IT資格道場

GitHub Advanced Security (GH-500) 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

SAMPLE

無料サンプル(第1章まで)

GH-500 出題範囲(Study guide・Skills measured as of July 2026)準拠
2026年11月版

ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

SAMPLE

目次

この試験の範囲 [overview]

第1章 Describe GitHub Security suites, features, and ecosystem [1.1.1/1.1.2/1.1.3/1.1.4/1.2.1/1.2.2/1.2.3/1.2.4/1.3.1/1.3.2/1.3.3/1.3.4/1.4.1/1.4.2]

- セキュリティ スイートの構成と画面の場所 [1.1.1]
- 3 つの守りの対象と、要件からの製品選び [1.1.2]
- 公開リポジトリと私有リポジトリ、プランごとの違い [1.1.3]
- Security overview で見られる事 [1.1.4]
- Secret Protection と Code Security の違いと重なり [1.2.1]
- 開発の流れ全体にセキュリティ機能を当てはめる [1.2.2]
- 防止を先に置く策と、関門で止める策 [1.2.3]
- security campaigns でアラートの積み残しを減らす [1.2.4]
- 秘密と脆弱性を見つける仕組み [1.3.1]
- アラートの一覧・優先付け・取る行動 [1.3.2]
- アラートを閉じる影響と、良い慣行 [1.3.3]
- 開発者・セキュリティ担当・管理者の責任分担 [1.3.4]
- アラートを見られる人と、通知の受け手 [1.4.1]
- サプライ チェーンの考え方と Dependabot アラートの中身 [1.4.2]

サンプルはここまでです

SAMPLE

このテキストの位置づけ: 学習用の本編です。**これだけで問題が解けるようになる**ことが目標で、仕組み・設定・YAML の読み方を合格ラインの深さで扱います。

SAMPLE

この試験の範囲 [overview]

GitHub Advanced Security(試験コード GH-500) は、GitHub のセキュリティ機能を、設定し、使い、運用し、組織に広げる力を問う試験です。出題は設定画面の項目や YAML の断片、組織の状況を見せて、「どの機能を使うか」「誰が何をできるか」「どこを直すか」を選ばせる形が中心です。用語の暗記だけでは足りません。

範囲は6つの分野に分かれます。このテキストも、同じ順に章を立てます(下の表)。配点は Microsoft の公式学習ガイド(Skills measured as of July 2026)に書かれた幅です。

分野	配点	内容
1 GitHub のセキュリティ機能の全体像	15〜20%	製品の構成、開発の流れへの当てはめ、アラートの扱い、権限
2 Secret Protection	15〜20%	有効化、push protection、validity checks、片付け、bypass、独自パターン
3 サプライチェーンの安全	15〜20%	dependency graph、Dependabot、dependency review、SBOM
4 Code Security	10〜15%	code scanning、CodeQL、SARIF、ワークフロー、アラートの読み方
5 セキュリティ運用	15〜20%	アドバイザリ、優先付け、ルールセット、キャンペーン、クエリ スイート

SAMPLE

分野	配点	内容
6 セキュリティ機能の管理	10〜15%	有効化の単位、security configuration、ポリシー、役割、大規模な自動化

受験料・合格ラインは、申し込み前に Microsoft の公式ページで最新情報を確認してください。

SAMPLE

第1章 Describe GitHub Security suites, features, and ecosystem [1.1.1/1.1.2/1.1.3/1.1.4/1.2.1/1.2.2/1.2.3/1.2.4/1.3.1/1.3.2/1.3.3/1.3.4/1.4.1/1.4.2]

この章は GitHub のセキュリティ機能の全体図を扱います。配点は 15〜20% で、どの機能がどの製品に入っているか、誰が何をできるか、アラートをどう扱うかを問われます。[1]

流れは 4 つの塊です。1 つ目は製品の構成と画面の場所（1.1）。2 つ目は開発の流れに機能を当てはめる考え方（1.2）。3 つ目はアラートの検出と対応（1.3）。4 つ目はアクセスの管理と、依存関係の守り方（1.4）です。設問は「この要件にはどの製品が要るか」「この場面で誰が承認するか」「この設定は誰が変えられるか」を選ばせる形が中心になります。

セキュリティ スイートの構成と画面の場所 [1.1.1]

GitHub のセキュリティ機能は、全プランで使える機能と、買わないと使えない機能の 2 層に分かれています。買う側の製品は、かつては GitHub Advanced Security という 1 つの製品でした。今は GitHub Secret Protection と GitHub Code Security の 2 つに分かれています。[2]

SAMPLE

層	名前	入っている機能の系統
全プラン	サプライ チェーンの機能など	dependency graph、Dependabot alerts、security updates、version updates、GitHub Advisory Database、SBOM の書き出し、security advisories、artifact attestations、ルールセット
有料	GitHub Secret Protection	秘密情報の漏えいを見つけて防ぐ機能
有料	GitHub Code Security	脆弱性を見つけて直す機能

2 つの製品を買えるのは GitHub Team か GitHub Enterprise のプランだけです。Free や Pro の個人は買えません。[2] 旧 GitHub Advanced Security のライセンスを持つ顧客は、2 つの製品の機能を全部まとめて使えます。

画面の場所は設問になりやすい項目です。現在の画面では、アラートの置き場所の呼び名が「Security」から「Security and quality」に変わっています。

見たい物	場所
secret scanning のアラート	リポジトリの Security and quality タブ
code scanning のアラート	リポジトリの Security and quality タブ
Dependabot のアラート	リポジトリの Security and quality タブ（左の Findings の Dependabot から Vulnerabilities）。dependency graph にも出る
dependency graph	リポジトリの Insights タブ
Security overview（GitHub.com・GHEC）	組織の Security and quality タブ

SAMPLE

見たい物	場所
Security overview（GHES の最新版）	組織の Security タブ

[3456]

設定を切り替える入口は 3 段あります。リポジトリでは Settings の "Security and quality" にある Advanced Security で各機能を切り替えます。組織では Settings の "Security" にある Advanced Security の Configurations で security configuration を作ります。企業向けの設定は、Settings の左サイドバーにある Advanced Security です。[78]

組織の Advanced Security 設定には 2 つの柱があります。security configurations は「リポジトリごとの機能の入り切り」の束で、global settings は「組織全体の振る舞い」を決める設定です。global settings では、アクティブなコミッターの数やリポジトリの一覧も見られます。[79]

設問の勘どころは名前の対応です。即答できるようにします。secret scanning は Secret Protection、code scanning は Code Security、dependency graph は全プランです。

3 つの守りの対象と、要件からの製品選び [1.1.2]

3 つの束は、守る対象で区別します。コードの脆弱性は Code Security、漏れた秘密情報は Secret Protection、依存関係の弱点は主にサプライ チェーンの機能の担当で、混同はできません。[3]

SAMPLE

製品	守る物	入っている機能
Secret Protection	シークレット	secret scanning、push protection、AI-detected secrets、custom patterns、delegated bypass と delegated alert dismissal、security campaigns、Security overview
Code Security	コード	code scanning、CodeQL CLI、Copilot Autofix、AI Scan、security campaigns、custom auto-triage rules、dependency review、Security overview
サプライ チェーンの機能	依存関係	dependency graph、dependency review、Dependabot alerts、malware alerts、Dependabot updates、immutable releases、artifact attestations

SAMPLE

210

表を読むときの注意が 4 つあります。1 つ目に、dependency review は Code Security の機能です。名前から「依存関係だから無料」と思いがちですが、私有リポジトリでは Code Security が要ります。公開リポジトリでの扱いは文書で書き方が分かれます（GitHub security features と Dependency review の説明では公開リポジトリで使える、About GitHub Advanced Security の表では Code Security の無い公開リポジトリは使えない印）。2 つ目に、delegated bypass と delegated alert dismissal は Secret

Protection の機能として載っています。ただし delegated alert dismissal は code scanning と Dependabot のアラートにも使えるので、Secret Protection 専用の機能ではありません。3 つ目に、security campaigns と Security overview は両方の製品に載ります。4 つ目に、Dependabot の custom auto-triage rules は Code Security の機能で、無視・スヌーズ・セキュリティ更新の自動実行を決めます。基本の Dependabot alerts は全プランで、購入は要りません。[211]

AI Scan は CodeQL が対応しない言語やフレームワークの脆弱性を、AI の検出エンジンで pull request のレビュー中に探す Code Security の機能です。CodeQL を置き換えるのではなく、CodeQL の穴を埋めます。code scanning の有効化は必要ですが、CodeQL の default setup は要りません。[28]

dependency graph はサプライ チェーン機能の中心です。dependency review、Dependabot alerts、security updates の 3 つはグラフの情報に頼ります。例外は Dependabot version updates で、グラフを使わず依存のセマンティックバージョンに頼ります。[10]

要件から製品を選ぶ問は、次の対応で解けます。

要件の言い方	選ぶ物
API キーやトークンの誤コミットを防ぎたい	Secret Protection（push protection）
アプリのコードに SQL インジェクションがないか調べたい	Code Security（code scanning）
使っているライブラリの既知の脆弱性を知りたい	Dependabot alerts（全プラン）
pull request で脆弱な依存の追加を止めたい（私有リポジトリ）	Code Security（dependency review）

SAMPLE

要件の言い方	選ぶ物
組織固有の形式の秘密を検出したい	Secret Protection（custom patterns）
悪意あるパッケージの混入を知りたい	Dependabot malware alerts（GitHub.com と GHES 3.22 以降）

artifact attestations には条件が付きます。Free・Pro・Team のプランでは、公開リポジトリだけが対象です。私有・internal のリポジトリでは GitHub Enterprise Cloud が要ります。[3]

場面問題の読み方

難しい設問は、複数の要件を 1 つの場面に混ぜます。たとえば「Team プランの組織が、私有リポジトリで秘密の誤コミットを止め、さらに依存の脆弱性の通知も受けたい。予算は最小にする」という場面です。秘密を止める push protection は Secret Protection が要ります。Dependabot alerts は全プランなので追加の購入は要りません。要件を機能に分解し、機能ごとに「どの層か」を当てはめれば、買う製品は Secret Protection だけと決まります。

公開リポジトリと私有リポジトリ、プランごとの違い [1.1.3]

公開リポジトリでは、有料製品の一部の機能が無料で既定で有効になります。私有・internal のリポジトリで同じ機能を動かすには、該当する製品を買う必要があります。[2]

製品	公開リポジトリで既定で使える機能
Code Security	code scanning、CodeQL CLI、Copilot Autofix、dependency review
Secret Protection	secret scanning のアラート、push protection

SAMPLE

公開リポジトリでも無料にならない機能があります。About GitHub Advanced Security の表で、公開リポジトリでも製品の購入が要る印になっているのは、security campaigns、custom auto-triage rules、Security overview、AI-detected secrets、custom patterns、delegated bypass for push protection です。同じ表は dependency review も購入が要る印にしていますが、dependency review と custom auto-triage rules は、ほかの文書では公開リポジトリで使えると書かれていて、文書で扱いが分かれます。[2] AI Scan と agentic autofix は Code Security AI credits を消費し、公開リポジトリでも同じです。[9]

個人アカウントには別の保護があります。push protection for users は、個人が公開リポジトリへ秘密を push するのを防ぐ機能で、既定でオンです。個人アカウントの設定でいつでも無効にできます。リポジトリ側で push protection が有効でない限り、個人側でバイパスしてもアラートは作られません。[312]

公開リポジトリや公開 npm パッケージで漏えいした秘密が見つかると、GitHub はそのサービスの提供元へ知らせます。これは secret scanning の partner program で、提供元が資格情報を失効させるなどの対応をします。[3]

プラン	買えるか	要点
Free・Pro	買えない	公開リポジトリの無料機能だけ
GitHub Team	買える	組織ごとに 2 製品を選んで買う
GitHub Enterprise Cloud	買える	artifact attestations が私有リポジトリでも使える

SAMPLE

プラン	買えるか	要点
GitHub Enterprise Server	買える	2 製品または旧 GitHub Advanced Security ライセンス

25

GHES でも Code Security と Secret Protection を使えます。ただし機能の一覧は GHEC・GitHub.com と同じではありません。GHES の最新版の一覧には Copilot Autofix・AI Scan・security campaigns が載っていません。5

課金はアクティブなコミッターの数で決まります。直近 90 日以内にコミットを push した人を数え、GitHub App の bot は数えません。ある組織だけ、または 1 つのリポジトリだけにコミットする人は unique committer と呼び、その製品を無効にするとその分のライセンスが空きます。9

購入前の評価用に無料のリスク評価があります。GitHub Team と GitHub Enterprise の組織は、漏れた秘密の評価を走らせられます。利用の多い最大 20 リポジトリをコードスキャンし、Copilot Autofix でどれだけ自動修正できたかを見ることもできます。2

場面で確かめます。ある会社は GitHub Enterprise Cloud の組織に私有リポジトリを 200 個持ち、予算の都合で Code Security だけを買いました。code scanning と dependency review は私有リポジトリでも動きます。push protection は Secret Protection の機能なので、私有リポジトリでは使えません。同じ会社の公開リポジトリでは、製品を買わなくても secret scanning のアラートと push protection が既定で働きます。買った範囲と、公開リポジトリで無料の範囲は、別々に数えます。

SAMPLE

Security overview で見られる事 [1.1.4]

Security overview は、組織や企業全体のセキュリティの状況を見て、対応が要るリポジトリを探す画面です。[4]

使える条件は 2 段です。Security overview は、secret risk assessment を実行した GitHub Team か GitHub Enterprise の組織で使えます。GitHub Team のすべての組織は、無料の secret risk assessment で、漏れた秘密の状況を評価できます。Secret Protection か Code Security を買った GitHub Team のアカウントは、さらに詳しい情報の入ったビューを使えます。[4]

ビュー	見られる事
Overview	検知・修復・防止の傾向を示すダッシュボード
Risk	アラートの種類別のリスク。脆弱な依存・コードの弱点・漏れた秘密
Coverage	リポジトリごとのセキュリティ機能の導入状況
Assessments	無料の秘密情報の評価レポート
Campaigns	関連タスクのまとまり・担当者・進み具合
Enablement	チームごとの導入の速さ
Code scanning on pull requests	CodeQL と AI Scan の pull request での効き方
Dependabot	重大な脆弱性の優先付けと追跡
Secret scanning	push protection で止めた種類とバイパスしたチーム

[4]

SAMPLE

すべてのビューは、見る権限があるリポジトリの既定ブランチの情報と指標を出します。フィルターを重ねると、ビューの全数値が選択に合わせて変わります。いくつかのページから CSV で書き出せます。[4]

権限は、設問に出やすい対応表です。

立場	見える範囲
組織オーナー、security manager	組織の全リポジトリの全ビュー
組織メンバー（admin 権限）	その admin のリポジトリのデータ
組織メンバー（write 権限）	そのリポジトリの code scanning と Dependabot のデータ。Coverage は見られない
組織メンバー（read・triage だけ）	見られない
企業オーナー	自分が組織オーナーか security manager の組織について、企業レベルで集計を見る

[1][3][4]

組織メンバーが見る組織レベルのページは、最近更新された 3,000 リポジトリの結果だけを表示します。組織オーナーと security manager は全リポジトリの結果を見ます。Assessments ビューは組織オーナーと security manager だけが使えます。[1][3]

ダッシュボードの数値には 2 つの落とし穴があります。同じ過去の期間でも、見る時期によって数値が変わることがあります。リポジトリの削除やアドバイザリの変更が原因です。コンプライアンスの報告や監査に、ダッシュボードの数値は使えません。audit log を使います。もう 1 つは、Security overview が出すのはアクティブなアラートだという点です。何も表示されなくても、未検出の問題があるか、機能が有効になっていない可能性が残ります。[4]

SAMPLE

場面で確かめます。組織メンバーが、あるリポジトリで write 権限だけを持っているとします。Security overview では、そのリポジトリの code scanning と Dependabot のデータは見えます。secret scanning のデータは見えず、Coverage ビューも使えません。見るには admin 権限か、セキュリティ アラートへのアクセス権が要ります。

Security overview のアラートから security campaign を作って管理できます。作業の入口が Security overview にあるので、次の節の campaign と対でつなげて覚えます。

Secret Protection と Code Security の違いと重なり [1.2.1]

2 つの製品は、検出の仕組みが同じではありません。Secret Protection は秘密の文字列のパターンを照合します。Code Security はコードの意味を解析します。

観点	Secret Protection	Code Security
見つける物	ハードコードされた秘密（キー・トークン・パスワード）	コードの脆弱性とバグ（SQL インジェクションなど）
検出の方法	提携先・汎用・独自のパターンの照合と、AI による非構造の秘密の検出	CodeQL などの静的解析、データの流れの追跡
見る範囲	全ブランチの Git 履歴全体	解析したコード。既定ブランチと pull request が中心
防ぐ機能	push protection	pull request での code scanning、dependency review
修正の手助け	秘密の失効が基本	Copilot Autofix

SAMPLE

1415

2 つの製品は独立して有効にできます。片方だけを買った場合、もう片方の機能は動かない仕組みです。security configuration で Secret Protection を有効にすると、secret scanning のアラートが有効になります。dependency review が有効になるのは、Code Security と dependency graph が両方有効なときです。[7]

AI の機能は製品ごとに別の AI credits を消費します。AI Scan と agentic autofix は Code Security AI credits を消費します。Secret Protection 側では、AI 検出シークレットの push protection が Secret Protection AI credits を消費します。AI の利用分は AI usage insights の AI credit consumption に出ます。Copilot Autofix は agentic autofix とは別物で、AI credits を消費しません。[9]

Security overview は両方の製品に含まれ、組織全体のリスクの分布を見せる共通の入口になります。組み合わせて使う場面の典型は、同じリポジトリで push protection と code scanning を併用することです。push protection が秘密を持ち込む commit を止め、code scanning が pull request でコードの弱点の混入を止めます。片方がもう片方の穴を埋める関係にはなりません。別の穴をそれぞれ塞ぎます。

security configuration は、Secret Protection と Code Security の設定を 1 つにまとめて、組織内のリポジトリへ配る道具です。

開発の流れ全体にセキュリティ機能を当てはめる [1.2.2]

secure SDLC は、書く・push する・pull request を出す・マージする・リリースする・運用するの各段に、それぞれ働く機能を割り当てる考え方です。

SAMPLE

段	働く機能	出るもの
書く・push する	push protection	秘密を含む push のブロック
pull request	code scanning、dependency review、secret scanning の結果	新しい問題の指摘、脆弱な依存の変更の表示
マージ	ルールセットの必須チェック	条件を満たさない pull request のブロック
スケジュール・イベント	code scanning の定期実行、Dependabot alerts	既存コードの問題、新しい脆弱性のアラート
修復	security campaigns、Dependabot security updates、Copilot Autofix	まとめて渡す修復の依頼、更新の pull request
測定・監査	Security overview、audit log、アラートのタイムライン	傾向、履歴

31617

Dependabot は dependency graph と GitHub Advisory Database を突き合わせて alerts を作ります。GitHub Actions のビルド中に依存の情報を dependency graph へ提出することもできます。ビルド時に依存を引き込む言語では、これで推移依存まで含んだ完全なグラフになります。[10]

大きな組織への導入は、次の 6 つのフェーズに分けて進める方法が公式に示されています。

1. 導入の戦略と目標をそろえる
2. 有効化の準備をする
3. パイロットで一部のプロジェクトとチームから試す（任意）

SAMPLE

4. 社内向けの文書を作る
5. code scanning を広げる
6. secret scanning を広げる

[17]

順序には意味があります。最初に戦略と目標をそろえ、パイロットは任意で、code scanning を secret scanning より先に広げます。secret scanning は設定が少なく導入が簡単ですが、新しい検出と過去の検出をどう扱うかの方針が要ります。[17]

監査に使える道具は 5 つあります。アラートの履歴タイムライン、Security overview、audit log、API、webhook です。[16]

防止を先に置く策と、関門で止める策 [1.2.3]

セキュリティの戦略は 2 種類に分けて考えます。防止優先は、問題が入る前に止める考え方です。ゲート型は、問題が入った状態を検査で見つけてマージを止める考え方です。

分類	機能	動き
防止優先	push protection	push の途中で秘密を調べ、見つけたら push を止める
防止優先	dependency review	マージ前に依存の変更の影響を見せる
防止優先	code scanning（pull request）	修正してからマージできるようアラートを示す
ゲート型	ルールセットの "Require code scanning results"	必須ツールが規定の重大度のアラートを出したらマージを止める

SAMPLE

分類	機能	動き
ゲート型	ルールセットの "Require secret scanning alerts are resolved"	開いたアラートがある pull request のマージを止める（公開プレビュー）
ゲート型	delegated bypass、delegated alert dismissal	バイパスや dismiss を承認制にする

[1][2][18][19]

push protection のバイパスは、既定ではリポジトリの write 権限があれば、理由を指定してできます。バイパスすると audit log に記録され、オーナーや security manager などへメールが飛びます。bypass の理由が "used in tests" か "false positive" なら、その理由で閉じたアラートが作られます。"I'll fix it later" なら開いたアラートが作られます。[1][2]

delegated bypass を有効にすると、権限のない人はバイパスの申請をして承認を待ちます。有効でなければ、貢献者は自分の判断でバイパスできます。つまり delegated bypass は、防止優先の push protection に承認という関門を足す機能です。delegated alert dismissal を入れると、write 権限の人でもアラートを直接閉じられず、申請になります。[20][21]

ルールセットの code scanning ルールは、3 つの条件のどれかでマージを止めます。必須ツールが規定の重大度のアラートを見つけたとき、必須ツールの解析がまだ終わっていないとき、必須ツールがリポジトリに設定されていないときです。重大度は 2 本立てで指定し、1 つに絞る必要はありません。Alerts は None・Errors・Errors and Warnings・All から選びます。Security alerts は None・Critical・High or higher・Medium or higher・All から選びます。[18][22]

SAMPLE

secret scanning のルールは、2 つの条件で止めます。pull request の先頭コミットのスキャンが完了していないとき、または、選んだ種類に合う開いたアラートを pull request が持ち込んだときです。提携先・汎用・独自のパターンに設定でき、AI-detected secrets は対象外です。[18]

ルールセットの enforcement status は Active（作成した時点から強制）と Disabled（強制しない）を選べます。上限はリポジトリごとに最大 75、組織全体のルールセットも最大 75 です。[18]

もう 1 つの対比は、新しく入れない策と、既存の積み残しを減らす策です。push protection・dependency review・pull request の code scanning は新しい問題の混入を防ぎます。すでにある大量のアラート、つまりセキュリティ負債は、security campaign と Copilot Autofix でまとめて減らします。設問では、「今ある 500 件のアラートを減らしたい」なら campaign、「今後、秘密の混入を止めたい」なら push protection と読み分けると迷いません。

場面で確かめます。開発チームが、秘密の誤コミットを減らしたいと考えています。write 権限を持つ全員が自分の判断でバイパスできる状態は避けたい、という条件も付いています。この場合は、push protection を有効にして、delegated bypass で bypass list を絞る形が当てはまります。bypass list に入っていない貢献者は申請を出し、レビュー担当が承認するまで push は通りません。申請は 7 日で期限切れになるので、放置はできません。ゲートは 1 段増えますが、すでにある 500 件のアラートは減りません。そちらは campaign の仕事です。

security campaigns でアラートの積み残しを減らす [1.2.4]

security campaign は、関連するアラートをまとめ、開発者と協力してセキュリティのバックログを減らす仕組みです。使えるのは、Secret Protection か Code Security を有効にした GitHub Team の組織です。[23]

SAMPLE

campaign の中身を、日々の運用に沿って見ます。

項目	内容
目的	関連アラートを束ね、優先度の高い物を開発者に直してもらう。セキュリティ負債を減らす
窓口	各 campaign に、質問・レビュー・協力の窓口になる担当者が決まっている
通知	開発者は、自分が貢献できる campaign の通知を受け取る
テンプレート	性質の近いアラートをまとめる。1 件の修正で得た知識を他の件にも使える
自動修正	code scanning のアラートでは Copilot Autofix が自動で修正案を出す
進み具合	Security overview の Campaigns ビューで、関連タスクのまとまり・担当者・進み具合を追う

[2][3][4]

作る場所は Security overview です。アラートを選んで campaign を作り、管理します。REST API でも作成・操作できます。作成には name・description・ends_at（終わりの日時。未来の日時にする）が必須で、managers（campaign の管理者。現時点では 1 人だけ指定できる）、team_managers、code_scanning_alerts（リポジトリ ID とアラート番号）などを渡します。この API を使えるのは、組織のオーナーか security manager です。[23][24]

code と secret の campaign には違いがあります。

SAMPLE

項目	code	secret
状態	一般提供	公開プレビュー
対象にできるアラート	既定ブランチのアラートだけ	対応（既定ブランチだけという制限は書かれていない）
開発者への通知の条件	リポジトリの write 権限が要る	アラート一覧を見られる権限が要る
アラートの割り当て	権限は上がらない	権限が一時的に上がることがある
自動修正	Copilot Autofix	なし

[23]

アラートの割り当てには規則があります。code scanning と secret scanning のアラートは、リポジトリの write 権限があるユーザーに割り当てます。secret scanning のアラートの担当者がアラート一覧を見られない場合は、そのアラートについて権限が一時的に上がり、担当から外れると取り消されます。GitHub は、アラートに割り当てられたときと、そのアラートが dismiss されたときにユーザーへ通知します。[23]

agentic autofix が有効なら、1 つの campaign で code scanning のアラートを最大 25 件まで Copilot に割り当てられます。1 回の割り当てで agent の session が 1 つ起動し、選んだアラートに対する pull request が 1 つ作られます。[23]

プログラムで扱うには、code scanning のアラートの担当者の割り当てと解除、担当者での絞り込みに REST API があります。アラートの割り当てと割り当て解除を知らせる webhook もあります。[23]

SAMPLE

campaign を使う場面の見分け方は単純です。大量の既存アラートを、期限を決めて開発者に直してもらうなら campaign です。新しい混入を止めるなら campaign ではなく push protection や pull request の検査です。

秘密と脆弱性を見つける仕組み [1.3.1]

検出の仕組みは、秘密、コード、依存関係の 3 系統で異なり、同じ道具では見られません。

secret scanning

secret scanning は、リポジトリの全ブランチの Git 履歴全体を調べます。Git 履歴に加えて、issues・pull request・Discussions の説明とコメント、wiki、secret gist の文章も対象です。新しい種類が追加されると、GitHub は定期的にリポジトリを再スキャンします。[14]

検出するパターンは 4 種類です。

種類	内容	アラートの行き先
提携先のパターン	サービス提供元と連携した、提供元固有の資格情報	公開リポジトリと公開 npm パッケージでは提供元へ通知（partner alerts）。secret scanning が有効なリポジトリでは既定のアラート一覧にも出る
汎用のパターン	秘密鍵、データベースの接続文字列など、提供元に紐づかない構造のある秘密	リポジトリの generic alerts の一覧
独自のパターン	組織固有の秘密を、正規表現で定義する custom patterns	既定のアラート一覧

SAMPLE

種類	内容	アラートの行き先
AI-detected secrets	パスワードのような構造のない秘密を AI で検出	generic alerts の一覧

[1425]

アラートの一覧は、既定のアラート（サポートされたパターンと独自のパターン）と、generic alerts（汎用のパターンと AI で検出した秘密）の 2 つに分かれます。generic alerts は誤検知やテスト用の値が多めで、1 つのリポジトリに最大 5,000 件（開いたアラートと閉じたアラートの合計）という上限があります。Security overview の要約ビューには出ず、Secret scanning ビューにだけ出ます。[25]

認証に 2 つの値の組が要る資格情報では、同じファイルに両方がそろったときだけアラートが作られます。部分的な漏れの情報に、重大な漏れが埋もれないようにするためです。[25]

アラートの種類も 3 つあります。user alerts はリポジトリの Security and quality タブに出る通常のアラートです。push protection alerts は、貢献者が push protection をバイパスしたときに出ます。partner alerts は提供元だけに送られ、リポジトリの管理者には届きません。管理者の対応は要りません。[25]

push protection は、push の最中に秘密を調べ、見つけたらブロックして理由のメッセージを出します。ブロックできるのは、提携先のパターン、汎用のパターン、custom patterns、AI で見つけた汎用の秘密です。コマンドラインの push に加えて、GitHub の画面でのコミットやファイルのアップロード、REST API の要求でも効きます。ただし AI で見つけた秘密は、REST API の要求では止まりません。[12]

SAMPLE

code scanning

code scanning は、GitHub が作る CodeQL か、サードパーティのツールでコードを解析します。CodeQL は、コードを問い合わせ可能なデータにして、脆弱性の型（variant）を探す言語とツール群で、結果を code scanning alerts として出します。[1526] サードパーティのツールは、SARIF 形式で結果を出せば code scanning と連携できます。SARIF は、解析ツールの結果を表す標準のファイル形式です。GitHub が読むのは SARIF 2.1.0 の対応する部分だけで、全部ではありません。CodeQL のワークフローと CodeQL CLI の結果は、この形式を自動で満たします。[27]

code scanning が解析するのは、既定ブランチと pull request です。スケジュールや push などのイベントでも走らせられます。解析には GitHub Actions を使うので、実行ごとに Actions の分を消費します。私有リポジトリで使うには Code Security が要ります。[15]

依存関係の脆弱性

Dependabot alerts は、dependency graph と GitHub Advisory Database を突き合わせて、脆弱な依存を見つけます。dependency graph は、リポジトリの manifest と lockfile に書かれた依存を解析して作り、直接依存と推移依存の両方が入ります。既定ブランチへ manifest か lockfile の変更が push されたときに自動で更新されます。[1028]

Dependabot は、新しい脆弱性が GitHub Advisory Database に追加されたとき、または dependency graph が変わったときにアラートを出します。GitHub が審査したアドバイザリだけがアラートの元になります。アーカイブされたリポジトリはスキャンしません。GitHub Actions では、セマンティックバージョンを使うアクションだけがアラートになり、SHA で固定したアクションは対象外です。[28]

SAMPLE

dependency graph が対応しないエコシステムの依存は、dependency submission API で提出できます。[10]

reachability analysis は、Dependabot alert の脆弱な関数にコードから到達できるかを、CodeQL の静的解析で調べて判定を付ける公開プレビューの機能です。JavaScript（npm・pnpm・Yarn）と TypeScript のリポジトリで、lock ファイルが要ります。判定が Unreachable のアラートは優先度を下げる目安にとどめ、それだけを理由に dismiss はしません。[29][30]

アラートの一覧・優先付け・取る行動 [1.3.2]

アラートごとに、使える状態が違います。

種類	状態	閉じる操作
code scanning	open・closed・dismissed・fixed	直す、または dismiss
secret scanning	open・resolved	理由を選んで resolve
Dependabot	open・dismissed（と、修正されて閉じる）	理由を選んで dismiss

[33][32]

REST API でアラートを更新する時に指定する state は、Dependabot が dismissed か open、secret scanning が open か resolved です。アラートを閉じる方法は 2 つです。コードを直して閉じるか、dismiss して閉じます。アラートの履歴タイムラインには、状態の変化が原因を問わず記録されます。[34][16]

優先付けの手がかりを種類ごとにそろえます。code scanning の重大度は 2 段です。すべてのアラートに Error・Warning・Note の severity があります。CodeQL のセキュリティのアラートには、さらに Critical・High・Medium・

SAMPLE

Low の security severity が付きます。security severity がある場合は、画面も判定もそちらが優先で、severity は表に出ません。security severity は CVSS の区分に従っていて、独自の尺度ではありません。[35] Dependabot は、重大度、EPSS スコア、依存のスコープ（開発か本番か）、依存の関係（直接か推移か）、reachability の判定を合わせて優先を付けます。開発専用の依存のアラートは、scope:development のフィルターで見分けて後回しにする手があります。[2930] secret scanning では validity checks が手がかりで、検出した秘密がまだ有効かを確かめて優先を付けます。[14]

Dependabot のアラートは、複数を選んでまとめて閉じられます。code scanning のアラートも、CWE などで絞り込んで一括で dismiss できます。[3034]

アラートを受けて取る行動は、場面で選びます。

場面	取る行動
コードの問題で、直せる	直す。Copilot Autofix の修正案を使える
直すのに時間がかかり、別のチームが担当する	担当者を付ける（write 権限のユーザー）、または Issue にする
誤検知、テスト専用のコード	理由を選んで dismiss
漏れた秘密	先にローテート（失効して作り直す）。そのあとにアラートを閉じる
脆弱な依存	Dependabot security updates の pull request で更新する

code scanning のアラートは、Issue にひも付けられます。1 つのアラートに 1 つの Issue、1 つの Issue で最大 50 件のアラートを追跡できます。ひも付ける先の Issue は、別のリポジトリの物でも構いません。アラートと Issue の状態

SAMPLE

は自動で同期されません。コードを直してアラートが自動で閉じても、Issue は開いたままで、手動で閉じます。Issue を削除するとひも付けが消えますが、アラートは開いたままです。[36]

絞り込みには REST API も使えます。code scanning のアラートは assignees パラメータで絞り込めます。`*` は担当者が 1 人以上、`none` は担当者なしです。secret scanning の API では、結果を `"push_protection_bypassed": true` のもので絞り込むと、push protection のバイパス由来のアラートだけを見分けられます。[31][16]

アラートの対応を受け取る webhook のイベント名は、code_scanning_alert、dependabot_alert、secret_scanning_alert の 3 つです。audit log には secret scanning と Dependabot のイベントが出ますが、code scanning のイベントは出ません。[16]

Dependabot には auto-triage rules があり、アラートを自動で無視・スヌーズし、または更新の pull request を作らせます。GitHub が用意した既定のルール（GitHub presets）はすべてのリポジトリで使え、たとえば開発専用の npm の依存の低影響なアラートを自動で閉じます。自分で作る custom rules は、私有リポジトリでは Code Security が要ります（公開リポジトリで使えるかは文書で書き方が分かれます）。ルールは通知より先に適用されるので、自動で閉じたアラートは通知を出しません。自動で閉じたアラートは、あとで開き直せます。依存のスコープが開発から本番へ変わったときなどに、自動で開き直されます。[11]

場面で確かめます。Dependabot が、開発専用の依存で重大度 High のアラートを出し、修正された版も示しています。reachability の判定は Unreachable です。この判定は優先度を下げる目安にとどめ、dismiss の理由にはしません。結果はコードや advisory の変化で変わるためです。本番の

SAMPLE

依存のアラートを先に直し、この件は Dependabot security updates の pull request で更新するのが自然です。

アラートを閉じる影響と、良い慣行 [1.3.3]

アラートを閉じる理由は、種類ごとに選べる値が決まっています。

種類	理由の値
code scanning	false positive、won't fix、used in tests、mitigated
Dependabot	fix_started、inaccurate、no_bandwidth、not_used、tolerable_risk
secret scanning	false_positive、wont_fix、revoked、used_in_tests（API の一覧にはほかに pattern_edited と pattern_deleted がある）

313233

閉じた事は記録されます。code scanning のアラートを dismiss すると、全ブランチで閉じ、理由が記録されます。任意でコメントを添えられ、コメントはアラートのタイムラインに残って、監査の根拠に使えます。code scanning の REST API ではフィールド名が dismissed_comment で、Dependabot の GraphQL API では dismissComment です。コメントを書かなくても dismiss はできますが、監査の根拠は残りません。理由の選び方しだいで、そのクエリが今後の解析から外れる場合もあるので、雑に選べません。[3430]

状態の規則もあります。Dependabot は open のアラートだけ dismiss できます。修正されていない dismiss 済みアラートは再び開けますが、修正されて

SAMPLE

閉じたアラートは開き直せません。リポジトリをアーカイブすると、そのリポジトリの開いたアラートは自動で閉じます。[30]

push protection のバイパス理由 "used in tests" と "false positive" は、その理由で閉じたアラートを作ります。"I'll fix it later" は開いたアラートを作ります。[12]

良い慣行は 4 つです。

1. dismiss の前に、アラートを開いて内容を調べ、適切な理由を選ぶ。
2. 理由とコメントを書いて、あとで読める根拠を残す。
3. 閉じる権限を絞る。delegated alert dismissal で、誰がアラートを直接 dismiss できるかを承認制にできる。
4. 閉じたあとも、新しい情報に合わせて見直す。直す労力が利益を上回る場合やテスト専用のコードに限って dismiss を使い、残りは直す。

漏れた秘密には例外があります。閉じる前にローテートします。私有リポジトリで漏れた秘密を GitHub へ報告する（Report leak）時も、先にローテートします。報告するとその秘密が失効し、使っているワークフローが止まる恐れがあるためです。Git 履歴からの除去は手間がかかり、すでに失効させていれば、多くの場合は要りません。secret scanning は、トークンがリポジトリから取り除かれても、アラートを自動では閉じません。手動で閉じます。[14][37]

CodeQL のアラートを false positive として dismiss する理由が、サニタイズ用のライブラリが未対応だから、という場合があります。GitHub はこのとき、CodeQL のリポジトリへ貢献して解析を改善するよう勧めています。[34]

SAMPLE

開発者・セキュリティ担当・管理者の責任分担 [1.3.4]

役割	主な責務	根拠となる権限
開発者	アラートの修正を担う。割り当てられた担当を直す	write 権限。campaign の割り当て先は write 権限のあるユーザー
セキュリティ担当（security manager）	優先を決める。組織のアラートを確認し、設定を変える。dismiss の申請を承認する	全リポジトリの read、組織内の全セキュリティアラートの write、組織とリポジトリのセキュリティ設定の変更
組織オーナー	security manager の任命、security configuration の作成、delegated 機能の設定	組織の管理権限
企業オーナー	Advanced Security の有効化ポリシーとライセンスの管理。全リポジトリで無効にもできる	企業の管理権限
監査担当	対応状況の確認	履歴・Security overview・audit log・API・webhook

3823216

security manager は組織レベルのロールで、オーナーが個人やチームに割り当てます。権限を絞った custom role も作れます。たとえば、secret scanning の結果とバイパス申請だけを扱うロールです。security manager のロールを持つチームについて、チームとリポジトリの両方で admin 権限を持つ人は、そのチームのアクセスレベルを変えられます。ただしアクセスそのものは取り除けません。[38]

SAMPLE

場面問題は 2 つの型が出ます。1 つ目は、delegated dismissal が有効なときの流れです。write 権限の開発者はアラートを直接 dismiss できず、dismiss の申請を出します。承認か却下をするのは組織オーナーと security manager で、開発者本人は承認できません。レビュー担当にはメールが届き、結果は申請者にメールで返ります。承認されればアラートが閉じ、却下されればアラートは開いたままです。[21]

2 つ目は、漏れた認証情報への対処です。直ちにローテートします。修正の主体は開発者です。Git 履歴からの除去は手間がかかり、失効させたあとなら、多くの場合は要りません。

アラートを見られる人と、通知の受け手 [1.4.1]

アラートの種類ごとに、見られる人の条件が違います。

アラート	見られる人
code scanning	write 権限のユーザー
secret scanning	リポジトリのオーナー、組織オーナー、security manager、admin 権限のユーザー、見る権限を渡された人。write・maintain の人は自分のコミットの alert 情報だけを見られる（一覧は見られない）。割り当てられた人が一覧を見られない時は、そのアラートについて権限が一時的に上がる
Dependabot	リポジトリの管理者、組織オーナー、write か maintain 権限の人、明示的に権限を渡された人とチーム

353730

SAMPLE

通知の既定の受け手は、次の 2 つを両方満たす人です。リポジトリで write・maintain・admin のどれかの権限を持つこと。そのリポジトリを watch していて、セキュリティ アラートかすべての活動の通知を有効にしていること。個人の通知設定で変えられます。Dependabot は、有効にした直前までの脆弱な依存について通知を出さず、有効にしたあとに見つかったものから通知します。[28] push protection のバイパスの通知は宛先が別です。個人アカウントと組織と企業のオーナー、security manager、そのリポジトリを watch する管理者に、理由つきのメールが飛びます。[12]

custom role では、見られるアラートの種類を絞った組織ロールを作れます。Security overview では、その custom role は許可された種類のアラートのデータをすべてのリポジトリで見られます。[13]

delegated bypass と delegated alert dismissal

2 つの仕組みは「許可の流れ」を変えるものです。

項目	delegated bypass	delegated alert dismissal
守る操作	push protection を迂回して push する	アラートを直接閉じる
申請する人	bypass list に入っていない貢献者	write 権限のユーザー
承認する人	組織オーナー、security manager、bypass list の人、特定の fine-grained 権限を持つ custom role	組織オーナー、security manager、レビュー用の権限を持つ custom role
対象	組織が所有するリポジトリで Secret Protection が有効なもの	code scanning、secret scanning、Dependabot

SAMPLE

項目	delegated bypass	delegated alert dismissal
設定名	Bypass privileges	Prevent direct alert dismissals

20217

対象は、Secret Protection を有効にした組織所有のリポジトリです。GitHub.com の説明は GitHub Team の組織を対象に書かれています。バイパスの申請は 7 日で期限切れになります。

delegated bypass には 3 つの機能があります。選んだ個人・役割・チームに bypass 権限を渡す。選んだ actor に exemption を渡す。それ以外の貢献者に、レビューを経る申請の流れを作る。bypass 権限は止められた push を通す権限で、exemption はその人のコミット全部で push protection を飛ばす権限です。exemption は、移行用の bot やサービス アカウントなど、信頼できる自動化に渡します。2012

常にバイパスできる人は 4 種類です。

1. 組織オーナー
2. security manager
3. bypass list に入れたチーム、既定の役割、custom role の人
4. "review and manage secret scanning bypass requests" の fine-grained 権限を持つ custom role の人 20

申請の確認場所は、リポジトリの Bypass requests ページか、組織の Security overview です。fine-grained 権限を持つ GitHub App を使えば、申請を自動で承認・却下できます。承認されたらそのコミットを push できます。却下されたら、秘密を取り除かないと push できません。delegated

SAMPLE

bypass は、GitHub 上で作成・編集・アップロードしたファイルにも適用されます。[20]

bypass list の追加場所は、リポジトリの Settings の Advanced Security にある "Bypass list" の Add role or team です。追加できるのはロールとチームで、secret team は追加できません。[12]

delegated alert dismissal の提供状況は、GitHub.com では 3 種類とも使えます。GHES では code scanning と secret scanning が 3.17 以降、Dependabot が 3.21 以降です。[21]

組織オーナーと security manager 以外にレビューさせるときは、custom role を作ります。そのロールには、アラートの dismiss 申請をレビューして迂回する組織権限と、アラートを表示・dismiss・再オープンするリポジトリ権限が要ります。[21]

場面で確かめます。組織オーナーが、全リポジトリで delegated alert dismissal を強制したいとします。リポジトリごとの設定に任せると、管理者が Settings で外せます。そこで security configuration の "Prevent direct alert dismissals" を有効にし、"Enforce configuration" を選んで、組織内に適用します。こうするとリポジトリの管理者は変更できません。dismiss の申請を見るのは組織オーナーと security manager です。申請が多いときは、レビュー用の custom role を作ってチームに割り当てる手があります。

security configuration の enforce

security configuration の enforce は、リポジトリの管理者が、設定で入り切りが決まった機能の状態を変えられないようにする仕組みです。"Policy" の Enforce configuration で選びます。設定していない機能（not set）は、強制の対象になりません。[739]

SAMPLE

組織オーナーが enforce した security configuration で delegated dismissal を入れると、リポジトリの側では変えられません。組織や企業へ配る設定は、security configuration の "Prevent direct alert dismissals" です。1 つのリポジトリだけなら、Settings の Advanced Security で有効にします。[40]

enforce していない configuration は、リポジトリの管理者が設定を上書きできます。上書きすると、リポジトリの状態は removed になります。設定は関連づいたままですが、全部の設定を継承しなくなり、組織のカバレッジの数にも入りません。再び適用すれば戻ります。[41]

状態	意味
attached	設定が適用されている
enforced	適用されて強制されている。管理者は変えられない
removed	強制していない設定を管理者が上書きした
removed_by_enterprise	企業レベルの変更が組織の設定と衝突した
failed	既存の設定との衝突で適用できなかった
detached	設定が外れている。リポジトリの設定は変わらない

enforce が効かなくなる場面もあります。code scanning の有効化が適用されないのは、次の 3 つの場合です。

1. リポジトリで GitHub Actions を無効にしたとき
2. code scanning に必要な Actions が使えないとき

SAMPLE

3. default setup で解析しない言語の定義が変わったとき REST API から強制された設定を変えようとすると、呼び出しは成功したように見えますが、設定は変わりません。[39]

default setup を既存の advanced setup のリポジトリへ強制しないための選択肢もあります。"Enabled with advanced setup allowed" を選ぶと、CodeQL 解析が動いていないリポジトリにだけ default setup を有効にします。[7]

サプライ チェーンの考え方と Dependabot アラートの中身 [1.4.2]

サプライ チェーンは、アプリが頼るライブラリやツール（依存）全体です。依存の弱点が攻撃の入口になります。保護の最優先は、脆弱な依存にパッチを当てるか、マルウェアを入れ替えることです。[10]

依存の入り方は 2 通りです。manifest や lockfile に書いた物は直接依存です。他の依存が使うために入る物は推移依存です。dependency graph はどちらも持ち、推移依存は "Show paths" で、どの依存から入ったかを追えます。[10]

機能	役割	層
dependency graph	依存の一覧。SBOM の書き出しの元	全プラン
SBOM	dependency graph を SPDX 互換で書き出す。read 権限があれば UI と REST API から可能	全プラン
GitHub Advisory Database	閲覧・検索・絞り込みができる、脆弱性の整理済みリスト	全プラン

SAMPLE

機能	役割	層
Dependabot alerts	既知の脆弱性のある依存を通知	全プラン
Dependabot malware alerts	悪意ある依存の通知。通常の脆弱性アラートとは別に出る	全プラン
Dependabot security updates	脆弱な依存の更新の pull request を作る	全プラン
Dependabot version updates	脆弱でなくても新しい版へ更新する pull request を作る	全プラン
dependency review	pull request で依存の変更の影響を見せる	Code Security（公開は無料）
security advisories	公開リポジトリの脆弱性を非公開で議論・修正し、あとで公開する	全プラン
artifact attestations	作ったソフトに改ざんできない由来と完全性の保証を付ける	全プラン（私有は GHEC）

3101942

malware alerts は、依存が悪意あるパッケージとしてフラグされたときに、既定ブランチで出ます。対応する ecosystem は Composer・Go・Maven・npm・NuGet・pip・RubyGems・Rust で、GitHub.com と GHES 3.22 以降で使えます。社内のパッケージが、公開の悪意あるパッケージと ecosystem・名前・版が同じだと、誤検知が出ることがあります。423

SAMPLE

Dependabot のアラートには、次の情報が載ります。影響を受けるファイルへのリンク、脆弱性の詳細と重大度、修正された版（ある場合）、元になるアドバイザリです。アラートは、既定ブランチの依存を対象に、新しい脆弱性が追加されたときか、dependency graph が変わったときに出ます。[28]

dependency review の action は、pull request の依存の変更を、dependency review の REST API で調べます。action の既定では、脆弱なパッケージを見つけるとチェックが通りません。リポジトリで必須にすれば、マージを止められます。失敗させる重大度や、ライセンスの許可・拒否のリストを設定できます。組織では、ルールセットで dependency review の action を必須の workflow にして、全リポジトリへ強制する手もあります。[19]

Dependabot alerts は「すでに入っている脆弱性」を、dependency review は「これから入る脆弱性」を見ます。設問では、既存か新規かで読み分けます。

default の auto-triage rules は、GitHub が用意したルールで、誤検知をかなり自動でふるい落とします。開発専用の npm の依存の低影響な問題を自動で閉じる preset は、公開リポジトリでは既定で有効で、私有リポジトリでは選ばないと有効になりません。パッケージの malware alerts を自動で閉じる preset は既定で無効です。すべてのバージョンが悪意あると判定された依存を本当に追加した場合も自動で閉じてしまうため、有効にするときは注意が要ります。[11]

SAMPLE

サンプルはここまでです

続き(第2章〜巻末)は、GH-500 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

Footnotes

1. https://learn.microsoft.com/en-us/credentials/certifications/resources/study-guides/gh-500 ↩
2. https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8 ↩9 ↩10
3. https://docs.github.com/en/code-security/getting-started/github-security-features ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8
4. https://docs.github.com/en/code-security/security-overview/about-security-overview ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8
5. https://docs.github.com/en/enterprise-server@latest/get-started/learning-about-github/about-github-advanced-security ↩ ↩2 ↩3
6. https://docs.github.com/en/enterprise-server@latest/code-security/security-overview/about-security-overview ↩
7. https://docs.github.com/en/code-security/securing-your-organization/enabling-security-features-in-your-

SAMPLE

organization/creating-a-custom-security-configuration ↩ ↩2 ↩3 ↩4 ↩5 ↩6

8. https://docs.github.com/en/code-security/getting-started/securing-your-repository ↩ ↩2
9. https://docs.github.com/en/billing/concepts/product-billing/github-advanced-security ↩ ↩2 ↩3 ↩4
10. https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-supply-chain-security ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8
11. https://docs.github.com/en/code-security/dependabot/dependabot-auto-triage-rules/about-dependabot-auto-triage-rules ↩ ↩2 ↩3
12. https://docs.github.com/en/code-security/secret-scanning/introduction/about-push-protection ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8
13. https://docs.github.com/en/code-security/reference/permissions/security-overview ↩ ↩2 ↩3
14. https://docs.github.com/en/code-security/secret-scanning/introduction/about-secret-scanning ↩ ↩2 ↩3 ↩4 ↩5
15. https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning ↩ ↩2 ↩3

SAMPLE

16. https://docs.github.com/en/code-security/concepts/security-at-scale/audit-security-alerts ↩ ↩2 ↩3 ↩4 ↩5 ↩6

17. https://docs.github.com/en/code-security/adopting-github-advanced-security-at-scale/introduction-to-adopting-github-advanced-security-at-scale ↩ ↩2 ↩3

18. https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-rulesets/available-rules-for-rulesets ↩ ↩2 ↩3 ↩4

19. https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-dependency-review ↩ ↩2 ↩3

20. https://docs.github.com/en/code-security/concepts/secret-security/about-delegated-bypass-for-push-protection ↩ ↩2 ↩3 ↩4 ↩5

21. https://docs.github.com/en/code-security/concepts/security-at-scale/delegated-alert-dismissal ↩ ↩2 ↩3 ↩4 ↩5

22. https://docs.github.com/en/code-security/how-tos/find-and-fix-code-vulnerabilities/manage-your-configuration/set-merge-protection ↩

23. https://docs.github.com/en/code-security/securing-your-organization/fixing-security-alerts-at-scale/about-security-campaigns ↩ ↩2 ↩3 ↩4 ↩5 ↩6 ↩7 ↩8

24. https://docs.github.com/en/rest/campaigns/campaigns ↩

SAMPLE

25. https://docs.github.com/en/code-security/concepts/secret-security/about-alerts ↩ ↩2 ↩3 ↩4

26. https://codeql.github.com/docs/codeql-overview/about-codeql/ ↩

27. https://docs.github.com/en/code-security/code-scanning/integrating-with-code-scanning/sarif-support-for-code-scanning ↩

28. https://docs.github.com/en/code-security/dependabot/dependabot-alerts/about-dependabot-alerts ↩ ↩2 ↩3 ↩4

29. https://docs.github.com/en/code-security/concepts/supply-chain-security/about-reachability-analysis-for-dependabot-alerts ↩ ↩2

30. https://docs.github.com/en/code-security/dependabot/dependabot-alerts/viewing-and-updating-dependabot-alerts ↩ ↩2 ↩3 ↩4 ↩5 ↩6

31. https://docs.github.com/en/rest/code-scanning/code-scanning ↩ ↩2 ↩3

32. https://docs.github.com/en/rest/secret-scanning/secret-scanning ↩ ↩2

33. https://docs.github.com/en/rest/dependabot/alerts ↩ ↩2

34. https://docs.github.com/en/code-security/code-scanning/managing-code-scanning-alerts/resolving-code-scanning-alerts ↩ ↩2 ↩3 ↩4

SAMPLE

35. https://docs.github.com/en/code-security/concepts/code-scanning/code-scanning-alerts ↩ ↩2

36. https://docs.github.com/en/code-security/concepts/code-scanning/alert-tracking-with-issues ↩

37. https://docs.github.com/en/code-security/secret-scanning/managing-alerts-from-secret-scanning/resolving-alerts ↩ ↩2

38. https://docs.github.com/en/organizations/managing-peoples-access-to-your-organization-with-roles/managing-security-managers-in-your-organization ↩ ↩2

39. https://docs.github.com/en/code-security/reference/security-at-scale/configuration-enforcement ↩ ↩2

40. https://docs.github.com/en/code-security/how-tos/manage-security-alerts/manage-code-scanning-alerts/enable-delegated-alert-dismissal ↩

41. https://docs.github.com/en/code-security/reference/security-at-scale/configuration-statuses ↩

42. https://docs.github.com/en/code-security/concepts/supply-chain-security/malware-alerts ↩ ↩2

SAMPLE

GitHub Advanced Security (GH-500) 学習用テキスト(無料サンプル)

版

2026年11月版(GH-500 出題範囲(Study guide・Skills measured as of July 2026)準拠)

発行

IT資格道場(https://www.shikaku-doujou.com)

お問い合わせ

info@shikaku-doujou.com

GitHub、GitHub Advanced Security および GitHub の製品名・サービス名は、GitHub, Inc. の米国およびその他の国における商標または登録商標です。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式シラバスの内容を基に作成したもので、GitHub, Inc. および Microsoft Corporation との提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。

SAMPLE