Faculdade de Tecnologia de Americana
Curso Superior de Bacharelado em Análise de Sistemas e Tecnologia da Informação

ENGENHARIA REVERSA NA SEGURANÇA DE SISTEMAS

Marshall Moshe Mauricio do Nascimento

Americana, SP
2013

Faculdade de Tecnologia de Americana

Curso Superior de Bacharelado em Análise de Sistemas e Tecnologia da Informação

ENGENHARIA REVERSA NA SEGURANÇA DE SISTEMAS

Marshall Moshe Mauricio do Nascimento

Projeto desenvolvido em cumprimento curricular da disciplina Projeto Articulador Bacharelado II do Curso Superior de Tecnologia em Bacharelado em Análise de Sistemas e Tecnologia da Informação da FATEC – Americana, sob orientação do Prof. Rogerio Nunes de Freitas.

Área: Engenharia de *Software*- Segurança de *Software*.

Americana, SP

2013

BANCA EXAMINADORA

Prof. Esp. Rogério Nunes de Freitas
(Orientador)

Prof. Me. Alberto Martins Junior

Prof. Me. Alexandre Mello Ferreira

Americana, 21 de Junho de 2013

Deus Por me dar saúde e sabedoria para chegar até aqui.

A minha família por me dar suporte e condições durante todo o caminho percorrido.

Aos meus amigos pelos momentos que passamos durante todo o curso.

AGRADECIMENTOS

Agradeço ao professor Rogério Nunes de Freitas, por toda compreensão e comprometimento durante toda a orientação, por todos os conselhos e dicas.

Agradeço a minha família, por todo carinho motivação e suporte que forneceram todo o tempo, sendo imprescindível para chegar até.

Agradeço aos meus amigos Josiane Gaia, por ter acreditado todo o tempo e por toda motivação fornecida, todas as broncas e cobranças, pelos conselhos e apoio em todos os momentos. A Natalia Antonino e Tamara por todos os momentos, e pela força que foi um fator importante para o desenvolvimento deste trabalho assim como chegar até o final do curso. Ao Tiago Altomare, Rodrigo Brito, Lucas Custódio e Marcos Cavalcanti por todos os momentos de brincadeiras e companheirismo que proporcionaram e proporcionam boas risadas e momentos descontraídos. E a todos os amigos e conhecidos que de alguma forma foram importantes e contribuíram pra eu chegar até aqui.

Agradeço a todos os professores que de alguma forma contribuíram para minha formação não apenas como aluno mais como pessoa, pois durante esses 4 anos de curso eles me proporcionaram condições para evoluir em vários aspectos.

“O segredo do sucesso é saber algo que ninguém mais sabe”.

(Aristóteles)

RESUMO

Atualmente um assunto em destaque se tratando em Sistema é a segurança, com a divulgação de vários ataques, faz-se necessário proteger os Sistemas, utilizar técnicas e recursos que os atacantes usam para invadir, mais com intuito de proteger é uma tática quando se quer aumentar o grau de segurança dos Sistemas. A engenharia reversa é uma técnica que possibilita conhecer por inteiro um Sistema mesmo quando só se possui o executável do programa, combinando suas técnicas é possível diminuir as chances de invasões e consequentemente se torna mais seguro o Sistema. A engenharia reversa para realizar todas as suas atividades conta com algumas ferramentas e técnicas que conseguem extrair informações tanto de baixo nível quanto a nível de projeto, e com essas informações os profissionais da área possuem uma gama muito ampla de dados relativos ao programa. O base nos resultados dessas ferramentas é imprescindível para determinar o quão seguro um Sistema é não somente o seu executável, mais também o código fonte e também do projeto e toda sua documentação, determinando pontos chaves que podem ser melhorados e explorados de forma a aumentar o nível de segurança e integridade do Sistema.

PALAVRAS CHAVES: Reverse engineering, systems, security, tools, techniques, design.

ABSTRACT

Currently an issue highlighted in the case system is security, with the release of several attacks, it is necessary to protect the systems, using techniques and resources that attackers use to invade more to protect against a tactic when you want increase the degree of safety systems. Reverse engineering is a technique that allows to know whole system even when one only has the executable program, combining their techniques can decrease the chances of invasions and consequently becomes more secure system. Reverse engineering to conduct all of its activities, with some tools and techniques that can extract both low-level information about the project level, and with this information the professionals have a very wide range of data relating to the program. The basis of the results of these tools is essential to determine how secure System is not only your executable, plus also the source code as well as the design and all your documentation, determining key points that can be improved and exploited in order to increase the level of security and integrity of the system.

KEYWORDS: Reverse engineering, systems, security, tools, techniques.

SUMÁRIO

LISTA DE FIGURAS 11

1. INTRODUÇÃO 12

1.1 DELIMITAÇÃO DO PROBLEMA 12
1.2 QUESTÃO PROBLEMA 13
1.3 RELEVÂNCIA 13
1.4 VIABILIDADE 14
1.5 HIPÓTESES OU PRESSUPOSTOS 15
1.6 OBJETIVOS 16
1.7 OBJETIVO GERAL 16
1.8 OBJETIVOS ESPECÍFICOS 16

2 FUNDAMENTAÇÃO TEÓRICA 17

2.1 CONTEXTO HISTÓRICO 17
2.1.1 ATARI CORP CONTRA NINTENDO DA AMERICA, INC. 17
2.1.2 O QUE É ENGENHARIA DE *SOFTWARE* 18
2.2 O QUE É ENGENHARIA REVERSA 19
2.3 O QUE É SEGURANÇA DE *SOFTWARE* 21

3 ENGENHARIA REVERSA 22

3.1 POR QUE USAR ENGENHARIA REVERSA 22
3.2 TÉCNICAS USADAS NA ENGENHARIA REVERSA 23
3.2.1 SEM CÓDIGO FONTE 23
3.2.1.1 ANÁLISE DE FLUXO DE DADOS 23
3.2.1.2 DISASSEMBLER 24
3.2.1.3 DESCOMPILAÇÃO 25
3.2.2 COM CÓDIGO FONTE 26
3.2.2.1 EXTRAÇÃO DAS INFORMAÇÕES 26
3.2.2.2 TRATAMENTO DOS FATOS 28
3.3 FERRAMENTAS 31
3.3.1 *DEBUGGER*S 31

3.3.2 *DISASSEMBLERS* 32
3.3.3 *DECOMPILERS* 33
3.3.4 FERRAMENTAS DE MONITORAMENTO 34
3.3.5 SISTEMAS OPERACIONAIS 37

4 PROCEDIMENTOS METODOLÓGICOS 38

5 CONSIDERAÇÕES FINAIS 41

6 REFERÊNCIAS 44

LISTA DE FIGURAS

FIGURA 1 CICLO TRADICIONAL DE ENGENHARIA DE SOFTWARE 19
FIGURA 2 CICLO ENGENHARIA REVERSA 20
FIGURA 3 OLYDBG DEBUGADOR.....32
FIGURA 4 IDA PRO DISASSEMBLER 33
FIGURA 5 VB DECOMPILER 34
FIGURA 6 FILEMON FERRAMENTA DE MONITORAMENTO DE TRAFICO ENTRE APLICATIVOS E O SISTEMA OPERACIONAL 35
FIGURA 7 REGMON MONITOR DE REGISTROS.....36
FIGURA 8 PROCESS EXPLORER MONITOR DE PROCESSOS - SIMILAR AO GERENCIADOR DE TAREFAS DO WINDOWS.....37

1. INTRODUÇÃO

A engenharia reversa é importante, englobam processos como manutenção e proteção de *software*, que pode evitar invasões, roubos de informações, gastos desnecessários em resolver problemas de SI (Sistema de Informação). Em alguns casos refazer o que existe, é uma ótima forma de aprimorar o Sistema atual, evitando gastos com a compra de um novo, o que pode gerar gastos extras com consultoria e a compra em si de um novo SI.

A engenharia reversa não se aplica exclusivamente a *software*s, mais também a processos de uma companhia ou mesmo em outras áreas além da Informática, sua abrangência e maleabilidade podem facilmente ser adaptadas além da área de origem. A engenharia reversa apesar de ser uma ferramenta poderosa da engenharia de *software* tradicional é pouco usada no Brasil, pois exige um mínimo de conhecimento de Informática (programação, lógica etc.) e isso é um empecilho, pois os profissionais da área não dão tanta importância assim ao tema.

Com ela é possível minimizar os riscos de que o Sistema tenha sua segurança burlada por indivíduos mal intencionados, a indústria de *software* (Sistemas, jogos etc.) sofre com o pirateamento de seus produtos, pois mesmo com medidas de segurança (seriais, chaves etc.) para evitar isso, pessoas com um pouco de conhecimento em programação se aproveitam da engenharia reversa para achar brechas e poder usar o *software* em questão sem pagar pelo *software*.

Atualmente várias empresas adotam a engenharia reversa como método de proteção aos Sistemas a fim de achar o máximo possível de brechas e assim tornar mais difícil que o mesmo seja burlado e com isso garantir não somente a elas mais a seus usuários uma experiência mais segura durante o uso desses Sistemas.

1.1 DELIMITAÇÃO DO PROBLEMA

A engenharia reversa apesar de ser uma técnica poderosa que auxilia na proteção de Sistema, porém ainda não é uma pratica amplamente usada, pois demandam tempo e despesas que poucas empresas possuem condições de adotar

essa ferramenta, além de profissionais qualificados para essa tarefa, atualmente grandes empresas adotam essa técnica para garantir a integridade de seus Sistemas e assim minimizar as perdas com as versões piratas ou invasões de seus produtos.

Permeh e Edwards (2010) afirmam que a o processo de analisar um produto ou algo do final para o inicio e tentar entender como funciona é conhecido como engenharia reversa. E se tratando de segurança de *software* é costume quebrar o produto em partes, e geralmente são as partes executáveis desses programas. Usando essas partes é possível ver claramente como é o funcionamento interno do programa, a engenharia reversa não é apenas usada para invadir Sistemas mais também para analisar e compreender como funcionam os vírus e outros tipos de ameaças.

1.2 QUESTÃO PROBLEMA

Segundo Hoglund e Mcgraw (2006) quando uma máquina – computador, celular etc. – é invadida, na maioria das vezes isso acontece explorando falhas do *software*, e por esse motivo vira seu calcanhar-de-aquiles.

Com base nessa informação a engenharia reversa é uma eficiente técnica para proteção de *software*s?

1.3 RELEVÂNCIA

Em Abril de 2012, Igor Soumenkov (especialista da *Kaspersky Lab*), ajudou na descoberta de uma rede de computadores zumbi, ele explica que foi utilizada a engenharia reversa no *software* malicioso, para produzir e registrar o nome de domínio.

> “Realizamos a engenharia reversa do algoritmo de criação da praga e utilizamos a data atual, 06 de Abril de 2012, para gerar e registrar o nome de domínio ‘*krymbrjasnof.com’* “. (SOUMENKOV, 2012)

De acordo com Permeh e Edwards (2010) ter uma visão ampla é uma vantagem que proporcionada pela engenharia reversa, de fato ela mostra o código executado pela *CPU* (C*entral Processing Unit* ou Unidade de Processamento Central), a quantidade de informação obtida é muito grande e muitas vezes pode-se ver códigos que normalmente ficam ocultos e se o compilador – programa que transforma instruções de uma linguagem de programação (ex., Cobol, Pascal) para uma linguagem que possa ser entendida pela maquina. – está adicionando coisas não desejadas no produto. Ainda segundo os autores, a demora de fazer o caminho oposto é uma desvantagem, mesmo com pessoas excepcionais que são capazes de ler o código-fonte, esse processo gasta muito tempo. Para auxiliar nesse demorado trabalho os programadores – pessoa que desenvolve programas usando alguma linguagem de programação – mais experientes costumam possuir kits de ferramentas feitas para tornar mais rápida as tarefas comuns, dentre as principais ferramentas desses kits há *disassemblers* – Tradutor de programas para o código de máquina – e depuradores – programa de computador usado para testar outros programas.

1.4 VIABILIDADE

No Brasil ainda enxerga a engenharia reversa como algo ruim – *crackers* usam técnicas de engenharia reversa para criar soluções gratuitas de *softwares* pagos (*cracks*) – e a dissertação é extremamente viável, tendo em mente que esta é uma visão errada sobre a engenharia reversa, seu uso incrementa a segurança do Sistema (torna-lo mais difícil de *crackear*, invadir, extraviar informações etc.).

Para Permeh e Edwards (2010), tornar o programador hábil é necessário investir uma porção significativa de tempo adquirindo conhecimento sobre as plataformas e linguagens de “*assembly*” especifica de da plataforma que será

utilizada. Conhecimento aprofundado sobre o funcionamento do Sistema e como é feita a geração de códigos pelas linguagens de programação também são exigidos pela engenharia reversa.

Os autores ainda dizem que quando os programadores estão escrevendo o código, os compiladores produzem um código-objeto, feito de pedaços de código em linguagem de maquina (binário), um *linker* (ligador) liga esses pedaços, criando executáveis e bibliotecas. Durante o processo de dissecar programas, eles possuem o costume de dividir em blocos de código-objeto e isso é conhecido como "*disassembly*".

1.5 HIPÓTESES OU PRESSUPOSTOS

Quando o Sistema em questão é muito conhecido e aceito pelos usuários ou de extrema importância para o funcionamento de uma empresa ou órgão publico, há necessidade de protegê-lo contra qualquer tentativa de modificação (*crackear*) ou invasão, para que assim seus usuários ou clientes não sejam prejudicados por esses ataques de pessoas mal intencionadas.

A invasão ou a pirateamento de um *software* pode gerar grandes prejuízos para as corporações. Informações extraviadas podem ser vendidas para outras empresas e isso torna a concorrência desleal, pois a empresa que adquiri essas informações consegue uma ampla vantagem. A pirataria também é um problema que as empresas enfrentam atualmente, pois alguns usuários não estão dispostos a gastar com as licenças dos *software*s originais e também não estão propensos a usar uma solução gratuita.

O argumento usado para justificar a pirataria, é o alto preço cobrado pelas empresas, usuários de *software* pirata alegam que se o produto fosse vendido com preço mais baixo não haveria necessidade de adquirir um *software* pirata. Entretanto não são apenas usuários comuns que usam de engenharia reversa para descobrir e modificar produtos já existentes no mercado, empresas também adotam esse tipo de técnica para poder aprender sobre o produto de seus concorrentes e assim aprimorar o próprio.

1.6 OBJETIVOS

O seguinte projeto objetiva apresentar algumas técnicas tais como análise do fluxo de dados, descompilação entre outras auxiliam na proteção de Sistemas, minimizando assim os riscos de invasão e modificação em um Sistema.

1.7 OBJETIVO GERAL

Identificar como a engenharia reversa pode ser eficaz em manter a integridade de *software*s, através de estudos de ferramentas como *debuggers* e desmontadores dentre outras, que a engenharia reversa dispõe.

1.8 OBJETIVOS ESPECÍFICOS

- para determinar o quão seguro um Sistema pode ser, e definir meios viáveis a essa proteção.

- para definir como realizar a engenharia reversa em um Sistema de modo que o mesmo evolua e dificulte a invasão por parte de pessoas mal intencionadas.

- para definir quando empregar técnicas de engenharia reversa a fim de demonstrar a mesma como ferramenta viável para solucionar problemas com segurança de Sistemas.

2 FUNDAMENTAÇÃO TEÓRICA

A engenharia reversa não é algo novo, e na área de tecnologia da informação não é diferente o caso da sega x NINTENDO é um exemplo disso, graças a sua grande flexibilidade e eficiência algumas pessoas conseguiram adapta-la se tratando de software conseguindo assim tirar proveito das informações coletadas durante todo o processo.

2.1 CONTEXTO HISTÓRICO

Raja e Fernandes (2008) citam o exemplo da Nintendo vs Atari por quebra de patentes onde foi utilizada engenharia reversa em um produto existente a fim de entender como esse funciona:

2.1.1 ATARI CORP CONTRA NINTENDO DA AMERICA, INC.

A Nintendo entrou com processo contra a Atari, pelos direitos autorais e violação de patentes. A Nintendo era a fabricante de um vídeo game chamado de *Nintendo Entertainment System* (*NES*), e a Atari queria produzir cartuchos que fossem compatíveis com *NES*. O *NES* havia sido projetado com uma complexa chave eletrônica e uma tranca, e com isso apenas cartuchos com essa chave funcionavam no *NES*. Para conseguir essa "chave" a empresa ou individuo interessado tinha de obter a licença de uso com a Nintendo, e então adquirir cartuchos especiais que posssuissem a chave eletrônica da Nintendo, a um determinado valor.

No caso de um jogo que não foi desenvolvido nesses cartuchos, o mesmo não seria executado no *NES*. Usando a engenharia reversa a Atari tentou decifrar a chave criada pela Nintendo tentando chegar ao código binario – linguagem de maquina 0 e 1 – dessa chave de segurança e assim desbloqueiar o Sistema. Porém não obteve sucesso com este método, então ela decidiu monitorar os sinais eletrônicos que eram emitidos pelo cartucho ao *NES* e vice versa com a intenção de descobrir um padrão e com isso revelar o código. Entretanto não obteve o resultado esperado.

Depois de algumas tentativas em vão de burlar o Sistema de proteção da Nintendo a Atari decidiu obter de forma fraudulenta uma cópia do código fonte do *NES* no Escritório de Direitos Autorais afirmando que necessitava do código para usar no conflito sobre direitos autorais com a Nintendo. Com a posse deste código-fonte a Atari abriu o NES e isso permitiu que os jogos produzidos pela Atari fossem executados no Sistema da Nintendo. A Nintendo processou a Atari pela violação dos direitos autorais, o tribunal entrou com uma liminar contra a Atari. No recurso, a Atari argumentou que microprocessador usado por ela era diferente, e a linguagem usada era totalmente diferente na criação de seu programa e, por isso, não houve violação de direitos autorais.

A corte entendeu que não se pode obter patente como forma de proteção, usando uma ideia, processo ou método operacional usando um formato não comprrensivel e fazer alegação de quebra de patente contra quem tenta entender o processo, idéia, ou método utilizado na operação.

2.1.2 O QUE É ENGENHARIA DE *SOFTWARE*

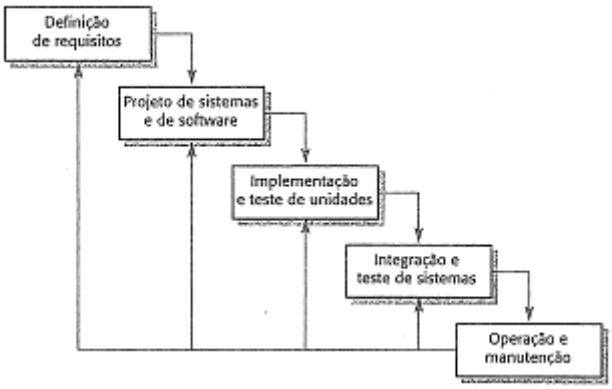
Segundo Sommerville (2003) engenharia de *software* é uma matéria que foca nos aspectos de fabricação de *software*, desde as fazes iniciais de especificações até a manutenção do mesmo. De modo geral engenharia de *software* tem haver com selecionar os métodos mais apropriados para as circunstancias e uma abordagem criativa e informal para desenvolvimento pode ser eficaz em alguma circunstancias.

Pressman (2005) descreve engenharia de *software* como uma tecnologia de camadas, e que deve estar vinculada com gestão de qualidade total, compromisso organizacional, e filosofias semelhantes que conduzem a uma cultura de um processo continuo de aprimoramento, e esse ato permite o crescimento de abordagens que funcionem de fato para a engenharia de *software*. Na figura 2 é demostrado o modelo cascata de desenvolvimento de software uma para a realização de uma etapa depende do fechamento da etapa anterior e apenas na manutenção poderá voltar às etapas anteriores.

Na figura 1 podemos ver mais claramente essas fazes, podemos notar que a comunicação de todas as partes é algo extremamente importante para a satisfação

do cliente e uma comunicação ineficiente ou a falta dela pode causar problemas não só durante o desenvolvimento do Sistema mais em todo seu ciclo de vida.

Figura 1 Ciclo Tradicional de Engenharia de Software



Fonte(Sommerville 2003).

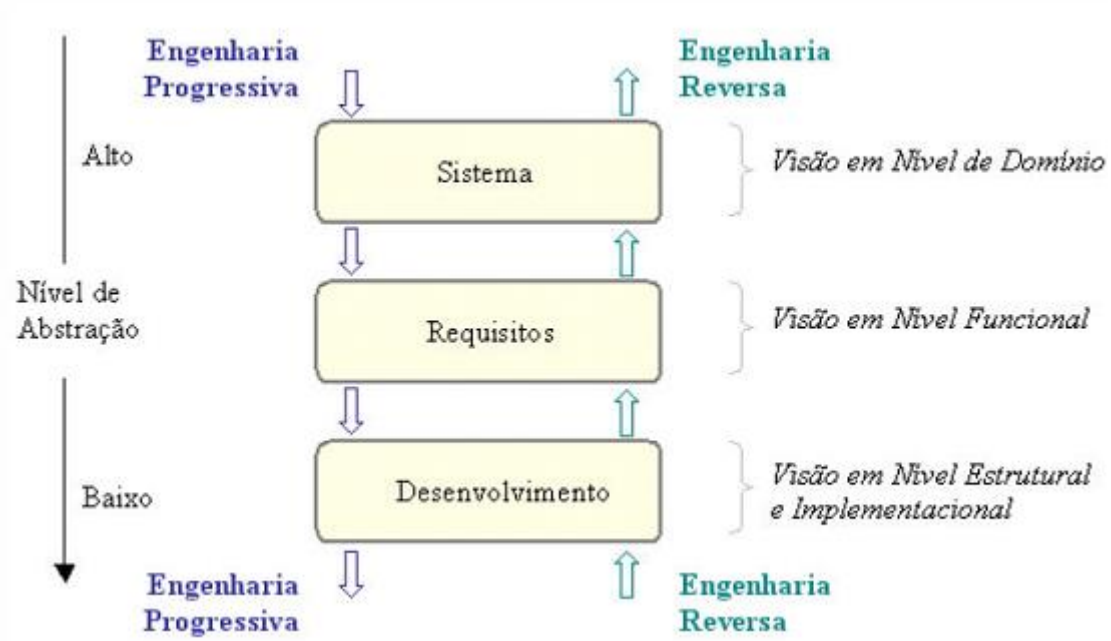
2.2 O QUE É ENGENHARIA REVERSA

Segundo Pressman (2005) a engenharia reversa pode extrair informação do código fonte, mais a completeza da documentação, abstração, a forma com que ferramentas e analistas trabalham e a direção do projeto é variável. Ainda segundo o autor abstração de um processo de engenharia reversa e as ferramentas usadas para efetuar refere-se à sofisticação da informação que pode ser tirada do projeto, o nível de abstração deve ser o mais alto possível. A engenharia reversa deveria poder originar informações do programa e estrutura de dados (baixo nível) e também modelos de fluxo de dados e controle (alto nível).

A completeza da engenharia reversa deve-se ao detalhamento que é fornecido em um nível de abstração. Em vários casos quanto maior a abstração menor a completeza. Pode haver melhora em uma proporção direta com a quantidade de análise realizada por aqueles que fazem a engenharia reversa. A interatividade é o grau que a pessoa está familiarizada com a ferramenta, segundo Pressman (2005). A figura 2 mostra o caminho percorrido pela engenharia de software começando pelos requisitos indo até a manutenção e também mostra o

caminho inverso e nessa parte começa a engenharia reversa, percorrer todo ou parcialmente o caminho contrario a engenharia de software.

Figura 2 Ciclo Engenharia Reversa



Fonte (Moura 2008)

Segundo Sommerville (2003) engenharia reversa é o processo de analisar o *software* objetivando recuperar suas especificações e projeto, o *software* em si continua sem mudanças durante o processo de engenharia reversa. A engenharia reversa difere de reengenharia de *software*, pois a engenharia reversa se preocupa em derivar o projeto ou a especificação de um Sistema a partir de seu código fonte, mais a reengenharia busca produzir um novo Sistema mais fácil de aplicar manutenção. A engenharia reversa é usada durante o processo de reengenharia com o propósito de recuperar a parte de projeto do programa eu os engenheiros utilizam para ajuda-los a entender um programa antes de reorganiza-lo, porém a engenharia reversa não precisa ser sempre seguida de reengenharia.

Segundo o autor Eylan (2005) a engenharia reversa é um processo de descontrução de um artefato de engenharia (como um carro, um motor a jato, ou um programa de *software*) de modo que revele seu design e arquitetura, ou seja, é parecido com uma pesquisa científica que estuda os fenômenos naturais, havendo

diferença que ninguém geralmente refere-se à investigação científica como engenharia reversa, simplesmente porque ninguém sabe ao certo se ou não a natureza nunca foi projetada. Em alguns casos, o código-fonte está disponível, mas os desenvolvedores originais que a criaram não estão disponíveis. Em alguns casos, é possível recuperar código-fonte (ou uma representação de alto nível semelhante) a partir do programa binário, o que simplifica a tarefa de leitura porque o código apresentado em uma linguagem de alto nível é muito mais fácil do que a leitura de um código apresentado em baixo nivel.

2.3 O QUE É SEGURANÇA DE *SOFTWARE*

De acordo com Hoglund, Mcgraw (2006), o bom comportamento de um *software* faz parte de e um processo, e isso inclui identificação e codificação da política, logo após utilizar alguma tecnologia para executar essa política. Não há nenhuma solução simples se tratando de segurança de *software*. Análise de código cada vez mais avançadas é uma boa ferramenta na hora de encontrar erros durante o processo de implementação, mesmo assim nada consegue substituir a experiência.

O investimento em tecnologia de segurança para aplicativos é uma ótima maneira de assegurar que apenas Sistemas aprovados sejam executados, porém não é boa se tratando de encontrar brechas em *software*s. Durante o final da década de 1990, aconteceu uma grande expansão no mercado de segurança, e muitas "soluções de segurança" foram comercializadas. Mesmo com o investimento em segurança, utilizando vários produtos, isso não conteve o aumento dos ataques a *software*s.

3 ENGENHARIA REVERSA

Neste capitulo, será abordado o porquê de se usar engenharia reversa bem como usa-la na como ferramenta auxiliar para a proteção de softwares, visando assim mostrar o quanto ela pode ajudar desenvolvedores e empresas a evitar fraudes ou pelo menos dificulta-las.

3.1 POR QUE USAR ENGENHARIA REVERSA

Para Raja e Fernandes (2008) listam alguns motivos para se usar engenharia reversa:

• Quando o fabricante não existe mais, e o cliente necessita do produto.
• O produto caiu em desuso e o fabricante não o disponibiliza no mercado.
• A ausencia ou inexistencia de documentação do projeto.
• Identificar e reforçar os pontos fortes do produto basedo no uso em longo prazo.
• Examinar o produto do concorrente a fim de identificar pontos fortes e fracos
• Estudar caminhos diferentes visando à melhoria não só dos produtos mais também dos recursos.

De acordo com Eylan (2005), *antireversing* – metodo que busca previnir o uso de engenharia reversa – na maoria das vezes faz sentido, independente da aplicação que está sendo desenvolvida, e o código não é aberto, você deve levar em conta utilizar alguma forma de *antireversing* no Sistema. Porem, nem todo *software* é vantajoso o uso dessa técnica. Em alguns casos o código é relativamente simples, e nesse caso é muito mais fácil refazer o *software* do que usar a *antireversing*.

Em algums casos se fazem necessários o uso de medidas *antireversing*. Um bom exemplo é a tecnologia de *copyrights*. Evitar ou dificultar reversores – pessoas ou ferramentas que buscar fazer o caminho inverso – olhar dentro das cópias é muitos casos uma importante etapa no processo de criar um meio que funcione de fato na proteção.

Ainda segundo Eylan (2005) além do mais, algumas plataformas de desenvolvimento efetivamente precisam de algum meio de usar medidas *antireversing*, porque de outra forma existe a possibilidade do Sistema ser de maneira facil transformado em uma reprodução com pouca diferença do código-fonte. E isto é um fato na maior parte das plataformas baseadas em bytecode – forma intermediária de código –, como Java e *.NET*, e por isso foram criados ofuscadores de códigos para estas plataformas (ainda que seja possível ofuscar os *software*s compilados em uma linguagem nativa de maquina). O obscurecimento é um instrumento automatizado que dimninui a possibilidade de leitura de um *software*, e com isso fazer modificações ou eliminar algumas informações a partir dele.

3.2 TÉCNICAS USADAS NA ENGENHARIA REVERSA

Para a realização da engenharia reversa é necessário conhecer algumas técnicas bem como as ferramentas que permitem e eficiências delas, juntando isso com conhecimentos intermediários em Sistemas Operacionais, proporcionam um conjunto fundamental para a realização da engenharia reversa.

3.2.1 SEM CÓDIGO FONTE

Para Canhota Junior (2005) existem vários jeitos de efetuar Engenharia reversa de software. E esse jeitos estão divididos em 3 (Três) grupos básicos da engenharia de software.

3.2.1.1 ANÁLISE DE FLUXO DE DADOS

Analisar observando a troca de informações que utilizam “analisadores de bus” assim como “pacotes de sniffers”, como exemplo, para conseguir "ouvir" dentro do bus – conjunto de ligações físicas (cabos, pistas de circuitos impressos, etc.) que podem ser usadas em conjunto por vários elementos físicos a fim de comunicar – de um computador ou através de uma conexão de rede, trazendo a tona assim o tráfico

de dados "escondidos". A maneira que os dados transmitidos via bus ou rede se comporta podem ser analisados e depois utilizados para fazer uma nova implementação do software que imitará esse comportamento observado. Essa técnica é muito utilizada durante o processo de engenharia reversa para drivers de dispositivos de acordo com Canhota Junior (2005).

De acordo com Willian (2009), análise de fluxo de dados busca conseguir informações de como o programa em questão manipula os dados. Isso acontece inspecionando o grafo do fluxo de controlo, examinar esse fluxo de valores através de um programa. Existe também varias métodos e abordagens que podem ser usadas, dentre todas as mais comuns: a abordagem iterativa, o método que utiliza árvore de controle através da análise estrutural e método baseado em árvore de controle utilizando análise de intervalos.

Para Willian (2009) dentre os métodos listados anteriormente o que se faz mais simples de implementar é a abordagem iterativa, é nesta abordagem que é feita a definição das equações que irão modelar o fluxo de dados e fazer repetições nelas até não haver dados como resultado. Mesmo os outros modos sendo mais difíceis de usar do que a analisa iterativa, porém, alterações no código são bem vistas e uma adaptação ao modelo que foi gerado se torna mais fácil de ser realizada, devido ao método de repetição da análise iterativa fazendo com oque o algoritmo seja executado de novo. Análise de fluxo de dados pode gerar também um grafo como forma de saída, o GFD (Grafo de Fluxo de Dados), neste as linhas são o fluxo de dados de um bloco mais básico para outro e ainda pode-se gerar um grafo de cada variável, se houver a necessidade de observar uma variável em particular de acordo com Willian (2009)

3.2.1.2 DISASSEMBLER

Para Canhota Junior (2005), utilizando uma ferramenta chamada disassembler (esse tipo de ferramenta será explicada mais adiante) é possível obter o código do programa em linguagem de baixo nível – assembly. O código fornecido

pela ferramenta é lido e compreendido em seus próprios termos, contando apenas com o “mneminics” – recurso utilizado para associar um conjunto complexo ou extenso de informações a algo que seja simples EX: o DOS utiliza o comando dir (de directory) para listar arquivos e diretórios – da linguagem de máquina. E isso funciona em todos os programas de computadores e pode consumir algum tempo principalmente pessoas que não estão habituados com o código de maquina.

De acordo com Eylan (2005) um disassembler é uma importante ferramenta para todo reversor. Um disassembler é capaz de decodificar o código binário de maquina (que é apenas um monte números) em código legível em linguagem assembly. Isto parecido com o processo que acontece dentro de uma CPU em tempo real enquanto o programa ainda está em execução. O que difere é o fato de o disassembler não executa as tarefas estabelecidas pelo código igual um processador faz, a ferramenta decodifica todas as instruções e cria uma representação em forma de texto para cada instrução. Mais também é desnecessário dizer que o formato da codificação é especifico assim como a epresentação textual gerada também é totalmente especifica da plataforma em questão. Cada plataforma fornece suporte à um conjunto distinto de instruções e possui um conjunto diferente de registros. Sendo assim um disassembler também é especifico mesmo havendo alguns que consigam fornecer suporte a mais de uma plataforma.

3.2.1.3 DESCOMPILAÇÃO

Segundo Canhota Junior (2005), através de programas denominados *Decompilers* é possível recriar o código escrito em uma linguagem de baixo em um código em uma linguagem de mais alto nível tornando assim mais fácil o entendimento por parte do reversor.

Para Eylan (2005), descompiladores são instrumentos que tenta reproduzir um código-fonte de um Sistema binário para um código-fonte em uma língua de alto nível. Porém, é impossível recuperar o código-fonte original em sua forma exata, por

que durante o processo de compilação algumas informações são removidas, essa quantidade de informação que é retirada no executável do programa varia de acordo com linguagem de alto nível usada assim como da linguagem de baixo nível que o compilador está traduzindo e também do próprio compilador utilizado.

Alguns programas feitos em .NET (*dot net*) compilado para MSIL (*Microsoft Intermediate Language*) com frequência podem ter bons resultados em sua descompilação (partindo do principio que nenhuma técnica de ofuscamento de código foi utilizada nesse programa). Para códigos baseados em IA-32 (*Intel Architecture, 32-bits)* a situação se torna mais complicada, pois executáveis que utilizam IA-32 possuem pouca as informações de alto nível, e recuperar de forma decente uma representação em alto nível a partir deles são nos dias de hoje impossível.

3.2.2 COM CÓDIGO FONTE

Como auxilio a engenharia reversa ainda conta várias técnicas que podem ser muito uteis, quando se tem acesso ao código fonte do Sistema em questão.

3.2.2.1 EXTRAÇÃO DAS INFORMAÇÕES

Para Canhota Junior (2005) a primeira coisa que deve ser feita é a coleta de informações do Sistema que vai ser estudado. Com base nas informações obtidas é possível usar a engenharia reversa e as informações podem ser extraídas de arias formas como por exemplo usando o próprio código fonte da aplicação, usando os dados do próprio banco de dados, através da documentação da aplicação dentre outras formas.

- **Código (análise estática)**

De acordo com Canhota Junior (2005) a primeira fonte a ser usada é o código e também é a mais utilizada. Essa análise do código também é denominada de

análise estática (por se contraria a análise dinâmica que será explicada mais a frente). Com ela é possui obter informações básica sobre o Sistema

- Componente básico que o Sistema possui tais como arquivos, rotinas, tipos, variáveis, classes, etc.;
- Mostra o relacionamento entre componente e conteúdo (onde se encontram).
- Relacionamentos de referência que conecta um componente com os que o utilizam (uma rotina A invoca alguma outra rotina B. A rotina A possui dependência da rotina B, porque se houver uma modificação na rotina B isso pode causar consequências na execução da rotina A).

Os tipos de ferramentas que são usadas para fazer esse tipo de análise são chamados de "*parser*". O "*parsing*" é considerado o primeiro passo da Compilação do código fonte. Para realizar isso, é preciso possuir conhecimento da sintaxe da linguagem de programação que foi usada. De acordo com as necessidades, usar *parsers* específicos que irão pesquisar um tipo especifico de informação. Como exemplo, em um programa desenvolvido em Pascal, conseguimos obter o nome de todas as funções utilizadas pelo programa segundo Canhota Junior (2005).

- **Trace de execução (análise dinâmica)**

Segundo Canhota Junior (2005), utilizando a análise estática é possível obter várias informações sobre um Sistema, mais é importante lembrar que não são todas as informações que são possíveis obter com essa técnica. Em um comando IF para saber qual parte dele é de fato usada, isso depende de quais dados foram usadas para chamar o programa. Esse tipo de informação é obtida através da análise dinâmica, que é basicamente rodar o programa e ficar monitorando as variáveis e seus valores, funções chamadas entre outras coisas.

- **Dados**

A base de dados também pode ser usada para conseguir informação e auxiliar o processo de engenharia reversa em um Sistema**.** Porém independentemente do Sistema que consiga manipular os dados, ainda pode-se usar a engenharia reversa de dados, pois se trata de um trabalho muito especifico.

Para demonstrar isso, considere que um antigo banco de dados sobre um *mainframe* – computador de grande porte, dedicado ao processamento de um volume gigantesco de informações – e é necessário fazer uma conversão dessa base de dados para uma versão relacional e distribuída sobre diversos PCs – *Personal Computer* – de acordo com Canhota Junior (2005).

- **Documentação**

Para Canhota Junior (2005), a documentação é tudo aquilo que não está em uso pelo computador para que o Sistema funcione, mais é utilizada pelos engenheiros que usam o código: diagramas gerados pela análise ou pelo projeto, relatórios, comentários feitos no código entre outros documentos. Por ela ser feita para humanos e não maquinas é difícil automatizar esse processos. Usando essa abordagem é possível obter conceitos muito importantes sobre o Sistema e isso é baseado na sobreposição de que os conceitos importantes que com mais frequência aparecem durante toda a documentação.

3.2.2.2 TRATAMENTO DOS FATOS

Canhota Junior (2005) o tratamento de fatos é uma das principais atividades englobadas pela engenharia reversa de Software, esse tipo de atividade usa todas as informações que foram obtidas no Sistema, pense em tratamento de fatos como uma tentativa de abstrair informações no nível mais alto dos fatos do Sistema, ou seja inúmeros detalhes serão descartados, gerando assim um problema, dentre todos os fatos quais são importantes e quais podem ser descartados.

- **Anomalias no código**

Em Sistemas legados (antigos), no código pode haver várias deformidades, como algumas partes do código que não poderão ser executadas (código morto) assim como copia de partes do código que foram superficialmente modificados para suprir alguma necessidade. Esses tipos de deformidade deixam o código maior bem

como algumas partes inutilizadas, deixando assim o estudo por parte do programador mais extenso e complexo, nesses casos remover as partes que contenham o código morto e comentando o código deixando explicito que há copias do trecho em questão em outra parte do código ajudam durante o processo de estudo desse código (Canhota Junior,2005).

- **Encapsulamento**

Essa técnica pode ser classifica mais como parte da reengenharia de software do que engenharia reversa, ela consiste em ocultar o código antigo em uma nova camada ao invés de tentar reestrutura-la de acordo com Canhota Junior (2005).

De acordo com Sintes (2002), o encapsulamento é a divisão de um software em partes independentes, que possuem implementações próprias e realiza sua função sem sofrer interferência das outras, essa independência é mantida através do ocultamento do código, que só é possível acessa-las utilizando alguma interface externa.

- **Slicing**

Para Canhota Junior (2005) Slicing ou fatiar é uma técnica que decompõe o código conforme as variáveis em questão são utilizadas, utilizar essa técnica em uma parte do código nada mais é do que extrair toda e qualquer instrução que tiver influencia no valor em uma variável definida em um ponto especifico do código e isso pode auxiliar na identificação de erros no Sistema, fazendo com que a varredura atrás de erros seja feita somente nos partes do código que realmente são necessárias

- **(Re-) Modularização**

De acordo com Canhota Junior (2005) uma definição para (re)-modularização é o processo de separação de várias partes do Sistema em partes menores (também conhecidos como módulos)**.** Normalmente na engenharia de software é esperado que todos os módulos possuam uma coesão interna muito forte e no

exterior um acoplamento. Esse agrupamento de arquivos em subSistemas não é uma pratica comum, em um Sistema, por exemplo, pode ser feito de vários arquivos, escritos em várias linguagens de programação (scripts, código assembly, criados para serem compilados e executados.).

Para realizar o agrupamento de componentes são usados algoritmos que são capazes de medir as distancias que existe entre os componentes, usando como base todas as informações extraídas do Sistema. Geralmente os resultados obtidos com essa técnica são satisfatórios, porém alguns módulos podem conter alguns detalhes errôneos, esse tipo de algoritmo é bom quando se deseja tratar vários componentes, e também é uma das primeiras abordagens usadas no desbravamento de um Sistema.

- **Reconhecimento de Clichés**

Para Canhota Junior (2005), um clichê pode ser considerado um padrão que traça uma visão mais ampla de como codificar um conceito de programação. O reconhecimento de clichês conta com uma base de clichês e o que se faz é um procura no código buscando identificar esses clichês. A ideia básica por traz é que um clichê é implementado usando várias linhas de código e o reconhecimento pode ajudar na simplificação do código por que ocorre a substituição do conceito pelas suas respectivas linhas de código. Mais para se fazer esse reconhecimento o responsável enfrentará várias dificuldades, dentre elas:

- A construção da base de clichês.
- Dualidade, precisão e cobertura (um clichê deve ser descrito de modo geral, mais há o risco de falsos positivos).
- Tempo de execução;
- Clichês interligados (Pode haver uma mistura entre as instruções que implementam um ou mais clichês).

3.3 FERRAMENTAS

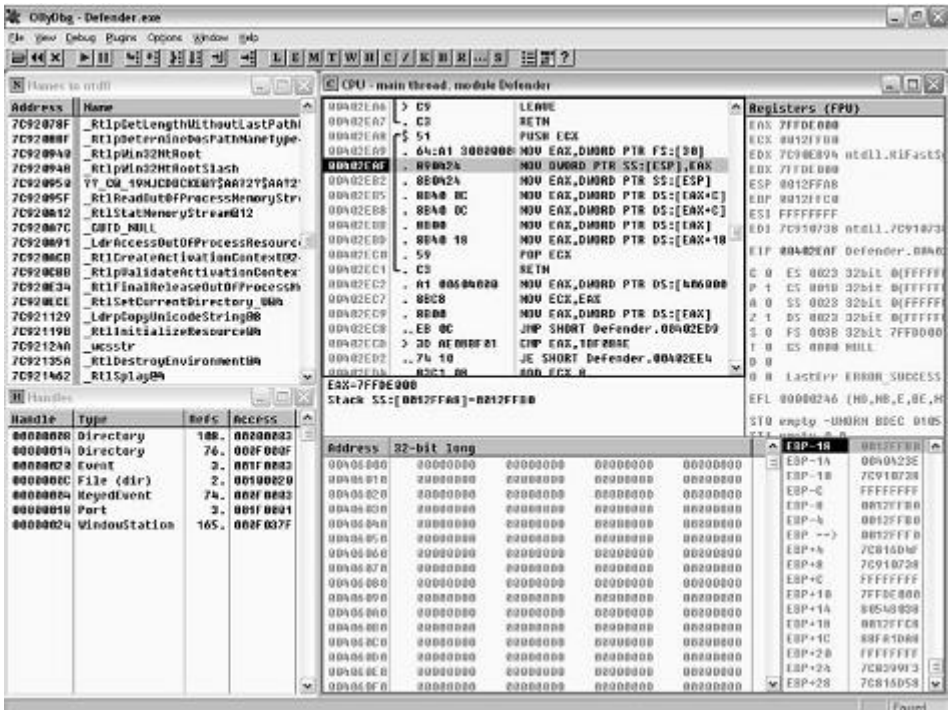
Nessa seção serão abordadas algumas ferramentas essenciais para todos os interessados em realizar engenharia reversa tanto na proteção quanto na invasão de Sistemas.

3.3.1 *DEBUGGERS*

Para Eylan (2005), um programa *debugger* ou depurador é a ferramenta que possibilita a quem desenvolve um software olhar o seu Sistema durante sua execução. E esse tipo de programa possui duas caracteristicas essenciais, ele consegue definir pontos onde o Sistema será interrompido, e rastrear por entre o código. Breakpoints possibilitam que o usuário escolha qualquer função ou linha do código em qualquer parte do código e o depurador irá executar até o ponto determinado, quando ele alcança esse ponto ele para (da um break) e mostra a situação atual do programa, e quando isso acontece deve-se tomar a decisão de continuar a depuração ou não.

O depurador para quem vai invadir um Sistema é tão importante quanto uma *IDE* (*Integrated Development Environment* ou ambiente integrado para desenvolvimento), pois o *debugger* possui um modo de desmontagem, quando se usa esse modo, a desmontagem acontece em tempo real. Do mesmo modo que a depuração realizada pelos desenvolvedores, os invasores podem colocar pontos de paradas em locais que for mais interessante no código desmontado, e depois ver como o Sistema está, segundo Eylan (2005). Na figura 3 podemos observar o SplineTech um debugador de Java Script em ação.

Figura 3 OlyDbg Debugador



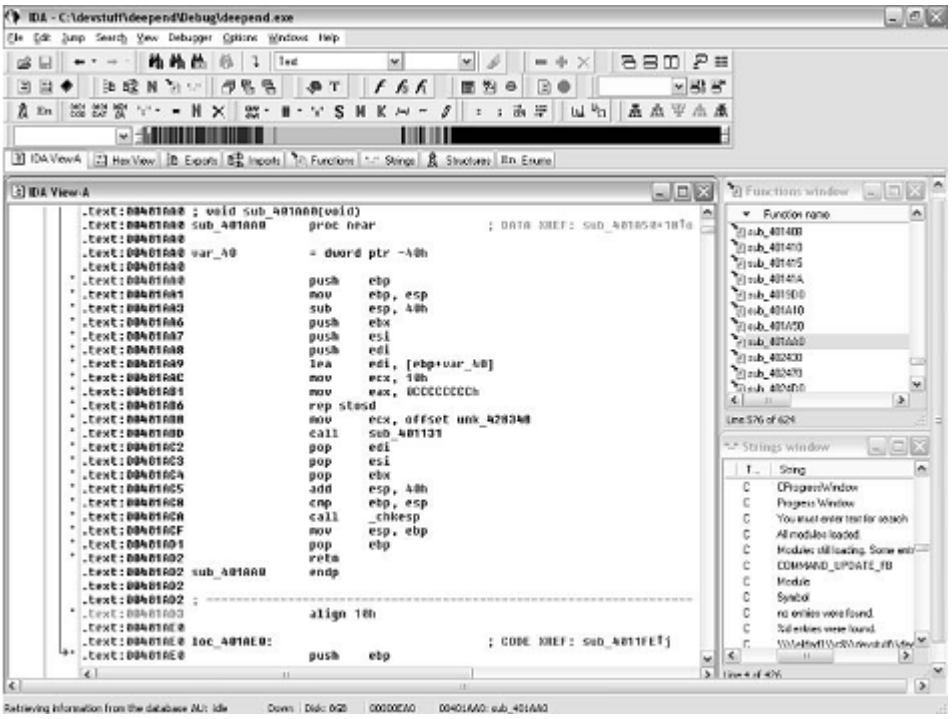
Fonte (Eylan 2005)

3.3.2 *DISASSEMBLERS*

Para Eylan (2005) *Disassemblers* basicamente são programas que transformam o executavel – que nada mais é do que um programa escrito em código de maquina – em arquivos de texto que possuem as linhas de código do programa em questão. Levando em conta que o código *assembly* é mapeado e transformado em código fonte. Esse processo de desmontagem é especifico para cada processador, mais algumas ferramentas possuem suporte a muitas arquiteturas de CPU. E possuir um *disassembler* com alto desempenho é algo essencial para um invasor mesmo hávendo preferencia apenas por *debugger*s, e usam os *disassembler*s embutidos que alguns debugadores possuem. Na figura 4 pode-se ver como funciona um disassembler, uma ferramenta muito util no processo de engenharia reversa.

.

Figura 4 Ida Pro Disassembler



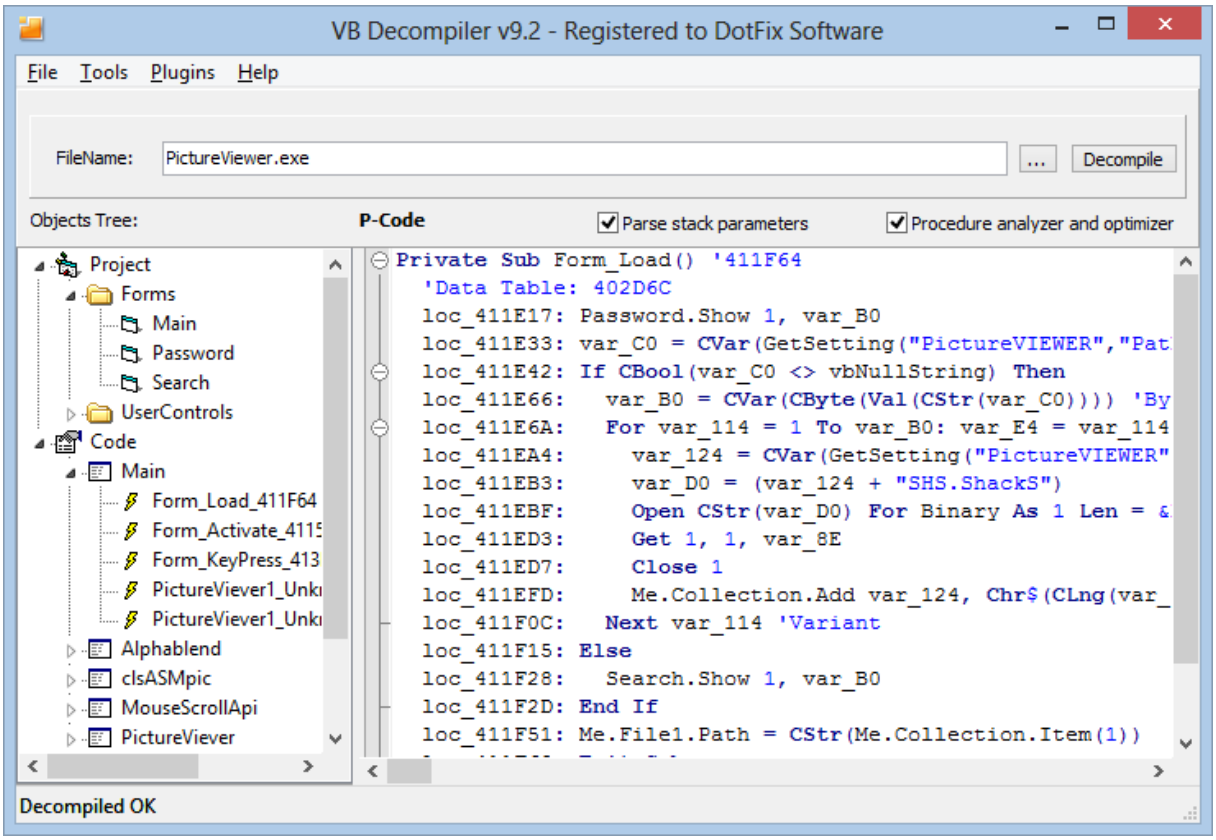
Fonte(Eylan 2005)

3.3.3 *DECOMPILERS*

*Decompiler*s ou descompiladores é próximo passo depois dos *disassembler*s, os descompiladores pegam um código binário e o transformam em um código de alto nível possibilitando assim sua leitura. O plano é reverter o processo de compilação e assim conseguir o código original ou algo semelhante a ele.

Em muitas plataformas, nos dias de hoje a recuperação do código fonte não é algo possível. A omissão de elementos importantes nas linguagens de alto nível torna o processo de recuperação impossível. Entretanto há ainda alguns descompiladores poderosos que em alguns casos pode reconstruir o código de alto nível e legível a partir do programa binário, segundo Eylan (2005). Na figura 5 é retratado o VB Decompiler, assim como é sua interface.

Figura 5 VB Decompiler



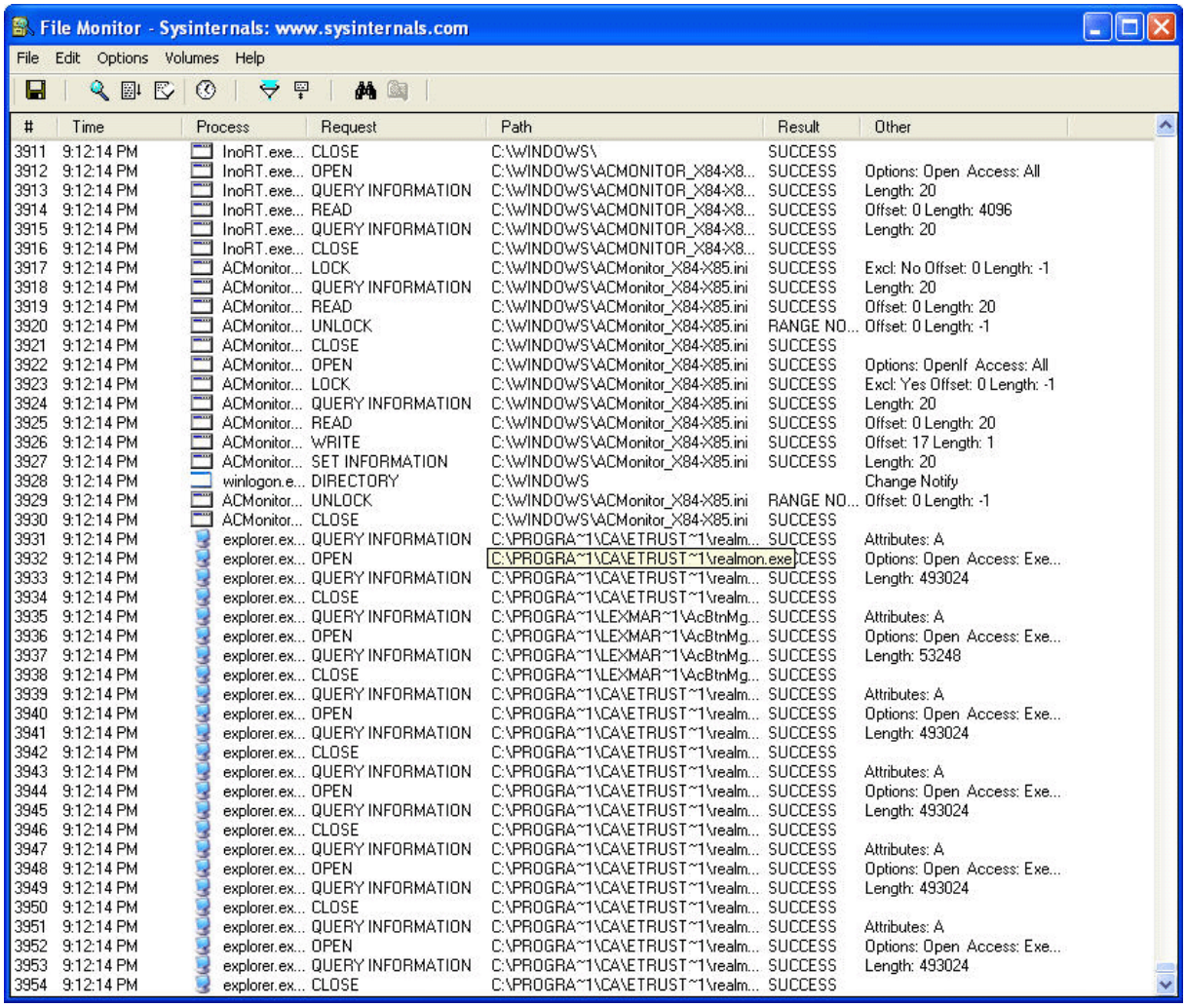
Fonte (http://www.vb-decompiler.org/products.htm)

3.3.4 FERRAMENTAS DE MONITORAMENTO

Segundo Eylan (2005) monitorar o Sistema é um importante passo para a inversão, mesmo sem ter visto o código fonte, em alguns casos o monitoramento pode auxiliar a entender e tirar duvida. Em geral há uma categoria de ferramentas para observar entradas e saida (I/O) em aplicativos ou Sistemas Operacionais. Esse tipo de ferramenta monitora e mostra todas as operações em um arquivo – tais como leitura e escrita, criação de um novo arquivo etc. – e isso feito a partir dos aplicativos do Sistema. Ligando algun tipos de componentes de mais baixo nivel no Sistema operacional e monitorando toda e qualquer tipo de chamada é feito através das aplicações mais relevantes ao reversor. A seguir algumas ferramentas de monitoramento:

- **Filemon:** Esta ferramenta monitora todo o trafico entre aplicações e o Sistema operacional e pode ser usada para visualizer todo arquivo de entrada e saida gerado por cada processo rodando no Sistema.

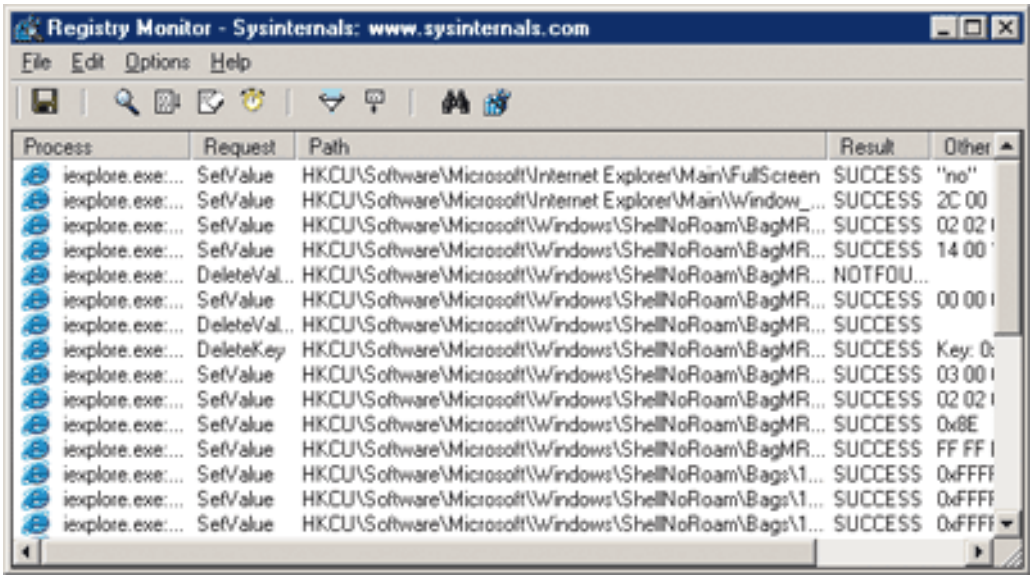
Figura 6 Filemon ferramenta de monitoramento de trafico entre aplicativos e o Sistema Operacional



Fonte (http://support.microsoft.com/kb/890960)

- **RegMon:** Este monitor reporta todos os registros acessados por cada programa, é muito útil para localizar registros chaves e configuração de manutenção de dados de alguns Aplicativos específicos.

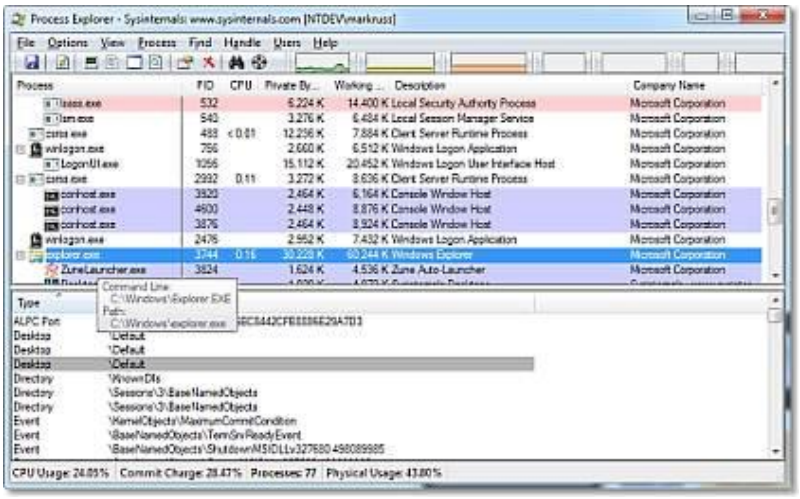
Figura 7 RegMon monitor de registros



Fonte(http://www.devmedia.com.br/revista-msdn-magazine-edicao-03-proteja-as-strings-de-conexao-a-banco-de-dados-e-outras-configuracoes-sigilosas-em-seu-codigo/4251)

- **Process Explorer:** É basicamente uma versão melhorada do Gerenciador de Tarefas do Windows, e foi criado para substitui-lo. Esse aplicativo consegue mostrar processos, DLLs, informações muito detalhadas sobre as conexões de rede, CPU além de gráficos sobre o uso de memória. Essa ferramenta consegue mostrar também alguns detalhes relativos ao código do Sistema como pilhas do kernel de cada *thread* em cada processo.

Figura 8 Process Explorer monitor de processos - similar ao gerenciador de tarefas do Windows.



Fonte (http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx)

3.3.5 SISTEMAS OPERACIONAIS

Para Eylan (2005), um Sistema pode ser considerado como um programa que gerencia o computador, tanto o hardware quanto o softwares. O Sistema Operacional é responsável por tomar conta de muitas coisas diferentes, e pode ser visto como um intermediário entre os vários elementos de um computador. Sistemas Operacionais é um dos elementos principais para qualquer um que queira realizar a Engenharia reversa, indivíduos que já praticam a engenharia reversa possuem um grande conhecimento sobre o funcionamento de um Sistema Operacional. Várias técnicas de engenharia reversa possuem uma forte ligação com ele, pois serve como um portão entre a aplicação e o meio externo.

4 PROCEDIMENTOS METODOLÓGICOS

Neste capitulo será apresentada todos os modos de pesquisas realizados para o desenvolvimento deste trabalho, bem como responder aos objetivos descritos no capitulo 1.

MODALIDADE DE PESQUISA

A pesquisa realizada visa apresentar dados sobre uma das técnicas de engenharia reversa que apesar de eficaz não é largamente utilizada no ciclo de segurança de *software*

A pesquisa quanto aos objetivos caracteriza-se como explicativa, pois os conteúdos observados mostram informações concretas e apresentadas em diversos livros. Em relação aos procedimentos da investigação, a pesquisa tem o caráter documental, pois os relatos apresentados foram buscados através de pesquisas em livros e conteúdo da internet.

Segundo Eylan (2005) *Software* é uma das tecnologias mais complexas e intrigantes hoje em dia, e engenharia reversa de *software* fala comparada como abrir "caixa". Assim como engenharia de *software*, engenharia reversa é um processo virtual, que envolve apenas uma CPU e uma mente humana. Engenharia reversa necessita de algums requisitos como uma combinação de habilidades e uma alta compreensão de computadores e desenvolvimento de *software*, Engenharia reversa integra várias artes: análise de quebra de códigos, resolução de quebra-cabeça, programação e lógica. O processo é utilizado por uma variedade de pessoas diferentes para uma variedade de fins diferentes, muitos dos quais irá ser discutida em todo o livro.

CAMPO DE OBSERVAÇÃO

Os dados apresentados por Eylan (2005) demonstra como utilizar a engenharia reversa e o modo como obter informações do *software*. Ele explica métodos de como quebrar código fonte e assim conseguir ter uma visão macro do funcionamento do programa.

Utilizando como base o material utilizado durante todo o processo de pesquisa e investigação, é possível observar que as técnicas apresentadas neste trabalho auxiliam a descobrir não apenas brechas no Sistema, mais também aprofunda o conhecimento sobre seu funcionamento comunicação com outros arquivos ou processos assim como o fluxo de dados interno e externo que é feito durante sua execução. Através da análise do fluxo de dados, desmontagem e utilizando ferramentas que monitoram processos, arquivos e registros do Sistema e também do Sistema Operacional.

Com todas essas informações em mãos é possível decidir o que é realmente necessário ou não, assim como será feito a comunicação entre módulos, arquivos e as próprias variáveis internas, minimizando os riscos durante um processo de invasão. Tendo em vista que a pessoa que atacar o Sistema não terá acesso a documentação ou mesmo o código fonte, é interessante utilizar algumas técnicas que necessitam desse mesmo código pois falhas e brechas a nível de projeto acarretam e meios e facilidades de invasão ao Sistema depois de pronto. Durante o processo de pesquisa e também foi apresentado neste trabalho a técnica de encapsulamento que é basicamente guardar um método ou função em uma capsula, e depois de feito isso não é possível ter acesso ao que está dentro da capsula, você consegue utilizar a capsula mais não abri-la.

INSTRUMENTOS DE COLETA DE DADOS

As informações apresentadas têm como fonte *sites* de editoras, instituições de ensino e através de livros que falem sobre engenharia reversa cujo um dos autores que será usado é Eylan.

CRITÉRIOS PARA ANÁLISE DOS DADOS

Os dados coletados foram elaborados por autores renomados na área que cujas publicações são referencia para vários trabalhos acadêmicos e livros de engenharia reversa e artigos de instituições de ensino renomados no cenário acadêmico nacional.

DESCRIÇÃO DAS ETAPAS DA INVESTIGAÇÃO

Para realizar as etapas de investigação o autor procurou por material didático (livros, artigos etc.), discussão com professores da área, análise dos dados coletados. Com base em todo o material pesquisado e também como resultado da análise é possível propor algumas técnicas e ferramentas que podem auxiliar no processo de garantir a integridade de um Sistema.

5 CONSIDERAÇÕES FINAIS

Atualmente com o aumento da preocupação com a segurança de software, avaliar todas as ferramentas disponíveis para auxiliar a aumentar a segurança do Sistema, tendo em vista que a engenharia reversa já muito difundida por pessoas que possuem muito conhecimento em informática e que atacam Sistemas, mais ela também pode ser usada não somente para roubar ou fraudar Sistemas, com ela é possível aprender sobre o próprio Sistema, e com esse tipo de informação é possível diminuir e até evitar alguns tipos de invasões.

Alguns Sistemas devidos as suas funcionalidades extremamente uteis, ou sua fama, são alvos mais frequentes de pessoas que não querem pagar para usufruírem desses Sistemas, utilizando a engenharia reversa para conhecer onde é feita a autenticação da chave ou simplesmente entender o padrão do código da chave, eles conseguem burlar os métodos de segurança mesmo os de grandes empresas.

Esse trabalho tem como objetivo mostrar um pouco mais sobre a engenharia reversa, e como ela pode ser útil para aumentar o nível de segurança de um software, reduzindo assim os riscos de invasão ou modificações sem autorização do proprietário.

Para a realização desse trabalho, foram feitas várias pesquisas em livros cujos autores têm possuem uma ampla aceitação não somente com os profissionais da área como também invasores utilizam alguns autores aqui utilizados e foram usados artigos que tratam sobre o tema como também sites de empresas especializadas em segurança que trabalham incessantemente em novas soluções para segurança eletrônica.

Durante a elaboração deste trabalho, algumas das dificuldades encontradas dizem respeito ao material para embasar a pesquisa, devido a falta de livros traduzidos e artigos pertinentes, sendo necessário assim buscar fontes alternativas que suprissem essa falta.

Como já foi visto a engenharia reversa conta com inúmeras técnicas, e meios para abrir e modificar um Sistema, bastando a pessoa ou grupo responsável possuir um conhecimento muito aguçado em determinadas linguagens ou ferramentas, este autor buscou mostrar não somente ferramentas como também técnicas que são utilizadas por invasores assim como profissionais da área.

A engenharia reversa não necessita de um código fonte e muito menos de toda a documentação do Sistema para se concretizar, é possível através do executável do Sistema descobrir como o mesmo funciona e como os módulos se comunicam entre si, existem algumas ferramentas para analisar esse fluxo de dados e com isso decifrar alguns mistérios relativos ao Sistema.

Mais caso haja o código fonte ou também a documentação parcial ou completa do Sistema, também se pode usar a engenharia reversa, pois com o uso dela ela trará um aprofundamento e atualização sobre o conhecimento do Sistema e também na documentação do mesmo, facilitando assim futuras modificações que se fizerem necessário.

Utilizar técnicas que não necessitem do código fonte é uma maneira muito interessante de analisar e verificar brechas em um Sistema, tendo em vista que muitos ataques são feitos utilizando algumas dessas técnicas, para auxiliar no êxito dessas atividades, são utilizadas ferramentas que proporcionam os resultados esperados por quem ataca um programa.

Este trabalho trouxe da maneira mais simples que o autor conseguiu mostrar o que mesmo sendo necessário conhecimento em programação e a algumas linguagens de programação, a engenharia reversa é um ferramenta muito importante para proteger Sistemas, pois se o desenvolvedor conhece as fraquezas dos seus projetos ele conseguirá não apenas reduzir isso como também evitar ou dificultar quem tente utilizar a engenharia reversa.

. Para obter uma analise do nível de segurança do programa é muito interessante utilizar a técnicas que necessitam de mais informações, tais como

documentação, o código fonte do software em questão. Analisar como é feita a comunicação de componentes, arquivos e processos que envolvem o Sistema.

Analisando o fonte é possível encontrar, falhas no código, tais como anomalias que são basicamente partes de funções que são reaproveitadas em outro lugar no fonte e isso pode gerar informações muito uteis à quem estiver analisando o software. Uma técnica muito interessante é a de encapsulamento que assegura a segurança de determinada função, essa abordagem foi criada para linguagens mais novas, ela consiste em encapsular a função, e quem for usar a função, só pode enviar informações e a capsula processa e devolve o resultado, e assim não é possível ver dentro da capsula.

Algo interessante a ser realizado, é uma análise pratica do que foi apresentado neste trabalho a fim de verificar o funcionamento e o grau de dificuldade em aplicara a engenharia reversa não somente no executável de um Sistema assim como um com toda sua documentação e código fonte.

6 REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR: 6023. Informação e documentação**: Referências - Elaboração. Rio de Janeiro: ago. 2002.

_____. **NBR: 10520. Informação e documentação:** Apresentação de citações em documentos. Rio de Janeiro: nov. 2003.

_____. **NBR: 14724. Informação e documentação**: Trabalhos acadêmicos - Apresentação. Rio de Janeiro: 30 jan. 2006.

_____. **NBR: 6028. Informação e documentação:** Resumo - Apresentação. Rio de Janeiro: ago. 2002.

BRAGA, Rosana T. Vaccare. **Engenharia Reversa e Reengenharia.** Curitiba: UFPR, 2006. 36 p. Disponível em:<http://www.inf.ufpr.br/silvia/ES/reengenharia/reengenharia.pdf>. Acesso em: 25 mai 2012.

CANHOTA JUNIOR, Antonio Jorge Sapage da. et al. **Engenharia reversa**. Rio de Janeiro, 2005. Disponível em: <http://www.ic.uff.br/~otton /graduacao/informaticaI/apresentacoes/eng_reversa.pdf>. Acesso em: 08 out. 2012.

DIAS, Adriano Batista. **Engenharia reversa**: uma porta ainda aberta. Recife: abrepo, 2012. Disponível em:<http://www.abepro.org.br/biblioteca/ENEGEP1997_T6109.PDF>. Acesso em: 04 out. 2012.

EYLAN, Eldad. ***Reversing:*** *Secrets of Reverse Engeneering. Indianapolis: Wiley Publishing, Inc*, 2005. 561 p.

FLORES, Nuno Honório Rodrigues. **engenharia reversa de padrões em arquitecturas reutilizáveis.** Porto: Faculdade de Engenharia da Universidade do Porto, 2005. 139 p. Disponível em: < http://repositorio-aberto.up.pt/bitstream/10216/12643/2/Texto%20integral%20.pdf>. Acesso em: 20 mai 2012.

HOGLUND, Greg; MCGRAW, Gary. **Como quebrar códigos:** A arte de explorar (e proteger) *software*. São Paulo: Pearson, 2006. 202 p.

HOGLUND, Greg; MCGRAW, Gary. **Como quebrar códigos:** A arte de explorar (e proteger) *software*. São Paulo: Pearson, 2006. 404 p.

IGOR SOUMENKOV. ***Flashfake Mac OS X botnet confirmed*.** *Moscow: Securelist - Information About Viruses, Hackers And Spam*, 2012. Disponível em: <http:// www.securelist.com/en/blog/ 208193441/Flashfake_Mac_OS_X_botnet_confirmed>. Acesso em: 01 abr. 2013.

LIMA, Willian Dos Santos. **Compilação de bytecodes Java para um ambiente de arquitetura reconfigurável**. São José do Rio Preto: Unesp, 2009. 103 p. Disponível em:<http://www.dcce.ibilce.unesp.br/ppgcc/dissert/diss-10-wlima.pdf>. Acesso em: 21 mai 2013.

MOURA, Ronildo Cesar De. **O uso da engenharia reversa no desenvolvimento seguro**. Vitória: Esab, 2008. Disponível em: <ttp://www.esab.edu.br/arquivos/ monografias/ronildo-cesar-de-moura.pdf>. Acesso em: 28 mai 2013.

PAULINO, Jorge. **A engenharia reversa.** , 2009. Disponível em: <http:// www.artigonal.com /ensino-superior-artigos/a-engenharia-reversa-1308116.html>. Acesso em: 09 out. 2012.

PERMEH, Ryan E EDWARDS, Brandon. **Invasão de aplicativos:** engenharia reversa. 6. ed. São Paulo: *Mcafee Security Journal*, 2010. 34 p. Disponível em: <http://www.mcafee.com/br/resources/reports/rp-security-journal-summer-2010.pdf>. Acesso em: 07 abr. 2013 .

PRESSMAN, Roger S.. **Engenharia de *software*.** 6. ed. Porto Alegre: Mcgraw-Hill, 2006. 711 p.

RAJA, Vinesh; FERNANDES, Kiran J.. ***Reverse engineering*****:** *an industrial perspective. London: Springer,* 2008. 242 p. (*Springer series in advanced manufacturing*).

SINTES, Tony. **Aprenda orientação a Objeto em 21 dias.** São Paulo: Makron Books, 2002. 762 p.

SOMMERVILLE, Ian. **Engenharia de *software***. 6. ed. São Paulo: Pearson Addison Wesley, 2003. 567 p.