UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

METODOLOGIA PARA INTERCEPTAÇÃO DE DADOS CIFRADOS APLICADA EM INVESTIGAÇÕES CRIMINAIS

RODRIGO ALVES CARVALHO

ORIENTADOR: Prof. Dr. FLAVIO ELIAS DE DEUS

CO-ORIENTADOR: Prof. Me. DINO MACEDO AMARAL

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA
ÁREA DE CONCENTRAÇÃO INFORMÁTICA FORENSE E SEGURANÇA DA INFORMAÇÃO

PUBLICAÇÃO: PPGENE.MD – 95 A/12

BRASÍLIA/DF: FEVEREIRO/2012

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

METODOLOGIA PARA INTERCEPTAÇÃO DE DADOS CIFRADOS APLICADA EM INVESTIGAÇÕES CRIMINAIS

RODRIGO ALVES CARVALHO

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE PROFISSIONAL EM INFORMÁTICA FORENSE E SEGURANÇA DA INFORMAÇÃO.

APROVADA POR:

FLAVIO ELIAS DE DEUS, Doutor, UnB
(ORIENTADOR)

ANDERSON CLAYTON ALVES NASCIMENTO, Doutor, UnB
(EXAMINADOR INTERNO)

HELVIO PEREIRA PEIXOTO, Doutor, DPF
(EXAMINADOR EXTERNO)

DATA: BRASÍLIA/DF, 06 DE FEVEREIRO DE 2012.

FICHA CATALOGRÁFICA

CARVALHO, RODRIGO ALVES
METODOLOGIA PARA INTERCEPTAÇÃO DE DADOS CIFRADOS APLICADA EM INVESTIGAÇÕES CRIMINAIS [Distrito Federal] 2012.
(xx), (91)p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2012).

Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Informática forense 2. Criptografia
3. Interceptação de dados 4. Investigação

I. ENE/FT/UnB. II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

CARVALHO, R. A. (2012). METODOLOGIA PARA INTERCEPTAÇÃO DE DADOS CIFRADOS APLICADA EM INVESTIGAÇÕES CRIMINAIS. Dissertação de Mestrado, Publicação PPGENE.MD – 95 A/12, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, (91)p.

CESSÃO DE DIREITOS

Rodrigo Alves Carvalho
Universidade de Brasília
Campus Universitário Darcy Ribeiro – CEP 70910-900
Brasília/DF/Brasil

Para minha família, meus amigos
e todos que lutam de verdade
por um Brasil melhor.

AGRADECIMENTOS

A Deus, pelas oportunidades que me tem concedido;
A meus pais e irmãos, que mesmo longe se fazem presentes em minha vida;
A minha esposa, pelo compreensão e motivação ao longo destes 2 anos;
Ao Professor Dino, que sempre se mostrou disposto a ajudar;
Aos colegas da turma do Mestrado, com quem dividi momentos sérios e momentos não tão sérios assim;
Aos organizadores deste curso de Mestrado, por acreditarem e tornarem realidade esta iniciativa.

RESUMO

METODOLOGIA PARA INTERCEPTAÇÃO DE DADOS CIFRADOS APLICADA EM INVESTIGAÇÕES CRIMINAIS

Autor: Rodrigo Alves Carvalho
Orientador: Prof. Dr. Flavio Elias de Deus
Programa de Pós-graduação em Engenharia Elétrica
Brasília, fevereiro de 2012

A aplicação de técnicas criptográficas para proteger dados está sendo cada vez mais disseminada entre usuários de computador, tanto em nível doméstico, empresarial e governamental. E isto está impondo uma dificuldade cada vez maior para a recuperação de dados pelos peritos criminais da área de informática. Dados cruciais que podem incriminar ou inocentar determinada pessoa ou organização acerca do cometimento de algum crime tornam-se inúteis para a justiça se estiverem cifrados com algoritmo e senha fortes, pois sua recuperação torna-se inviável. Ainda, a legislação brasileira, ao afirmar que não se pode exigir que nenhum cidadão produza provas contra si mesmo, elimina a possibilidade de obrigar algum suspeito a fornecer a senha que decifre seus dados criptografados. Assim, este trabalho propõe uma metodologia que vise a recuperação destes dados antes de serem cifrados, durante uma interceptação telemática autorizada judicialmente. Para isso, além da metodologia, foi implementado um programa que, ao ser instalado no computador de um suspeito, captura informações inseridas por ele e as envia de maneira segura para um servidor remoto. Os testes realizados mostraram que foi possível obter dados como senhas e conteúdo de bate-papo cifrado às claras, tornando esta metodologia uma alternativa viável, de baixo custo, exclusiva e adaptada para ser utilizada por órgãos de investigação brasileiros.

ABSTRACT

METHODOLOGY FOR ENCRYPTED DATA INTERCEPTION APPLIED IN CRIMINAL INVESTIGATIONS

Author: Rodrigo Alves Carvalho
Advisor: Prof. Dr. Flavio Elias de Deus
Programa de Pós-graduação em Engenharia Elétrica
Brasília, February of 2012

Cryptographic techniques are increasingly being applied in systems, partitions and files by computer users, whether they are home, bussiness or government users. And this is imposing a great difficulty for computer forensics analysts to retrieve these data. After all, crucial data that can incriminate or make someone innocent about a crime commitment becomes useless if it is encrypted with strong cryptographic algorithm and password, as it is innacessible. Moreover, Brazilian laws eliminate the possibility to force a suspect to provide the password to decrypt his encrypted data, as it states that no citizen must produce evidence against himself. Thus, this work proposes a methodology aimed at the recovery of the data before it gets encrypted, during a court authorized wiretap. Besides the methodology, it was implemented a program that, when installed on a suspect computer, capture information entered by him and sends it securely to a remote server. Tests showed that it was possible to obtain data such as passwords and plain contents of encrypted chats, turning this methodology into a low-cost and viable alternative, exclusive and adapted for use by Brazilian law enforcement agencies.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

API Application Programming Interface
BLOB Binary Large Object
BSOD Blue Screen of Death
CBC Cipher-block Chaining
CIPAV Computer and Internet Protocol Address Verifier
CNG Cryptography API: Next Generation
DLL Dynamic Link Library
DNS Domain Name System
DPC Deferred Procedure Call
ECB Eletronic Code Book
FBI Federal Bureau of Investigation
HAL Hardware Abstraction Layer
HKLM Hive Key Local Machine
HTTP Hypertext Transfer Protocol
IP Internet Protocol
IRP I/O Request Packet
ISR Interrupt Service Routine
IV Initialization Vector
I/O Input/Output
KMCS *kernel* Mode Code Signing
KPP *kernel* Patch Protection
MAC Media Access Control
MTU Maximum Transfer Unit
NPI Network Programming Interface
PCI Peripheral Component Interconnect
PGP Pretty Good Privacy
PHP Hypertext Preprocessor
RSA Rivest Shamir Adleman
SET Social Engineering Toolkit
TCP Transmission Control Protocol
TLS Transport Layer Security
URL Uniform Resource Locator
WDK Windows Driver Kit
WSK WinSock *kernel*

1 INTRODUÇÃO

Um dos desafios da informática forense que, apesar de já existir há bastante tempo, começa a ser encontrado cada vez mais frequentemente, é a recuperação de dados protegidos.

Segundo (IASHCHENKO, 2002), a criptografia, ou estudo das cifras, foi por muito tempo uma ciência secreta porque era utilizada, primordialmente, para garantir a segurança do Estado e segredos militares.

Entretanto, durante os anos 80 e 90, a criptografia se tornou um importante aspecto das comunicações comerciais. O crescimento da computação pessoal e da internet mudaram a criptografia de uma exótica tecnologia exclusivamente militar para uma tecnologia crucial em comércio eletrônico (WHITEFIELD, 2007).

Notou-se que, apesar de não ser mais tratada como uma tecnologia de uso exclusivamente militar, o usuário comum, ao utilizar seu computador pessoal, ainda ignorava a existência de tais sistemas de segurança, por vários motivos: por falta de interesse, por não possuir acesso a tal tecnologia, ou mesmo por não achar necessário.

Neste cenário, infere-se que, até alguns anos atrás, eram poucos os casos em que era necessário "desproteger" os dados de uma pessoa para, só então, ter acesso aos mesmos; a maioria dos dados não possuía quaisquer barreiras para sua visualização, como senhas de acesso, criptografia, e outros.

1.1 JUSTIFICATIVA

Existe uma grande oferta de dispositivos com criptografia embarcada (celulares, notebooks, entre outros) e de aplicativos para criptografia e autenticação no mercado. Podem ser encontrados aplicativos com diferentes faixas de preço (alguns inclusive sem custo,

como o *GPG*[1] - *Gnu Privacy Guard* e o *Truecrypt*[2]); direcionados para objetivos específicos (proteção de e-mail, de servidores de acesso remoto, de arquivos, de discos rígidos, de comunicação falada, de bate-papo instantâneo, de comércio eletrônico, entre outros); e para diferentes tipos de usuários (uso pessoal, corporativo, governamental).

Tais dispositivos e aplicativos já são objetos de busca e apreensão ou interceptações telemáticas, e a tendência é que essa situação torne-se cada vez mais comum. Ainda, a depender do nível de sofisticação e dos cuidados tomados nas escolhas de algoritmos, chaves e senhas utilizados nos mesmos, ataques de dicionário, força bruta, ou mesmo a exploração de uma possível falha de segurança tornam-se inviáveis, ou no mínimo extremamente complicadas e com poucas chances de sucesso, pois envolvem:

- Pesquisar sobre as tecnologias empregadas;
- Reproduzir o ambiente original em que o aplicativo funcionava;
- Entender o funcionamento do aplicativo ou dispositivo de segurança;
- Realizar engenharia reversa e estudar o código, a fim de descobrir alguma falha no aplicativo ou alguma brecha que possa ser utilizada em um ataque de força bruta, de dicionário ou outro;
- Dependendo da complexidade da senha, a lista a ser utilizada num ataque de força bruta será extremamente extensa.

Todos esses fatores expostos atingem diretamente a eficácia das investigações policiais e dos exames periciais. A utilização de um aplicativo que cifra conversas instantâneas, por exemplo, torna inútil boa parte de uma interceptação telemática autorizada judicialmente. Um disco rígido completamente cifrado pode comprometer fatalmente sua análise.

O método ideal para se ter acesso a tais dados é obter a chave ou qualquer outro dado/dispositivo que desproteja os mesmos. Isto elimina a necessidade de tomar os passos descritos, e aumenta consideravelmente as chances de sucesso na decifração dos dados. Para

[1] Aplicativo de código aberto utilizado tanto para criptografar quanto para autenticar dados e comunicações em uma rede de computadores.

[2] TrueCrypt é um aplicativo em código aberto utilizado para proteger dados. Ele pode tanto criar volumes criptografados (contêineres) que podem ser montados como unidades virtuais, quanto criptografar um disco inteiro, exigindo uma chave para decifrá-lo antes do *boot*.

atingir este objetivo, a engenharia social, que consiste no ato de manipular uma pessoa para que esta realize uma ação que possa ou não ser de seu interesse próprio (HADNAGY; WILSON, 2010), pode ser bastante útil.

De acordo com (OPPENHEIMER et al., 1997), a segurança computacional, assim como uma corrente, é tão forte quanto seu elo mais fraco. Em muitas organizações, os usuários do sistema são o elo mais fraco na cadeia de segurança.

Vários são os motivos que fazem do fator humano, comumente, o elo mais fraco em um sistema de segurança: falta de treinamento para operar determinados aplicativos; falta de interesse em aprender novas tecnologias e seu funcionamento; negligência com a segurança (não aplicação de *patchs*[3], por exemplo); entre outros.

Assim, em sistemas com forte segurança, técnicas de engenharia social aliadas à utilização de alguma ferramenta destinada à recuperação de chaves / senhas pode ser bastante eficaz.

Apesar da seriedade deste problema e das possibilidades existentes, nota-se uma carência de metodologia a ser aplicada em casos que envolvam dados protegidos, explicada talvez pela criptografia ser um empecilho relativamente novo em investigações policiais. Esta carência, somada à urgência em tornar os dados inteligíveis, pode acarretar consequências como:

- Possível vazamento de informações sigilosas ao utilizar algum *software* proprietário (em que não seja possível avaliar o código fonte);
- Possível descoberta da investigação pelo investigado, se cuidados necessários não forem tomados;
- Possível detecção de *softwares* proprietários por ferramentas antivírus;

Assim, é mais indicado o desenvolvimento de metodologia e ferramentas próprias, baseados na realidade interna da corporação ao qual servirão.

[3]Pequeno programa de computador criado para atualizar ou sanar problemas de outros programas de computador, ou ainda otimizar sua performance. Para isso, modifica diretamente o código binário destes programas.

1.2 OBJETIVOS

O objetivo deste trabalho é desenvolver uma metodologia para recuperação remota de dados protegidos, a ser utilizada em investigações policiais e com autorização judicial.

Esta metodologia utiliza tecnologias de *rootkit*[4] e *drivers*[5] para sistemas *Windows* aliadas a técnicas de engenharia social embasadas em informações obtidas em investigações policiais. Envolve 3 etapas:

1. Estudo do alvo e definição das melhores técnicas de engenharia social a serem aplicadas, a depender das informações levantadas junto à investigação policial;
2. Instalação da *FID* - Ferramenta para Interceptação de Dados - no alvo, verificando seu correto funcionamento;
3. Recebimento dos dados enviados pelo programa.

Para a etapa 1 será criada uma metodologia que vise reunir e agrupar todas as informações sobre os indivíduos investigados, obtidas pelas equipes policiais. O objetivo é facilitar a escolha da técnica de engenharia social a ser utilizada para a instalação da *FID* no computador do investigado.

Para a etapa 2 será desenvolvida a ferramenta citada na etapa 1, que consistirá em um *driver* que armazene caracteres digitados no teclado, nomes de programas em execução e outras informações obtidas do sistema. Ao final, todos estes dados serão enviados cifrados a um servidor remoto, para que sejam aproveitados pela investigação policial.

Para a etapa 3, será implementada toda a estrutura para recebimento, decifração e disponibilização dos dados. Será necessário também de algum meio de controle de acesso às informações decifradas, pois são sigilosas. Este controle fará parte da metodologia proposta, e seu efetivo cumprimento será observado, juntamente com novas autorizações judiciais, para definir a continuação ou não da interceptação dos dados.

[4]Conjunto de programas e códigos destinados à espionagem e controle de um sistema, no qual mantêm uma presença permanente e dissimulada. (HOGLUND; BUTLER, 2006) (BLUNDEN, 2009)

[5]Software que provê uma interface para que o sistema instale e se comunique apropriadamente com os diferentes dispositivos físicos ou lógicos presentes no computador.

1.3 METODOLOGIA DE TRABALHO

O primeiro passo desse trabalho foi identificar as funcionalidades básicas que a *FID* deverá implementar, quais sejam: obtenção de dados do sistema, captura de dados digitados, identificação de programas em execução e envio de informações pela rede de maneira discreta e cifrada.

Pelo fato de ser uma tecnologia relativamente nova, o passo seguinte consistiu em pesquisar na internet por páginas, fóruns e artigos que tratassem de *rootkits*, em busca dos autores mais reconhecidos pela comunidade de segurança da informação e também de códigos existentes que implementassem algumas das funcionalidades especificadas.

A partir daí, esses códigos foram estudados, a fim de identificar as qualidades e as limitações de cada um. Ao mesmo tempo, livros que tratam dos assuntos *rootkits*, *drivers* e engenharia social ajudaram a sanar dúvidas de implementação e indicar as melhores maneiras de abordar o problema.

Então, foram implementados um protótipo funcional da *FID* e toda a infra-estrutura necessária para seu correto funcionamento. Há de se observar que nenhuma dessas etapas é "estanque": são abordadas em ciclo, visando aperfeiçoamento ou implementação de novas funcionalidades.

Ao final, foi desenvolvida uma metodologia dentro da qual o *FID* será utilizada. Para isso, três investigadores que trabalham rotineiramente com interceptações telemáticas foram entrevistados, com o objetivo de entender o funcionamento destas interceptações e também descobrir as principais dificuldades enfrentadas por eles. De posse desta informação, as etapas da metodologia foram definidas.

1.4 RESULTADOS ESPERADOS

Espera-se, a partir da definição de uma metodologia completa para obtenção de dados protegidos, desde a análise dos alvos até a disponibilização dos dados capturados aos analistas, que seu uso seja difundido em investigações policiais em que dados interceptados legalmente não estejam sendo aproveitados na persecução penal por estarem cifrados, aumentando a eficácia e a eficiência destas investigações.

1.5 LIMITAÇÕES

Este projeto não implementará o mecanismo de instalação da *FID* na máquina alvo: apesar de existirem alternativas que podem ser exploradas para este fim, a escolha de uma está ligada diretamente ao conhecimento e comportamento do usuário do sistema alvo, e também aos aplicativos e atualizações de segurança instalados no mesmo.

Além disso, a *FID* será desenvolvida apenas para sistemas de 32 *bits*. Esta limitação será explicada na seção 2.2.

1.6 ORGANIZAÇÃO DOS CAPÍTULOS

No capítulo 2, são apresentados conceitos fundamentais ao entendimento da ferramenta a ser desenvolvida, além dos programas utilizados na implementação. Também serão discutidas outras soluções existentes, similares à solução buscada neste projeto.

O capítulo 3 trata exclusivamente da implementação da *FID*, enumerando seus módulos, as funções dentro de cada um deles e apresentando diagramas que ilustram o fluxo de dados dentro da solução.

No capítulo 4 são detalhadas as etapas da metodologia sugerida, em que são explicadas a necessidade e o resultado esperado em cada uma delas.

O capítulo 5 apresenta a validação e os testes executados em um sistema em que se encontra instalada a *FID*, ilustrando o tráfego de dados cifrados por determinadas aplicações e os dados capturados pela ferramenta. Também são realizados testes a fim de verificar se aplicativos de detecção de *malware*[6] geram alertas quando executados concomitantemente à *FID*.

Já o capítulo 6 apresenta a conclusão do trabalho, explicando a contribuição conseguida e sugerindo tópicos de aprimoramento e de projetos futuros.

[6] *Software* utilizado para infiltrar em um sistema de computador alheio de forma ilícita, com o intuito de causar algum dano ou roubo de informações

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo, serão relacionadas algumas soluções similares à ferramenta a ser desenvolvida neste trabalho, e serão explicadas as características inerentes que impossibilitam ou tornam desaconselhável a adoção das mesmas por órgãos investigativos. Também serão explicados conceitos fundamentais para o entender o desenvolvimento e o funcionamento da solução, além de embasar algumas decisões tomadas no decurso do projeto. Finalmente, a última seção lista todos os programas utilizados e a função de cada um.

2.1 TRABALHOS CORRELATOS

A ideia deste projeto não é inédita. Pelo fato do problema tratado ser comum a diversos países, é bastante factível que vários deles já tenham desenvolvido ou estejam desenvolvendo soluções similares. Apesar de se desconhecer a existência de metodologias desenvolvidas para interceptação de dados criptografados, a utilização de aplicativos de espionagem não é novidade.

Ainda, existem aplicativos comerciais pagos, que são vendidos tanto para governos de países, quanto para empresas e pessoas. Finalmente, podem ser encontrados alguns códigos-fonte na internet, em sua maioria sem garantia de funcionamento e testados apenas pelos próprios desenvolvedores.

2.1.1 Soluções governamentais

Devido às grandes sensibilidade e polêmica do assunto, poucos governos admitem publicamente utilizar tais soluções. E, mesmo que utilizem, o intercâmbio de informações relacionadas se mostra bastante difícil, pelos mesmos motivos.

O governo da Alemanha é uma exceção: desde 2007, não apenas admite, mas incentiva seus cidadãos a baixarem e instalarem em seus próprios computadores o aplica-

tivo *Bundestrojaner*, com a justificativa de investigar e conter possíveis atentados terroristas (BUNDESTROJANER, 2011). Consiste basicamente de uma ferramenta de vigilância que envia dados capturados a um servidor remoto e também permite enviar comandos para o cliente.

Conforme relatório do (CHAOS, 2011), o *malware*[7] consiste de uma *DLL* [8] - *Dynamic Link Library* - e de um *driver*, e o o envio de tais comandos em modo "aberto" (sem criptografia) torna o aplicativo suscetível a ser controlado por terceiros não autorizados.

Outro exemplo é o *CIPAV* - Computer and Internet Protocol Address Verifier, utilizado pelo *FBI*. Essa ferramenta veio a público também em 2007, durante uma investigação do FBI. Segundo (LYNCH, 2011), ela captura informações como endereço *IP*, endereço *MAC*[9], sistema operacional instalado, serviços em execução, usuário logado, selos de data e hora, última *URL*[10] visitada, entre outros, da máquina em que se encontra instalada, e envia todas essas informações para servidores do *FBI*.

Além da dificuldade já relatada em obter seus códigos-fonte, para efeito de avaliação, comparação e até mesmo utilização, peculiaridades das rotinas do órgão policial e do poder judiciário de cada país, somadas à necessária compartimentação do assunto, torna recomendado o desenvolvimento de solução própria.

2.1.2 Soluções comerciais

Um aplicativo comercial difundido da internet e que, apesar de não ser direcionado a governos, implementa várias funcionalidades relacionadas a vigilância eletrônica, é o *Ardamax Keylogger*[11]. Esta ferramenta grava toda a atividade de usuário no sistema, e armazena num arquivo criptografado, para conferência posterior, ou envia por *email*. Porém, difere da solução buscada nesse projeto nos seguintes pontos:

- Seu código fonte é fechado, impossibilitando uma auditoria completa sobre seu real funcionamento ou possíveis vazamentos de informações;

[7] Nome genérico dado a softwares desenvolvidos para se infiltrar em computadores, causando algum prejuízo a este.

[8] Arquivos de biblioteca compartilhada, que possuem funções comuns utilizadas por diferentes aplicações.

[9] Endereço físico de 48 bits da interface de rede. O protocolo é responsável pelo controle de acesso de cada estação à rede *Ethernet*.

[10] Endereço de um recurso (um arquivo ou uma impressora, por exemplo) disponível em uma rede. Apresenta a seguinte estrutura: protocolo://máquina/caminho/recurso.

[11] http://www.ardamax.com/keylogger/

- É bem mais factível que sua assinatura já conste em pelo menos uma base de dados anti-vírus, tornando-a detectável;
- Seu objetivo principal é oferecer ao proprietário de determinado computador uma ferramenta para vigilância de terceiros que utilizam o referido computador.

Ainda, de acordo com uma comparação entre várias ferramentas do tipo *keylogger* realizada pelo sítio *keylogger.org*[12], nota-se que a maioria das ferramentas comerciais normalmente possuem um mesmo conjunto de funcionalidades básicas, quais sejam: interceptação de teclas digitadas e do conteúdo armazenado na área de transferência, captura de tela e de clique de mouse, e envio de dados pela rede. Algumas apresentam funcionalidades extras, como monitoramento do estado dos arquivos e da fila de impressão. Destas, as que não estão implementadas na *FID* podem vir a ser no futuro, após análise da necessidade das mesmas.

Das 8 melhores ferramentas, avaliadas pelo sítio, apenas uma delas, a *Spytech SpyAgent Stealth Edition*[13] permite instalação remota, e nenhuma delas captura a senha do usuário do *Windows*, dado capturado pela *FID* e que pode ser utilizado para recuperar as chaves criptográficas do sistema de proteção *EFS*[14] - *Encrypting File System* - utilizado em alguns sistemas *Windows*. Ainda, por serem comerciais, possuem as mesmas características intrínsecas: código fonte fechado e grande possibilidade de sua assinatura já constar em bases de dados anti-vírus.

Existem soluções comerciais mais completas, destinadas a governos de países, como o *FinFisher*[15]. Segundo Mikko Hypponen, chefe de pesquisas da *F-Secure*[16], esta solução pode ter sido oferecida e utilizada pelo governo do Egito. Ainda, de acordo com sua política, a empresa *F-Secure* não se eximirá de adicionar à sua base de dados a assinatura do *FinFisher* ou de qualquer outro programa destinado à invasão, comercial ou não, que chegue a seu conhecimento.(HYPPONEN, 2011)

Além de possuir seu código fonte fechado, o fato de um *software* confidencial ser comercial torna-o menos discreto. Bastou a descoberta que o governo do Egito poderia estar

[12] http://www.keylogger.org/

[13] http://www.spytech-web.com/spyagent.shtml

[14] Tecnologia presente a partir do sistema de arquivos *NTFS 3.0* e que provê criptografia em nível de sistema de arquivo, transparente ao usuário

[15] http://www.finfisher.com/FinFisher/en/index.php

[16] empresa desenvolvedora de produtos para segurança computacional, como *softwares* antivírus e anti-intrusão

utilizando a solução *FinFisher* para que ampla publicidade fosse dada à mesma, influenciando diretamente investigações de outros governos que porventura possam estar utilizando a mesma solução.

Dito isso, nota-se que ferramentas comerciais, tanto destinadas a usuários comuns quanto a governos, não suprem os requisitos recomendáveis para serem utilizadas em investigações policiais sigilosas.

2.1.3 Códigos-fonte encontrados na internet

A pesquisa para este trabalho iniciou na internet, em fóruns e sítios relacionados ao tema, a fim de determinar as soluções existentes, os autores mais reconhecidos no meio e as possibilidades de implementação.

Especificamente, a procura por códigos-fonte de *rootkits* que implementassem as mesmas funções pretendidas neste projeto se mostraram muito limitadas. Poucos desenvolvedores postam seus códigos na rede. E os sítios que se dispõem a postar tais códigos ficam constantemente indisponíveis, fruto de ataques de grupos de *crackers*[17], que não gostam de ver seu conhecimento disseminado.

Ainda assim, após estudos e testes em códigos-fonte encontrados na internet, foi possível selecionar dois, que tratavam especificamente de captura de teclas digitadas e de transmissão de dados pela internet, os quais foram adaptados ao código principal do *rootkits*, tornando-se o ponto de partida para a implementação do mesmo.

2.2 CONCEITOS UTILIZADOS

O programa a ser desenvolvido neste trabalho será um *software-only driver*[18] que operará em modo *kernel*[19]. A seguir, serão apresentadas estas definições e outros conceitos, tecnologias e ferramentas empregados.

[17] Especialistas em informática que fazem mau uso de seus conhecimentos, invadindo computadores, roubando dados e outras ações danosas.

[18] *Driver* utilizado para operar um dispositivo lógico. É considerado uma extensão do *kernel*, adicionando funcionalidades a este.

[19] Componente central do sistema operacional dos computadores; ele serve de ponte entre aplicativos e o processamento real de dados feito a nível de hardware.

2.2.1 Arquitetura dos sistemas operacionais Windows

Os sistemas operacionais Windows implementam 2 modos de execução de programas: modo usuário e modo *kernel*, cada modo com seu espaço próprio de memória virtual. Essas distinções de modo e memória são necessárias para impedir que aplicações de usuário acessem ou modifiquem dados de sistema, seja de forma inadvertida ou proposital, evitando instabilidade e inconsistência operacionais. Na família de processadores *x86* e *x64*, tais distinções são implementadas utilizando um conceito bastante intuitivo: *rings*, ou anéis em português, conforme apresenta a figura 2.1.

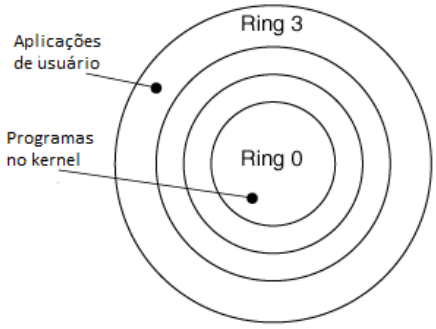

Figura 2.1: Conceito de anéis, adotado pelos processadores x86 e x64.

Apesar destes processadores suportarem até 4 anéis ou níveis de privilégio de execução, os sistemas Windows implementam apenas 2 níveis: o de menos privilégios, *ring* 3, atribuído a códigos e dados provenientes do modo usuário, e o mais privilegiado, *ring* 0, atribuído a códigos e dados provenientes do modo *kernel*. Assim, o processador, por exemplo, ao constatar uma tentativa de acesso a um endereço alocado no espaço de memória destinado ao modo *kernel*, verifica o nível a que a aplicação solicitante pertence. Se for uma aplicação *ring* 3, então esse acesso será negado. As seguintes diferenças são observadas(PERLA; OLDANI, 2010):

Aplicações que executam em modo usuário:

- São executadas com privilégios restritos;
- não podem acessar a memória do *kernel*;
- não podem comunicar-se diretamente com o *hardware*;
- não podem invocar instruções privilegiadas do processador.

Aplicações que executam em modo *kernel* têm acesso a todos os recursos do sistema, a saber:

- São executadas com todos os privilégios existentes;
- têm acesso a toda a área de memória, incluindo a memória do usuário;
- podem comunicar-se diretamente com o *hardware*, utilizando para isso uma *DLL* específica;
- podem invocar instruções privilegiadas do processador.

Na figura 2.2, é apresentada uma representação dos componentes presentes no modo usuário e no modo *kernel*.

Existem três camadas que compõem o modo *kernel*:

- Camada executiva: Implementa as funções que podem ser chamadas no modo usuário. Tais funções são denominadas ”serviços de sistema”, e são exportadas via a biblioteca *NTDLL.dll*. Outras funções abrigadas são as de *drivers* de dispositivos, ou seja, funções desenvolvidas por terceiros e que se destinam a controlar dispositivos de *hardware* ou de *software*. Esta camada também contém vários gerenciadores, a saber: de *I/O - Input/Output*, de objeto, de *plug and play*[20], de segurança, de memória, de processos, entre outros. Basicamente, essa camada define “o que se tem que fazer”.
- Camada *kernel*: Provê mecanismos essenciais utilizados pelos componentes da camada executiva, tais como sincronização e agendamento de *thread*[21], e também suporte para funcionalidades dependentes de *hardware*, tais como lançamentos de interrupções e exceções. Em resumo, a camada *kernel* especifica “como deve ser feito”.
- Camada *HAL*: Provê serviços para as camadas executiva e *kernel*. Sua função principal é a portabilidade entre os sistemas *Windows* e as diferentes plataformas de *hardware* utilizadas por meio de uma interface em baixo nível. Essa interface possibilita que detalhes específicos de cada plataforma de *hardware* (por exemplo, implementação da comunicação entre processadores, dos controles de interrupção, entre outros) não sejam levados em conta por quem está solicitando o serviço, tornando mais simples sua utilização (VISCAROLA, 2010).

[20] Tecnologia que permite que um computador reconheça e configure automaticamente qualquer dispositivo que seja instalado.

[21] Menor unidade de processamento que pode ser agendada por um sistema operacional.

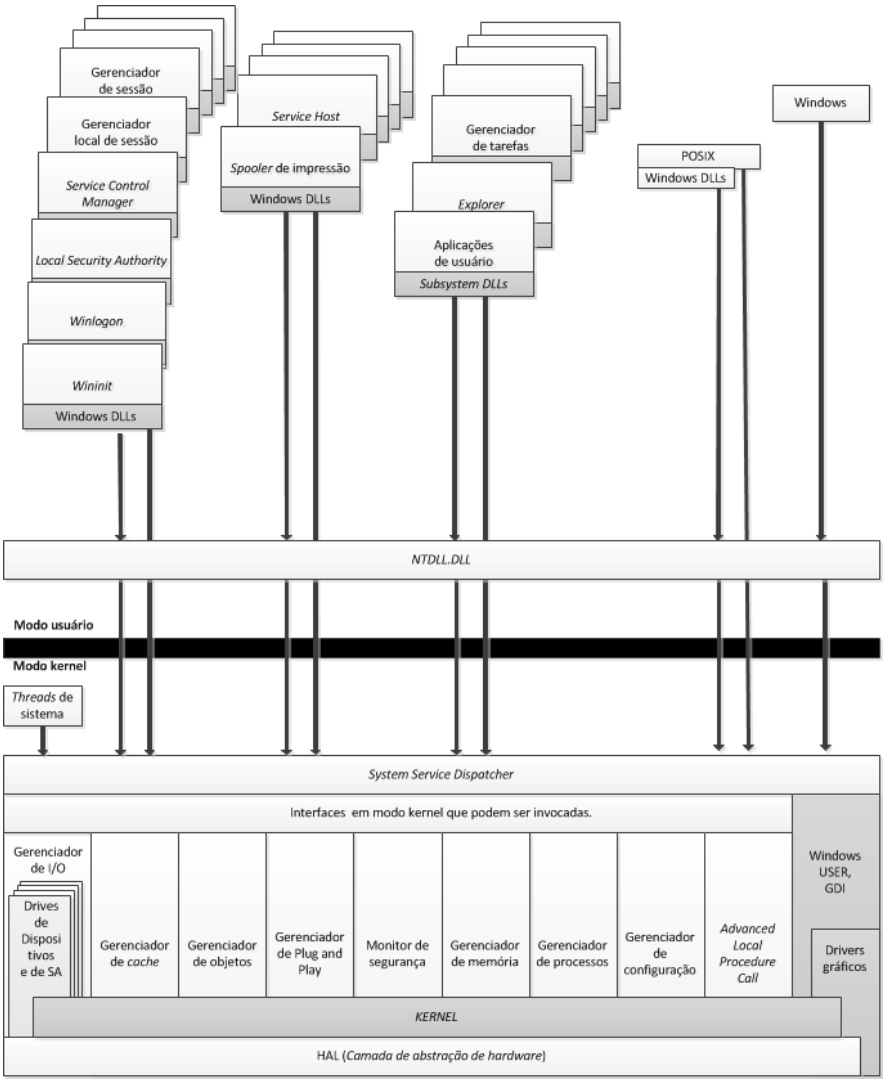

Figura 2.2: Arquitetura do sistema Windows. Imagem adaptada de (RUSSINOVICH; SOLOMON; IONESCU, 2009)

.

Sobre os processos, os serviços e as aplicações que executam em modo usuário, a Figura 2.2 mostra que eles nunca chamam diretamente as funções oferecidas pela camada executiva do Windows (por exemplo, a função *CreateFile*, utilizada para criar um arquivo). Ao contrário disso, eles chamam uma ou mais *DLLs*, que regulamentam e traduzem as funções descritas na *API*[22] - *Application Programming Interface* - do *Windows* em chamadas de sistema para o modo *kernel*, via as versões "Nt" correspondentes (função *NtCreateFile*, no caso).

É a biblioteca *NTDLL.DLL* que contém as versões "Nt"das funções especificadas na *API* do *Windows*. E dentro de cada uma dessas funções existe uma instrução especial que causa a transição para o modo *kernel*, invocando o despachante de sistema. Este último, após verificar alguns parâmetros, chama a função correspondente no modo *kernel*, que efetivamente cria o arquivo.

Por causa de todo este controle existente na transição entre modos, e também das limitações a que aplicações de usuário estão sujeitas, foi definido que a *FID* operará em modo *kernel*. Assim, ele possuirá privilégios ilimitados de sistema, podendo, por exemplo, executar as seguintes tarefas sem ser bloqueada por outros aplicativos ou políticas de segurança:

- Criar arquivos em pastas de sistema;
- Enviar pacotes pela rede livremente;
- Capturar entradas digitadas no teclado;
- Driblar políticas de segurança.

Em contrapartida, existem alguns dificultadores para essa abordagem em modo *kernel*:

- Confiabilidade: todo código que executa em no modo *kernel* é tido como confiável, ou seja, não são feitas verificações de argumentos passados e instruções invocadas. Em outras palavras, um código em modo *kernel* mal desenvolvido ou pouco testado

[22] Conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços.

certamente ocasionará inconsistências ou travamentos de sistema, mais conhecidos como *BSOD*[23] - *Blue Screen of Death*. Em modo usuário, o máximo que poderia ocorrer seria um erro e o posterior fechamento do aplicativo com erro;

- Segurança: A *Microsoft*, ao observar que os ataques a seu sistema operacional estavam migrando do modo usuário para o modo *kernel*, desenvolveu as seguintes contramedidas: (BLUNDEN, 2009)

  a) *Kernel* Patch Protection, ou *KPP*. Ao operar em modo *kernel*, um *driver* podia modificar como bem entendesse qualquer componente do *Windows*, inclusive os essenciais para seu funcionamento (por exemplo, tabelas de sistema(*GDT*[24], *IDT*[25], *SSDT*[26]), pilhas na área de memória reservada ao modo *kernel*, definições de objetos, entre outros). É certo que uma modificação mal feita acarretaria o travamento do sistema. Porém, pelo fato das técnicas que exploram este componentes já estarem bastante disseminadas na internet e em livros, o número de possíveis atacantes com chances reais de sucesso cresceu bastante. Assim, a *Microsoft* criou este sistema, que, ao detectar quaisquer mudanças nesses componentes, derruba o sistema com o código `0x109 -CRITICAL_STRUCTURE_CORRUPTION`.

  b) *kernel* Mode Code Signing, ou *KMCS*. Trata-se de uma política de sistema que exige que todo *driver* a ser carregado no sistema esteja assinado por uma chave criptográfica, denominada código *Authenticode*[27], assinada por autoridades certificadores às quais a *Microsoft* se credenciou. Caso o código seja inválido, uma mensagem é exibida na tela, abortando a instalação do *driver*. Porém, por questões de compatibilidade retrógrada, esta política é totalmente aplicada apenas em sistemas de 64 *bits*. Sistemas de 32 *bits* possibilitam a instalação de *drivers* não assinados, gerando apenas um alerta no *log* de eventos do Windows. Isso porque tais sistemas estão presentes em vários computadores que possuem dispositivos antigos, ou que não possuem mais suporte, ou até mesmo cuja empresa desenvolvedora não existe mais, inviabilizando a assinatura dos *drivers* para estes dispositivos. Nesta situação, a aplicação completa de tal política certamente faria com que vários sistemas parassem de funcionar. Já em relação aos sistemas de 64 *bits*, o fato da sua distribuição no mercado de consumo ser bem mais recente possibilita à *Microsoft* certificar as empresas autorizadas a desen-

[23]Tela apresentada nos sistemas operacionais Windows em caso de erro grave de sistema.

[24]Tabela utilizada pelo *kernel* que contém informações sobre áreas de memória, como endereço base, tamanho e privilégio de acesso.

[25]Tabela utilizada pelo *kernel* que contém o mapeamento entre as diferentes interrupções e as rotinas para tratar cada uma delas.

[26]Tabela utilizada pelo *kernel* que contém o mapeamento entre as diferentes chamadas de sistema e as rotinas para tratar cada uma delas.

[27]Código empregado pela *Microsoft* para assinar digitalmente arquivos executáveis, a fim de conferir sua autoria e garantir que seu código fonte não foi modificado ou corrompido.(MICROSOFT, 2011)

volver dispositivos e *drivers* para tais sistemas, autorizando a atribuição de códigos *Authenticode* para os mesmos.

Tendo em vista as dificuldades elencadas, foi definido que a *FID* se destinará a sistemas Windows Vista ou Windows 7 de 32 *bits*, de modo a evitar o bloqueio pelo *KMCS*. Ainda, suas funcionalidades serão implementadas por meio de um *driver* instalado na máquina alvo, não sendo necessário alterar os componentes de sistema protegidos pelo *KPP*.

2.2.2 Estrutura do driver

Um *driver*, quanto a sua natureza, pode ser de 3 tipos:

1. *Driver* de barramento, ou *Bus Driver*: gerenciam barramentos lógicos e físicos, tais como *USB*, *PCI*[28], entre outros, informando ao gerenciador de *plug and play* quando um novo dispositivo é conectado ao seu barramento;
2. *Driver* de função, ou *Function Driver*: controlam um dispositivo. São eles que exportam a interface operacional do dispositivo, informando todas as funções do mesmo, para o sistema operacional;
3. *Driver* de filtro, ou *Filter Driver*: são *drivers* lógicos que se encontram logo acima ou abaixo de um *driver* de função, podendo adicionar, monitorar ou alterar *IRPs*[29] destinados ou enviados ao dispositivo controlado pelo *driver* de função.

Neste trabalho será desenvolvido um *Filter Driver*. Afinal, ele não será o responsável pelo controle de nenhum dispositivo, mas sim interceptará dados retornados pelo *driver* de teclado, e utilizará as funções especificadas pelo *driver* da placa de rede. Em outras palavras, ele interceptará pacotes *IRP* provenientes do *driver* de teclado, e enviará pacotes *IRP* para o *driver* da placa de rede.

É obrigatório que todos os *drivers* para sistemas Windows contenham as seguintes rotinas:

[28] Barramento utilizado para conectar dispositivos periféricos a computadores.

[29] Estruturas do modo *kernel* utilizadas para possibilitar a comunicação entre *drivers* - *I/O Request Packets* do sistema *Windows*

```
DRIVER_INITIALIZE DriverEntry;
...

NTSTATUS DriverEntry(
   __in struct _DRIVER_OBJECT  *DriverObject,
   __in PUNICODE_STRING  RegistryPath
   )
 {
     ...
 }
```

Código 2.1: Rotina DriverEntry

```
pDriverObject->MajorFunction[IRP_MJ_CREATE] = DispatchCreate;
pDriverObject->MajorFunction[IRP_MJ_CLOSE] = DispatchClose;
pDriverObject->MajorFunction[IRP_MJ_READ] = DispatchRead;
pDriverObject->MajorFunction[IRP_MJ_WRITE] = DispatchWrite;
pDriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = DispatchControl
   ;
```

Código 2.2: Function Codes

- *DriverEntry*: É a rotina de inicialização do *driver*, implementada pelo programador. É chamada apenas quando o *driver* é carregado, e recebe dois parâmetros: um ponteiro para o respectivo *Driver Object*, já criado pelo gerenciador de *I/O*, e uma string *UNICODE* contendo o caminho para a chave do *driver* no registro do *Windows*. Esta chave encontra-se dentro da *HKLM*[30] - *Hive Key Local Machine*, conforme a rotina 2.1.

  O objetivo da rotina *DriverEntry*, além de inicializar todas as estruturas que serão utilizadas pelo *driver*, como *spinlocks*[31], *queues*[32], entre outras, consiste também em exportar os *Dispatch Entry Points* definidos pelo *driver*;

- *Dispatch Entry Points*: São ponteiros para rotinas implementadas pelo *driver* para cada requisição de *I/O* suportada por este. Para cada uma dessas requisições é definido um código de função, ou *Function Code*[33], que indicará qual rotina será chamada. Assim, o gerenciador de *I/O*, ao receber um *IRP* destinado a um *driver* específico, verifica se o driver está carregado e iniciado, identifica o código de operação especificado no *IRP* e chama diretamente um das rotinas implementadas por aquele *driver*. No código 2.2, um exemplo contendo um mapeamento entre os códigos de função e os *Dispatch Entry Points*:

[30] Handle para uma estrutura de dados do registro do *Windows* que é carregada para a memória durante o *boot* e que contém informações específicas do computador.(MICROSOFT, 2011)

[31] Mecanismo utilizado pelo *kernel* para implementar a exclusão mútua.

[32] Lista encadeada implementada pelo *kernel*, que funciona como filas de objetos.

[33] Código contido no IRP que informa ao driver qual operação de I/O deve ser executada. (RUSSINOVICH; SOLOMON; IONESCU, 2009)

No caso, `IRP_MJ_CREATE` é um exemplo de código de função e `DispatchClose` é um exemplo de rotina implementada pelo driver.

- *DriverUnload*: Rotina que descarrega o *driver*, invocada quando todos os *Device Object* associados ao *driver* foram deletados e não resta nenhuma referência ao mesmo. Em resumo, esta rotina desfaz todas as ações executadas pela *DriverEntry*.

Além dessas rotinas básicas que todos os *drivers* obrigatoriamente devem implementar, os *drivers* de *hardware* devem conter, ainda, rotinas como *AddDevice*[34], *InterruptServiceRoutine*[35], e outros *Dispatch Entry Points*, como *DispatchPnP driver*[36] e *DispatchPower driver*[37].

2.2.3 Objetos do sistema de *I/O*

Existem três objetos essenciais para o funcionamento do sistema de *I/O*(VISCAROLA, 2010): *Driver Object, Device Object e FileObject*. Alguns campos foram omitidos nas representações destas estruturas (Tabelas 2.1, 2.2 e 2.3) ou por serem "parcialmente opacos" - campos que não devem ser acessados ou modificados, ou por serem indiferentes ao *driver* de *software* que será implementado neste projeto:

- *Driver Object*: Criado pelo gerenciador de *I/O* quando o *driver* é carregado. O ponteiro para esta estrutura é passado como primeiro argumento na chamada da rotina *DriverEntry* citada na seção 2.2.2, que preencherá os campos da estrutura *Driver Object*, reproduzida na tabela abaixo, com informações como o tamanho do *driver* `DriverSize`, qual o dispositivo controlado por ele `DeviceObject`, além de ponteiros para as rotinas *Dispatch Entry Points* e *DriverUnload*, além de outras informações.

- *Device Object*: Estrutura de dados que representa um dispositivo lógico (por exemplo, um volume ou um sistema de arquivos) ou físico (por exemplo, um teclado ou um disco rígido) e que descreve as suas características. Pode ser criado por meio da rotina *AddDevice*, no caso de *drivers* compatíveis com *plug and play*, ou por meio da rotina

[34]Rotina que cria objetos para dispositivos enumerados pelo gerenciador de Plug and Play. (RUSSINOVICH; SOLOMON; IONESCU, 2009)

[35]Rotinas que tratam interrupções.

[36]Rotina que trata IRPs contendo código de função `IRP_MJ_PNP`. (RUSSINOVICH; SOLOMON; IONESCU, 2009)

[37]Rotina que trata IRPs contendo código de função`IRP_MJ_POWER`. (RUSSINOVICH; SOLOMON; IONESCU, 2009)

Tabela 2.1: Estrutura Driver Object

CSHORT Size	CSHORT Type
PDEVICE_OBJECT DeviceObject	
ULONG Flags	
PVOID DriverStart	
ULONG DriverSize	
...	
PDRIVER_EXTENSION DriverExtension	
UNICODE_STRING DriverName	
...	
PDRIVER_UNLOAD DriverUnload	
PDRIVER_DISPATCH MajorFunction[IRP_ MJ_MAXIMUM_FUNCTION + 1]	

IoCreateDevice, que pode ser chamada durante a rotina *DriverEntry* caso um *driver* de *software* necessite criar um dispositivo lógico. Apresenta os campos citados na Tabela 2.2:

Tabela 2.2: Estrutura Device Object

CSHORT Size	CSHORT Type
LONG Reference Count	
PDRIVER_OBJECT DriverObject	
...	
ULONG Flags	
ULONG Characteristics	
...	
PVOID DeviceExtension	
DEVICE_TYPE DeviceType	
	CCHAR StackSize
LIST_ENTRY ListEntry	
...	
PDEVOBJ_EXTENSION DeviceObjectExtension	
PVOID Reserved	

Todos os *Device Object* podem possuir um nome, a ser definido pelo *driver* ou gerado automaticamente pelo gerenciador de *I/O*. Porém, este nome estará disponível apenas no diretório *\DosDevices*[38], ao qual a *API* do *Windows* não tem acesso. Para tornar um dispositivo acessível no modo usuário , é necessário criar um link simbólico, dentro do diretório *\Global??*[39], que remeta ao dispositivo presente no diretório *\DosDevices*. Como ilustração, apresenta-se a Figura 2.3:

Um dos campos mais importantes na estrutura *Device Object* é o *DeviceExtension*.

[38] Diretório que armazena os nomes MS-DOS dos dispositivos.

[39] Diretório que contém o mapeamento entre nomes de dispositivos e seus nome simbólicos.

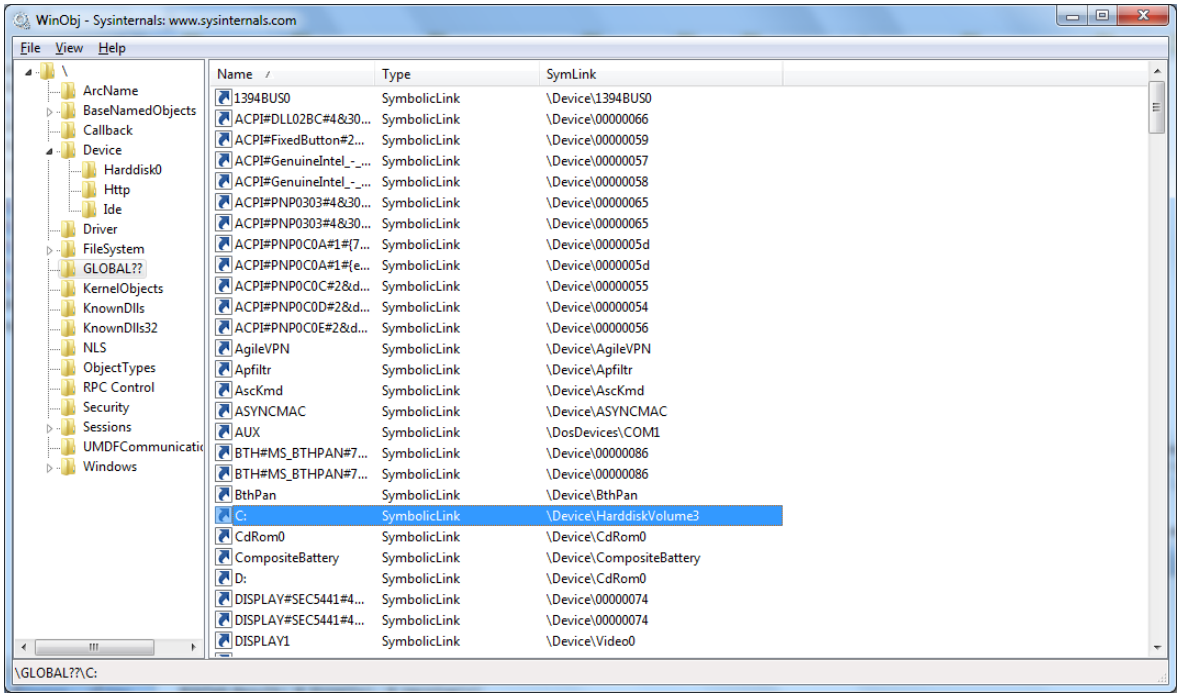

Figura 2.3: Link simbólico c: destacado.

Nele, podem ser armazenados quaisquer dados definidos pelo programador. Normalmente, é utilizado para armazenar estatísticas, estado do dispositivo e das estruturas que ele utiliza (*spinlocks*, por exemplo), além de qualquer dado que seja necessário estar residente em memória para completar a operação de *I/O* (é no campo *DeviceExtension* que o código de uma tecla digitada é armazenado, a fim de ser repassado para o próximo dispositivo). Outros campos importantes são os ponteiros para o *Driver Object* e para o pacote *IRP* sendo processado (CurrentIrp).

- *File Object*: É uma estrutura de dados que representa uma instância ativa de um *Device Object*. Ele é criado para executar qualquer operação de *I/O* solicitada no *Windows*. Por exemplo, quando o usuário abre um arquivo ou um dispositivo, o gerenciador de *I/O* retorna um *File Object*. É necessário frisar que o *File Object* contém apenas informações relativas à instância aberta, e não dados contidos no arquivo em si. Ainda, se o mesmo arquivo for aberto duas vezes, serão criadas duas instâncias *File Object*. Isso é necessário, por exemplo, para armazenar a posição atual do cursor em cada arquivo. Algumas informações presentes em um *File Object* são apresentadas na Tabela 2.3:

  Todas essas informações são relacionadas à instância. Por exemplo, `CurrentByteOffset` indica a posição do cursor no arquivo, e `SharedDelete, SharedRead e SharedWrite` indicam se um terceiro usuário podem executar tais ações simultaneamente ao usuário atual.

Tabela 2.3: Estrutura File Object

CSHORT Size	CSHORT Type
LONG Reference Count	
PDEVICE_OBJECT DeviceObject	
PVPB Vpb	
...	
BOOLEAN SharedDelete, SharedRead, SharedWrite	
...	
UNICODE_STRING FileName	
LARGE_INTEGER CurrentByteOffset	
ULONG Flags	
...	
PIO_COMPLETION_CONTEXT CompletionContext	

2.2.4 Gerenciador de *I/O*

Pela natureza do projeto, que basicamente consiste em obter dados e enviá-los pela rede, a maior parte da implementação relaciona-se estreitamente com sistema de *I/O* do *Windows*. O gerenciador de *I/O*, presente na camada executiva no *kernel*, é o coração deste sistema. Ele é o responsável por conectar aplicações e componentes do sistema a dispositivos virtuais, lógicos e físicos, além de prover toda a infra-estrutura de suporte aos *drivers* destes dispositivos.

Outros gerenciadores, como de *plug and play* e de energia, que também fazem parte do sistema de *I/O* do Windows, não serão abordados neste trabalho, pois são utilizados diretamente apenas por *drivers* de dispositivos físicos, ou, em outras palavras, *drivers* que controlam *hardware*.

O driver de filtro que será desenvolvido neste projeto é um *driver* de *software*. Ele pode ser considerado uma extensão do *kernel*, adicionando funcionalidades ao último. Ainda, sua implementação é mais simples que um *driver* de *hardware*, pois como não controla nenhum dispositivo físico, não se envolvendo assim em questões relacionadas, por exemplo, ao gerenciador de *plug and play* e à camada de abstração de *hardware HAL*.

Não obstante, um *driver* de *software* pode procurar e utilizar serviços de plataformas de *hardware* instaladas, bastando para isso enviar uma requisição para o *driver* de *hardware* responsável. Essa comunicação dentro do sistema de *I/O* é feita via pacotes denominados I/O *Request Packets*, ou *IRPs*.

2.2.5 IRP

*IRP*s são pacotes criados pelo gerenciador de *I/O* assim que este recebe uma requisição. Cada pacote *IRP*, por descrever completamente uma única operação de *I/O*, contém toda as informações para que decisões sobre esta requisição sejam tomadas. Tanto o gerenciador de *I/O* quanto os dispositivos para os quais o pacote *IRP* é enviado necessitam destas informações, tais como:

- Operação a ser executada: criação, deleção, leitura, escrita, controle, entre outras definidas pelo sistema ou pelo programador;
- *File Object* associado: é o objeto para o qual foi solicitado a operação de *I/O*;
- Modo do requisitante: se a requisição partiu do modo *kernel* ou usuário.

Essa necessidade se explica pelo fato de cada *driver*, ao receber um pacote *IRP* enviado ao dispositivo controlado por ele, ter o poder de decidir o que fazer com o pacote, se repassa para outro dispositivo, ou se completa a requisição com sucesso ou erro.

Na Figura 2.4, é apresentado um exemplo simplificado de uma requisição de *I/O* para um dispositivo:

No passo 4 da Figura 2.4, é gerada uma interrupção de *hardware* assim que o mesmo finaliza a requisição. Basicamente, uma rotina específica para esta interrupção é chamada (*ISR*[40], ou *Interrupt Service Routine*), a qual captura a informação retornada pelo dispositivo e finaliza a interrupção, liberando o processador para outras tarefas. A informação retornada é disponibilizada em uma fila, da qual é extraída por uma outra rotina ((*DPC*[41], ou *Deferred Procedure Call*), cuja função é interpretar a informação retornada e avisar o gerenciador de *I/O* se a requisição de *I/O* pode ser finalizada com sucesso ou não. No caso de sucesso, o pacote *IRP* utilizado é descartado.

[40] Rotina para o qual o *kernel* transfere a execução quando um driver gera uma interrupção.

[41] Técnica que permite o reagendamento de tarefas menos prioritárias que estão em execução, para dar lugar a tarefas de maior prioridade.

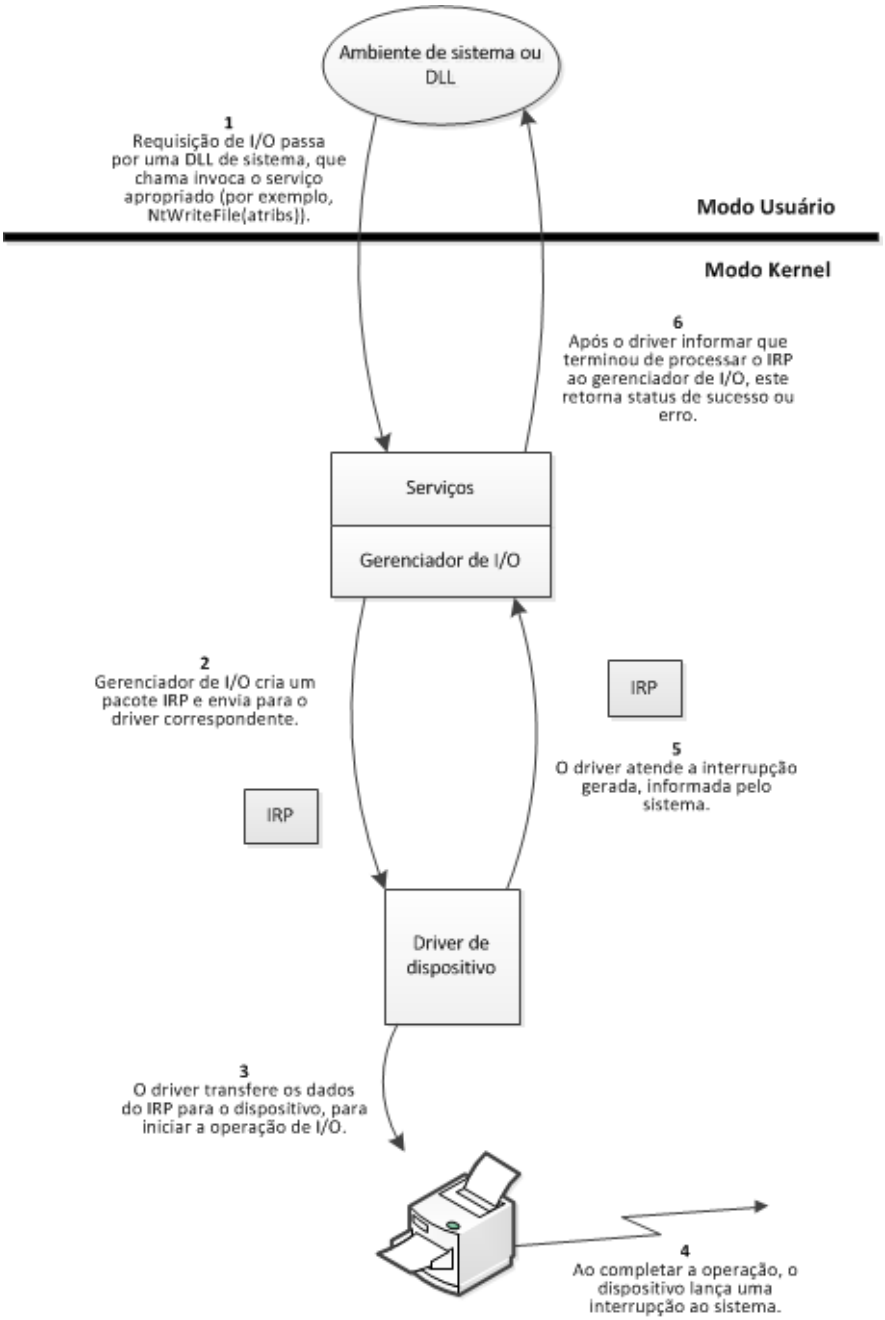

Figura 2.4: Requisição e finalização de uma requisição de I/O. Imagem adaptada de (RUSSINOVICH; SOLOMON; IONESCU, 2009)

.

2.2.6 Sockets

Soquetes, ou *sockets*, são as extremidades de um canal de comunicação bi-direcional estabelecido entre dois processos, amplamente utilizados para envio e recebimento de informações dentro de uma rede de computadores.

Segundo (TANENBAUM, 2002), as primitivas de soquetes para uma conexão *TCP* são:

- `SOCKET`: Criação de um novo ponto de comunicação. Nesta etapa, além de serem alocados recursos de sistema, são especificados o formato de endereçamento a ser utilizado (`IPv4`[42] ou `IPv6`[43]), o tipo de serviço desejado - não orientado a conexão (`Datagram sockets`) ou orientado a conexão (`Stream sockets`), e o protocolo;
- `BIND`: Atribuição de um endereço de rede ao soquete, constando endereço IP e número de porta;
- `LISTEN`: Anuncia a disposição para aceitar conexões e mostra o tamanho da fila, caso vários clientes tentem se conectar ao mesmo tempo;
- `CONNECT`: Tenta estabelecer uma conexão ativamente;
- `ACCEPT`: Aceita a solicitação de conexão recebida, e aloca recursos para a mesma;
- `SEND`: Envia dados através da conexão;
- `RECEIVE`: Recebe dados da conexão;
- `CLOSE`: Encerra a conexão. É necessário que ambos os lados executem esta primitiva.

2.2.7 Criptografia

A confidencialidade do fluxo de tráfego é um dos principais serviços que devem ser oferecidos pela *FID*, pois serão interceptadas informações sigilosas e pessoais, às quais só é permitido acesso por meio de uma autorização judicial.

[42] Quarta versão do protocolo IP, que utiliza endereços de 32 bits.
[43] Versão sucessora do IPv4, que utiliza endereços de 64 bits.

Para fornecer esse serviço de confidencialidade, será utilizado tanto o mecanismo de criptografia simétrica quanto o de criptografia assimétrica.

Segundo (STALLINGS, 2005), a criptografia simétrica é uma forma de criptosSistema em que a criptografia, ou a conversão do texto claro (que se deseja cifrar) para texto cifrado, e a decifração, ou a conversão do texto cifrado para o texto claro original, são realizadas utilizando a mesma chave.

Já a criptografia assimétrica, ou criptografia de chave pública, é uma forma de criptossistema em que a criptografia e a decifração são realizadas utilizando chaves diferentes - uma pública e outra privada. Assim, se a chave pública for utilizada para converter um texto claro em texto cifrado, apenas a chave privada correspondente conseguirá converter o texto cifrado no texto claro original. E a segurança desse criptossistema reside no fato que é computacionalmente inviável determinar a chave de decifração dado apenas o conhecimento do algoritmo utilizado e da chave pública.

O algoritmo simétrico *DES triplo* (NIST, 1999) será utilizado para cifrar o conteúdo interceptado a ser enviado para o servidor. Já o algoritmo assimétrico *RSA* (RSALABORATORIES, 2011) será utilizado para cifrar a chave do algoritmo simétrico, para que também possa ser enviada ao servidor de maneira confidencial.

- Cifra simétrica *DES triplo*: técnica de criptografia múltipla que utiliza o algoritmo *DES*. O algoritmo *DES* é uma cifra de bloco, ou seja, ele divide o tamanho total do texto a ser cifrado em blocos menores, de 64 bits, que gerarão, cada um, um criptograma de igual tamanho, todos cifrados de forma independente, porém utilizando a mesma chave. Assim, para que funcione, o algoritmo espera uma entrada de texto claro de tamanho múltiplo de 64 bits. Pode-se preencher o texto em claro com zeros, por exemplo, até que este tamanho seja atingido. A chave esperada também é de 64 bits, porém apenas 56 bits são efetivamente usados. Os 8 bits restantes podem ser utilizados para outros fins, como por exemplo paridade (ou seja, verificação da integralidade da chave).

  No algoritmo *DES triplo*, o algoritmo *DES* é aplicado 3 vezes consecutivas, conforme a figura 2.5:

  1. Passo 1: A chave K1 é utilizada para cifrar o texto em claro P em texto cifrado C1;

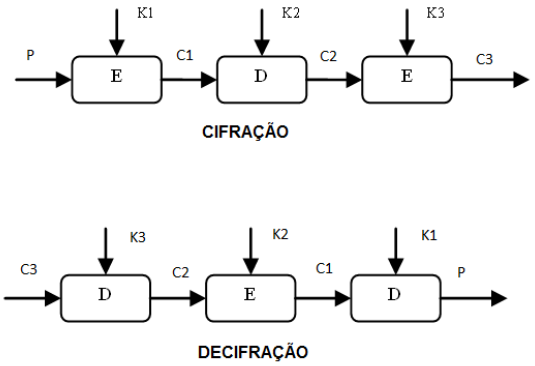

Figura 2.5: Criptografia e decifração *DES triplo*

2. Passo 2: A chave K2 é utilizada para decifrar o texto cifrado C1 no texto cifrado C2. Ainda segundo (STALLINGS, 2005), não existem razões de ordem criptográfica para aplicar a decifração ao invés da cifração no passo 2, sendo a única vantagem permitir que usuários do *DES triplo* decifrem dados criptografados pelos usuários do *DES* simples, caso K1 seja igual a K2;
3. Passo 3: A chave K3 é utilizada para cifrar o texto C2 no texto cifrado C3, que é o criptograma final do algoritmo *DES triplo*.

Para obter novamente o texto claro P, basta aplicar os mesmos passos da criptografia, porém de maneira inversa, também conforme ilustrado na figura 2.5.

Existem 5 modos de operação para uso com cifras de bloco. O modo CBC[44]-*Cipher Block Chaininig*, que será utilizado na solução, foi desenvolvido para contornar as deficiências do modo *ECB-Electronic Codebook*, em que todos os blocos são cifrados de maneira independente, porém utilizando a mesma chave. Assim, em um texto de grande tamanho cifrado utilizando o modo *ECB*, podem surgir criptogramas idênticos, o que pode facilitar o trabalho de um criptoanalista em tentar descobrir o texto claro.

Para isso, no modo CBC, em cada etapa é realizada uma operação *XOR*[45] entre o bloco de 64 bits de texto claro P^x e o bloco de texto cifrado resultante da etapa anterior C^{x-1}. Por não existir etapa anterior à primeira, nesta é utilizado um vetor de inicialização *IV*, conforme a figura 2.6. Ainda, nota-se que, tal como no modo ECB, a mesma chave K é utilizada em todas as etapas.

Dessa maneira, dois blocos de texto claro idênticos não gerarão criptogramas idênticos.

- Cifra assimétrica *RSA*: Os passos para se gerar uma par de chaves *RSA* são:

[44]Modo de operação utilizado em cifras de bloco em que a cifragem de um bloco depende do resultado da cifragem do bloco anterior.

[45]Operação lógica entre dois operandos que só resulta em um valor verdadeiro se um de seus operandos também possuir valor verdadeiro.

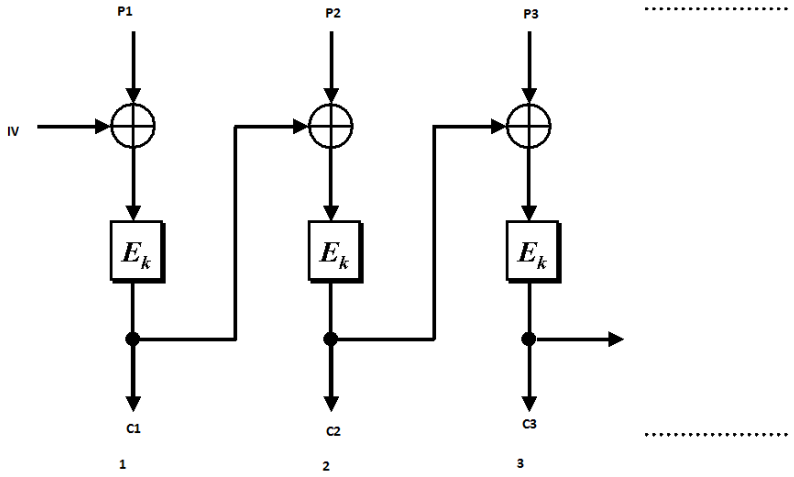

Figura 2.6: Modo de operação CBC

1. Escolher dois números primos p, q , tal que $p \neq q$;
2. Calcular $n = p * q$. n será o módulo;
3. Calcular $\phi(n) = (p-1)(q-1)$;
4. Selecionar e tal que $mdc(e, \phi(n)) = 1; 1 < e < \phi(n)$;
5. Calcular d, tal que $d \equiv e^{-1}(mod(\phi(n)))$;

Assim, a chave pública será o par e, n e a chave privada será o par d, n.

Para converter o texto claro P no texto cifrado C, primeiro há de se obter a representação em número inteiro de P. Supondo que seja M, conforme a equação 2.1:

$$C = M^e mod n; M < n \tag{2.1}$$

Já para converter o texto cifrado C no original M:

$$M = C^d mod n \tag{2.2}$$

A força do algoritmo *RSA* vem da dificuldade em fatorar números grandes, no caso descobrir os primos p e q a partir de n, que é um número grande, da ordem de, tipicamente, 1024 *bits* , ou 309 dígitos decimais.

2.3 PROGRAMAS UTILIZADOS

A ferramenta primordial para desenvolvimento de *drivers* para sistemas *Windows* chama-se *WDK*[46], ou Windows Driver Kit. Ela pode ser baixada gratuitamente do sítio da *Microsoft*[47] e contém documentação, exemplos, *build environments*[48] e outras ferramentas úteis aos desenvolvedores.

A versão do *WDK* utilizada no desenvolvimento do *driver* é 7600.16385.0, e o *build environment*, *x86 Checked Build for Windows 7*[49]. Existem duas possibilidades para cada ambiente: *Checked Build*[50] e *Free Build*[51]. É altamente recomendável que, durante o desenvolvimento do código, os executáveis sejam gerados utilizando a primeira abordagem: nela, verificações são incluídas no código binário a fim de detectar erros e variáveis para *debug*[52] são setadas. Ao final do desenvolvimento, deve-se gerar o executável utilizando a *Free Build*, para se obter um código mais otimizado.

Outros programas utilizados são descritos a seguir:

- Ambiente de programação: Visual Studio 2008[53]. *Framework* desenvolvida pela Microsoft para desenvolvimento de programas, permite integração com WDK;
- Depuração, ou *debug*: *WinDBG*[54]. Possibilita depuração do *kernel* em tempo de execução. Para isso, faz-se necessário conexão com o sistema a ser depurado, seja por máquina virtual ou alguma porta de comunicação física;
- Identificação de erros: *OACR*[55]. Procura por erros durante a compilação;
- Identificação de erros: *PREfast*[56]. Realiza testes específicos para *drivers*;

[46] Principal conjunto de ferramentas destinado ao desenvolvimento de *drivers* para sistemas *Windows*, desenvolvido pela *Microsoft*.

[47] http://msdn.microsoft.com/en-us/windows/hardware/gg487463

[48] Ambiente alvo em que determinado *software* será executado. Assim, esse *software* precisa ser construído obedecendo as condições especificadas por este ambiente.

[49] Versão especial do *Windows* que implementa controles adicionais destinados à depuração do sistema. (RUSSINOVICH; SOLOMON; IONESCU, 2009)

[50] Versão especial do *Windows* que implementa controles adicionais destinados à depuração do sistema. (RUSSINOVICH; SOLOMON; IONESCU, 2009)

[51] Outra forma de identificar a versão comercial do *Windows*. (RUSSINOVICH; SOLOMON; IONESCU, 2009)

[52] Processo de encontrar e reduzir defeitos num aplicativo de software ou mesmo em hardware.

[53] http://msdn.microsoft.com/pt-br/vstudio/aa718325

[54] http://www.windbg.org/

[55] http://msdn.microsoft.com/en-us/library/ms933794.aspx

[56] http://msdn.microsoft.com/en-us/library/ms933794.aspx

- Gerenciador de máquina virtual: [57]. Nela, foi emulada a máquina alvo, um sistema *Windows* 7 de 32 *bits*. Testes em sistemas instalados em máquinas virtuais são recomendados pela possibilidade de restaurar o sistema em poucos minutos e com o mínimo de trabalho, caso o mesmo se corrompa. Também foi emulado o servidor, em um sistema *Windows XP* de 32 *bits*, cuja função é receber os dados enviados pela máquina alvo;

- Instalação do *driver*: *OSR Loader*[58]. Possibilita setar várias opções de instalação e inicialização do *driver*;

- Análise de pacotes: *Wireshark*[59]. Solução gratuita com inúmeros recursos para análise de tráfego de rede;

- Servidor *WEB*[60]: Apache[61]. Ferramenta de retaguarda instalada na máquina Windows XP. Possibilita a conexão com o*driver* remoto, recebendo os pacotes enviados por este;

- Tratamento dos dados recebidos: *PHP*[62]. Linguagem utilizada para desenvolver o *script*, que será executado no servidor Apache e cuja função é receber,interpretar e disponibilizar os dados recebidos.

Neste capítulo, foi apresentado o escopo do projeto a ser implementado. Primeiramente foram citadas soluções que apresentam algum grau de similaridade com a solução proposta neste trabalho e, ao analisar aspectos inerentes às mesmas em relação à necessidades elencadas na seção 1.1, foi explicado porque não seria aconselhável a sua adoção. Após, foram apresentados os principais conceitos necessários ao desenvolvimento da solução proposta e também foram listados as aplicações utilizadas e o objetivo de cada uma, a fim de embasar o entendimento do próximo capítulo, que tratará dos aspectos técnicos da implementação da solução.

[57] https://www.virtualbox.org/

[58] http://www.osronline.com/article.cfm?article=157

[59] http://www.wireshark.org/

[60] Aplicação Web é o termo utilizado para designar, de forma geral, sistemas de informática projetados para utilização através de um navegador, na internet ou em redes privadas (Intranet).

[61] http://httpd.apache.org/

[62] http://br.php.net/

3 IMPLEMENTAÇÃO

Neste capítulo, a implementação da Ferramenta para Interceptação de Dados *FID* será detalhada. Ela consiste de três módulos: um *driver*, um serviço e uma aplicação de usuário, a saber:

1. `filtec.sys` - é o *driver* propriamente dito. Ele realiza tarefas como captura de dados digitados, armazenamento e cifragem de informações e transmissão pela rede.
2. `serv1.exe` - é um executável cuja função é instalar e iniciar tanto o *driver* `filtec.sys` quanto o serviço `serv1`. Além disso, este executável recebe os dados de interação do usuário transmitidos pelo processo `sendproc.exe`.
3. `sendproc.exe` - é um processo de usuário, criado pelo serviço `serv1`, para obter informações relacionadas à interação do usuário.

Nas próximas seções, serão expostos os motivos pelos quais se decidiu pela arquitetura apresentada. Na Figura 3.1, uma ilustração das funções destes módulos e do relacionamento entre eles:

3.1 filtec.sys

O arquivo `filtec.sys` é o *driver* propriamente dito, e consiste no principal módulo da solução.Na seção 3.2.1 será visto que uma chamada à rotina `StartService`, invocada pelo aplicativo `serv1.exe`, transfere a execução para a rotina `Driver Entry`, que inicializa o *driver*. As tarefas executadas pela rotina `Driver Entry` são ilustradas na Figura 3.2: ela cria uma *thread* e dois dispositivos, um de filtro e outro de controle (setas contínuas), além de informar ao subsistema de *I/O* as rotinas para tratar cada pacote *IRP* que recebe (setas tracejadas) e iniciar outras estruturas (setas pontilhadas).

Na Figura 3.2, um diagrama contendo os principais parâmetros e processos executados pela rotina *DriverEntry*:

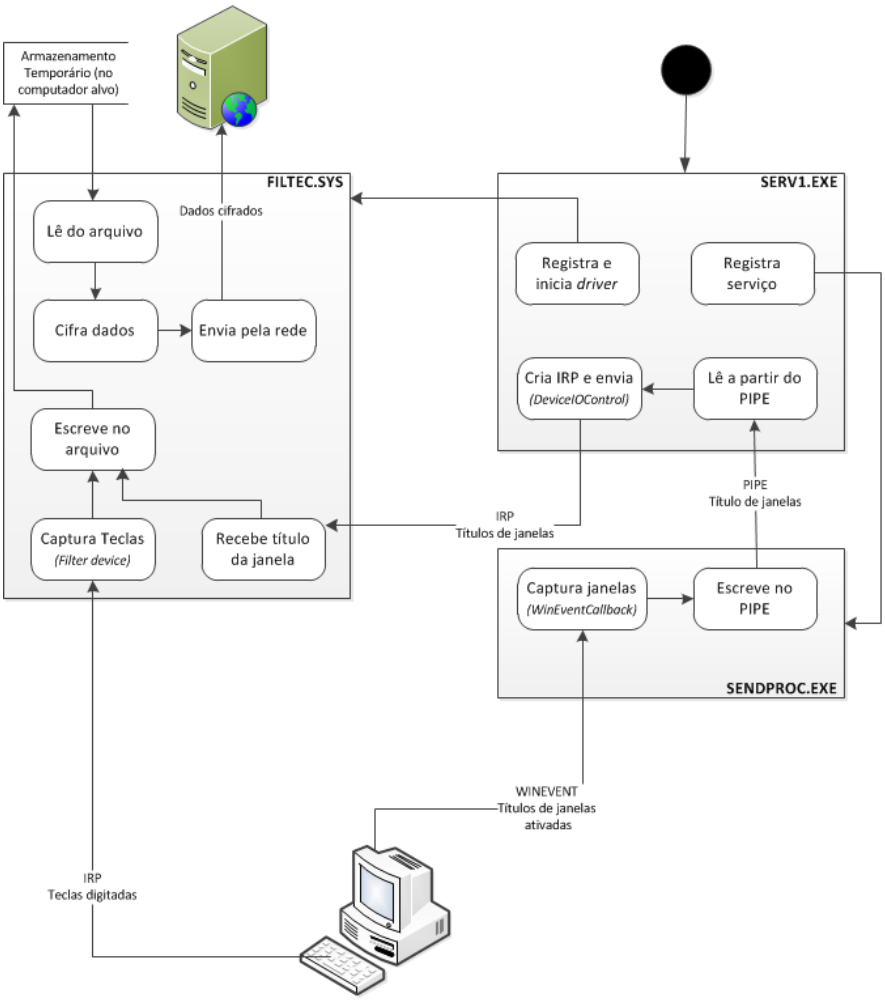

Figura 3.1: Relacionamento entre os diferentes módulos do rootkit.

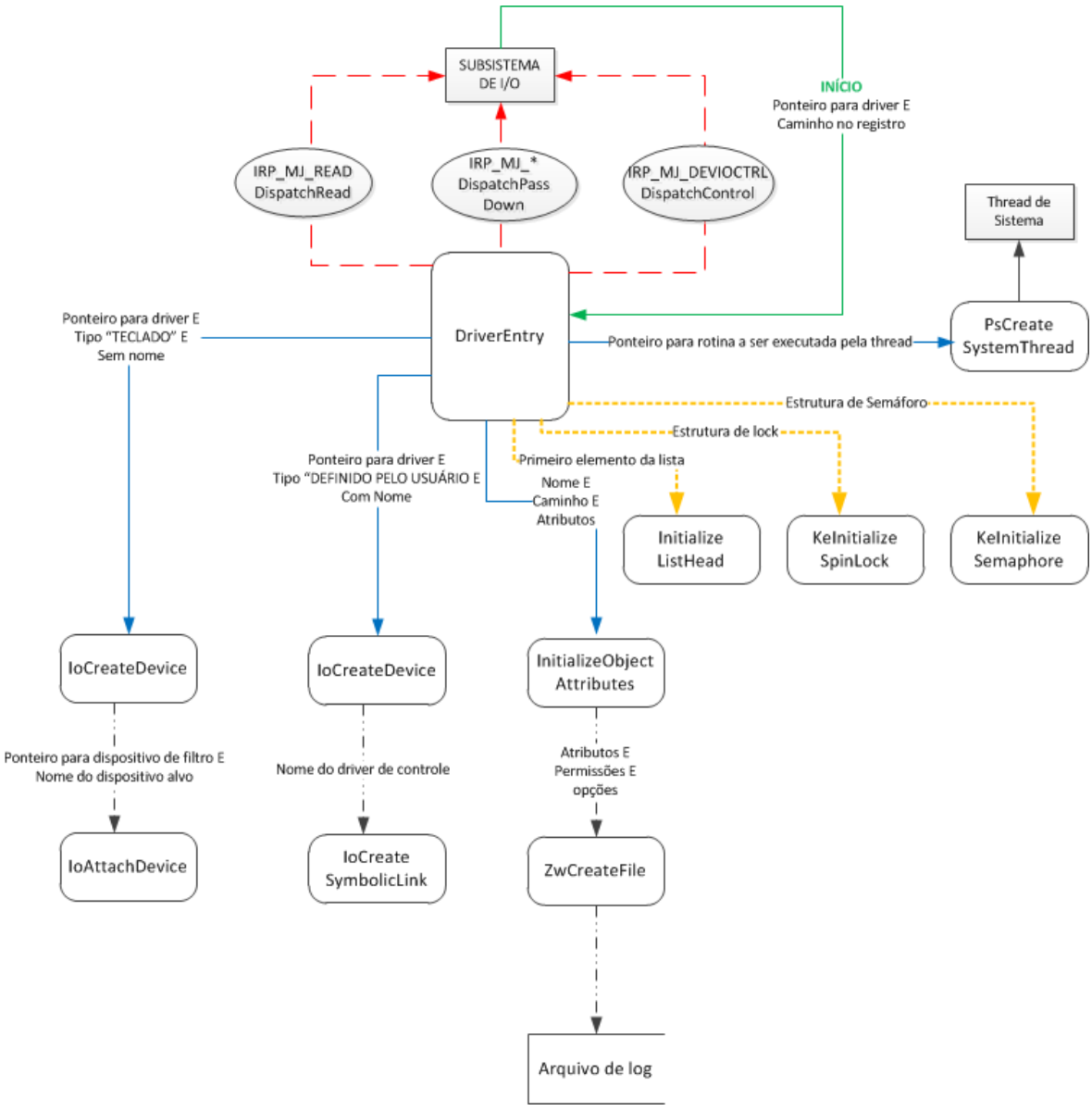

Figura 3.2: Tarefas executadas pela rotina DriverEntry.

```
NTSTATUS IoCreateDevice(
  __in      PDRIVER_OBJECT DriverObject,
  __in      ULONG DeviceExtensionSize,
  __in_opt  PUNICODE_STRING DeviceName,
  __in      DEVICE_TYPE DeviceType,
  __in      ULONG DeviceCharacteristics,
  __in      BOOLEAN Exclusive,
  __out     PDEVICE_OBJECT *DeviceObject
);
```

Código 3.1: Rotina IoCreateDevice

3.1.1 Captura de teclas pressionadas

A captura de teclas pressionadas pelo usuário é feita por meio de um dispositivo de filtro (*filter device* criado durante a inicialização do *driver* (chamada da rotina `DriverEntry`). O dispositivo é criado por meio de uma chamada à rotina `IoCreateDevice`, ilustrada no codigo 3.1:

Os principais parâmetros desta rotina são:

1. DriverObject: Entrada. Ponteiro para o *driver*. É o mesmo argumento recebido pela rotina *DriverEntry*;
2. DeviceName: Entrada, opcional. Ponteiro para nome do dispositivo a ser criado, em formato *UNICODE*[63]. É parâmetro obrigatório se for necessário instanciar o dispositivo a partir do modo usuário;
3. DeviceType: Entrada. Tipo do dispositivo a ser criado. No caso, será utilizado `FILE_DEVICE_KEYBOARD`;
4. DeviceObject: Saída. Ponteiro para a variável que vai receber o ponteiro para o dispositivo recém criado.

Um retorno com valor `STATUS_SUCCESS` indica que o dispositivo foi criado com êxito. Como é necessário que este dispositivo de filtro seja semelhante ao dispositivo de teclado, pois aquele interceptará todas as informações oriundas deste, além de possuir o tipo `FILE_DEVICE_KEYBOARD`, suas *flags* devem estar configuradas tais quais as de um dispositivo de teclado convencional. Finalmente, faz-se necessário inicializar a extensão de

[63] Sistema de codificação de caracteres que suporta o intercâmbio, processamento e exibição de textos em diversas linguagens existentes no mundo. São necessários 16 bits para representar um caractere Unicode.

```
NTSTATUS IoAttachDevice(
  __in   PDEVICE_OBJECT SourceDevice,
  __in   PUNICODE_STRING TargetDevice,
  __out  PDEVICE_OBJECT *AttachedDevice
);
```

Código 3.2: Rotina IoAttachDevice

dispositivo (*DeviceExtension*[64], que é uma estrutura de dados que armazenará, entre outras informações, o código de cada tecla pressionada pelo usuário.

Após o dispositivo ser criado, é necessário anexá-lo ao dispositivo do qual se deseja interceptar dados, ou seja, o dispositivo nativo de teclado. A rotina `IoAttachDevice` está ilustrada no código 3.2:

Seus parâmetros são:

1. SourceDevice: Entrada. Ponteiro para o dispositivo de filtro recém criado pela rotina `IoCreateDevice`.
2. TargetDevice: Entrada. Ponteiro para nome do dispositivo ao qual se deseja anexar. No caso, `\Device\KeyboardClass0"`;
3. *AttachedDevice: Saída. Ponteiro para a variável que vai receber o ponteiro para o dispositivo de filtro, caso a rotina retorne sucesso. Por meio deste ponteiro se buscarão os códigos das teclas digitadas.

A figura 3.3 apresenta uma abstração de uma árvore de dispositivos em que a *FID*, que corresponde ao dispositivo de filtro criado, foi anexada.

Como todo dispositivo de filtro é inserido no topo da árvore que contém o dispositivo ao qual se deseja anexar (módulo `/device/rootkit` da Figura 3.3), quaisquer pacotes *IRP* destinados ao último ou originários dele passarão obrigatoriamente pelo primeiro, que poderá decidir o que fazer com cada um deles. É neste estágio que a interceptação ocorre, basicamente da seguinte maneira:

1. Pacotes `IRP_MJ_READ` que receberão códigos de teclas são constantemente criados pelo gerenciador de *I/O* e enviados "em branco" para o dispositivo de teclado, onde

[64] Estrutura de dados que armazena informações específicas do dispositivo. (RUSSINOVICH; SOLOMON; IONESCU, 2009)

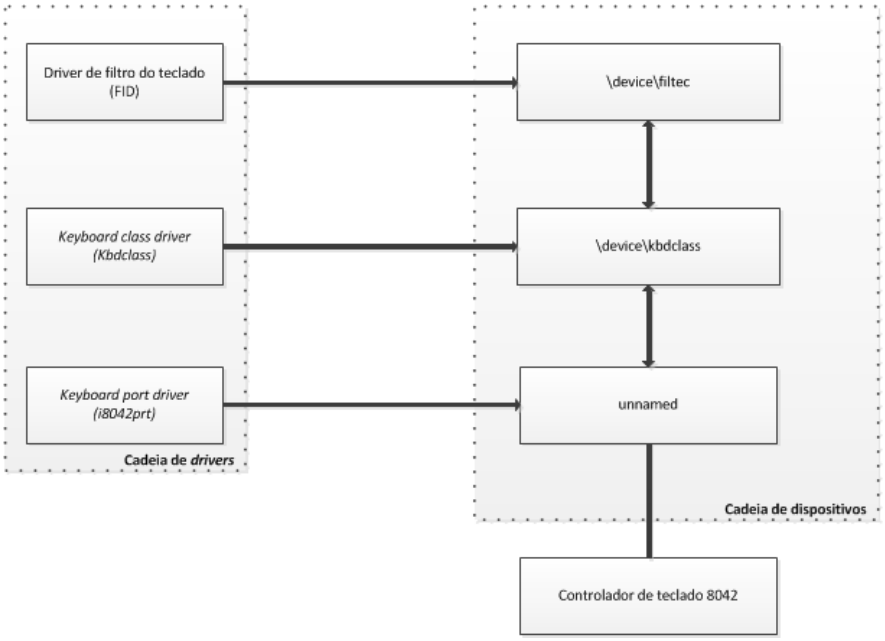

Figura 3.3: Árvore de dispositivos. Imagem retirada de (HOGLUND; BUTLER, 2006)

```
VOID IoSetCompletionRoutine(
  __in      PIRP Irp,
  __in_opt  PIO_COMPLETION_ROUTINE CompletionRoutine,
  __in_opt  PVOID Context,
  __in      BOOLEAN InvokeOnSuccess,
  __in      BOOLEAN InvokeOnError,
  __in      BOOLEAN InvokeOnCancel
);
```

Código 3.3: Rotina IoSetCompletionRoutine

aguardam uma tecla ser pressionada para então serem enviados de volta ao gerenciador de I/O, já contendo o código da tecla pressionada;

2. Como o dispositivo de filtro foi inserido na árvore que contém o dispositivo de teclado, os pacotes destinados a este último passarão pelo primeiro pelo dispositivo de filtro;

3. O *driver* do dispositivo de filtro está programado para repassar diretamente ao próximo dispositivo quaisquer pacotes que chegarem a ele (figura 3.4), exceto os dos tipos `IRP_MJ_READ` e `IRP_MJ_DEVICE_CONTROL`.

   Estes pacotes, ao serem identificados, são tratados por rotinas específicas, definidas em `DriverEntry`. A rotina `DispatchRead`, entre outras atividades, "marca" o atual pacote `IRP_MJ_READ` por meio da rotina `IoSetCompletionRoutine` ilustrada no código 3.3:

   Sua função é especificar um ponteiro para a rotina de *callback*[65] `CompletionRoutine`,

[65]Referência para um função no código que é passada como argumento para outra função, a fim da primeira ser invocada futuramente.

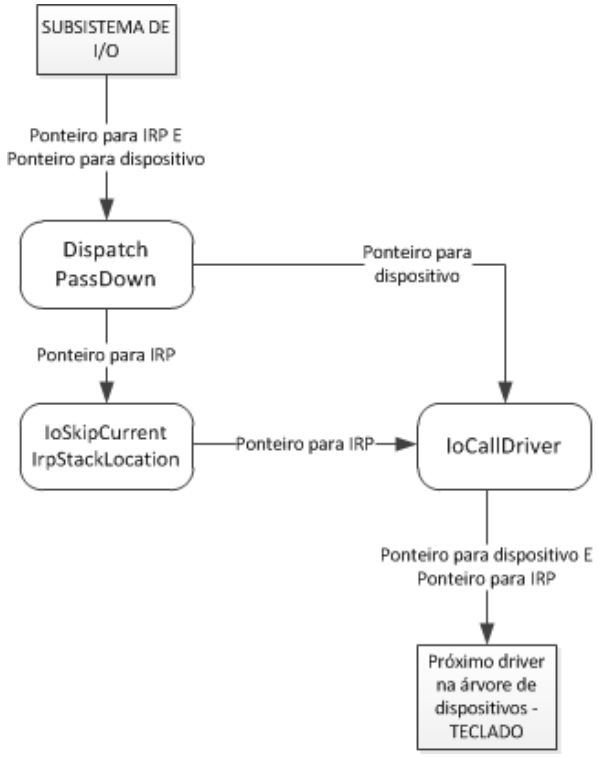

Figura 3.4: Repassando pacotes IRP.

```
NTSTATUS IoCallDriver(
  __in     PDEVICE_OBJECT DeviceObject,
  __inout  PIRP Irp
);
```

Código 3.4: Rotina IoCallDriver

a ser chamada apenas quando o *driver* logo abaixo na árvore de dispositivos completar a requisição de *I/O*, ou seja, invocar a rotina `IoCompleteRequest`. Conforme se nota nos três últimos parâmetros do código 3.3, pode-se configurar esta rotina para ser invocada em caso de sucesso, erro ou cancelamento.

O corpo da rotina `CompletionRoutine` é especificado pelo dispositivo de filtro, e um ponteiro para ela é inserido no próprio pacote *IRP*, possibilitando assim que seja chamada logo após o *driver* do dispositivo de teclado invocar a rotina `IoCompleteRequest`, ou seja, logo após o código de uma tecla digitada for inserido no pacote *IRP*.

4. Após ser "marcado", o pacote *IRP* é encaminhado para o dispositivo de teclado por meio rotina `IoCallDriver`:

   Seus parâmetros são:

   (a) DeviceObject: Entrada. Ponteiro para o dispositivo ao qual foi solicitado *I/O*.

   (b) Irp: Entrada e Saída. Ponteiro para o pacote *IRP* que descreve completamente a

```
typedef struct _KEYBOARD_INPUT_DATA {
  USHORT UnitId;
  USHORT MakeCode;
  USHORT Flags;
  USHORT Reserved;
  ULONG  ExtraInformation;
} KEYBOARD_INPUT_DATA, *PKEYBOARD_INPUT_DATA;
```

Código 3.5: Estrutura KEYBOARD_INPUT_DATA

requisição *I/O*.

Após a rotina `IoCallDriver` ser invocada, o pacote *IRP* passa a pertencer ao dispositivo apontado por `DeviceObject`, não podendo mais ser alterado ou encaminhado para outro destinatário pelo dispositivo de filtro. Nesta etapa, resta esperar pela chamada da rotina especificada em `IoSetCompletionRoutine`. Os passos descritos até aqui correspondem ao fluxo de setas pontilhadas da Figura 3.5.

5. Ao ser invocada pelo pacote *IRP* assim que a requisição de *I/O* se completou (ou seja, um código de tecla foi inserido no mesmo) , a rotina `OnReadCompletion` copia este código, presente no *buffer* do pacote *IRP*, para uma estrutura criada especificamente para recebê-lo. Esta estrutura é do tipo `KEYBOARD_INPUT_DATA`:

   Os principais subtipos desta estrutura são:

   (a) MakeCode: Código da tecla pressionada;

   (b) Flags: Normalmente indica se a tecla foi pressionada ou solta.

6. Finalmente, estas informações são inseridas no final de uma lista duplamente encadeada (setas contínuas da Figura 3.5), utilizando a rotina `ExInterlockedInsertTailList`, a fim de serem disponibilizados para uma *thread* encarregada de traduzir tais códigos em letras, números ou símbolos e escrevê-los em um arquivo (setas tracejadas da Figura 3.5) . Esta tradução é necessária porque o código armazenado em `MakeCode` não corresponde ao código *ASCII* referente ao caractere impresso na superfície da tecla pressionada, mas sim a um valor aleatório definido pela *IBM* quando esta criou o primeiro teclado para PCs. (GREBENNIKOV, 2007).

Na figura 3.5 um diagrama contendo os principais parâmetros e processos envolvidos:

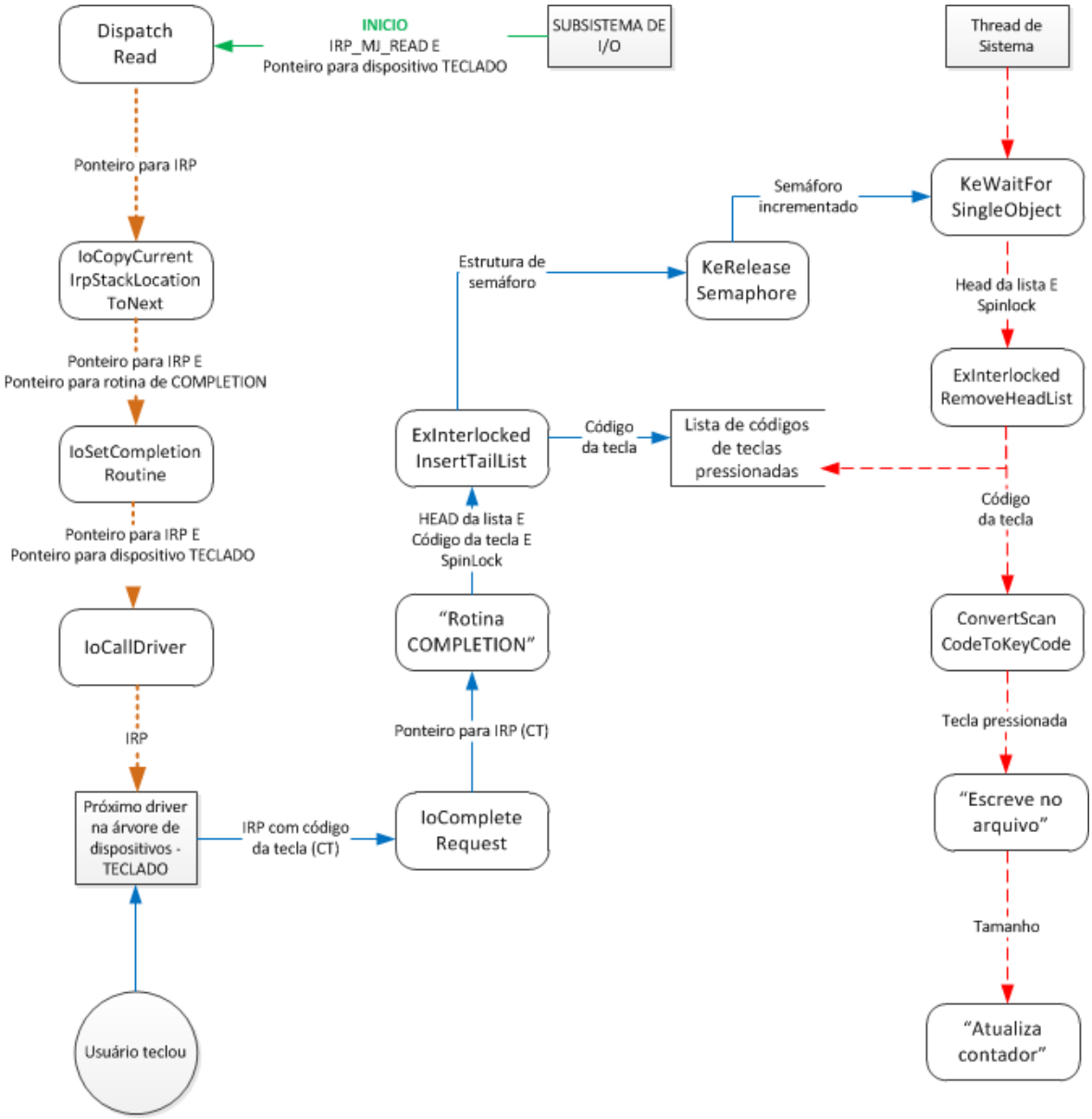

Figura 3.5: Captura das teclas digitadas

```
BOOL WINAPI DeviceIoControl(
  in         HANDLE hDevice,
  in         DWORD dwIoControlCode,
  in       LPVOID lpInBuffer,
  in         DWORD nInBufferSize,
  out       LPVOID lpOutBuffer,
  in         DWORD nOutBufferSize,
  out      LPDWORD lpBytesReturned,
  inout      LPOVERLAPPED lpOverlapped
);
```

Código 3.6: Rotina DeviceIoControl

3.1.2 Recebimento de títulos das janelas ativas

O motivo que levou à decisão de capturar o título da janela ativa em vez do nome do programa em execução veio do fato que o primeiro reúne muito mais informações sobre o que o usuário do computador está fazendo. Por exemplo, um título de janela contendo "Internet Explorer - Yahoo Mail! Email gratuito com armazenamento ilimitado."é mais informativo do que apenas o nome do aplicativo "iexplorer.exe".

Porém, essa decisão acarretou uma dificuldade adicional, que só pôde ser contornada por uma alteração da arquitetura da *FID*. Esta dificuldade será explicada na seção 3.2.3.

Os títulos das janelas ativas são enviados ao *driver* de controle a partir de um serviço, por meio do Código 3.6.

Para que o *driver* receba corretamente esses dados, é necessário definir um código de controle específico - `dwIoControlCode`, conhecido tanto pelo *driver* quanto pelo serviço remetente. Este código será enviado ao *driver* indicado em `hDevice` juntamente com o título da janela ativa, este último armazenado no local apontado por `lpInBuffer` e de tamanho `InBufferSize`.

A chamada da rotina `IoDeviceControl` faz com que o gerenciador de *I/O* crie um pacote *IRP* do tipo `IRP_MJ_DEVICE_CONTROL` e o envie para o *driver* especificado.

A rotina para tratar os pacotes `IRP_MJ_DEVICE_CONTROL` destinados ao *driver* de controle copia os dados existestes no *buffer* do pacote *IRP* para *buffers* locais e faz um *switch*, de modo que consiga tratar todos os códigos `dwIoControlCode` recebidos. Por

```
NTSTATUS ZwCreateFile(
  __out     PHANDLE FileHandle,
  __in      ACCESS_MASK DesiredAccess,
  __in      POBJECT_ATTRIBUTES ObjectAttributes,
  __out     PIO_STATUS_BLOCK IoStatusBlock,
  __in_opt  PLARGE_INTEGER AllocationSize,
  __in      ULONG FileAttributes,
  __in      ULONG ShareAccess,
  __in      ULONG CreateDisposition,
  __in      ULONG CreateOptions,
  __in_opt  PVOID EaBuffer,
  __in      ULONG EaLength
);
```

Código 3.7: Rotina ZwCreateFile

enquanto, existe apenas o código "Envio do título da janela". Porém, essa abordagem é expansiva, podendo acomodar futuros códigos para tratar novas funcionalidades.

Basicamente, a rotina que trata o código "Envio do título da janela"copia os valores dos *buffers* locais para variáveis cujos tipos são aceitos pela rotina "Escreve no arquivo", invocada também para escrever as teclas digitadas no arquivo.

Após isso, a requisição se completa. Para tanto, é invocada a rotina IoCompleteRequest, e o valor `STATUS_SUCCESS` é retornado caso tudo tenha funcionado.

3.1.3 Armazenamento temporário de dados

Todos os dados obtidos (data e hora, informações de sistema, títulos das janelas ativas e teclas digitadas) são primeiramente armazenados em um arquivo de texto discreto, até que este arquivo atinja um determinado tamanho. A partir daí, é enviado pela rede.

Para criar o arquivo de texto discreto que receberá as informações capturadas, o código 3.7 é invocado durante a inicialização do *driver*:

Seus principais parâmetros são:

- FileHandle: Saída. *Handle*[66] do arquivo recém criado ou aberto;

[66] É um ponteiro opaco, ou seja, que não contém explicitamente o endereço apontado, e que se refere a um recurso alocado em memória.

- DesiredAccess: Entrada. Máscara contendo o tipo de acesso desejado para este arquivo, como “leitura”, “escrita”, “deleção”, entre outras permissões de objeto e de arquivo;
- ObjectAttributes: Ponteiro para uma estrutura que contém o nome e o endereço do arquivo, entre outros atributos, como `OBJ_EXCLUSIVE` (apenas um *handle* pode ser aberto para este objeto) ou `OBJ_KERNEL_HANDLE` (este *handle* só pode ser acessado a partir do modo *kernel*);
- IoStatusBlock: Saída. Ponteiro para uma estrutura que recebe o *status* da operação recém efetuada. Pode ser: `FILE_CREATED, FILE_OPENED, FILE_OVERWRITTEN, FILE_SUPERSEDED, FILE_EXISTS ou FILE_DOES_NOT_EXIST`.

Foram escolhidos um nome e um endereço de modo que este arquivo seja confundido com um arquivo de sistema. Optou-se pela dissimulação em vez de tornar o arquivo invisível pelo fato desta última técnica ser facilmente detectável(RUTKOWSKA, 2006).

Ao contrário de data e hora e informações do sistema, que são solicitadas pelo *driver*, dados contendo as teclas digitadas e os títulos das janelas ativas podem chegar a qualquer momento. Por isso, é necessário criar *threads* para copiar tais dados no arquivo de texto tão logo cheguem.

Então, também durante a inicialização do *driver*, são invocadas as seguintes rotinas:

- `InitializeListHead`: Inicializa uma lista duplamente encadeada que receberá os códigos enviados pelo dispositivo de teclado;
- `KeInitializeSpinLock`: *spinlock* que sincronizará o acesso à lista. Em outras palavras, impedirá que mais de um item da lista seja inserido ou retirado ao mesmo tempo;
- `KeInitializeSemaphore`: Inicializa o semáforo que sinalizará quando um novo código for inserido na lista;
- `PsCreateSystemThread`: Cria uma *thread* que executa em modo *kernel*.

O ponteiro para o primeiro elemento da lista e os estados do semáforo e do *spinlock* são armazenados na extensão do dispositivo de filtro (*DeviceExtension*). Assim, eles mantêm seu estado e permanecem disponíveis para consulta sempre que for preciso.

```
PLIST_ENTRY ExInterlockedRemoveHeadList(
  __inout  PLIST_ENTRY ListHead,
  __inout  PKSPIN_LOCK Lock
);
```

Código 3.8: Rotina ExInterlockedRemoveHeadList

```
NTSTATUS ZwWriteFile(
  __in      HANDLE FileHandle,
  __in_opt  HANDLE Event,
  __in_opt  PIO_APC_ROUTINE ApcRoutine,
  __in_opt  PVOID ApcContext,
  __out     PIO_STATUS_BLOCK IoStatusBlock,
  __in      PVOID Buffer,
  __in      ULONG Length,
  __in_opt  PLARGE_INTEGER ByteOffset,
  __in_opt  PULONG Key
);
```

Código 3.9: Rotina ZwWriteFile

A *thread* criada é posta em estado de espera indefinida, por meio da rotina `KeWaitForSingleObject`, até que o semáforo sinalize que um novo item foi inserido na lista.

Após a sinalização, o item da lista é retirado da mesma por meio do código 3.8.

Esta rotina recebe o ponteiro para o primeiro item da lista `ListHead` e, respeitando a sincronia de acesso proporcionada por `Lock`, retorna este ponteiro.

Então, o valor apontado por este ponteiro é inserido numa estrutura similar à do item da lista, porém que pode ser manipulada. Esta estrutura, que contém o código da tecla pressionada e o estado da tecla (se foi pressionada ou se foi solta), é enviada como parâmetro para uma função que converterá o código em um número, letra ou símbolo. Esta função se utiliza de um mapeamento código/tecla que pode ser customizado, além de interpretar o estado das teclas CAPS LOCK, SHIFT, ALT.

Após a conversão, a tecla resultante será escrita no arquivo. Para isso, é invocada a função "Escreve no arquivo"que recebe o *handle* para o arquivo, o texto a ser inserido nele e o tamanho deste texto. Esta função recebe todos os dados capturados pelo sistema, escrevendo-os num único arquivo.

Dentro desta função, o código 3.9 é invocado:

Esta rotina, além de receber os parâmetros citados no parágrafo anterior, recebe também a posição de *offset* em que se deseja escrever no arquivo apontado pelo *handle* - `ByteOffset`. No código, este parâmetro pode assumir dois valores:

1. `NULL`: Um dos parâmetros utilizados na rotina para criar o arquivo (`CreateOptions`) faz com que o gerenciador de *I/O* armazene a posição de *offset* atual do arquivo. Assim, ao passar o valor `NULL` em `ByteOffset`, esta posição é considerada quando se deseja escrever novos dados no arquivo;

2. `0`: Logo após os dados serem enviados pela rede, o arquivo de texto começa a ser sobrescrito com novos dados capturados. Assim, consegue-se manter um pequeno tamanho de arquivo, além de apagar evidências antigas de dados capturados.

Na figura 3.6, um diagrama contendo os principais parâmetros e processos envolvidos no recebimento dos títulos das janelas ativas e escrita destes no arquivo de log.

3.1.4 Cifragem de dados

Os dados armazenados no arquivo de texto discreto são cifrados utilizando o algoritmo *DES triplo*, no modo de operação *CBC*, antes de serem enviados pela rede. Por se tratar de informação sigilosa, tal artifício faz-se extremamente necessário, a fim de evitar vazamento de dados no caminho até o servidor.

Para isso, será utilizado a *API* de criptografia *CNG*, ou *Cryptography API: Next Generation*(NIST, 2011)

Inicialmente, é necessário gerar uma chave simétrica para cifrar e decifrar dados. Isto é feito durante a primeira carga do *driver*. São necessários, basicamente, os seguintes passos:

1. Adquirir um *handle* para um provedor do algoritmo *DES triplo*, que deverá ser informado quando da criação da chave simétrica. Tal *handle* é adquirido por meio da rotina `BCryptOpenAlgorithmProvider`;

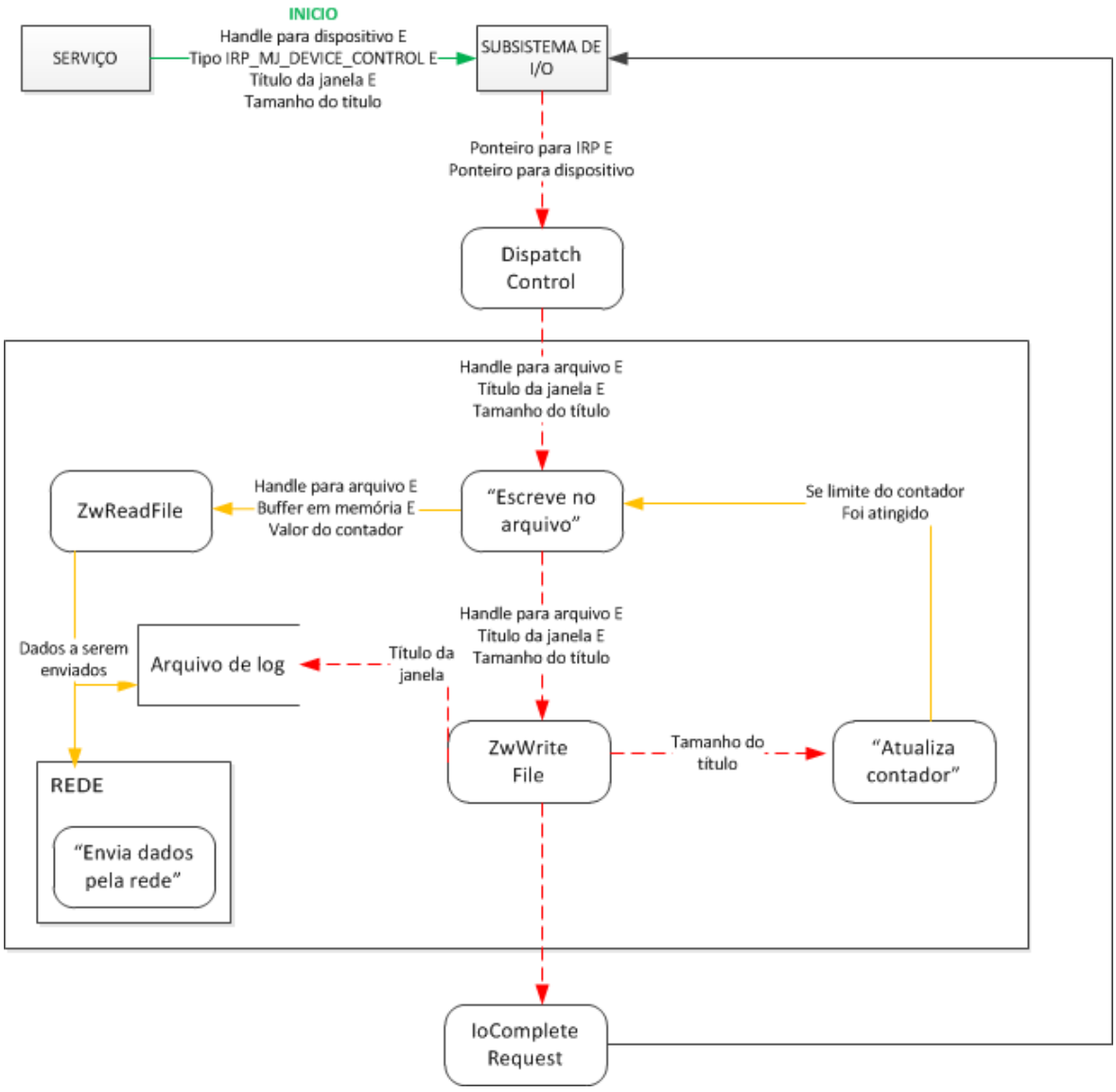

Figura 3.6: Armazenamento temporário de dados.

```
NTSTATUS WINAPI BCryptGenerateSymmetricKey(
  __inout     BCRYPT_ALG_HANDLE hAlgorithm,
  __out       BCRYPT_KEY_HANDLE *phKey,
  __out_opt   PUCHAR pbKeyObject,
  __in        ULONG cbKeyObject,
  __in        PUCHAR pbSecret,
  __in        ULONG cbSecret,
  __in        ULONG dwFlags
);
```

Código 3.10: Rotina BCryptGenerateSymmetricKey

2. Alocar um *buffer* do tamanho da chave para recebê-la. O tamanho deste *buffer* é obtido por meio de uma chamada à rotina `BCryptGetProperty` passando o parâmetro `BCRYPT_BLOCK_LENGTH`;

3. Gerar uma sequência pseudo-aleatória de bytes que formarão a chave, tarefa realizada pela rotina `BCryptGenRandom`;

4. Descobrir o tamanho do bloco utilizado pelo algoritmo *DES triplo*. Novamente é invocada a rotina `BCryptGetProperty`, porém desta vez passando o argumento `BCRYPT_OBJECT_LENGTH`. Isto é necessário para definir o tamanho necessário de *IV* para ser corretamente utilizado pela cifra;

5. Definir o modo de operação *CBC*, o que é feito pela rotina `BCryptSetProperty`, passando como parâmetros a propriedade que se deseja configurar (`BCRYPT_CHAINING_MODE`) e o valor (`BCRYPT_CHAIN_MODE_CBC`;

6. Gerar a chave, utilizando a rotina ilustrada no código 3.10:

   Seus principais parâmetros são:

   - hAlgorithm: *handle* para o provedor de algoritmo escolhido, no caso o *DES triplo*;
   - *phKey: Ponteiro para o *handle* da chave recém criada;
   - pbKeyObject e cbKeyObject: ponteiro para o *buffer* que receberá a chave e o tamanho do mesmo, respectivamente;
   - pbSecret e cbSecret: ponteiro para o *buffer* contendo os dados segundo os quais será gerada a chave, e o tamanho do mesmo.

Após a chave simétrica ser gerada, é necessário enviá-la ao servidor.

Para isso, primeiramente há de se exportar a chave, em um formato compatível para que a mesma possa ser empregada externamente ao aplicativo em que foi gerada, no caso pelo *script*[67] *PHP*[68] rodando no servidor *WEB Apache*. Esta tarefa é executada pela rotina `BCryptExportKey`, que exporta os dados da chave para um *BLOB*[69] (a fim de permitir que seja importada posteriormente), no formato `BCRYPT_KEY_DATA_BLOB`. Tal formato consiste basicamente na chave precedida de um cabeçalho contendo um código identificador, a versão do formato e o tamanho da chave.

Tal como o envio dos dados, o envio da chave simétrica também requer proteção contra vazamento de informações. Afinal, quem tiver acesso a ela poderá decifrar os dados. Assim, decidiu-se pela aplicação de criptografia assimétrica nesta chave, gerando um par de chaves *RSA* pública/privada, a fim de garantir a confidencialidade.

Para gerar o par de chaves assimétricas, são tomados basicamente os mesmos passos utilizados na geração da chave simétrica, porém obedecendo à peculiaridades do novo algoritmo, como por exemplo ser desnecessária a utilização de um vetor de inicialização.

A criação destas chaves deve ocorrer antes de gerar o executável da *FID*, pois a chave pública será escrita no código da *FID*, para que possa ser importada e aplicada no momento da cifragem da chave simétrica. Então, foi desenvolvido um aplicativo à parte, cuja única função é gerar um par de chaves pública/privada, escrevendo cada uma em arquivos separados devidamente identificados.

Assim, o *driver*, ao ser iniciado, executa as seguintes funções:

- Criação e exportação de uma chave simétrica para um *BLOB*, conforme já explicado anteriormente;
- Importação do *BLOB* que contém os dados da chave pública, presente no código fonte, por meio da rotina `BCryptImportKeyPair`. Além da chave, são passados para ela o *handle* para o provedor *RSA*, um *buffer* que armazenará a chave resultante e o tipo de chave que se está importando;
- Cifragem do *BLOB* que contém os dados da chave simétrica, enviando-o como parâmetro para a rotina `BCryptEncrypt` juntamente com a chave pública recém importada,

[67] Termo genérico que se refere a uma linguagem de programação interpretada, que pode ser executada dentro de um programa.
[68] Linguagem de *script open source*.
[69] Estrutura de dados que contém dados binários.

além de outros parâmetros;

- Envio do criptograma resultante pela rede, utilizando a mesma rotina de envio dos dados capturados, porém com um diferente identificador de conteúdo anexado ao corpo do pacote *HTTP*[70] *POST*[71].

Uma vez criada a chave simétrica, ela será utilizada sempre antes de enviar os dados capturados, cifrando-os logo após serem lidos a partir do arquivo discreto. A rotina utilizada é a `BCryptEncrypt`.

Por razões de segurança, uma nova chave simétrica é gerada após o decurso de determinado período, estipulado em código. Para isso, a cada inicialização a *FID* confere se a data atual do sistema é maior do que a data limite estipulada. Caso positivo, a nova chave simétrica será gerada, cifrada com a chave pública e enviada para o servidor.

3.1.5 Envio de dados pela rede

Após uma determinada quantidade de dados for escrita no arquivo de texto discreto, estes serão enviados pela rede, e novos dados sobrescreverão os antigos. Esta quantidade foi definida de modo que o tamanho total de dados enviados não ultrapasse o valor de *MTU*[72] estabelecido para redes *ethernet*[73], ou seja, `1500 bytes`. A rotina "Verifica tamanho para envio"é invocada quando esta quantidade é atingida, e recebe como parâmetros um *handle* para o arquivo e a quantidade de bytes `size` presente nele.

Ela então invoca a rotina `ZwReadFile`, que transfere os dados lidos do arquivo para um *buffer* previamente alocado em memória não paginada e de tamanho `size`. De posse dos dados a serem enviados, serão adicionados um número sequencial e um selo de data e hora. Todo esse conjunto de dados será criptografado, conforme explicado na seção 3.1.4, e só então será invocada a rotina "Envia pela rede", que recebe quatro parâmetros:

[70] Protocolo de aplicação responsável pelo tratamento de requisições e repostas num modelo "'Cliente-Servidor"' dentro de uma rede de computadores.

[71] Método de requisição HTTP utilizado quando o cliente necessita enviar dados para o servidor, tais como enviar um arquivo ou submeter um formulário.

[72] Unidade máxima de transferência. O segmento a ser transmitido em uma rede deve caber em sua MTU. (TANENBAUM, 2002)

[73] Rede de difusão de barramento e de controle descentralizado, em que todos computadores podem transmitir sempre que desejam. Em geral opera em velocidades entre 10 Mbps e 10 Gbps. (TANENBAUM, 2002)

1. *IP* ou *URL* para onde os dados serão enviados;

2. porta especificada para receber os dados no servidor remoto;

3. dados a serem enviados;

4. tamanho total dos dados a serem enviados.

Obrigatoriamente, por se tratar de uma investigação policial, este *IP* deve ser discreto, sendo inclusive recomendado que pertença a um provedor de outro país, de modo a não levantar suspeitas.

A função "Envia pela rede"utiliza rotinas da interface de programação de rede (*NPI*[74] - *Network Programming Interface*) *WSK*[75] -*Winsock Kernel*. Esta *NPI* foi implementada nos sistemas *Windows* a partir do *Vista*, e seu principal objetivo foi prover uma interface em modo *kernel* mais eficiente e simples (CORPORATION, 2006). Apesar de ser inspirada na *NPI* de modo usuário *Winsock2*[76], a *WSK* é uma interface de programação nova, e inclui novidades como a utilização de pacotes *IRP* para completar operações de *I/O* de rede de forma assíncrona.

Basicamente, as etapas para o envio são:

1. Criação de um soquete por meio da rotina `WskSocket`, que inicializa o soquete e retorna um ponteiro para o mesmo. Nele são especificados o formato de endereçamento desejado (*IPv4*), o tipo de serviço desejado (orientado a conexão), o protocolo (TCP) e o ponteiro para um pacote *IRP* alocado pelo usuário;

2. Associação de um endereço completo (endereço IP e número de porta) ao soquete recém criado, por meio da rotina `WskBind`. Este é o endereço do remetente;

3. Conexão ao servidor remoto, por meio da rotina `WskConnect`. É passado como parâmetro, além do soquete, os endereço IP e número de porta remotos, ao quais se deseja enviar as informações.

[74] Interface de comunicação entre dois módulos de rede que contém funções para que ambos possam comunicar-se entre si.

[75] NPI para desenvolvimento em modo *kernel*.

[76] Especificação técnica que define como *softwares* de rede do *Windows* devem utilizar os serviços de rede. (**??**)

Após a conexão, é criada uma área em memória não paginada com tamanho suficiente para armazenar temporariamente os dados a serem enviados. Nesta área, é criado um pacote *HTTP POST*, de conteúdo tipo `application/x-www-form-urlencoded` (são enviadas duplas chave-valor, em que chave é um identificador, como `captura=` e valor corresponde aos dados capturados). Optou-se pelo *HTTP POST* por permitir a criação de um script em linguagem *PHP*, que ficará rodando no servidor, e cuja função seja receber e disponibilizar os dados.

Ao pacote montado e armazenado em memória, são anexados um selo de data e hora (para fins de identificar possíveis discrepâncias entre o horário da máquina alvo e do servidor) e um número sequencial (para identificar possíveis perdas de pacotes na transmissão). Então, todo este conteúdo é cifrado por meio da rotina `BCryptEncrypt` e enviado ao servidor remoto por meio da rotina `WskSend`. Ambas rotinas recebem como parâmetro um ponteiro para este *buffer* e o tamanho do mesmo.

A resposta do servidor é escrita em um vetor de caracteres previamente criado por meio da rotina `WskReceive`. Então, algumas posições neste vetor são verificadas a fim de descobrir o código da resposta do servidor. Um código `200` informa que os dados foram recebidas com sucesso.

Finalmente, são invocadas as rotinas `WskDisconnect` e `WskCloseWskSocket`, que finalizam a conexão e fecham o soquete. Optou-se por não manter as conexões abertas para evitar detecção das mesmas. Uma execução do comando `netstat`, por exemplo, que ilustra todas as conexões que estão abertas no momento, identificaria a porta utilizada pela solução, e poderia ser um ponto de partida para uma análise mais profunda com o intuito de descobrir qual aplicação está utilizando a mesma.

A figura 3.7 apresenta um diagrama contendo os principais parâmetros e processos do sistema de envio.

3.2 serv1.exe

O aplicativo `serv1.exe` é o único dos três módulos que precisa ser executado para que a *FID* entre em funcionamento. Ele possui 3 responsabilidades principais: instalar e iniciar o *driver* `filtec`, instalar e iniciar o serviço `serv1` e enviar para o *driver* `filtec` os títulos capturados das janelas ativas.

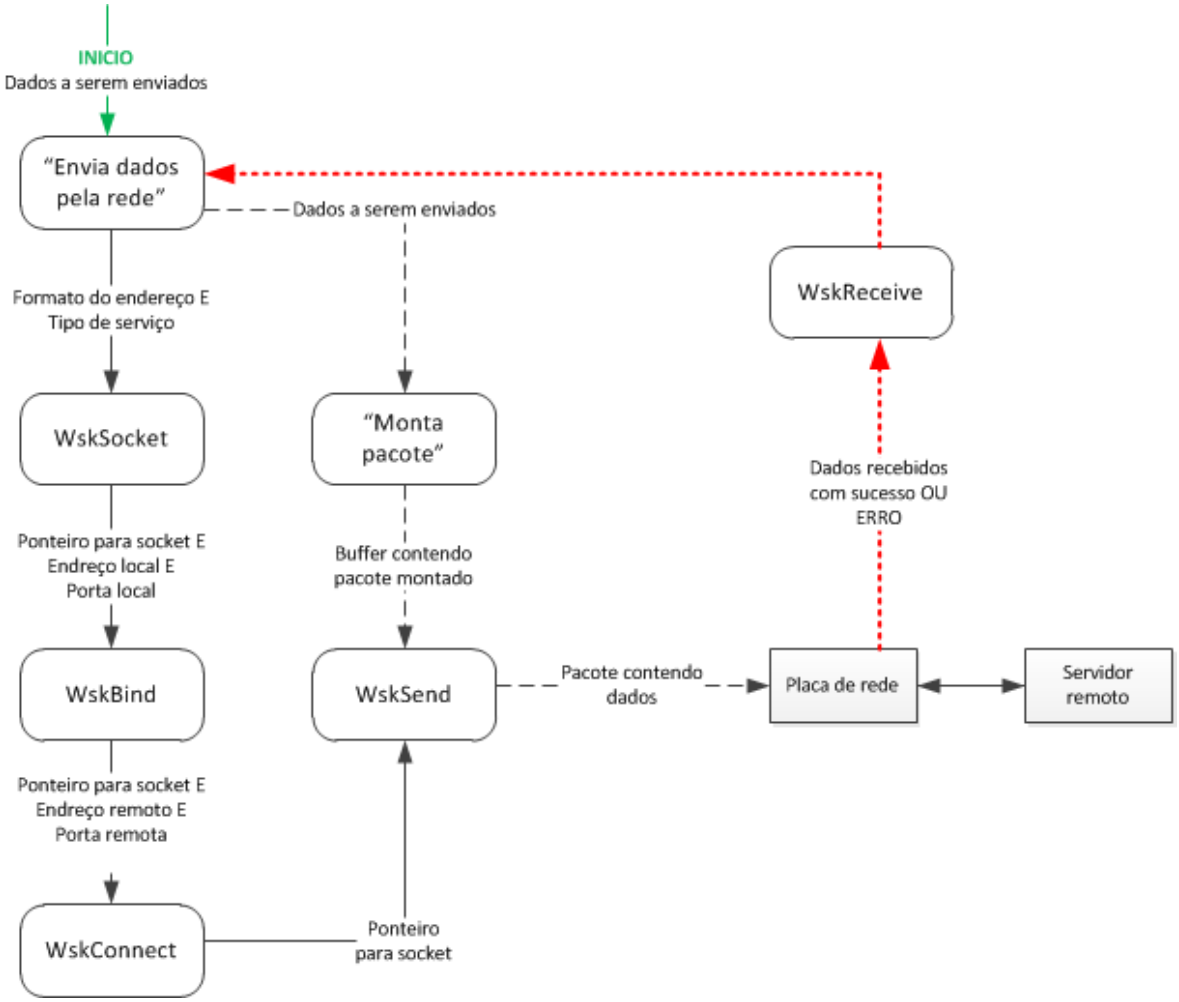

Figura 3.7: Envio de dados pela rede

```
SC_HANDLE WINAPI CreateService(
  __in        SC_HANDLE hSCManager,
  __in        LPCTSTR lpServiceName,
  __in_opt    LPCTSTR lpDisplayName,
  __in        DWORD dwDesiredAccess,
  __in        DWORD dwServiceType,
  __in        DWORD dwStartType,
  __in        DWORD dwErrorControl,
  __in_opt    LPCTSTR lpBinaryPathName,
  __in_opt    LPCTSTR lpLoadOrderGroup,
  __out_opt   LPDWORD lpdwTagId,
  __in_opt    LPCTSTR lpDependencies,
  __in_opt    LPCTSTR lpServiceStartName,
  __in_opt    LPCTSTR lpPassword
);
```

Código 3.11: Rotina CreateService

3.2.1 Instalar e iniciar o *driver* filtec.sys.

Ao ser executado, a primeira tarefa executada pelo aplicativo é registrar o *driver* no *Service Control Manager*, ou apenas *SCM*. Este é um serviço especial do Windows, que se destina a iniciar, parar e interagir com outros serviços. Para se instalar um *driver*, a rotina ilustrada em 3.11 é invocada:

Seus principais parâmetros são:

- hSCManager: Entrada. *handle* para o *SCM*, obtido por meio de uma chamada À rotina `OpenSCManager`;
- lpServiceName: Entrada. Nome do *driver*, a ser exibido na lista de *drivers* instalados. Pode ser escolhido qualquer nome, de preferência um que possa ser confundido com um *driver* nativo do sistema.
- dwServiceType: Entrada. Tipo de serviço. No caso, foi escolhido SERVICE_KERNEL_DRIVER;
- dwStartType: Entrada. Tipo de inicialização do serviço. Para sempre iniciar junto ao sistema, foi escolhido SYSTEM_AUTO_START;
- lpBinaryPathName: Entrada. Caminho completo para o local onde o arquivo do *driver* (`filtec.sys`) está armazenado;

- lpServiceStartName: Entrada, opcional. Nome da conta dentro da qual o serviço deve executar. Ao ser atribuído o valor `NULL` a este parâmetro, o serviço será executado na conta *LocalSystem Account*[77], que possui privilégios de administrador.

Notadamente, esta rotina cria uma chave de registro, de mesmo nome que o *driver*, abaixo da seguinte chave de registro: `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services`. Assim, é possível, por exemplo, a inicialização do *driver* junto com o sistema.

Depois da criação, é necessário iniciar o *driver*, o que é feito pela rotina `StartService`, passando como parâmetros o número de argumentos, os argumentos em si e um *handle* `schService` para o serviço desejado. Este último é obtido pelo retorno da rotina `OpenService`, para a qual é passado o nome do serviço (no caso, `filtec`). Como resultado, será chamada a rotina `Driver Entry` do *driver* em questão, explicada na seção 2.2.2.

Apesar de ainda não haver sido implementado um contador regressivo para que a *FID* se desinstale automaticamente, as rotinas para tal já estão implementadas, inclusive foram alvo de testes de estabilidade, em que seguidamente a *FID* era instalada e iniciada, e, após, parada e desinstalada manualmente. Ao invocar-se a rotina `ControlService`, passando como parâmetro o mesmo *handle* `schService` acima, obtido por meio da rotina `OpenService`, e também a opção `SERVICE_CONTROL_STOP`, o serviço de *driver* é parado. E ele é desinstalado invocando-se a rotina `DeleteService` passando-se como parâmetro também o mesmo *handle* `schService`.

A partir deste ponto, os dispositivos de filtro e teclado já foram criados e iniciados, ou seja, as teclas já estão sendo capturadas, e o *driver* já está preparado para receber e tratar os títulos das janelas ativas, faltando apenas serem enviadas.

3.2.2 Instalar o serviço serv1 no Windows.

Como segunda tarefa, o aplicativo `serv1.exe` é responsável por criar o serviço `serv1`.

[77] Conta do *Windows* na qual os principais componentes do modo usuário do *Windows* executam. (RUSSINOVICH; SOLOMON; IONESCU, 2009)

Serviços são aplicações "especiais"que executam em segundo plano, independente de contexto de usuário, e que podem ser configurados para permanecerem em execução concomitantemente com o sistema e interagirem com este, sem que seja necessário que o administrador inicie-os a cada *boot*. Por exemplo, um servidor *WEB* é um serviço.

A decisão de se criar e manter em execução o serviço `serv1` foi tomada após constatar-se que esconder os arquivos da *FID* em diretórios comuns levanta menos suspeitas do que torná-los invisíveis. (RUTKOWSKA, 2006). Ainda, o fato do serviço criado executar com privilégios de administrador e ser sempre iniciado juntamente com o sistema torna possível uma presença permanente da *FID* no sistema, evitando a necessidade um novo ataque de engenharia social.

O serviço `serv1` é registrado no *SCM* por meio da mesma rotina utilizada pelo *driver*, `CreateService`. Porém, além de nome e caminho diferentes, o seu tipo é declarado como `SERVICE_WIN32_OWN_PROCESS` e `SERVICE_INTERACTIVE_PROCESS`, ou seja, é um serviço que executa em seu próprio processo (não divide o mesmo processo com outros serviços) e também necessitará interação com o usuário.

Após isso, a rotina `SvcMain` é cadastrada no `SCM` como sendo a rotina a ser invocada quando o serviço for iniciado.

Então, é necessário iniciar o serviço, por meio da chamada `net start serv1`. Esta chamada invocará a função cadastrada no `SCM` `SvcInit`.

Na Figura 3.8, um diagrama retratando resumidamente a inicialização do *driver* `filtec` e do serviço `serv1`.

3.2.3 Enviar para o *driver* de controle os títulos capturados das janelas ativas

Outra decisão de projeto foi a de enviar ao *driver* de controle o título da janela ativa em vez do nome do programa em execução, por motivo explicado na seção 3.1.2. Essa decisão trouxe uma dificuldade que não existia no *Windows XP*: a impossibilidade de interação direta entre um serviço do *Windows* com a interface gráfica do usuário logado.

A captura do título da janela ativa é feita por meio do código 3.12.

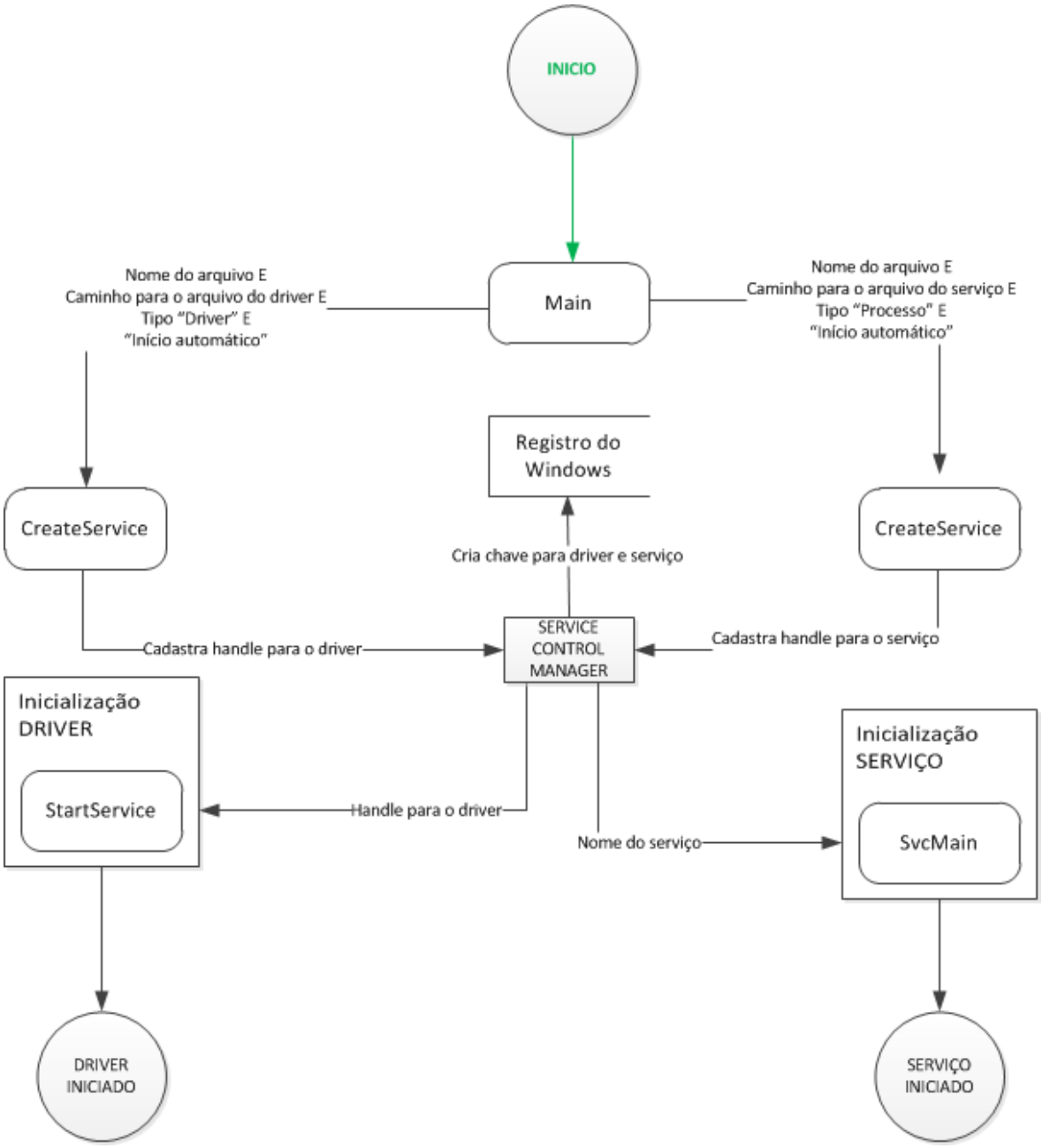

Figura 3.8: Registro e inicialização do *driver* e do serviço.

```
HWINEVENTHOOK WINAPI SetWinEventHook(
  __in  UINT eventMin,
  __in  UINT eventMax,
  __in  HMODULE hmodWinEventProc,
  __in  WINEVENTPROC lpfnWinEventProc,
  __in  DWORD idProcess,
  __in  DWORD idThread,
  __in  UINT dwflags
);
```

Código 3.12: Rotina SetWinEventHook

Ao ser invocada, é possível especificar uma rotina de *callback* (`lpfnWinEvent Proc`) a ser chamada quando um evento ocorrer. Para isso, o código deste evento tem que estar presente dentro do intervalo definido entre `eventMin` e `eventMax`. Existe uma lista de eventos que podem ser registrados para notificação, dentro deles vários relacionados a interface de usuário. Para receber notificações de alteração de janela ativa, é necessário cadastrar o evento `EVENT_SYSTEM_FOREGROUND`. Esta mesma abordagem pode ser utilizada em futuras atualizações da ferramenta, caso seja necessário capturar, por exemplo, cliques de *mouse*.

Essa abordagem funciona para aplicações de usuário e, até o *Windows XP*, funcionava também para serviços. Porém, nas versões *Vista* e 7 do *Windows*, são necessários mais alguns passos. Isto porque, segundo (KIRIATY, 2009), no *Windows XP* e versões anteriores, os serviços e as aplicações de usuário executavam em uma mesma sessão, iniciada pelo primeiro usuário que logasse no sistema (sessão 0, conforme ilustrado na Figura 3.9:

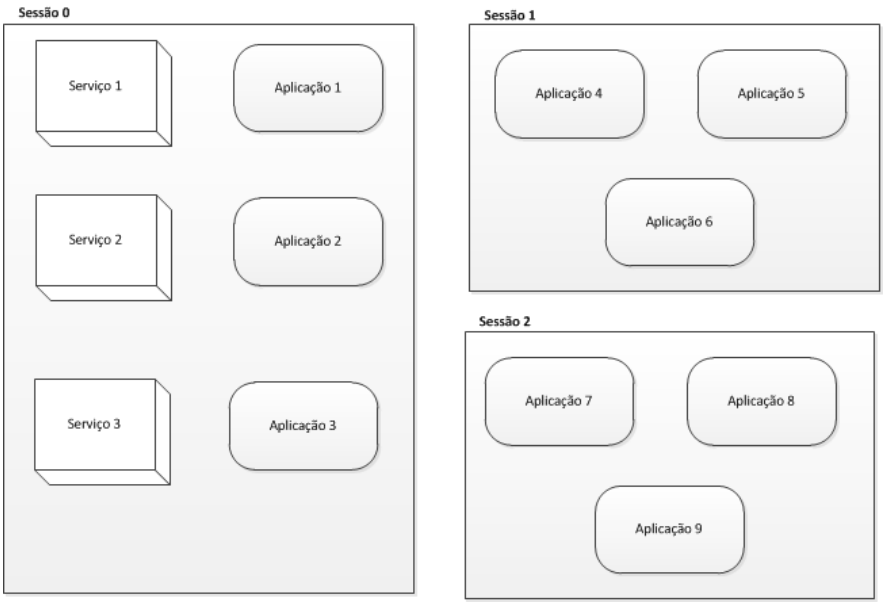

Figura 3.9: Serviços e aplicações dividindo a mesma sessão.Imagem adaptada de (KIRIATY, 2009).

Ainda segundo (KIRIATY, 2009), isto criava um risco de segurança pois, por compartilharem a mesma sessão, os serviços, que executam com privilégios de administrador, eram alvo de aplicações de usuários que desejavam aumentar seus privilégios "sequestrando"estes serviços.

A partir do *Windows Vista*, apenas serviços são executados na sessão 0, mantendo-se assim isolados das aplicações de usuário que executam em sessões subsequentes, criadas à medida que usuários logam no sistema, conforme ilustrado na Figura 3.10.

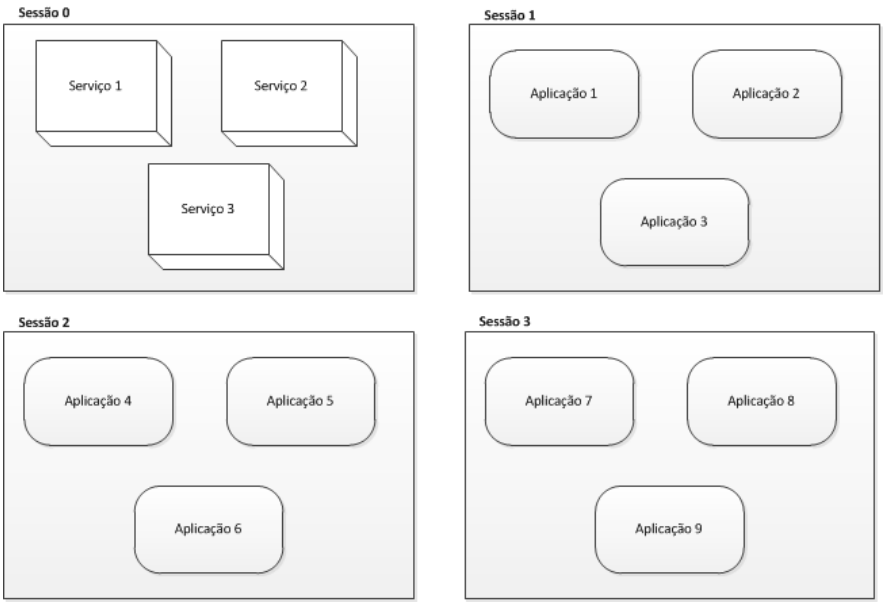

Figura 3.10: Serviços e aplicações isolados. Imagem adaptada de (KIRIATY, 2009).

Assim, entidades (aplicações e serviços) executando em diferentes sessões não podem enviar mensagens entre si, nem compartilhar elementos da interface gráfica (as janelas, por exemplo) nem tampouco objetos de *kernel* diretamente, como era feito no *Windows XP*.

Caso seja necessária esta comunicação, é necessário que ambos se identifiquem para o sistema, além de possuírem determinados privilégios de controle de acesso. Isto torna os ataques de elevação de privilégios bem menos propícios.

Por isso, o serviço `serv1` não é capaz de receber as notificações de eventos ocorridos, no caso, de que a janela ativa foi alterada. Então, torna-se necessária a criação de um processo em modo usuário no *desktop*[78] ativo para tal.

Isto é feito da seguinte forma:

1. Aguardar até um usuário logar no sistema. De acordo com (GOLUB, 2010), isto pode ser feito por meio da rotina `WTSWaitSystemEvent`, passando o parâmetro `WTS_EVENT_LOGON`. Essa função só retorna após detectar um logon;

2. Obter o código da sessão iniciada pelo usuário, por meio da rotina `WTSGetActiveConsoleSessionId`;

[78] Conjunto de objetos que formam a interface para o usuário, como janelas e menus.(MICROSOFT, 2011)

3. De posse do identificador da sessão, obter o *token*[79] de acesso do usuário logado à sessão corrente, o que é feito pela rotina `WTSQueryUserToken`. Esta é uma rotina destinada a ser invocada apenas por serviços confiáveis. Como o serviço `serv1` roda na conta `LocalSystem Account`, ele possui privilégios dos usuários `NT AUTHORITY\SYSTEM` e `BUILTIN\Administrators`, e consegue obter o *token* desejado ao utilizar esta rotina;

4. Realizar uma cópia do *token* recém obtido. Isto é necessário porque qualquer processo executado no escopo deste usuário deve possuir uma cópia do *token* de acesso do mesmo. Essa cópia é realizada por meio da rotina `DuplicateTokenEx`, que produz uma duplicata do *token* de acesso passado como parâmetro;

5. Recuperar as variáveis de ambiente do usuário corrente. Elas serão necessárias para criar um novo processo, e são obtidas por meio da função `CreateEnvironment Block`;

6. Criar um processo de usuário, por meio da rotina `CreateProcessAsUser`. Para ele são passadas, dentre outras informações, o *token* duplicado, o caminho completo para o aplicativo para o qual será criado um processo e as variáveis de ambiente configuradas para o usuário corrente.

Após isto, o processo está criado, com privilégios do usuário corrente. O nome do processo é o mesmo do aplicativo.

Para receber as informações oriundas do aplicativo em modo usuário, é criado um *pipe*[80], por meio da rotina `CreateNamedPipe`.

Finalmente, é criada uma *thread*, por meio da rotina `CreateThread`. Um de seus parâmetros é um ponteiro para uma variável a ser recebida pela *thread* recém criada. No caso, é passado um ponteiro para o *pipe* também recém criado, para que a *thread* possa verificá-lo por novas informações recebidas.

Esta *thread* executa num *loop*[81] infinito. Porém, para evitar ocupar demasiadamente o processador, ao final de cada execução do *loop* é invocada a rotina `Sleep`, passando

[79] Objeto que descreve o contexto de segurança de um processo ou de uma thread. Em outras palavras, contém a identidade e os privilégios da conta de usuário associada ao processo ou thread. (MICROSOFT, 2011)

[80] Canal estabelecido entre dois programas, que encaminha os dados da saída de um deles para a entrada do outro.

[81] Bloco de instruções que é executado indefinidamente, ou até determinada condição ser satisfeita.

o valor `0` como parâmetro. Normalmente, o valor deste parâmetro indica a quantidade de milissegundos durante os quais a execução da *thread* é suspensa. Porém, ao invocar a rotina `Sleep` passando o valor `0` como parâmetro, a *thread* libera o restante do seu *quantum*[82] para outra *thread* em estado "pronta para executar", ao mesmo tempo em que se torna "pronta para executar"novamente. Assim, ela nunca sai da fila, e garante sua execução sempre que houver disponibilidade de processador.

A Figura 3.11 apresenta um diagrama contendo os principais parâmetros e processos citados:

3.3 sendproc.exe

Este é um processo criado em modo usuário pelo serviço `serv1`. Seu objetivo é capturar o título da janela ativa no momento, ou seja, a janela em que o usuário está inserindo ou recebendo *I/O*. Qualquer mudança da janela ativa (por exemplo, quando o usuário clicar com o *mouse* dentro de outra janela) será registrada. Com isso, é possível contextualizar o texto que o mesmo porventura venha a digitar (por exemplo, um texto digitado dentro de uma janela titulada “Entrar - Windows Live Messenger” pode indicar as credenciais (nome de *login* e senha) do usuário para aquele serviço).

Ele foi criado devido à impossibilidade, já citada na seção 3.2.3, de um serviço interagir com a interface gráfica de um usuário diretamente. Tanto a rotina `SetWinEventHook`, que registra o evento a ser capturado, quando a rotina de *callback* `HandleWinEvent`, que trata o evento recebido, pertencem ao aplicativo `sendproc.exe`.

A rotina `HandleWinEvent` retorna, como um dos seus parâmetros, o *handle* para a janela que gerou o evento, caso exista. E, utilizando esse *handle*, o nome do objeto (da janela) é recuperado por meio das rotinas `AccessibleObjectFromEvent` e, posteriormente, `get_accName`.

Os títulos das janelas são enviados ao serviço `serv1` por meio do *pipe* criado por ele. Uma nova instância para este *pipe* é criada por meio da rotina `CreateFile`, passando como parâmetros o nome do *pipe* e a flag `OPEN_EXISTING`. Após obter o *handle* para a instância do *pipe*, é invocada a rotina `WriteFile` para escrever os dados no mesmo.

[82] Tempo disponível para que uma thread possa executar até que a próxima thread na fila, de prioridade igual ou superior, comece a execução.

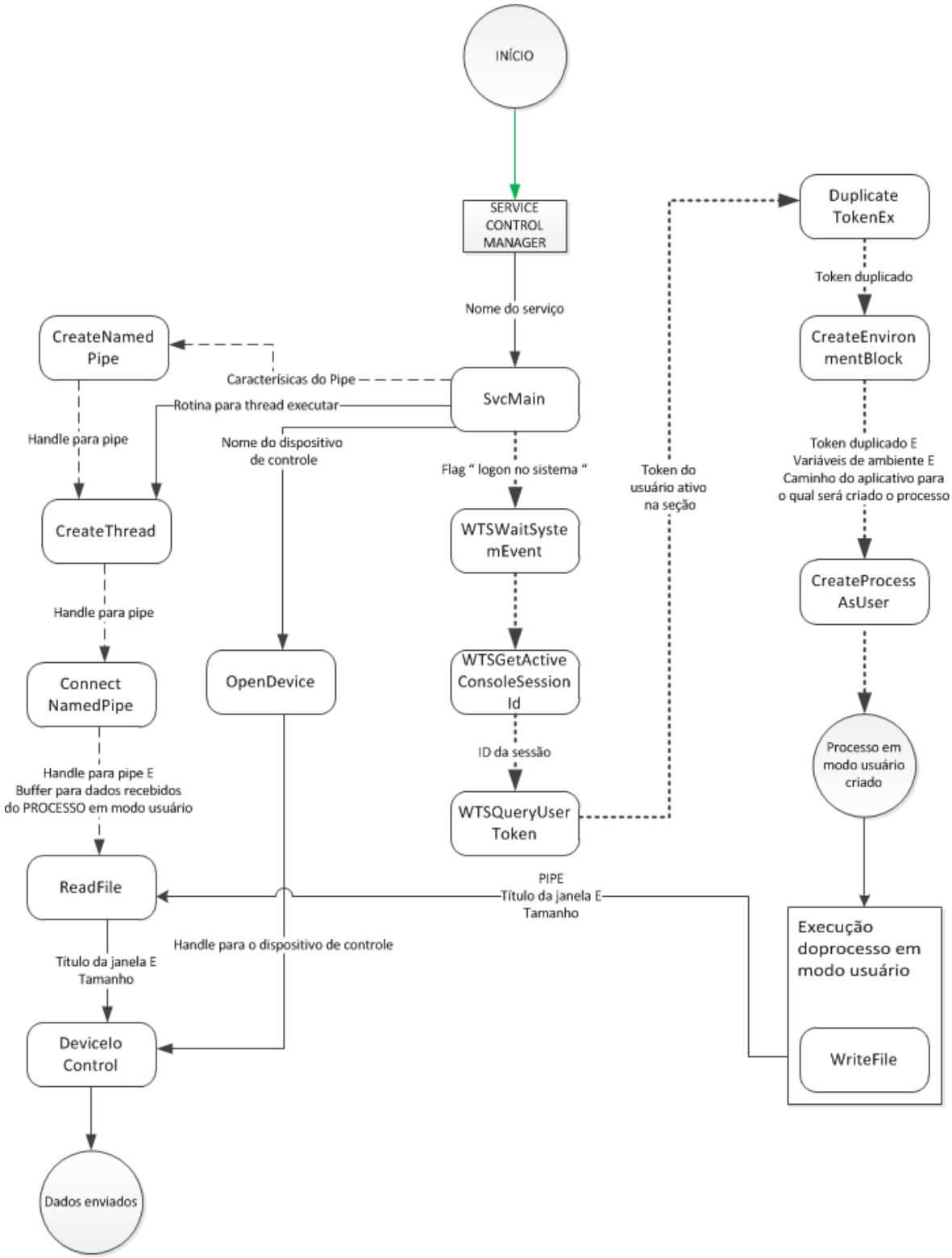

Figura 3.11: Inicialização do serviço.

Daí, a *thread* criada pelo serviço `serv1` recuperará os dados enviados para o *pipe* (no caso, os títulos das janelas ativas) utilizando a rotina `ReadFile`, e por fim enviará os mesmos para o *driver* de controle por meio da rotina `DeviceIoControl` já explicada.

Na figura 3.12, um diagrama contendo os principais parâmetros e processos citados ilustra a execução do processo `sendproc.exe`.

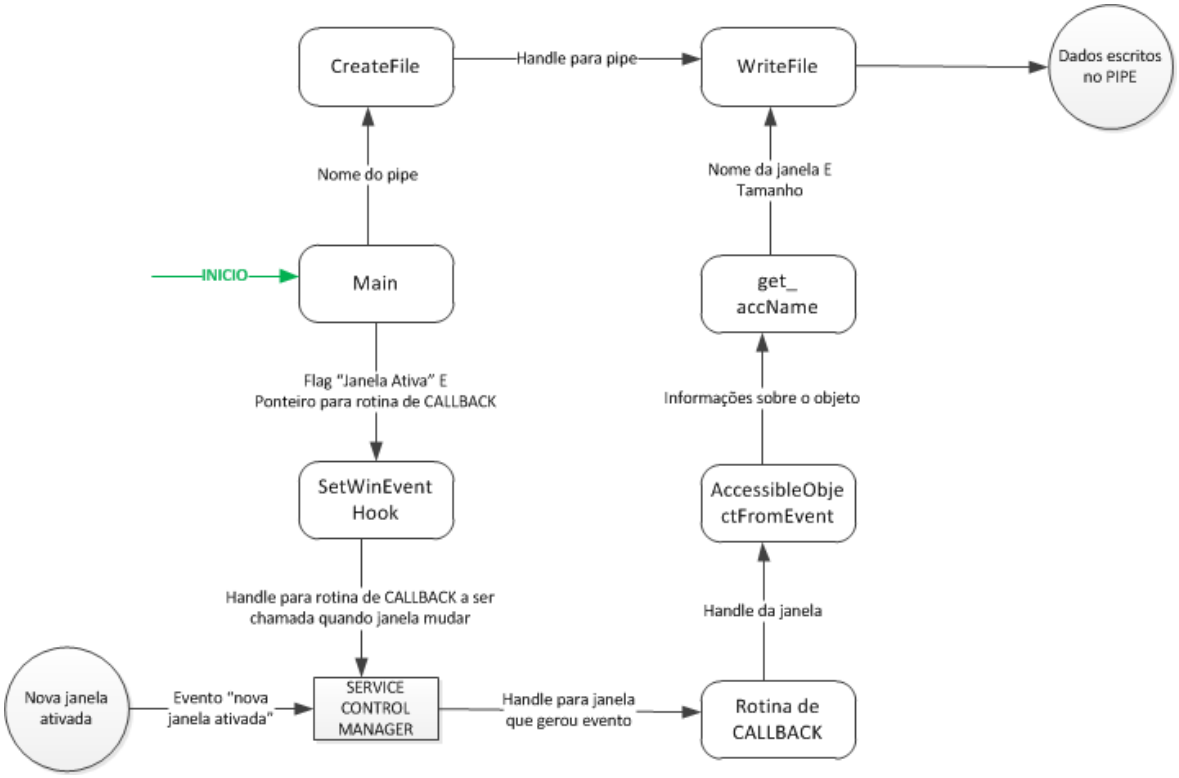

Figura 3.12: Inicialização do processo em modo usuário sendproc.exe.

Este capítulo buscou explicar os principais detalhes da implementação da *FID*, além de apresentar diagramas de fluxo de dados contendo as funções citadas no decorrer do capítulo, com o intuito de fornecer uma visão macro de cada funcionalidade implementada. O próximo capítulo tratará da metodologia a ser aplicada em casos em que o tráfego capturado por interceptações telemáticas encontra-se criptografado. E é em uma das etapas desta metodologia que a *FID* será utilizada.

4 METODOLOGIA PARA INTERCEPTAÇÃO DE DADOS

Após a implementação da parte *software* da solução, faz-se necessário definir em que etapa do processo ela será empregada, pois sozinha não é suficiente para obter os dados protegidos de maneira remota. Também, a sua utilização sem obedecer a metodologia pré-definida pode resultar em fracasso ou, pior, descoberta da investigação por parte do alvo.

Assim, foram definidas 7 etapas que compõem a metodologia:

1. Confirmação da necessidade e aplicabilidade da solução;
2. Solicitação de autorização judicial para instalação remota da solução;
3. Reunião e organização de informações obtidas pela investigação policial;
4. Preparação da estrutura de retaguarda;
5. Definição da estratégia de instalação a ser utilizada;
6. Análise dos dados coletados;
7. Desativação da *FID*.

O relacionamento entre estas etapas está ilustrado na Figura 4.1.

4.1 Confirmação da necessidade e aplicabilidade da solução

Em situações normais, uma investigação policial é conduzida por uma equipe especialmente montada para tal. São estas pessoas que, ao analisar interceptações telefônicas e telemáticas autorizadas judicialmente, realizar vigilâncias e levantar informações com terceiros, possuem o maior conhecimento acerca dos alvos da investigação.

Existem aplicações, algumas inclusive desenvolvidas internamente, que auxiliam nas interceptações. Elas proveem:

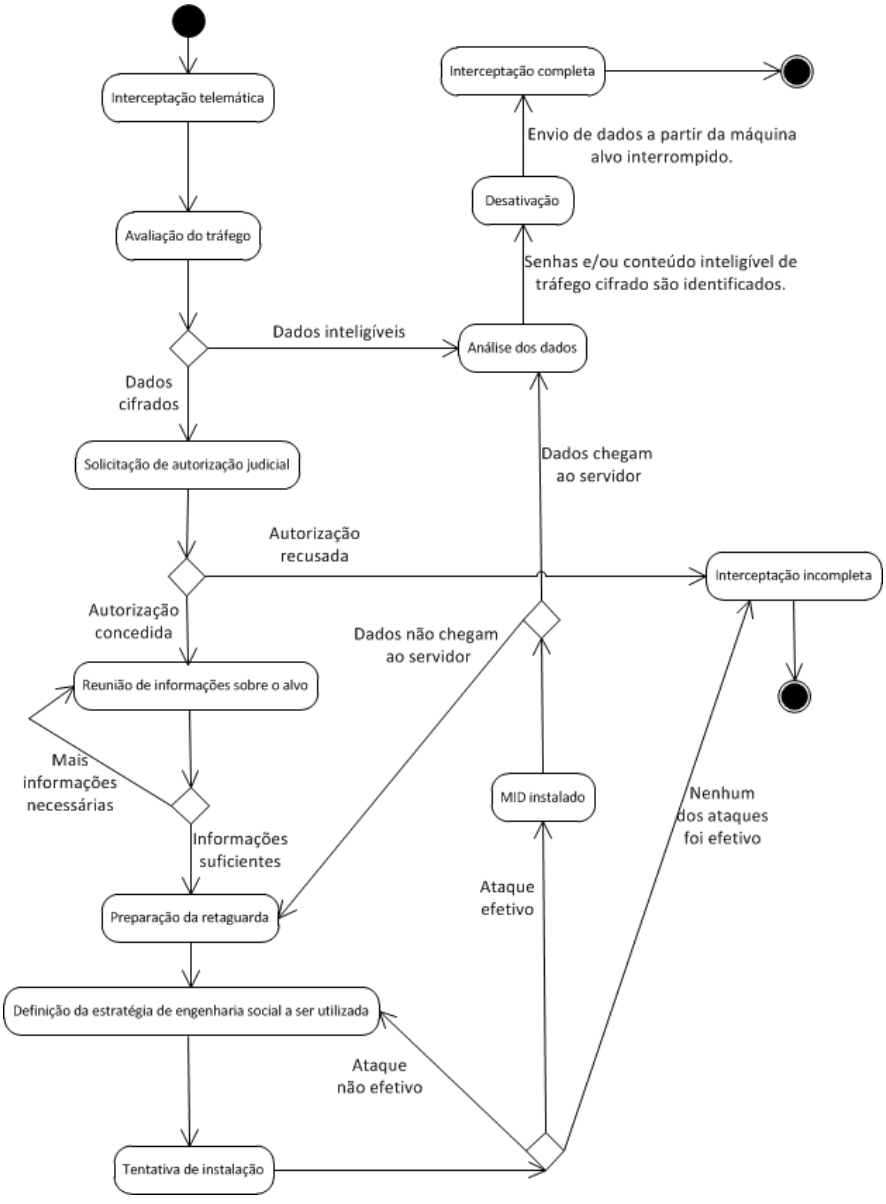

Figura 4.1: Metodologia para interceptação de dados cifrados.

- Uma estrutura para receber os dados enviados pelas operadoras de telefonia e de internet;
- Interpretação dos cabeçalhos dos pacotes *raw*[83] recebidos, agrupando-os caso pertençam à mesma requisição;
- Remontagem dos pacotes, com o objetivo de disponibilizar seu conteúdo tal como o alvo interceptado o visualiza (por exemplo, uma mensagem de email, ou uma página da *WEB*).

Algumas vezes, tanto a solução interna, quanto uma solução comercial utilizada para o mesmo fim, falham na interpretação do conteúdo dos pacotes. Um dos motivos mais corriqueiros para isto é a tentativa de interpretação de tráfego telemático criptografado.

Por exemplo, uma bate-papo *online* entre dois alvos investigados, cujo conteúdo normalmente é disponibilizado pela ferramenta de interceptação sem maiores problemas, pode tornar-se ininteligível se for utilizado algum aplicativo de criptografia. Este é um caso em que se justifica o emprego da *FID*.

Assim, para confirmar a necessidade e aplicabilidade da solução, deve ser preenchido um "formulário de requisição", contendo:

1. Identificação da investigação (nome ou número);
2. Nome do solicitante, cargo, local, data e assinatura;
3. Prováveis aplicações, endereços IP e número de porta pelos quais estão trafegando dados cifrados;
4. Anexo em CD, contendo amostragem do tráfego interceptado.

4.2 Solicitação de autorização judicial para instalação remota da solução

Após a confirmação da real necessidade de empregar a *FID* na investigação, o próximo passo é solicitar autorização judicial para tal. Esta solicitação deve conter um documento técnico informando:

[83] Nome genérico dado a arquivos contendo dados brutos.

1. Cada alvo para o qual se deseja enviar a *FID*;
2. As possibilidades de instalação que serão tentadas;
3. Breve explicação sobre o funcionamento da *FID*;
4. Quais dados serão capturados, e como será feito o controle de acesso aos mesmos.

Uma vez autorizada a instalação e a captura de dados, o próximo passo consiste em reunir o maior número de informações úteis possível.

4.3 Reunião e organização de informações obtidas pela investigação policial

O fato de existir uma investigação policial em andamento possibilita uma grande reunião de informações sobre determinada pessoa ou caso. Elas incluem:

- Nomes dos alvos e de amigos ou de parentes;
- Lugares frequentados pelos alvos;
- Número de identidade, CPF, placa de veículos;
- Endereços de email e IPs utilizados;
- Informações pessoais, trocadas por e-mails ou bate papos *online*;
- Aplicativos utilizados;
- Interesses pessoais (por meio das páginas acessadas).

Para que todas estas informações sejam aproveitadas ao máximo, é necessário catalogá-las, de modo a descobrir correspondências e obter uma visão geral das mesmas. Alguns aplicativos foram desenvolvidos para atender este propósito. (HADNAGY; WILSON, 2010) sugere o *Maltego*, pois, além de catalogar, permite que dezenas de buscas por domínios, endereços IP e pessoas na internet sejam realizadas de forma automatizada por uma mesma ferramenta. Na Figura 4.2, uma tela do *Maltego*.

A partir dessa primeira organização, ficará mais fácil buscar na internet por informações derivadas dos alvos:

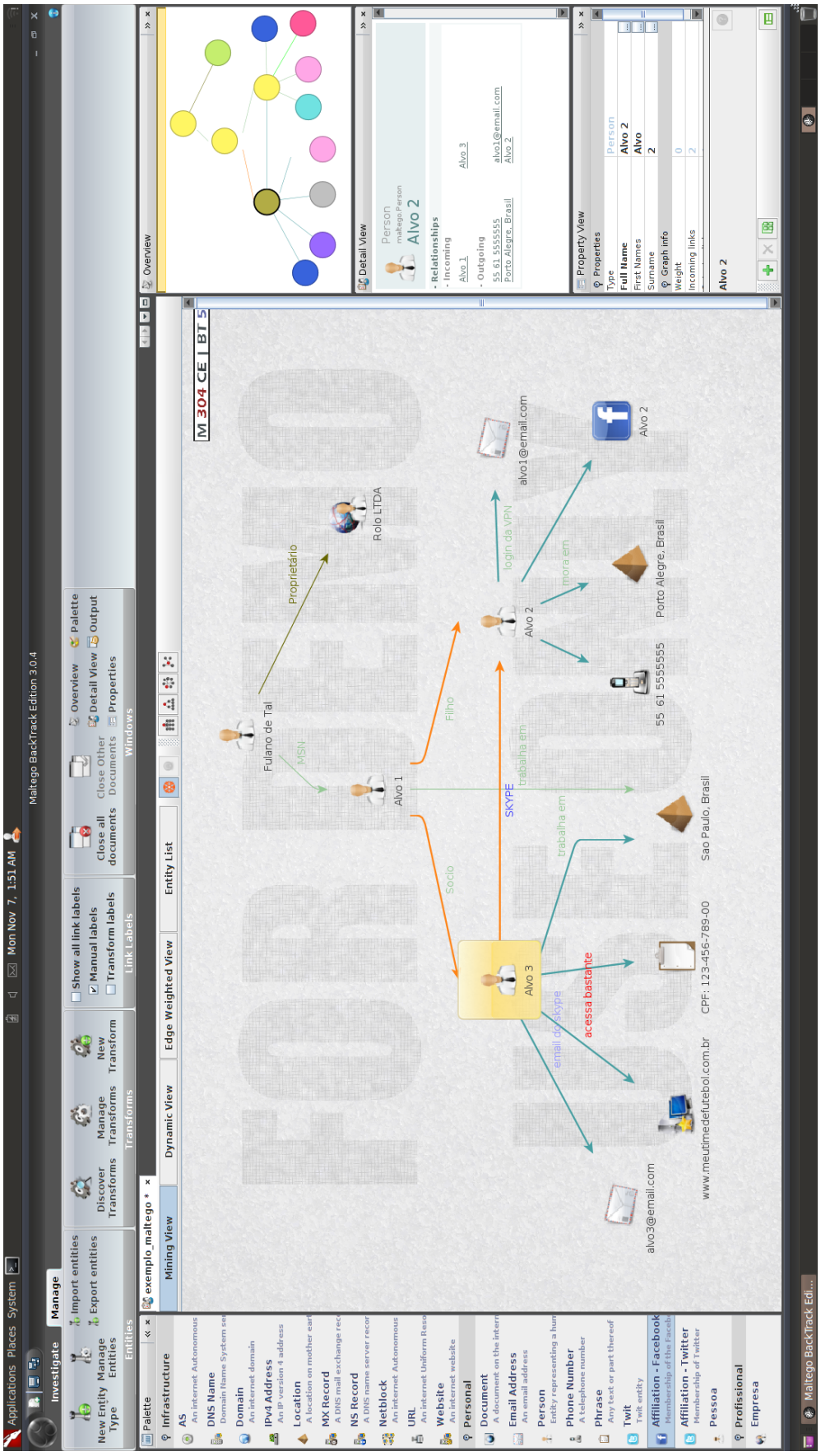

Figura 4.2: Exemplo de organização de informações por meio da aplicação *Maltego*.

- Escaneamento de portas [84] - nos endereços IPs: o objetivo é descobrir todas as aplicações que estão em execução;
- Pesquisa em sítios de busca como o *Google*: pode ajudar a descobrir informações, por exemplo, sobre uma empresa, tais como nome de funcionários e de empresas parceiras, endereços de *email*, entre outros;
- Pesquisa em redes sociais e *blogs*, como *Facebook* e *Twitter*, com o intuito de descobrir os nomes de amigos ou parentes, interesses pessoais, locais frequentados, e outras informações;
- Bases de dados governamentais e outras, para descobrir dados que não são de conhecimento comum, e que por isso podem tornar a engenharia social mais verossímil.

4.4 Preparação da estrutura de retaguarda

Para receber as informações capturadas do computador alvo, é necessário configurar um servidor dedicado para tal. Os seguintes requisitos são necessários:

- Possuir um endereço *IP* válido na internet, para que o computador alvo consiga acessar o servidor. Isto pode ser feito de suas maneiras:
    1. Adquirir um endereço *IP* fixo junto a um provedor de internet. Esta é a maneira mais direta, pois não necessita realizar consultas de nomes em servidores *DNS*[85], evitando assim ataques de *DNS spoofing*[86], os quais poderiam fazer os dados serem enviados para um servidor não autorizado a recebê-los;
    2. Conectar-se à internet por meio de um endereço *IP* dinâmico, que é a maneira mais utilizada por conexões residenciais. Nesta abordagem, é necessário, além de contratar um serviço de DNS - por exemplo, o *DynDNS*[87], configurar um *port forwarding*[88] no *modem*. Esta foi a abordagem utilizada durante os testes, pois não acarreta custos.

[84] Técnica utilizada para verificar o estado das portas de uma interface de rede.

[85] Servidor para resolução de nomes, utilizado pelos navegadores para traduzir uma URL em um endereço IP.

[86] Técnica de mascaramento, cujo objetivo é fazer uma entidade se passar por outra.

[87] Serviço *WEB* utilizado para associar URLs a uma interface de rede que utiliza endereços IP dinâmicos.

[88] Técnica utilizada para redirecionar o tráfego que chega por uma porta para outro dispositivo dentro da rede.

Também é recomendado possuir um endereço *IP* fixo adicional, contratado de um segundo provedor, de modo a evitar a interrupção na transmissão dos dados caso venha a acontecer algum problema com o primeiro provedor.

- Garantir a discrição da localização do servidor de retaguarda e dos dados enviados para ele: Por se tratar de uma investigação policial, este é um fator extremamente importante. Por isso, é recomendado que o servidor que receberá os dados cifrados possua um endereço *IP* pertencente a uma faixa reservada a um país estrangeiro. Essa discrição, somada à ininteligibilidade dos dados enviados, visa minimizar a suspeita que dados estão sendo interceptados por uma investigação criminal, caso a *FID* venha a ser descoberta.

Ainda como parte da estrutura, é recomendada a configuração de um segundo computador, cuja função é obter os pacotes criptografados no servidor, decifrá-los e disponibilizar seu conteúdo, controlando o acesso às informações por meio de definição de credenciais usuário e senha para cada investigador.

4.5 Definição da estratégia de instalação a ser utilizada

Com toda a informação reunida e analisada, e a estrutura de retaguarda pronta, é hora de definir o vetor de instalação que será utilizado.

Apesar de não haverem sido implementadas, as seguintes abordagens são sugeridas para a instalação da *FID* na máquina alvo:

- Envio de *email* com anexo no qual está embutido a *FID*. O fato de informações como interesses pessoais e nomes e endereços de *email* de amigos próximos ou colegas de trabalho serem conhecidas torna mais factível a aceitação do *email* como confiável e a consequente abertura de seu anexo pelo investigado;
- Clonagem de um sítio de interesse do alvo: Segundo (HADNAGY; WILSON, 2010), a ferramenta *SET*[89] pode ser configurada para realizar esta tarefa automaticamente, além de implementar outros ataques, como envio de *email* contendo código malicioso. Assim, bastaria que o alvo clicasse em um *link* ou abrisse um arquivo para que a *FID* fosse instalada;

[89] Conjunto de *softwares* utilizados para implementar ataques de engenharia social.

- Instalação física: caso se tenha acesso ao computador alvo, bastaria a inserção de um *pendrive*, pré configurado para tal tarefa, para instalar completamente a *FID*.

A abertura do arquivo ou o clique do *link* citados executaria um pequeno aplicativo, de tamanho reduzido e cujas funções seriam:

1. Baixar e armazenar os arquivos principais da *FID* (`filtec.sys`,`serv1.exe` e `sendproc.exe`) a partir de um servidor remoto;
2. Após determinado período, lançar uma janela para o usuário solicitando autorização para executar uma aplicação acreditada (por exemplo, uma atualização do *Java*), porém instalando a *FID*.

A decisão entre as abordagens apresentadas ou outras possíveis baseia-se na avaliação dos hábitos de cada alvo ao utilizar a internet: se costuma receber *emails* ou clicar em janelas *pop-up* com frequência, por exemplo.

Caso a abordagem escolhida não funcione, outras técnicas de engenharia social ainda podem ser tentadas. A definição destas técnicas, a implementação das mesmas, os resultados obtidos e a interpretação destes resultados são tópicos sugeridos para trabalho futuro, que consiste em uma parte fundamental para o sucesso da metodologia desenvolvida.

4.6 Análise dos dados capturados

Após o início da chegada dos pacotes, é necessário disponibilizar os dados contidos para investigadores autorizados. Assim, conforme citado na seção 4.4, um computador específico é o responsável tanto pela decifração dos pacotes quanto pelo provimento de uma interface gráfica para análise, à qual só é fornecido acesso por meio de credenciais usuário / senha.

Então, informações como senhas para bases de dados, para outros servidores ou mesmo para partições e arquivos cifrados podem ser procuradas, bastando para isso identificar títulos de janelas ou trechos de texto compatíveis. Ainda, o selo de data presente em cada

pacote recebido pode servir para relacionar as sessões cifradas de bate papo interceptadas com os dados capturados pela *FID*.

O fim da etapa da análise dos dados está condicionado a duas situações: não renovação do mandado de interceptação, ou finalização da investigação, com o possível posterior cumprimento de mandado de busca e apreensão. Para ambas, é necessário desativar a *FID*.

4.7 Desativação da *FID*

O objetivo desta etapa é garantir o fiel cumprimento do mandado de interceptação telemática, a fim de evitar posterior contestação das provas obtidas com o argumento de que a *FID* enviou dados em período não abrangido pelo mandado.

Assim, ao ser desativada, a *FID* cessa o envio de informações do usuário para um servidor remoto. Optou-se por apenas desativá-la ao invés de removê-la completamente pelo seguinte motivo: caso a renovação do mandado seja posterior à data limite do último mandado, ou, em outras palavras, seja necessário interromper a interceptação, a reativação da mesma dependerá apenas do envio de um comando específico para a *FID*. Assim, não será necessária a aplicação de uma nova técnica de engenharia social contra o alvo, o que poderia diminuir as chances de sucesso nesta "nova"interceptação.

Esta etapa, fundamental na metodologia, é objeto de aprimoramento futuro da ferramenta.

A metodologia tratada neste capítulo foi definida baseada na realidade atual observada quando uma interceptação telemática é autorizada judicialmente e conduzida por um órgão de investigação oficial. A contínua utilização desta metodologia em investigações e a análise das dificuldades encontradas e dos resultados obtidos poderá agregar novas etapas ou incrementar as existentes, otimizando a metodologia.

No próximo capítulo, serão explicados os testes realizados para validar a solução e as condições em que ocorreram. Buscou-se simular as condições mais comuns encontradas pelas equipes de investigação em interceptações telemáticas criptografadas, a fim de confirmar a obtenção dos dados. Também foram executados testes de módulo, ou seja, que verificassem as condições de funcionamento da *FID* junto ao sistema em que está instalada.

5 VALIDAÇÃO E TESTES

O objetivo deste capítulo é verificar a efetividade da solução desenvolvida e monitorar o seu funcionamento, a fim de descobrir supostos erros de execução e avaliar os resultados obtidos.

Primeiramente foi preparado um ambiente de modo a simular uma situação real. Para isso, são necessários 2 computadores, um fazendo o papel da máquina alvo, na qual será instalado a *FID*, e outro fazendo o papel do servidor, que receberá os dados enviados.

5.1 Montagem do ambiente

5.1.1 Alvo

O computador que atuará como "alvo"possui a configuração descrita na tabela 5.1:

Tabela 5.1: Configuração da máquina alvo

Natureza	Computador portátil da marca Lenovo, modelo T60
Processador	*Intel Core 2* T7600 2.33 GHz
Memória *RAM*	2 GB
Placa de rede	*Intel* PRO 10/100 em conexão *ethernet*
Sistema Operacional	*Windows* 7 Professional Service Pack 1, 32 *bits*
Aplicativo antivírus	AVG Internet Security 2012

5.1.2 Servidor

O computador que atuará como "servidor"possui a configuração descrita na tabela 5.2:

Tabela 5.2: Configuração do servidor

Natureza	Máquina virtual rodando no aplicativo *VirtualBox*
Processador	*Intel Core* i7 2630 2.0 GHz (virtual)
Memória *RAM*	512 MB
Placa de rede	Virtual em modo *bridge* com o *host* em conexão *ethernet*
Sistema Operacional	*Windows XP Professional Service Pack 3*, 32 *bits*;
Servidor *WEB*	*Apache* versão 2.2.17 e *PHP* versão 5.2.17
Servidor *DNS*	Serviço TZO

```
C:\Windows\System32: serv1.exe -init
C:\Windows\System32: net start serv1.exe
```

Código 5.1: Comandos para iniciar a *FID*.

5.2 Instalação da *FID*

A solução, no estágio em que se encontra, ainda não conta com um vetor de instalação, ou seja, um artifício de engenharia social que leve o usuário alvo a clicar em um botão, autorizando o *download* e a posterior instalação dos arquivos com privilégios de administrador, de maneira discreta. O desenvolvimento deste artifício é um dos projetos futuros sugeridos.

Por isso, para fins de teste, a instalação da solução ocorreu de maneira manual: em uma janela do *prompt* de comando do *Windows*, aberta com privilégios de administrador, foram digitadas as linhas de comando ilustradas em 5.1:

O primeiro comando instala e inicia o *driver* da solução (filtec), e também instala o serviço `serv1`. O segundo comando inicia o serviço `serv1.exe`. Este serviço criará o processo `sendproc.exe`, responsável por capturar o título das janelas ativas.

A partir desse ponto, a solução está completamente funcional.

5.2.1 Tráfego gerado pela *FDI*

O tráfego gerado pela solução corresponde a conexões *TCP* estabelecidas entre a máquina alvo e o servidor. Para efeito de comparação, a Figura 5.1 contém uma conexão TCP ordinária capturada na mesma interface, e a Figura 5.2 contém uma conexão TCP criada pela *FDI*.

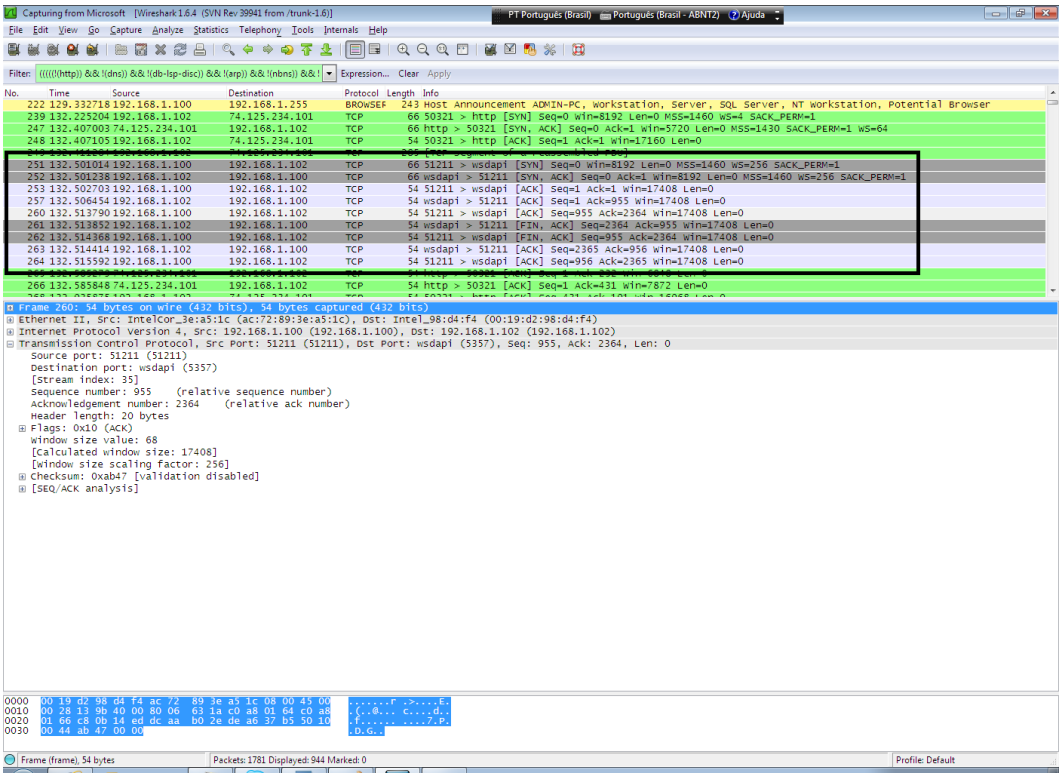

Figura 5.1: Tráfego TCP comum.

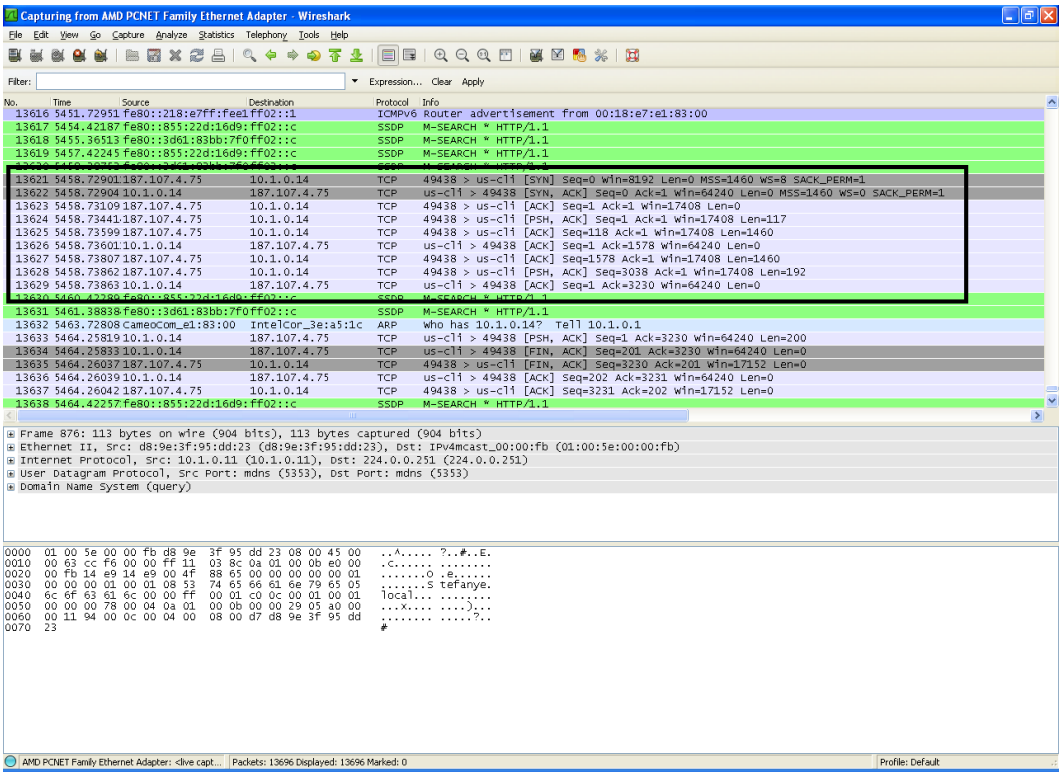

Figura 5.2: Tráfego TCP gerado pelo envio de um pacote contendo informações capturadas.

Cada linha das figuras corresponde a um pacote. As diferenças mais notáveis entre as duas capturas restringem-se à quantidade de *bytes* transferidos, que podem ser conferidos no atributo `Len` de cada pacote, e à utilização da *flag PSH*, ou *PUSH*, que obriga a transferência imediata dos dados à aplicação destinatária tão logo estes cheguem à interface de rede, não importando a quantidade *bytes* recebidos.

5.2.2 Recebimento e decifração dos dados

Todos criptogramas que chegam ao servidor *WEB* são extraídos dos pacotes *HTTP POST* e escritos em um arquivo de texto.

O primeiro conjunto de dados enviado pelo alvo, ainda durante a inicialização do *driver*, contém o criptograma da chave simétrica cifrado com a chave pública. Ao chegar ao servidor, ele é identificado por meio da marcação `init=`. Na Figura 5.3, o conteúdo do *payload* dentro do fluxo *TCP*, que reflete este criptograma.

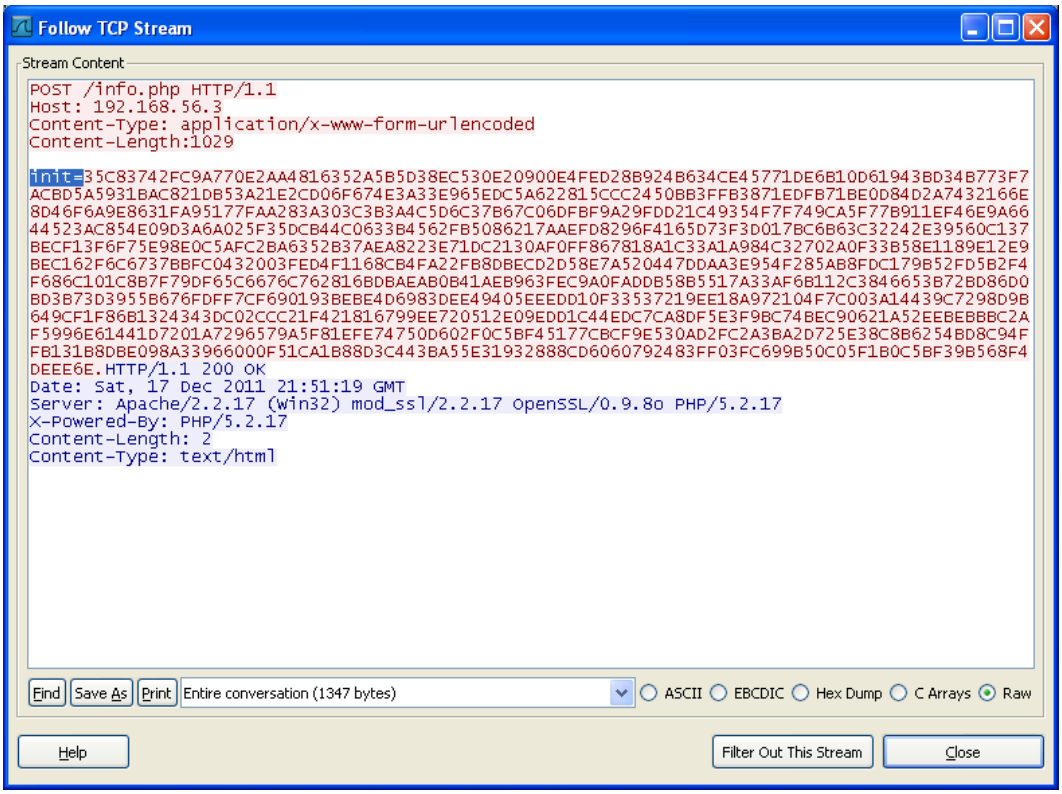

Figura 5.3: Criptograma contendo a chave simétrica, identificada pela marcação "init=".

Este criptograma é decifrado por um programa auxiliar, ao qual é informado, além do criptograma, a chave privada. O programa gera como saída um arquivo de texto

contendo um selo de data e hora seguido da chave simétrica. Uma posterior decifração de uma nova chave recebida não sobrescreve a antiga, mantendo-se assim um histórico de chaves utilizadas. Então, é necessário informar a chave simétrica ao *script PHP*, que terá por função decifrar os dados. Finalmente, os dados decifrados são escritos em um arquivo que será disponibilizado aos investigadores.

Logo após a chegada da chave simétrica, caso haja interação do usuário com o sistema, pacotes que contêm os dados capturados (teclas digitadas e título das janelas que foram ativadas) e os dados de controle (data, hora e número identificador do pacote) começam a chegar ao servidor. Na Figura 5.4, o conteúdo do *payload* dentro do fluxo *TCP*, que reflete o criptograma que contêm os dados citados.

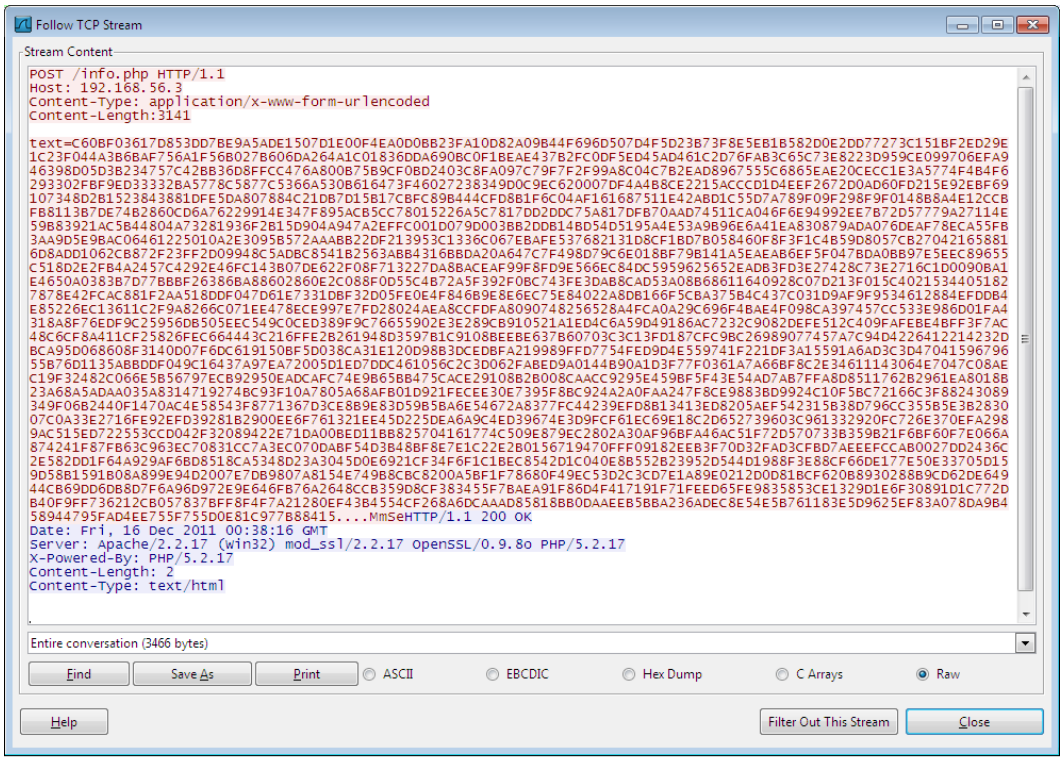

Figura 5.4: Criptograma contendo dados capturados, identificado pela marcação "text=".

Este *payload* contém os dados capturados cifrados com a chave simétrica. O criptograma foi convertido da base binária para base hexadecimal, a fim de ser transmitido pela rede.

5.2.3 Resultado da captura

Em uma situação real, o servidor que primeiramente receberá os dados não será um computador do órgão de investigação, mas sim um contratado de uma empresa especializada em alugar servidores *WEB*. O motivo desta precaução é evitar a identificação do destino final dos pacotes, caso os mesmos sejam rastreados.

Assim, neste servidor alugado, estará em execução permanente um *script PHP* cujo objetivo é receber os pacotes e escrever o conteúdo dos mesmos em arquivos de texto, além de reportar ao computador alvo o sucesso ou a falha no recebimento dos pacotes. Estes arquivos de texto serão armazenados em uma pasta específica do servidor, esta sim que será consultada periodicamente por uma aplicação rodando no servidor do órgão de investigação. O objetivo desta aplicação é verificar por novos arquivos. Caso positivo, os mesmos são baixados e seu conteúdo é decifrado e escrito em outros arquivos que serão disponibilizados à equipe de investigação.

Porém, como um dos objetivos dos testes é verificar se o envio dos pacotes produz uma diferença de performance perceptível ao usuário do computador alvo, e não aos usuários do servidor alugado ou do servidor do órgão de investigação, os dados estão sendo recebidos, decifrados e escritos no mesmo arquivo de *log*. A Figura 5.5 ilustra o conteúdo deste arquivo, em que o texto entre os marcadores `<P>` e `</P>` reflete o título de todas as janelas que foram acessadas em sequência, e o texto sem marcadores reflete o texto digitado pelo usuário.

5.3 Cenários para testes

Os cenários de testes foram definidos de acordo com as dificuldades encontradas pelas equipes de investigação em interpretar comunicação cifrada trafegando entre alvos de interceptações telemáticas autorizadas judicialmente. Apesar de não ser possível prever de antemão todos as situações que serão enfrentadas, buscou-se abranger tanto as situações para quais a solução foi desenvolvida quanto as que porventura pudessem interferir no funcionamento normal da mesma.

É importante frisar que a solução não foi avaliada e comparada em relação a uma solução comercial por impossibilidade de aquisição da mesma. As soluções que aten-

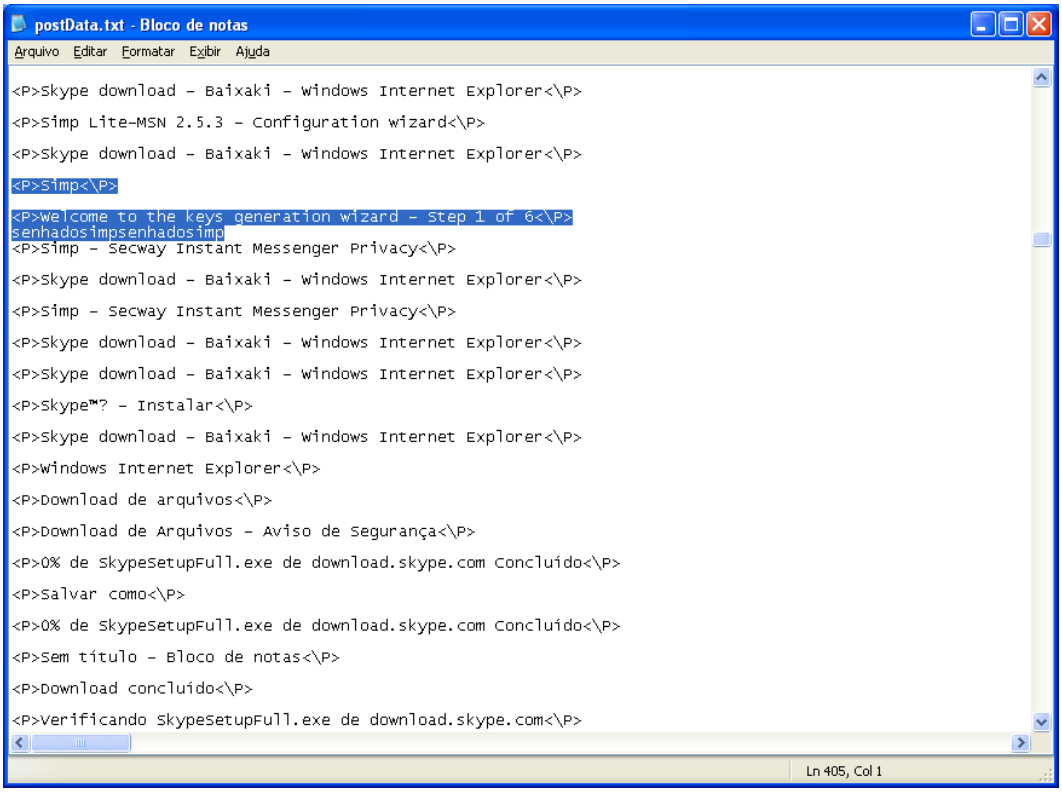

Figura 5.5: Trecho do arquivo contendo os dados decifrados. Em destaque, a senha definida na instalação do Secway Simp, que foi capturada.

dem órgãos governamentais, ou seja, que foram desenvolvidas com o mesmo objetivo da apresentada neste projeto, como o *FinFisher* (HYPPONEN, 2011), chegam a custar centenas de milhares de reais.

5.3.1 Principais situações para as quais a solução foi desenvolvida

Para avaliar o funcionamento da solução em situações de bate-papo cifrado ou de envio e recebimento de emails utilizando criptografia, foram definidos os critérios constantes na Tabela 5.3.

Para os testes, foram escolhidos os aplicativos de bate papo cifrados *MSN Live Messenger*[90] com *Secway Simp*[91] e *Skype*[92], pois são os mais constantemente encontrados pelas equipes de investigação em interceptações telemáticas.

[90] http://explore.live.com/messenger

[91] Aplicativo utilizado para cifrar o conteúdo de conversas entre clientes de bate-papo online, como o MSN Live Messenger. http://www.secway.fr/us/products/simplite_msn/getsimp.php

[92] http://www.skype.com/intl/ptbr/getskype/

Tabela 5.3: Critérios para avaliação da captura de bate-papo

Identificador	Critério
A	Integridade dos dados: se todos os dados capturados foram recebidos
B	Correção dos dados dentro do pacote: se os dados interceptados estão dispostos de maneira equivalente aos originais
C	Totalidade dos pacotes recebidos: se todos os pacotes enviados foram recebidos
D	Manutenção da performance: se a performance manteve-se semelhante em comparação à ausência da solução
E	Resultado diferenciado: se o resultado obtido pela solução poderia ser obtido também por uma interceptação telemática tradicional

1. **MSN Live Messenger com Secway Simp:** No caso da utilização apenas do *MSN Live Messenger*, tanto os dados de controle quanto o texto digitado são transmitidos completamente às claras, conforme se nota na Figura 5.6.

   A utilização do aplicativo *Secway Simp* em conjunto com o *MSN Live Messenger*, desde que ocorra em ambas partes da conversa, faz com que o texto digitado pelas partes durante o bate-papo seja cifrado, tornando-o ininteligível. Na Figura 5.7 um exemplo de um fluxo *TCP* contendo os dados de controle e o criptograma resultante.

   Nota-se o texto cifrado e um cabeçalho adicionado pelo *Secway Simp* logo acima dele.

   Já a Figura 5.8 apresenta o texto digitado contido no criptograma da Figura 5.7, extraído do arquivo de *log*.

   Nota-se que algumas teclas estão invertidas. Esse fenômeno foi observado sempre que a digitação ocorreu de maneira muito rápida e, apesar de não ser determinante ao entendimento do texto, é um problema a ser resolvido futuramente.

2. **Skype**: Ao analisar os pacotes de dados enviados e recebidos pelo *Skype*, notou-se que todo o seu tráfego nativo é ininteligível, conforme a Figura 5.9.

   Da mesma maneira que o *MSN Live Messenger com Secway Simp*, foi possível obter os dados digitados pela parte antes de serem cifrados, conforme a Figura 5.10.

   Ao aplicar os critérios da tabela 5.3 nestas situações de utilização de aplicativos de bate-papo, obteve-se a seguinte tabela:

- Sobre o critério A: Em ambos cenários, todos as informações digitadas foram recebidas no servidor.

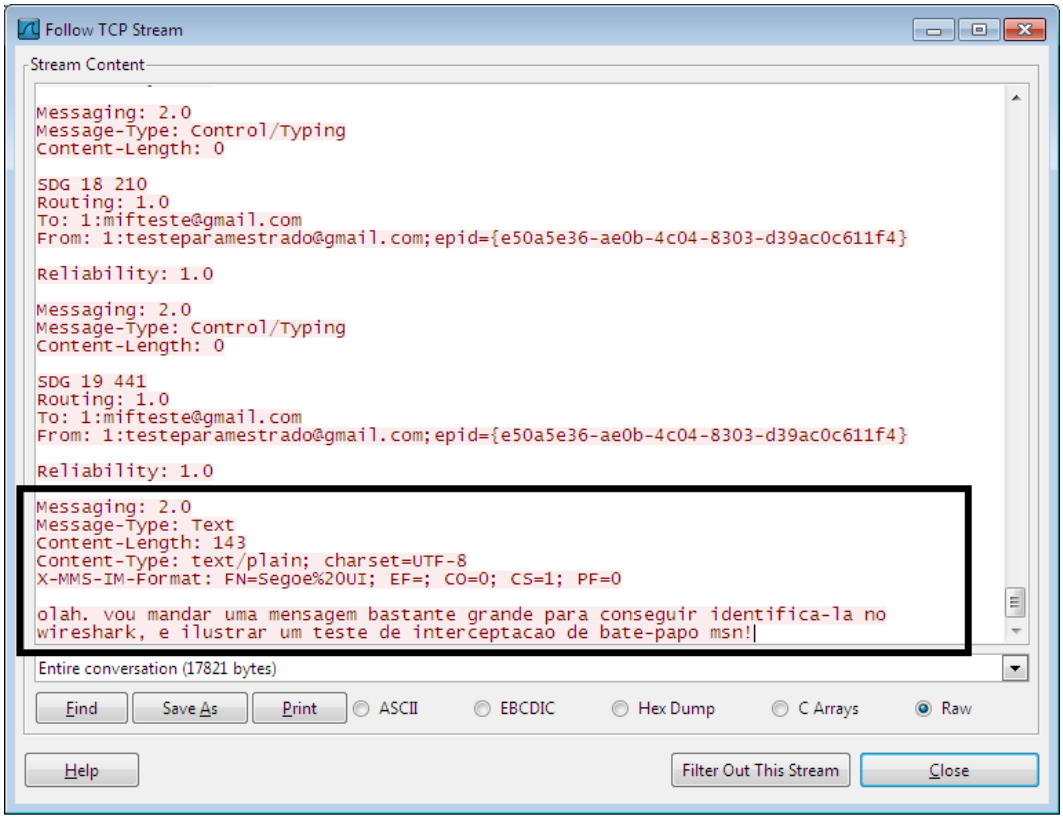

Figura 5.6: Fluxo contendo um trecho de um bate-papo do *MSN Live Messenger*.

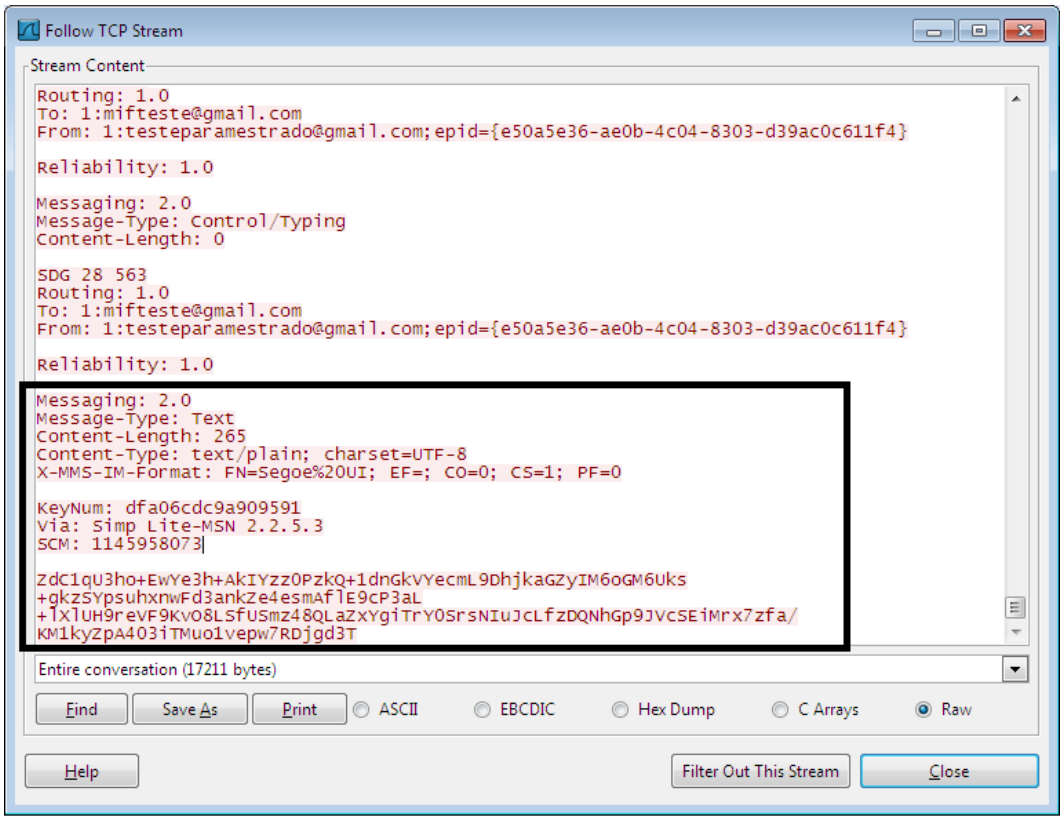

Figura 5.7: Fluxo contendo um trecho de um bate-papo do *MSN Live Messenger* utilizando o*Secway Simp*.

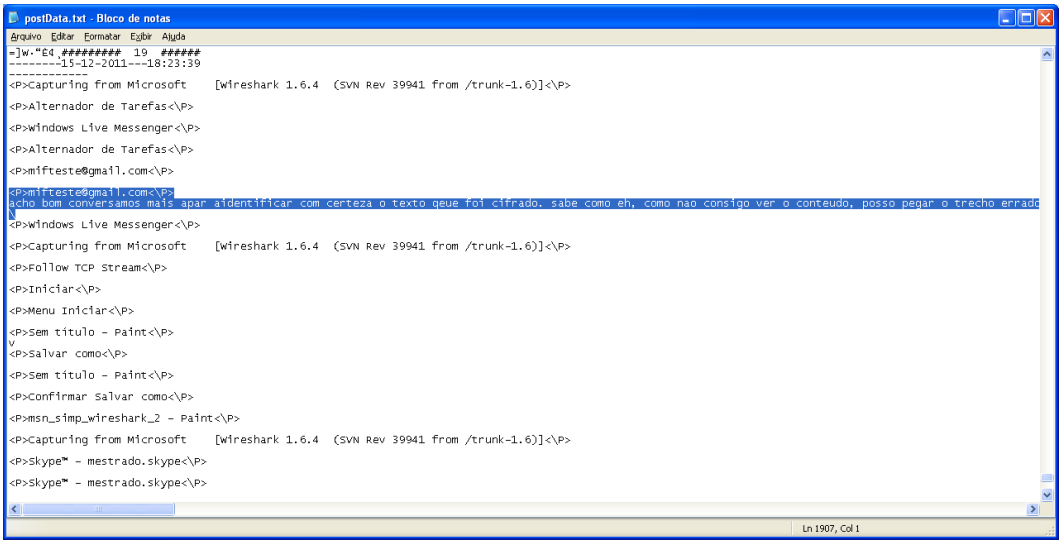

Figura 5.8: Trecho de texto capturado durante do bate-papo do *MSN Live Messenger* cifrado com *Secway Simp*.

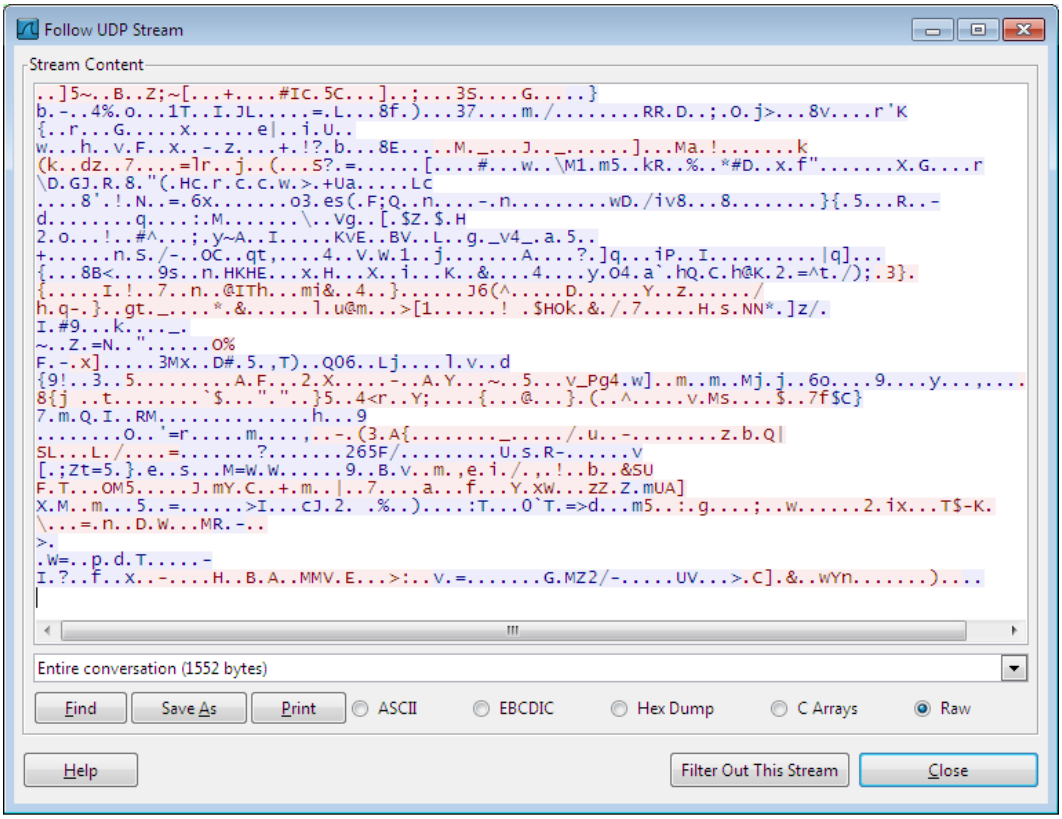

Figura 5.9: Fluxo *TCP* contendo trecho de bate-papo do *Skype*.

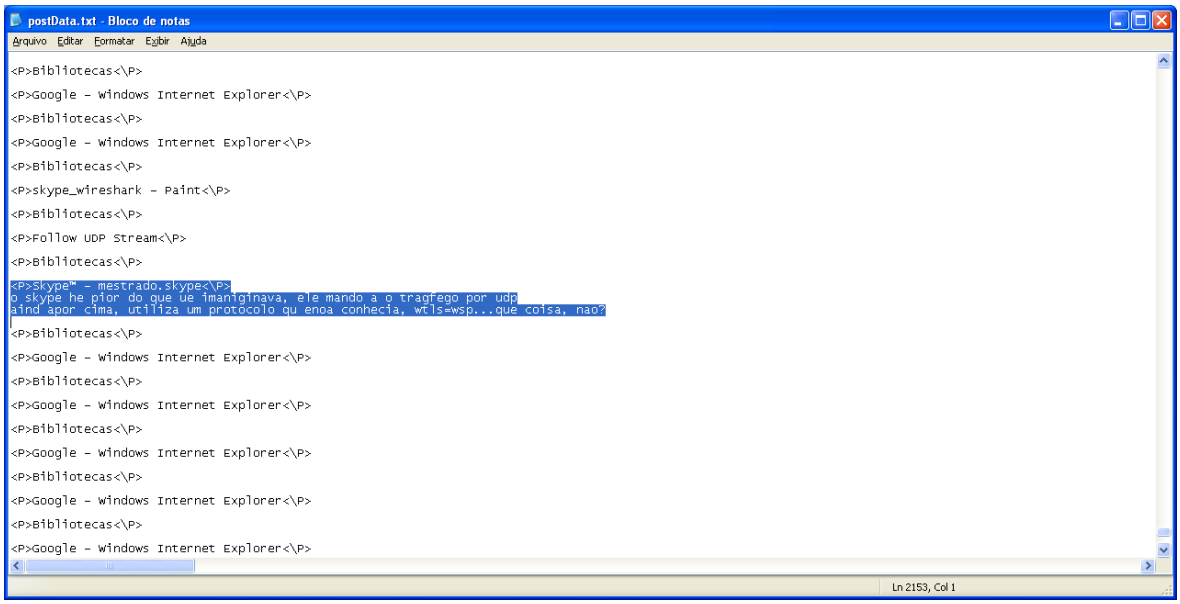

Figura 5.10: Trecho de texto capturado durante do bate-papo do *Skype*

Identificador	MSN com Secway Simp	Skype
A	Sim	Sim
B	Algumas letras trocadas	Algumas letras trocadas
C	Sim	Sim
D	Sim	Sim
E	Não	Não

Tabela 5.4: Availação da captura de bate-papo

- Sobre o critério B: Em ambos cenários, algumas letras foram trocadas em virtude de digitação rápida.
- Sobre o critério C: Desde o envio da chave até o final dos testes foram enviados no total 23 pacotes com aproximadamente 1500 *bytes* de conteúdo em cada. Não houve perda de pacotes recebidos.
- Sobre o critério D: Em ambos cenários, não foi gerada alteração de tempo de resposta da aplicação perceptível ao usuário.
- Sobre o critério E: Em ambos cenários, os dados obtidos pela solução estariam ininteligíveis se obtidos a partir de uma interceptação telemática tradicional.

Dito isto, apesar do problema em relação à troca de letras, a solução mostrou-se eficaz. Um ponto a ser observado é que as capturas são referentes à apenas uma das partes da conversa. Para capturar a conversa inteira, é necessário instalar a *FDI* no computador da outra parte também.

5.3.2 Envio de mensagens eletrônicas (*email*)

Para realizar este teste, foi criada uma conta para testes no serviço *Gmail*.

Ao acessar a conta, digitando usuário e senha, o fluxo *TCP* capturado pelo *Wireshark* mostrou-se conforme na Figura 5.11:

Esse fluxo reflete o conteúdo de um *handshake TLS*, que é um protocolo criptográfico para comunicação segura na internet. Assim, o nome de usuário e senha trafegam cifrados.

O fluxo *TCP* referente ao posterior envio de uma mensagem eletrônica apresentou conteúdo semelhante, ou seja, cifrado.

Ao analisar o arquivo de resultados, tanto os dados de autenticação quanto o conteúdo da mensagem eletrônica enviada foram capturados, conforme as Figuras 5.12 e 5.13.

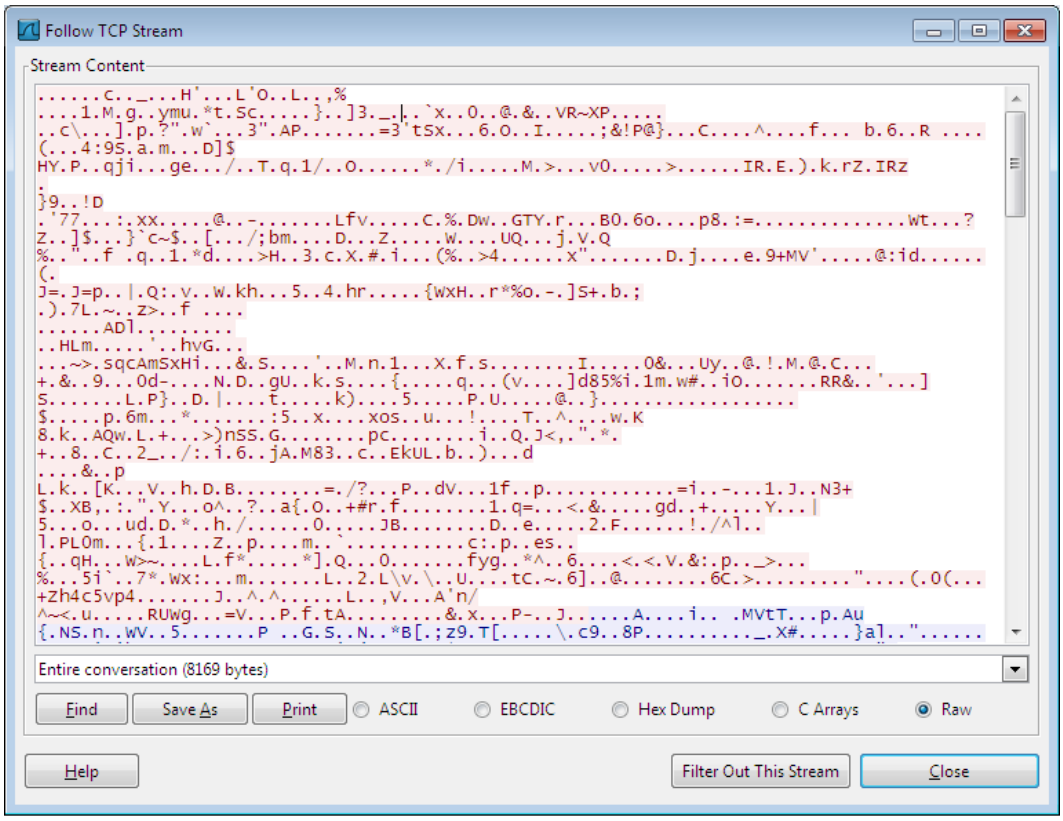

Figura 5.11: Fluxo *TCP* de uma autenticação ao serviço Gmail

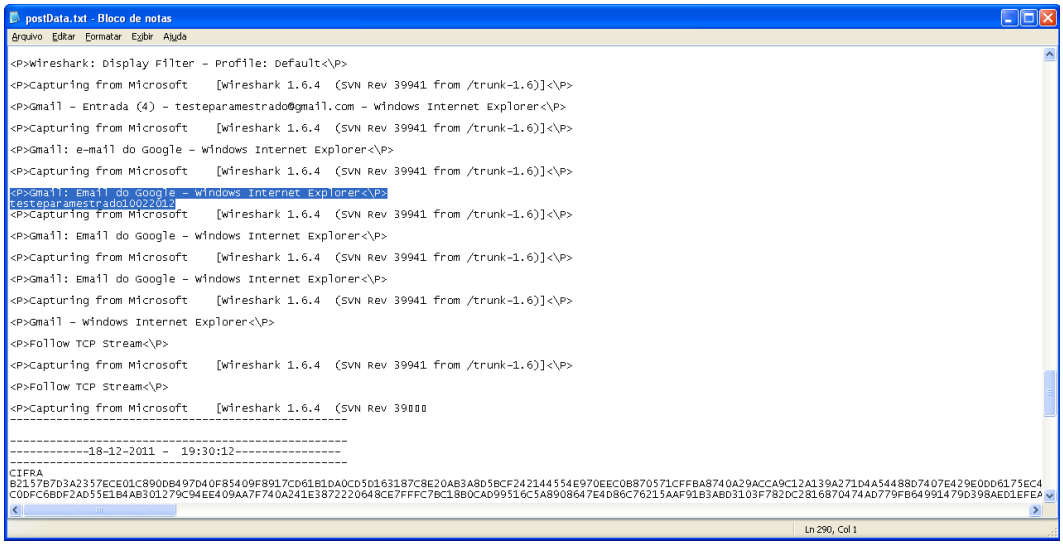

Figura 5.12: Usuário e senha de uma conta do *Gmail* capturados.

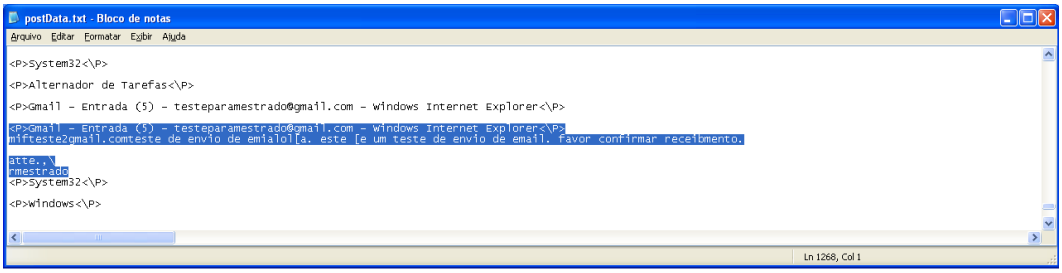

Figura 5.13: Captura do texto digitado no *email*.

Ao aplicar os critérios estabelecidos na tabelatab:criterios em situações de envio de mensagens eletrônicas, obteve-se a seguinte Tabela 5.5:

Tabela 5.5: Availação da captura de email

Identificador	Gmail
A	Sim
B	Não
C	Sim
D	Sim
E	Não

- Sobre o critério A: Todos os dados digitados na mensagem eletrônica chegaram ao servidor.
- Sobre o critério B: Notou-se que o mapeamento das teclas apresenta algumas discrepâncias: por exemplo, na Figura 5.13, o "@"consta como "2", e o acento agudo consta como "]". Ainda, para uma melhor visualização, faz-se necessário implementar a captura do clique do *mouse* e associá-lo ao mesmo código da tecla "Enter". Assim, neste exemplo, o endereço de email, o assunto e o corpo da mensagem estariam dispostos em linhas consecutivas.
- Sobre o critério C: Desde o envio da chave até o final dos testes Foram enviados no total 18 pacotes com aproximadamente 1500 *bytes* de conteúdo em cada. Não houve perda de pacotes recebidos.
- Sobre o critério D: Não foi gerada alteração de tempo de resposta na digitação do *email* que fosse perceptível ao usuário.
- Sobre o critério E: Os dados digitados na mensagem eletrônica enviada pelo serviço *Gmail*, que cifra as mensagens enviadas, estariam ininteligíveis se capturados por uma interceptação telemática tradicional.

Novamente, tal como no cenário de aplicativos de bate-papo, a falha no critério B não invalida os resultados da solução.

5.3.3 Instalação e execução de aplicativos antivírus e anti-rootkits

Antes da instalação da *FID*, foi instalado o aplicativo *AVG Antivirus* 2012, em sua versão completa e grátis para avaliação por 30 dias. Este antivírus foi escolhido para monitorar constantemente o sistema durante os testes por dois motivos:

1. De acordo com uma pesquisa do sítio PC Magazine, a versão 2012 do *AVG Antivirus* foi eleita uma das melhores soluções para bloquear e remover vírus e *rootkits*(RUBENKING, 2011).
2. Na página oficial do aplicativo AVG Antivírus, é informado que esta versão realiza análise heurística de *malwares*. Esta é uma análise comportamental, útil para detectar novos *malwares* cujas assinaturas ainda não são conhecidas.

O aplicativo *AVG Antivirus* 2012 foi configurado para o modo de "proteção contínua"durante os testes, e não identificou ameaças em nenhum momento.

Então, foram baixados e executados alguns aplicativos anti-*rootkits*, com o intuito de executar buscas mais direcionadas. Um por vez, todos fizeram uma varredura do disco, à procura de *rootkits*. Os resultados estão na tabela 5.6.

A *FDI* não provocou o disparo de alerta em nenhuma destas ferramentas, talvez por não esconder nenhum de seus objetos. A falta de assinatura de seu *driver* e a criação de um dispositivo anexado a outro (no caso, ao dispositivo de teclado) foram reportadas pelas ferramentas TDSSKiller e GMER, respectivamente. Ambas situações não são incomuns em sistemas *Windows* de 32 *bits*. O mesmo não se pode dizer de sistemas Windows de 64 bits a partir da versão 7, que só permite a instalação de um *driver* se este for assinado. Esta obrigação, entretanto, pode ser desabilitada.

Assim, em sistemas *Windows* de 32 *bits*, o rastro deixado pela solução não difere muito do rastro deixado por outros aplicativos correntes instalados no computador.

Tabela 5.6: Execução de aplicativos anti-*rootkit*

Aplicativo	Alerta de Rootkit?	Observações
AVG 2012 (módulo anti-*rootkit*)	Não	-
Sophos 1.5	Não	Busca baseada em objetos escondidos
Kaspersky TDSSKiller 2.6.23.0	Não	Na busca por *drivers* não assinados (desabilitada por padrão), detectou o *driver* utilizado pela *FDI*.
GMER 1.0.15.15641	Não	Na aba "Rootkit/-Malware"listou o dispositivo da solução juntamente com outros dispositivos utilizados pelo AVG.
Avast ASWmbr 0.9.8.986	Não	-

Assim, os testes foram considerados satisfatórios, pois foi possível interceptar dados antes de serem cifrados e enviados pela Internet, sem acarretar uma diferença perceptível de tempo de resposta da aplicação ao usuário, e também sem levantar alertas de *malware* por aplicativos anti-*rootkit*.

6 CONCLUSÕES

Este trabalho apresentou uma metodologia para interceptação de dados de maneira remota, a ser utilizada em investigações policiais. A necessidade desta metodologia teve origem na impossibilidade enfrentada por investigadores em interpretar dados que navegam cifrados pela internet, em interceptações telemáticas autorizadas judicialmente.

Por motivos expostos na seção 2.1, foi verificado que o desenvolvimento de solução própria que atenda aos objetivos listados na seção 1.2 é a alternativa mais recomendada.

Assim, foi implementado um programa cujo objetivo é, após instalado no sistema alvo, interceptar e enviar dados inseridos pelo investigado para um servidor remoto, de maneira segura.

As seções 3.1, 3.2 e 3.3 detalham as funcionalidades implementadas na solução, quais sejam:

1. Captura de teclas pressionadas;
2. Captura da hora do sistema;
3. Captura dos títulos das janelas de aplicativos ativas;
4. Cifragem de todos os dados capturados;
5. Envio dos criptogramas pela rede;
6. Recebimento e disponibilização dos dados no servidor.

Os testes conduzidos e apresentados no capítulo 5 mostram que o programa se mostrou eficaz, apesar de não ter sido empregado em uma situação real. Como as funcionalidades implementadas atenderam ao objetivos estipulados na seção 1.2, tanto a metodologia quanto o programa podem começar a ser utilizados em investigações, fato que proporcionará novas avaliações e consequentemente acarretará melhorias para a solução como um todo.

Por se tratar de uma área em constante evolução, o desenvolvimento de novos sistemas operacionais e a pesquisa sobre seu funcionamento, supostas falhas descobertas e técnicas para se aproveitar destas falhas obrigam o contínuo aprimoramento da solução. Entretanto, após ter atingido certa maturidade e estabilidade, espera-se que a solução possa ser distribuída para todos os órgãos investigativos oficiais brasileiros que demonstrem interesse em sua utilização. Há de se frisar, porém, que será sempre necessário o acompanhamento de algum especialista em informática, previamente treinado, a fim de poder dar resposta a imprevistos que possam ocorrer.

6.1 Aprimoramento da solução

Como em qualquer processo de desenvolvimento de software, são eleitas as funcionalidades primordiais a serem implementadas, que foram discutidas neste trabalho. Conforme a disponibilidade de tempo, novas etapas de desenvolvimento são iniciadas. Assim, na relação abaixo são listadas algumas sugestões de aprimoramento que, apesar de não serem suficientes para tornar-se tema de projeto futuro, são bastante importantes para melhoria das funcionalidades existentes e complementação da *FID*:

- Definição de um vetor de instalação da *FID*, compreendendo tanto técnicas de engenharia social quanto a implementação de um *downloader*, conforme sugerido na seção 4.5;
- Implementação de ativação e desativação da *FID* baseado em respostas do servidor *WEB*;
- Opção de envio dos dados por *email*, caso algum *Firewall* bloqueie o acesso à porta utilizada pela *FID*.
- Reinício dos serviços, em caso de término ou falha dos mesmos;
- Implementação de conexões *TLS*, a fim de dissimular as conexões *TCP* criadas;
- Melhoria da interface gráfica a ser utilizada pela equipe de investigação e implementação de controle de acesso aos dados capturados;

6.2 Projetos Futuros

A solução, apesar de estar funcional e implementar os mecanismos básicos para atingir os objetivos a que se presta, possui também limitações importantes, como a sua inaplicabilidade em sistemas *Windows* 7 de 64 *bits* e a ausência de um módulo para recuperar senhas criptográficas informadas antes do *boot* do sistema.

Para a primeira, é necessário desabilitar sistemas de proteção inerentes a esta versão, conforme detalhado na seção 2.2. Para a segunda, é necessária uma implementação de um módulo totalmente novo, pois visará obter dados informados fora do escopo do sistema *Windows* 7.

Ou seja, para ambas, é necessário o desenvolvimento de módulos à parte, e, devido à sua complexidade, ficam como sugestão para trabalhos futuros.

Também é sugerida a portabilidade da solução para sistemas operacionais de dispositivos móveis, como o *Windows Mobile*, e também novas implementações para os sistemas *iOS* e *Android*. Tais dispositivos, por apresentarem grande poder de processamento, armazenamento e transmissão de dados, podem substituir o *PC* em várias tarefas, principalmente nas relacionadas à comunicação entre pessoas.

6.3 Contribuição

Apesar das limitações elencadas, a solução mostrou-se eficaz, conforme explicado nas seções 5.3 e 5.3.3. Após a confirmação dos resultados positivos, tem-se que a principal contribuição desta trabalho foi a definição de uma metodologia, talvez a primeira desenvolvida completamente e internamente por um órgão de investigação brasileiro, cujo objetivo é a obtenção do conteúdo inteligível a partir de dados cifrados em interceptações telemáticas autorizadas judicialmente. Por não possuir custos de aquisição, ela apresenta potencial de ser disseminada e utilizada por outros órgãos oficiais, independente de disponibilidade de dotação orçamentária, tornando a investigação e elucidação de crimes mais eficazes e eficientes.

REFERÊNCIAS BIBLIOGRÁFICAS

BLUNDEN, B. *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. [S.l.]: Jones & Bartlett Learning, 2009. 908 p. ISBN 1598220616.

BUNDESTROJANER. *Bundestrojaner.net - Hier gibt es den Bundestrojaner zum Download*. 2011. Disponível em: <http://www.bundestrojaner.net/>. Acesso em: 20 jan. 2012.

CHAOS. *ANALYSEEINERREGIERUNGS - MALWARE*. [S.l.], 2011. Disponível em: <http://www.ccc.de/system/uploads/76/original/staatstrojaner-report23.pdf>. Acesso em: 20 jan. 2012.

CORPORATION, M. *Introduction to Winsock Kernel (WSK)*. 2006. Disponível em: <http://blogs.msdn.com/b/wndp/archive/2006/02/24/538746.aspx>. Acesso em: 20 jan. 2012.

GOLUB, E. *Monitoring of logon/logout in terminal and client sessions*. 2010. Disponível em: <http://www.apriorit.com/our-company/dev-blog/213-monitoring-of-logon-logout-in-terminal-and-client-sessions>. Acesso em: 20 jan. 2012.

GREBENNIKOV, N. *Keyloggers: implementing keyloggers in windows. part two - securelist*. 2007. Disponível em: <http://www.securelist.com/en/analysis/204792178-/Keyloggers_Implementing_keyloggers_in_Windows_Part_Two?print_mode=1>. Acesso em: 20 jan. 2012.

HADNAGY, C.; WILSON, P. *Social Engineering: The Art of Human Hacking*. [S.l.]: John Wiley and Sons, 2010. 408 p. ISBN 0470639539.

HOGLUND, G.; BUTLER, J. *Rootkits: subverting the Windows kernel*. [S.l.]: Addison-Wesley Professional, 2006. 324 p. ISBN 0321294319.

HYPPONEN, M. *Egypt, FinFisher Intrusion Tools and Ethics - F-Secure Weblog : News from the Lab*. 2011. Disponível em: <http://www.f-secure.com/weblog/archives/00002114-.html>. Acesso em: 20 jan. 2012.

IASHCHENKO, V. V. *Cryptography: An Introduction (Student Mathematical Library)*. [S.l.]: American Mathematical Soc., 2002. 229 p. ISBN 0821829866.

KIRIATY, Y. *Session 0 Isolation*. 2009. Disponível em: <http://windowsteamblog.com-/windows/b/developers/archive/2009/10/01/session-0-isolation.aspx>. Acesso em: 20 jan. 2012.

LYNCH, J. *New FBI Documents Provide Details on Government's Surveillance Spyware | Electronic Frontier Foundation*. 2011. Disponível em: <https://www.eff.org-/deeplinks/2011/04/new-fbi-documents-show-depth-government>. Acesso em: 20 jan. 2012.

MICROSOFT. *MSDN - Desenvolvimento na Plataforma . NET da Microsoft.* 2011. Disponível em: <http://msdn.microsoft.com/pt-br/>. Acesso em: 20 jan. 2012.

NIST. *DATA ENCRYPTION STANDARD (DES)*. [S.l.], 1999. 22 p. Disponível em: <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>. Acesso em: 20 jan. 2012.

NIST. *Microsoft Windows 7 Kernel Mode Cryptographic Primitives Library (cng.sys) Security Policy Document*. [S.l.], 2011. 31 p. Disponível em: <http://csrc.nist.gov-/groups/STM/cmvp/documents/140-1/140sp/140sp1328.pdf>. Acesso em: 20 jan. 2012.

OPPENHEIMER, D. L. et al. *System Security: A Management Perspective*. [S.l.]: USENIX Association for Sage, 1997. 91 p. ISBN 1880446855.

PERLA, E.; OLDANI, M. *A Guide to Kernel Exploitation: Attacking the Core*. [S.l.]: Elsevier Science & Technology, 2010. 442 p. ISBN 1597494860.

RSALABORATORIES. *PKCS #1: RSA Cryptography Standard*. 2011. Disponível em: <http://www.rsa.com/rsalabs/node.asp?id=2125>. Acesso em: 20 jan. 2012.

RUBENKING, N. J. *The Best Antivirus for 2012 | PCMag.com*. 2011. Disponível em: <http://www.pcmag.com/article2/0,2817,2372364,00.asp>. Acesso em: 20 jan. 2012.

RUSSINOVICH, M. E.; SOLOMON, D. A.; IONESCU, A. *Windows internals*. [S.l.]: Microsoft Press, 2009. 1232 p. ISBN 0735625301.

RUTKOWSKA, J. Rootkit Hunting vs. Compromise Detection. In: *Black Hat Federal*. [s.n.], 2006. Disponível em: <http://www.blackhat.com/presentations/bh-federal-06/BH-Fed-06-Rutkowska/BH-Fed-06-Rutkowska-up.pdf>. Acesso em: 20 jan. 2012.

STALLINGS, W. *Cryptography and network security*. 4. ed. Prentice Hall, 2005. 592 p. Disponível em: <http://www.amazon.com/Cryptography-Network-Security-William-Stallings/dp/0131873164>.

TANENBAUM, A. S. *Computer Networks (4th Edition)*. Prentice Hall, 2002. 912 p. ISBN 0130661023. Disponível em: <http://www.amazon.com/Computer-Networks-4th-Andrew-Tanenbaum/dp/0130661023>.

VISCAROLA, P. G. *Windows Internals and Software Driver Development*. [S.l.]: Open Systems Resources, 2010.

WHITEFIELD, D. Cryptography: Past, Present and Future. In: *On the Move to Meaningful Internet Systems 2007*. [S.l.]: Springer, 2007. p. 683. ISBN 3540768351.