Detecção de artefatos maliciosos a partir de indicadores de comprometimento

Heyder Andrade[1], **Henrique Ferraz Arcoverde**[2]

[1]Departamento de Gestão de Sistemas da Informação – Faculdade Santa Maria (FSM)
Recife – PE – Brasil

[2]Software Security Labs – Tempest Security Intelligence
Recife – PE – Brasil

emailredacted , emailredacted

Abstract. *Given the growth in the volume of malicious artifacts in the network traffic, and the inefficiency of the classical techniques of antivirus, this paper suggests use of IOCs as a basis for detecting malicious artifacts. once undergoing classification algorithms based on learning techniques machine, it is possible to reach detection rates than the rates obtained by classical approaches.*

Resumo. *Diante do crescimento no volume de artefatos maliciosos que trafegam na rede, e da ineficiência das técnicas clássicas de antivírus, este trabalho sugere a utilização de IOCs como base para a detecção de artefatos maliciosos. Uma vez submetidos à algoritmos de classificação baseados em técnicas de aprendizado de máquina, é possível chegar a taxas de detecção acima das taxas obtidas pelas abordagens clássicas.*

1. Introdução

Apesar das diversas tentativas de inviabilizar os problemas relativos à ataques provenientes de artefatos maliciosos, tais artefatos continuam presentes em grande volume. Um exemplo disso, especificamente no cenário brasileiro, reside nas sucessivas perdas relatadas pela FEBRABAN [FEBRABAN 2011].

Com o intuito de mitigar as consequências das ações de artefatos maliciosos, faz-se necessário distinguir arquivos legítimos de arquivos mal-intencionados. Tal incumbência, de maneira manual, dado o grande volume de dados trafegados na *web*, torna-se extremamente onerosa e difícil. Diante de tal fato, faz-se necessário automatizar, com um grau de acertividade conveniente, a tarefa de detecção de artefatos maliciosos.

A detecção automática de artefatos maliciosos se baseia em técnicas clássicas como *scanning*, checagem de integridade, heurísticas de detecção, emulação de código, etc. [Szor 2005]. O desenvolvimento de tais técnicas tem se mostrado ineficiente diante da constante evolução dos artefatos maliciosos. Alguns afirmam que os antivírus estão mortos e fadados ao fracasso [Goodin 2014]. Além disso, dados apontam que os antivírus não são capazes de identificar 45% dos ataques cibernéticos realizados [Yadron 2014].

A fim de expandir as possibilidades de detecção de artefatos maliciosos, o presente visa verificar a viabilidade da utilização de IOCs (*indicators of compromise*) como indicadores de atividades maliciosas. Neste trabalho, o uso de tais indicadores está associado a algoritmos de aprendizado de máquina, com o objetivo de determinar a presença ou não de atividades maliciosas a partir de tais artefatos.

2. Conceitos básicos

A presente seção descreve os conceitos básicos necessários para o entendimento das atividades descritas neste trabalho.

2.1. Malware

Um *malware* é todo e qualquer *software* que faça algo que cause detrimento ao usuário, à máquina ou à rede de computadores [Sikorski and Honig 2012]. Uma vez que este trabalho não visa realizar a classificação de *malwares* em classes específicas, não é realizada nenhuma distinção entre tipos ou variantes de *malware*, sendo a distinção entre artefatos maliciosos e não maliciosos suficiente para as atividades.

2.2. Análise de malwares

Sendo *malware* um vetor amplamente utilizado em ataques contra computadores, faz-se necessário conhecer as técnicas que possibilitam extrair informações sobre os mesmo. A disciplina responsável por extrair tais informações é conhecida por análise de *malwares* e pode se dar a partir de seu código fonte – o que é incomum, haja vista a dificuldade de se obter acesso a esses arquivos em tempo, ou a partir de seu binário. De maneira geral, existem duas abordagens macro para a realização da análise de um *malware*: análise estática e análise dinâmica [Sikorski and Honig 2012] [Ligh et al. 2010].

Genericamente, tem-se por análise dinâmica o processo de análise de um determinado programa durante sua execução. Em contrapartida, a análise estática refere-se a

todas as técnicas que analisam um programa sem a necessidade de execução do mesmo [Sikorski and Honig 2012] [Siddiqui 2008].

Uma boa prática na disciplina de análise de *malware*, quando se deseja obter informações (como comportamento, técnicas, tráfego de rede, *logs* de arquivos, alterações em chaves de registro, etc.) sobre um artefato, é combinar as informações da análise estática e dinâmica. [Siddiqui 2008]. A experiência mostra que a utilização de diferentes técnicas de análise de *malware* permite que o analista de *malware* rapidamente, e em detalhes, entenda o risco e a intenção de uma determinada amostra.

Em resumo, a análise de um *malware* se constitui através de um somatório de técnicas que permitem identificar o comportamento, características, técnicas, etc. [Ligh et al. 2010]. Ademais, uma definição razoável do que é uma análise de *malware* é a seguinte: "Análise de malware é a arte de dissecar e entender como funciona, como identificar e como se defender ou eliminar um *malware*."[Sikorski and Honig 2012].

2.2.1. Análise estática

Em concordância com o seu nome, a análise estática consiste em analisar o potencial artefato malicioso sem que este seja executado, ou seja, estando este estático. [Arcoverde 2013]

Apesar de algumas limitações e inconvenientes (como o tempo e dificuldade de análise manual), a análise estática oferece informações sobre o programa, como o fluxo de dados e outros recursos estáticos sem realmente executar o programa [Siddiqui 2008]. Através de análise estática é possível obter minimamente:

- Tamanho do binário;
- *Packer* utilizado;
- Tecnologia de desenvolvimento do malware;
- Presença de algoritmos de criptografia;
- Presença de técnicas anti-debugging;
- Tipo de arquivo;
- Lista de funções importadas.

Outro sério inconveniente reside em "despackear" um *malware* quando este está "packeado", comprimido ou mesmo cifrado. Normalmente tal característica dificulta a extração de informações estáticas, pois suas reais funções estão encapsuladas dentro de um código que não representa sua "regra de negócio". [Sikorski and Honig 2012]

Resumidamente, a ideia da análise estática é obter informações sobre o funcionamento interno do artefato, através da realização de engenharia reversa no código objeto, seções do arquivo, bibliotecas utilizadas, etc. [Sikorski and Honig 2012].

Ademais, em contrapartida aos inconvenientes da análise dinâmica, a análise estática tende a ser executada mais rapidamente uma vez que não necessita da interação do artefato malicioso com nenhuma entidade externa. [Sikorski and Honig 2012]

2.2.2. Análise dinâmica

A análise dinâmica é realizada através da observação e registro do comportamento de um *malware*, enquanto em execução na máquina *host*, normalmente em um ambiente controlado – é comum que tais ambientes sejam compostos por máquinas virtuais e *softwares* de *sandboxes*. Além disso, este tipo de análise também envolve depurar o *malware* durante a sua execução usando um *debugger* para assistir o seu comportamento passo-a-passo, enquanto suas instruções estão sendo processadas [Sikorski and Honig 2012] [Ligh et al. 2010].

Uma dos efeitos colaterais benéficos da utilização da análise dinâmica de *malwares* reside no conhecimento de quais recursos devem ser modificados de modo a remover o *malware* de um sistema [Ligh et al. 2010]. Tal fato se dá pois durante a análise dinâmica de *malware* é possível verificar, dentre outras, as seguintes informações:

- arquivos criados;
- arquivos editados;
- arquivos removidos;
- chaves de registro criadas;
- chaves de registro removidas;
- chaves de registro alteradas;
- tráfego de rede;

Apesar das vantagens da análise dinâmica, tal qual ocorre com a análise estática, possui seus inconvenientes [Ligh et al. 2010]. Uma das maiores limitações da análise dinâmica ocorre devido a presença de gatilhos, que determinam se uma ação deve ou não ser tomada [Sikorski and Honig 2012]. Tais gatilhos estão normalmente ligados à regra de negócio do artefato malicioso e são mais fáceis de serem encontrados através de análise estática [Sikorski and Honig 2012]. Um exemplo hipotético deste tipo de gatilho pode ser o condicionamento de um artefato em se comunicar com o servidor de comando e controle, tão somente quando a vítima acessar uma determinada página. Com esse tipo de comportamento, uma análise pode ser ludibriada caso os gatilhos não sejam corretamente disparados.

Ademais, a coleta de informações em uma análise dinâmica pode ser realizada de diversas formas: com sensores externos ou internos ao ambiente de execução. Para o presente trabalho, será utilizado o ambiente de análise *Cuckoo Sandbox*. Por fim, a utilização de mecanismos de *sandbox* e máquinas virtuais pode trazer consequências à execução do artefato caso o mesmo realize a detecção de tais mecanismos.

2.3. Indicador de comprometimento

No contexto de análise de *malware*, um IOC, do inglês *Indicators of Compromise* é um artefato forense ou remanescente de uma intrusão que pode ser identificado em um host ou rede [Gragido 2012].

Esses indicadores, reunidos, devem permitir a um analista descrever as características técnicas que identificam uma ameaça conhecida, como: a metodologia de um atacante, evidências de comprometimento, técnicas de ataque, potenciais informações subvertidas, etc [Gragido 2012].

A ideia de IOC traz a ideia de substituir as tradicionais assinaturas que são facilmente subvertidas por ameaças mais sofisticadas [OpenIOC 2014]. Tal abordagem é realizada através da aglutinação de informações que descrevem a ação de um artefato malicioso [OpenIOC 2014]. Um exemplo de IOC que determina quais as características de um determinado artefato malicioso pode ser descrito, por exemplo, pela presença, na máquina comprometida, de arquivos, chaves de registro, acesso a URLs, etc. [OpenIOC 2014].

Dentro da comunidade de analistas de *malwares* é possível citar a presença de iniciativas para o compartilhamento de tais indicadores de comprometimento, sendo talvez o mais importante o OpenIOC. Segue abaixo um exemplo do OpenIOC que representa os indicadores do *malware Stuxnet*:

```
<ioc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.
w3.org/2001/XMLSchema" xmlns="http://schemas.mandiant.com/2010/ioc"
id="ea3cab0c-72ad-40cc-abbf-90846fa4afec" last-modified="2011-11-04T19:35:05">
<short_description>STUXNET VIRUS (METHODOLOGY)</short_description>
<description>
Generic indicator for the stuxnet virus. When loaded, stuxnet spawns lsass.exe
in a suspended state. The malware then maps in its own executable section and
fixes up the CONTEXT to point to the newly mapped in section. This is a common
task performed by malware and allows the malware to execute under the pretense
of a known and trusted process.
</description>
<keywords>methodology</keywords>
<authored_by>Mandiant</authored_by>
<authored_date>0001-01-01T00:00:00</authored_date>
<links/>
<definition>
<Indicator operator="OR" id="73bc8d65-826b-48d2-b4a8-48918e29e323">
<IndicatorItem id="b9ef2559-cc59-4463-81d9-52800545e16e" condition="contains">
<Context document="FileItem" search="FileItem/PEInfo/Sections/Section/Name" type="mir"/>
<Content type="string">.stub</Content>
</IndicatorItem>
<IndicatorItem id="156bc4b6-a2a1-4735-bfe8-6c8d1f7eae38" condition="contains">
<Context document="FileItem" search="FileItem/FileName" type="mir"/>
<Content type="string">mdmcpq3.PNF</Content>
</IndicatorItem>
<IndicatorItem id="e57d9a5b-5e6a-41ec-87c8-ee67f3ed2e20" condition="contains">
<Context document="FileItem" search="FileItem/FileName" type="mir"/>
<Content type="string">mdmeric3.PNF</Content>
</IndicatorItem>
<IndicatorItem id="63d7bee6-b575-4d56-8d43-1c5eac57658f" condition="contains">
<Context document="FileItem" search="FileItem/FileName" type="mir"/>
<Content type="string">oem6C.PNF</Content>
</IndicatorItem>
<IndicatorItem id="e6bff12a-e23d-45ea-94bd-8289f806bea7" condition="contains">
<Context document="FileItem" search="FileItem/FileName" type="mir"/>
<Content type="string">oem7A.PNF</Content>
</IndicatorItem>
<Indicator operator="AND" id="422ae9bf-a1ae-41f2-8e54-5b4c6f3e1598">
<IndicatorItem id="e93f1610-daaf-4311-bcf3-3aecef8271c0" condition="contains">
<Context document="DriverItem" search="DriverItem/DeviceItem/AttachedToDriverName"
type="mir"/>
<Content type="string">fs_rec.sys</Content>
</IndicatorItem>
```

<IndicatorItem id="72476f35-8dea-4bae-8239-7c22d05d664f" condition="contains">
<Context document="DriverItem" search="DriverItem/DeviceItem/AttachedToDriverName"
type="mir"/>
<Content type="string">mrxsmb.sys</Content>
</IndicatorItem>
<IndicatorItem id="f98ea5aa-9e23-4f18-b871-b3cf5ba153fe" condition="contains">
<Context document="DriverItem" search="DriverItem/DeviceItem/AttachedToDriverName"
type="mir"/>
<Content type="string">sr.sys</Content>
</IndicatorItem>
<IndicatorItem id="32f61140-0f58-43bc-8cdd-a25db75ca6c4" condition="contains">
<Context document="DriverItem" search="DriverItem/DeviceItem/AttachedToDriverName"
type="mir"/>
<Content type="string">fastfat.sys</Content>
</IndicatorItem>
</Indicator>
<Indicator operator="AND" id="eb585bf5-18d8-4837-baf0-80ac74104ca3">
<IndicatorItem id="8d85b559-4d18-4e15-b0c9-da5a9b32f53c" condition="contains">
<Context document="FileItem" search="FileItem/FileName" type="mir"/>
<Content type="string">mrxcls.sys</Content>
</IndicatorItem>
<IndicatorItem id="8a3e425d-fa87-4a31-b20d-8f8630d77933" condition="contains">
<Context document="FileItem" search="FileItem/PEInfo/DigitalSignature/CertificateSubject"
type="mir"/>
<Content type="string">Realtek Semiconductor Corp</Content>
</IndicatorItem>
</Indicator>
<Indicator operator="AND" id="bc8d06dd-f879-4609-bb1c-eccded0222ce">
<IndicatorItem id="89f194d3-3ee6-4218-93f8-055ea92a9f00" condition="contains">
<Context document="FileItem" search="FileItem/FileName" type="mir"/>
<Content type="string">mrxnet.sys</Content>
</IndicatorItem>
<IndicatorItem id="c2dae8bf-81b1-49fb-8654-396830d75ade" condition="contains">
<Context document="FileItem" search="FileItem/PEInfo/DigitalSignature/CertificateSubject"
type="mir"/>
<Content type="string">Realtek Semiconductor Corp</Content>
</IndicatorItem>
</Indicator>
<Indicator operator="AND" id="00538c36-88fe-42ea-a70f-136a2fb53834">
<IndicatorItem id="a779b811-345f-4164-897e-0752837d0c1e" condition="contains">
<Context document="RegistryItem" search="RegistryItem/Path" type="mir"/>
<Content type="string">
HKEY_LOCAL_MACHINE001
</Content>
</IndicatorItem>
<IndicatorItem id="ee981f06-b713-40aa-ac98-c6f4fd82b78d" condition="contains">
<Context document="RegistryItem" search="RegistryItem/Text" type="mir"/>
<Content type="string">mrxcls.sys</Content>
</IndicatorItem>
</Indicator>
<Indicator operator="AND" id="d8d9b32c-f648-4552-9805-93c05ed48219">
<IndicatorItem id="c08044e7-e88c-433c-b463-763bdddeff82" condition="contains">
<Context document="RegistryItem" search="RegistryItem/Path" type="mir"/>
<Content type="string">
HKEY_LOCAL_MACHINE001
</Content>

```
</IndicatorItem>
<IndicatorItem id="38dfb382-ebbe-4685-bbb7-60675b91bd15" condition="contains">
<Context document="RegistryItem" search="RegistryItem/Text" type="mir"/>
<Content type="string">mrxnet.sys</Content>
</IndicatorItem>
</Indicator>
</Indicator>
</definition>
</ioc>
```

No presente trabalho, a ideia é utilizar os indicadores de comprometimento como subsídio para a detecção de artefatos maliciosos.

2.4. Cuckoo Sandbox

Nesse trabalho, foi escolhido para estudo e adequação as necessidades da pesquisa o *framework* Cuckoo Sandbox que está sob licença GNU GPL.

O *Cuckoo Sandbox* foi iniciado em 2010 como um projeto do *Google Summer of Code* e posteriormente tornou-se um *framework* bastante consolidado, e de código aberto, com uma larga quantidade de contribuidores e funcionalidades. O *Cuckoo Sandbox* funciona de forma independente ao esquema de virtualização/emulação escolhido, podendo esse ser: QEMU, VirtualBox ou VMWare. Com ele é possível realizar a análise dinâmica e estática de diversos tipos de arquivos, como: .jar, .dll, .doc e outros.

O processo de análise no *Cuckoo Sandbox* começa após o envio de um arquivo malicioso para ele, durante esse processo é possível informar para o cuckoo alguns parâmetros, como por exemplo, o tempo limite para a execução do arquivo. Os dados são armazenados localmente e, assim que uma máquina virtual estiver disponível será realizado o processamento desse artefato.

Antes de carregar o analisador para a máquina virtual, juntamente com o arquivo malicioso, a máquina virtual é restaurada para um estado limpo. Na máquina virtual só é necessário executar o *script* de agente do *cuckoo* - que consiste basicamente em um servidor XMLRPC que recebe um pacote comprimido e executa o analisador contido nele.

A arquitetura modularizado do *framework* permite que quase todas as fases da análise de *malwares* sejam customizáveis. O módulo *Machinery* define como a solução de virtualização é gerenciada e como as máquinas virtuais são iniciadas, restauradas e paradas. Atualmente com suporte a QEMU, VirtualBox e VMWareThere - mas pode-se escrever novos módulos para dar suporte a novos *softwares*.

Os módulos de *packages* define como um determinado tipo de arquivo é executado depois de ser carregado para a máquina virtual. Um arquivo PE (.exe) pode ser iniciado diretamente, enquanto um documento do Word precisa ser carregado no Office.

Outro tipo de módulo no *Cuckoo* é a classe *auxiliary*. Estes módulos são executados simultaneamente com o processo de análise e não influenciam diretamente a instrumentação ou de registro. Um caso de uso para essa classe é o módulo *Human*. Ele emula o comportamento de um usuário, como o movimento do mouse e cliques do mouse, a fim de subverter técnicas anti-análise. A instrumentação real dos processos em execução atualmente é feita através da injeção de uma DLL que intercepta a cha-

mada de certas funções da API do Windows e registra seus parâmetros quando chamado [Bremer 2013].

2.5. Aprendizagem de máquina

Com o objetivo de validar a utilização de IOCs como subsídio para a detecção de artefatos maliciosos, este trabalho faz uso de algoritmos de aprendizado de máquina. Dentre as diversas definições, tem-se que aprendizado de máquina é a capacidade de melhorar a realização de determinadas atividades através da experiência [Mitchell 1997].

A abordagem de aprendizado de máquina está presente em diversos ramos de atuação, como jogos digitais, mineração e correlação de dados, reconhecimento de caracteres escritos, reconhecimento de linguagem natural, visão computacional, fraudes digitais, etc.

Dentre as possibilidades de aplicação e tecnologias dentro da disciplina de aprendizagem de máquina, podemos citar a presença de classificadores [Mitchell 1997]. Classificadores são utilizados, de forma supervisionada, para determinar em qual conjunto uma determinada entrada pode ser enquadrada.

Dentre os diversos algoritmos de classificação disponíveis, é possível citar: Random Forest, Árvores de Decisão e Naive Bayes.

3. Experimento

3.1. Fluxo de execução

O fluxo de execução do experimento se dá basicamente segundo as seguintes atividades:

- Configuração do ambiente de execução;
- Coleta de artefatos maliciosos;
- Coleta de artefatos não maliciosos;
- Execução dos artefatos;
- Extração dos IOCs e conversão para formato adequado;
- Treinamento dos algoritmos de aprendizagem de máquina;
- Teste e avaliacão dos resultados obtidos.

3.2. Configuração do ambiente de execução

O ambiente utilizado para a execução do presente experimento consiste em duas máquinas, sendo uma máquina *host* e uma máquina servidora. A máquina servidora é quem executa o *software* Cuckoo é composto de uma máquina real (não virtual) com as seguintes configurações:

- Sistema operacional - Ubuntu 14.05
- Processador - Intel(R) Xeon(R) CPU E3110 @ 3.00GHz
- Memória - 8GB
- Armazenamento
    1. 160 GB (SSD)
    2. 1.5 TB
    3. 240 GB

Nesta máquina está instalado o **Cuckoo Sandbox** versão 1.2-dev [Cuckoo 2014] e todas as dependências necessárias para a sua execução. Ademais, o sistema de virtualização escolhido para a execução dos teste, de acordo com a recomendação dos desenvolvedores do *Cuckoo Sandbox*, é o VirtualBox.

A máquina que executa o *malware*, aqui chamada de *host* é uma máquina virtual Windows XP sp2 com instalação padrão, tal qual recomendado pelos desenvolvedores do *Cuckoo Sandbox*. Para a realização do experimento, a fim de aumentar a janela de atuação, são adicionados os *softwares* da suíte Office, Flash e Java.

O procedimento de instalação e configuração da máquina servidora está além deste trabalho e pode ser realizado segundo o tutorial presenta no *site* do *Cuckoo Sandbox* [Cuckoo 2014]. Também não faz parte deste trabalho elencar o mecanismo de comunicação entre a máquina servidora e a máquina *host*, bem como técnicas de detecção do ambiente de virtualização ou *sandboxing*.

3.3. Coleta de artefatos

Para a realização do experimento faz-se necessário uma massa de artefatos a serem executados. Tais artefatos são binários nativos do Windows (PE32), tanto maliciosos quanto não maliciosos. A captura dos artefatos maliciosos foi realizada a partir do *site* http://www.virusshare.com, e os artefatos não maliciosos foram obtidos a partir de repositórios públicos, como *source forge*, *baixaki* e *superdownloads*.

3.3.1. Amostra

A amostra total é composta de 1403 artefatos, que se dividem em apenas duas partes: artefatos maliciosos e artefatos não maliciosos.

A parte da amostra que se refere aos artefatos maliciosos é composta por 1200 artefatos do tipo PE32 selecionados, de forma aleatória através de um *script* automatizado, a partir de um universo de 131000 distintos, baixado a partir do repositório *site* http://www.virusshare.com.

A segunda parte da amostra, é relativa aos artefatos conhecidamente não maliciosos, e se refere a 203 arquivos, também PE32 baixados a partir dos repositórios públicos *sourceforge*, baixaki e *superdownloads*.

Como o número de arquivos maliciosos é bem maior que o de arquivos não maliciosos é criado duas bases distintas:

- **Não-Balanceada**: Nessa base constam 1200 arquivos maliciosos e 203 não maliciosos.
- **Balanceada**: Já nessa base, a quantidade de amostras maliciosas e não maliciosas são idênticas, 203.

3.4. Execução dos artefatos

A execução dos artefatos se dá através da submissão submissão do artefatos ao *software* de *sandbox Cuckoo* a partir do *script submit.py*, disponível como utilitário do próprio *Cuckoo*. A fim de automatizar a submissão dos 1200 artefatos, foi criado um *script* que

monitora a quantidade de artefatos na fila de execução e mantém esse valor sempre em 5. Tal mecanismo foi desenvolvido para facilitar a continuação do experimento em caso de eventuais problemas durante a execução, como: falta de energia, indisponibilidade de rede, sobrecargas no sistema, etc.

Uma vez submetido o artefato ao *Cuckoo*, este restaura a máquina *host* para o *snapshot* onde ainda não foi executado nenhum artefato malicioso, e envia o artefato para a máquina *host*. Quando presente na máquina *host*, os sensores do *Cuckoo* são ligados e o artefato é executado. Após a execução, as informação dos sessores são transmitidas à máquina servidora que as recebe, realiza um *parsing* e as salva em banco de dados e sistema de arquivos (em formato JSON).

3.5. Extração dos IOCs e conversão para formato adequado

Munido com os dados brutos provenientes da execução dos artefatos, faz-se necessário convertê-los em um formato aceitos pelos algoritmos de classificação escolhidos para o presente trabalho, a saber: *Random Forest*, árvore de decisão e *Naive Bayes*. A implementação escolhida de tais algoritmos está presente no *Rapid Miner*.

3.6. Escolha de atributos relevantes

Diferente de outros trabalhos, onde o foco na escolha dos atributos se baseia em dados como *n-grams* como em [Andrade 2013], o presente trabalho utiliza como atributos presentes no vetor que será utilizado pelos algoritmos de classificação os seguintes dados:

- `http_gets;`
- `http_posts;`
- `bind;`
- `autorun;`
- `encrypted;`
- `api_calls;`
- `childrens;`
- `n_dropped;`
- `dll_count;`
- `peid_sig;`
- `n_proc;`
- `n_files;`
- `n_keys;`
- `n_mutexes;`
- `f_size;`
- `packer_upx;`
- `cuckoo_signs;`
- `n_drop_pe32;`
- `n_drop_data;`
- `enhanced;`
- `files_read;`
- `files_write;`
- `reg_read;`
- `reg_write;`
- `libs_loaded;`

- udp;
- irc;
- http;
- smtp;
- tcp;
- hosts;
- pcap_sha256;
- dns;
- domains;
- icmp;
- n_yara_dropped;
- n_yara_target;

O vetor submetido aos algoritmos de aprendizado de máquina é composto pelos 37 atributos supracitados, onde; http_gets representa a quantidade de requisições HTTP GET foram realizadas; http_posts representa a quantidade de requisições HTTP POST; bind é um booleano que indica se alguma porta local foi bindada ou não; autorun é um booleano que indica se o artefato se auto-instalou na inicialização do sistema; encrypted indica ser o artefato é cifrado ou não; api_calls total de chamadas a API do sistema; childrens quantidade de processos filhos criados; n_dropped indica o número de arquivos baixados durante a execução do artefato malicioso; dll_count indica o número de DLLs importadas pelo artefato; peid_sig quantidade de assinaturas de *packers* identificadas no cabeçalho do PE32; n_proc representa a quantidade de processos criados; n_files representa o número de arquivos criados, editados ou removidos; n_keys representa o número de chaves de registro criadas, alteradas ou removidas; n_mutexes indica o número de *mutexes* criados durante a execução do artefato; f_size representa o tamanho do artefato em *bytes*; packer_upx é um booleano que indica se o artefato foi *packeado* com o UPX; cuckoo_signs indica a quantidade de assinaturas do *cuckoo* identificadas, esse atributo é considerado como peso para os algorítimos de AM; n_drop_pe32 representa a quantidade de aquivos do tipo PE32 baixada pelo artefato; n_drop_data indica a quantidade de aquivos do tipo data baixada pelo artefato; enhanced indica o número de ações executadas pelo artefato, ações do tipo: bibliotecas carregadas, chaves do registro e arquivos criados, copiados, movidos e excluídos, *windowshook* criados, serviços criados ou removidos e etc; files_read e files_read representam, respectivamente, a quantidade arquivos lidos e escritos, assim como reg_read e reg_write; libs_loaded indica a quantidade total de bibliotecas carregadas em tempo de execução; udp, tcp, http, irc, smtp, dns e icmp representam a quantidade de pacotes interceptados para cada protocolo; hosts representa o total de endereços IPs externos interceptado nos protocolos de rede, e por sua vez domain representa os nomes; pcap_sha256 é usado como *flag* para determinar se o tráfego de rede foi interceptado com sucesso, terá valor igual a 64 em casos positivos e valor 0 em caso negativos; n_y_dropped representa o número de assinaturas YARA identificadas nos arquivos baixados e n_y_target boleano que indica se alguma assinatura YARA foi identificada no próprio artefato.

3.7. Treinamento dos algoritmos de aprendizagem de máquina

Nessa fase, técnicas de Aprendizado de Máquina são aplicadas nos arquivos de vetores de atributos para o aprendizado e avaliação de *malwares*. Como os dados foram representados em forma de vetores, diversos algoritmos de classificação podem ser escolhidos e comparados uns com os outros. Na Avaliação é verificado o desempenho da metodologia, levando-se em conta parâmetros como acurácia, falsos positivos e falsos negativos.

Para os testes e experimentos é utilizado a versão gratuita do software RapidMiner Studio 6.0.003, que fornece um ambiente integrado para a aprendizagem de máquina, mineração de dados, mineração de texto e análise preditiva.

A figura 1 apresenta o fluxo de processamento criado no *software* RapidMiner, para testar cada algorítimo.

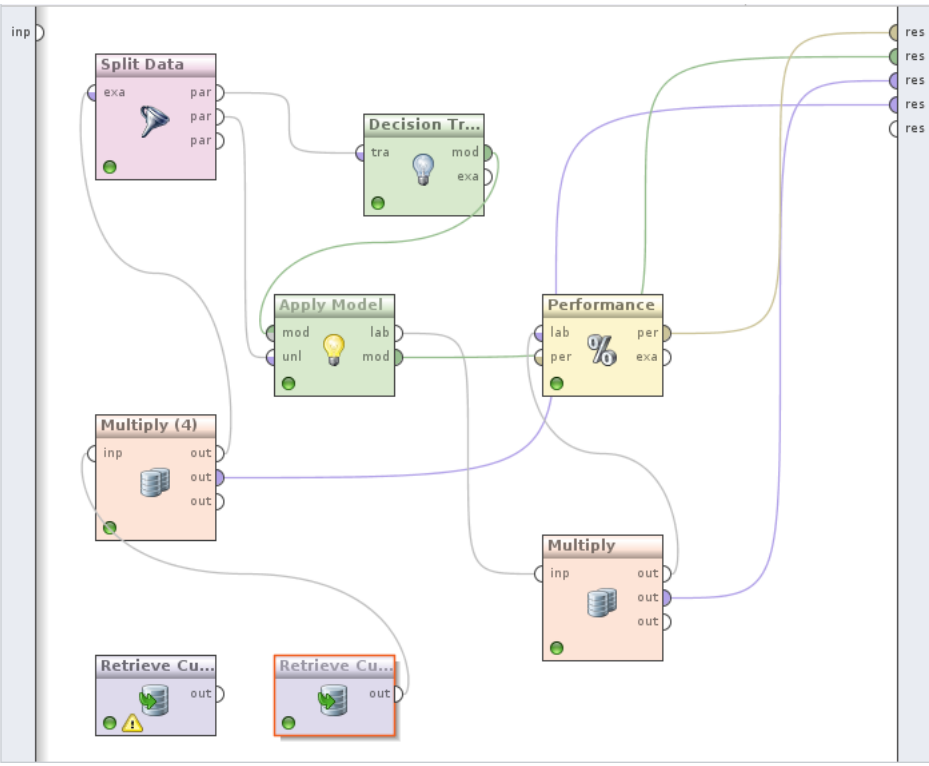

Figura 1. Tela da estrutura de processamento do RapidMiner

3.8. Resultados experimentais

Como explicado na seção 3.6, os dados são representados em forma de vetores, assim permitindo que diversos algoritmos de classificação possam ser escolhidos e comparados uns com os outros. Foram realizados 2 experimentos onde o experimento 01 utiliza a base de dados balanceada e o experimento 02 analisou o conjunto composto pela base de dados não balanceada, conforme descrito na seção 3.3.1.

Para cada experimento foram executados e comparados os seguintes algoritmos de aprendizado supervisionado:

- Naive Bayes

- Random Forest
- Decision Tree

A figura 2 representa uma árvore de decisão gerada no experimento 01.

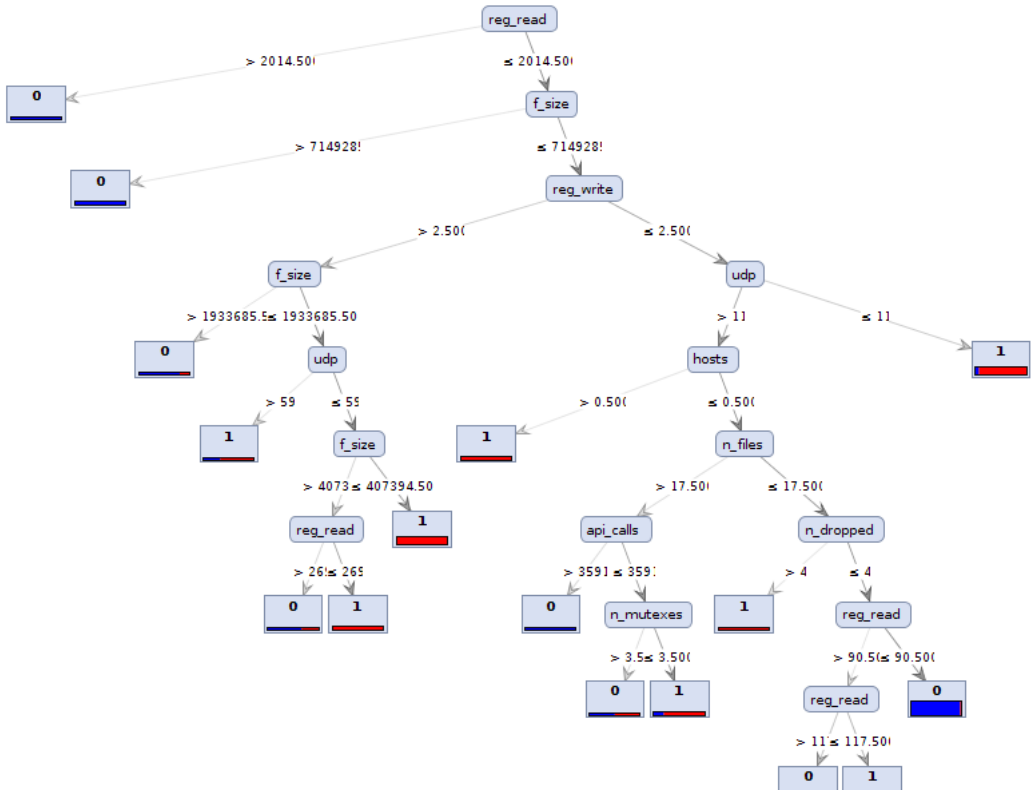

Figura 2. Árvore de decisão

A tabela 1 amostra os resultados obtidos no experimento 01 através do método proposto.

Tabela 1. Comparação dos classificadores - Experimento 01

	Acurácia	Falso Positivo	Falso negativo
Decision Tree	88,48%	7,90%	12,36%
Naive Bayes	84,33%	35,9%	11,24%
Random Forest	72,35%	7,69%	32,02%

Nesse experimento, os algoritmos de melhor desempenho foram *Decision Tree* e *Naive Bayes* com **88,48%** e **84,33%** de acurácia, respectivamente, e com baixos indices de falsos positivos.

A tabela 2 amostra os resultados obtidos no experimento 02 através do método proposto.

Nesse experimento, os algoritmos de melhor desempenho foram *Random Forest* e *Decision Tree* com **96,25%** e **95,53%** de acurácia, respectivamente, e com baixos índices de falsos positivos.

Tabela 2. Comparação dos classificadores - Experimento 02

	Acurácia	Falso Positivo	Falso negativo
Decision Tree	95,53%	47,37%	2,9%
Naive Bayes	54,43%	18,42%	46,57%
Random Forest	96,25%	100%	0%

4. Conclusões

Podemos verificar o algoritmo *Random Forest* apresenta o melhor resultado geral. Ao analisar os números de falsos positivos e falsos negativos gerados por esse algorítimo no experimento 02, percebe-se claramente que o algorítimo prediz todos os dados como positivo. Ainda no experimento 02 observa-se que *Naive Bayes* obteve uma taxa de precisão baixa e um alto índice de falso negativos. No experimento 01 tem-se uma taxa de precisão relativamente alta nos três algorítimos testados, destacando-se os baixos índices de falso negativos para os algorítimos *Naive Bayes* e *Decision Tree*. Em linha gerais o algorítimo ***Decision Tree*** demostrou-se mais estável e eficiente em ambos os experimentos.

Diante do evolução dos *malwares* e da ineficácia das técnicas clássicas utilizadas contra essas ameaças, este trabalho propõe criar um eco sistema integrado de mecanismos já existentes – análise dinâmica e estática, assinaturas, aprendizado de máquina – que permita a análise de *malware* de forma automática e com um grau satisfatório de precisão. Com a quantidade de artefatos analisados, foi obtida taxa de precisão acima de 90%.

Com o objetivo de melhorar o desempenho da metodologia é possível selecionar e criar novos atributos, além de remover alguns que demonstraram-se irrelevantes para o experimento, também é possível aumentar o grau de automatização da extração desses atributos, escrevendo um modulo de relatório para o *Cuckoo*. Ainda com o objetivo de melhorar a metodologia, é possível tratar as técnicas de anti-virtualização e *sleeping*.

Referências

Andrade, C. A. B. (2013). Análise automática de malwares utilizando as têcnicas de sandbox e apendizado de máquinas. Dissertação de mestrado - Instituto Militar de Engenharia (IME).

Arcoverde, H. F. (2013). Malwares brasileiros: técnicas, alvos e tendências. Dissertaçãoo de mestrado - Centro de Informática UFPE.

Bremer, J. (2013). Cuckoo Sandbox workshop at Blackhat US 2013. https://media.blackhat.com/us-13/US-13-Bremer-Mo-Malware-Mo-Problems-Cuckoo-Sandbox-WP.pdf. [Acesso em: 21 abril 2014].

Cuckoo, F. (2014). Cuckoo sandbox book. http://docs.cuckoosandbox.org/en/latest/. [Acesso em: 03 mar. 2014].

FEBRABAN (2011). Perdas com fraudes eletrônicas aumentam 36% no primeiro semestre de 2011. http://www.febraban.org.br/Noticias1.asp?id_texto=1321. [Acesso em: 20 maio 2014].

Goodin, D. (2014). Antivirus pioneer Symantec declares AV is dead and doomed to failure. http://arstechnica.com/security/2014/05/antivurus-pioneer-symantec-declares-av-dead-and-doomed-to-failure/. [Acesso em: 10 maio 2014].

Gragido, W. (2012). Understanding Indicators of Compromise (IOC) Part I. http://blogs.rsa.com/understanding-indicators-of-compromise-ioc-part-i/. [Acesso em: 15 abr. 2014].

Ligh, M., Adair, S., Hartstein, B., and Richard, M. (2010). Tools and techniques for fighting malicious code. In *Malware Analyst's Cookbook and DVD*. Wiley.

Mitchell, T. M. (1997). In *Machine Learning*. McGraw-Hill Science/Engineering/Math.

OpenIOC (2014). An Open Framework for Sharing Threat Intelligence. http://www.openioc.org/. [Acesso em: 10 maio 2014].

Siddiqui, M. A. (2008). Data mining methods for malware detection. http://etd.fcla.edu/CF/CFE0002303/Siddiqui_Muazzam_A_200808_PhD.pdf. [Acesso em: 02 fev. 2014].

Sikorski, M. and Honig, A. (2012). The hands-on guide to dissecting malicious software. In *Practical Malware Analysis*. No Starch Press.

Szor, P. (2005). Virus research and defense. In *The art of computer*. Addison-Wesley.

Yadron, D. (2014). Symantec Develops New Attack on Cyberhacking: Declaring Antivirus Software Dead, Firm Turns to Minimizing Damage From Breaches. http://online.wsj.com/news/articles/SB10001424052702303417104579542140235850578. [Acesso em: 10 maio 2014].