# 内部処理分析を基にした Web アプリケーションのセキュリティ SaaS の開発
## ―内部処理分析による新たな攻撃検知―

### 1.　背景

近年、インターネットを介した多岐にわたるサービスが広く提供されるようになり、その中でもサービス提供にあたって媒体として Web アプリケーションが広く用いられている。しかし、実装上の不備や実装に際し用いるライブラリの脆弱性により、Web アプリケーションにおいては様々なセキュリティリスクが存在している。稼働している Web アプリケーションに対しセキュリティ対策を提供するツールとして Web Application Firewall （WAF） が広く利用され、システムコールによる攻撃検知ツールも提供されているが、それぞれのセキュリティ対策ツールには検知及び対策ができない攻撃事例が存在しており、セキュリティリスクが未だ存在しているという現状がある。

### 2.　目的

このような現状において、既存のセキュリティ対策ツールでは検知及び対策ができないセキュリティ侵害を検知可能な対策ツールが必要となっている。本プロジェクトでは、稼働している Web アプリケーションに内部で実行された処理を記録するためのプラグインであるプロファイラを導入し、そこから得た内部処理情報から攻撃に特有の処理を発見することで、既存のセキュリティ対策ツールでは検知及び対策ができないセキュリティ侵害を検知するシステムを開発することを目的とした。本システムの導入により、既存のセキュリティ対策ツールでは検知ができなかったセキュリティ侵害を検知し、セキュリティリスクの低減を目指した。

### 3.　開発の内容

本プロジェクトでは内部処理分析という新たな仕組みを用いたセキュリティ SaaS として、Web アプリケーションの内部処理を収集するプロファイラ、ルールに基づく攻撃検知機構、ダッシュボードを開発した。本システムを Phrude（Profiling History-based Runtime Detection System）と命名し、システムの構成図を図 1 に示す。

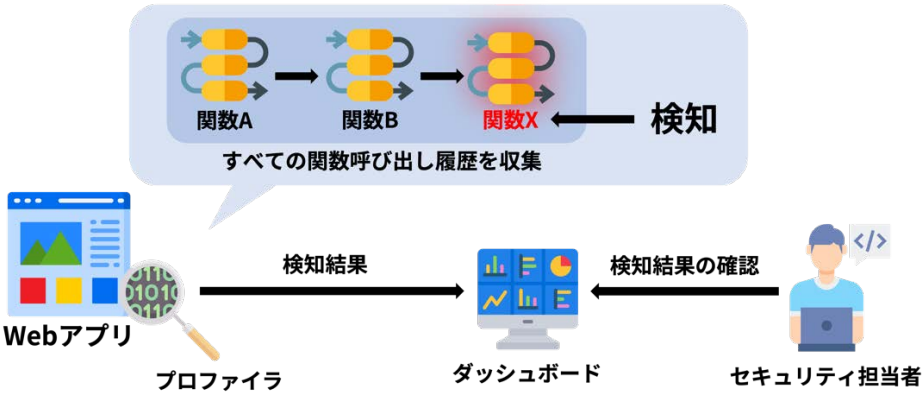


**図 1　Phrude のシステム構成図**

Web アプリケーションのセキュリティ担当者がまず Web アプリケーションにプロファイラを導入し内部処理を収集する。ここで収集される内部処理は図 2 で示されているように実行された関数名・関数が含まれるプログラムファイル・関数呼び出しの引数情報・呼び出し元関数をプロファイラからなる関数呼び出し履歴であり、Web アプリケーションに対して処理を挿入する攻撃があった際、攻撃による処理が記録される。

```
...
関数名 :  call
ファイル : /usr/local/lib/python3.10/site-packages/jinja2/runtime.py
引数 :  {
    "args": "('CLOUD_SECRET_KEY',)",
    "kwargs": "{}",
    "_Context__obj": "CLASS:method-wrapper",
    "_Context__self": "CLASS:Context“
}

関数名 :  getenv
ファイル :  /usr/local/lib/python3.10/os.py
引数 :  {
    "key": "CLOUD_SECRET_KEY",
    "default": null
}
...
```

**図 2 収集する内部処理の例**

そしてプロファイラとともに実装されている攻撃検知機構が動作し攻撃を検知する。プロファイラで収集した内部処理である関数呼び出し履歴に対し、事前に設定した攻撃の処理としてよく用いられる関数が指定されたルールに一致するような処理が実行されたか判定することで攻撃を検知する。

そして攻撃が検知された場合、検知した処理の関数名・関数が含まれるプログラムファイル・関数呼び出しの引数情報・呼び出し元関数をダッシュボードに送信し、Web アプリケーションのセキュリティ担当者が確認する。図 3 で示しているように、記述ルールではパラメータ name はルール名、パラメータ func は検知する関数名、パラメータ file は検知する関数が含まれているプログラムファイル名となっている。次のパラメータ level は検知の重要度を示し、指定できる値は info, warn, alert の 3 つの段階となっている。最後のパラメータ value では、特定の値を指定することで検知結果の中でそれを取り上げて表示することができる。図 3 左の例では取り上げる値の名称 key に対して、関数の引数を示す$func_args の中で getenv 関数において取得する環境変数名が格納されている key パラメータを指定している。図 3 右の検知結果では指定されたルール名が表示されるとともに、パラメータ value で指定した、getenv 関数において取得する環境変数名が格納されている key パラメータの値である CLOUD_SECRET_KEY が取り上げて表示されている。


記述ルール　　検知結果

**図 3 記述ルールと検知結果の対応例**

また、図 4 で示されているように検知結果において関数呼び出しの引数情報および呼び出し元関数が確認できるため、検知された攻撃がどのような処理を行ったのかを把握することができ、攻撃原因の追求を行うことができる。

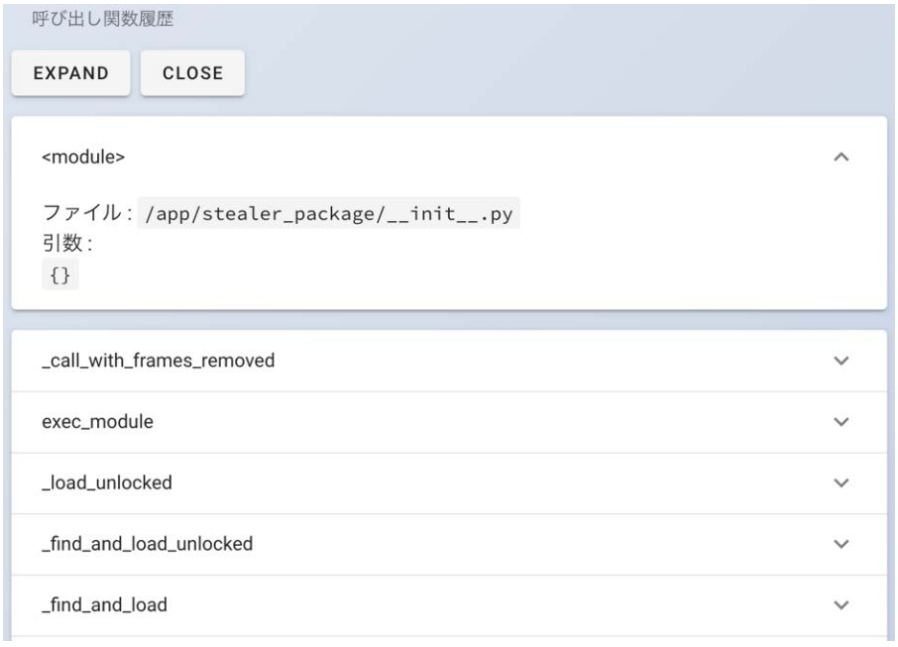


**図 4 OSS のサプライチェーン攻撃時の呼び出し関数履歴表示**

## 4. 従来の技術（または機能）との相違

従来の攻撃検知手法として、WAF やシステムコールによる攻撃検知が挙げられる。これらの攻撃検知手法と Phrude に対して、検知が得意な攻撃手法を表 1 に示す。

表 1 既存攻撃検知手法及び Phrude が検知を得意とする攻撃手法

| 攻撃手法 | WAF | システムコールによる攻撃検知 | Phrudeによる攻撃検知 | Phrudeによる原因追跡 |
|---|---|---|---|---|
| Cross Site Scripting | ○ | × | × | × |
| SQL Injection | ○ | × | × | × |
| OSSのサプライチェーン攻撃 | × | △ | ○ | ○ |
| 任意の処理を挿入するWAFの検知を回避した/未知の攻撃 | × | △ | ○ | ○ |

WAF は、特定の文字列や文字パターンを見つけることで検知をしているため、典型的な攻撃に対しては効果を発揮するが、検知されうる特定の文字列や文字パターンを無くす細工をした HTTP リクエストによる攻撃については検知ができない。また、システムコールによる攻撃検知は新規プロセスの作成などのシステムコールを発行する処理からなる攻撃を検知できるものの、環境変数の取得などのシステムコールを発行しない攻撃に対しては検知ができない。

Phrude では、これらの攻撃検知技術では防げない攻撃を内部処理の分析によって検知ができ、相補的な関係にある。

## 5. 期待される効果

Phrude の導入により、既存のセキュリティ対策ツールでは検知不可能な Web アプリケーションに対する攻撃を新たに検知できるようになる。Phrude の普及によってセキュアな Web アプリケーションが広まり、安全なサービスが運営され、さらにインターネットを介した多岐にわたるサービスの利用が促進されることを期待する。

## 6. 普及（または活用）の見通し

Phrude は内部処理分析による Web アプリケーションへの攻撃検知という新たな仕組みを提唱するもので、今後は新たに発見される攻撃手法に対しても継続して検知可能かを検証することで内部処理分析の有用性を証明し、Phrude 及び内部処理分析のさらなる普及を目指す。

## 7. クリエータ名（所属）


- 赤松 宏紀（大阪大学大学院情報科学研究科 博士前期課程）
- 大迫 勇太郎（大阪大学大学院情報科学研究科 博士前期課程）

（参考）関連 URL

- Phrude ティザーサイト
  https://phrude.com/
- Phrude ドキュメントページ
  https://phrude.com/docs/intro
- Phrude デモサイト
  https://demo.phrude.com/
- Phrude の検証で用いた Python Flask 製のデモ用 Web アプリ
  https://github.com/phrude/flask-demoapp