xFile: Uma Ferramenta Modular para Identificação de *Packers* em Executáveis do Microsoft Windows

Victor F. Martins[1,2], André R. A. Grégio[1,2], Vitor M. Afonso[1,2], Dario S. Fernandes Filho[1,2], Paulo L. de Geus[1]

[1]Universidade Estadual de Campinas (Unicamp) – Campinas – SP – Brasil

[2]Centro de Tecnologia da Informação Renato Archer (CTI)– Campinas – SP – Brasil

furuse@gmail.com, argregio@cti.gov.br, {vitor, dario, paulo}@las.ic.unicamp.br,

Abstract. *Malicious activity in cyberspace has been growing exponentially in the last years and it has been a lucrative business to criminals. In order to avoid defense mechanisms such as antivirus, malware developers use packers, a special kind of software to obfuscate the malicious code and to difficult its detection and analysis. This article presents a multiplataform and object oriented tool to identify packers in Microsoft Windows binaries which allows new modules aggregation to aid in gathering information about malware.*

Resumo. *A atividade maliciosa no espaço cibernético tem crescido muito nos últimos anos e se tornado um negócio lucrativo para criminosos. Para evitar mecanismos de proteção como antivírus, os desenvolvedores de malware usam packers, programas para ofuscar o código malicioso e dificultar sua detecção e análise. Este artigo apresenta uma ferramenta multiplataforma e orientada a objetos para identificar packers em arquivos de executáveis do Microsoft Windows, permitindo a agregação de novos módulos para auxiliar na obtenção de informações sobre malware.*

1. Introdução

A atividade maliciosa no espaço cibernético tem crescido exponencialmente nos últimos anos. Ao longo de 2009 a produção de novos *malware* – software de comportamento malicioso – alcançou o número de 25 milhões de novos exemplares, sendo que cerca de 66% desses *malware* são falsos antivírus e *bankers*, que visam a captura de informações bancárias do usuário [Panda 2009]. Ainda em 2009, notou-se o aumento do comprometimento de sites vistos como confiáveis, isto é, páginas de sites de busca, de governos e de notícias, as quais estão sendo usadas para hospedar e distribuir *malware*. No primeiro semestre de 2009 ocorreram 508% mais comprometimentos deste tipo, segundo relatório anual da IBM [IBM 2009].

A segurança da informação é crucial para as organizações, visto que ataques aos sistemas computacionais resultam em grande perda financeira e de reputação da empresa. Com isso, a preocupação com a segurança da informação acaba ganhando maior importância, passando à frente de desastres naturais e terrorismo. Em 2009 foi

constatado que 75% das empresas em geral sofreram ataques contra seus sistemas, onde 92% deles resultaram em perdas monetárias. As principais perdas devem-se ao roubo de propriedade intelectual e de informações pessoais dos clientes, como dados de cartão de crédito [Symantec 2010]. Esses ataques em geral são efetuados por meio de *malware*.

Em virtude desse cenário, diversas soluções de proteção são estudadas por centros de pesquisa e empresas privadas. Uma forma muito difundida de mecanismo de defesa contra *malware* é o uso de programas antivírus, que geralmente funcionam com base em assinaturas e heurística (análise de comportamento). Antivírus possuem algumas características peculiares, como a necessidade de examinar rapidamente os arquivos a fim de identificar se algo é malicioso, não devendo prover falso-positivos, ou seja, acusar que um executável é malicioso sendo que não é. Outra necessidade é o baixo consumo de processamento, não podendo prejudicar o usuário no uso do computador ou exceder os recursos da máquina. Desta forma, percebe-se que há um grande esforço por trás do programa final que o usuário utiliza, sendo que as organizações que trabalham com antivírus necessitam compreender o funcionamento dos novos exemplares de *malware* a fim de fazer assinaturas para os mesmos sob demanda, visando garantir a qualidade do seu produto em identificar programas maliciosos.

Para tentar lidar com a quantidade excessiva de novos exemplares que surgem diariamente, uma abordagem que tem crescido é a análise minuciosa dos artefatos através de engenharia reversa, para que assim medidas de prevenção por meio de assinaturas de antivírus e atualizações de segurança dos softwares sejam efetuadas rapidamente. Porém essa abordagem tem uma grande dificuldade que é o uso de *packers* nos *malware*. *Packers* ou empacotadores são módulos de cifragem que ofuscam o código *assembly* a fim de dificultar a análise dos artefatos. [Davis et al. 2010]

Este artigo apresenta *xFile*, uma ferramenta modular para identificação de *packers* em arquivos executáveis de sistemas operacionais Microsoft Windows, visando prover uma análise preliminar acerca da maliciosidade nestes arquivos. Na Seção 2 é apresentado o conceito de *packer* e algumas ferramentas para identificação de tipos de arquivos e *packers*. Na Seção 3, são detalhados a arquitetura e o funcionamento da ferramenta. A Seção 4 mostra resultados iniciais na validação da ferramenta enquanto que na Seção 5 as conclusões e trabalhos futuros são discutidos.

2. Análise de arquivos

A utilização de *packers* é uma técnica comum utilizada por programadores de artefatos maliciosos. Um dos motivos é que, dessa forma, antivírus e aplicações de análise estática (análise do *malware* sem executá-lo) não conseguem identificar os *malware* por meio de assinaturas convencionais, visto que as *strings* e seqüências de *opcodes* (instruções de máquina convertidas para hexadecimal) que os caracterizam estão ofuscadas. Então, assinaturas que antes funcionavam precisam ser revistas e adaptadas a inúmeros *packers* que podem estar presentes nos *malware*. Vale salientar que se diferentes *packers* forem aplicados a um certo exemplar de *malware*, o resultado será vários executáveis similares, com as mesmas funcionalidades, porém diferentes. Isto acarreta na propriedade de polimorfismo de *malware*.

O polimorfismo torna a análise aprofundada dos *malware* fundamental para ser possível descobrir as características marcantes que os identifiquem. Caso contrário, não seria possível fazer uma assinatura confiável, visto o alto grau de variação dos *malware.* A identificação do *packer* presente em um *malware* tem um papel fundamental em sua análise e caracterização. Alguns *packers* são conhecidamente usados somente por criminosos digitais, o que já é um forte indício de que o executável é malicioso. Além disso, ao saber qual *packer* está presente no *malware*, a chance de se conseguir analisá-lo é maior, inclusive verificando pela existência de algum modo de *unpacking*.

Para realizar o *unpacking*, existem algumas técnicas que retiram os *packers* de arquivos, desde que se saiba qual tipo está sendo usado. A partir deste conhecimento é possível utilizar o método adequado, deixando somente o conteúdo original do *malware*. Normalmente, o processo de identificação verifica por modificações no cabeçalho do executável e/ou busca por funções de cifragem e decifragem do *packer*.

Com o *malware* sem o *packer,* é possível estudá-lo minuciosamente, permitindo principalmente a criação de assinaturas de identificação mais genéricas e a aplicação de engenharia reversa e *debugging* seguindo o fluxo normal de execução do *malware*. Todavia, na análise do *malware* não basta apenas identificar o *packer* presente, mas também o tipo do arquivo. Essa informação permite saber para qual sistema operacional o *malware* foi desenvolvido e como este interage com o sistema, se necessita de uma aplicação específica, se é uma biblioteca ou executável *stand-alone*, por exemplo.

2.1 Identificadores de *packers*

Quanto à identificação de *packers*, as ferramentas mais comumente utilizadas são o PEiD e o *pefile*. Ambas atuam sobre arquivos em formato *Portable Executable*, mais conhecido como PE [PECOFF 2008], que é uma estrutura de dados na qual os executáveis do Microsoft Windows são encapsulados. Nas subseções a seguir, o padrão PE é explicado, bem como as ferramentas citadas.

2.1.1 *Portable Executable* (PE)

O *Portable Executable* (PE) é um formato criado pela Microsoft para executáveis, códigos objetos e DLLs das diversas versões do sistema operacional Microsoft Windows, padronizado em 1993 pelo *Tool Interface Standard Committee.*

O PE é uma estrutura muito bem projetada, a qual suporta diversas versões do sistema operacional Microsoft Windows em várias arquiteturas. Nesta estrutura estão todas as informações necessárias para o sistema operacional carregar (*system loader*) e executar um programa. Sua macro-estrutura está representa na Figura 1.

O estudo e a compreensão da estrutura do PE possibilitam a manipulação total deste tipo de arquivo, seja para fins de *debugging*, *dumping* de memória, subversão do fluxo de execução ou *cracking* (violação de direitos autorais). Assim, uma das técnicas utilizadas em *packers* é a alteração de *Entry Points* presentes no PE para dificultar a análise do arquivo. *Entry Points* são endereços de memória que direcionam o fluxo de execução do programa (através de *jump*) para um código interno como uma função ou externo, por exemplo, uma biblioteca. Desta forma, um *packer* pode, ao alterar *Entry Points* de um programa, inserir novas funções ou embaralhar todo o conteúdo do programa sem alterar suas funcionalidades.

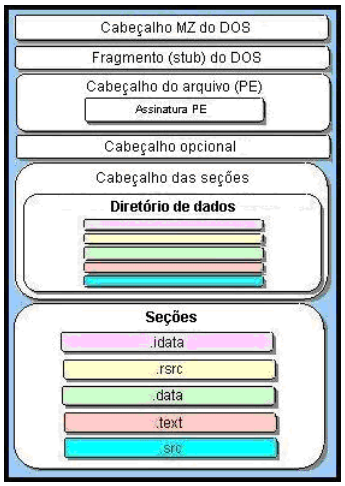

Figura 1: Estrutura do *Portable Executable*[1]

O estudo e a compreensão da estrutura do PE possibilitam a manipulação total deste tipo de arquivo, seja para fins de *debugging*, *dumping* de memória, subversão do fluxo de execução ou *cracking* (violação de direitos autorais). Assim, uma das técnicas utilizadas em *packers* é a alteração de *Entry Points* presentes no PE para dificultar a análise do arquivo. *Entry Points* são endereços de memória que direcionam o fluxo de execução do programa (através de *jump*) para um código interno como uma função ou externo, por exemplo, uma biblioteca. Desta forma, um *packer* pode, ao alterar *Entry Points* de um programa, inserir novas funções ou embaralhar todo o conteúdo do programa sem alterar suas funcionalidades.

2.1.2 PEiD[2]

O PEiD [PEiD 2010] é um software desenvolvido para detectar os mais comuns *packers* em executáveis Microsoft Windows, através de uma biblioteca interna de assinaturas. Também há suporte para a adição de bibliotecas externas com assinaturas criadas por terceiros.

As assinaturas são compostas por três campos, sendo eles: nome do *packer*, *signature* e *ep_only*. O campo "nome do *packer*" representa o identificador nominal de certo *packer*. O "*signature*" é o campo da assinatura, isto é, uma seqüência hexadecimal que caracteriza um determinado *packer*. As assinaturas geralmente são cadeias de *opcodes* que representam funções específicas do *packer*, permitindo a identificação de seu uso. Por fim, a variável *ep_only* diz se o *packer* deve ser procurado somente nos *Entry Points* (estrutura do PE) do programa ou em todo o arquivo.

A varredura de uma assinatura somente nos *Entry Points* diminui o risco de falso-positivos. Um exemplo de assinatura utilizada no PEiD pode ser vista na Figura 2, na qual os pontos de interrogação presentes no campo "*signature*" indicam que qualquer valor hexadecimal pode estar presente.

---

1 *Fonte:* http://www.devmedia.com.br/imagens/clubedelphi/nov2005/1.gif

2 Fonte: http://www.peid.info/

```
[Themida 1.0.x.x - 1.8.0.0 (compressed engine) -> Oreans
Technologies]
signature = B8 ?? ?? ?? ?? 60 0B C0 74 58 E8 00 00 00 00 58 05
43 00 00 00 80 38 E9 75 03 61 EB 35 E8 00 00 00 00 58 25 00 F0
FF FF 33 FF 66 BB 19 5A 66 83 C3 34 66 39 18 75 12 0F B7 50 3C
03 D0 BB E9 44 00 00 83 C3 67 39 1A 74 07 2D 00 10 00 00 EB DA
8B F8 B8
ep_only = true
```

Figura 2: Exemplo de assinatura do PEiD

O PEiD possui uma comunidade bastante ativa e que contribui bastante com o desenvolvimento da ferramenta. Em fóruns mantidos pelos desenvolvedores facilmente encontra-se bibliotecas externas com inúmeras novas assinaturas de *packers*. Entretanto, apesar da ferramenta ser livre (*freeware*), não é aberta (*open source*), o que dificulta adaptá-la a necessidades mais específicas de um usuário, além de não identificar tipos de arquivos. Adicionalmente, a ferramenta PEiD possui muitas outras funcionalidades, porém fogem ao escopo deste trabalho e não serão citadas.

2.1.3 *Pefile*

Pefile [Pefile 2010] é uma ferramenta *freeware* e *open source* que, em linhas gerais, lê e manipula todas as informações presentes na estrutura PE de um arquivo, assim como verifica a presença de valores incorretos e suspeitos. Outras funcionalidades do *pefile* são: a detecção de *packers* por meio de assinaturas do PEiD e a criação destas.

A detecção de assinaturas em um executável Microsoft Windows ocorre nos mesmo moldes do PEiD, que foram citados na subseção 2.1.2, sendo que as assinaturas são no mesmo padrão mostrado anteriormente na Figura 2. *Pefile* foi escrito em linguagem Python podendo funcionar em diversos sistemas operacionais e da mesma forma que o PEiD, não é capaz de identificar tipos de arquivo.

2.2 Identificadores de tipo de arquivo

Para identificar o tipo de um arquivo existe uma ferramenta amplamente utilizada em sistemas operacionais baseados em Unix, chamada "file" [File 1994]. Um outro programa interessante com o mesmo objetivo é o TrID [TrID 2010]. Ambos serão expostos nas subseções adiante.

2.2.1 File

A ferramenta "file" [File 1994] identifica o tipo de um arquivo por meio da execução de um conjunto de três testes, sendo que o primeiro teste bem-sucedido passa a ser o tipo identificado para o arquivo.

O primeiro teste realizado é referente ao sistema de arquivos (*filesystem*) utilizado, o segundo é a busca de *magic numbers* no arquivo e o último refere-se a testes de linguagem (codificação).

O segundo teste é o mais poderoso para identificar e especificar tipos de executáveis Microsoft Windows, pois utiliza *magic numbers* para classificar os arquivos. *Magic numbers* são dados, *strings* e números, presentes em certos *offsets* que caracterizam certo tipo de arquivo, dado que esses tipos seguem padrões determinados.

Por exemplo, executáveis Microsoft Windows compilados para máquinas de 32 bits possuem campos no PE informando essa situação.

Para reconhecer os *magic numbers,* "file" possui uma biblioteca contendo as assinaturas de tipos de arquivos, chamada de *magic* [Magic 1994]. Tais assinaturas são compostas por quatro campos, sendo eles: *offset*, *type*, *test* e *message*.

Na Figura 3 é possível ver um exemplo de assinatura *magic*: o primeiro campo é do *offset* e ele indica o nível de teste a ser realizado no campo *test* e a posição do *offset* onde deve ser buscado o tipo indicado em *type*. O *type* pode ser desde *unsigned byte*, passando por *little endian short*, até *strings.* Já o campo *test,* indica qual teste deve ser feito com o *type* lido, por exemplo, se o número lido deve ser maior que um certo número e assim por diante. Por fim, o campo *message* é impresso quando o teste é bem-sucedido, podendo ser nulo.

```
# MS Windows executables are also valid MS-DOS executables
0           string  MZ
>0x18       leshort <0x40   MZ executable (MS-DOS)
```

Figura 3: Exemplo de assinatura do *Magic*

As configurações permitidas na biblioteca Magic são bem extensas e permitem inúmeras possibilidades de assinaturas. [Magic 1994]

2.2.2 TrID

A ferramenta TrID [TrID 2010] possui uma singularidade bem interessante com relação às outras ferramentas de identificação de tipo de dados. Ela não é determinística quanto à classificação de um arquivo, mas probabilística.

Com base em um biblioteca interna, o TrID calcula qual a probabilidade de um arquivo ser de determinados tipos. A Figura 4 mostra um exemplo de classificação.

```
C:\TrID>trid c:\test\doc\lasik_info.doc
70.7% (.DOC) Microsoft Word document (58000/1/5)
29.3% (.) Generic OLE2 / Multistream Compound File (24000/1)
```

Figura 4: Exemplo de assinatura do TrID

3. Detalhamento da Ferramenta

O projeto da ferramenta *xFile* teve por objetivo principal a integração com um sistema de análise de *malware* desenvolvido pelos autores deste trabalho, a fim de dar início ao processo da coleta de informações gerais sobre um binário malicioso. Assim, *xFile* foi desenvolvida de forma modular e orientada a objetos (Java), que além de possibilitar o uso multiplataforma, permite a agregação de novas funcionalidades. Portanto, o foco da ferramenta foi portar para Java um identificador de tipo de arquivos do Microsoft Windows em conjunto com um identificador de *packers*.

As subseções a seguir detalham a arquitetura e o funcionamento de cada módulo da ferramenta.

3.1 Arquitetura

A arquitetura da ferramenta *xFile* foi dividida em três módulos para atender os objetivos propostos: o Módulo *xFile*, Módulo *Magic* e o Módulo *PE*, conforme a Figura 5.

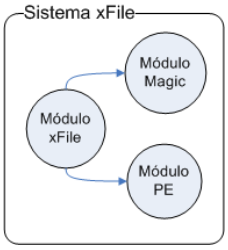

Figura 5: Módulos do Sistema xFile

O Módulo *xFile* provém dados para o funcionamento correto dos outros dois módulos. Essa relação será descrita com detalhes nas seções 3.3, 3.4 e 3.5.

3.3 Módulo xFile

O Módulo *xFile* é o responsável por ler o binário malicioso e apresentá-lo em hexadecimal para os outros módulos, Módulo *Magic* e Módulo PE; além de guardar informações referentes ao *malware,* como nome e *path.*

3.4 Módulo Magic

O Módulo *Magic* é encarregado por classificar um arquivo quanto ao tipo de dado que possui, através de *magic numbers*. Este módulo é subdividido em quatro partes internas - o Carregador, o Pesquisador, a Estrutura de dados e as Assinaturas *Magic* - e em duas externas - a Biblioteca *Magic* e o Módulo *xFile* - apresentadas na Figura 6.

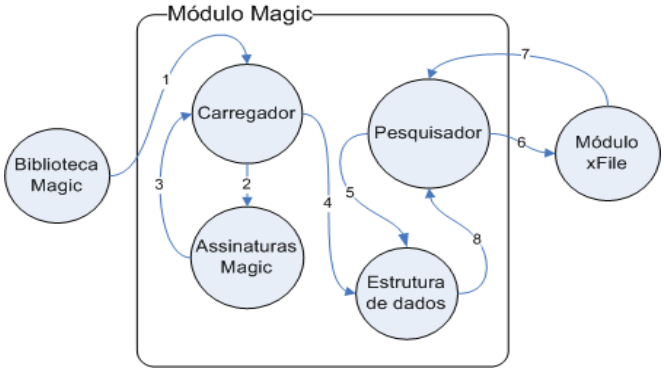

Figura 6: Detalhamento do funcionamento do Módulo Magic

Inicialmente deve-se fornecer uma Biblioteca *Magic*, na forma de arquivo texto, com as assinaturas *magic* que pretende-se trabalhar. A biblioteca utilizada foi parte do arquivo original *magic* [Magic 1994], contendo apenas as assinaturas de Windows, como aplicativos executáveis e bibliotecas dinâmicas (DLLs).

O Carregador do Módulo *Magic*, ao obter a biblioteca, lê as informações contidas na mesma e divide nos campos apresentados na Figura 3. Logo em seguida, esses dados são passados para um objeto Assinatura *Magic*, o qual encapsula as informações da assinatura e dá significado semântico para os campos *offset*, *type* e *test.*

Para exemplificar, no objeto criado o campo *offset* estará interpretado, ou seja, o objeto já conterá o nível do teste e caso seja um *offset* indireto terá o calculo a ser realizado para a obtenção do valor do *offset*. No campo *type,* se houver máscara de *AND* para ser aplicada no valor lido, essa máscara será separada durante o carregamento da assinatura. Portanto, todo o trabalho de processamento das *strings* em algum significado semântico é realizado durante o carregamento da biblioteca, pelo Carregador.

Duas coisas são importantes frisar com relação ao carregamento da biblioteca. A primeira é que a partir do momento que a biblioteca está carregada na memória, a mesma não precisa ser lida novamente. A segunda é que cada assinatura *magic* ocupa um objeto de Assinatura *Magic*, ou seja, cada linha do arquivo texto da biblioteca é repassado para apenas um objeto. Depois que um objeto Assinatura *Magic* é montado, ele é enviado para o Carregador novamente, o qual repassa para a Estrutura de dados armazenar o objeto.

A Estrutura de dados do Módulo *Magic* é uma lista encadeada de árvores, na qual cada árvore não é balanceada e não possui número máximo de nós por nível, assim sendo um nó pode ter de zero a *n* filhos. Essa estrutura de dados escolhida é resultado da análise da estrutura e da hierarquia de como foi definido o arquivo *magic* [Magic 1994].

No arquivo texto da biblioteca, o nível de um teste é definido pela quantidade de "**>**" presentes no campo *offset,* porém na Estrutura de dados o nível do teste é o mesmo da árvore. Já o nó pai de um objeto Assinatura *Magic* é a primeira assinatura antecessora que tiver um nível a menos que seu próprio nó. Essa relação é de extrema importância para a classificação do arquivo.

O Pesquisador é a parte do Módulo *Magic* responsável por identificar o tipo de um arquivo. Para isto ele utiliza a biblioteca carregada na memória (Estrutura de dados) e as informações fornecidas pelo Módulo *xFile*. Primeiramente, o Pesquisador percorre a Estrutura de dados testando cada Assinatura *Magic* no *malware*, e caso alguma esteja correta o campo *message* da assinatura é impresso e o teste continua no próximo nível do nó. Essa pesquisa somente acaba quando não houver mais nós para serem testados, ou seja, o Pesquisador tiver atingido as folhas da árvore.

3.5 Módulo PE

O Módulo PE possui a mesma subdivisão que o Módulo *Magic*, porém com diferenças quanto à complexidade das funcionalidades das partes, devido a maior simplicidade do aplicativo PEiD. Assim, a Figura 6 pode ser usada para detalhar o funcionamento do módulo.

A biblioteca utilizada no Módulo PE também é um arquivo texto e é encontrado no próprio site do PEiD [UserDB 2010]. Inicialmente o Carregador lê a biblioteca e separa as informações nos campos do PEiD, mostrados na Figura 2, e posteriormente esses dados são encapsulados em objetos de Assinaturas PE. Depois esses objetos são guardados em uma lista ligada (Estrutura de dados).

Com a biblioteca carregada na memória a identificação de *packers* no *malware* ocorre testando cada assinatura PE com o arquivo malicioso, e caso algum teste seja bem sucedido o nome do *packer* em questão é armazenado em uma pilha de resultados. O teste ocorre como descrito na seção 2.1.2, ou seja, se for encontrado uma seqüência de

hexadecimal que case perfeitamente a assinatura com uma parte do *malware,* é entendido como teste bem-sucedido. Esse teste é implementado no *xFile* através de expressões regulares (*Pattern*).

É importante salientar que a Estrutura de dados é totalmente percorrida. E no fim da execução pode haver mais de um *packer* identificado na pilha de resultados. Assim a resposta que o *xFile* dará é do último *packer* encontrado e que tenha a variável *ep_only* como *true,* pois como foi citado na seção 2.1.1 e 2.1.2 a busca de características marcantes dos *packers* nos *Entry Point* reduz falsos positivos; e é escolhido a última assinatura, pois na biblioteca as assinaturas são colocadas no arquivo em ordem decrescente de generalização, ou seja, as assinaturas mais genéricas aparecem primeiro e por último estão as mais específicas.

4. Testes e Resultados

Para verificar a corretude dos resultados apresentados pela ferramenta *xFile* na análise de um executável malicioso, submeteu-se um conjunto de *malware* de teste e comparou-se as saídas obtidas com as dos programas *file* e *pefile*, obtendo-se as mesmas identificações. Uma amostra dos resultados é apresentada na figura 6.

```
Malware1
Md5: 9dfda32f28c1e11e3760a9237b031524
Packer: ASPack v2.12 -> Alexey Solodovnikov
Tipo: PE 32 executable  for MS Windows(GUI) Intel 80386 32-bit
Malware 2
Md5: 15f60476b36297d6f73f08bc0560a80d
Packer: UPX 2.90 [LZMA] -> Markus Oberhumer, Laszlo Molnar & John Reiser
Tipo: PE 32 executable  for MS Windows(GUI) Intel 80386 32-bit
Malware 3
Md5: eca8de7442bcb2c768074040d2025a19
Packer: PECompact 2.xx --> BitSum Technologies
Tipo: PE 32 executable  for MS Windows(GUI) Intel 80386 32-bit
Malware 4
Md5: e693ca9255c357a2357cec87132a379b
Packer: tElock 0.98 -> tE!
Tipo: PE 32 executable for MS Windows(GUI) Intel 80386 32-bit
```

Figura 6: Amostra dos resultados obtidos com a ferramenta xFile

5. Considerações Finais

A análise de *malware* deve ser uma atividade executada de maneira rápida a fim de promover a tomada de contra-medidas em casos de comprometimento de sistemas. Alguns analisadores de *malware* não executam certos arquivos dependendo do *packer* utilizado, pois o sistema de análise pode não suportá-lo [Balzarotti et al. 2010]. Além disso, o conhecimento do tipo de arquivo é vital para saber com qual aplicação ou linha de comando o *malware* deve ser executado, caso seja uma biblioteca, por exemplo.

A ferramenta *xFile* proposta neste artigo integra a identificação do tipo de arquivo com a identificação do *packer*, possibilitando que se obtenha algumas informações básicas em apenas uma verificação e já se identifique como o *malware* deve ser analisado, se deve ser

desempacotado primeiro ou se é necessário uma aplicação específica para executá-lo.

Além disso, a ferramenta permite que novos módulos sejam agregados com a finalidade de realizar mais etapas da análise estática, podendo inclusive ser integrada a um *framework* de análise, dado que o tipo de arquivo e o *packer* são informações importantes para o início da análise e mesmo da engenharia reversa de um *malware*.

Como trabalhos futuros, *xFile* pode ser otimizada alterando-se a estrutura de árvores, para que os tipos mais comuns sejam os primeiros a serem checados e o processo de identificação seja mais eficiente. A ferramenta também pode ser expandida facilmente para identificar outros tipos de arquivos, não só executáveis do Microsoft Windows.

Referências

Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Kirda, E. and Vigna, G. (2010) "Efficient detection of split personalities in malware", 17th Annual Network and Distributed System Security Symposium, February 28th-March 3rd.

Davis, M., Bodmer, S. and LeMasters, A. (2010). "HACKING EXPOSED MALWARE AND ROOTKITS", McGraw-Hill, Inc., 1th edition.

File, manual (1994); http://www.opengroup.org/onlinepubs/9699919799/utilities/ /file.html, acessado em 01/08/2010.

IBM, Report (2009), http://www.servicemanagementcenter.com/main/pages/ /IBMRBMS/SMRC/ShowCollateral.aspx?oid=74711, acessado em 01/08/2010.

Magic, manual (1994); http://unixhelp.ed.ac.uk/CGI/man-cgi?magic, acessado em 01/08/2010.

PANDA (2009) "Anual Reports – PandaLabs", http://www.pandasecurity.com/img/enc/Annual_Report_PandaLabs_2009.pdf, acessado em 01/08/2010.

PECOFF, Microsoft (2008), "Microsoft Portable Executable and Common Object File Format Specification", http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/pecoff_v8.docx, acessado em 01/08/2010.

Pefile (2010), http://code.google.com/p/pefile/, acessado em 01/08/2010.

PEiD (2010), http://www.peid.info/, acessado em 01/08/2010.

SYMANTEC (2010) "State of Enterprise Security", http://www.symantec.com/content/en/us/about/presskits/SES_report_Feb2010.pdf, acessado em 01/08/2010.

TrID, Tool (2010); http://mark0.net/soft-trid-e.html, acessado em 01/08/2010.

UserDB (2010), http://www.peid.info/BobSoft/Downloads.html, acessado em 01/08/2010.