問1　ソフトウェア開発に関する次の記述を読んで，設問1～3に答えよ。

U社は，IoT機器の開発を行う，従業員数400名の企業である。IoT機器のソフトウェアの開発にはC/C++言語を使っている。IoT機器に搭載するOSには，Linuxを利用してきたが，今後はLinux以外も利用する予定である。これまで製品に大きなトラブルはなかったが，2020年東京オリンピック・パラリンピックに向けてIoT機器に関するセキュリティリスクが高まると経営層が判断し，開発におけるセキュリティ対策を強化することになった。そこで，開発部のL部長とX主任がセキュリティ対策技術を調査した。

〔メモリ破壊攻撃の概要〕

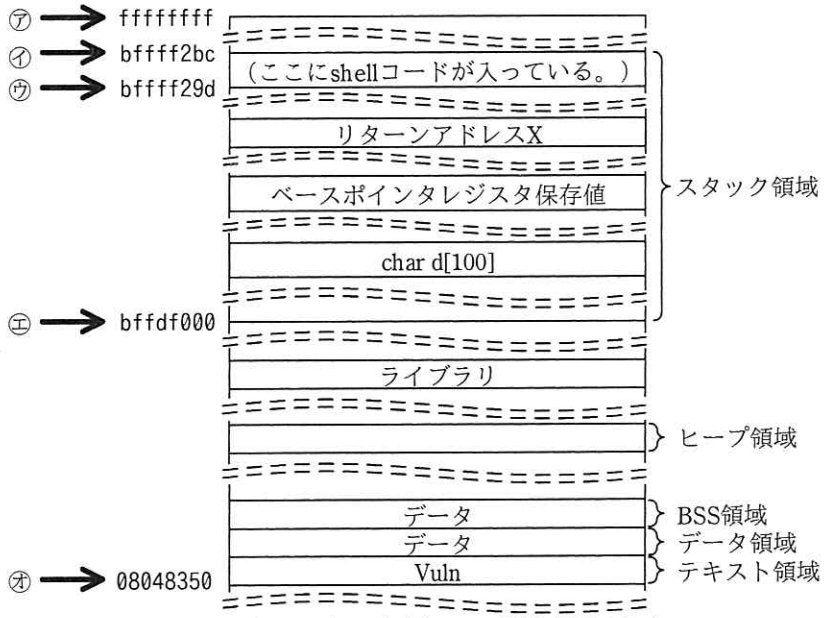
メモリ破壊脆弱(ぜい)性は，プログラム実行時に，メモリ上にある制御情報を書き換えることによって，実行制御を奪うなどのメモリ破壊攻撃に悪用される。メモリ破壊脆弱性の一種にバッファオーバフロー脆弱性がある。

例えば，図1に示すプログラムVulnがあったとする。Vulnは，スタックバッファオーバフロー脆弱性の学習用に作成した，32ビット版Linuxで実行可能なプログラムである。図2はVuln内の関数fooが呼び出された後のメモリマップである。プログラム実行時に，変数bが指し示すデータが不正な場合，そのデータによって，［a］が［あ］に書き換えられると，関数fooの終了時にshellコードへ処理が遷移する。しかし，①このような遷移があっても，データ実行防止機能（以下，DEPという）が機能していると，攻撃は成功しない。

攻撃者が，DEPを回避するため，図2中のshellコードへ処理を遷移させる代わりに，［b］中の実行可能なコードや［c］領域にマップされているVulnの断片コードを利用するケースがある。例えば，攻撃の際，［a］を［b］関数の先頭アドレスで書き換えて，攻撃者の意図した［b］関数を呼び出す［d］攻撃もある。

```
      (省略)
 1:   int main(int argc, char *argv[]) {
 2:     char *a, *x;
        (省略, argvに応じてサイズを確保する。)
        (省略, ここでa, xがポイントする領域にargvからデータをコピーする。)
 3:     foo(a, x);
        (省略, ここでその他の必要な処理をする。)
 4:   }
 5:   int foo(char *b, char *c) {
 6:     char d[100];
        (省略)
 7:     strcpy(d, b);
 8:     if (d[0] == 0) {
 9:       err_out(c);
          (省略)
10:     }
        (省略)
11:     return 0;
12:   }
13:   int err_out(char *errmsg) {
14:     char s1[100];
15:     int i=0;
        (省略)
16:     while ((s1[i++] = *errmsg++) != '¥0');
17:     fprintf(stderr, "Error : %s ¥n", s1);
        (省略)
18:     return 0;
19:   }
```

図1　スタックバッファオーバフロー脆弱性のあるプログラム Vuln



注記　メモリアドレスは4バイトの16進数表記である。

図2　関数 foo が呼び出された後のメモリマップ

〔メモリ破壊攻撃に対する対策技術〕

現在，メモリ破壊攻撃に対する対策技術（以下，M 対策技術という）が普及している。X 主任は，DEP を含めた代表的な M 対策技術を調査し，表 1 にまとめた。

表 1　M 対策技術の概要

技術名	概要	期待される効果	備考
DEP	（省略）	（省略）	プログラムによっては適用できない。
SSP（Stack Smashing Protection）	スタック領域で canary と呼ばれる値を利用してスタックバッファオーバフローの有無を確認する技術	スタックバッファオーバフローを検知し，抑制する。	（省略）
ASLR（Address Space Layout Randomization）	プログラムの実行時に，データ領域，ヒープ領域，スタック領域及びライブラリを，ランダムにマップする OS の技術	（省略）	32 ビット OS の場合，効果が限定的である。
PIE（Position Independent Executable）	プログラムの実行時に，ASLR が対象とする領域に加えて，テキスト領域もランダムにマップする技術	（省略）	プログラムによっては適用できない。
Automatic Fortification	バッファオーバフロー脆弱性の原因となりうる脆弱なライブラリ関数を，コンパイル時に境界チェックを行う安全な関数に置換する技術	境界チェックによって，オーバフローを抑制する。	境界チェックにおいて，書込み先のサイズが不明な場合は機能しない。

〔M 対策技術の動作概要〕

例えば，Vuln のコンパイル時に SSP が適用されていると，関数 foo を呼び出す際，図 2 のベースポインタレジスタ保存値より下位に［　e　］が挿入される。もしも，［　e　］が上書きされた場合は，攻撃と判断し，Vuln の実行を停止する。

なお，Vuln の場合は簡単ではないが，攻撃者が［　e　］の値を正確に推測して上書きできてしまうと，［　a　］の書換えが可能となり，［　d　］攻撃を防げない。その対策としては，ライブラリ関数のアドレス推定を困難にさせる［　f　］が有効である。

しかし，［　f　］は［　c　］領域にある実行可能なコードを用いる攻撃に対しては効果がない。そうした攻撃は PIE によって緩和される。さらに，Vuln の場合，Automatic Fortification によって，ライブラリ関数［　g　］を安全な関数に置き換えることで，バッファオーバフローの原因を排除することができる。

〔脆弱性対策強化〕

L 部長と X 主任が表 1 の技術を確認したところ，表 1 の備考欄の指摘以外にも②

Automatic Fortification ではバッファオーバフローの原因を排除できないケースがあると分かった。

L 部長は次に，表 1 の技術を適用することによる影響を確認した。その結果，次のことが分かった。例えば，ソースコードに脆弱性があっても，SSP を適用してコンパイルしていると，メモリ破壊攻撃が成立しないが，そのソースコードを③別の開発環境でコンパイルすると問題となる場合があることが分かった。

これらについては，U 社内でコーディングスタンダードを定め，それによって対処することにした。検討の結果，U 社は表 1 の技術を全て採用することにした。

設問 1　〔メモリ破壊攻撃の概要〕について，(1) ～(3) に答えよ。

(1) 本文中の［ a ］～［ d ］に入れる適切な字句を，解答群の中から選び，記号で答えよ。

解答群

ア　Return-to-libc　　イ　ROP

ウ　テキスト　　エ　ヒープ

オ　ベースポインタレジスタ保存値　　カ　ライブラリ

キ　リターンアドレス X

(2) 本文中の［ あ ］に入れる適切なアドレス値を図 2 中から選び，㋐ ～ ㋔ の記号で答えよ。

(3) 本文中の下線①について，攻撃が成功しない理由を 35 字以内で述べよ。

設問 2　〔M 対策技術の動作概要〕について，(1)，(2) に答えよ。

(1) 本文中の［ e ］，［ f ］に入れる適切な字句を，表 1 中の用語を用いて答えよ。

(2) 本文中の［ g ］に入れる適切なライブラリ関数名を答えよ。

設問 3　〔脆弱性対策強化〕について，(1)，(2) に答えよ。

(1) 本文中の下線②について，図 1 のプログラムにおいて，排除できないケースに該当する処理を行番号で答えよ。また，排除できない理由を 30 字以内で述べよ。

(2) 本文中の下線③について，どのような問題か。また，どのような開発環境の場合に問題となるか。それぞれ 25 字以内で述べよ。