INSTITUTE FOR SECURITY AND OPEN METHODOLOGIES

SPSMM

PADRÂO PARA PROGRAMAÇÃO SEGURA

TRADUÇÃO PARA PORTUGUES – Lilian Vital/Sandro Melo
Revisão Técnica – Sandro Melo

PADRÂO PARA PROGRAMAÇÃO SEGURA - Victor A.Rodriguez (Bit-Man)

Quando se apresenta uma interface (de programação, usuário, etc), seu uso pode ser subvertido (uso abusivo). É este abuso que deve ser evitado. Já que planejamos definir um padrão que é esta metodologia, desde o lado da programação, e seguramente você que é um programador, arquiteto de sistemas , lider de equipe ou outra coisa no mesmo estilo, queira manter a interface limpa, utilizável e sem componentes que possam ser violados ou mal usados.

À medida que as aplicações da internet começam a ser mais complexas, e a geração de código começa a ser cada vez mais automatizada através do uso de wizards (assistentes), ambientes de programação, frameworks e outras ferramentas deste estilo, começamos a estar em grande desvantagem. As técnicas para programações seguras é algo que tem sido abordado de muitas formas e em muitas linguagens. O que se trata de fazer é uma única metodologia independente de linguagens de programação, ambientes de programação, e ferramentes de desenvolvimento.

A idéia de converter em um padrão aberto nos dá o benefício imediato que pode ser usado por qualquer um, entretanto também podemos incluir aos script kiddies. No entanto se é seguido uma metodologia seguramente os erros mais comuns já não aparecem. Em contrapartida é que quanto mais programadores leiam e entendam este manual, maior é a possibilidade de desenvolver aplicações de forma segura como também para ajudar a melhorar esta tecnologia.

Agradecimentos

Gostaria de agradecer a David A. Wheieler por escrever “Secure Programing for Linux and Unix HOWTO” () que tanto me inspirou para fazer este trabalho também um agradecimento às pessoas que escreveram “SECPROG@SecurityFocus.com working document”(9), algo assim como “Primeiro Dicionário” sobre “Programação Segura”.

Devo também dar um agradecimento muito especial a Pete Herzog por toda a ajuda proporcionada em muitos dos aspectos deste trabalho.

Todos que contribuíram com este manual de alguma forma são mencionados no início deste documento. Cada pessoa recebe um reconhecimento pelo tipo de sua contribuição e não pelo que contribuiu. O uso deste ocultamento é para prevenir a polarização no momento da interpretação. Somente na tradução do documento, sabe-se que as pessoas que realizaram a contribuição, já que o tradutor se converte no contato do documento traduzido que inclue, não somente em manter atualizado o documento, como também as FAQs, agregados e comentários feitos no original, em inglês e traduzido para linguagem desejada.

Termos

- **DoS (Denial of Service)**: Ataque cujo propósito é não permitir o uso do sistema. Não afeta os dados armazenados no sistema.
- **Out-of-the-box**: instalação padrão, utilizando os parametros de configuração básicas como meio de exploração.
- **Super-user**: é um usuário especial (conhecido como root ou administrador) que não tem restrições de acesso dentro do sistema.

Audiência

Este manual foi escrito para profiossionais da programação que queiram estabelecer lineamentos e/ou procedimentos para a programação segura em qualquer projeto de Tecnologia da Informação (TI), independente da linguagem da programação, ambiente de execução e ferramentas de desenvolvimento.

Este manual não avalia a forma correta de usar uma ferramenta de desenvolvimento em particular ou como programar em uma ldada inguagem.

Os desenvolvedores encontrarão muitas utilidades neste manual e realizarão melhores programas, mais resistentes em ambientes hostis tais como a internet.

Alcance

O objetivo é realizar um padrão de metodologia de programação segura que possa ser usado em qualquer processo, manual ou automático, e permita os requerimentos de segurança para maximizar o uso e evitar seu abuso. O resultado indireto é a criação de uma disciplina que possa atuar como um ponto central em todas as provas de segurança independente da linguagem de programação, ambiente de execução de desenvolvimento.

Exceções

Os casos não cobertos, ainda, neste trabalho são:

- Programas cujo propósto é o uso da segurança
- Programas que devem ser executados como super usuário

Preparação para trabalhar em um ambiente hostil

Não é uma tarefa fácil, e foi feito em diferentes maneiras, cada uma cobrindo seu ponto de vista, começando em diferentes formas, etc. Aqui estarão estabelecidas

as minhas.

Se vemos a Secure Programing for Linux and Unix HOWTO (Programando com Segurança para Linux e Unix) (1) (seção 22, Security principles) a programação segura pode ser interpretada como o cumprimento dos seguintes objetivos: confidencialidade, integridade e disponibilidade. Isto é correto e pode ser moldado para qualquer ambiente em particular, mas é uma boa resposta genérica. O problema começa em como alcaçar estes objetivos. O que tratará de enfatizar não é o arranjo dos mesmos, e sim uma metodologia Open Source que provém da base para o desenvolvimento de métodos e/ou procedimentos.

O trabalho de um sistema é implementar requerimentos funcionais que, devidamente acordados possam ser provados. Em uma analise mais detalhada, a codificação se faz cumprir com este objetivo, mas em uma escala atômica (ao nível de subrotina o procedimento de programação). O que não é parte dos requerimentos funcionais, é a disponibilidade do sistema, que estabelece as condições de uso tais como que em dia será usado (dias de trabalho, feriados, etc), janela diária de uso (9:00 a 17:00 hs, etc), máximo tempo permitido sem acesso ao sistema (2 horas mensais, etc), e toda uma série de requerimentos que definem o denominado. Acordo de Nível de Serviço (SLA o Service Level Agreement).
É neste ponto de onde, executando em ambientes hostis como Internet, poder oferecer os níveis requeridos do SLA (Integridade), impossibilitar o roubo de informações (segurança) ou fornecer a informação à pessoa correta (confidencialidade) é de interesse particular.

Uma vez que conhecemos sobre este e seu impacto negativo, há uma grande preocupação em como adotar uma aproximação sistemática para ser resolvido. Para começar, veremos os fatores que dão começo a esta problemática.

- **O que não fazer**: é muito difícil falar sobre o que não se deve fazer. Proteger-se de algo que não se sabe a sua forma exata, é uma tarefa muito díficil (um bom arquivo sobre este tema pode ser visto em Newsletter Crypto-gram de Novembro de 1999 (2). É similar à defesa contra os vírus, não podemos nos proteger a menos que saibamos de sua existência e como derrotar-los.
- **Complexidade**: Isto dá origem a um dos mais velhos e difíceis problemas da humanidade. Chama-se comunicação, e segundo se agrega complexidade a um sistema este que se pode manusear melhor caso se divida em tarefas menores, se designa a equipes diferentes que devem comunicar-se entre si para alcançar o objetivo: quanto mais equipes há uma maior necessidade de quantidade de comunicação, e como já se sabe, converte a construção do sistema em um problema de crescimento exponencial (o melhor é ver ao título *software essential difficulties* de “No silver bullet”[3]). Isto favorece a existência de erros ocultos que são cobertos pela síndrome de “há muitos lugares onde olhar”.

Metodologia

Como este trabalho é parte de “The Open-Source Security Testing Methodology Manual” (OSSTMM), a mesma metodologia será usada, estabelecendo os Parâmetros e tarefas orientadas à mesma Programação Segura, mas com uma visão um tanto diferente. Em OSSTMM se descobre que há um erro, e agora queremos saber onde se está localizado, como reparar e evitar suas aparições em futuras codificações.

Esta seção trata de dois primeiros objetivos (descobrimento e reparação) enquanto que o trabalho de como evitar estes problemas é um tema a ser tratado em um trabalho a ser desenvolvido (não incluso neste).

Dados entrantes

Deve se notar que as interfaces se baseiam na implementação de alguma tecnologia e que a exploração dos problemas de segurança devem ser corrigidos para obter uma completa solução do problema. Não é suficiente corrigir a implementação da interface ou o programa que esta sendo testado.

Algo que devemos ter em mente é que os dados de entrada não são somente adquiridos através dos usuários ou de outras aplicações, são também obtidos desde o sistema (data, variáveis do ambiente, etc) a outro software que formam a base para o sistema. Resumindo, cada porção de dados que alimenta o nosso programa ou sistema deve ser analisado.

Validação dos padrões de dados de entrada

Devemos provar os dados de entrada contra padrões de aceitação e rejeição. Isto pode ser dividido entre diferentes telas, scripts, URLs, etc, para manuseio segundo os recursos (tempo, custos) e pessoas disponíveis. Se recomenda provar todas as interfaces disponíveis, (GUI, API, etc) e não somente as que são expostas ao usuário.

Neste ponto quem realiza os testes deve ter foco em como elaborar os grupos de dados que não somente não coincidam com os padrões estabelecidos, como também que trate de explorar algum elemento proveniente da arquitetura em que o sistema está baseado (sistema operativo, linguagem de programação, base de dados, etc).

Resultados esperados:

- Uma lista de padrões de dados aceitos e rejeitados.

Tarefas a realizar:
Aceitação/rejeição de dados:

- Verificar os dados aceitos que devem ser aceitos.
- Verificar os dados rejitados que devem ser aceitos (falso negativo).
- Verificar os dados aceitos que devem ser rejeitados (falso positivo).
- Verificar os dados rejeitados que devem ser rejeitados.

Trabalho sobre o código:

- Identificar os segmentos de código que manejam cada dado de entrada e sua validação
- Corrigir o código

Referente à correção do código, é somente para esta prova, a chave é não somente corrigir o código para rejeitar os dados nocivos, como também explorar em padrões genéricos. Neste ponto o método usado para filtrar e rejeitar deve ser discutido. O método escolhido depende das limitações do ambiente escolhido, como se explica no “Ingresso de Dados” em El Apéndice C, mas ainda há algo sobre o filtro que se deve mencionar: o método de aceitação e rejeição.

Há dois métodos básicos:

- **Padrão de aceitação:** Os dados ingressados se provam contra padrões de aceitação, fazendo-os mais seletivos segundo se descobrem novas vulnerabilidades. Se nenhum dos padrões é satisfeito os dados são rejeitados.
- **Padrão de rejeição:** Os dados ingressados se provam contra padrões de reijeição, agregando padrões à medida que se descobrem novas vulnerabilidades. Se nenhum dos padrões são satisfeitos os dados são aceitos.

Se prestamos atenção em cada método temos os prós e os contras. O segundo é mais fácil agregar novos padrões, mas requer provas condicionais para cada padrão de rejeição. O primeiro é mais limpo e genérico. Por isso o método do **Padrão de aceitação** é o preferido, basicamente porque se trabalha em reduzir os padrões de aceitação, sendo um esquema mais proativo, mais fácil de manter e menos propenso a erros.
Um exemplo típico dos bugs encontrados com esta prova podem ser encontrados em “Apêndice A” no título “Compromisso Remote”.

Validação dos limites de dados de entrada

Uma vez que os dados de entrada se provam contra os padrões correspondentes, pode ser que os obtidos não sejam tão bons. Por exemplo se um identificador de uma conta bancária é numérica, não todas as combinações, nem todas as

longitudes de conta são válidas. Notar que estes limites não estão relacionados com o significado do mundo real, senão com as implicâncias de segurança quanto a longitude das variáveis. Falando de outra forma , é quando a implementação da tecnologia (linguagens, sistemas operativos, etc) se levam a limites de stress que são desviados de seu comportamento normal (uma forma comum de provar é chamar as funções com parâmetros que estão em ordem de kilobytes e que suponham não devem ser mais que alguns bytes).

Resultados esperados:

- Comportamento do sistema a entradas validadas mas não esperadas

Tarefas a realizar
Monitoramento do Sistema

- Uso CPU (porcentagem, tempo, picos de uso, etc)
- Uso de memória (real, virtual, cache, etc)
- Uso I/O (espaço em disco, acesso em rajadas (simultaneos), etc)
- Uso de red (rejeição de pacotes, generalização, etc)

Trabalho sobre o código:

- Identificar os segmentos de código que manuseiam cada dado que de entrada e sua validação
- Corrigir o código

Neste ponto, a análise e codificação é uma espécie de arte, porque há mais importância no comportamento do sistema do que nos valores devolvidos. Isto requer um amplo conhecimento da tecnologia que se utiliza, principalmente os manuseios internos da linguagem, tais como o manuseio da memória, generalização de processos e threads, bloqueios, IPC, etc.

Um exemplo dos bugs descobertos com este tipo de provas podem ser vistos no “Apêndice A” sob o título “Bug de Format String”. Seu efeito pode ser minimizado se os testes correspondentes sobre o tamanho e formato são realizados.

Processos

Uma vez que nos protegemos do mundo exterior, devemos fazer nossa própria forma de codificação e práticas internas.

Uma análise de como proteger o processo é basicamente a proteção de todos os recursos usados pelo sistema: memória, disco, red, etc, para evitar seu abuso pelo mesmo ou por terceiros.

Consumo de recursos

Cada programa faz uso de recursos (tempo de CPU para cálculos, memória para armazenamento interno e externo, arquivos para armazenamento, etc) para seus propósitos. Sob certas condições é possível fazer que o programa comece um ciclo de consumo que termine em uma caída de sistema (system thrashing) ou uma finalização abrupta (program dumping). Estes são as chamadas negações de serviços (DoS-Denial of Service).

É prática normal realizar os testes somente com os dados provenientes de padrões aceitos, mas é uma boa prática fazer também com os provenientes de padrões rejeitados e variações destes (conhecido como teste “sujo” ou dirtv test).

Resultados esperados:

- Consumo de recursos e uso para cada padrão de entrada e estado (referido a máquina de estados implementada).

Tarefas a realizar:
Monitoramento do sistema: por cada classe de recurso (CPU, memória, etc.)

- Conveniente aquisição de recursos
- Conveniente o uso de recursos
- Conveniente liberação de recursos
- Analise/monitoramento de deadlocks

Trabalho sobre o código:

- Identificar os segmentos de código que manejam cada dado de entrada e sua validação
- Corrigir o código

Para evitar estas condições de DoS é aconselhável limitar a quantidade de recursos usados, e rejeitar pedidos quando se alcaça um nível de consumo de recursos é alcançado.

Um exemplo de bugs descobertos com este tipo de teste pode ser visto em “Apéndice A” sob o título “Retenção de Recursos”.

Teste padrões de uso de recursos

Às vezes ser muito padronizado no uso de recursos (o que significa ter um padrão

de uso) não é bom, basicamente porque ajuda a prever o seguinte movimento a ser feito pelo sistema.

Isto pode ser usado por um atacante para personificar o sistema (oferecendo a mesma resposta que o sistema original, conhecendo o seguinte nome de arquivo a ser usado e o redirecionar para sobreescrever um arquivo suscentível, prever identificadores de sessão, etc.)

Resultados esperados:

- Padrão de uso de recursos

Tarefas a realizar:
Monitoramento do sistema:por cada classe de recurso (CPU, memória, etc.)
Análise de uso de recursos: Buscar padrões no uso de recursos.

Trabalho sobre o código:

- Identificar os segmentos do código que manuseiam cada dado de entrada e sua validação

- Corrigir o código

Prevenção de ataques

Quando se processam os dados obtidos tende-se a crer que os dados disponíveis são seguros, mas...passarão os dados pelos controles de segurança se o programa não é executado sob o ambiente específico, para que ele foi programado?? O que ocorre se um atacante encontrar a forma de executar o módulo de processamento passando por alto estes controles??

Alguns programas fazem adicionais tais como de funcionamento no ambiente indicado e modificação do código.

Resultados esperados:

- Resposta detalhada do sistema sob condicões anomaris de funcionamento.

Tarefas a realizar:
Execução do Sistema:

- Usar pontos de entrada ao sistema distintos dos especificados
- Injeção e modificação do código
- Modificando as condicões de funcionamento

Trabalho sobre o código:

- Identificar os segmentos de código que manuseiam cada dado de entrada e sua validação
- Corrigir o código

A primeira forma de reparação que nos veem à mente é a de incluir um código similar ao de validação dos dados de entrada, mas esta tarefa está reservada aos módulos de verificação de dados, pelo que esta tarefa deve ser duplicada. A tarefa a ser levada em conta é distinta: detecção de condições de execução anormais e não a detecção de dados impróprios.

Saída

Todos os dados de entrada estão filtrados, se tomarão precauções sobre o uso de recursos mas não é suficiente. Que tal ele dizer aos seus competidores a chave de êxito?? Às vezes não é tão drástico, mas para ele dar algumas pistas é somente uma questão de tempo para obter a informação completa.

Algo deve ser posto em conta, que alguns erros podem ser vistos como erros de processo porque produzem informação sutil, produto de uma saída não esperada (**espúrea/spurious)**). O que se trata de mostrar é a passagem de informação que os canais normais de saída, tais como tags de HTML ocultos, erros que podem dar informação sobre a estrutura interna, e probelmas pelo estilo.

Níveis de autorização dos dados

Este conceito é muito claro no uso militar. Em forma resumida, têm-se categorias (ultra secreto, secreto, público, etc) que marcam cada objeto Iinformação, gente, etc) e uma pessoa pode acessar qualquer recurso marcado com o mesmo ou menor nível que o seu (por exemplo, uma pessoa com níevel “secreto” pode acessar recursos marcados como “secreto” e “público” mas não marcá-los”ultra-secretos”). Além de que um recursos tem o mesmo nível que qualquer dos objetos que contém (por exemplo, um documento que contenha dados marcados como “público” e “secreto”; se agrega uma parte “ultra secreta” se converte em um documento “ultra secreto”). O mesmo se aplica aos canais de comunicação, no se pode transmitir um documento “ultra secreto” através de um canal “secreto” (o canal tem menos mecanismos de proteção que o documento pode aceitar).

Este mesmo conceito deveria ser aplicado aqui.

Resultados Previstos:

- Patamares de nives par cada objeto de saída (dados, pessoas, canais, etc..)

Tarefas a realizar:

- Identificação dos segmentos de código que realizam a saída de dados.
- Classificação do assunto que está destinado à saída
- Classificação dos canais de saída
- Classificação dos dados a serem enviados pela saída

Trabalho sobre o código:

- Dados marcantes com níveis de autorização
- Corrigir o código

Às vezes é necessário oferecer uma boa quantidade de informação, e não é sempre uma boa prática, e a mesma contém os erros que nos dá o sistema operacional tais como nome de arquivos, que podem dar alguma pista cuja intenção é não oferecê-la.

Um claro exemplo disto são os Web Servers. Um server MS IIS (internet Information Server) instalado out-of-the-box mostra uma mensagem de erro indicando que a página não existe e mostrando a suposta rota de acesso (file path) ao arquivo (no caso que este não exista) ou a saída de um script em caso de erro (que pode conter estruturas internas de dados ou informação de debugging cuja divulgação não é desejável). Sua contrapartida (Apache) mostra uma mensagem indicando que ocorreu um erro, o administrador será notificado e a informação mais proveitosa é armazenada no arquivo error_log.

Apêndice A – Erros mais freqüentes

Nesta seção as falhas serão mostradas, às vezes em detalhes e outras somente uma idéia das mesmas, principalmente no caso onde há estudos ou papers que a discutem.

Não se pretende que esta seja uma lista exaustiva de falhas, mas sim das mais frequentes e representativas, para poder eleger uma guia quanto a informação, ferramentas e resultados que podem ser encontradas.

Stack smashing

Nesta versão em português foi decidido manter o nome em inglês devido a sua

tradução não ser muito utilizada e há mais de uma nomenclatura do mesmo, também conhecemos como “buffer overflow”.

Será descrita muito brevemente, porque há muitos bons arquivos que as descrevem (ver (5) e (6).

Quando se definem duas variáveis em uma linguagem de alto nível e estão adjacentes no código fonte, há uma grande probabilidade que usem área de memórias adjacentes. Porisso se escrevemos uma destas e se superam os limites, então está sendo escrito na área de memória pertencente a outra. Se a primeira tem 30 bytes, quando se acessa ao byte 31 é na realidade o primeiro byte da segunda variável.

Não parece ser de muito uso, mas façamos uma pequena investigação. Quando se usa uma linguagem de alto nível normalmente usamos sub rotinas (também chamadas funções, procedimentos, etc.) que usam suas próprias variáveis locais. É prática comum que os compiladores usam uma porção de memória, chamada stack, onde é armazenada a direção da próxima instrução a ser executada , chamada direção de retorno, quando a sub rotina termina e também se armazenam a todas as variáveis locais. O que podemos fazer é utilizar esta localização porque se acessamos as variáveis locais podemos ir mais além de seus limites e acessar a direção de retorno .Não se pode fazer magia para adivinhar esta direção, mas sim podemos injetar uma nova peça no código e apontar a direção de retorno a este podendo fazer que ao terminar execute nosso código em lugar do original.

Isto pode ser evitado usando alguns truques tais como inserir alguns testes de integridade para a direção de retorno. Isto também pode ser evitado, mas há menos probabilidades de ocorrência de ataques. Um bom artigo sobre este assunto pode ser lido em (7).

Format String bug

Há funcões e funções...algumas mais flexíveis que outras, mas...a flexibilidade tem seus custos e limitações, uma delas é a segurança, há um conjunto de funções em C (a família dos printf) que um dos parâmetros é um string contendo o formato, e ordem, em que os parâmetros serão passados para a função. Por exemplo, a seguinte chamada a função:

printf (“ola, %s. Nos visitou %u vezes”, nome, contador);

nos diz que os parâmetros são três (o string e as variáveis nome e contador), e a informação relativa a quantidade e tipo de parâmetros pode ser lido desde o primeiro parâmetro (também chamado “format string”): um string (%s) e um número sem sinal (%u).

Agora que este “format string” se encontra como uma constante na aplicação, que há de aproximação entre este string e o mundo exterior?? Tomemos a seguinte porção de código:

```
scanf ( “%s”, *format_string );
printf (format_string, name, counter );
```

onde format_string é equipado por um usuário não verificado e/ou não confiável (através da função scanf). Isto podia permitir ver alguma informação interna, muito útil para obter dados e explorar os mesmos posteriomente (tais como um “stack smashing” ou buffer overflow”) através da manipulacão do string, tais como a troca de %s ou %u por uma forma mais conveniente de ver a informação de memória (por exemplo, usando um contador hexadecimal e declarar o format_string que é um apontamento a um string ou um char). Para uma completa descrição , e exemplos “in the wil”, ver a referência (10).

Compromisso remoto

Este ataque combina dois erros muito usuais; ausência de filtro (ou filtro defeituoso) e execução de um programa com privilégios excessivos.

A continuação se mostra um breve resumo extraído do aviso de segurança indicado em (11):

“O motor de base de dados MS Jet (que permite acessar a bases Access)permite a inclusão para absorver expressões de VBA em seus comandos, o que permite executar comandos de Windwos NT. Isto, combinado com um erro em IIS executando comandos de ODBC como o usuário system_local permite um ataque remoto possuir um controle total do sistema. Outros web servers podem ser afetados. Muitos mecanismo de MS Jet são afetados, mas podem não conduzir a elevação de privilégios”.

Basicamente se pode absorver comandos VBA (Visual Basic for Aplications) dentro de uma sentença de SQL e deixar que ODBC os execute (filtro inadequado), tais como comandos do sistema através de um intérprete de comandos (shell). Isto pode minimizar se é executado com um usuário que tem privilégios mínimos para executar esta tarefa de uma forma eficiente. Este não era o caso.

Não há mais explicações. O Aviso de segurança mencionado é uma obra mestre

que merece ser lida em detalhe.

Retenção de recursos

Quando se inicia uma conexão de TCP a máquina de estados que devem ser seguidas em três passos:

- Requerer a conexão ao extremo remoto
- O extremo remoto responde com uma aceitação (acknowledge), ou a conexão se rejeita.
- O extremo local responde com um acknowledge

Devido a simplicidade de seu mecanismo não possui um controle para evitar a contenção de recursos. Suponhamos que quando seu computador requer uma conexão ao computador remoto ele deixa de funcionar, então o acknowledge será enviado mas o computador local não receberá porque o computador remoto espera até que certo time-out se cumpra e considera a conexão como encerrada, devido a que TCP/IP foi desenhado para trabalhar sobre vínculos não confiáveis e de alta latencia, este esperará durante muito tempo até que considere a conexão fechada (que tenha falhado).

Este usa recursos no computador remoto (principalmente CPU e memória) que podem ser reduzidos a sua mínima expressão sem muitas conexões permanecem neste estado.

Um relatório muito completo pode ser encontrado em CERT, na referência (12).

Apêndice B – Ferramentas

O que vem na sua mente quando se fala de ferramentas. Se a resposta é “programas para reforçar a programação segura” é mais ou menos correto, somente se, não significa somente recursos para usar. Se deve estar preparado para algum tipo de trabalho interno ou a adaptação de algum que outro programa e suas necessidades.

Logging extensivo

Depois de todos estes probelmas a serem resolvidos algo deve estar claro, e é que apesar de tudo o esforço os programas não são perfeitos, e as imperfeições podem e, seguramente vão aparecer. Para descobrir estes problemas devem usar as ferramentas apropriadas tais como debuggers e analisadores de dump, mas às

vezes não são suficientemente úteis, particularmente quando não se sabe por onde começar a busca.

Lembram-se do conto de “Hansel e Gretel”? Iam tirando migalhas de pão durante seu caminho, assim poderiam voltar mais tarde. Nós usaremos o mesmo truque, deixaremos uma marca de cada pedaço de código que é executado junto com os dados mais importantes durante um ataque (ou uma simples falha), de tal maneira que se possa reaver os passos seguidos.

FIX ME: What should and shouldn't (DoS attacks, special chars, passwords,...)

Listas para usar

Isto é uma simples lista das ferramentas conhecidas para assegurar a programação segura, sendo os prediletos os do tipo Open Source.

Nombre: RATS (Rough Auditing Tool for Security)
Resumen: É uma ferramenta de segurança para C e C++que faz uma busca sobre o código fonte, buscando chamadas a funções potenciamente perigosas.
Licença: Versão 2 da Licença Pública GNU (GPL – GNU Public License).
Link: http://www.securesw.com/rats/
Nome: Flawfinder
Resumo: Examina o código fonte buscando vulnerabilidades no código C ou C++.
Licença: Licença Pública GNU (GPL – GNU Public License).
Link: http://www.dwheeler.com/flawfunder/
Nome: ITS4
Resumo: É uma ferramenta simples que que busca no código fonte C ou C++, em forma estatística, por potenciais vulnerabilidades de segurança.
Licensa: ITS4 NO-COMERCIAL para o código fonte
Link: http://cigital.com/its4
Nome: LCLint
Resumo: Prova estatística de programas em C
Licença: Licença Pública GNU (GPL – GNU Public License.)
Link: http://iclint.cs.virginia.edu/
Nome: StckGuard
Resumo: StackGuard é um compilador que generalisa programas mais resistentes aos ataques do tipo “stack smashing” ou “buffer overflow”
Licença: StackGuard é Free Software: é uma melhora a GCC e se distribue sob licença GPL em forma de fontes e binários.
Link: http://www.immunix.org/stackguard.html
Nome: FormatGuard
Resumo: FormatGuard protege programas C e C++contra o “format bug”, principalmente usado na família de funções printf
Licença: FormatGuard é Free Software: é uma melhora a GCC e se distribue sob

licensa GPL em forma de fontes e binários.
Link: http://www.immunix.org/formatguard.html
Nome: RSX
Resumo: RSX é um extensor de espaço de endereçamento em tempo de execução (Runtime addressSpace eXtender) que fornece um remapeamento de código em tempo de execução para binários Linux, que implementa um stack não executável e áreas de heap pequenas e grandes, ataca o problema de "buffer overflow" prevenindo que se execute código nas áreas reubicadas (definidas como somente leitura).
Licença: Licença proprietária que inclue os fontes
Link: http://freshmeat.net/projects/rsx
Nome: PageExec
Resumo: Implementação de páginas não executáveis para processadores IA-32
Licença: ???
Link: http://pageexec.virtualave.net/
Nome: Libsafe
Resumo: Permite evitar "buffer overflow" e "format string", empacotado em forma de biblioteca (library) que intercepta as chamadas conhecidas como vulneráveis
Licença: Libsafe version 20 (código fonte) é licenciadopela licença GNU Lesser.
Link: http://www.avayalabs.com/project/libsafe/index.html

Apêndice C – Falhas

Uma vez que temos navegado no velho costume, perto de **cosat,** necessitamos traçar um mapa mais detalhado. É minha preferência pessoal ir do genérico até o particular, assim que comecemos.

Dados de entrada

Começamos com o básico sobre programas ou manipulação da informação. O esquema típico é obter dados, processar (transformar) e oferecer os resultados: entrada-processo-saída.

Não é grande coisa, mas para evitar funcionamentos incorretos e efeitos colaterais (alguns relacionados com as dependências do sistema) devem ser filtrados para obter os dados apropriados.

Suponhamos que estamos lidando com um sistema bancário onde deve ingressar um número de conta e que deve obter o balanço desta conta, então a entrada (número de conta) deve ter um formato determinado (digamos três números, cinco letras e oito números), os que devem ser validados (provados contra um gabarito de correção) e revisados (rejeitados se o formato não é apropriado ou se a conta não existe).

Neste exemplo as opções do formato da conta são realmente simples, somente não podem dar o balanço de uma conta se esta não existe, mas há problemas mais sutis (que se mostrarão mais adiante) que não são do tipo funcional. Pode se tomar uma regra: todo o que seja usado deve ser evitado.

Há muitas formas de fazer, e cada uma depende dos objetivos de cada sistema:

- **Finalização anormal**: esta é a mais radical; no caso de detecção de um dado não válido o sistema interpreta que está sob ataque (alguém buscando formatos de dados impróprios) e finaliza.

- **Rejeição:** a metade do caminho, quando um dado não é válido é detectado se pergunta novamente pelo mesmo, isto tem uma vantagem para um atacante, e é que pode se perguntar uma quantidade limitada de vezes até que um dado válido é detectado.

- **Utilização:** as partes impróprias são quitadas e somente o que concide com o formato apropriado é usado. Tipicamente se usa dados de longitude variável tais como direções de email, URLs e direções de emails, URLs e direções postais.

Uma linguagem de programação que tem capacidades para a programação segura é Perl. Pode ser executado utilizando o “tainted mode” onde, essencialmente, cada dado obtido desde o mundo exterior é marcado como “sujo” e não pode ser usado até que seja limpo.

Normalmente a máquina que vai “limpar” (filtrar) estes “dados sujos” está composta de filtros implementados como reforço, se estes dados são usados “sem limpos” Perl interrompe sua execução (para uma explicação mais profunda ver a referência (4).

Processamento (A caixa de Pandora)

Às vezes se tem muito cuidado sobre os dados obtidos, mas não é suficiente. Porque não somos perfeitos, cometemos erros, devemos proteger o programa, e o nosso sistema. Recordemos que em qualquer programa há muitos fatores a serem considerados (é tão certo que a análise de confiabilidade de software o comportamento dos programas se estudam em forma estatística) e nem todos devem ser tomados em conta na mesma vez, porque a possiblidade de introduzir erros é realmente alta (esta probabilidade decresce à medida que o programa é provado e corrigido com o correr do tempo).

Não é fora do comum que em todas as empresas e software bem estabelecidas, o uso de ambientes de prova, execução, etc, que simulem as condições de campo (execução em seu destino final) e executando em um ambiente protegido, onde este falhe, retire erros e o que ocorra. O mesmo ocorre em seu destino final, é executando com muita precaução até que se execute como um processo produtivo, sendo seu monitoramento um processo crítico nas primeiras semanas.

A maioria das linguagens modernas de programação como C++, Java, Perl e Phyton contém mecanismos de proteção. Basicamente se executam partes de código em um ambiente protegido, e se pega a qualquer erro (ou excessão) que possa ser produzido pelo programa.

Por exemplo, se pode rodar o código com um bloqueio que o isole de seu ambiente em caso de falha, e devolva o controle a uma porção do código que maneje esteerro.

```
Try()  {
        Code under suspicion ();
                } catch (Exception e)  {
                        System.println “Code_under_suspicion(),
                        Exception generated : “ + e;
        }
```

Agora suponhamos que estamos desenvolvendo uma série de programas de uso crítico, como para medicina, aeronáutica, etc, onde devem continuar correndo ainda em condições de falha.
Imaginemos que durante seu vôo, o programa que segue a rota até o aeroporto envia a seguinte mensagem à torre:

“Excessão em 0x3F745DE9, o sistema está sendo baixado”

Estes programas estão feitos com estas situações em mente, e devem prosseguir sua execução ainda sob condições especiais (digamos que se isole o código que causou o problema, realiza algum tipo de teste sobre o hadware e software, etc).

Para ler uma muito boa revisão sobre excessões aplicado a Java, ver a referência

(8).

Às vezes não é uma boa opção que seja feito desta forma, mas ao colocar algumas barreiras de contenção ao redor do programa pode ser um pouco mais fácil: não se requer programação extra, nem recompilação para sua ativação. Para este caso pode ser usado uma série de ferramentas padrão tais como dump analyzers (analisis post-mortem - análise pós-encerramento) ou debuggers (debugadores) envolvem o programa, e quando se produz um erro este é brecado e manejado pelo debugger. Isto pode imprimir a situação do stack, contidos de variáveis, e oferecer ao programador/operador uma interprete de comandos (shell) que permita a inspeção do programa, seu reinicio, dumping, etc.

Saída de Dados

Falando de efeitos colaterais, temos em mente que sua saída é a fonte de entrada de outro processo? Ou ainda pior... que há perto da sujeira que se produz?

Comecemos pelo princípio. O mesmo problema que temos com nossos dados de entrada se transfere ao processo que manejará nossos dados de saída e os processará. Este processo não manejará dados rigorosos porque nosso programa/sistema atuará como filtro, mas assim mesmo pode gerar problemas (a propósito e sim com certeza) que pode estragar um bom trabalho.

Tomemos um exemplo onde se processam uma série de dados e, como parte do processo de saída, se gera um arquivo. Tudo parece que está correto, tudo está dentro das especificações e estamos seguindo as regras do jogo. Seguro??? Sim por exemplo, geramos um arquivo com um nome fixo, digamos output.data, simplesmente:

```
filehandle = open ( “output.data” , w ) ;
        write (filehandle, data ) ;
        close (filehandle ) ;
```

Seguramente se terá agradado algumas precauções sobre a existência do arquivo, perguntar ao operador se o arquivo pode ser apagado no caso que já exista, fazer um backup, e todo esse tipo de coisas, mas...o que ocorre se a infraestrutura em que nos baseamos foi comprometida e não estamos escrevendo o arquivo que realmente pensamos?? Há um “caminho limpo” ao arquivo??

Além disso, com segurança saberá, uma fonte importante de informação sem fundamento a análise e o recolhimento dos conteúdos de sujeira (depois de tudo, por isso que existem os trituradores de papel). O exemplo mais claro é a resposta para a pergunta “onde você envia suas mensagens de advertências e erros?”. Seguramente haverá notado que em alguns web servers quando ocorre um erro,

tais como uma página não encontrada ou uma falha em um CGI, o erro envia ao browser, e as mensagens aparecem da seguinte forma:

The request CGI program in /home/httpd/cgi-bin/script'failed to execute. The next lines contain the error:

Couldn't connect to database 'Customers in server 'internal_db'.
Send a message to hostmaster@this.domain.com

Que fonte de informação excelente!!! Isto DEVERIA ser redigido ao log de erros ou de alguma outra forma onde podería ser lido pelo administrador/operador. Depois de tudo isso não tem sentido para a maioria dos usuários e essa é uma muito boa fonte de informações para os crackers. Você não pode crer? A parte triste da história é que a maioria das web servers estão configuradas de forma insegura out-of-the-box.

Appendix D - Links

[1] **Title:** Secure Programming for Linux and Unix HOWTO
Author: David A. Wheeler
Location: http://www.dwheeler.com/secure-programs
[2] **Title:** Why Computers are Insecure
Author: Bruce Shneier
Location: http://www.counterpane.com/crypto-gram-9911.html#WhyComputersareInsecure
[3] **Title:** No silver bullet
Author: Frederick P. Brooks, Jr.
Location: http://www.undergrad.math.uwaterloo.ca/~cs212/resource/Articles/SilverBullet.html
[4] **Title:** Perl security
Location: http://www.perl.com/pub/doc/manual/html/pod/perlsec.html
[5] **Title:** Smashing The Stack For Fun And Profit
Author: Aleph One
Location: http://www.phrack.com/search.phtml?view&article=p49-14
[6] **Title:** Avoiding security holes when developing an application
Author: Frederic Raynal, Christophe Blaess, Christophe Grenier
Location: http://www.linuxfocus.org/English/March2001/article183.meta.shtml
[7] **Title:** Bypassing StackGuard and StackShield
Author: Bulba and Kil3r
Location: http://www.phrack.com/search.phtml?view&article=p56-5
[8] **Title:** What's an Exception and Why Do I Care?
Author: Sun Microsystems
Location: http://java.sun.com/docs/books/tutorial/essential/exceptions/definition.html
[9] **Title:** SECPROG@SecurityFocus.com working document
Author: Secure Programming list contributors
Location: http://www.securityfocus.com/forums/secprog/secure-programming.html
[10] **Title:** Analysis of format string bugs
Author: Andreas Thuemmel
Location: http://www.securityfocus.com:80/data/library/format-bug-analysis.pdf

[11] **Title:** NT ODBC Remote Compromise
Author: Matthew Astley & Rain Forest Puppy
Location: http://www.securityfocus.com/archive/1/13882
[12] **Title:** CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks
Author: CERT
Location: http://www.cert.org/advisories/CA-1996-21.html

Open Methodology License (OML)

PREAMBLE

A methodology is a tool that details WHO, WHAT, WHICH, and WHEN. A methodology is intellectual capital
that is often protected strongly by commercial institutions. Open methodologies are community activities which
bring all ideas into one documented piece of intellectual property
which is freely available to everyone.
With respect the GNU General Public License (GPL), this license is similar with the exception for the right for
software developers to include the open methodologies which are under this license in commercial software.
This makes this license incompatible with the GPL.
The main concern this license covers for open methodology developers is that they will receive proper credit for
contribution and development as well as reserving the right to allow only free publication and distribution where
the open methodology is not used in any commercially printed material of which any monies are derived from
whether in publication or distribution.
Special considerations to the Free Software Foundation and the GNU General Public License for legal concepts
and wording.

TERMS AND CONDITIONS

1. The license applies to any methodology or other intellectual tool (i.e. matrix, checklist, etc.) which contains a
notice placed by the copyright holder saying it is protected under the terms of this Open Methodology License.
2. The Methodology refers to any such methodology or intellectual tool or any such work based on the Methodology. A "work based on the Methodology" means either the Methodology or any derivative work by copyright law which applies to a work containing the Methodology or a portion of it, either verbatim or with modifications and/or translated
into another language.
3. All persons may copy and distribute verbatim copies of the Methodology as are received, in any medium,
provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and
creator or creators of the Methodology; keep intact all the notices that refer to this License and to the

absence
of any warranty; give any other recipients of the Methodology a copy of this License along with the Methodology, and the location as to where they can receive an original copy of the Methodology from the copyright holder.
4. No persons may sell this Methodology, charge for the distribution of this Methodology, or any medium of which this Methodology is apart of without explicit consent from the copyright holder.
5. All persons may include this Methodology in part or in whole in commercial service offerings, private or internal (non-commercial) use, or for educational purposes without explicit consent from the copyright holder
providing the service offerings or personal or internal use comply to points 3 and 4 of this License.
6. No persons may modify or change this Methodology for republication without explicit consent from the copyright holder.
7. All persons may utilize the Methodology or any portion of it to create or enhance commercial or free software,
and copy and distribute such software under any terms, provided that they also meet all of these conditions:
a) Points 3, 4, 5, and 6 of this License are strictly adhered to.
b) Any reduction to or incomplete usage of the Methodology in the software must strictly and explicitly state
what parts of the Methodology were utilized in the software and which parts were not.
c) When the software is run, all software using the Methodology must either cause the software, when started
running, to print or display an announcement of use of the Methodology including an appropriate copyright notice and a notice of warranty how to view a copy of this License or make clear provisions in another form
such as in documentation or delivered open source code.
8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not
limited to patent issues), conditions are imposed on any person (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If said person cannot satisfy simultaneously his obligations under this License and any other pertinent
obligations, then as a consequence said person may not use, copy, modify, or distribute the Methodology at all.
If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of
the section is intended to apply and the section as a whole is intended to apply in other circumstances.
9. If the distribution and/or use of the Methodology is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an
explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or
among countries not thus excluded. In such case, this License incorporates the limitation as if written in the
body of this License.
10. The Institute for Security and Open Methodologies may publish revised and/or new versions of the Open
Methodology License. Such new versions will be similar in spirit to the present version, but may differ in detail
to address new problems or concerns.
NO WARRANTY

11. BECAUSE THE METHODOLOGY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR
THE METHODOLOGY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE
THE METHODOLOGY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED,
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE IN USE
OF THE METHODOLOGY IS WITH YOU. SHOULD THE METHODOLOGY PROVE INCOMPLETE OR
INCOMPATIBLE YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY
COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY USE AND/OR REDISTRIBUTE THE
METHODOLOGY UNMODIFIED AS PERMITTED HEREIN, BE LIABLE TO ANY PERSONS FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF
THE USE OR INABILITY TO USE THE METHODOLOGY (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY ANY PESONS OR THIRD
PARTIES OR A FAILURE OF THE METHODOLOGY TO OPERATE WITH ANY OTHER METHODOLOGIES),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.