GitHub

セキュアなソフトウェア開発のためのガイドブック

目次

はじめに

企業と顧客の間の関係、およびその相互関係の基盤にあるものは信頼です。企業が機密データを悪者の手に渡さないようにできるかどうかが、この信頼の要です。世界各国の政府は、さまざまな規制の枠組みによって対応しており、顧客データを保護することを企業に義務付け、違反すると厳しい制裁を課しています。データを適切に保護できなければ、企業の評判やビジネスに深刻な影響が長期的に残る可能性があります。

さらに複雑なことに、ソフトウェアは既に企業のビジネスの中心的な要素となっており、その重要性は高まり続けています。しかし、厳格な規制や法令遵守要件が、従うべきプロセスから使用するツールまで、企業のソフトウェア構築方法を統制しています。これらの要件は、パフォーマンスの低下やイノベーションの阻害を招くほど厳しいことが少なくありません。

一方で、セキュアな開発手法はコラボレーションやイノベーションを必ずしも妨げるわけではありません。何千もの組織が GitHub を活用して鎖国状態の開発からワークフローを解放し、安全なプロセスを構築して、最高の仕事ができる柔軟性をエンジニアリングチームに提供しています。

消費者の期待がこれまでにないほど高まり、コスト削減の圧力が大きくなる中、効率的かつ協力的で安全なワークフローがあれば、チームは顧客に適した革新的なソフトウェアを構築するという最重要事項に集中することができます。

この記事では、第一部と第二部にわけて、企業が直面する独自の規制上および技術上の課題とその対処方法、そして GitHub がどのようにサポートできるかについて概説します。

顧客データの保護

多くの企業は機密性の高いデータを扱っており、犯罪目的でこのデータを手に入れようとする者たちと激しい闘いを続けています。闘いの相手は、個人のハッカーから、国の支援を受けた非常に高い能力とリソースを有する者まで、セキュリティを危険にさらそうとするあらゆる者たちです。幸いなことに、企業が開発プロセスを保護し、機密データを安全に保つのに有効なベストプラクティスとツールがあります。

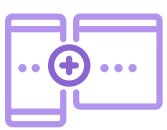
1つのソリューションやアプローチでセキュリティを保証することはできません。効果的なセキュリティは、階層化アプローチや、データパスとワークフローを複数のポイントで強制的にチェックおよび保護することによって実現します。この階層化アプローチには、ツール、プラクティス、そしてセキュリティを重視する文化が必要です。

セキュリティ文化を築く

セキュリティポリシーを全社規模で実施するための鍵となるのが、セキュリティをすべての中心に据える企業文化です。セキュリティ文化を創り出すということは、セキュリティ保護が全員の仕事になるようなプロセスを構築するということです。たとえば、安全なプラクティスに関するトレーニングを定期的に行い、フィッシング攻撃のような侵入に対するソーシャルパスの危険性だけでなく、技術的な手段にも焦点を当てます。

具体的なベストプラクティスの例：

複雑なパスワード、定期的なパスワード変更、パスワードマネージャーやパスワードジェネレーターの使用など、パスワードセキュリティを強化するパスワードポリシーを施行する



2 要素認証を使用する

ノート PC、タブレット、携帯電話といったデバイスの物理的な保護だけでなく、パブリックネットワークの危険に関する教育とトレーニングを行う

VPN を使用し、社内ネットワークへのアクセスを保護する

常にデータを暗号化する

承認されていないアクセスを回避する最良の方法の 1 つは暗号化です。 悪意を持つ者が伝送中のデータにアクセスしたとしても、強力な暗号化により、そのアクセスはほとんど、またはまったく価値のないものになります。繰り返しになりますが、暗号化戦略を導入する場合、階層化アプローチを採用するのが最も効果的です。
これは、以下を意味します。

永続性記憶装置とエンドユーザーの間でデータをやり取りする際は、伝送中のデータを暗号化する

データベースに保存されているデータを暗号化する

伝送時の暗号化

標準的なプラクティスとして、あらゆる Web ベースの作業では、TLS などの最新の暗号化プロトコルを使用したエンドツーエンドの暗号化を採用する必要があります。TLS は、実質的にまず解読できない強力な証明書ベースの暗号化を使用して、安全な通信を支援します。また、TLS は、ネットワークトランザクション内の各当事者を検証し、悪意を持つ者がトランザクション内で信頼できるパートナーになりすます「中間者」攻撃を防ぎます。Let's Encrypt などのサービスは、効果的な TLS の実装や必要な証明書の取得にかかるコストと複雑な作業を大幅に削減しています。

保存時の暗号化

一部のアプリケーションプロバイダーは、ソフトウェア層で暗号化サービスを提供しています。この種の暗号化はやらないよりましですが、複雑さとパフォーマンス低下の観点から見て相当な負荷となります。ファイルシステムの暗号化やハードウェアアプライアンスレベルの暗号化など、可能な限りストレージハードウェアに近いところで実行される暗号化ソリューションは、**すべて**のアプリケーションに透過的な暗号化レイヤーを提供します。

GITHUB 使用時の暗号化

Let's Encrypt を利用することで手頃な費用で簡単に証明書を管理することができます。Let's Encrypt を使用すると、GitHub Enterprise とのすべての通信が TLS で保護されます。また、GitHub は、開発者が使用する Git クライアントとサーバー間の通信も、SSH を使用して保護します。いずれも強力な暗号プロトコルであり、不正な傍受や機密データの漏洩に対する堅牢な壁となります。

強力な認証管理プラクティスを採用する

ユーザーが覚えなければならないパスワードが増えれば増えるほど、パスワードを紙に書いたり、覚えやすい (それゆえクラッキングしやすい) パスワードを使い回したりするなど、安全ではないプラクティスに流れがちになります。LDAP のような ID の一元管理システムをシングルサインオン (SSO) ソリューションと一緒に使用することで、ユーザーが管理しなければならないパスワードの数が大幅に減り、正しいパスワード管理を行えます。

GITHUB と認証管理システムを併用する

GitHub Enterprise は、Active Directory や LDAP など、さまざまな認証管理ソリューションを使用した認証と承認をサポートします。また、GitHub は SAML もサポートしているため、企業はユーザーにシングルサインオンを使わせることができます。

データベースへの攻撃を防ぐ

組織に入ってくるデータは、悪意を持つ者がシステムにアクセスするために利用した、危険なものである場合があります。攻撃の多くは、適切に設計されていないシステムに危険なペイロードを実行し、欠陥を悪用してシステムを乗っ取ります。SQL インジェクションはその典型例です。悪意を持つ者が、Web アプリケーションの顧客名など、無害に見えるデータに SQL コードを意図的に追加すると発生します。基盤となるソフトウェアに欠陥があると、このコードが任意に実行され、意図せずにユーザーへデータが返されます。このような攻撃はセキュリティを監督する IT 管理者が気づきにくいため特に危険です。

ここでもまた、検証されたデータのみを社内環境に入れるようにするには階層化アプローチが最も効果的です。これは、ユーザーインターフェイス内のコードから始まり、ミドルウェアを介して基盤となるデータストアまで、データパス全体が対象です。各ステップでコードに検証を組み込むことで、こうした攻撃に対する脆弱性は大幅に軽減されます。この取り組みをサポートするうえで非常に重要なのは、手作業での頻繁なレビューと、自動化されたレビューおよびテストを組み合わせることです。

一般的な脆弱性を回避する 3 つの方法：

- 検証されたデータのみを運用システムに受け入れる
- システムのエラーやエッジケースを監視する
- 認証管理システムを導入し、可能な限りデータシステムレコードへのアクセスを制限する

機密データを隔離する

機密データが存在する場所が少なければ少ないほど、攻撃者が悪用できる機会 (接触できる「表面積」) が減少します。この「表面積」を減らすには、個人に任せず、パスワードマネージャーで強力なパスワードを管理する、パスワード、アクセス /API トークン、暗号鍵などの機密データをだれでもアクセスできるリポジトリに決してコミットしない、などのプラクティスを実施します。

絶対に、機密データを非運用システムで使用したり、保存したりしないでください。もちろん、チームがテストのために実際の環境を模した大量のデータを必要とし、エンジニアが開発およびテスト環境を運用データで「リフレッシュ」したいと考える可能性はあります。そのような場合は、運用データを使用する代わりに、大量のダミーデータを設定および生成できるツールの活用を検討してください。こうすることで、機密情報を危険にさらすことなく、信頼性の高い現実的なテストを実施できます。

PRE-RECEIVE フックを使って、機密データがリポジトリに保存されないようにする

機密データを隔離して保存することに加え、pre-receive フックを使用して、保護されているデータが不用意に (または故意に) リポジトリへコミットされないようにします。pre-receive フックは、GitHub アプライアンス上の隔離された環境で実行するシンプルなスクリプトです。これらのスクリプトは、コードが GitHub にプッシュされる**前に**トリガーされ、コミットの検証、機密情報の特定、リポジトリへの追加の回避を行います。

コードを検証した後、pre-receive フックは成功または失敗の 2 つの値のいずれかのみを返します。失敗した場合、コミットが失敗したことを伝えるメッセージが、チームが選んだその他の役立つ情報と一緒に表示されます。pre-receive フックが成功した場合にのみ、コードはリポジトリにコミットされます。その結果、望ましくないコードがリポジトリにコミットされることがなくなり、会社は侵害、責任に対する脅威、規制当局からの処罰を避けることができます。

ソースコードの保護を強制する

効果的なセキュリティでは、機密情報へのアクセス、およびそれを管理するソフトウェアのソースコードへのアクセスを制御する必要があります。詳細に制御すれば、アクセスを徹底的に拒否するというセキュリティで固められた環境を構築しなくても、こうした情報を効果的に保護できます。GitHub は、保護されたブランチでアクセスポリシーに対応できます。**保護されたブランチ**では、管理者が保護することにしたブランチ上の Git 機能をいくつか制限することにより、コードの整合性を維持します。たとえば、管理者はブランチにポストできるユーザーを、特定のユーザーやチームに制限できます。また、特定のブランチで、コードを変更または削除する可能性があるフォースプッシュを無効にすることも可能です。

その他のセキュリティ対策：

ブランチを削除できないようにする

1 名以上の担当者による、監査可能な手作業でのレビューを必須にする

自動テストや継続的インテグレーション (CI) チェックで失敗したコードがマージされないようにする

コードベースのあらゆる部分に「コード所有者」を指定し、彼らによるレビューを必須にする

顧客の最も重要な情報を保護するために取るべきステップ：

- すべての場所で安全な接続を使用し、データの伝送には SSH や SFTP などのプロトコルのみを使用する
- パスワード、アクセス /API トークン、暗号鍵などを、だれでもアクセスできるリポジトリに決してコミットしない
- 保護されたブランチと pre-receive フックを使ってヒューマンエラーに対処し、機密データを運用環境から隔離して保護する
- 認証管理ソリューションを利用してアクセスを制限し、適切なユーザーだけが機密データにアクセスできるようにする

開発プロセス全体を保護する

セキュアなソフトウェア開発を行うにあたり、ソフトウェアのコードとその開発プロセスは重要です。多くの開発者が協業し、ソースコードに対するさまざまな変更が実施される環境において、GitHub が提供するセキュリティ機能とベストプラクティスを組み合わせてることで、よりセキュアなソフトウェア開発を行えるようになります。

レガシーなシステムのセキュリティも注意する

メインフレームなどのツールは、初期導入されてから 50 年以上にわたり、多くの企業のシステム運営の中心であり続けています。これらのシステムは他にはない機能を提供しますが、安全上の独自の問題ももたらします。企業は、定期的にすべてのレガシーシステムを検査し、システムの更新にかかるコストと、セキュリティ侵害の観点から考えられるリスクのバランスを検討する必要があります。罰金額が数十億ドルになるまで、あるいは世間の信用を失う壊滅的なセキュリティ侵害が問題になるまで、レガシーシステムの入れ替えはコスト効率が悪く見え、システム入れ替えは実施されないかもしれません。

メインフレームシステムの機能拡張を検討してみる

GitHub のパートナーエコシステムには、メインフレームシステムを製造、販売、保守するベンダーが多数参加しています。例えば、IBM は最近、GitHub の土台となる Git の拡張を発表しました。この拡張によって、メインフレームを使った開発ワークフローにおいても、モダンな DevSecOps のプラクティスを行うことができるようになります。

プロセスを自動化してバグやエラーを防ぐ

依存関係の自動管理を
導入した組織では、
リリースしたソフトウェアの
セキュリティの脆弱性が、
導入前より **60 パーセント**
減少しました。

手動でのセキュリティチェックは、効果的な階層化セキュリティ戦略の重要な要素です。ただし、人に依存し過ぎることも危険です。人は疲労し、注意力が散漫になります。また、単にミスを犯します。大切なのは適材適所です。繰り返しで単調であるものの重要なタスクは、機械に任せましょう。GitHub ワークフローでは、完全に自動化された **継続的インテグレーションと継続的デリバリー (CI/CD)** も活用できます。

CI/CD ツールは、コミットがリポジトリにプッシュされるたびに、コードをテストおよび評価します。GitHub で CI/CD を活用すると、新規コードを既存の運用コードと一緒にテストし、提案された変更が意図したとおりに機能することを確かめ、既存のシステムにセキュリティ上の欠陥が生じないようにすることができます。これらのテストを、対象のコードが依存する (依存関係がある) すべてのコードに拡大して適用することも可能です。

自動化の導入は難しくありません。また、目に見える結果がすぐに出ます。2017 年の IEEE の調査によると、依存関係の自動管理を導入した組織では、リリースしたソフトウェアのセキュリティの脆弱性が、導入前より 60 パーセント減少したことが明らかになっています。

CI/CD と GITHUB の統合

CI/CD と GitHub の統合は簡単です。自動テストの結果が対応作業を必要とするものでない限り、開発者の既存ワークフローにも影響しません。自動化によって開発者の煩わしさが減っていけば、さらに広く自動化が適用されるようになり、組織全体でその利点が感じられるようになります。

GitHub は、Jenkins、Travis、CircleCI など、さまざまな CI ツールと統合します。これらのツールは、コードがプッシュされるたびに GitHub リポジトリからコードを自動的にフェッチし、テストを実行して、結果を成功 / 失敗ステータスのどちらかで GitHub に返します。また、各テストのステータスに関する通知も返すため、開発者はコードがどこで失敗したのかがわかります。

CI テストの結果を受け入れてコードがコードベースにマージされるのを回避するか、何も対処せずただ開発者に警告するかは、あなたとチームが決めることができます。CI ステータスが必要な場合、必要なすべての CI ジョブが完了するまでプルリクエストをマージできません。マージしないようにした場合、必要なテストが成功ステータスを返すまでプルリクエストをマージできません。

セキュリティツールとワークフローを統合する

GitHub と統合できるツールは何百とあります。その多くは、コードや顧客データの保護に役立ちます。

- **CI:** Travis CI、CircleCI、AppVeyor などの CI ツールは、GitHub にコードをプッシュすると自動的にコードをビルドおよびテストし、バグが運用環境に展開されないようにします。
- **エラー報告：** Snyk、Sentry、Dependabot などのツールは、チームによる脆弱性の検出、修正、回避に役立ちます。
- **コード品質：** Code Climate や Codeacy は、セキュリティチェックと品質チェックによるレビューを自動化し、ヒューマンエラーを防ぐとともにチームのコードレビュープロセスを効率化します。

GitHub がどのようなツールと連携できるかを確認するには、Marketplace (github.com/marketplace) もしくは github.com/works-with をご覧ください。

レビュープロセスに、マニュアルレビューを必須化する

自動化は重要ですが、マニュアルでのコードレビューは常にセキュリティ戦略の重要な要素です。特定のコードベースをチェックする「目」が増えれば、検出されるエラーや脆弱性も増えます。また、レビューはセキュリティの他にも、組織の知識を共有し、あらゆるスキルレベルの開発者に学びと指導の機会を提供するという重要な役割も果たします。定期的に組織公認のレビューを実施することで、コードベースのセキュリティが強化されるだけでなく、より健全で一貫性のあるものになります。

GitHub は、コードレビューを義務付ける柔軟性の高いフレームワークを提供します。たとえば、以下のような作業が可能です。

変更を保存するリポジトリへの書き込みアクセス権を持つ、1 名以上のユーザーによるレビューを義務付ける

提案された変更について、行レベルで詳細なフィードバックをする権限をレビュアーに与える

1 名以上の**コード所有者**による追加のレビューと承認を義務付ける。コード所有者は、コードベースのセクション、特定の技術、またはその 2 つの組み合わせに対して特定の責任が割り当てられた、任意の組み合わせの個人ユーザーとチームにすることができます

今後のためにすべてのレビューを記録し、監査できるようにする

バグやエラーの回避に役立つ 3 つのステップ

1. セキュリティの自動スキャンを構築してヒューマンエラーを減らし、発見しにくいバグや脆弱性を発見する。
2. ステータスチェックのために自動化をワークフローに導入する
3. 安全なソフトウェアの構築と保守に対し、チームメンバー全員が共通の責任を等しく感じるようにする

ワークフローを監査し、コンプライアンスを遵守する

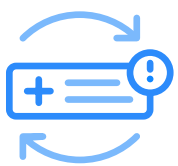
多くの企業では堅牢なログ作成機能と監査機能を必要とします。企業は効果的なセキュリティ戦略を策定するだけでなく、規制機関にそのことを証明する必要があります。GitHub は、コンプライアンス遵守に役立つ柔軟で強力なログ作成、監査、報告フレームワークを提供します。また、コンプライアンスを遵守していることを証明する機能も備わっています。

監査ログ：

GitHub Enterprise は、ユーザーおよびシステム活動の包括的なログを保持します。監査される活動にはすべての git push 操作が含まれ、プッシュを開始したユーザーとその IP アドレス、操作が影響するリポジトリなどが記録されます。ログは、分析、報告と保管、規制要件の確実な遵守のために外部システム (Logstash や Splunk など) へ転送できます。

リポジトリのアーカイブ：

保守しなくなったリポジトリは、アーカイブして「読み取り専用」モードに設定できます。こうすることで、新たなコードがリポジトリにコミットされなくなると同時に、ユーザーは引き続きコンテンツを確認できます。



フォースプッシュのブロック：

Git では、変更をフォースプッシュすることで、開発者はコミット履歴を書き換えることができます。ミスを修正するために必要となるときもありますが、規制によりデータを変更できないようにしなければならない場合、この機能は問題となります。GitHub Enterprise では、管理者は個々のリポジトリまたはアプライアンス全体に対するフォースプッシュをブロックし、コミット履歴を一切書き換えられないようにすることができます。

ユーザーがリポジトリを削除できないようにする：

リポジトリの削除機能は管理者に制限されます。

セキュリティをシフトレフトする

これまで、セキュリティはたいてい後から考えるものでした。実際、インターネットを実行するプロトコルは、セキュリティを**一切**考慮せずに設計されました。後から追加されたセキュリティはないよりましですが、今日の激しい脅威環境ではとても十分とは言えません。効果を上げるには、プロジェクトが開始してすぐに、セキュリティをすべてのソフトウェア開発プロジェクトに組み込む必要があります。セキュリティのシフトレフトとは、タイムラインの右側にあるリリース目前のタイミングから、セキュリティ設計をずっと左側の開始時点に移すことを意味します。こうすることで、製品のライフサイクル全体を通してセキュリティが常に中心となります。

また、早い段階からセキュリティチームに関与させると、変更が難しくコストも高くなる前に、設計、開発、保守に関する決定に影響を与えることができます。開発の早い段階からセキュリティチームを関与させるには、組織の決定権を持つような役職者と、エンジニアチームの合意が必要です。安全な開発には、全員が効果的なセキュリティ文化を受け入れる必要があります。

セキュリティ文化を創り出す

DevSecOps (Development, Security, Operation) は、開発サイクル全体を通してセキュリティを考慮し、チームとロールに分散する方法論です。アプリケーションが出来上がってからセキュリティチームがプロセスに加わっても意味がありません。悪用可能な欠陥が見つかるだけです。セキュリティの専門家が最初から密接に協力していれば、チームは開発しながらセキュリティを積極的にサポートする協力的なプロセスを構築できます。

DevSecOps の原則のために、組織はインフラだけでなくチームの文化の変革が必要になる場合がありますが、ソフトウェアの信頼性の向上と予期しない事態の減少という、それだけの価値のある結果がもたらされます。企業によっては、セキュリティの専門家をスクラムチームに招き、プロセス全体を通して最優先事項にしているところもあります。

ユーザー事例：
Societe Generale

Societe Generale (ソシエテ・ジェネラル) は、フランスに本社を置く、銀行および金融サービスの多国籍企業です。世界 66 か国で 14 万 6,000 人の従業員を雇用しています。同銀行の主要な 3 事業の 1 つであるグローバルバンキングおよび投資家ソリューション (GBIS) 部門は、コーポレートおよびインベストメントバンキング、資産管理、プライベートバンキング、セキュリティサービスを一つにまとめています。

「デジタル変革は当行の戦略的優先事項です。。。私たちは、世界のソフトウェア企業のトップと同じツールやプラクティスを採用することが成功への鍵であると確信しています。」

アミール・ジャバラ氏
Societe Generale、継続的デリバリープラットフォーム担当グローバル責任者

最高情報責任者カルロス・ゴンサルヴェス氏が率いる GBIS は、継続的デリバリーの導入を主な目的として、情報システムの大規模な改革に着手しました。2014 年、情報システム部門は GitHub Enterprise を利用して継続的デリバリープラットフォームを備えました。それ以来、世界中で 3,000 人以上の従業員が利用しています。

「Societe Generale では管理コストが 80 パーセント減少しました。GitHub Enterprise のおかげで、より多くの時間をチェンジ・マネジメントのために割けるようになりました。」

アミール・ジャバラ氏

2016 年末までに GBIS ソフトウェアアプリケーションの 50 パーセントを変えるという戦略目標を達成するため、GBIS の IT 部門は、部門のインテグレーションおよびテストシステムにシームレスに適合する、高速かつ高性能なバージョン管理プラットフォームを必要としていました。また、開発者に高い水準の機能と性能を提供するために、同銀行は一元化されたバージョン管理から、そのようなニーズを満たすのに最適なソリューションである GitHub Enterprise に移行しました。

Societe Generale の経営陣を納得させた、GitHub が最良のソリューションにもたらした 5 つの主要機能：

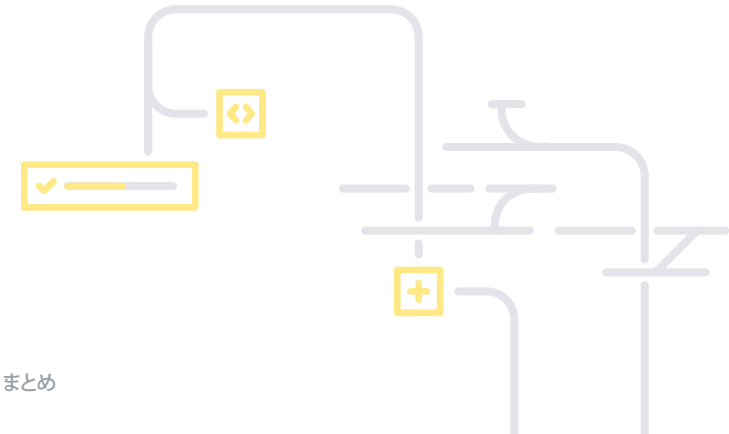
1. **リポジトリ：**開発者は独自のリポジトリを作成してチームで作業できるため、プロジェクトの開発期間を大幅に短縮できます。
2. **コードレビュー：**エラーをなくすために、開発者は同僚が開発したコードをレビューし、変更を提案できます。
3. **コラボレーション：**開発者は、他のプロジェクトで開発されたフィールドテスト済みのソリューションを開発し直す代わりに、組織内の既存のコードを検索して再利用することができます。
4. **ドキュメントのバージョン管理：**プログラマーは、できる限りアプリケーションに合わせたドキュメントを作成し、ホストすることができます。
5. **セキュリティ：**プラットフォーム全体が社内サーバー上でホストされます。GitHub Enterprise では、エンタープライズディレクトリを使用してユーザー認証を管理し、ユーザーおよびシステム活動をすべて記録します。

まとめ

成功を収めるにはデータを安全に保つことが何よりも重要です。ビジネスの生き残りにおいても同様です。効果的なセキュリティの導入は大変なことに見えるかもしれませんが、必ずしもそうとは限りません。階層化した防御の利点の 1 つは、セキュリティを導入するというタスクを、それほど厄介ではない複数のサブプロジェクトに分割できることです。そのために最も重要なことは、まず始めることです。既に始めている場合は、テスト、調査、確認、改善を続けることです。攻撃者が休むことはありません。あなたも休むことはできません。

効果的なセキュリティの導入における課題の 1 つは、ソフトウェア開発者が革新、成長し、生産的になるために必要な自由とのバランスです。セキュリティがゆるすぎると効果がありません。一方、厳し過ぎると、創造性が失われたり、セキュリティへの取り組みに対する他の人たちの意欲を完全に削ぐことになったりする可能性があります。GitHub のようなツールは、組織に適したバランスを見つけるために必要な管理機能を提供します。

結局のところ、わたしたちの目標は、セキュリティに対する認識を単なる脅威から、顧客を満足させ、新たな顧客を惹き付け、ビジネスをさらに差別化する機会に変えることです。優れたセキュリティは顧客の信頼を生みます。また、GitHub のようなパートナーが、効果的なセキュリティ戦略の実現をサポートします。



ポイント：

1. データは常に攻撃を受けています。侵害は、ビジネスに深刻な影響をもたらす場合があります。また、個人への影響も大きくなっています。
2. セキュリティに魔法の薬はありません。複数のツール、プロセス、プラクティスで防御を階層化した方がもし侵害されても修復しやすく、効果的です。
3. 効果的なセキュリティには、効果的なセキュリティ文化が必要です。セキュリティは、あらゆる重要なイニシアティブや意思決定で常に考慮する必要があります。
4. すべてのデータを暗号化しましょう。例外はありません。
5. 多層化アプローチを活用して、システムに入ってくるすべてのデータを精査、検証し、SQL インジェクションのような悪意のある攻撃を回避します。
6. メインフレームなど、老朽化していてもミッションクリティカルなシステムもセキュリティ対策の対象です。罰金、民事責任や信用の喪失に対するレガシーツールのアップグレードコストの費用対効果を考えましょう。
7. 特にソフトウェアのテストなど、可能なところでは自動化を活用します。

わたしたちがお手伝いします。

GitHub は、ご説明したすべての方法に加え、その他の方法でも、堅牢で安全なコードの構築をサポートします。お気軽にご連絡ください。

jp-sales@github.com

GitHub