Programação Segura: Um estudo de falhas e ferramentas de auxílio

Luiz Gustavo C. Barbato, Luiz Otávio Duarte e Antonio Montes

Instituto Nacional de Pesquisas Espaciais – INPE
Laboratório Associado de Computação e Matemática Aplicada
Av. dos Astronautas, 1758 – 12227-010 – São José dos Campos, SP.
E-mail: {lgbarbato,duarte,montes}@lac.inpe.br

Resumo

A diferença entre o número de pessoas mal intencionadas, que procuram por vulnerabilidades em sistemas e pessoas que tentam suprir as falhas nos mesmos é muito grande. Com isso, empresas e desenvolvedores devem cada vez mais ficar atentos à maneira como a programação de seus sistemas é realizada. Algumas ferramentas auxiliam no processo de auditoria de códigos fontes. Entretanto, é importante conhecer o funcionamento de cada uma delas, o que estas podem induzir e do que realmente são capazes. O objetivo deste trabalho é mostrar alguns dos problemas atualmente encontrados na codificação dos sistemas, como estouro de buffers, condições de corrida e validações de entrada Além disso, tem o objetivo de mostrar algumas ferramentas que auxiliam a auditoria de códigos, como Flawfinder, ITS4, PScan e RATS.

Palavras-chave: *segurança, programação segura, auditoria de código*

1. Introdução

Há bem pouco tempo, segurança de sistemas de informação era sinônimo exclusivamente de proteção, assumindo sempre uma posição puramente defensiva. O número de pessoas tentando violar a segurança de sistemas, passando dias e noites em busca de vulnerabilidades nos sistemas com intuito de desenvolver ferramentas que permitam o acesso ao sistema alvo, é muito grande.

Além disso, se para cada organização existem alguns poucos profissionais dedicados a estudar e manter os sistemas seguros, por outro lado, existem inúmeras pessoas tentando encontrar uma maneira de penetrar e comprometer os mesmos.

Atualmente, o número de vulnerabilidades encontradas em softwares é um problema em expansão. Esta expansão é decorrente tanto do aumento no número de sistemas no mercado, da forma como estes softwares são concebidos e da maneira como são mantidos, configurados e utilizados.

Dados do ICAT/NIST mostram que no ano de 2004 uma grande porcentagem dos problemas de segurança encontradas em sistemas é devida a má codificação dos mesmos. Do início do ano até agosto, 52% dos erros eram devidos aos erro de validação de entradas, que envolvem problemas de *estouro de buffers*. Do total de vulnerabilidades encontradas, pelo menos 74% eram decorrentes de falhas de codificação.

A má codificação do software é a causa imediata das vulnerabilidades do mesmo. Muitas empresas desenvolvem sistemas com códigos inseguros mesmo sem intenção. Existe um grande número de razões para isto.

Na maior parte das vezes, empresas que desenvolvem *software* são contratadas para especificar, projetar, desenvolver, testar e documentar em um tempo pré-determinado. Esta pressão induz, muitas vezes, a erros graves, como, por exemplo, a não preocupação com a escolha de funções seguras para exercer determinadas tarefas, ou a utilização incorreta das interfaces de programação que a linguagem utilizada disponibiliza. Além do que, muitas das vezes a segurança não faz parte do projeto.

Um outro aspecto que deve ser levado em consideração é a má formação dos desenvolvedores de sistemas. São poucas as universidades que ensinam seus alunos a programarem de maneira correta, aplicando técnicas de programação segura. E são em menor número ainda as que fornecem a disciplina de programação segura em suas grades curriculares.

Algumas ferramentas de auxilio a auditoria de códigos estão disponíveis hoje e servem como apoio aos analista de sistemas. Entender como estas ferramentas funcionam, o que podem induzir e o que de fato são capazes é muito importante para qualquer profissional ligado direta ou indiretamente com segurança e programação de sistemas.

O objetivo deste trabalho é ilustrar o atual quadro de problemas encontrados na codificação de sistemas e mostrar quais ferramentas podem ser utilizadas com

a finalidade de trazer um maior nível de segurança para o sistema.

2. Problemas na Codificação

A maioria dos problemas na codificação dos programas são causados porque os desenvolvedores não escrevem códigos seguros. Algumas das razões pelas quais isto ocorre foi sumarizada por Aleph One num *email*[1] enviado para a lista *Bugtraq* em dezembro de 1998.

- A maioria das universidadas não se preocupam com disciplinas voltadas a segurança de computadores;
- Livros de programação não ensinam técnicas de programação segura;
- Não existem métodos formais de verificação;
- A linguagem C é insegura;
- Programadores não pensam em ambientes ”multi-usuários“;
- Programadores são humanos. Humanos são preguiçosos;
- A maioria dos programadores não são bons programadores;
- A maioria dos programadores não trabalham com segurança;
- Consumidores não se preocupam com segurança;
- Custo extra no tempo de desenvolvimento;
- Custo extra nos testes;

Dentre os vários problemas que podem existir na codificação de programas, serão descritos apenas os três mais comuns e que serão utilizados durante a explicação das ferramentas:

2.1. Estouros de Buffer

Os estouros de *buffers*[2] são problemas causados devido a não validação do tamanho da memória utilizada, excedendo em certas circunstâncias, a sua capacidade de armazenamento. Os *buffers* são alocados com tamanhos fixos de memória previamente determinados, e a utilização deste espaço para armazenar uma quantidade de dados que ultrapasse este tamanho causa o estouro de *buffer*.

[1]Este email pode ser encontrado em `http://seclists.org/bugtraq/1998/Dec/0062.html`

[2]*Buffers* são regiões de memória alocada para o armazenamento de determinado tipo de dado.

Estes estouros, podem ser classificados de duas formas, de acordo com a forma em que foram criados. Eles podem ser alocados estaticamente(*stack*) ou dinamicamente (*heap*), onde a primeira forma é utilizada para o armazenamento de variáveis locais e parâmetros passados por valor em funções. Já na segunda, são armazenadas variáveis alocadas dinamicamente e variáveis globais.

2.2. Validações de Entradas

Este problema consiste da não verificação dos valores de entrada nos programas, gerando assim, situações inesperadas como execução de programas indevidos, estouros de *buffers*, execução de comandos SQL não permitidos, dentre outras. A entrada de dados em um programa pode ser feita de várias maneiras como parâmetros de execução do programa, leituras de teclado e arquivos, através de comunicação interprocessos (memória compartilhada, *pipe*,...) e via rede (*sockets*), etc. Cada entrada possui uma validação específica de acordo com a finalidade da mesma. Há entradas que esperam por números interos ou reais e de tamanhos específicos[3], outras esperam por caracteres possíveis em um endereço de *email*, outras confiam somente na seleção de poucos dados apresentados em um formulário (*combo box*), outras esperam por determinados padrões de dados em um arquivo, e a mudança proposital destes valores para valores não esperados, pode mudar o comportamento do programa.

2.3. Condições de Corrida

As condições de corrida ocorrem em ambientes que suportam multiprogramação. Este problema acontece quando dois ou mais processos utilizam a mesma variável, mesmo arquivo ou outros recursos simultâneamente. Um recurso pode ser modificado, intencionalmente ou não, por um processo, e que será requisitado por um segundo, fazendo que este se comporte de maneira não esperada. Um exemplo deste tipo de problema esta relacionado com a utilização de arquivos temporários criados pelos programas, onde estes podem ser antecipadamente trocados por *links* simbólicos que apontam para um outro determinado arquivo cujo usuário que está rodando o programa possui permissão de escrita.

3. Auditoria de Código para Segurança

A auditoria de código para segurança é a análise do conjunto de códigos do sistema para que problemas de segurança possam ser identificados e sanados. A auditoria não tem a tendência de encontrar falhas maiores

[3]Inicialmente são representadas por *strings*

que somente uma análise da arquitetura do software poderia encontrar.

O trabalho requerido para se entender e analisar todo o código fonte de um programa é muito grande, já que é necessário um grande conhecimento do sistema, além de ser muito dispendioso. Isto torna esta abordagem dificilmente adotada.

Um bom método que pode ser adotado para auditoria começa através da identificação de todos os pontos que recebem entradas de usuários, que obtém dados de outros programas, ou alguma outra fonte não confiável. É importante identificar estes pontos, pois atacantes podem passar entradas específicas para partes frágeis do programa.

Além da identificação dos pontos que recebem entradas, é importante mapear o caminho que as entradas tomam no programa. Ou seja, quais funções recebem estas entradas. O próximo passo seria identificar sintomas de problemas, analisando manualmente trechos de códigos para identificar se possuem ou não algum tipo de vulnerabilidade.

Este método de análise apesar de ser bom para encontrar falhas comuns e levar um menor tempo, não garante completude. Entretanto, as análises podem ser realizadas em diversos níveis que quanto mais abrangentes, mais efetivos.

Um outro método, utilizado por ferramentas que suportam a análise, é o que procura por vulnerabilidades específicas. Ou seja, que procura por vulnerabilidades bem documentadas como *estouro de buffers* e *condições de corrida* . Assim, não levando em consideração o funcionamento do programa e sim a existência ou não de funções inseguras.

4. Uso de Ferramentas para Auxílio

Algumas ferramentas estão disponíveis hoje para análise estática de códigos. As mais difundidas são: *Flawfinder*, *ITS4*, *PScan* e *RATS*.

Estas ferramentas são destinadas a pessoas que realizam análises de códigos, ajudando-as a encontrar problemas através de uma lista de potenciais problemas.

As ferramentas não entendem a semântica do código, simplesmente casam o código contra potenciais falhas conhecidas. Tanto que grande parte do trabalho realizado por estas ferramentas pode ser reproduzido com auxílio do comando *grep*. Entretanto, no caso do *grep*, todas as falhas precisariam ser lembradas toda vez que um trecho novo de código fosse analisado.

5. Ferramentas de Auxílio à Auditoria de Código

5.1. Flawfinder

O *Flawfinder* [12] é uma ferramenta escrita em *Python* que examina códigos fontes da linguagem *"C"* e reporta possíveis fraquezas de segurança. É uma ferramenta de código aberto e licença GPL[4] (*"GNU Public License"*).

Este programa trabalha com um banco de dados de funções C/C++ com problemas bem conhecidos. Como: *"strcpy();"*, *"strcat();"*, *"gets();"*, *"sprintf();"* e funções da família *"scanf();"*.

Além disso, identifica problemas de formatação de cadeias de caracteres nas funções: *"[v][f]printf();"*, *"[v]snprinf();"* e *"syslog();"*.

Identifica, também, *condições de corrida* encontradas em funções do tipo: *"access();"*, *"chown();"*, *"chgrp();"*, *"chmod();"*, *"tmpfile();"*, *"tmpnam();"* e *"mktemp();"*.

O *Flawfinder* ainda identifica funções da família *"exec();"*, *"system();"* e *"popen();"*, produzindo como saída uma lista de potenciais problemas de segurança organizadas pelo risco de cada função encontrada. O risco não depende exclusivamente da função utilizada, mas também dos parâmetros que são passados para estas funções. Isto é, se os parâmetros são constantes ou variáveis.

Os riscos são ordenados do maior risco "5" ("*maximum risk*") para o menor risco "0" ("*no risk*"). O usuário pode setar a partir de qual risco deseja ser alertado. Além disso o relatório gerado pelo programa pode ser tanto em "texto plano" quanto em "html".

5.2. ITS4

ITS4 [1] é o acrônimo de *"It's The Software, Stupid! [Security Scanner]"*. Como o *Flawfinder*, é uma ferramenta que identifica possíveis problemas em trechos de códigos *"C/C++"*.

Esta ferramenta permite que o usuário defina qual o método de ordenação do relatório de saída. Ou seja, o relatório pode ser ordenado de maneira crescente quanto ao risco, ordenado pelos arquivos que foram analisados, pela vulnerabilidade ou pela ordem em que cada possível problema foi identificado.

Outra característica que distingue o ITS4 das outras ferramentas é o fato de permitir que o programador defina quais partes do código devem ser ignoradas, como no exemplo:

```
strcpy(dst, src);  /* ITS4: ignore */
```

[4] A licença GPL pode ser encontrada em `http://www.gnu.org/copyleft/gpl.html`

5.3. PScan

O PScan é uma ferramenta de licença GPL que procura por problemas de formatação de cadeias de caracteres nas funções da família *printf*.

Como esta ferramenta não checa por quaisquer outras funções ou tipo de problema o seu uso é restrito. A menos que definições adicionais de funções sejam passadas para o programa.

Entretanto, a checagem de formatação realizada nestas funções é a mesma realizada para as funções da família *printf*.

5.4. RATS

RATS, acrônimo de "*Rough Auditing Tool for Security*", é uma ferramenta de código aberto distribuído sob a GPL e é desenvolvida pela *Secure Software Inc*.

O objetivo desta ferramenta é encontrar possíveis problemas como *estouro de buffer* e *TOCTOU*[5] *condições de corrida* em códigos fontes escritos em "*C/C++*", "*Perl*", "*PHP*" e "*Python*".

Para cada uma destas linguagens o RATS possui um banco de dados XML correspondente. Isto facilita a inserção de novos possíveis problemas, mas faz com que seja necessário que a biblioteca *Expat* esteja instalada.

6. Conclusões

Apesar de ser um bom recurso de auxílio, as ferramentas de auditoria de código ainda requerem um alto grau de conhecimento da pessoa que estiver auditando o código. Embora estas ferramentas possuam um certo conhecimento sobre algumas vulnerabilidades que não precisam mais serem mantidas na cabeça dos analistas, segundo [10], um analista experiente ainda faz um melhor trabalho do que um inexperiente. Pois, consegue discernir mais facilmente quais dos possíveis problemas encontrados pelas ferramentas são de fato problemas reais ou não.

Mesmo para analistas experientes a análise estática toma muito tempo. Isto pois, as ferramentas de auxilio por vezes retornam um grande número de falsos positivos, o que faz com que muito do trabalho seja desperdiçado tentando-se encontrar problemas que na realidade tendem a não existir.

Por outro lado, estas ferramentas auxiliam desenvolvedores e analistas inexperientes a encontrarem alguns possíveis problemas, conscientizando-os, desta maneira, que o código "pode" possuir falhas. Além disso, estas ferramentas possibilitam que possíveis problemas possam ser encontrados nos primeiros minutos de análise.

[5]TOCTOU: Time of Check, Time of USE

Portanto, estas ferramentas devem ser utilizadas com a devida cautela. Pois, ao mesmo tempo que auxiliam e facilitam o trabalho de análise, podem deixar que uma grande gama de vulnerabilidades permaneça oculta causando uma falsa impressão de completude e segurança.

À medida que novas ferramentas forem desenvolvidas, as atuais se tornem mais maduras e a pesquisa nesta área aumente; o número de falsos positivos e de falhas não identificadas tendem a diminuir, tornando assim estas ferramentas mais confiáveis e efetivas.

Referências

[1] "ITS4: Software security tool". Disponível on-line em: `http://www.cigital.com/its4/verificadoemagostode2004.`

[2] "RATS: Rough auditing tool for security." Disponível on-line em: `http://www.securesw.com/download_rats.html` verificado em agosto de 2004.

[3] Anderson, R. J. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 1st edn., 2001. ISBN 0-471-38922-6.

[4] DeKok, A. "PScan: A limited problem scanner for c source files". Disponível on-line em: `http://www.striker.ottawa.on.ca/~aland/pscan/` verificado em agosto de 2004.

[5] Graff, M. G. & Wyk, K. R. V. *Secure Coding: Principles and Practices*. O'Reilly & Associates, 1st edn., 2003. ISBN 0-596-00242-4.

[6] Heffley, J. & Meunier, P. "Can source code auditing software identify common vulnerabilities and be used to evaluate software security?" 2004. Proceedings of the 37th Hawaii International Conference on System Sciences.

[7] Hoglund, G. & McGraw, G. *Exploiting Software: How to Break Code*. Addison-Wesley Professional, 1st edn., 2004. ISBN 0-201-78695-8.

[8] Howard, M. & LeBlanc, D. C. *Writing Secure Code*. Microsoft Press, 2nd edn., 2002. ISBN 0-735-61722-8.

[9] Kernighan, B. W. & Pike, R. *The Practice of Programming*. Addison-Wesley, 1st edn., 1999. ISBN: 0-201-61586-X.

[10] Viega, J. & McGraw, G. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, 1st edn., 2001. ISBN 0-201-72152-X.

[11] Viega, J. & Messier, M. *Secure Programming Cookbook for C and C++*. O'Reilly & Associates, 1st edn., 2003. ISBN 0-596-00394-3.

[12] Wheeler, D. A. "Flawfinder". Disponível on-line em: `http://www.dwheeler.com/flawfinder` verificado em agosto de 2004.

[13] Wheeler, D. A. *Secure Programming for Linux and Unix HOWTO*. 2003. Disponível on-line em: `http://www.dwheeler.com/secure-programs` verificado em agosto de 2004.

[14] Whittaker, J. A. & Thompson, H. H. *How to Break Software Security*. Pearson Education, 1st edn., 2003. ISBN: 0-321-19433-0.