既存 Web アプリケーションの入力処理の脆弱性調査と対策

木村 勇一[1,a)]　後藤 厚宏[1]

概要：本研究は，Web アプリケーション（WebAP）の入力処理の脆弱性に注目し，現在の開発手法に適用可能な対策方法を考察したものである．WebAP に対する古典的な攻撃手法であるパラメタ改ざんのうち，サーバが取り込み対象としているパラメタならばクライアントサイドからパラメタが見えなくてもパラメタが取り込まれることを利用した負のパラメタ改ざん攻撃と呼ばれる攻撃がある．このような攻撃は国内外ともに注目されることは少なかったが，近年，WebAP 開発に用いられる WebAP フレームワークや自動生成ツールの組み合わせ利用の増加によって，入力処理の脆弱性が発生する懸念があり，負のパラメタ改ざん攻撃を受ける可能性がある．そこで本研究では Java で検証用 WebAP を構築し，これを用いて脆弱性の原因調査を行い，入力処理の脆弱性の対策について対策方法を実装し検証を行った．その結果，単一の方法では十分な対策が困難であることが分かった．十分な対策を行うためには，適用する WebAP ごとにいくつかある対策方法を組み合わせることが重要となる．

1. はじめに

WebAP 開発では，高品質／短納期／低コストの実現に向けて，WebAP フレームワークや自動開発ツールの利用が推進されている．しかし，これらを複数組み合わせて利用する場合，WebAP のアーキテクチャは複雑化し，内部構造はブラックボックス化したものとなる．これらを原因とする WebAP の意図しない振る舞いには，取込対象外の入力情報が不正に取込まれる「入力処理の脆弱性」がある．

パラメタ改ざんに関する研究は，イリノイ大学の Skrupsky らの研究 [7][8] では，パラメタ改ざんの攻撃手法のひとつとして，負のパラメタ改ざん攻撃を挙げ，その対策方法を述べている．この攻撃手法は，本研究が注目する入力処理の脆弱性を突いたものである．

Skrupsky らは PHP で作成された CMS[*1] を研究対象としている．PHP の CMS は，ソースコードが閲覧可能であるため研究対象としやすいが，Java で構築された WebAP の場合，基本的な枠組みは類似するものの，それぞれが個々のソフトウェアであり，また，ソースコードを容易に確認できないため研究対象とすることが難しい．

本研究では，Java で構築された WebAP を研究対象とするため，実開発で用いられる技術を使って検証用の WebAP を作成し，これを研究対象とした．

原因を確認したのち，対策方法を実装して検証を行ったところ，従来より提案されている方法も含め，単一の対策を適用するだけでは，十分な対策を取ることができないことが分かった．そのため，各方法を適用した場合に不十分となる点，および，WebAP ごとに対策方法を組み合わせることで十分な対策とする方法を提案の評価として述べる．

2. パラメタ改ざんの脆弱性

2.1 脆弱性の定義

パラメタ改ざんは， WebAP 開発のセキュリティ向上を図る団体である OWASP [*2] では「クライアントとサーバ間で交わされるパラメタの操作に基づく攻撃」[1] と定義されている．OWASP で重要とされる脆弱性をまとめた 2013 年の Top10[2] においては，A1 の「インジェクション」や A4 の「安全でないオブジェクトの直接参照」と関連する．ソフトウェアの脆弱性を収集し，識別を行っている CWE [*3] の分類では「CWE-472（External Control of Assumed-Immutable Web Parameter）」[3] や「CWE-639（Authorization Bypass Through User-Controlled ）」[4] と関連する．

用語としては「パラメタの改ざん（Parameter Tampering）」のほか，「パラメタの操作（Parameter Manipulation）」

[1] 情報セキュリティ大学院大学
INSTITUTE of INFORMATION SECURITY
[a)] mgs124503@iisec.ac.jp
[*1] Content Management System
[*2] The Open Web Application Security Project
[*3] Common Weakness Enumeration

「パラメタの汚染（Parameter Pollution）」[5] と呼ばれ，「汚染の解析（Taint Analysis）」[6] が関連の用語となる．

正規の入力値が変更されて不正な値が入力チェックをすり抜けて内部に取り込まれるため，例えば，パラメタ改ざんの脆弱性を含むオンラインショッピングサイトの場合，変更を想定されていない価格情報が書き換えられたり，想定量以上の購入数が設定されたりするといった被害が想定される．

2.2 パラメタ改ざん攻撃

Skrupsky の研究 [8] では，パラメタ改ざん攻撃を次のように分類している．

基本的なパラメタ改ざん攻撃

（Basic Parameter Tampering Attack）

クライアントサイドから，テキストフィールドの maxlength 設定などを無視したり，リストボックスや隠し項目の値を書き替えたりして，不正なリクエストパラメータを送信することで，不正な情報参照や登録，更新を行う攻撃．

負のパラメタ改ざん攻撃

（Negative Parameter Tampering Attack）

クライアントサイドからは確認できなくても，サーバが取り込み対象としているパラメタならばパラメタが取り込まれることを利用して，不正なパラメタを設定してサーバに送信し，不正な操作を行う攻撃．

順序性を突いたパラメタ改ざん攻撃

（Tampering based Sequencing Attack）

WebAP の画面遷移パターンが，商品選択画面，商品確認画面，購入画面というように順序性を持つ場合に，リンクやアクションパスを不正に書き換えて，商品選択画面から購入画面に遷移するといった遷移パターンを無視した処理を実行する攻撃．この攻撃は，CSRF[*4] と同様の脆弱性を突いた攻撃である．

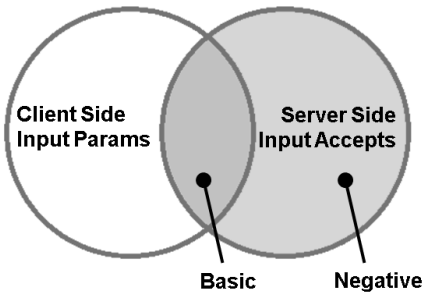

図 1 入力項目とパラメタ取り込みのベン図

パラメタ改ざんは特に目新しくもない古典的な攻撃手法である．2006 年ごろまでは脆弱性に関する注意喚起が行われていたが，近年ではほとんど取り上げられなくなった．しかし，2012 年に発生した CMS の Joomla! 2.5.2 の不正に管理者のアカウント登録が行われる脆弱性 [9] は，負のパラメタ改ざん攻撃に関連するものである．

基本的に Web は，特定の URL に向けて送られたリクエストに対してレスポンスを返すものなので，開発者としてはありがちな操作と言えなくもないが，この操作による意図しない振る舞いは AP の開発者の入力処理の設計不備に起因するため，開発者がこの操作に対する WebAP の振る舞いを意識することは重要である．

2.3 パラメタ改ざんの対策と課題

パラメタ改ざんは，一般的なインジェクション攻撃の攻撃手法として用いられる．二次的な被害として，SQL インジェクションやセッションハイジャック，ディレクトリトラバーサル，CSRF などがあり，パラメタ改ざんをひとまとめに定義すると「開発者の意図しない値でサーバ内部の情報を汚染する操作」となる．

このパラメタ改ざんの脆弱性が埋め込まれる原因は，外部から送信される入力値に対して適切なチェックや無害化が行っていないことである．そのため，基本的な対策方針として，処理の取得情報は不正操作されるという前提で処理を設計することが重要である．

Prithvi は 2011 年に，ホワイトボックスの検査ツール WAPTEC[7] を作成し，ブラックボックスで検査を行うより，ホワイトボックス観点での検査を行う方が，パラメタ改ざんの脆弱性を多く検出できると主張し，実際に PHP の CMS「DCP-Portal」の権限昇格の脆弱性を検出した．

Skrupsky は 2013 年に，WAF[*5] のようにクライアントとサーバの中間位置に設置し，HTTP リクエストを検査するフィルタリングツールとして TamperProof[8] を作成した．TamperProof は，HTTP レスポンスから自動的にチェックルールを作成し，パッチ ID と遷移元のアクションパスを付与することで，パラメタ改ざんや CSRF の防止を試みている．また，HTTP レスポンスからチェックルールを生成する場合，クライアントサイドでの Ajax などの動的な処理を検知することが困難であることも明らかにしている．

最近では，ブラウザに開発者ツールが標準で搭載されているため，パラメタ改ざんのような比較的単純な操作は容易に実行できるようになった．また，パラメタ改ざんに関して以下のように脆弱性が生じる可能性がある．

(1) ブラウザを無視した操作

WebAP 開発において開発者は，ユーザが仕様書にて規定されたブラウザを使用して WebAP を操作することを前提として設計を行うため，悪意のあるユーザによるブラウザを無視した不正操作に対して十分に考慮されない可能性がある．

[*4] Cross Site Request Forgeries

[*5] Web Application Firewall

(2) リッチ化したクライアントサイドの処理
現在，Ajax や HTML5 といったクライアントサイドの技術が発展を遂げている．そのため WebAP もクライアントサイドに処理を実装する比重が高まっている．入力チェック処理や入力抑止処理もクライアントサイドで実装され，サーバサイドでは処理を設けていない WebAP となる可能性がある．
(3) リクエスト処理の自動化
クライアントから送信された入力情報は HTTP リクエストとしてサーバ上の WebAP に取り込まれるが，HTTP リクエストを取得し，内部で保持する情報に変換する処理は WebAP フレームワークの機能や開発者の実装により自動化されている場合がある．このとき，入力情報の設計に不備があると不要な入力項目が取得される．

リッチ化されたクライアント処理があっても，ブラウザを無視した操作が行われた場合に備えてサーバサイドでのチェック処理が必要である．パラメタ改ざんの脆弱性はリクエスト処理の自動化によって，標準的な手順で開発を進めていても埋め込まれる可能性がある．

本研究では，負のパラメタ改ざんのように，クライアント上からは見えないパラメタが WebAP に取り込まれ，後続処理にて不正な動作を引き起こす可能性について着目し，入力処理の脆弱性とした．

3. WebAP の入力処理の脆弱性調査

入力処理の脆弱性を検証するため検証用 WebAP を作成した．そしてこの WebAP に入力処理の脆弱性があることを確認し，その根本原因を考察した．

3.1 検証用 WebAP

攻撃対象の WebAP は，実際に開発されている WebAP を想定して作成した検証用 WebAP である．一般的な画面遷移パターンを持ち，基本的な機能として，ログイン機能，情報参照機能，情報更新機能があり，業務要件として権限ごとに参照と更新可能な情報に制限がある．クライアント端末のブラウザから WebAP にアクセスし，WebAP から DB サーバにあるデータベースへアクセスする．

WebAP フレームワークには TERASOLUNA FW [*6] を使用している．TERASOLUNA FW は WebAP フレームワーク内に，Struts1，MyBatis，Spring framework を利用する複合型の WebAP フレームワークであるため，複数の WebAP フレームワークを組み合わせ，内部構造がブラックボックス化したものの例となる．

3.1.1 画面遷移パターン

一覧→詳細→変更入力→確認→完了という順で画面遷移し，それぞれ隣接する画面にのみ遷移可能とする．ただし，完了画面からは確認画面には遷移できず，一覧画面には遷移することができる．また，ログイン画面を持ち，権限による操作を行う場合は，ログイン画面からログインを行う必要がある．

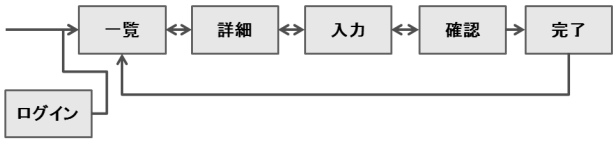

図 2 画面遷移図

3.1.2 ビジネスルール

権限ごとに参照／更新可能な情報を**表 1** に示す．

表 1 権限による振る舞い

権限	ルール
システム管理者	一覧と詳細情報を参照，変更可能．
特権ユーザ	一覧と，自分のグループの詳細情報を参照し，グループメンバの情報の一部※を変更可能．
一般ユーザ	一覧と，自分のグループの情報を参照し，自分の情報の一部※を変更可能．
未ログイン	一覧のみ参照可能．

※権限，グループについては更新できない．

3.1.3 開発規模

規模数は詳細を省くが，プロパティファイル，Java で記述された実処理，データ交換用のオブジェクト，WebAP フレームワークの設定ファイル，画面ファイル（JSP）を含めた総ステップ数は，3.5 キロステップほどである．

3.2 脆弱性を突いた攻撃

攻撃の手順は，まず HTML ソースを参照し，送信する入力情報を調査，推測する．次に入力情報をツールを用いて書き換えや追加を行い，不正な入力値を送信する．そして送信した値がデータベースに登録されるかを確認する．検証用 WebAP のテストデータは，インターネット上から利用できる擬似個人情報ジェネレータ [*7] を利用した．

HTML ソースでは，input タグのパラメタ名の他，CSS の id や class に入力処理の変数名が使用されている場合がある．これを元に入力情報とする項目名が特定できる．また，入力項目の命名規則から表示されていない項目も推測が可能である．form タグのアクションパスを確認することで，確認処理や更新処理のアクションパスも特定できる．

攻撃の操作として以下を行った．
(1) 単純な入力チェックをすり抜ける攻撃
(2) アクションパスを書き替える攻撃
(3) コードの値をマスタに存在しない値に更新する攻撃

[*6] 株式会社 NTT データ
TERASOLUNA Server Framework for Java Web

[*7] kazina.com なんちゃって個人情報，
http://kazina.com/dummy/

(4) キー情報を変更して他ユーザの情報を更新する攻撃
(5) 更新できない権限情報を更新する攻撃
攻撃を実行した結果を下表にまとめる.

表 2 攻撃の実行結果

操作	パラメタ	実行結果	実害
1	特定可能	成功	△対処なし
2	特定可能	成功	△対処なし
3	特定可能	成功	△対処なし
4	推測可能	失敗	—
5	推測可能	成功	○要対処

実行した攻撃のうち，1 番 2 番 3 番は攻撃が成功し，DB の値を入力チェックに反する値に更新することができた．しかし，この攻撃については攻撃者の管理下にある情報の更新であり，他ユーザやシステム全体に影響を与えるものではないため，実際の被害は軽微なものと評価した．

4 番は，他ユーザの情報を書き替える操作であるため，成功すれば問題であったが，更新処理にて楽観的ロックエラーが発生し，成功しなかった．楽観的ロックとは，入力情報取得処理から更新処理までの間のデータ更新を検知する制御である．取得したレコードの更新日時をデータ取得時と更新処理時とで比較を行い実装する．これは，基本的な AP の業務設計にて用いられる制御処理である．つまり，他ユーザのキー情報差込には成功したものの，更新対象レコードが情報取得時のものと異なると検知し，異常終了した，ということになる．

5 番は，権限の昇格に成功した．入力画面にて権限情報のパラメタを推測し，WebAP の内部情報に送り込むことで，不正な権限情報で更新を行うことができた．管理者ユーザでしか権限情報を入力できないように作られた WebAP であるが，不正なタイミングで不正情報が差し込まれた場合を想定していない．権限昇格が行われるため，実害のある故障であり，修正は必須である．

3.3 発生の原因

不正な入力情報が取り込まれる脆弱性の根本原因は，自動的に入力情報をサーバ内部へ格納していること，入力情報と内部で保持する情報をひとつのオブジェクトにまとめていること，この 2 つの要素の組合せである．

その他の要因として，パラメタが推測可能であったことも挙げられるが，本研究では，別の要因として議論は行わない．[*8]

3.3.1 WebAP フレームワークの機能

WebAP フレームワークが提供するリクエスト処理により，入力情報とサーバ内部で保持すべき情報のマッピング（対応付け）の記述を省略することができる．

オブジェクト指向で構築された WebAP の場合，Data Transfer Object と呼ばれるオブジェクトを設けて，オブジェクト間で情報を入れ替える処理が必要となる．入力情報である HTTP リクエストも，リクエスト処理にて内部のセッションの情報に変換する必要がある．

Struts1 や TERASOLUNA FW で行われるリクエスト処理は，直接セッションに値を設定するわけではなく，アクションフォーム機能と呼ばれる一時的な情報保持のためのオブジェクトを生成することでこの自動マッピングを実現している．開発者は，具体的な取得項目を設定したオブジェクトをアクションフォームとすることで，入力情報を取得する．通常，アクションフォームもまた入力情報と変わりないのだが，AP のセッションと同じ有効期限で持ちまわることができる利便性があるため，セッションの代用物として利用されることがある．このアクションフォームに内部情報が定義された場合，自動マッピングの仕組みにより，入力処理の脆弱性の可能性が生じる．[*9]

リクエスト処理，及びセッション管理は WebAP フレームワークごとに異なるため，WebAP フレームワークごとに調査を行う必要がある．**表 3** に主要な Java の WebAP フレームワークについてまとめた．

表 3 WebAP フレームワークごとのリクエスト処理機能

	リクエスト処理機能	セッション管理
Struts1	RequestProcessor	ActionForm
TERASOLUNA	RequestProcessorEX	ActionForm
Struts2	ActionInvocation	ValueStack
SpringMVC 2.5-	DispacherSevlet	Session ※

※ Session の情報を DispacherSevlet が POJO クラスに設定して Controller に渡す.

3.3.2 アクションフォームへの入出力定義

一般的なコーディング技法として，定義情報や処理を目的ごとにひとつのオブジェクトとして定義する方法がある．この方法により，複数項目の取得情報を 1 項目の情報として扱えるようになり情報の持ち回りや流用が容易になり，可読性が向上する．

しかし，入力情報とサーバ内部で保持するため情報をひとつのオブジェクトとしてまとめ，HTTP リクエストとセッションの自動的なマッピング処理が行われた場合，オブジェクト内の情報が全て入力値からの取り込み対象として扱われる．ユーザ ID や権限種別のように，入力値として想定していない内部のみで保持する必要がある情報について，不正な入力によりオブジェクトのユーザ ID や権限情報が書き替えられる可能性が生じる．

[*8] 例えば Salseforce.com が提供する PaaS 型のサービス Force.com を利用して構築された WebAP の場合は，自動生成された画面から入力項目を推測することは困難である．

[*9] Struts1 のアクションフォームは，2005 年の WAS フォーラムカンファレンスにて，「ActionForm クラスをデータの格納に使用しているプログラムを見かけるが，ユーザからの入力を受け取る以外の目的で使ってはいけない．プロパティ名がわかれば，ブラウザから値を上書きできてしまう」と既に指摘されている．

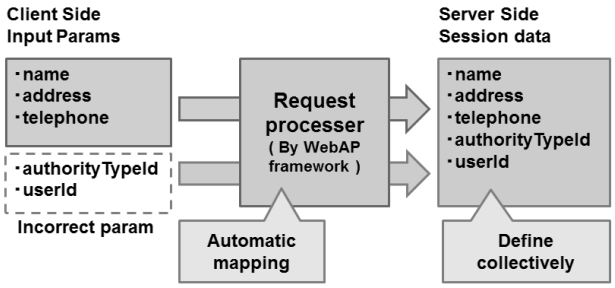

図 3 意図しない取込の例図

3.4 対処の重要性

このような入力処理の脆弱性を突いたパラメタ改ざんは，業務仕様外の操作であり，実際のところ多くの場合は危惧されていない．また，他ユーザやシステムに影響のないものが多く，検出されたとしても実害がないものとして許容され，対処なしとなることが多い．

基本的に Web は，特定の URL に送られたリクエストに対してレスポンスを返すものなので，Web 上で動作する AP は，ブラウザからの入力に限らず指定した URL に，サーバが取り込み可能なパラメタが送信されることで，パラメタが取り込まれるのは当然のことであり，このような振る舞いは当たり前のことと考えることもできる．

しかし，ユーザ ID のようなキー情報や権限情報，金額などの重要な情報など，運営者や管理者，他ユーザに実害を与えるものは対処する必要がある．また，想定しない箇所での入力や不正な情報の取り込みは，本来の業務モデルに反する操作であるため，セキュリティ要求として対策が必要かを上流工程にて検討することは重要である．さらに，後続の処理にて二次的な被害の可能性があるため，開発者は負のパラメタ改ざんも視野に入れ，入力が想定されるパラメタを把握し，それらについて実装が適切であるか，また異常な振る舞いをすることなく後続処理が行われるかを保証することは WebAP 開発において重要である．

4. 入力処理の脆弱性対策の検討

4.1 フィルタに関する既存の技術

パラメタ取り込みに関するフィルタリング処理は既に WAF や WebAP フレームワークにより提供されている．調査結果を**表 4** にまとめている．

4.1.1 OWASP Stinger

OWASP Stinger [*10] は OWASP が提供するオープンソースの WAF ライブラリで，リクエスト情報のフィルタリングを行うため，SQL インジェクションやクロスサイトスクリプティングのほか，パラメタ改ざんの防止機能もある．Java の標準の機能である ServletFilter として WebAP に設定し，入力チェックを行うパラメタを設定ファイルに定義することで，パラメタ取り込み時に入力チェックを行う．

本来，入力チェックを行うものであるため，入力処理の脆弱性に対しては指定したパラメタに関しては，リクエストからの入力がないことをチェックすることが対策となる．

4.1.2 Struts1 のパラメタ取り込み抑止

Struts1 は 2001 年に初版がリリースされ，既に 2013 年にサポート期間が終了となっている Java の WebAP フレームワークである．Java による WebAP フレームワークのデファクトスタンダートとして利用されているため，多くの既存 WebAP が Struts1 を利用している．

既にリリースされた WebAP において WebAP フレームワークを変更する改修は，WebAP フレームワークが WebAP の骨組みであるため困難である．また WebAP フレームワークごとに習熟期間が必要となるため，習熟度の高い技術者の多い Sruts1 で構築された WebAP は今後も保守されていくと想定する．

Struts1 では，入力情報の物理名を特定の接頭語や接尾語とすることで，パラメタ取り込みの識別を行うオプション機能がある．しかし，この機能は公に周知されているものではなく，また実際にこの機能を用いて既存の WebAP の改修を行う場合，入出力の物理名を変更することはその修正の影響範囲が大きくコストが高くなるため，この機能を利用した修正は難しい．

4.1.3 TERASOLUNA FW のパラメタ取り込み抑止

Struts1 を拡張して作られた TERASOLUNA FW には独自機能として，パラメタ取り込みを抑止する機能がある．WebAP フレームワークの設定ファイル上で各アクションパスに抑止可否を設定することで，アクションパスにおけるリクエスト処理時のパラメタ取り込みを抑止することが可能となる．補助的な設定ではあるが，アクションパスの設定ファイルに記述することで実装可能であり，修正のコストは業務ロジックの修正に比べると低い．

この設定は，指定したアクションパスに対する入力取込をすべて抑止するものであるため，確認画面から完了画面への画面遷移といった意図しないタイミングでのパラメタ改ざんには有効だが，アクションフォーム内の特定のパラメタに対する取込抑止は設定できない．

4.1.4 SpringMVC のリクエスト処理

SpringMVC は，Spring Framework を MVC モデルに従いフルスタック化したもので，柔軟にオブジェクト間を結びつける DI 機能を標準的に持つことから，コンポーネントの再利用性の高い WebAP フレームワークとして今後 Struts1 の代替となる WebAP フレームワークとなると想定されている [10]．SpringMVC では，HTTP リクエストを取り込み時に独自にフィルタリングできるデータバインダ機能 [11] を標準的に備えている．これはリクエスト処理において，入力情報をアノテーションによって業務ロジッ

[*10] OWASP Stinger Project,
https://www.owasp.org/index.php/Category:
OWASP_Stinger_Project

表 4　フィルタに関する既存の技術と提案手法

	実装箇所	設定方法	対策方法	汎用性	修正コスト	備考
使用しない	業務ロジック	実装	初期化	×	×	作業量が最も多い
TamperProof	プロキシサーバ	HTTP レスポンス	取込抑止	◎	◎	クライアント処理×
OWASP Stinger	ServletFilter	設定ファイル	入力チェック※	○	△	動的画面項目×
Struts1	リクエスト処理	パラメタ物理名	取込選別	△	×	フレームワーク依存
TERASOLUNA	リクエスト処理	設定ファイル	取込抑止	△	○	同上
SpringMVC	業務ロジック	アノテーション	取込抑止	△	△	同上
	（データバインド機能）		／初期化			
提案手法 A	ServletFilter	アノテーション	取込抑止	○	○	
提案手法 B	ServletFilter	設定ファイル	取込抑止	○	△	
提案手法 C	ServletFilter	HTTP レスポンス	取込抑止	◎	◎	

【凡例】　◎：最適　○：良い　△：可　×：不可
※ Stinger は入力チェックを行うものであり，入力がないことのチェックは本来行わない．

ク上に定義し，情報取得時に初期化処理を行うものである．データのフォーマット変換に使用されることが多いが，パラメタ取込抑止の場合は，内部情報やデータベースの値を再取得し，その値によって初期化を行う．

取込抑止の実施方法が初期化であるため，設定時に初期化処理を実装する必要があり，修正コストがかかる．

4.1.5　既存の対策方法の総括

既存の対策方法ではそれぞれに効果的な点があった．TamperProof は HTTP レスポンスからチェックルールを自動生成しており，クライアント処理による書き換えがなければ，WebAP の修正が不要となる．OWASP Stinger は ServletFilter を利用しているため，WebAP フレームワークに依存しない設定が可能である．TERASOLUNA FW は設定ファイルにプロパティを追加するだけで設定が可能である．SpringMVC はアノテーションを利用しており，設定ファイルに依存しない実装が可能である．一方，Struts1 や TERASOLNA FW，SpringMVC はそれぞれ機能を有しているが，WebAP フレームワーク固有の機能であるため，他 WebAP フレームワークで構築された WebAP に適用できる汎用性はない．また，それぞれ項目ごとに設定が必要となり，修正コストを最小化する方法が求められる．

4.2　対策案

対策を行うに当たり，その対策コストを最小限とするため，作業量の低減とメンテナンス性の高い方法が求められる．また，WebAP フレームワークや自動生成ツールの制約のため，入力情報と内部で保持する情報はひとつにまとめられており，自動的にリクエスト処理が行われることを前提とする．

入力処理の脆弱性対策として，対策のアプローチと具体的な方法としてパラメタのフィルタリングを行う機能とその設定方法について以下より述べる．

4.2.1　対策方法のアプローチ

想定する不正操作は，次のとおりである．[*11]

(1) 入力項目ではない値のパラメタ改ざん
(2) 意図しないタイミングでのパラメタ改ざん
(3) アクションパス (URL) の改ざん

入力処理の脆弱性の対策方法は 2 つのアプローチがある．

一つは，ブラウザから入力できるパラメタのみが取り込まれ，それ以外のパラメタは取り込まれないようにするホワイトリストのフィルタリングである．もう一つはブラックリストのフィルタリングである．これは取り込みを抑止するパラメタの一覧を作成してサーバサイドで保管しておき，HTTP リクエストのパラメタ情報と突き合わせて一覧にあるパラメタが取り込まれないようにすることである．入力処理の脆弱性対策において，ホワイトリスト方式，ブラックリスト方式ともに対策案の検討を行った．

ホワイトリスト方式のように，詳細設計工程において画面設計書の入力項目一覧にあるものを取り込み許可するとした方が，業務ロジック処理の入力項目一覧から取り込みをキャンセルする項目を洗い出すよりも効率的に対象範囲を選択できるため，優れていると考えられる．実現性においても WAF やフィルタリングツールを使用すればホワイトリスト方式での対策実現は不可能ではない．しかし，この方法を実施するのは WAF などのツールと変わりなく，実装時の許可対象項目の管理作業が発生する．

一方，フィルタリングを使用せず特定の項目の初期化や入力チェックを行う場合は，ブラックリスト方式で個別に対処することとなる．既存 WebAP のように修正を局所的に実施したい場合などは，ブラックボックス方式で厳密に対象項目の精査を行い，影響範囲が最小限となるように進めるほうが要件に合致している．

4.2.2　脆弱性対策の具体的方法

WebAP フレームワークに依存せず異なる AP アーキテクチャにも汎用的に適用可能な機能として，Servlet の標準機能である ServletFilter を利用したフィルタクラスを作成する．フィルタクラスは HTTP リクエストを読み取り，チェックルールをもとにパラメタの取り込み許可／抑止処

[*11] 隠し項目やリストボックスの値の書き換えについては対象外としている．

理を行う．

チェックルールにはアクションパスと指定したアクションパスにおける取り込み許可／抑止パラメタの一覧がセットで定義される．これにより不正操作 1 だけでなく，不正操作 2，3 に対しても対策が取れるようになる．

TamperProof では以上の要件のほか，パッチ ID と遷移元アクションパスも保持してチェックを行っているが，本提案では，WebAP フレームワークの機能を利用したトークンチェックを行うこと，リファラ情報はクライアントサイドから書き換えが可能であることを考慮して，要件から除外している．

4.2.3 フィルタ機能へのチェックルールの設定方法

フィルタ機能へのチェックルールの設定方法について以下の通り，3 つの案がある．

A) アノテーション

Java のアノテーション（注釈）機能を利用してブラックボックス方式でチェックルールを作成する．アノテーションは外部の設定ファイルに切り出されていた設定情報を Java のソースコードに設定可能であるため，煩雑な設定ファイルの削減が見込める．業務ロジックの入力値についてブラックリストのフィルタルールを用いて，取り込み抑止を示すアノテーションを記述する．

B) 設定ファイル

外部の設定ファイルに，アクションパスとクライアントから入力可能な入力パラメタの一覧を定義して，フィルタ処理が設定ファイルを読み込み，定義されたチェックルールに基づいてパラメタの取り込み許可を行うホワイトボックス方式の制御手法である．クライアントサイドの定義情報をサーバサイドの業務ロジックと依存せずに保持するためには，設定ファイルとして保持することが妥当である．

C) HTTP レスポンスデータ

フィルタ処理が HTTP レスポンスから入力項目を抽出し，検証用パラメタ一覧を作成し，後続する HTTP リクエストの内容を検証するホワイトボックス方式の方法であり，先行研究の TamperProof と同様の手法である．この手法は，サーバにて生成される画面情報をそのままチェックルールとし，入力処理の脆弱性のフィルタ処理を自動化する．サーバサイドにクライアントサイドの情報を埋め込むこともなく，また状態ごとの可変的な入力項目の懸念もないため，効率的で最適な案である．

4.3 設定方法の評価

フィルタ機能へのチェックルールの設定方法について，机上での実現性検証を行い，詳細な点については実装を通じて確認をした．その結果，3 つの案にはそれぞれ課題があった．課題の観点を以下の通り列挙し，その後，各案の課題を述べる．

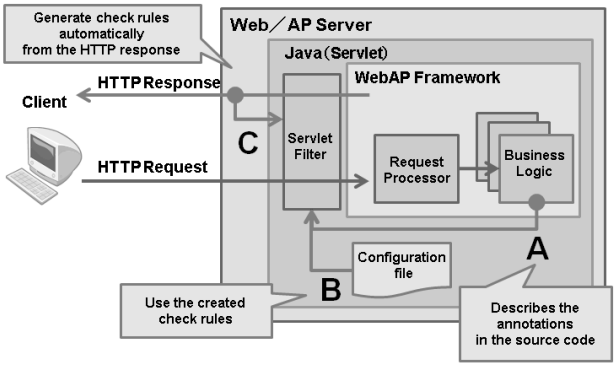

図 4 提案の概要図

ア) 依存性

MVC モデルのモジュール分割のルールに従い，画面項目名が業務ロジックに記述されていないこと．

イ) 状態ごとの可変的な入力項目

権限など業務要件による動的な画面表示処理が行われると想定し，事前の状態によって入力項目が動的に変化する場合でも取り込みの許可／抑止が行えること．

ウ) 作業量の肥大化

記述する項目数が多いほど作業量が大きいと想定し，修正作業量が従来の手法と比べて低減していること．

エ) クライアント処理

TamperProof の研究結果に従い，JavaScript などのクライアントサイドの操作を想定し，クライアント処理による動的な入力項目／ URL の変更や，クライアントとサーバ間の非同期通信がある場合でも取り込みの許可／抑止が行えること．

評価結果を**表 5** にまとめた．

表 5 各案と課題との評価表

	A 案	B 案	C 案
ア）依存性	×	○	◎
イ）状態変化への対応	(△)	×	◎
ウ）作業量	(○)	△	◎
エ）クライアント処理	(◎)	○	×

案 A は未実装であるため想定の評価である．

A 案について，アノテーションを記述するソースコードは，クライアントサイドの情報（入力パラメタ名）を保持していないため，単純なアノテーションの付与だけでは取り込み抑止対象を特定できない．そのため，アノテーションの引数として，クライアントサイドの取込キャンセルを行うパラメタ名やアクションパスを指定できるようにする必要がある．しかし，Java のクラスである業務ロジック内にクライアントサイドの入力項目情報を設定することは，観点アに反するため採用すべきではない．また，依存性を生じないような実装方法を検討したが，この課題に対して理想的な修正を行う場合，サーバ内のデータストアに格納さ

れた情報との突き合わせ確認を実装するため，業務ロジックにチェック処理を追加するのと変わりないと考えた．

ただし，観点イに対しては業務要件を考慮した修正は本来，業務ロジックで行うものなので，観点イについて A 案を採れるような実現方法を考察することは今後にために有効である．観点エに対しては，サーバサイドのみを考慮すればよいため，問題外とすることができる．

B 案について，ホワイトリスト方式での取込対象パラメタの定義は，観点イに対応することは難しい．設定ファイルで許可設定を行う場合，想定されるすべての入力可能なパラメタ名を記述するため，状態によって項目が取込対象が変化する場合に対応できない．仮に可変的な入力項目を制御するようにフィルタが動作したとしても，設定ファイルでは，状態ごとにさらに取込の可否設定を記述しなければならないので，設定項目の管理が煩雑になり，観点ウとして挙げている作業量が増大が起こる．

一方，B 案はクライアントサイドの情報である入力パラメタをソースコードに記述しないで済むため，観点アに対応できている．また，開発者自身が柔軟に定義を設定できるため，観点エを解決することが可能である．そのため，観点ア，エへの対策として B 案を採用し，観点イについては業務ロジックの修正を実施することが，作業量を低減させる対策となる．

C 案について，自動的にチェックルールを作成するため，既存 WebAP への対策方法として最適であるが，観点エに対しては有効な解決方法がないため，クライアント処理を利用した WebAP に対しては採用できない．

しかし，クライアント処理を行わないポリシーの元で開発される WebAP や，クライアントツールもあるため，WebAP のアーキテクチャに基づいて C 案を採用することは可能である．

5. まとめと課題

本研究では，WebAP の内部構造がブラックボックス化した際に生じる可能性のある入力処理の脆弱性に注目した．この入力処理の脆弱性は，国内外ともに十分に取り上げられていない負のパラメタ改ざん攻撃を引き起こすものである．研究対象として，負のパラメタ改ざん攻撃としては未検証である Java の WebAP について検証用 WebAP を作成することで研究対象とした．

検証用 WebAP を用いて入力処理の脆弱性の調査を行った．脆弱性の発生原因は自動的に入力情報をサーバ内部へ格納する仕組みと入力情報と内部で保持する情報をひとつのオブジェクトにまとめていることの組み合わせである．その対策としてフィルタ機能の検討と実装を行い，単一の方法による対策が難しいことを明らかにした．この結果を受けて，十分な対策を行うために，各対策の困難な点を整理し，WebAP ごとに各対策を組み合わせる方法を考察した．

課題としては，入力処理の脆弱性について，他の WebAP フレームワークを使用した場合に再現されるか，また Java 言語以外の PHP などの場合には再現されるかといった，他アーキテクチャでの継続的な再現検証が今後も必要となる．一例として，SpringMVC はステートレスな RESTfulWeb サービスに対応するものだが，本研究の対象 WebAP はステートフル WebAP であった．そのため，状態管理を標準機能として持つ SpringWebFlow と SpringMVC を比較することは価値がある．

参考文献

[1] OWASP: Web Parameter Tampering. (online) , 入手先 ⟨https://www.owasp.org/index.php/Web_Parameter_Tampering⟩, (2014.1.20).
[2] OWASP: Top 10 2013. (online), 入手先 ⟨https://www.owasp.org/index.php/Top_10_2013-Top_10⟩, (2014.1.20).
[3] CWE: CWE-472: External Control of Assumed-Immutable Web Parameter. (online), 入手先 ⟨http://cwe.mitre.org/data/definitions/472.html⟩, (2014.1.20).
[4] CWE: CWE-639: Authorization Bypass Through User-Controlled Key. (online), 入手先 ⟨http://cwe.mitre.org/data/definitions/639.html⟩, (2014.1.20).
[5] Marco Balduzzi: *HTTP Parameter Pollution Vulnerabilities in Web Applications*, Black Hat Europe 2011 (2011).
[6] David Brumley: Taint Analysis. (online), 入手先 ⟨http://users.ece.cmu.edu/ dbrumley/courses/18487-f10/files/taint-analysis-overview.pdf⟩, (2014.2.19).
[7] Prithvi Bisht, Timothy Hinrichs, Nazari Skrupsky, V.N. Venkatakrishnan.: *WAPTEC: Whitebox Analysis of Web Applications for Parameter Tampering Exploit Construction*, CCS 2011 (2011).
[8] Nazari Skrupsky, Prithvi Bisht, Timothy Hinrichs, V. N. Venkatakrishnan, Lenore Zuck.: *TamperProof: A Server-Agnostic Defense for Parameter Tampering Attacks on Web Applications*, CODASPY 2013 (2013).
[9] 徳丸浩: Joomla2.5.2 の権限昇格脆弱性. 徳丸浩の日記. (online), 入手先 ⟨http://blog.tokumaru.org/2012/12/joomla-privilege-escalation-vulnerability.html⟩, (2014.1.20).
[10] 掌田津耶乃: Eclipse ではじめる Java フレームワーク入門 第 4 版 クラウド開発対応, 秀和システム (2013).
[11] soracane: *03.* 基本概念：バインダー（*Binder*）とは (online), 入手先 ⟨https://sites.google.com/site/soracane/home/springnitsuite/spring-mvc/04-ji-ben-gai-nian-controllerno-chu-lifuro⟩, (2014.1.21).