Engineer

運用シークレット

プログラムの動作に必要な鍵やパスワードなどの認証情報

どのようなものか

運用シークレットとは、ソフトウェアやその運用者がシステムに対して操作するための値であり、それを読んだ他の人にも同じ権限を与えるものです。データベースの接続パスワード、APIトークン、SSH鍵ペアの秘密鍵、バックアップを暗号化するパスフレーズは、いずれも運用シークレットです。普段は単にシークレットと呼び、その生成、保存、更新、破棄を行うことをシークレット管理といいます。

周囲のコードと異なる点は二つあります。一つは、誰が読むかです。利用者のパスワードは人が覚えていて入力を求められたときに打ち込みますが、運用シークレットはプログラム、デプロイスクリプト、定期実行ジョブが読み取り、人が覚えておく必要はありません。そのため、OWASP Secrets Management Cheat Sheetは、シークレットに独自のライフサイクルを設けています。必要最小限の権限で生成し、定期的に更新し、漏えいした可能性があれば失効させ、プラットフォームが対応する場合は有効期限を設定します。有効期間は数分から数年までさまざまです。利用者のパスワードは、この定期更新の対象外です。もう一つは、シークレットと、その所在を示す設定との境界です。ホスト名やバックアップリポジトリのパスは、何かがある場所を示します。シークレットは、保持者のアクセスを可能にする値であり、その複製の数を把握する必要があります。

シークレットには入れ子の関係もあります。保存先のストアにも独自のシークレットが必要です。このチートシートは、クラウド事業者のルート認証情報など、その最上位のシークレットは、通常、別の場所に保管しなければならないと指摘しています。

概念を判断に生かす

「運用シークレットとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 アカウント、パスワード、権限を整理し、業務に必要なアクセスだけを付与し、退職者の権限は当日に解除します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Linuxサーバーを守る1,000件のRBACルールを作成 — 1,000 セキュリティ上の穴を塞ぐアクセスルール

Linuxサーバーを守る1,000件のRBACルールを作成 お問い合わせ →

なぜ重要なのか

シークレットは、それを提示した人に対して機能します。トークン自体には、本来誰が持つべきものだったかという記録がないため、複製が増えるたびに、それを使える人やプログラムの範囲が広がります。ソースリポジトリは、その値が入り込む場所として最も不適切です。クローンには全履歴が含まれるため、一度コミットされた値は、それを取得するすべてのノートパソコン、ビルドランナー、ミラーへ渡り、該当行を削除しても履歴に残ります。

標準は、これを要件として定めています。OWASP Application Security Verification Standardは、バックエンドのシークレットを生成、保存し、アクセスを制御し、破棄するために、キーストアなどのシークレット管理ソリューションを求めています。また、アプリケーションのソースコードやビルド成果物にシークレットを含めてはならないとしています。アクセスには最小権限を求め、最も高い検証レベルでは、有効期限と更新も要求しています。NIST SP 800-53には、管理策の拡張要件IA-5(7)があり、暗号化されていない固定の認証子を、アプリケーションやその他の静的な保存領域に埋め込んではならないとしています。その指針は抜け道も塞いでいます。保存されている値をそのまま認証に使うのであれば、どのファイルに入っていても、暗号化されていないものと見なされます。

トレードオフもあります。集中管理ストアを置けば、最も必要なときに稼働していなければならないシステムが一つ増えます。チートシートも、インシデント時には新しい認証情報がすぐに求められると指摘しています。複数のサービスで一つのシークレットを共有すれば手間は減りますが、漏えい時にどの保持者から流出したか分からなくなります。手作業の負担はシークレットが増えるたびに大きくなるため、チートシートは更新の自動化を推奨しています。また、アプリケーションを再起動しても、盗まれた認証情報は有効なままだと注意しています。使えなくするには、失効させる必要があります。

実際の活用例

このサイトは、安全な業務の進め方を説明するページに、シークレットをリポジトリに入れないという原則を掲げています。認証情報はシークレットストアやデプロイ層に置き、ソースにもビルドの生成物にも含めません。コミットのたびに自動スキャンを実行し、検出があればコミットを停止します。

リポジトリの技術ノートには、その方法が記されています。スキャナーはgitleaksで、コミットのたびに作業ツリーを調べ、完全な品質検査でも再度実行します。検出は必ず処理を止めるものであり、許可リストに加えて回避することは認められていません。依存関係のフォルダーや生成出力のツリーは、毎回作り直される複製なので除外します。ノートには、より気付きにくい失敗も記されています。スキャンの途中でファイルが消えると、gitleaksはエラー行を出力しながら、正常終了します。そのため、実行タスクはエラー行が一つでもあれば失敗とします。サイト自体は静的で、アプリケーションのシークレットを持たず、所有者が保管する鍵を使ってSSH経由でデプロイします。

シークレットをリポジトリの外に置くと、その検査からも見えなくなります。シークレット用の事前検証プレイは、この隙間に対処するものです。シークレットをホスト別に分けた直後、コードは正しかったものの、運用者のマシン上にあり、品質ゲートからは見えない暗号化Vaultには、以前の単一ホスト用の値が残っていました。最初の収束処理は稼働中のホストに接続し、8タスクを実行した後、9番目で処理を拒否しました。その時点で、本番のスワップ設定はすでに書き換わっていました。対策として、どのホストにも接続せず、factsも収集せず、本番の収束処理と同じ二つのタスクファイルをVaultに対して実行するプレイを作りました。約1秒で終わり、プロビジョニングが他の作業を始める前に実行します。ルールを別に書き直すのではなくコードを再利用することで、検査と収束処理が食い違うのを防ぎます。現在、シークレットに不備があれば、ノートパソコンでは2秒で処理が拒否されます。

その分割とは、すべての運用シークレットをホストごとの値にした変更です。プラットフォームは1ホスト構成から2ホスト構成に移るところで、バックアップリポジトリ、パスフレーズ、アラートのスイッチなど、いくつかのシークレットが単一の値として書かれていました。それぞれをインベントリーのホスト名をキーとするマップに変更し、その理由を二つ、設定のそばに記録しました。2ホストでデッドマンズスイッチを共有すると、どちらかが生存通知を送り続ける限り正常表示のままなので、2台目のホストが1台目の停止を隠してしまいます。別々のresticリポジトリは、同じパスフレーズを使っていても、異なるマスター鍵を持つことができます。しかし、そのパスフレーズを入手し、両方のリポジトリの暗号化データと鍵ファイルを読み取れる人は、両方を復号できます。そのため、パスフレーズを共有すると、ホストごとの分離にはならず、両方がまとめて解読

され得る状態になります。既存のホストは四つのマップすべてに記載されていますが、リポジトリと二つのスイッチは空欄のままです。これは、仕組みは用意できていても、値の設定はまだ運用者が行う必要があることを示しています。

問題が起きるところ

シークレットがコミットに入り込みます。 コミット時に設定ファイルやテスト用データに値が入っていたため、スキャナーが差分で検出したり、プッシュされたファイル内のトークンに誰かが気付いたりします。OWASPのチートシートは、対処の順序を示しています。まず漏えいした値を失効させ、次に新しい値へ更新し、その後で初めてコードやログから取り除きます。コミットを除くために履歴を書き換えると、それ以降のすべてのコミットが変わり、それらへのリンクも切れるため、計画が必要です。検出はコミット前に、pre-commitフックやエディターで行います。その段階なら、検出による手戻りはありません。

スキャンが、読んでいないファイルを合格扱いにします。 途中でファイルが消えた、除外範囲が広すぎた、シークレットに判別できる形式がなかったなどの理由で、ツリーの一部を調べていないのに実行が成功します。パターンルールは固定形式のトークンを検出できますが、人が考えたパスワードは見逃します。そのため、チートシートは複数のツールの使用を勧めています。スキャナーのエラーはすべて失敗として扱い、除外するのはビルドで作り直される複製に限ります。

一つのシークレットを多くの保持者が使います。 2ホスト、2サービス、あるいはチーム全体が同じ値を使います。漏えいしても、ログではどの複製から流出したか分からず、交換するには、すべての保持者に対して同時に新しい値を反映しなければなりません。SP 800-53は、グループやロールのアカウントの認証子を、メンバーが変わるたびに変更するよう求めています。保持者ごとにシークレットを分ければ、一斉の調整ではなく、一つの値の交換で済みます。

シークレットがビルドとともに配布されたり、ログに出たりします。 コンテナーイメージ、バンドルされたスクリプト、デバッグ出力によって、値が本来置かれるべき場所の外へ運ばれます。ASVSは、ソースコードと並んでビルド成果物も対象として明記しています。チートシートは、シークレットをログに出さず、どうしてもログ行で参照する必要がある場合はマスクするよう求めています。リポジトリだけでなく、ビルドの生成物もスキャンします。

参考資料

- OWASP Secrets Management Cheat Sheet — シークレットのライフサイクル、コミット前の検出、漏えい後の対処順序を説明しています。

- OWASP Application Security Verification Standard — グループ13.3にあるシークレット管理の要件です。Vault、最小権限、更新を扱っています。
- NIST SP 800-53 Rev. 5 — IA-5の認証子管理と、埋め込まれた固定の認証子を禁じるIA-5(7)を説明しています。
- Gitleaks — このサイトがコミットのたびに作業ツリーを調べるために使うスキャナーです。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「運用シークレットとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く 連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/secret/