Analise de Software

Rodrigo Rubira Branco
rodrigo@firewalls.com.br

O que é a Firewalls?

- Empresa Especializada em Segurança;
- Profissionais Certificados;
- Atenta a Padrões Internacionais;
- Parceira das maiores empresas de Segurança do Mundo;
- Visão de Negócios e Ética Empresarial;
- Soluções Personalizadas para a Realidade das Empresas;
- Independente de Fornecedores e Tecnologias proporcionando a melhor solução para o cliente específico.

Objetivos da Apresentaçao

- Demonstrar normas de segurança no desenvolvimento de software

- Apresentar quesitos de segurança em software

- Expor os problemas mais comuns existentes no desenvolvimento de Software e como identifica-las

- Mostrar abordagens para se evitar falhas em software

Conceitos de Segurança

CIDAL =

C onfidencialidade

I ntegridade

D isponibilidade

A utenticidade

L egalidade

LEMBRAR-SE SEMPRE DISTO NO DECORRER DA APRESENTAÇAO!

Ambiente Existente

Trabalhamos em um mundo real de sistemas mal configurados:

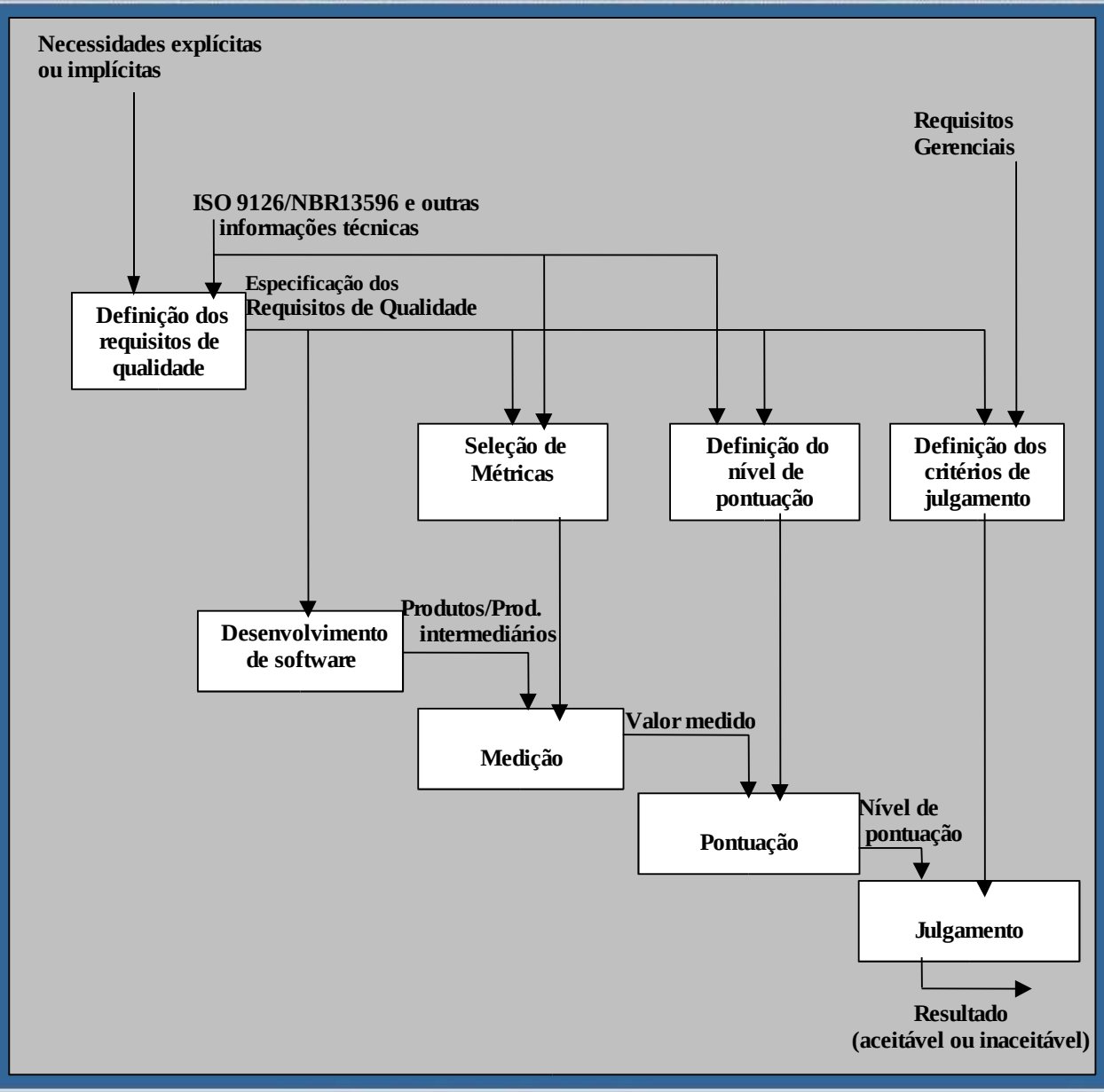
**** Bugs de Software***
**** Empregados Insatisfeitos***
**** Administradores de Sistemas Sobrecarregados***
**** Acomodação de necessidades empresariais***
*** Falta de educação em segurança**
**** B2B,B2C,B2E,C2C,X2X?***
*** Tempo disponivel para desenvolvimento**
*** Linguagens de 3 e 4 geraçoes**

Defense In-Depth

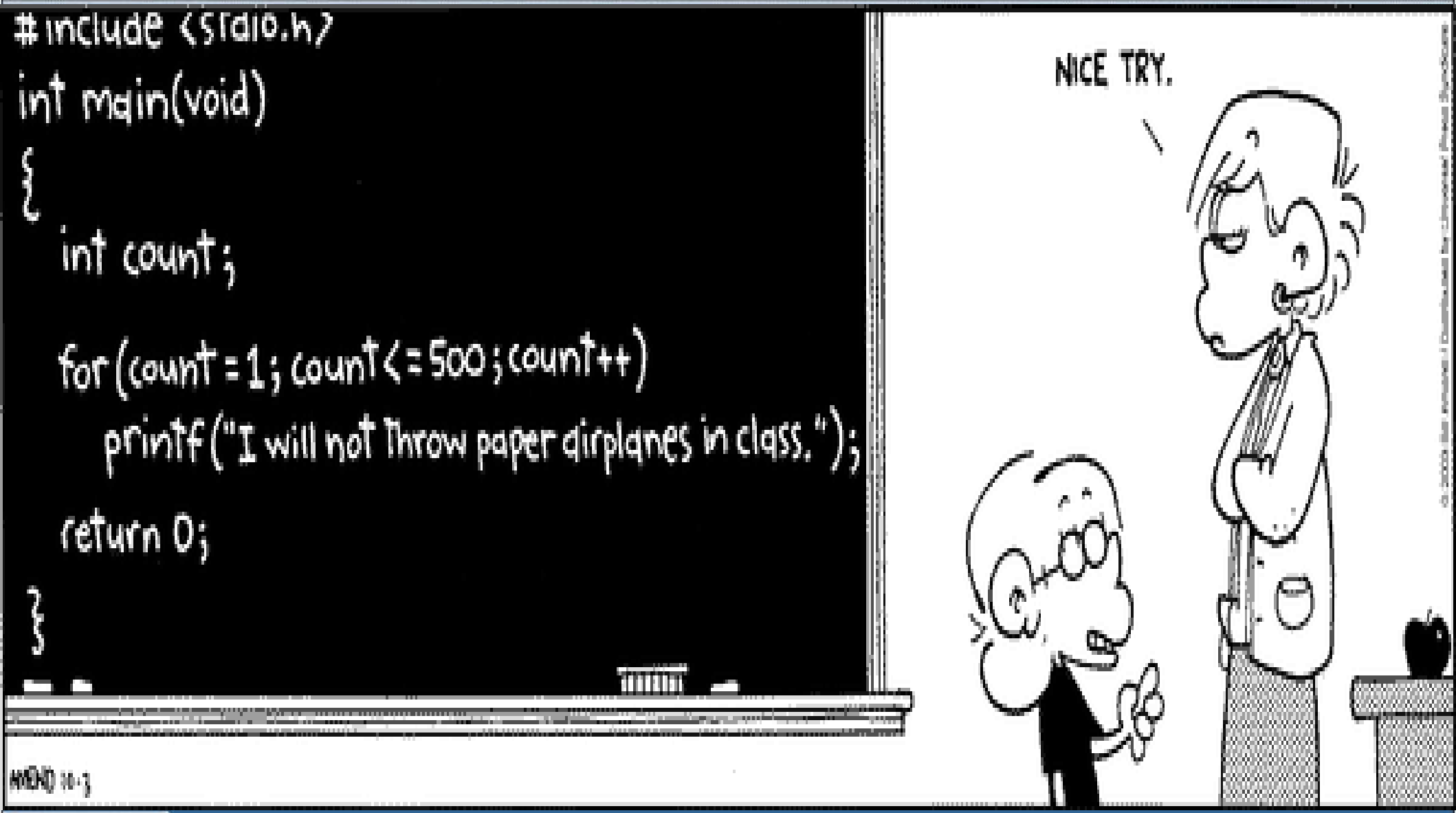
- Softwares sempre terao falhas
- Falhas inerentes ao sistema operacional, se nao previstas pelo software causam problemas (vide race condition do /tmp)
- Fortificaçoes possiveis no sistema operacional, poderiam tornar as aplicaçoes “livre de falhas” (vide Win2003, OpenBSD e patchs para o kernel do Linux)

Quando pensar em Segurança pense em uma CEBOLA.

Mundo IDEAL


Necessidades explícitas
ou implícitas
Requisitos
Gerenciais
ISO 9126/NBR13596 e outras
informações técnicas
Especificação dos
Requisitos de Qualidade
Definição dos
requisitos de
qualidade
Seleção de
Métricas
Definição do
nível de
pontuação
Definição dos
critérios de
julgamento
Desenvolvimento
de software
Produtos/Prod.
intermediários
Medição
Valor medido
Pontuação
Nível de
pontuação
Julgamento
Resultado
(aceitável ou inaceitável)

"O" desenvolvedor



Norma ISO 15.408 (CC)

- **Segurança como quesito implicito de qualidade – criterio evoluido da norma estatica TCSEC**
  - CIDAL
  - Nao repudio
- **Segurança do Ambiente de Desenvolvimento**
- *Segurança da Aplicaçao Desenvolvida*
- **Garantia de Segurança da Aplicaçao Desenvolvida**

Norma ISO 15.408 (CC)

- **Segurança na Aplicaçao Desenvolvida**
  - Funçoes intrisicamente seguras
  - Verificar codigos de erros
  - Atentar para tamanhos de buffer
  - Documentaçao

Intrisicamente Seguras

- **A funçao deve ser segura por si propria, e nao contar com “testes” antes de sua chamada**
  - Ex: funcao insegura: strcpy()

- **char teste[10], teste2[12];**
- **if (! (strlen(teste2) > strlen(teste)))**
  - **strcpy(teste,teste2);**

Hook de Funcoes

- **Para testarmos codigos prontos, podemos utilizar hook de funcoes**
- **Consiste em interceptarmos a chamada da funcao feita pelo nosso software e passarmos novos argumentos**

Verificar retorno de erros

- As funcoes podem retornar valores, ler a documentacao sobre estes retornos e fazer testes
  - Ex: malloc()

- #define MALLOCTEST(pointer) if (pointer == NULL) serror(“\n No memory\n”);
- teste=(char *)malloc(3);
- MALLOCTEST(teste);

Norma ISO 15.408 (CC)

- Niveis de garantia
  - EAL1 – Testado Funcionalmente
  - EAL2 – Testado Estruturalmente
  - EAL3 – Metodicamente Testado e Verificado
  - EAL4 – Metodicamente Projetado, Testado e Revisado

Recursos In-depth

- Stack nao executavel

- Endereços de bibliotecas randomizados

- Inserçoes nao contiguas na Stack --> Finalizadores de bloco

- Heap “segura” -> Nao insercao de structs de controle

Falhas de Software

- Stack Overflow
- Heap Overflow
- Race Condition
- Format String
- Off-by-one
- Off-by-few
- Integer Overflow
- Double-free

Stack Overflow

- Falha muito comum (provavelmente a mais comum entre todas e a mais facil de ser explorada)

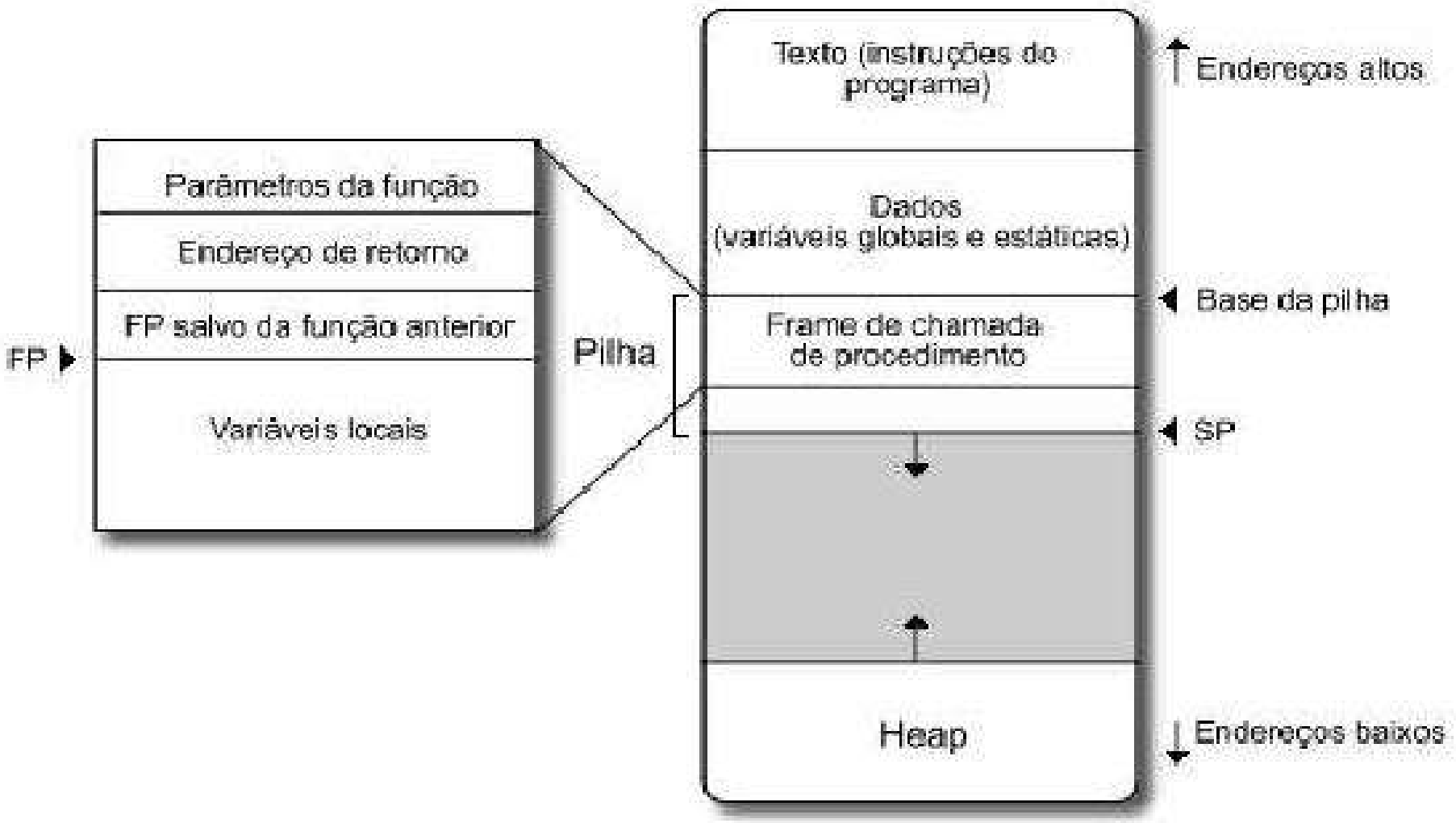
- Existe grande dificuldade de detecçao por tratar-se de interaçoes e loops, uso de funçoes do proprio usuario

- Provavelmente poderia ser detectada atraves de analise de funçoes perigosas e como estao sendo usadas, dando-se grande numero de FALSOS NEGATIVOS

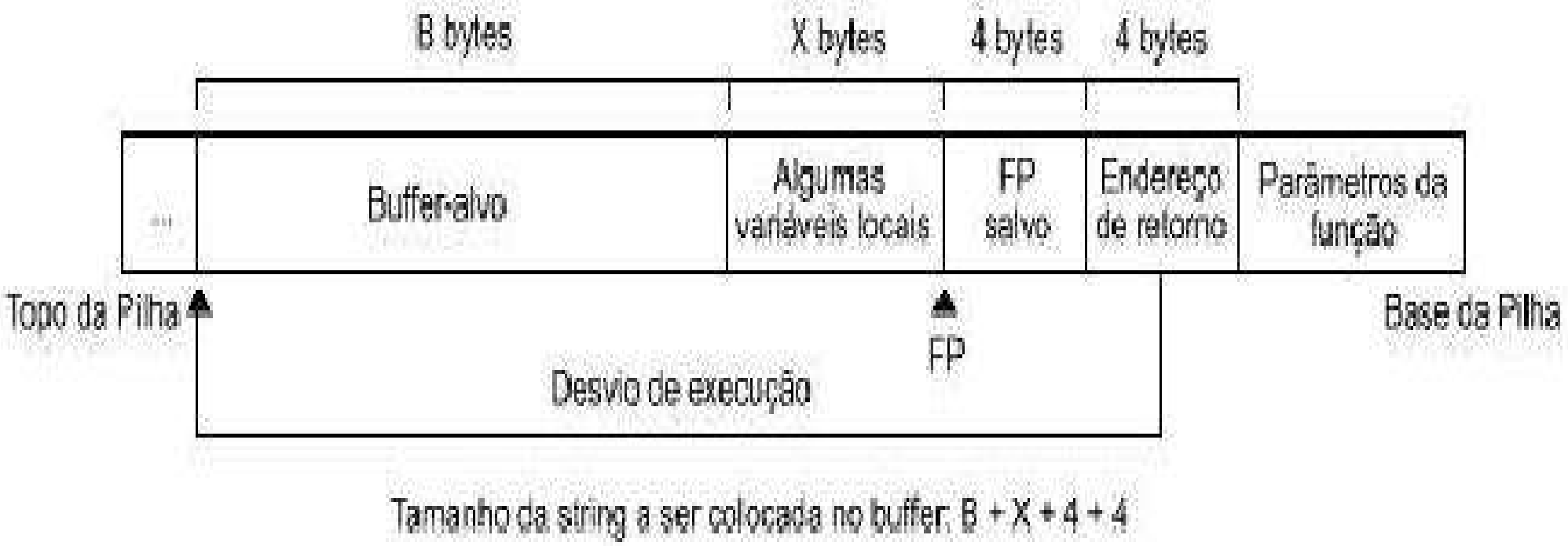
- Consiste da alocaçao de uma variavel com tamanho fixo e da tentativa de armazenar-se nela mais dados do que sua declaraçao continha (geralmente dado pelo usuario).

- Sobrescreve-se na pilha o RET, que quando a funçao retornar sobrescreve o registrador EIP e permite execuçao de codigo arbitrario.

Stack Overflow



Stack Overflow



Heap Overflow

-Deve-se levar em consideraçao quase todos os quesitos dos itens anteriores

- Acontece com variaveis alocadas dinamicamente e ponteiros

- Um estouro em uma variavel acarreta na sobrescrita do ponteiro de outra, permitindo execuçao de codigo arbitrario ou sobreposicao de ponteiros permitindo passar-se por autenticaçoes e escrita em arquivos aleatorios

Heap Overflow

```
#define BUFSIZE 16


int main(int argc, char **argv)
{
        FILE *tmpfd;
        static char buf[BUFSIZE], *tmpfile;
        tmpfile=”/tmp/lalala.tmp”;
        printf(“Digite o que colocar em tmp”);
        gets(buf); // vulneravel
        ...


}
```

Integer Overflow

- Enquanto os programadores e auditores de codigo cada vez mais se atentam para falhas de Stack/Heap Overflow, uma nova modalidade surge

- Integer overflow nada mais e do que a tentativa de armazenamento de valores maiores do que os possiveis em variaveis inteiras (short, long, etc)

- Devido a um integer overflow, pode-se passar por testes de condiçoes e conseguir-se acarretar situaçoes de Stack/Heap overflows mesmo onde nao existiam, dai a dificuldade em detecçao automatizada deste tipo de procedimento

Integer Overflow

Exemplo utilizando-se “ carne morta “

Race Condition

- Uma das falhas mais dificeis de se detectar em auditorias manuais de codigo e em auditorias automaticas de fonte

- Geralmente pode ser descoberta automaticamente atraves do uso do FUZZER, por acarretar a situaçao de condiçao de corrida com o stress do software

- Consiste em se explorar condiçoes de acessos a recursos compartilhados, onde geralmente o resultado obtido consiste em execuçao de codigos com privilegios elevados (escalaçao de privilegios)

Race Condition

Exemplo com o /tmp do sistema

Format String

- Consiste em se explorar condiçoes em que uma funçao tem o objetivo de ser utilizada para diversas finalidades, recebendo nao apenas parametros (variaveis/ponteiros) como tambem o formato como estes devem ser impressos (se serao strings, hexa, inteiros, etc).

- Funçoes da familia printf, syslog e etc sao muito suscetiveis a este tipo de ataque

- Muitas vezes o programador esquece da string de formato, o que tambem permite o acontecimento destas falhas

Format String

Exemplo com o WU:

site index %d %d %d

site exec %d %d %d

Off-by-one

- Erro comum e de dificil exploraçao consiste em se declarar uma variavel (ou alocar dinamicamente memoria para ela) esquecendo-se de um byte

- Muito comum principalmente porque programadores esquecem do terminador de string “\0” que ocupa um byte em qualquer string de caracteres em C

- Atraves deste tipo de falha costuma-se passar por condiçoes de testes e acarretar-se outros tipos de falhas para execuçao de codigo arbitrario

Off-by-one

```
char matriz[10];

for (i=0; i<=10; i++)
        matriz[i]='a';
```

Off-by-few

- Mesmo principio de funcionamento do off-by-one, mas neste caso seriam alguns poucos bytes

- Nao se iguala ao stack overflow por exemplo, pois esta quantidade de bytes nao e suficiente para sobrescrever-se o RET (EIP).

- A exploraçao se da igualmente o off-by-one e tambem existem tecnicas de sobreposicao de outros registradores e manipulacao de espacos de memoria para execuçao direta de codigo arbitrario

Off-by-few

./alloc 4
./alloc 8
./alloc 12
./alloc 13

Exemplo mais complexo
./off

Double-free

- Apos se alocar memoria atraves do uso de funçoes da familia alloc em C, e muito comum livrar-se destes junks atraves da funçao free(ponteiro)

- Muitas vezes estas condiçoes de free estao dentro de variaveis condicionais

- Alguns casos podem ocorrer em que estas condicionais sao manipuladas (atraves de outras falhas) para serem verdadeiras e causarem o free duas vezes em um ponteiro

- Perceba-se que a falha em si consiste em se liberar memoria que ja nao mais esta alocada, pois a funçao free podera executar o codigo arbitrario armazenado naquele setor

Double-free

“

void free(void *ptr);

free() frees the memory space pointed to by ptr, which must have been returned by a previous call to malloc(), calloc() or realloc(). Otherwise, or if free(ptr) has already been called before, undefined behaviour occurs.”

Exemplo: ./double

Directory Traversal

- Acontece quando se aceita dados fornecidos pelo usuario e atraves deste tenta-se acessar/criar/gravar/manipular algum arquivo

- O usuario pode fornecer paths relativos, como ../.. ou ././ para passar por checagem de Strings ou executar/carregar arquivos indevidos

- Kernel do solaris ja apresentou este tipo de problema: Determinada system call recebia do usuario o nome do modulo a ser carregado e procurava o mesmo no diretorio /usr/modules. O usuario poderia especificar por exemplo ../../../home/meumodulo e carregar modulo proprio, ganhando privilegios de administrador

Directory Traversal

Exemplo ./directory

Arquivo 1: mostrar.txt

Arquivo 2: ../etc/shadow

Dificuldades

- Sistemas operacionais e compiladores tem suas proprias versoes de funcoes perigosas (como strcpy)

- Testes de funcao em tempo de execucao podem ser performados atraves de Hooks (sejam de funcao compartilhadas como estaticamente linkadas)

- Funcoes estaticamente compartilhadas podem ser testadas atraves da injecao de codigo diretamente na memoria, dando “jump” para areas de nosso sistema de testes

LEMBRETE

"Segurança, segundo nossa filosofia e um processo complexo que vai desde uma boa interface com o usuario, ate uma boa administraçao e atualizaçao dos servidores, aliados a uma programaçao em constante teste e correçao.

- **INDEPENDE DO USUARIO**

- **EMBORA SEJA REGULAMENTADA, DEVE SER IMPOSTA**

Referências

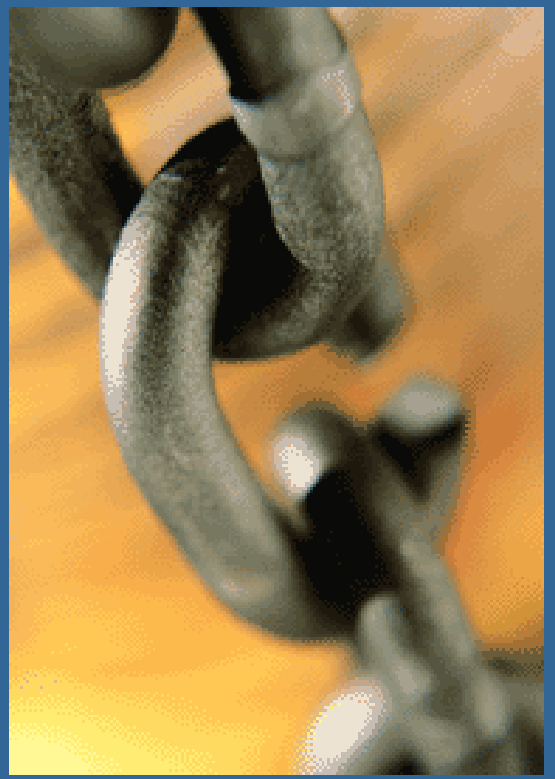
- SCMorphism
  http://www.bsdaemon.org
- SANS
  http://www.sans.org
- Security Focus
  http://www.securityfocus.com
- NFR Security
  http://www.nfr.com
- ISS
  http://www.iss.net

Agradecimentos

- **Eventos, Workshops, Seminários, Palestras, RoadShows, Conversas**

- **Comunicação**

- **Elo mais fraco da segurança da informação**



FIM! Será mesmo?

DÚVIDAS ?

Rodrigo Rubira Branco
rodrigo@firewalls.com.br