情報処理学会論文誌 Vol.59 No.1 1–6 (Jan. 2018)

Webアプリケーションを安全にする新しいフレームワークの機能

久保田 康平[1,a)] 小出 洋[2]

受付日 2020年5月10日，再受付日 2015年7月16日 / 2015年11月20日，
採録日 2016年8月1日

概要：Web アプリケーション開発者が実装するコードを自動的に解析し，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．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ず，脆弱性が残る実装を残してしまうことがある．そのため，実装されたコードを自動的に解析し，修正する機能はフレームワークが持つべき機能のひとつである．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修正を行わないものとほとんど変わらないことを確認した．

キーワード：セキュアプログラミング，Web セキュリティ，Web アプリケーションフレームワーク

A New Framework Feature to Secure Web Applications

Kohei Kubota[1,a)] Hiroshi Koide[2]

Received: May 10, 2020, Revised: July 16, 2015/November 20, 2015,
Accepted: August 1, 2016

Abstract: We propose, implement, and evaluate a web application framework that automatically analyzes the code implemented by web application developers and modifies the code if necessary. Developers can't always write perfectly secure code, leaving implementations that remain vulnerable. Therefore, the ability to automatically analyze and modify the implemented code is one of the features that a framework should have. We demonstrated and evaluated the proposed method and confirmed that this feature can fix some vulnerabilities in the implemented code.

Keywords: secure programming, web security, web application framework

1. 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．その達成のために，開発者が実装したプログラム中の関数や引数を解析し，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 アプリケーションフレームワークは開発者に，Web アプリケーションのセキュリティを向上させる機能を提供している．具体的には，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）などの脆弱性に対する防御手法を，関数やモジュールとして開発者に提供している．したがって，開発者は Web アプリケーションフレームワーク上の関数を用いることで，それらの脆弱性が低減された Web アプリケーションを実装できる．しかし，開発者は時として Web アプリケーションフレームワークを利用して，脆弱性があるアプリケーションを実装することがある．開発者が，Web アプリケーションフレームワークが提供する関数やモジュールを，適切に利用できないことがあるためである．これは開発者が実装したプログ

[1] 九州大学システム情報科学府
Graduate School and Facility of Information Science and Electrical Engineering, Kyushu University
[2] 九州大学情報基盤研究開発センター
Research Institute for Information Technology, Kyushu University
[a)] kouhei.kubota@icloud.com

ラム中の脆弱性を，Web アプリケーションフレームワークが解消できないという問題である．この問題は，もし全ての脆弱性への対策を提供している Web アプリケーションフレームワークが存在し，そのフレームワークを開発者が利用しても，脆弱性があるアプリケーションを開発しうることを意味している．また，学習コストの高さも問題である．Web アプリケーションフレームワークは，それぞれが独自の関数を開発者に提供しているので，開発者は異なるフレームワークを利用する際にそれぞれのフレームワークが提供している関数を知っている必要がある．セキュリティについても同様で，開発者の知識不足や誤解が，脆弱性がある Web アプリケーションを実装する要因になる．

これらの問題を解決するために，本論文では開発者が実装したソースコードを修正し，必要に応じて修正する Web アプリケーションフレームワークが備えるべき機能を提案する．フレームワークは，開発者が実装したコールバック関数中に存在する脆弱な関数を安全な関数に変換したり，その関数の引数に対して入力検証を行う関数を自動で挿入したりすることで，脆弱性を取り除くことができる．

このフレームワークの機能は，Python[1] を用いて実装された．Python は関数を引数や戻り値に取れる言語であることから本論文の手法を実装するのに適した言語の一つである．このフレームワークは，具体的には 4 つの工程によりプログラムを修正する．まず活動中のオブジェクトから，開発者が実装した関数のソースコードを取得する．開発者が実装した関数とは，クライアントのリクエストヘッダに基づいて呼び出されるコールバック関数である．このフレームワークは，コールバック関数を実行可能な形式で受け取り，その関数のソースコードを組み込み関数を利用してフレームワークに格納する．次にそのソースコードを抽象構文木 [2] に変換する．その後，その構文木を修正する．そして最後に実行可能なオブジェクトに変換しなおすことで，ソースコードを修正する．この工程により，開発者が実装した関数はフレームワークに修正されたのちに，フレームワークに格納される．

検証として，この Web アプリケーションフレームワークの機能を利用して，開発者が実装したコードに脆弱性がある Web アプリケーションを実装して評価を行った．結果として，組み込み関数によってできた SQLi 脆弱性を，フレームワークが自動で修正できることがわかった．

2. 関連研究

Web アプリケーションフレームワーク上で実装したソースコードを自動的に解析する技術として，[4] がある．この論文は，Web アプリケーションフレームワーク上のソースコードをセキュアにするという目的と，そのソースコードを自動で解析することができる点で類似している．この論文ではシンボリック実行を利用して，到達可能性を調べることにより脆弱性を探索し，脆弱性と判定されれば，開発者に警告を返す．このシステムの長所は，既存の Web アプリケーションフレームワークである Ruby-on-Rails[5] のソースコードの解析を行うことができることである．

抽象構文木を用いたセキュリティに関する研究として [6] があげられる．この論文は抽象構文木の特徴を利用して難読化された JavaScript の悪性を判断する手法を提案している．抽象構文木はプログラム内の無意味な部分を除去することができるので，プログラムの意味を抽出しやすいことと，抽象構文木を実装するために，動的な解析を必要としないので高速な点が長所である．抽象構文木は，即時性を求められる Web プログラミングの解析に向いた手法だと言える．

3. 提案手法

コールバック関数を修正する機能は，Web アプリケーションフレームワーク開発者にとって編集しやすいことが望ましい．もしコールバック関数を直接修正しようとすると，Python の実行可能なオブジェクトはバイナリデータであるため，バイナリである関数を引数にとって，それを編集する関数をフレームワーク上に実装する必要がある．これは編集を行いやすいとは言えない．このフレームワークはコールバック関数の修正を簡単にするために，抽象構文木を引数に取り，その抽象構文木を修正することで，コールバック関数を修正する手法を提案する．抽象構文木は，プログラムにとって意味がない部分をソースコードから取り除いているため，プログラムの意味を修正しやすい特徴がある．また，構文木のノードには型が存在するため，その型に応じた修正を行うことで，自動的に修正する関数を実装してもソースコードを編集するのと比較して，バグを作りにくい特徴がある．

加えて，このフレームワークはデコレータ [3] を利用して開発者が抽象構文木を修正する関数を実装し，コールバック関数に適用することが可能である．デコレータとは，ソースコードの可読性をあげることができるようにする関数を修正する関数の表現方法である．フレームワークが提供している抽象構文木の自動修正を行ったのち，開発者が追加した修正を行う．この手法により開発者は以下のような実装を行うことで，コールバック関数を修正させることができるようになる．

```
1 @app.funcFixer.addNodeFixer
2 def rewriteNode(node):
3     new_node = Rewrite().visit(node)
4     return new_node
5
6 @app.route("^/$", "GET")
7 @app.funcFixer.fix("rewriteNode")
```

```
8  def index(request):
9
10     return "Hello"
```

行頭の@がデコレータを表している．したがって，1 行目から 4 行目は，
app.funcFixer.addNodeFixer(rewriteNode)
と等価である．

このソースコードでは，1 行目から 4 行目がアプリケーション開発者が実装できるコールバック関数の抽象構文木を修正する関数であり，5 行目から 10 行目がコールバック関数である．1 行目の app.funcFixer.addNodeFixer は，フレームワーク内にコールバック関数の抽象構文木を修正する関数を格納するメソッドである．コールバック関数を修正する関数は addNodeFixer メソッドによってフレームワークに渡され，関数名とコードオブジェクト辞書形式で格納される．2 行目の関数の宣言における引数の node はフレームワークから与えられるコールバック関数の抽象構文木である．3 行目の Rewrite は Python が提供する抽象構文木を修正するヘルパーである ast[7] モジュールのうち，ast.NodeTransformer の子クラスであり，そのインスタンスの生成している．この Rewrite クラスに，抽象構文木を修正する条件や修正する方法を記述する．また，visit メソッドは抽象構文木の探索と修正を行うメソッドである．

コールバック関数の実体は，8 行目から 10 行目までであり，引数の request はクライアントから送信されたリクエスト情報をコールバック関数内で利用しやすいように整形したものである．6 行目は，コールバック関数をフレームワークに格納するデコレータで，第一引数がパスの正規表現，第二引数がリクエストメソッドである．この二つの引数を満たすリクエストが送信されると，コールバック関数が呼ばれる．7 行目がコールバック関数を修正するデコレータである．これは修正関数名を受け取り，その名前に一致する，addNodeFixer メソッドによってフレームワークに格納された関数を利用して，コールバック関数を修正するデコレータである．このデコレータは複数の引数を受け取ることができ，複数の修正関数によって抽象構文木を修正したのち，修正された抽象構文木をコードオブジェクトに変換する．

4. 実装

図 1 は，このフレームワークがコールバック関数を修正するために必要な工程を表している．

図 1 に示す通り，Web アプリケーション開発者が実装したコールバック関数を抽象構文木に変換して，解析・修正するために 4 つの工程が必要である．一つ目が，コードオブジェクトからテキストを受け取ることである．これは，Python が提供している抽象構文木に対するヘルパーである ast モジュールが，実行可能なバイナリデータを直接抽象構文木に変換できないためである．コードオブジェクトからテキストを受け取るために，inspect[8] モジュールを利用する．inspect モジュールは，活動中のオブジェクトから情報を取り出すモジュールである．このモジュールのうち，getsource メソッドを利用して関数のソースコードを取得する．getsource メソッドは，ソースコードの該当部分をテキストファイルとして読み込むことで，ソースコードを取得する．動的に実行したコードオブジェクトから，ソースコードを取得しようとすると OSError を起こすことがあるため注意が必要である．特に，フレームワークによって修正された関数は動的に生成された関数であるため，ソースコードを取得できない．ゆえに，このフレームワークが一度修正した関数をもう一度修正することはできない．

二つ目に，関数のソースコードを抽象構文木に変換する．ソースコードを構文木に変更できる形に整形する．具体的にはソースコードの先頭にあるインデントの分だけ各行のインデントを取り除く．これはソースコードの先頭にインデントがある状態では，抽象構文木を作成できないためである．

三つ目は構文木を解析・修正する工程である．この工程では，まず抽象構文木からコールバック関数のデコレータ部分を取り除く．これは，ソースコードを取り出した際にデコレータを含んでおり，動的に生成されたコードオブジェクトのソースコードを取得しようとしてエラーを起こすためである．この工程でデコレータを取り除く理由は，コールバック関数を修正するデコレータではコールバック関数内のデコレータを取り除かないようにするためである．抽象構文木を利用すると，モジュール名とメソッド名からデコレータを取り除くことができるため，誤りが少ない．デコレータを取り除いたのちに，関数の内容を修正する．この工程はアプリケーションフレームワークで自動的に行われる修正を行い，その後 Web アプリケーション開発者が実装した修正を行う．

最後に修正された構文木を活動中のオブジェクトに変換する．Python には抽象構文木を実行可能なオブジェクトに変換する exec[9] という組み込み関数がある．この関数を利用してできた関数はデフォルトでは，ローカル変数であるため注意が必要である．

以下の工程を経てコールバック関数は修正され，フレームワーク内に格納される．

5. 実験と結果

本論文が提案するフレームワークを評価するために，意図的に脆弱なコールバック関数を実装して，その脆弱性をフレームワークが修正できているかを調べる実験を行った．また，コールバック関数を修正した時としてない時の，リクエストを送ってレスポンスを受け取るまでの時間を計測

図 1 コールバック関数を修正する工程の簡易図.
Fig. 1 Simplified diagram of the process to fix a callback function.

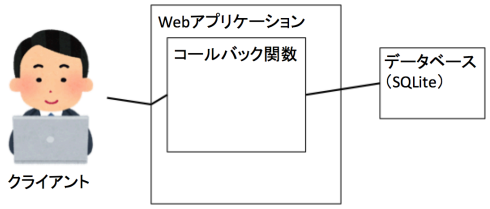

図 2 実装したアプリケーションの簡易図.
Fig. 2 Simplified diagram of the implemented application.

した．実験に使用した実装とその結果を以下に示す．

5.1 実験

この実験は，Python3.7 上で実装されている．アプリケーションは図 **2** に示す構成である．

5.1.1 コールバック関数の修正

コールバック関数はデータベースに接続しており，リクエストパラメータからクエリを生成して，データベースに問い合わせ，データベースから受け取った値をブラウザに返す．クライアントは ID とパスワードを入力するフォームから，これらの情報を入力し，サーバに送信する．データベースには，ID とユーザ名，パスワードを持つテーブルがある．ID は，主キーに設定しており同一の ID は存在しないものとする．このテーブルに対して SQL 文を実行し，ID とパスワードが一致するレコードを取得し，そのうちユーザ名と ID をブラウザに返す．

```
1 cur.execute("select␣*␣from␣table1␣" + \
2 "where␣ID␣=␣" + request.params.get["ID"]\
3  + "␣and␣Password␣=␣"\
4  + request.params.get("Password"))
```

上に示すソースコードは，SQLi 脆弱性があるコールバック関数の例のうちの一部である．cur.execute という SQL 文を実行させるメソッドを利用して，データベースから情報を取り出す．この時，クエリを文字列連結によって組み立てて，そのままデータベースに送っている点に SQLi の脆弱性がある．例えば，ID に’1 or 1 == 1 order by desc;–’，パスワードを空白にした入力を行うと，生成されるクエリは，”SELECT * from table1 where ID = 1 or 1 == 1 order by desc;– and password = ”である．このクエリはパスワード部分をコメントアウトにより，条件にしないため，テーブル内の ID が最も下にあるレコードを取り出すクエリになってしまう．

このような脆弱性に対して，コールバック内の SQL 文を実行する関数の引数をエスケープすることで，脆弱性を低減するように修正する関数を実装した．具体的には，SQL を利用するモジュールを探し出し，そのメソッド名が execute である関数を探索し，その引数の内，リクエストパラメータを利用したものを修正する関数を実装した．引数を修正する際，コールバック関数内の変数がどのリクエストパラメータを利用しているかを代入する時に調べて，リクエストパラメータ部分だけを修正した．

5.1.2 ベンチマーク

ローカル上にコールバック関数を修正する機能を利用したアプリケーションと利用してないアプリケーションをを立ち上げ，レスポンスタイムを測定した．コールバック関数を修正していないアプリケーションと修正したアプリケーションのレスポンスタイムを，cURL コマンドを用いてそれぞれ 100 回実行して計測した．

5.2 結果

結果として，まず一部のコールバック関数を修正し，脆弱性の一部を低減することができることが分かった．

```
1  import sqlite3
2
3
4  @app.route("^/$", "GET")
5  @app.addNodeFixer("rewriteExecuteArgs")
6  def func1(request):
7      conn = sqlite3.connect("test1.db")
8      cur = conn.cursor()
9      cur.execute("select␣*␣from␣table1" \
10      + "where␣ID␣=␣" \
11      + request.params.get("ID") \
12      + "␣and␣Password␣=␣" \
13      + request.params.get("Password"))
14
15     data = cur.fetchone()
       conn.commit()
```

```
16      return "Hello, " + data[1] \
17       + ". Your ID is " + data[0]
```

このコールバック関数は，リクエストのパラメータから ID とパスワードを取得し，その情報を元にデータベースを検索し，ID とユーザ名をクライアントに返す．コールバック関数を修正する関数は 8 行目から 12 行目を修正した．具体的には，cur.execute メソッドの引数のうち，request.params.get("ID") と request.params["Password"] を，それらをエスケープした変数に差し替えた．

このコールバック関数について以下にソースコードの説明を記述する．3 行目か 5 行目は，提案手法で説明したデコレータと関数の宣言である．6 行目と 7 行目でコールバック関数は，データベースと利用できるようにしている，具体的には，6 行目でデータベース内のテーブルに接続しており，7 行目でそのテーブルを操作できるようにしている．8 行目の cur.execute メソッドは引数にクエリを取り，そのクエリを SQL で実行する．10 行目の fetchone メソッドは，SQL の実行結果のうち，最初に取り出されたデータをタプルの形で出力する．11 行目の commit メソッドはデータベースを更新するメソッドである．その後，SQL が返した値を整形してクライアントに返している．

下に示す二つ目のソースコードを修正できたことによって，cur.execute メソッドの引数に直接リクエストパラメータが入ってはいない場合でも修正することが可能であることがわかった．

```
1  import sqlite3
2
3  @app.route("^/$", "GET")
4  @app.addNodeFixer("rewriteExecuteArgs")
5  def func1(request):
6      conn = sqlite3.connect("test1.db")
7      cur = conn.cursor()
8
9      s1 = "select * from table1"
10     s2 = " where ID = {}" \
11      + request.params.get("ID")
12     s3 = " and Password = {}" \
13      + request.params["Password"]
14     s4 = s1 + s2 + s3
15     cur.execute(s4)
16     data = cur.fetchone()
17     conn.commit()
18     return "Hello, " + data[1] \
19      + ". Your ID is " + data[0]
```

一つ目とのソースコードの違いは，9 行目から 15 行目である．一つ目のソースコードは，cur.execute メソッドの引数に直接リクエストパラメータが入っていたが，今回は入っていない．変数の代入を調査し，リクエストパラメータを利用した変数を利用することで，修正することができることが確認された．

コールバック関数を修正する関数の実装で脆弱性の対策ができる実装例がある一方で，脆弱性を修正できない場合もあった．それは，ループによってクエリを動的に作成する場合である．リクエストパラメータをイテレータとし，各処理でイテレータから要素を変数に代入するが，その代入が通常の代入と異なるために，リクエストパラメータを利用していない扱いになっていたからであった．

レスポンスタイムは修正された関数と修正されていない関数の場合で大きく変わらないことが測定から分かった．結果は，提案手法を使わない場合のレスポンスタイムが 4.98ms/req. で，提案手法を適用した場合が 5.28ms/req. であったため，オーバーヘッドは 6.09%程度であった．

6. 考察

実験の結果から，このフレームワークはコールバック関数を修正し，一部の脆弱性を低減させることが確認できた．今回は，SQLi を修正する関数を実装して実験を行ったが，他の脆弱性に対しても修正できると考えられる．このフレームワークはコールバック関数を修正する関数を追加することができるからである．これにより，広範なセキュリティ機能をアプリケーション開発者がほとんど意識することなく利用することが可能になると考えている．

また，レスポンスタイムの比較から，オーバーヘッドは 6.09%と，本論文で提案する手法がレスポンスに大きな影響を与えないことが分かった．修正後の方がレスポンスタイムが少し大きいのは，修正前にはない処理をコールバック関数に追加しているからだと考えられる．この時間の追加は，コールバック関数をセキュアにするために必要なレスポンスタイムの追加であると考えられる．

加えて，本提案手法は異なる修正関数を定義して，コールバック関数を複数回修正できる．本論文では実装していないが，例えば XSS は，コールバック関数の戻り値を解析・修正する関数を実装することで対策することができると考えている．

このフレームワークには長所がある一方で制限があることも分かった．その一つが，コールバック関数内で修正すべき箇所を適切に見つけるのが難しいことである．その理由は，Web アプリケーション開発者によってソースコードの記述に幅があるからである．アプリケーションフレームワーク開発者が，同じ意味になるソースコードの記述全てに対して，解析・修正する関数を実装するのは難しい．

6.1 課題

本論文では，Web アプリケーションフレームワークが

コールバック関数を自動で修正する．それにより，フレームワークが脆弱性だと判断したら，それがもし脆弱性ではなかったとしても修正されてしまう．したがって今後，脆弱性を正しく判定するために，修正関数内の判定方法や判定を行いやすくするヘルパーの実装について考察する必要がある．

また，今後，修正関数が増えると，オーバーヘッドも増加すると考えられる．ゆえに修正関数とそれに伴うオーバーヘッドの関係性について調査する必要がある．

6.2 今後の展望

本論文が提案するフレームワークは，コールバック関数を修正する関数をフレームワークで定義しておけば，Web アプリケーション開発者がコールバック関数上で見逃す脆弱性を修正できる，というものである．この特徴により，コールバック関数に未知の脆弱性に対して，フレームワーク開発者が修正関数を実装すれば，アプリケーション開発者はフレームワークをアップデートをするだけで，脆弱性の対策ができるようになると考えている．今後，フレームワークに様々なコールバック関数を修正する関数を追加していくことで，現在発見されていない脆弱性を含めた広範なセキュリティ機能を持つ Web アプリケーションフレームワークになっていくと考えている．

また本提案手法は単一のコールバック関数に対して，複数の修正関数を適用できるので，脆弱性を複合的に低減できるようになると考えている．これは，複数の Web アプリケーションフレームワーク開発者が各々の修正関数を定義し，このフレームワーク上に実装を行うことで，フレームワーク開発者ごとの手法を用いて脆弱性を低減できるという長所だと言える．

7. おわりに

本論文では，アプリケーションが持つべき機能として，コールバック関数を自動的に解析し，必要ならば修正する機能を提案し，実装したのちに評価した．この実装と評価から SQLi 脆弱性を一部対策できること，その他の脆弱性に関しても修正する関数を実装することによって対策可能であることが分かった．一方で，この実装は改良が必要なことも分かった．今後改良をしていくことで，このアプリケーションフレームワークを利用して，セキュアなアプリケーションを簡単に実装できるようになると考えている．

謝辞 本研究は，国立研究開発法人科学技術振興機構 (JST) の助成を受けている．また，日立システムズとの共同研究の一部である．

参考文献

[1] Python Software Foundation: python, 入手先 ⟨https://www.python.org/⟩（参照 2020-05-12）
[2] Brett Cannon: PEP 339 – Design of the CPython Compiler, 入手先 ⟨https://www.python.org/dev/peps/pep-0339⟩（参照 2020-05-12）.
[3] Python Software Foundation：用語集, 入手先 ⟨https://docs.python.org/ja/3.7/glossary.html#term-decorator⟩（参照 2020-05-14）
[4] A. Chaudhuri and J. S. Foster: Symbolic Security Analysis of Ruby-on-rails Web Applications, CCS(2010), pages 585–594.
[5] Ruby on Rails, 入手先 ⟨https://rubyonrails.org/⟩（参照 2020-05-14）
[6] 神薗雅紀， 西田雅太， 小島恵美， 星澤裕二：抽象構文木による不正な JavaScript の特徴点検出手法の提案，情報処理学会論文誌，Vol. 54, No. 1, pp. 349-356 (2013).
[7] Python Software Foundation：ast — 抽象構文木, 入手先 ⟨https://docs.python.org/ja/3.7/library/ast.html⟩（参照 2020-05-12）.
[8] Python Software Foundation：inspect — 活動中のオブジェクトの情報を取得する, 入手先 ⟨https://docs.python.org/ja/3.7/library/inspect.html⟩（参照 2020-05-13）
[9] Python Software Foundation：組み込み関数，入手先 ⟨https://docs.python.org/ja/3.7/library/functions.html⟩（参照 2020-05-13）