モデルテストによるセキュリティ設計パターンの適用検証

小橋孝紀[†1]，鷲崎弘宜[†1]，深澤良彰[†1]，大久保隆夫[†2]，海谷治彦[†3]，吉岡信和[†4]

ソフトウェア開発では開発者が必ずしもセキュリティの専門家ではなく，開発の早期段階で,システムに起こりうる脅威・脆弱性を確認・検証が十分ではない．またセキュリティ対策を施したところで，本当にセキュリティ要求仕様を満たしているかの検証も不十分である．そこで我々は UML モデルのシミュレーション環境を用いたモデルテストによるセキュリティパターンの適用検証を提案する．本手法を用いることによりセキュリティに精通しない開発者であっても設計パターンの正しい適用と，パターン適用による設計上の脆弱性の有無を検証可能とし，セキュアな設計を実現する．

1. 背景

現在ソフトウェア開発者が必ずしもセキュリティの専門家ではなく,開発の早い段階でシステムにおけるセキュリティ項目について十分に考慮されていない．そこでセキュリティの専門家の知識を活用・再利用する方法としてセキュリティパターン[1,2]がある．セキュリティパターンとは,セキュリティに関する特定の状況下で幾らか抽象化された問題およびその解決法である．ソフトウェアにおける脅威とその対策の特定をする手法として,セキュリティ要求パターン(Security Requirements Patterns)(以下、SRP)[1] の利用がある．またセキュリティ要求仕様を満たすソフトウェア設計の具体的な設計手法の一つにセキュリティ設計パターン(Security Design Patterns)(以下 SDP)の利用がある．

2. 関連研究とその問題点

SDP の例として[2]では,25 個のセキュリティパターンが示されている．しかしながら,現状では設計モデルに対し特定済みの脅威に対しセキュリティ対策としてパターンを適用した際,パターンを適切に適用できたか,またセキュリティ被害を引き起こしうる設計上の脆弱性が解消できたか,これらを検証することは十分おこなわれていない．

これに対し Arnon らはデータベースアプリケーションに対してステレオタイプを用いたセキュリティパターンの適用検証を提案している[3]．パターン適用の検証を行うという点では本研究と似ているが,この研究では構造的なパターンの適用検証のみに留まり,パターン適用後の設計モデルにおいて,適切な振る舞いによって実際に脆弱性が解消されたかの検証がなされていない．

3. 提案手法

先の背景と関連研究を踏まえ本稿で扱う Research Question を以下に示す．

RQ1 モデル上でのセキュリティ設計パターンの適切な適用を検証できるか．

RQ2 セキュリティ設計パターンの適用前や適用後におけるモデル上で,特定済みの脅威に対する脆弱性の有無を正確に検出できるか．

我々は既存のセキュリティパターンを拡張した上で，新たにパターンの適用検証プロセスとモデルテストを実行する為のテストツールを提案し，上記の RQ1，RQ2 に応える．本研究の貢献は次の 3 つである．

- 既存のセキュリティ要求・設計パターンに OCL 記述を用い拡張した新たなパターンの提案．
- テスト駆動開発の概念を基に設計上でパターンの適用検証と脆弱性を検証するテストプロセスの提案．
- モデリングやパターン適用をサポートし，テスト実行の為のスクリプトを自動生成するツールの提案．

3.1. 本手法の全体像

本手法では既存の SRP,SDP ごとに拡張した拡張 SRP,拡張 SDP を予め用意しておく．拡張 SRP,SDP の全体構成と本手法の流れを図 3.1,3.2 に示す．

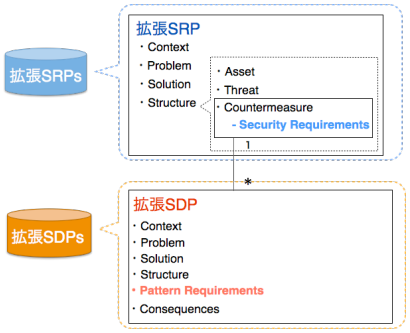

図 3.1 拡張 SRP, SDP の全体構成

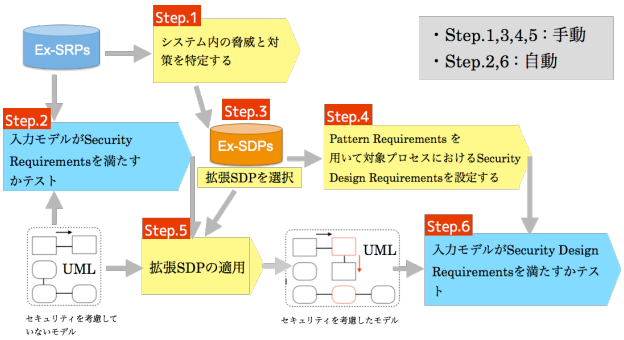

図 3.2 本手法の全体像

本手法の流れを以下に示す．

1. 要求に対し，拡張 SRP を適用し，要求段階において開発中のソフトウェアにどういったアセット(資産)，脅威，対策があるかを特定する．
2. セキュリティを考慮していないモデルを入力とし，特定した対

†1 早稲田大学、†2 情報セキュリティ大学院大学、†3 神奈川大学、†4 国立情報学研究所

策毎に用意された Security Requirements (図 3.1)を満たしているか確認する. ここで Security Requirements を満たしていない, つまり要求段階で特定した脅威に対する脆弱性が検出されることを確認する.

3. セキュリティを考慮していないモデルが Security Requirements を満たしていないことを確認した上で, 拡張 SRP の対策に対応する拡張 SDP を選択する.
4. 各 SDP に用意された Pattern Requirements (図 3.1)を用いて, 対象プロセスにおける Security Design Requirements(OCL 記述)を設定する.
5. 拡張 SDP を適用する. 具体的には入力として与えられたモデル図に対し拡張 SDP の構造・振る舞いを適用する. 適用の際にはパターン毎に定義されたステレオタイプをもとにパターンをマッピングする.
6. パターンを適切に適用できたかを検証する為, 対象プトセスの Security Design Requirements が満たされているかを検証する.

4. ケーススタディ

ケーススタディではソフトウェア工学を専攻する修士学生らに作成させた, 学生管理システム[4] を対象にパターンを適用することで, RQ1, RQ2 を評価する. このシステムは特にセキュリティを考慮していない. 対象となるシステムの規模を表 4.1 に示す.

表 4.1 対象となるシステムの規模

ユースケース数	24個
クラスの要素数	31個 (View : 18, Controller : 5, Model : 8)
開発期間	3ヶ月

ケーススタディでは「Student Controller」の「delete」関数, つまり「学生情報を削除する」というユースケースに対して脅威と対策を特定した上で本手法を適用する. 今回は学生情報の削除に対する脅威を[Privilege Exploitation:権限昇格]と[SQL Injection]とし, その対策として[Access Control:アクセス制御], [Input and Data Validation:入力データの検証]が有効なケースを想定し評価を行った. 上記のユースケースにおいて満たされるべき Security Requirements を以下に示す.

```
context StudentController
 inv SecurityRequirement :
  if self.DeleteUI.Actor.right = true and
     self.DeleteUI.Actor.valid_input_data = true then
   self.delete = true
 else
   self.delete = false
 endif
```

図 4 学生情報削除処理における Security Requirements

まずは上記の OCL で記述された Security Requirements が満たされているかを検証し, パターン適用によって脆弱性が解消されることを検証する. Security Requirements を満たしていないことを確認した上で, 次にパターンを適用する. 拡張 SDP には「RBAC」と「Prevent SQL Injection」を選択しパターンを適用する. 表 4.2 に満たすべき Security Design Requirements を示す.

表 4.2 学生情報削除処理における Security Design Requirements

		1	2	3	4
条件	<<UserData>>が属する<<Role>>にアクセス権限<<Right>>が与えられている	Yes	Yes	No	No
	削除が行われる<<UI>>で入力データのエスケープ処理が行われている	Yes	No	Yes	No
行動	<<Actor>>に権限があると見なす。	×	×		
	<<Actor>>に権限が無いと見なす。			×	×
	<<UI>>に安全な入力を受けたと見なす	×		×	
	<<UI>>に危険な入力を受けたと見なす		×		×
	学生情報削除処理を実行できる	×			
	学生情報削除処理を実行できない		×	×	×

モデルテストにて上記の 4 つのテストケースのテストを行い, パターン適用後のモデルが各テストケースに対し期待する行動をとるか確認した. このように, 学生情報削除処理おいて Security Design Requirements を満たすか検証することで **RQ1** であるセキュリティパターンの適切な適用を検証できた.

最後に学生情報削除処理おける Security Requirements を満たしているか再度モデルテストによって検証し, パターン適用後のモデルが Security Requirements を満たすかを確認した. これにより **RQ2** であるパターンの適用前後でのモデルにおける要求段階で特定した脅威に対する脆弱性の有無の検証した. 具体的にはセキュリティを考慮していない入力モデルに対し Security Requirements を満たしているかを検証し, パターン適用後, 再度 Security Requirements を満たしているかを検証することで, 特定済みの脅威に対する脆弱性が解消されたことを本ケーススタディで確認した.

5. 終わりに

我々は UML デルのシミュレーション環境を用いたモデルテストによる, セキュリティ設計パターンの適用検証を提案した. システムにおける設計上の脆弱性というのは, 漠然と考えていても考えつくものではないため, 本手法ではまず, 既存の問題とその対策が記述されているセキュリティパターンを拡張した. この拡張 SRP, SDP を参照することで開発の早い段階でシステムにおける資産, 脅威, 対策を特定し, パターンの正しい適用と, 脅威によってセキュリティ被害を引き起こしうる設計上の脆弱性の解消の有無をモデルテストにより検証した.

参考文献

[1] T.Okubo H.Kaiya N.Yoshikawa Effective Security Impact Analysis with Patters for SoftwareEnhancement IJSSE 3(1): 37-61 (2012)

[2] M.Schumacher Eduardo Fernandez- Buglioni Duane Hybertson, Frank Sommerlad.SECURITY PATTERNS”Wikey. 2006

[3] Arnon Sturm, Jenny Abramov, Validatingand Implementing Security Patterns for Database Applications SPAQu ’09 2009

[4] SSR-Project https://github.com/SSR-Project