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報告書

第Ⅴ部

オープンソース・ソフトウェアの効率的な検査技術の利用ガイド

- RATS -

情報処理振興事業協会

セキュリティセンター

目　次

1. はじめに

RATS（Rough Auditing Tool for Security）は、米国セキュアソフトソリューションズ社によって開発が行われているオープンソース・ソフトウェアである。GPL[1]の規約の元でコピー・再配布が許可されており、容易な入手・利用が可能である。RATS では C、C++、Perl、PHP、Python といったさまざまな言語のソースコードの検査を行うことが可能である。本利用ガイドでは、C 言語で書かれた UNIX 上のソースコードを対象とした利用方法を述べる。尚、使用した RATS のバージョンは 2.1 である。

2. インストール

2.1. RATS のインストールに必要となるツール

RATS をインストールするにあたっては、XML のパーサライブラリである expat がインストールされていることが必要である。インストールされている expat のバージョンについては、特に指定はされていない。 expat が原因で RATS の動作に不具合が生じる場合は、下記の要領で最新の expat を導入することが推奨される。

2.1.1. expat

expat は James Clark 氏によって開発されたオープンソース・ソフトウェアである。ライセンスは、MIT/X Consortium であり、自由に利用可能である。expat は C 言語で書かれた XML パーサである。RATS では脆弱性に関して記述されたデータベース（脆弱性データベース）が XML 形式で記述されている。その為、RATS では expat を利用して、起動時に XML 形式の脆弱性データベースが取り込まれる。

2.1.2. expat の入手

expat は以下のアドレスから入手可能である。

http://prdownloads.sourceforge.net/expat/expat-1.95.6.tar.gz?download

[1] 本報告書では特に断らない限り、GNU Gneral Public Licence Version2 を GPL2 と表記する。

最新バージョンは 1.95.6 であり、ソースコードの他にバイナリ形式として RPM での配布が行われおり、Red Hat 系 Linux のユーザは、これを利用することが可能である。RPM は以下のアドレスから入手可能である。

```
http://prdownloads.sourceforge.net/expat/expat-1.95.6-1.i386.rpm?download
```

2.1.3. expat のソースコードからのインストール方法

expat をコンパイルして、インストールする為には、以下のコマンドを実行する。標準では/usr/local ディレクトリ以下にインストールされる。最新のバージョン（1.95.6）の expat のインストールを Red Hat Linux 7.2 で行った例を、以下に示す。

```
% ./configure
% make
% make install
```

インストール先を変更する場合には、以下のようにコマンドを実行する。

```
% ./configure --prefix="ディレクトリ名"
% make
% make install
```

RATS では、expat のライブラリのみを使用する。その為、expat のライブラリのみをインストールすることで、RATS は利用可能である。

2.2. RATS のインストール

2.2.1. RATS の入手先

RATS は以下のアドレスから入手可能である。

```
http://www.securesoftware.com/download_form_rats.htm
```

RATS を入手する際には、氏名・所属組織・連絡先（電話番号、E-mail アド

レス）の入力が要求される。幾つかの情報の提供が必要となるが、入手できるソフトウェアのライセンスはGPLに基づいており、この条件下で誰でも利用・再配布が行える。

ソースコードだけでなく、win32 実行形式のファイルも配布されている。現在の所、win32 環境以外でのバイナリ形式の配布は行われていない。

2.2.2. インストール方法

以下のようなコマンドを実行し、ソースコードをコンパイルし、インストールする。以下の例では、インストールを Red Hat Linux 7.2 上で行うことを想定している。

```
% ./configure
% make
% make install
```

通常では、RATS は/usr/local/bin ディレクトリ以下にインストールされる。脆弱性データベースは/usr/local/share ディレクトリ以下にインストールされる。これらのインストール先を変更したい場合には、以下のようにする。

```
% ./configure --prefix="ディレクトリ名"
% make
% make install
```

また、RATS 本体と脆弱性データベースのインストール先をそれぞれ個別に定めたい場合には、以下のようにする。

```
% ./configure --bindir="RATS のインストール先" --datadir="DB のインストール先"  ¥
% make
% make install
```

RATS に付属されている README ファイルには、脆弱性データベースは/usr/local/lib ディレクトリ以下にインストールされると記述されている。しかし、これは誤りであり、実際には/usr/local/share ディレクトリ以下にインストールされる。

尚、インストール作業時に必要とされる expat は/usr/local/lib ディレクトリ以下と/usr/local/includeディレクトリ以下に導入されていると想定されている為、それ以外のディレクトリに expat を導入している場合には、--with-expat-lib または--with-expat-include オプションを configure コマンド実行時に指定する。

```
% ./configure --with-expat-lib="ディレクトリの絶対パス"¥
 --with-expat-include="ディレクトリの絶対パス"
% make
% make install
```

3. RATS の使用方法

3.1. 基本的な使い方

RATS を使ってソースコードの検査を行うためには、以下のようにコマンドを指定する。

```
% rats　"検査を行いたいディレクトリ名、またはファイル名のリスト"
```

ディレクトリ名を指定した場合（もしくはシェルのメタキャラクタを指定した場合)、RATS は該当するディレクトリ中の.c、.cpp、.pl、.php、.py といった拡張子のファイルの検索及び検査を自動的に行う。従って、ファイルの検査を一括で行う場合、ディレクトリ名を指定する。

標準では、指定したディレクトリのサブディレクトリ下のファイルに対しても検査を行う。サブディレクトリに対する検査を行わない場合は、以下のようにする。

```
% rats -R　"検査を行いたいディレクトリ名、またはファイル名のリスト"
```

3.2. 出力のカスタマイズ

RATS は検査終了後に、標準出力に検査結果を出力する。出力結果の概要は以下のようである。

```
ヘッダー
各ファイルのステータス情報
各ファイルの検査結果
フッター
```

最初にヘッダーが出力される。ヘッダーには、情報として脆弱性データベース及び検査対象となるファイル名の一覧が含まれる。ヘッダー情報の出力後に

検査機能が起動し、検査が行われる。検査が行われるファイル名は、各ファイルの検査実行毎にファイルのステータス情報として出力される。全てのファイルに対しての解析が終了した後、検査が行われた順にファイルの検査結果が出力される。検査機能が検査結果を出力した後に、検査に要した時間、総検査行数、一行あたりの検査時間がフッターとして出力される。

ヘッダー、フッター情報の出力は省略することが可能である。ヘッダー情報の省略は、以下のようにする。

```
% rats --noheader   "検査を行いたいディレクトリ名、またはファイル名のリスト"
```

フッター情報の省略は、以下のようにする。

```
% rats --nofooter   "検査を行いたいディレクトリ名、またはファイル名のリスト"
```

同様に、ファイルのステータス情報は以下のようにして省略することが可能である。

```
% rats --quiet   "検査を行いたいディレクトリ名、またはファイル名のリスト"
```

上記の３つの情報を同時に省略するには、以下のようにする。

```
% rats --resultonly   "検査を行いたいディレクトリ名、またはファイル名のリスト"
```

上記のように付属情報を省略することで、各ファイルの検査結果のみを出力することが可能である。

RATS では、検査結果の出力をテキスト形式だけでなく XML 形式と HTML 形式で行うことができる。XML 形式で出力を行うには、以下のようにする。

```
% rats --xml   "検査を行いたいディレクトリ名、またはファイル名のリスト"
```

HTML 形式で出力を行うには、以下のようにする。

```
% rats --html   "検査を行いたいディレクトリ名、またはファイル名のリスト"
```

以下に、HTML 形式の出力を Web ブラウザで表示した例を示す。

3.3. コマンドオプションによる柔軟な検査

3.3.1. 指定した関数の検査

-a オプションを使うことで、ある関数名がプログラム中に使用されているかについてのみ検査を行うことができる。./src ディレクトリ以下にあるソースファイルに対して random 関数と gets 関数が使用されているかを調べたい場合には、次のようにコマンドを入力する。

```
% rats -a random -a gets "検査を行いたいディレクトリ名、またはファイル名のリスト"
```

3.3.2. 明示的な検査の省略

前述のように、ユーザが明示的に指定した関数名については、-a オプションで検査が行える。しかし、検査対象から省略したい関数名を指定することを、オプションで行うことはできない。ユーザが特定の関数に関して検査を省略したい場合には、ソースコード中に以下のようなコメントを挿入する必要がある。

```
strcpy(buf, dst); /*rats: ignore*/
```

上記の記述は以下のような C++スタイルコメント文で記述可能である。

```
strcpy(buf, dst); //rats: ignore
```

上記の方法による検査の省略は、コメントが書かれた行に対してのみ有効である。この為、以下のような記述においては、最初の strcpy の検査の省略は行われるが、2 番目の strcpy の省略は行われない。これは、C++スタイルのコメント文でも同様である。

```
strcpy(buf1, dat1); /*rats:ignore*/
strcpy(buf2, dat2);
```

以下のような形式でコメントを書いた場合には、コメントが書かれた次の行のみ、検査を省略することが可能である。これは、C++スタイルのコメント文でも同様である。

```
/* rats:ignore */
strcpy(buf, dst);
```

検査省略の為にコメントを挿入する場合は、必ず rats:ignore という記述でなければならない。コメントの記述は、拡張可能な仕様になっているが、現在の所は ignore コマンドしか受け付けない。これ以外の文字列が記入されると、検査の省略は行われない。

また、rats:と記述された個所は、its4:という記述で代用することが可能である。この記述は ITS4 というソースコードの検査ツール用に用意されたものである。このような記述を許すのは、RATS、ITS4 のツールによる二重検査が容易に行えるよう考慮された結果である。現在、RATS の開発では Flawfinder というソースコード検査ツールと相互に連携・メンテナンスを行う試みが検討されている。Flawfinder でも、RATS、ITS4 同様にコメント文でコマンドを入力することが可能であり、将来的には RATS と Flawfinder によるソースコードの二重検査が可能となることが期待されている。

3.3.3.　検査言語の指定

RATS では、ファイル名の拡張子から使用されている言語を自動的に判定している。ユーザが明示的に検査対象の言語を指定したい場合には、-l または --language オプションで言語名を指定する。以下は、.c 拡張子のファイル foo.c を perl で記述されたものとして検査を行う例である。

```
% rats -l perl　foo.c
```

3.3.4.　ヘルプの表示

以下のように、RATS のヘルプを呼び出すことが可能である。

```
% rats -h
```

-h オプションの代わりに、--help オプションを利用することも可能である。

3.3.5.　ワーニングレベルの指定

脆弱性データベースに登録された関数には、3 つの危険度が定められている。脆弱性データベースでは、それぞれ High、Medium、Low と記述されている。これらに対応して、レベル 1、レベル 2、レベル 3 と呼ばれる検査レベルが用意されている。レベル 1 では、High の危険度の関数に対してのみ検査が行われる。レベル 2 では、High と Medium の危険度の関数に対して検査が行われる。レベル 3 では、すべての危険度の関数に対して検査が行われる。検査レベルを指定する場合には、以下のように-w オプションを使用する。

```
% rats -w 2　"検査を行いたいディレクトリ名、またはファイル名のリスト"
```

検査レベルを特に指定しない場合には、レベル 2 で検査が行われる。

3.3.6.　脆弱性データベースの指定

ユーザは、明示的に使用する脆弱性データベースを選択することができる。インストールされた脆弱性データベース以外に、使用する脆弱性データベースを追加するに為には、-d、-db、--database のいずれかのオプションを使用する。複数の脆弱性データベースを追加する場合は、このオプションを繰り返し使用する。以下に、/usr/share ディレクトリ下にある original_1.db と original_2.db を使用する時の例を示す。

```
% rats -d /usr/share/orginal_1.db -d /usr/share/original_2.db　¥
"検査を行いたいディレクトリ名、またはファイル名のリスト"
```

インストールされた脆弱性データベースの使用を省略するには、-x オプションを使用する。以下に、標準の脆弱性データベースを使用せずに、上記の例で使用した脆弱性データベースを用いて検査を行う例を示す。

```
% rats -x -d /usr/share/orginal_1.db -d /usr/share/original_2.db　¥
"検査を行いたいディレクトリ名、またはファイル名のリスト"
```

3.4.　脆弱性データベースの作成

ユーザは、脆弱性データベースを独自に作成することができる。RATS における脆弱性データベースは XML で記述されている。このため、ユーザは比較的容易に脆弱性データベースの改変及び作成が行える。

脆弱性データベースの記述にあたっては、以下の 2 点が記述されていることが推奨される。

- XML 宣言
- DOCTYPE 宣言

これらの記述は以下のようにして行う。

```
<?xml version="1.0"?>
<!DOCTYPE RATS []>
```

上記の記述の他に、以下のようなルートエレメントの記述が行われていることが必須となっている。各脆弱性の記述は、ルートエレメント内にエレメントとして行う。以下の記述では、C 言語が対象の脆弱性データベースとして宣言している。この為、RATS が C 言語以外の言語と判別した場合には、この脆弱性データベースが検査時に使われることはない。

```
<VulnDB lang="c">
```

脆弱性データベース中の各脆弱性の記述について、以下の 2 点が記述されていることが必須である。

宣言の種類	使用するタグの例
各脆弱性の区切りの宣言	<Vulnerability> </Vulnerability>
関数名の記述	<Name> </Name>

また、上記に加えて以下の項目の記述も可能である。

宣言の種類	使用するタグの例
危険度	<Severity> </Severity>
脆弱性に付随する情報	<Info> </Info>
脆弱性の概要	<Description> </Description>
脆弱性の可能性のある引数の個所	<Arg> </Arg>, <FormatArg> </FormatArg>, <SrcBufArg> </SrcBufArg>
参照となる URL	<URL> </URL>
脆弱性の種類名	<RaceCheck> </RaceCheck>, <RaceUse> </RaceUse>, <InputProblem> </InputProblem>, <FSProblem> </FSProblem>, <BOProblem> </BOProblem>

通常、以下のような形式の記述を行うだけで十分な検査を行うことが可能である。

```
<Vulnerability>
<Name>検出行したい関数名をここに記述する</Name>
<info>
<Description>脆弱性の概要・対策の概要をここに記述する</Description>
<Severity>危険度（High、Low、Medium）を個々に記述する</Severity>
</info>
</Vulnerability>
```

尚、脆弱性データベースの内容を日本語で記述することはできない。日本語で記述を行った場合、RATS はその脆弱性データベースを無視する。

4. ケーススタディ：脆弱性の可能性の高い関数の検出

本ケーススタディでは、脆弱性データベースに登録されている fopen の検出が行えるかを確認する。確認においては、以下のようなソースコードを利用する。

```
#include <stdio.h>
int main(int argc, char *argv[]){
   char *fmt = "%d:%s¥n";
   int i=1;
   FILE *fp;

   if(argc = = 3){
      fp = fopen(argv[1], "w");
      fprintf(fp, fmt, i, argv[2]);
      fclose(fp);
   }
   return 0;
}
```

検査のために、上記のソースコードを test.c というファイル名で保存し、検査は以下のようにして行う。

```
% rats test.c
```

検査結果は以下のようである。

```
Entries in perl database: 33
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing test.c
test.c:9: High: fprintf
Check to be sure that the non-constant format string passed as argument 2 to
this function call does not come from an untrusted source that could have added
formatting characters that the code is not prepared to handle.

Total lines analyzed: 14
Total time 0.000293 seconds
47781 lines per second
```

fprintf が危険度（High）として警告される。fopen 関数に関する警告は特に出力されなかった。RATS をデフォルトの状態で実行すると、危険度（Low）レベルの関数の検査は行われない。従って、以下のようにオプション-w で警告レベルを設定して検査を行う。

```
% rats -w3 test.c
```

検査結果は以下のようである。

```
Entries in perl database: 33
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing test.c
test.c:9: High: fprintf
Check to be sure that the non-constant format string passed as argument 2 to
this function call does not come from an untrusted source that could have added
formatting characters that the code is not prepared to handle.

test.c:8: Low: fopen
A potential race condition vulnerability exists here.  Normally a call to this
function is vulnerable only when a match check precedes it.  No check was
detected, however one could still exist that could not be detected.

Total lines analyzed: 14
Total time 0.000285 seconds
49122 lines per second
```

警告レベルを変更した結果、fopen の使用を危険度（Low）、fprintf の使用を危険度（High）として検出した。現在の出力結果では、ヘッダー情報・フッター情報が付随している。検査結果のみを出力させるために以下のようにする。

```
% rats --resultsonly -w3 test.c
```

出力結果は以下のようである。

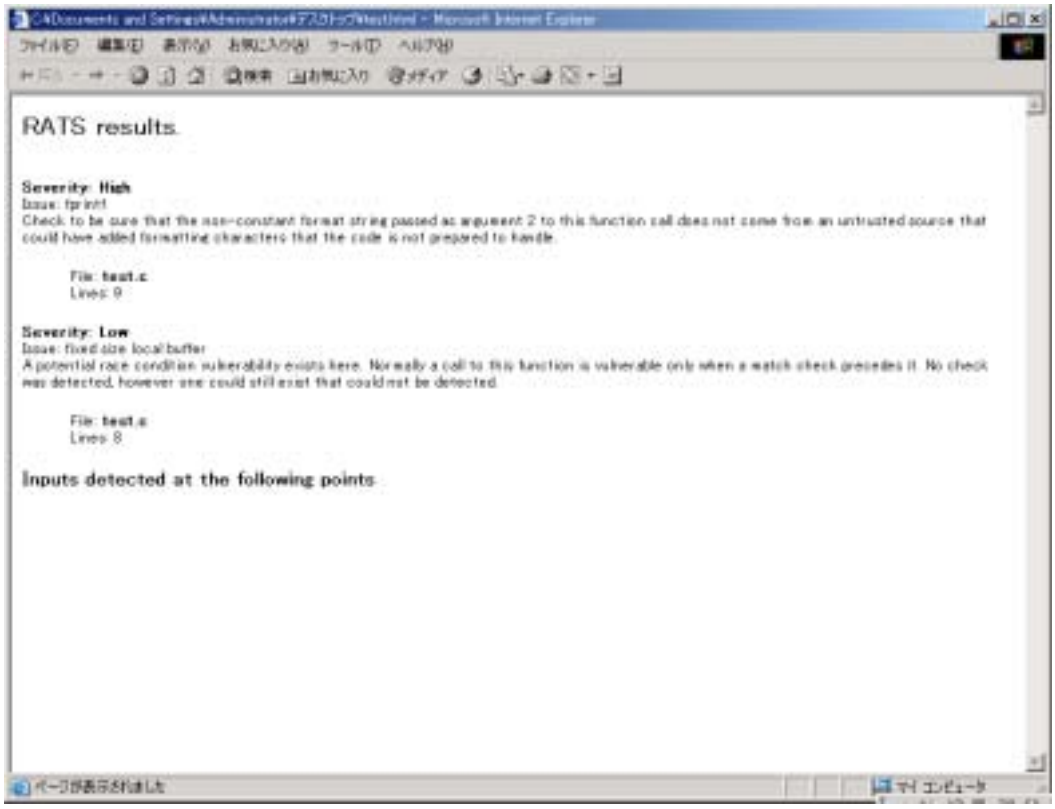
```
test.c:9: High: fprintf
Check to be sure that the non-constant format string passed as argument 2 to
this function call does not come from an untrusted source that could have added
formatting characters that the code is not prepared to handle.

test.c:8: Low: fopen
A potential race condition vulnerability exists here.  Normally a call to this
function is vulnerable only when a match check precedes it.  No check was
detected, however one could still exist that could not be detected.
```

6 行目からの出力結果に、fopen 関数の使用に関して警告が出力されている。この結果を HTML 形式で出力する。なお、出力した HTML ファイルのファイル名を test.html とする。

```
% rats --html --resultsonly -w3 test.c > test.html
```

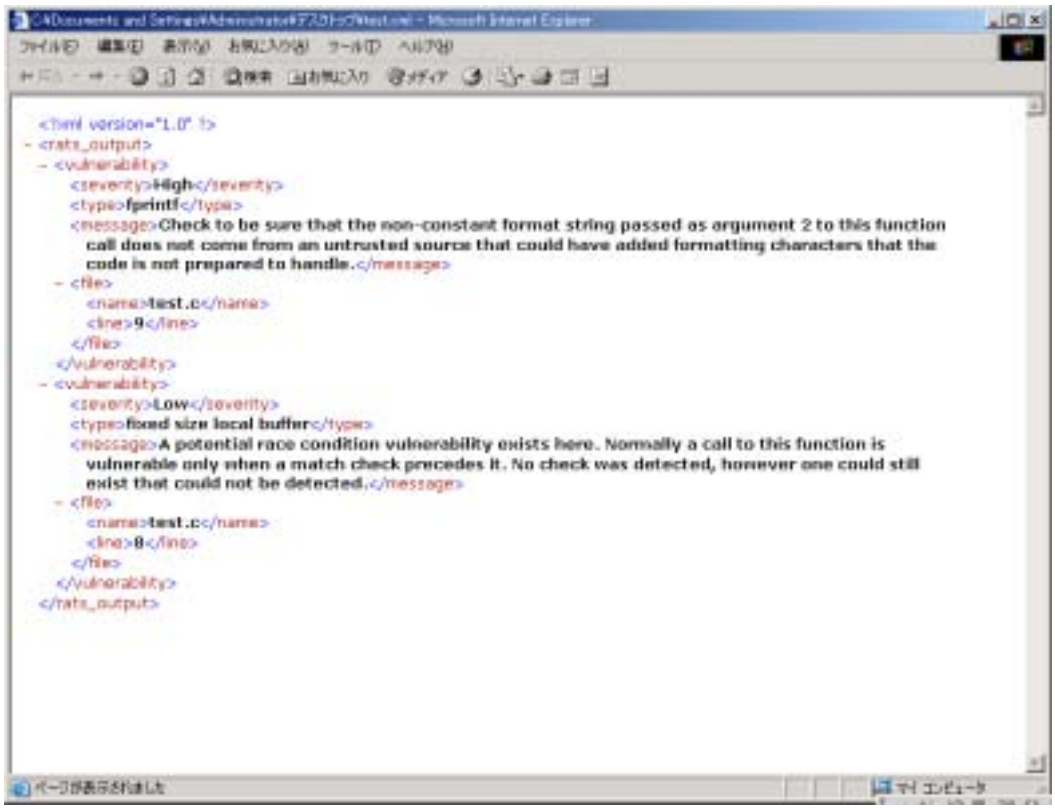
出力結果を Web ブラウザ（Microsoft Internet Expolorer 6.0）で表示すると以下のようである。



同様に XML 形式で出力を行うには、以下のようにする。

```
% rats --xml --resultsonly -w3 test.c
```

出力結果を上記と同様に Web ブラウザ[2]で表示すると以下のようである。



[2] XML 形式の表示を Web ブラウザで表示するには、Web ブラウザが XML 対応していなければならない。

5. 付録

5.1. RATS のコマンドオプションの一覧

オプション	説明
-a　<*fun*>	ユーザが明示的に指定した関数 *fun* を検査する時に用いる。 例：read_line_from_socket と read_line_from_user という関数を検査したい場合は、次のようにコマンドを入力する。 > rats –a read_line_from_socket –a ¥ read_line_from_user ./*.c
-d -db --database	脆弱性データベースのファイルを明示的に指定する時に使用する。複数のファイルを指定したい場合は、-d オプションを繰り返し使用する。
-h --help	コマンドの使用方法及びオプションの概要を表示する。
-l --language	指定した言語に関する脆弱性データベースを用いて解析を行う。 例：C で書かれたソースコードを perl の脆弱性データベース用いて検査する。 > rats –l　perl ./*.c
-w --warinig	ワーニングレベルの設定が行える。有効なレベルは 1，2，3 である。脆弱性データベースでは、各脆弱性毎に重要度に応じたラベル（High, Medium, Low）が振られている。レベル 1 で検査を行うのは High についてであり、レベル 2 では High と Medium について、レベル 3 で全ての脆弱性について検査をする。尚、-w オプションでワーニングレベルを指定しない時に使われるのは、レベル 2 である。
-x	デフォルトの脆弱性データベースの参照を行わずに検査を行う。
-R --no-recursion	サブディレクトリに対しての検査は行わない。
--xml	出力を XML 形式で行う。
--html	出力を HTML 形式で行う。
--noheader	最初のヘッダー情報を出力しない。
--nofooter	最後のフッター情報を出力しない。
--quiet	解析中のファイルのステータス情報を出力しない。
--resultsonly	ヘッダー、フッター及びステータス情報を出力しない。
--columns	問題個所が行の先頭から何文字目で問題が発生したかを示す。
--context	問題個所の行を出力する。

参考文献

[1] Secure Software,
http://www.securesoftware.com/download_form_rats.htm