バイナリデータ圧縮に関する機能の作成に対して取り組んだ内容

このシステムでは、複数のバイナリデータをまとめて圧縮し取り扱えるように、圧縮解凍機能を設けております。設けた理由としては以下のようになります。

・ユーザーからのリクエストで多くのバイナリデータを受け取れるようにするため。
・ユーザーへのレスポンスの際のデータ量を抑えて、多くのバイナリデータを送信できるようにするため。

この資料では、主に圧縮ファイルの取り扱い時のセキュリティに気を使った内容をプレゼンします。

目次

1 採用したライブラリ
2 圧縮データからの抽出時の注意点
3 ファイルのデータ量が大きすぎる場合の対処
4 処理できないファイルの場合の対処
5 ZIP爆弾への対処

採用したライブラリ

選定したのは、Zip4jとなります。

理由としては以下のようになります。

- 圧縮ファイルの暗号化に対応できる圧縮ライブラリが必要だったため。

一応、Java標準の機能にもZIPファイルを取り扱える機能が存在しますが、そちらは暗号化された圧縮ファイルには対応できません。ユーザーから暗号化された圧縮ファイルが渡された際の対処も、処理に定義したいため外部ライブラリを用いる必要がありました。

- 調べた限り、圧縮ファイル用ライブラリは、この一択だったため。

先ほどの暗号化ファイルへの要件を満たすためには、標準のJavaのZIP機能は用いることができませんが、他のライブラリを探した際にはこのライブラリしか情報が存在せず、選択肢が限られたためです。

圧縮データからの抽出時の注意点

圧縮データからの抽出時に、特に気を付けなければいけない点は、以下のようになります。
これらの問題への対策について、以降の資料で順番に説明します。

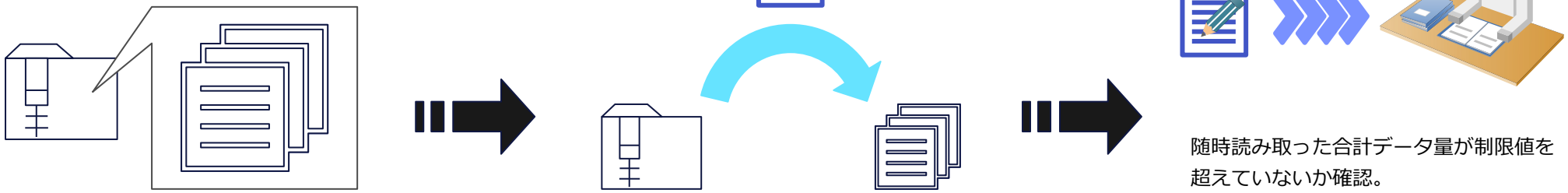
バッファオーバーフロー攻撃

ファイルのデータ量が大きすぎる場合の対処

データを抽出した際に、取り出したデータが大きすぎる場合があります。この対処に関しては、以下の方法で対応しています。

- 一つのファイル当たりのデータ量の制限を設け、超過したら処理を中断する。
- 受け付ける抽出ファイル数に制限を設け、規定の数量を超過したら処理を中断する。
- ファイルのバイト配列を読み取る際は、一気に読み取らず少しづつ読み取っていく。

一つのファイルのデータ量のカウント

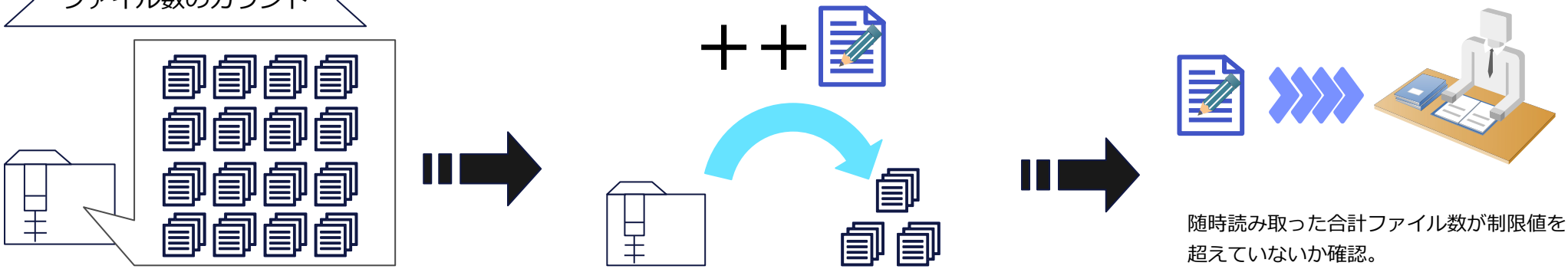


すごく大きいファイルが
入ったZIPファイル

少しずつデータを読み取り、
読み取ったデータ量を加算してメモしていく。

随時読み取った合計データ量が制限値を
超えていないか確認。
超過したら不正と判断し処理を中断。

ファイル数のカウント



大量のファイルが入ったZIPファイル

ファイルを一つ読み込もうとする際に、
現在の読み取りファイル数をカウントする。

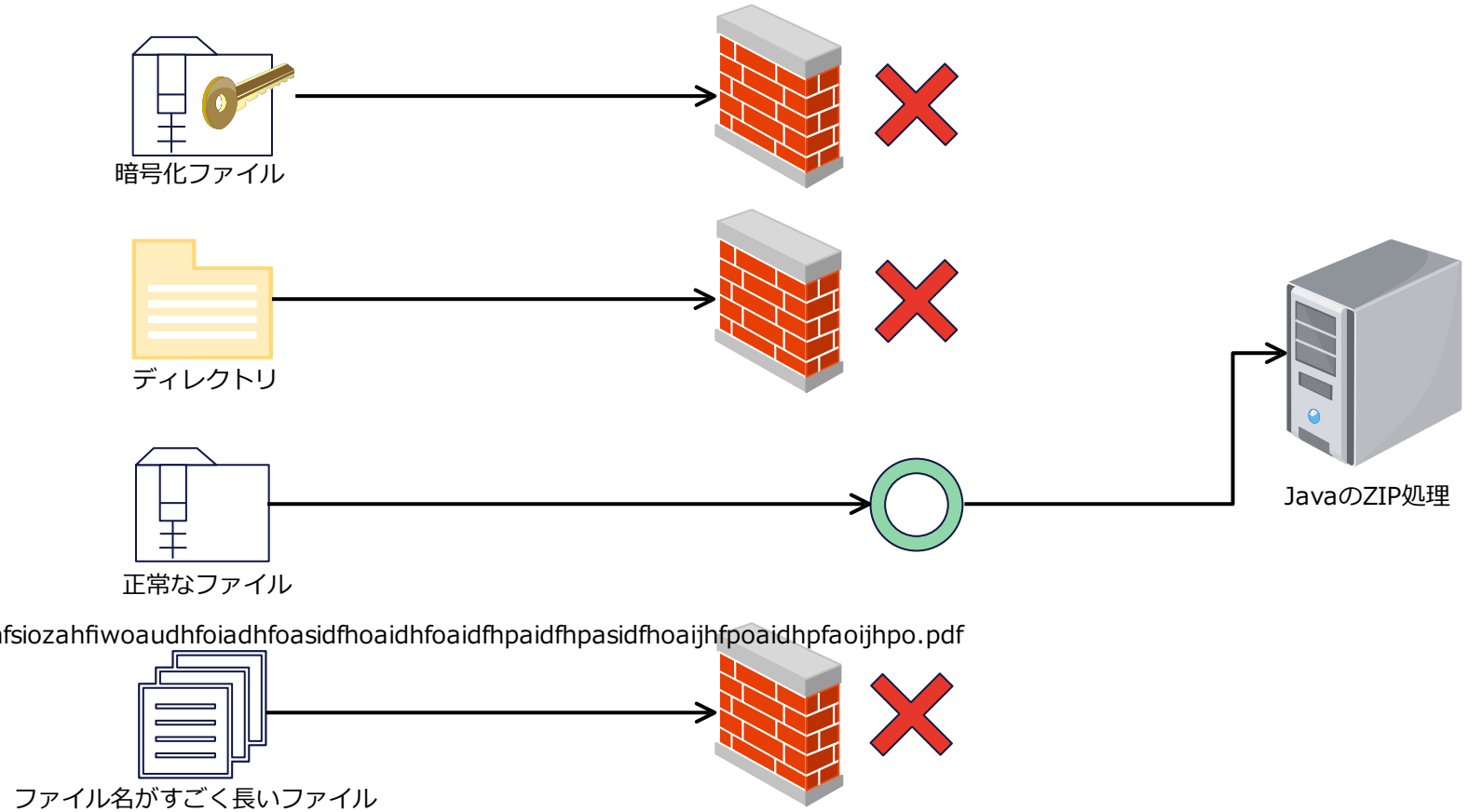
随時読み取った合計ファイル数が制限値を
超えていないか確認。
超過したら不正と判断し処理を中断。

処理できないファイルの場合の対処

このシステムは、要件上以下のようなファイルは処理できません。

- 暗号化されたファイル
- ファイルではなく、ディレクトリ
- ファイル名が長すぎるファイル

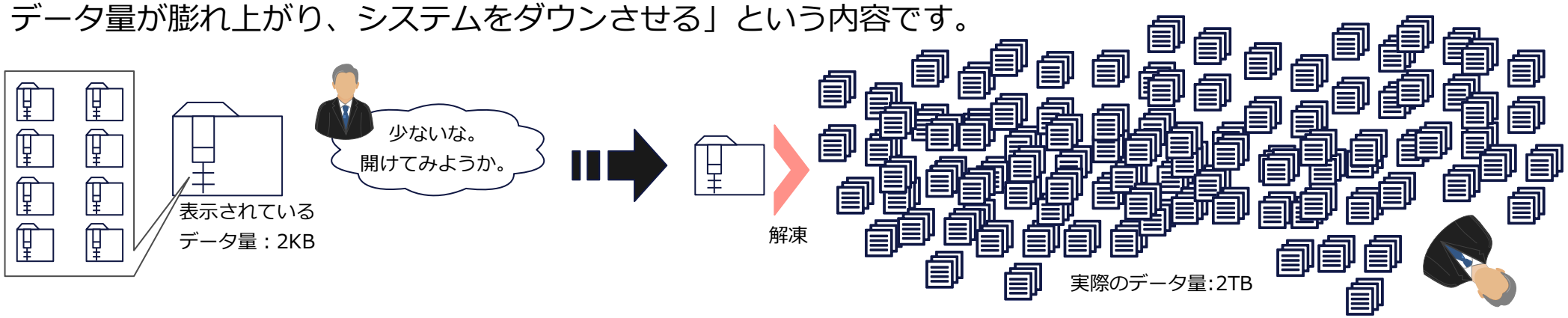
採用したライブラリの機能を用いてファイルの属性を判別し、これらに該当するファイルは、抽出処理にそもそも投入させません。



ZIP爆弾への対処

圧縮ファイルを用いた攻撃手法として代表的なのが、「ZIP爆弾」であり、それに対する対処が必要です。

この攻撃手法の簡単な説明をすると、圧縮ファイルの中にさらに圧縮ファイルを格納し、それを繰り返して格納した圧縮ファイルを生成して送信することで、「見かけ上のデータ量は少なくても、解凍したとたんデータ量が膨れ上がり、システムをダウンさせる」という内容です。



このシステムでの対処は、「再帰的な圧縮ファイルの解凍の回数に制限を設ける」事にしています。
そのため、圧縮ファイルを解凍した際に、中にさらに圧縮ファイルがあった際は解凍は行わないようにしており、圧縮ファイルが存在すればエラーとしています。

なお、このシステムの要件上、再帰的に解凍する必要性がない為、このようにしています。