

オープンソース・ソフトウェアのセキュリティ確保に関する調査

平成 15 年 2 月 3 日

1 はじめに

情報インフラの構成要素として、オープンソース・ソフトウェア（Open Source Software）が利用されるケースが増えている。その一方で、オープンソース・ソフトウェアのセキュリティが重要な問題となってきた。オープンソース・ソフトウェアでは基本的にセキュリティは保証されておらず、自己責任において利用されるべきものとされている。しかしオープンソース・ソフトウェアのセキュリティについて深く検討もせず導入を行ったために、その脆弱性をつかれて、不正にアクセスされる例は年々増大しており、利用者側でのオープンソース・ソフトウェアのセキュリティ確保が急務とされてきている。

オープンソース・ソフトウェアのセキュリティに関しては、オープンソース・ソフトウェアを利用する側と、作成する側の 2 側面から考える必要がある。オープンソース・ソフトウェアを作成する側からは、ISO/IEC 15408 などの評価制度がセキュリティ確保の手段として挙げられる。しかし、オープンソース・ソフトウェアは開発そのものがオープンであり、商用のソフトウェアの開発とは違った開発モデルが採用されており、必ずしも ISO/IEC 15408 には適していない。ISO/IEC 15408 では保証要件として技術的要素だけでなく、開発体制などの保証要件も要求されるため、オープンソース・ソフトウェアの開発には適用しにくい。一方、オープンソース・ソフトウェアを利用する側からすればこうした側面が、セキュリティの問題となる。コストが低いから仕方ないという考え方もあるが、利用する以上、オープンソース・ソフトウェアのセキュリティを自己責任において評価するための仕組みやガイドが示されていることが重要である。

2 目的

オープンソース・ソフトウェアの利用は今日ビジネスにおいても急激に増大しており、オープンソース・ソフトウェアのセキュリティ確保のための技術的、方策的な対策に対する重要性は増している。本調査では、オープンソース・ソフトウェアの特徴を考慮し、オープンソース・ソフトウェアの利用者（SIベンダー等を含む）の立場から、オープンソース・ソフトウェアを安全に利用するための技術的、方策的な項目について調査を実施し、オープンソース・ソフトウェア利用者に有用な情報を取りまとめ、オープンソース・ソフトウェアの利用に際してセキュリティを確保するためのフレームワークを作成することを目的とし、実施されたものである。

3 セキュリティ確保のためのフレームワーク

オープンソース・ソフトウェアの利用におけるセキュリティの確保において最も重要なことは、オープンソース・ソフトウェアのリスクを自己の管理において把握し、対象のシステムに応じて適切にリスクを把握し、低減の手段を講じていく事である。ここでいうリスクとはオープンソース・ソフトウェアが抱える脆弱性を指す。オープンソース・ソフトウェアは商用のソフトウェアと比べてセキュリティが低いわけではないことは前述したとおりである。しかしながら、オープンソース・ソフトウェアも商用のソフトウェアもソフトウェアであることに変わりなく、数量的な違いはあれ、セキュリティに対するリスクが存在する。商用ソフトウェアの場合はリスクをエンドユーザが把握する手段は少ない。それはソースコードそのものが見えないため、リスクを把握することが困難であるということである。一方オープンソース・ソフトウェアでは、ソースコードが公開されているために、さまざまな手段を利用して、リスクをある程度把握し、さらにそれに応じた対応

策を講じることが可能となる。オープンソース・ソフトウェアの特徴である、ソースコードの開示の特性は、こうしたセキュリティの管理において非常に重要な意味を持つこととなる。

オープンソース・ソフトウェアの利用におけるセキュリティ確保を形式化するために、以下のようなオープンソース・ソフトウェア利用の3ステップからなるセキュリティ・フレームワークを定義する。

第1段階：ソースコード検査

第2段階：脆弱性対応

第3段階：運用管理

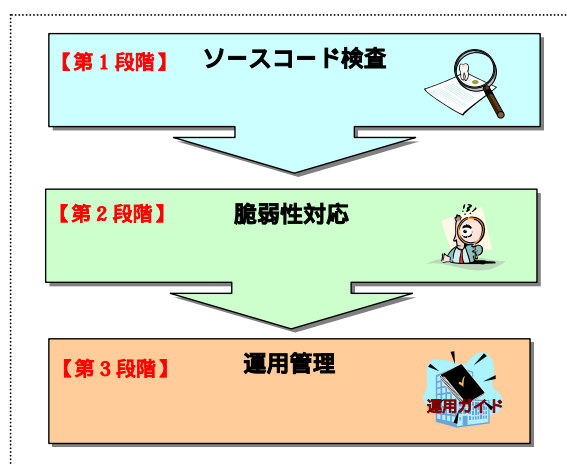


図1. セキュリティ・フレームワーク

4 ソースコード検査

オープンソース・ソフトウェアでは一般的な商用ソフトウェアと違い、ソースコードを利用することが可能である。そこで、利用者側で、ソースコードを利用して、ソフトウェアに含まれているバグや脆弱性の検査を行うことが可能である。この検査の結果に従い、脆弱性の評価を行い、システムのセキュリティ上のリスクの把握を行うことは、利用者にとって重要である。しかしながら、ソースコードの検査を手作業でやることは、非常に困難で専門的な知識を必要とする。そこで、こうした検査をシステムで自動的に検出するツールの存在が重要となると考えられる。現在、ソースコードの検査を効率的に行うための幾つかのツールがオープンソース・ソフトウェアとして公開されている。ソースコード検査技

術は大きく2つに大別され、それぞれ以下のようなツールが開発されている。

《パターンマッチング技術》

脆弱性に関するデータベースを元に、ソースコード中に脆弱性を持つ関数や変数が使用されているかどうかを検査する技術である。検査は、データベースに登録された脆弱性情報とソースコードのパターンマッチングによって行われる。

- (1) RATS (Rough Auditing Tool for Security)
- (2) ITS4 (It's the Software Stupid Source Scanner)
- (3) Flawfinder

《構文解析技術》

ソースコードをそのまま検査するのではなく、構文解析を行い抽象化した構文木に変換し、検査を行う技術である。従来の構文解析技術に、セキュリティ検査の為に機能を付加したものである。セキュリティ検査のための付加的な注釈を使用し、このデータ遷移の安全性まで検査可能である。

- (4) Splint (LCLint)
- (5) Cqual

ソースコードの検査技術は、ソースコードレベルでの脆弱性の検出を効率的に行えるようにサポートしてくれる。その成熟度はまだ低いものの、ソースコードの検査という非常に重要な作業を効率的に行うための基盤として非常に重要であり、利用者の立場からすれば、よく知られた脆弱性を利用するオープンソース・ソフトウェアに含まれていないかを簡易的にチェックするためにも有益である。

5 脆弱性対応

オープンソース・ソフトウェアのセキュリティを確保するためには、前章で説明したソースコードの検査技術だけでは不十分である。たとえソースコードの検査によってセキュリティに関連する問題を検出したとしても、それを利用者側で修正することは、事実上困難な場合もあり、ソースコードを直接修正することなく、検査によって検出されたセキュリティの問題に対応するための技術が重要となる。ここでは、オープンソース・ソフトウェアのセキュリテ

ィを確保するために、利用者がソースコードそのものを修正することなく、セキュリティを確保するための技術には以下のようなものがある。

《カーネル技術》

プログラムの実行環境であるオペレーティングシステムそのものを拡張して、プログラムをセキュアに実行するための環境を提供するための技術である。

- (1) Solar Designer

《コンパイラ技術》

GCCなどのコンパイラを拡張して実行コードを拡張して、セキュアなコードを生成する技術である。

- (2) StackGuard
- (3) Bounds Checking
- (4) Stack Smashing Protector

《ライブラリ技術》

アプリケーションや、オペレーティングシステムそのものを拡張するのではなく、アプリケーションで利用されるライブラリを拡張し、プログラムをセキュアに実行するようにするための技術である。

- (5) FreeBSD stack integrity patch
- (6) Libsafe

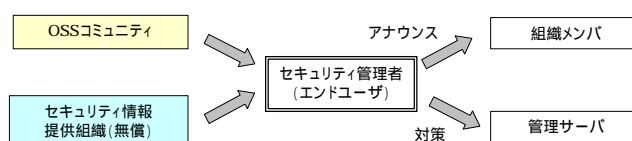
セキュアな実行コード・実行環境技術は、ソースコード検査の結果を踏まえて、オープンソース・ソフトウェアの安全性を高めるための技術として非常に重要である。本調査では、こうした技術を取り上げ、実際のオープンソース・ソフトウェアに適用してその効果について調査したが、すべての脆弱性に対応はできないものの、特定の脆弱性には非常に効果がある。少なくともソースコードの検査で検出される脆弱性をカバーするための技術としては整備が進んできている。

6 運用管理

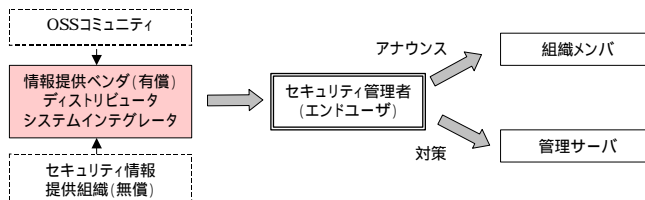
技術的な対策だけでは、十分ではない。ソフトウェアのセキュリティは固定的なものではなく、時間とともに変化するものである。この変化に対応するためには、ソフトウェアを適切に運用していくことが重要である。ソフトウェアに脆弱性が検出された場合は、速やかにパッチをあて、ソフトウェアをア

ップデートしなければならないし、そうした情報を積極的に利用しなければならない。オープンソース・ソフトウェアの運用は、商用のソフトウェアに比べて、自由度が高い。つまり、利用者が自分でさまざまな対応を取ることが可能である。そのため、どのような体制でオープンソース・ソフトウェアを運用していくかについて、コストや体制を踏まえて検討する必要がある。本調査では3つのオープンソース・ソフトウェア運用のモデルを提案した。

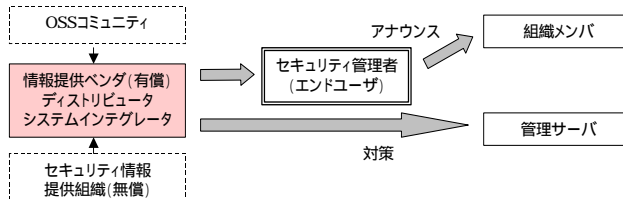
《モデル1》 自組織のみで情報の収集と対応をするケース



《モデル2》 有償外部リソースから情報を収集して、自組織で対応をするケース



《モデル3》 有償外部リソースに情報の収集と対応を依頼するケース



3つのモデルを通じてセキュリティ管理者が行わなければならないことを整理すると以下の3項目が挙げられる。

- (1) セキュリティ情報の収集
- (2) セキュリティ情報への対応(パッチ適用等)
- (3) インシデントへの対応

どのモデルで実際に自己のオープンソース・ソフトウェアを運用していくかによって、上記3項目のどの部分を自己で実施し、どの部分を有償で外部に委託するのか変わってくるため、コストも異なれば、自組織に構築しなければならない体制も実施すべき作業項目も変わってくる。しかしサービスの中には従来の運用形態とまったく同じ形態をとるものもある。そのためオープンソース・ソフトウェアの利用者は、安全にオープンソース・ソフトウェアを運用していくための方策を理解していくとともに、上記3つのモデルをベースに自組織でどのように運用に取り組んでいくのか、コスト、技術等を見て検討する必要がある。

7 まとめ

本調査で提案したフレームワークは、利用者の立場からすれば、技術的には高度で適用できない場合もある。また、利用者が個別で同じオープンソース・ソフトウェアのセキュリティを検討することは効率的ではない。こうしたことを考えると、本フレームワーク自身もオープンなものとすることが重要である。つまり、つまりソースコードの検査や、セキュアなコード生成・実行環境技術などについての情報をオープンに公開・利用できる環境が重要になると思われる。米国では、Sardonixのように、オープンソース・ソフトウェアのソースコード検査の結果を公開している組織などが政府の支援の下で運営されている。こうした組織により、オープンソース・ソフトウェアのセキュリティに関しての情報を共有するための場の構築が今後非常に重要となると思われる。