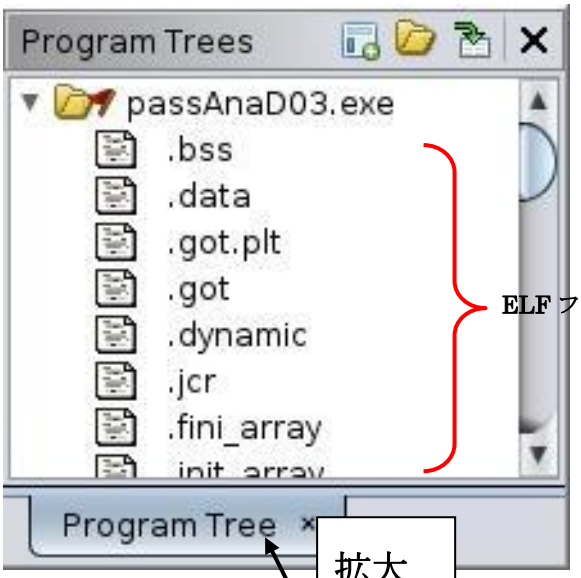
第３１話（マルウェア解析Ⅲ）

パスワード入力プログラムの解析例

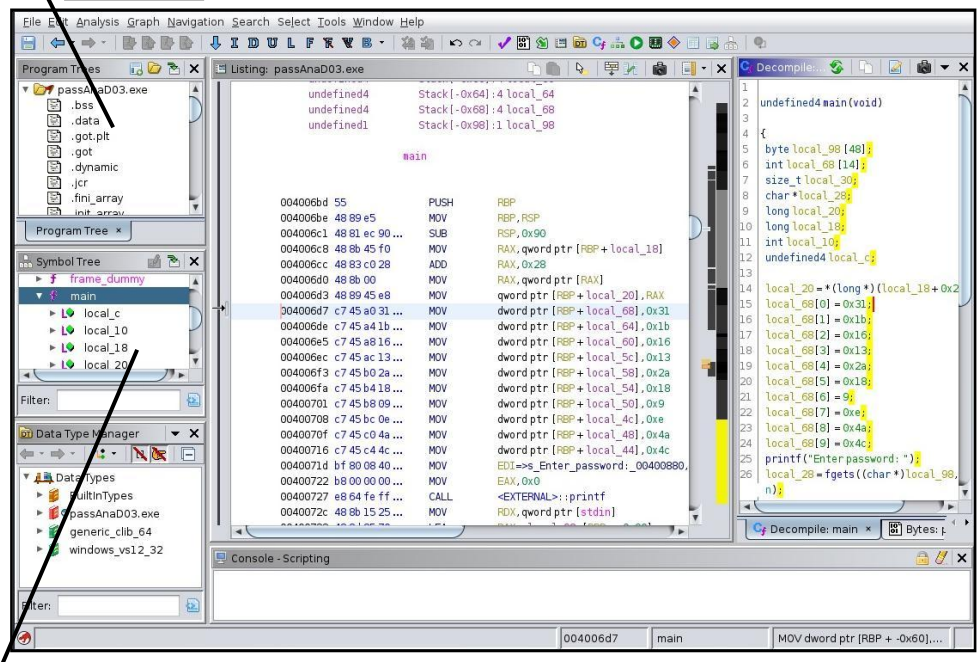
タヌキ！最初は、実行ファイルであるパスワード入力プログラム（passAnaD03）を解析して既に記憶されているパスワードを割り出す、ということをやってみる。
このプログラム（passAnaD03）の場合、正しいパスワードがエンコード（１６進数への変換）されてスタック領域に積まれている。その状態は、Ghidra でデコンパイルされたプログラム（C 言語）でも見ることができる。
新規に入力されたパスワード（仮に Raccoon とする）と正しいパスワード（仮に yxxxxxxxxy とする）は両方ともエンコードされ、エンコードされたビット列を相互に XOR 演算をし、全てが０になれば入力されたパスワードが正しいと認証し、ログインできると言う訳だ。
今回は、この正しいパスワード（仮に yxxxxxxxxy とする）を割り出し、ASCII 文字として表示しようという試みだ。
ただ、パスワードは１６進にエンコードされるという単純なものでは無く、実際にはハッシュ関数などでメッセージダイジェスト形式に変換されているので、簡単に復元することができない、ということに注意してもらいたい。ここは、あくまでも仕組みの勉強だ。

キツネ、これから実習に使う「passAnaD03」実行ファイルは、PE フォーマットと ELF フォーマットのどちらだ？

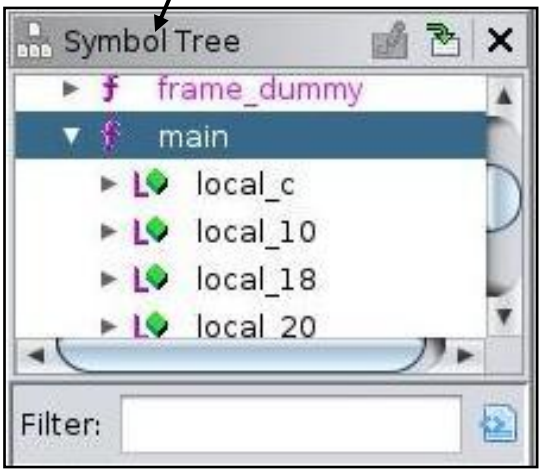
「passAnaD03」はソースプログラムが Linux 上の C 言語で開発されたものなので、ELF フォーマットだ。以下に、「passAnaD03」からパスワードを割り出す方法を図示する。



キツネ、これが「passAnaD03」実行ファイルを Ghidra にインポートした状態か。左の Program Trees を見ると、ELF フォーマットになっているな。



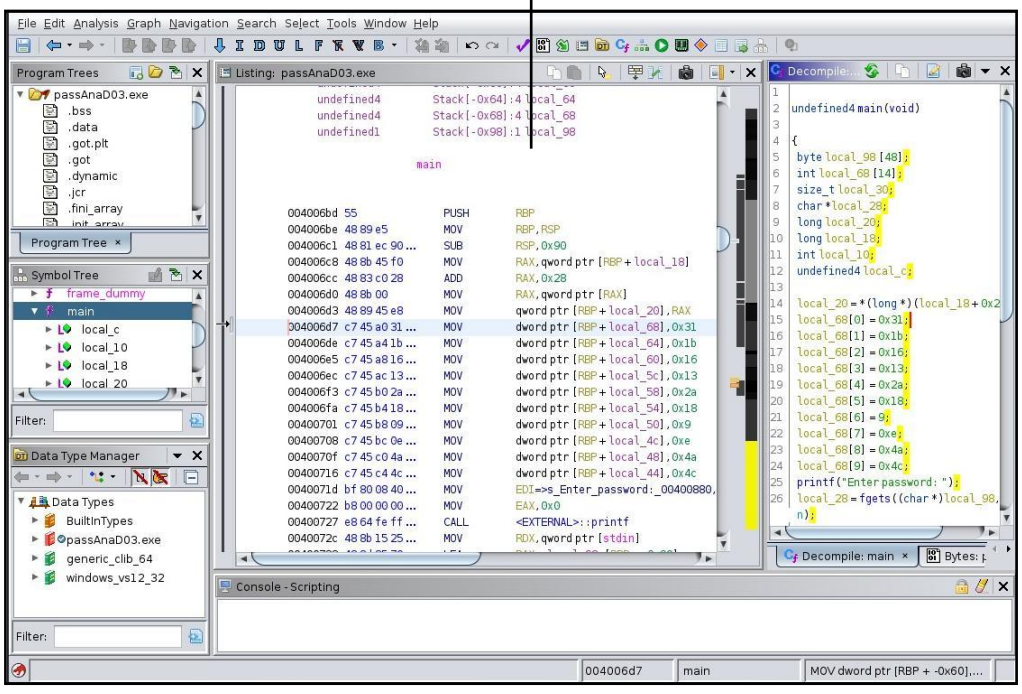
拡大

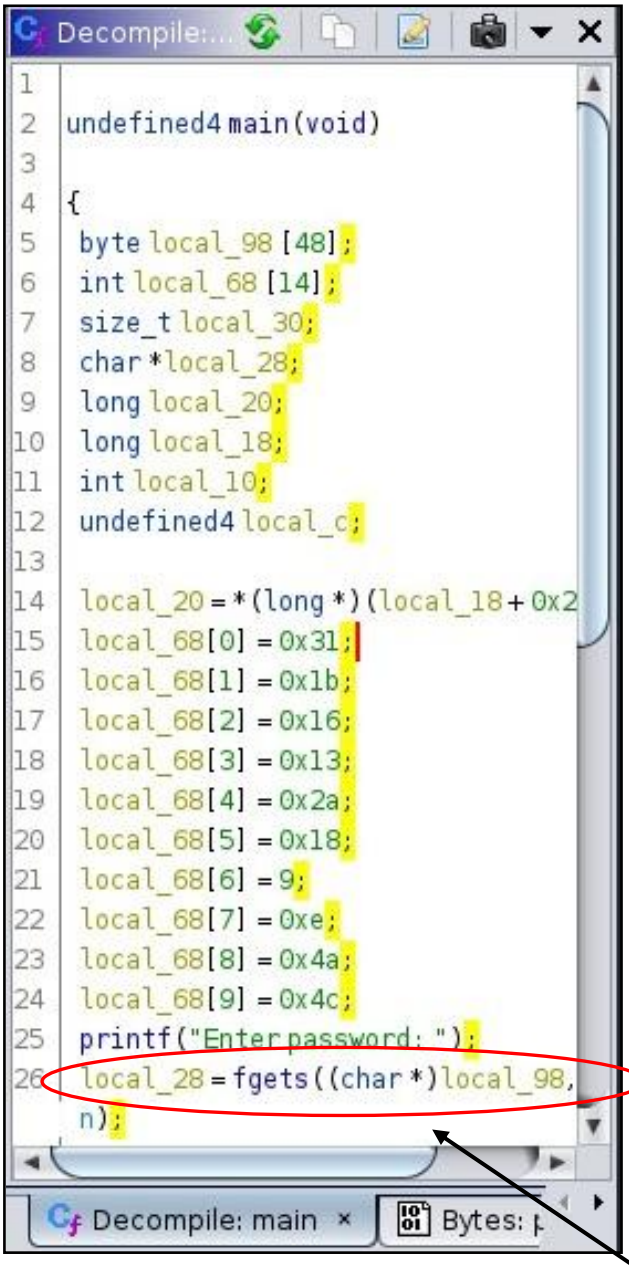


ELF バイナリ（実行ファイルのこと）は、main（）で始まるのでここをクリックすれば、逆アセンブラスクリプトや逆コンパイラのスクリプトが表示されるのか。
Windows の PE バイナリの場合も main()で始まるが、main という表示でないので、main()を探すのが大変だよな。

これが逆アセンブラされた main 関数の部分か。確かにスタックに積まれたパスワードの最初の仮想アドレス（004006d7）を読み取ることができる。16 進数の 31 がエンコードされたパスワードの最初の文字だな。

拡大

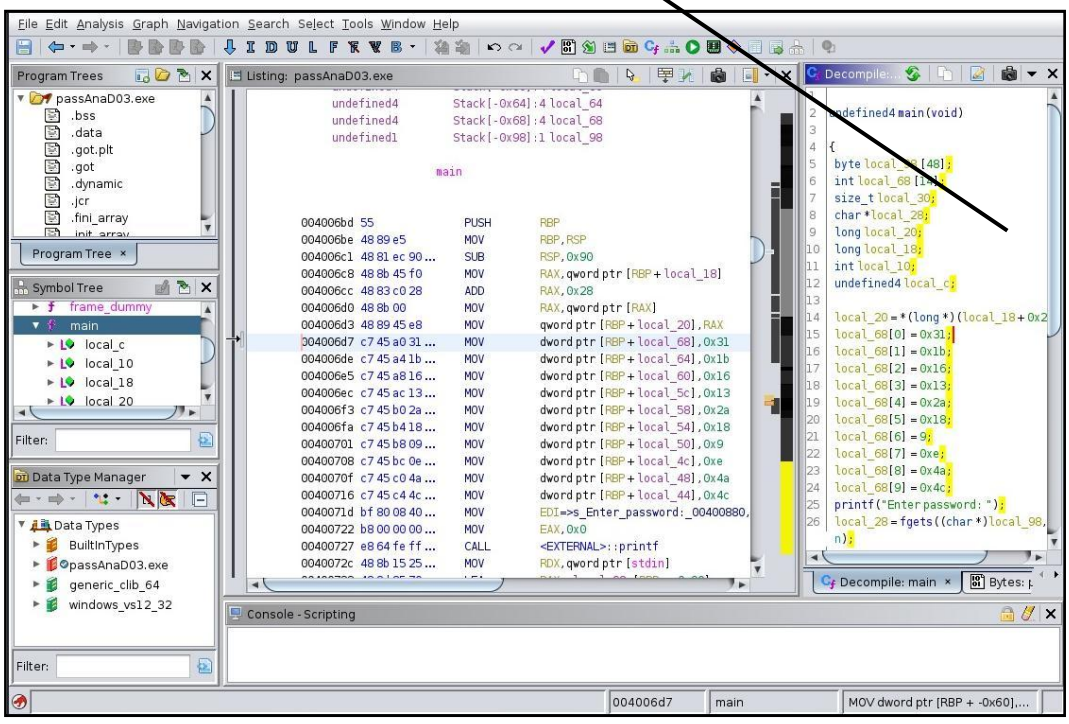




バイナリファイルからリバースエンジニアリングするのは大変だが、このように逆コンパイルしてくれれば、一目瞭然ではないか。スタックに積まれているパスワードも見えるし、「passAnaD03」を実行した時に入力されるパスワードを取り込む fgets()関数も読み取ることができるし、その後の XOR 処理も読み取れる。

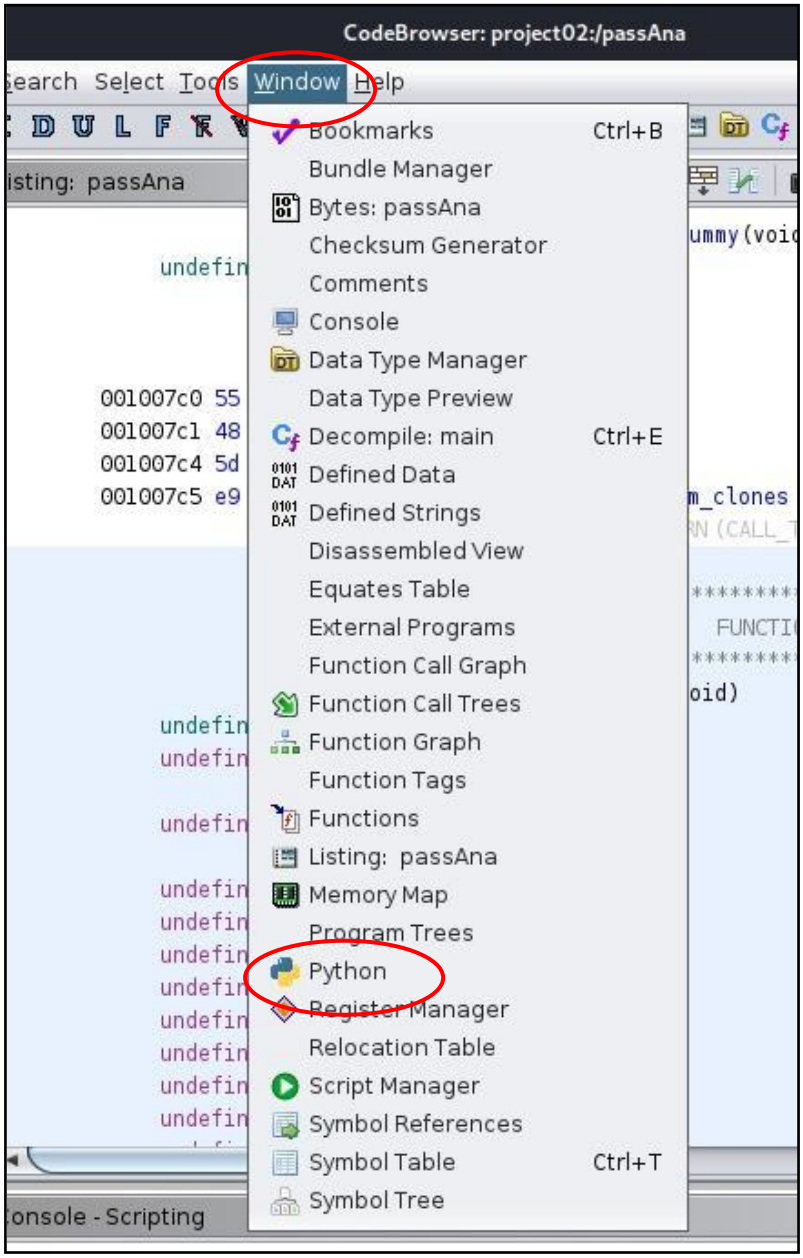
「passAnaD03」のコンパイル前のソースと違うのは変数名ぐらいなものかな。

拡大



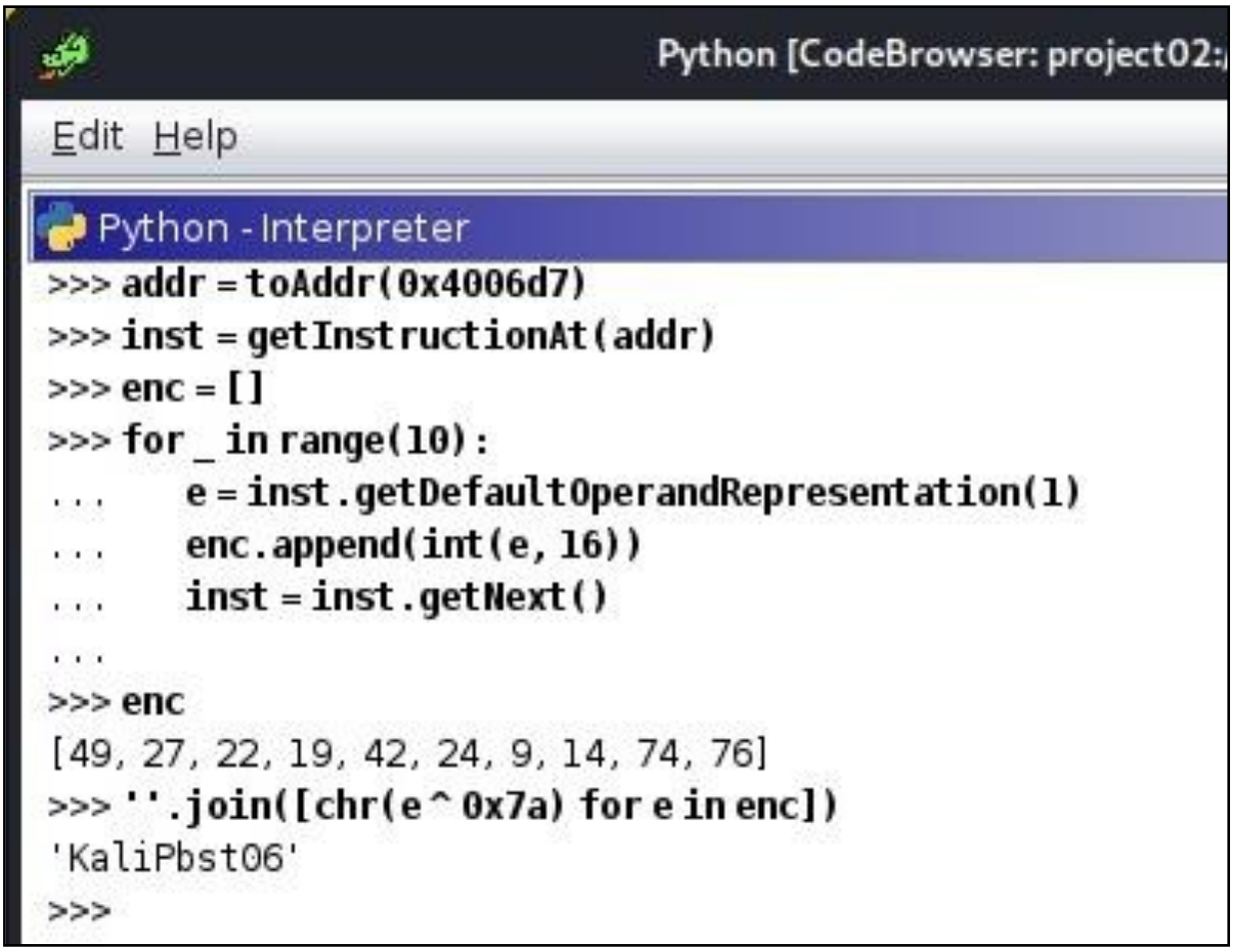
キツネ！バイナリファイルからディス（逆）アセンブルやデ（逆）コンパイルし、内容を読み取ることができることは理解した。読み取った後、どうするのだ？

タヌキの言うように、読み取った後が大切だよな。
手順は、パスワードが積まれているスタック領域の最初の仮想アドレスを設定し、上から順に１０個のエンコードされた１６進データを取り出して、それを ASCII 文字に変換していくのだ。
この作業をするために、以下の図で示すように Ghidra に標準で付いている Python のコマンドプロンプトを使うのだ。



なるほどな、Python も使うのか。Python はインデント（字下げ）を守らないとエラーになるからな。

Python のコマンドプロンプトが開いたら、入力する命令やデータは次図のようになる。



なるほどな、toAddr()関数でスタックの最初の仮想アドレスを addr に保存し、enc 配列を定義し、getInstructionAt()関数で、仮想アドレス（4006d7）の部分のアセンブルされた 1 行を取り出している。繰り返し処理（for）でパスワードの文字数分、10 回繰り返している。getDefaultOperandRepresentation(1)関数で inst からデータ（オペランドともいう）部分（0 x 31）を取り出している。取り出したものを、append（）関数で 16 進数と断って enc 配列に順に保存している。getNext()は言うまでも無く、次のアセンブルされた行に移動することだ。For 以下を入力する時には、インデントに注意しなければエラーになるよね。

繰り返し処理から抜ける場合には、単に「Enter」キーを押すだけで OK だ。enc と入力するだけで、配列に保存されたデータを確認することができる。次に enc 配列の値を 1 個づつ変数 e に代入し、chr（e ^ 0x7a）で、XOR 処理をした文字コードを ASCII 文字に変換している。さらに、join()関数で 10 個の文字を結合し、一連の文字列にしている。それが実際のパスワードである「KaliPbst06」だ。キツネ、こんな解釈でいいかな？

タヌキの解釈で良いよ。
正しいかどうかは、下図のように端末を開いて実行ファイル「passAnaD03」が存在するディレクトリに移動し、「passAnaD03」を実行してみることだ。

```
┌──(kali㉿kali)-[~]
└─$ ./passAnaD03.exe
Enter password: abcdefghij
Wrong ...
```

キツネ！これは、パスワードを割り出していないので、適当に「abcdefghij」と１０個入力し、間違っているよ（Wrong）と拒否された状態だな。

```
┌──(kali㉿kali)-[~]
└─$ ./passAnaD03.exe
Enter password: KaliPbst06
Correct !!
```

次は、割り出したパスワード「KaliPbst06」を入力し、正しい（Correct）と受け入れられた状態だな。
すごいなキツネ！

タヌキ、もう少し丁寧に説明しておこうか。
このパスワード解析には前提条件がある。
つまり、最初のパスワード「KaliPbst06」は、XOR（排他的論理和）でエンコードする、ということだ。XOR でエンコードされた文字は、XOR で元に戻すことができる、という前提条件で成り立っている。
論理演算子（AND、OR、XOR）については、タヌキよ、自分で勉強してね。以下に図解しようか。

パスワード：KaliPbst06 のＫ（大文字）で考えよう！

Ｋ（大文字）の ASCII コードは１６進数で「４ｂ」だ。２進数で「01001011」だ。これを、16 進数「７a」（２進数で「01111010」）（任意の数値で良い）を用いて XOR でエンコードすると次のようになる。

［エンコード］

```
４ｂ　：　０１００１０１１
７ａ　：　０１１１１０１０
ＸＯＲ　―――――――――――
　　　　００１１０００１　――→　３１（エンコードした１６進数）
```

［元に戻す］

```
３１　：　００１１０００１
７ａ　：　０１１１１０１０
ＸＯＲ　―――――――――――
　　　　０１００１０１１　――→　４ｂ（元に戻した１６進数）
                              （ASCII コード：大文字のＫ）
```

なるほど、これを上記の Python で行っているのか。
すごいなキツネ！

タヌキ、今回オイラが参考にした文献はマイナビ発行の「Ghidra 実践ガイド」だ。そこが提供している学習用実行ファイル「Lebel1」を使用したのだが、「Lebel1」では、内容が把握しずらいのでパスワード解析ということで「passAnaD03」と変更して使わせてもらった。ここで断って置くぞ！
他に、技術評論社の「UNIX　ネットワークプログラミング入門」も参考にしている。