Secure SketCH Books VOL.7

セキュリティの勘所

～品質管理編～

Secure SketCH

Secure SketCH Books VOL.7

セキュリティの勘所

～品質管理編～

システムの大規模化や複雑化に伴い年々セキュリティ脅威も高度化する中、セキュリティ品質の重要性も高まっている。そのためには、品質を低下させるセキュリティリスク要因となる脆弱性を自動で排除する魔法のツールではなく、地道な品質管理が重要である。

Topic

※本記事は2018 年 3 月に発行したNRIセキュア ニュースレターで掲載されたレポートを基に再構成しています。

1．品質管理の目的と意義

ISO9001によると、品質管理は「買い手の要求に合った品質の品物またはサービスを作り出し、維持するための活動」であり、セキュリティ開発においては、ユーザが満足するセキュリティ品質（機密性・完全性・可用性）を有するシステムを、コストや納期とのバランスを考慮しながら計画・実現し、維持・向上していくことであると解釈できる。

システムのセキュリティ品質を低下させる脆弱性がシステムに組み込まれるリスク要因を図1 に示す*1。脆弱性は、バグと同じく工程を経るごとに作りこまれる。これらリスク要因を可視化して計画的・効率的に対処していくためには、品質管理の基本に立ち返り、PDCA に沿ったセキュリティ品質管理プロセスを定義して開発工程を進めるべきである。

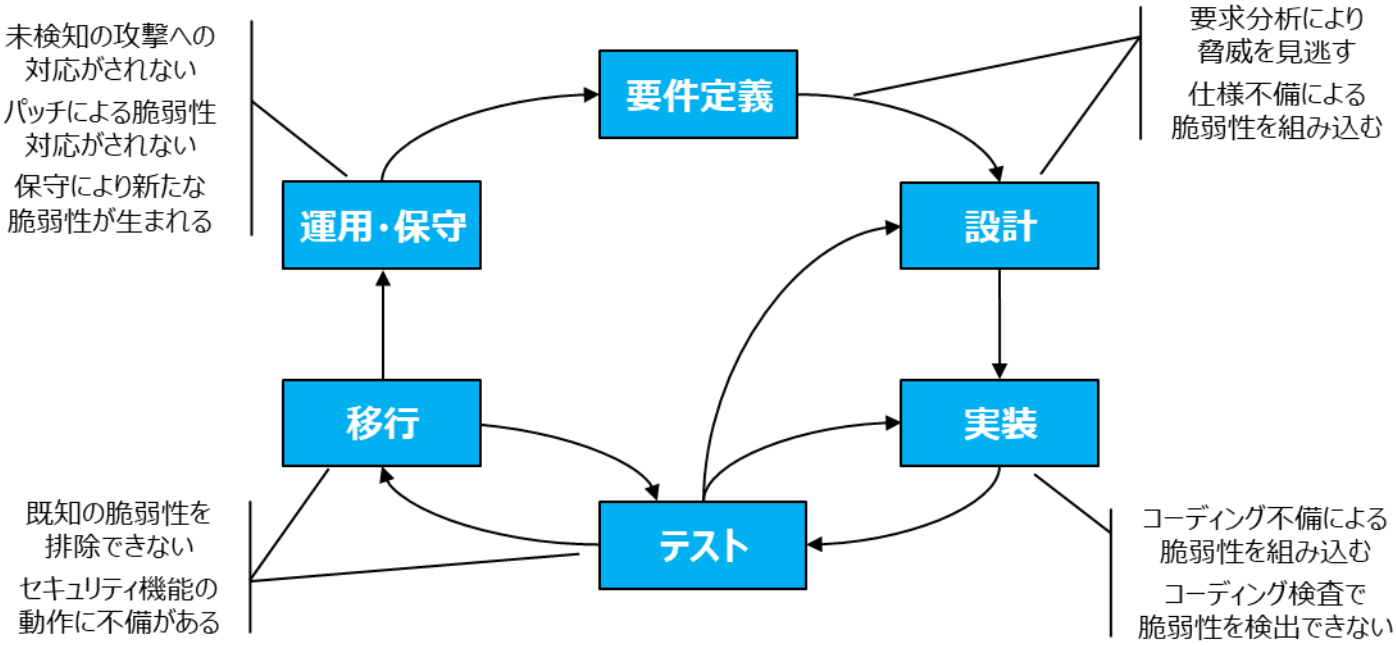


図1 工程ごとのセキュリティ品質低下（脆弱性発生）の要因

*1 IT ロードマップ2017 (野村総合研究所, 2017) https://www.nri.com/jp/news/2017/170309_1.aspx

2. セキュリティ品質管理プロセス

目標達成のための計画・実行・検査タスクを定義する

システムの設計者・開発者は想定される脅威や脆弱性を全て考慮しながら設計・開発・テストを行う必要があるが、攻撃者は脆弱性を1 つ見つけるだけでも目的達成ができることもある。このように開発者に不利な状況でセキュリティ品質目標を本番稼動まで、またはリリース後も維持・向上していくには、工程ごとに品質を細分化して計画・設計し(Plan)、計画に則り設計・開発を行い(Do)、品質基準に沿って検証する(See)必要がある。

Microsoft のSDL*2を参考に、工程ごとの品質管理プロセスの ”例” を図2に示す。

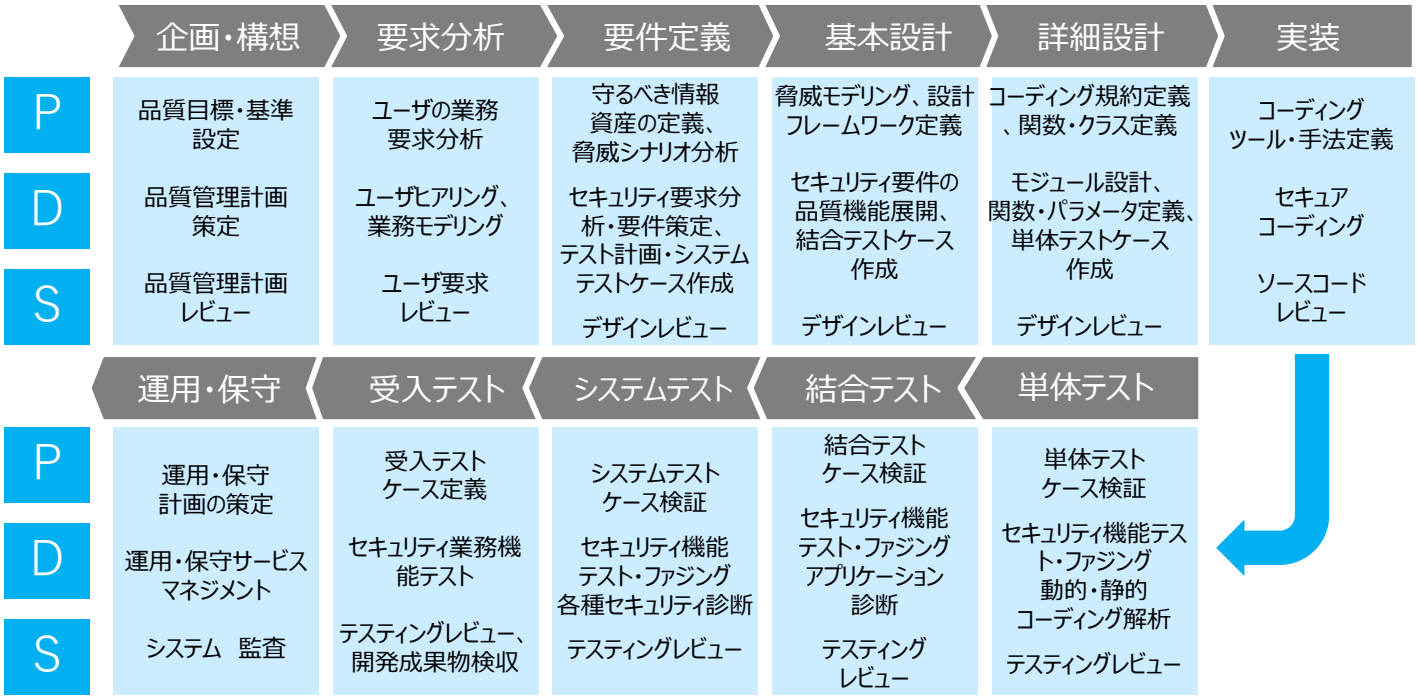


図2 工程ごとの標準的なセキュリティ品質管理プロセス

例えば基本設計工程では、前工程で定義した情報資産を細分化して「脅威モデリング」を行い、「設計フレームワーク」に沿って情報資産ごとの対策方針を明確にする。

次に、設計要件の「品質機能展開」を行って前工程で定義したセキュリティ業務要件を、技術対策領域（アクセス制御、アカウント管理等）やNIST Cybersecurity Framework の「特定・防御・検知・対応」*3や、実装レイヤー（アプリケーション、データベース、OS、ネットワーク）に展開する。そして「デザインレビュー」で、仕様の記載は明確か、実装方式や設定値に無理はないか、稼動後の運用現場が回るかなどの観点で妥当性を検証する。

検証結果は設計文書に反映し、システムオーナー承認を得た上で詳細設計工程へ進む。

*2 Microsoft Security Development Lifecycle https://msdn.microsoft.com/en-us/library/windows/desktop/cc307748.aspx
*3 NIST Cybersecurity Framework 1.1

2-1．企画・構想
品質管理のプロセスを定義し、実行計画を策定する

企画・構想工程では、品質の計画を行う。ビジネス戦略やシステムの目的を踏まえセキュリティの品質目標を設定し、本番稼動および稼動後まで継続して維持・向上する一連の管理プロセス（上述）を計画書に落としこみ、必要なトレーニングを実施する。

○ セキュリティ品質管理計画

例えばEC システムでは、他システムでのセキュリティ品質の評価結果を参考に、システム目的である売上向上を阻害しない機密性・完全性・可用性の目標を言語化し、そのための品質管理コストを算出し、他システムと比較して費用の妥当性を検証する。

品質管理の目的と、目標とする品質基準、それらを維持・向上する体制と役割・責任、工程ごとの活動内容、開発成果物、品質設計・開発／検査手法、品質管理コスト、トレーニングなどを定義する。品質基準は「品質管理の結果達成されるべきセキュリティ水準」を指す。

守るべき情報資産を明確にして情報重要度などから資産ごとに満たすべきセキュリティ品質(機密性、完全性、可用性)の目標を設定する。

プロジェクト計画と整合をとり、QCD（品質・コスト・納期）観点からベンチマーキングやコスト／リスク・ベネフィット分析等で具体化する。

システム開発において、スケジュール遅延とコスト超過を防ぎ高い品質を確保するためには計画的な設計・開発と適時の品質検査・欠陥除去が有効なのと同様、セキュリティ品質管理計画に沿った各工程のセキュリティ品質点検と欠陥(脆弱性)除去が重要といえる。

この工程ではセキュリティ品質の抽象的な概念を具体的内容にまで落とすことは難しいが、仮説ベースで計画し・設定しておき、工程が進む都度具体化しながら見直しを行う。

○ セキュリティ開発成果物

各工程で作成すべきセキュリティの開発成果物、すなわち各種セキュリティ設計文書や設定パラメータシート、セキュリティテスト文書などを一覧で定義する。

一覧化にあたり各成果物間のインプット・アウトプットの依存関係を整理しておくと、要件のトレーサビリティを確保できる。

また、成果物作成のツールをあわせて考えておくことも有益である。これらは単独で作成することもあるが、通常の開発成果物に取り込むと管理が楽になる。

○ 品質管理体制

各工程においてセキュリティ観点で関与すべきステークホルダーを洗い出し、設計・開発および維持・向上するために必要な体制、役割・責任、会議体を定義して可視化する。

セキュリティ品質目標に向かい各ステークホルダーが役割・責任を果たせる体制を工程ごとに設計する。例えば品質検査体制はユーザ、開発者に加えセキュリティの担当者やベンダを含め、第三者視点でセキュリティ品質を検査してセキュリティ責任者が承認する。

図3に、品質検査における工程ごとのステークホルダーの役割・責任を一覧化する。会議体の設計も重要である。

例えば、「**変更管理委員会(CAB)**」では、セキュリティ仕様の変更（例：認証強化）によって、使用性や性能が低下するなど業務品質も考慮する必要があるため、セキュリティ責任者やユーザを交え多様な視点で検討すべきである。

システム重要度	企画・構想				要件定義、基本設計				詳細設計、実装、単体テスト				結合テスト、受入テスト			
	開発者	PM	ｾｷｭﾘﾃｨ	ﾕｰｻﾞ	開発者	PM	ｾｷｭﾘﾃｨ	ﾕｰｻﾞ	開発者	PM	ｾｷｭﾘﾃｨ	ﾕｰｻﾞ	開発者	PM	ｾｷｭﾘﾃｨ	ﾕｰｻﾞ
高	○	○	○	●	○	○	●		○	●	△		○	○	○	●
中	○	○	●		○	○	●		○	●			○	●	●	
低	△	△			○	●			△				○	●		

○：自分で机上または実機で成果物を検査する
●：他者の検査結果（チェックシート）をレビューする
△：参考にする（検査やレビューは実施しない）

図3 工程ごとの品質検査にかかる役割・責任の定義

○セキュリティ品質の設計・開発基準

これらは設計・開発の工程で、セキュリティ品質を作り込むよりどころとなる。

設計工程では「セキュリティ設計文書雛形」や「フレームワーク」や、馴染みの薄いセキュリティ用語をユーザ・開発者間で統一する「用語集」、セキュリティ要件レビュー時の「OK/NG 判定基準」などを整備する。実装工程では、脆弱性を作り込まないための開発者の共通言語「セキュアコーディング規約」が該当する。

○ セキュリティ品質設計手法

各工程でまず最初に計画・設計しないといけないのが、セキュリティ品質をセキュリティ要件に変換し、順次具体化して作り込んでいくための考え方や手法である。

例えば、要件定義工程では、「脅威シナリオ分析」「脅威モデリング」などの手法を用いて情報資産ごとに脅威シナリオとその対策に必要なセキュリティ要件を洗い出す。基本設計工程では「ミスユースケース分析」などの結果洗い出されたセキュリティ対策要件を「セキュリティ設計フレームワーク」などを使って具体化していく。実装工程では「セキュアコーディング規約」の定義、各テスト工程では、各種「セキュリティテストケース設計方針」の検討、セキュリティ診断の「診断スコープ設計」などが必要になってくる。

品質設計をうけ、実際に設計・開発・テストを行って品質を具現化し、さらに品質検査でそれらの実現度合い（品質基準の達成率）や妥当性を確認していく。その際、品質設計の質が、設計・開発・テストの効率や、品質検査による脆弱性検出数の多寡を左右する。

○ セキュリティ品質検査手法

計画・設計されて作り込まれるセキュリティ品質は、各工程の品質検査で検証され、その結果をうけて改善・対応される。セキュアなシステムを構築するためには、まずセキュリティ機能設計を正しく実装してセキュリティ要求を満たすことを確認する。さらに、コーディングや設定に付随する脆弱性(欠陥のうち悪用される可能性のあるもの)を排除する必要もある。これらは、通常のシステム開発での品質検査や欠陥修正では対応が難しい。

近年様々なセキュリティ品質の検査手法が登場している。図4に各工程での代表的な品質検査手法を紹介する。これらを組み合わせて品質管理計画書に盛り込んでおくことで、各工程の始めにそれを見据えた品質設計が行われ、検証速度向上と手戻り減少によってシステム開発全体の効率化に繋がる。

検査種別	検査手法	分類	実施担当者	実施方法	実施形態	工程									
						企画・構想	要件定義	基本設計	詳細設計	実装(製造)	単体テスト	結合テスト	システムテスト	受入テスト	移行・運用
ドキュメント検査	デザインレビュー	静的	設計者	手動	–	○	○	○	○						
	テスティングレビュー	静的	設計者	手動/ﾂｰﾙ	–						○	○	○	○	○
コーディング検査	ソースコードレビュー	静的	開発者	手動	ﾎﾜｲﾄﾎﾞｯｸｽ					○					
	静的コード解析(SAST)	静的	ﾍﾞﾝﾀﾞ/開発者	ﾂｰﾙ	ﾎﾜｲﾄﾎﾞｯｸｽ					○	○				
	動的コード解析(DAST)	動的	ﾍﾞﾝﾀﾞ/開発者	ﾂｰﾙ	ﾎﾜｲﾄﾎﾞｯｸｽ					○	○				
セキュリティテスト	セキュリティ機能テスト	動的	開発者	手動/ﾂｰﾙ	ﾌﾞﾗｯｸﾎﾞｯｸｽ							○	○	○	
	ファジング	動的	開発者	ﾂｰﾙ	ﾌﾞﾗｯｸﾎﾞｯｸｽ						○	○	○		
セキュリティ診断	アプリケーション診断	動的	ﾍﾞﾝﾀﾞ/開発者	手動/ﾂｰﾙ	ﾌﾞﾗｯｸﾎﾞｯｸｽ							○	○		○
	プラットフォーム診断	動的	ﾍﾞﾝﾀﾞ/開発者	手動/ﾂｰﾙ	ﾌﾞﾗｯｸﾎﾞｯｸｽ								○		○
	ペネトレーションテスト	動的	ユーザ/ﾍﾞﾝﾀﾞ	手動	ﾌﾞﾗｯｸﾎﾞｯｸｽ								○		○
セキュリティアセスメント	セキュリティ評価	静的	ユーザ/ﾍﾞﾝﾀﾞ	手動	–										○
	セキュリティ監査	静的	ﾍﾞﾝﾀﾞ	手動/ﾂｰﾙ	–										○

図4 セキュリティ品質管理手法の一覧

○ 品質管理トレーニング

これまで紹介した品質管理計画、設計/開発・検査手法などに関する要員の教育は各工程の品質を左右するため、プロジェクトの開始前・初期段階で重要である。セキュリティは設計者・開発者の意識やスキル・経験に依存するところ大きいこともあり、セキュリティ目線で各工程において何をすべきか／すべきでないかの意識付けがポイントになる。

2-2．要件定義・基本設計
品質は上流工程で作りこむ

要件定義・基本設計工程では、品質計画で設定した品質目標を達成するセキュリティ機能・非機能要件を品質設計手法で言語化し、仕様上の不備（脆弱性）や漏れがないかを検証する。さらに、この段階からテスト計画やテストシナリオ／ケースも策定していく。

○ 品質目標の具体化

まずは、開発システムが保有する情報資産（データベース、ファイル）をシステムの要件定義書を参照して洗い出し、データの重要度などから情報資産ごとに満たすべき機密性・完全性・可用性をできる限り定量的に定義する。

これを満たすには、守るべき資産に対して想定される脅威を洗い出して分析し、リスク（脅威×脆弱性）の大きさ＝リスク顕在率×影響度をもとに対策要件に優先度をつけて仕様化することが必要となる。

○ 脅威シナリオの洗い出しと要求分析(フォールトツリー&ユースケース分析)

要件定義工程においては、守るべき情報資産とシステムの概要から想定される脅威シナリオをフォールトツリーを使って洗い出す。木構造の頂点にセキュリティ被害（情報漏洩など）を記述し、その要因となる脅威シナリオを攻撃の基点まで辿っていく（図5）。

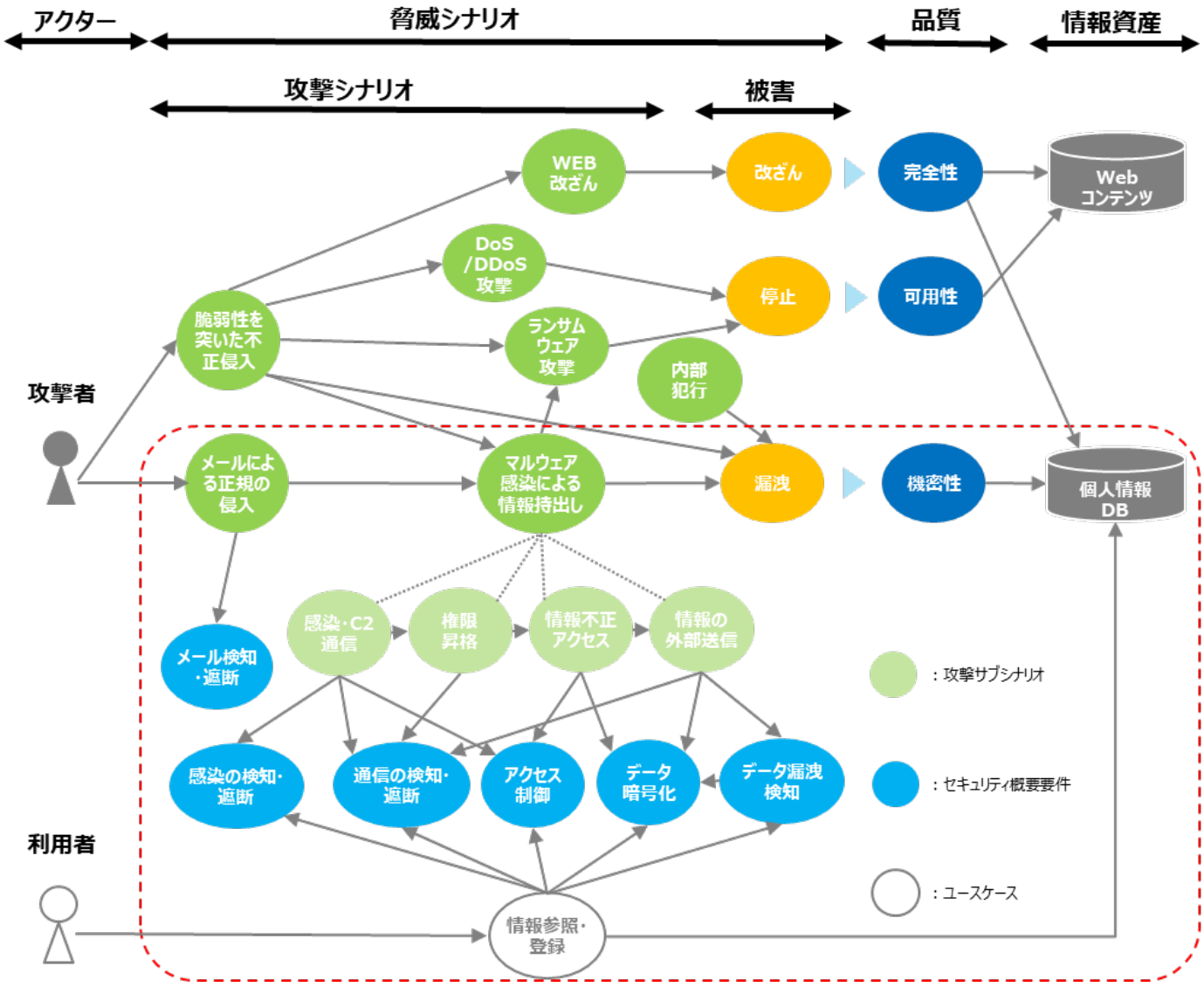


図5 脅威シナリオのフォールトツリー分析と要件のプロット

ここでUML のユースケースの考え方を用いて、アクターとして「利用者」と「攻撃者」を設定し、対策要件をこのフォールトツリーにプロットすることで要求仕様を洗い出す。

図は、網羅的に洗い出した脅威シナリオのうちマルウェア感染シナリオを例にとり、サブシナリオにまで分解した上でセキュリティ要件を洗い出すケースを示している。

1．DFDを使った脅威の識別（洗い出し）

基本設計工程に入ると、セキュリティ要求仕様とシステム全体の基本設計をもとに、脅威をモデリングし、要件定義の結果を踏まえてセキュリティ設計仕様を定義していく。この段階で検討する脅威は情報資産の用途やシステム構成、利用者や攻撃者の能力などで大きく異なり、さらに要求分析を終えた段階では設計や実装がないため脅威の予測も難しい。そのため、以下のような定量的・可視的な一貫性あるアプローチが必要といえる。

脅威モデリングでは、まずシステムをコンポーネントに分解する。アプリケーションとデータベースでは、エンティティ間のデータの流れをDFD等で可視化し、要求分析で洗い出した脅威シナリオを意識しながら攻撃対象になりうる箇所を洗い出し脅威を抽出する。漏洩や改竄などの脅威はDFDにおける信頼境界(管理組織やインターフェスが変わるシステム境界線)で発生するため、データの入出力箇所を分析することで脅威が抽出できる。

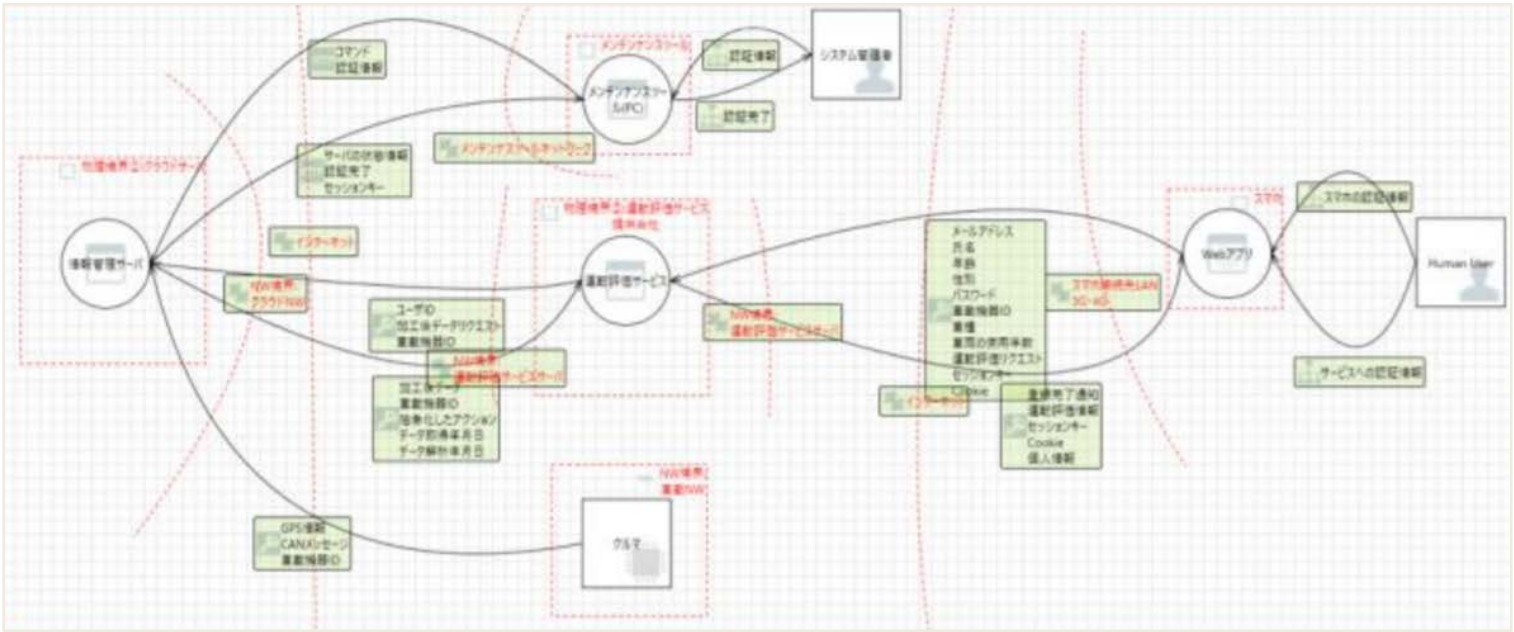


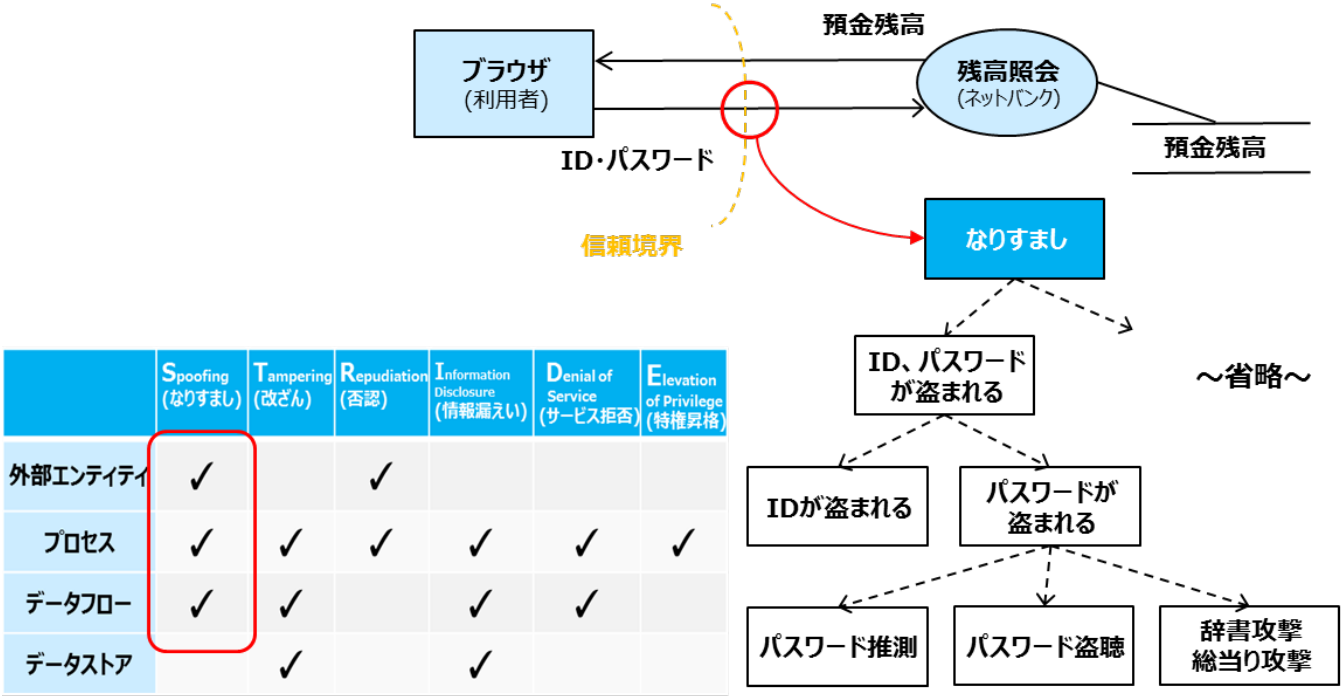
図6 DFD による脅威モデリング（Microsoft のSDL Threat Modeling Tool）

2．脅威ツリーによる脅威の評価と対策の検討

フォールトツリーを応用して、脅威ツリー(基本設計版)を作成する。例としてアプリケーションについて、DFDで抽出された脅威の要因分析を行って下位の脅威と脆弱性を段階的に詳細化し、セキュリティ被害を構成する脅威と脆弱性の集合を得る。

これらの評価を行って打ち手を考えることでセキュリティ機能を定義できる。

なお、OS やネットワークなどプラットフォームでも、同様な考え方で脅威と脆弱性をツリー上に表現して分析できる。



	Spoofing (なりすまし)	Tampering (改ざん)	Repudiation (否認)	Information Disclosure (情報漏えい)	Denial of Service (サービス拒否)	Elevation of Privilege (特権昇格)
外部エンティティ	✓		✓			
プロセス	✓	✓	✓	✓	✓	✓
データフロー	✓	✓		✓	✓	
データストア		✓		✓		

図７脅威ツリーによる脅威の評価

○ セキュリティ品質機能展開と、設計段階での品質検査

品質機能展開（QFD）とは、要求されるセキュリティ品質とそれを実現する機能を分解して可視化する作業である。ここまでに分析結果抽出されたセキュリティ仕様は要件の粒度や網羅性の観点で不足が予想されるため、図8のような抜け漏れのないセキュリティ要件フレームワークを活用して展開し、検討の抜け漏れがないことを担保する。

品質検査手法としては、要件定義書・基本設計書のデザインレビューが中心となる。仕様が曖昧でないか、想定リスクは妥当か、実装方式に無理はないか、稼動後の運用現場がまわるのか、などの観点から妥当性を検証する。

要件定義		特定	防御	検知	対応	復旧
基本機能	アカウント管理	アカウント管理対象の特定	アカウント統制	アカウント監視	アカウント停止	アカウント復旧
	認証	認証対象の特定	認証	認証監視	認証の停止	認証の復旧
	アクセス制御	アクセス制御対象の特定	アクセス制御	アクセス監視	アクセスの遮断	アクセスの復旧
	データ保護	データ保護対象の特定	データ保護	データ改ざん・漏洩範囲の調査	データ改ざん・漏洩対処	データの復旧
	バックアップ	バックアップ対象の特定	バックアップ取得	バックアップ監視	バックアップ復元	システムの復旧
共通機能	標準構成	標準構成の特定	構成管理	構成監視	構成復元	標準構成の復旧
	脆弱性管理	脆弱性情報の特定	脆弱性管理	脆弱性診断・監視	脆弱性暫定対処	脆弱性監査・報告
	マルウエア対策	マルウエア情報の特定	マルウエア遮断	マルウエア監視	マルウエア駆除・封じ込め	マルウエア感染対象の復旧
	ログ管理	ログ管理対象の特定	ログ取得・保管	ログ監視	ログ調査・分析	ログ監査・報告

基本設計（アプリ）		特定	防御	検知	対応	復旧
基本機能	アカウント管理	アカウントの申請・登録	アカウントポリシー統制	アカウント棚卸・利用監視	アカウントの無効化	アカウントの有効化
	認証	認証方式・対象の定義	ID・端末・生体による認証	ログイン監視	ログインの無効化	ログインの有効化
	アクセス制御	役割の定義	役割によるアクセス制御	アクセス状況の監視	アクセス権限の無効化	アクセス権限の有効化
	データ保護	暗号化方式の定義	通信・処理の暗号化	—	—	—
	バックアップ	—	—	—	—	—
共通機能	標準構成	プログラム構成の把握	プログラム構成管理	プログラム構成監視	モジュールの復元	プログラム構成の復旧
	脆弱性管理	プログラムの脆弱性の把握	セキュアコーディング	アプリケーション診断	プログラム修正・パッチ適用	バージョンアップ
	マルウエア対策	—	—	—	—	—
	ログ管理	ログ出力要件の定義	アプリケーションログ出力	—	—	—

図8 セキュリティ要件の設計・検証チェックリスト

○ セキュリティテスト計画とテストケース（ドラフト）

テスト計画の検討は要件定義工程または基本設計工程の段階から開始するが、セキュリティテスト計画もあわせて検討を開始する。従来この意識は希薄で、そもそもセキュリティ観点ではユーザアクセス制御の確認など最低限しか事前計画されないことが多かった。理由はセキュリティが「テストの必要がある設計要件」として認識されておらず、そもそも設計書にセキュリティ仕様が記載されることが少なかったからである。

セキュリティテスト計画書には、テスト工程全体の構成、各テストの目的・粒度、テスト手法、完了基準、テストケース定義方法、使用データ、スケジュール、役割・責任、テスト環境、テスト結果判定基準、欠陥(脆弱性)の管理ツールなどが記述される。後段のテスト工程で脆弱性をコントロールするには、各設計書をにらんで品質検査方法を定義してテストケースを作成しておくのが理想的である。この段階では、脅威モデリングの成果を利用し、実際の攻撃シナリオをベースとした結合・システムテストケースが生成できる。

なお、セキュリティテスト計画やテストケースは、セキュリティ設計文書と整合をとって工程ごとに常に見直し、スムーズにテスト工程を進められるようにする。

2-3．詳細設計

上流設計文書を漏れなく具体化し、
実装での脆弱性をコントロールする

詳細設計工程では、セキュリティ要件の実装方法（関数、クラス、基盤の 設定項目やパラメータ）を定義し、セキュリティ設計仕様を実装可能なレベルまで具体化する。

まず、プログラム構造設計のセキュリティ上の留意点や既知の脆弱性をソースコードに組み込みを防ぐ考慮事項を言語化して、コーディング規約に追加し、単体テスト開始までにコーディング作業を行うメンバーへ周知しておき、実装工程で確実に遵守徹底させる。

次に、システムの各種セキュリティ機能要件・非機能要件をプログラムの入力・処理・出力に置きえながら、入力パラメータ制御、処理ロジック、出力制御などの仕様を具体化してセキュリティ詳細設計書に落としこむ。アプリケーションでは、SQL インジェクションなどの脆弱性を突いた攻撃へのリスク要因を実装工程の段階で生まないようにする。 基盤では、ミドルウェアやOS、ネットワーク機器のセキュリティ設定項目やパラメータを整理し、設定値をシステム要件やセキュリティポリシーに適合させ脆弱性混入を防ぐ。標準構成イメージや自動化ツールで抜け漏れなく正確に・効率的に設定することもできる。

最後に、前工程で作成したテスト計画書を見直して単体テストケースを作成しておく。

品質検査手法としては、詳細設計書のデザインレビュー、コーディングサンプルなどに対するソースコード検査などが中心となる。

2-4．実装・単体テスト

設計文書や規約・ガイドラインに基づいて仕様を具現化する

実装・単体テスト工程では、詳細設計で考慮した観点やセキュアコーディング規約に沿ってプログラミングやシステム基盤の構築を行い、それらの動作を部品単位で検証する。

品質検査手法としては、ソースコードレビュー、ソースコードの動的・静的な検査（後述）、または基盤の設定手順書のレビューなどが中心となる。

最近、コーディングに連動してソースコード検査や単体テストの一種である静的セキュリティ診断(SAST）や動的セキュリティ診断(DAST）を使うケースが出てきた。

SAST は主にプログラム作成時におけるセキュリティ観点の構文チェックを、DAST は主に作成済みのプログラムを連動動作させたうえで実施する脆弱性チェックを意味する。

2-5．結合テスト・システムテスト
基本設計で定義したセキュリティ機能を検証する

結合テスト・システムテスト工程では、基本設計書や要件定義書のセキュリティ要件がシステムに正しく実装されていることを、機能やサービスの単位で確認していく。

品質管理手法としては、大きく2つに分類される。1つはセキュリティ機能テストで、ホワイトボックステスト形式（結合テスト）、ブラックボックステスト形式（システムテスト）によって、認証やアクセス制御、暗号化などセキュリティ機能の動作を確認する。
もう1つは実装過程で生じる脆弱性の外部検査で、アプリケーション診断（結合テスト以降）、プラットフォーム診断・ペネトレーションテスト（システムテスト）などのセキュリティ診断やファジングなどの動的検査が活用できる。

2-6．受入テスト
基本設計で定義したセキュリティ機能を検証する

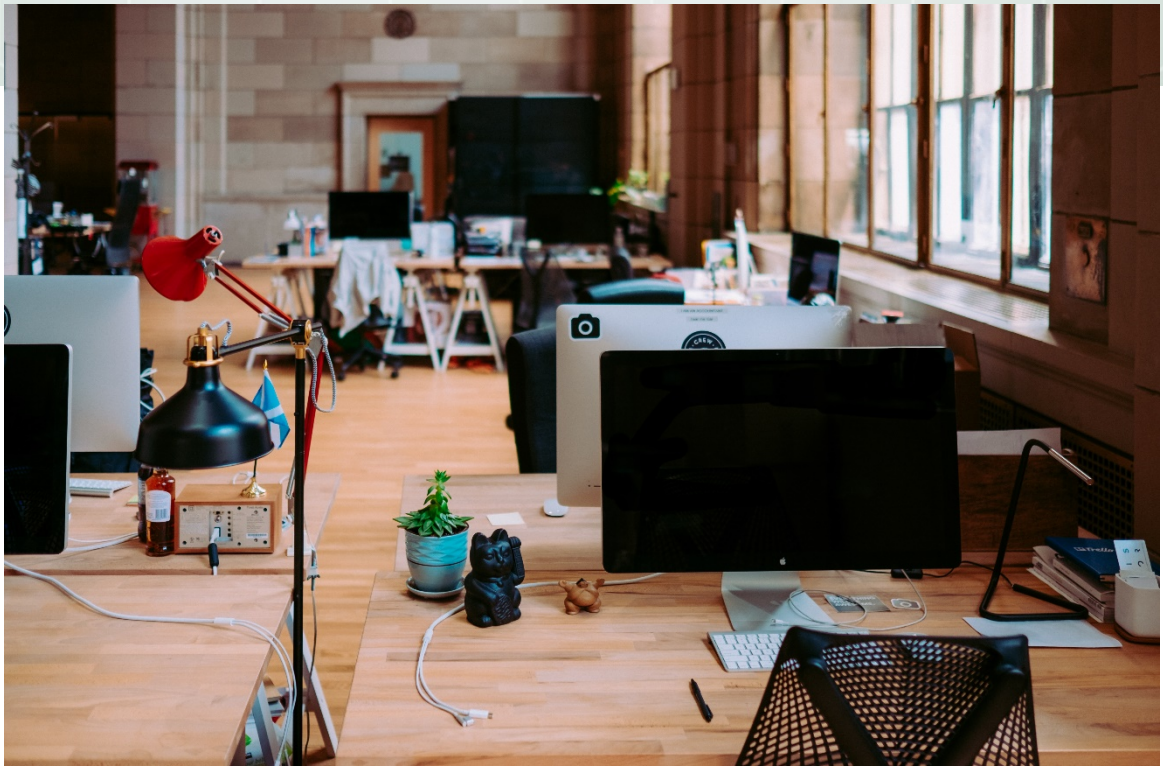
受入テスト工程では、規約や上位工程で定めた設計に基づいて実装された結果、当初定めた品質基準やシステムの非機能要件、脅威モデルまたはリスクシナリオへの対策が有効であること、セキュリティ品質強化によるビジネスへの影響が最小限であること、などを検証する。リリース後に対応する脆弱性の特定と対応方針設定もここで必ず行う。

品質管理手法としては、セキュリティテストのうち、主にブラックボックステスト形式による各種セキュリティ機能のテストを、業務要件に従ってユーザ目線で行う。

2-7．移行・運用
リリース後のセキュリティ品質を定期的に検証する

移行・運用の工程では、脆弱性管理と変更管理、定期的なアセスメントなどにより、リリースされたセキュリティ品質を維持・向上させることが目的となる。業務要件やセキュリティポリシーの変更、機能追加、新たな脆弱性の発見やセキュリティパッチなどによりセキュリティのどの機能や設定に変更が必要かを、基本文書をたどって影響分析する。

品質管理手法としては、移行ではリリース時のセキュリティ品質の最終確認、リリース後の運用では定期的な脆弱性のスキャンや脆弱性情報の評価とセキュリティパッチ管理、セキュリティ評価や監査などのアセスメントが中心となる。



おわりに

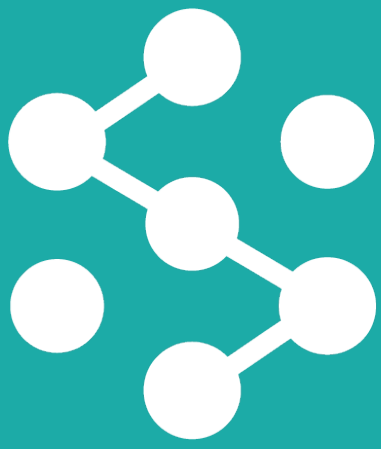
セキュリティ品質管理とは、セキュリティ品質の実現を最終的にかなえるための活動である。すなわち、上流設計工程で言語化されたセキュリティ要件を、詳細設計以降で段階的に具体化・可視化・具現化していくに従い、それによって達成されるべきセキュリティ品質をリリース後も見据えて維持・向上していくことが品質管理の目的となる。

“Security by Design” の体現を裏から支えるセキュリティ品質管理は、セキュリティ担当者や開発のプロジェクトマネージャーだけの問題ではなく、情報システムをビジネスに用いる企業全体の責務であるといえる。

筆者略歴

岡部 拓也

外資系IT企業とコンサルティングファームで延べ9年システムの構想策定～設計・導入～運用、ITガバナンス、プロジェクト管理、リスクマネジメント等に従事。2016年に野村総合研究所入社後、NRIセキュアテクノロジーズへ出向し、中長期計画策定、セキュリティ上流設計、セキュリティリスク評価、IT組織改革等、数多くのコンサルティング案件に従事している。



Secure SketCH

セキュリティ経営をシンプルに

https://www.secure-sketch.com