Web アプリケーションを防御する新しい機能

久保田　康平

2020 年 8 月 11 日

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 アプリケーションフレームワークはアプリケーション開発者に，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．したがって，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上の関数を用いることで，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，アプリケーション開発者は時として 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．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．

上記の問題に対して Web アプリケーションフレームワークの自動防御手法が提案されている．J. Weinberger らの論文 [1] では，クロスサイトスクリプティング脆弱性 (XSS) の対策として Web アプリケーションフレームワークが自動でクライアントの入力を無害化する自動サニタイズについて書かれている．自動サニタイズによって Web アプリケーション開発者は XSS を対策する実装をすることなく，XSS の対策を適切に行うことが可能になる．一方で自動サニタイズの多くは XSS や SQL インジェクション (SQLi) の対策である．したがって多くの Web アプリケーションフレームワークは認可制御不備の脆弱性を自動的に防御する機能を持っていない．

Web アプリケーションの脆弱性を低減する手法の研究として，Web アプリケーションファイアウォール（WAF）の研究 [2, 3] がある．WAF は，XSS や SQLi などの Web アプリケーションの脆弱性を利用する攻撃から Web アプリケーションを保護するシステムである．Web アプリケーションとクライアント間の通信を監視し，攻撃である通信を検出した時，その通信を遮断したり修正したりする．WAF は，アプリケーションを修正することなく，脆弱性に対する攻撃を低減することができるため，アプリケーションを直接修正できない状況の時に，特に有効である．一方で WAF では対策しづらい攻撃が存在する．WAF はリクエスト内の特殊文字によって攻撃かどうかを判断するが，主に認証や認可について脆弱性がある Web アプリケーションに対する攻撃は，特殊文字を利用することなく行うことができるため，WAF では防御しづらい．

これら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．アプリケーションの実行時に，アプリケーション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，脆弱性を低減するものである．この機能はアプリケーション開発者が，脆弱性の解析のために追加の実装を行うことなく，アプリケーションを動的

に検証する関数や変数を追加することができる．ここでのコールバック関数とはアプリケーション開発者が実装し，Web アプリケーションフレームワークに格納される関数である．この関数はパスやメソッドに紐づけられていてパスやメソッドに応じてフレームワークから呼び出される．コールバック関数を修正する機能により，これまでの Web アプリケーションフレームワークや WAF では対策がしづらかった，管理者しか利用できない関数やデータについての脆弱性の対策をフレームワークが行うことができる．権限が必要なメソッドをコールバック関数内で呼び出した時に，フレームワークが，リクエスト情報を利用して，正しい権限を持っているかどうかを検証するメソッドをコールバック関数内に追加してくれることが可能だからである．加えて，この機能は実行時にアプリケーションを自動で修正するので，アプリケーション内の脆弱性を直接修正できる点で WAF とは異なる．

このフレームワークの機能は，Python を用いて実装された．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，アプリケーション開発者が実装した関数のソースコードを取得する．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 (AST) に変換する．その後，その AST 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．この工程により，アプリケーション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，アプリケーション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 と認可制御の不備という二つの脆弱性を低減できることがわかった．

2　提案手法

この章ではコールバック関数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記述する．このフレームワークが実行される時，4 つの過程を経てコールバック関数が修正される．

まずはじめにフレームワークにコールバック関数とルーティングに必要な情報が格納される．コールバック関数はルーティングに必要な情報とともに一つの辞書形式のデータになり，リストに格納される．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，つまりバイナリデータである．

次にコールバック関数を処理しやすくするために，格納されているコールバック関数を活動中オブジェクトから AST に変換する．AST はプログラムを実行可能なバイナリデータにするプロセスで得られる中間コードである．AST はノードで表され，各要素は意味のあるプログラムのコードオブジェクトを表している．AST の各ノードは意味のある名前または属性名を持っているため，バイナリを直接変更するよりも修正が容易である．また，AST はソースコードを修正するよりも修正が容易であるという利点がある．フレームワーク内でコールバック関数を修正する際，Web アプリケーションフレームワークの開発者はアプリケーション開発者が実装するコールバック関数を直接見ることができない．そのため，ソースコードを修正するために正規表現や文字列パターンを利用するのは困難である．AST のノードはソースコードから意味のない部分が削除されているため，解析と修正が容易である．

三つ目のステップとして，コールバック関数を修正する．コールバック関数を修正する，脆弱性ハンドリング関数をフレームワーク内に実装する．脆弱性ハンドリング関数はコールバック関数の AST を要素とするリストを引数として受け取り，再帰的に探索して脆弱性を検出する．その後検出された脆弱性を修正して AST のリストを返す．AST のリストを受け取ることで，脆弱性ハンドリング関数はコールバック関数間の内容の違いから脆弱性を検出できるという利点がある．

最後の工程として，修正された AST を活動中のオブジェクトに変換することで，リクエストに対して修正されたコールバック関数が呼び出されるようになる．

3　実装

この章では提案手法で示したコールバック関数を修正する 4 つの過程についての実装をそれぞれ記述する．一つ目が，コールバック関数を受け取り，フレームワークに格納することである．コールバック関数を受け取る際に，フレームワークはパスとメソッドを受け取る．これらのパスとメソッドの情報を元にコールバック関数は呼び出される．このフレームワークはパス，メソッド，コールバック関数を一つにまとめた辞書形式のデータにし，一つにまとめられたデータを要素としてリストに格納する．

二つ目に，コールバック関数を AST に変換する．Python が提供している AST 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 AST に変換できないため，コードオブジェクトからソースコードを取得する．コードオブジェクトからテキストを受け取るために，inspect 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．その後，Python の ast モジュールの parse メソッドを利用してソースコードを ast に変換する．

三つ目はコールバック関数の AST を解析し修正する工程である．コールバック関数を解析・修正する脆弱性ハンドリング関数は複数あり，リスト形式で格納されている．脆弱性ハンドリング関数が複数あるのは，脆弱性ハンドリング関数の保守性を高めるためである．コールバック関数の AST はそれぞれの脆弱性ハンドリング関数によって解析・修正される．フレームワークは脆弱性ハンドリング関数をリストから一つずつ取り出しそれぞれを実行する．脆弱性ハンドリング関数の引数は AST に変換されたコールバック関数のリストであり，解析・修正された AST のリストが脆弱性ハンドリング関数の戻り値である．

最後に修正された AST を活動中のオブジェクトに変換する．Python の AST を実行可能なオブジェクトに変換する exec という組み込み関数を利用する．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．

4　実験

脆弱なコールバック関数を持つ簡易的な Web アプリケーションを，本報告における Web アプリケーションフレームワーク上に実装し実験を行った．脆弱性は SQLi と認可制御の不備である．それぞれの脆弱性とその対策ついて記述する．このフレームワークの実装は Python3.7 上で実装された．実験を行った環境は，OS が Mac で OS X El Capitan 10.11.6, CPU が Intel Core i5 (2.95GHz), メインメモリが 8GB であった．

4.1　SQLi の修正

SQLi はデータベースを不正に利用される脆弱性である．SQLi はデータベースに利用されているリクエストの一部が適切に無害化されていないという脆弱性であり，SQLi 攻撃はこの脆弱性を利用して行われる脆弱性攻撃である．悪意のあるリクエストを Web アプリケーションに送信することで，そのアプリケーションに接続されたデータベースを不正に動作させ，データを取得したりデータベースを改ざんしたりする攻撃である．

SQLi を修正する実験として，SQLi 脆弱性があるコールバック関数を実装した．脆弱性は具体的には以下のコードである．

```
1 cur.execute('{query}'.format(action
    =request.params["query"]))
```

cur.execute メソッドが SQL を実行するメソッドである．request.params はリクエストパラメータである．このソースコードは request.params を無害化していないため脆弱である．この脆弱性を低減するために，cur.execute メソッドを検出し，テーブルを消去する"drop"命令と where 句が含まれない"select"命令がクエリに挿入されていると，そのクエリを空文字のクエリに変換する脆弱性ハンドリング関数を実装した．

4.2 認可制御の不備

認可制御の不備は権限が必要なデータを権限を持たないユーザが閲覧可能である脆弱性である．以下に脆弱なコールバック関数を示す．

```
1  @app.route("^/login$", "POST")
2  def do_login(request):
3      id = request.params["ID"]
4      password = request.params["
           PASSWORD"]
5      if is_admin(id, password):
6          return "ADMIN_PAGE"
7      else:
8          return "LOGIN_PAGE"
9
10 @app.route("^/home$", "GET")
11 def home(request):
12     return "ADMIN_PAGE"
```

/login を要求すると呼び出されるコールバック関数である do_login 内の is_admin 関数は ID とパスワードにより認証を行っており，ID とパスワードを知っている管理者のみが"ADMIN_PAGE"を閲覧可能になっている．is_admin 関数はこの Web アプリケーションフレームワークの組み込み関数であり，ID とパスワードが正しければ True，そうでなければ False を返す．このアプリケーションは/home を要求すると権限を持っていないユーザも"ADMIN_PAGE"を閲覧できてしまう点で脆弱である．この対策として，is_admin 関数が条件である戻り値（上記のコードでは"ADMIN_PAGE"）のうち，is_admin 関数による認証がない戻り値に対して，自動で同様の条件を追加する脆弱性ハンドリング関数を実装した．

5 結果

二つの脆弱性を対策するためのそれぞれの脆弱性ハンドリング関数は実験における脆弱なコールバック関数をそれぞれ修正した．SQLi に対しては drop を含むクエリが実行されていないことを確認した．認可制御の不備では，/home を要求すると，修正前は"ADMIN_PAGE"に遷移されたが，脆弱性ハンドリング関数を実装した後は"LOGIN_PAGE"へ遷移されることを確認した．この二つの結果からこのフレームワークは脆弱性の影響を低減できることがわかった．

6 まとめ

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 と認可制御の不備を一部対策可能であるとわかった．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．

参考文献

[1] J. Weinberger, P. Saxena, D. Akhawe, M. Finifter, R. Shin, and D. Song, “A systematic analysis of xss sanitization in web application frameworks,” in *European Symposium on Research in Computer Security*. Springer, 2011, pp. 150–171.

[2] C. Kruegel and G. Vigna, “Anomaly detection of web-based attacks,” in *Proceedings of the 10th ACM conference on Computer and communications security*, 2003, pp. 251–261.

[3] N. Epp, R. Funk, C. Cappo, and S. Lorenzo-

Paraguay, “Anomaly-based web application firewall using http-specific features and one-class svm,” in *Workshop Regional de Segurança da Informação e de Sistemas Computacionais*, 2017.