Credential Stuffing の攻撃対象について

Akamai

はじめに

Credential Stuffing 攻撃のリスクにさらされている組織にとって、攻撃を効果的に緩和できるかどうかは、選択するボット管理ベンダーやソリューション以上に、Web サイトのアーキテクチャに依存します。Web サイトのアーキテクチャは、どのセキュリティソリューションを利用する場合でも、その実効性に大きな影響を与えます。

その理由を理解するには、攻撃の仕組みを知り、セキュリティソリューションがどのように攻撃を防御するのかを考える必要があります。Credential Stuffing 攻撃は、ボットネットを利用してアプリケーションへのログイン情報を盗み出し、その認証を自動化します。ボットと正当な利用者とを区別するために、今日の高度なボット検知テクノロジーでは、ウェブページの保護に JavaScript インジェクションが使用され、ネイティブ・モバイル・アプリが使用する API の保護にモバイルソフトウェア開発キット（SDK）が使用されています。Web サイトのアーキテクチャやサイトとやり取りするクライアントのタイプによっては、アタックサーフェスを最小限に抑える機能が制限される可能性があります。

このホワイトペーパーでは、今日のボット管理ソリューションを効果的に採用するうえでのアーキテクチャ上の課題の背景、Credential Stuffing 攻撃を緩和する理想的な Web サイトアーキテクチャ、およびアタックサーフェス（攻撃の対象となり得る領域）を縮小するための具体的な中間オプションと各オプションのリスクや制限についても説明します。

ボット検知の仕組み

ボットの検出には、リクエストの送信元が人間かどうかを特定することが求められます。単純なボット検知手法では、ヘッダーを見るなどして、リクエストのコンテンツを調べます。ところが、リクエストヘッダーは簡単にスプーフィングできるため、このようなアプローチはより高度なボットには効果がありません。より高度なボット検知手法では、JavaScript チャレンジ、ブラウザーフィンガープリント、およびふるまい異常分析が使用されています。

高度なボット検知手法では、JavaScript チャレンジ、ブラウザーフィンガープリント、およびふるまい異常分析を使用しています。

ブラウザーセッションに JavaScript を挿入する

ボット検知手法の多くは、クライアントのブラウザーに配信されるウェブページの HTML に JavaScript コードのスニペットを挿入しています。JavaScript コードのスニペットは、ブラウザーに HTML がロードされるときに実行され、人間とボットを区別するためのさまざまな手法に採用できます。

上記では最も洗練度が低い手法である JavaScript チャレンジでは、JavaScript を実行できるブラウザーがリクエストの送信元であるかどうかの確認が行われます。JavaScript チャレンジは、JavaScript を実行できない単純なボットの検出には効果があると言えます。

より高度な手法としてあげられるブラウザーフィンガープリントでは、ブラウザーのタイプやバージョン、画面の解像度、インストールされているプラグインやフォントなど、ブラウザーのさまざまな特性が収集されます。ボット管理ソリューションでは、このようなブラウザーフィンガープリントを検査して、クライアントが正当なブラウザーをスプーフィングしようとするボットであることを示す異常を特定できます。

今日利用できる最も高度な手法としてあげられるふるまい異常分析では、クライアントの入力デバイスからコンピューターのキーボードストロークやマウスの動き、モバイルデバイスのタッチイベントやジャイロスコープ／加速度計の測定値などの、ふるまいに関するデータが収集されます。このデータはボット管理ソリューションに返され、ボットであることを示すふるまいパターンの偏差が分析されます。

アプリのモバイル SDK

JavaScript インジェクションベースの手法は、デスクトップやモバイルのブラウザーなど、ブラウザーベースのクライアントには効果的です。ところが、ネイティブ・モバイル・アプリや自動サービスなど、API ベースのクライアントの多くは、JavaScript を実行できません。その結果、JavaScript インジェクション手法の場合、クライアントからは応答を得られません。適切な応答がないと、ボット管理ソリューションではクライアントがボットであると仮定し、対策が講じられます。

このような状況を打開するために、ボット管理のベンダーはモバイル SDK を提供することで、ボット検知をネイティブ・モバイル・アプリと統合しています。モバイル SDK により、モバイルアプリではアプリ内で必要なデータを収集し、それを分析するボット管理ソリューションに転送することができます。

Web サイトアーキテクチャを理解する

昨今の Web サイトは多数のウェブページで構成され、さまざまなタイプのクライアントやトラフィックに対応できるため、複雑で無秩序という特性があります。さらに、Web サイトの所有権は、多くの場合、組織全体に分散しています。Web サイトのアーキテクチャを理解し、クライアントがどのように別のページからログインエンドポイントに取り込まれるのかを知ることは、Credential Stuffing 攻撃を緩和し、リスクレベルを把握するために不可欠です。

エンドポイントとは？

エンドポイントとは、クライアントがアクセスできる個別の URL のことです。Credential Stuffing 攻撃の緩和には、ユーザー認証情報を検証するトランザクション URL の特定と保護が必要です。たとえば、URL は既存のアカウントへのログインやアカウントの新規作成に使用されることがあります。Credential Stuffing にとどまらず、保護が必要なエンドポイントにはギフトカードの残高確認、フライト情報の検索、ショッピングカードへの商品の追加などを行うための URL が含まれている場合があります。

あらゆるエンドポイントを特定する

組織の多くは、複数のエンドポイントを持つ Web サイトを運用しているため、Credential Stuffing の格好の標的になる可能性があります。たとえば、金融サービス機関は消費者、小規模ビジネス、雇用主などにサービスを提供しており、それぞれに異なるログインエンドポイントがあります。さらに、これらの業務には、個別のアカウントにサインアップするためのエンドポイントが存在する場合もあります。

組織には IT や業務担当者の数人しか知らない二次エンドポイントが存在していることがあります。それらは多くの場合、そのエンドポイントの存在がほとんど誰にも知られていないレガシーアプリですが、執拗な攻撃者はそれを見つけ出すことができます。それらには、API、リテールキオスク、カスタマー・サービス・チャットボットなどにサービスを提供するエンドポイントが含まれている場合があります。

ボット管理の実装を計画するための最初のステップは、保護が必要なすべてのものを洗い出すことです。

保護が必要なものを特定する

ボット管理の実装を計画するための最初のステップは、保護が必要なすべてのものを洗い出すことです。ただし、そのような作業は、多くの場合、昨今の Web サイトのサイズや複雑さが原因で、たやすいことではありません。以下に例を示します。

- 保護が必要なものという観点から、ウェブページについて考えることは多くても、エンドポイントについては必ずしてもそうではありません。
- シングルログインのエンドポイントは、複数のページで使用されることが多く、たとえば、ユーザーは多くの場合、ログイン認証情報をオンライン小売サイトのページから入力します。ところが、認証情報はすべて、同じログインエンドポイントに送信され、検証が行われます。
- エンドポイントの維持を担当するチームは、エンドポイントにつながるページを作成するチームでないことがよくあります。たとえば、ネイティブ・モバイル・アプリは、デスクトップユーザー向けのウェブページを開発するチームと連携していない、別のチームによって開発されていることがよくあります。
- 特定のエンドポイントのセキュリティを確保するには、エンドポイントにクライアントがアクセスするあらゆる方法を理解し、対象ページの保護を展開することが求められます。すべての当事者が密接に連携しないと、保護されていないエンドポイントにアクセスしてしまうようなやり方に簡単にたどり着くことになります。

多様なタイプのクライアントの保護を構築する

Web サイトは、組織のニーズに応じて、多様なタイプのクライアントとやり取りする場合があります。例として以下が挙げられます。

- デスクトップ、ノートパソコン、またはモバイルのブラウザーを使用する人間のクライアント
- 組織のネイティブ・モバイル・アプリ使用する人間のクライアント
- 財務情報アグリゲーター、販売代理店パートナー、予約パートナーなどのサードパーティによる自動サービス

あらゆるタイプのクライアント向けに設計する場合、組織には多くの作業が必要になります。

消費者のそれぞれのタイプに合わせたエンドポイントを設計する

Credential Stuffing のアタックサーフェスを最小限に抑えるには、必要なアクセスにのみ限定した正しいアクセス権限を適切なクライアントに提供することが必要です。Web サイト利用者のタイプによってニーズは異なります。たとえば、銀行利用者（顧客）は残高や取引明細の表示、金融取引の実行、ユーザープロファイルの更新などを行う必要があります。
ところが、銀行のユーザーが利用する財務情報アグリゲーターに必要なのは、銀行口座の残高情報を確認するためのアクセス権限だけです。利用者のタイプごとにニーズを特定し、それぞれのタイプに適切なレベルのデータアクセスと機能を提供するエンドポイントを個別に作成することは、ボット管理ソリューション以上にセキュリティ上のメリットがあります。

モバイル SDK の制約

モバイル SDK は、ネイティブ・モバイル・アプリを提供する API の保護には有効ですが、このアプローチには制約もあります。まず、アプリケーションのライフサイクルを考えてみましょう。ウェブページの保護機能の有効化に比べ、モバイルアプリのアップグレードにはかなり多くの時間がかかり、ソフトウェアの開発やリソースのテストが必要になります。その上、ボット緩和を有効化する前に、すべてのエンドユーザーのモバイルアプリを最新バージョンにアップグレードする必要もあります。何よりも、モバイル SDK では、サードパーティの自動サービスがアクセスできるよう設計された API を保護することはできません。当然のことながら、自動サービスはボットやその他の自動ツールを活用しているからです。ブラウザーフィンガープリントやふるまい異常分析などの高度なボット検知手法は、人間とボットを区別することはできても、正当なボットと不正なボットを区別することはできません。

ブラウザーフィンガープリントやふるまい異常分析などの高度なボット検知手法は、人間とボットを区別することはできても、正当なボットと不正なボットを区別することはできません。

ハイブリッド URL

ハイブリッド URL は、多様なタイプのクライアントにサービスを提供するよう設計されたエンドポイントであるため、あらゆるボット管理ソリューションによる保護には課題が残ります。関係するクライアントのタイプによっては、ブラウザーベースクライアント向けの JavaScript インジェクションとネイティブ・モバイル・アプリ向けのモバイル SDK を組み合わせて実装されたボット検知を使用して、エンドポイントを保護することは可能かもしれません。たとえば、ブラウザーや組織のネイティブ・モバイル・アプリを介した人間によるトラフィックのみを処理するよう設計されたエンドポイントは、ボット管理ソリューションで完全に保護できます。

ただし、別のケースでは、Web サイトにアクセスせずにアタックサーフェスを完全に最小化するのは不可能です。たとえば、モバイル SDK では、ネイティブ・モバイル・アプリとサードパーティ自動サービスの両方で使用される API を保護することはできません。

Credential Stuffing の緩和 — 中間オプションとリスク

Credential Stuffing は、ボットが高度で、Web サイトアーキテクチャが保護を妨げる可能性があるという 2 つの理由から、複雑な問題であると言えます。組織や Web サイトのニーズによっては、100%効果的なソリューションであり、理論上は可能であっても、受け入れられない可能性もあります。Credential Stuffing 緩和戦略には、セキュリティ上のリスクと変更に伴う組織、Web サイト、およびモバイルアプリへの影響、コスト、期間とのバランスを取ることが必要になる場合もあります。

オプション 1：Web サイトアーキテクチャを多様なタイプのクライアントに合わせる

アタックサーフェスを可能な限り最小限に抑えることが目標であるなら、理想的なソリューションにするために、Web サイトアーキテクチャをさまざまなタイプのクライアントに合わせる必要があります。一部の組織では、さまざまなレベルですでに実施しています。
ただ、その他の組織では、それを実現するために、Web サイトの再構築が必要になる場合があります。既存のエンドポイントを個別の URL に分けることで、トランザクション URL トラフィックを最も細かく制御できるため、アタックサーフェスを縮小できます。
たとえば、以下のようにクライアントを URL 単位で分割できます。

URL 1：デスクトップ、ノートパソコン、モバイルのブラウザーを利用している人間ユーザー

URL 2：ネイティブ・モバイル・アプリ

URL 3：業界アグリゲーターやパートナーなどのサードパーティの自動サービス

このようなアプローチにより、適切なボット検知を URL 1 と URL 2 に適用し、他のタイプの利用者を強制的に URL 3 に割り当てることができます。この場合、ふるまい異常ボット検知を使用して URL 3 を保護することはできませんが、URL 3 のユーザーが利用できるデータやアプリケーション機能を制御することはできます。

メリット	デメリット
リスクを最小限に抑え、アタックサーフェスを最小限に縮小できる	プロジェクトの規模が大きすぎて、Web サイトや組織の多くの部分に手を入れることになる Web サイトの可用性が最優先課題である場合は、運用の中断が多すぎることがある

オプション 2：ネイティブ・モバイル・アプリをホワイトリストに登録する

ブラウザーベースクライアントとネイティブ・モバイル・アプリの両方にサービスを提供するエンドポイントは、JavaScript インジェクションとモバイル SDK の両方を提供するボット管理ソリューションで保護できます。ただし、一部の組織ではモバイルアプリをベンダーのモバイル SDK で更新することに、少なくとも現時点では積極的に取り組む意思がない場合があります。

モバイルアプリをベンダーの SDK に統合し、ユーザーベース全体のアップグレードを図ることが、かなり大掛かりな作業を伴うことを踏まえれば、JavaScript ベースのインジェクションをブラウザーベースクライアントのエンドポイントに展開し、ヘッダーフィールドの値（たとえば、ユーザーエージェントなど）やヘッダーフィールドの並び順など、リクエストヘッダー内のデータを介して、モバイルアプリを暫定的なソリューションとしてホワイトリストに登録するという方法もあります。

メリット	デメリット
簡単に実施できる 中断が発生しない	モバイルアプリをホワイトリストに登録したことが攻撃者に見つかってしまうまでしか効果がない アプリをスプーフィングして、防御を回避することは攻撃者にとって比較的簡単である

オプション 3：既知のサードパーティの自動サービスをホワイトリストに登録する

クライアントトラフィック（ブラウザーベースまたは API ベース）および限定的で既知のサードパーティによる自動サービスがあり、すべて同じエンドポイントにアクセスしている場合は、サードパーティのサービスをホワイトリストに登録するという選択肢もあります。たとえば、銀行では Intuit Mint などのよく知られた財務情報アグリゲーターの IP アドレスをホワイトリストに登録できます。これらのサービスは、通常、よく知られている IP アドレス空間から運用されているため、攻撃者がスプーフィングするのは困難です。これにより、明示的にホワイトリストに登録されていないサードパーティの自動サービスは、ボットとして処理されます。

メリット	デメリット
簡単に実施できる IP アドレスは HTTP／HTTPS プロトコルを使用したスプーフィングが比較的難しい	サードパーティを個別にホワイトリストに登録する必要がある ホワイトリストに登録されたサードパーティは、ユーザーとして引き続き同じデータにフルアクセスできる すべてのサードパーティサービスが知られていて、数を限定した場合のみ適している

次のステップ：エンドポイントのアーキテクチャ、リスク、オプションを理解する

組織でクライアントのタイプ（ウェブブラウザー、ネイティブ・モバイル・アプリ、サードパーティの自動サービス）ごとに専用の URL を使用していない場合は、ウェブアーキテクチャを見直すことが不可欠です。

最初のステップは、現在のアタックサーフェス、リスクのレベル、オプションを理解することです。Akamai のクラス最高レベルの Credential Stuffing セキュリティエキスパートは、包括的な分析を行い、Credential Stuffing 緩和戦略のプレイブックを作成できます。その結果、Web サイトアーキテクチャについて、および、それが次のようなボット検知ソリューションの実装にどのような影響を及ぼすのかを理解できるようになります。

1. Credential Stuffing の防御に必要なトランザクション URL およびリクエストを URL に送信するすべての HTML フォーム・エントリー・ポイントを特定する
2. 利用者タイプごとに URL を分類する
3. ハイブリッド URL を特定し、URL クライアントの再構築を推奨する
4. 既知の技術的な複雑化要因を特定して評価する
5. ターゲットとポリシー構造を一致させることで、URL クライアントのタイプ別に適切な防御戦略を策定する
6. Credential Stuffing アタックサーフェスのプレイブックを作成して、ボット検知実装計画への次のステップを特定する