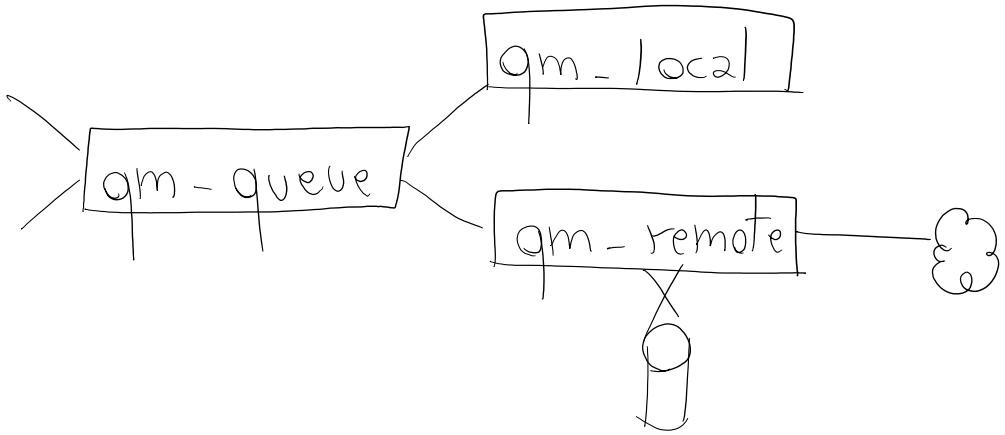
Segurança da Computação

Matheus Venturyne Xavier Ferreira
Universidade Federal de Itajubá
20 de Outubro de 2015

Como escrever código seguro?

- Qmail: agente de transferência de e-mail (MTA)
  - Desenvolvido em 1997 e 16617 linhas de código
  - Último bug: desconhecido
- Sendmail
  - 134976 linhas de código
  - Último bug: Maio de 2009



Como escrever código seguro?

- Resposta
  - Eliminar bugs
  - Eliminar código
  - Eliminar código confiável
- Distrações
  - Seguir o invasor
    - Número de bugs é infinito
  - Minimizar privilégios
  - Velocidade

Eliminar BUGs

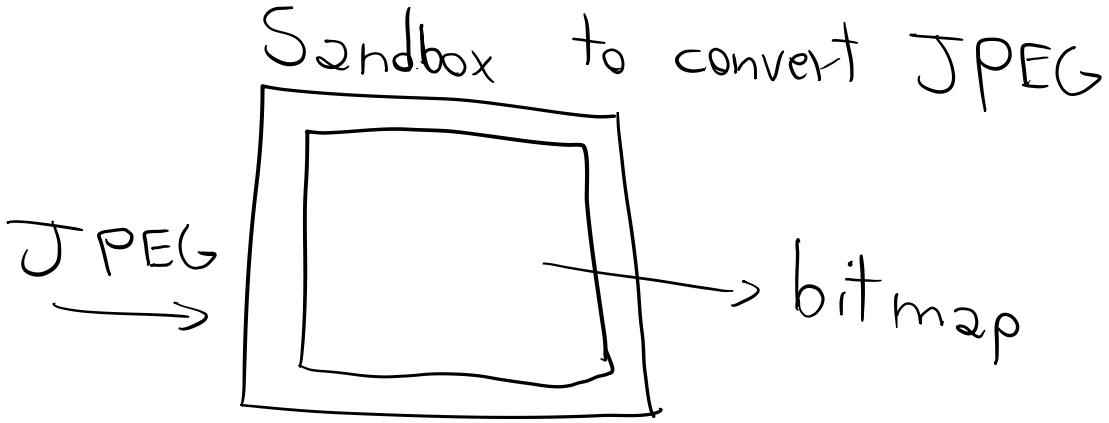
- Reforçar dataflow (sem variáveis compartilhadas)
  - Qmail é formado por processos individuais que podem apenas se comunicar apenas por pipelines
- Evitar parsing
- Faça possível testar condições especiais
- Simplificar a semântica de inteiros
  - Y = X + 1 (Y pode ser menor que X)
- APIs são importantes sprint (snprintf) strcpy (strncpy)

Eliminar Código

- Há uma relação entre o número de linhas de código e o número de bugs
- Identifique funções comuns
- Manipulação automática de erros
- Reutilize mecanismos do Sistema Operacional
  - Reutilize protocolos de comunicação
  - Reutilize controle de acesso
  - Reutilize Sistema de arquivos

Eliminar código confiável

- TCB (Trusted computing base)
- Não importa o quão severo um bug seja se ele está em uma prisão para códigos inseguros
- Processo é criado para manipular JPEG e é deletado quando complete
  - A primeira conversão pode alterar a forma como se converte JPEGs futuros



Ataques à memória

- Todo software é escrito com suposições que implicam em um risco
  - Para um dado software se assume que ele não conecta à internet – nem sempre verdadeiro em C/C++
- Muitas aplicações escritas em linguagens inseguras C/C+
  - Bilhões de linhas já escritas
  - Necessidade por acesso de baixo nível e performance (e.g. sistemas operacionais)
- Começam com corrupção da memoria, um overflow ou um dangling pointer
- Suposições para defesas
  - Probabilidade de diferentes tipos de ataque
  - Capacidades de um invasor
  - Ambiente de execução

Ataque 1

- Corromper o endereço de retorno da função (Buffer overflow)
  - Limitações
    - Requer que o hardware esteja disposto a executar código encontrado na stack
    - O payload não deve conter null bytes
  - Variantes
    - Adicionar some indirecionamento: retornar normalmente mas o Segundo retorno está comprometido
    - Sobrescrever ponteiros para retorno de tratamento de exceções (normalmente reside na stack)

Ataque 2

- 2) Corromper ponteiros de funções armazenados na heap
  - Overflow de ponteiros na heap
  - Limitações
    - Necessita a habilidade de determinar o endereço da memória da heap que está sendo corrompida

Ataque 3

- Corromper ponteiro e executar código já existente (jump-to-libc ou return-to-libc)
  - Executar código em uma ordem diferente
  - Limitações
    - Deve desenvolver o ataque com conhecimento do endereço do alvo

Ataque 4

- Corromper dado para alterar o comportamento (data-only, non-control-data attack)
  - Limitações
    - Somente alguns dados podem ser corrompidos
    - Mesmo que todo o dado possa ser alterado o ataque é limitado pelo comportamento do software

Ataque 5 – Corromper Código

- Sobrescrever uma instrução da memória
- Hoje código pode ser apenas lido (nunca escrito); no entanto, há compilação just-in-time (JIT) em navegadores (JavaScript ou Flash).
  - Não se pode reforçar integridade de código nessas tecnologias

Ataque 6 – Information Leak

- Ler a memória
- Dados importantes podem ser encontrados na memória
- Pode ser usado para contra-atacar defesas probabilísticas baseadas em randomização (e.g. ASLR) e segredos

Defesas

- Probabilísticas
  - Randomizar conjunto de instruções
  - Randomizar o espaço de endereços
- Determinístico
  - Monitor de baixo nível que gera faults em caso de problemas

Propriedades das Defesas

- Custo (Overhead)
  - Performance (crítico)
  - Memória
- Compatibilidade
  - Source: impraticável modificar código fonte
  - Binary: ainda devem lincar com bibliotecas (modularidade, dll, so)
- Proteção: força, efetividade, precisão
- Faso negativos vs falso positivos

Defesa 1 – Stack Canaries

- Bytes reservados colocados antes do endereço onde EBP é salvo
- Overhead
  - Baixo, alguns por cento
- Limitação
  - Defesa apenas contra o ataque 1)
- Contra-ataque:
  - O dado corrompido pode ser usado antes do retorno da função
  - O invasor pode tentar adivinhar o cookie

Defesa 2 – Variáveis Locais

- Mover variáveis locais da função abaixo do buffer da stack
- Overhead
  - Nenhum – feito em tempo de compilação
- Limitação
  - Proteção apenas contra stack-based buffer overflow

Defesa 3 – W(+)X

- Útil contra injeção de código
- Um segmento da memória só pode ser escrita ou executada nunca ambos
- Overhead
  - Pode ser alto, mas alguns por cento nas arquiteturas de hoje
- Limitação
  - Software legado pode fazer uso de dado que é executável
  - Não defende contra jump-to-libc

Defesa 4 – Control-Flow Integrity (CFI)

- É gerado um grafo das possíveis caminhos de execução o que gera uma exceção se o caminho é desrespeitado.
- Overhead
  - Em media 16%
- Limitação
  - Não defende contra data-only attack
  - Não Evita ataques onde o caminho de execução são caminhos possíveis

Defesa 5 - Criptografia

- Criptografar o endereço em código e ponteiros de dados
- Overhead
  - Para se manter, se utiliza criptografia com xor-ing
- Limitação
  - Somente Evita ataques que envolvem ponteiros
  - Operações aritméticas com ponteiros

Algoritmo de criptografia - XOR cipher

XOR		
Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Xor-cipher: 0101

Cipher	0101-0101-01
Data	0011-0001-10
Encryption (Cipher XOR Data)	0110-0100-11
Decryption (Cipher XOR Encrypted)	0011-0001-10

Defesa 6 – ASLR (Address Space Layout Randomization)

- A cada execução do programa o layout da memória é alterado. A linguagem de programação não se preocupa onde o dado será armazenado
- Overhead
  - Se bibliotecas compartilhadas podem ser colocadas em endereços diferentes em cada processo haverá um overhead e consumirá memória
- Limitação
  - Ainda permite data-only attacks
  - O número de possibilidades não é tão grande
  - Se o invasor é capaz de repetir o ataque até que ele encontre o endereço ele irá suceder
- Contra-ataque
  - Memory Leak

Segurança da memória

- Parar com toda a corrupção da memória
- Segurança espacial com limite de ponteiro: cada ponteiro pode ser usado para acessar somente a memória ele é especificado (alto overhead)
- Segurança especial com limite de objetos: associar limite para cada objeto
- Segurança temporal (use-after-free ou double-free)
  - Nunca usar a mesma memória virtual depois de liberar uma memória (desperdício)
  - Somente utilizar uma memória com objetos que cabem no mesmo espaço
  - Valgrind
    - Tentar detectar use-after-free-bugs
  - Tabela global de ponteiros

Referências

- U. Erlingsson: Low-Level Software Security: Attacks and Defenses
- L. Szekeres, M. Payer, T. Wei, and D. Song: Eternal War in Memory
- D.J. Bernstein: Some thoughts on security after ten years of qmail 1.0
- D. Engler et al.: Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code