ソニーの製品セキュリティへの取り組み

渡邉　聡
swatanabe@jp.sony.com
Software Security Assurance Group
Sony Digital Network Applications, Inc.

お題目

- セキュリティの作りこみのライトウェイトなアプローチ
  - 私たちの活動の基本的な考え方
  - LWSSA
- セキュリティ専門組織によるセキュリティの作りこみ
  - ライトウェイトアプローチの最初の取り組み
  - LWSSA 1.0
- 開発者自身によるセキュリティの作りこみ
  - LWSSA 1.0の拡張
  - LWSSA 2.0

セキュリティの作りこみのライトウェイトなアプローチ

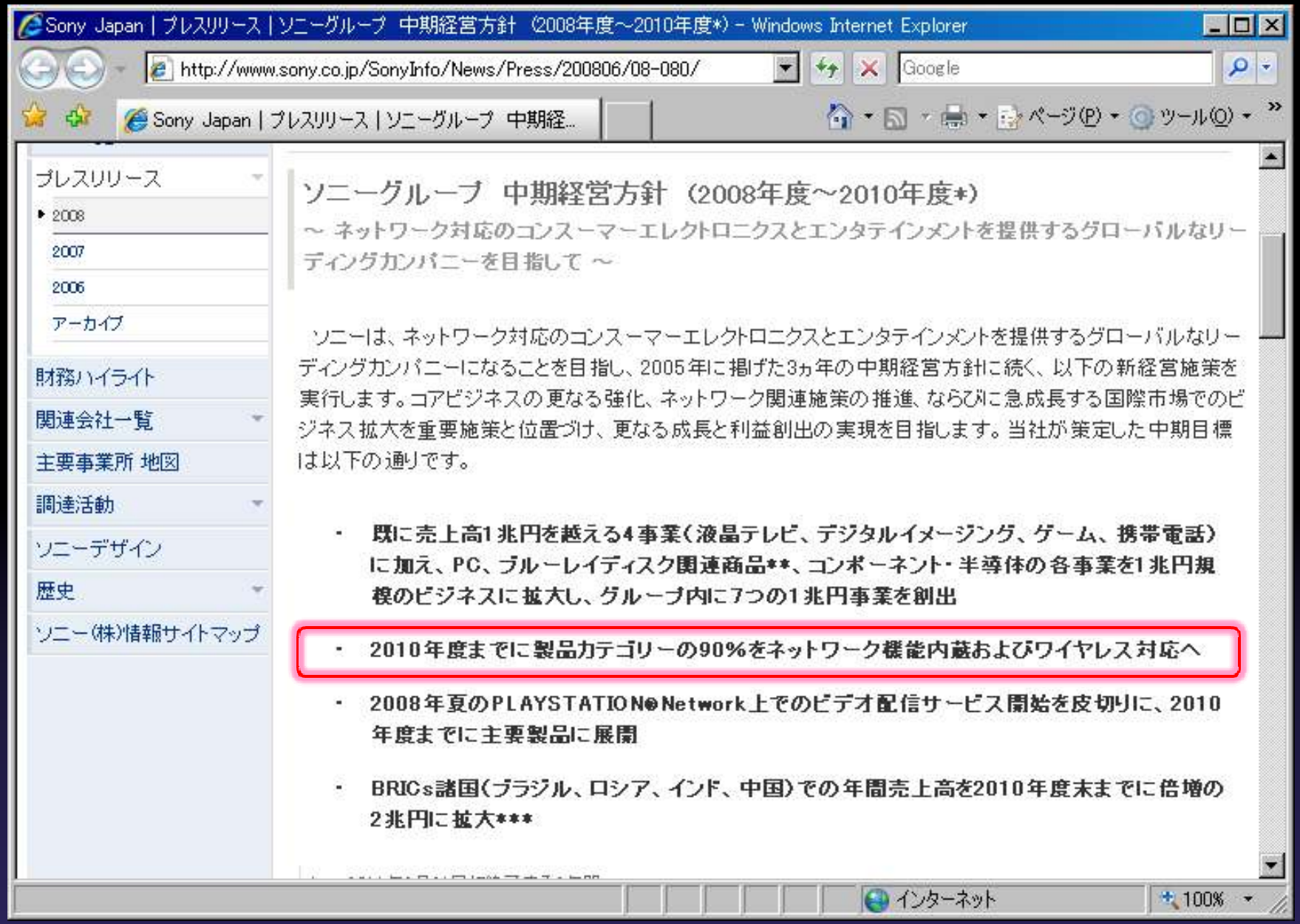
私たちの活動の基本的な考え方

家電製品のセキュリティリスクが高まってきた

近年、これらの家電製品の
ネットワーク化が進んできた

セキュリティリスクが
高まってきた

加速する家電のネットワーク化



家電製品のセキュリティ状況（私見）

- 実際のところ被害は多発していない
  - セキュリティ対策の価値がエンドユーザーにあまり認知されていない
- とはいっても、無防備であるわけにはいかない
  - 被害はゼロではない
  - 被害があると企業イメージの低下にもつながる
  - 何よりエンドユーザーが被害に遭うのは避けたい
- セキュリティ対策のために製品価格が上がることは許されない
  - 「価格」は製品の売れ行きに大きく影響する

セキュリティを高めたくても、セキュリティのために十分な予算を確保するのは難しい状況

ライトウェイトアプローチ

各セキュリティ対策の費用対効果を高める工夫を重ねる

費用対効果が大きいセキュリティ対策から始めて、与えられた時間、費用内でできるところまで実施する（できないところはリスク受容する）

Lightweight Software Security Assurance

セキュリティレビュー実績の一部

製品開発費用全体に占める
セキュリティレビュー費用の比率

製品名	脆弱性指摘件数	コード規模	レビューコスト比率	開発言語	製品種別
製品1	12 件	265,000 行	1.6 %	C/C++	PCアプリ
製品2	13 件	146,000 行	1.2 %	C/C++	PCアプリ
製品3	6 件	7,000 行	2.6 %	C/C++	PCアプリ
製品4	10 件	172,000 行	5.9 %	ASP	社内ITシステム
製品5	20 件	18,000 行	4.0 %	C/C++	PCアプリ
製品6	41 件	331,000 行	1.4 %	C/C++	組み込み家電
製品7	29 件	198,000 行	2.7 %	C/C++	PCアプリ
製品8	0 件	12,000 行	1.5 %	C/C++	PCアプリ
	131 件				

製品出荷前に修正した脆弱性の件数

約1億3100万円のパッチ作成費用のリスクを
削減できたことになる

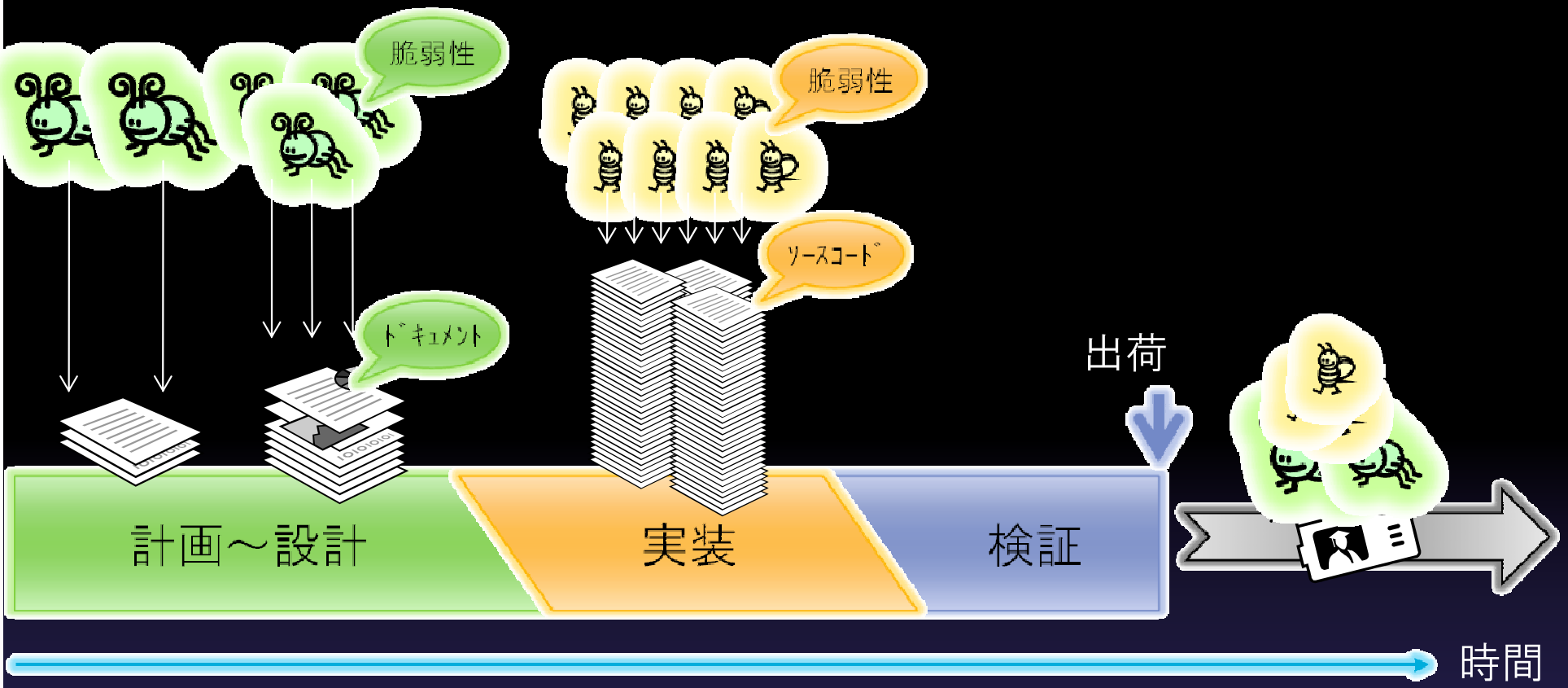
弊社過去実績値

パッチ作成費用
約100万円／1脆弱性

セキュリティ専門組織による セキュリティの作りこみ

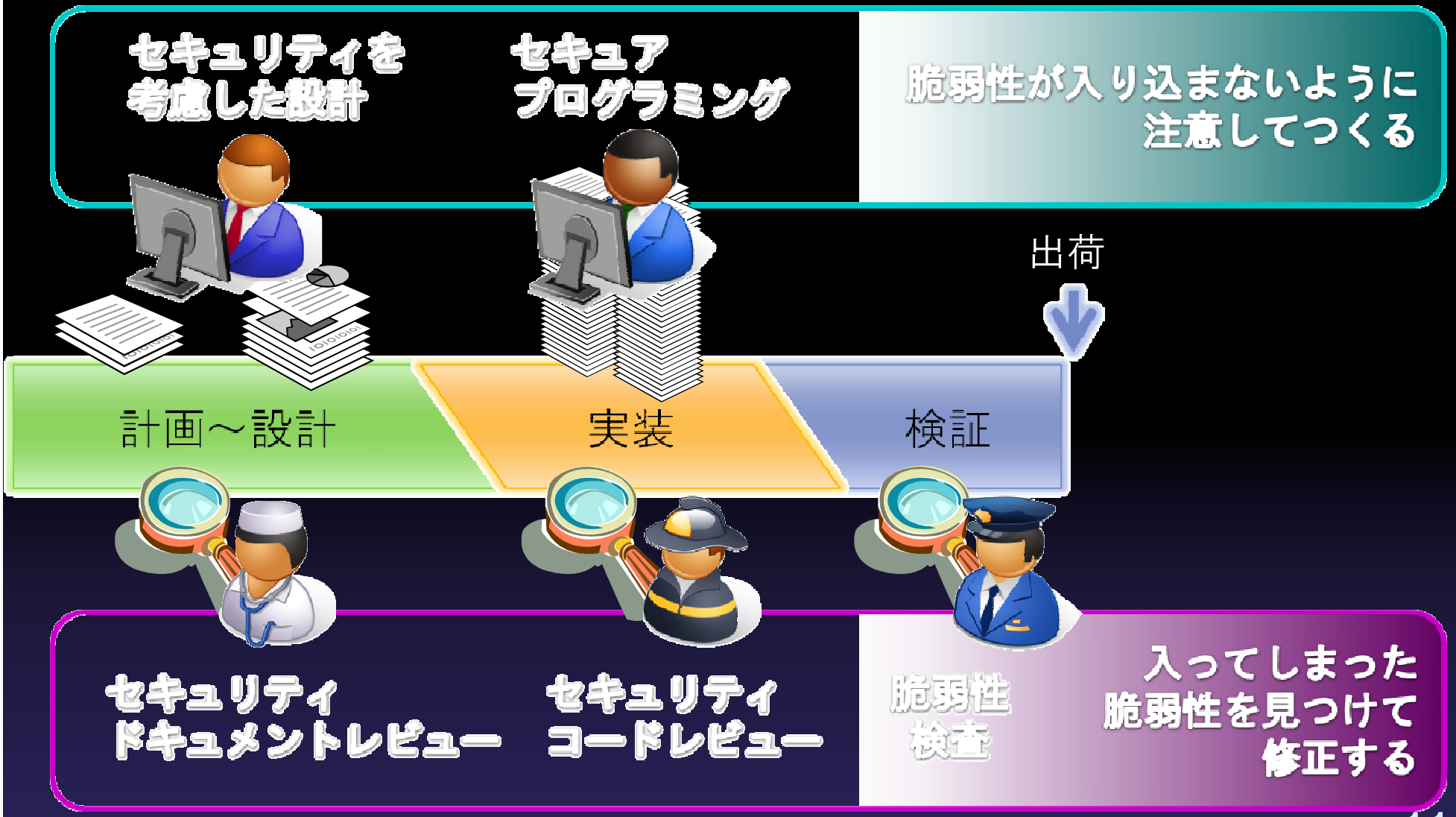
ライトウェイトアプローチの最初の取り組み

脆弱性が製品に入り込んでいく様子



（セキュリティを作りこむ）

脆弱性を予防する5つの活動

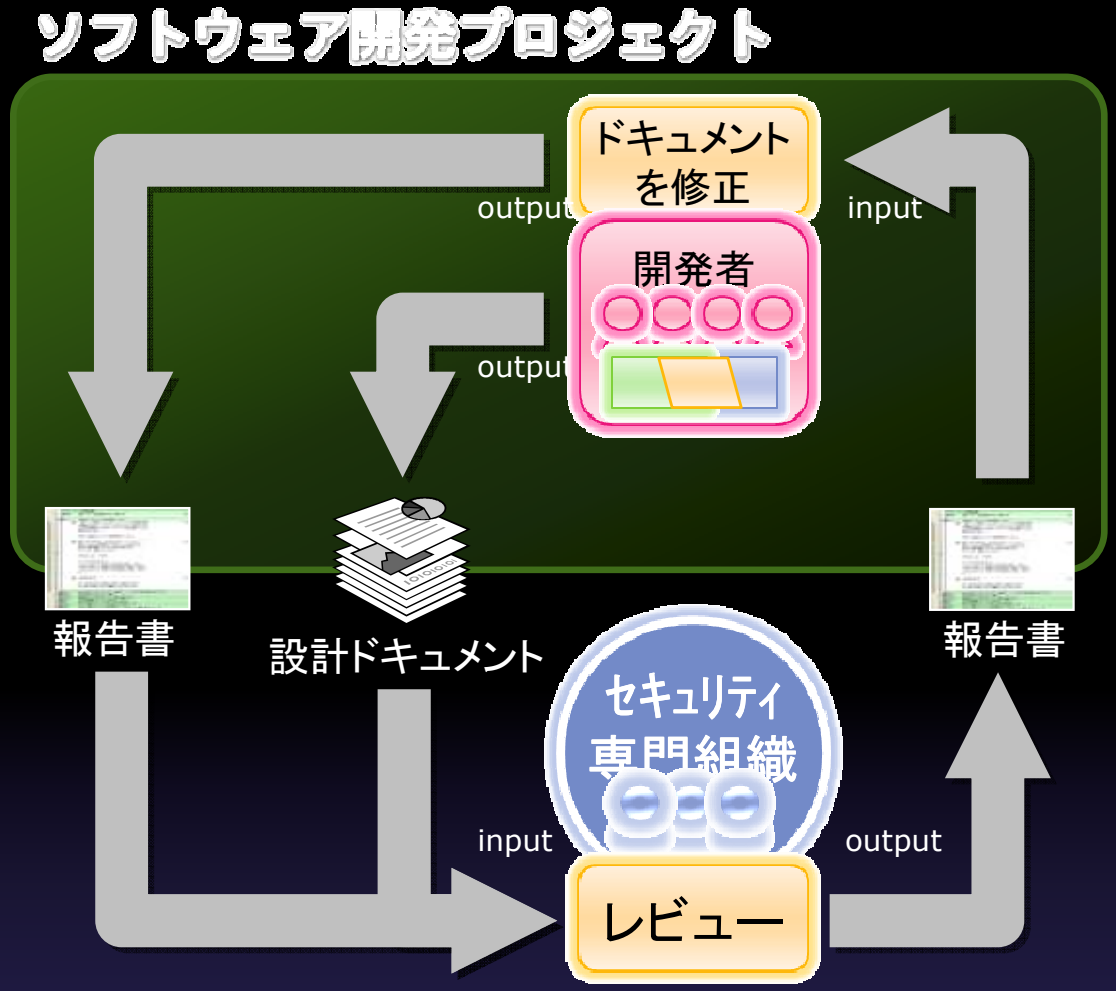


セキュアプログラミングセミナー



- セキュアプログラミングの基礎知識を身につける４時間の座学研修
  - 短時間の研修であるが効果は大きい
  - 例）コードレビューツールの指摘内容を理解できるようになる
- すべてのプログラマに受講が義務付けられている
  - プログラマは大量のソースコードを生産
  - 大量のソースコードの中から脆弱性を見つけ出すのは効率が非常に悪い
  - 最初からセキュアに実装するのが一番効率がよい

セキュリティドキュメントレビュー



- セキュリティ専門組織はセキュリティの観点から設計ドキュメントをレビュー
- 見つかった脆弱性が対策されるまで追跡
- 開発者はセキュリティについてあまり気にしなくてよい（開発に専念）

Microsoft SDLの脅威分析手法を超簡素化した独自のレビュー手法

① 脅威分析表を準備する

目に付いたキーワードや文章など		条件、脅威、質問			分析時コメント
分析対象	分析対象から見つけたキーワード、文章	TH-XX CD-XX Q-XX DFD-XX	だれがどのように困る？ だれが何をすると？ PL/Architectへの質問 Architectに作ってもらいたいDFD	依存する CD-XX 依存する CD-XX 関係する TH-XX, CD-XX 関係する TH-XX, CD-XX	
		TH-01 CD-01 Q-01 DFD-01	脅威の内容 条件の内容 質問の内容 書いて欲しいDFDの内容	CD-01 CD-02 CD-01 TH-01,CD-01	分析時のコメントfor脅威 分析時のコメントfor条件 分析時のコメントfor質問 分析時のコメントforDFD

簡単な書式でドキュメントコストを最小化

すべてのドキュメントの脅威分析結果を1つの脅威分析表に集約してまとめる

② キーワード、文章を見つける

気になるキーワード、文章を脅威分析表に転記する

設計ドキュメント：機能仕様.xls

機能	環境／入力条件(OS条件含む)	出力(期待)結果
	一番目の階層を選択し、検索ボタンを押す	一番目の階層を表示される
	二番目の階層を選択し、検索ボタンを押す	二番目の階層を表示される
	三番目の階層を選択し、検索ボタンを押す	三番目の階層を表示される
	四番目の階層を選択し、検索ボタンを押す	

検索ボタン

SQLインジェクション？

脅威分析表

目に付いたキーワードや文 ど		条件、脅威、質問		
分析対象	分析対象から 出した キーワード 章	TH-XX CD-XX Q-XX DFD-XX	だれがどのように困る？ だれが何をすると？ PL/Architectへの質問 Architectに作ってもらいたいDFD	依存する CD-XX 依存する CD-XX 関係する TH-XX, CD-XX 関係する TH-XX, CD-XX
機能仕様.xls	検索ボタン			

③ 脅威と、脅威の顕在化条件をまとめる

① 攻撃者がやりそうなこと or 製品開発者がやりそうな設計や実装（条件：CD-XX）を連想して記入する。

② 起こって欲しくないこと（脅威：TH-XX）を連想して記入する。

③ 脅威（TH-XX）が顕在化する条件（CD-XX）を記入する。

目に付いたキーワードや文章など		条件、脅威、質問		
分析対象	分析対象から見つけた キーワード、文章	TH-XX CD-XX Q-XX DFD-XX	だれがどのように困る？ だれが何をすると？ PL/Architectへの質問 Architectに作ってもらいたいDFD	依存する CD-XX 依存する CD-XX 関係する TH-XX, CD-XX 関係する TH-XX, CD-XX
機能仕様.xls				
	検索ボタン	CD-03 CD-04 TH-04	ユーザ入力文字列に特殊文字を入れてしまったら SQL文を(プレースホルダでなくて)文字列連結で組み立てていたら SQLインジェクションでDBが破壊されてしまう	 AND(CD-03,CD-04)

1

2

3

脅威ツリーをテキストで表現することでドキュメントコストを最小化

- 連想する順序は任意。脅威、条件の関係を構築できればOK。
  - 詳しくは日経BPソフトプレスの 「脅威モデル」を参照。

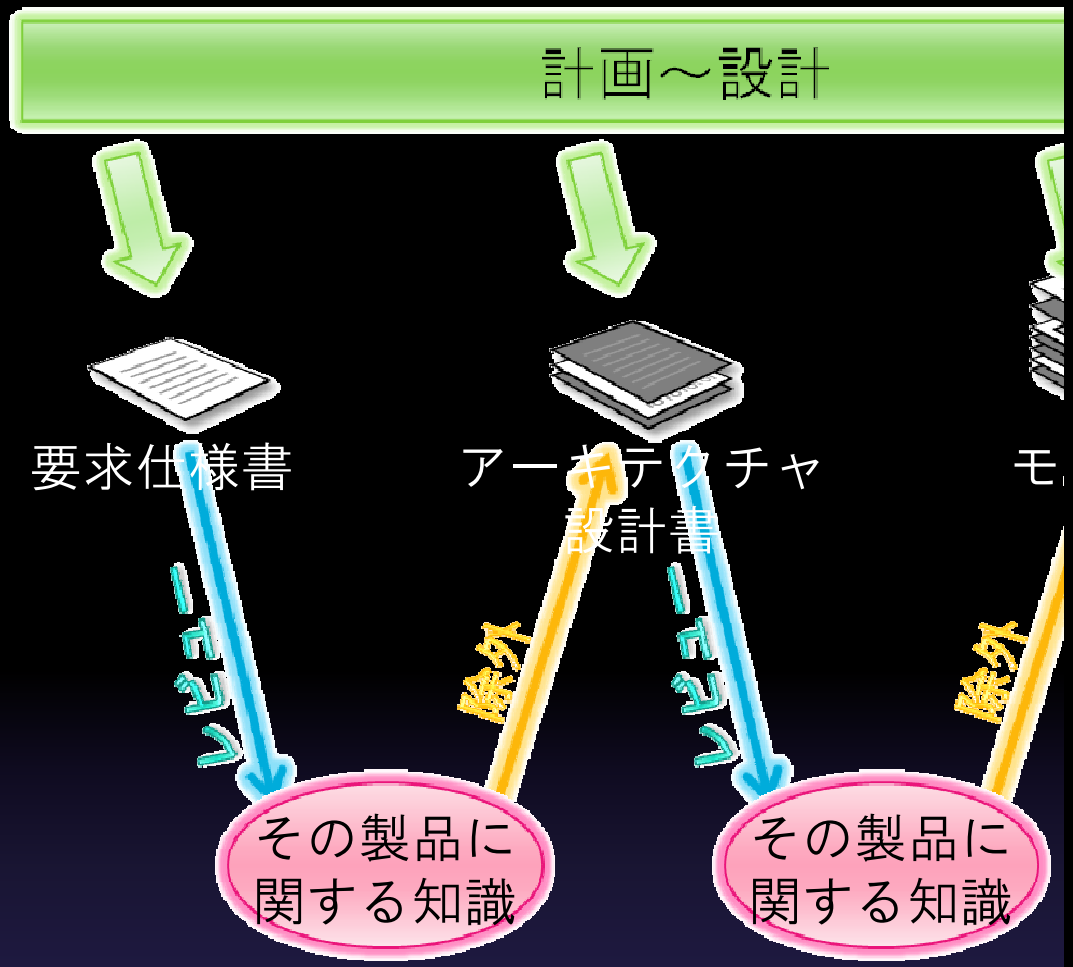
報告書サンプル

TH-05	大量の検索結果が生じるような検索文字列を発行された場合、システムがDoS状態に陥る	
ステート	S39 - 脆弱性なし確認済み	
松並	検索機能において、大量の検索結果が生じるような入力に対する対処を行っていますでしょうか？ たとえば以下のようなケースがあります。 (1) 「あ」など大量にヒットするようなケース (2) LIKE節でワイルドカードになる「%」を入れられてしまうようなケース	11/21
	★対策★ (1)については、一度に生成されるレコード数に制限(LIMIT句？)を設けるなどの対策方法があります。 (2)については、「%」が単なる「%」という文字として扱われるようにエスケープしてSQL文に組み込むなどの対策方法があります。	
飯坂さん	(1)について、今回のアプリケーションでは、マスタデータの検索しかなく、マスタのデータが Dos攻撃に匹敵するほどの量になることは想定していませんが、それでも対応は必要でしょうか。 (2)については、[redacted] では元々ほとんどの検索機能で、LIKE 文を使用しています。単純に「%」が検索文字列として認識されないということが問題ですので、開発側に確認します。	11/21
MTGメモ	(1)はOKです。 (2)は確認していただいて、結果をお知らせください。	
飯坂さん	検索機能において、「%」が入力された場合に、適切にエスケープされて、「%」という文字として使用されることを確認しました。	12/12
渡辺	確認ありがとうございました。よって本件は問題ありません。 クローズ。	01/17

対策方法を提供し、修正コストを最小化

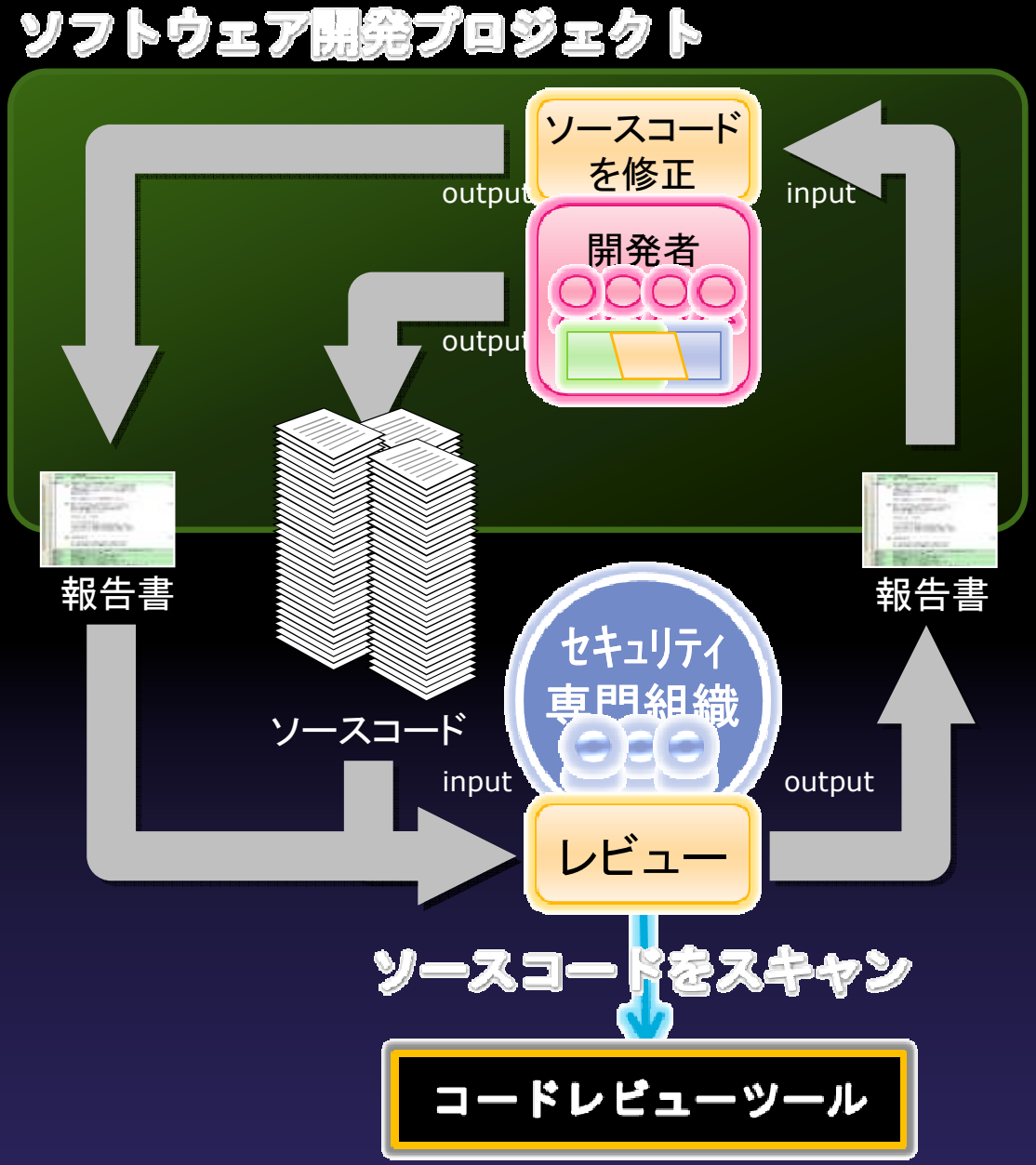
簡単な書式でドキュメントコストを最小化

さらなるドキュメントレビューの効率化



- セキュリティに関係ある記述は
  ドキュメントのほんの一部

セキュリティコードレビュー



- セキュリティ専門組織はコードレビューツールを使ってソースコードをスキャン
- ツールの出力をレビューし、精査したうえで、対策提案を含めた報告書を開発者に渡す
- 見つかった脆弱性が対策されるまで追跡

コードレビューツールの必要性

- 人力コードレビューではまったく不十分
  - 弊社事例：
    約４０万行程度のソースコードに対して、
    ３人のセキュリティレビュアーで、
    約３．５日レビューして、
    たったの４２８０行（全体の１%）しか、
    レビューできなかった
- 人力によるコードレビューにはカバレッジの面で限界がある
  - そこでコードレビューツールの登場です

コードレビューツールの検出精度を測ってみた

これらの脆弱性を仕込んだソースコードをとある2製品でスキャンさせてみた（2006年夏）。

- 演算トラップ
  - ゼロ割　割り算オーバーフロー　SIGFPEの補足による無限ループ
- シグナルハンドラ
  - 非同期シグナルアンセーフ関数の使用　シグナルハンドラからのthrow　シグナルハンドラからの不適切な型の変数の更新　シグナルハンドラ登録とシグナル発生のレース
- スレッドセーフティ
  - 非スレッドセーフ関数の同時使用　スレッドの非同期キャンセル　シグナル送出によるスレッドの終了　スレッドの協調キャンセル　bool変数のロックなしでの使用　関数スコープの静的記憶期間オブジェクトの使用　複数スレッド動作中のフォーク　スレッド共有リソースの変更
- OS環境
  - 環境変数の信用　不適切なumask　コアの生成許可
- プロセス生成
  - PATHの信用　シェルコマンドインジェクション　パストラバーサル
- 一時ファイル
  - 予測可能な一時ファイル名　一時ファイルの消し忘れ
- 乱数
  - 予測可能な乱数
- TOCTOU
  - 既存ファイルの安全なつもりのオープン　新規ファイルの安全なつもりの作成
- グレーなAPI
  - realpathの使用　globの使用　fnmatchの使用　wordexpの使用
- スタック溢れ
  - 過剰再帰　可変長自動配列
- メモリリーク
  - 関数内でのメモリリーク　クラス内でのメモリリーク　例外安全性1
- C++
  - デストラクタでのthrow　type-punning　クラスのメンバのポインタのshallow-copy(→dbl-free)　大域なオブジェクト　未初期化自動変数の読み込み　未初期化自動変数への書き込み
- 整数オーバーフロー
  - real world example
- 文字列
  - デコードによるヌルバイト混入
- well known vuln
  - ダブルフリー　フリー後のアクセス　off-by-one　フォーマット文字列 スタックオーバーフロー (API)　スタックオーバーフロー (配列のインデクス不正)　ヒープオーバーフロー (API)　整数オーバーフロー
- パーサー性能チェック
  - GCC拡張への対応 C99への対応　C++への対応

結果（2006年夏時点）

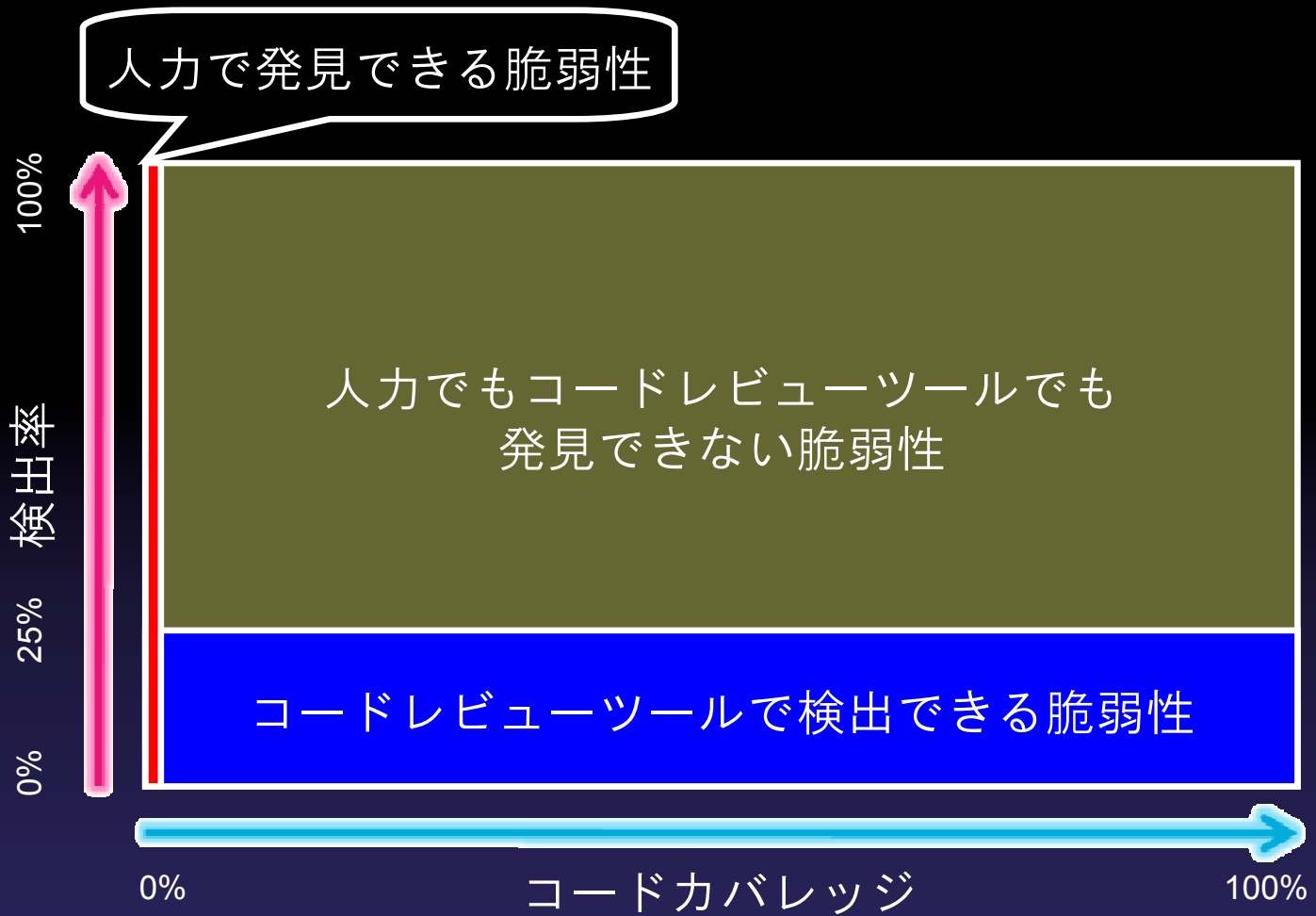
- ２５％の脆弱性を検知
  - ７５％の脆弱性は見つけられなかった

  - 他社同等製品もほぼ同じ２５％の脆弱性を検出
    つまり、これが2006年時点のソースコード解析テクノロジの限界である
    ツールによる明確な能力差は認められなかった

- それでもわたしたちはコードレビューツールを使っています。その訳は・・・

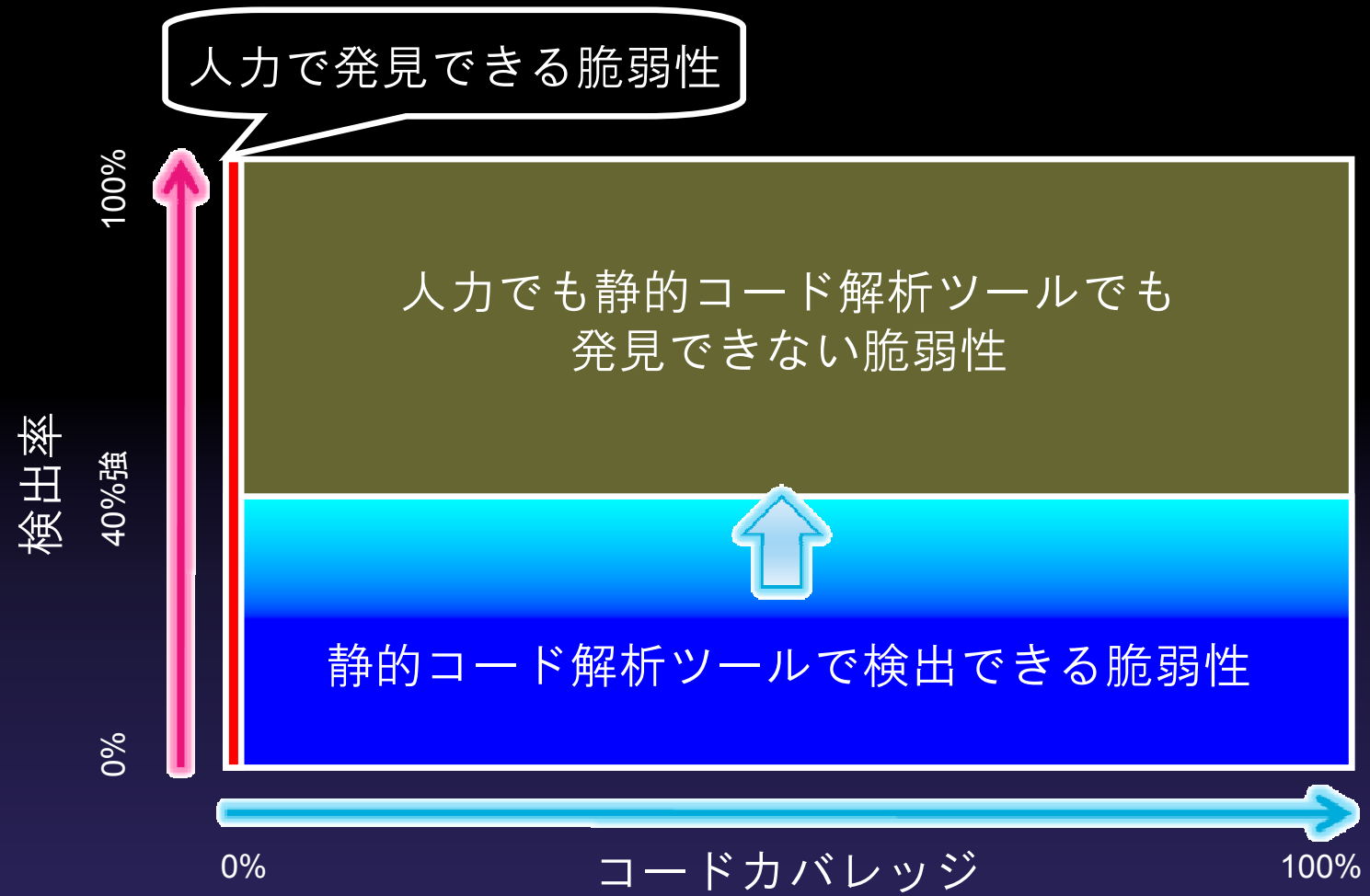
カバレッジが大きな効果を生む

たとえ検出率が低くても、人力が発見できる脆弱性よりもはるかに多くの脆弱性をコードレビューツールは発見できる。



静的コード解析ツールも勝手に日々進化

2008年秋頃に再度計測してみたら、40%強まで検出率が改善されていた。



人力でもコードレビューツールでも発見できない脆弱性はどうするの？

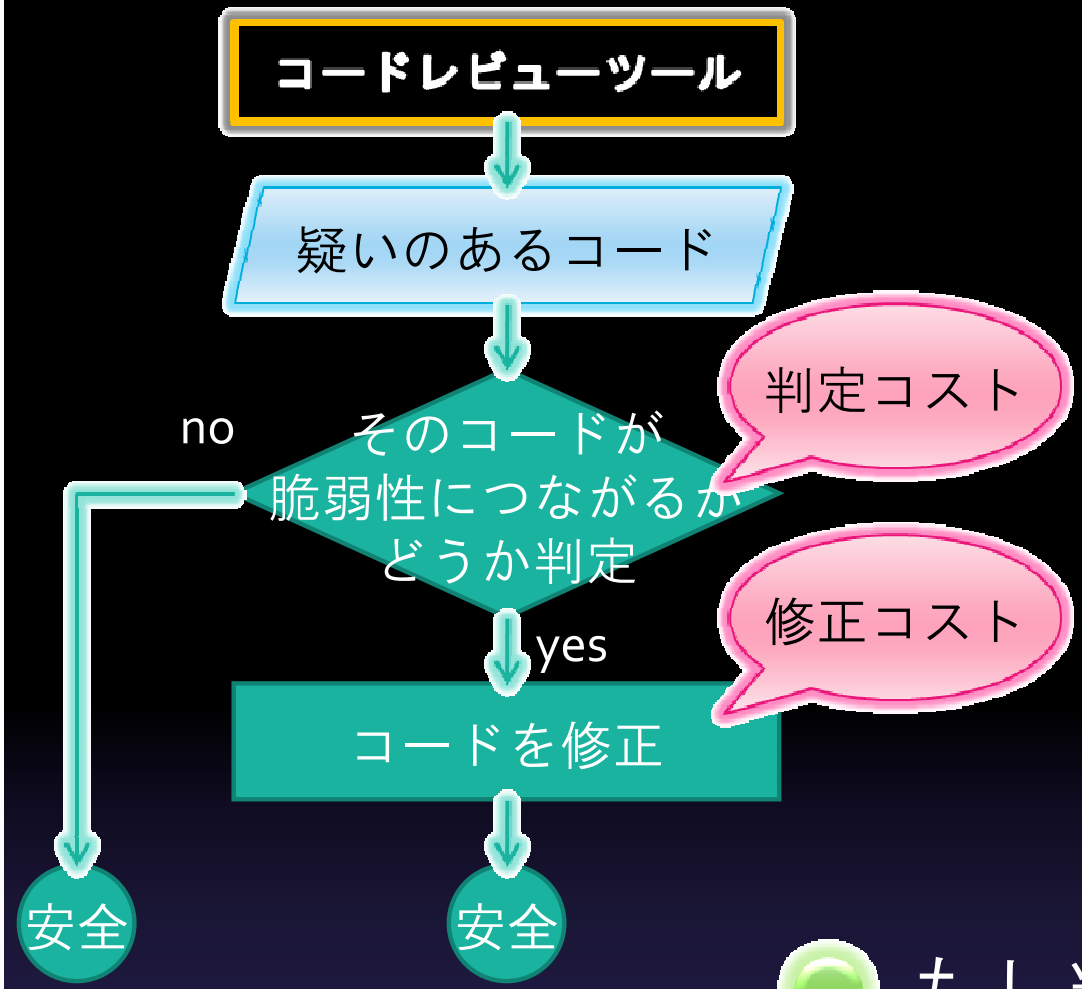
- 見つけられない脆弱性は**リスク受容**
  - 完璧なセキュリティを求めるのではなく、費用対効果の大きいところから攻める（LWSSAの基本）

    人力で全件見つけるのは工数の観点から非現実的

    そもそも人力でも見落とす場合がある

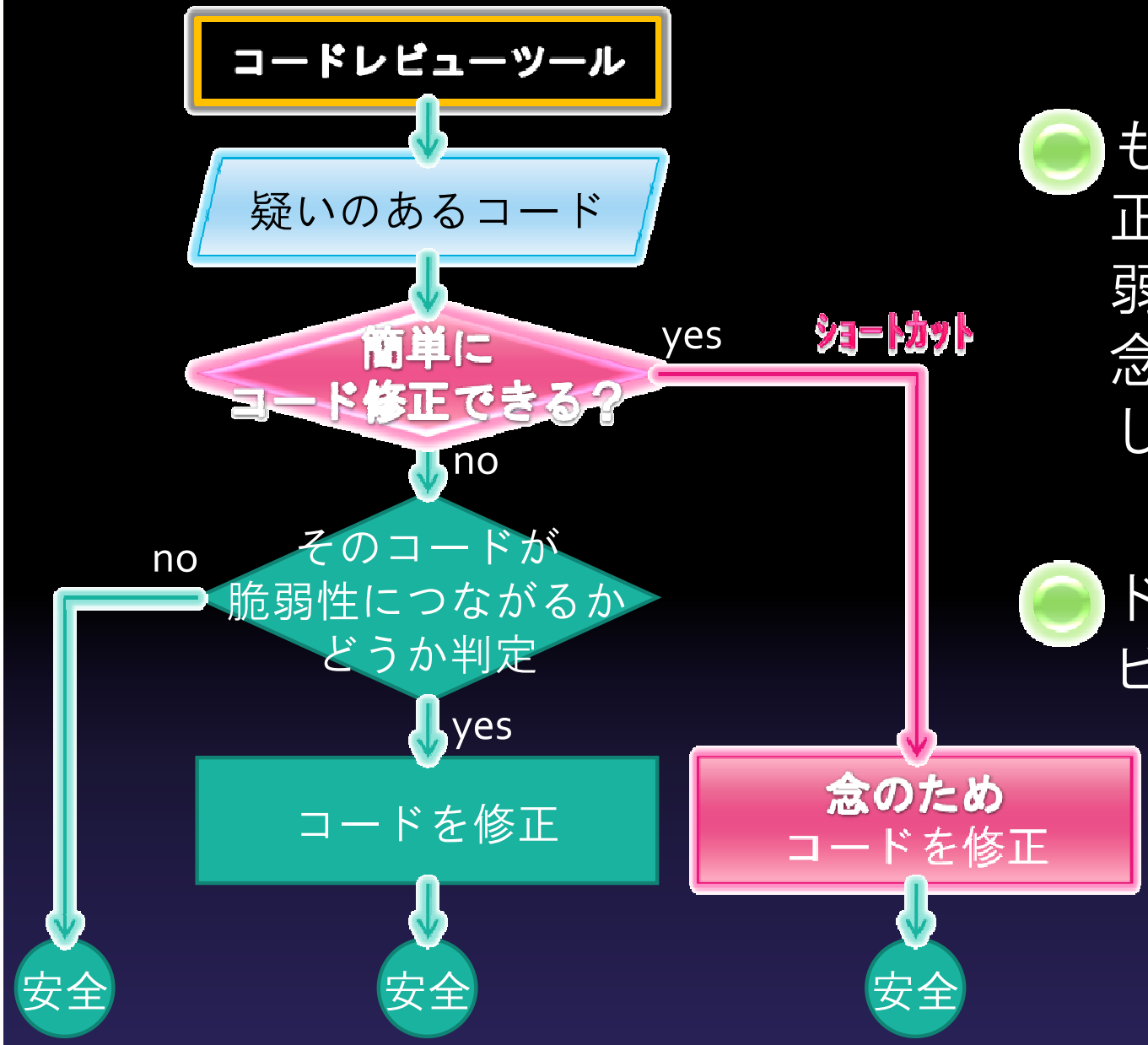
- さらに**その他の施策**でカバー
  - 最初からセキュアに実装するためのセキュアプログラミング教育を実施
  - クリティカルな部分は、ドキュメントレビューの結果からレビュー範囲を絞ったうえで、人力でコードレビュー（※必要な場合だけ実施）

コードレビューの効率化



- もし判定コストが修正コストを上回る場合、この手順は効率が悪い

効率的な手順



- もし簡単にコード修正できる場合は、脆弱性の判定を省略し、念のためコード修正してしまう
- ドキュメントレビューでも同じ工夫

開発者自身による セキュリティの作りこみ

LWSSA 1.0の拡張

LWSSA 1.0 の限界

- 製品数カバレッジの限界（量の限界）
  - セキュリティ専門組織がすべての製品開発プロジェクトをレビューするのは無理がある
  - 常時多数のプロジェクトが走っている
- 技術分野カバレッジの限界（質の限界）
  - セキュリティ専門組織がすべての技術分野の知識を習得するのは無理がある
  - 特に、家電製品は幅広い技術領域の上に成り立っている

    Ex) Blu-ray, DLNA, Bluetooth, HDMI, Wi-Fi, MPEG ...

LWSSA 2.0 方針

製品数カバレッジの限界を超えるために

- 開発者自身にセキュリティレビューを実施してもらう

技術分野カバレッジの限界を超えるために

- 各技術領域の開発者にセキュリティを教え込んでしまう
  - Ex) Blu-ray × Security の専門家を養成

つまり

製品開発におけるすべてのセキュリティ活動を開発者自身に任せてしまう

実際のトレーニングの段取り

セキュリティレビュアートレーニングプログラム

4カ月

最初の６週間

	Mon	Tue	Wed	Thu	Fri
10:00 - 12:00	C/C++コーディング問題の学習				→ﾚﾋﾞｭｰ会
12:00 - 13:00	昼食				
13:00 - 15:00	進捗会議	セキュリティレビューアーツール			
15:00 - 17:00	設計問題の学習				→ﾚﾋﾞｭｰ会

次の４週間

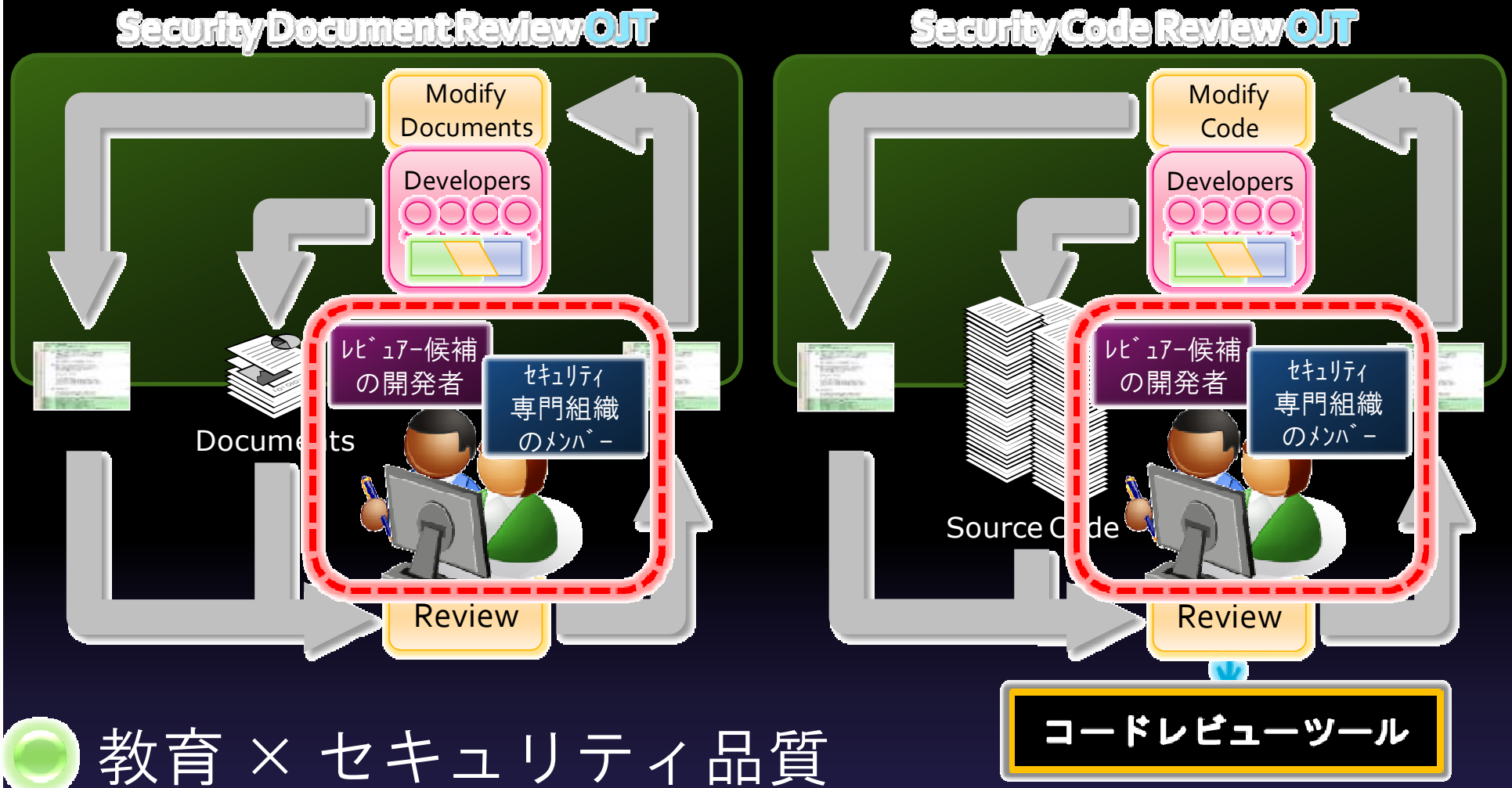
	Mon	Tue	Wed	Thu	Fri
10:00 - 12:00	C/C++ｺｰﾃﾞｨﾝｸﾞ問題学習		C/C++ｺｰﾃﾞｨﾝｸﾞ問題学習		→ﾚﾋﾞｭｰ会
12:00 - 13:00	昼食				
13:00 - 15:00	進捗会議	セキュリティレビューアーツール			
15:00 - 17:00	設計問題の学習				→ﾚﾋﾞｭｰ会

最後の６週間

	Mon	Tue	Wed	Thu	Fri
10:00 - 12:00	設計問題の学習		設計問題の学習		→ﾚﾋﾞｭｰ会
12:00 - 13:00	昼食				
13:00 - 15:00	進捗会議	セキュリティレビューアーツール			
15:00 - 17:00	調査研究				→ﾚﾋﾞｭｰ会

Sony Digital Network Applications, Inc.

セキュリティレビュー On the Job Training



- 教育 × セキュリティ品質
- 実在する製品 → トレーニングが実戦的

さいごに

- 我々が行っているセキュリティに対しての取り組みもまだまだ発展途上です
- また、セキュリティはソフトウェア業界全体として取り組んでいかないといけないと考えております
- 今後も皆様と情報交換していき、安全な製品開発について一緒に取り組めればと思います

- 今回このような場を提供してくださいましてありがとうございました
- 今後も情報交換等、宜しくお願い致します

ご清聴、誠に有難う御座いました

不景気の今こそ、無二の好機

- 好景気でソフトウェア開発の需要が大きかったときの反省
  - 一度つくられたソフトウェアの脆弱性を見つけてコード修正していくことは極めて効率が悪かった
  - 最初からセキュアにつくられていればどんなに楽だったか・・・
- 景気回復したとき、ソフトウェア開発の需要が爆発する
  - そのときは絶対、最初からセキュアにソフトウェアをつくりたい
- ソフトウェア生産が減速した今こそ、次に重点を置く
  1. ツールを使った地道な既存ソフトウェアの脆弱性除去
  2. セキュアプログラミング教育によるセキュアプログラマの育成
  3. セキュリティレビューによる新規開発ソフトウェアの脆弱性除去
  4. セキュリティレビュアーの育成