第７章　セキュア Unix/Linux プログラミング

[7-7.] Unix パス名の安全対策

相対パス記法を悪用したディレクトリトラバーサル攻撃から保護するために，パス名チェックは欠かせない。しかし相対パス記法を仕様上認めるアプリケーションもある。パス名を正規化して，意図するディレクトリ／ファイルを指しているかどうかをチェックしよう。

● ● ●

ディレクトリトラバーサル攻撃

リスト１の Perl で書かれたサンプルプログラムを見ていただきたい。ファイルを操作するアプリケーションによく見られるコーディング例である。このアプリケーションではデータファイルを「/var/data/」ディレクトリ下に配置し，そのディレクトリ下のファイルへのみアクセスすることを前提としている。1行目はこれをそのまま反映させたコーディングで，データディレクトリ「/var/data/」のパスに，参照しようとしているファイル名$fileを連結して，パス名$filepathを構築している。その後，2行目のsysopen文で$filepathにしたがいデータファイルをオープンしている。

ちなみに，2行目のファイルオープン処理ではopen文ではなくsysopen文を使用している。これはPerlプログラムを標的にした「ダイレクト OS コマンドインジェクション攻撃」（注１）の危険を回避するためである。

（**注１**・「ダイレクト OS コマンドインジェクション攻撃」については Perl に関する関連記事『4-1. ファイルオープン時のパスにご用心』を参照していただきたい。）

リスト１　データディレクトリ /var/data/ 配下のファイルへアクセス

```
1  $filepath = "/var/data/".$file;
2  sysopen(DATA, $filepath, O_RDONLY);
```

実は**リスト１**のようなコードのままだと，「ディレクトリトラバーサル攻撃」を受けてしまう。**図１**はその図解である。ここでは，$fileに攻撃文字列が与えられたときの$filepathがどのように解釈されるかを示している。

図１の先頭で述べているように，もし$fileに「../../etc/passwd」という文字列が与えられた場合，まず**ステップ１**

図１　$filepath の解釈：ディレクトリトラバーサル攻撃の例

```
もし $file が "../../etc/passwd" だったなら $filepath の解釈は・・・

ステップ１  "/var/data/"."../../etc/passwd"       ← "/var/data/".$file
                         ↓
ステップ２  "/var/data/../../etc/passwd"          ← $filepath はこの文字列となる
                         ‖  等価
ステップ３       "/etc/passwd"                    ← sysopen 文の解釈はこうなる
```


Copyright © 2002 IPA, All Rights Reserved.

で「/var/data/」と $fileが連結され，$filepathは**ステップ２**のようなパス名文字列となる。**ステップ２**のパス名は相対パス記法「../」を含んでおり，これは**ステップ３**の「/etc/passwd」と等価である。したがって $filepath が sysopen 文に渡されるときには，「/etc/passwd」をオープンするという意味になってしまう。

このように相対パス記法「../」を使用することで，システム内のディレクトリ間を自由に横断（トラバース）し，任意のファイルへアクセスできてしまう。これが「ディレクトリトラバーサル」という攻撃名称の由来である（注２）。

（**注２**・「ディレクトリトラバーサル攻撃」に関する各種の話題については，参考文献『Directory Traversal』も参照していただきたい。）

Unix のパス名の特徴

UnixおよびLinuxのファイルシステムの特徴は，すべてのファイルとディレクトリがルートディレクトリ「/」を根とする単一の木構造を構成していることである。このことは，ファイルシステム上のあらゆるリソースをすっきりした形で取り扱うことができる優れたデザインである反面，親ディレクトリを辿る相対パス記法「../」を悪用した「ディレクトリトラバーサル攻撃」の格好の餌食になることも意味している。

パス名を正規化してからチェックする

ディレクトリトラバーサル攻撃は，パス名構築時に相対パス記法「../」が混入できることにより成立する。当然，防御の手段として「../」を禁止してしまうことが考えられる。しかしアプリケーションの用途によっては相対パス記法を許可せざるをえないものもあるだろう。その場合，入力データに含まれる「../」が正当なものか不正なものかを単純には判別できない。相対パス記法を許可するアプリケーションでは，構築したパス名を正規化してからチェックするとよい。

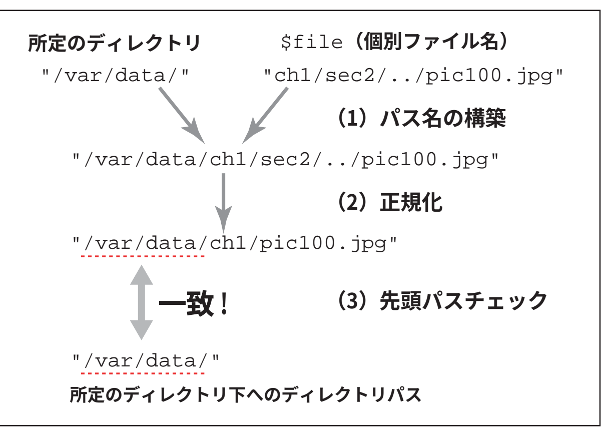
パス名の正規化とは，次の要件を満たすようにパス名を変換することである。ただし変換後のパス名も同じファイルを指すよう，パス名の意味が変わらないように変換する。

- 絶対パス名である（/ から始まる）
- パス名中に // を含まない（/home//foo/data は /home/foo/data と等価）
- パス名中に /./ を含まない（/home/./foo は /home/foo と等価）
- パス名中に /../ を含まない（/home/foo/../bar/ は /home/bar/ と等価）

パス名を正規化することにより，相対パス記法「../」を含まないパス名へと変換できる。正規化したパス名の先頭部分が「/var/data/」であるかどうかをチェックすることで，パス名が「/var/data/」ディレクトリ配下のファイルを指すかどうか判断できる。正規化したパス名は相対パス記法を含まないため，パス名の先頭が「/var/data/」で始まるならば，「/var/data/」の外のファイルへは決してアクセスできないからだ。

図２にパス名を正規化してからチェックを行う例を示す。「(1) パス名の構築」では単純に「/var/data/」とファイル名$fileを連結して，「../」を含んだパス名を生成している。「(2) 正規化」を施すことで相対パス記法「../」「./」「//」を一切含まないパス名に変換している。「(3) 先頭パスチェック」により，正規化したパス名の先頭部分とデータディレクトリのパス「/var/data/」を比較して，一致していることを確認している。正規化された

図２　パス名を正規化してからチェック

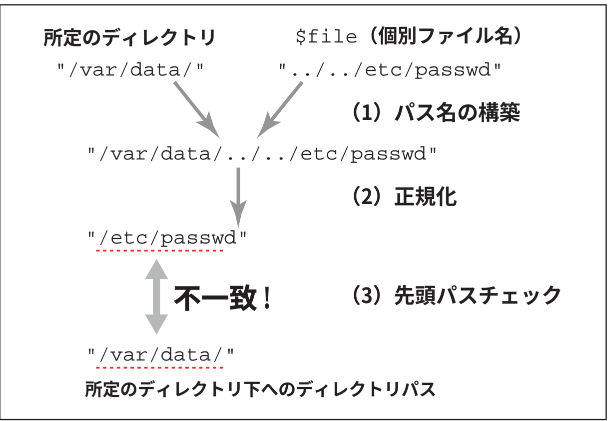


Copyright © 2002 IPA, All Rights Reserved.

パス名は「/var/data/」で始まっているので，このディレクトリ下のファイルであることが確認できた。そのまま sysopen 文に渡しても安全である。

図３は同様の手法により，ディレクトリトラバーサル攻撃を検知する例である。「(1) パス名の構築」では攻撃者が送り込んだファイル名$fileにより，相対パス記法「../」を使ったパス名が生成される。しかし「(2) 正規化」によって相対パス記法を排除することで，「/etc/passwd」という直接的なパス名へと変換される。「(3) 先頭パスチェック」により，正規化したパス名の先頭部分とデータディレクトリのパス「/var/data/」と比較すると，不一致であることが分かる。

図３　ディレクトリトラバーサル攻撃を検知



このように，所定のディレクトリ下にないファイルへのパス名が生成されていることが検知できるので，意図しないファイルへのアクセスを未然に検知・防御できる（注３）。不正なパス名を検知した場合，プログラムを中断するなどのエラー処理を行うことになる。

（**注３**・これでパス名については無害化された。しかしこの記事の後半でも触れるが，より安全を期すためにはローカルなユーザによって不正なシンボリックリンクが設けられていないことを確かめる必要がある。）

チェック処理を部品化して毎回利用する

リスト２は**リスト１**を改良したもので，パス名を正規化してチェックする手順を追加したものである。**リスト１**の処理に該当する部分は1～4行目である。6～31行目はパス名の正規化関数canonicalize，33～41行目は先頭パスチェック関数 checkpath である。

1行目は**リスト１**の1行目と同じである。2行目は「./」や「../」が含まれる可能性のある $filepath を正規化関数canonicalizeによって正規化し，結果を$canopathに保存している。3行目は正規化されたパス名$canopathの先頭部分を「/var/data/」と比較して一致するかどうかを調べている。一致しなかった場合，die 文によりプログラムをその場で終了している。2～3行目の検査を通過した場合に限り，正規化したパス名は「/var/data/」ディレクトリ下のみを指すことが保証される。よって4行目のsysopen 文で「/var/data/」ディレクトリの外のファイルへアクセスしてしまうことはない。

正規化関数 canonicalize

6～31行目のパス名正規化関数 canonicalize について簡単に解説する。7行目で関数の引数に与えられた正規化すべきパス名をローカル変数 $path に受け取っている。10～14行目は $path が相対パス名であった場合に，プロセスのカレントディレクトリを元に絶対パス名に変換している。17～26行目で正規化処理を行っている。17行目の@components配列は正規化したパス名の中間データを収める変数である。最初は空にしておく。18行目の foreach ループでは，パス名を '/' で区切ったパス名の構成要素ごとに以下の検査・処理を行う。

- 「/」の連続は無視（19 行目）
- 「.」は無視（20 行目）
- 「..」ならば @components 配列から構成要素を１つ削除（21 ～ 24 行目）
- それ以外なら @components 配列へ構成要素を追加（25 行目）

foreachループが終了すると，@components配列に正規化したパス名の構成要素が含まれている。28行目では @components 配列に含まれる構成要素を '/' で連結してパス名文字列を生成している。

Copyright © 2002 IPA, All Rights Reserved.

リスト２　改良されたプログラム

```
 1  $filepath = "/var/data/".$file;         # パス名を構築
 2  $canopath = &canonicalize($filepath);   # パス名を正規化
 3  &checkpath("/var/data/", $canopath) or die("'$canopath'へのアクセスは禁止。");
 4  sysopen(DATA, $canopath, O_RDONLY);     # $canopathでオープン
 5
 6  sub canonicalize {
 7      my $path = shift;
 8
 9      # 相対パス名→絶対パス名
10      if($path!~m|^/|) {
11          my $cwd = `/bin/pwd`;
12          chop($cwd);
13          $path = "$cwd/$path";
14      }
15
16      # 正規化パス名の作成
17      my @components = ();                # 正規化パス名中間データを初期化
18      foreach $component (split('/', $path)) {
19          next if($component eq "");      # //は無視
20          next if($component eq ".");     # /./は無視
21          if($component eq "..") {        # /../なら
22              pop(@components);           # 1つ前の構成要素も無視
23              next;
24          }
25          push(@components, $component);  # 構成要素を追加
26      }
27
28      $path = '/'.join('/', @components); # パス名文字列を生成
29
30      return $path;
31  }
32
33  sub checkpath {
34      my $leading = shift;    # 第1引数はお手本先頭パス
35      my $path = shift;       # 第2引数にパス名
36
37      # $pathが$leadingで始るときのみ1(true)を返す
38      return 1 if(index($path, $leading)==0);
39
40      return 0;   # $pathは$leadingで始っていない
41  }
```

先頭パスチェック関数 checkpath

33～41行目の先頭パスチェック関数について簡単に解説する。第1引数はプログラマが範囲の限定を意図しているディレクトリ「/var/data/」などのパス名で，34行目のローカル変数$leadingで受け取っている。第2引数はチェック対象の正規化されたパス名で，35行目のローカル変数$pathで受け取っている。$pathの先頭が$leadingと一致するかどうかの検査は，38行目のindex文で行っている。ここでのindex文は，$path文字列中で$leading文字列が含まれる個所をインデックスとして返すので，先頭を表すインデックス0（ゼロ）と比較している。インデックスが0の場合，$pathは$leadingで始っているので，true値の1を返している。そうでない場合はfalse値の0を返している。

以上はPerlの例であるが，こうしたcanonicalizeやcheckpathのようなソフトウェア部品を対象アプリケーションの開発言語に合わせて用意し，パス名のチェックを徹底することをお奨めする。

Copyright © 2002 IPA, All Rights Reserved.

より厳密なパス名の検査

一般的なアプリケーションにおけるパス名の安全対策としては前節で紹介した手法で十分な場合も多い。しかし非常に機密性・重要性の高い情報を扱うアプリケーションの場合には，さらに以下のような項目も検査すべきである。

- ・パス名の各階層のディレクトリが実際に存在すること
- ・パス名の各階層のディレクトリのオーナやグループ，ファイルモードが適切であること
- ・パス名中にシンボリックリンクが含まれていないこと

シンボリックリンクは関連記事『7-1. シンボリックリンクの悪用』で紹介しているように特に悪用されやすい。シンボリックリンクがパス名中に含まれていると，パス名先頭部分が期待するディレクトリパスと一致しても，期待するディレクトリの外へリンクされている可能性があるなどだ。パス名中のシンボリックリンクを展開して，シンボリックリンクを含まないパス名を形成した後，パス名先頭部分が期待するディレクトリパスと一致するかどうかを確認すべきである。

ただし上記のような厳密な検査は，アプリケーションの運用にとって不便な場合もある。シンボリックリンクを使って容量に余裕のあるディスクパーティションへ大量データ用のディレクトリを再配置するといった運用ができなくなる。アプリケーションの性質や運用形態に合わせて，検査項目は調整すべきだ。

まとめ

ファイルを扱うアプリケーションはパス名の構築に注意を払わなければならない。そのためにはパス名を正規化し，先頭部分が期待通りのディレクトリ下を指しているかどうか確認すべきである。

関連記事

『4-1. ファイルオープン時のパスにご用心』

『7-1. シンボリックリンクの悪用』

参考文献

『Directory Traversal』（英文），Open Web Application Security Project
http://www.owasp.org/projects/asac/iv-directorytraversal.shtml

Copyright © 2002 IPA, All Rights Reserved.