# 「重要な処理」の際に混入する脆弱性

# 重要な処理とは？

取り消しできない処理

- 決済
- 退会処理
- パスワードの変更
- メッセージの送信

# CSRF: Cross Site Request Forgeries

- ユーザーが意図しない処理を実行させる攻撃方法
- ユーザーは攻撃されていると認識できない場合も
- CSRFはユーザーが実行できる処理しか実行できない。データを盗むなどはできない（パスワードリセットなどを除く）

# 攻撃手順（例）

1. 被害者は脆弱性のあるページにログインします
2. 被害者は悪意のあるユーザーから送られたリンクを踏みます（スパムメールなど）
3. 以上！

# なぜこんなに簡単に攻撃できるのか

あるサイトのパスワードリセットをする場合必要な条件は

- ログインしていること
- /reset にPOST Request を送ること
- POSTにnew_passwordを送ること

これらを満たすことができれば攻撃できる！

# なぜこんなに簡単に攻撃できるのか

- ブラウザーはクッキーに保存されたセッションIDを自動的に送信してしまう
- 例え、オリジンが違っていても！

# 問題点

- リンクを踏ませるだけと攻撃を大人数に仕掛けやすい
- 開発者にXSSほど知られていない？
- 個人情報などは直接漏れないが、パスワード変更などをすることによって情報漏洩することも
- レスポンスは読めるためそこから情報が漏洩することも
- ファイルアップロードなどを行うフォームも、XMLHttpRequestを使えばCSRF攻撃ができる（CORSでエラーを吐いてもPOSTは成功)
- 認証機能のない掲示板などのフォームなどにCSRF攻撃を行うこともできる

# 確認画面

- 重要な処理の確認画面の実装方法としては

1. Hiddenパラメータでパラメータを渡す
2. セッション変数を使う

どちらの方法でもCSRF対策にはならない

# セッション変数を使った確認画面

1. POSTリクエストでデータをサーバーへ送る(email, password etc)
2. サーバー側はセッション変数に保存
3. 確認画面返す
4. POSTリクエストで確認する
5. セッション変数をデータベースなどに保存

- 画面を二つ用意し時間差でPOSTするとCSRF攻撃ができる！

# 攻撃方法

罠観覧

↓

仕掛けのあるHTML、JavaScriptがサーバーにリクエストを送る

↓

攻撃用リクエストをサーバーが受信。
Session Cookieがついているためユーザー権限の処理を行うことができる

# 攻撃方法

罠観覧

- 防げない…

# 攻撃方法

仕掛けのあるHTML、JavaScriptがサーバーにリクエストを送る

- 防げない…
- 他人のサイトの機能を制限することはできない

（iframeなどを表示できないようにするなどは可能だが、POSTリクエストの実行権限をなくすことはできない）

# 攻撃方法

攻撃用リクエストをサーバーが受信

- ここで対策するしかない
- 任意の行動と（ユーザーがパスワード変更を意図的にすること）悪意のある行動を識別する必要がある

# 対策

- CSRF対策が必要なページは少ない
- 対策が必要なページは重要な処理を行うページのみ
- 要件定義で対策が必要な機能をマークする
- 基本設計工程でCSRF対策の必要なページをマークする
- 開発工程でCSRF対策を作り込む
- このように、CSRF対策は初めから考慮しないといけない

# 対策

CSRFトークン

- POSTリクエストを送るフォームにランダムに生成されたトークンを仕込む

# 対策

CSRFトークン

- 実装が楽、frameworkによってはオプションをつけるだけで
- UXは何も変わらない
- GETだとRefererからトークンが漏れる場合があるのでPOSTを使う
- しっかりと予想できない(しづらい)トークンを生成することで攻撃者はCSRF攻撃を素直に仕掛けることができない
- XSSなど、他の脆弱性があった場合は別

# 対策

CSRFトークン

- SessionIDを使うと万が漏れた場合セッションハイジャックなど他の攻撃に繋がるため、危険
- ハッシュしても、seed値を求められたらアウト(自身のSessionID, CSRFトークンを比較し、bruteforceでシード値を求められる）
- 時間などだけでもbruteforceされる場合も
- 暗号論的擬似乱数生成器などいろいろ難しそうなものを使うべき
- ほとんどの言語ではCSPRNGが実装されてる

# 対策

Refererの確認

- リクエストのReferer（前のページのURL)を確認する
- 正規のリクエストとCSRFでのリクエストではRefererが違う
- これも実装が簡単

しかし

- ブラウザの設定によってはRefererを送らない場合も
- それに気づかないユーザーのUXが下がる

# 対策

Refererの確認

注意点

Stringの比較などでRefererを確認する場合しっかりと/まで比較しないといけない

Eg. example.jpの確認だけだとexample.jp.example.comを許してしまう

# 対策

パスワードの再確認

- 処理を行うたびにパスワードを求める
- 実装が簡単
- 間違ってユーザーが処理を行わないようにできる（間違ってクリックしたなどの対策）

しかし

- UXが下がる

# アンチCSRF対策

- XSSでCSRFトークンを取ってくる
- ユーザーにCSRFトークンを教えてもらう
  https://www.symantec.com/connect/blogs/facebook-csrf
- トークンはInput Hiddenでも、Cookieでもデベロッパーツールを開けば見つかる
- CSRFは他の脆弱性があると簡単に対策を突破できてしまう。

# 対策の保険

- なので、保険をかけましょう
- 重要な処理が行われた場合（パスワード変更）メールを送るなど
- 履歴・ログが見れるようにする

# 疑問点？

CSRFトークンがhiddenパラメータで入っている場合攻撃者がそのままPOSTしたらCSRF攻撃されてしまうのでは？

- 攻撃用のサイトのオリジンは違う
- 同一オリジンポリシーによってオリジンの違うサイトを操作することはできない
- CSRFを取ってくるにはXSSなどで一度トークンを盗まないといけない

# 疑問点？

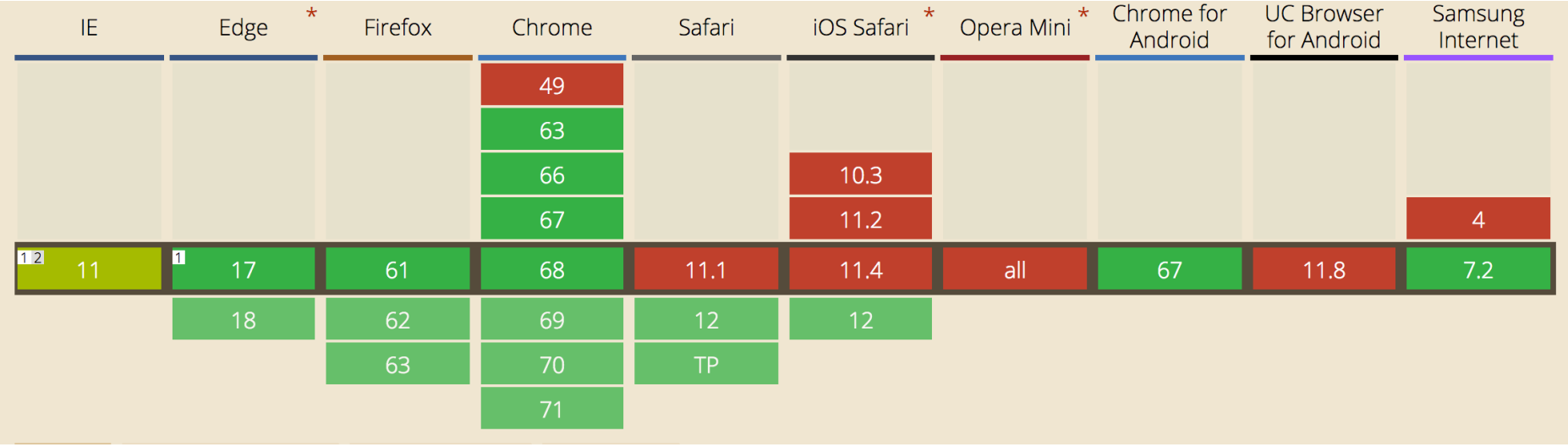
複数タブを立ち上げて同じ処理をしようとするとトークンはどうなる？

- CSRFトークンが毎回作られている場合（ワンタイム、最初に開いたタブからの処理はエラーを吐く（CSRFトークンがあっていない）
- CSRFトークンをユーザー毎に固定すると（sessionIDをHashするなどで）複数のタブからの同時に処理が行える

- トークンを盗まれてしまった場合が大変(時間制限などを設ける）

# おまけ1: Same Site Cookie

- SameSite以外にはCookieをつけない
- サポート率が上がっている



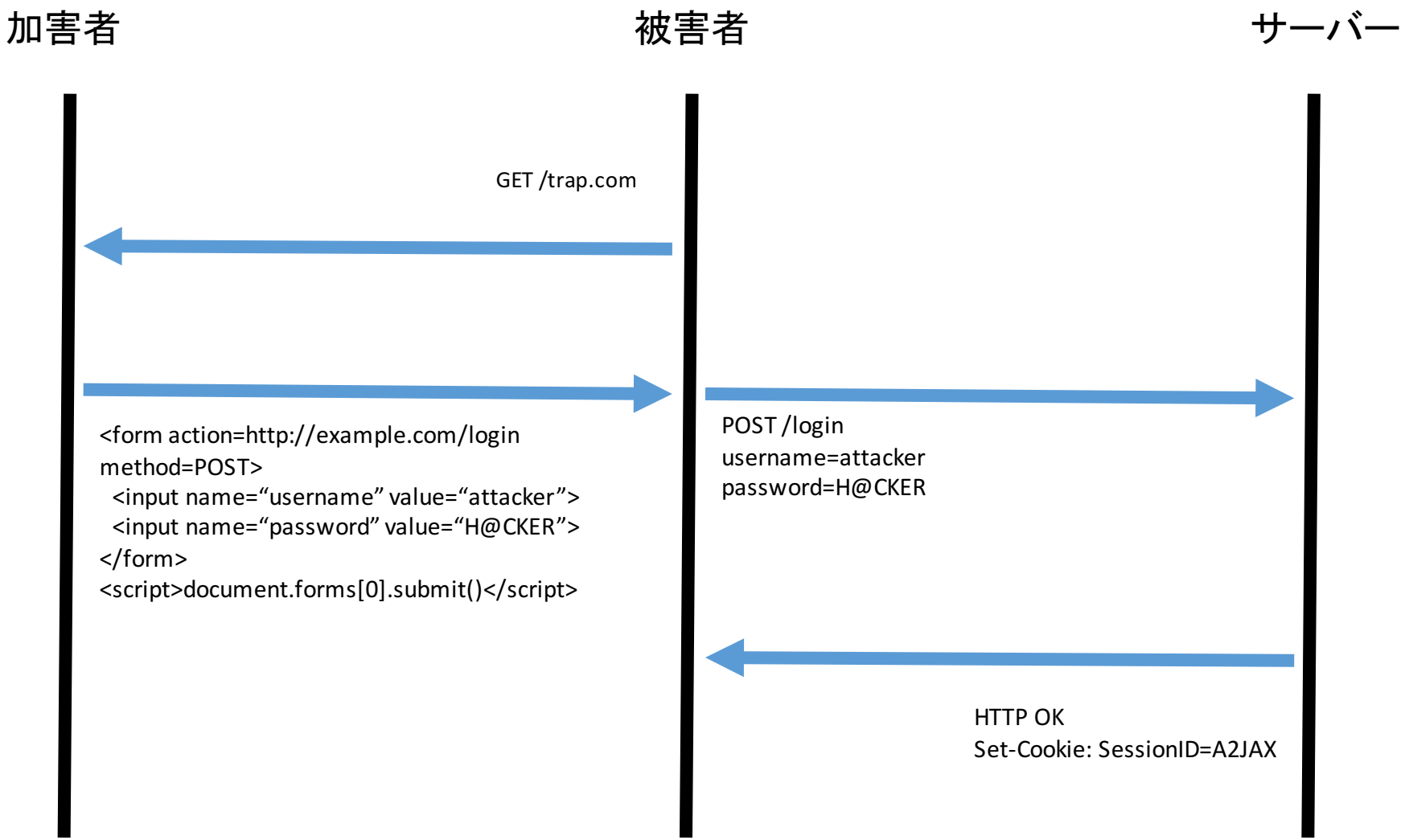
# おまけ1: Same Site Cookie

| request type, | example code, | cookies sent |
|---|---|---|
| link | `<a href="…">` | normal, lax |
| prerender | `<link rel="prerender" href="…">` | normal, lax |
| form get | `<form method="get" action="…">` | normal, lax |
| form post | `<form method="post" action="…">` | normal |
| iframe | `<iframe src="…">` | normal |
| ajax | `$.get('…')` | normal |
| image | `<img src="…">` | normal |

https://www.sjoerdlangkemper.nl/2016/04/14/preventing-csrf-with-samesite-cookie-attribute/

画面遷移が起きるはずの処理は許可されている(トップレベルの処理)

# おまけ2：ログインCSRF



# おまけ2：ログインCSRF

- 被害者を攻撃者のアカウントでログインさせることで被害者の行動を見ることができる
- YouTubeなどで視聴動画の情報を手にしたり、ユーザーログをかき集め被害者のデータを集めることができる
- ログイン画面にもCSRF対策をしましょう！

# クリックジャッキング

- iframeのボタンなどをJavaScriptでは押せない
  （同一オリジンポリシー）
- ユーザーにボタンを押してもらえばいい

# CSRFとの比較

- CSRFとクリックジャッキングは似ている
- 攻撃者は処理が行われた結果を得ることはできない
- 攻撃用サイトに誘導する必要性がある

# 対策

- アプリケーションのバグではないのでブラウザの協力が必要
- X-Frame-OptionをDENY,SAMEORIGINに設定する
- DENYはiframeでのページの表示を禁止
- SAMEORIGINはオリジンが同じ時のみiframe内で表示できる

# 対策の保険

- 重要な処理が行われた場合（パスワード変更）メールを送るなど
- 履歴・ログが見れるようにする
- CSRFの保険と同じなので、実装しましょう！

# おまけ4:レガシーブラウザ

- X-Frame-Optionが実装されていないブラウザもある
- self === topの確認
- Redirectはブロックされる(Chrome)

  Require user gesture for framebusting in cross-origin iframes

  https://www.chromestatus.com/features/5851021045661696
- window.confirmでの確認

おしまい

# おまけ5: CSRF攻撃の被害

- https://www.exploit-db.com/exploits/45078
- Router設定をリンクを踏ませるだけで変えられる（ログイン時）
- http://net.ipcalf.com/ IPアドレスをJavaScriptで取得