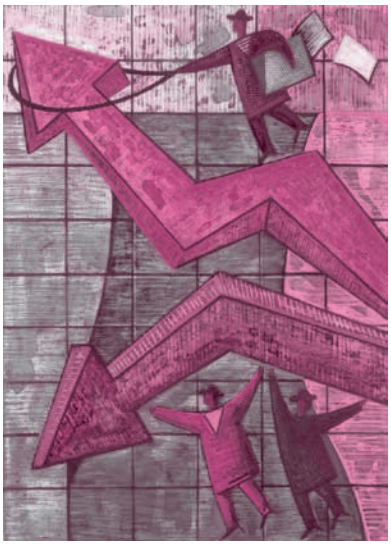

セキュリティの知識を共有するセキュリティパターン

吉岡信和 国立情報学研究所 GRACE センター

[セキュリティの知識とは?]

セキュリティ対策を施したソフトウェアを構築，提供することは，重要であり，現状のネット社会ではある種義務といっても過言ではない．2010 年に報告された日本での個人情報の漏えいに対する想定被害総額は 1,215 億円にも上る☆1．情報セキュリティは，社会全体の経済活動をも揺るがす関心事として避けて通ることはできない．しかし，セキュアなソフトウェアを開発するためには，セキュリティに関する専門知識を必要とする．さらに，新しい攻撃，新しい脆弱性が日々発見されるため常に新しいセキュリティの知識を共有して，ソフトウェアに適用する必要がある．そのため，そのようなセキュリティの知識を活用，共有する仕組みとして，ソフトウェアパターンは優れたツールとなり得る．

ソフトウェアパターンは，本特集のソフトウェアパターン概観でも述べているように，特定の局面・状況での問題に対する解法を表現できる．そのため，セキュリティに関する問題，すなわち攻撃，脆弱性やセキュリティの要求を明文化することができる．そして，その問題に対するセキュリティ対策を解法として指し示すことができる．言い換えれば，セキュリティの専門知識は，守ろうとするソフトウェアに対してどのような悪影響が起こる可能性があるかといった問題領域に関する知識と，なにをどのように守るかという解決領域の知識の 2 種類存在する．

☆1 JNSA 2010 年 情報セキュリティインシデントに関する調査報告書 http://www.jnsa.org/result/incident/2010.html

情報セキュリティの定義は，ISO/IEC 27001 では，「情報の機密性，完全性及び可用性を維持すること．さらに，真正性，責任追跡性，否認防止及び，信頼性のような特性を維持することを含めてもよい」としている．セキュリティの問題とは，このような特性を成り立たたなくさせる攻撃が発生するために起こる．たとえば，機密性を守りたい個人情報があった場合，その情報を漏えいさせる攻撃により，機密性が保たれなくなるというセキュリティ上の問題が発生する．

こういったセキュリティ上の問題を発見するのは深い専門知識がないと容易なことではない．新しい攻撃は日々発見されており知識は常に更新する必要がある．また攻撃の方法は複雑化し，巧妙になってきている．そして，攻撃を可能にするソフトウェアの脆弱性が，安全にしたいソフトウェア上に存在するかを調べるためには，対象となるソフトウェアの中身をよく知っている必要がある．ソフトウェアの作り方も時代によって変わってくる．C 言語であれば，バッファオーバフローといったメモリ管理の脆弱性をついた攻撃をまず考えなければならないが，Web を活用したソフトウェアでは，Cookie を使ったパスワード漏えいや，セッションハイジャックなどがあり得る．

機密性などのセキュリティに関する特性を保つためには，上記の問題が発生しないようにソフトウェア上の情報を守る必要がある．その方法には，暗号機能や認証フレームワークなどセキュリティの機能を駆使して情報を直接的に守る方法と，言語のバグやフレームワークの機能を悪用した攻撃を防ぐよう

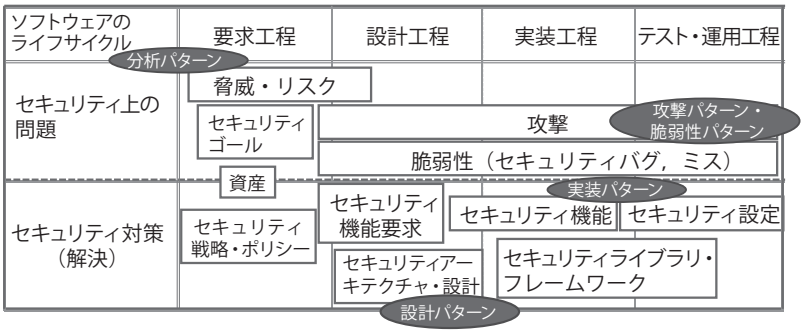

図 -1　セキュリティに関するパターン

に脆弱性をなくす方法の 2 種類ある．前者に対しては，セキュリティの設計に関するパターンがあり，後者に関しては，セキュアプログラミングなど実装に関するガイドラインが存在する．図 -1 に，ソフトウェアの開発・運用工程ごとに，問題・解決領域の関心事と，それぞれの関心事に関するパターンを分類した．

以下ではセキュリティ上の問題をどのように分析するかを示したパターン，攻撃や脆弱性に関するパターン，セキュリティ設計のパターン，攻撃を分析するためのパターン，脆弱性を防ぐ実装方法を示したガイドラインをそれぞれ紹介する．

[セキュリティ上の問題を分析するためのパターン]

セキュリティ上の問題を認識するには，そもそも情報の機密性，完全性および可用性といったセキュリティの特性がどれくらいの確率で攻撃により破壊されて，どれくらい被害をこうむる危険性があるかといったリスク☆2 を考える必要がある．リスクの高い問題は優先的に内容を分析し，対処する必要があるからである．セキュリティ上の問題の把握やその解決（対策）には，膨大なコストがかかる．起こり得ない問題（リスクがない事象）をいくら考えていても無駄なコストにしかならない上，問題を効率よく絞り込まなければ，タイムリーなソフトウェアの構築・提供ができなくなる．

安全にしたいソフトウェアシステムのセキュリティ上のリスクを明確にするためのパターンとして，文献 1）にはいくつかのパターンが紹介されている．セキュリティの特性を考えるべき情報（資産）に関するパターン（Security Needs Identification for Enterprise Assets），資産の価値を推定するためのパターン（Asset Valuation），資産に対する被害（脅威）の度合い，大きさを分析するためのパターン（Threat Assessment）などである．資産に関するパターンでは，資産として考えられる情報の分類として，個人情報，会計情報，契約に関する情報，特許など知財に関する情報，顧客に関する情報，公開情報の 6 種類を挙げており，対象となっているソフトウェアシステムに対してどのような情報を守るべきかを分析するガイドラインとなっている．資産の価値を推定するパターンでは，その価値をセキュリティの要求の度合い，金銭的な大きさ，ビジネスへの影響の大きさをそれぞれ 6 段階で評価し，総合的に推定する方法を述べている．被害を分析するパターンも同様に，脅威がどのような状況で起こるのかのガイドラインと，それに対する発生頻度を 6 段階で評価する方法を説明している．

これら分析に関するパターンは，情報システム一般に適用でき，特定の状況に適用できるパターンというよりも，情報システムを設計する際のガイドラインに近い．ネットゲームでは，ゲームに登場するアイテムが高額で売買されるなど，守るべき情報がビジネスの分野や時代の移り変わりにより異なってくる．また，情報システムを利用するユーザは，リテラシーが高い成人だけではなく，ゲームをする子供のように層が広がっており，人に関連した問題が発生する確率もさまざまな可能性が考えられる．今後は，情報システムが使われる多種多様な領域の分析パターンが整理されることが望まれる．

☆2　本稿では，セキュリティ上のリスクを，攻撃によって被害が起こる確率およびその被害の大きさの掛け算で定義し，「リスクがある」とは，その大きさが問題視すべき状態にある場合を言う．

[セキュリティを設計するためのパターン]

機密性など，情報(資産)に対するセキュリティ特性を守るためには，暗号機能や認証フレームワークなどセキュリティの機能を利用し，セキュリティを設計する必要がある．セキュリティ機能は，できる限り安全性が確認されている既存のものを利用し，独自に実装しないことが望ましい．セキュリティ機能の安全性は，多くの時間やコストをかけて確認され，国際標準になる．言い換えれば，脆弱性のないセキュリティ機能を作ることは難しく，独自の実装は，セキュリティホールを作る可能性が高いからである．たとえば，独自に考えた暗号アルゴリズムを深い検討，検証なしで用いることはソフトウェアの脆弱性を高めることになりかねない．しかし，安全が担保されたセキュリティ機能を使えば，どんなソフトウェアも安全になるとは限らない．セキュリティ機能の安全性は，秘密鍵が漏れない，鍵は容易に推測されないなど，特定の前提のもとで保証されている．その前提を崩すような使い方をしてしまえば，強力なセキュリティ機能を使っても脆弱なソフトウェアができてしまう．

セキュリティ機能をどのように使えばよいかの設計方針を示したセキュリティパターンがある．文献1）には，セキュリティを設計するときに役に立つ7種類，38パターンが紹介されている．具体的には，セキュリティの基盤やアプリケーションによって，IDの証明，アクセス制御モデル，アクセス制御アーキテクチャ，OSのアクセス制御，真証性確認・監査，ファイヤウォール，インターネットアプリケーションの7種類にセキュリティパターンを分類している．

情報の機密性を実現するために，一般に情報へのアクセスを制限するアクセス制御のメカニズムが利用される．その際に，機密の範囲を特定の個人に制限するだけではなく，役員やプロジェクトのメンバなど組織内に限定して情報を共有する場合も少なくない．そのような組織や役割ごとに情報のアクセス制御を行う方法としてロールベースアクセス制御が使われ，その使い方を説明したのが，ロールベースアクセス制御パターン（Role-Based Access Control Pattern）である[1)]．このパターンでは，図-2のようにロールベースアクセス制御を行う際のユーザと情報との静的な構造がクラス図によって示されている．そして，J2EE（Java 2 Enterprise Edition）やWindows 2000などのシステムでその制御法が採用されていることが説明されているため，このパターンによって，どのような実装や実行環境を選べばよいかの判断基準が得られる．

機密性を確保するためには，情報に誰がアクセスできるかといった制御方法を定義するだけではなく，実際にどのように情報を制御するかといったソフトウェアの振舞いを設計する必要がある．言い換えれば，許可のないユーザは，情報にアクセスできないようにソフトウェアの振舞いを制限する必要がある．セキュリティ設計の多くは，このようにソフトウェアの振舞いを制限するように，既存の設計を変更する必要がある．情報の提供を制限する方法として，セッションを用いる方法がある．

文献1）には，どのようにセッションを使うかを説明したセキュリティセッションパターン（Security Session Pattern）が紹介されている．図-3に，このパターンで説明されているセッションの管理方法のシーケンス例を示す．この例では，情報にアクセスするComponentに対して，認証を行うCheck Pointクラス，セッション管理を行うManagerクラス，そしてセッションを保持するSessionクラスを追加することにより，そのクラスの利用を制限している．このとき，認証のためのCheck Pointクラスを呼び出さずにComponentクラスのメソッドが呼び出せてしまうと，情報が漏えいする可能性がある．そのような脆弱性がないかどうかを分析するために，攻撃，脆弱性を見つけるためのパターンが有効になる．

文献1）で紹介されているパターンは，どちらかと言えば，セキュリティのアーキテクチャや考え方，概念的な設計に関するパターンである．これらのパターンを実装するためには，各言語や開発環境で提

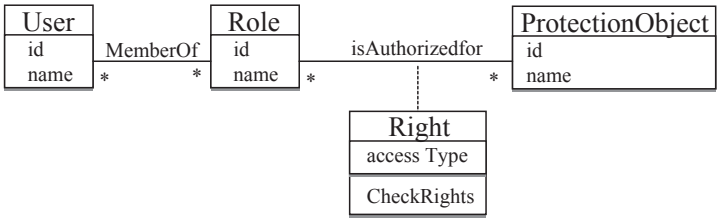

図 -2　ロールベースアクセス制御の構造

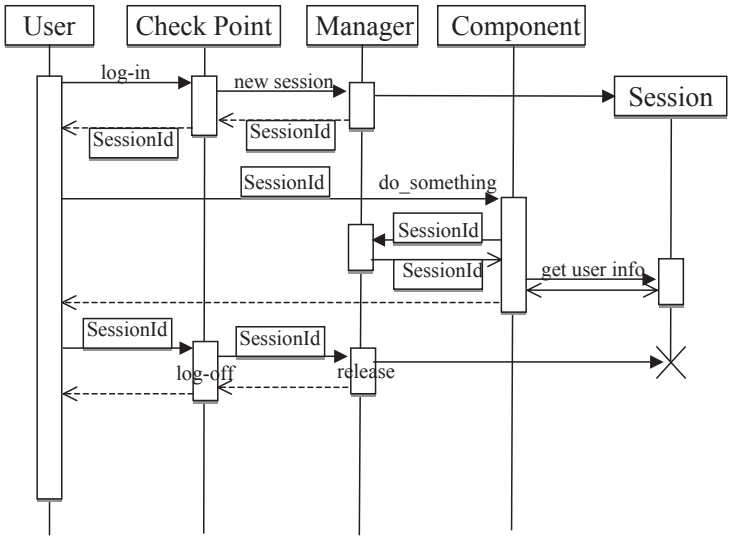

図 -3　セキュリティセッションパターン

供しているセキュリティ機能・フレームワークを使いこなす必要がある．そこで，言語やライブラリに依存した詳細設計・実装レベルのパターン，ガイドラインが重要になる．Java 言語に関しては，詳細設計や実装のガイドラインを示すパターンとして，文献 2）がある．この文献では，J2EE でセキュリティライブラリを使ってセキュリティを実装する場合の 23 個のパターンが紹介されている．

図 -4 に文献 2）で紹介されているセキュアロガーパターン（Secure Logger Pattern）のクラス図を示す．このパターンでは，安全にログを送信，保存するクラスを構築し，それを生成するクラス（Factory）に関連付けている．このようにセキュリティの詳細設計は，通常のデザインパターンを利用することで，セキュリティを考慮しない場合と切り替えて使うことが可能になり，保守性が保たれる．しかし，そのために新たな脆弱性ができる可能性がある．脆弱性があるかを分析するために，攻撃をパターン化した攻撃パターンが有益である．

［攻撃を知るパターン］

図 -1 に示した通りソフトウェアの開発工程ごとに攻撃され被害が起こる可能性がある脆弱性が混入してしまう可能性がある．要求工程では，内部犯など，ソフトウェアを取り巻く状況に関して脆弱性がないか考慮する必要がある．セキュリティ設計のミスや言語上のバグは，ソフトウェアの脆弱性につながる．脆弱性そのものの知識については次の章で述べるが，ここでは，脆弱性の発見に役立つ攻撃のパターンを紹介する．

設計に対する攻撃パターンの情報は実装や運用に比べて整理されていない．たとえば，SSL プロトコルに対する攻撃が見つかると，論文や報告書として公開される．しかし，それを他の同様のプロトコルに発生するかどうかを分析するために，パターン化されることはない．その理由は，攻撃を見つけるのは，それを専門とするセキュリティ専門家であり，パターン化せずとも論文や報告書で十分情報共有できているからである．少なくともセキュリティ機能の設計に関しては，それで十分であるが，最近多くなってきているセキュリティ機能以外の設計上の不備に対する攻撃に対しては，パターンによる知識の明文化は有効である．

セキュリティ以外の設計に対する攻撃のパターンとして，VoIP（Voice over IP）などアプリケーション領域に依存したものがまとめられている．図 -5 は，VoIP のプロトコルに関して DoS（Denial of Service）攻撃をかけたときのシーケンス図である [3]．この例では，PBX（構内交換機）に対して，電話の設定直後に無効なメッセージを送り続けることで，通話の接続を妨害している．文献 3）では，攻撃パターンの記述形式として，攻撃に対する対策だけではなくイ

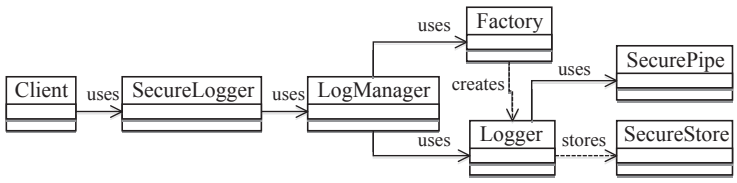

図 -4　セキュアロガーパターン

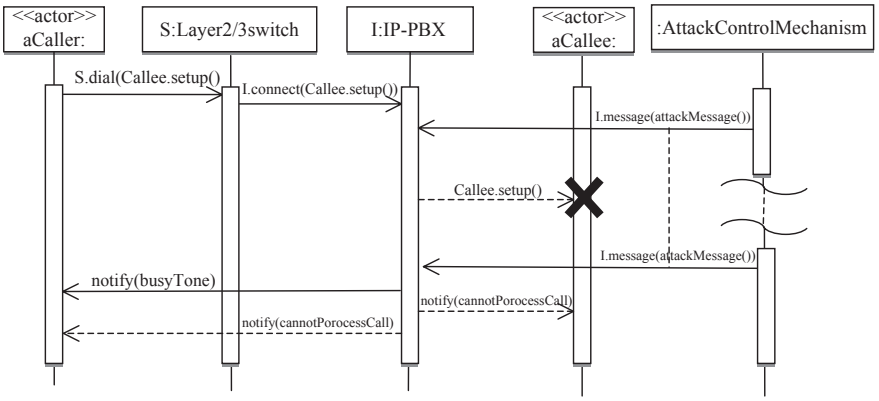

図 -5　DoS 攻撃のパターン

ンシデントが発生したときの犯罪の証拠（フォレンジック）の分析，取得方法も記述することを提案している．

実装の攻撃・脆弱性のパターンとして，多くの書籍や Web で情報が整理されている．文献 4）では，インジェクション攻撃や HTTP Cookie を利用した攻撃など，48 個のよく知られた攻撃パターンが整理されている．それぞれの攻撃に対して，実装上のどの部分が攻撃の対象となるか，どのような脆弱性が原因になるかを具体的なコード例や攻撃手順をもとに説明をしている．この書籍は 2004 年出版と比較的古いが，その多くが現在の開発においても考慮すべき攻撃となっている．

攻撃パターンを Web 上で共有する枠組みとして Common Attack Pattern Enumeration and Classification（CAPEC）☆3 がある．CAPEC では，既知の攻撃を，その分類，過酷さ，対応方針，関連する攻撃などとともに登録して情報を共有することができる．また，多数の攻撃の分類や関連を自動化するために，XML 形式で共有している．そのデータベースには，現在 380 パターン以上が登録されている．また，日本語の資料として，内閣官房セキュリティセンターが，代表的なマルウェアの振舞いパターンを分析し，公開している☆4．具体的には，Web 閲覧による感染など 4 つのマルウェアの振舞いパターンを説明し，その影響と問題点を論じている．

[脆弱性を作りこまないためのガイドライン]

実装上のバグに対する攻撃が発生しないように，脆弱性を作らないプログラミングを行う必要がある．すなわち，実装する言語・ライブラリ特有の原因で脆弱性が発生する可能性がある．たとえば，C 言語のメモリ管理機構の貧弱さにより，バッファオーバフローの脆弱性が発生する．C 言語では多くの脆弱性が発見されており，文献 5）では，C 言語でプログラミングする際に，脆弱性が入り込まないようにするためのプログラミングガイドラインが示されている．たとえば，バッファオーバフローを発生させないための，sprintf 関数の書式の指定方法などである．

発見された脆弱性の情報を共有する仕組みには，脆弱性対策情報 DB JVN iPedia☆5，The Open Source Vulnerability Database（OSVDB）☆6，Common Weakness Enumeration（CWE）☆7 などがある．CWE には現在，690 個以上の脆弱性が登録されており，脆弱性ごとに ID が振られ，他の脆弱性との関連やコード例，対策などが記載されている．OSVDB に登録された脆弱性の傾向では，図 -6 にあ

☆3 http://capec.mitre.org/
☆4 リスク要件リファレンスモデル作業部会報告書 http://www.nisc.go.jp/inquiry/index.html
☆5 http://jvndb.jvn.jp/
☆6 http://osvdb.org/
☆7 http://cwe.mitre.org/

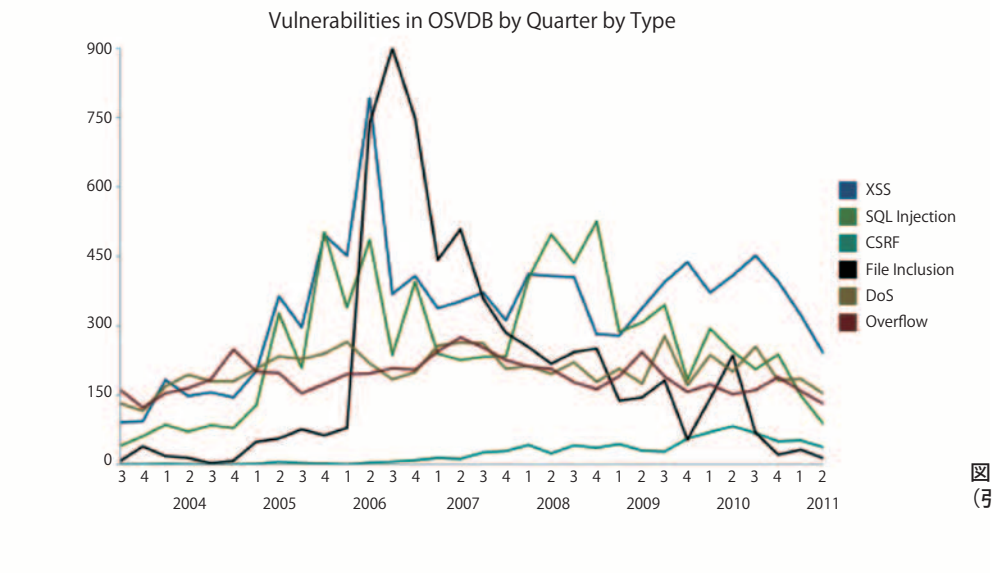

図 -6　脆弱性の登録件数
（引用元：http://osvdb.org/）

る通り 2009 年からクロスサイトスクリプティング（XSS）が上位になってきており，ファイルインジェクションが 2006 年をピークに減少傾向になっている．

[セキュリティパターンの展望]

現在，提案・整理されているパターンを使いこなすにはまだまだ難しい．具体的には，パターン適用時の難しさと，ソフトウェアの開発に合わせたパターン連携の難しさがある [6]．

実装レベルのパターンは数が多く，必要なパターンを見つけ出すのが困難である．設計レベルのパターンはそれほど数があるわけではないが，記述が抽象的なため，設計方針は理解できても，それを具体的にどう実装に落としていけばよいかが分かりにくい．なぜならば，現状，設計方針，詳細設計，実装のそれぞれのパターン間での関連が定義されていないため，設計時に選ばれたパターンがどう実装と関連するのかが明確ではない．また，実装レベルで発見させる攻撃パターンが，設計とどう関係するかが明確になっていないため，対策のパターンが十分適用できているかどうかが不明確になっている．

今後，ソフトウェアの開発を通して，セキュリティ上の問題発見からその解決（対策）へ導く一連の知識活動を，ソフトウェアパターンがサポートしてくれることが望まれる．

参考文献

1）Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F. and Sommerlad, P. : Security Patterns : Integrating Security and Systems Engineering, Wiley (2006).
2）Steel, C., Nagappan, R. and Lai, R. : Core Security Patterns : Best Practices and Strategies for J2EE, Web Services, and Identity Management, Prentice Hall (2005).
3）Fernandez, E., Pelaez, J. and Larrondo-Petrie, M. : Attack Patterns : A New Forensic and Design Tool, IFIP International Federation for Information Processing, Vol.242, pp.345-357 (2007).
4）Hoglund, G. and McGraw, G. : Exploiting Software : How to Break Code, Addison-Wesley Professional (2004).
5）久保正樹 , 戸田洋三：JCERT C セキュアコーディングスタンダード，アスキー・メディアワークス（2009）．
6）Yoshioka, N., Washizaki, H. and Maruyama, K. : A Survey on Security Patterns, Progress in Informatics, National Institute of Informatics, No.5, pp.35-47（May 2008）．

（2011 年 7 月 7 日受付）

吉岡信和（正会員）　nobukauz@nii.ac.jp

1998 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．博士（情報科学）．同年（株）東芝入社．2002 年より国立情報学研究所に勤務，現在，同研究所 准教授，2007 年より総合研究大学院大学 准教授を兼務，セキュリティ技術，エージェント技術，ソフトウェア工学の研究に従事，2011 年より日本ソフトウェア科学会理事，現在に至る．日本ソフトウェア科学会，電子情報通信学会各会員．