問1　ソフトウェアの脆弱性に関する次の記述を読んで，設問1～9に答えよ。

V社は，従業員数100名のソフトウェア開発会社である。V社では，開発に関わる全員が情報セキュリティを意識した実装を行えるよう，開発経験が浅い従業員にセキュリティ教育を行っている。次は，任意の攻撃コードが実行され得る脆弱性について，開発チームのT主任が部下のUさんに教えていた時の会話である。

T主任：任意の攻撃コードが実行され得る脆弱性は幾つかある。確保済みメモリ領域を超えてデータを書き込んでしまう［　a　］と呼ばれる脆弱性の報告が以前から多かった。最近は解放したメモリ領域を後から使用してしまう［　b　］と呼ばれる脆弱性の報告も多くなってきている。

Uさん：［　b　］という脆弱性は具体的にはどのようなものなのですか。

T主任：例えば，図1のC++ソースコードからなるプログラム（以下，例示プログラムという）があったとする。例示プログラムは，図2に示すシステム構成の中で動作し，ノートと呼ぶメモ書き機能を実現するものであり，クライアントから利用者が自身の名前とメッセージを登録したり，それを他の利用者が参照したりする。例示プログラムでは，ノートはNote構造体で表現され，利用者の操作に応じて，NoteManagerクラスの各メンバ関数が個別に呼び出される。各メンバ関数では，ノートの生成（CreateNote），利用者の名前の登録（RegisterName），メッセージの登録（RegisterMsg），ノートの登録内容表示（DisplayNote），ノートの破棄（DeleteNote）を行う機能を実装している。例示プログラムにおいて，DeleteNoteメンバ関数内でm_noteの指すメモリ領域を解放しているが，仮にDeleteNoteメンバ関数が呼び出された直後にRegisterNameメンバ関数が呼び出されると，解放したm_noteの指すメモリ領域にアクセスできてしまう。これが［　b　］という脆弱性だ。例示プログラムでは，悪意をもつ利用者（以下，攻撃者という）の操作によって，任意の攻撃コードを実行され，サーバを乗っ取られてしまうおそれがある。

```
 1: #include <cstdio>
 2:
 3: struct Note{
 4:   char *name;
 5:   char *msg;
 6: };
 7:
 8: class NoteManager{
 9:   Note *m_note;
10: public:
11:   NoteManager(){ m_note = NULL; }
12:   void CreateNote(){
13:     if(m_note = new Note()){
14:       m_note->name = NULL;
15:       m_note->msg = NULL;
16:     }
17:   }
18:   void RegisterName(){
19:     if(m_note && !m_note->name) m_note->name = new char[8];
20:     if(m_note && m_note->name){
21:       printf("Input name: ");
22:       scanf("%7s%*[^¥n]%*c", m_note->name);
23:     }
24:   }
25:   void RegisterMsg(){
26:     if(m_note && !m_note->msg) m_note->msg = new char[100];
27:     if(m_note && m_note->msg){
28:        printf("Input message: ");
29:        scanf("%99s%*[^¥n]%*c", m_note->msg);
30:     }
31:   }
32:   void DisplayNote(){
33:     if(m_note && m_note->name) printf("Name: %s¥n", m_note->name);
34:     if(m_note && m_note->msg) printf("Message: %s¥n", m_note->msg);
35:   }
36:   void DeleteNote(){
37:     delete[] m_note->name;
38:     delete[] m_note->msg;
39:     delete m_note;
40:   }
41: };
42: (省略)
```

注記　メモリアドレスが32ビットの環境で動作させるものとする。

図1　脆弱性が存在するC++ソースコード

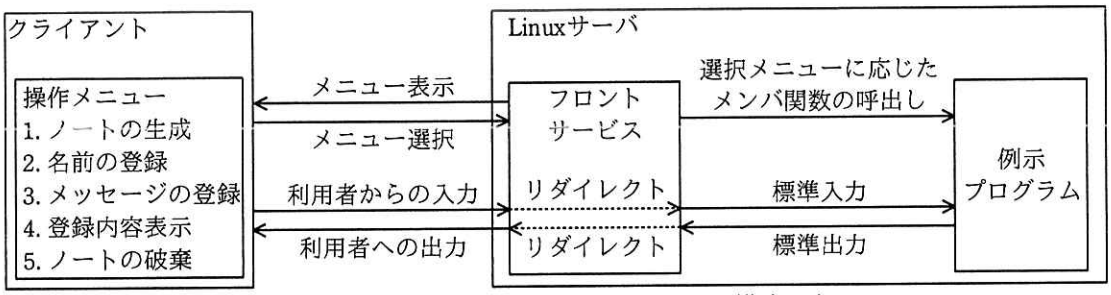
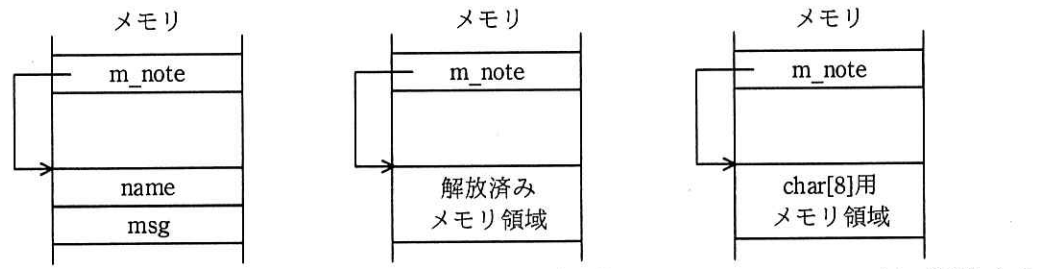


図 2　例示プログラムが動作するシステム構成の例

U さん：解放したメモリ領域にアクセスされると，どのように攻撃コードが実行されるのですか。

T 主任：例示プログラムにおいて，図 3 に示す(1)~(3)の順でメンバ関数の呼出しが行われたとしよう。その場合，図 3 の(1)で確保されていた Note 構造体用のメモリ領域と，図 3 の(3)で確保された char[8]用のメモリ領域が同じアドレスに割り当てられる可能性がある。その場合，①RegisterName メンバ関数内で読み込まれる攻撃者からの入力値によって，元々Note 構造体用であったメモリ領域が上書きされる。このときの攻撃者からの入力値がうまく細工されていると，②次に RegisterName メンバ関数が呼ばれた際，その際に読み込まれる攻撃者からの入力値が，攻撃者の指定したアドレスに書き込まれることになる。このように，攻撃者からの入力値が攻撃者の指定したアドレスに書き込まれる場合には攻撃コードが実行され得る。



(1) CreateNoteメンバ関数実行後　(2) DeleteNoteメンバ関数実行後　(3) RegisterNameメンバ関数内のnew実行後

図 3　脆弱性を悪用する関数呼出しの過程とメモリマップ

Uさん：もう少し具体的に説明してください。

T主任：関数テーブルの例で説明しよう。ここでいう関数テーブルとは，プログラム中で呼び出している共有ライブラリに含まれる関数（以下，ライブラリ関数という）の実行コードの先頭アドレスが記録されたテーブルだ。ライブラリ関数の呼出し時には，図 4 に示すように関数テーブルに記録された実行コードの先頭アドレスの値を参照してライブラリ関数の実行コードに処理が遷移する。

図4　ライブラリ関数の呼出し時の動き

T主任：攻撃者が既に，攻撃コードをメモリ上に書き込んでいるとしよう。この状態で，例えば，攻撃コードが存在するアドレスを関数テーブルに書き込まれた場合，関数の呼出し時に関数テーブルが参照されると，攻撃コードに処理が遷移してしまう。

Uさん：例示プログラムを攻撃する場合だと，関数テーブルに書き込むアドレスは具体的にどのような値になりますか。

T主任：例えば，RegisterMsg メンバ関数の呼出しによって m_note->msg が指し示すメモリ領域に攻撃コードが書き込まれていて，その先頭アドレスが 0x0b123400 と分かっていたとする。その場合，関数テーブルが表 1 に示すようになっていたとすると，アドレス［　c　］番地に値［　d　］を書き込むことによって，次に CreateNote メンバ関数が呼び出された際，攻撃コードに処理が遷移することになる。

表1　関数テーブル

アドレス	値	値の意味
（ア）0x08049e30	（キ）0xf7cfcd70	delete の実行コードの先頭アドレス
（イ）0x08049e38	（ク）0xf7ce9370	scanf の実行コードの先頭アドレス
（ウ）0x08049e3c	（ケ）0xf7cd7670	printf の実行コードの先頭アドレス
（エ）0x08049e40	（コ）0xf7cff150	new の実行コードの先頭アドレス
（オ）0x08049e44	（サ）0xf7cff230	new[]の実行コードの先頭アドレス
（カ）0x08049e4c	（シ）0xf7cfcdd0	delete[]の実行コードの先頭アドレス

U さん：RegisterMsg メンバ関数の呼出しによって入力された攻撃コードは［　e　］領域に書き込まれるので，データ実行防止と呼ばれる機能が有効化されていた場合，実行されませんよね。

T 主任：確かにそのとおりだ。ただし，③関数テーブルに書き込むアドレスとして，例えば，共有ライブラリ内のメモリアドレスを選べば，データ実行防止が有効化されていた場合でも，攻撃者が任意の処理を実行できる可能性がある。例示プログラムにおいて，共有ライブラリ内のメモリアドレスが表 2 のようになっていたとすると，関数テーブルに書き込むアドレスを［　f　］番地にすることによって，データ実行防止が有効化されていた場合でも，/bin/sh を起動して任意のシェルコマンドを実行できる可能性がある。

表2　共有ライブラリ内のメモリアドレス

アドレス	内容
（ア）0xf7cc8da0	system の実行コード
（イ）0xf7cd7670	printf の実行コード
（ウ）0xf7ce9370	scanf の実行コード
（エ）0xf7cfcd70	delete の実行コード
（オ）0xf7cfcdd0	delete[]の実行コード
（カ）0xf7cff150	new の実行コード
（キ）0xf7cff230	new[]の実行コード
（ク）0xf7de99ab	"/bin/sh"の文字列

U さん：共有ライブラリ内のメモリアドレスは，表 2 のように前もって知ることができるものなのですか。

T 主任：共有ライブラリをメモリに読み込む際，それを配置するアドレスを毎回ラン

ダムに選ぶASLR（Address Space Layout Randomization）と呼ばれるセキュリティ機能がある。この機能が有効な場合，共有ライブラリ内のメモリアドレスを前もって知ることは難しい。しかし，例示プログラムにおいて，図3の(3)の状態をつくり出せれば，RegisterNameメンバ関数と［ g ］メンバ関数を利用することによって，ASLRが有効化されていた場合でも，共有ライブラリ内のメモリアドレスを特定できる可能性がある。データ実行防止やASLRなどは効果的な機能ではあるが，根本的な対策にはならない。やはり脆弱性そのものを修正することが重要だ。

U さん：では，［ b ］の脆弱性を修正するには，例示プログラムではどうすればよいのでしょうか。

T 主任：図1の39行目の直後に［ h ］という1文を加えればよいだろう。

U さんは，今回学んだことをコードレビューの観点として生かしていくことにした。

設問1　本文中の［ a ］，［ b ］に入れる適切な脆弱性名を，解答群の中から選び，記号で答えよ。

解答群

ア　CSRF	イ　SQLインジェクション
ウ　Use-After-Free	エ　クロスサイトスクリプティング
オ　コマンドインジェクション	カ　バッファオーバフロー
キ　フォーマットストリングバグ	ク　レースコンディション

設問2　本文中の下線②のようになるためには，本文中の下線①で読み込まれる攻撃者からの入力値はどのような値である必要があるか。攻撃者の指定したアドレスを0x12345678，改行コードを0x0aとした場合について，入力値の具体的なバイト列を14字以内の16進数文字列で答えよ。ここで，アドレスは32ビットであり，バイトオーダがリトルエンディアンのバイトマシンによって扱われるものとする。

設問3　本文中の［ c ］に入れる適切なアドレスを表1中の（ア）～（シ）から選び，記号で答えよ。

設問4　本文中の［ d ］に入れる適切な値を16進数で答えよ。

設問5　本文中の［ e ］に入れるメモリ領域の名称を答えよ。

設問6　本文中の下線③の理由を，45字以内で述べよ。

設問7　本文中の［ f ］に入れる適切なアドレスを表2中の（ア）～（ク）から選び，記号で答えよ。

設問8　本文中の［ g ］に入れるメンバ関数の名前を答えよ。

設問9　本文中の［ h ］に入れる適切なソースコードを答えよ。