Webアプリケーションを安全にするフレームワークの新しい機能

久保田 康平

令和　3年　2月

情報知能工学専攻

概 要

本論文は，Webアプリケーションのセキュリティ機能向上を目的にしている．そのために本論文では，Webアプリケーション開発者が実装するコードを実行時に自動的に解析し，必要ならば修正する機能をWebアプリケーションフレームワークに持たせることを提案し，実装して評価を行う．Webアプリケーションはインターネットを通して世界中から誰でも接続でき，対話的に通信できるという特徴から様々な攻撃の対象になる．また，インターネットの普及に伴いWebアプリケーションの重要性は増し，同様にWebアプリケーションの防御もまた重要になっている．脆弱性攻撃は，Webアプリケーションの設計上の欠点や仕様上の問題点である脆弱性を利用する攻撃である．脆弱性攻撃の対策の一つは，Webアプリケーションに脆弱性を作らないことであり，そのためWebアプリケーション開発者はWebアプリケーションフレームワークを利用することがある．Webアプリケーションフレームワークは，Webアプリケーション開発において利用することが多いメソッドを持つライブラリである．それらのメソッドを利用すること

で効率よくアプリケーションを開発することができる．セキュリティ面において，Web アプリケーションフレームワークが提供するメソッドは脆弱性対策がなされているものが多い．したがって，Web アプリケーションフレームワークを利用した方が，利用しない時と比較して効率的にセキュアな Web アプリケーションを開発しやすい．一方で，開発者は常に完全にセキュアなコードを書くことはできないため，Web アプリケーションフレームワークを利用して，脆弱性がある Web アプリケーションを実装してしまうことがある．その理由の一つが，Web アプリケーション開発者が Web アプリケーションフレームワークを適切に利用できないことである．Web アプリケーション開発者が，フレームワークのメソッドが持つセキュリティ機能を正しく理解していなかったり，セキュリティ機能を持つメソッドを知らなかったりすることによって脆弱な Web アプリケーションが実装される．この問題に対して本論文では，Web アプリケーション開発者が実装したソースコードを修正する機能を持つ Web アプリケーションフレームワークを提案する．提案手法を実証し評価を行った結果，この機能は実装されたコードの脆弱性を一部修正でき，レスポンスタイムは提案手法を適用しなかった場合とほとんど変わらないことを確認した．実装された修正関数の蓄積は将来のアプリケーションのセキュリティの向上に寄与できるものである．

目 次

第 1 章　はじめに **1**

第 2 章　関連研究 **5**

2.1　論文 1 5

2.2　論文 2 5

2.3　論文 3 5

第 3 章　提案手法 **6**

第 4 章　実装 **9**

4.1　コールバック関数の修正機能 9

4.1.1　コールバック関数の格納 9

4.2　コールバック関数の修正 11

4.3　リクエスト処理システム 13

4.3.1　リクエスト情報の取得 13

4.3.2　コールバック関数の呼び出し 13

4.3.3　レスポンスの作成 13

第 5 章　実験 **14**

5.1 脆弱性の影響低減評価 14
5.1.1 SQLi を持つコールバック関数の修正 15
5.1.2 不適切な認証を持つコールバック関数の修正 17
5.1.3 結果 17
5.2 オーバーヘッドの評価 17
5.2.1 結果 17

第 6 章 考察 **18**

第 7 章 おわりに **19**

図 目 次

表 目 次

第1章

はじめに

本論文は，Web アプリケーションのセキュリティ機能向上を目的にしている．その目的の達成のために，アプリケーション開発者が実装したプログラム中の関数や引数を解析し，実行時にその関数に脆弱性があった時には修正することができる Web アプリケーションフレームワークを提案，実装し評価する．

Web アプリケーションセキュリティは，セキュリティ分野において重要である．インターネットの普及に伴い，Web システムは様々な場所や階層において様々な攻撃にさらされている．Web システムへの攻撃のうちアプリケーション層への攻撃の多くはアプリケーションのプログラムが持つ論理的な問題が原因である．そのため Web アプリケーション開発者は攻撃を回避するために，アプリケーションの論理的な問題や設計上の欠点である脆弱性を作らない実装をする必要がある．一方で，Web アプリケーション開発者は常にセキュアなコードを記述することはできず，脆弱性を残す実装をすることがある．加えて Web アプリケーション層に

はセキュリティに関するプロトコルや標準的な仕様がないため，Web アプリケーションの安全性は，Web アプリケーション開発者のセキュリティに関する知識や技術に依存する．これらの Web アプリケーションの問題を解決しセキュリティを向上するために，Web アプリケーションの自動防御手法として Web アプリケーションファイアウォール（WAF）や Web アプリケーションフレームワークの利用などが検討されている．

WAF は，Web アプリケーションを脆弱性攻撃から保護するためのシステムである．WAF は Web アプリケーションとクライアントの間に配置され，クライアントからのリクエストを監視し，リクエストが攻撃リクエストかどうかを検証する機能を持つ．攻撃を検出した場合，そのリクエストを遮断もしくは無毒化することで，Web アプリケーションへの攻撃の影響を低減する．WAF は Web アプリケーションを修正することなく，脆弱性攻撃を低減することが可能であるため，アプリケーションを直接修正できない時に有効な対策である．一方で WAF はアプリケーションを修正しないので，アプリケーション内の脆弱性を根本的に修正できないという欠点がある．また WAF はアプリケーション内の論理的な設計や仕様を知らないため，一部の脆弱性を対策することが難しい．WAF は通常，特殊文字を含むリクエストを攻撃として検出する．したがって，リクエスト内に特殊文字を含まない攻撃を WAF が検出することは難しい．

Web アプリケーションフレームワークは，Web アプリケーションを効率よく開発するために，Web 開発に多用される機能を関数やメソッドとして提供するライブラリである．自動防御手法としては，クロスサイトスクリプティング（XSS）や SQL インジェクション（SQLi）のようなインジェクション攻撃に対する入力検証と自動サニタイズという機能を提

供していることがある．自動サニタイズとは特殊文字をエスケープする機能であるサニタイズを Web アプリケーションフレームワークが行う一部の Web アプリケーションフレームワークが持つ機能である．自動サニタイズの長所は Web アプリケーションのセキュリティの一部を Web アプリケーションフレームワークが負担することが可能なことである．自動サニタイズによって Web アプリケーション開発者はサニタイズについて考慮することなく，セキュアな Web アプリケーションを実装することが可能になる．一方で自動サニタイズは限定的な対策で，インジェクション攻撃ではない攻撃を対策することが難しい．

Web アプリケーションの自動防御は Web アプリケーションの論理的な設計を検証し脆弱性の影響を低減する機能を持たないため，一部の攻撃を自動的に防御することができない．具体的には，Web アプリケーションの不適切な認証への攻撃を自動で対策する手法を Web アプリケーションフレームワークは持たない．不適切な認証は，アプリケーションの利用者が権限を所持していると主張した時に，アプリケーションがその主張が適切かどうかを証明しない，もしくは不適切に証明する脆弱性である．

この問題を解決するために，本論文ではアプリケーション開発者が実装したソースコードを解析し，必要であれば修正する Web アプリケーションフレームワークである VHF（Vulnerability Handling Framework）を提案する．VHF は脆弱なソースコードの条件とそのソースコードを修正するプログラムを持っている．Web アプリケーション開発者が記述したソースコードを実行開始時に静的に解析することで脆弱性を検出する．その後，脆弱なソースコードを保護するための関数を挿入したり，安全な関数に置き換える．挿入された関数は実行中にアプリケーションを動的に

検証し，攻撃を検出すると無毒化する．この提案手法の貢献は，Web アプリケーション開発者のソースコードを自動で修正するので，そのアプリケーションの論理的設計の不備を修正することが可能であることである．したがって VHF は通常の Web アプリケーションフレームワークでは対策できない認証の不備に対する攻撃の低減を行うことが可能である．

VHF は実行開始時にアプリケーション開発者が実装したソースコードを修正するシステムとリクエストを処理するシステムの 2 つで構成されている．ソースコードを修正するシステムは Web アプリケーション開発者が実装したソースコードを解析し修正する機能である．VHF は実行開始時にアプリケーション開発者が実装したソースコードをフレームワーク内に格納し，その後格納したソースコードを解析・修正する．リクエストを処理するシステムはクライアントからリクエストを受け取りレスポンスを作成して応答するシステムである．具体的にはまず受け取ったリクエストをアプリケーションが処理しやすい形式に変更する．次にそのリクエストを用いてレスポンスボディを作成する．最後にレスポンスを作成する．リクエストに基づいてレスポンスヘッダーを作成し，レスポンスボディと組み合わせてレスポンスを作成する．作成されたレスポンスはクライアントに返される．

本論文では Web アプリケーション開発者が実装したソースコードを解析して脆弱性の影響を緩和することが可能であるかを確認するために実験を行った．その結果，不適切な認証と SQL インジェクションの脆弱性を修正できることを確認した．

第2章

関連研究

2.1　論文1

2.2　論文2

2.3　論文3

第3章

提案手法

VHF は，コールバック関数を解析し必要であれば修正する．コールバック関数とは Web アプリケーションへのリクエストを基に，Web サーバー側で行う処理を記述した関数である．VHF は，リクエストのメソッドとパスによってコールバック関数を呼び出す．リクエストのパスとメソッドが一致するコールバック関数が存在する場合そのコールバック関数が呼び出され，その戻り値をレスポンスボディとしてクライアントに送信する．一致するコールバック関数が存在しない場合 VHF が HTTP ステータスコードを 404，レスポンスボディを Not Found にしてクライアントにレスポンスを送信する．

VHF は実行開始時に 4 つの工程によってコールバック関数を修正する．まず最初に，コールバック関数，リクエストパス，メソッドを一つの辞書式データとして VHF に格納する．この辞書式データはリストの一要素として VHF に格納される．この時点ではコールバック関数は活動中のオブジェクト，つまり実行可能なバイナリ形式のオブジェクトである．次に

コールバック関数を修正しやすくするために，VHF はコールバック関数の形式を活動中のオブジェクトから抽象構文木（Abstract Syntax Tree: AST）に変更する．AST は，プログラムを実行可能なバイナリ状態にする処理の途中で取得される中間生成物であり，ソースコードから実行可能なオブジェクトを生成するために必要ない部分を削除した表現である．AST はバイナリよりもプログラムの論理的設計を把握しやすいため，コールバック関数の解析と修正が容易である．3 つ目がコールバック関数の解析と修正である．VHF はコールバック関数を解析し修正する脆弱性ハンドリング関数を持っている．脆弱性ハンドリング関数は格納されたコールバック関数のリストを引数として受け取りコールバック関数のリストを返す．脆弱性ハンドリング関数はコールバック関数のリストを受け取ると，脆弱性ハンドリング関数が持っている脆弱なコールバック関数の条件とその条件を満たすコールバック関数の修正方法に基づき脆弱性を検出・修正する．脆弱性ハンドリング関数はコールバック関数のリストを受け取るため，コールバック関数間の論理的な設計の誤りを脆弱性の条件にすることが可能である．具体的には，あるコールバック関数では権限を求めたページを他のコールバック関数でも同じ権限が必要になるように修正することが可能になる．この手法を用いることで，従来では自動で対策が困難な不適切な認証の不備を低減できる．また脆弱性ハンドリング関数は脆弱性ごとに複数の関数で実装されている．この実装方法により新たな脆弱性が発見された時，その脆弱性の影響を低減する新たな関数を追加で実装することが可能である．修正が終わると、VHF はコールバック関数のリストを返す．最後にコールバック関数を実行可能なオブジェクトに変更することで，修正されたコールバック関数がリク

エストを処理することが可能になる.

第4章

実装

この章では，VHF の実装について記述する．VHF は Python3.7 によって実装されている．VHF は 2 つのシステムによって構成されている．コールバック関数を修正するシステムとリクエストを処理するシステムである．

4.1　コールバック関数の修正機能

VHF は実行開始時にコールバック関数を解析・修正する．この時に行われる処理はコールバック関数の格納とコールバック関数の修正の 2 つである．

4.1.1　コールバック関数の格納

VHF はコールバック関数を格納するためにデコレータを利用するためのメソッドとして route メソッドを持つ．実効開始時に route メソッドはコールバック関数を VHF 内のリストに格納する．以下のソースコードは

コールバック関数の例である．

```
1 @app.route(path='^/$', method='GET')
2 def index(request):
3   return "Hello"
```

Listing 4.1: コールバック関数の一例

ソースコード 4.1 の 1 行目がデコレータである．デコレータは関数を修飾する関数であり，下記のソースコード 4.2 はソースコード 4.1 と糖衣構文である．デコレータを利用することで関数を引数にする関数の記述を簡易にしてくれる．

```
1 def index(request):
2   return "Hello"
3 index = app.route(path="/", method="GET")(index)
```

Listing 4.2: ソースコード 4.1 と糖衣な表現

ソースコード 4.1 の 1 行目にある app は VHF のモジュールであり，route は app モジュールが持つメソッドの一つである．route メソッドはリクエストパスとリクエストメソッドを引数としている．ソースコード 4.1 の path がリクエストパスの正規表現，method がリクエストメソッド，2 行目と 3 行目の関数が第 3 引数のコールバック関数である．コールバック関数は request を引数として受け取る．request はリクエストの情報を格納している変数である．ソースコード 4.1 の 3 行目は戻り値であり，この戻り値はその後レスポンスボディになる．route メソッドはリクエストパスとリクエストメソッドをコールバック関数と対応付けて VHF に格納す

る．以下のソースコード 4.3 が route メソッドの実装である．

```
1 class App():
2   ...
3     def route(self, path=None, method='GET'):
4         def decorator(callback_func):
5             self.router.add(method, path,
                callback_func)
6             return callback_func
7         return decorator
```

Listing 4.3: route メソッド

route メソッドを実行すると route メソッド内部の decorator 関数を返す．その後コールバック関数を引数とした decorator 関数が実行される．decorator 関数内の router.add メソッドはコールバック関数を VHF に格納するメソッドである．decorator 関数はコールバック関数を引数に取り，decorator 関数内にあるリクエストメソッドとリクエストパスを一つにした辞書形式にして VHF 内に格納する．

4.2　コールバック関数の修正

VHF に格納された時点ではコールバック関数は活動中のオブジェクト，つまり機械語である．機械語を解析・修正するのは容易ではないため格納されたコードを一度修正しやすい形式に変換する．具体的には，活動中のオブジェクトを AST に変換する．Python は，AST を処理するライ

ブラリとして ast モジュールを提供している．このモジュールは，ソースコードを AST に変換したり AST を探索したりするメソッドを持っている．parse メソッドは ast モジュール内にある，ソースコードを AST に変換するメソッドである．parse メソッドはソースコードを引数として受け取るため，格納された時点での活動中のオブジェクトとしてのコールバック関数を受け取ることはできない．したがって，一度コールバック関数をソースコードに変換したのち，parse メソッドを利用してコールバック関数をソースコードから AST に変換する．

その後 VHF は AST になったコールバック関数を解析・修正する．VHF はコールバック関数を修正する関数として脆弱性ハンドリング関数を実装している．脆弱性ハンドリング関数は AST であるコールバック関数のリストを引数に取り，解析・修正された AST であるコールバック関数のリストを返す．脆弱性ハンドリング関数は特定の AST のノードを脆弱性とみなしている．脆弱性ハンドリング関数は脆弱性と判断したノードがコールバック関数内にあるかを解析し，検出されたノードは脆弱性ハンドリング関数内で定義したノードの修正方法に基づいて修正される．脆弱性ハンドリング関数は脆弱性ごとに分割して構成されており，VHF に AST であるコールバック関数をひとまとめにしたリストとして格納されている．コールバック関数のリストは順に脆弱性ハンドリング関数によって解析・修正される．脆弱性ハンドリング関数は分割して構成されているので，新しい脆弱性が見つかるたび追加の脆弱性ハンドリング関数を実装することが可能である．この実装により，既存の脆弱性ハンドリング関数を変更せずにその脆弱性を対策することが可能である．

最後にコールバック関数を活動中のオブジェクトに変更する．AST を

実行可能な形式に変更するために，exec() 関数を利用する．これにより，コールバック関数は活動中のオブジェクトとなり，リクエストを処理することが可能になる．

4.3　リクエスト処理システム

リクエスト処理システムはリクエストを基にコールバック関数を呼び出しレスポンスを作成するシステムである．このシステムはリクエスト情報の取得，コールバック関数の呼び出し，リクエストの作成の 3 つの機能から構成されている．

4.3.1　リクエスト情報の取得

リクエスト情報の取得は，クライアントからの HTTP リクエストを取得し，アプリケーションが処理しやすいように加工する機能である．

4.3.2　コールバック関数の呼び出し

4.3.3　レスポンスの作成

第5章

実験

本研究では二つの実験を行った．一つ目に脆弱性ハンドリング関数が脆弱性の影響を低減したかどうかの評価を行った．二つ目に脆弱性ハンドリング関数が実装されたことによる実行開始時のオーバーヘッドの計測を行った．

5.1　脆弱性の影響低減評価

脆弱性ハンドリング関数を実装することにより脆弱なコールバック関数を修正し，脆弱性攻撃への影響を低減することが可能か評価した．本実験では二つの脆弱性を持つアプリケーションを一つ実装した．実装されたアプリケーションが持つ二つの脆弱性は SQLi と不適切な認証のである．この脆弱性に対して，それぞれ脆弱性ハンドリング関数を実装した．その後，ローカル上でアプリケーションを実行し，攻撃することで脆弱性攻撃の影響を低減できたか評価した．本実験は以下の環境で行われた．Mac OS X El Capitan 10.11.6, Intel Core i5(2.95GHz), メインメ

モリ 8GB. 以下には，それぞれの脆弱なコールバック関数と脆弱性ハンドリング関数を記述する．

5.1.1　SQLi を持つコールバック関数の修正

SQLi はリクエスト内の値を利用して直接クエリを作成することで起こる脆弱性である．リクエストに特殊文字を挿入することで，アプリケーション開発者が意図していない命令がデータベースで実行される．これにより，データベースが改ざんされたり不正に削除されたりする．下記のコールバック関数は SQLi の脆弱性を持つコールバック関数である．

```
1  @app.route("^/access$", "POST")
2  def access(request):
3    import sqlite3
4    conn = sqlite3.connect("test.sqlite3")
5    cur = conn.cursor()
6    action = request.forms.get('action')
7    name = request.forms.get('name')
8    password = request.forms.get('password')
9    query = '{action} * from user'.format(action=
       action)
10   if action='select':
11     query += " where name = '{name}' and password =
         'password'".format(name=name, password=
         password)
```

```
12      cur.execute(query)
13      data = cur.fetchone()
14      return tmpl("access.html", action='select',tel=
            data[2],mail_address=data[3])
15    else:
16      cur.execute(query)
17    return tmpl("access.html", action=action)
```

Listing 5.1: SQLi 脆弱性を持つコールバック関数

上記のソースコード 5.1 の 1 行目は，コールバック関数を格納するメソッドである．リクエストパスが”/access”でリクエストメソッドが POST の時，このコールバック関数が呼び出される．2 行目以降の access() 関数がコールバック関数である．ソースコード 5.1 の 3 行目から 5 行目がデータベースと接続する準備である．3 行目でリレーショナルデータベースとして sqlite3 をインポートしている．4 行目でデータベースに接続し，5 行目でカーソルを宣言している．その後ソースコード 5.1 の 6 行目から 8 行目では，クエリを作成するために必要な情報をリクエストパラメータから取り出している．取り出される変数は action, name, password である．action は SQL のコマンド，name はユーザー名，password はユーザーのパスワードである．ソースコード 5.1 の 9 行目と 11 行目でこれらの変数を利用してクエリを作成する．ソースコード 5.1 の 12 行目で作成したクエリがデータベースで実行される．

上記のソースコード 5.1 は，リクエストパラメータを直接利用してクエリを作成しているため SQLi 脆弱性を持っている．このソースコードに対

して，下記の脆弱性ハンドリング関数を実装した．

hogehoge

5.1.2　不適切な認証を持つコールバック関数の修正

5.1.3　結果

5.2　オーバーヘッドの評価

VHF は実行開始時にコールバック関数を解析するため Web アプリケーションの実行よりも，

5.2.1　結果

第6章

考察

第7章

おわりに