# Programação Segura

# Semana 8: Tempo e Estado, Tratamento de Exceções

Profª Denise Goya
Denise.goya@ufabc.edu.br – UFABC - CMCC

# Classes de Erros de Segurança

- Validação e representação dos dados de entrada
- Abuso de API
- Características de segurança
- Tempo e estado
- Tratamento de exceções
- Qualidade do código
- Encapsulamento
- Ambiente

# Tempo e Estado

- A classe de problemas relacionada a **Tempo e Estado** ocorre em ambientes que suportam **multiprogramação**, em que dois ou mais processos compartilham um estado e dependem de um escalonamento de processos
- Mais comum em sistemas distribuídos ou com *multithreading*

# Tempo e Estado

- Processos não são executados de forma atômica
- Um processo pode interferir em outro, entre a execução de duas instruções, por exemplo

# Tempo e Estado

- **Race Condition** é a expressão mais comumente usada para indicar um comportamento não desejável em sistemas eletrônicos ou de software em que a saída depende da sequência ou do tempo de outros eventos sobre os quais não se tem controle
  - Race Condition = Condição de Concorrência (ou condição de corrida)

# Race Condition

- Exemplos de recursos que podem ter acesso concorrido ou compartilhado:
  - Sistemas de arquivo
    - Acesso simultâneo ao mesmo arquivo por programas diferentes: pode provocar corrupção de dados se mal gerenciado
  - Memória (RAM ou disco) ou ciclos do processador
    - Controle inadequado do compartilhamento leva a estados não desejáveis

# Tipos de Race Condition

1. Interferência causada por processos não-confiáveis
   - Tipo também conhecido por problemas de sequenciamento, condição não-atômica ou TOCTOU: *time-of-check, time-of-use*
2. Interferência causada por processos confiáveis
   - Tipo também conhecido por problemas de *deadlock* ou por condição de falha de travas

# Race Condition: Exemplo Tipo 1

- **Reuso de um buffer** para armazenar um texto em claro num dado instante e o texto cifrado, logo em seguida:
  - Imagine uma aplicação com o seguinte fluxo:
    1. Carregar um buffer com um texto em claro (*plaintext*)
    2. Cifrar o buffer (isto é, o buffer passa a conter um texto cifrado, *ciphertext*)
    3. Enviar o buffer para um destinatário

# Race Condition: Exemplo Tipo 1 (cont)

- Agora, imagine que a aplicação é multithread e, por uma Race condition de sequenciamento, os passos 2 e 3 são executados em ordem trocada:
  1. Carregar um buffer com um texto em claro (*plaintext*)
  2. Enviar o buffer para um destinatário
  3. Cifrar o buffer (isto é, o buffer passa a conter um texto cifrado, *ciphertext*)
  - O destinatário recebe o texto em claro
  - Caso real com o Microsoft IIS 4

# Race Condition: Exemplo Tipo 1

- **Acesso a Arquivo** com vulnerabilidade TOCTOU:
  - Em sistemas de arquivo, uma vulnerabilidade de *time-of-check, time-of-use* é possível pois um mesmo nome de arquivo pode referenciar arquivos distintos em diferentes momentos.

# Race Condition: Exemplo Tipo 1

- Ataques relacionados a TOCTOU em **Acesso a Arquivo** costuma seguir a seguinte sequência:
  1. Um programa verifica uma propriedade de um arquivo, usando um nome (string filename) em vez de um tratador adequado para o objeto
  2. Um atacante altera o nome (filename)
  3. O programa realiza a operação sobre o arquivo usando o mesmo filename, pressupondo que já está verificado

# Race Condition: Ex. Acesso a Arquivo

- As funções *access()* e *open()* aceitam como parâmetro uma string com o nome do arquivo, em vez de um file handle.
- Entre uma chamada *access()* e *open(),* o atacante tem uma janela de oportunidade para alterar o nome do arquivo
- Existem técnicas para um atacante aumentar o tempo dessa janela, por exemplo, enviando um sinal para o processo atacado para dar a preferência à CPU (e atrasando o processo)

# Race Condition: Ex. Acesso a Arquivo

- Serviço de impressão no Red Hat 6:
  - lpr é um utilitário instalado como setuid root, para possibilitar comunicação com impressora

**Example 12.8** File access race condition vulnerability in code from lpr in Red Hat 6.

```
for (int i=1; i < argc; i++) {
  /* make sure that the user can read the file, then open it */
  if (!access(argv[i], O_RDONLY)) {
    fd = open(argv[i], O_RDONLY);
  }
  print(fd);
}
```

# Race Condition: Ex. Acesso a Arquivo

- Um atacante aciona **lpr** com parametro /tmp/atack e o redireciona para o arquivo de senhas /etc/shadow durante a janela de vulnerabilidade

| | lpr | Attacker |
|---|---|---|
| Time 0: | `access("/tmp/attack")` | |
| Time 1: | | `unlink("/tmp/attack")` |
| Time 2: | | `symlink("/etc/shadow","/tmp/attack")` |
| Time 3: | `open("/tmp/attack")` | |

**Figure 12.4** Possible sequence of instructions for an attack against the TOCTOU vulnerability in lpr from Example 12.8.

# Race Condition: Correção no lpr

**Example 12.9** Code from Example 12.8 rewritten to drop privileges instead of relying on access().

```
for (int i=1; i < argc; i++) {
  int caller_uid = getuid();
  int owner_uid = geteuid();

  /* set effective user id before opening the file */
  if (setresuid(-1, caller_uid, owner_uid) != 0){
    exit(-1);
  }

  if (fd = open(argv[i], O_RDONLY);

  /* reset the effective user id to its original value */
  if (setresuid(-1, owner_uid, caller_uid) != 0){
    exit(-1);
  }
  if (fd != -1) {
    print(fd);
  }
}
```

Essa correção vale para qualquer programa.

É importante evitar o *access()* como forma de teste de permissões

# Race Condition: Ex. Acesso a Arquivo

- De modo geral, é importante evitar o uso de nomes de arquivos em strings: use descritores

**Table 12.3** Common filesystem calls that accept paths and their file descriptor–based equivalents.

| Operates on Path | Operates on File Descriptor |
|---|---|
| `int chmod (const char *filename, mode_t mode)` | `int fchmod (int filedes, int mode)` |
| `int chown (const char *filename, uid_t owner, gid_t group)` | `int fchown (int filedes, int owner, int group)` |
| `int chdir (const char *filename)` | `int fchdir (int filedes)` |
| `int stat (const char *filename, struct stat *buf)` | `int fstat (int filedes, struct stat *buf)` |

# Race Condition: Ex. Acesso a Arquivo

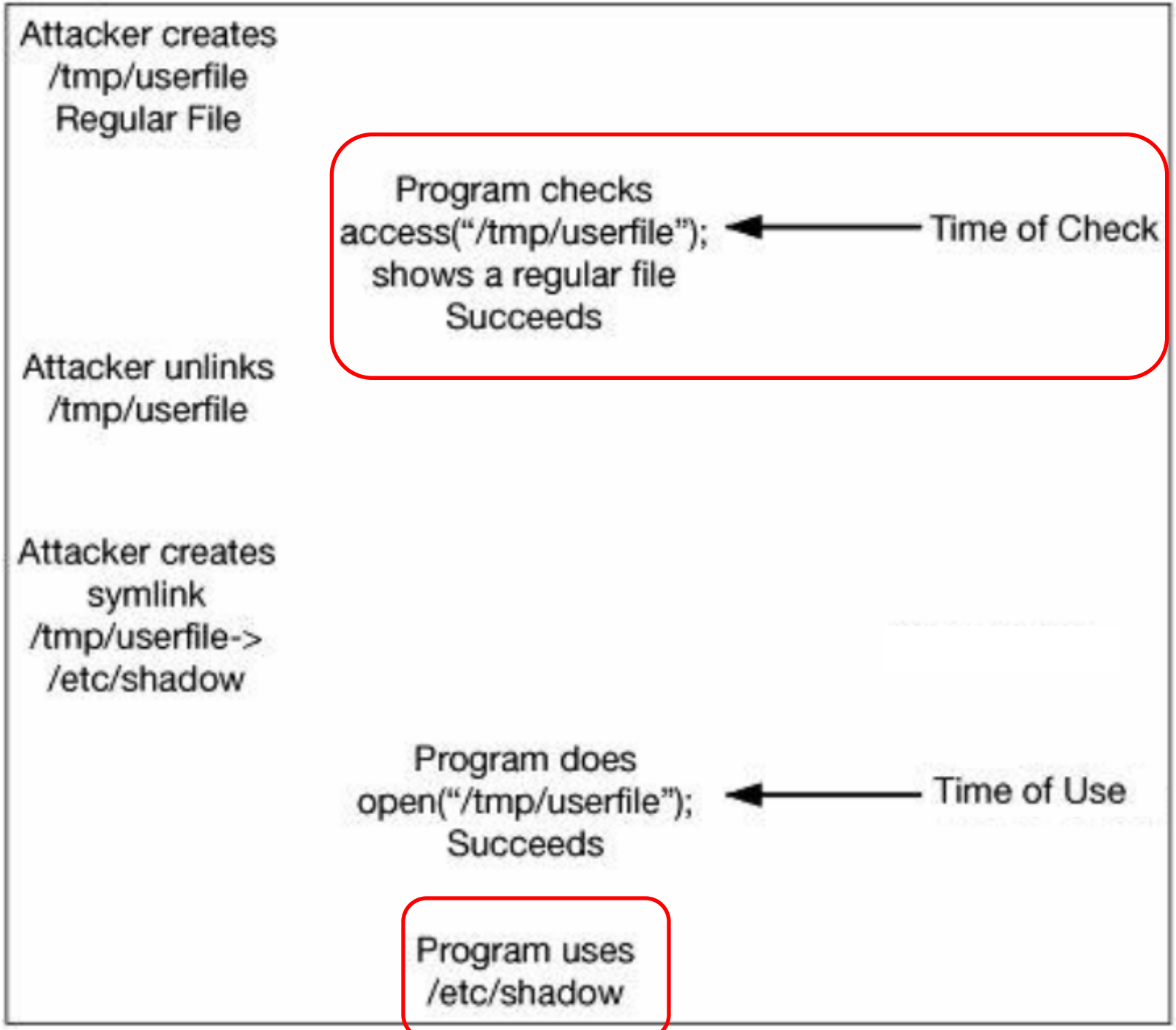
- TOCTOU usando stat ou lstat

**Listing 9-4. Race Condition from Kerberos 4 in lstat() and open()**

```
    errno = 0;
    if (lstat(file, &statb) < 0)
        goto out;
    if (!(statb.st_mode & S_IFREG)
#ifdef notdef
        || statb.st_mode & 077
#endif
        )
        goto out;
    if ((fd = open(file, O_RDWR|O_SYNC, 0)) < 0)
        goto out;
```

Verifica que é um arquivo regular e não um link simbólico:

Então abre o arquivo


Attacker creates
/tmp/userfile
Regular File
Program checks
access("/tmp/userfile");
shows a regular file
Succeeds
Time of Check
Attacker unlinks
/tmp/userfile
Attacker creates
symlink
/tmp/userfile->
/etc/shadow
Program does
open("/tmp/userfile");
Succeeds
Time of Use
Program uses
/etc/shadow

# Race Condition: Ex. Acesso a Arquivo

- De modo geral, é importante usar descritores

- Como em Java a classe java.io.File usa strings para arquivos (em vez de um file handle): não há como evitar este tipo de race condition
  - Portanto, não use Java para escrever programas privilegiados

# Race Condition: Exemplo Tipo 2

- **Deadlock**: situação de impasse em que, p.ex.
  - Processo 1 acessa e bloqueia recurso A e espera pelo recurso B, ao mesmo tempo que
  - Processo 2 acessa e bloqueia recurso B e espera pelo recurso A
- Em Unix, de modo geral, é comum se usar a criação de arquivos para indicar tais bloqueios
  - Em situação como essa, é preciso prevenir deadlocks

# Race Condition: Ex. Acesso a Arquivo

- Há outras situações em que podem ocorrer Race Conditions envolvendo acesso a arquivos

- Para saber mais, consulte:
  - CHESS, WEST. "**Secure Programming with Static Analysis**". Addison-Wesley Professional, 2007, Capítulo 12.
  - Dowd, McDonald, Schuh. "**The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities**". Pearson, 2006, Capítulo 9.

# ARQUIVOS TEMPORÁRIOS E TRAVESSIA DE DIRETÓRIOS

# Mais Problemas sobre Arquivos

- Arquivos temporários
  - Muitos aplicativos precisam trabalhar com arquivos temporários, mas há vulnererabilidades relacionadas
- Travessia de diretórios
  - O acesso a arquivos depende do caminho completo de diretórios: a travessia é um problema que deve ser evitado

# Arquivos Temporários

- A biblioteca C padrão oferece várias formas de se criar arquivos temporários:
  - Existem funções que criam nomes únicos
    - Suscetíveis a ataques quando o nome do arquivo temporário é previsível
  - Existem funções que abrem descritores de arquivos
    - Se o nome for previsível, atacante pode realizar escalada de privilégio

**Table 12.4** Common functions that attempt to generate a unique temporary filename.

| Function | Description |
|---|---|
| `char* mktemp (char *template)` | The `mktemp()` function generates a unique filename by modifying template `[...]`. If successful, it returns the template as modified. If `mktemp()` cannot find a unique filename, it makes template an empty string and returns that. |
| `char* tmpnam (char *result)` | This function constructs and returns a valid filename that does not refer to any existing file. If the `result` argument is a null pointer, the return value is a pointer to an internal static string, which might be modified by subsequent calls and therefore makes this function non-reentrant. Otherwise, the `result` argument should be a pointer to an array of at least `L_tmpnam` characters, and the result is written into that array. |
| `char* tempnam (const char *dir, const char *prefix)` | This function generates a unique temporary filename. If `prefix` is not a null pointer, up to five characters of this string are used as a prefix for the filename. The return value is a string newly allocated with `malloc()`, so you should release its storage with free when it is no longer needed. |

Em Windows, a função GetTempFileName() é similar e sofre da mesma vulnerabilidade

# Temporário com Nome Previsível

- Se o nome for previsível, um atacante pode criar um arquivo temporário e impedir que os programas executem suas tarefas
  - Pode, por exemplo, levar o programa a uma situação de lock
- Solução: gerar nomes de arquivos temporários aleatoriamente, mas os aleatórios devem ser imprevisíveis

# Temporário com Nome Previsível

- Funções que criam e abrem temporários únicos

**Table 12.5** Common functions that attempt to open a unique temporary file.

| Open Temporary Files | Description |
|---|---|
| `FILE* tmpfile (void)` | This function creates a temporary binary file for update mode, as if by calling `fopen()` with mode `wb+`. The file is deleted automatically when it is closed or when the program terminates. |
| `int mkstemp (char *template)` | The `mkstemp()` function generates a unique file name just as `mktemp()` does, but it also opens the file for you with `open()` [with the `O_EXCL` flag]. If successful, it modifies the template in place and returns a file descriptor for that file open for reading and writing. If `mkstemp()` cannot create a uniquely named file, it returns -1. The file is opened using mode 0600. |

# Criação e Abertura de Temporário

- As funções tmpfile e mkstemp são mais seguras, mas precisam ser usadas com cautela para um atacante não tentar escalar privilégio:
  - mkstemp() cria um temporário e o abre no modo 0600 (leitura e escrita apenas para quem criou)
    - Em sistemas mais antigos, é usado o modo 0666, de modo que o programador deve aplicar umask 077
  - mas o nome aleatório é relativamente fácil de prever: usar então diretório não públicos ou geradores de aleatórios mais robustos

# Travessia de Diretórios

- Imagine um **servidor Web** sob o Windows que permita que arquivos cgi sejam executados no diretório (“C:\inetpub\wwwroot\cgi-bin”)
- **Pergunta**: um arquivo com o nome

“C:\inetpub\wwwroot\cgi-bin\..\..\..\Windows\System32\cmd.exe”

- poderia ser executado? Sua tradução é: “C:\Windows\System32\cmd.exe”
- Ou seja, ao executar tal comando, o servidor abre um prompt de comando para o usuário

# Travessia de Diretórios

- Vulnerabilidades de travessia de diretórios são bastante comuns
- Podem permitir a escrita, leitura ou execução remota de um arquivo
- Podem violar a confidencialidade, integridade e disponibilidade da informação
- Está relacionada a Validação de entrada, em que aplicações precisam que o usuário informe arquivos a serem abertos

# Travessia de Diretórios: Exemplo

**Listing 8-15. Directory Traversal Vulnerability**

```
use CGI;
...
$username = $query->param('user');
open(FH, "</users/profiles/$username") || die("$!");
print "<B>User Details For: $username</B><BR><BR>";
while(<FH>){
    print;
    print "<BR>"
}
close(FH);
```

Como o nome de usuário não é devidamente sanitizado, um atacante pode digitar "../../../../../etc/passwd" e ter o arquivo de senhas exibido

# TRATAMENTO DE EXCEÇÕES

# Tratamento de Exceções

- Além de problemas com exceções não capturadas (vistos em aula anterior), há outros relacionados ao mal tratamento de erros no programa
- Citaremos:
  - mais algumas boas práticas e
  - um caso que foi decorrente da estrutura empregada no Windows para tratamento de exceções

# Tratamento de Exceções

- Declaração de uma captura genérica (catch) ou de um lançamento (throws) muito genérico:
  - faz com que o código de tratamento da exceção fique cada vez mais complexo, para tratar muitos casos
  - códigos complexos podem criar vulnerabilidades de segurança: evite-os

# Captura Genérica de Exceções

- Suponha que sejam equivalentes:

*Example Language:* **Java** *(Good Code)*

```
try {
  doExchange();
}
catch (IOException e) {
  logger.error("doExchange failed", e);
}
catch (InvocationTargetException e) {

  logger.error("doExchange failed", e);
}
catch (SQLException e) {

  logger.error("doExchange failed", e);
}
```

*(Bad Code)*

```
try {
  doExchange();
}
catch (Exception e) {
  logger.error("doExchange failed", e);
}
```

- o código à esquerda é preferível, pois se doExchange precisar ser alterado posteriormente, é melhor que não misture os casos em um código complexo

# Throws Genérico de Exceções

- idem para:

*Example Language:* **Java** *(Good Code)*

```
public void doExchange() throws IOException, InvocationTargetException, SQLException {
  ...
}
```

*(Bad Code)*

```
public void doExchange() throws Exception {
  ...
}
```

# SEH (Structured Exception Handler)

- O SEH é uma estrutura de tratamento de exceção utilizada pelo Windows
  - Os blocos de códigos são encapsulados e cada bloco possui um ou mais tratadores associados
  - Quando uma exceção é disparada, um tratador adequado é procurado
  - O primeiro que satisfaça as condições da exceção é executado.

# SEH (Structured Exception Handler)

- Uma vulnerabilidade no SEH quando ocorre um **transbordamento de memória** de tal maneira que sobrescreve a Estrutura de tratamento de exceções:
  - Um transbordo que ocorra dentro de um bloco try/except pode provocar a **alteração de endereço** de um próximo tratador de exceção

# SEH (Structured Exception Handler)

- Um SEH possui dois ponteiros:
  - o apontador para o próximo registro SEH e
  - o ponteiro para o tratador de exceção

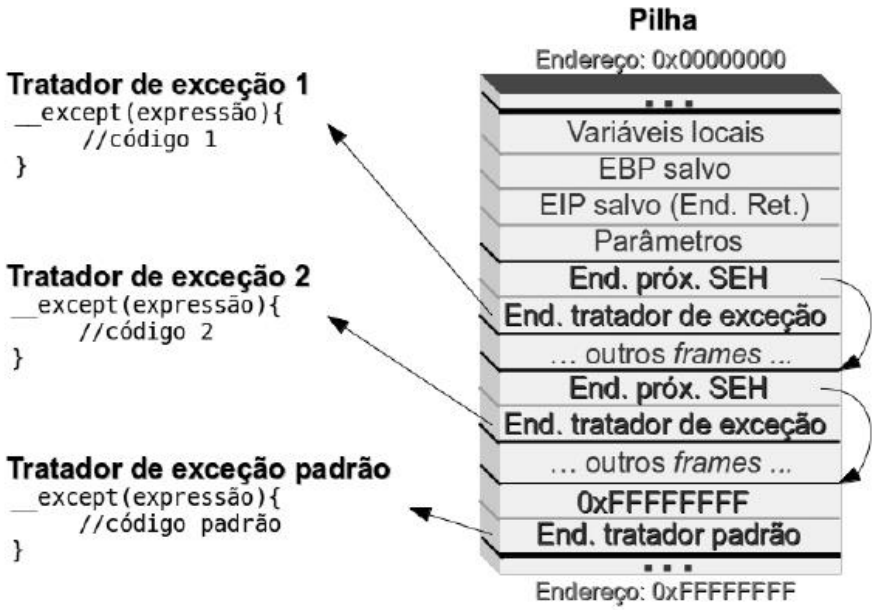



Figura 1.6. Exemplo de pilha com SEH. [Anwar, 2009]

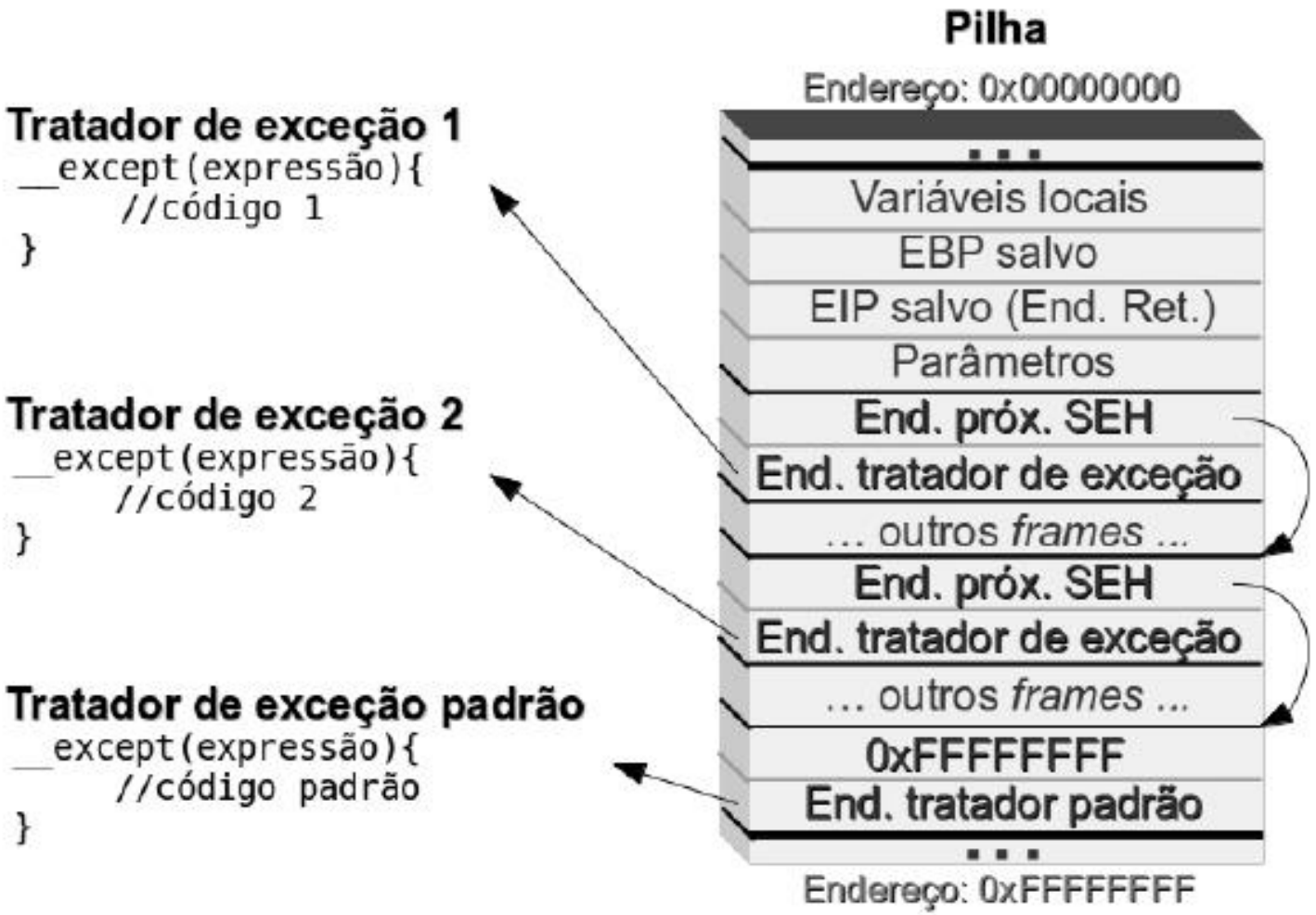



Figura 1.6. Exemplo de pilha com SEH. [Anwar, 2009]

# SEH (Structured Exception Handler)

- Cada tratador de exceção possui a chance de tratar a exceção disparada ou passá-la para o próximo tratador de exceção que se encontra na lista
- A partir do Windows XP SP1 foi feita uma melhoria: zerar todos registradores antes de uma chamada do SEH
  - Mas isso não foi suficiente

# SEH (Structured Exception Handler)

- No momento em que o tratador de exceção é chamado, o **endereço do SEH** fica armazenado duas palavras (8 bytes) abaixo do topo da pilha (ESP – stack pointer).
  - Um ataque que realize dois desempilhamentos (POP) e depois retorne (RET) desviará o fluxo de execução para a posição originalmente ocupada pelo ponteiro para o próximo SEH

# SEH (Structured Exception Handler)

- É possível escrever um exploit para sobrescrever o SEH e subverter o fluxo de execução. Em geral, o padrão é:
  1. Sobrescreve o ponteiro para o próximo tratador de exceção com alguma instrução de desvio (ex: JMP) que leve ao shellcode;
  2. Sobrescreve o ponteiro para o código do tratador de exceção de modo a voltar o fluxo de execução para a área sobrescrita pelo atacante na pilha (ex: POP POP RET);
  3. Gera uma exceção

# SEH (Structured Exception Handler)

- Exemplo de um exploit escrito dessa forma pode ser encontrado em
  - http://www.exploit-db.com/exploits/19625/
  - que foi publicado em 2012
- A Microsoft já vinha atuando em melhorias como
  - SEHOP (SEH Overwrite Protection), habilitado por default a partir do Windows Server 2008
  - SafeSEH: flag acionada durante a compilação

# SEH (Structured Exception Handler)

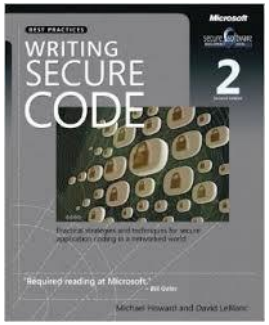
- mais informações em
  - Ferreira, Rocha, Martins, Feitosa e Souto. "**Análise de vulnerabilidades em Sistemas Computacionais Modernos: Conceitos, Exploits e Proteções**", 2012, disponível em http://dainf.ct.utfpr.edu.br/~maziero/lib/exe/fetch.php/ceseg:2012-sbseg-mc1.pdf

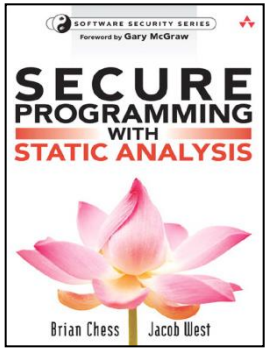
# ESTUDO INDIVIDUAL

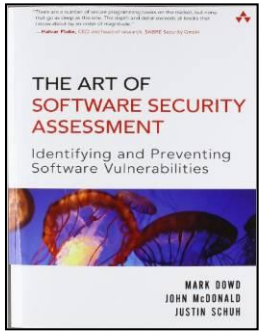
# Leitura Recomendada

Além dos links indicados ao longo dos slides:

- WEELER, D.: "**Secure Programming for Linux and Unix HOWTO – Creating Secure Software**". Ebook disponível em http://www.dwheeler.com/secure-programs/
- HOWARD, M.; LEBLANC, D.: "**Writing Secure Code**" Microsoft Press, 2a edição, 2002.
- CHESS, WEST. "**Secure Programming with Static Analysis**". Addison-Wesley Professional, 2007.
- Dowd, McDonald, Schuh. "**The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities**". Pearson, 2006.



# Exercício

1) Em Common Weakness Enumeration (CWE), é possível navegar em outros tipos de vulnerabilidades relacionados:

- Tempo e estado:

http://cwe.mitre.org/data/definitions/361.html

- Tratamento de erros:

http://cwe.mitre.org/data/definitions/388.html

Descreva algumas vulnerabilidades que lhe chamaram a atenção.