Research on Stack Overflow

Autor: Matheus Rotta Alves (184403)

- Descrição:

Ataques do tipo Buffer Overflow exploram a falta de checagem do tamanho do input sendo guardado em um vetor buffer. Sobrescrevendo dados além do fim do vetor alocado, o atacante pode fazer mudanças arbitrárias ao estado do programa adjacente ao vetor. (1)

Deste modo, o termo Buffer Overflow é um termo genérico para descrever exploits que se valem da falta de “bounds checking”.

Stack Overflow é um tipo específico de Buffer Overflow em que a estrutura de dados corrompida é a pilha (“stack smashing attack”). (1)

A linguagem C é muito sujeita a ataques do tipo Buffer Overflow pois nela falta a checagem automática de limites do array, e também porque seus programadores tendem a focar em performance, o que significa que em geral vão evitar checagem de erros quando possível. É o caso das funções gets() e strcpy().

Os dois objetivos mutuamente dependentes de um ataque Stack Overflow são:

1. **Injetar o código de ataque:**
   O atacante insere uma string de entrada que na verdade é código executável, podendo ser código binário nativo à máquina sob ataque. Um exemplo é o código fazer algo como abrir uma root shell com exec(“sh”);
2. **Mudar o endereço de retorno:**

Existe um stack frame para uma função ainda ativa que fica acima do buffer sendo atacado. O overflow do buffer troca o endereço de retorno que seria para a stack frame original para o código malicioso. Obviamente, a instrução de retorno vai mudar PC (program counter) para o código do atacante.

- Alvos: normalmente daemons privilegiados, por dois motivos:
    1. Daemons são servidores, então aceitam input. É nesse ponto que é possível causar o overflow. Se o daemon usa funções que não fazem “bounds checking” como a gets(), é vulnerável.
    2. Se o daemon é privilegiado, abrir uma shell durante sua execução fará com que a shell tenha privilégios root também.

- Como desenvolver um ataque Stack Overflow?

Esse tipo de ataque é normalmente baseado em fazer engenharia reversa do programa a ser atacado, para determinar o offset exato do buffer até o endereço de retorno no stack frame, e o offset do endereço de retorno até o código injetado.

Como “relaxar” tais exigências? Duas opções:

1. a localização do endereço de retorno pode ser aproximada se replicarmos o endereço de retorno desejado várias vezes por volta da região do endereço de retorno;
2. o offset até o código malicioso pode ser aproximado se colocarmos antes do código malicioso um monte de instruções NOP (No OPeration). Se a sobrescrição do end. de retorno fizer com que o programa pule para o meio do “campo” de NOP’s, então inevitavelmente a execução chegará ao código malicioso. (aumentar o tamanho do alvo).

Já que praticamente todas as informações expostas até agora vieram do mesmo paper[(1)], nada mais justo que eu exponha também qual era o seu propósito original:

- Como proteger seu programa de um Stack Overflow?

A referência (1) propõe seu próprio software para tal fim: StackGuard. Usa dois métodos diferentes:

1. detectar mudança na palavra de return address antes do retorno;
2. prevenir mudanças no return address com MemGuard.

1- Como detectar mudanças na word do endereço de retorno? Colocar uma “canary word” antes da word de return address. Assim, a suposição é que o endereço de retorno só pode ser sobrescrito se a canary word também for. Isso é verdade para ataques buffer overflow pois o atacante é obrigado a sobrescrever tudo de maneira linear e sequencial. A partir daí, basta verificar se a canary word está intacta antes de pular para o valor na palavra do endereço de retorno. É importante gerar a canary word de maneira aleatória para que o atacante não consiga simulá-la sobrescrevendo-a de maneira feliz por ela mesma.

2- Como prevenir, de fato, mudanças na palavra do endereço de retorno?
Grosso modo, MemGuard funciona marcando páginas virtuais de memória que contêm termos que raramente mudam como read-only. Então, por exemplo, se return address quase não varia ao longo da execução do programa, poderia ser marcado como read-only para ficar protegido. Caso seja necessário um write numa página protegida, teria um trap handler que emularia o write para uma palavra não

protegida numa página protegida, o que custa 1800 vezes mais que um write normal. Então esse tipo de operação não pode ser frequente.

- Exemplo de incidente:

-Como visto nas aulas, o Morris Worm usava dois métodos diferentes de exploit: vulnerabilidade no programa sendmail usando a opção debugger e buffer overflow no programa fingerd (finger daemon).
Entrando em mais detalhes nesta ocasião, o fingerd esperava tinha um buffer de 512 bytes para receber input, e o worm mandava uma string de 536 bytes que justamente trocava o endereço de retorno para que esse apontasse para o buffer no stack. As instruções que foram escritas nessa porção foram uma série de no-ops (como exposto anteriormente, servem para “aumentar o alvo”) e, em seguida, uma série de instruções assembly que acabavam executando:

execve(“/bin/sh”,0,0)

esse comando abre uma shell. Como fingerd rodava com privilégios root, a shell abria com privilégios root.

Bibliografia: (1) StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, SteveBeattie, Aaron Grier, Perry Wagle, and Qian Zhang, Oregon Graduate Institute of Science & Technology; Heather Hinton, Ryerson Polytechnic University