

**6.考察・まとめ**

6.1　望月　紋之丞

　今回大会主催側であるMBSDからの問題定義では、Webサイト運営会社の社内で使

用されることが背景状況として提示されていた。このことから一度の操作によりで　　きる限り脆弱性を検出できるツールの作成を目標に動いた。

今回のMBSDでのツールを作成するにあたっての反省点としては、リーダとして日　　程管理をしきれなかった点である。いつ頃にどのようなツールを作るのか、どのプ　　ログラムコードを使い作成をするのか、搭載する機能はどのようなものにするのか　　などを考える日程の管理を怠ってしまい最後に詰めすぎてしまった点です。

　次回このような大会に参加する場合には、開発初めに日程管理を考えながら開発を　　進めていきたいと考える。

　今回自分は、自動巡回ツールのHTML・CSS・HTMLとPython間で文字列を明け渡　　すプログラムを作成したがCGIやローカルサーバの接続・使い方が最初は、わから　　ならなかったが今回のプログラム作成でそれらローカルサーバの使い方などを理解　　することが出来た。また、仲間たちの協力合って今回の自動巡回ツールは、作成す　　ることが出来たと考えてるのでチームの人たちには、とても感謝している。

6.2　望月　健翔

　今回のMBSDの巡回ツールの作成で関数を作成することの大切さを知ることができ　　た。またそれぞれが作成した関数を結合する際にその人なりの変数の名前やプログ　　ラム構造など癖が出てきて結合するのに大変だったので、複数人でプログラムなど　　のシステムを作成する際はしっかりとどのような変数の名前やファイルの名前にす　　るのか決めるのが重要な事だと知ることができた。またプログラムを作成するにあ　　たり、pythonは便利なライブラリが標準でついていて利用することで簡略化できる　　事もしれた。しかし、現状のプログラムでは実行時間がとても長かったので非同期　　処理や並列処理などに変更する事も時間があったら追加したいと思った。また実行　　している最中もプログラム中の何％が終了しているかなどの進捗具合が出ないのは　　pythonプログラムをcgiスクリプトとして使用してるからなのでcgiスクリプトでは　　ないやり方で作成する改善できるのではないかと感じた。また文字コードエラーも沢山出て出力する文字コードや取得する際の文字コードなどでエラーが沢山でて解決するのに時間がかかった。これをきっかけに文字コードについてしれたのでインターネットなどから情報を拾うときは注意すべきだと知ることができた。

6.3　法村　寿史

Pythonについては顔見知り程度で、チームでのツールの作成は初めてでした。　　　チームメイトとの認識の齟齬があることでプログラムの作成に遅れが生じたことや　　プログラム作成での分担によって効率化したもののデバッグに時間が想像以上にか　　かってしまったこともあり、思い通りにいかないことの多い開発でした。自分で試　　行錯誤しながらの開発であり、プログラミング全般の知識が問われました。上手く　　いかないことも多く、作業時間についてもかなりギリギリになってしまいました。　　開発でのハプニングに対応できるようにさらに余裕をもって取り組むべきと考えま　　す。プログラム中での改良を行いたいと感じる部分が散見した部分からも時間の管　　理の大切さを感じます。予想外の所でプログラムが上手くいかないこともあり大変　　な開発であったと感じます。特に文字コードによるエラーが幾度となく発生してそ　　の度にプログラムを改変し修正を試みかなり骨の折れた点であると思います。URLの検出方法についての効率化については検討の余地があると考えます。

6.4　大石　翔亜

　今回はMBSD初参加で、Pythonを触るのも初めてだったが、自分の中では一番良

いプログラムをかけたと思う。また、ライブラリを使用するのも初めてだった。

集団で一つのものを作ったり、一からシステムの構造を考えたり、普段の学校や生活で　　　　　　　　　　　　　は得ることのできない経験を得ることができ、とても貴重なものだった。改善点は、作業をするときに、完成したプログラムをDiscordに貼り付けて共有しており、効率が悪かったり、綿密な計画を立てていなかったので、今回は効率が悪かった。ほかにも、用事があったりして個人の作業量にばらつきがあったりしたので、作業時間にゆとりを持たせて作業をすればよかった。今回の経験を踏まえて、次回はもっと効率のいい作業をしていきたいと思った。