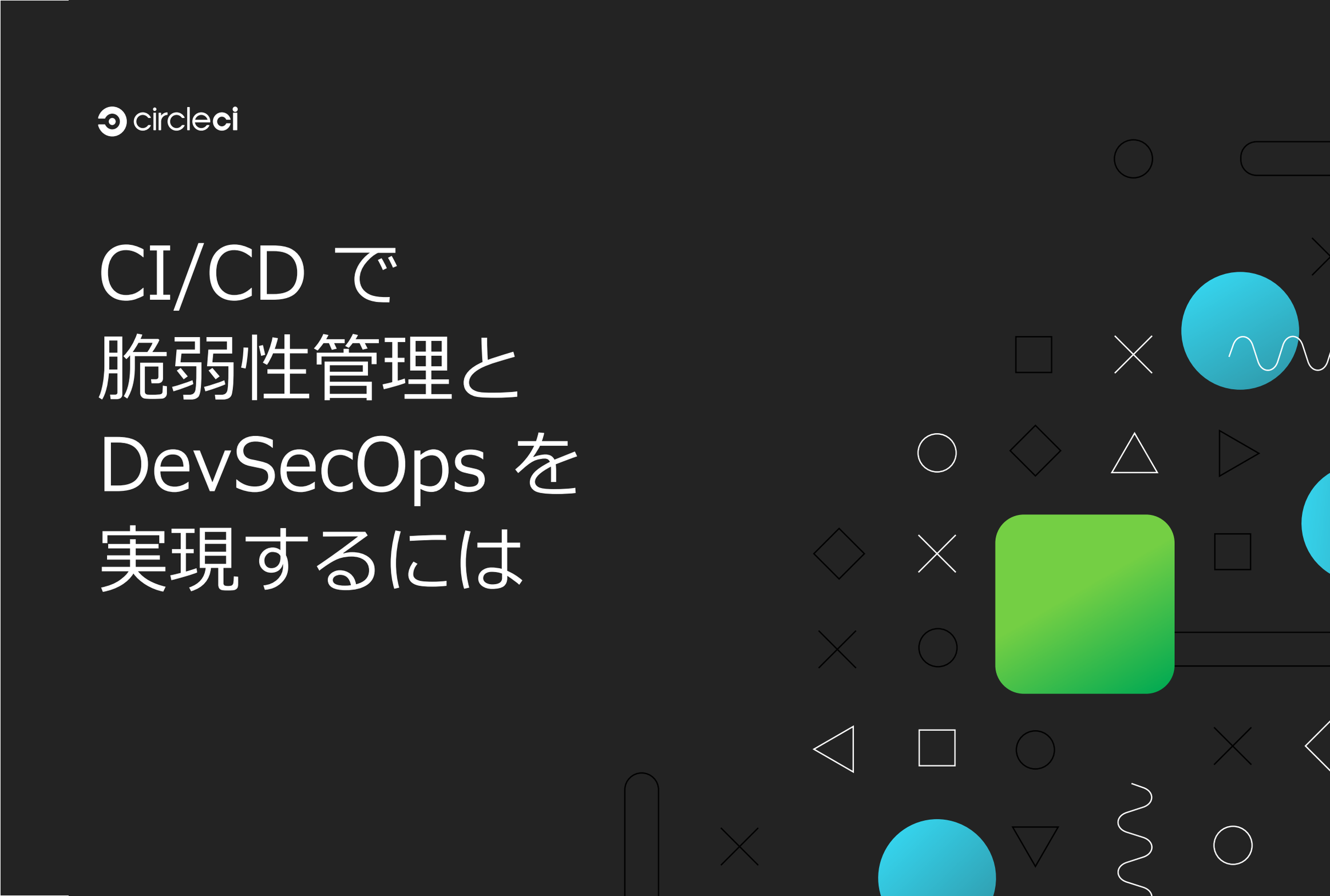
circleci

CI/CD で脆弱性管理と DevSecOps を実現するには



世界トップクラスのソフトウェア開発チームでは、すでに DevOps 手法が主流です。たしかに DevOps を取り入れれば、ソフトウェアのビルドからテスト、デプロイメントまでをシンプルにできますが、それだけでは重要な問題、つまりセキュリティと脆弱性の管理が抜け落ちています。

脆弱性は、DevOps パイプラインのさまざまなポイントで紛れ込んできます。たとえば、開発者が誤ってセキュリティ上の問題を抱えたコードを書いてしまうかもしれません。また、アプリケーションが使用しているライブラリに、セキュリティに関わるバグが存在しているかもしれません。コンテナ化したアプリケーションであっても例外ではありません。コンテナ内で実行される OS が脆弱性を抱えている可能性があるからです。

大小問わずさまざまな企業が DevOps やコンテナ化に多額の投資を行う傾向を受け、脆弱性管理にもアプローチの転換が求められています。もはや開発者は、サーバーの最新バージョンへの更新を運用チーム任せにしていてはなりません。開発チーム自体が、ワークフローにセキュリティを組み込む必要があるのです。

本書では、モダンな開発手法や DevOps を実践している開発チーム向けに、CI/CD を活用して DevSecOps 手法を取り入れ、脆弱性管理を実現する方法についてご説明します。

開発手法の進化がさらなる責任を生む

現在、開発者は有用な自動化ツールを自己裁量で数多く導入しています。しかし、こうしたツールを不用意に扱うと、セキュリティ上の脆弱性が生まれかねません。私たちが扱うコードやパッケージ、パイプライン、コンテナ イメージは、価値の源泉であると同時に、脆弱性の原因でもあるのです。

ハッカーやセキュリティ研究者は、ソフトウェアやライブラリ、オペレーティング システム、インフラストラクチャに潜む脆弱性を見つけようと日々取り組んでいます。脆弱性管理とは、ソフトウェアに脆弱性がないかどうかをスキャンし、脆弱性があれば分類し、対応の優先順位を付け、パッチを適用するという作業を繰り返す、終わることのないプロセスなのです。こうしたプロセスのすべてを、DevSecOps の手法で自動化できます。

脆弱性はソフトウェアやハードウェアだけでなく、社内のプロセスにも知らぬ間に潜んでいることがあります。人員の新規雇用プロセスや、リソース アクセス権の管理プロセスがその一例です。そのため、セキュリティ ポリシーを定めてこうした脆弱性の発生を最小限に抑え、発生した場合はポリシーに従い対応する必要もあります。

DevSecOps ではこうしたセキュリティ プロセスを明確に定めて文書化しますが、一番の重要な点はプロセス全体の自動化にあります。これにより、先に挙げたような問題を解消できます。

コードの脆弱性

開発において、他者が書いたコードを使うのは珍しいことではありません。ソフトウェアの新規プロジェクト作成時には、パッケージ マネージャーでパッケージをインポートすることが一般的です。こうしたパッケージはソースが公開されている場合もあれば非公開の場合もあり、さらに別のパッケージとの間で依存関係があります。特に JavaScript で開発する際には npm を使うというように、パッケージは開発と切っても切り離せません。

ただ、こうした外部との依存関係にはリスクがつきものです。パッケージに誤ったコードが含まれていたり、悪意のあるコードが入っている可能性もあります。企業から自社ソフトウェアやパッケージの新バージョンがリリースされる場合、セキュリティの更新が含まれていることが普通です。しかし、こうした更新によって新しい脆弱性 (あるいはバグ) が生まれることもあります。

とは言え、自分で開発したソフトウェアにも脆弱性は潜んでいるものです。その原因は、誰もがやっている間に合わせの修正や、見落としから来ているかもしれません。また、コードが古く、セキュリティに関する最新のベスト プラクティスに従っていないことが原因である可能性もあります。

CI/CD パイプライン

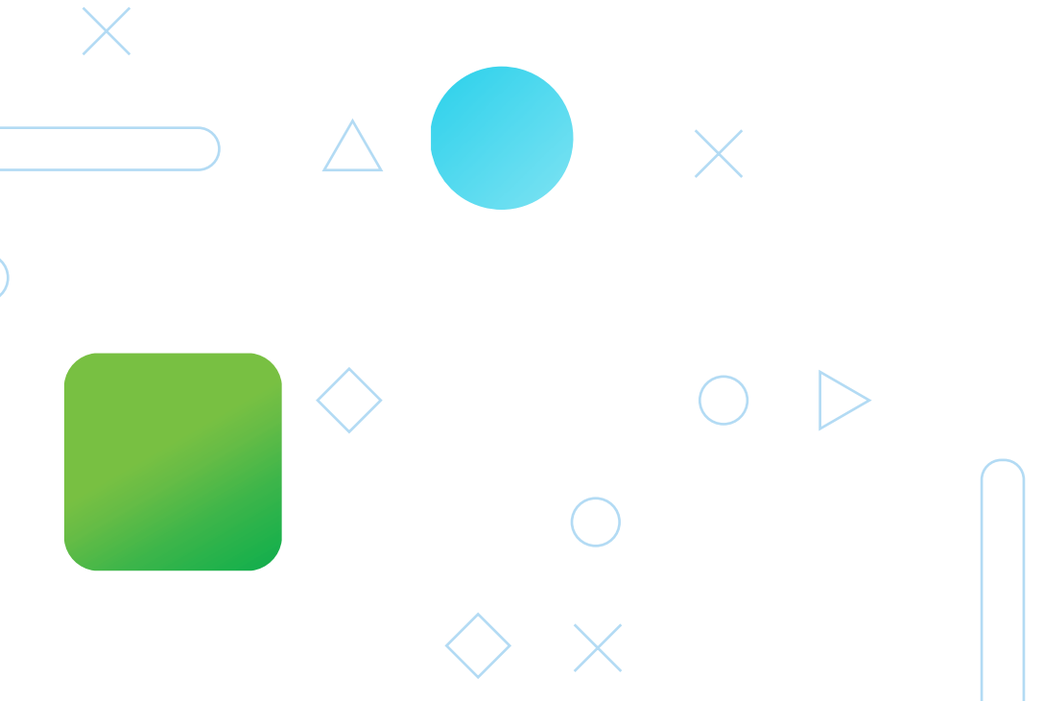
DevOps 手法の狙いは、作業を可能な限り自動化して、時間を節約するとともに人為的ミスを最小限に抑えることにあります。こうした自動化の要となるのが、CI/CD パイプラインです。パイプラインとは、ビルドのトリガーからテストの実行、ソフトウェアのデプロイまでを自動的に実行する仕組みであり、クラウド上または社内サーバー上で実行されます。

CI/CD パイプラインは DevOps の実現を成功させる鍵ですが、同時にセキュリティ上の脅威を生む原因にもなり得ます。パイプラインに不正にアクセスされると、会社全体がリスクにさらされる可能性があるからです。後ほど説明するように、CI/CD パイプラインは脆弱性管理で重要な役目を果たしますが、まずパイプライン自体をセキュアにすることが先決です。

データベースの起動を自動化したり、アプリケーションからデータベースに接続する必要がある場合、CI/CD パイプラインにユーザー名とパスワードなどのシークレットを組み込むことがあります。こうしたケースでは、適切な方法でシークレットを指定、保護できなくてはなりません。そうでなければ、機密データが保存されている本番環境のデータベースに開発者が直接アクセスできてしまいます。

開発者が退職したりチームを抜けたりする場合、管理者がすべきことは、その開発者がパイプラインにアクセスできないよう、ID やパスワードを無効化することだけではありません。既存のパスワードを開発者がメモしていたり、記憶していたりする可能性に備えて、すべてのパスワードを更新することも必要です。

サービスの自動展開に合わせて強固なパスワードを生成するなどの手法を用いれば、シークレット管理の一部を自動化することは可能です。AWS パラメータ ストアや Azure Key Vault、HashiCorp 社の Vault などのシークレット管理ツールを必ず使用して、シークレットそのものとシークレットへのアクセスを管理することを忘れないでください。



コンテナ化

開発者にとって、DockerやKubernetesのようなコンテナ機能を活用したツールは便利な存在です。コンテナにより、アプリケーションのランタイム環境をさらに細かく制御できるからです。アプリケーションに必要な依存関係とシークレットのすべてをコンテナにまとめることで、本番環境と同一のテスト環境を用意できるので、テスト環境はより理解しやすいものになります。

しかし、コンテナの利用は新しい問題の原因にもなります。開発チームは使用するライブラリの脆弱性だけでなく、コンテナ イメージの脆弱性にも気を配らなくてはいけなくなるからです。古いイメージやソフトウェアを使用していると、脆弱性が潜んでいる可能性があります。また、イメージ自体は安全でも、構成に脆弱性があるかもしれません。最新のDevOps手法を取り入れるのなら、それに応じてこれまでと違った脆弱性管理の手法が必要になるのです。

また、コンテナを使用するからといって、サーバーの管理が不要にはなることはありません。コンテナはハードウェア上で実行されるものであり、また、コンテナと合わせて他のソフトウェアを実行することもあるからです。予測可能なパスワードを使用したり、誤ってポートを開放してしまえば、社内サーバーとクラウドにホストしているサーバーの両方を危険にさらすことになります。

問題解決の鍵は DevSecOps

DevOps をすでに実践済みのチームなら、CI/CD パイプラインによるビルド、テスト、デプロイメントの自動化はすでにおなじみのものでしょう。では、DevOpsプロセスに脆弱性管理をスムーズに組み込むにはどうすればよいのでしょうか。

その答えは、DevSecOps にあります。

DevSecOps の主眼は、組織の DevOps プロセスにセキュリティを一体化させることです。『2019 年版 State of DevOps レポート (英語)』 (Puppet 社、CircleCI、Splunk 社共著) によると、平均的に言って DevOps を導入済みのチームは、そうでないチームと比べてすでにセキュリティが強固であることがわかっています。しかし、DevOps のワークフローにセキュリティは含まれません。DevSecOps なら、セキュリティがこのワークフローと明示的に統合されることになります。これにより、関係者全員がセキュリティに対して責任を持つことができるようになります。

DevOps の狙いは、チームをまとめてコラボレーションを促進し、責任を分かち合うことです。DevOps 以前の手法では、ソフトウェアがデプロイされるまでには、責任範囲の異なる複数のチームが関わることが通例でした。たとえば、開発チームがソフトウェアの開発、テスト チームがテスト、システム管理チームがサーバーへのデプロイを担当するといった具合です。

場合によっては、さらに別のチームがソフトウェアのセキュリティ脆弱性のテストを担当します。しかも、こうしたテストのタイミングは、往々にして最終リリース段階です。チーム間ではミーティングどころか、会話さえないこともあります。チームそれぞれが違うオフィスやフロア

ー、あるいは違う国で働くことさえ珍しくありません。そのため、セキュリティは定位置のないブラックボックス扱いとなっていました。

ソフトウェアのリリース サイクルが数か月や数年であるなら、こうした状況もあまり問題にはなりません。しかし DevOps 手法ではリリースを 1 日に複数回行うこともあります。セキュリティ手法が旧来のままでは、素晴らしい DevOps を導入しても、過去の開発手法に引き戻され、ソフトウェアのリリースが大幅に遅れかねません。

DevOps では開発チーム、テスト チーム、運用チームの連携を促し、一般に分野横断的なチームを結成します。この手法を導入することで、アプリケーションの開発を担当するチームが、アプリケーションのビルドやテスト、デプロイ、モニターの各方法まで把握できます。

チームに DevOps を導入する方法について知りたい方は、CircleCI 提供の eBook 『チームの DevOps 成熟度を高めるには (英語)』および『DevOps 駆動型チームによるソフトウェア テスト (英語)』をご覧ください。

DevOps にセキュリティ チームを組み込む (つまり DevSecOps を実現する) と、アプリケーション自体やそのニーズを最もよく知る担当者が、ソフトウェア ライフサイクルのあらゆる段階でセキュリティの問題に対処できるようになります。セキュリティ チームは支援チームとなり、組織の他チームがビルドやテスト、デプロイを不安なく短時間で進める準備をする役割を担います。この仕組みでは、各開発チームが、それぞれのアプリケーションのパッケージ管理、シークレット管理、コンテナ イメージ脆弱性管理を担当することになります。

では、組織の開発チーム全体に DevSecOps を導入するにはどのようにすればよいのでしょうか。

DevOps なくして DevSecOps は成り立ちません。つまり、DevOps がそのための前提条件となります。他の必須条件としては、各チームの連携と、ソフトウェアのビルドおよびデプロイメントの自動化があります。まだ DevOps を実践していないチームにとっては、DevSecOps 導入の第一歩は DevOps 手法の導入です。

すでに DevOps を実践しているチームなら、CI/CD パイプラインを実装済みでしょう。そうでない場合は、まずパイプラインの実装から始めましょう。DevSecOps 導入に向けた次の手順としては、ソフトウェア資産の目録の作成 (把握していないソフトウェアは保護できません)、各資産の責任者の特定、CI/CD パイプラインでの問題検出機能の実装などがあります。

DevSecOps の導入に向けた準備

初めにすべきことは、組織内で運用しているソフトウェア、パイプライン、コンテナ イメージの把握です。この対象には、社内ユーザーが利用しているソフトウェア、社内で管理しているパッケージ、開発チームで使用しているツール、現在開発中のソフトウェアなどが含まれます。

次に、把握した資産をどの担当チームで利用しているのかを特定します。この作業は困難なものであり、争いに発展することさえあります。しかし、こうして責任の範囲を特定し、各チームの利用状況を明確にすることで、すべてのチームにメリットがあることを忘れないでください。責任範囲の特定は、自社に最適なやり方で進めましょう。スプレッドシートやかんばんボードのほか、Airtable のようなツールを使用する手もあります。この作業の成果として、脆弱性検査が必要なソフトウェア、パイプライン、イメージの一覧と、脆弱性が見つかった場合の対応を担当するチームのリストが得られます。

全員を巻き込もう

脆弱性やバグの検査を始める前に、まずは全開発チームで話し合いを行い、次の事項について周知します。

- DevSecOps手法の導入に向けて変革を実施する理由
- 変革を行ううえで各チームに必要なもの
- 変革によってもたらされる業務の改善点
- 例外なく、すべての開発チームがソース、パイプライン、コンテナイメージのパッチ適用および更新を担当すること

この際には、各チームに変革を実践するうえで必要なリソースがそろっているか確認しなければなりません。ソフトウェア開発では、メンテナンスしやすい安定したコードを書くように求める声と、できるだけ早く新機能をリリースするように求める声の板挟みになることがよくあるからです。

そのため、開発チームに脆弱性管理を押し付けるだけではうまくいきません。まずは手法の転換を実現するうえで必要な時間とリソースを各チームに確保するとともに、今後もそれらの時間とリソースを維持していく必要があります。

DevOpsプロセスにセキュリティを組み込むことは、新機能の開発に似ています。どちらも時間をかけて計画を立て、開発とテストを実施しなければなりません。DevSecOps への転換は、スイッチを切り替えるようには進みません。DevOps と同じく、DevSecOps を定着させるには、社内全体での取り組みが必須なのです。

CI/CD パイプラインを活用する

すでにDevOpsプロセスの一環としてCI/CDパイプラインを取り入れている場合は、それらのパイプラインに脆弱性がないか確認してください。

たとえば、スクリプトや設定ファイル内にパスワードがプレーン テキストとして格納されていないか調べる必要があります。このような問題は、あなたが考える以上に存在しているものです。

多くのパイプラインでは、パラメーターをプレーン テキストではなくシークレットとして格納できます。シークレットの設定は可能ですが、デザイナービューやログにシークレットの値は表示されません。

できることなら、シークレット管理ツールを導入することをお勧めします。これらのツールでは、各種アプリケーション間でシークレットを共有しながら、シークレットを安全に一元管理できます。シークレット管理ツールへのアクセスは、マネージャーや管理者、リード開発者など、許可を受けた人だけが可能です。許可を受けていないメンバーは、シークレット管理ツールはもちろん、格納されているシークレット自体にもアクセスできません。

パイプラインに脆弱性検査ツールを追加する

CI/CD パイプラインの安全性を確認できたら、各種脆弱性を検査するツールの追加を検討してください。既存の CI/CD パイプラインにこうした検査ツールを追加すると、開発者は、すでに使用しているツールから脆弱性についての重要な情報が得られるようになります。この手法は、メール、ミーティング、レポートなどと比べて効率性がはるかに優れています。それらは作業の邪魔になることがあるのに対し、パイプラインは日常業務の一部になっているからです。また、検査ツールを導入することで、セキュリティ チームが日々行っている間違いの起こりやすい作業を自動化できるというメリットもあります。

お勧めのスキャナーには次のものがあります。

- Alcide: Kubernetes クラスタ向けの脆弱性検査ツール
- Snyk: オープン ソースのライブラリやコンテナに含まれる脆弱性を開発者が継続的に検出し、修正できるようサポート
- Stackhawk: コードに含まれるバグの検査ツール

こうした検査ツールを導入すれば、コードにセキュリティ上の脆弱性が含まれている場合に、開発者がその旨を知ることができます。重大な脆弱性であればビルドは失敗するので、本番環境にそうした脆弱性が持ち込まれることはありません。

なお、こうした検査ツールにはすべて CircleCI Orbs (再利用可能な CircleCI 設定ファイルのパッケージ) が提供されているので、既存の CI/CD パイプラインへの組み込みは手早く行えます。CircleCI Orbs については詳しくは、本書の後半で解説します。

コンテナを利用する場合、イメージの脆弱性を検査できるツールも組み込むことをお勧めします。Docker コンテナは自己完結型であり、依存関係をコンテナで指定するので、CI/CD パイプラインにコンテナの脆弱性検査ツールを追加するのは簡単です。有料ツールにもオープン ソースツールにも良いものが揃っていますが、CircleCI ではオープン ソースの Clair をお勧めします。

初回ロールアウト時にやるべきこと

脆弱性検査ツールを追加する際には、検査ツールをビルドのストッパーとしてではなく、脆弱性レポートとして使用する導入期間を用意しましょう。この期間は、各チームがセキュリティの問題を修正するだけでなく、新しい脆弱性管理の手法に慣れるための時間でもあります。繰り返しになりますが、各チームにはセキュリティの問題を修正できるだけの時間が必要です。時間の目安は、次のスプリントにおけるストーリー ポイント数個分です。

開発チームからは、脆弱性管理を任されることに不満の声が上がるかもしれません。そのため、初めて検査ツールを導入するときには、このツールを使用して、組織のコンテナに潜むすべての脆弱性がまとめられた包括的なレポートを作成することをお勧めします。そして、このレポートを基に、関係するチームと協力して、他の対応すべき作業もこなしながら、余裕を持って脆弱性の修正を行えるような計画を策定してください。

最初の検査で見つかった脆弱性の修正が一通り完了したら、今後は検査ツールが脆弱性を検出した際には、そこでビルドが中断されるような設定にします。これにより、たとえばコンテナの変更が原因で脆弱性が生じても、開発者が問題を修正しない限り、CI/CD でビルドやデプロイが自動実行されてしまうことがなくなります。

パイプラインに自動脆弱性検査ツールを組み込むと、脆弱性修正パッチのテストやバリデーションも容易になります。また別のメリットとして、セキュリティの管理が(週1回や月1回などの)定期作業ではなく、開発サイクルの日常業務になります。

セキュリティ管理の手間を見直す

重大な脆弱性の修正が終わったら、セキュリティ管理の手間を減らす方法を検討しましょう。

有効な方法としては、頻繁に使うコードを共有ライブラリに追加することが挙げられます。開発ではコードのコピー&ペーストがよく行われるので、別々のチームが、コードベースのあちこちに同じコードをコピー&ペーストしてしまうという事態になりがちです。パイプラインやコード エディター向けに、コードの重複箇所(や類似した箇所)を検出できるツールが提供されています。

こうしたツールで検出されたコードは、社内の全チームが活用できる共有ライブラリに追加することをお勧めします。この方法には、使いまわしのコードにセキュリティの問題が見つかっても、パッチをライブラリに1回当てるだけで良いというメリットもあります。こうしたメリットが特に大きいコードには、データベースにアクセスするコードや、インターネット経由での呼び出し、認証および承認、基盤 OS の機能を直接使用するコードなどが挙げられます。

大規模な組織では、共有ベース イメージを導入するという方法もよく見られます。一般的な開発チームでは、他チームが使用しているコンテナ ベース イメージを調べることなく、独自のイメージを利用してしまいがちです。ベース イメージは、可能な限りすべてのチームで共有することが推奨されます。

先ほどと同じくこの方法にも、脆弱性が見つかった場合に共有ベース イメージだけを更新すれば、そのイメージを利用している全チームでただちに問題が解決するというメリットがあります。チームごとに別々のイメージを使用していると、各自がイメージの更新を行わなくてはなりません。また、イメージを共有すれば、ビルド/テスト/デプロイの各サイクルにおいて、使用されるベース イメージが自動的に最新バージョンになるという効果もあります。

DevSecOps の実施

新しいセキュリティ手法をすべて導入し、開発プロセスと一体化させても終わりではありません。継続的にモニターすることが極めて重要です。現時点でセキュリティにまったく問題がコードでも、明日には新しい脆弱性が見つかるかもしれないからです。モニターの対象には、すでに運用中のソフトウェアだけでなく、開発が進行中のコードも含まれます。

こうしたモニタリングには、Splunk や Prisma Cloud などのツールが有用です。これらのツールでは、関係者全員にわかりやすい形式でレポートを自動生成する機能が付いています。Honeybadger や Honeycomb、LogDNA のような監視とログ記録ツールも非常に役立ちます。これらは、CircleCI を使用することでパイプラインに簡単に組み込むことができます。

ホスト環境がクラウドであるなら、クラウド ベンダーが提供するモニタリング ツールもお勧めです。たとえば、Azure では Application Insights、AWS では CloudWatch Application Insights が提供されているので、これらを使うことをお勧めします。こうしたベンダー製ツールは、不審なログイン試行や不正アクセス、アプリケーションのエラーなどの検出に対応しています。

サードパーティ各社も工夫をこらしたツールを提供しており、検出の簡素化や他チームへのモニタリングの提供、レポートの生成、追加のメトリックの測定などの機能が搭載されています。

ソフトウェアへのパッチ適用

検査ツールにより脆弱性が検出されたら、できるだけ速やかにパッチをソフトウェアに適用することが重要です。しかし残念ながら、パッチを当てると、これまで動作していた箇所が動かなくなることがあります。また、オープン ソース プロジェクトやサードパーティ ベンダーからパッチが提供されることもあります。

パッチ適用で生じるリスクを抑えるには、単体テストと結合テストの自動化などの DevOps の原則をはじめとする、適切な開発手法をパッチ適用プロセスで遵守することが大切です。特に結合テストなら、修正により問題が追加発生しないことを確かめたうえで、ソフトウェアにパッチを適用できるようになります。また、結合テストを自動化すれば、パッチ適用時の手間を大きく抑えられます。ソフトウェア資産をチーム間で統一し、共有している場合は、全チームにパッチ適用を反映できます。

一歩先へ: CircleCI Orbs

CI/CD パイプラインに CircleCI を使用している方には、CircleCI Orbs の使用をお勧めします。Orb とは、CircleCI の設定ファイルをまとめた再利用および共有可能なオープン ソース パッケージです。CircleCI Orbs を使用することで、コード検査ツールのような便利なセキュリティ サービスをはじめとして、多くのサードパーティ製サービスをスピーディにパイプラインに組み込むことができます。CircleCI では、セットアップにかかる時間を極力抑えてパイプラインに脆弱性検査を手早く組み込めるよう、脆弱性検査用の Orb を多数ご用意しています。Orb があれば、設定不要なソリューションでセキュアなパイプラインを実現できます。

脆弱性検査ツールとして Alcide や Snyk、Stackhawk をご紹介しましたが、他にも次のようなツールがあります。

- Anchore (イメージ検査ツール)
- AWS Systems Manager パラメータストア (環境のシークレットを管理および読み込むためのツール)
- Checkmarx (アプリケーションのセキュリティに特化した静的および対話型のテストツール)
- Probely (Web アプリケーションの脆弱性検査ツール)
- SecretHub (アプリケーションにパスワードやキーをプロビジョニングするツール)
- SonarCloud (コード品質の継続的検査ツール)

他にも、Docker、Kubernetes、Heroku、AWS、Azure、Google Cloud など、Web アプリケーション上での開発に役立つ Orb がこちらで提供されています。

まだOrb化されていないセキュリティ検査ツールをお持ちの場合は、コミュニティのために、ぜひOrb化してオープンソースのCircleCI Orbレジストリに公開してください。

まとめ

DevOps にセキュリティ意識を高めた DevSecOps 手法なら、CI/CD を活用して、大きな手間をかけることなく既存のデプロイ パイプラインに脆弱性の検査ツールや管理ツールを追加できます。

こうした追加ツールの導入では、まず DevSecOps に開発チームが慣れるために基本的な検査ツールから始め、その後、時間をかけて検査対象の脆弱性の数や種類を増やしていくとよいでしょう。

脆弱性管理は、CI/CD が開発チームのパワーアップに役立つ領域の例の一つに過ぎません。変化に強いシステムを構築すれば、チームはより少ないリスクですぐれた品質のコードを素早く開発できるようになります。また、CI パイプラインを実装することで、競合との差別化につながるレバレッジ ポイントを手に入れることができます。

興味を持たれた方は、ぜひ CircleCI で完全な DevSecOps パイプラインを実装してみてください。

CircleCI の利用は、https://circleci.com でアカウントを作成して無料で始められます。