

15-B-12

セキュア開発手法の考察と診断ツールの活用事例の紹介¹

～お客様に「安心してご利用ください」と言えるための脆弱性対策～

1. 概要

ビッグロブ株式会社（以降、当社）では、接続サービスを始めとして様々なインターネットサービス（クラウドサービス）を展開しており、会員データベースなどの基幹系システムやポータルサイト、スマートフォンアプリまで多種多様なシステムを開発・運営している。また、その開発体制はアウトソース・内製さまであり、お客様の安心・安全を維持するための最適な開発手法を模索・整備し続けている。本編では、その活動を通じて見えてきた脆弱性の仕分け方と診断手法の工夫および診断ツールの活用事例を紹介する。

2. 取り組みの目的

お客様に安心・安全な Web サービスやクラウドサービスを提供するためには、既知の脆弱性を残さぬようセキュアな開発と検証を行わなければならない。しかし、プロジェクトの予算や開発規模も多種多様であり、検査専門会社に委託するなど一回数 10 万～100 万円以上かかる手法で一律化するのは、事業として現実性を損なってしまう。ツールを用いて診断を内製化するにしても検査作業の工数（コスト）は消費するので、一律様な検査方法を義務化する事も現実的には難しい。検証精度の属人化は抑止しつつも、対象システムの規模感・予算感に合わせた検証法を選択できるようにしなければならない。

2.1. 解決すべき課題

サービスを常に「既知の脆弱性が一切無い状態」でリリースするのが理想ではあるが、世の脆弱性は文字通り無数に存在し、リスクの捉え方にも様々な考え方がある。そのため、100%の対処を常に続けるのは現実的には不可能でもある。それでも、リリース後に脆弱性が発見された際に、お客様から「不適切な検証方法や判断だったのではないか」と言われぬだけの検証は必要である。従って、たとえ低コスト・短納期な開発計画でも相応の検証と対処ができるよう、適切な制度やツールの整備が必要となる。これらの事情を踏まえて、以下を本質的課題とした。

- (1) 属人性の少ないセキュア開発手法を整備・展開する。

¹ 事例提供: ビッグロブ株式会社 クラウド・スマートサービス事業部 橋田 幸浩 氏

(2) 対象システムの規模や予算に合った検証方法を選択可能にする。

2.2. 目標

一律一様の検証方法を義務づけることができない以上は、開発チームが「このプロジェクトでは、どの工程でどのような検証方法を採用することが最適なのか」を自ら誤解なく選定できる必要がある。そこで、検証ツール毎の守備範囲や開発プロセス毎への適正を分かりやすく可視化し、「開発者自身がセルフサービスで脆弱性の検証方法を適切に選別・実行できるようにする」ことを目標とした。

3. 取り組みの対象

2005 年から積み重ねてきた自社サービスのセキュリティ検査（20 件／四半期～最大で約 200 件／年）を通して、脆弱性リスクの見極め所や開発手法に合わせた最適な検証手法を整理した。対象サービスの中には当社の自主事業だけでなく、受託開発案件も含む。

情報を整理するツールや検証サービスは、自社で利用あるいは商材化している物、およびその候補となった物を対象とした。

4. 取り組みの実施

4.1. 「この世に存在する脆弱性と、対処が必要な脆弱性」の考え方の整理

まず、脆弱性をどこまで検証・対処すればお客様に「安心してご利用ください、と自信を持って言えるのか」を見極めるために、脆弱性に対する考え方を単純化した（図 15-B-12-1）。

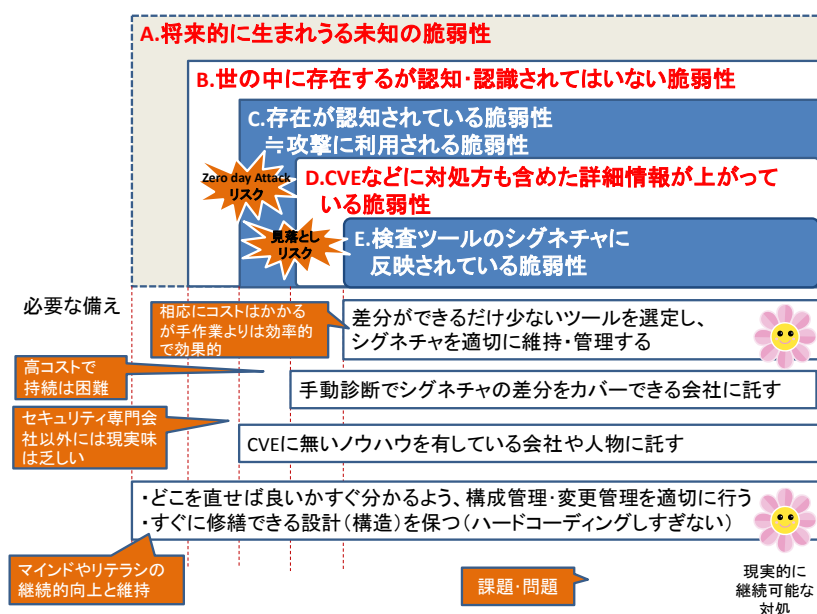


図 15-B-12-1 脆弱性の概念図

時折お客様からは「未知の脆弱性にも対応していること」という要件が定形文的に示されることがあるが、A（将来的に生まれうる未知の脆弱性）にまで予め対処することはそもそも不可能である。また、B（世の中に存在するが認知・認識されていない脆弱性）は、実際の攻撃に使われることも無く対処法も存在しないことになるので、これも対応は不要かつ不可能である。従って、AとBの領域については「新たな脆弱性が認知された場合は、それに追従できる仕組みであること」が適切な要件と言える。

実害が生じ得るのはC（存在が認知されている脆弱性≒攻撃に利用される脆弱性）よりも内側の領域である。CとD（CVE²などに対処法も含めた詳細情報が上がっている脆弱性）の差分が zero day attack のリスクとなるわけだが、この差分を実際に認識できるのは攻撃者自身か情報セキュリティの学者（研究者）くらいであろう。セキュリティ事業者は正にこの領域を生業としているわけだが、すべての事業者に学者や専門業者と同等のレベルの保持を求めることは、やはり現実的とは言えないだろう。以上のことから、一般の事業者が自主的に対処すべき脆弱性はDよりも内側の領域であり、Dの範囲を適切にカバーできればお客様に自信を持ってご提供することができると判断した。

4.2. リスク分類法の見直し

4.1 で述べたDよりも内側のリスクにすべて対処できれば理想だが、予算や納期の都合もあるので現実にはそうはいかない。また、一般的なリスク分類法として表 15-B-12-1 に示す「回避の容易性」と「実害の大きさ」による4象限分類があるが、「被害を受ける可能性があるユーザの数が少ない」ということは「実害が小さい」ということと同義では無い。

表 15-B-12-1 一般的なリスク分類法

影響度	大	1	3
	小	2	4
		容易	困難
		回避性	

※通常、1→3→2→4の順に対処される

なぜならば、被害を受けたユーザにとっては、同じ被害を受けた人が100万人いようが自分一人であろうが自分の受けた損害に変わり無く、企業側が「対処すべき事なのに利益優先のために対処しなかった」とすれば「不適切な判断」としか評価していただけないからである。そこで、脆弱性リスクを表 15-B-12-2 のように分類した。

² Common Vulnerabilities and Exposures

表 15-B-12-2 対処の要否を判断するためのリスク分類法

		実際に攻撃を受けるリスクが...	
		事実上、ない	ある
		(例) 攻撃を成立させるためには中間者攻撃(予めroot権限やネットワーク機器の操作権を奪取する等)の必要がある等の前提条件が多く、それらの対処はできている脆弱性	(例) エンドユーザと同じ立場でどこからでも攻撃できる脆弱性(SQLインジェクションやXSS、Strutsやbashの脆弱性など)
技術的回避策 対応策が?	事実上、ない	a. 対処が不要と言える脆弱性	c. 対処の必要性和現実的可否、方法論を考えるべき脆弱性 ↓ 判断を誤ると「不適切な判断」と評されるリスク
	ある	b. 対処の必要性を考えるべき脆弱性 ↓ 判断を誤ると「不適切な判断」と評されるリスク	d. 対処が必要な脆弱性 ↓ 対処せずにいると「不適切な判断」と評されるリスク

a.に分類できる事象であれば、本質的に対処は不要と言え、対処しなかったとしても後日「不適切な判断」と評される可能性も小さいと言える。

b.に分類された場合は、費用対効果や対外的な品質アピールをどうするかに併せて判断すれば良いが、対処不要と判断した場合はリスクが事実上無いことを適切にユーザへ説明できる必要がある。

c.の場合、基本的には対処が不要あるいは不可能なリスクとも言えるのだが、予想される実害の度合いによっては事業継続可否の判断が求められる。例えば、プロバイダ側による回避が不可能なc.のリスクとしてWindows XPやInternet Explorer ver.6の保守切れのようなユーザ端末側の問題がある。保守の切れたOSやアプリケーションの脆弱性の責任まではサービスプロバイダ側には及ばないが、プロバイダとしては保守が終了したレガシーなOSやブラウザでなければ動かないようなサービスは漸次改善・縮小してユーザの環境更新を促していく等の啓発活動が望まれるだろう。なお、「対処のしようが無いならば致命的な脆弱性であっても放置が許される」と誤解もされやすいので、指針を示す際は注意が必要である。

d.に分類されながら「収益上の理由」だけで対処を遅らせ、その結果ユーザに実害が及んだとなれば、ユーザからは「不適切な判断」と評されることになる。

以上のことから、迅速に対処すべき順序としてはd.→b.→c.→a.となる。この順序は、表15-B-12-1の分類法による優先順位とは異なる場合があるので注意が必要である。また、時間の経過とともにa.、c.の脆弱性もb.、c.に変化するのが常なので、分類も世の中の情勢に合わせて随時見直し続ける必要がある。過去の脆弱性情報が各々どのカテゴリに分類されるかの説明は本編では省略するが、事業部門に判断を任せてしまうとa.、b.に判断が偏りやすいので、実際にはCSIRT³などの組織が事象を適切に分析・判断し、社内へ指針を示すこと

³ Computer Security Incident Response Team 自社内でセキュリティ上の問題が起きていないかどうかを監視するとともに、万が一問題が発生した場合にその原因解析や影響範囲の調査、あるいは対応の取りまとめを実施・支援する組織の総称

が望ましい。

4.3. 診断ツール毎の特性（守備範囲）の整理

次に、脆弱性の作り込みや放置を予防するにあたり、世の中に存在する品質検証ツールの守備範囲と検証精度を整理した（表 15-B-12-3～表 15-B-12-6 および図 15-B-12-2）。なぜならば、何らかの診断ツールを導入すると必ずと言って良いほど「このツールで診断しさえすれば、他は何もしなくて良い」という誤解が広まりやすかったからである。

表 15-B-12-3 は、診断要素別に手法やツールを整理した物である。脆弱性を検証するツールは、基本的にアプリケーション診断（以降、AP 診断）、プラットフォーム診断（以降、PF 診断と記す）、ソースコード診断（以降、ソース診断）の 3 種に分類される。AP 診断は動的診断（DAST: Dynamic Application Security Testing）と静的診断（SAST: Static Application Security Testing）とに分けるのが現在の主流ではあるが、非技術者や非 IT 系のお客様企業には AP 診断・ソース診断の表記で紹介した方が直感的に理解しやすいようなので、このように表している。

表 15-B-12-3 必要な診断対象の分類

- ポイント

- ツール毎に「意味」が異なる。
- どれか一つで検査すれば十分という物は無い。

種別	検査対象	検査手法	注意事項	主なツール
アプリケーション (AP)層診断	アプリケーションの作り として攻撃の余地がな いかを検査する	実際に攻撃 する	検査対象の品質が悪いと、 対象システムを破壊する 可能性もある。	AppScan Standard(IBM) WebInspect(HP)
プラットフォーム (PF)層診断	OSやミドルウェアの バージョンや設定に脆 弱性が無いかを検査す る		XSSやインジェクション、 セッションハイジャック等 のアプリ層の脆弱性は 分からない。	Retina Nessus
ソースコード 診断	ソースコードの作法とし て既知の脆弱な書き方 が無いかを検査する パラメータがサニタイズ されているかどうかを 構造的に検査する	ソースファ イルを全文検 索・解析して いく	コードの作法としては 完全なつもりでも、 アプリの実装として 攻撃ができる可能性は ある。 あくまで修正作業の効率化 /Webアプリ診断がどうし てもできない際の次善策 という位置付け	Fortify360(HP) AppScan Source(IBM)

表 15-B-12-4 は、ツールやサービス毎に「何を検証・防護するのか」と「どのプロセス(工程)で使える物なのか」を整理した表である。

表 15-B-12-5 は、脆弱性の脅威に対し何を目的とするのかを軸にして整理した表である。往々にして検証と防護と監視が混同されるケースや、IPS⁴ や Firewall さえ使っていれば他の防護措置は不要だと誤解されているケースが見られたため、ツール毎の主目的の違いを明確にするために作成した。なお、IPS/IDS⁵ は本来防護を目的としたツールでもあるが、当

⁴ Intrusion Prevention System

⁵ Intrusion Detection System

社での使い方の特徴上△(条件・制約付きで使えなくもない)にしている。この整理によって、自社内においては、被害を予防するためのツールは抜け漏れも無いが重複も多く、ダメージコントロール機能はまだまだ人的運用に頼らざるを得ない状態になっていることが浮き彫りになり、今後の投資戦略が考えやすくなった。

表 15-B-12-4 検証要素毎の適用ツール

◎:最適、○:適している、△:条件・制約付きで使えなくもない ×:使えない

ツール名	検証・防護対象			適用プロセス			
	コード	AP層	PF層	リリース前		リリース後	
				コード検証	システム診断	検知・防護	
FortifySCA,AppScan Source	◎	×	×	◎	×	×	×
AppScan Standard, WebInspect	×	◎	×	×	◎	△	×
Nessus, Retina	×	×	◎	×	◎	○	×
Androidアプリ診断	◎	△	×	◎	×	○	×
ワンショット型診断	△	◎	◎	×	○	◎	×
クラウド型診断(SCT Secure)	×	○	○	×	△	◎	×
クラウド型改ざん検知(gred)	×	×	×	×	×	×	◎
IPS/IDS	×	×	○	×	×	×	◎
Firewall	×	×	○	×	×	×	◎
WAF	SiteShell	×	○	?	×	×	◎
	Scutum, WAPPLES	×	○	△	×	×	◎
e-mining (情報流出監視)	×	×	×	×	×	×	○

表 15-B-12-5 目的別での適用ツール

○:適している、△:条件・制約付きで使えなくもない ×:使えない -:対象外

ツール名	被害の予防 (攻撃の余地を 最小化する)	被害の防護 (攻撃を直接 的に防護し、 実害を減らす)	被害の検知 (被害の早期 発見と最小化 を図る)	ダメージ コントロール (被害状況の 把握と拡大 防止を図る)
Fortify360, AppScan Source	○	×	×	×
AppScan Standard, WebInspect	○	×	×	×
Nessus, Retina	○	×	×	×
Androidアプリ診断	○	×	×	×
ワンショット型診断	○	×	×	×
クラウド型診断(SCT Secure)	△	○	×	×
クラウド型改ざん検知(gred)	○	×	○	×
IPS/IDS	△	△	○	×
Firewall	○	○	△	×
WAF	△	○	△	×
e-mining(情報流出監視)	×	×	△	○

表 15-B-12-6 は、ツールを適用すべきプロセスをさらに分解してマッピングした物である。AppScan や負荷テストで代行検査を依頼する際、完成度が不十分でテスト時に正常動作せず何度もやり直しとなるケースが多かったため、「この検証法は、この段階まで進んだシス

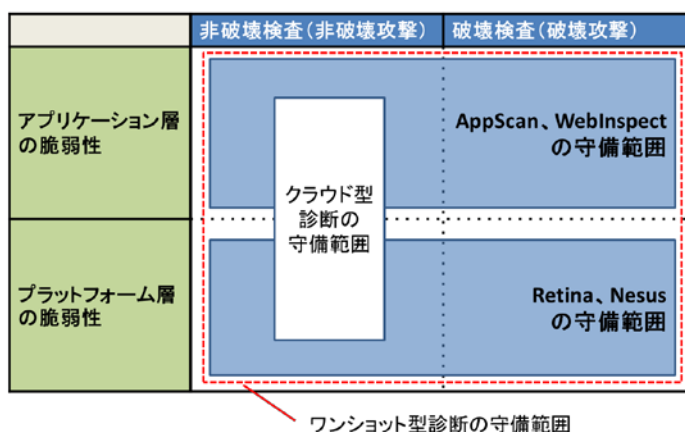
テムでないと適用しづらい」ということを分かりやすくするために作成した。この整理によって、自社内において完成後の検証とリリース後の監視・検査については十分なメニューがあるが、開発途中（CD/FT 段階）での検証を支援するためのメニューが不足していることも明確化できた。同時に、お客様から脆弱性の検証についてご相談があった際にどの工程でお困りなのかを伺うことで、どのツールやサービスが最も適しているかを考えやすくなった。

表 15-B-12-6 プロセス別の適用推奨ツール

◎:最適、○:適している、△:条件・制約付きで使えなくもない ×:使えない

サービス名	リリース前			リリース後		備考
	Coding中	FT/ST	出荷前検査	常時監視	定期検査	
FortifySCA, AppScan Souce	◎	△	○	×	×	Webの無いサービスでは出荷前検査(出荷判定)に用いるのも可
AppScan Standard, WebInspect	×	○	◎	×	○	
Nessus, Retina	×	×	◎	×	◎	
Androidアプリ診断	×	×	◎	×	◎	
ワンショット型診断	×	×	◎	×	◎	
クラウド型診断(SCT Secure)	×	×	△	◎	○	評価系に適用することも出来るが制約あり。
クラウド型改ざん検知(gred)	×	×	×	◎	○	汚染/改ざんの有無の監視のみ。行為自体は監視できない。
IPS/IDS	×	×	×	◎	×	
Firewall	×	×	×	◎	×	
WAF	×	×	×	◎	×	《不正なアクセス》の監視のみで、脆弱性の監視ではない
e-mining	×	×	×	○	×	情報流出の有無の監視

図 15-B-12-2 は、ワンショット型診断とクラウド型診断との検査精度(強度)の違いを分かりやすく示すために作成したモデルである。



※手作業による診断＝表中の隙間を埋めるか、更に限定的な範囲の検証を行う場合の手法

図 15-B-12-2 診断ツール（サービス）の検出精度モデル分類

「クラウド型診断を適用すればリリース前検査はしなくても良い」という誤解が広まりかけてしまったので、それを防ぐために作成した。また、「ツールによる検証は必ず不足があるから買うだけ無駄。手作業検査でないと駄目だ」という意見が社内外から何度も出たため、

実際の守備範囲の違いを分かりやすく示すためにも用いている。高品質な検証ツールは自動で数万通りあるいはそれ以上の攻撃手法を試してくれるが、手作業でそれと同じだけのパターンを試すのは現実的に不可能である。手作業による検査はあくまでツールによる診断結果の検証や、検査パターン(シグネチャ)に反映されていない最新の脆弱性をピンポイントに検証するための手法と考えるべきである。

当社で最初に採用したツールは PF 層診断ツールで、順次 AP 層診断ツール、ソースコード診断ツールと強化していった。共通ツールとして採用したのはそれぞれ Nessus、IBM Rational AppScan Standard、HP Fortify SCA（当時の名称。現在は Fortify360）であり、要件やプロジェクトの予算等に応じて社外の診断サービスや他の診断ソフトを用いる事も可、という前提にした。重要度の高いサービスでは複数の手法を組み合わせることで多重チェックすることもある。

4.4. サービスの開発手法と推奨される検証手法の分析と分類

ツールやサービス類の特徴・特性の分類と併せて、実際にサービスが作られている様子や開発手法(フレームワーク)も観察し、必要な脆弱性検証の方法やその必要性を分析した。その結果が表 15-B-12-7 である。

表 15-B-12-7 サービス開発手法のセキュリティレベル分類

レベル	概要	例
5	個別の開発者(制作者)が各々でセキュリティを意識する必要がない仕組みで作られている	ECサイトやCMSで所定の商品データやコンテンツを追加するだけ
4	各開発者は必要最低限のセキュリティ(サニタイズとバリデーション)さえ押さえておけば、簡単にセキュリティが確保できる仕組みで作られている	既存のAPIのパラメータを変更するだけ
3	開発者のスキルやセンスに依存しない手法で、開発者自身が定量的にチェックするプロセスが働いている	新規サービスの開発中～大規模な案件 検査ツールは主にここで活用
2	専門チームが検査を代行して、一定の品質を保証できるようにする	
1	開発チームにて個別にチェックを実施。もし内外からの指摘や定期診断で問題が見つかれば、迅速に対処する	htmlやスタイルシートの制作、簡単なスクリプトの制作のみ等
0	セキュリティが全く考慮されていない(対処できる体制になっていない)	リリース不許可
-1	セキュリティ対策はお客様マターで、自社では検査を実施できない(制度化できない)	クラウドホスティングやハウジングで構築されたお客様のシステム

※出典: 2011.11.25AppScanユーザ会パネリスト資料

当社では、会員管理や課金系等の基幹系システムは個々のサービス開発者が自由にデータを操作できるアプリケーションを作り込める仕組みではなく、品質が担保された所定の API を経由した処理しか行えないようにしている。従って、レベル 4～5 が担保されていると言

える。故に、簡単なサービスであれば都度セキュリティ診断を行う必要がない仕組みが実現できている。

個別の作り込みが必要な案件については、概要設計の段階で識者によるアセスメント（設計審査）を行うことで、レベル 0 になりそうな案件やレベル 3 以下でも検証の度合いが足りないと思われる案件に対して指摘・改善指導をしている。

AppScan や Nessus のような自動診断ツールは、すべてのプロジェクトに対して適用できれば確かに理想的でもあるが、相応に工数も消費するので、レベル 4 や 5 のプロジェクトに対してまで闇雲に利用させる必要は無い。「担当者が個々に考える必要をなくす」ためにツールの適用を義務化させるより、レベル 2、3 のプロジェクトで診断ツールを活用するように働きかけつつ、レベル 1 の案件がレベル 0 に滑り落ちないように監督・指導することに注力する方が、個々人だけでなく組織全体のスキル向上のためには効果的と考える。

5. 取り組みの結果

5.1. ツールによる検証結果の考察

ツール毎の特性を前述のように分類して実際の案件の開発手法や検証状況を観察した結果、以下の 3 点に気づいた。

- (1) コード規約やガイドラインを見かけ上は満たしていても、攻撃の余地が残っている場合はよくある。
- (2) コード規約やガイドラインを完全には満たしていなくとも、攻撃の余地が無い状態を作れていることは多い。
- (3) AppScan や Retina のような著名な有償パッケージで検証すれば、4.2 の d、b に該当する脆弱性はほぼ確実に潰せている。

サービスとして真に満たすべき目標は「コード規約などのガイド類を守ること」や「あらゆるリスクを完全に排除すること」ではなく、「脆弱性によって生じる実害のリスクを排除あるいは最小にすること」なので、図 15-B-12-3 に示す考え方に基づいて当社では AP 診断と PF 診断で安全性が証明できれば良いと判断し、Fortify は残念ながら予算等の都合もあり契約を解除した。現在は、IPA のテクニカルウォッチ⁶などを参考にしつつ、プロジェクト毎に個別に工夫している状態である。

⁶ IPA テクニカルウォッチ「ウェブサイトにおける脆弱性検査手法の紹介(ソースコード検査編)」
～オープンソースのソースコード脆弱性検査ツールを用いた検査手法の紹介～
<http://www.ipa.go.jp/files/000036973.pdf>

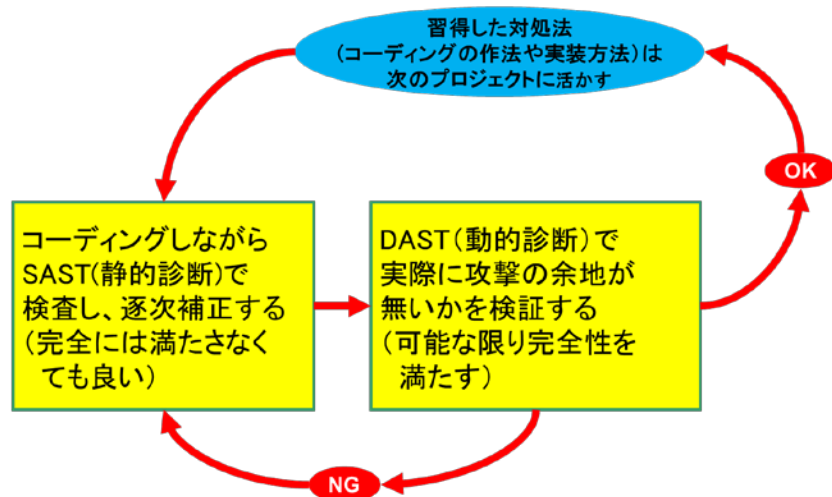


図 15-B-12-3 AP 診断ツールの最も効果的・効率的な使い方

5.2. 見えてきた検証時の注意点とコツの展開方法

AppScan のような動的検証ツールを展開するに当たっては、利用者へ適切な使い方を教育することが重要である。それには、単に「誰でも分かる簡単なマニュアル」を作ることでは解決できず、むしろ誤解を広めやすい場合もあることが分かった。

開発者自身にセルフサービスで適切に利用させるためにマニュアルを作成し注意事項なども示していたが、多くの利用者が「入力フォームのあるページを表示させてテスト実行ボタンを押せば良い」と思い込むか、「評価系とはいえ、DB への書込まで検証すると事後のデータ消去（クレンジング）が大変だから、確認画面までの遷移（寸止め）で止めておきたい」と思い、正しい検証ができていないケースが見られた。

動的検証ツールは、ブラウザに表示されたページそのものを解析するというより、ページとページ間の処理(遷移)において攻撃命令を仕込む物なので、実際は入力→書込→結果表示のすべての遷移を検証する必要がある（図 15-B-12-4）。また、ブラウザの戻るボタンの操作などによる二度書き処理・セッションハイジャックのリスクなどもありうるため、書込処理のコミット後の遷移(「スタートページへ戻る」あるいは「ログアウト」など)までも含めて検証することが理想的である。予算等の都合で本番環境と評価環境を完全には分離できず「寸止め」しかできないという場合は、最後のコミット処理やログオフの処理に穴が残らないよう留意しておく必要がある。

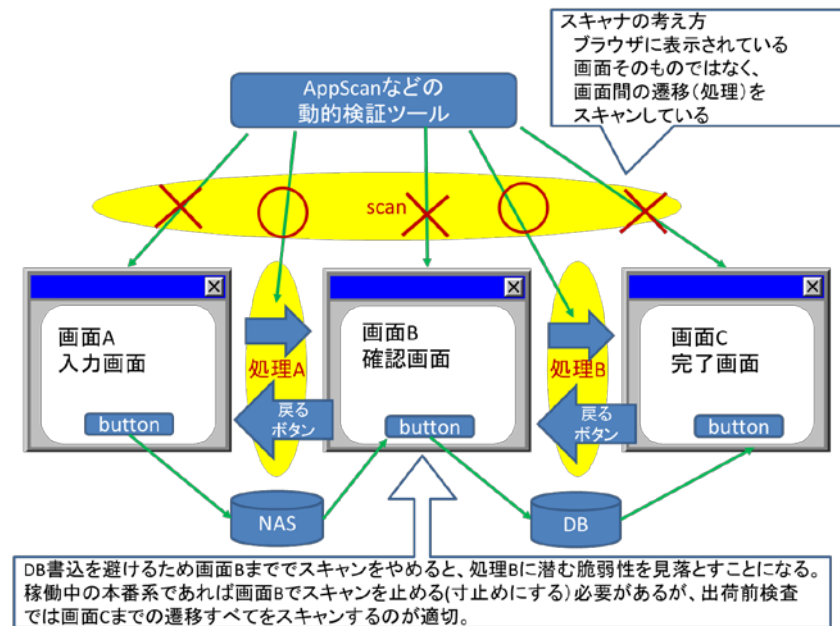


図 15-B-12-4 脆弱性検証ツールが検査する要素

こうした事柄を細かくマニュアルに反映させても、冊子が分厚くなって逆に読む気を失わせ、斜め読みと思い込みでテストを開始してしまうので、マニュアルの強化よりもハンズオンセミナーなどの体験活動に注力するようにしている。

AppScan の導入当初は専門の担当者に検証のみを代行させることでこうした勘違いを予防してきたが、開発者が AppScan による解析結果の読み解き方に慣れるに併せて徐々に開発者自身にツールを利用させるようシフトさせてきた。現在では AppScan による検証の 8 割以上が開発者自身によるセルフサービスとなっている。

5.3. ツールによる検証結果を分析する際の注意点

ツールによって検知された問題点を解析するにあたり、最も多い誤解は「危険度のレベルが低い警告は無視しても良い」という思い込みであった。低レベルの警告は結果として無視して良いことは確かに多いが、「無視して良い事柄」ならばそもそも警告する必要自体がないので、必ず一度は全件を確認することが重要である。例えば、「ある URL のサービスや特定のポートが空いている」という事が検知された場合、ツールでは「情報」というレベルで警告を出すことが多い。しかし、開発者が意図していないサービスやポートが知らぬ間に開いていたならば、単なる設定間違いだけでなくバックドアを仕込まれている危険性もあり得るので、見落としは禁物である。「レベルの高いものだけをチェックする」のはあくまで「時間や工数が限られている場合の優先順位としてやむを得ない措置」あるいは「低レベルの警告は過去に確認済みの場合のやり方」である。

5.4. 開発者のスキルアップと修正効率の向上

診断ツールの活用に際しては、開発者がツールの使い方に慣れるほど、修正のための手戻り工数も減っていった。筆者が直轄したプロジェクトにおいては、最初は十数件の脆弱性修正に際しても一週間から半月の修正期間を要していた SE が、3 回ほど経験を積んだ後は 100 件を超える問題が検出されても 0.5～3 日程度で修正できるようになった。それ以外のプロジェクトでも、5～6 回（四半期～半年ほど）の経験を積み同様の修正が可能になり、検査ツール導入から 3 年ほど経った後は 3 日以内あるいは即日中に再検査を依頼されることが殆どになった。

5.5. テスト工程のコストと期間の圧縮

AppScan による検証は、スタート当初は熟練者による代行サービスを中心としていたが、徐々に開発者自身によるセルフサービス化を進め、ピーク時には四半期あたりで 600 万円ほどかかっていた代行コストが、現在は 1/4（150 万円四半期）以下まで削減できている。また、代行検査にはテストパターンの事前確認などで毎回数日～一週間のリードタイムを要していたが、セルフサービスであればスキャン用端末さえ空いていれば即日テストが実施可能な状態となった。

また、AppScan のような動的診断ツールが使えるということは、そのシステムは正常に動作し、異常な操作にも耐えられる、ということの意味する。さらに、ツールは同時に複数のセッションを走らせることでテストの効率化が図られているので、あくまで簡易ではあるが性能試験も兼ねている、と考えることもできる。この特性を踏まえて、筆者のプロジェクトで試みに「セキュア開発手法」に拘ってみる、という前提でチームを立ち上げ、システム構造設計にも注意しつつ、その上で正常系試験と異常系試験とセキュリティ診断とを 1 プロセスに省略した。結果、リリース後もテスト不足による問題は起こらず、検証工程のコストを 0.5 人月から 3 人日（約 1/3）に圧縮することができた。負荷テストだけは必要な最大負荷が AppScan では出せないことが明白だったので別途実施したが、AppScan で相応の性能が出せていることを事前に検証できたので、「想定外の性能が全く出ず、負荷テストが完遂できない」という手戻りのリスクを避けられた。

5.6. 運用コストの抑止

前述した筆者のプロジェクトでは総計で 324 台のサーバ（うち、24 台が基盤側、300 台がクライアント側）で構成されるシステムを構築し、セキュア開発手法に基づき全ての機能について input/output 制御の厳密性を高めるように設計した。その結果、プラットフォームの未知バグ（新規バグ）やパッチの影響によって想定外の異常が起こる回数も極少となり、仮に不具合が発生しても原因箇所が簡単に特定できるようになった。アプリケーションの保守・運営には四半期あたりで延べ 1 人月前後のコストで対応できており、これは当社においては中級エンジニアが 1 営業日あたり 1 時間で全サーバ（324 台）のアプリケーション保守が行えている計算になる。数万本単位で出荷されるパッケージウェアと比べれば遙かに工数

はかかっているが、完全スクラッチ開発の独自システムとしては極めて少ない保守工数と考えている。リリース前のセキュリティバグ修正コストもゼロで、手戻りがあったのは筆者自身のシステム要件定義書の間違い指摘漏れによるものが、一カ所のみであった。その開発費は3年間で約5千万円で、セキュアな設計のために要した追加開発コストは、そのうちわずかに50万円ほどであった。

5.7. デグレードの検知と抑止の効率化

クラウド型診断では、デグレードによる脆弱性の埋め込みを複数回検知し、即時修正させることができた。また、CVEには記述の無いバージョンのミドルウェアの脆弱性（ツールのシグネチャには適切に反映されていた）も1件検知し、ミドルウェアの保守部門も認識できていない脆弱性情報をいち早く認知することもできた。結果として、自社のサービスシステムの脆弱性によって個人情報を丸ごと窃取されたり、Web ページを改ざんされたりといった事故（事件）は、攻撃手法が日々高度化する昨今も含め、事業を開始して以来ゼロ件を維持できている。

5.8. 副次的効果

Fortify によるソース診断は、現在はコストの都合で廃止してしまったが、利用中は延べ200件ほどのスキャンを行い、その全てで何らかの脆弱性や性能劣化に繋がる要素が発見できた。特に検出効果が顕著だったのは、「データベースから読み出したデータの大多数が利用されていない」といった処理上の無駄と、Null Pointer Exception であった。どちらも致命的な脆弱性ではなかったが、サーバリソースの無駄遣いや性能劣化に繋がりやすい要素の一つである。手作業でのコードレビューではこれらの問題は全く検出できていなかったため、改善効果は高く、今後も予算の都合さえ付けばぜひ再導入しておきたいツールの一つである。

6. 考察

6.1. PDCA サイクルとプロセス毎に適用すべきツールの組み合わせ

企画設計の段階からサービスリリース後に至るまで、セキュリティ要件のチェックは複数のプロセスにおいて行うべき事の一つである。しかしながら、闇雲にすべてのプロセスにチェックリストや検証ツールを適用していても、無駄や無理が生じる。実際、当社の中でも事前チェックを重要視しようとするあまり、企画や設計審査の段階で「当該システムは出荷判定に合格しているのか」という的外れな質問が出ることもあった。また、AppScan や WebInspect のような強力な検証ツールでリリース後にも定期的に検証する事ができれば理想ではあるが、スキャンや結果の解析にも相応のスキルを持った要員と工数が必要な上に、お客様の予算の都合で検証用の環境を維持することが難しい場合もある。

そこで、手持ちの商材や世の中のサービスを組み合わせ、サービスのライフサイクルにおいて最も効率的な検証と監視の組み合わせを図 15-B-12-5 のように整理した。

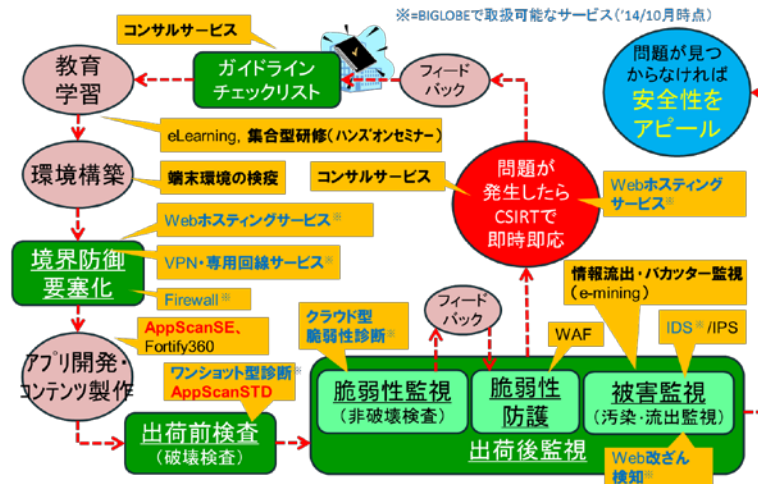


図 15-B-12-5 ライフサイクルと最適な対策の組合せ ver. 2.0

コストの都合もあるので全サービスを利用する（ご契約いただく）のは難しいが、チェックリストの穴埋めなど書類作りに関わる人的工数を極力減らせるようにした組み合わせである。

ただし、検証あるいは監視の結果を正しく分析・解釈・判断できる人（あるいは部門）の育成（設置）は必要不可欠である。当社では、社内向けには **CSIRT** で診断結果の解釈とリスク判断の支援をしつつ、お客様へ提供する際はオプションとして自主事業で培ったノウハウを踏まえた解説と提言を添えるサービスも行っている。

6.2. 検証を重ねて見えてきた、設計時点で考慮しておくべき工夫

AP 層診断（動的診断：DAST）は、「疑似攻撃」と言いつつも実態は「実際に攻撃をかけ、攻撃の余地があるかどうかを調べている」に等しい。検証を受けるサーバから見れば、「検査の通信」と「攻撃の通信」との違いは通信元 IP くらいで、実際の区別は付かないと言って良い。この点を踏まえ、アプリケーション設計において次のような注意と工夫が必要となる。

6.2.1. 他社のサービスと連携するアプリケーションを作る際の注意と工夫

入力されたパラメータをそのまま他社(例えば、Google の検索や地図など)の API に引き渡すような作りであると、検証の際に改ざんした値(攻撃命令)までそのまま引き渡してしまう。その場合、他社から見れば被検査システムが攻撃をかけてきているようにしか見えない(図 15-B-12-6)。

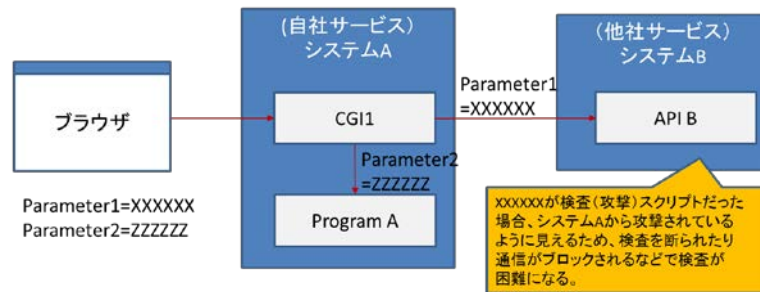


図 15-B-12-6 検証が困難になるアプリ設計

脆弱性検証ツールには基本的に「検証先の IP や URL」を限定し、余計なサーバへは検査パケットを流さないためのガードがかかっているが、図 15-B-12-6 に示すような構造ではこのガードは働かないので、調整が必要となる。とはいえ、検証の都度ソースコードから改造していたのではコストがかかり修正ミスの危険性も増すので、例えば図 15-B-12-7 のようにパラメータ変更だけで簡単かつ確実に他社 API の応答をダミー化できる構造にすることが望ましい。

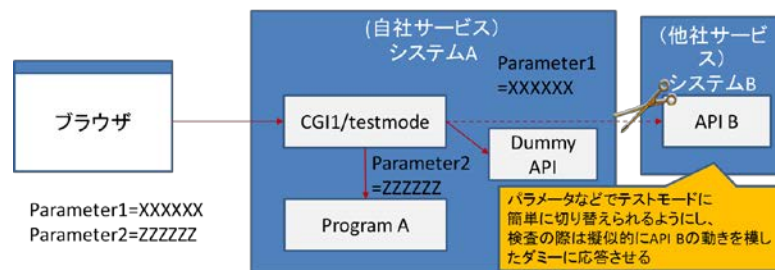


図 15-B-12-7 検証しやすいアプリ設計

6.2.2. リリース後のサービスに対して動的検証をかける際の注意と工夫

リリース前に強力なツールを用いてデータベースへの書込処理まで検証したとしても、日々のプログラム更新や新たな脆弱性の発見によって新たな穴が生まれる可能性はある。稼働中の本番環境と全く同じ構成の評価環境を永続的に保持してそこに日々AppScan や Retina のようなツールをかけ続けることができれば理想ではあるが、これも現実的とは言えない。そこで当社では、AppScan や Retina ほど強力で高精度ではないが、リリース後の本番環境に対して日々軽めの検査（非破壊検査）を行うためのサービスとして、三和コムテック株式会社のクラウド型脆弱性診断サービス（SCT SECURE クラウドスキャン）を採用し、商材としてお客様へも提供している。

クラウド型診断サービスは基本的に非破壊系の検査のみを全自動で行うため、リリース済みの本番環境に対して任意の周期で安価にスキャンをかけ続けられるというのが最大の利点であるが、注意点も多い。例えばブログや Facebook へのコメント投稿、問合せフォームへの書込などは、コミットすれば一般のユーザによる操作と同様に書込処理が実行される。評

価環境であれば後でデータを削除（クレンジング）すれば良いが、本番環境ではそうはいかないので、予め「除外指定」によって最後の書込処理は実行させない(寸止めする)必要がある。

しかし、昨今では CMS⁷ などのパッケージツールが使われることも多く、入力画面・確認画面・結果画面すべてが java などの動的処理もしくは内部パラメータによってのみ区分・処理されているケースが増えている（図 15-B-12-8）。クラウド型診断サービスでは目下こうした遷移の区別が付けられないため、安全のためには入力から書込までのすべての処理を除外せざるを得なくなり、検査の有効性が損なわれてしまう。それを避けるためには、図 15-B-12-9 のように敢えてページ毎に URL が変わるようにするか、クエリストリングスなどの引数で遷移の区別が付けられるようにしておき、寸止めでスキャンできるようにすることが望ましい。

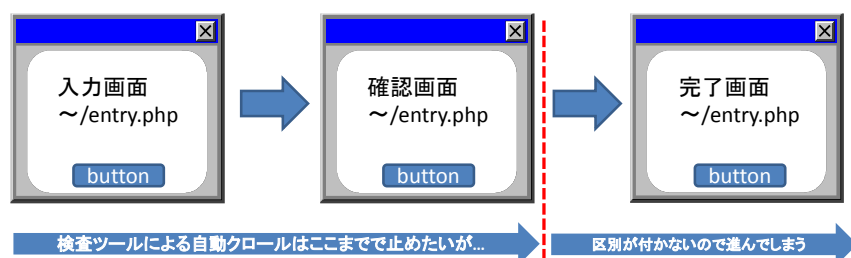


図 15-B-12-8 自動クロールでの「寸止め」が効かなくなりやすい遷移
すべて同じ URL で、スクリプトのみで遷移するパターン

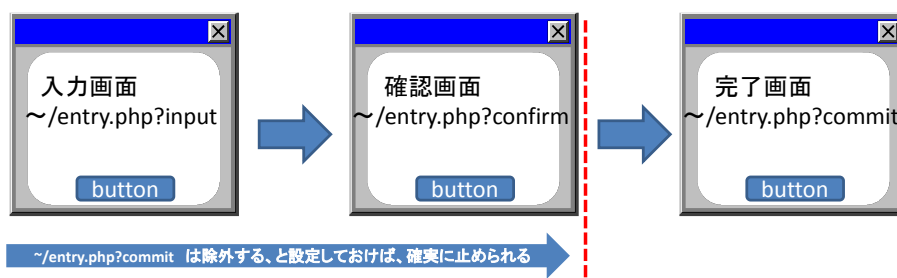


図 15-B-12-9 URL は同じでも、自動クロールでの「寸止め」を効かせやすい遷移
ページ毎に URL 中に識別子が付いているパターン

⁷ Content Management System ウェブコンテンツを構成するテキストや画像などのデジタルコンテンツを統合・体系的に管理し、配信など必要な処理を行うシステムの総称。WordPress や WebRelease、Microsoft SharePoint などがある。

7. 今後の課題

今回ご紹介した手法やツールは、残念ながらいずれも「機械的にチェックすれば、思考する必要もなく判断・対処できる」という物では無い。適切にツールを活用するには、お客様の安心・安全と事業性とを踏まえ、最適な解釈と判断ができるリテラシとマインドを持った担当者の育成が不可欠である。育成方法も、ハッキングスキル（テクニック）やコーディング規約等のルールだけを覚えさせても効果は乏しいと感じている。セキュリティ技術は日進月歩で進化も変化もするため、解釈・判断方法をマニュアル化するのも難しい。そうした事を踏まえ、現在は下記に注力して社内教育を行うように心がけている。

- (1) 「事業者は、お客様の安心・安全を守ったサービスを提供することで初めて対価を頂く資格を持てる」というマインドをまず植え付けること
- (2) セキュリティ技術は日々進化するものであり、変化への対応力にこそセキュリティ人材の価値がある、ということ

実際の教育内容の充実と効果の計測は、まだこれからの状態である。

8. おわりに

当社において、9 年間に渡り多種多様なサービスのセキュリティ診断に携わり、その活動を通じて得られた「対処が必要な脆弱性の考え方」や品質検証ツールの使い分け方、ツールを適用する際の諸注意やコツをご紹介させていただいた。脆弱性を検証するツールや手法は世の中に多々あるが、「これさえ使っていれば万全である」と言えるほど便利な物は、残念ながらいまだ存在しないというのが結論である。

激しい変化を続けるセキュリティの脅威に対し、事業として収益を上げつつ、従業員自身が自らの家族や友人も含めたお客様に「自分が作ったサービスだから安心して使ってくれ」と自信を持って言えるサービスを提供できるよう、効果的・効率的な手法の整備に継続的に努めていく所存である。

また、この経験を振り返り、セキュア開発は「最も TCO⁸ に優れた開発手法」であると実感した。「セキュア開発＝採算性を横に置いた、原理原則主義の高コスト手法」と評する人も少なからず居ると思うし、チームが不慣れな間は短期的な開発費だけを見れば確かに高コストにもなる。しかし、長期的に見れば「余計な運用コストがかからないシステムを作り上げられる開発手法」であり、手法に慣れれば短期的コストも軽減する。開発に関わる方々、予算管理に関わる方々には、セキュア開発手法は短期的な開発費（投資額）や手間だけで評価するのではなく、中長期での運用費も含めたトータルコストで評価していただきたいと思う。

⁸ Total Cost of Ownership

掲載されている会社名・製品名などは、各社の登録商標または商標です。

独立行政法人情報処理推進機構 技術本部 ソフトウェア高信頼化センター（IPA/SEC）