Atividade 2

Charles Henrique Porto Ferreira
RA: 11103409

19 de Fevereiro de 2015

1. Quais são os problemas de segurança contidos na codificação desse programa?

   Ele permite que seja acessada uma área que exceda o limite de dados alocada inicialmente para a variável, dependendo do valor da entrada de dados do usuário.

2. Compile o programa usando as opções default do compilador gcc, acionando:
   $ gcc buffov.c -o buffov

   - Você observou alguma mensagem de erro ou aviso?
     Sim, o compilador da um aviso de segurança informando que a função *gets* não é segura e deveria ser evitada. Além disso ele também avisa sobre a formatação dos dados de saída da função *printf* o qual está usando uma formatação para o tipo de dado long int porém o dado inserido é do tipo int.
   - O arquivo executável foi gerado?
     O arquivo executável foi gerado porque o problema de segurança não impede o funcionamento/compilação do programa. É um problema que pode ocorrer de acordo com a entrada de dados.
   - Que conclusão você tira a respeito?
     O mais importante a se observar é que o compilador nao vai impedir o seu programa de ser executado se houver alguma falha de segurança deste tipo, logo o simples fato do programa ter sido compilado e gerado não significa que ele está pronto para ser usado. Outro detalhe que deve ser observado e em relação as bibliotecas usadas no programa. É muito importante conhecer a documentação delas e verificar quais são mais ou menos seguras e qual a melhor forma de usar elas.

3. Execute o código gerado, acionando: **$ ./buffov**

   - Teste o programa com diferentes tamanhos para a string path. O que ocorre quando o tamanho é menor do que 10?

     O programa executa normalmente.

   - E quando o tamanho é igual a 20?
     O programa também executa normalmente.

   - E com tamanho igual a 23?
     O seguinte erro aparece:
     *** stack smashing detected ***:
     E o programa é abortado.

   - E com tamanho maior que 50?
     O seguinte erro aparece:
     *** stack smashing detected ***:
     E o programa é abortado.

   - E para quais desses tamanhos, o programa fornece respostas incorretas ou inesperadas?
     Para tamanhos acima de 22 o programa responde de forma incorreta e termina a execução.

   - Para cada erro, explique o que houve e dê uma justificativa
     O problema que esta ocorrendo com as saidas que apresentaram erros é ocasionado sempre que uma

entrada de dados tem um valor acima do tamanho reservado para a variável que vai receber os dados. Neste caso os dados sobressalentes começam a sobreescrever outra área de memória que não havia sido originalmente resevada para ela ocasionando perda de dados e a falha da execução do programa.

4. O que é Stack Smashing?

   Stack Smashing é estouro de um buffer alocado na pilha.

5. O que é Stack Guard?

   Stack Guard e a deteção de modificação do endereço de retorno de funções na pilha.

6. Agora, compile o programa desligando a opção "Stack Guard" do compilador gcc, acionando o comando abaixo. (a opção "Stack Guard" é ativada por default, como na primeira compilação acima, a partir da versão 4.3.3 do gcc):
   $ gcc buffov.c -o bufnosp -fno-stack-protector
   Execute o código gerado, acionando:
   $ ./bufnosp

   - Teste o programa com diferentes tamanhos para a string path. Que diferenças você percebe nas execuções, em relação à compilação com a opção "Stack Guard"?
     O programa executa porém desta vez não aparece o aviso de stack smaching e o programa libera um erro de falha de segmentação.

   - Como você explica essas diferenças?
     Sem a proteção o acesso a área de memória ficou livre, ocasionando um acesso indevido e a devida falha do programa.

7. Quais são as vantagens e desvantagens em se ativar a opção "Stack Guard"?

   A grande vantagem consiste me proteger o programa contra ataques a área de memória que não deveriam ser acessadas.

8. Quais são as vantagens e desvantagens em desativar a opção "Stack Guard"?

   Uma vantagem em desatiar é que seria feito menos verificações no programa ocasionado uma maior agilidade na execução do mesmo.

9. Cite um exemplo em que poderia ser vantajoso e com baixo risco o não uso da opção "Stack Guard"

   Caso tenh certeza de que não existe vulnerabiliade no código poderia ser retirado o StackGuard para aumentar a velocidade de execução do programa.

10. Quais são os problemas de segurança contidos na codificação desse programa?

    Ele não trata o dado na hora de imprimir.

11. Compile o programa usando as opções default do compilador gcc, acionando:
    $ gcc form.c -o form

    - Você observou alguma mensagem de erro ou aviso?
      Sim a mensagem apresentada foi:
      warning: format not a string literal and no format arguments [-Wformat-security]
      printf(msg);
      Ele reclama exatamente da formatação do dado no função *printf*.
    - O arquivo executável foi gerado?
      O arquivo executável foi gerado, pois o problema nao impede a execução do programa.

- Que conclusão você tira a respeito?
  Uma vez que o erro nao é um erro de programa, mas um erro que pode ser ocasionado durante a execução, o compilador nao deve realmente impedir a criação do programa, porém deve-se sempre ficar atendo a essa minúcias das bibliotecas que sao usadas.

12. Agora, compile o programa desligando a opção Wformat-security do compilador gcc, acionando o comando abaixo: **$ gcc form.c -o form -Wno-format-security**
    Se tudo tiver dado certo, seu programa foi compilado, sem nenhum aviso. Teste-o, acionando:
    $ ./form
    Para que serve a opção Wformat-security e qual a importância dela?
    Essa opção adiciona avisos sobre chamadas de funções formatadas que representam possíveis problemas de segurança. Atualmente, estes avisos são nas chamadas do *printf()* onde a string de formatação nao é um literal e não possui argumentos.

13. Teste o programa, fornecendo como entrada para msg as strings abaixo. Indique o que foi observado e suas interpretações:

    - Entrada: um %d:
      Saida: um 50
    - Entrada: um %d dois %d:
      Saida: um 50 dois -1217029088:
    - Entrada: um %d dois %d três %d:
      Saida: um 50 dois -1216975840 tres -1218111050:
    - Entrada: um %s:
      Saida: Falha de segmentação (imagem do núcleo gravada)
    - Entrada: um %s dois %s:
      Saida: Falha de segmentação (imagem do núcleo gravada)
    - Entrada: um %s dois %s três %s:
      Saida: Falha de segmentação (imagem do núcleo gravada)
    - Entrada: um %s dois %s três %s quatro %s:
      Saida: Falha de segmentação (imagem do núcleo gravada)
    - %s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s:
      Saida: Falha de segmentação (imagem do núcleo gravada)

    Quando se digita um opção de formatação no dado a ser impresso pelo printf ele começa a imprimir o valor contidos em algum endereço de memória ou acessa um área proibida e encerra o programa.

14. No sistema operacional Ubuntu e em vários Linux é possível controlar a aleatorização dos espaços de endereçamento, com a opção sysctl -w kernel.randomize_va_space

    - O que significa essa opção?
      Faz com que a escolha do espaco de endereçamento seja aleatória e não sequencial.
    - Qual a importância de manter ativa uma das opções de aleatorização?
      Ela pode ajudar a se defender contra ataques de buffer overflow, tornando difícil para um hacker predizer o endereço de memória da próxima instrução.