UNIVERSIDADE FEDERAL DO CEARÁ

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

DARWIN DE OLIVEIRA PINHEIRO

UM ESTUDO EXPERIMENTAL DAS FERRAMENTAS DE ENGENHARIA REVERSA APLICADAS ÀS VULNERABILIDADES DO SOFTWARE

QUIXADÁ

2013

DARWIN DE OLIVEIRA PINHEIRO

UM ESTUDO EXPERIMENTAL DAS FERRAMENTAS DE ENGENHARIA REVERSA APLICADAS ÀS VULNERABILIDADES DO SOFTWARE

Monografia submetida à coordenação do curso de Sistemas de Informação Universidade Federal do Ceará (UFC), como requisito para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Marcos Dantas Ortiz

QUIXADÁ-CE

2013

DARWIN DE OLIVEIRA PINHEIRO

UM ESTUDO EXPERIMENTAL DAS FERRAMENTAS DE ENGENHARIA REVERSA APLICADAS ÀS VULNERABILIDADES DO SOFTWARE

Monografia submetida à coordenação do curso de Sistemas de Informação Universidade Federal do Ceará (UFC), como requisito para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Marcos Dantas Ortiz

_____

Profº. Ms. Marcos Dantas Ortiz
(Orientador)

_____

Profº. Ms. Jefferson de Carvalho Silva
(Membro)

_____

Profº. Ms. Enyo José Tavares Gonçalves
(Membro)

_____ /Fevereiro/ 2013.

DEDICATÓRIA

A minha família Francisco Kermedy, Dalva Maria e Dalker, por sempre me motivar, indiretamente, a alcançar minhas metas e meus objetivos.

AGRADECIMENTOS

Nesse período de quatro anos e meios de graduação, posso dizer que amadureci bastante e absorvi muito conhecimento, onde este foi passado também pela comunicação que estabelecia com quem estava a minha volta.

Primeiramente quero agradecer a Deus, por estar presente em minha vida, sempre me dando tudo aquilo que é melhor para mim, dando saúde que é a essência da vida, me motivando com pensamentos de grandezas e dando forças para sempre ir em frente e alcançar minhas metas e objetivos.

Minha família por me mostrar a importância de ter mais conhecimento, e de me incentivar a alcançar meus objetivos.

A todos meus amigos que consegui fazer, nesse período de graduação, pela amizade, companheirismo, e por sempre está compartilhando conhecimento, ajudando a descobrir que a informação é a moeda desses novos tempos.

A todos meus professores Marcos, Davi, Vitor, Enyo, Fábio, Camilo, Flávio, Lincoln, Atslands, Wladimir, Diana, Ciro, Aragão, Alberto, Arthur, Andréia, Paulyne, Esdras, por me passar conhecimento contribuindo para meu aprendizado, e alguns pela excelente metodologia de ensino que foi compatível com a base de meu aprendizado.

A professora Tânia pela ajuda que sempre deu quando necessário e por estar sempre ali presente com uma capacidade surpreendente de entender um problema e logo encontrar uma solução, achando formas de ajudar sempre quem demonstra interesse.

Ao meu orientador, professor Marcos Dantas. Por sua dedicação, paciência, motivação, ensinamentos e pela oportunidade de trabalhar com o mesmo.

Aos professores da minha banca, Enyo e Jefferson, por me ajudarem dando grandes contribuições para melhoria do meu trabalho de conclusão de curso.

Finalmente, agradeço a todos que de alguma forma me ajudaram, contribuíram e torceram por mim.

Em vez de dizer: isto não pode ser feito, continue questionando o seu cérebro: como isto pode ser feito?

Lair Ribeiro

RESUMO

Na última década a Tecnologia da Informação se consolidou como diferencial estratégico e se expandiu em diversas áreas, como: comercial, industrial e pessoal. A quantidade de informação que é gerada diariamente no mundo é proporcional a esse crescimento, que induz o aumento da quantidade de dados, às vezes restritos, que são armazenados nos computadores em volta do mundo. Juntamente com esse crescimento, aumenta também o número de usuários mal-intencionados que possuem conhecimentos técnicos na área da tecnologia da informação, e os utilizam para roubar dados ou deferir ataques diversos, praticando atividades ilícitas que acabam prejudicando pessoas e organizações, indo contra as leis e a ética. Contribuindo com essas práticas, novos vírus aparecem a cada dia, como também os *spywares* e *malwares*, ameaçando a privacidade, veracidade e segurança desses dados. Com isso, é motivado a criação de atualizações diariamente de sistemas de segurança como os antivírus, Firewalls, e IDS, entre vários outros sistemas que são usados diariamente no trabalho ou para fins pessoais, e que efetuam alguma funcionalidade. Contudo, se o foco durante o processo de desenvolvimento destes sistemas fosse destinado à segurança, teríamos uma economia considerável de tempo e dinheiro. A criação de *softwares* de maior qualidade evitaria a frequente reparação destas ferramentas. A Engenharia Reversa de *software* contribui para que o desenvolvedor entenda como o *software* trabalha, tornando possível estudar sua estrutura lógica a partir do produto final. Além disso, permite efetuar testes necessários para aprimoramento, alterar funções do *software*, recuperar códigos-fontes, incrementar sua qualidade nos quesitos desempenho e segurança. Logo, este trabalho tem por objetivo apresentar a utilização das ferramentas de Engenharia Reversa aplicando-as em *softwares* na plataforma *Windows*. Nesse contexto, é demonstrado como um programa funciona internamente a partir do monitoramento e análise via ferramentas de engenharia reversa. Este trabalho também classifica as diversas ferramentas com base nas suas características e funcionalidades, além de apresentar a importância da linguagem *assembly*.

Palavras chave: engenharia reversa; segurança; *windows*; *malwares*.

LISTAS DE FIGURAS

LISTAS DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

A Engenharia Reversa tem sua origem na análise de *hardware*, em que é comum a prática de obter projetos de produtos finalizados, onde é normalmente aplicada para melhorar os próprios produtos, bem como para analisar os produtos de competidores ou de adversários em situações militares ou de segurança nacional. Normalmente seu objetivo é obter um entendimento do sistema em um nível maior de abstração (REKOFF, 1985).

Segundo Dias (1996), ela comporta uma larga diversidade de definições, devido aos diferentes empregos e processos adotados. Não há uma definição consensual do que seja Engenharia Reversa, mas dentre as definições variadas há duas que agem diferentemente. Uma primeira se constitui na obtenção de informação que caracteriza e especifica o objeto da ação, identificando seus componentes e seu padrão de inter-relacionamento, e na segunda, o objeto é representado com um nível mais elevado de abstração. Essa prática não altera o objeto da ação sendo um processo não destrutivo, mas de exame que não ocorre modificação do objeto.

Sua aplicação consiste em usar a criatividade para, a partir de uma solução pronta, retirar todos os possíveis conceitos empregados naquela solução, como por exemplo, definições de estruturas de um modelo de *software*. Existem alguns sistemas, tanto comerciais como não comerciais, que são distribuídos apenas no formato binário sem o código fonte, já compilado na linguagem de máquina, de maneira que o usuário apenas possa usar e executar suas funcionalidades propostas. Como o usuário não tem acesso ao código fonte, dificulta-se a criação de cópias e falsificações (DIAS, 1996).

A engenharia reversa proporciona uma importância maior na compreensão de sistemas, através de um estudo dos processos de forma inversa, partindo de um objeto pronto. Destaca-se também sua importância para utilização em correção e/ou adaptação de *software* de terceiros, recuperação de dados em sistemas não mais utilizados, como também analise de vírus, *trojans* e *malwares* (HOGLUND, 2006).

Neste trabalho, apresenta-se a utilização de ferramentas de Engenharia Reversa aplicadas a software em três estudos de caso, com objetivo de analisar e apresentar seus artefatos. As ferramentas foram classificadas em relação às suas funcionalidades e, onde e como aplicá-las na prática.

Este trabalho é composto de seis capítulos. No primeiro capítulo foi apresentada a introdução, que aborda o conceito e a importância da engenharia reversa em *software*. No segundo capítulo, será apresentado o objetivo geral e os específicos. No terceiro capítulo, será apresentada a revisão bibliográfica sobre Engenharia Reversa. No quarto capítulo, serão abordados os procedimentos metodológicos que determinam como será feita e analisada a aplicação das ferramentas, para que no quinto a seja mostrado a demonstração de sua utilização. Finalizando, o sexto capítulo, aborda os pontos mais relevantes deste estudo em relação à aplicação da Engenharia Reversa.

2 **OBJETIVOS**

2.1 **Objetivo geral**

Apresentar a utilização das ferramentas de Engenharia Reversa sobre *software*, aplicadas em diferentes cenários de segurança.

2.2 **Objetivos específicos**

- Apresentar conceitos, importância e utilidade da Engenharia Reversa;
- Identificar e distinguir ferramentas de Engenharia Reversa usadas em *softwares*;
- Apresentar cenários de uso da Engenharia Reversa no ambiente *Windows*;
- Apresentar a importância da linguagem *assembly*;
- Abordar o processo de descompilação de um *software*;
- Identificar vulnerabilidades de *softwares*;
- Identificar ações maliciosas de *malwares*
- Propor ações anti-reversão.

3 **REVISÃO BIBLIOGRÁFICA**

3.1 **Engenharia Reversa**

A Engenharia Reversa é uma prática que envolve uma vasta gama de atividades apresentando vários conceitos multidisciplinares que se complementam (LIMA, 2003).

Enquanto a engenharia convencional transforma conceitos e modelos em objetos, produtos ou sistemas reais já existentes, a Engenharia Reversa transforma conceitos e modelos em engenharia. (VARADY, 1997; DICKIN, 1996, ARAÚJO, 2010). Dessa forma, ela consiste em analisar estes conceitos e modelos para identificar seus componentes e como se relacionam para criar representações diferentes do sistema ou em mais alto nível de abstração (CHIKOFSKY, 1990).

A expressão ‘Engenharia Reversa’ tem sua origem na análise de produtos, e é normalmente utilizada pelas empresas para melhorar não apenas os próprios produtos, como também analisar os produtos dos concorrentes. Ela por si própria não envolve modificação do sistema analisado ou criação de um novo sistema, mas trata-se apenas de um exame que não envolve o processo de mudança ou replicação (CHIKOFSKY, 1990).

O processo de Engenharia Reversa começa a partir deum produto e termina em um modelo virtual, enquanto o processo de engenharia convencional começa com um modelo virtual e produz um produto (PUNTAMBEKAR, 1994; LIMA, 2003). Com o modelo virtual obtido a partir do modelo físico, é possível formular as etapas da engenharia convencional (LIMA, 2003).

As aplicações da Engenharia Reversa estão relacionadas com a criação de novos produtos, cópia de modelos existentes, correção e melhoria de modelos, inspeção e documentação de produtos, tornando o setor produtivo mais ágil e flexível. Desta forma, é uma tecnologia que contribui para a melhoria da qualidade dos produtos e para que as organizações sejam mais competitivas por possibilitar o desenvolvimento de novos produtos e serviços com custos e prazos menores (JARDINI et al., 2011).

Sendo a Engenharia Reversa um processo que analisa uma aplicação ou um sistema, identificando os componentes e inter-relações, ela pode ser confundida com pirataria. Mas em ambientes de grandes corporações, com milhares de programas, com particularidades que interessam apenas à instituição onde foram desenvolvidos, e que dificilmente seriam adaptados à realidade de outras organizações, a Engenharia Reversa é uma atividade lícita. Nestas organizações, a Engenharia Reversa é aplicada para melhor entendimento de programas e sistemas, para manutenção e melhorias ou para reengenharia de sistemas (CIFUENTES, 2001).

3.2 **Reengenharia**

Enquanto que a Engenharia Reversa tem objetivo de desmontagem das “caixas pretas” do *software*, de seus segredos, de trás para frente, recuperando do processo de análise de

projeto, especificação e documentação, a Reengenharia, também chamada de renovação ou recuperação, não somente recupera informações de projeto de um *software* existente, mas usa estas informações para alterar ou reconstruir o sistema existente, reimplementando a função do sistema, adicionando novas funções, com o intuito de melhorá-lo (PRESSMAN, 1995).

Segundo Vallim (2002), as pesquisas e práticas de Engenharia Reversa e a reengenharia estão focados ao nível de código, em que importantes regras de negócio estão inseridas e durante toda a evolução do *software*, são efetuadas mudanças, são acrescentadas novas funções, corrigidos problemas e implementadas melhorias, que nem sempre são perfeitos. Em sistemas com documentação pobre, o código é a única fonte de informação confiável do sistema, porém, nem sempre contêm todas as informações necessárias, como também o conhecimento sobre a arquitetura e regras de negócio. Restrições do sistema e domínio das aplicações existem apenas na cabeça dos engenheiros de *software*, além dos problemas de esquecimentos, saída ou troca de pessoal, depreciação da documentação e aumento da complexidade. Tudo isso aumenta a distância entre saber o que é informação útil e as informações de requisitos para alteração do *software*. Como resultado, surge a necessidade do uso de Engenharia Reversa focando o entendimento do código conjuntamente com a Reengenharia caso a criação novos produtos seja a solução.

A reengenharia de *software* é uma forma de reuso, em que se reutiliza a funcionalidades do código na reconstrução do sistema, como também se revitaliza a estrutura do código, obtendo o entendimento das funcionalidades da aplicação, e recuperando as informações do projeto que são organizadas de forma coerente e reutilizável (NOVAIS, 2002; PENTEADO, 1996).

3.3 **Acordos e leis**

Existem problemas de legalidade que a Engenharia Reversa pode gerar, como uma empresa querendo criar uma cópia de produtos da concorrência que vendem bem. No entanto essa questão depende das leis de cada país, havendo países que não possuem leis específicas sobre o assunto.

Uma das leis mais conhecidas é a "*Digital Millenium Copyright Act*", ou Lei dos Direitos Autorais do Milênio Digital, conhecida como DMCA. É uma lei de direitos de autor dos Estados Unidos da América que criminaliza não apenas a infração do direito autoral em si, mas também a produção e a distribuição de tecnologia que permita evitar medidas de proteção dos direitos de autor. Aprovada em 1998, dentre várias medidas para proteger direitos autorais na informática, também faz restrições em relação à Engenharia Reversa que só é

permitida para fins de analisar compatibilidade com outros *software*s e/ou *hardware* (USA, 1998).

Na União Europeia existe o "*EU Copyright Directive*", de 2001, parecida com o "*Digital Millenium Copyright Acts*", porém menos restritiva. Restringe o uso da Engenharia Reversa quando seu objetivo final é a cópia de algum programa ou quebra de patente visando lucro. Porém, caso seja pra fins acadêmicos ou de compatibilidade, não existe muitas restrições (EU, 2001).

No Japão, a Lei Japonesa de Concorrência Desleal de 1993 proíbe a imitação servil, mesmo no caso de produtos que não foram patenteados, nem protegidos por direitos autorais, impondo limites claros à aplicação da norma de apropriação ilícita. É válida apenas por três anos, ressalva-se o caso de modificações ou aperfeiçoamento técnico efetuado pelo competidor com base no item copiado, e a necessidade de padronização e compatibilização de produtos e o uso de elementos de caráter estritamente funcional. Ou seja, a proibição de imitação não impede o progresso técnico, ressalva o domínio das patentes para proteger ideias e conceitos, e o interesse social na padronização e compatibilização industrial (JAPÃO, 1993).

Já no Brasil, não existe uma lei específica sobre Engenharia Reversa. Quando ocorre Engenharia Reversa, costuma-se proceder de duas maneiras: se a Engenharia Reversa não tem como objetivo a pirataria ou infração de algum direito autoral, não é considerada crime, porém, em caso contrário, a Lei de *software* e também de Direitos Autorais protege seus autores (BRASIL, 1998).

3.4 **Aplicações da Engenharia Reversa**

A Engenharia Reversa permite obter um modelo virtual de um objeto virtual ou físico, o que é uma capacidade muito interessante para diversas aplicações. Para Christensen (2000) várias áreas têm utilizando a Engenharia Reversa em suas aplicações.

No campo do *hardware*, a Engenharia Reversa é mais utilizada em conjunto com outras técnicas, duplicação de um artefato, que também serve para o campo de *software*. No entanto, o entendimento suficiente do projeto é o verdadeiro objetivo, como também melhorias funcionais, e manutenção, que consiste em corrigir falhas, melhorar seu desempenho ou outros atributos, ou adaptar o produto a mudanças (CHIKOFSKY, 1990).

De acordo com Vallim (2002), nesta técnica o objeto de um sistema pode ser um programa, parte de um programa, um conjunto de programas complexos interagindo, instruções

de controle de serviços, interfaces e até mesmo arquivos. Na engenharia convencional este objeto é o produto do processo de desenvolvimento, enquanto que na Engenharia Reversa este dá origem a um processo de desenvolvimento.

Chikofsky (1990) destaca duas partes importantes da Engenharia Reversa: redocumentação e recuperação do projeto. Redocumentação tem como objetivo criar ou reescrever a documentação do sistema, já a recuperação do projeto visa recriar abstrações de projeto a partir do código-fonte, como também documentações de projeto existentes, experiências pessoais, conhecimento sobre o problema e domínio da aplicação. Deve reproduzir toda a informação requerida para que uma pessoa possa entender perfeitamente o que um programa faz, como faz e porque faz.

Na análise de *malware*, esta técnica é utilizada quando se deseja descobrir, de forma detalhada, quais os passos desenvolvidos pelo *malware* no sistema. Isso possibilita, por exemplo, identificar quais técnicas ele utilizou para comprometer o sistema, como ele esconde suas atividades para ocultar sua execução, quais dados do sistema comprometido ele captura, dentre outras atividades normalmente executadas por *malware*. A partir dessas informações, é possível inclusive criar procedimentos que retirem da máquina comprometida todas as modificações feitas pelo *malware*, levando-a a um estado integro, anterior ao comprometimento (AFONSO et al., 2011, p. 19).

O sucesso ou fracasso da Engenharia Reversa em sistemas depende dos objetivos específicos do processo. A maioria dos esforços de Engenharia Reversa falha ao tentar alcançar os objetivos tradicionais de extrair automaticamente especificações completas para o processo de engenharia, enquanto que se os objetivos de extração automática de especificações fossem mais modestos, a chance de sucesso seria maior, e com suas chances elevadas por assistência humana especializada (QUILICI, 1995).

Para Vallim (2002), a chance de sucesso é maior se o foco do problema a ser resolvido for menor e mais bem definido. Também ressalta que é preciso fazer uma avaliação criteriosa de quais informações podem ser obtidas automaticamente do código fonte e de que forma elas podem contribuir para o processo de engenharia. Algumas pesquisas focam na busca de padrões no código que permitam reconhecer abstrações de alto nível, tais como domínio, estrutura de dados e algoritmos, mas infelizmente não é possível extrair um projeto hierárquico completo em sistemas do mundo real.

Pesquisadores têm criado soluções para explorar, manipular, analisar, sumarizar, sintetizar e visualizar componentes de *software*. Muitas destas ferramentas de reengenharia de *software* estão focadas na estrutura de sistemas antigos com o objetivo de transferir estas informações para os engenheiros de *software* e ajudá-los na reconstrução ou reuso destas informações. O investimento em tecnologia para entendimento de programas é crítico para a indústria da tecnologia da informação, com intuito de controlar os altos custos e riscos inerentes da evolução de sistemas antigos (VALLIM, 2002).

3.5 **Ferramentas**

De acordo com Vallim (2002), enquanto a necessidade de qualidade e produtividade motivava o surgimento de ferramentas CASE para novos sistemas, os sistemas antigos se transformaram em sistemas legados que continuaram a consumir um volume considerável de tempo e recursos para sua manutenção e melhorias, motivando o surgimento de ferramentas de Engenharia Reversa. Um exemplo é um sistema antigo para o qual qualquer manutenção leva cada vez mais tempo do que se deseja e qualquer pequena melhoria consome cada vez mais tempo e recurso. Na hora de implementar uma modificação no sistema, a questão custo benefício esbarra no aspecto do tempo necessário para implementá-la. Por isso, uma forma de minimizar o impacto do aspecto tempo, é a necessidade da construção de uma ferramenta que auxilie quem vai fazer a modificação a obter informações que reduzam o tempo de busca de informações do sistema.

O principal desafio das ferramentas de Engenharia Reversa está na necessidade de dar suporte a uma variedade de linguagens e de serem capazes de ser estendidas para outras linguagens. Dessa forma, construí-las não é uma tarefa fácil (CIFUENTES, 2001).

3.5.1 **Depuradores (*Debuggers*)**

Segundo Hoglund e McGraw (2006) um depurador é aquele programa que se conecta a outro programa e passa a controlá-lo, garantindo analisar durante a sua execução o fluxo lógico. Através dele é possível rastrear códigos, percorrer sua execução através de *breakpoints*, verificar o estado das variáveis na memória, em tempo de execução.

Um depurador pode atuar por *software*, que ocorre quando o usuário coloca um *breakpoint* em uma instrução, e ele troca esta instrução por uma chamada à instrução especial, "INT3h*" (*instrução *em assembly* para interrupção*)*, que retorna a informação que a interrupção foi alcançada, logo é substituído pela instrução original e a execução é congelada, possibilitando

a análise do estado da memória em qualquer ponto. A depuração também pode ser por *hardware*, porém mais complexa, pois utiliza alguns recursos do processador, podendo rastrear e analisar dados e estados lógicos, emular instruções e examinar os registradores.

Os depuradores podem ser em modo de usuário, onde trabalham como programas normais sob sua plataforma, depurando outro programa específico. Um exemplo de depurador em modo de usuário (*freeware*) é o OllyDbg (2013).

Existem também os depuradores em modo de *kernel* que são instalados no sistema operacional, e torna possível a depuração de programas, dispositivos e até o próprio sistema operacional. Um exemplo dos depuradores em modo *kernel* mais conhecidos é o *shareware* SoftIce (2012).

3.5.2 **Desassembladores (Disassemblers)**

Hoglund e McGraw (2006) explicam que, são classificadas como *desassemblers* aquelas ferramentas que convertem a linguagem de máquina, binário, em código *assembly*. As instruções e a tradução do código são dependentes da plataforma, suportando para cada plataforma um *desassembler* diferente. Os *desassemblers* usam algoritmos de conversão lineares e recursivos.

A conversão linear é feita através da sequencial conversão dos *bytes* de um bloco em instruções de máquina, enquanto que a conversão recursiva analisa as instruções de acordo com o fluxo de dados, podendo detectar e descartar blocos inúteis de dados, tornando esta conversão mais eficaz. (EILAM, 2005). Os *desassemblers* mais conhecido para processadores IA-32 é o IDA PRO (2013). Para .NET existe um *desassembler* chamado ILDASM (2013), que converte desde o código MSIL.

3.5.3 **Descompiladores ou compiladores reversos**

Os descompiladores são praticamente iguais aos *desassemblers*, só que vai um posso à frente, chegando a obter além da linguagem de baixo nível, *assembly*, a linguagem de alto nível através conversão do código *assembly* ou código de máquina em um código-fonte. Dessa forma, essas ferramentas são úteis para identificar a lógica usada em alto nível. Os descompiladores melhoram e facilitam a compreensão do código gerado, mas geralmente não é gerado no mesmo modelo em que foi desenvolvido. (Hoglundet al., 2006)

Mesmo com diversos descompiladores disponíveis para as diversas linguagem alto nível, destacam-se, para a linguagem C, o DCC (2012) e o Bommerang (2012), *freeware* de código aberto.

3.5.4 **Ferramentas de Patching**

De acordo com Eilam (2005), estas ferramentas convertem qualquer arquivo em formato binário, em hexadecimal, possibilitando a edição dos valores obtidos. Não são exclusivas para a prática da engenharia reversa, mas não deixa de ser um item indispensável para ela.

Essas ferramentas possibilitam ajustar *bytes* nas instruções de máquina, que ocasionam erros, corrigindo o fluxo de dados de seu processamento. Podem ser usadas também para desativar métodos de proteção ou validação e até mesmo integrar novos recursos.

São considerados editores hexadecimais o Hiew (2013) – dono de um método muito interessante que é a possibilidade de edição através da digitação do código *assembly* e o Hex Workshop (2013).

3.5.5 **Packers e Unpackers**

Esse grupo de ferramentas refere-se aquelas que são usados para compressão de dados, com redução de sua banda, e consequentemente redução de seu espaço. Com a possibilidade de utilização da técnica de Engenharia Reversa hoje em *software*, o auxilio dessas ferramentas são de grande importância, tanto para esconder código fonte através de algoritmos de embaralhamento, quanto para mostrar o que foi modificado (EILAM, 2005).

Existem diversas outras ferramentas que verificam se o executável do *malware* contém alguma proteção, e analisam os tipos de compressores. Elas são consideradas identificadores para que seja utilizado um *unpacker* correspondente. Pode ser classificado o PEiD (2012) e também o Exeinfo PE (2012).

3.5.6 **Linguagem Assembly**

Para Eilam(2005) a linguagem *assembly* não é considerada obsoleta, mesmo sendo uma linguagem primitiva da segunda geração. Esta existirá enquanto os microprocessadores existirem, mesmo que apareçam novas linguagens com um mais alto nível, pois em algum momento está será convertida em código de máquina para ser executada pelo microprocessador. Sendo possível, sempre, obter o código fonte em *assembly* de qualquer outro programa,

independente da linguagem em que foi desenvolvido, com exceção de Java e da *Common Intermediate Language* da Microsoft .Net, que os códigos-fontes originais podem ser recuperados.

Assembly é a principal linguagem para o entender e compreender o funcionamento da prática de Engenharia Reversa, mas não é um aprendizado trivial, geralmente consome mais tempo de prática de programação, e ainda é preciso entender os conceitos e instruções de cada modelo de processador para se trabalhar na linguagem, já que o *assembly* é dependente do *hardware* (EILAM, 2005).

3.5.7 **Instruções**

As instruções são códigos de operação que são utilizados na comunicação com o processador. Cada instrução *assembly* possui seu correspondente em hexadecimal, por exemplo: POP EAX (instrução *assembly*), corresponde aos *bytes* em hexadecimal *66 58*. A lista completa dessas instruções é encontrada nos manuais dos seus respectivos processadores que diferem algumas vezes dependendo da arquitetura (INTEL, 1999).

Todas as instruções são formadas de um, dois ou nenhum parâmetro, variando conforme sua funcionalidade, denominados operandos. No quadro 1 é apresentado algumas das principais instruções.

Quadro 1 – Principais instruções da linguagem *assembly*

INSTRUÇÃO	**SINTAXE**	**DESCRIÇÃO**
AND	AND destino, fonte	AND realiza uma operação lógica AND bit a bit nos seus operandos, colocando o resultado no destino.
TEST	TEST destino, fonte	AND realiza uma operação lógica AND bit a bit nos seus operandos sem alterá-los, modifica apenas os *flags*.
CMP	CMP destino, fonte	Esta instrução compara efetuando uma subtração da fonte no destino. Atualiza as *flags*, mas não armazena resultado.
DEC	DEC valor	Decrementa o valor de um registrador ou de uma variável.
INC	INC valor	Incrementa o valor de um registrador ou de uma variável.
MOV	MOV destino, fonte	Copia um byte do operando fonte para o operando destino.
NOT	NOT destino	Realiza uma operação lógica invertendo cada bit do operando destino.

OR	OR destino, fonte	Realiza uma operação lógica OR INCLUSIVE bit a bit nos seus operando e põe o resultado no destino (retorna 0 quando operando for 0).
XOR	XOR destino, fonte	Realiza uma operação lógica OR INCLUSIVE bit a bit nos seus operandos e põe o resultado no destino (retorna 1 quando operandos diferentes).
PUSH	PUSH fonte	Transfere o valor da fonte para o topo da pilha (área de memória que armazena dados temporariamente), e decrementa o registrador SP (contém o endereço do topo da pilha) pelo tamanho do operando.
POP	POP destino	Transfere o valor do topo da pilha para o operando destino e incrementa o registrador SP para apontar par ao o topo da pilha.
SUB	SUB destino, fonte	Subtrai a fonte do destino, e armazena o valor no operando destino.
JMP	JMP destino	Salto incondicional, para destino.
JE/JZ	JE destino	Salta se igual. Salto feito se ZF(*flag*) ativado.
JNE/JNZ	JNE destino	Salta se não for igual. Salto feito se ZF desativado

Fonte: Adaptado de INTEL, 2013, p. 1496-1505.

3.6 **Segurança da Informação**

Segundo Campos (2005) a informação é todo e qualquer conteúdo ou dado que tenha algum valor para uma determinada pessoa ou organização. Podendo até estar exposta ao público, mas na maioria dos casos essas informações necessitam de restrições de acesso. Para isso, são implementadas técnicas que promovem segurança da informação.

3.6.1 **Políticas de Segurança**

A segurança da informação pode ser afetada por diversos fatores comportamentais, como o ambiente, a infraestrutura em que está, ou por pessoas mal intencionadas que objetivam modificar, danificar ou furtar a informação. Com tais ameaças, são crias políticas de segurança, que se constituem de um conjunto de regras e práticas, que definem a proteção e o compartilhamento das informações e dos recursos. Cada política aplicada a um sistema deve promover cinco elementos, são eles: Disponibilidade – o sistema deve estar disponível para o usuário sempre que o requisitar; Utilização – os dados do sistema devem ser utilizados para a devida finalidade; Integridade – os dados do sistema devem estar sempre íntegros, e em condições de ser utilizados; Autenticidade – o sistema deve ter condições de verificar a

identidade do usuário; Confidencialidade – os dados devem ser apresentados apenas para os seus respectivos autores (CAMPOS, 2005).

3.6.2 ***Malwares***

O termo *malware* é atribuído a todo programa de computador que se infiltra, de forma ilícita, em um sistema alheio para causar algum dano ou furtar informações sensíveis, os principais são: *Vírus* – um programa que de computador que se hospeda em outro e se multiplica visando destruir e inutilizar um computador; *Trojan* – programa que lembra a lenda do cavalo de troia, hospedado em outro arquivo visa criar um canal de acesso um invasor; *Worm* – mesmo conceito do vírus, a diferença é que o *vírus* necessita de um hospedeiro para se propagar, enquanto que o *worm* já é um programa completo; *Adware* – trazem para a tela do usuário várias propagandas; *Backdoor*– se infectam no computador com intuito de deixar uma ou mais portas de comunicação com algum servidor; *Keylogger* – programa de computador que tem como finalidade monitorar tudo que é digitado; *Phishing* - caracterizada por tentativas de adquirir informações, como senhas e números de cartão de crédito, se fazendo passar um uma pessoa ou organização; *Rootkit* – sua intenção é se camuflar, impedindo que seu código seja encontrado por qualquer antivírus(YIN, 2007).

São nomeados de *hackers*, os indivíduos com alta capacidade técnica que elaboram e modificam *softwares* e *hardwares*, implementando novas funcionalidades ou adaptando antigas, dentre eles estão os *crackers*, que são os verdadeiros invasores que possuem intenções maliciosas de violar direitos de pessoas e sistemas, sendo os responsáveis por criar os *malwares* (MARTINS, 2007).

3.6.3 **Risco de Softwares**

Segundo McGraw (2006), classificar os risco de softwares pode auxiliar no entendimento do problema, se resumem em:

- Defeito – São considerados defeitos quando os problemas se destinam em implementação e projeto. Um comportamento inadequado do sistema que pode provocar falhas.
- Falha – Operação incorreta ou dos componentes do sistema. A falha não induz necessariamente a ocorrência de um erro, porém pode acarretar erros.
- *Bug* (Erro) – Problema na implementação do código fonte de um software. Resultado que foge de suas especificações.
- Risco – Probabilidade de um bug ou falha impactar o *software*.

4 PROCEDIMENTOS METODOLÓGICOS

Realizou-se até aqui um levantamento das ferramentas disponíveis que implementam Engenharia Reversa (ver seção 3.5). Para cada ferramenta estudada, foram identificadas características que serviram para classificá-las e distingui-las em suas utilizações (ver seções 3.5.1 a 3.5.5). Em seguida, elas foram aplicadas a diferentes cenários que foram construídos com a finalidade de validar sua eficácia através da análise dos seus artefatos.

O fator que norteou a escolha das ferramentas para os cenários foram suas características, como mostrado no quadro 2, e a eficiência observada nos trabalhos dos praticantes de Engenharia Reversa Eilam (2005), Hoglund e McGraw (2006).

Quadro 2 – Características das ferramentas

	Debbuger	***Hex Editor***	***Decompiler***	***Packer Detector***	***Disassember***
Exeinfo				X	
Hex Workshop		X			
Hview		X			
DCC			X		
IDA PRO	X				X
ILDASM					X
PEiD				X	
OllyDBG	X	X			X
Bommerang			X		

Fonte: própria (2013)

Para resolução desse trabalho foram usadas quatro ferramentas. A OllyDBG (2013) é caracterizada como *hex editor*, *debugger* e *disassembler*. Atua em processadores de *32-bits*, rodando em ambiente *Windows*. Tem ênfase na análise de código binário tornando útil em casos em que o código fonte não está disponível. A IDA PRO(2013) é outra ferramenta que é classificada como *debugger* e *disassembler*. Ela é está disponível nas plataformas *Windows*, *Linux*, e *Mac OS*.

A conversão e alteração de valores hexadecimais para plataforma *Windows*, com recursos que incrementam facilidades e flexibilidades de leitura e adaptação do binário do *software*, são funções do HEXWORKSHOP (2013). Também é apresentando o *packer/unpacker*, que detecta tipos de compressões, *EXEINFO*(2012).

4.1 **Metodologia de análise**

Este trabalho utilizou a metodologia proposta por Vênere (2009) para executar os cenários e analisar os resultados. O autor defende que a análise inicie-se a partir dos itens considerados mais fáceis, onde podem ser encontradas informações mais prontamente acessíveis, sem que seja necessário um grande desprendimento de tempo de análise *off-line* de código.

Para o autor, o item elencado primeiramente foi à busca por cadeias de caracteres no arquivo executável, cuja finalidade é encontrar vestígios de comportamento da aplicação que possam ser utilizados como ponto de partida para uma análise mais aprofundada. Outra informação de suma importância é a detecção de funções importadas pela aplicação, já que algumas delas referem-se à leitura de teclado, tratamento de cadeias de caracteres, criação e abertura de arquivos, dentre outras. Além disso, as aplicações em ambiente *Windows* compartilham funções daquele sistema operacional que são amplamente conhecidas pelos desenvolvedores de *softwares* e que integram a sua API (*application programming interface*).

O autor sugere também que a análise do código binário seja feita após examinados os nomes de estruturas que tenham sido identificadas automaticamente pela ferramenta de análise, no caso a IDA (2013), bem como os gráficos de chamadas de funções, sendo essa última realizada depois de renomeadas aquelas cujo sentido é possível identificar, já que facilitam a compreensão do relacionamento existente entre elas e, consequentemente, do comportamento geral da aplicação. Após essa análise inicial tenha sido concluída, é favorável seguir uma análise em tempo real, onde a aplicação é executada em um ambiente controlado permitindo monitorar o seu comportamento sem comprometer a segurança do sistema que a hospeda.

O autor demonstra a eficácia da sua aplicação, após definida sua metodologia de análise, em um *worm*, buscando identificar: os dados que são modificados e acessados; seu meio de propagação; seu comportamento no sistema hospedeiro; os possíveis canais de acesso para *hackers*; recursos que o mantém no sistema; e o prejuízo que pode causar.

Com isso, o autor comprovou, sem a necessidade de uma análise aprofundada do código binário e utilizando-se das ferramentas de Engenharia Reversa, a possibilidade de se extrair dados e informações que tem grande importância para a compreensão do comportamento por inteiro de *malware*.

4.2 **Cenários**

Um cenário, neste trabalho, é representado por um programa de computador que é analisado pela técnica de engenharia reversa, através das ferramentas pré-determinadas, tendo como métrica de escolha dos cenários à análise sobre código binário. Objetivando verificar se os resultados esperados são alcançados (efetividade).

4.2.1 **Burlando a autenticação de usuário**

Neste cenário foi encontrado um "pseudo programa" (PROJECTLOGIN, 2012) que roda em ambiente *Windows*, que possui duas telas. A primeira é apresentada para o usuário com dois campos de *login* e senha, para que faça a autenticação. Caso autenticado com sucesso, será direcionado para a tela seguinte que mostra apenas uma mensagem de boas-vindas, exemplificando um sistema que exige autenticação de usuário.

Para isso foi utilizada a ferramenta *OllyDBG*, classificada como *debugger*, caracterizado por métodos de interrupção em tempo de execução para análise, que necessita de pelo menos o programa em linguagem de máquina para analisá-lo em tempo de execução e decifrar a senha que autentica o usuário.

Como resultado, através da Engenharia Reversa via execução da ferramenta *OllyDBG* foi verificada a descoberta da senha autêntica do usuário que dá acesso à segunda tela do programa.

4.2.2 **Burlando o período de avaliação**

Para este cenário foi utilizado o software *TimePuter*(2012), classificado como *shareware*, é um programa que calcula o período de um determinado ponto no tempo até a data atual. Ao abrir o programa uma tela de validação é exibida, informando que está disponível por30 dias para testes, e logo após será necessário fazer a compra do mesmo para continuar sua utilização. Disponível na plataforma *windows*.

Apenas para fins acadêmicos, foi utilizada a ferramenta IDA, classificada como *disassembler*, para gerar o código-fonte em *assembly* deste programa, localizando as instruções que chamam a tela de validação, e identificando seus valores hexadecimais equivalentes, possibilitando a alteração com a ferramenta Hex Workshop e gerando um executável que implementa as alterações.

Como o resultado, as ferramentas de Engenharia Reversa criaram um novo executável onde a tela de validação foi esquecida, e a interface principal do programa passou a ser a primeira apresentada.

4.2.3 **Análise de *trojan***

Diferente dos demais, neste cenário será estudado um *trojan*(2013), recebido via conta de e-mail do google, que é classificado como um *malware* para ambiente *windows*.

Foi necessário identificar se o arquivo existia proteções utilizando o PEiDeExeinfoPE.Com a remoção das proteções através do *unpacker* sugerido pela ferramenta, o executável ficou livre ser para depurado e disassemblado com o OllyDBG e o IDA, respectivamente.

Como resultado, foram apresentadas, a partir do código obtido, as ações maliciosas que o *trojan* executaria caso fosse acessado na plataforma *windows* sobre o processador IA-x86.

5 **RESULTADOS**

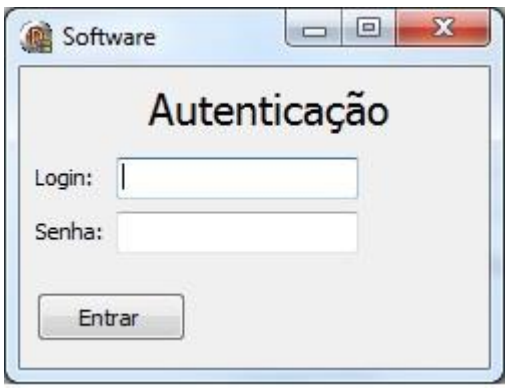
Os resultados aqui apresentados estão divididos de acordo com os cenários apresentados na seção 4.1. Os cenários foram montados e avaliados neste estudo para validar os conceitos de engenharia reversa na prática.

5.1 **Burlando a autenticação de usuário**

No primeiro cenário o pseudo programa, alcançou os resultados esperado a partir da ferramenta *OllyDBG,* demonstrado da seguinte maneira:

Ao abrir o executável do pseudo programa é aberta uma tela simples, que espera do usuário duas entradas para preencher os campos de *login* e senha, como mostrado na Figura 1.

Figura 1 – Tela inicial do pseudo programa



Fonte: própria (2013).

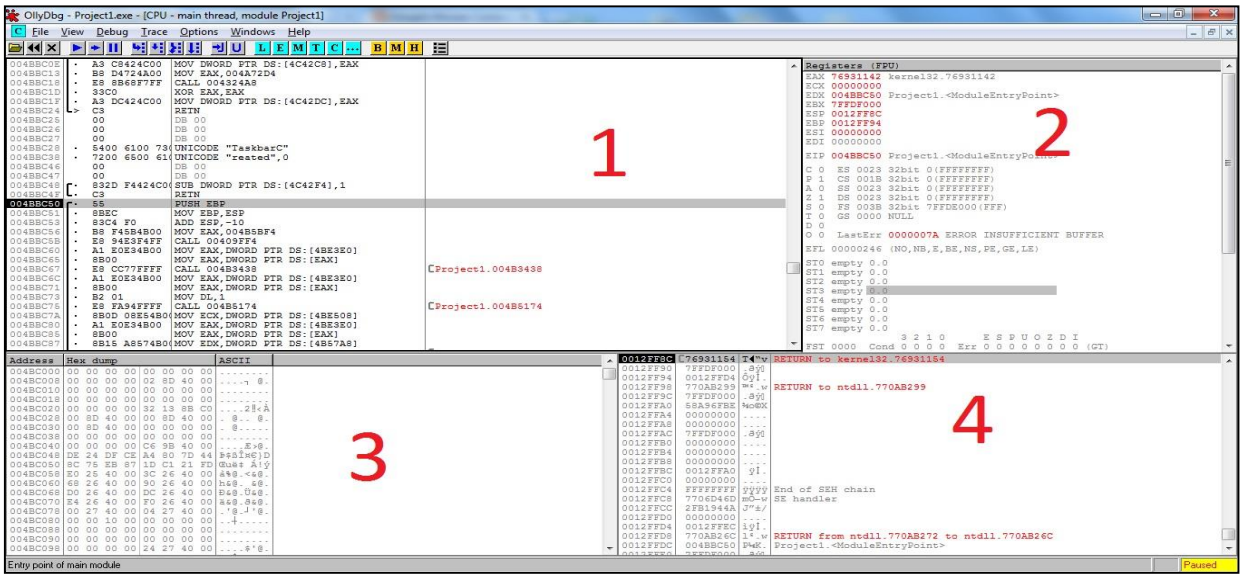
Ao entrar com valores que não correspondem aos programados, o programa lança uma mensagem de erro (**fig. 2**).

Figura 2 – Tela de mensagem de erro

Fonte: própria (2013).

Com isso, percebemos um cenário comum de autenticação em que quase todos os sistemas ou serviços que fazem uso de credencias. Então, para conseguir as credenciais desse programa foi aplicada a técnica de Engenharia Reversa a partir da ferramenta OllyDBG(2013). A figura 3 representa a ferramenta em execução depurando o cenário apresentado.

Figura 3 – Tela inicial do OllyDBG, com o primeiro cenário sendo depurado

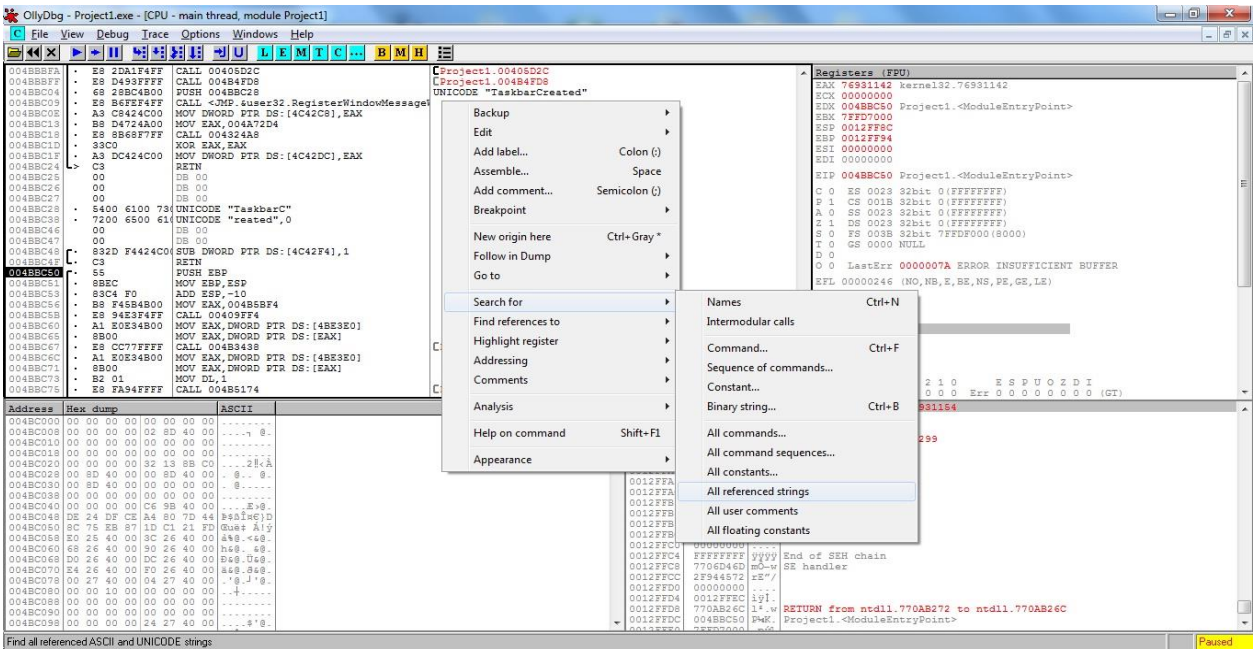


Fonte: própria (2013).

A interface é dividida em quatro regiões principais. A primeira região representa a tela principal do programa, onde é apresentado o *disassembly* do *software.* Na segunda região, atualizada a cada instrução, são mostrados os registradores, *flags* e outros valores. Na terceira região, é mostrada a memória física (RAM) destinada ao aplicativo, e na quarta região, o estado atual da pilha, que é amplamente utilizada durante as chamadas de função.

Quando o botão direito do *mouse* for pressionado, um menu será apresentado para que o usuário selecione uma funcionalidade. Para nosso caso o sub-menu "*All referenced strings*" foi escolhido como ilustrado na figura 4.

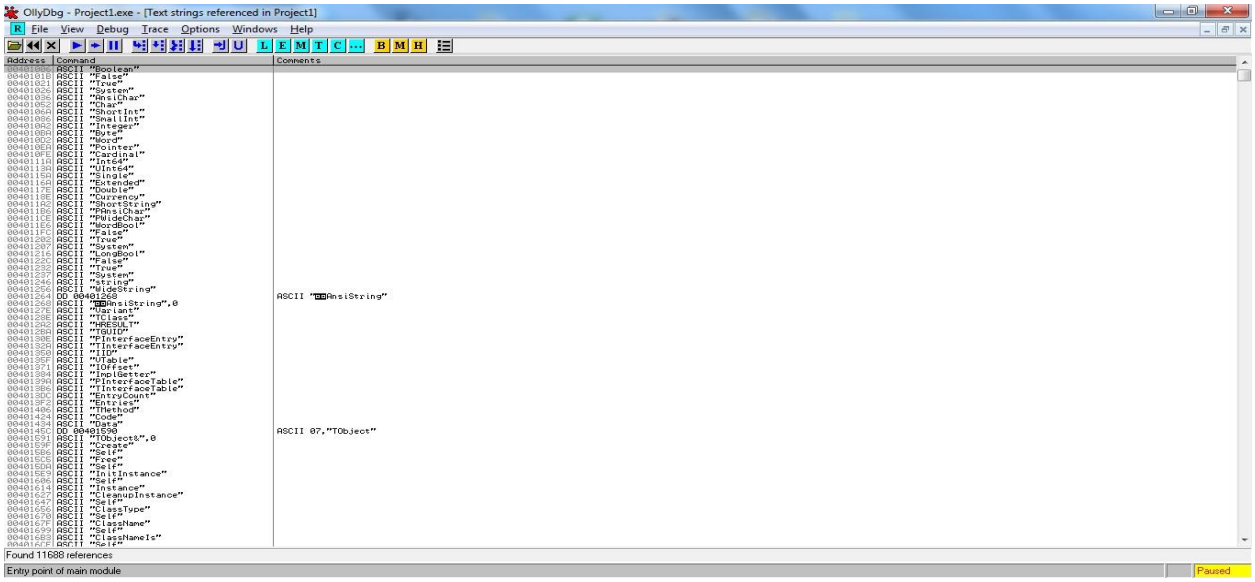
Figura 4 – Tela do OllyDBG selecionando uma opção do menu



Fonte: própria (2013).

Com isso, uma tela com todas as referências de *strings* utilizadas no programa é apresentada através de três colunas como ilustrado na figura 5. A primeira contém seu endereço de memória, a segunda contém a *string* utilizada, e a terceira, um comentário da ferramenta que serve para o entendimento da instrução.

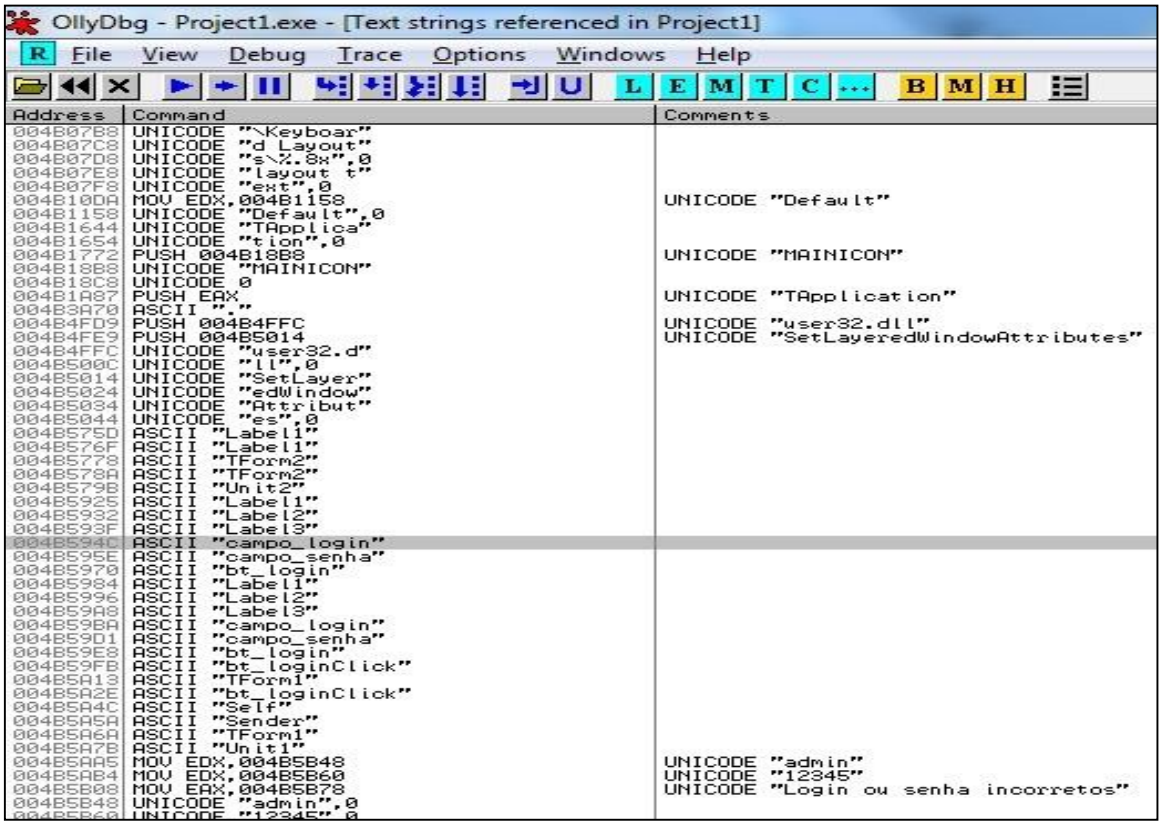
Figura 5 – Tela do OllyDBG referenciando de strings



Fonte: própria (2013).

Logo, todas as *strings* utilizadas no pseudo programa foram apresentadas, e com uma busca sequencial e um pouco de paciência, os valores que correspondem ao login e senha são encontrados como mostra a figura 6.

Figura 6 – Tela do OllyDBG referenciando strings, e localizando login e senha.



Fonte: própria (2013).

No endereço de memória "*0x004B5AA5",* existe a instrução *MOV* referência de memória "*0x004B5B48"* que possui o valor "*admin*", para o registrador "*EDX*" do processador. E no endereço de memória seguinte, o mesmo acontece para o endereço "*0x004B5B60*" que tem o valor "*12345*". Atribuindo esses valores como credenciais no pseudo programa, a tela de boas-vindas é apresentada.

Apenas com o código em linguagem de máquina (executável do programa) foi possível alcançar o resultado esperado para o cenário descrito anteriormente, em que era esperada a descoberta das credenciais que dão acesso ao programa, com a utilização da ferramenta de Engenharia Reversa.

Nesse contexto, uma técnica anti-reversão seria a eliminação de informações simbólicas, classificado por Eilam (2005). Em que o mais óbvio passo para afastar reversores é a eliminação de qualquer dados sensíveis no programa compilado e dados sensíveis escritos no código fonte. Estas informações podem ser extremamente úteis e devem sempre ser

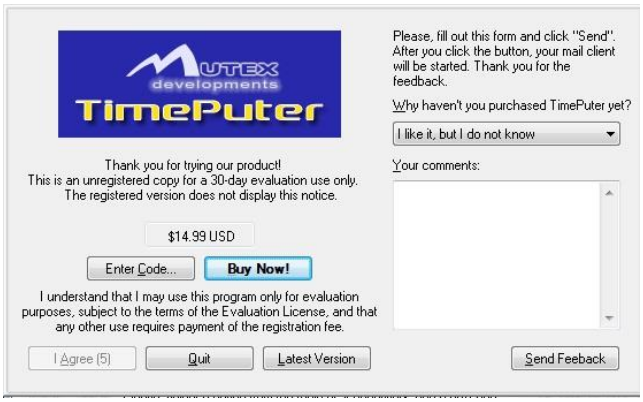
renomeadas e substituídas por sequências de caracteres que desvinculem o nome de seu significado.

5.2 **Burlando o período de avaliação**

No segundo cenário o software TimePutter, após ter sido baixado da internet e instalado em máquina com ambiente *windows*, foi submetido a ferramenta IDA-Pro para ser dissasemblado. O seu funcionamento foi analisado para verificar os resultados esperados.

Ao executar o software "alvo" é exibida uma tela que mostra a necessidade da compra de uma licença para uso continuo do software, como mostra a figura 7. Havendo à possibilidade de utilizar por trinta dias para testes, pressionando o botão "*I agree*".

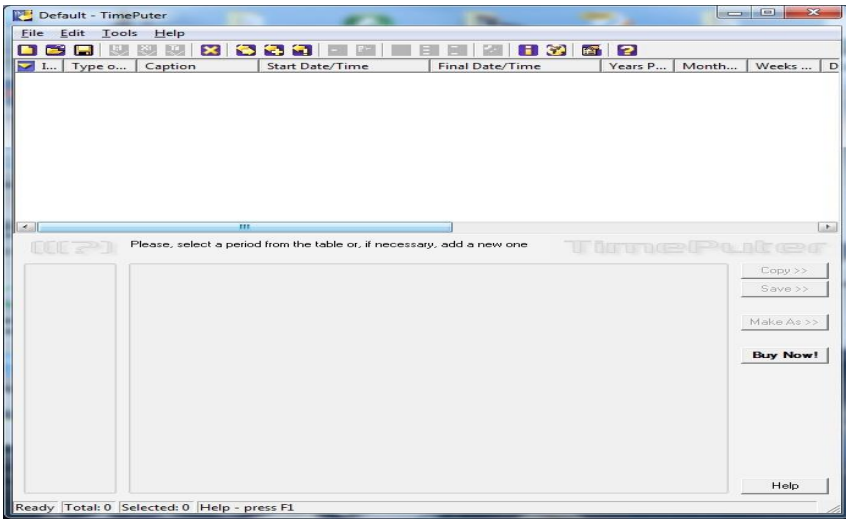
Figura 7– Tela de registro do TimePutter



Fonte: própria (2013).

A tela principal do programa (**fig. 8**) só é apresentada se estiver dentro do prazo de trinta dias, ou se efetuada a compra da licença.

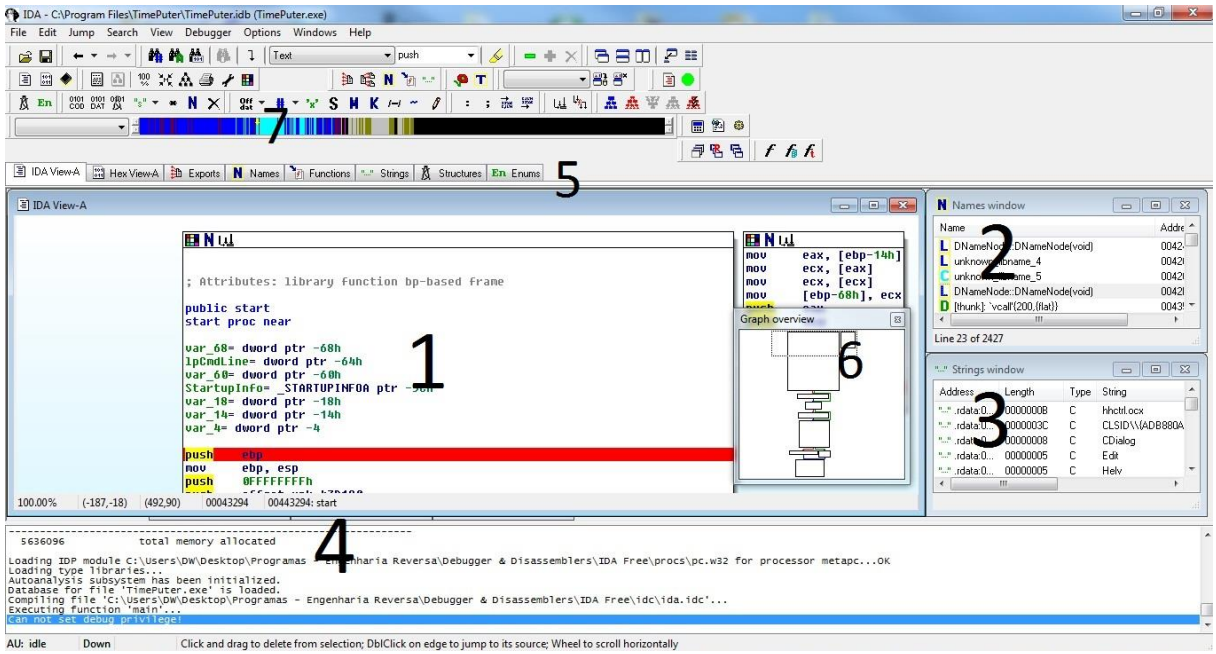
Figura 8 – Tela principal do TimePutter



Fonte: própria (2013).

Com isso, para fins acadêmicos, e com intuito de aprender como proteger seu software de usuários mal intencionados que possuem conhecimentos aprofundados, o TimePutter foi submetido a ferramenta “IDA-pro” que por sua vez, depurou e disassemblou todo o código binário do “software-alvo”.

Figura 9 – Tela inicial do IDA, executando o TimePutter



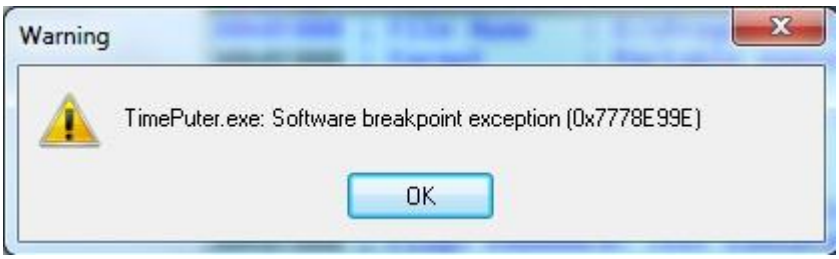
Fonte: própria (2013).

A figura 9 apresenta a interface principal do IDA depurando o “*software*-alvo”, com várias pequenas janelas que mostram informações necessárias para o entendimento de seu funcionamento.

A janela 1 mostra todo o código na linguagem *assembly* e comentários feitos pela ferramenta, sendo possível colocar breakpoints em qualquer linha de código. A janela 2 mostra uma resolução dos nomes usados que aparecem no decorrer do código e endereço. A janela 3 apresenta todas as *strings* encontradas no decorrer do programa com seu tamanho e endereço na memória. A janela 4– Barra de notificações–mostra algumas informações do comportamento da ferramenta. A janela 5– Abas de navegação– mostra diferentes tipos de informação do resultado da análise da ferramenta como: o código na linguagem *assembly*, o código em hexadecimal, as importações e exportações feitas pelo programa, *strings*, declarações de funções, entre outras. A janela 6– Gráfico –ilustra as funções em *assembly* na forma de uma estrutura que facilita o entendimento do programa. Finalmente, a janela 7– Barra –mostra, em cores, a análise do programa que, se estiver mudando, significa que o código binário ainda está sendo analisado.

Aplicando as funcionalidades da ferramenta, o código em *assembly* fica disponível para estudo. Depura e executado o programa instrução por instrução, a seguinte mensagem é mostrada como na figura 10.

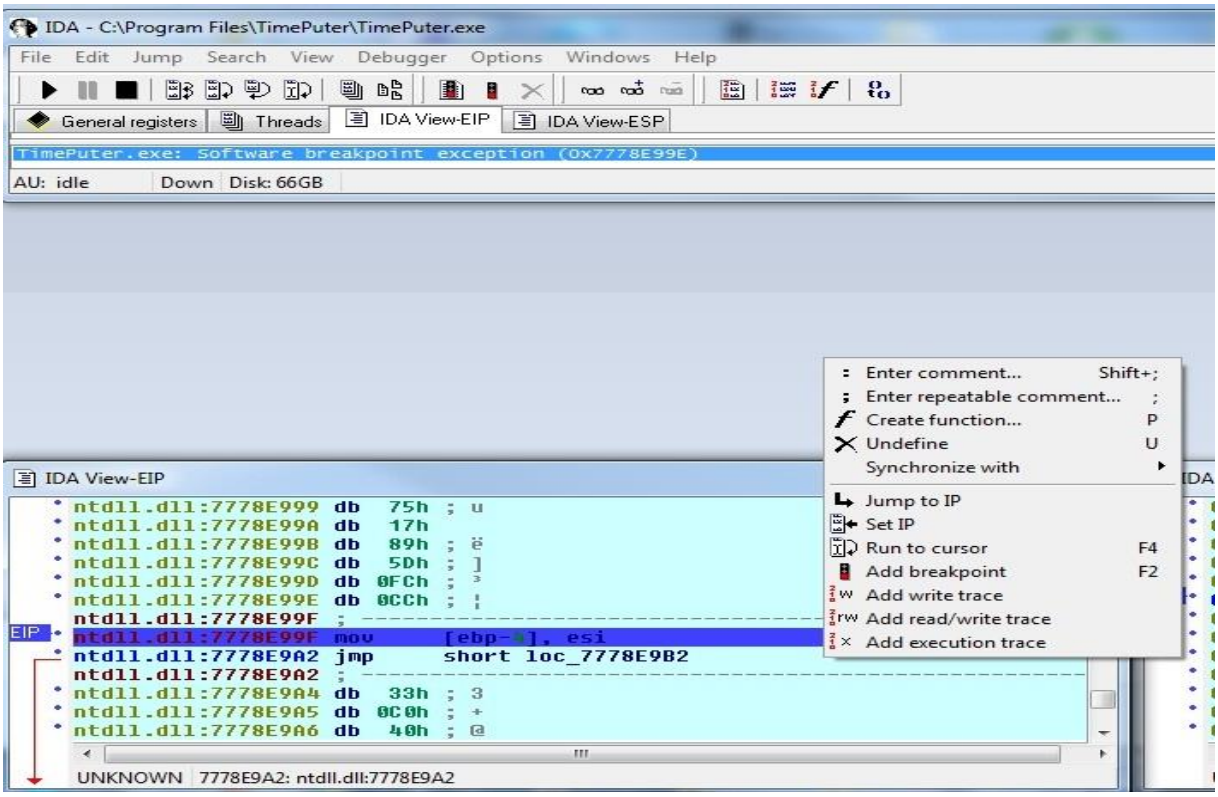
Figura 10 – Tela de exceção



Fonte: própria (2013)

Esta exceção é lançada ao tentar atribuir um ponto de parada e debugar o programa entrando em uma biblioteca do Windows (ntdll.dll). Para driblar essa restrição, basta pular a instrução, como mostra na figura 11, clicando com o botão direito em cima da instrução desejada e escolhendo a opção “Set IP”.

Figura 11 – Tela de execução do IDA, passando pela biblioteca do Windows (ntdll.dll)

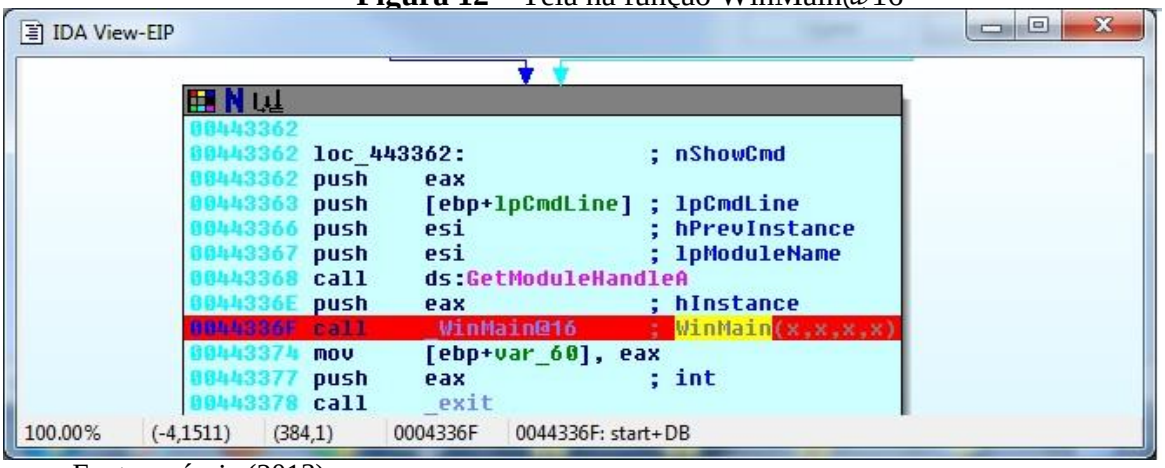


Fonte: própria (2013).

Após pular a instrução da exceção, o objetivo passa a ser localizar a função que chama a tela que queremos remover.

Para executar instrução por instrução no *debbuger*, basta pressionar a tecla F7. Por outro lado, a tecla F8 executa a instrução e as rotinas por completo, que foi o método para encontrar a primeira função que chama tela*callWinMain@16*, como mostra a figura 12. Logo a tecla F7 é pressionada para entrar na rotina dessa função.

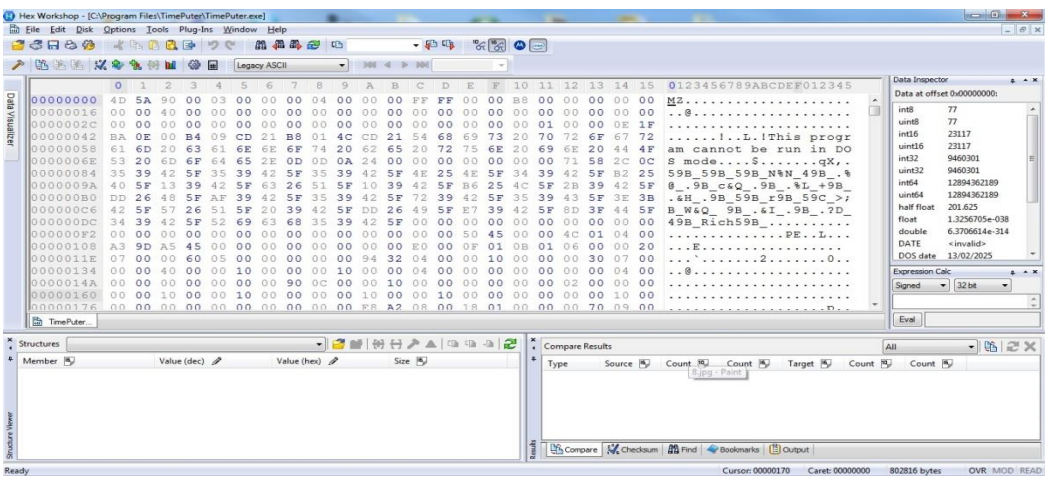
Figura 12 – Tela na função WinMain@16



Fonte: própria (2013).

Seguindo a rotina callWinMain@16, foram localizadas as seguintes funções subsequentes que abrem a janela: call sub_45BBC3, call dword prt[eax+50h] e, a última e principal rotina, call 455829. Descoberta a função chamadora da tela, é preciso inutilizá-la. Para isso foi preciso visualizar na aba Hex-view para obter seu valores *hexadecimais* correspondentes: E8 1B 35 04 00. Dessa forma, foi necessário preciso trocar esses valores para "90" que é uma instrução que o processador interpreta como "não fazer nada". Para isso, foi utilizada a ferramenta Hex Workshop, figura 13.

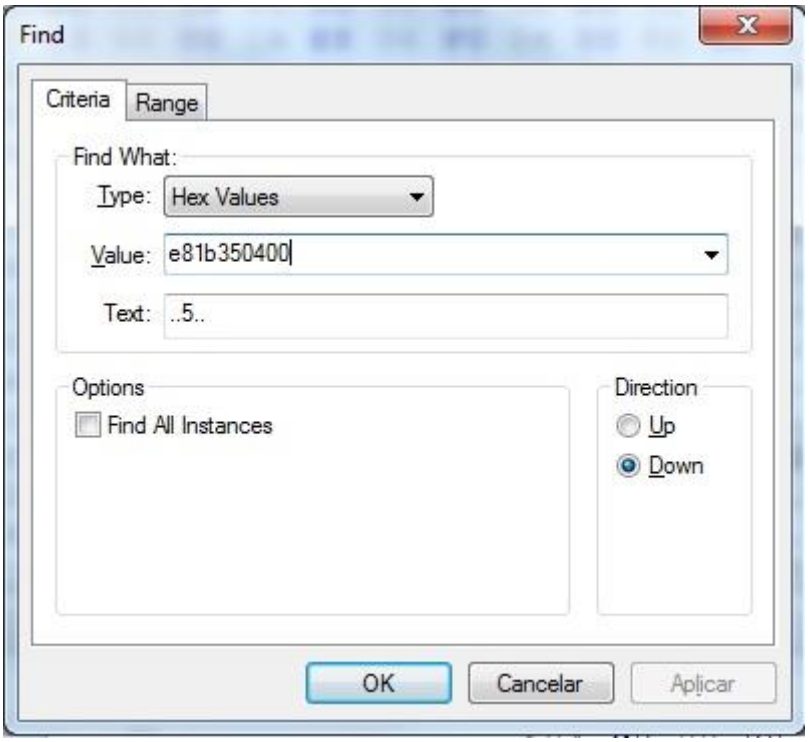
Figura 13 – Tela inicial da HexWorkShop visualizando o "software-alvo"



Fonte: própria (2013).

A figura 14 mostra a tela do programa Hex Workshop, que possibilita a alteração do código binário a partir da alteração dos seus valores hexadecimais correspondentes. Com isso, foram localizados os valores pressionando-se a tecla “Ctrl + F” e inserindo-se no campo “*Value*” o código, como mostra na figura 14.

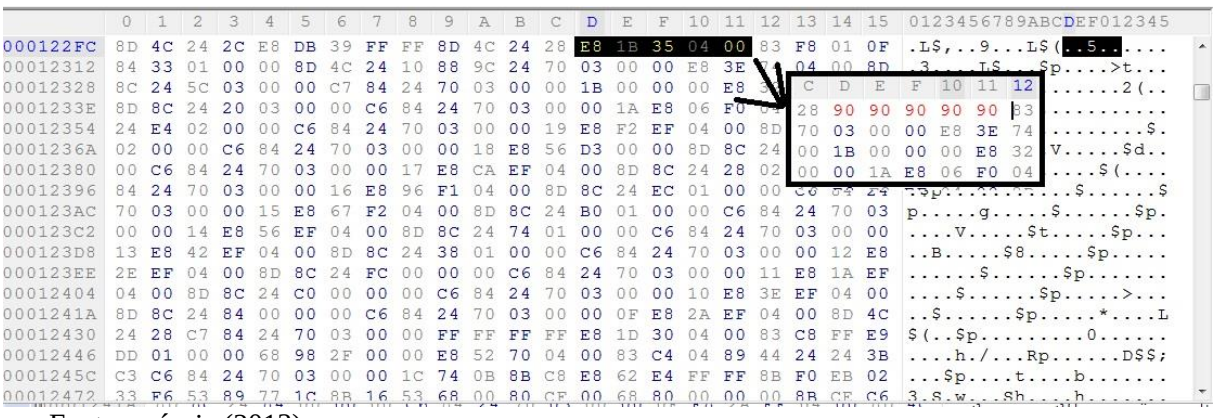
Figura 14 – Tela de localização do Hex Workshop



Fonte: própria (2013).

Após pressionado o botão “OK” a ferramenta indica a linha que possui o código pesquisado. Portanto, o próximo procedimento constituiu-se em substituir esses valores por “90” como mostra na Figura 15.

Figura 15 – Tela do Hex Workshop, substituindo os valores



Fonte: própria (2013).

Feito isso, basta ir ao menu “Arquivo>Salvar como” para criar um novo executável que remove apenas a chamada para a tela de registro. É importante ressaltar que o procedimento todo não foi removido, mas apenas a sua chamada, indo direto para a tela principal do programa.

Concluindo-se assim o segundo cenário, verificou-se seu objetivo principal que é remoção da tela de validação do “software-alvo”, partindo direto para a tela principal.

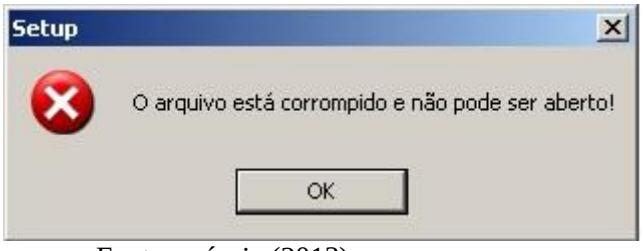
Após ter sido verificado a não existência da tela de validação, o executável modificado foi removido do disco rígido, pois esse procedimento foi realizado apenas para fins didáticos e não deve ser colocado em prática para fins de quebra de direitos autorais. Portanto, este trabalho ausenta-se da responsabilidade de incentivar tal prática.

Para esse contexto uma boa alternativa para anti-reversão seria utilizar as técnicas de ofuscação de código, que segundo Eilam (2005), se baseiam numa escrita de código que, do ponto de vista funcional, não altera em nada o programa, porém tornam a compreensão e leitura muito mais complexa, reescrevendo, reestruturando e alterando o fluxo de operações de tal maneira a confundir o reversor. Outra técnica que também pode ser usada é a de códigos anti-reversão embutidos, que consiste em inserir códigos específicos na aplicação que servem como armadilhas, no intuito de identificar, desabilitar e/ou danificar quaisquer ferramentas ou técnicas de reversão.

5.3 **Análise de *Trojan***

No terceiro cenário foi analisado, em uma máquina virtual, um *trojan* baixado após clicar em uma propaganda de e-mail. Ao ser executado, é lançada uma tela como ilustrado na figura 16, que apresenta uma mensagem de erro informando que o arquivo está corrompido. Então, foi analisado se ele realmente ele está.

Figura 16 – *Trojan*, mensagem de alerta.

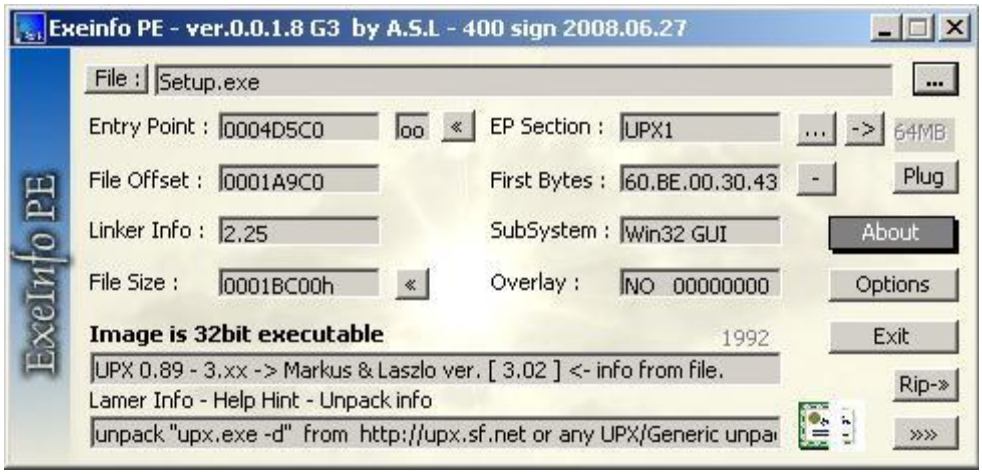


Fonte: própria (2013).

Primeiramente foi averiguado se o arquivo fora modificado por algum *packers*, garantindo que o código binário analisado seja o original. Após aberto com a ferramenta

ExeinfoPe foi comprovado que ele fora modificado por UPX, tornando necessário utilizar o *unpacker* especificado na ferramenta como mostrado na figura 17.

Figura 17 – Descobrindo proteção

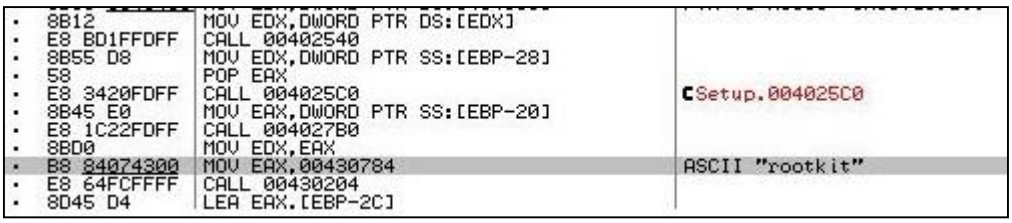


Fonte: própria (2013).

Quando o arquivo voltou a sua forma original, o binário ficou disponível para ser disassemblado, chegando a vez da ferramenta *OllyDBG*.

Com o OllyDBG aberto em um código que não se sabe como se comporta, e com má intenções, todo cuidado foi pouco. As informações foram buscadas primeiramente nas *Strings* que aparecem no decorrer do programa (como apresentado no cenário anterior) em busca de características suspeitas. Logo, alguns *Strings* já trouxeram informações suspeitas como mostrado na figura 18.

Figura 18 –*Stringrootkit*

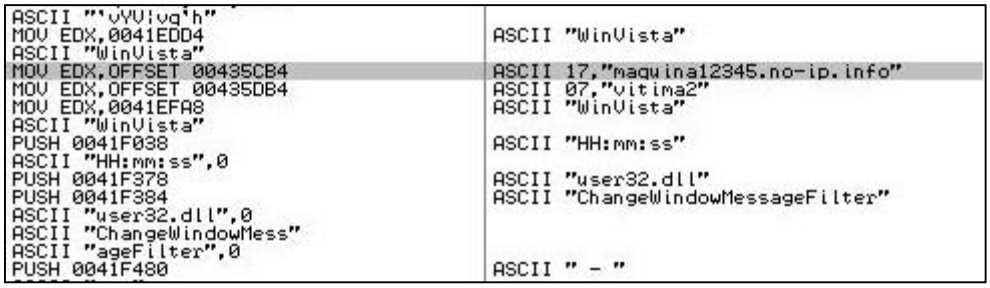


Fonte: própria (2013)

A instrução mostra a utilização de uma *string* chamada *rootkit*, que é utilizado por softwares maliciosos, que além de esconder processos, dão acesso contínuo e privilegiado a um computador (EILAM, 2005).

Em seguida, foi encontrada uma informação que serve para identificar que tipo de *malware* está sendo analisado, como ilustrado na figura 19.

Figura 19 – Endereço ip dinâmico

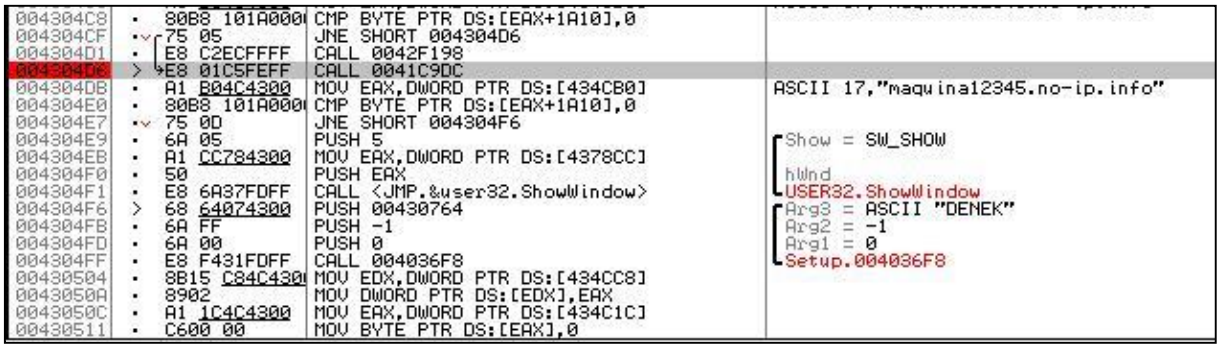


Fonte: própria (2013).

Percebe-se que existe um endereço IP dinâmico, possivelmente o endereço que o *hacker* utiliza para não ser identificado, definindo o destino para onde as informações serão encaminhadas, ou o endereço para um canal de comunicação com a vítima (CHAVES, 2004).

O trabalho continuou com a análise das instruções *assembly* para identificar as demais instruções ameaçadoras.

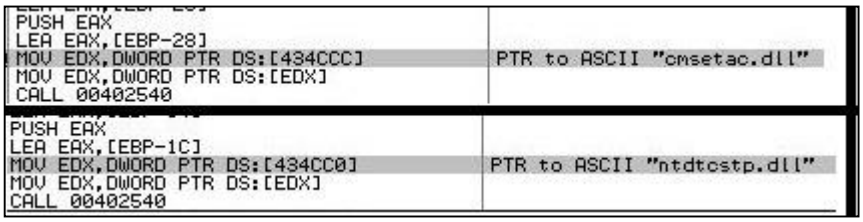
Figura 20 – Função que chama a mensagem de erro



Fonte: própria (2013)

Na figura 21, é mostrada a função (*CALL 0041C9DC*) que chama a tela de exibição da mensagem de erro, descrita no início do cenário, e abaixo o endereço utilizado pelo atacante. Mais abaixo são apresentadas as instruções que geram duas *dlls*, são elas: cmsetac.dll e ntdtcstp.dll (figura 22).

Figura 21 – Instruções envolvendo as *dlls*



Fonte: própria (2013)

Essas *dlls* são geradas e colocadas na pasta onde está o executável, e como o *windows* trabalha com *dlls*, quando algum arquivo for executado, ele executará essas *dlls*. Estas, que são usadas para instanciar um servidor que dá acesso ao atacante.

Dessa forma, foi concluído que o *trojan* analisado através das ferramentas de engenharia reversa é do tipo *backdoor* e *rootkit*. Ao ser executado, ele abre um canal de comunicação com o endereço do atacante: “maquina12345.no-ip.info”.

6 CONCLUSÃO

No contexto deste trabalho é abordada a utilização da Engenharia Reversa com auxílio de ferramentas de software, disponibilizando esse conhecimento como material de estudo para iniciantes entusiastas que se prontificam em descobrir como funciona um *software* por dentro, analisando suas instruções detalhadamente e interagindo com o centro de processamento de um computador.

A realização deum estudo sobre a estrutura e funcionamento de um programa, que é extremamente importante na hora de qualificar um software, torna possível ao programador/analista otimizar o desempenho, e a segurança dos sistemas desenvolvidos.

A análise detalhada de programas no ambiente *windows possibilitou* conhecer os procedimentos internos em um processador com arquitetura Intel x086, apresentar as informações estruturadas na memória RAM, verificar, inspecionar, e também a alterar valores chave. Além do entendimento, o estudo proporcionou uma base simplificada sobre o assunto, já que esta é um pouco escassa, apresentando a forma como os dados são organizados na linguagem *assembly* e traduzindo do formato binário, interpretado pelo processador, a partir das ferramentas de reversão.

Com isso, este trabalho apresentou a análise e utilização das ferramentas de Engenharia Reversa aplicadas a *softwares*, abordando como se comporta o código binário de algumas aplicações. Pretende-se com isso, promover uma base que ajude a aprimorar a eficiência e/ou a segurança do software, pensando em técnicas de proteção para garantir a integridade dos dados do seu produto ou de forma diferente, recuperar o código de um programa que não é mais desenvolvido.

Como trabalho futuro propõe-se, com base no conhecimento obtido neste trabalho, o desenvolvimento de uma ferramenta de Engenharia Reversa que disassemble o código binário de um executável e o recompile para uma linguagem de alto nível especificada, além de aplicar técnicas de proteção diretamente no código com objetivo de elevar seu nível de segurança comparado ao código binário original.

REFERÊNCIAS

AFONSO, V. M. et al. **Técnicas para análise dinâmica de malaware.** 2011. Disponível em: <http://www.ppgee.unb.br/sbseg2011/resources/downloads/minicursos/91936.pdf>. Acesso em: 22 nov. 2011.

BOOMERANG. 2012. Disponível em: <http://boomerang.sourceforge.net/>. Acesso em: 11 dec. 2012.

BRASIL. **Lei n° 9.609, de 19 de fevereiro de 1998**. Dispõe sobre a proteção da propriedade intelectual de programa de computador, sua comercialização no País, e dá outras providências. Diário Oficial da República Federativa do Brasil, Brasília, DF, 19 fev. 1998. Disponível em: <http://www.abes.org.br/old/consjuri/legislac/lei9609.htm>. Acesso em: 19 out. 2011.

CAMPOS, A. **"Sistema de Segurança da Informação: controlando os riscos",** 2ª edição, Visual Books, 2005.

CIFUENTES, C**.** Reverse Engineering and Computing Profession, **IEEE The Profession,** 2001.

CHAVES, C. H. P. C.; MONTES, A. "Backdoors e Canais Dissimulados: uma metodologia para detecção." **Anais do VI Simpósio sobre Segurança em Informática** (SSI'2004), 2004.

CHIKOFSKY, E.; CROSS II, J. **Reverse Engineering and Design Recovery:** A Taxonomy IEEE Software. 13-17, 1990.

CHRISTENSEN, J.; BANDYOPADHYAY, A.Reverse engineering of clear solids using refractive index matching.**RapidPrototypingJournal.** v. 6, n. 2, p.87-96, 2000.

DCC. 2012. Disponível em: <http://www.itee.uq.edu.au/~cristina/dcc.html/>. Acesso em: 11 dec. 2012.

DIAS, A. B. **Engenharia Reversa: uma porta ainda aberta**. Porto Alegre, 1998. Disponível em: <http://www.abepro.org.br/biblioteca/ENEGEP1997_T6109.PDF>. Acesso em: 20 out. 2011.

DICKIN, P. **"Reverse Engineering regains popularity".** IEE Review.v. 42, 1996.

EILAM, E. Reversing: Secrets of Reverse Engineering.Indianápolis, Estados Unidos da America: Wiley Publishing, Inc., 2005. 589p.

EU. (2001). **Directive 2001/29/EC of the European Parliament** and of the Council of 22 May 2001 on the Harmonisation of Certain Aspects of Copyright and Related Rights in the Information Society. Brussels: European Commission. Disponível em: <http://eurlex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32001L0029:EN:html> Acesso em: 23 out 2011.

EXEINFO. 2012. Disponível em: <http://www.exeinfo.antserve.com/>. Acesso em: 15 dec. 2012.

GUILFANOV IDA Pro Disassembler. Disponível em:<http://www.datarescue.com/idabase/ida.htm> Acesso em 20out. 2011.

HEXWORKSHOP. 2013. Disponível em: <http://www.bpsoft.com>. Acesso em: 07 jan. 2013.

HOGLUND, G.; McGRAW, G.**Como quebrar códigos**: a Arte de Explorar (e Proteger) Software. São Paulo: Pearson Education do Brasil, 2006. 424p.

IDA PRO. 2013. Disponível em: <http://https://www.hex-rays.com/products/ida/support/download.shtml>. Acesso em: 07 jan. 2013.

ILDASM. 2013. Disponível em: <http://msdn2.microsoft.com/en-us/library/f7dy01k1(vs.80).aspx>. Acesso em: 19 jan. 2013.

HVIEW. 2013. Disponível em: <http://www.hiew.ru>. Acesso em: 07 jan. 2013.

INTEL. Intel Architecture Software Developer's Manual: Volume 2: Instruction Set Reference, A-Z. The United States. 2013.

JAPÃO**,** Lei no. 47/1993.Vide Rahn e Heath, What is Japanese about the Japanese Unfair Competition Act, 25 Int' **Rev. Indus. Prop**. & Copyright L. 343, 1994.

JARDINI, A. et al. **Integração de prototipagem rápida e Engenharia Reversa no desenvolvimento rápido do produto**. Caxias do Sul, 2011. Disponível em: <http://alvarestech.com/temp/cobef2011/grima.ufsc.br/cobef2011/media/trabalhos/COF11-0291.pdf>. Acesso em: 27 out. 2011.

LIMA, C. B. **"Engenharia Reversa e Prototipagem Rápida Estudos de Casos"**. Dissertação (Mestrado), Unicamp, Campinas, São Paulo, 2003.

MARTINS, B. C. "Cooperação e livre fluxo da informação: A influência da cultura hacker na definição dos padrões da Comunicação Mediada por Computador." Encontro Anual Da Associação Nacional Dos Programas De Pós-Graduação 15, 2007.

MÜLLER, H. **A Visual Tool for understanding legay systems.**UNIVERSITY OFVICTORIA, BC, Canada. Disponível em:<http://www.rigi.csc.uvic.ca> Acesso em: 22 out. 2011.

NOVAIS, E. R. A.; PRADO, A. F. **Reengenharia de software orientada a componentes distribuídos**. Rio de Janeiro, 2001. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbes/2001/015.pdf>. Acesso em: 20 out. 2011.

OLLYDBG. 2013. Disponível em: <http://www.ollydbg.de>. Acesso em: 19 jan. 2013.

PEID. 2012. Disponível em: <http://www.downloadsource.net/2819/PEiD/>. Acesso em: 15 dec. 2012.

PENTEADO, R.D. **Um Método para Engenharia Reversa Orientada a Objetos.** São Carlos, 251 f, 1996. Tese (Doutorado). Universidade de São Paulo, 1996.

PRESSMAN, R. S. **Engenharia de software.** São Paulo: Mkron, 1995. Project Management.

PROJECTLOGIN. , 2012. Disponível em: <http://www.4shared.com/file/0yxZia2p/Project1.html?>. Acesso em: 19 dec. 2012.

PUNTAMBEKAR, N. V.; JABLOKOW, A. G., SOMMER III, H. J."Unified review of 3D model generation for Reverse Engineering"**. Computer Integrated Manufacturing Systems.** v. 7, p. 259-268,1994.

SOFTICE. , 2012. Disponível em: <http://myweb.tiscali.co.uk/reverseengineering/html/softice.html>. Acesso em: 15 dec. 2012.

QUILICI, A.Reverse Engineering of Legacy Systems: A Path Toward Success -Proceedins of the 17Ih Annual International Conference on Software Engineering. IEEE Press**. Seattle. WA,** p. 331-336, 1995.

REKOFF, M. G. On Reverse Engineering. IEEE Transactions on Systems, **Man, and Cybernetics,** v.15, n.2, 1985.

SOBH, T. M, et al**.** Industrial Inspection and Reverse Engineering. CAD-Based Vision Workshop. **Proceedings of the 1994 Second. IEEE.** p. 228-235, 1994.

TIMEPUTER. , 2012. Disponível em: <http://www.superdownloads.com.br/download/121/timeputer/>. Acesso em: 11 dec. 2012.

TROJAN. , 2012. Disponível em: <http://https://mail.google.com/mail/u/0/?shva=1#inbox>. Acesso em: 07 jan. 2013.

USA. **The digital millennium copyright act of 1998**, Pub. L. n. 105-304, 1998.

VARADY, T., MARTIN, R. R.; COXT, J**.** "Reverse Engineering of geometric models – an introduction**". Computer-Aided Design.** v. 29, p. 255-268, 1997.

VALLIM, D. A. **Extração de regras de integridade referencial**. 2002. Dissertação (Mestrado em Informática) - Universidade Federal do Paraná. Curitiba.

VÊNERE, G. **Engenharia Reversa de Código Malicioso**. São Paulo: Escola Superior de Redes/Rede Nacional de Pesquisa. 2009, 149P

YIN, H. et al. "Panorama: capturing system-wide information flow for malware detection and analysis." **Proceedings of the 14th ACM conference on Computer and communications security**. ACM, 2007.