Understanding Your Credential Stuffing Attack Surface

Akamai

Introduction

For any organization at risk for credential stuffing attacks, its ability to effectively mitigate these attacks will depend on more than the bot management vendor or solution selected. How your website is architected will play a critical role in the effectiveness of any security solution.

To understand why, consider how these attacks work, and how security solutions protect against them. Credential stuffing attackers use botnets to automate the validation of stolen credentials against your application login. To separate automated bots from legitimate human users, today's advanced bot detection technologies use JavaScript injection when protecting web pages and a mobile software development kit (SDK) when protecting APIs used by native mobile apps. Depending on how your website is architected and the types of clients that interact with it, your ability to minimize your attack surface could be limited.

In this white paper, we explain what's behind the architectural challenge to employing today's bot management solutions effectively; the ideal website architecture to mitigate credential stuffing attacks successfully; and specific intermediate options to reduce your attack surface — along with the risks and limitations of each option.

How Bot Detection Works

Detecting bots requires identifying whether the sender of a request is a human or not. Simple bot detection techniques examine the content of a request, such as by looking at the header. However, this approach is not reliable against more sophisticated bots, because request headers are easily spoofed. More advanced bot detection techniques use JavaScript challenges, browser fingerprinting, and behavior anomaly analysis.

Advanced bot detection techniques use JavaScript challenges, browser fingerprinting, and behavior anomaly analysis.

Injecting JavaScript in a browser session

Most bot detection techniques inject a snippet of JavaScript code into the HTML of a web page delivered to a client's browser. The JavaScript code snippet is executed when the browser loads the HTML and can employ a variety of techniques to distinguish humans from bots.

The least sophisticated technique is a JavaScript challenge, which checks to see if the request is coming from a browser that can execute JavaScript. A JavaScript challenge can be useful for detecting simple bots that cannot execute JavaScript.

A more sophisticated technique is browser fingerprinting, which collects a variety of characteristics about the browser, such as the browser type and version, screen resolution, installed plugins, and installed fonts. A bot management solution can inspect the browser fingerprint to identify anomalies that indicate that the client is a bot attempting to spoof a legitimate browser.

The most sophisticated technique available today is behavioral anomaly analysis, in which behavioral data is collected from the client's input devices — keyboard strokes and mouse movements on a computer, or touch events, gyroscope readings, and accelerometer readings on a mobile device. This data is returned to the bot management solution, which analyzes the behavioral patterns for deviations that indicate a bot.

Mobile SDK for apps

JavaScript injection-based techniques work on browser-based clients, including desktop and mobile browsers. However, most API-based clients, such as native mobile apps and other automated services, cannot execute JavaScript. As a result, JavaScript injection techniques will not get a response from these clients. Without the proper response, the bot management solution will assume the client is a bot and take action against it.

To overcome this situation, bot management vendors will often provide a mobile SDK to integrate its bot detection with a native mobile app. With a mobile SDK, mobile apps can collect the required data within the app and transmit it to the bot management solution for analysis.

Understanding Your Website Architecture

Modern websites are complex, sprawling properties that can comprise hundreds or thousands of web pages and support multiple different types of clients and traffic. In addition, website ownership is often dispersed across an organization. Understanding your website architecture and how clients flow from different pages to your login endpoints is essential to successfully mitigating credential stuffing attacks — and understanding the level of risk you face.

What is an endpoint?

An endpoint is a discrete URL that can be accessed by a client. Mitigating credential stuffing attacks requires identifying and protecting the transactional URLs that validate user credentials. For example, the URL may be used to log in to an existing account or to create a new account. Beyond credential stuffing, endpoints that may need to be protected may include URLs that check gift card balances, search for flights, or add products to a shopping cart.

Identifying every endpoint

Many organizations operate websites with multiple endpoints that can be susceptible to credential stuffing. For example, a financial services institution may have consumer, small business, and employer services — each with different login endpoints. In addition, each of these lines of business may have separate account signup endpoints.

Your organization may have secondary endpoints that are unknown to all but a few IT or line-of-business personnel. There are often legacy apps with endpoints that few know about, but a persistent attacker can find them. These may include endpoints that service an API, a retail kiosk, or a customer service chatbot.

The first step in planning any bot management implementation is taking inventory of everything that needs to be protected.

Identifying what needs to be protected

The first step in planning any bot management implementation is taking inventory of everything that needs to be protected. However, this is often a difficult task due to the size and complexity of modern websites. For example:

- People often think about the page and not necessarily the endpoint in terms of what needs to be protected.
- A single login endpoint is often used by multiple pages. For example, users can often enter login credentials from any page on an online retail site. However, the credentials are all sent to the same login endpoint for validation.
- The team responsible for maintaining an endpoint is often not the one creating the pages that lead to it. For example, native mobile apps are often developed by a different team that does not coordinate with the team developing web pages for desktop users.
- Securing any given endpoint requires an understanding of all the ways in which a client can get to it and deploying protection on those pages. Without close coordination between all parties, it is easy to end up with ways to get to the endpoint that are unprotected.

Architecting Protection for Multiple Types of Clients

Websites may interact with multiple types of clients, depending on the needs of the organization. These can include:

- Human clients using desktop, laptop, or mobile browsers
- Human clients using an organization's native mobile apps
- Automated third-party services, such as financial aggregators, reseller partners, and booking partners

Architecting for all types of clients may require substantial work by your organization.

Design Endpoints for Each Type of Consumer

Minimizing the attack surface for credential stuffing requires providing the right access to the right clients — and no more than is necessary. Different types of website consumers have different needs. For example, banking customers need to view balances and statements, perform financial transactions, and update their user profiles. A financial aggregator employed by that user, however, only needs access to check account balances. Identifying the different needs of different types of consumers — and creating different endpoints that offer appropriate levels of data access and functionality for each — will provide security benefits beyond any bot management solution.

Limitations of a Mobile SDK

While a mobile SDK will help your organization to protect the APIs serving your native mobile apps, there are limitations to this approach. First, consider your application lifecycle. Compared to enabling protections for web pages, upgrading a mobile app will take significantly more time and require the engagement of software development and testing resources. What's more, your organization will need to upgrade all end users to the latest version of the mobile app before enabling bot mitigation. Most importantly, a mobile SDK cannot help protect APIs designed to be accessed by automated third-party services. By definition, these services leverage bots and other automated tools. Advanced bot detection techniques such as browser fingerprinting and behavior anomaly analysis can only help distinguish between humans and bots, and not between legitimate and illegitimate bots.

Advanced bot detection techniques such as browser fingerprinting and behavior anomaly analysis can only help distinguish between humans and bots, and not between legitimate and illegitimate bots.

Hybrid URLs

Hybrid URLs are endpoints designed to serve multiple types of clients and are therefore challenging for any bot management solution to protect. Depending on the types of clients involved, it may be possible to protect an endpoint using bot detection implemented through a combination of JavaScript injection for browser-based clients and a mobile SDK for native mobile apps. For example, an endpoint that is designed to serve only human traffic through browsers and an organization's native mobile apps can be fully protected by a bot management solution.

In other cases, however, it may be impossible for an organization to fully minimize the available attack surface without rearchitecting its website. For example, a mobile SDK does not help protect an API used by both native mobile apps and automated third-party services.

Mitigating Credential Stuffing — Intermediate Options and the Risk

Credential stuffing is a complex problem, both because bots are sophisticated and because website architecture may hinder protection. Depending on the needs of your organization and your website, a 100% effective solution, while theoretically possible, may not be palatable. Your credential stuffing mitigation strategy may require the balancing of security risk against the impact, cost, and time frame of changes to your organization, websites, and mobile apps.

Option 1: Align Website Architecture with Different Types of Clients

If achieving the smallest possible attack surface is your goal, then the ideal solution has to align the website architecture with different types of clients. Some organizations have already done this to varying degrees. Others may need to rearchitect their website in order to do so. Breaking existing endpoints into separate URLs helps reduce the attack surface by providing the most granular control of transactional URL traffic. For example, you might segment your clients by URL as follows:

URL 1: Humans on desktop, laptop, and mobile browsers

URL 2: Native mobile apps

URL 3: Automated third-party services, such as industry aggregators and partners

With this approach, you will be able to apply the appropriate bot detection to URL 1 and URL 2, and force other types of consumers to URL 3. While you can't use behavioral anomaly bot detection to protect URL 3, you can control the data and application features available to its users.

Pros	Cons
Lowest risk and smallest attack surface	May be too big of a project, touching too many parts of the website or organization May be too disruptive to operations when website availability is a top goal

Option 2: Whitelist Native Mobile App

Endpoints serving both browser-based clients and native mobile apps can be protected by a bot management solution that provides both JavaScript injection and a mobile SDK. However, some organizations may be unwilling to update their mobile apps with a vendor's mobile SDK — at least not right away.

Given that integrating your mobile apps with a vendor's SDK and upgrading your entire user base may be a significant undertaking, you could deploy JavaScript-based injection on the endpoint for browser-based clients, and choose to whitelist your mobile apps via something in the request header — such as header field values (e.g., user agent) or header field order — as a stopgap solution.

Pros	Cons
Easy to do Non-disruptive	Only works until an attacker figures out that you've whitelisted your mobile app It's relatively trivial for an attacker to spoof your apps and bypass protections

Option 3: Whitelist Known Automated Third Parties

If you have client traffic (either browser- or API-based) and a limited and known set of automated third-party services — all accessing the same endpoint — you may choose to whitelist the automated third-party services. For example, a bank could choose to whitelist the IP addresses for a popular financial aggregator such as Intuit Mint. These services typically operate from a well-known IP address space, making it difficult for an attacker to spoof. Any automated third parties that are not explicitly whitelisted will be treated as bots.

Pros	Cons
Easy to do IP addresses are relatively hard to spoof using HTTP/HTTPS protocols	Each third party must be whitelisted individually Whitelisted third parties will still have full access to the same data and functionality as a user Only feasible if all third-party services are known and limited in number

Next Step: Understand Your Endpoint Architecture, Risks, and Options

If your organization does not use exclusive URLs for each type of client (web browsers, native mobile apps, and automated third-party services), then it is essential to review your web architecture.

The first step is to understand your current attack surface, the level of risk you face, and your options. Akamai's best-in-class credential stuffing security experts can perform a comprehensive analysis and develop a playbook for your credential stuffing mitigation strategy. As a result, you will understand your website architecture and how it will impact the implementation of any bot detection solution, including:

1. Identifying transactional URLs that need protection from credential stuffing, and all HTML form entry points that send requests to them.
2. Categorizing those URLs by consumer type.
3. Identifying hybrid URLs and recommending URL client restructuring.
4. Identifying and assessing known technical complications.
5. Matching target and policy structure to determine the appropriate protection strategy, by URL client type.
6. Developing a credential stuffing attack surface playbook and identifying the next steps to a bot detection implementation plan.