Redmineの脆弱性診断を「Ruby × Selenium」を使って自動化

ファーエンドテクノロジー株式会社　坂本 想

自己紹介

坂本　想

Sakamoto　So

■　所属

・ファーエンドテクノロジー株式会社
入社3年目

■　業務

・既存プロダクトの改善
・テストの自動化

アジェンダ

1 脆弱性診断の必要性

2 Redmine脆弱性診断の流れ

3 Ruby × Selenium メソッドの紹介

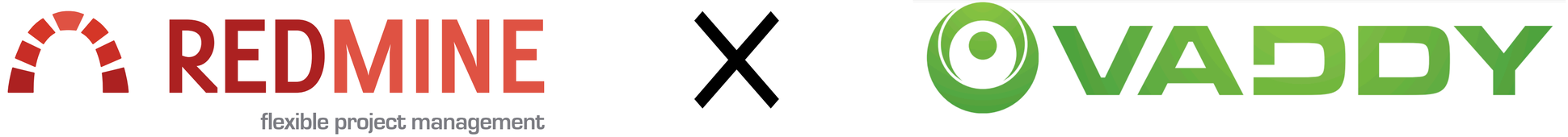
4 脆弱性診断を自動化してみて

5 本日のまとめ

1. 脆弱性診断の必要性

脆弱性診断の定期実行

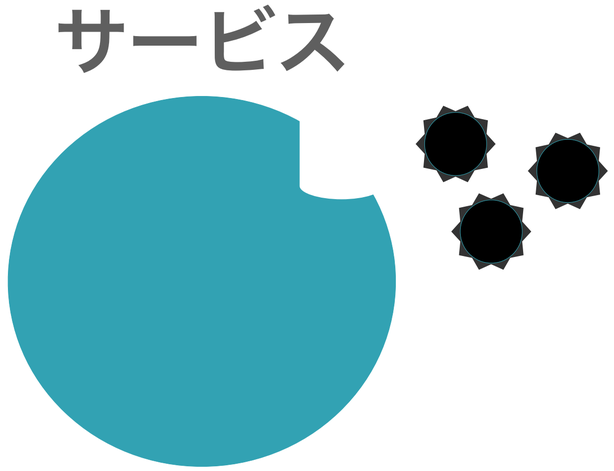
Redmineの**脆弱性診断**を**定期的**にしています！



脆弱性とは？

- サービスの**問題点**

脆弱性とは？



- サービスの**問題点**

- 脆弱性があると、**ウイルス感染の危険性**がある

  ・個人情報漏洩
  ・サイバー攻撃

脆弱性診断とは

- サービスの**問題点**がないか**チェック**する

脆弱性診断とは

- サービスの**問題点**がないか**チェック**する
- より**品質の高いサービス**を**提供**できる

2. Redmine脆弱性診断の流れ

脆弱性診断の流れ

大きく分けて2つのステップ

Step1

クロールデータ の作成

・リンクをクリックするなどのRedmineの操作を記録したデータ

脆弱性診断の流れ

大きく分けて2つのステップ

Step1

クロールデータ の作成

・リンクをクリックするなどのRedmineの操作を記録したデータ

Step2

Scan

・クロールデータ を選択して、脆弱性診断を開始

脆弱性診断の流れ

本日メイン

Step1

クロールデータ の作成

・リンクをクリックするなどのRedmineの操作を記録したデータ

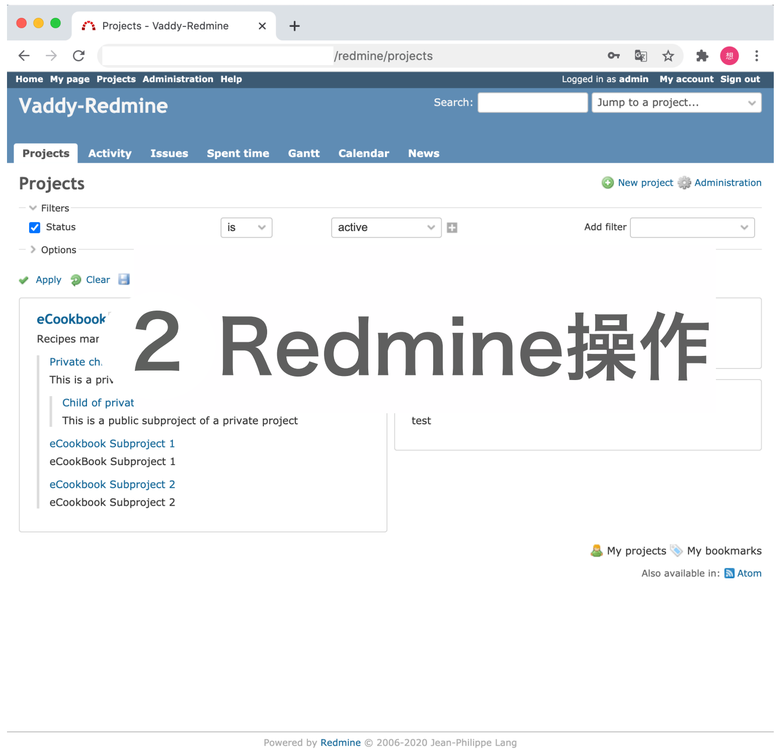
Step2

Scan

・クロールデータ を選択して、脆弱性診断を開始

クロールデータ の作成

Step1



2 Redmine操作

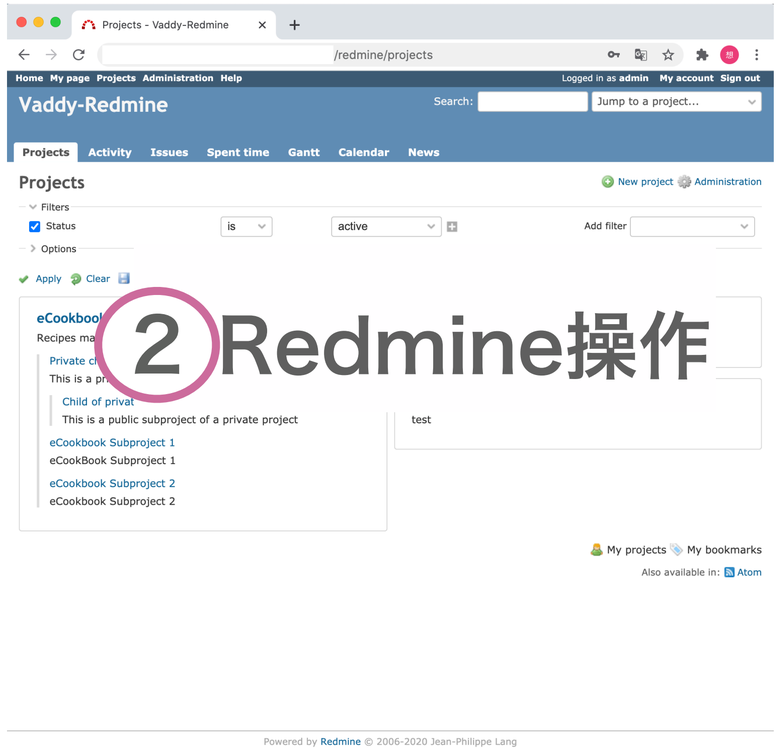
① 開始URL

Proxy

③ 終了URL

クロールデータ の作成

Step1



2 Redmine操作

1

Proxy

3

VADDY

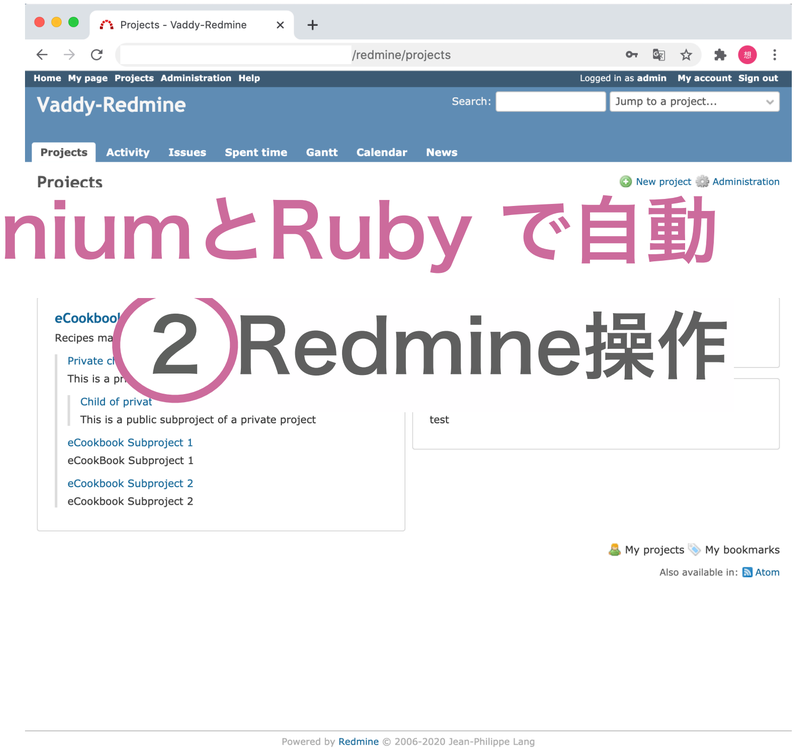
■ 開始URL

■ 終了URL

クロールデータ の作成

Step1

SeleniumとRuby で自動



② Redmine操作

1

Proxy

3

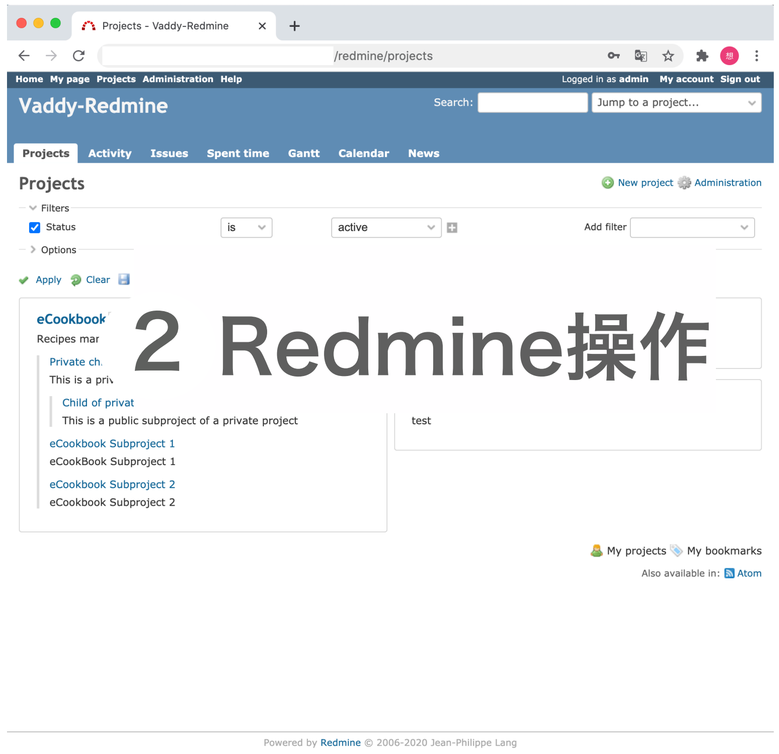
VADDY

■ 開始URL

■ 終了URL

クロールデータ の作成

Step1



2 Redmine操作

1

Proxy

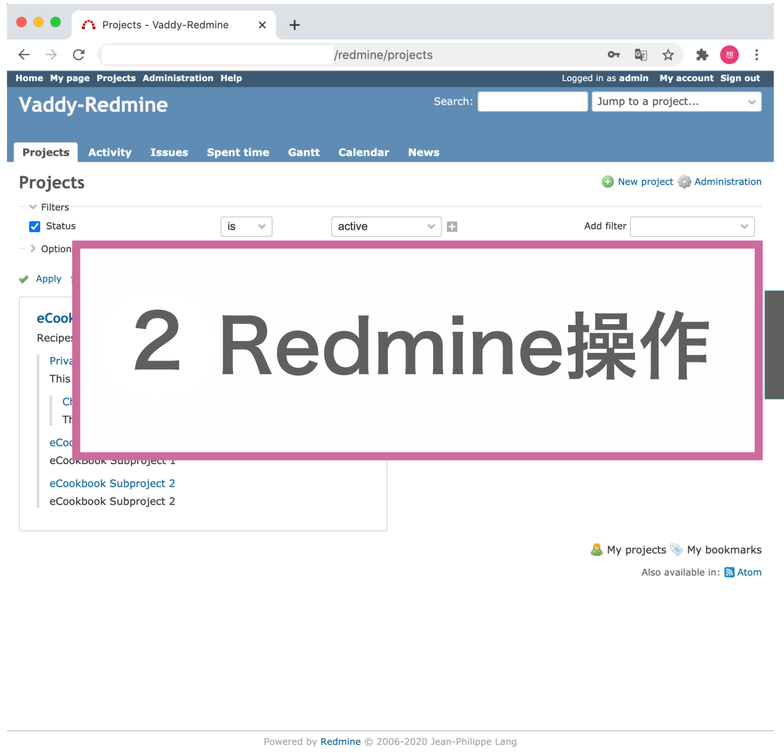
3

VADDY

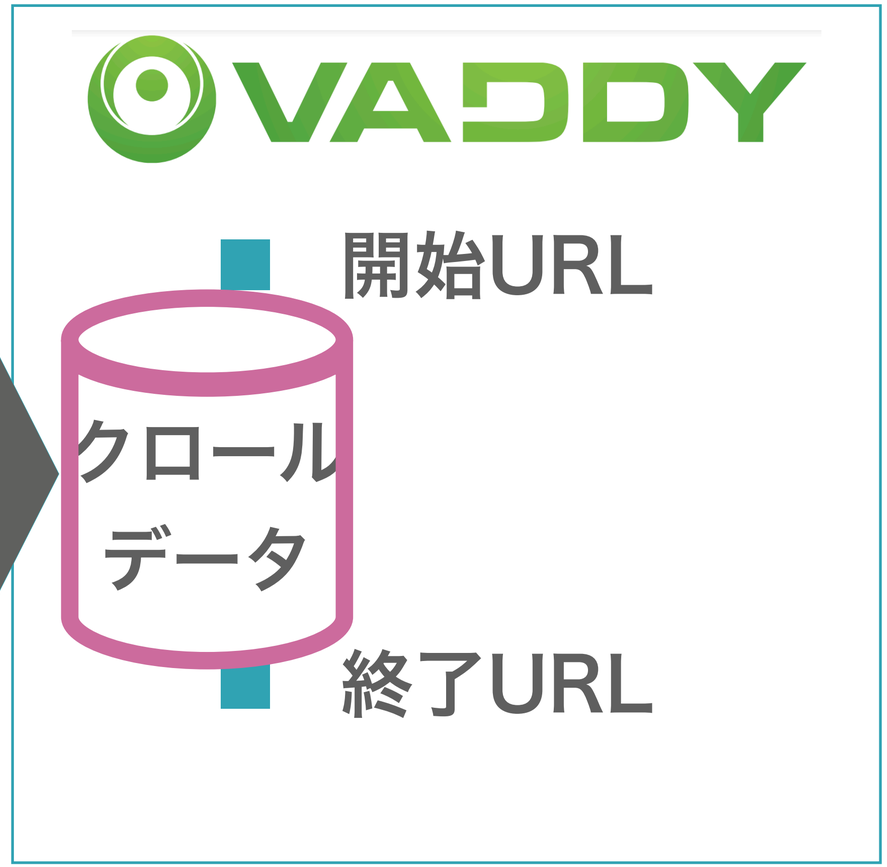
- 開始URL
- 終了URL

クロールデータ の作成

Step1

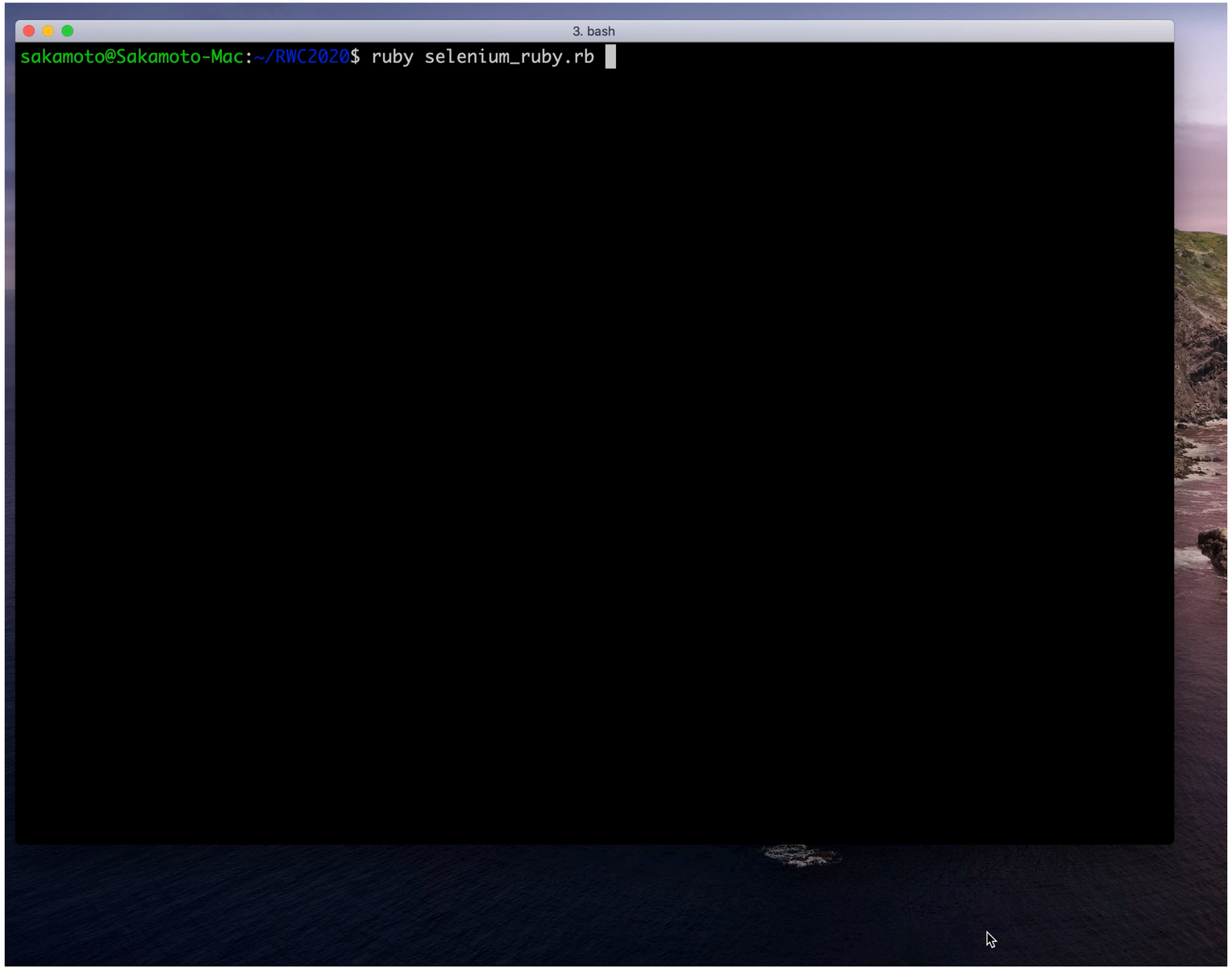


2 Redmine操作



3. Ruby × Selenium メソッドの紹介

Seleniumが動く動画



コードの中身

```ruby
require 'selenium-webdriver'

driver = Selenium::WebDriver.for :chrome
driver.manage.timeouts.implicit_wait = 10

# redmineにログイン
driver.navigate.to('http://vaddy-redmine.jp/redmine/login')
driver.find_element(:id, 'username').send_keys 'admin'
driver.find_element(:id, 'password').send_keys 'admin'
driver.find_element(:id, 'login-submit').click

# 新しいチケット作成画面へ移動
driver.navigate.to('http://vaddy-redmine.jp/redmine/issues/new')

# 新しいチケット(ticket1)を作成
driver.find_element(:id, 'issue_subject').send_keys 'ticket1'
driver.find_element(:id, 'issue_description').send_keys 'ticket1'
# ファイルを添付
driver.find_element(:xpath, '//*[@id="attachments_form"]/span/span[2]/input').send_keys '~/dummy.png'
driver.find_element(:xpath, '//*[@id="attachments_1"]/input[2]').send_keys 'dummy'
driver.find_element(:xpath, '//*[@id="issue-form"]/input[3]').click

driver.quit
```

紹介する操作

- ブラウザ起動
- ページ遷移
- 要素取得（テキスト入力、クリック）
- 待機処理

紹介する操作

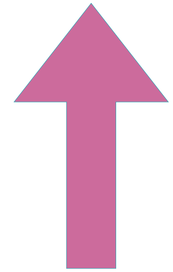
- ブラウザ起動
- ページ遷移
- 要素取得（テキスト入力、クリック）
- 待機処理

ブラウザ起動

```
driver = Selenium::WebDriver.for :chrome
```

ブラウザ起動

```
driver = Selenium::WebDriver.for :chrome
```



- 起動するドライバを指定

ブラウザ起動

firefox

```
driver = Selenium::WebDriver.for :chrome
```

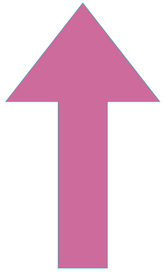
- 起動するドライバを指定
- firefoxドライバも指定できる

紹介する操作

- ブラウザ起動
- ページ遷移
- 要素取得（テキスト入力、クリック）
- 待機処理

ページ遷移

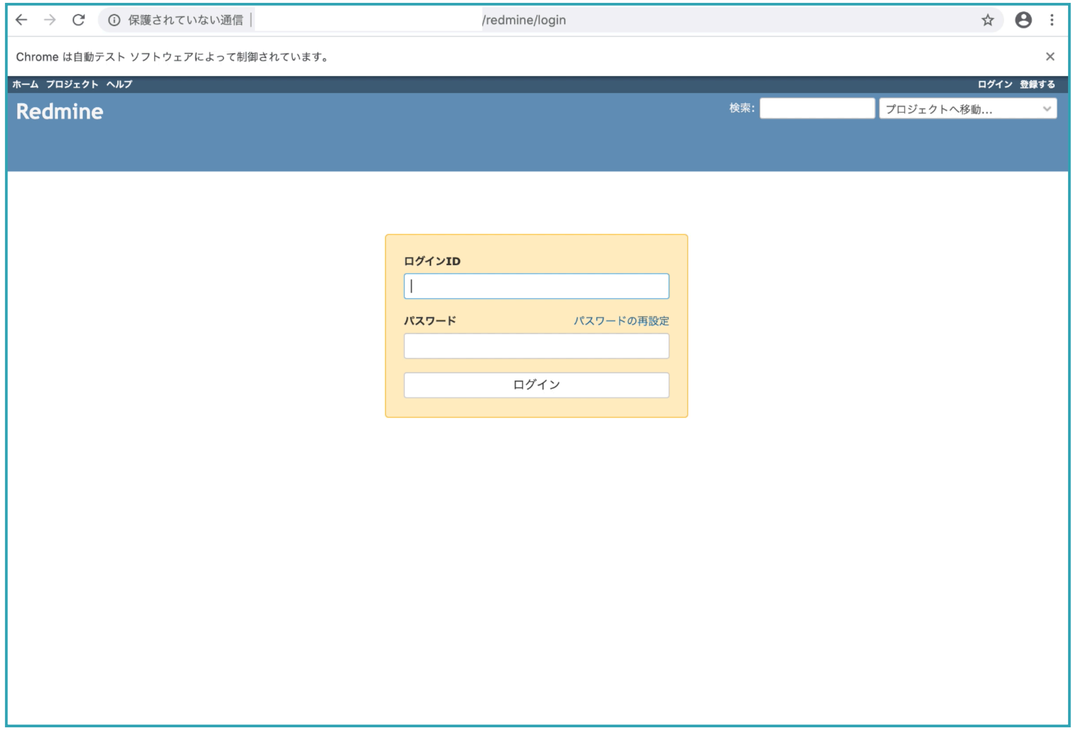
```
driver.navigate.to('http://vaddy-redmine.jp/redmine/login')
```



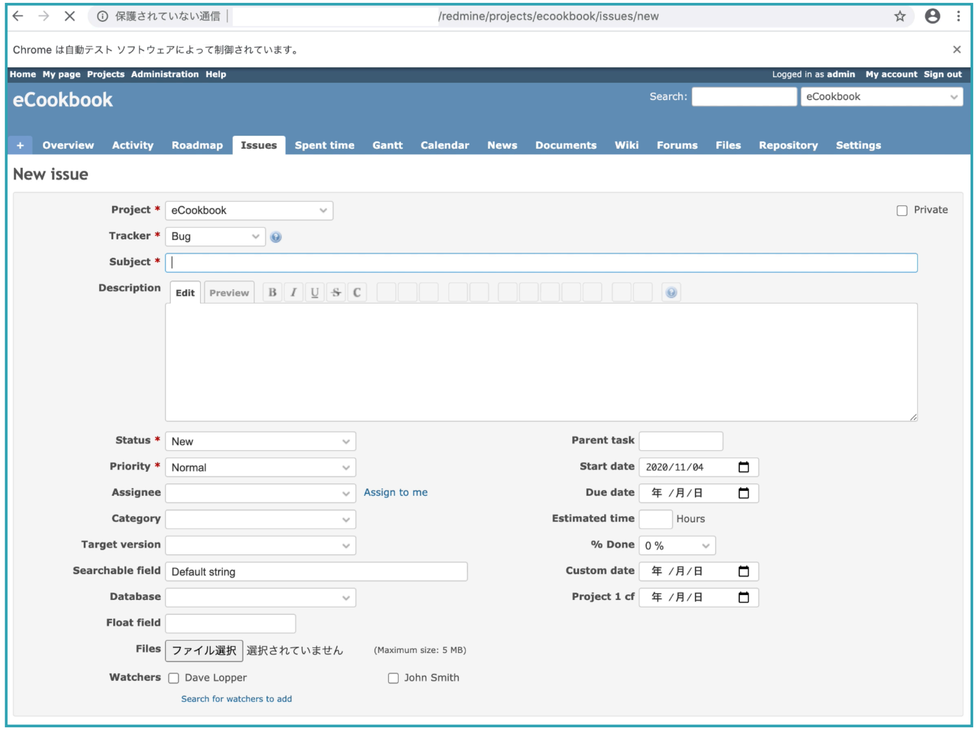
- アクセスしたいURLを指定

ページ遷移

ログイン画面

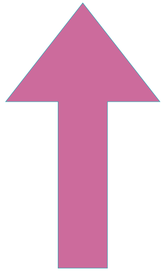


チケット作成画面



ページ遷移

```
driver.navigate.to('http://vaddy-redmine.jp/redmine/login')
```



- アクセスしたいURLを指定
- ログインページなどにアクセスできる

紹介する操作

- ブラウザ起動
- ページ遷移
- 要素取得（テキスト入力、クリック）
- 待機処理

要素取得

```
driver.find_element(:id, 'username')
```

属性

要素取得

```
driver.find_element(:id, 'username')
```

属性値

要素取得

```
driver.find_element(:id, 'username')
```

要素取得

```
driver.find_element(:id, 'username')
```

- idがusernameの要素を取得

要素取得

```
driver.find_element(:id, 'username')
```

- idがusernameの要素を取得
- xpath、name属性の要素も取得できる

要素取得

```
driver.find_element(:id, 'username')
```

＋

- 要素に文字を入力するメソッド send_keys

要素取得

```
driver.find_element(:id, 'username')
```

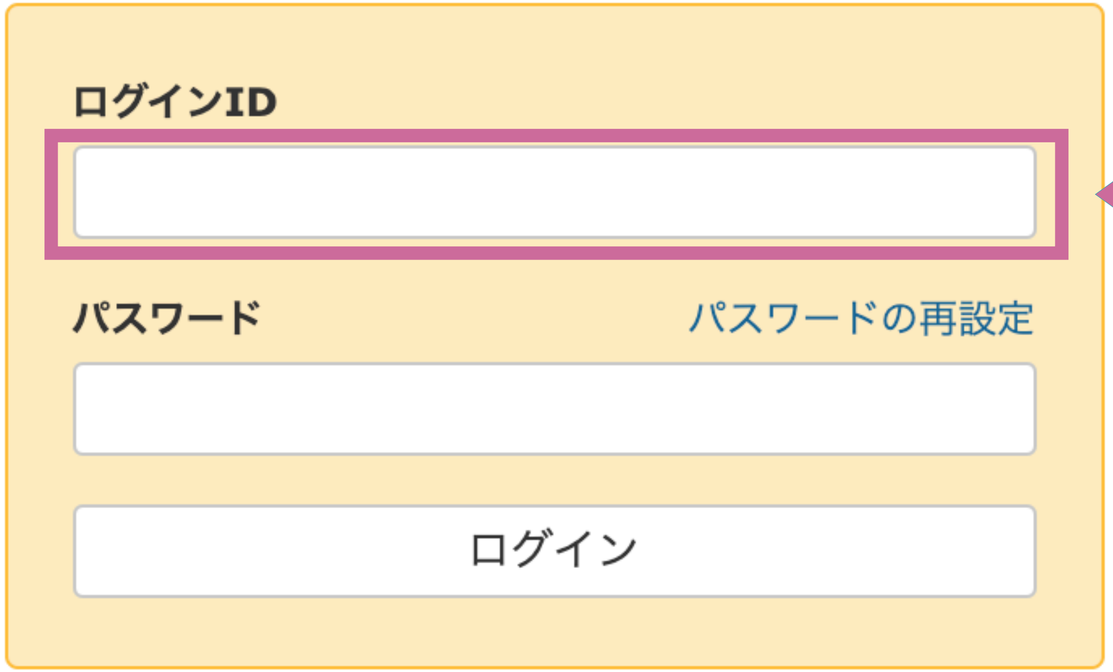
＋

- 要素に文字を入力するメソッド send_keys
- 要素をクリックするメソッド click

など…

文字入力: send_keys

```
driver.find_element(:id, 'username').send_keys 'admin'
```

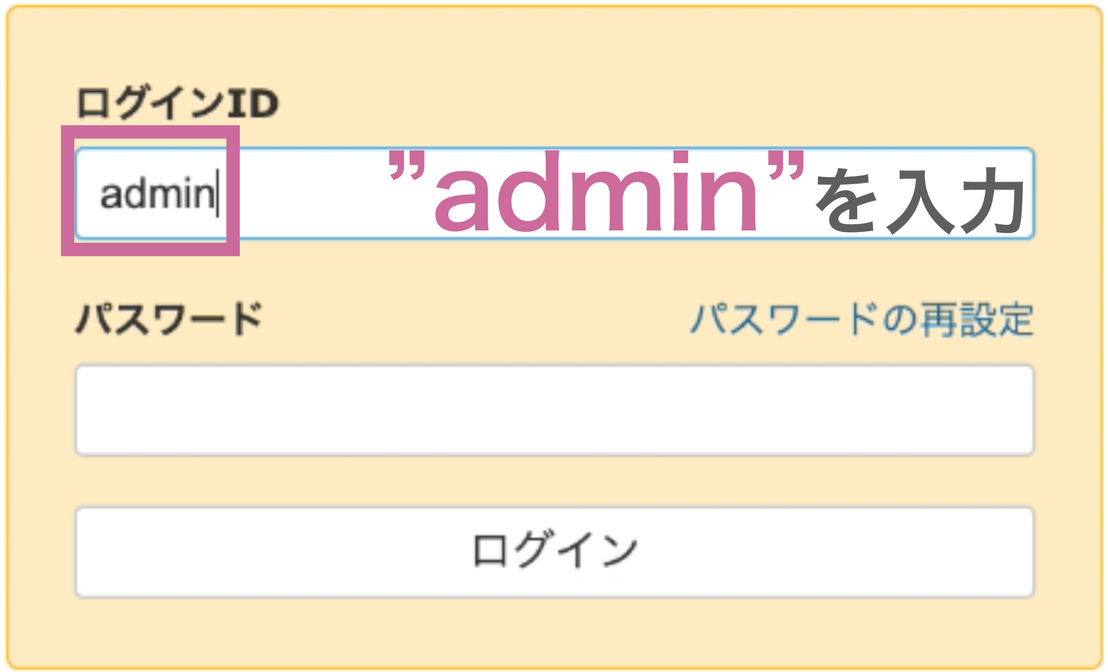


send_keys ‘admin’

を指定

文字入力: send_keys

```
driver.find_element(:id, 'username').send_keys 'admin'
```



文字入力: send_keys

```
driver.find_element(:id, 'username').send_keys 'admin'
```

- send_keysメソッドは
  使用頻度が高い

クリック: click

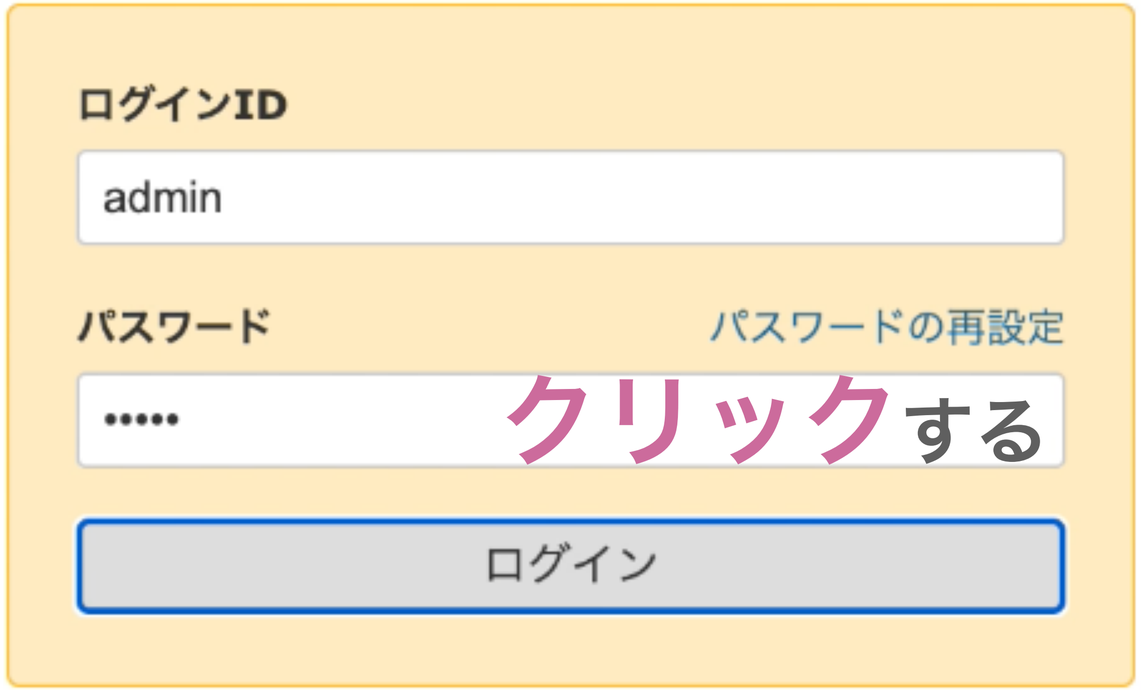
```
driver.find_element(:id, 'login-submit').click
```

クリック: click

```
driver.find_element(:id, 'login-submit').click
```

クリック: click

```
driver.find_element(:id, 'login-submit').click
```

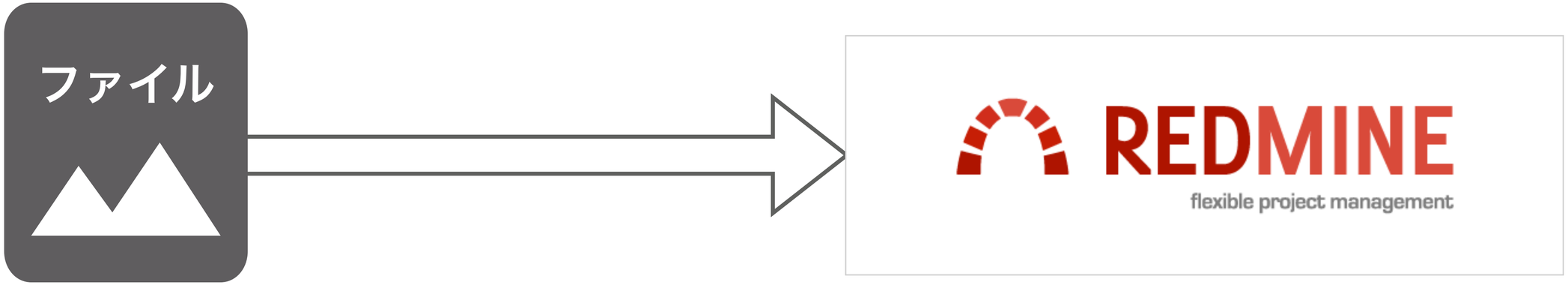


- ボタンのクリック
  をするときに使う

紹介する操作

- ブラウザ起動
- ページ遷移
- 要素取得（テキスト入力、クリック）
- 待機処理

待機処理



- 添付処理は時間がかかる
- 処理が終わる前に次の処理を実行するとエラーとなる
- 処理が終わるまで待機する

待機処理

3種類試した中で、1番良いのはimplicit_wait

① Sleep

② Wait.until

③ implicit_wait

① 待機処理: Sleepメソッド

```
sleep 5
```

5秒の間、一時停止状態にする

① 待機処理: Sleepメソッド

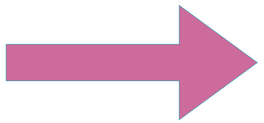
```
sleep 5
```

5秒の間、一時停止状態にする

明確な秒数がわからない

必要以上にスリープしてしまう

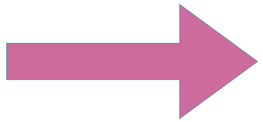
② 待機処理: Wait.untilメソッド



```ruby
# waitに50秒のタイマーを持たせる
wait = Selenium::WebDriver::Wait.new(:timeout => 50)

# 指定した要素が表示されるまで待つ
wait.until{driver.find_element(:id, 'test').displayed?}
```

② 待機処理: Wait.untilメソッド



```ruby
# waitに50秒のタイマーを持たせる
wait = Selenium::WebDriver::Wait.new(:timeout => 50)

# 指定した要素が表示されるまで待つ
wait.until{driver.find_element(:id, 'test').displayed?}
```

② 待機処理: Wait.untilメソッド

```ruby
# waitに50秒のタイマーを持たせる
wait = Selenium::WebDriver::Wait.new(:timeout => 50)

# 指定した要素が表示されるまで待つ
wait.until{driver.find_element(:id, 'test').displayed?}
```

毎回呼び出しが必要

待機処理

3種類試した中で、1番良いのはimplicit_wait

① Sleep

② Wait.until

③ implicit_wait

③ 待機処理: implicit_waitメソッド

```
driver.manage.timeouts.implicit_wait = 10
```

指定した要素が見つかるまでの**待ち時間**を**設定**する

③ 待機処理: implicit_waitメソッド

```
driver.manage.timeouts.implicit_wait = 10
```

指定した要素が見つかるまでの**待ち時間**を**設定**する

- **1度の設定**で良い
- 待ち時間を**一律**に指定することができる

4. 脆弱性診断を自動化してみて

脆弱性診断の流れ

自動化

Step1

クロールデータ の作成

・リンクをクリックするなどのRedmineの操作を記録したデータ

Step2

Scan

・クロールデータ を選択して、脆弱性診断を開始

脆弱性診断の流れ

Step1

- **クロールデータ の作成**
  - リンクをクリックするなどのRedmineの操作を記録したデータ

Step2

- **Scan**
  - クロールデータ を選択して、脆弱性診断を開始

クロールデータ の作成

Step2 シェルスクリプトとVAddyの機能を使って自動化

シェルスクリプト

~~~~~~~~~~
~~~~~~~~~
~~~~~~~~~
~~~~~~~

×

クロールデータ の作成

Step2　シェルスクリプトとVAddyの機能を使って自動化

シェルスクリプト

~~~~~~~~~~
~~~~~~~~~
~~~~~~~~~
~~~~~~~

今後の取り組み

- シェルスクリプト → Rubyで書く

脆弱性診断を自動化してみて

今までは…

- すべて**手作業**
- 脆弱性診断をするのに**時間**がかかっていた
- **丸一日**かかっていた

脆弱性診断の流れ

自動化

Step1

クロールデータ の作成

・リンクをクリックするなどのRedmineの操作を記録したデータ

Step2

Scan

・クロールデータ を選択して、脆弱性診断を開始

脆弱性診断を自動化してみて

自動化して

- 脆弱性診断をする**時間**が**少なくなった**
- 実行する**頻度**を**増やせた**
- 人為的作業ミスの**対策**にもなった

5. 本日のまとめ

今日のまとめ

- 自動化することで効率よく確実に実行することができる
- send_keysメソッドを活用する
- Seleniumを使う時は待機処理を使う
- implicit_waitメソッドが一番使いやすい