JPCERT CC®

HTML5 Security & Headers
- X-Crawling-Response-Header-

一般社団法人JPCERTコーディネーションセンター
早期警戒グループ 情報セキュリティアナリスト
重森 友行

JPCERT/CCとは

一般社団法人JPCERTコーディネーションセンター

（JPCERT/CC (ジェーピーサート・コーディネーションセンター)）

Japan Computer Emergency Response Team Coordination Center

- https://www.jpcert.or.jp/
- サービス対象: 日本国内のインターネット利用者やセキュリティ管理担当者、ソフトウエア製品開発者等（主に、情報セキュリティ担当者）
- コンピュータセキュリティインシデントへの対応、国内外にセンサをおいたインターネット定点観測、ソフトウエアや情報システム・制御システム機器等の脆弱性への対応などを通じ、セキュリティ向上を推進
- インシデント対応をはじめとする、国際連携が必要なオペレーションや情報連携に関する、我が国の窓口となる　CSIRT

※各国に同様の窓口となる　CSIRTが存在する

(例えば、米国のUS-CERT 、中国のCNCERT, 韓国のKrCERT/CC、など)

- 経済産業省からの委託事業として、コンピュータセキュリティ早期警戒体制構築運用事業を実施

- JPCERT/CCをご存知ですか? -

JPCERT/CCの活動

インシデント予防　インシデントの予測と捕捉　発生したインシデントへの対応

脆弱性情報ハンドリング

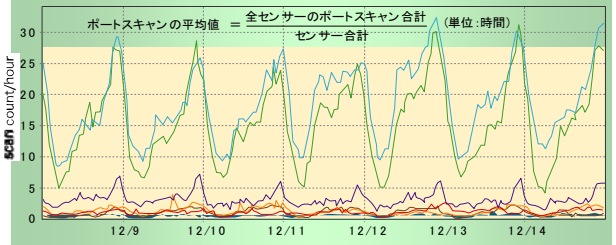
- 未公開の脆弱性関連情報を製品開発者へ提供し、対応依頼
- 関係機関と連携し、国際的に情報公開日を調整
- セキュアなコーディング手法の普及
- 制御システムに関する脆弱性関連情報の適切な流通

JVN Japan Vulnerability Notes

情報収集・分析・発信

定点観測（TSUBAME）

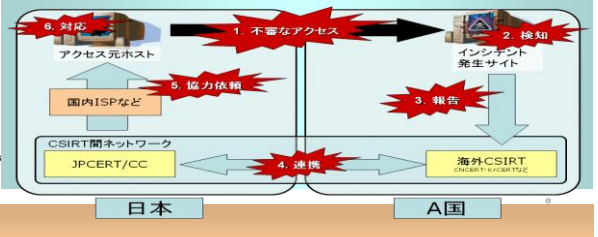
- ネットワークトラフィック情報の収集分析
- セキュリティ上の脅威情報の収集、分析、必要とする組織への提供



インシデントハンドリング

（インシデント対応 *調整*支援）

- マルウエアの接続先等の攻撃関連サイト等の閉鎖等による被害最小化
- 攻撃手法の分析支援による被害可能性の確認、拡散抑止
- 再発防止に向けた関係各関の情報交換及び情報共有



早期警戒情報

重要インフラ、重要情報インフラ事業者等の特定組織向け情報発信

CSIRT構築支援

海外のNational-CSIRTや企業内のセキュリティ対応組織の構築・運用支援

アーティファクト分析

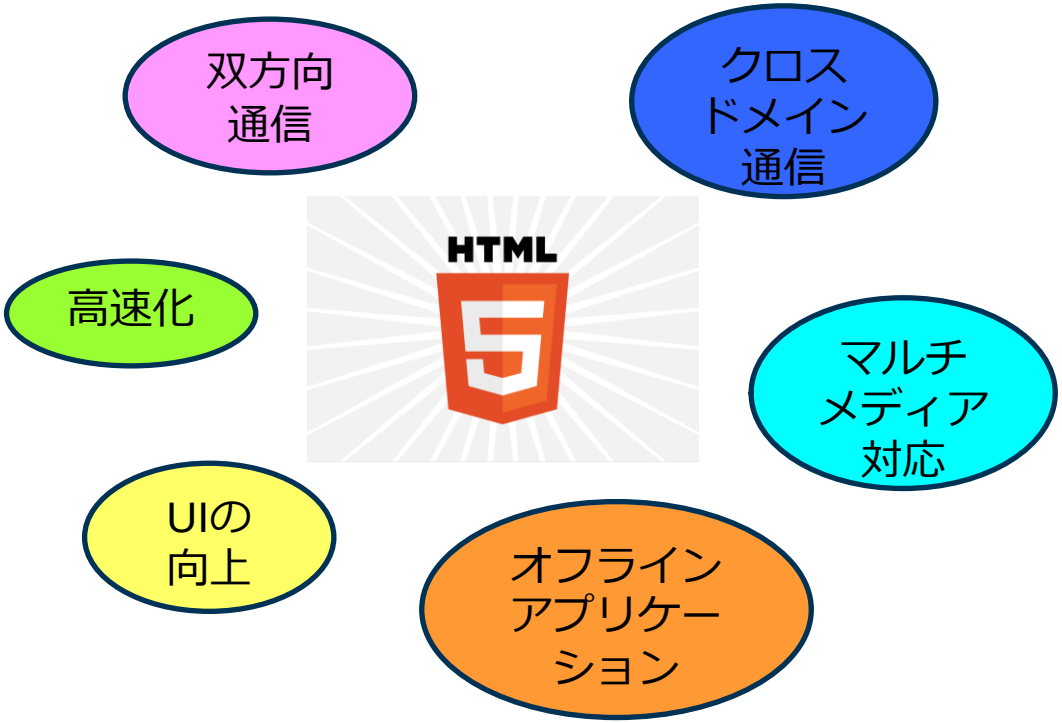
マルウエア（不正プログラム）等の攻撃手法の分析、解析

国際連携

各種業務を円滑に行うための海外関係機関との連携

HTML5とは

- 従来のHTMLに代わる次世代のHTML
- ブラウザでのデータ格納、クライアントとサーバ間での双方向通信、位置情報の取得など、従来のHTMLよりも柔軟かつ利便性の高いWeb サイトの構築が可能となる
- 日本を含むアジア太平洋地域においても急速に普及が進みつつある
- 最近のブラウザの多くは、HTML5に対応している（一部実装されていない機能もある）



HTML5とは

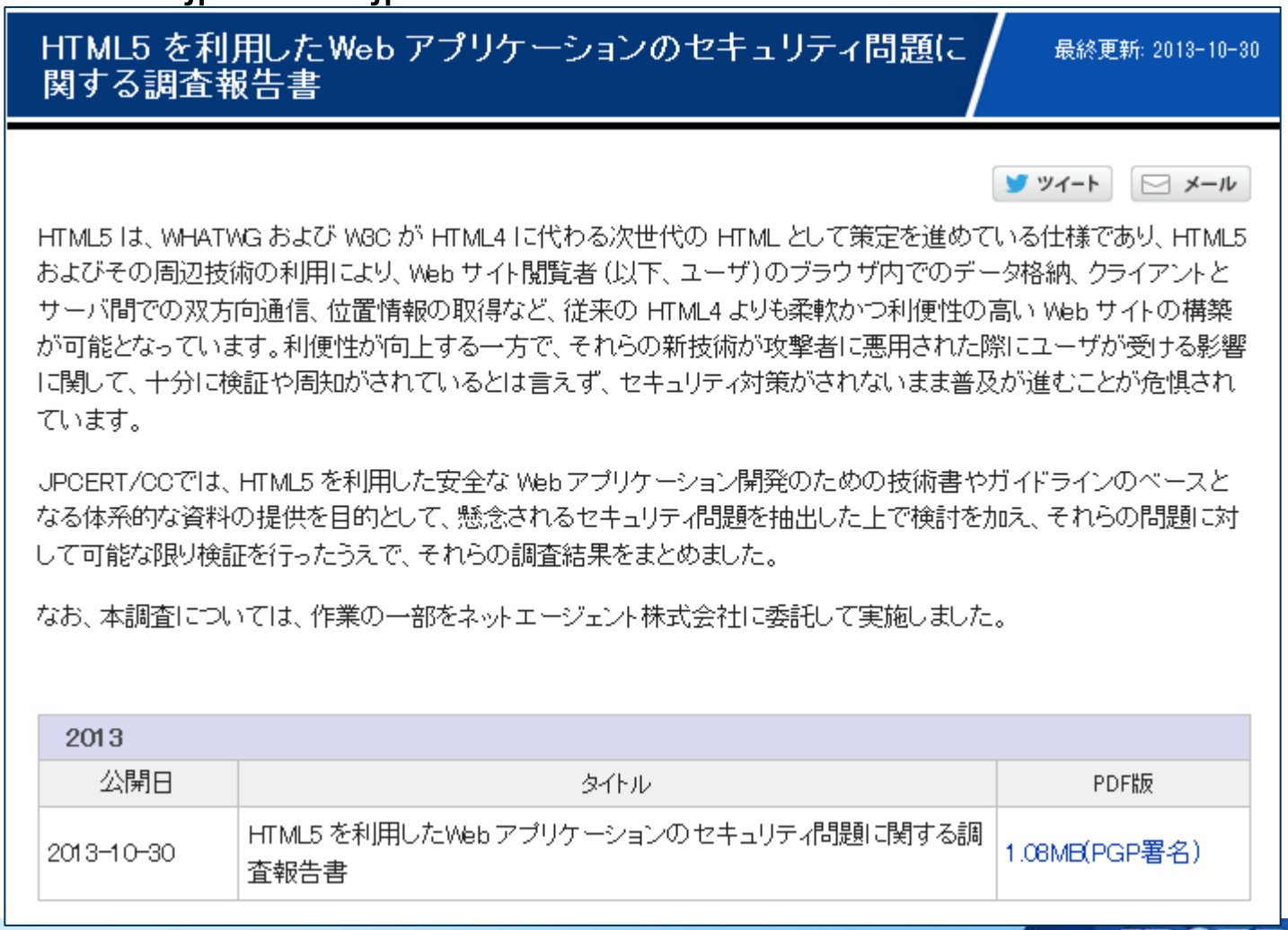
- 従来のHTMLに代わる次世代のHTML
- ブラウザでのデータ格納、クライアントとサーバ間での双方向通信、位置情報の取得など、従来のHTMLよりも柔軟かつ利便性の高いWeb サイトの構築が可能となる
- 日本を含むアジア太平洋地域においても急速に普及が進みつつある
- 最近のブラウザの多くは、HTML5に対応している（一部実装されていない機能もある）

- 従来のHTMLでは影響のなかったケースが、ブラウザのHTML5対応により、脆弱性となってしまうケースが存在する

- 利便性が向上する一方で、それらの新技術が攻撃者に悪用された際にユーザが受ける影響に関して、十分に検証や周知がされているとは言えない

HTML5 を利用したWeb アプリケーションの セキュリティ問題に関する調査報告書

- 10月30日に公開
- https://www.jpcert.or.jp/research/html5.html



HTML5 を利用したWeb アプリケーションのセキュリティ問題に関する調査報告書

最終更新: 2013-10-30

ツイート　メール

HTML5 は、WHATWG および W3C が HTML4 に代わる次世代の HTML として策定を進めている仕様であり、HTML5 およびその周辺技術の利用により、Web サイト閲覧者（以下、ユーザ）のブラウザ内でのデータ格納、クライアントとサーバ間での双方向通信、位置情報の取得など、従来の HTML4 よりも柔軟かつ利便性の高い Web サイトの構築が可能となっています。利便性が向上する一方で、それらの新技術が攻撃者に悪用された際にユーザが受ける影響に関して、十分に検証や周知がされているとは言えず、セキュリティ対策がされないまま普及が進むことが危惧されています。

JPCERT/CCでは、HTML5 を利用した安全な Web アプリケーション開発のための技術書やガイドラインのベースとなる体系的な資料の提供を目的として、懸念されるセキュリティ問題を抽出した上で検討を加え、それらの問題に対して可能な限り検証を行ったうえで、それらの調査結果をまとめました。

なお、本調査については、作業の一部をネットエージェント株式会社に委託して実施しました。

2013

公開日	タイトル	PDF版
2013-10-30	HTML5 を利用したWeb アプリケーションのセキュリティ問題に関する調査報告書	1.08MB(PGP署名)

報告書の使い方

- 技術書・ガイドラインのベース資料
- 仲間内の勉強会資料
- セミナの参考資料

などにどうぞ

引用・転載にあたっては以下を参照してください。

JPCERT/CC ご利用にあたってのお願い
https://www.jpcert.or.jp/guide.html

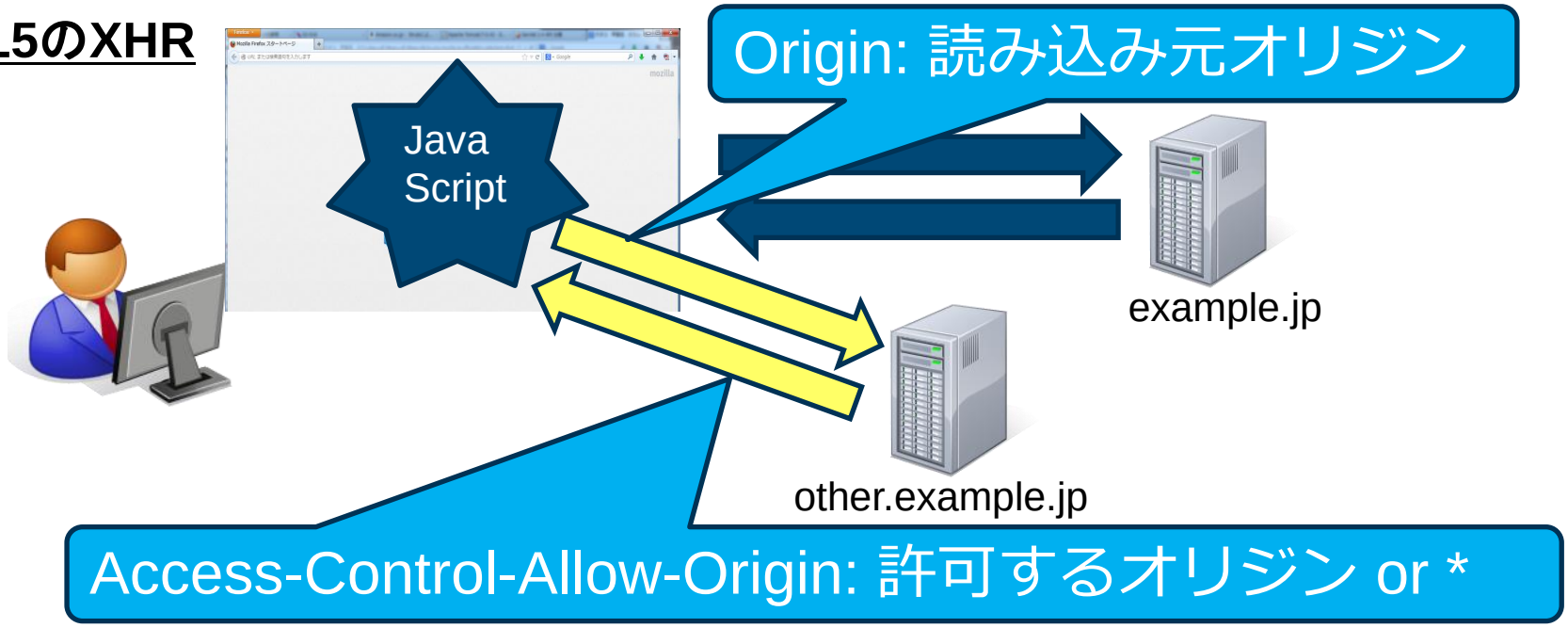
記載例）
引用元: JPCERTコーディネーションセンター
「HTML5 を利用したWeb アプリケーションのセキュリティ問題に関する調査報告書」
https://www.jpcert.or.jp/research/HTML5-20131030.pdf

JavaScript API
XMLHttpRequest

XMLHttpRequest(XHR)概要

- XMLHttpRequest(XHR)とは
  - —JavaScriptでHTTP通信を行うためのAPI
  - —非同期通信によりインタラクティブな表現が可能
  - —AJAXの普及に伴い使用される機会が増加
  - —HTML5以前は同一オリジンとの通信のみに制限されていた

HTML5のXHR



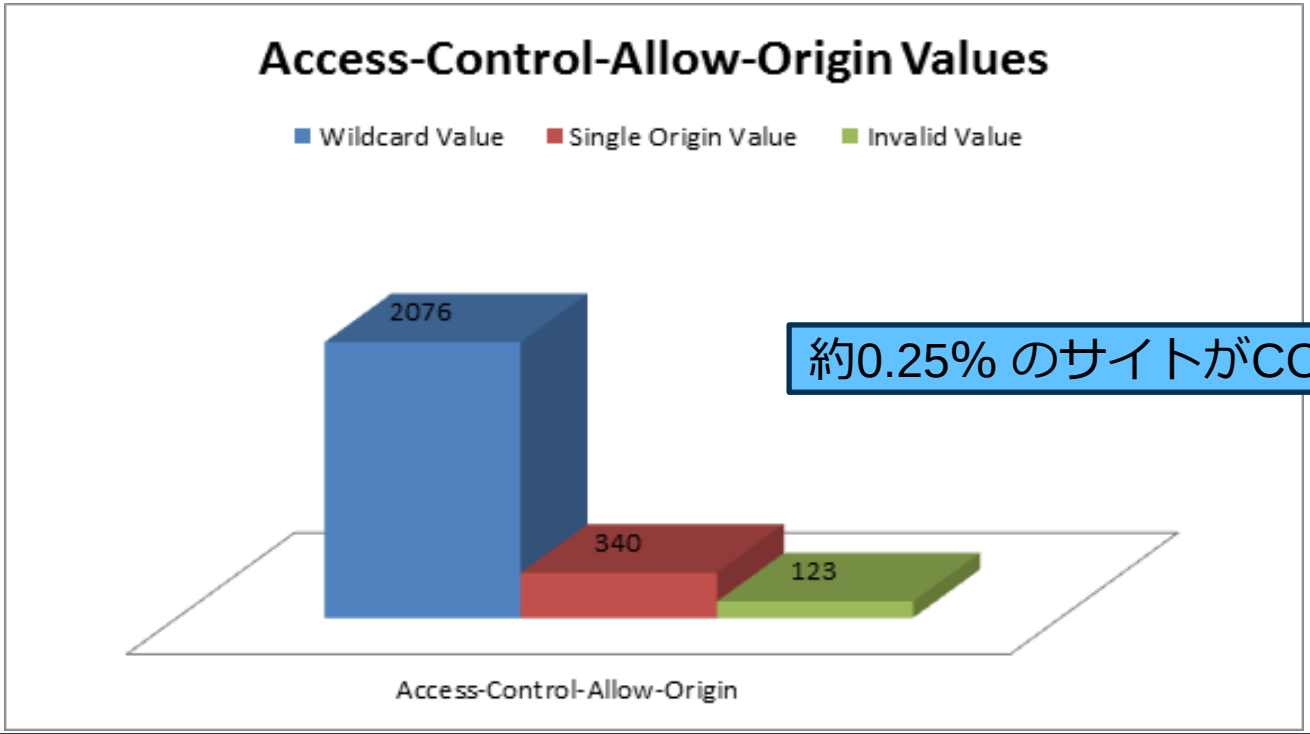
Cookieを許可するためには別途Access-Control-Allow-Credentialsが必要

※オリジン： ホスト・ポート・スキームの組み合わせ

HTML5の現状

以前に行われた調査

- 2012年11月に行われた2539サイトに対する調査



約0.25% のサイトがCORSに対応

引用元：http://www.veracode.com/blog/2012/11/security-headers-report/

- Access-Control-Allow-Origin以外のCORS関連のヘッダの状況は？？

HTML5の利用についての調査

- HTML5の利用状況について確認するため、以下を調査した
  - CORS関連ヘッダ(*1)の利用状況
  - セキュリティ関連ヘッダ(*2)の利用状況

- 調査方法
  - Alexa Top 1,000,000 (*3)に含まれるサイトのトップページをクロール
  - Curlコマンドのレスポンスの HTTPヘッダを調査
  - リダイレクトされる場合は、最終リダイレクト先が対象
  - 送信するリクエストには、Originリクエストヘッダを追加
  - 調査期間は、2013/12/26 – 2013/12/30

(*1) CORSで使われるヘッダ名が「Access-Control-」から始まるレスポンスヘッダ

(*2) 調査報告書で紹介したセキュリティ関連機能に関するヘッダ

(*3) http://www.alexa.com/topsites

CORS関連レスポンスヘッダ

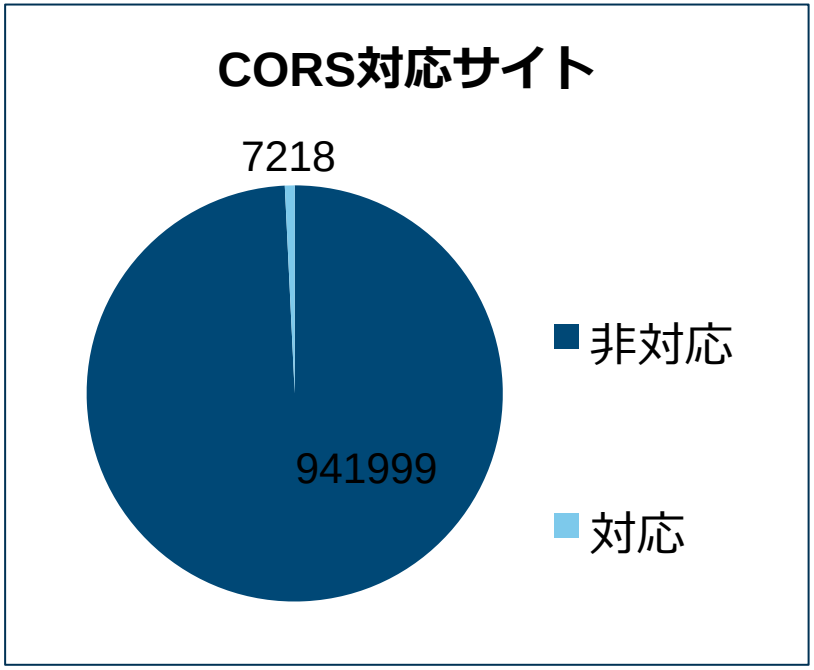
- Access-Control-Allow-Origin
  - —リソースへのアクセスを許可するオリジンを指定
- Access-Control-Allow-Credentials
  - —Cookie等の認証情報を含んだリクエストに対するレスポンスへのアクセスを許可する場合にtrueを指定
- Access-Control-Expose-Headers
  - —ブラウザが使用してもよいヘッダを指定
- Access-Control-Allow-Methods（プリフライト）
  - —送信を許可するメソッドを指定
- Access-Control-Allow-Headers（プリフライト）
  - —送信を許可するヘッダを指定
- Access-Control-Max-Age（プリフライト）
  - —プリフライトレスポンスをキャッシュする時間を指定

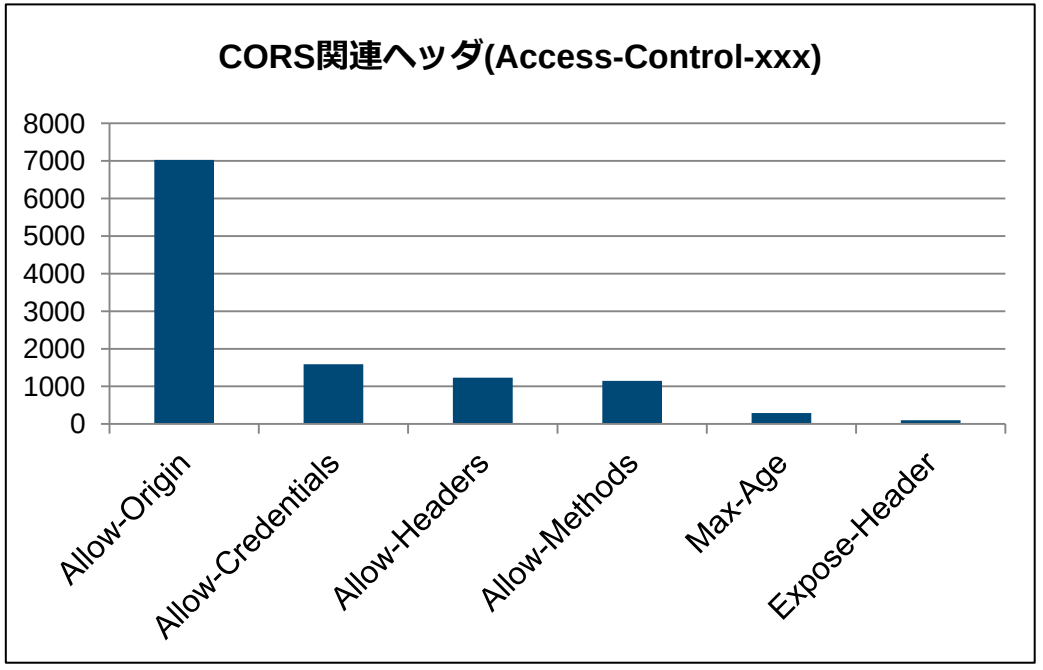
セキュリティ関連レスポンスヘッダ

- X-XSS-Protection
  —XSS攻撃からの保護
- X-Content-Type-Options
  —Content-Typeヘッダに従ったコンテンツの取り扱い
- X-Frame-Options
  —フレームへの埋め込みを制限
- Content-Security-Policy
  —コンテンツの読み込み元を制限
- Content-Disposition
  —ファイルのダウンロードダイアログの制御
- Strict-Transport-Security
  —HTTPSの強制

調査結果

CORS対応状況

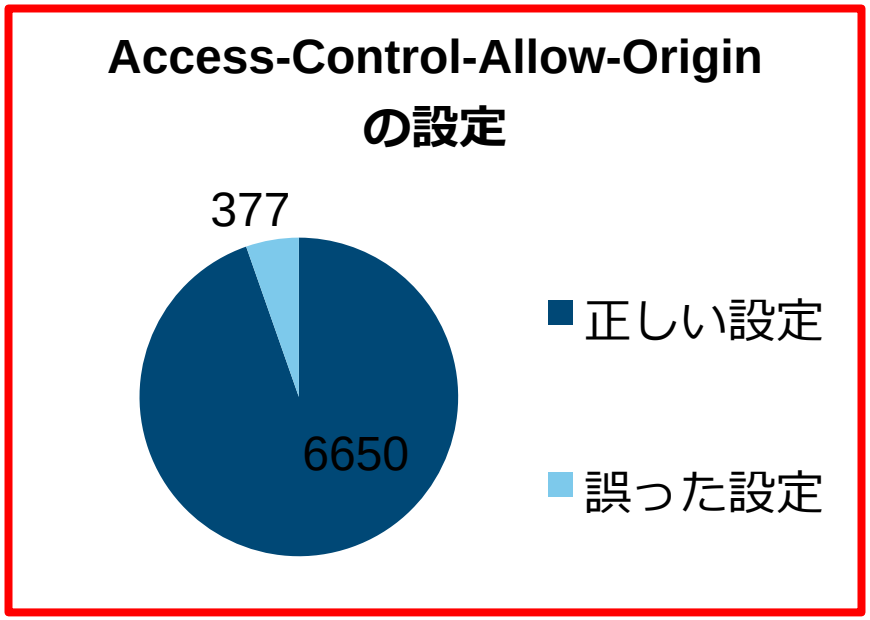


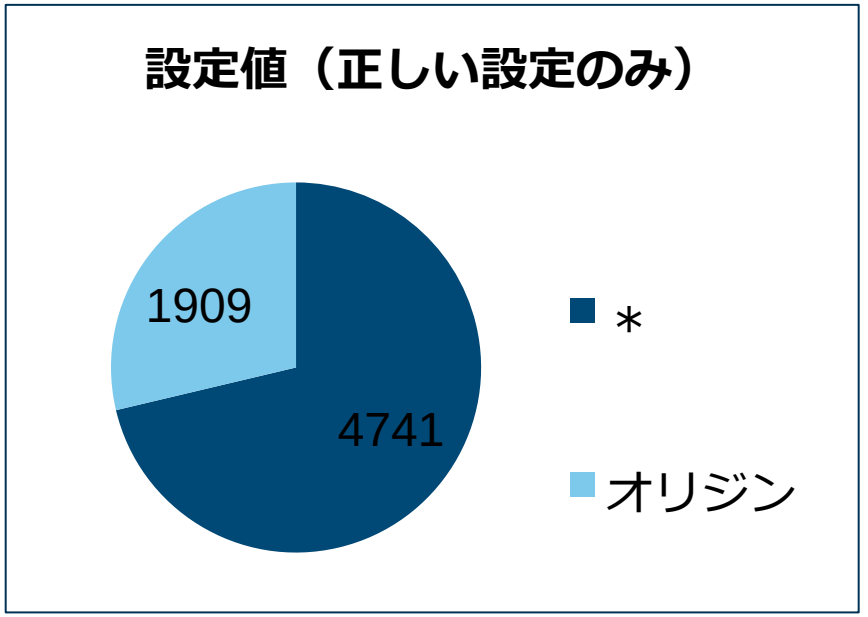


- レスポンスを受け取ったサイト数は、949,217サイト
- CORS対応サイト(*)は、7218サイト
- CORS対応サイトは、全体の約0.76%

(*) CORS関連ヘッダを返してきたWebサイト

Access-Control-Allow-Origin





- 約5.4%のサイトが誤った設定を行っていた
- 誤った設定では、設定された値は無視されるため、Acecss-Control-Allow-Originヘッダを設定しても効果がない

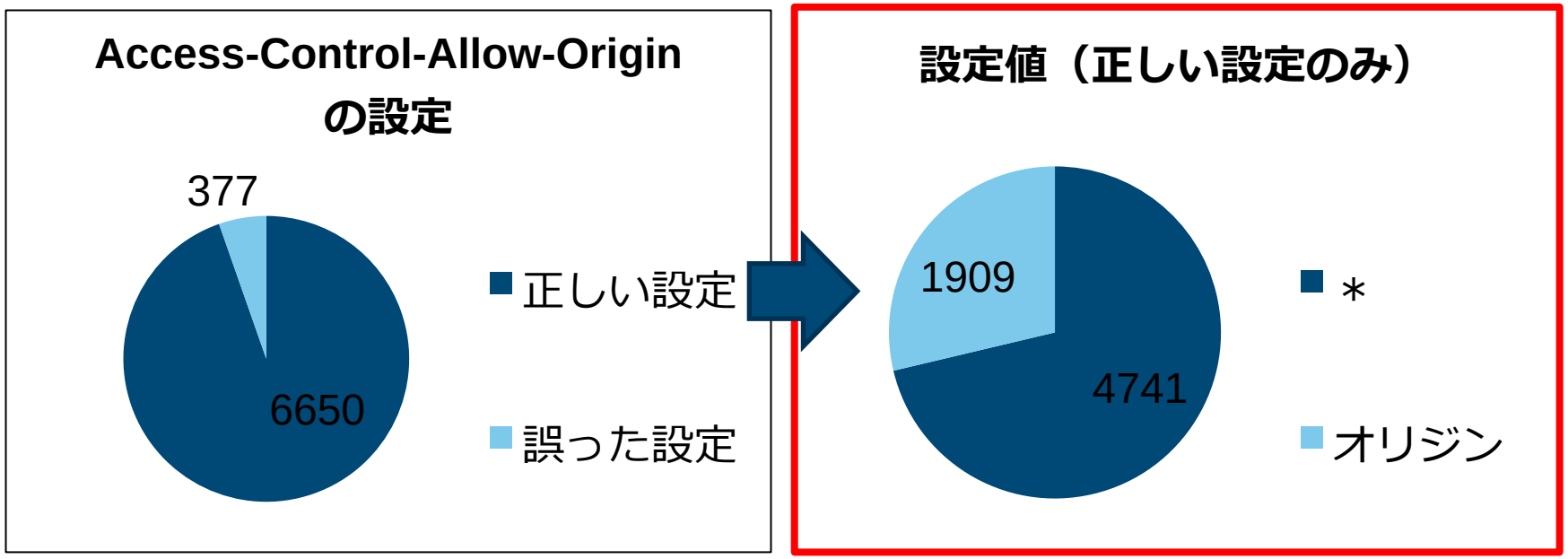
Access-Control-Allow-Originの設定

- 正しい設定
  - scheme://host[:port][scheme://host[:port]]* (※1)
  - null
  - *
- 誤った設定の例（いずれの設定も無効）
  - スキームなし：　example.com
  - カンマ区切りで複数：　http://example1.com,http://example2.com
  - ワイルドカード使用：　http://*.example.com
  - オリジンの末尾に /：　http://example.com/
  - 複数のヘッダ：　http://example1.com
http://example2.com

- 内容を理解せず安易にヘッダを付加した場合、アクセス制限が弱まり脆弱となる場合があることに注意

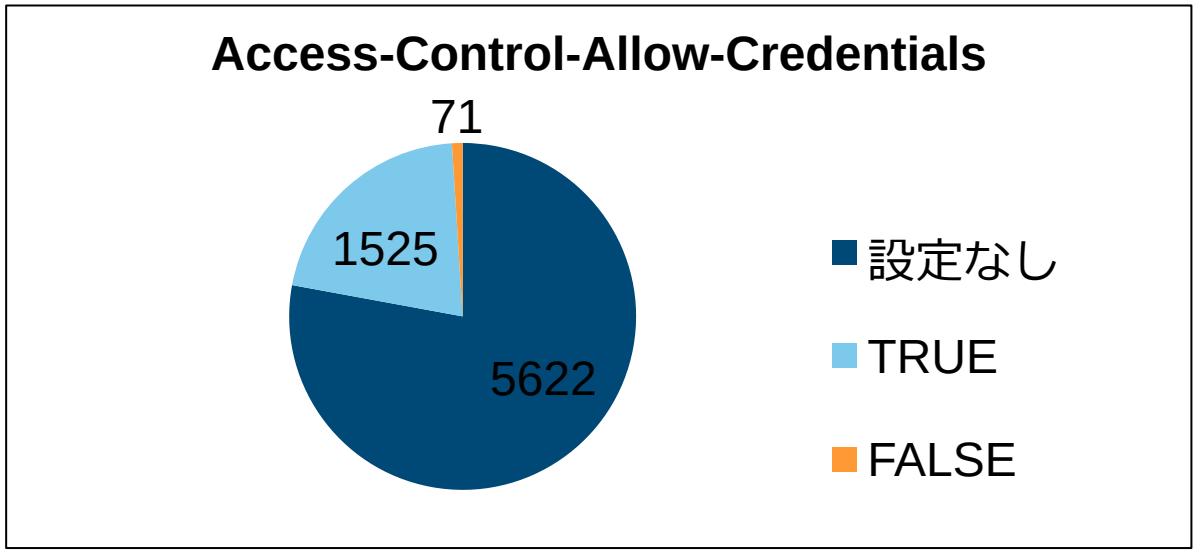
※1：スペース区切りの複数オリジンの記載は、ブラウザの実装によっては、禁止されている場合がある

Access-Control-Allow-Origin



- Access-Control-Allow-Originの設定を正しく行っているWebサイトのうち
  - 約71%が値に*を設定
    ⇒全てのWebサイトからのCORSを許可
  - 約29%が値にオリジンを設定
    ⇒オリジンで指定したWebサイトからのCORSのみを許可

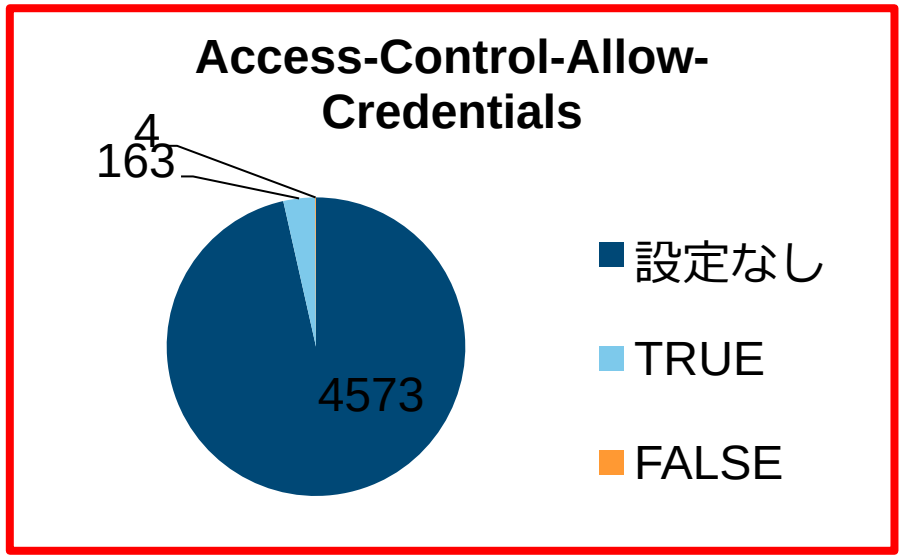
Access-Control-Allow-Credentials



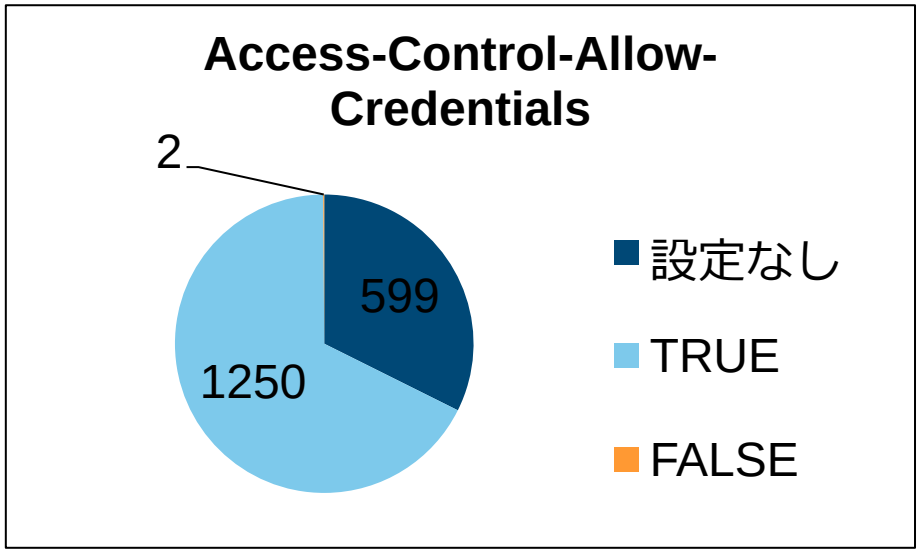
- Access-Control-Allow-Credentialsを設定しているサイトは、CORS対応サイト全体の約22%
- 設定値は「true」のみが意味を持つ設定値。それ以外は全て「true」が設定されていない場合と同様の動作となるが「false」と指定されているものも複数存在

Access-Control-Allow-Credentials

Access-Control-Allow-Originに*を設定しているWebサイト



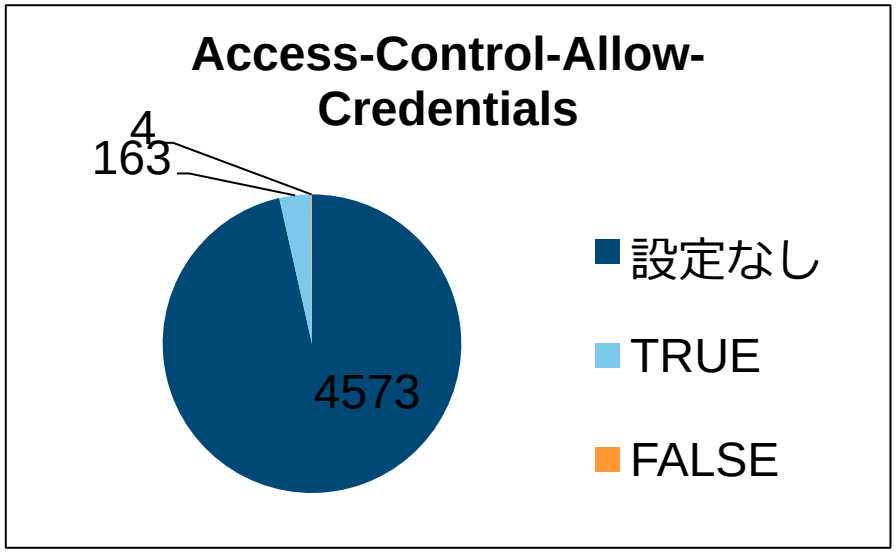
Access-Control-Allow-Originに適切にオリジンを設定しているWebサイト



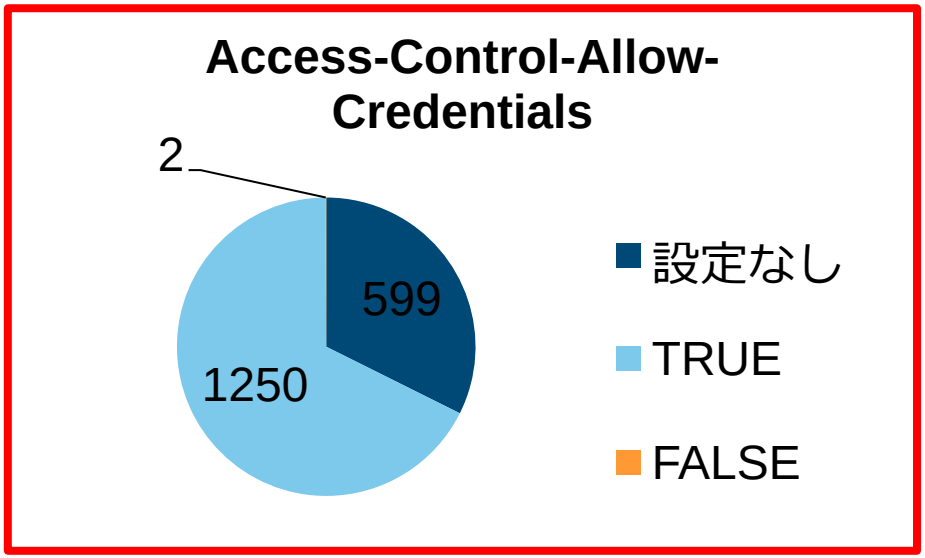
- Access-Control-Allow-Credentialsにtrueを指定する場合、Access-Control-Allow-Originに*を使うことはできないが163サイトが*を設定
- 意図したとおりの動作となっているか事前に確認を

Access-Control-Allow-Credentials

Access-Control-Allow-Originに
*を設定しているWebサイト



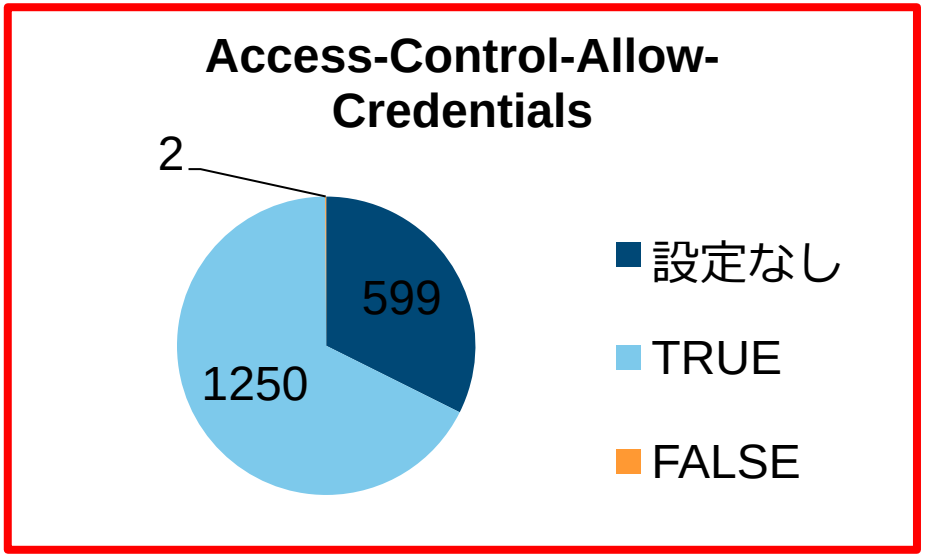
Access-Control-Allow-Originに
適切にオリジンを設定しているWebサイト



- Access-Control-Allow-Originでオリジンを指定しているサイトの約68%で、Access-Control-Allow-Credentialsの値にtrueを指定

Access-Control-Allow-Origin

Access-Control-Allow-Originに
適切にオリジンを設定しているWebサイト



送られるレスポンスヘッダ例：trueの場合
Access-Control-Allow-Origin: http://example.com
Access-Control-Allow-Credentials: true

- 上記例のスポンスヘッダが返った場合、http://example.comから読み込んだJavaScriptは、ユーザがブラウザを操作する場合と同様のふるまいをすることが可能。

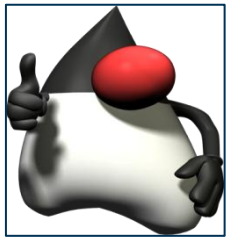
  ⇒Access-Control-Allow-Originで指定したサイトで、任意のJavaScriptが実行できる場合、非常に危険な状況となる
- 脆弱性がなければ大丈夫？

マッシュアップサイトの場合

A-C-Allow-Origin: siteA
A-C-Allow-Credentials:true

マッシュアップ元サイト

X さんの画像




A-C-Allow-Origin:siteB
A-C-Allow-Credentials:true

マッシュアップサイトA

Cookie

Origin: siteA

Origin: siteB

マッシュアップサイトB


各サイトにCookie付きCORSを許可するため、レスポンスヘッダを動的に生成

お気に入り画像を背景として利用

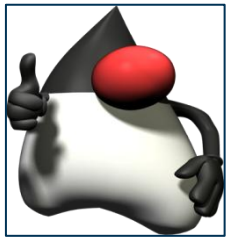
お気に入り画像の関連商品を提供

危険な動作の例

A-C-Allow-Origin:
http://evil.example.com
A-C-Allow-Credentials:true

マッシュアップ元サイト

X さんの画像



悪意あるサイト
（evil.examle.com）

Cookie

Origin: http://evil.example.com

個人情報
漏えい

外部に送信

Xさん

XHR2を用いた他源泉リクエスト

XHR2 を突破できるケース

セキュリティの配慮がある XHR2 であっても、次のような条件が成立する場合にはリクエスト強要攻撃が成立し得る。

- クライアントから送られてきた Origin: *origin* リクエストヘッダに対して、サーバが必ず Access-Control-Allow-Origin: *origin* レスポンスヘッダを返すようになっている。かつ、
- サーバが Access-Control-Allow-Credentials: true レスポンスヘッダを発行するとともに、Cookie を用いたセッション維持を行っている。

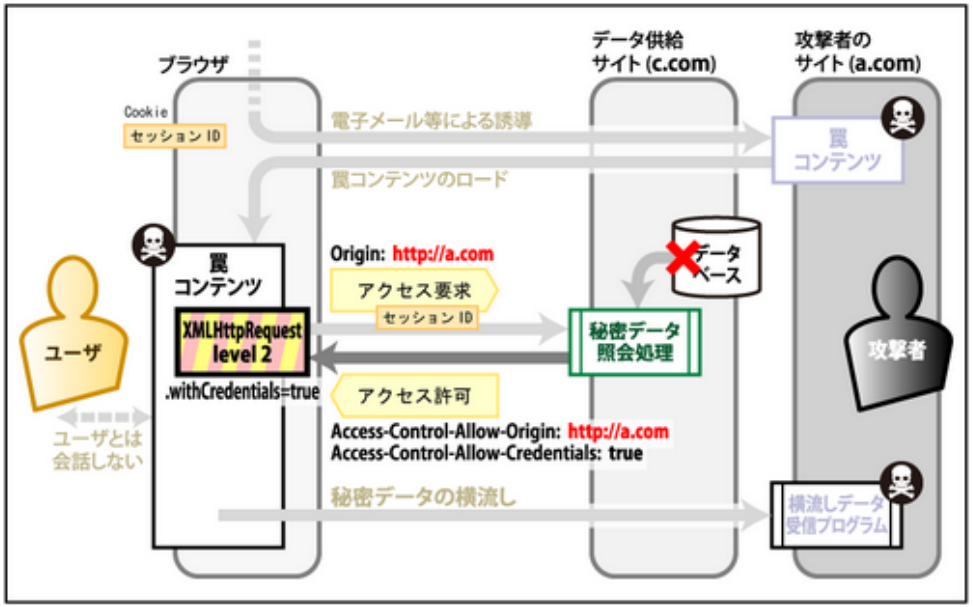


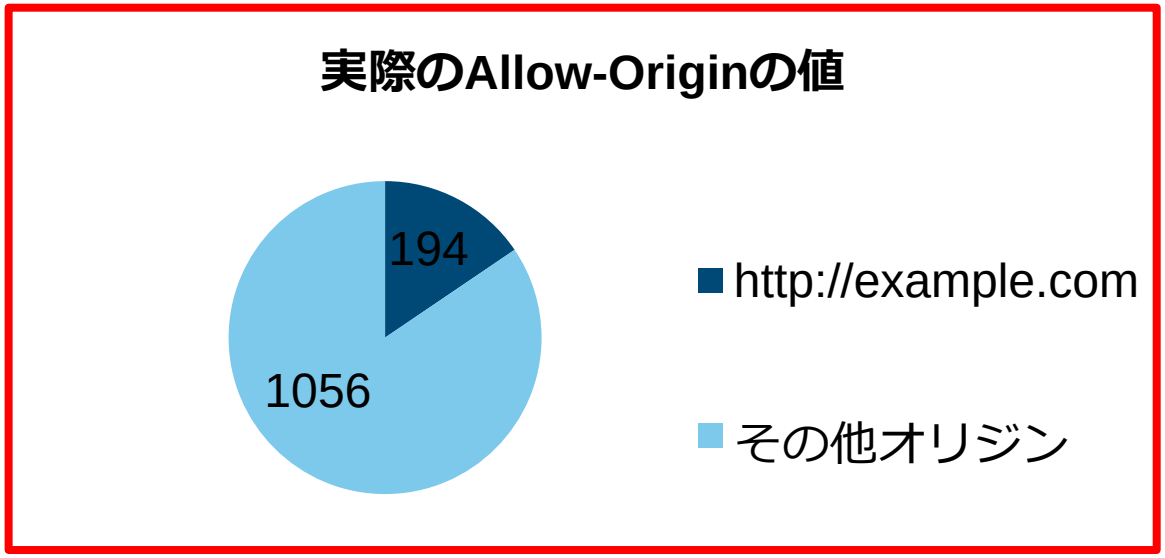
図8-16：XHR2 を突破できるケース

IPA セキュアプログラミング講座 Webアプリケーション編
https://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/704.html

Access-Control-Allow-Origin

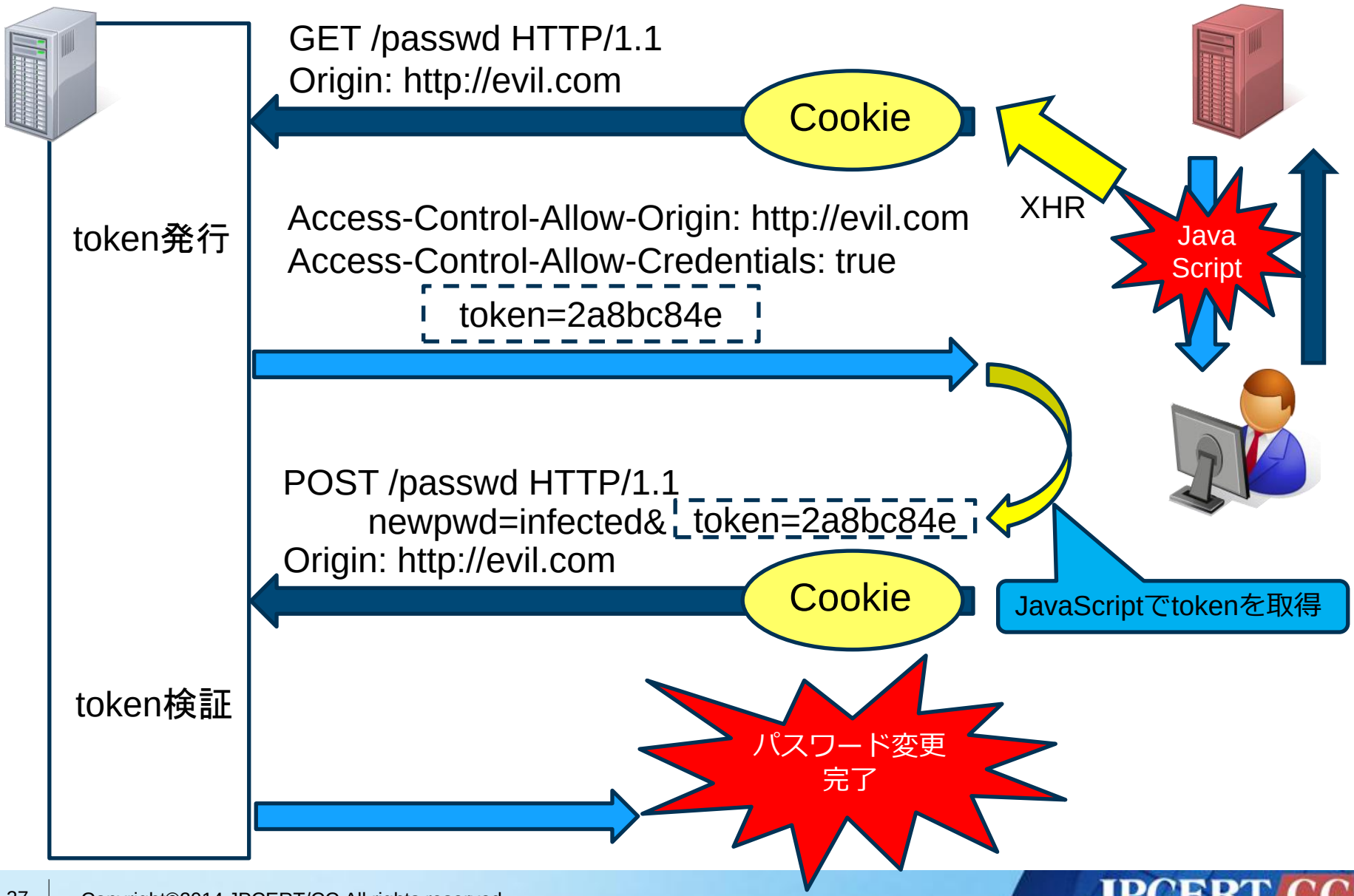
今回の調査で
実際に送ったリクエスト：

GET / HTTP/1.1
HOST: <target>
Origin: http://example.com



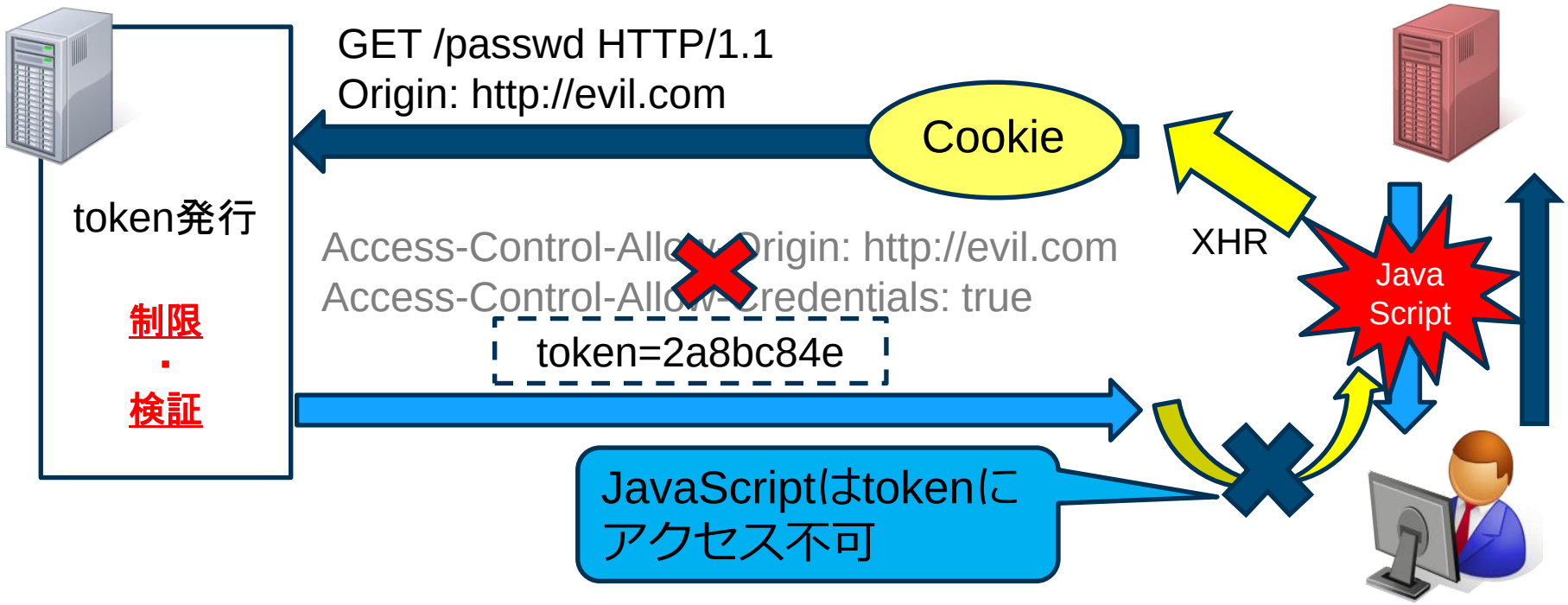
- 194サイトが、送ったOriginヘッダの値を、Access-Control-Allow-Originヘッダにそのまま設定して返送！！
- 認証済みのユーザのみに限定すべきページでも上記動作を行っている場合、情報が盗まれる可能性がある
- さらに、トークンによるCSRF対策が回避されるケースも

CSRFによるパスワード変更



正しい設定

- 許可するページの制限
  - 認証を行っているユーザのみに限定するべきページには、Access-Control-Allow-Origin、Access-Control-Allow-Credentialsを付加しない（設定変更ページ、パスワード変更ページ等）
- オリジンの検証
  - 特定のサイトに対して許可する場合には、Originの値を検証し、許可する場合にのみAccess-Control-Allow-Origin、Access-Control-Allow-Credentialsを付加する



設定例

※実際に設定する場合には、アプリケーションの仕様を踏まえた設定の検討と、十分なテストが必要です

■ 許可するページの制限

—認証を行っているユーザのみに限定するべきページには、Access-Control-Allow-Origin、Access-Control-Allow-Credentialsを付加しない（設定変更ページ、パスワード変更ページ等）

Apache での設定例：画像ファイルのみでリソースの使用を許可

```
<FilesMatch “¥.(jpeg|gif|png)$”>
  SetEnvIf Origin “^https?://.*$” ORIGIN=$0
  Header set Access-Control-Allow-Origin %{ORIGIN}e env=ORIGIN
</FilesMatch>
```

■ オリジンの検証

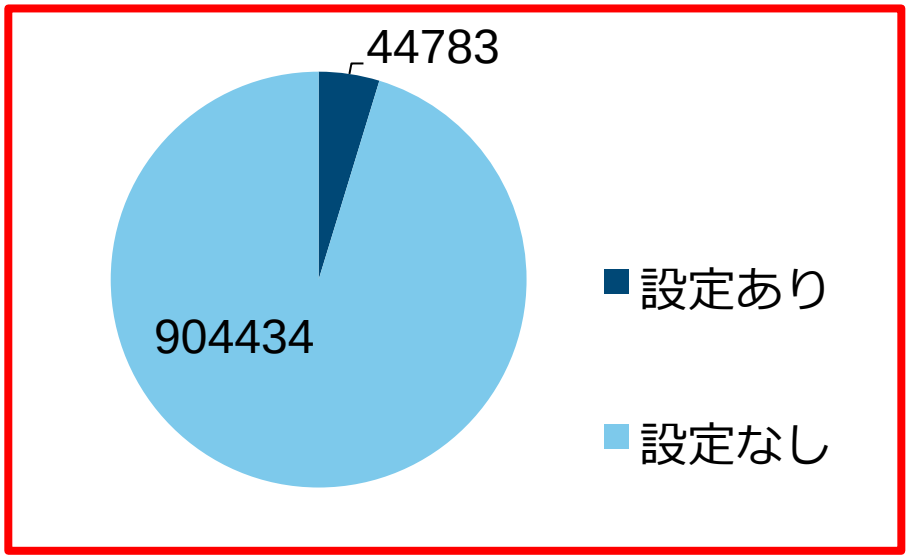
—特定のサイトに対して許可する場合には、Originの値を検証し、許可する場合にのみAccess-Control-Allow-Origin、Access-Control-Allow-Credentialsを付加する

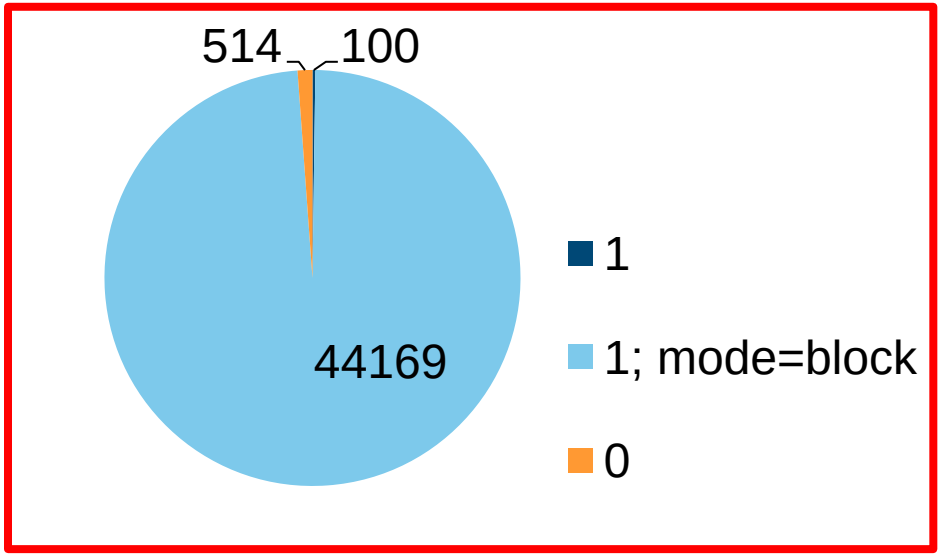
Apache での設定例：example.comのサブドメインのみにリソースの使用を許可

```
SetEnvIf Origin “^https?://.*¥.example¥.com$” ORIGIN=$0
Header set Access-Control-Allow-Origin %{ORIGIN}e env=ORIGIN
```

セキュリティ関連ヘッダ

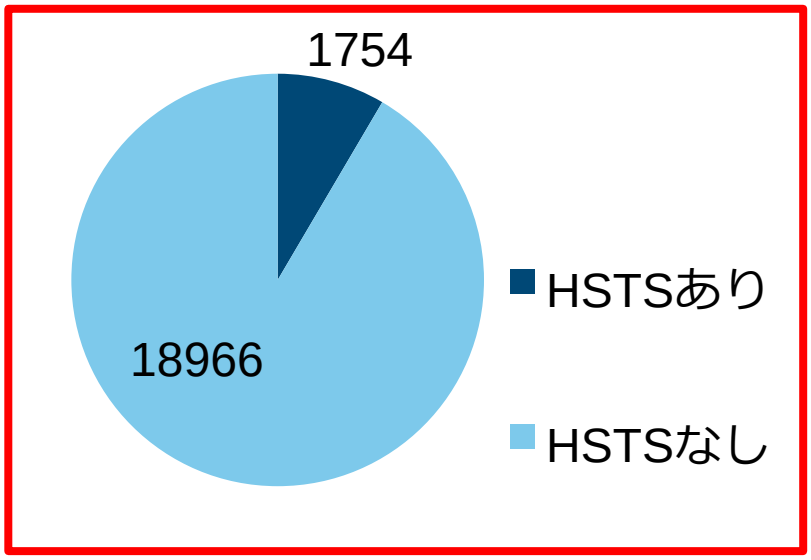
X-XSS-Protection

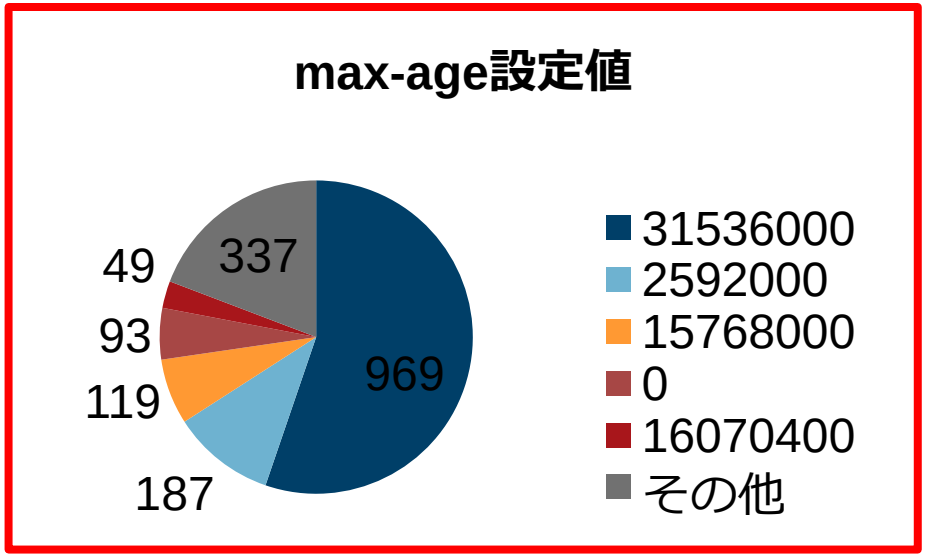




- X-XSS-Protectionを設定しているサイトのほとんどが1;mode=block(保護フィルタ有効、検出時には空白ページを表示)
- 514サイトが0 (無効)を設定
- **XSS保護フィルタの無効は、誤検出があるなど特別な理由があるページのみに限定すべき**

Strict-Transport-Security





- 全サイト中20720サイトがHTTPSにリダイレクト
- 1754サイトがHSTSを設定
- 半数以上がHSTSの期限を1年に設定
- **0を設定した場合、ブラウザはHSTSのリストから該当のサイトを削除することに注意**

まとめ

まとめ

- HTML5では便利になる一方で、使用には注意が必要な機能も多数ある

- 「HTML5を利用したWebアプリケーションのセキュリティ問題に関する調査報告書」をセキュアなWebアプリケーションの開発に役立ててください

- お問い合わせは以下まで
  —ご意見お待ちしています

JPCERTコーディネーションセンター
Email：ww-info@jpcert.or.jp
Tel：03-3518-4600
Web: https://www.jpcert.or.jp/

おまけ

Custom HTTP header

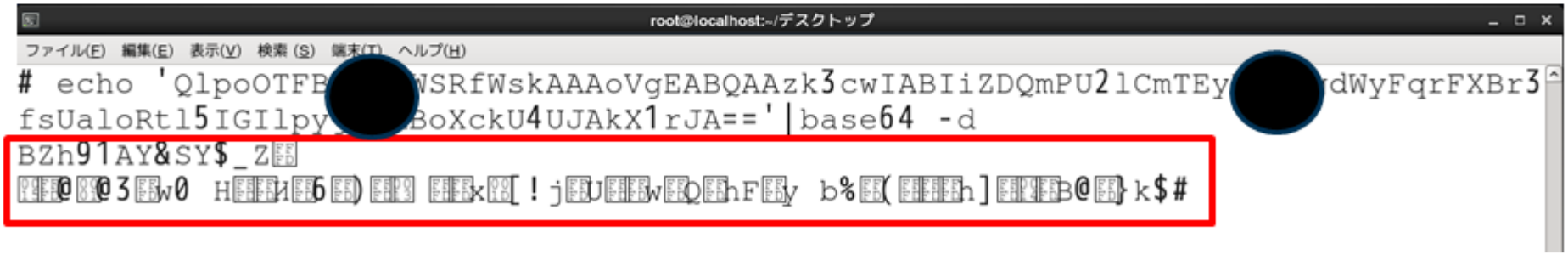
- X-Recruiting: We're looking for talented people, join us: <URL>(We have cookies!)
- X-Recruiting: Like HTTP headers? Come write ours: <URL>
- x-poetry: Choose Life. Choose a job. Choose a career.
- X-<CompanyName>-jobs: you're reading this … come work at xxx!
- Were-currently-looking-for-devs-like-you: Tweet @xxx for job details.

X-Want-A-Job-With-Us response header

X-Want-A-Job-With-Us:
QlpoOTFBW[redacted]WSRfWskAAAoVgEA[redacted]Azk
3cwIABIiZDQ[redacted]J2lCmTEy[redacted]gdWy[redacted]XBr3
fsUaloRtl5IGIlpyjl8dBoXckU4[redacted]AkX1rJA==

?

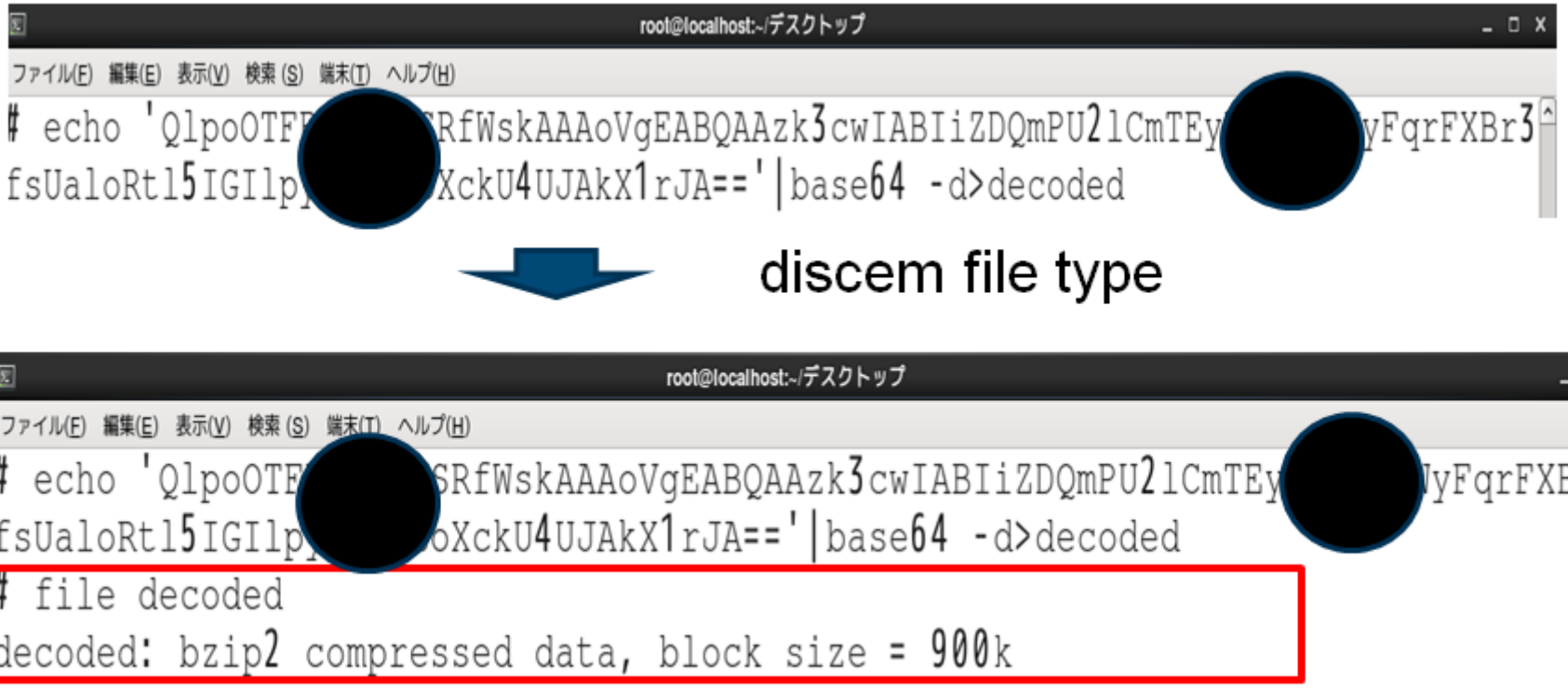
decode by Base64



Binary ! ! ?

discem file type

output to a file



Compressed file…. :-c

extraction by bzip2

extraction

```
root@localhost:~/デスクトップ
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
# bunzip2 -v decoded
bunzip2: Can't guess original name for decoded -- using decoded.out
  decoded: done
```

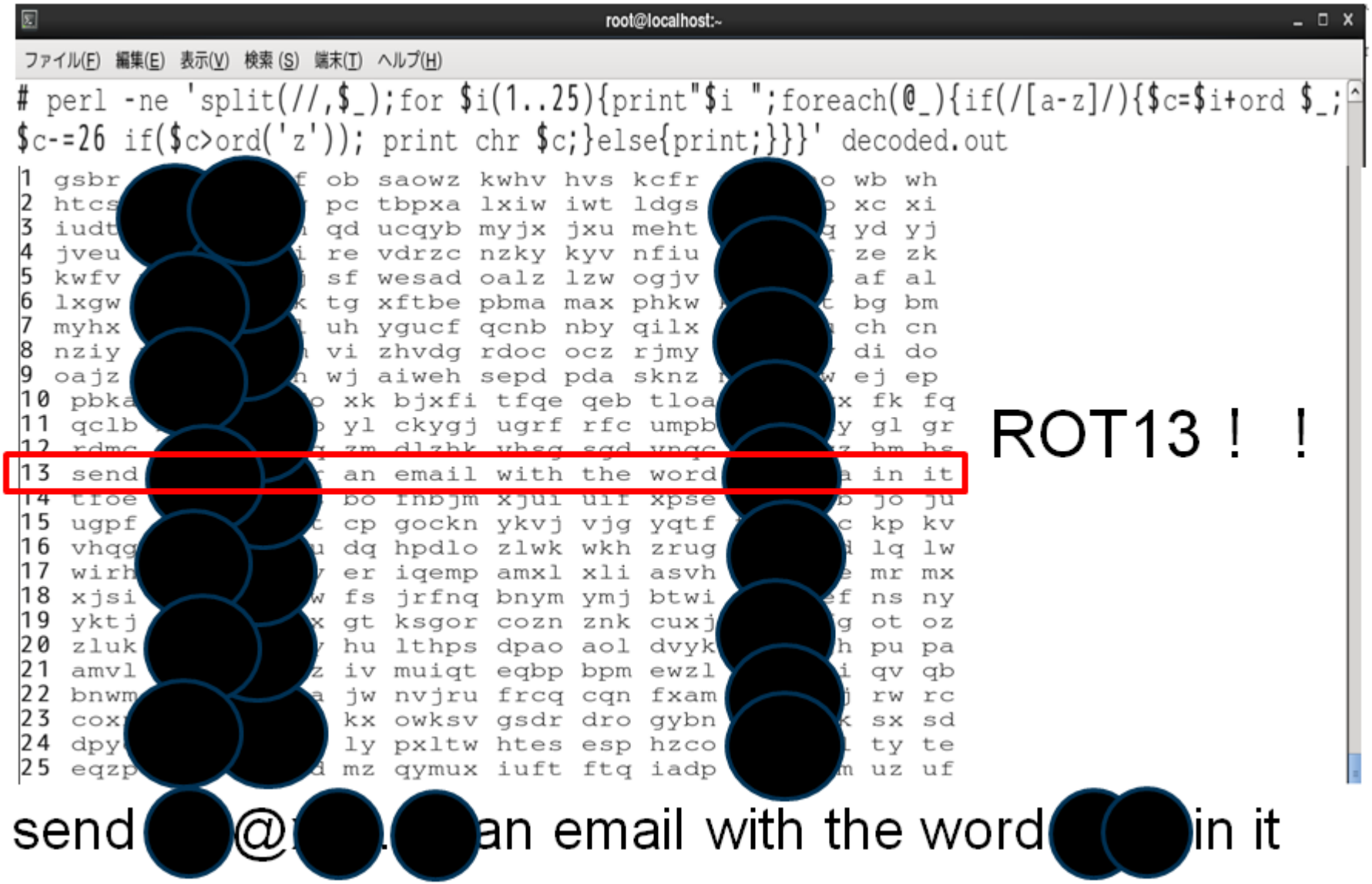
confirm file type

```
root@localhost:/
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
# bunzip2 -v decoded
bunzip2: Can't guess original name for decoded -- using decoded.out
  decoded: done
# file decoded.out
decoded.out: ASCII text, with no line terminators
```

ASCII！ Readable！ ！

finish?

perl



ご静聴ありがとうございました