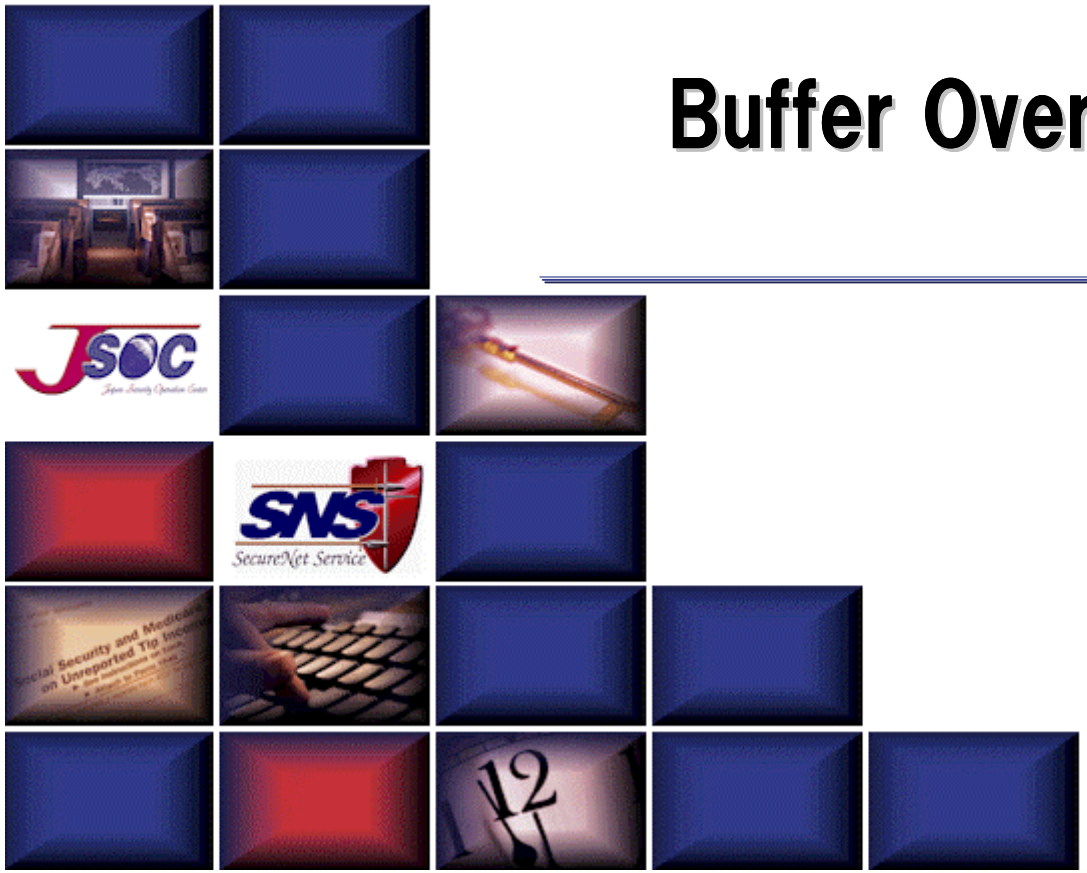


Buffer Overflowのケーススタディとデモンストレーション

株式会社ラック
新井 悠
y.arai@lac.co.jp
http://www.lac.co.jp/security/

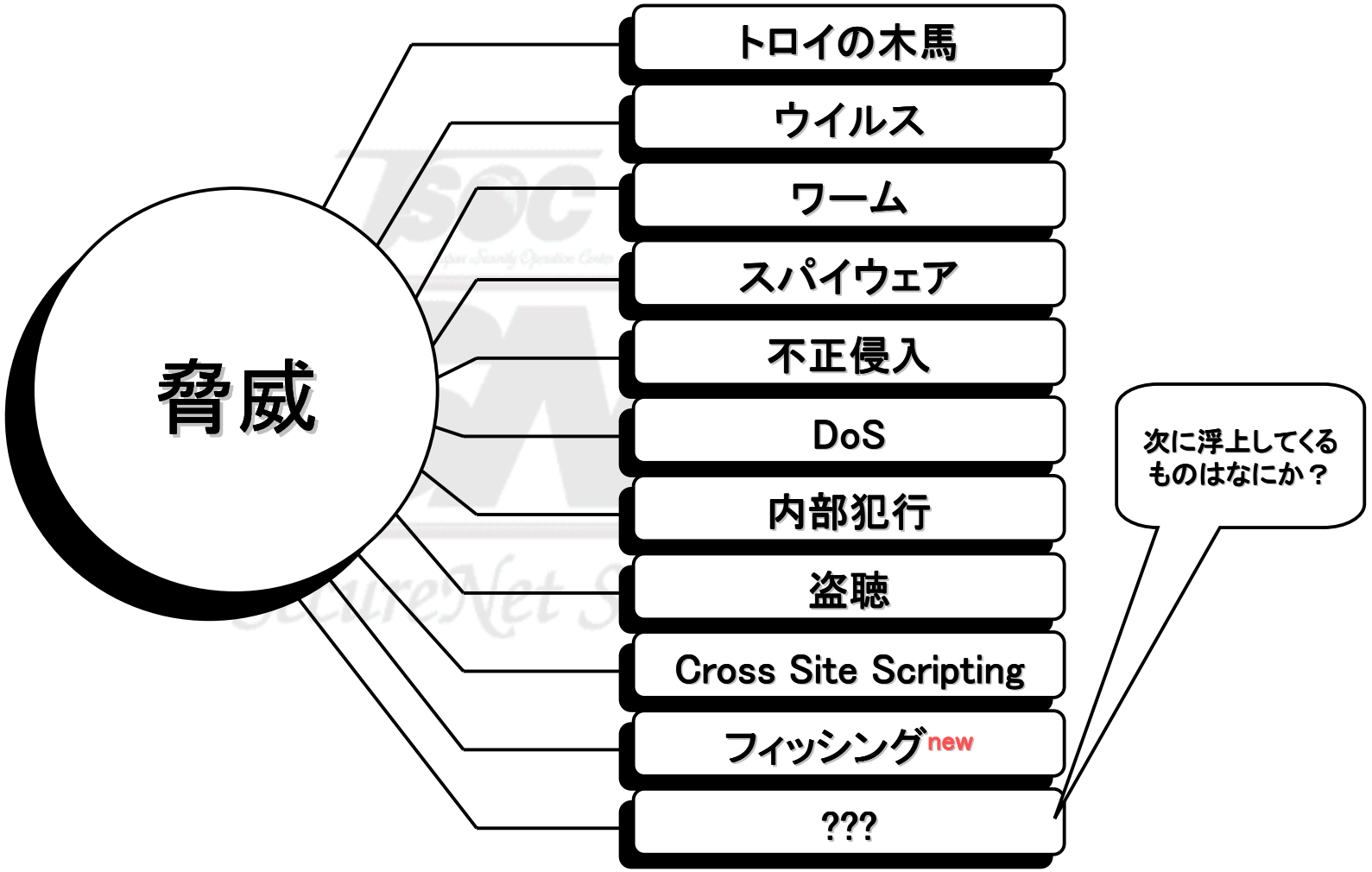
アジェンダ

- はじめに
  - インターネットセキュリティにおける脅威の現状
- バッファオーバーフローとは何か？
  - バッファオーバーフローのカテゴリ
  - スタックオーバーフロー
  - ヒープオーバーフロー
- ワームとバッファオーバーフロー
  - ワーム化する条件
- プロアクティブなバッファオーバーフロー対策
  - /GSオプション
  - DEP(Data Execution Prevention)
- まとめ

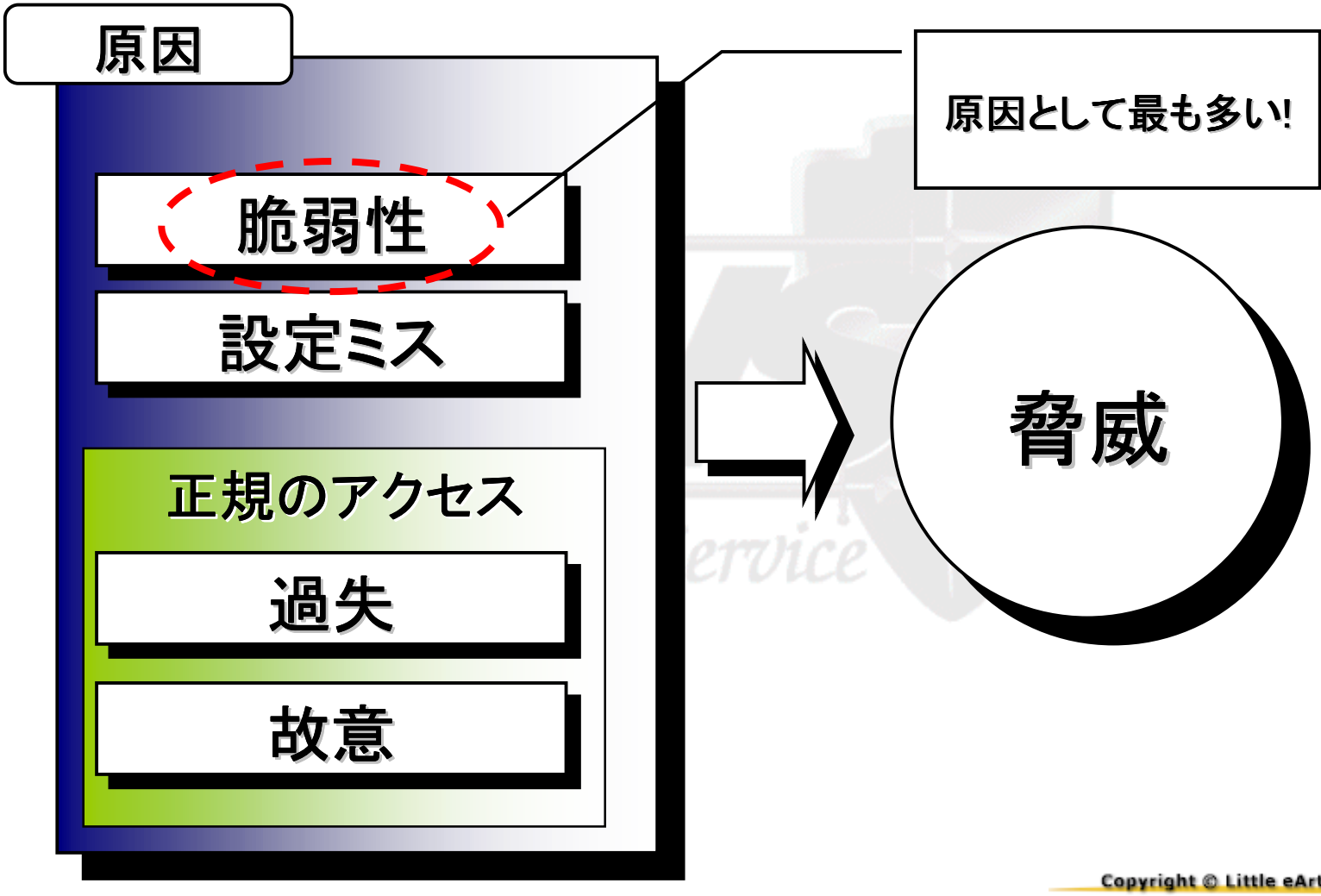
本セッションの前提

- Windowsにおけるバッファオーバーフローについての説明が主体
- バッファオーバーフローとは、CやC++、あるいはアセンブラといったプログラミング言語を用いて作成されたプログラムで発生する**現象**
- バッファオーバーフロー自体は現象でしかなく、それを攻撃に転化させるテクニックが必要となる

ネットワークセキュリティにおける脅威の多様化



脅威を生み出す原因



例:ウイルスの脆弱性悪用状況

参照元:トレンドマイクロ社の2003年ウイルス感染被害年間レポート
(http://www.trendmicro.com/jp/security/report/report/archive/2003/mvr2003-12.htm)

ランキング	名称	脆弱性の悪用有無
1	Klez	○(MS01-020)
2	Nachi	○(MS03-026)
3	Redlof	○(MS00-075)
4	Blaster	○(MS03-026)
5	Swen	○(MS01-020)
6	Bugbear	○(MS01-020)
7	Opaserv	○(MS00-072)
8	Dluca	
9	Fortnight	○(MS00-075)
10	Istbar	

現場の声

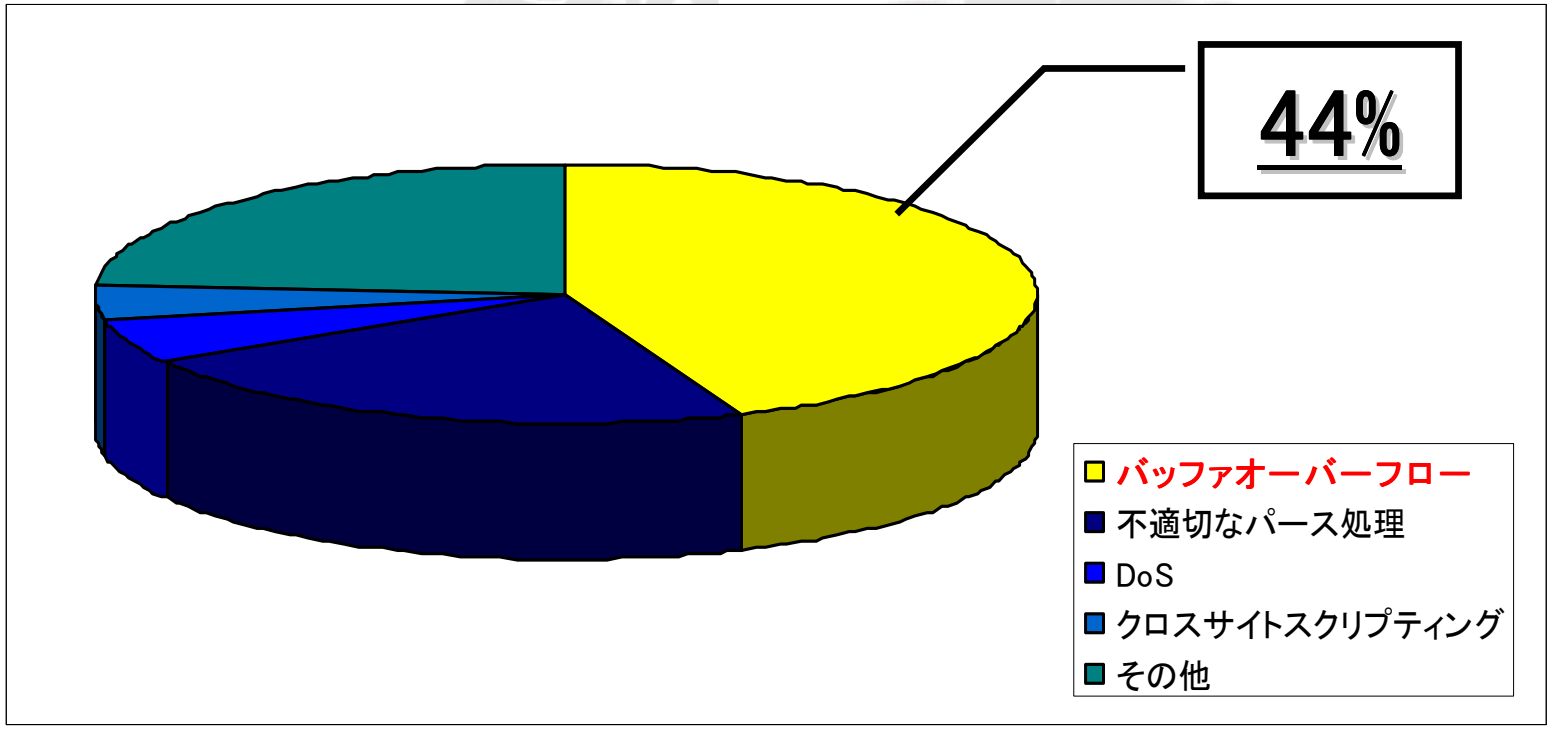
■ ソフトウェア開発でのセキュリティは？
- 要件仕様を満たすことが優先される
- コストが優先される
- 納期が優先される

■ システム運用でのセキュリティは？
- 安定稼動が優先される
- パッチではない代替策が優先される

Point

リスクはいつのまにかに『選択されて』いる

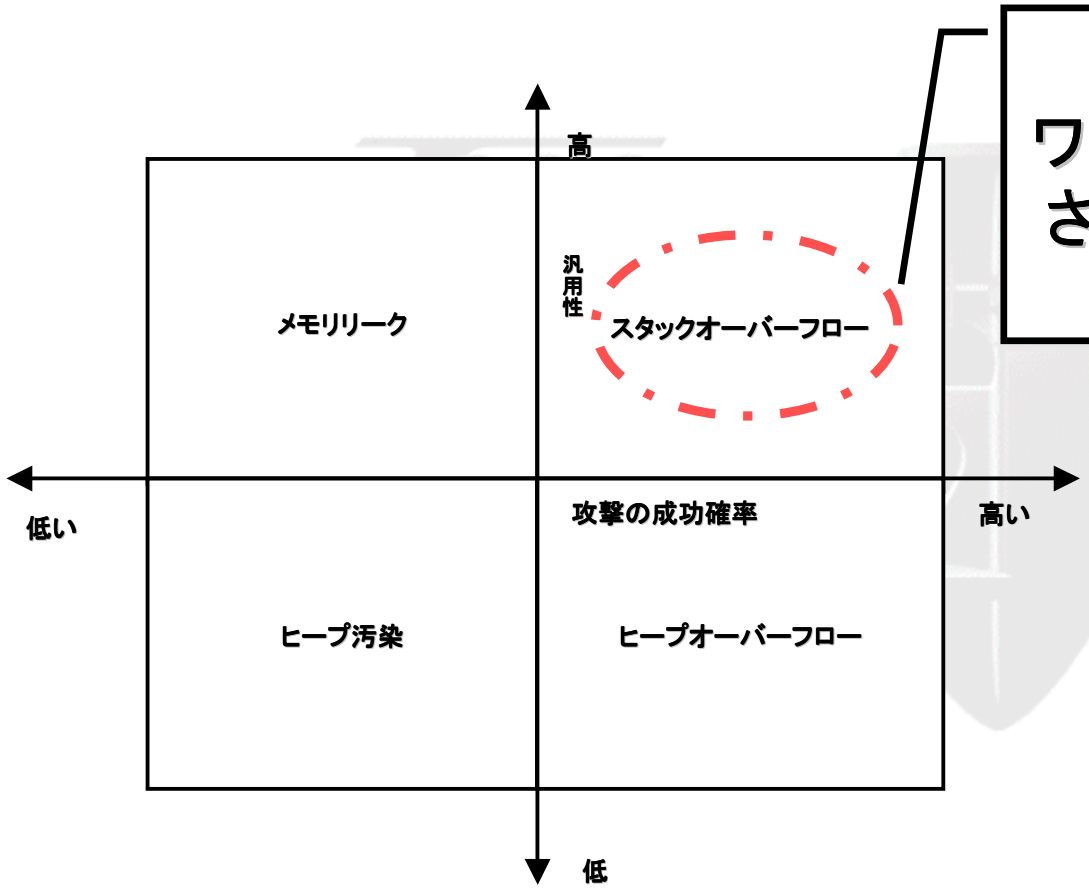
2003年にマクロソフトセキュリティ情報で公開/修正された脆弱性の内訳



バッファオーバーフローとは何か？

バッファオーバーフローとは何か？

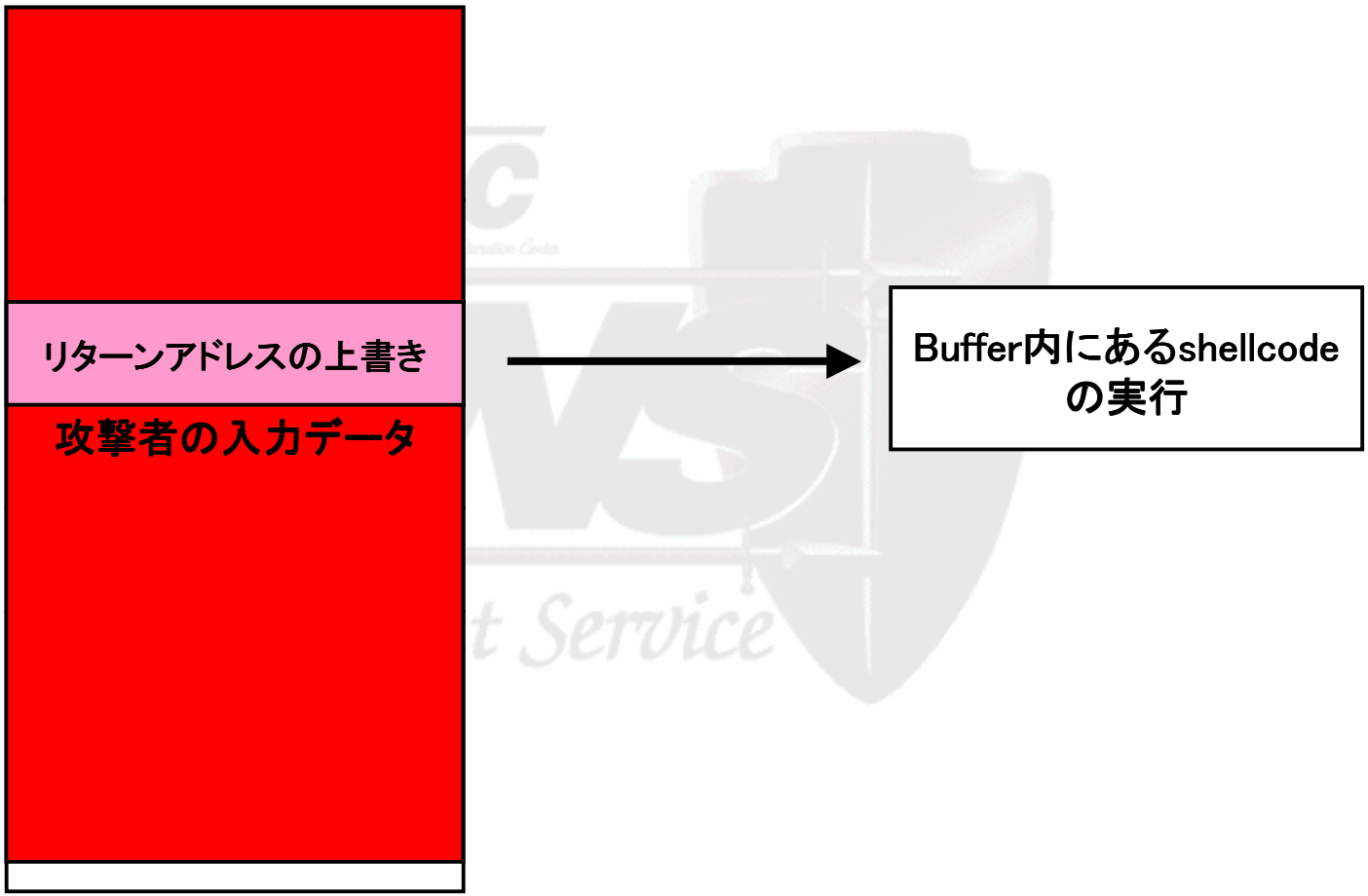
バッファオーバーフローの分類



ワームに利用されやすい!!!

バッファオーバーフローとは何か？

よくあるバッファオーバフローの例



スタックオーバーフローの特徴

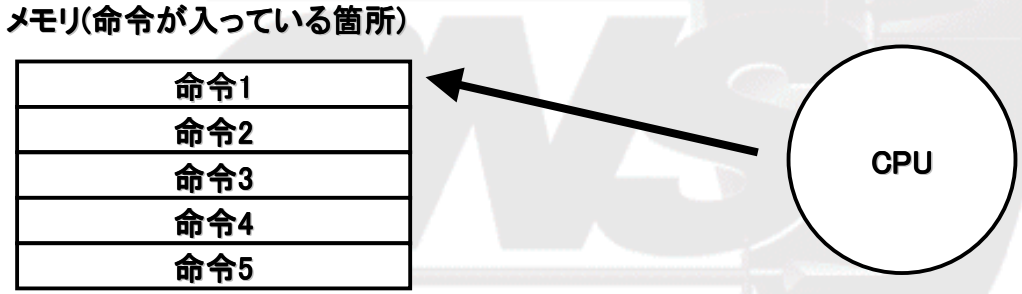
- EIPレジスタの値を操作可能
- strcpy(), gets(), spfintf(), memcpy(), …
- 攻撃に利用することは容易
- 絶滅危惧種

スタックオーバーフローの実例

脆弱性	ワームの例
MS03-026(RPC Locator Service Buffer Overflow)	Blaster、Welchia
MS03-049(Workstation Service Buffer Overflow)	Gaobot
MS04-011(LSASS Buffer Overflow)	Sasser

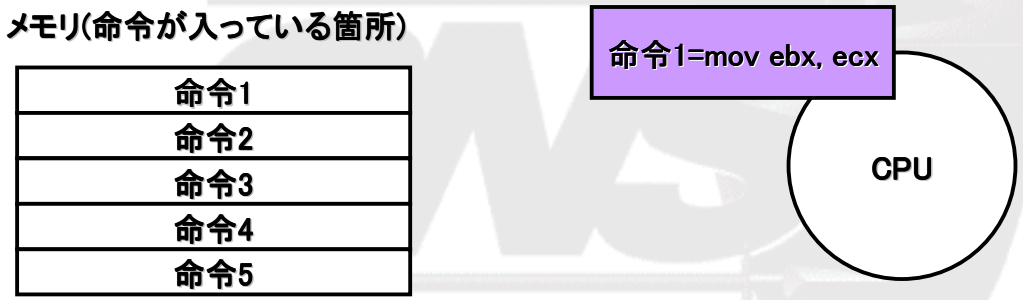
バッファオーバーフローとは何か？

- バッファオーバーフローを理解するために
  - Fetch, Decode, Executeのサイクル



①命令をとってくる(Fetch)

バッファオーバーフローとは何か？

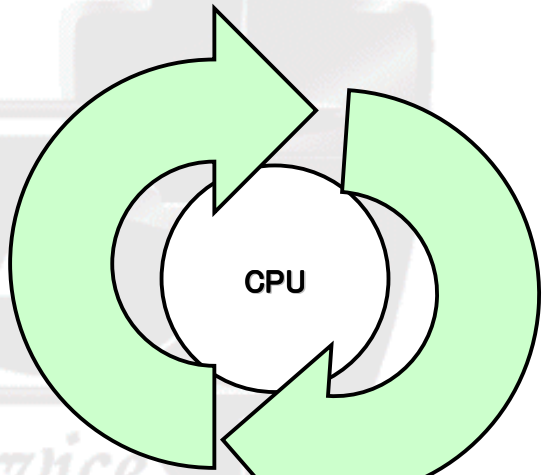


②とってきた命令の解読(Decode)

バッファオーバーフローとは何か？

メモリ(命令が入っている箇所)

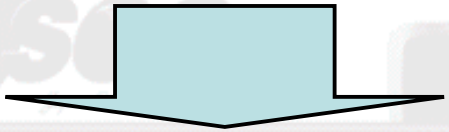
命令1
命令2
命令3
命令4
命令5



③命令の実行(Execute)

バッファオーバーフローとは何か？

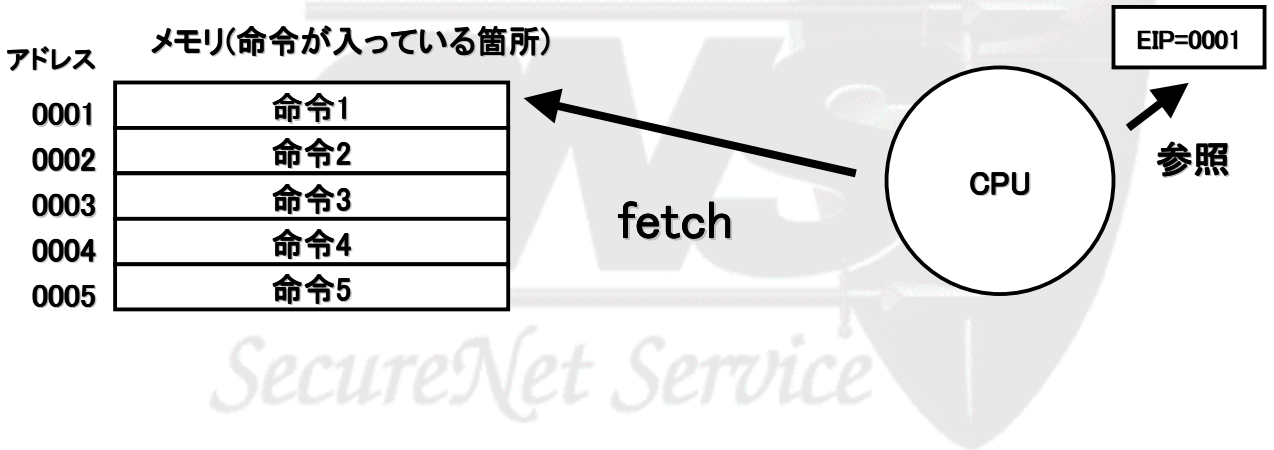
メモリは『アドレス』で管理されている



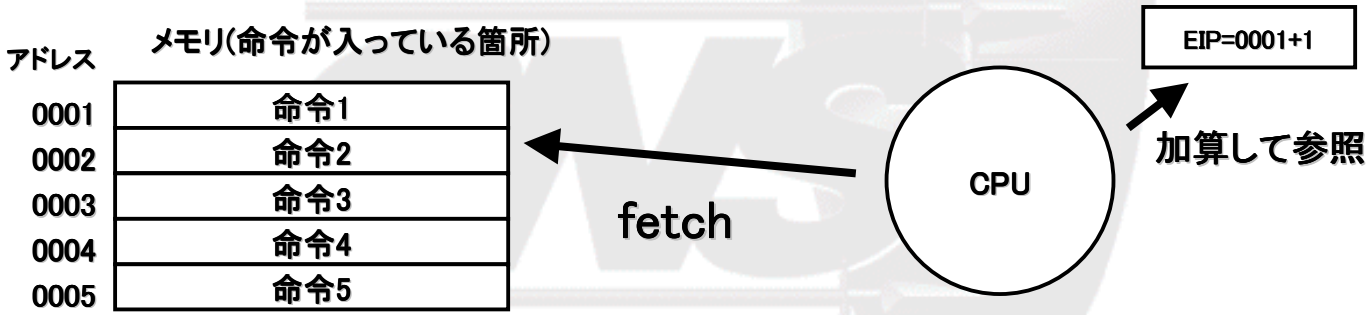
命令を読み出すためには、アドレスを
指定して読み出さなければならない

バッファオーバーフローとは何か？

- EIPレジスタ
- レジスタとは、CPUの中に含まれる小さな記憶領域



バッファオーバーフローとは何か？

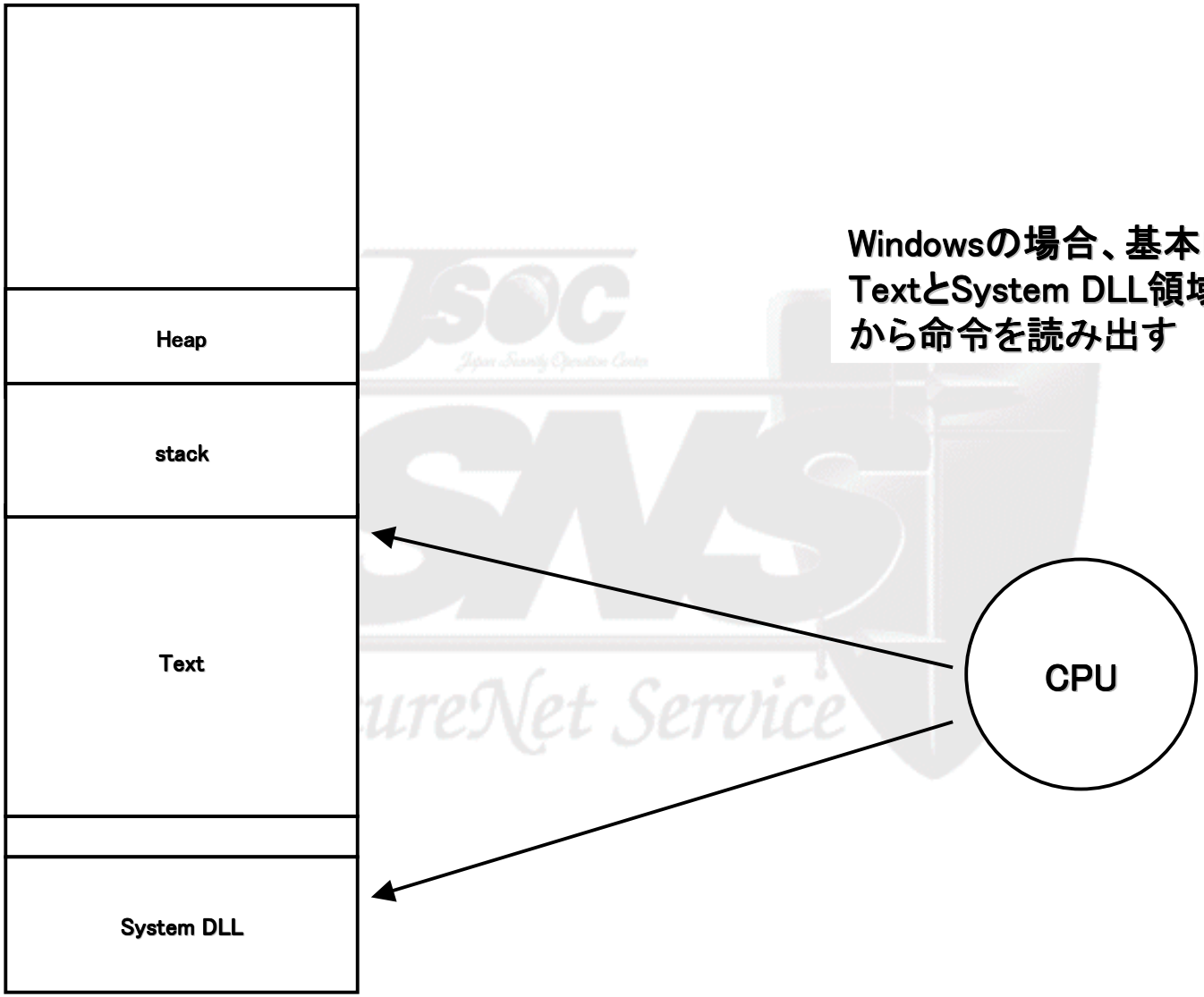


バッファオーバーフローとは何か？

Windows のメモリセグメント

名称	役割
Text	プログラムコードがロードされる
Stack	静的なデータがロードされる
Heap	動的なデータがロードされる
System DLL	システムAPIがロードされる

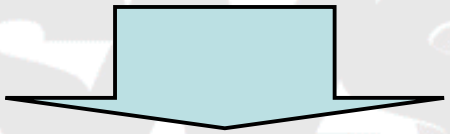
バッファオーバーフローとは何か？



Windowsの場合、基本的にTextとSystem DLL領域だけから命令を読み出す

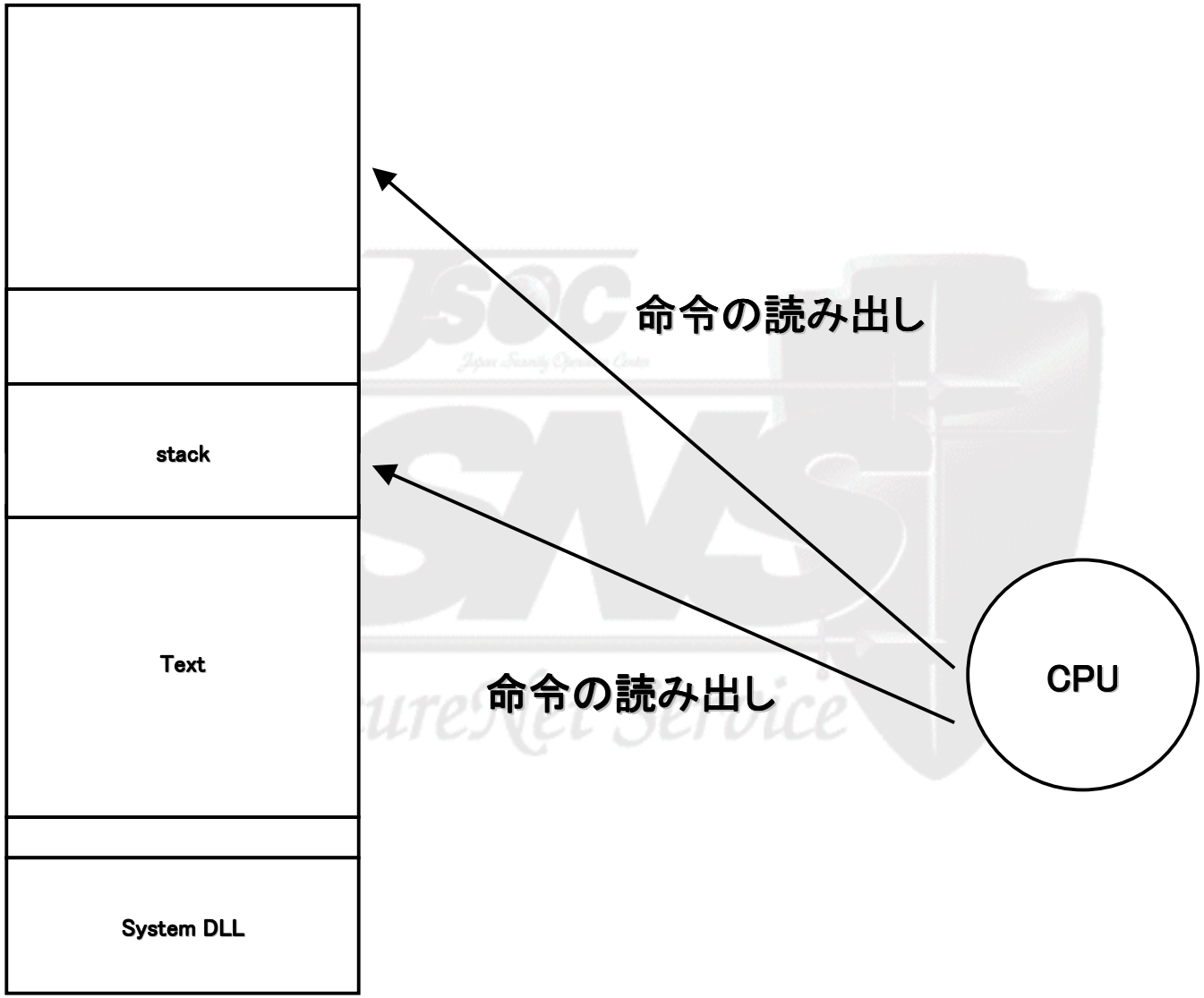
バッファオーバーフローとは何か？

スタックオーバーフローは:
・EIPレジスタの値を操作できる



EIPレジスタの値を操作できるので、
任意の場所から命令を読み出せる

バッファオーバーフローとは何か？



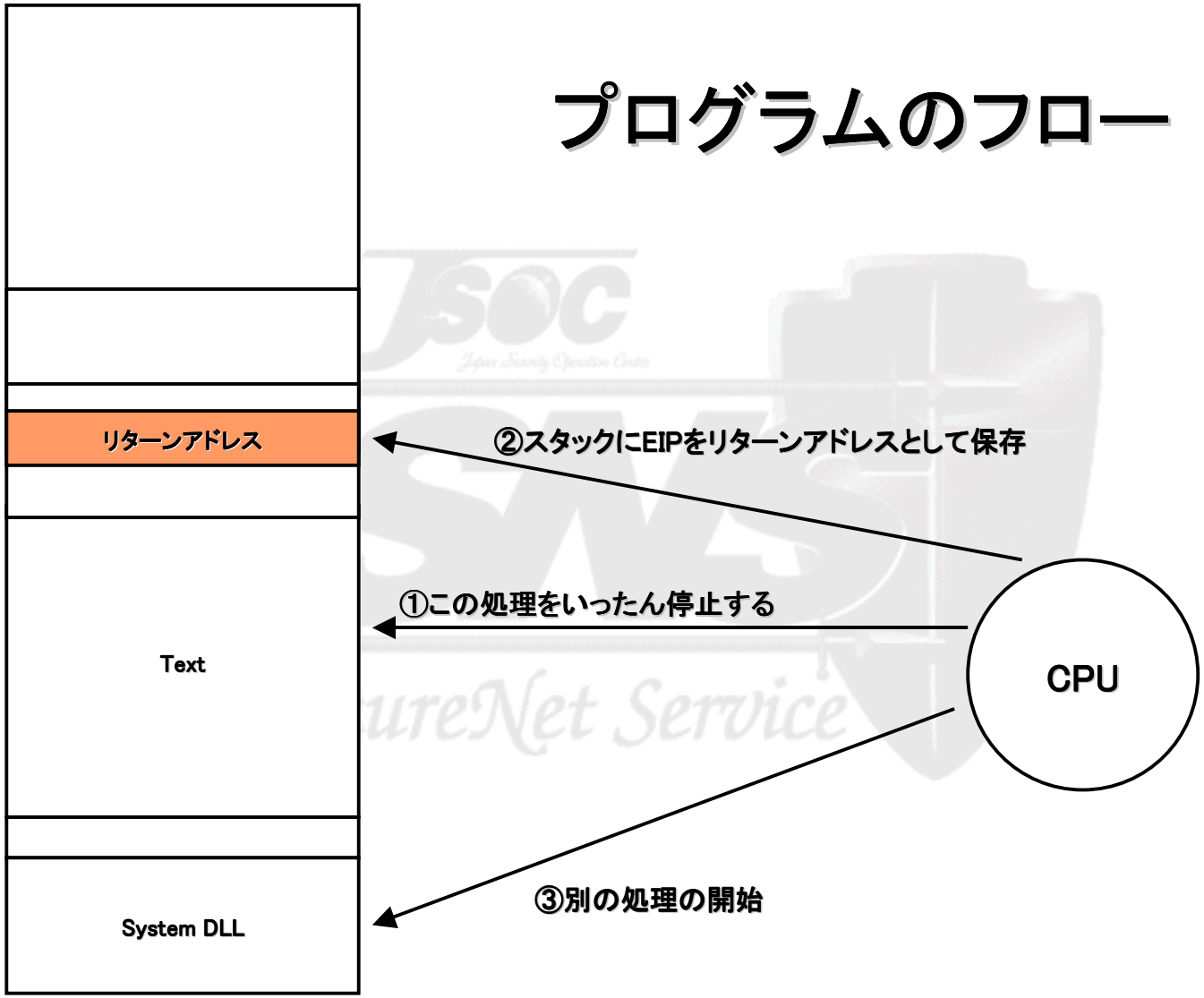
バッファオーバーフローとは何か？

- スタックオーバーフローがEIPレジスタの値を操作できる理由:リターンアドレスの上書き

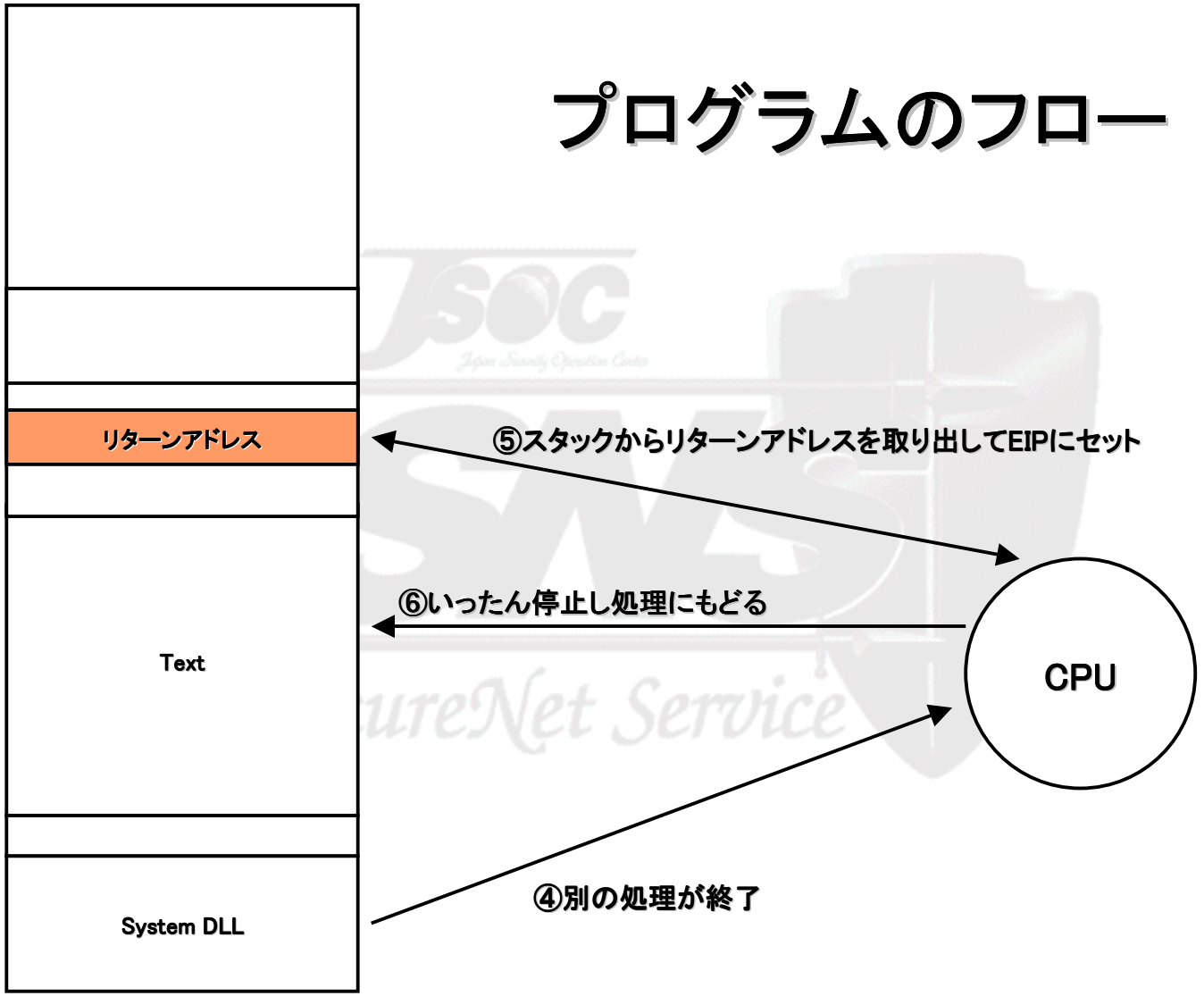
リターンアドレスとは？

•CPUがいったん別の処理をしてから、前の処理に戻るためにEIPレジスタの値を保存しておく箇所のこと

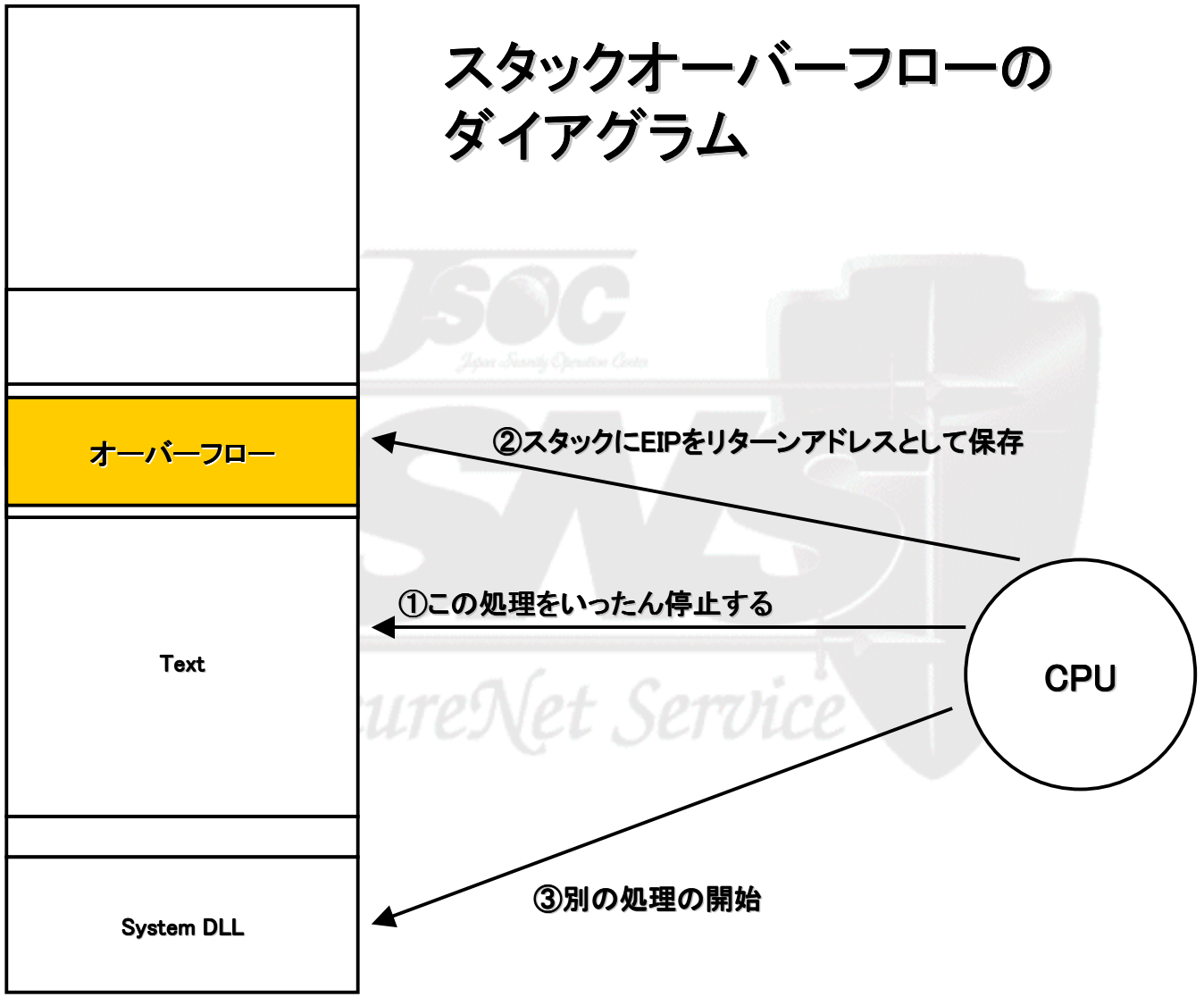
バッファオーバーフローとは何か？



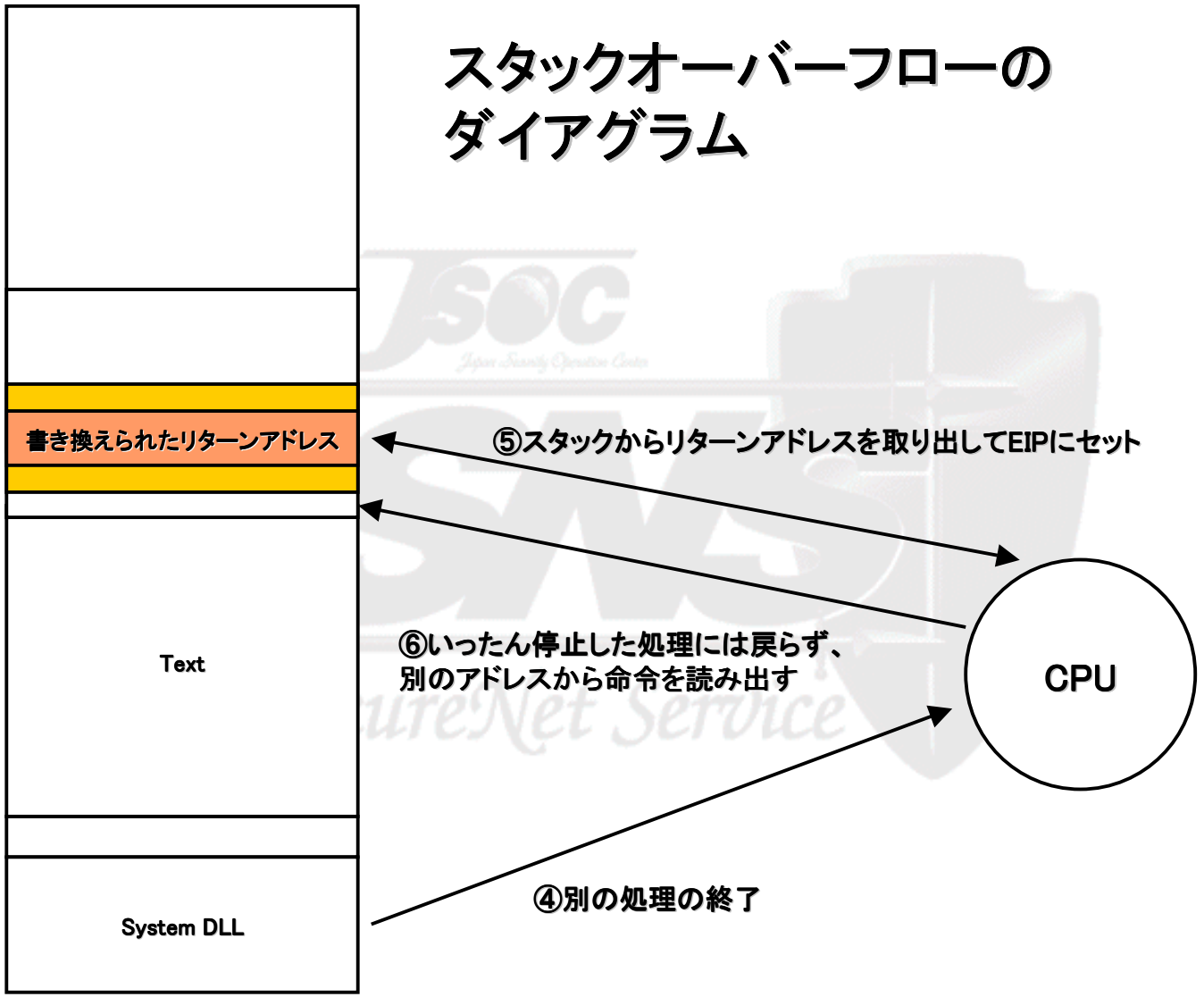
バッファオーバーフローとは何か？



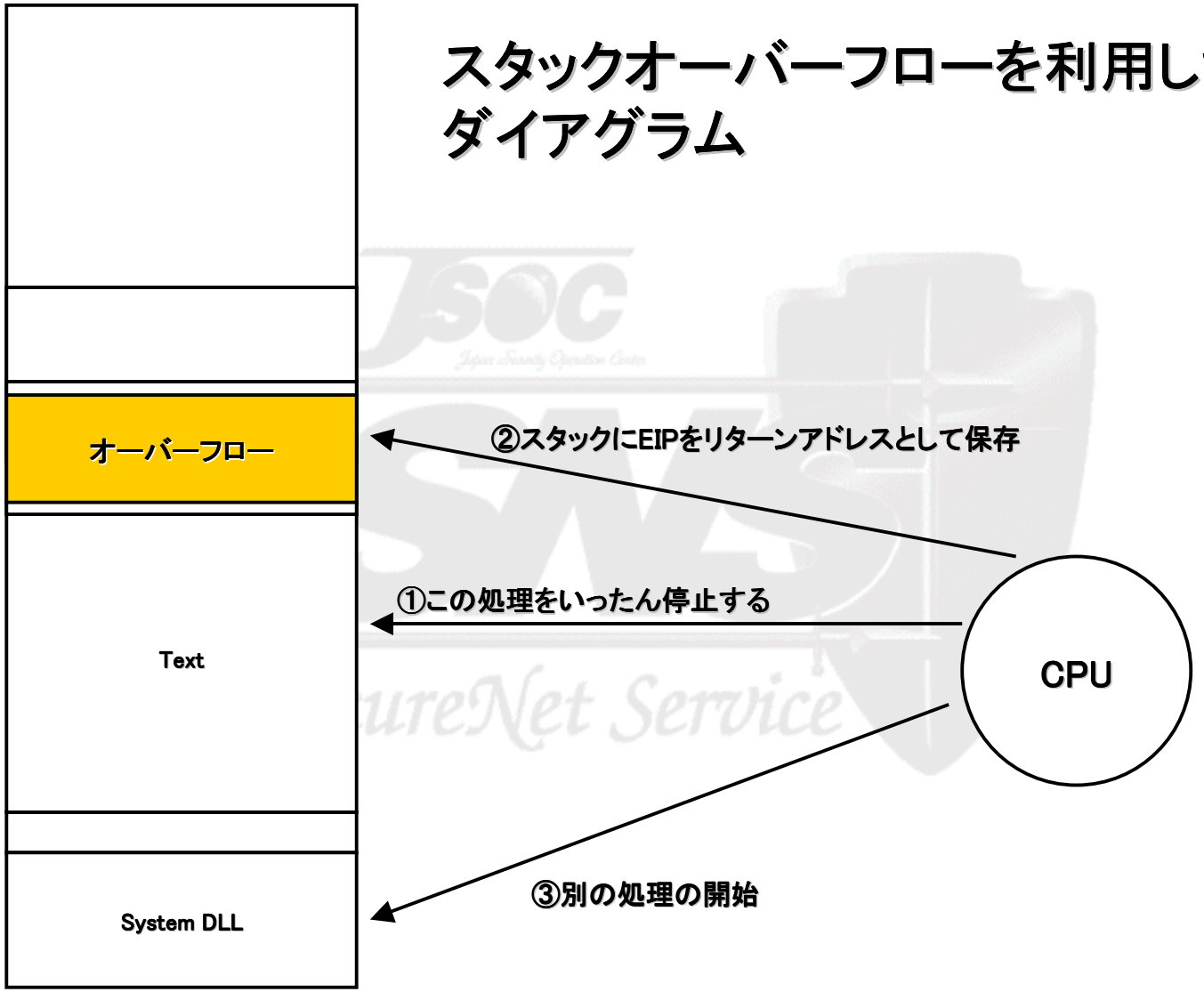
バッファオーバーフローとは何か？



バッファオーバーフローとは何か？



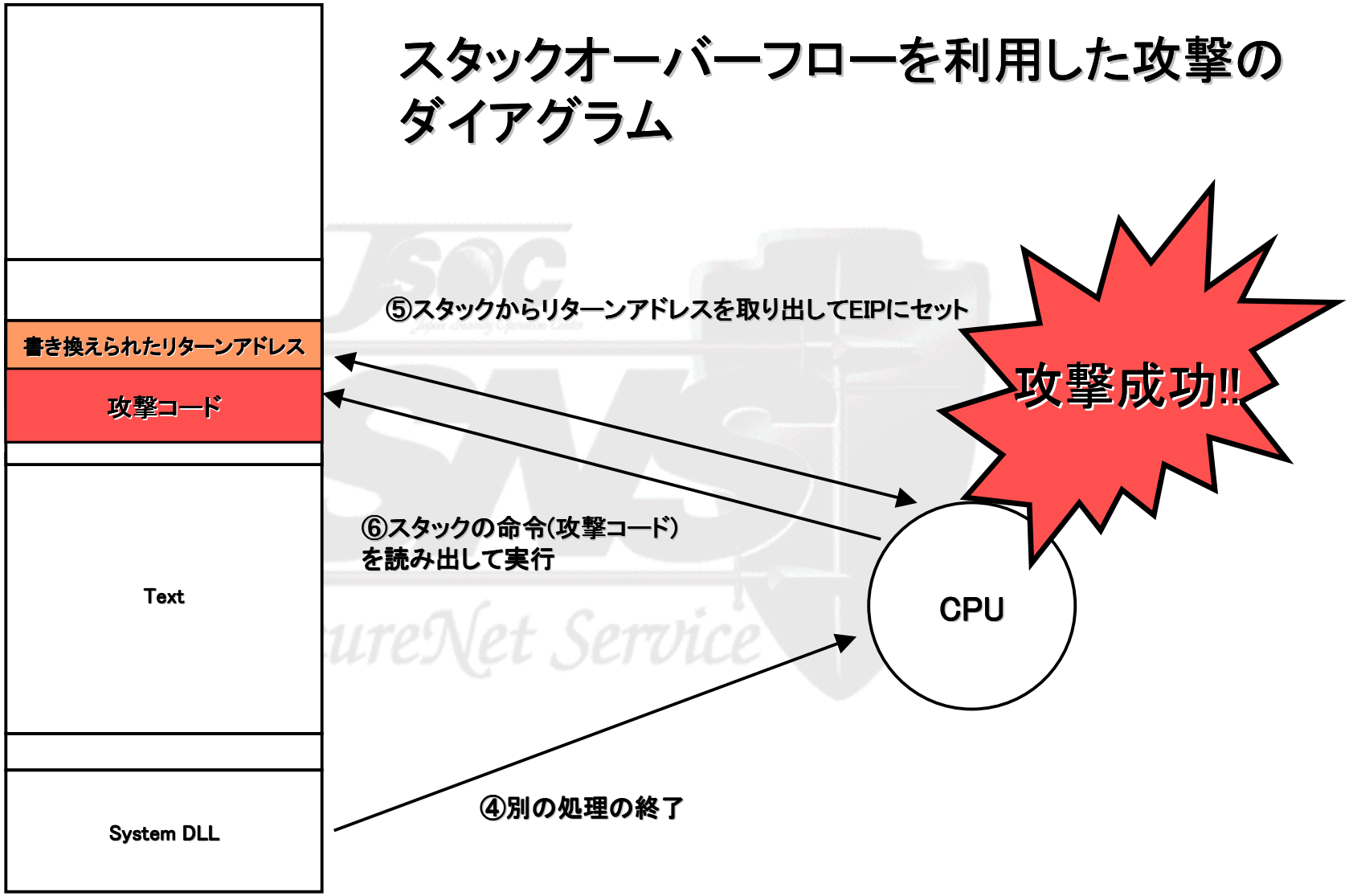
バッファオーバーフローとは何か？



スタックオーバーフローを利用した攻撃のダイアグラム

バッファオーバーフローとは何か？

スタックオーバーフローを利用した攻撃のダイアグラム



バッファオーバーフローとは何か？

・DEMO!!!

ヒープオーバーフローの特徴

- 2つ以上のレジスタの値を操作可能
- メモリ内の任意のアドレス内にある4バイト(32bit)の値を上書き可能
- strcpy(), gets(), spfintf(), memcpy(), …
- 攻撃に利用することはスタックオーバーフローよりは難しい
- 現在の主流

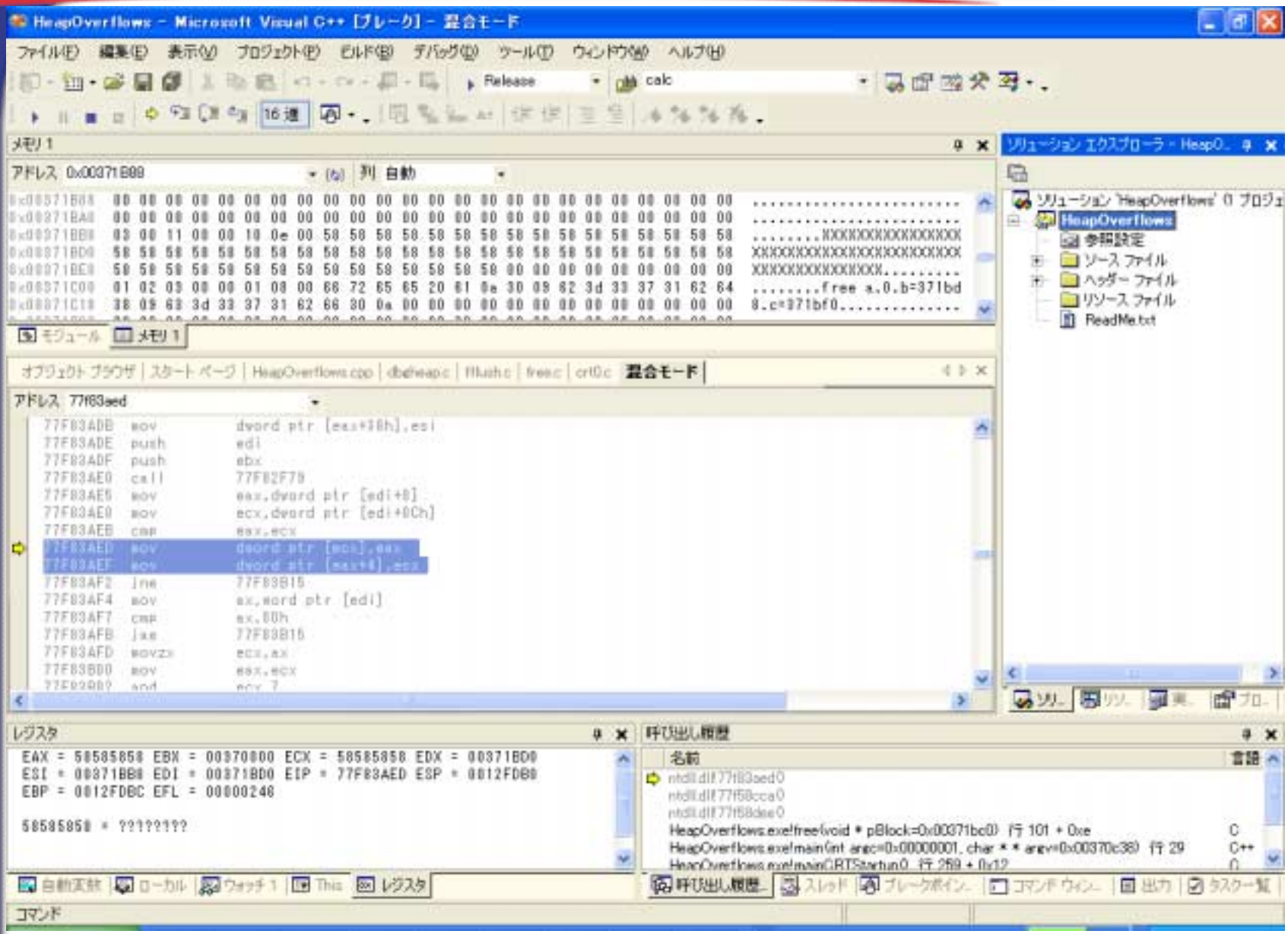
バッファオーバーフローとは何か？

ヒープオーバーフローはなぜ攻撃に利用できるのか？

- ヒープ領域が上書きされるだけでは攻撃に利用できない
  - free()関数による動的なメモリの開放が行われる必要性がある
- free()関数による開放が行われた際に、メモリ内のデータ操作が行われることを利用

上書きしたデータを使った、このデータ操作が行われた際に、任意のアドレスの書き換えが発生する

バッファオーバーフローとは何か？



バッファオーバーフローとは何か？

```
#include <stdio.h>
int main(int argc,char *argv[])
{
        char *smallbuf,*a,*b,*c;

        a = malloc(10);
        b = malloc(10);
        c = malloc(10);

        printf("a=%x¥tb=%x¥tc=%x¥n",a,b,c);

        memset(a,0x0,10);
        printf("copy¥n");
        strcpy(a,"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX");
        ↑ここでオーバーフロー自体が発生

        printf("free a¥n");
        free(a);
        ↑ここでデータ操作が発生し、任意のアドレスの上書きが可能になる

        printf("free b¥n");
        free(b);
        printf("free c¥n");
        free(c);

        printf("done¥n");
}
```

バッファオーバーフローとは何か？

- 上書きする値は？
  - トップレベル例外ハンドラ (Unhandled Exception Filter)
  - Process Environment Block (PEB)内の fast lockingポインタ (RtlEnterCriticalSection())
  - etc..

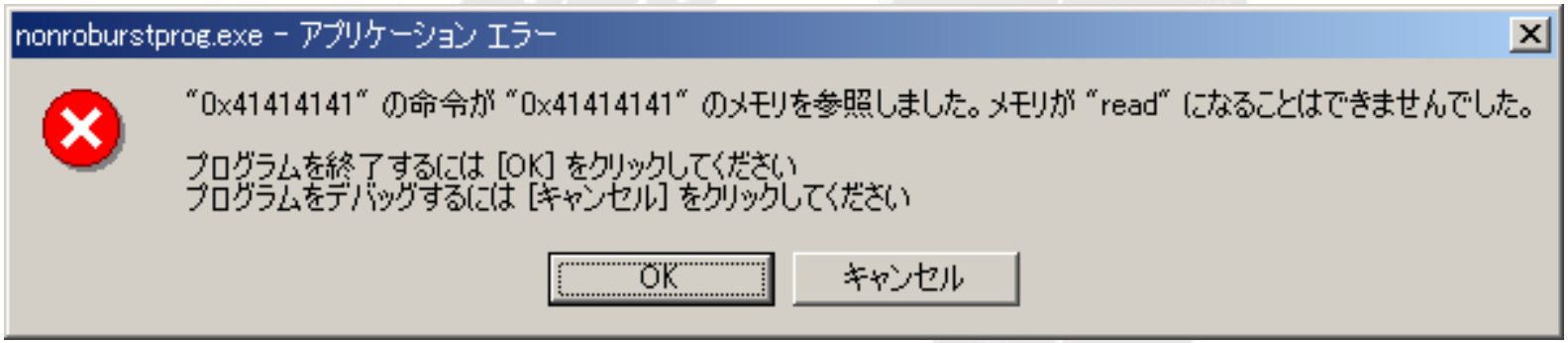
トップレベル例外ハンドラ(1)

- ハンドルできない例外が発生した場合に呼び出される関数

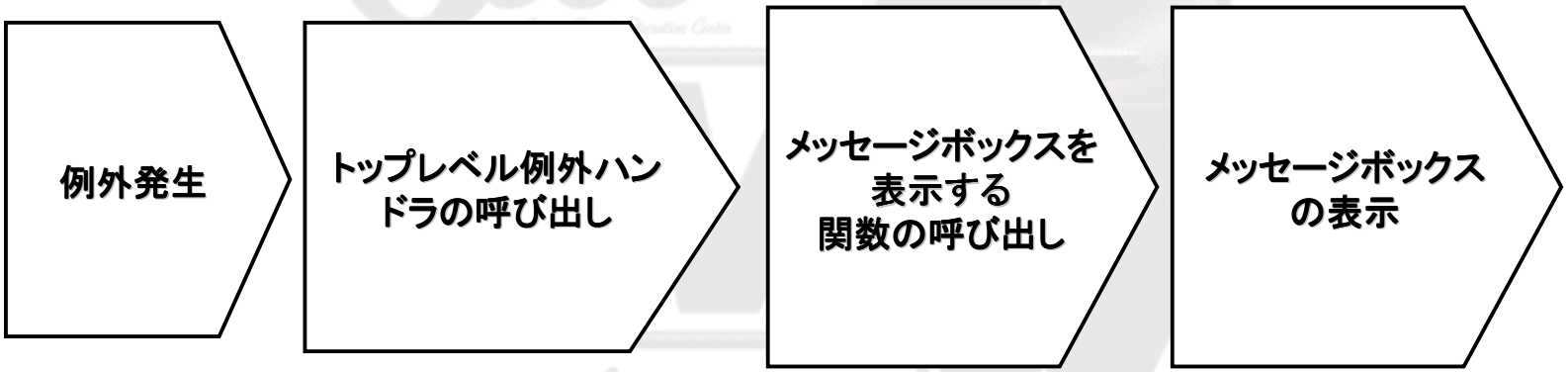
- ハンドルできない例外とは
  - ゼロによる除算
  - 不正なメモリの参照
  - etc…

バッファオーバーフローとは何か？

トップレベル例外ハンドラ(2)

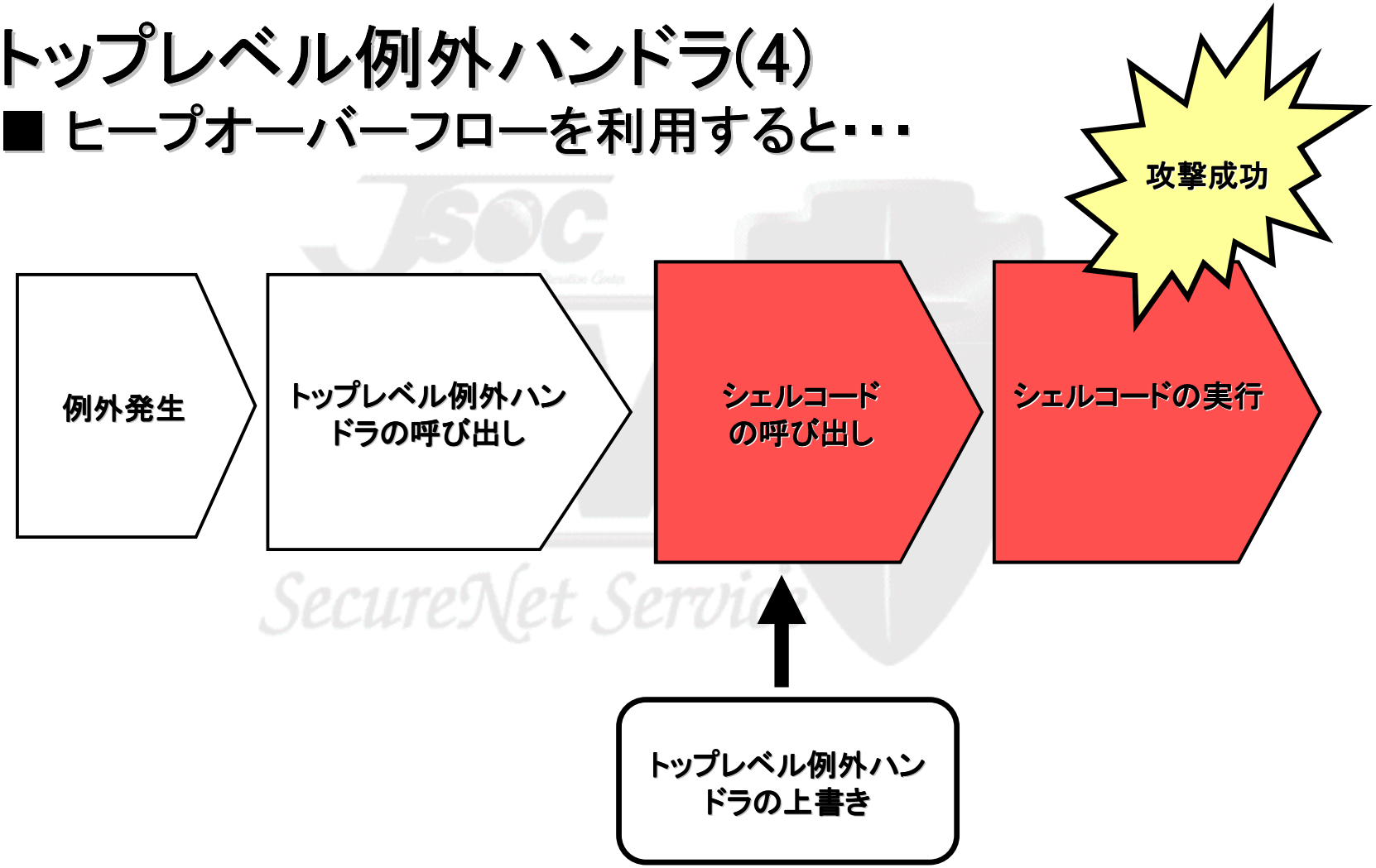


トップレベル例外ハンドラ(3)



トップレベル例外ハンドラ(4)

■ ヒープオーバーフローを利用すると・・・



PEB内のfirst lockingポインタ(1)

- プロセス毎に保持される構造体
(Process Environment Block:PEB)

- PEBの中に含まれるもの
  - ローダの情報
  - プロセスの各種パラメータ
  - First lockingポインタ

PEB内のfirst lockingポインタ(2)

- RtlEnterCriticalSection() へのポインタを保持
- いくつものWin32 APIが排他的処理を要求するため、参照されることがある

バッファオーバーフローとは何か？

- DEMO!!!

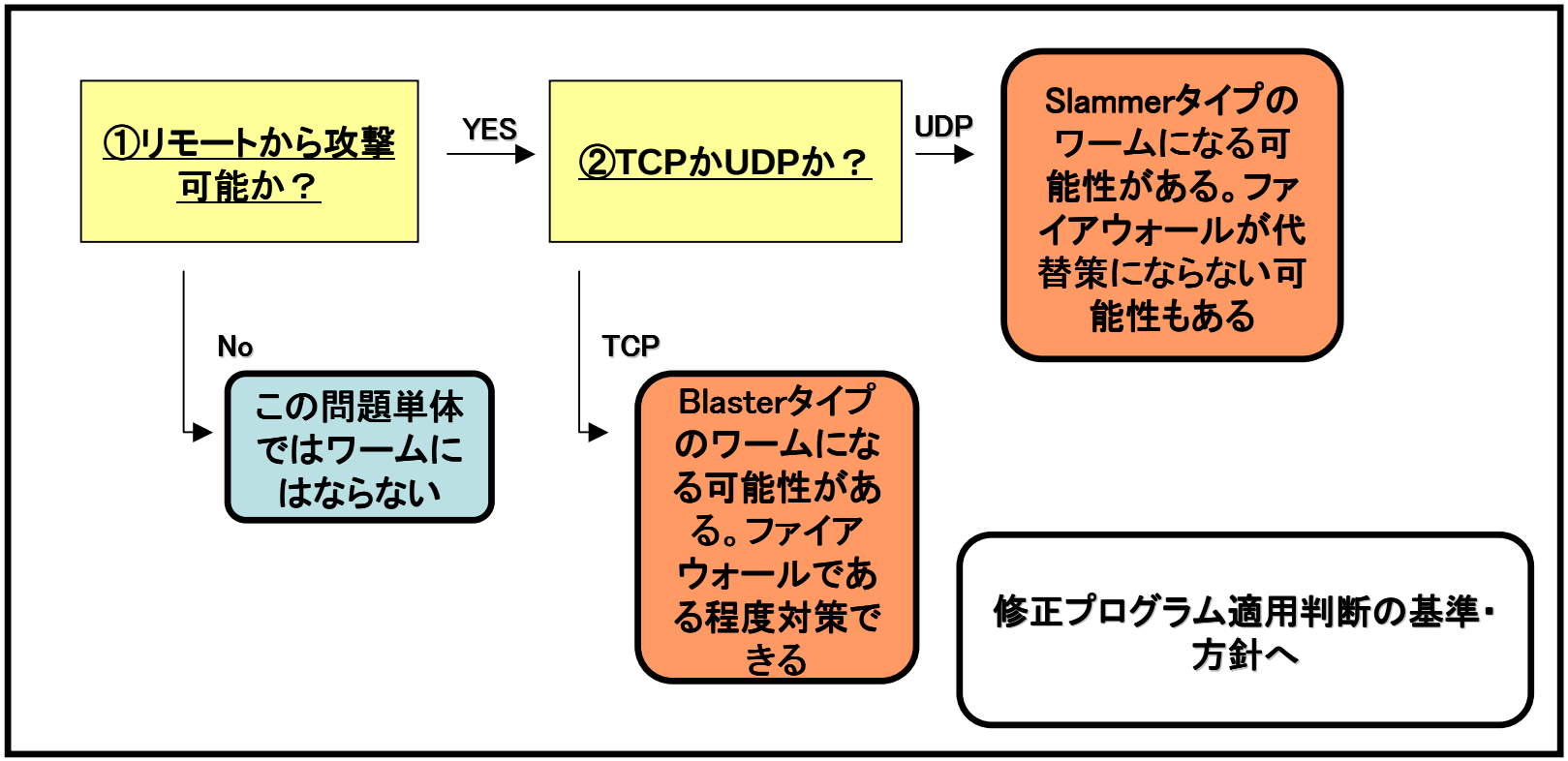
ワームとバッファオーバーフロー

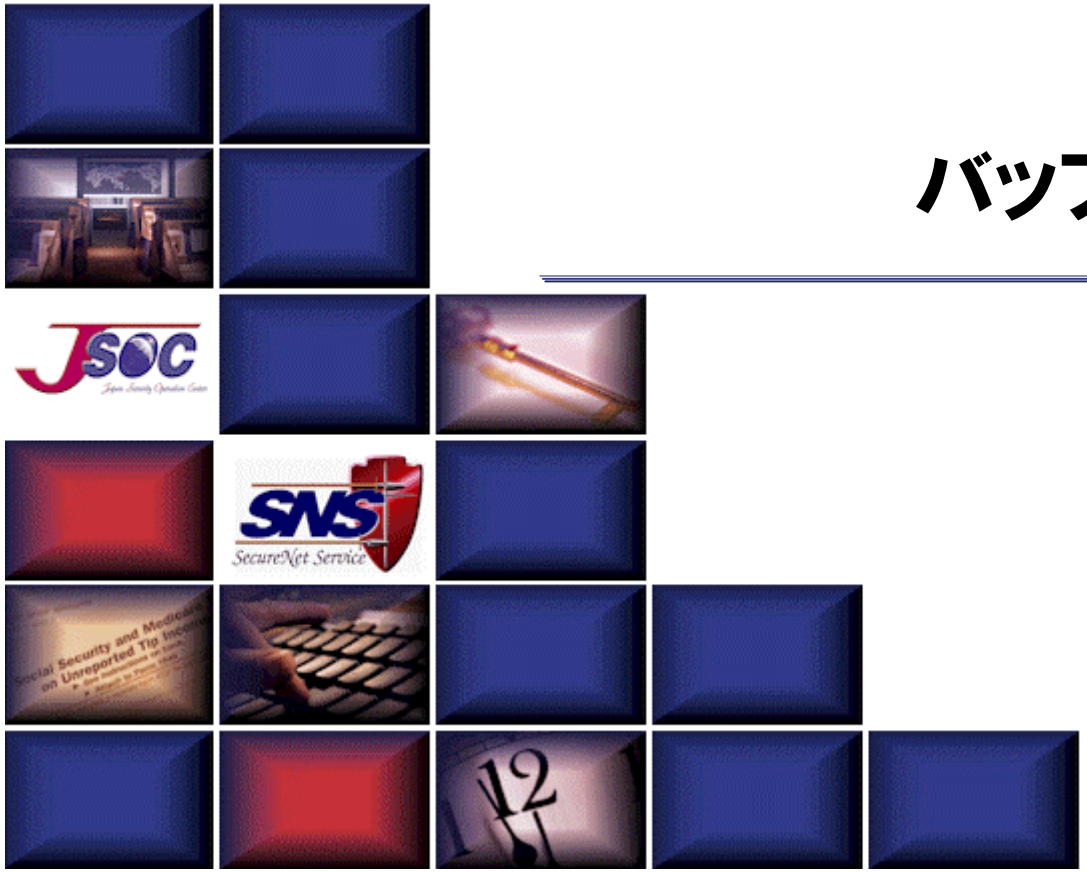
ワームとバッファオーバーフロー

- ワーム化を抑制または促進させる要素
  - リモート
  - 攻撃を成功させるために必要な権限
  - 攻撃のベクトル(TCPとUDP)
  - ペイロードの大きさ(ワームの自由度)
  - Windowsのバージョン・言語・サービスパックの差異を超えさせる要件(ワームの安定性)

ワームとバッファオーバーフロー

対処例(簡易版):
あるバッファオーバーフローの脆弱性が公開されたとき、それがワーム化するのかどうかについての判断要素

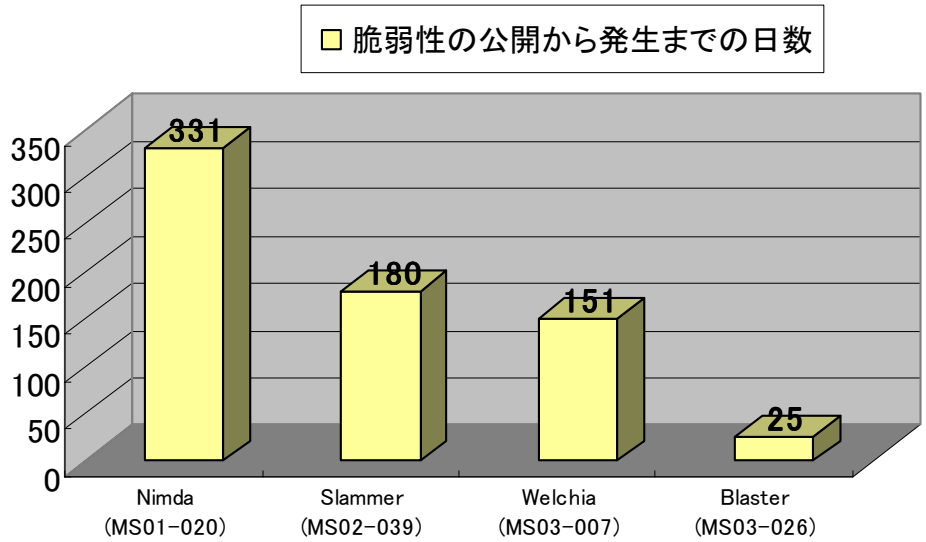




プロアクティブな
バッファオーバーフロー対策

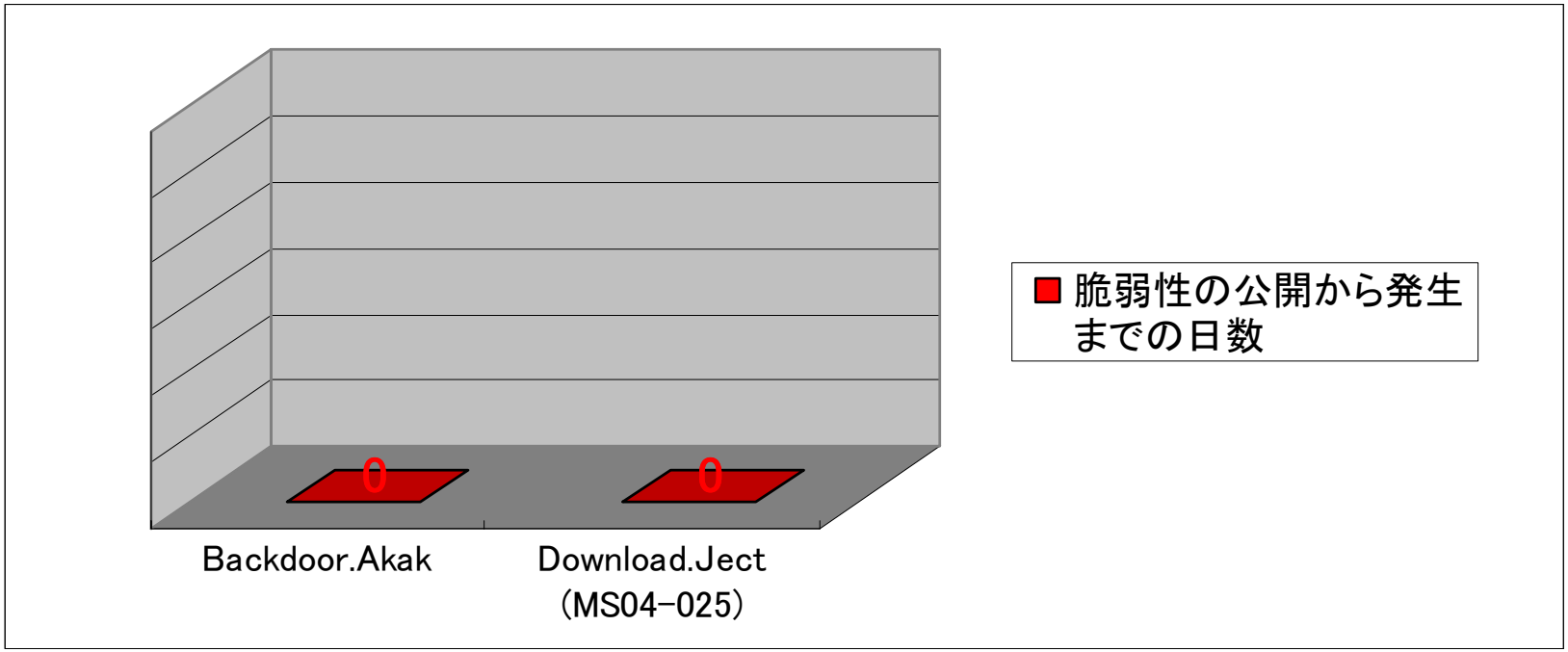
プロアクティブなバッファオーバーフロー対策

過去の脆弱性脅威モデル



プロアクティブなバッファオーバーフロー対策

現在の脆弱性脅威モデル



プロアクティブなバッファオーバーフロー対策

プロアクティブな対策をOSレベルに組み込む必要性

■ 修正プログラムよりもワームやウイルスが先にリリースされる傾向にある
- 先立つ情報はなにもないので対策は難しい
- 情報収集をするにもコストがかかる

■ 脆弱性の多くはバッファオーバーフローである
- バッファオーバーフローはワームになる可能性がある
- 攻撃プログラムも開発されやすい

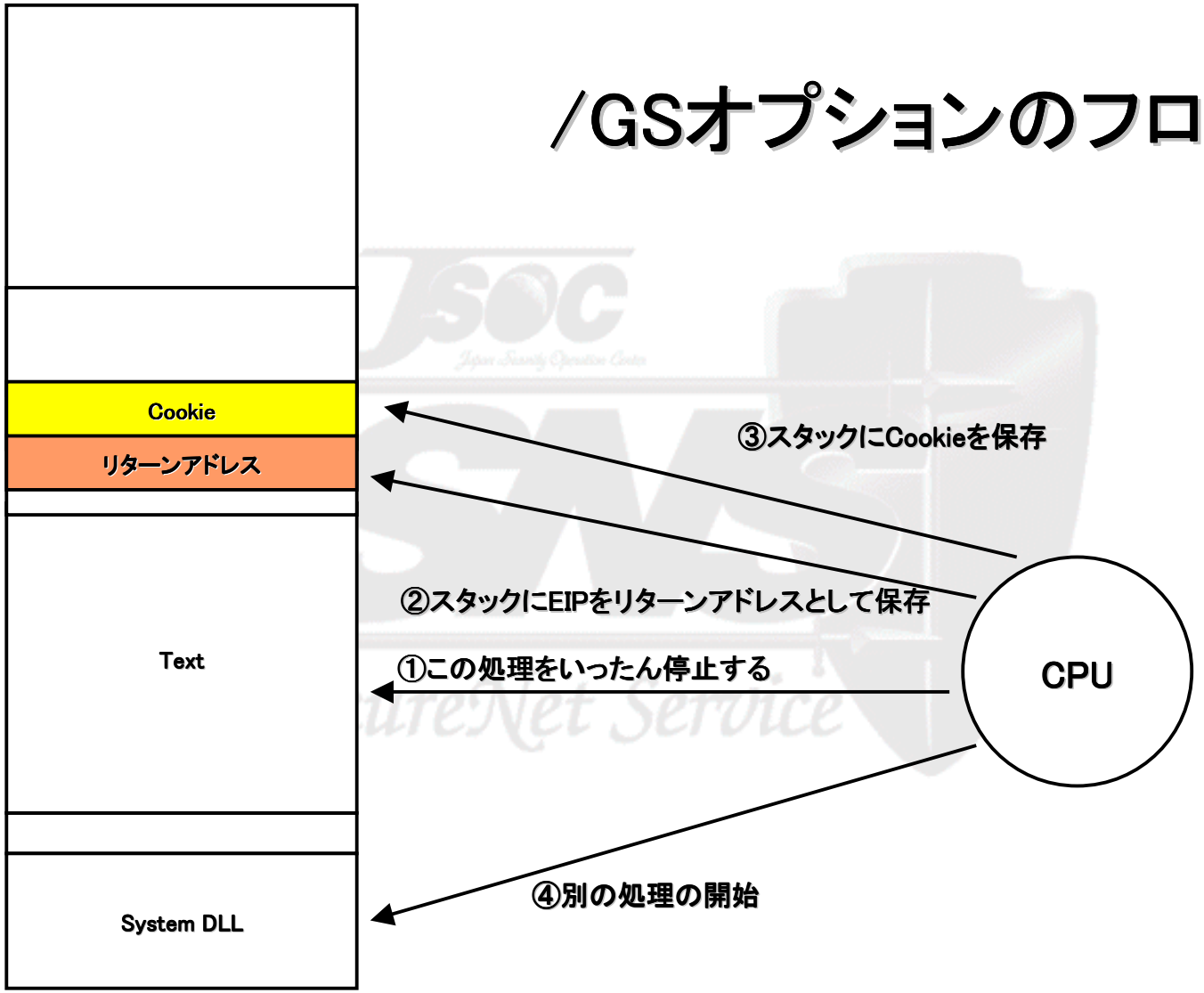
プロアクティブなバッファオーバーフロー対策

■ /GSオプションとは

- Visual C++ .NET以降で加わったランライム保護手段
- スタックに格納されるリターンアドレスの直前にCookieと呼ばれる値を設ける
- Cookieの上書きを検知して、プログラムを終了させる
- スタックオーバーフローにのみ対応

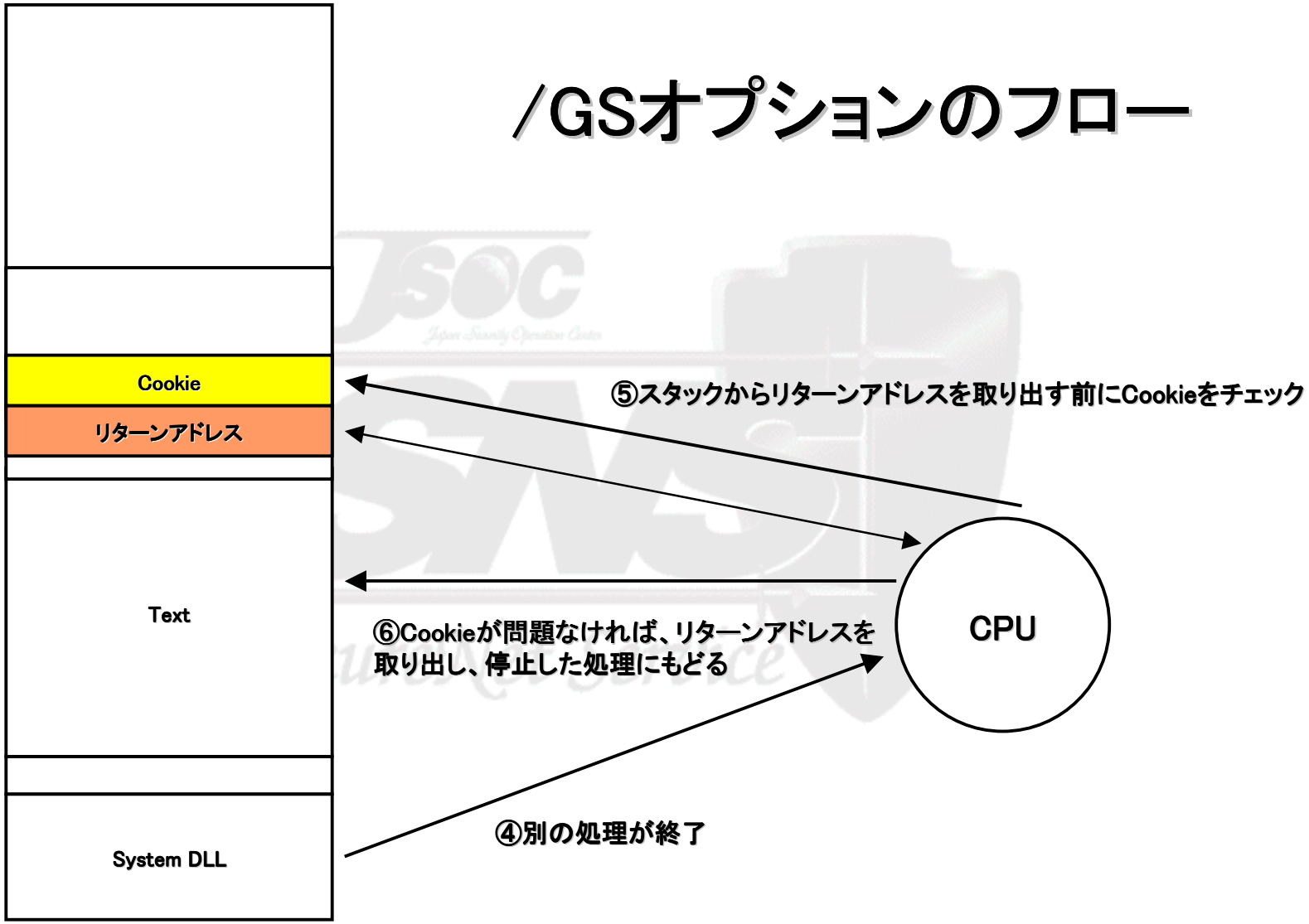
プロアクティブなバッファオーバーフロー対策

/GSオプションのフロー



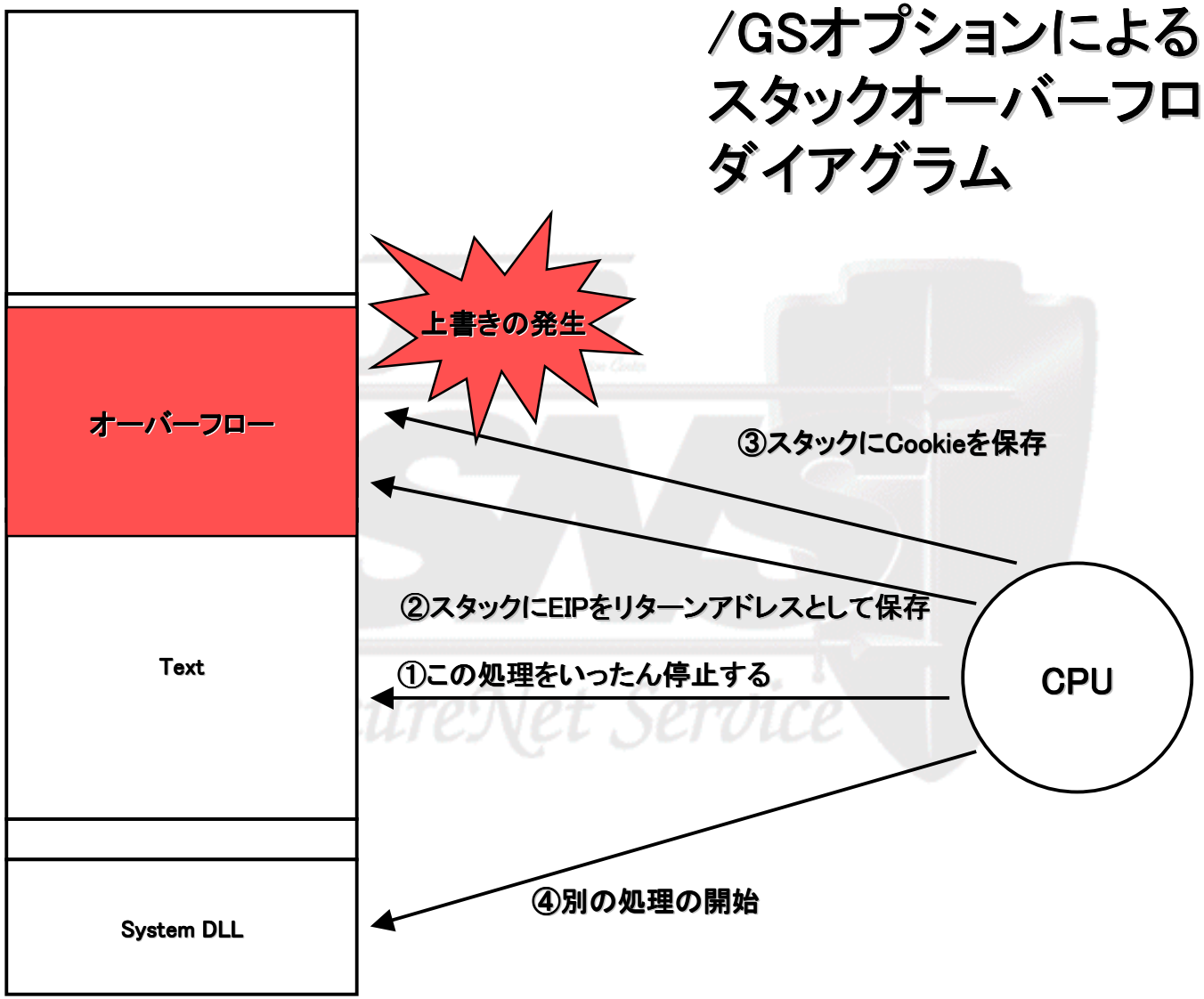
プロアクティブなバッファオーバーフロー対策

/GSオプションのフロー



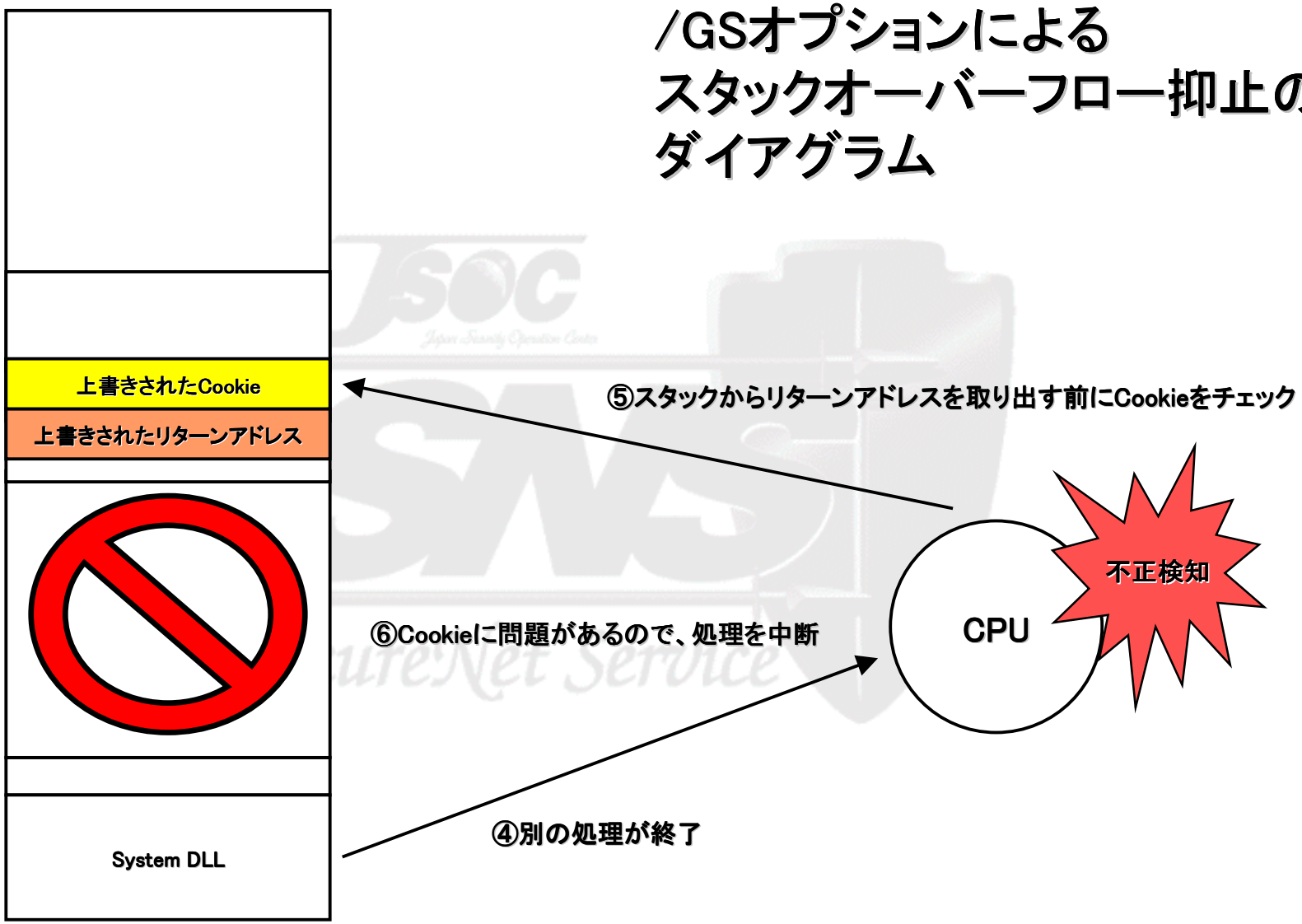
プロアクティブなバッファオーバーフロー対策

/GSオプションによる
スタックオーバーフロー抑止の
ダイアグラム



プロアクティブなバッファオーバーフロー対策

/GSオプションによるスタックオーバーフロー抑止のダイアグラム



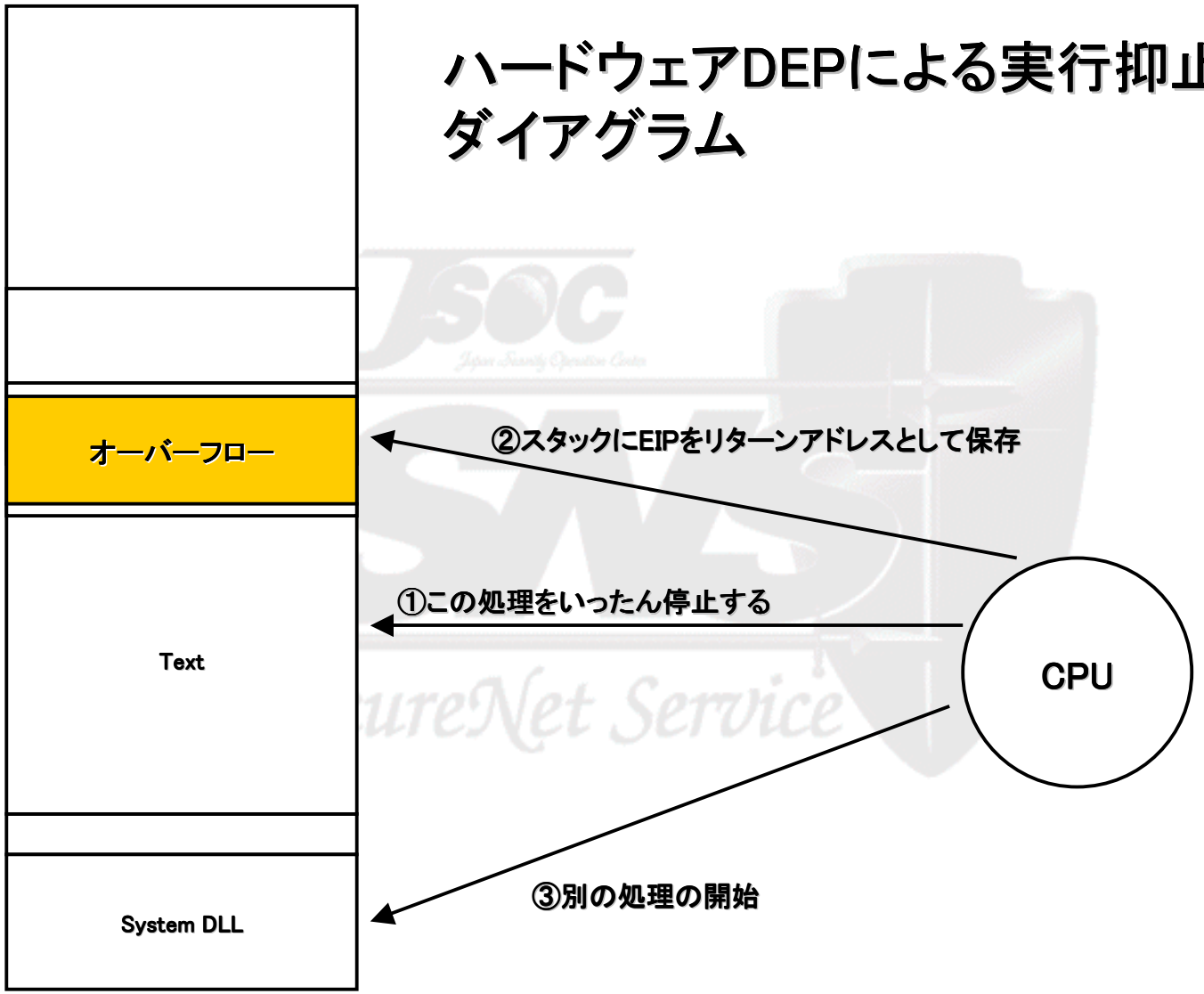
プロアクティブなバッファオーバーフロー対策

■ DEP(Data Execution Prevention)とは

- Windows XP SP2で加わったハードウェアおよびソフトウェアによる保護手段
- ハードウェアDEPでは、CPUに加わったセグメント管理機構を使用してスタックとヒープから実行権限を剥奪
- ソフトウェアDEPでは例外処理機構を利用してコードの実行を阻止

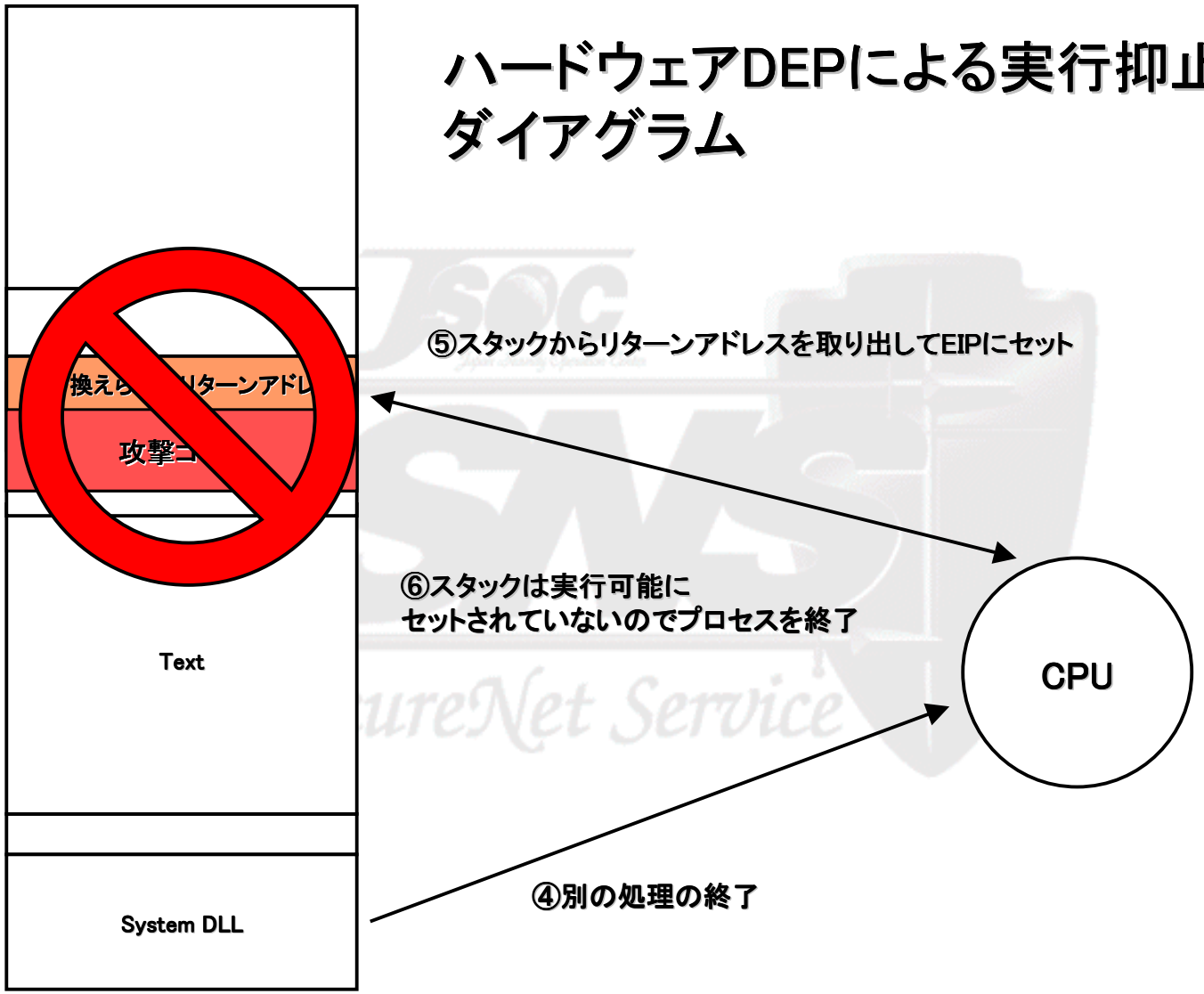
プロアクティブなバッファオーバーフロー対策

ハードウェアDEPによる実行抑止のダイアグラム



プロアクティブなバッファオーバーフロー対策

ハードウェアDEPによる実行抑止のダイアグラム



·まとめ

·バッファオーバーフローによるコード実行は
抑止されゆく方向にあり、いまはその
過渡期である

•これまでのレガシーなシステムを一斉に
リプレイスするのは不可能であるため、
今後しばらくはワームへの対策が必須