社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－SE－158 (6)
2007／11／2

セキュリティパターン技術に関する研究動向

吉岡 信和[1] 鷲崎 弘宜[2] 丸山 勝久[3]

[1] 国立情報学研究所 [2] 国立情報学研究所/総合研究大学院大学 [3] 立命館大学 情報理工学部

近年のインターネットなどを利用したオープンなサービスの利用拡大と、コンピュータの社会への浸透にともない、セキュリティは益々重要になってきている。しかしながら、セキュリティを考慮した安全なシステムの構築には、様々な要素を考慮しなければならず、その構築が困難であった。専門家の知識をさまざまなシステムで利用可能にする技術として、パターンが有用である。セキュリティに関してもそれに関する知識を、広く利用可能にしたセキュリティパターンが、近年多数提案されてきており、安全なシステムを開発するための情報が整いつつある。本稿では、このセキュリティパターンに関する最新動向を整理し、今後の研究動向について考察する。

Technologies of Security Patterns

Nobukazu Yoshioka[1] Hironori Washizaki[2] Katsuhisa Maruyama[3]

[1]National Institute of Informatics [2]National Institute of Informatics / The Graduate University for Advanced Studies [3]Department of Computer Science, Ritsumeikan University

As the growth of the Internet connected open systems and clients, security has become an important role for many software systems. Security patterns are reusable solutions to security problems. Although many security patterns and techniques for them have been proposed, the use of combination is difficult because of a luck of research on integration of patterns into systems.
This paper provides a survey of approaches to security patterns. As the result of classifying of the approaches, we illustrate a direction for the integration and future research topics.

1 はじめに

オープンなネットワークを利用したサービスが社会に浸透し、ビジネスとして利用されるようになってきたことに伴い、そのセキュリティが重要視されてきている。セキュリティの関心事は、ソフトウェア開発の要求から設計、実装、テスト、運用の全てのフェーズで扱えるようにしなくてはならない [1]。しかしながら、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い。そこで、専門化の知識を活用・再利用する方法としてソフトウェアパターン [2, 3] がある。パターンとは、ソフトウェア開発の様々な場面で繰り返し登場する構造、振る舞い、行為、プロセス、事 (Thing) である。特に、セキュリティに関する知識をまとめたものがセキュリティパターンである。近年、多くのセキュリティセキュリティパターンが提案されているが [4, 5, 6]、ソフトウェア開発の各工程でそれを適切に利用することは、困難である。

本稿では、セキュリティパターンの研究動向を紹介すると共に、既存のパターン技術が不十分な点を整理する。そして、今後の研究の方向性について論じる。具体的には、次の 2 節では、セキュリティの概念とセキュリティパターンに関する本稿での定義を述べる。そして、3 節では、ソフトウェア開発の工程、およびセキュリティパターンに関する活動の観点で、セキュリティパターンの技術を整理する。さらに、4 節では、セキュリティの概念やモデルの観点で、これまでのセキュリティパターン技術の不十分な点と、今後の研究方針を論じる。最後の 5 節で本稿をまとめる。

2 定義

2.1 セキュリティに関する概念

[5] では、セキュリティに関連する概念として、次の９つを挙げている。

アセット (Asset): 組織や人が価値があると認めることができる情報、または、リソース
ステークホルダ (Stakeholder): アセットに特定の価値を見出す組織や人
セキュリティ目標 (Security objective): 脅威に対する安全の要求を満たすための宣言文
脅威 (Threat): アセットに対する安全が脅かされる可能性があること
攻撃 (Attack): アセットの安全性を破壊する行為
攻撃者 (Attacker): 攻撃を実行する実体
脆弱性 (Vulnerability): アセットに関するセキュリティを破ることができる欠陥や弱点
対策 (Countermeasure): 脆弱性や攻撃に対してアセットを保護するための行為
リスク (Risk): 攻撃が成功する確率の高さ

ここで、重要なことは、セキュリティは、守るべき価値のあるアセットに対する故意の攻撃とその対策に関する概念であるということである。

本稿では、この概念に注目し、セキュリティパターンに関する技術がこの概念の観点で十分であるかを整理する。

2.2 セキュリティパターン

パターンとは、特定の状況 (Context) に関する問題に対する解決法 (Solution) を表している [5]。その規定方式はパターンによって様々であるが、[5] では、名前などのほかに、状況 (Context)、問題 (Problem)、解法 (Solution)、解法の構造 (Structure) や振る舞い (Dynamics)、結果 (Consequences) などが含まれる。但し、この記述は、複数のシステムに再利用可能であるように考慮されている必要がある。特にセキュリティパターンでは、2.1 節に説明したセキュリティの概念に関連する記述となる。

加えて、本稿では、パターン言語とは、「関連する問題に対して、それを体系的に解決するための手順を規定した、パターン間のネットワーク」 [7] のことをさす。すなわち、セキュリティパターン言語は、セキュリティに関連する問題を解決するためのセキュリティパターンのネットワークである。

3 セキュリティパターン技術

この節では、まず、既存のセキュリティパターンをソフトウェア開発のライフサイクルの観点で分類し、それぞれの開発工程でどのセキュリティの概念を扱っているかを 3.1 節から 3.3 節で整理する。次に、3.4 節で、セキュリティパターン自体のライフサイクルの観点でパターンを整理する。

3.1 要求・分析に関するセキュリティパターン

セキュリティに関する要求には、何を守るのかの規定が含まれる。その守る情報・およびリソースがアセットである。アセットを、どのようにどこまで守るのかの方針を定めるために、なぜ守るのかを分析する必要がある。

要求に関するセキュリティパターンには、プロセスパターン・分析パターンが多い。セキュリティパターンに関して最もよくまとまっている文献は、[5] である。[5] には、システム中のどのような情報がアセットかを識別するためのパターン (Security needs Identification for enterprise assets) が説明されている。このパターンでは、アセットの識別、セキュリティに関するビジネス要因の識別、アセットとビジネス要因の関連の分析、アセットに対してどのようなセキュリティの種類が必要かを識別するプロセスを説明している。このセキュリティの種類は、データを特定範囲にしか公開しない機密性 (Confidentiality)、望まない変更が存在しないという完全性 (Integrity)、必要な時にはいつでもアセットにアクセスできるという有効性 (Availability)、必要な人がアセットに関する影響を知ることができるという説明可能性 (Accountability) の 4 種類である。

さらに、[5] では、アセットに対する価値を設定する時の方針を示したパターン (Asset Valuation) が紹介されている。このパターンでは、アセットの価値をセキュリティの要求、経済価値、ビジネス要因などの観点で、極めて僅か (negligble) から極めて高い (extreme) まで６段階まで評価する指標を示し

ている。

セキュリティを実装するにはコスト (構築時間、リソース、金額) がかかるため、１００%安全なシステムを作るのは大変困難である。そのため、コストと安全性のトレードオフを考えるため、アセットには優先順位をつける必要がある。その優先度は、アセットの価値とともに、アセットに対するリスクにも依存する。そのために、リスク・脅威・脆弱性を分析する必要がある。これにより、アセットをなぜ守るのかも明確にすることができる。

[5] では、脅威分析を行うプロセスパターン (Threat Assessment)、システムの脆弱性を分類するパターン (VulnerabilityAssessment)、リスクを決定する方法をまとめたパターン (Risk Determination) が紹介されている。

さらに、Enterprise Security Approaches[5] のパターンは、アセットを守るための方針を示している。その方針には防御 (Prevention)、検知 (Detection)、対応 (Response) があり、この方針に沿ってセキュリティの機能が設計、実装される。

[8] では、セキュリティに関するアンチパターンが提案されている。そこでは、攻撃が成功してしまうパターンや不適切なセキュリティ機能を実装してしまうパターンが説明されている。

3.2 設計のためのセキュリティパターン

設計工程では、要求の規定の際に認識されたアセットに対して、セキュリティの種類 (機密性、完全性、有効性、説明可能性) を実現するための機能を設計する。具体的には、アクセス制御、認証、暗号化、署名、ロギングなどにより要求を満たす機能を設計する。具体的に、どのようなセキュリティ機能を設計するかは、セキュリティの方針 (防御、検知、対応) により決定される。例えば、機密性を破る攻撃に対しては、防御のためにはデータを暗号化し、検知のためにはロギングを行う。

Fernandez と Pan は、[4] で、セキュリティの機能に関するデザインを、デザインパターンと同様、モデルを使って説明している。具体的には、許可 (Authorization)、ロールベースアクセス制御、マルチレベルセキュリティ、ファイルの許可 (File Authorization) に関するパターンをまとめている。[5] では、概念的なアーキテクチャと設計レベルのパターンに関して 25 個のパターンを紹介している。

不特定多数がアクセスする状況で、機密性を実現するために、アクセス制御が用いられる。[5] では、アクセス制御モデルに関して５つのパターンが紹介され、それを実現するアーキテクチャに関して６パターンが紹介されている。具体的には、どこでどうアクセスを制御するかについてのパターンに関しては３つのパターン (Single Access Point, Check Point, Security Session) が紹介されている。そして、アクセス制御に関連したユーザインターフェースに関する２つのパターン (Full Access with Errors, Limited Access) が紹介されている。[9] では、UMLsec を使って安全にデータを送受信するパターンを表現している。

完全性は、許可されていないユーザに、データの変更を許さないことで実現する。そのために、電子署名が有効である。[10] には、VoIP の場合の完全性を保証するためのパターン (Signed Authenticated Call Pattern) が紹介されている。

有効性を実現するには、ファイアウォールが利用できる。ファイアウォールのパターンとして、IP レベル、トランスポーテーションレベル、サービスレベルの３つのレベルのパターンがある [5]。ファイアウォールは、攻撃を軽減することで有効性を実現するだけではく、アクセス制御リストを設定することにより機密性も実現可能である。

説明可能性を実現するパターンとして、[5] では、Security Accounting Requirements など５つのプロセスパターンが紹介されている。

上記で説明したパターンは、3.1 節のどのセキュリティの種類を具体的に満たすのかなど、セキュリティの要求との関係まで明確に言及していないものがほとんどである。セキュリティパターンとそれに関係するセキュリティ要求との関連を非機能要求モデルを使って表現する提案を Weiss らが行っている [11]。この提案では、セキュリティの性質、パターンを実施する理由と設計レベルのセキュリティパターンとの関連をモデル化している。さらに、パターン間の関連もモデル化することで、推論が可能となっている。

さらに、特定のアプリケーション領域に特化したセキュリティパターンも提案されている。例えば OS レベルのパターン [12, 5] や Web アプリケーション向けパターン [6] などである。[6] では、Web アプリケーションに関する 23 個のセキュリティパターンを紹介している。

3.3 実装に関するセキュリティパターン

3.3.1 セキュアプログラミング

これまで、様々なセキュリティに関する実装上の不都合が明文化されてきている。[13] では、取り除くべきプログラム上の問題に関して、１８個の実装ルールを定めている。同様のルールは、[14] にも紹介されている。[14] では、６個種類について推奨される実例と、２３個の悪い実例を整理している。

プログラマの視点で、セキュアなプログラムを書くための設計や実装をガイドラインとしてまとめたものを、一般に “セキュアプログラミング” と呼ぶ。[15] は、Linux や Unix システムに関するよく知られたセキュアプログラミングのためのガイドラインを示している。この中には、バッファオーバフローなどの脆弱性のガイドラインだけではなく、C/C++, Perl, Python など言語特有な問題も説明している。[16] は、安全な Java コードに関する１２のルールを定めている。Jslint [17] は、そのルールを使ってセキュリティ上の脆弱性を自動的に発見するツールである。さらに、[18] では、Java のプログラマが起こしやすい脆弱性について取り上げ、それを最小限にするガイドラインを示している。

これらのガイドラインは、熟練したプログラマにより実践的にまとめられている。具体的な内容は、セキュリティ上の不都合、脆弱性や不都合によって起こる被害、その被害の緩和などである。しかしながら、これらは、他に優先して実践すべき事柄ではなく、かつ、セキュリティパターンの観点で、2.2 節にあるように一貫性のある構造で説明されているわけではない。

3.3.2 アタックパターン

[19, 20] の本は、ソフトウェアを破壊する方法に焦点を当てパターンを説明しており、アタックパターン [21] のカタログとみなすことができる。[19] では、19 個のアタックパターンを紹介しており、Web ブラウザなど特定のアプリケーションの例を説明している。また、[20] では、Web アプリケーションに関する 24 個のアタックパターンを説明している。これらの本では、パターンは、WHEN、WHAT、HOW という文書構造で説明されている。WHEN の節では、アタックが成功する状況が説明されている。これはプログラムの脆弱性を示しており、事前にテストされるべきである。WHAT の節では、アタックが成功する原因となる欠陥 (フォールト) が説明されている。プログラマは、この欠陥 (バグ、もしくは、デザイン上の誤り) を取り除く必要がある。HOW の節では、どのような手順で攻撃が行われ、また、それをどのように防ぐのかを説明している。

これらのアタックパターンは、直接的にはコーディングを支援していないが、実装を改善するために有益である。

3.3.3 セキュアリファクタリング

リファクタリングは、プログラムの修正に関する一種のパターンとみなすことができる。このリファクタリングに関して、いくつかセキュリティを扱った文献が存在する。例えば、Fowler のカタログ [22] は、オブジェクト指向プログラムに存在するセキュリティの弱点を取り除くリファクタリングを含んでいる。具体的には、Encapsulate Field リファクタリングは、他からアクセス可能な公開 (Public) フィールドを、非公開 (Private) フィールドに変更する。これによって、攻撃者がフィールドの値にアクセスすることを困難にする。さらに、そのフィールドの値がオブジェクトの生成時に設定され、以降変更がない場合、Remove Setting Method リファクタリングが適用できる。これは、値の設定メソッドを削除し、フィールドを final として宣言する。

丸山は、既存のコードのセキュリティレベルを向上させる 4 つリファクタリングを提案している [23]。このリファクタリングは、名前や動機の他に、Mechanics とコード例という文書構造により説明されている。Mechanics は、リファクタリングすべき状況と、どのような手順でリファクタリングを行うかを説明している。具体的には、悪意のある変更からフィールドを守るためのリファクタリング (Introduce Immutable Field) や、攻撃者から内部の値を守るためのリファクタリング (Replace Reference With Copy) などが紹介されている。

3.4 セキュリティパターンに関する工学

パターンのライフサイクルの観点から、パターンに関する工学は、パターンの摘出の適用の２つに分類できる。これは、セキュリティパターンであっても同様である。そこでこの節では、これらに関連す

る研究を紹介する。

3.4.1 セキュリティ要求とパターンの表現

2.2 節で説明したとおり、パターンはゆるく構造化された文書である。しかし、この構造はセキュリティの要求との対応が取れているわけではなく、その要求を満たす適切なパターンを選択するのは容易ではない。そのために、[24, 25] では、パターンと要求とを正確に関連付けし、パターンの適用を支援する。[24] では、セキュリティに関するオントロジと組み合わせることでセキュリティパターンの検索エンジンを提供している。このエンジンでは、パターンの状況 (Context) や問題 (Problem) に関してクエリを出することで、関連するパターンに加え、そのパターンの適用に必要な他のパターンや代用パターン、コンフリクトが起こるパターンなども検索可能である。

[25] では、セキュリティ要件を拡張 BNF に基づく文法で記述する方法を提案している。これにより、システムが必要とするセキュリティから、関連するセキュリティ要件を導くことが可能なる。さらに、セキュリティ要件を定義し、その選択とそれに関するセキュリティパターンの適用を支援するツールを提案している。

これらの研究は、パターンの摘出のための具体的なパターン記述と、それを適用する際のパターンの選択を主に支援する。

3.4.2 セキュリティパターンの分類と品質

分類・整理されたパターンは、セキュリティパターンの適用、特に適切なパターンを選択する際のガイドラインになる。3.1 節から 3.3 節では、パターンを開発工程に沿って分類した。また、同時にセキュリティの種類 (機密性、完全性、有効性、説明可能性) の観点でもパターンを整理したが、[26] でも同様の分類を行っている。[26] では、さらに、顧客、アーキテクト、デザイナなどのステークホルダに基づいた分類や、盗聴、成りすまし、DoS などセキュリティの問題の種類に基づいた分類も行っている。その他には、Konrad らが、セキュリティパターンを、生成、構造、振る舞いなどの観点とネットワークレベル、ホストレベル、アプリケーションレベルの抽象度によって整理している [2]。

これらの分類は熟練者が経験に基づき行うことを前提としているが、久保らは、パターンの分類や関連付けを自動化することを目指している [27]。

さらに、パターンの品質の分析法は、摘出したパターンのレビューや選択を支援する。[28] や [29] では、１０個のセキュリティ原則に従いパターンを分析している。Heyman らは、パターンの質を、適切さ、文章の質、流通の３つの観点で評価している。

3.4.3 セキュリティパターンの保存

セキュリティパターンのリポジトリも、パターン選択のために有用である。Yskout らは、[30] の中で、セキュリティパターンのカタログを提示しており、セキュリティパターンの品質への影響やパターン間の関係などもまとめている。

既存のオンラインリポジトリ [31, 32] にもセキュリティパターンは登録されている。これらは、パターンを摘出した結果の登録に有効である。しかしながら、セキュリティ特有の登録方法や検索方法を提供しているわけではない。

4 議論

この節では、3 節で示した既存のセキュリティパターンの技術がセキュアなソフトウェアの開発に十分なのかを議論し、どのようなパターンが今後必要になるかを 4.1 節で示す。さらに、4.2 節では、従来のパターンがシステムモデルにどれくらいうまく組み込めるのかを議論し、今後の研究の方向性を示す。

4.1 セキュリティパターンの十分性

図 1 に、2.1 節で説明したセキュリティの概念と提案されているセキュリティパターンの数をソフトウェア開発の各工程で整理した。各列は、概念を表し、各行は開発工程を表す。また、各セルには、各概念に関する各工程でのアクティビティとセキュリティパターンの充実度を表した。ここで、セキュリティパターンの充実度は、“+”, “++” の順で充実していることを示す。

この一覧から、下記のことが読み取れる。

- 要求・分析工程では、攻撃に関するパターンが多く、対策に関するパターンが少ない

Concept \ Phase	Requirements and Analysis Phase		Architecture and Design Phase		Implementation Phase	
Countermeasure	Identified		Feasibility	++	Implemented	++
Risk	Identified	+	Estimated		Measured	+
Threat	Identified	+	Feasibility		Realized	+
Attack	Identified	++	Feasibility		Tested	++
Attacker	Identified	++	Feasibility		Tested	+
Vulnerability	Identified	+	Feasibility		Realize	++
Asset	Defined	+	Designed with security	+	Implemented with security	+
Stakeholder	Defined	+	Reviews		Tests	
Security objective	Defined	+	Reviewed	+	Reviewed	

specification

図 1: セキュリティの概念と各開発工程におけるパターンの数

- アーキテクチャ設計や詳細設計工程では、攻撃に関するパターンと次にセキュリティ仕様に関するパターンが少ない
- 実装工程では、攻撃や対策に関するパターンが多いが、セキュリティ仕様に関するパターンが少ない

セキュリティの概念を網羅性することを考えると、これらの中で数が少ない部分のパターンを今後充実すべきと思われる。しかしながら、全ての概念に関して平等にパターンが有益なわけではない。[24] には、各工程間のセキュリティ概念のオーバラップを説明している。図 1 中の破線の丸がそれである*。セキュアシステムの構築を考えた場合、このオーバラップ部分のサポートが特に重要である。なぜなら、工程間のオーバラップ部分に関してパターンを連続的に適用することにより、セキュリティの要求から実装までのギャップを適切に埋めることが可能になるからである。パターンをスムーズにつなぐためには、パターン言語としての洗練が重要である。

この分析の結果、要求と設計工程のギャップを埋めるリスクや攻撃に関するパターンが特に必要であることが分かる。そのため、この領域のパターンを充実させることにより、セキュリティの仕様と実装のギャップを埋められることが推察できる。

しかしながら、設計レベルの脅威、攻撃、脆弱性を見つけることは一般に容易ではない。なぜなら、過去の攻撃の事例を見ても、内部の設計に及ぼす影響が分かりにくいためである。そのため、アタックパターンを適切に構築する方法が今後必要になってくるであろう。その方法として、攻撃のモニタリングの仕組みを活用し、設計の内部への影響を調べる手法や、通常の設計と同様、攻撃を設計する方法が考えられる。

*いくつかの工程については紙面の都合上、省略している。

4.2 セキュリティパターンとシステムとの融合

[1] には、セキュリティのためのソフトウェア工学の挑戦 (Challenge) として次の 2 点を挙げている。

- セキュリティとシステム工学の融合
- セキュリティとシステムモデルの融合

前者は、セキュリティパターンの充実により支援ができると思われる。しかしながら、セキュリティに関する分析や、セキュリティ方針・機能の選択には、攻撃に対するより深い分析や形式化が必要になるであろう。

後者は、セキュリティパターンに、システムモデルとの親和性を持たせることで支援できると思われる。現在、要求や設計工程でセキュリティをモデル化する手法として下記のような様々なモデルが提案されている。

- Problem Frame[33] による表現: Abuse frame、Anti-Requirements など
- i*/Tropos[34], Secure Tropos による表現: Attacker agent、信頼モデルなど
- UML による表現: Abuse case, Misuse case など

要求や設計工程のセキュリティパターンでは、これらをうまく使い分けることで、システムモデルとの親和性が高くなるであろう。このような研究は、既

にいくつか存在する。具体的には、セキュリティの要件を分析・定義するために、問題領域を定義するためのProblem Frameや、複数のステークホルダ間の要求を分析するTroposを使ったパターンが提案されている。

Troposでは、攻撃者をエージェントとすることにより攻撃の状況をモデル化することができる。Mouratidisらは、Troposを使って悪意のあるエージェント(Malicious Agent)からの攻撃をまもるためのパターンを提案している[35]。

さらに、Hateburらは、Problem Frameを使ったSecurity Problem Frameによりセキュリティ要求を明らかにしようとしている[36, 37, 38]。具体的には、Problem Frameの中にMalicious Subjectを設定することにより、重要な情報が他に洩れない性質を明確化している。

さらに、デザインのためのモデルとしては、UMLが標準になってきている。セキュリティをモデル化するための拡張として、UMLsec[39]やSecureUML[40]が提案されているが、パターンにもこれらのモデルを利用すべきであろう。

実際に、UMLを積極的にセキュリティパターンに使った提案もいくつかされている[9, 29, 41]。[9]や[29]では、形式的な定義を組み合わせることでセキュリティプロパティを厳密にチェック可能にしている。[41]では、パターンの状況(Context)にセキュリティを付加する前の配置図やシーケンス図を定義することで、セキュリティを付加すべきシステムの箇所の発見を容易にしている。

実装工程のモデルは、プログラムである。そのため、プログラムの解析や書き換え(リファクタリング)に利用がしやすいようなパターンの記述法が望まれる。

5 おわりに

本稿では、セキュリティパターンに関する技術を調査し、パターンを開発工程やパターン活動で整理した。さらに、それぞれが扱っているセキュリティの概念を分析し、パターンの十分性とシステムモデルとの融合性について議論した。その結果、設計工程の攻撃に関するパターンが特に今後必要であることが明らかになった。そして、パターンを表現するモデルについての今後の方向性を示した。

今後は、この本稿の考察を受けて、設計工程のアタックパターン、および、それぞれの工程にあったパターン表現と、それを使ったツールサポートについての提案を行う予定である。

参考文献

[1] Premkumar T. Devanbu and Stuart Stubblebine. Software engineering for security: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 227 – 239, 2000.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley, 1996.

[4] E. B. Fernandez and R.Pan. A pattern language for security models. In *Proceedings of PLoP 2001 Conference*, 2001.

[5] M. Schumacher, F. Buschmann E. B. Fernandez, D. Hybertson, and P. Sommerlad. *Security Patterns: Integrating Security And Systems Engineering*. John Wiley & Sons Inc, 2006.

[6] C. Steel, R. Nagappan, and R. Lai. *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall, 2005.

[7] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. John Wiley & Sons Inc, 2007.

[8] Miroslav Kis. Information security antipatterns in software requirements engineering. In *the PLoP 2002 conference*, 2002.

[9] Jan Jürjens, Gerhard Popp, and Guido Wimmel. Towards using security patterns in model-based system development. In *Proceedings of PLoP 2002 Conference*, 2002.

[10] E. B. Fernandez, J. C. Pelae, and M. M. Larrondo-Petrie. Security patterns for voice over ip networks. In *Proceedings of the International Multi-Conference on Computing in the Global Information Technology (ICCGI'07)*, 2007.

[11] M. Weiss. *Integrating Security and Software Engineering: Advances and Future Vision*, chapter Chapter VI: Modelling Security Patterns Using NFR Analysis, pages 127–141. Idea Group Publishing, 2006.

[12] E. B. Fernandez, T. Sorgente, and M. M. Larrondo-Petrie. Even more patterns for secure operating systems. In *Proceedings of PLoP 2006 Conference*, 2006.

[13] M. Bishop. *Computer Security: Art and Science*, chapter Chapter 29: Program Security, pages 869–921. Addison Wesley, 2003.

[14] M. G. Graff and K. R. Wyk. *Secure Coding: Principles and Practices*, chapter Chapter 4: Implementation, pages 99–123. O'Reilly, 2003.

[15] David A. Wheeler. Secure Programming for Linux and Unix HOWTO, 1999. http://www.dwheeler.com/secure-programs/.

[16] Gary McGraw and Edward Felten. Twelve rules for developing more secure java code, 1998. http://www.javaworld.com/javaworld/jw-12-1998/jw-12-securityrules.html.

[17] John Viega, Gary McGraw, Tom Mutdosch, and Edward W. Felten. Statically scanning java code: Finding security vulnerabilities. *IEEE Software*, 17(5):68–74, 2000.

[18] SunMicrosystems. Security code guidelines, 2000. http://java.sun.com/security/seccodeguide.html.

[19] James A. Whittaker and Herbert H. Thompson. *How to Break Software Security*. Addison Wesley, 2001.

[20] M. Andrews and J. A. Whittaker. *How to Break Web Software*. Addison-Wesley, 2006.

[21] Greg Hoglund and Gary McGraw. *Exploiting Software: How to Break Code*. Addison-Wesley, 2004.

[22] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[23] K. Maruyama. Secure refactoring: Improving the security level of existing code,. In *Proc. Int'l Conf. Software and Data Technologies (ICSOFT 2007)*, pages 222–229, 2007.

[24] Markus Schumacher. *Security Engineering With Patterns: Origins, Theoretical Models, and New Applications*. Springer, 2003.

[25] Kawin Supaporn, Nakornthip Prompoon, and Thongchai Rojkangsadan. An approach: Constructing the grammar from security pattern. In *Proc. 4th International Joint Conference on Computer Science and Software Engineering (JCSSE2007)*, 2007.

[26] Munawar Hafiz, Paul Adamczyk, and Ralph E. Johnson. Organizing security patterns. *IEEE Software*, 24(4):52–60, 2007.

[27] A. Kubo, H. Washizaki, and Y. Fukazawa. Extracting relations among security patterns. Submitted to 1st International Workshop on Software Patterns and Quality (SPAQu'07).

[28] John Viega and Gary McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, 2001.

[29] S. Konrad, B.H.C. Cheng, L.A. Campbell, and R. Wassermann. Using security patterns to model and analyze security requirements. In *International Workshop on Requirements for High Assurance Systems*, 2003.

[30] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen. An inventory of security patterns. In *Technical Report CW-469*. Katholieke Universiteit Leuven, Department of Computer Science, 2006.

[31] Microsoft. Patternshare. http://patternshare.org/.

[32] Inc. Cunningham & Cunningham. Portland pattern repository. http://c2.com/ppr/.

[33] M. Jackson. *Problem Frames: Analysing and structuring software development problems*. Addison Wesley, 2000.

[34] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *JAAMAS*, 8(3):203–236, 2004.

[35] Haralambos Mouratidis, Michael Weiss, and Paolo Giorgini. Modelling secure systems using an agent-oriented approach and security patterns. *International Journal of Software Engineering and Knowledge Engineering*, 16(3):471–498, 2006.

[36] D. Hatebur, M. Heisel, and H. Schmidt. Security engineering using problem frames. In *Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS)*, volume 3995, pages 238–253. Springer Berlin / Heidelberg, 2006.

[37] D. Hatebur, Maritta Heisel, and Holger Schmidt. A pattern system for security requirements engineering. In *Proceedings of the International Conference on Availability, Reliability and Security (AReS)*, pages 356–365. IEEE, 2007.

[38] D. Hatebur, Maritta Heisel, and Holger Schmidt. A security engineering process based on patterns. In *Proceedings of the International Workshop on Database and Expert Systems Applications (DEXA)*, pages 734–738. IEEE, 2007.

[39] Jan Jürjens. *Secure Systems Development with UML*. Springer, 2004.

[40] Torsten Lodderstedt, David A. Basin, and Jürgen Doser. Secureuml: A uml-based modeling language for model-driven security. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 426–441, 2002.

[41] N. Yoshioka, S. Honiden, and A. Finkelstein. Security patterns: a method for constructing secure and efficient inter-company coordination systems. In *Proceedings of Enterprise Distributed Object Computing Conference 2004 (EDOC'04)*, pages 84– 97, 2004.