# バイナリコードから脅威度を推定する脆弱性検出ツールの実装と評価

今井祥子 †　佐藤喬 †　多田好克 †
† 電気通信大学大学院情報システム学研究科

## 1. はじめに

現在、インターネット上では有益なプログラムが多数公開されている。しかし、それらのプログラムには脆弱性が含まれている可能性があり、ユーザに被害が及ぶ場合も存在する。CERT/CC や JPCERT/CC 等のサイトには多数の脆弱性が報告されており、また、広く使用されているプログラム内にも脆弱性が発見されている。例えば解凍ツール Lhaca[1]のバージョン 1.2.0 では、バッファオーバフローの可能性がある strcpy 関数を使用していたため、それを利用した攻撃者がシステムを乗っ取ることが可能になってしまった。他にも iTunes や Flash Player などからも脆弱性は発見されている。ゆえにプログラムの安全性を使用前に把握することが望まれるが、それは容易ではない。公開されているプログラムの多くが、ソースコードが公開されていないバイナリコード形式で配布されているためである。

本研究では、利用前にプログラムの安全性を把握可能にするため、バイナリコードを解析してその安全性を容易に計るシステムを提案する（図 1）。コード内に存在する脆弱性の原因となる関数とその使用法を分析し、それを点数化して分かりやすい形で出力する。

## 2. システムの設計

以下で本システムの検査対象や、バイナリコードの解析方法、検出結果の表示方法について述べる。

### 2.1. 検査対象

同一のソースコードを元にしていても、CPU アーキテクチャやコンパイラ、OS によって作成されるバイナリコードは異なってしまうため、今回は x86 アーキテクチャ上の Linux にて gcc でコンパイルされたプログラムを検査対象とした。

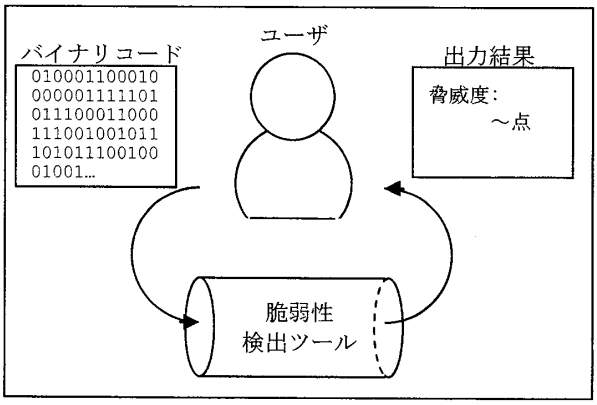


図 1　システムの概要図

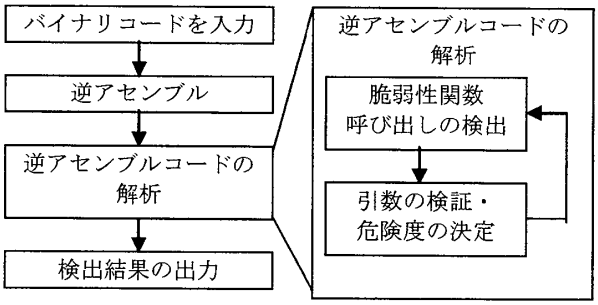


図 2　処理の流れ

### 2.2. バイナリコードの解析方法

バイナリコードはただの 0 と 1 の並びであるため、何らかの変換を行わなければ情報が抽出しにくい。本研究ではコード内で使用されている命令の把握を容易にするため、逆アセンブルを行うこととした。

本システムの処理の流れを図 2 に示す。逆アセンブルしてバイナリコードを解析しやすい形にした後、脆弱性を含む関数を呼び出している箇所を検出する。検出対象となる関数の情報は、既存のソースコード検査ツール ITS4[2][3]のデータベースから取得した。ITS4 はオープンソースのソースコード検査ツールであり、パターンマッチングによって脆弱性を含む関数の使用箇所を検出している。それだけではなく、その関

---

Implementation and Evaluation of Vulnerability Detection Tool Estimating the Risk of Binary Code
† Shoko Imai　† Takashi Satou　† Yoshikatsu Tada
† Graduate School of Information Systems, The University of Electro-Communications

数の呼び出し時に使用された引数の検証も行っている。これは引数によって脆弱性の危険度が左右されるためである。本システムではこの引数検証方法を逆アセンブルコードに応用した。

### 2.3. 検出結果の表示方法

データベース内にある、各脆弱性関数の危険度別に個数を表示する。また、それぞれに重みをつけ、全体的な点数も算出する。

また、詳細な情報を知りたいといった場合に備え、オプションといった形で詳細情報を出力可能とした。この情報は主に発見された脆弱性関数の危険性についてであり、また、脆弱性関数がどこから呼ばれているのかといった情報も含んでいるため、開発者へのフィードバックとして使用可能である。

## 3. システムの実装

先に述べたとおり、gcc でコンパイルされた C のプログラムを対象にし、Java 言語にて開発を行った。結果、1500～1800 行程度のプログラムとなった。また、逆アセンブルには Linux で使用されている既存のツール objdump を使用した。以下で逆アセンブルコードの解析について詳しく述べる。

### 3.1. 逆アセンブルコードの解析

逆アセンブルコードを解析し、脆弱性を含む関数を使用している箇所を検出する。また、呼び出し時に使用された引数の検証を行うことで脆弱性の危険度の調整も行う。

まず引数検証の前準備として、逆アセンブルコード内のリードオンリーデータセクションを解析する。ここには静的に定義された文字列のデータが格納されているため、それを解析、取得しておく。この情報は後の引数検証時に使用する。

次にテキストセクションを解析する。テキストセクションはプログラムのコードが書かれた領域であり、ここから call 命令を使用して脆弱性関数を呼び出している箇所を検出する。次にその call 命令の前に行われている、引数をスタックに置く命令を検出することで引数を把握し、検証する。引数検証方法は大きく分けて二つあり、一つ目は静的に定義された文字列を使用しているかどうかを判別するものである。これは引数として指定されたものがリードオンリーデータセクション内にあるかどうかを検証することで実現している。また、文字列データの内容自体を検証する場合もある。二つ目は同一ファイル名に対して処理を行っているかどうかというものであり、これは同一の領域を引数として指定しているかどうかを検証することで実現している。この二つの引数検証方法を使用して検出された脆弱性関数の危険度を調整し、検出された脆弱性関数を危険度別に整理して個数を計算、出力する。

## 4. 関連研究

ITS4 は先に述べたようにオープンソースのソースコード検査ツールであり、パターンマッチングで脆弱性関数を検出している。対象となる脆弱性関数はバッファオーバフローやレースコンディションを引き起こす関数、乱数生成に関する関数、ユーザが悪意あるコードを埋め込む可能性がある関数、フォーマット文字列の脆弱性をもつ関数で、とくにレースコンディションの検知に力を入れている。しかし、ITS4 ではソースコードを検査対象としているため、ソースコードが公開されていなければプログラムの安全性を判断することが出来ない。

また、Tevis ら[4]は Windows におけるバイナリコードの脆弱性検出を試みた。逆アセンブルは行わず、ヘッダ情報のみから脆弱性を検査する。脆弱性関数の使用の仕様も検出するが、ヘッダ情報のみでは引数の検証が不可能であるため、誤検出が多く含まれてしまう可能性がある。

## 5. 今後の予定

予定している実験としては、ITS4 と本システムでの脆弱性検出精度の比較や、引数検出精度の検証、実行時間の測定があげられる。また、今後の展望としては、フロー解析を用いた脆弱性のより精密な検出や、Windows への移植、実行時間の短縮等が考えられる。

## 6. まとめ

本論文では、バイナリコードの脆弱性検出を試み、ツールの設計・実装を行った。これにより、ソースコードが公開されていないプログラムにおいても、使用者がその安全性を確かめることが可能になる。

## 参考文献


[1] Lhaca http://park8.wakwak.com/~app/Lhaca/

[2] ITS4, http://www.cigital.com/its4/

[3] Amitabh Srivastava and Alan Eustace, "Token-based scanning of source code for security problems," ACM SIGPLAN Notices Volume 39, Issue 4(April 2004) Best of PLdl 1979-1999 SPECIAL ISSUE:1994 pp. 528-539.

[4] Jay-Evan J. Tevis and John A. Hamilton, Jr., "Static analysis of anomalies and security vulnerabilities in executable files," Proceedings of the 44th annual Southeast regional conference, pp. 560-565, 2006.