[2020]九州大学情報統括本部年報 : 2020年度

出版情報：九州大学情報統括本部年報. 2020, pp.1-, 2021-12-01. Information Infrastructure Initiative, Kyushu University
バージョン：
権利関係：

第 5 章　情報システムセキュリティ研究部門

5.1 スタッフ一覧

職名	氏名	研究キーワード
教授	小出 洋	サイバーセキュリティ、プログラミング、ネットワーク、並列分散処理、動的記憶領域管理

5.2 研究事例紹介

5.2.1「Web アプリケーションを安全にするフレームワークの新機能」

概要

情報システムセキュリティ研究部門では、情報システムを安全に構築する技術、情報システムをさまざまな攻撃から守る技術について研究をおこなっている。その研究成果の中から Web アプリケーションを安全にするフレームワークの新機能に関する研究について紹介する[14,15]。この研究は Web アプリケーションのセキュリティ機能の向上を目的としている。そのために本研究では、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、必要であれば修正する機能を Web アプリケーションフレームワーク[1-3]に持たせることを提案し、実装して評価までを行った。

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。その目的のため、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場合には修正を行う Web アプリケーションフレームワーク[1-3]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、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、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 (Web Application Firewall) [4-10]や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に情報システム上におかれるシステムである。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害化することにより、Web アプリケーションへの攻撃の影響を低減することができる。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そのものを修正することはしないため、アプリケーション内の脆弱性を根本的に修正することは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に関す情報は利用しないため、一部のタイプの脆弱性の対策は難しい。WAF は通常、パターンマッチングにより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、Cross Site Scripting（クロスサイトスクリプティング; XSS）[1]や SQL Injection(インジェクション；SQLi)[7,11]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[1]という機能を提供している。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り、一般的な 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担うことが可能なことである。自動サニタイズにより、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あり、インジェクション攻撃以外の攻撃の対策を行うことはできない。

例えば、一般的な 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。そのため、Web アプリケーションの不適切な認証に対する攻撃[12]については自動的に防御することができない。ここで不適切な認証とは、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[12]。

この問題を解決するため、我々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)を提案し実装して評価を行った。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つ。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、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害化する。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ため、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。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し、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。

本研究では 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評価実験を行った。その結果、不適切な認証と SQLi の脆弱性を修正できることを確認することができた。

2 関連研究

2.1 自動サニタイズに関する研究

自動サニタイズは Web アプリケーションフレームワークが提供する脆弱性の影響を低減する機能の一つである。この機能はフレームワークの特＝定の関数やメソッド内部で入力値を検証することで、アプリケーション開発者が実装することなく攻撃を無害化する機能である。Joel Weinberger らは、一般に普及している Web アプリケーションフレームワークの XSS に関する自動サニタイズを調査している[1]。本研究と Joel Weinberger らの研究との違いは、対策可能な脆弱性の幅である。Joel Weinberger らの研究は Web アプリケーションフレームワーク内の XSS を自動サニタイズする関数や機能について言及している。一方、本研究は Web フレームワークのコールバック関数内に自動サニタイズを行う関数を自動的に正しく挿入することで、脆弱性を自動で修正することを提案している。本研究はコールバック関数内で脆弱性の影響を低減する関数を適切に利用できているかを検証することでアプリケーションのうち、より論理的な設計に関する脆弱性の対策が可能である。

2.2 WAF に関する研究

WAF は Web アプリケーションのユーザと Web アプリケーションの間に配置される。Web アプリケーションユーザが送信したリクエストを検査することで Web アプリケーションへの攻撃を対策する。一般的な WAF はリクエストのみを利用して攻撃を検出するため、一部の攻撃を検出することができない。また WAF の検出を迂回する攻撃が存在し、検出率が低下することがある。この問題に対して Meixing Le らは、Web アプリケーションとデータベース間とリクエストを関連づけることで WAF を迂回する SQLi の脆弱性攻撃を対策する手法を提案している[7]。Meixing Le らの研究と本研究の類似点は、Web アプリケーションの情報を利用することで脆弱性攻撃を対策している点である。一方で本研究は、アプリケーション内の変数の検証を行うことが可能であるため、より多くのアプリケーションの情報を取得可能である。

3 提案手法

本節で我々は、コールバック関数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 を提案する。ここで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 サーバ側で行う処理を記述した関数である。図 3.1 は VHF を利用した Web アプリケーションの概要図である。VHF はコールバック関数の修正機能とリクエストの処理機能を持っている。VHF が実行されると、まず、コールバック関数を修正する。具体的には、実行開始時に VHF はコールバック関数を VHF 内に格納し、コールバック関数を修正する。その後実行中はクライアントからのリクエストに対して修正されたコールバック関数で処理を行うことで脆弱性の影響を低減する。

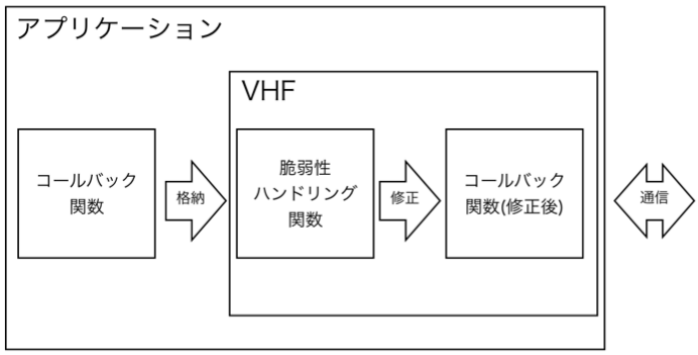

図 3.1: VHF の概要.

Figure 3.1: Abstract of VHF schematic diagram.

VHF はサイバーセキュリティの観点から 3 つの利点を持つ。第一は VHF が対策しているコールバック関数に関するセキュリティ機能の全てをアプリケーション開発者が負担しなくて良いことである。一般的な Web アプリケーションフレームワークにおける Web アプリケーションの実装では、Web アプリケーション開発者は安全なコールバック関数を実装する全責任を負う。VHF はコールバック関数を自動で解析・修正するので、VHF がコールバック関数の責任の一部を担うことが可能である。第二は、VHF は WAF で対策が難しい脆弱性攻撃を対策可能であることである。一般的な WAF はアプリケーションへのリクエストを解析することで脆弱性攻撃の影響を低減する。具体的には、リクエスト中の特殊文字や他のプログラミング言語に関する意味のある文字を脆弱性攻撃としそれらを検出する。この検出手法は Web アプリケーションを修正することなく Web アプリケーションを自動で防御できるが、リクエスト中に特殊文字を含まない攻撃を対策することが困難である。リクエスト中に特殊文字を含まない攻撃の 1 つが不適切な認証を持つアプリケーションに対する攻撃である。ここで不適切な認証は、Web アプリケーションの論理的な設計の間違いが原因で起こる脆弱性である。不適切な認証によって、正しいアクセス制御ができず、それにより特権を持たないユーザが特権リソースに接続する攻撃が発生する。VHF はコールバック関数間の論理的な設計を比較して他のコールバック関数に適用することで、不適切な認証の影響を低減することが可能である。最後に、VHF はコールバック関数を実行開始時に修正するので、脆弱性を根本的に対策することが可能である。WAF は、コールバック関数の脆弱性を削除できないため、特定の攻撃を対策してもその対策を迂回する攻撃が発生する可能性がある。

VHF は図 3.2 に示す通り、実行開始時に 4 つのステップによってコールバック関数を解析・修正する。図 3.2 の上部はコールバック関数の形式であり、下部が VHF 内部で行われる工程である。最初に、コールバック関数、リクエストパス、メソッドを 1 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[13](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の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つ。脆弱性ハンドリング関数は特定の属性と名前を持つノー

ドを脆弱なノードとし、このノードを再帰的に探索して検出して、その後脆弱性ハンドリング関数の処理によってノードの一部を修正する。具体的な修正方法は、脆弱なノードに VHF が持つ関数やメソッドを挿入する。脆弱性ハンドリング関数は図 3.3 に示すように、脆弱なノードの一部に攻撃を無害化する関数を挿入することで、実行時に変数を動的に検証し、攻撃を無害化する。図 3.3 では、vuln()関数を脆弱なノードの条件としており、vuln()関数の引数 input を無害化する関数 helper()を挿入することで、脆弱性の影響を低減している。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引数として受け取る。このリストは全てのコールバック関数が AST の状態で格納されている。脆弱性ハンドリング関数は全てのコールバック関数を受け取ることで、単一のコールバック関数内の脆弱性だけでなく、コールバック関数間の論理的な設計によって起きる脆弱性を対策することが可能である。脆弱性ハンドリング関数はコールバック関数を修正したのち、全てのコールバック関数が格納されたリストを返す。最後に、修正されたコールバック関数の AST を実行可能なオブジェクトに戻す。実行可能な状態になった修正されたコールバック関数は、クライアントの通信の際に呼び出されリクエストを処理することができる。

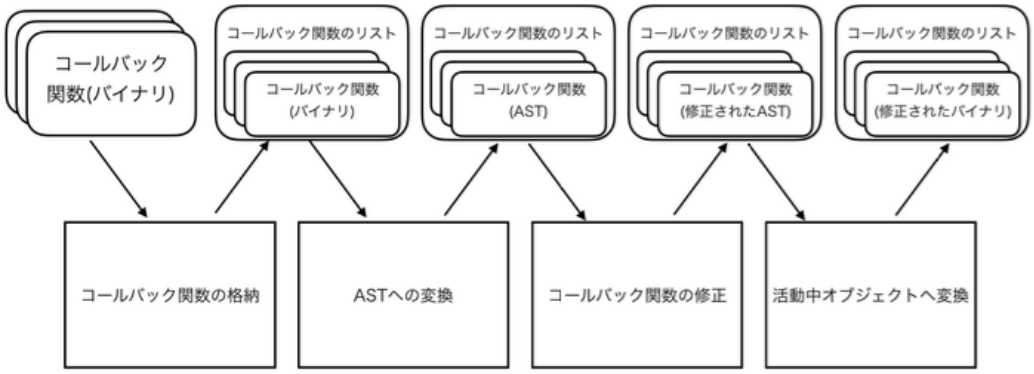

図 3.2: コールバック関数を自動修正する 4 つのステップ。

Figure 3.2: Four steps to modify call functions.

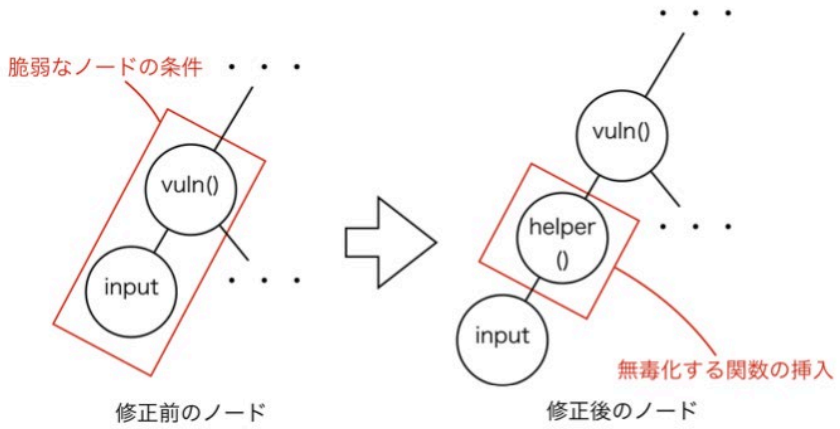

図 3.3: AST の修正による脆弱性影響低減手法の概要図。

Figure 3.3: Schematic diagram of vulnerability impact reduction method by modifying AST.

4　実装

本節で我々は、VHF の実装について記述する。VHF は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の 2 つのモジュールとして実装されている。実行開始時にコールバック関数を修

正するシステムにより、コールバック関数が修正され、実行中はクライアントのリクエストに基づきレスポンスを返す。

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。この時にコールバック関数の格納、コールバック関数の AST への変換、コールバック関数の修正、コールバック関数の活動中のオブジェクトへの変換が行われる。それぞれの実装について下に記述する。

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。

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いる。

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 1 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 は変数でリクエストの情報を格納している。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。

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とリクエストパス、リクエストメソッドの要素を持つ辞書形式にして、その辞書データをリストに格納する。

4.1.2 コールバック関数の AST への変換

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、つまり機械語の状態である。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 は活動中のオブジェクトを AST に直接変換できないため、活動中のオブジェクトを一度ソースコードに変換したのちに AST に変換する。活動中のオブジェクトからソースコードを取得するために、Python が提供している inspect モジュールの getsource メソッドによりソースコードを取得する。getsource メソッドは活動中のオブジェクトを引数に取り、そのソースコードを返すメソッドである。Python の活動中オブジェクトは関数名やソースコードのファイルのパスなどの情報を保持している。その情報を利用して getsource メソッドはソースコードのファイルを読み取り、ソースコードを取得する。動的に実行された活動中のオブジェクトには、ソースコードのファイルなどの情報がないため、getsource メソッドでソースコードを取得できない点

には注意が必要となる。getsource メソッドが取得したソースコードは route デコレータを含むため、そのまま AST に変換できない。そのため、コールバック関数の先頭にある route デコレータを取得したソースコードから取り除く。その後、取得したソースコードを 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。生成された AST は、リクエストメソッドとリクエストパス、AST を要素とする辞書形式にまとめられ、この辞書型式のデータはリストに格納される。

4.1.3 コールバック関数の修正

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の内部で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、検出されたノードを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する。

VHF は新しい脆弱性が発見された時、脆弱性ハンドリング関数を追加することで未知の脆弱性を対策する。この実装手法により、既存の脆弱性ハンドリング関数に修正を加えることなく、脆弱性ハンドリング関数を実装可能である。脆弱性ハンドリング関数は脆弱性ごとに分割されており、これらの分割された脆弱性ハンドリング関数はリストに格納される。脆弱性ハンドリング関数を格納するために NodeFixer クラスの add デコレータを利用する。ソースコード 4.4 は、脆弱性ハンドリング関数の例である。

コールバック関数のリストは図 4.1 に示すように順に add デコレータで格納された脆弱性ハンドリング関数によって解析・修正される。

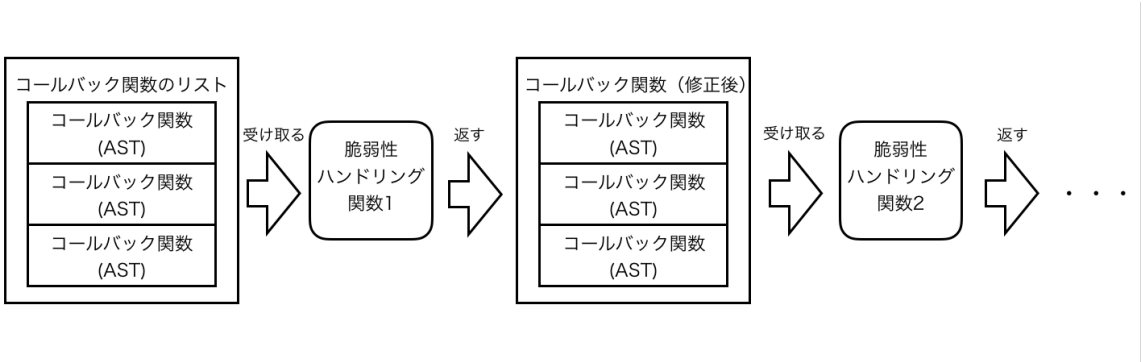

図 4.1:脆弱性ハンドリング関数がコールバック関数を修正する概略図。

Figure 4.1: Schematicdiagramofavulnerabilityhandlingfunctionmodifyingacallbackfunction.

4.1.4 コールバック関数の活動中のオブジェクトへの変換

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。

4.2 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 つの工程でリクエストを処理する。

4.2.1 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工程である。この工程では request インスタンスが作成される具体的には、リクエストパラメータを辞書形式に変換したり、リクエストボディをしたりして、HTTP リクエストを元に request インスタンスを作成する。request インスタンスのインスタンス変数がリクエスト情報になる。例えば request.path がリクエストパス、request.method がリクエストのメソッド、request.query がリクエストのパラメータである。このインスタンス変数はコールバック関数に引数として与えられる。

4.2.2 コールバック関数の呼び出し

リクエストパスとリクエストメソッドに基づき、コールバック関数を呼び出してレスポンスボディを作成する。VHF に格納されている修正されたコールバック関数のリストから、リクエストパスとリクエストメソッドが一致するコールバック関数を探す。一致するコールバック関数が存在する場合、requst を引数に与えてコールバック関数を実行する。コールバック関数の戻り値がレスポンスボディに相当する。一方、一致するコールバック関数がない場合、「NotFound」をレスポンスボディにする。

4.2.3 レスポンスの作成

レスポンスボディをエンコードし、レスポンスヘッダを作成したのち、レスポンスヘッダとレスポンスボディを組み合わせ、レスポンスを作成する。一致するコールバック関数が存在しレスポンスボディが作成された時は、レスポンスヘッダのステータスコードを 200、存在しない時は 404 とする。レスポンスヘッダとレスポンスボディを組み合わせクライアントに送信する。

5. 評価実験

VHF について、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実験、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実験を行った。本実験は MacOSX ElCapitan10.11.6,Intel Corei5(2.95GHz),メインメモリ 8GB の環境で実施した。

5.1 脆弱性の影響低減評価実験

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 2 つの脆弱性を持つアプリケーションを 1 つ実装した。

実装されたアプリケーションが持つ 2 つの脆弱性は SQLi と不適切な認証である。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。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と攻撃

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。本実験では、図 5.1 に示すように SQLi 脆弱性を持つアプリケーションが実装された。実装されたアプリケーションには、SQLi があるコールバック関数が 1 つ実装された。このコールバック関数は sqlite のデータベースと接続しており、通常のリクエストでは user テーブルからデータを取得するよう実装された。下記のソースコード 5.1 は SQLi 脆弱性を持つアプリケーションのコールバック関数の一部である。

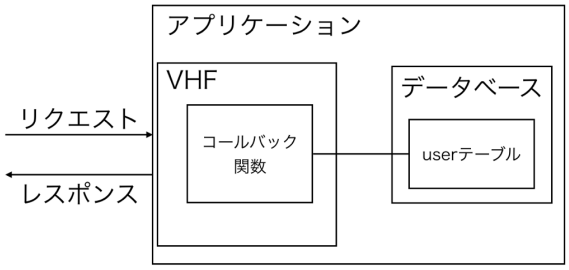

図 5.1:SQLi 脆弱性を持つアプリケーション。

Figure 5.1:SchematicofawebapplicationwhichhasaSQLi.

```
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)
```

リスト 5.1: SQLi 脆弱性を持つコールバック関数。

Listing 5.1: A callback function with SQLi vulnerability.

上記のソースコード（リスト 5.1）は、リクエストパラメータに基づいてデータベースを操作し、その結果をクライアントに返すコールバック関数である。このソースコードの 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 行目でデータベースと接続する準備をしている。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（リスト 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 はユーザ名、password はユーザ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（リスト 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（リスト 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、クエリを実行するコードを修正する脆弱性ハンドリング関数を実装した。この脆弱性ハンドリング関数は cur インスタンスの execute メソッドが持つ引数をエスケープする escape_special_query()関数を挿入した。escape_special_query()関数はクエリを引数に取りクエリに drop 命令が入っている場合クエリを捨てる関数である。上記ソースコードでは、リスト 5.1 の 12 行目と 16 行目の cur.execute()メソッドの引数を escape_special_query()関数でエスケープした。ソースコード（リスト 5.2）は実装された脆弱性ハンドリング関数の一部である。

```
1  class InsertQueryChecker(ast.NodeTransformer):
2    def visit_Call(self, node):
3      if isinstance(node.func, ast.Attribute):
4        if isinstance(node.func.value, ast.Name):
5          if node.func.value.id is 'cur':
6            if node.func.attr is 'execute':
7              new_args = []
8                for arg in node.args:
9                  new_arg = ast.Call(
10                   func=(ast.Name(id='
                         escape_special_query', ctx=ast.
                         Load())),
11                   args=[arg],
12                   keywords=[]
13                 )
14                 new_args.append(new_arg)
15                 new_node = ast.Call(
16                   func=node.func,
17                   args=new_args,
18                   keywords=node.keywords
19                 )
20               return ast.copy_location(new_node,
                   node)
21     return node
```

リスト 5.2:SQLi の影響を低減するための脆弱性ハンドリング関数の一部。

Listing 5.2: A vulnerability handling function for reducing affection of SQLi.

ソースコード（リスト 5.2）の 1 行目は、クラスの宣言である。ast.NodeTransformer クラスは AST を再帰的に探索する ast モジュールが持つクラスである。このクラスの visit 属性というメソッドは特定のノードを検出した時に実行されるメソッドであり、ソースコード(リスト 5.2)の 2 行目の visitCall()メソッドはノードの属性が ast.Call の時に実行されるメソッドである。ソースコード(リスト 5.2)3 行目から 6 行目が cur.execute()メソッドを検出するノードの条件である。その後条件に一致する AST を検出し、ソースコード(リスト 5.2)の 9 行目から 19 行目で AST を修正する。

脆弱性ハンドリング関数によって脆弱性の影響を低減できたか確認するために、まず脆弱性ハンドリング関数の実行部分をコメントアウトしたアプリケーションでローカル上の 8000 番ポート実行し、このアプリケーションに対して攻撃を行った。次に、脆弱性ハンドリング関数に解析・修正されたアプリケーションをローカル上の 8000 番ポートに立ち上げ、攻撃であるリクエストをアプリケーションに送信した。攻撃リクエストはパスが/login で、クエリの要素となるリクエストボディ部分が

```
action=drop user;--&id=name&password=password
```

とした。

5.1.2 不適切な認証を持つコールバック関数の修正と攻撃

不適切な認証を持つコールバック関数を VHF 上に実装した。図 5.2 は不適切な認証を持つアプリケーションの概要図である。このコールバックク関数は図に示すように/login へのリクエストでは認証を行い、アプリケーション内に登録されたユーザは ADMINPAGE が返却される。一方、/home へのリクエストは、認証なしに ADMINPAGE を返却する。このコールバック関数は適切に認証を行っていないため、全てのユーザが ADMINPAGE を閲覧可能である。ADMINPAGE が権限を必要とするページである時、このアプリケーションは不適切な認証の脆弱性を持つ。ソースコード（リスト 5.3）が実装したコールバック関数である。

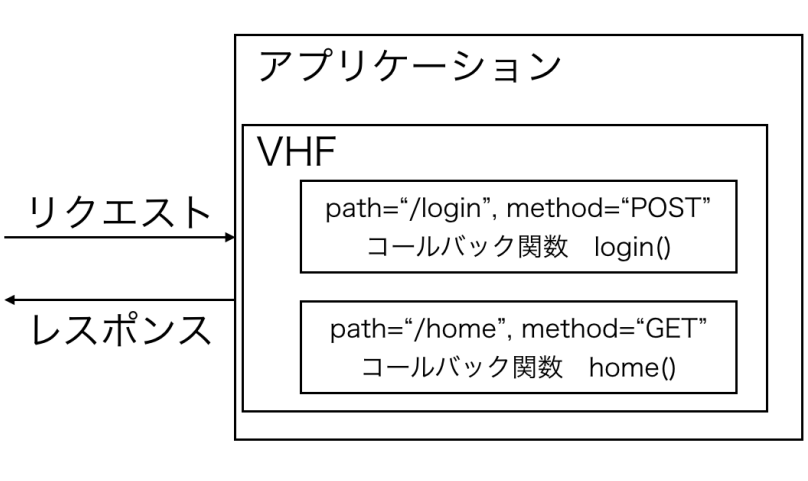

図 5.2:不適切な認証を持つアプリケーションの概要。

Figure 5.2:Schematic of an application with an unappropriated authentication.

```
1  class InsertQueryChecker(ast.NodeTransformer):
2    def visit_Call(self, node):
3      if isinstance(node.func, ast.Attribute):
4        if isinstance(node.func.value, ast.Name):
5          if node.func.value.id is 'cur':
6            if node.func.attr is 'execute':
7              new_args = []
8                for arg in node.args:
9                  new_arg = ast.Call(
10                   func=(ast.Name(id='
                         escape_special_query', ctx=ast.
                         Load())),
11                   args=[arg],
12                   keywords=[]
13                 )
14               new_args.append(new_arg)
15               new_node = ast.Call(
16                 func=node.func,
17                 args=new_args,
18                 keywords=node.keywords
19               )
20             return ast.copy_location(new_node,
                   node)
21     return node
```

リスト 5.3: 不適切な認証を持つ脆弱性のある関数。

Listing 5.3:A vulnerable function which has an unappropriated authentication.

このアプリケーションには2つのコールバック関数が実装されている。1つ目がdologin()関数であり、2つ目がhome()関数である。dologin()関数はリクエストパスが/loginでリクエストメソッドがPOSTの時に実行される。このコールバック関数は、ユーザのIDとパスワードによるユーザ認証を行う関数である。dologin()関数はソースコード5.3では、2行目から8行目に該当する。3行目と4行目でユーザのIDとパスワードをリクエストから取得する。その後、5行目で認証を行っている。5行目のisadmin()関数はユーザのIDとパスワードを引数に取る。ユーザIDとパスワードが一致するユーザが存在する時True

を返し、そうでない時は False を返す。ソースコード 5. 3 の isadmin()関数が True の時、dologin()関数は ADMINPAGE を返し、False の時は”LOGINPAGE”を返す。10 行目と 11 行目のコールバック関数は home()関数である。このコールバック関数はリクエストを受け取ると、ADMINPAGE を返す。

home()関数に対して認証機能を追加する脆弱性ハンドリング関数を実装した。この脆弱性ハンドリング関数は isadmin()関数下のノードではなく、かつ isadmin()関数下の戻り値ノードと同様のページを返すノードを脆弱なノードとした。この脆弱なノードを修正するために、脆弱性ハンドリング関数は 3 つの操作を行う。まず isadmin()関数が True である時の、属性が戻り値であるノードを検出しリストの形式にまとめる。まとめられたこれらのノードのリストをリスト A とする。次に isadmin()関数が True である戻り値と同様のページを返すノードを検出しリストにまとめる。このリストをリスト B とする。第 3 にリスト B に含まれ、かつリスト A に含まれないノードを検出する。検出されたノードは isadmin()関数下のノードではなく、isadmin()関数下の戻り値ノードと同様のページを返すため脆弱である。最後に脆弱性ハンドリング関数は、検出された脆弱なノードに isadmin()関数を追加する。

この脆弱性ハンドリング関数を評価するために、脆弱性ハンドリング関数をコメントアウトして実行したアプリケーションと脆弱性ハンドリング関数でコールバック関数を解析・修正したアプリケーションに対してそれぞれ同じ攻撃を行った。本実験の攻撃リクエストは/home へのリクエストである。

5.2 オーバーヘッドの評価実験

実行開始時にコールバック関数を解析・修正する時間を time()関数を利用して計測した。コールバック関数を解析・修正するソースコードの前後に time()関数を記述し、その差をとることでコールバック関数を修正するために必要になるオーバーヘッドを計測した。本実験で実装されている脆弱性ハンドリング関数は SQLi を対策する脆弱性ハンドリング関数が 1 つと不適切な認証を対策する脆弱性ハンドリング関数が 1 つの合計 2 つであった。コールバック関数は実験 5.1.2 の不適切な認証を持つアプリケーションを利用した。このアプリケーションは 2 つのコールバック関数を持っている。

6 評価結果

6.1 アプリケーションへの攻撃

脆弱性を持つ 2 つのアプリケーションに対して脆弱性ハンドリング関数を適用した場合と適用してない場合でそれぞれ攻撃を行った結果、脆弱性ハンドリング関数はそれぞれの脆弱性の影響を低減したことがわかった。

6.1.1. SQLi 脆弱性を持つアプリケーションへの攻撃結果

脆弱性ハンドリング関数をコメントアウトして実行しなかった時、攻撃リクエストがデータベースのテーブルを消去したことがわかった。攻撃リクエストはパスが/login で、クエリの要素となるリクエストボディ部分が

```
action=drop user;--&id=name&password=password
```

であった。コールバック関数内部を確認したところ、この攻撃リクエストは

```
drop user;--user where id=”name” and password=”password”
```

というクエリを実行していた。このクエリは dropuser 以降のクエリがコメントアウトされるため、user テーブルを消去するクエリが実行された。一方で脆弱性ハンドリング関数が実行されていると、テーブルが消去されていないことがわかった。攻撃リクエストから

```
drop user;--user where id=”name” and password=”password”
```

というクエリがコールバック関数内部で作成されたが、クエリを実行する前に無害化された。その結果、テーブルの消去は起こらなかった。

6.1.2 不適切な認証を持つアプリケーションへの攻撃結果

脆弱性ハンドリング関数を実行していない場合、不適切な認証を持つアプリケーションに対して、/home へリクエストを送信したところ home()関数がコールバック関数として呼び出された。その後、クライアントは”AD-MINPAGE”のレスポンスを受け取った。つまり、不適切な認証を利用した攻撃が可能だった。一方、脆弱性ハンドリング関数を実装した場合、home()関数がコールバック関数として呼び出されたが、挿入された isadmin()関数により、”LOGINPAGE”をクライアントに送信した。結果、不適切な認証への攻撃の影響を低減できることがわかった。この攻撃は、一般的な WAF では対策することが困難な攻撃である。一般的な WAF はアプリケーション内の情報を知らないため、アプリケーション内で認証が行われるかを知ることができない。もし一般的な WAF が今回の攻撃を対策する場合、WAF は認証が必要であるリクエストパスのパターンを全て知っている必要がある。しかし、認証を行っていないパスを全て対策することは困難である。対して本実験では、ある認証が必要なページに対して isadmin()関数を利用すれば、そのページへの遷移は isadmin()関数が適用されることがわかった。

6.2 オーバーヘッド

脆弱性ハンドリング関数が実行される前後に、time()関数を追加し、その差を調べたところ約 13 ミリ秒であった。この結果から脆弱性ハンドリング関数が実行開始時に与えるオーバーヘッドは大きくないとわかった。

7 おわりに

本研究では Web アプリケーションフレームワーク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て変更する機能を提案した(3 節)。提案したフレームワークを評価するために、脆弱性ハンドリング関数を持つ Web アプリケーションフレームワーク VHF を実装し、ローカル上で実行した脆弱な Web アプリケーションを攻撃する実験を行った。

この実験から、VHF は SQLi と不適切な認証の脆弱性をアプリケーション開発者が修正することなく自動的に修正し、これらの脆弱性の影響を低減することが可能であるとわかった(5 節)。また、脆弱性ハンドリング関数を実装した時の実行開始時のオーバーヘッドは小さく、実行に大きな影響を与えないことがわかった(6 節)。

一方で VHF は Web アプリケーション開発者がアプリケーションを実装する前に脆弱性ハンドリング関数を実装する必要がある。つまり Web アプリケーションが開発しうる様々な脆弱性に対して網羅的な脆弱性ハンドリング関数をあらかじめ実装する必要がある。これは今後の課題である。

本研究では、VHF の脆弱性ハンドリング関数により従来では対策が困難だった脆弱性の影響を低減できることがわかった。今後の改善により、VHF はより安全なアプリケーションの効率的な実装へ貢献すると考えられる。

参考文献

[1] Joel Weinberger, Prateek Saxena, Devdatta Akhawe, Matthew Finifter, Richard Shin, and Dawn Song. A systematic analysis of xss sanitization in web application frameworks. In European Sym- posium on Research in Computer Security, pages 150-171. Springer, 2011.

[2] Marcel Hellkamp. Bottle: Python web framework. https:// bottlepy.org/docs/dev/ (参照日時: 2021-02-04).

[3] Pallets. Welcome to flask - flask documentation (1.1.x). https://flask.palletsprojects.com/en/1.1.x/ (参照日時: 2021-02-04).

[4] Christopher Kruegel and Giovanni Vigna. Anomaly detection of web-based attacks. In Proceedings of the 10th ACM conference on Computer and communications security, pages 251-261, 2003.

[5] Nico Epp, Ralf Funk, Cristian Cappo, and San Lorenzo-Paraguay. Anomaly-based web application firewall using http-specific features and one-class svm. In Workshop Regional de Seguran ̧ca da Informa ̧c ̃ao e de Sistemas Computacionais, 2017.

[6] Avik Chaudhuri and Jeffrey S Foster. Symbolic security analysis of ruby-on-rails web applications. In Proceedings of the 17th ACM conference on Computer and communications security, pages 585- 594, 2010.

[7] Meixing Le, Angelos Stavrou, and Brent ByungHoon Kang. Dou- bleguard: Detecting intrusions in multitier web applications. IEEE Transactions on dependable and secure computing, 9(4):512-525, 2011.

[8] Tammo Krueger, Christian Gehl, Konrad Rieck, and Pavel Laskov. Tokdoc: A self-healing web application firewall. In Proceedings of the 2010 ACM Symposium on Applied Computing, pages 1846-1853, 2010.

[9] 独立行政法人情報処理推進機構セキュリティセンター. Web application firewall(waf) 読本 改訂第 2 版第 3 刷. https://www.ipa. go.jp/files/000017312.pdf (参照日時: 2021-02-04).

[10] Michiaki Ito and Hitoshi Iyatomi. Web application firewall using character-level convolutional neural network. In 2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA), pages 103-106. IEEE, 2018.

[11] William GJ Halfond and Alessandro Orso. Amnesia: analysis and monitoring for neutralizing sql-injection attacks. In Proceedings of 38 the 20th IEEE/ACM international Conference on Automated software engineering, pages 174-183, 2005.

[12] 独立行政法人情報処理推進機構. 不適切な認証 (cwe-287) - jvn ipedia. https://jvndb.jvn.jp/ja/cwe/CWE-287.html (参照日時: 2021-02-04).

[13] Python Software Foundation. ast ―抽象構文木― Python 3.9.1 ドキュメント.

[14] 久保田康平: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, 九州大学大学院システム情報工学科学研究院情報知能工学専攻修士論文, 2021 年 2 月.

[15] Kohei Kubota, Wai Kyi Kyi Oo, Hiroshi Koide: A New Framework Feature to Secure Web Applications, Proc. 2020 8th International Symposium on Computing and Networking Workshop, CANDARW 2020, pp.334-340,2020.

5.3 研究内容紹介

5.3.1 小出 洋

研究内容

1. Moving Target Defense（MTD）に関する研究

Moving Target Defense（MTD）は情報システムにおけるさまざまなパラメータ（例えば OS、システムコール番号、実行可能バイナリのマジックナンバー、ネットワーク識別子）を変化させ、攻撃を困難にする近年注目されている手法である。本研究では、特定のアプリケーションや特定の情報システムに向いた MTD の開発と評価、ある MTD を情報システムに適用したときの平均攻撃成功時間間隔などの MTD のサイバー攻撃に対する防御性を評価する方法に関する考察、ひとつの情報システムに複数の異なる MTD を適用した場合の防御性を評価する方法について研究を行っている。

2. 脅威トレースに関する研究

APT 攻撃に利用されるマルウェアに代表される脅威が情報システムに侵入したときの活動を予測し、脅威が行う攻撃を阻止したり、情報漏洩を防いだりするには何が必要かを明らかにし、情報システムの設計や運用に資することを目的として脅威トレースの提案、実装、評価を行っている。脅威トレースはマルウェアとそれが動作する情報システムと攻撃に使われるマルウェアを抽象度の高いモデルで表現し、その挙動をシミュレーションすることで解析している。

3. Web アプリケーションのための攻撃検知システムに関する研究

Web アプリケーションを作成する際には、Web アプリケーション・フレームワークが必ず利用される。サイバー攻撃の検知やサイバー攻撃からシステムを防御するための機能は Web アプリケーション・フレームワークが備えるべき機能といえるが、実際は Web アプリケーション・ファイヤーウォールやセキュリティアプライアンス等の別のシステムになっていることが多い。サイバー攻撃からの防御のための機能を Web アプリケーション・フレームに組み込んだ場合、Web アプリケーション内部の情報や Web アプリケーションの特徴にあわせた攻撃検知とすることができる。Web アプリケーションの特徴に合わせた攻撃検知や攻撃をハニーポットに誘導する機能を Web アプリケーション・フレームワークに実装して評価する研究を行っている。

所属学会名

ACM，ソフトウェア科学会，電子情報通信学会，情報処理学会

主な研究テーマ

1. 実践的プログラミング教育の支援環境の開発
   キーワード：実践的プログラミング教育，教育学習過程の情報化，プログラミング教育支援，学習過程，活動履歴，2017.04～2020.04.
2. ネットワークアプリケーションにおける攻撃検出に関する研究
   キーワード：サイバーセキュリティ，ウェブアプリケーション，ウェブアプリケーションフィアウォール，ハニーポット，攻撃検知，2017.04～2020.04.

研究プロジェクト

戦略的国際共同研究プログラム(SICORP)「国際共同研究拠点（インド）」「安全な IoT サイバー空間の実現」
2016.10～2021.09，代表者：岡村 耕二，九州大学，国立研究開発法人　科学技術推進機構（日本）

研究業績

● 原著論文

1. Zhao Hao, Yaokai Feng, Hiroshi Koide, Kouichi Sakurai, A Sequential Detection Method for Intrusion Detection System Based on Artificial Neural Network, International Journal of Networking and Computing, 10, 2, 213-226, 2020.07.
2. Kohei Kubota, Wai Kyi Kyi Oo, Hiroshi Koide, A New Feature to Secure Web Applications, 8th International Symposium on Computing and Networking Workshops, CANDARW 2020, 334-340, 2020.11.

● 総説，論評，解説，書評，報告書等

1. 小出　洋 , Moving Target Defense とサイバーセキュリティ教育, (一社)九州テレコム振興センター会員向け Web マガジン Key-Eye, 2020.04.

● 学会発表

1. Mbow Mariama, Hiroshi Koide, Kouichi Sakurai, Adversarial Attack Against Network Intrusion Detection Systems with Deep Learning, 情報処理学会九州支部火の国シンポジウム, 2021.03.
2. Yihui, Y., Koide, H., Sakurai, K, Anomaly Detection of C&C Traffic using Chebyshev Theorem and Machine learning Based on URL Anomaly features, 電子情報通信学会 総合大会, 2021.03.

研究資金

● 競争的資金

1. 2019 年度～2020 年度，教育訓練プログラム開発事業（2 年開発コース），代表，教育訓練プログラム開発事業（2 年開発コース）<<区分　ICTを開発した分野>>委託
2. 2017 年度～2022 年度，文部科学省研究拠点形成費等補助金（成長分野を支える情報人材の育成拠点の形成(enPiT) enPiT-Pro)，連携，企業・官公庁等の IT 実務、設計･製造実務における情報セキュリティに関わるプロ人材育成コースの開発・実施

● 共同研究、受託研究

1. 2020.06～2021.03，代表，情報システムを攻撃から防御するための Moving Target Defense に関する研究

教育活動

● 教育活動概要

1. 2017 年～現在　enPiT-Pro ProSec-IT の主体的実施

● 担当授業科目

1. 2020 年度・後期，通信工学通論 A
2. 2020 年度・後期，通信工学通論 B
3. 2020 年度・後期，コンピュータシステム通論 A
4. 2020 年度・後期，コンピュータシステム通論 B

社会貢献・国際連携

● 社会貢献活動

1. 2019.04.01～2022.3.31，福岡県警サイバー犯罪対策テクニカルアドバイザーとして委嘱された（再委嘱）。

大学運営

● 学内運営に関わる各種委員・役職等

1.　2019.04～2021.03, 情報統括本部情報共有基盤室長