Universidade Tecnológica Federal do Paraná
Departamento Acadêmico de Informática – DAINF
Curso: Engenharia de Computação
Disciplina: Segurança Computacional

Artigo: Identificação de Códigos Maliciosos Metamórficos pela Medição do Nível de Similaridade de Grafos de Dependência

AUTORES: GILBERTO B. MARTINS, PAULO DOS SANTOS, VITOR DANRLEY, EDUARDO SOUTO, ROSIANE DE FREITAS

DISCENTE: MAICON PAULO ASSMANN

DOCENTE: BRUNO CÉSAR RIBAS

Pato Branco
2017

Códigos maliciosos

- Códigos maliciosos/*malwares*:
  - É um arquivo ou código geralmente transmitido através da rede que infecta, explora, rouba ou realiza virtualmente qualquer comportamento que o invasor deseje.
- Tipos de detecção:
  - Extração de trechos de códigos de *malwares* (assinaturas).
  - Utilização de autômatos finitos.
  - A normalização do código e o levantamento do fluxo para reversão e identificação o uso de funções.
  - A utilização de grafos para modelar o uso de funções.
  - A relação de dependência entre instruções do código.
- Tendo 2 componentes furtivos:
  - Um Motor criptográfico.
  - Um corpo criptográfico do *malware*.

Motor criptográfico

- São duas responsabilidades principais:
- Restauração do corpo criptográfico ao seu estado original.
  - O *malware* possa agir de acordo com os objetos para qual foi construído.
- A criação de um novo corpo criptográfico.
  - Baseado numa chave aleatória.
  - Ocorre o processo de propagação.
- Conjunto de motores criptográficos.
  - Múltiplos motores criptográficos.
  - Ofuscação.
  - Metamorfismo.

Metamorfismo

- Técnicas de simples de metamorfismo:
  - 1) A inserção de instruções e variáveis irrelevantes, também conhecida como inserção de código lixo.
    - Não altera a lógica original do programa.
  - 2) Alteração no nome de variáveis
    - Troca mútua de variáveis entre instruções.
  - 3) Substituição de sequência de instruções por outra que produzam o mesmo resultado.
  - 4) Alteração na ordem de execução das instruções.
    - Seja pelo reposicionamento de blocos de código independente.
    - Pelo uso de instruções de desvio de fluxo.

Grafos de dependência

- Existe diferentes abordagem:
  - Entre ela os de grafos de dependência.
- Lidam com o Problema NP-Dificil.
  - Tem solução 100% certa.
    - Tempo exponencial.
  - Na identificação de similaridade entre os grafos.
    - Métricas para medir o nível de similaridade
  - Pode ser usado um algoritmo genético.
    - Para não ter um tempo muito alto de execução.

Algoritmo 1 Floyd-Warshall Modificado

```
1: for each v_k in G
2:   if v_k isn't a decision vertex
3:     for each v_o
4:       for each v_d
5:         if v_o isn't a decision vertex
6:           if (d(v_o,v_k) + d(v_k,v_d)) < d(v_o,v_d)
8:             set d(v_o,v_d)=(d(v_o,v_k) + d(v_k,v_d))
9:         else
10:          if (d(v_k,v_o) + d(v_k,v_d)) < d(v_o,v_d)
11:            set d(v_o,v_d)=(d(v_k,v_o) + d(v_k,v_d))
```

Algoritmo 1: Floyd-Warshall Modificado.

Modos de construção do grafos de dependência

- Forma de modelar, sendo uma dela do seguinte modo:
  - Cada vértice v_i está associado a uma função.
  - Uma aresta $v_a v_b$ é criada sempre que o corpo da função v_a existe uma chamada para a função v_b.
  - Extração de grafos de referencia de *malware* grande.
- Utilização de grafos que modelam chamadas biblioteca de APIs.
  - Procedimentos e Funções para identificação.
  - Chamadas de bibliotecas externas.
  - Chamadas de função de sistemas operacionais.

Construção do grafos de dependência do artigo

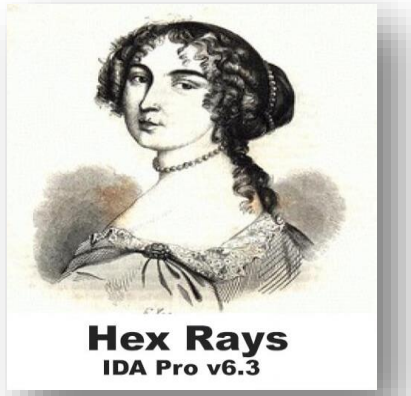
- Metodologia de identificação de códigos metamórficos Executáveis.
  - Baseada na utilização de grafos de dependência.
  - Tem 4 etapas principais de processo de obtenção do grafo.

- 1) **Reconstrução do código *assembly*:** Onde o programa executável passa por um processo de engenharia reversa para obtenção do código.

```
linha:01 .686p
linha:02 .model flat
linha:03    push eax
linha:04    push ebx
linha:05    push ecx
linha:06    call sub_01
linha:07 ini_loop:
linha:08    cmp ebx, eax
linha:09    jg  end_loop
linha:10    call sub_02
linha:11    jmp ini_loop
linha:12 end_loop:
linha:13    call sub_03
linha:14    end
linha:15 sub_01 proc near
linha:16    mov eax, 9
linha:17    mov ebx, 3
linha:18    mov ecx, 0
linha:19 retn
linha:20 sub_01 endp
linha:21 sub_02 proc near
linha:22     add ecx, 1
linha:23     sub eax, ebx
linha:24 retn
linha:25 sub_02 endp
linha:26 sub_03 proc near
linha:27    pop ecx
linha:28    pop ebx
linha:29    pop eax
linha:30 retn
linha:31 sub_03 endp
```

Figura 1: Código *assembly*.

- Auxilio de programas como:
  - OllyDbg.
  - IDA Pro.

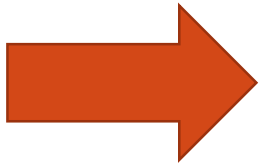


Construção do grafos de dependência do artigo

- **2) Geração do grafo de dependência:** Onde o programa código em *assembly* é usado para geração do grafo.

```
linha:01 .686p
linha:02 .model flat
linha:03    push eax
linha:04    push ebx
linha:05    push ecx
linha:06    call sub_01
linha:07 ini_loop:
linha:08    cmp ebx, eax
linha:09    jg  end_loop
linha:10    call sub_02
linha:11    jmp ini_loop
linha:12 end_loop:
linha:13    call sub_03
linha:14    end
linha:15 sub_01 proc near
linha:16    mov eax, 9
linha:17    mov ebx, 3
linha:18    mov ecx, 0
linha:19 retn
linha:20 sub_01 endp
linha:21 sub_02 proc near
linha:22     add ecx, 1
linha:23     sub eax, ebx
linha:24 retn
linha:25 sub_02 endp
linha:26 sub_03 proc near
linha:27    pop ecx
linha:28    pop ebx
linha:29    pop eax
linha:30 retn
linha:31 sub_03 endp
```

Figura 2: Código *assembly*.



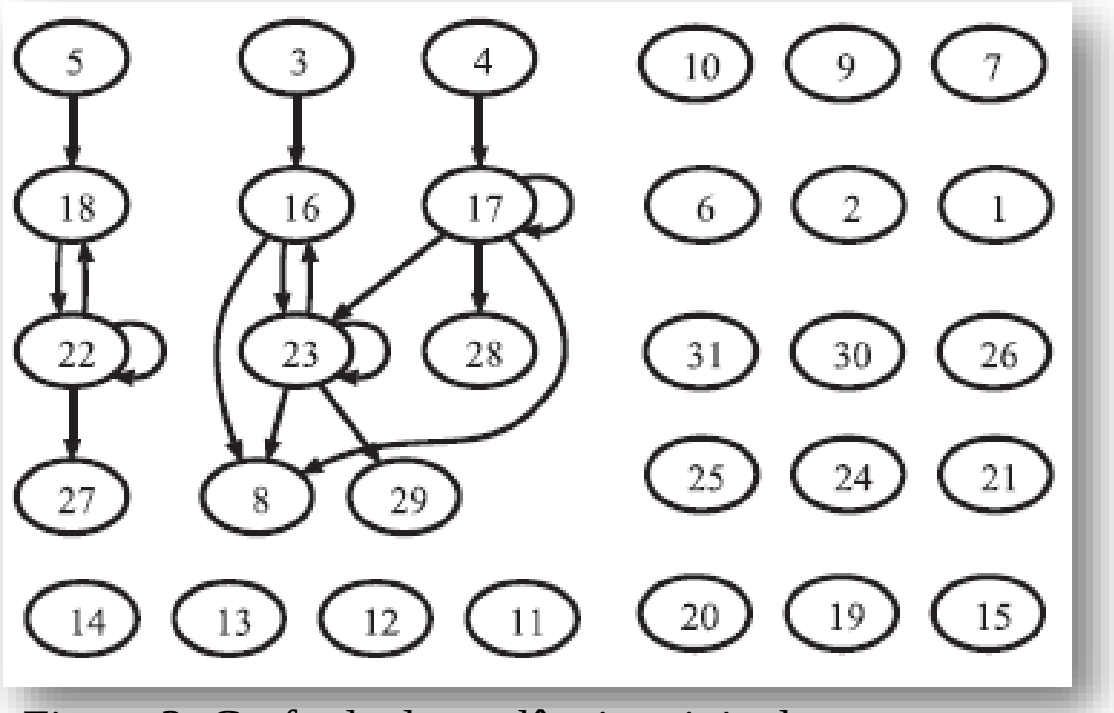


Figura 3: Grafo de dependência original.

Construção do grafos de dependência do artigo

- **3) Redução do grafo:** Utilizado o grafo de dependência obtido.
  - Partes do código/grafo onde o controle de fluxo nunca ira passar são removidos.

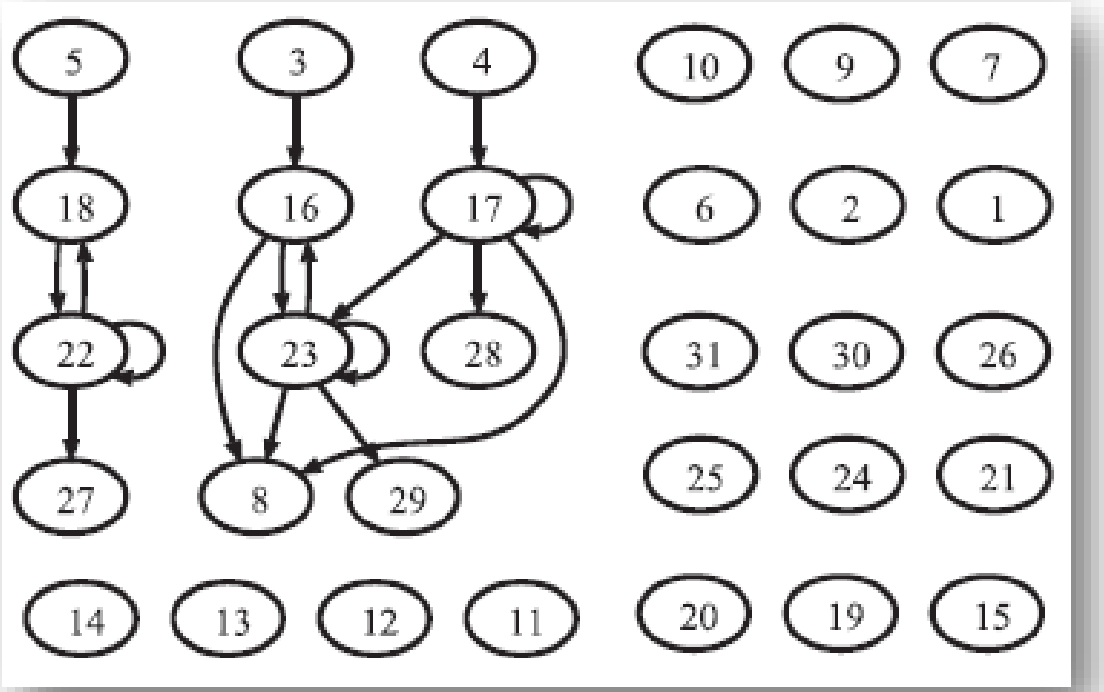


Figura 4: Grafo de dependência original.

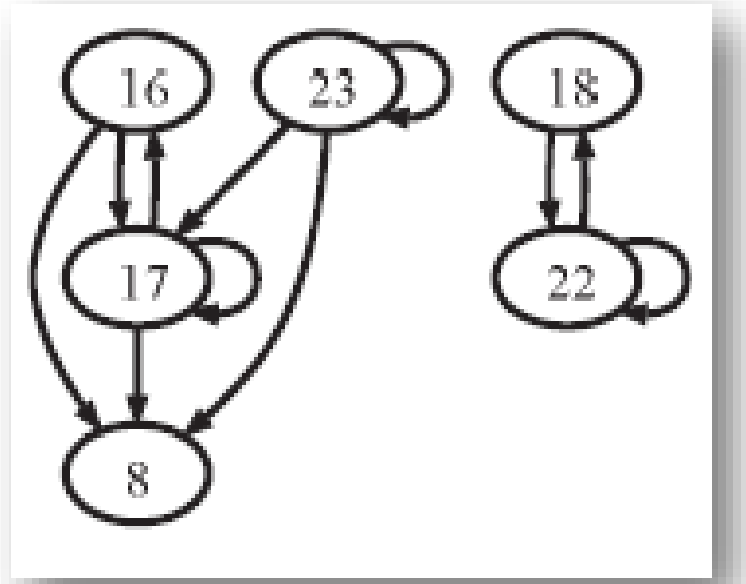


Figura 5: Grafo de dependência reduzido.

- **4) Comparação do grafo reduzido obtido o código:** Com um grafo correspondente a um *malware* previamente analisado.

Diferenciação dos vértices em grafos de dependência

- Os vértices são classificados em 3 categorias:
  - **1) Os vértices de partida** : Dizem respeito ao inicio da cadeia de dependência associadas à manipulação de uma variável ou registrador.

Vértice de partida

    - Não tem arestas entrando do vértice.
  - **2) Os vértices de processamento** : Estão associados a qualquer instrução que manipula e alteram o conteúdo dos registradores e variáveis presente no códigos.

Vértice de processamento

    - Tem tanto arestas saindo do vértice como entrando nele.
  - **3) Os vértices de Decisão** : São aquele gerados a partir das instruções CMP, utilizadas para avaliar e comparar o conteúdo de registradores e variáveis.

Vértice de decisão

    - Não tem arestas saindo do vértice, logo seria o vértice final/decisão.

Identificar os vértices de decisão

- É onde está as instruções que serão realmente será ou não executado o código.
  - O processo para idêntica vértices mais relevantes de decisão, são resumidos em 4 etapas.

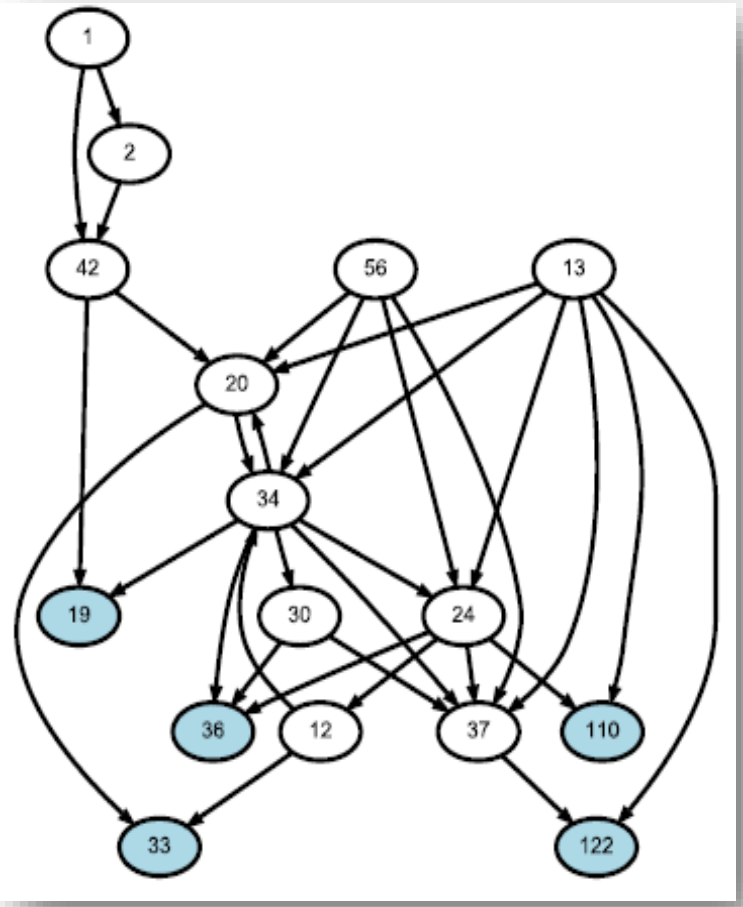


Figura 6: Grafo de dependência reduzido, com os vértices de decisão em destaque.

- 1) Cálculo de menor distancia relativa entre cada vértice de decisão.
- 2) Construção de um grafo virtual derivado, constituído apenas dos vértices de decisão, com as arestas representado a distancia relativa entre cada vértice.
- 3) Cálculo da clique máxima presente neste grafo virtual derivado.
- 4) Redução final do grafo de dependência, com a eleiminação de qualquer vértice e aresta que não estejam associados aos vértices de decisão presente na clique máximo do grafo virtual derivado.

Comparação dos grafos de dependência

- A comparação entre vértices de dois grafos, G_1 e G_2, que satisfação um conjunto de restrições do problema NP-Dificil.
  - Utilizado algoritmos genéticos.

Descrição	Equação
Definições iniciais	$G_1=(V_1, E_1), G_2=(V_2, E_2)$ e $\|V_1\| < \|V_2\|$
Detalhamento dos vértices	$V=V_d, \cup V_p$
Subgrafo G'	$G'=(V', E') \mid G' \subset G, V' \subset V, E' \subset E$
Subgrafos comparados	$G_1' \subset G_1, G_2' \subset G_2 \mid \|V_{d1}'\| = \|V_{d2}'\|, \|V_{p1}'\| = \|V_{p2}'\|$
Função de busca de uma aresta e em um conjunto de arestas E	$I(e,E) = \begin{cases} 1, & se \quad e \in E \\ 0, & caso \quad contrário \end{cases}$
Cálculo da similaridade entre G_1 e G_2	$similaridade(G_1', G_2') = \frac{\sum_{e \in E_1'} I(e, E_2') + \sum_{e \in E_2'} I(e, E_1')}{\|E_1'\| + \|E_2'\|}$

Tabela 1: Equações para realizar o cálculo da similaridade entre grafos.

- O conjunto de vértices V em cada grafo tratado é constituído por dois grupos:
  - 1) V_d,o subconjunto de V formado apenas por vértices de decisão.
  - 2) V_p,o subconjunto de V formado apenas por vértices de processamento.

Segmentação por tipo do vértice

- Os grafos que forem comparados sejam separados em dois subconjuntos.
  - No primeiro, ficam os nós de decisão e no segundo, os nó de processamento e nós de partida.

a) Disposição de vértices no cromossomo, sem o processo de segmentação

1	2	12	13	19	20	24	30	33	34	36	37	42	56	110	122

b) Disposição de vértices no cromossomo, com o processo de segmentação

19	33	36	110	122	1	2	12	13	20	24	30	34	37	42	56

Figura 7: Comparação entre a disposição de vértices em cromossomos sem segmentação e com o uso de segmentação.

Comparação segmentada com restrição de escopo

- Empregado as mesmas estratégias genéticas/algoritmo de:
  - População inicial.
  - Vértices de processamento serão ranqueados, por sua ligação com vértices de decisão.
- Este processo de construção da população inicial é dividia em 4 estapas:
  - 1) Os vértices de decisão são inseridos nas primeiras posições do cromossomo.
  - 2) A lista de vértices de processamento ligados diretamente a cada um dos vértices de decisão é recuperada.
  - 3) Os primeiros vértices de processamento inseridos tem suas listas de vértices adjacentes consultadas.
    - Este processo se repete até que todos os vértices de processamento presente no cromossomo sejam avaliados.
    - Tenham sido inseridos na etapa 2.

Comparação segmentada com restrição de escopo

- 4) Nesta etapa se encarrega de inserir todos os vértices que não tenham sido inseridos nas etapas anteriores, sendo divididos em 2 tipos de vértices
  - a) Vértice de partida, que não podem ser atingidos pelo processo de construção do cromossomo descrito na etapa anterior.
    - Pois estes vértices só possuem arestas originadas a partir deles, não existe qualquer caminho no grafo que leve até eles.
  - b) Vértices de processamento que só podem ser atingidos a parti de vértices de partida.

a) Disposição de vértices no cromossomo, com o processo de segmentação

19	33	36	110	122	1	2	12	13	20	24	30	34	37	42	56

b) Disposição de vértices no cromossomo, com o processo restrição de escopo

19	33	36	110	122	34	42	12	20	24	30	13	37	1	2	56

Figura 8: Comparação entre a disposição de vértices em cromossomos com segmentação e com restrição de escopo.

Resultados

- Empregada um a base de dados sintética como os valores de MCS (maior subgrafo comum).
  - Execução por algoritmo genético de comparação.

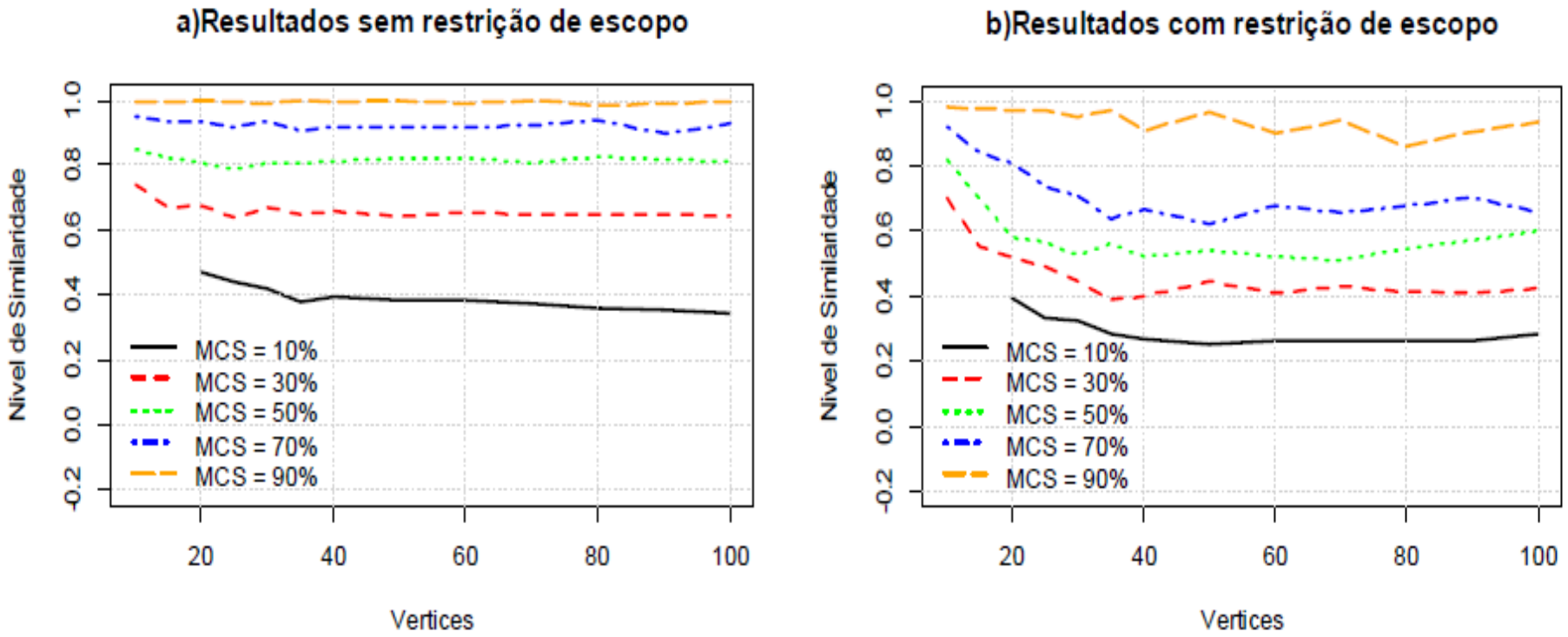


Figura 8: Gráfico de dispersão relacionando as médias das diferenças com os valores do MCS e da quantidade de vértices.

Resultados

- Empregada em vírus reais, com a base de dados sintéticas.
  - Em ciclos de 0 a 100 comparações.

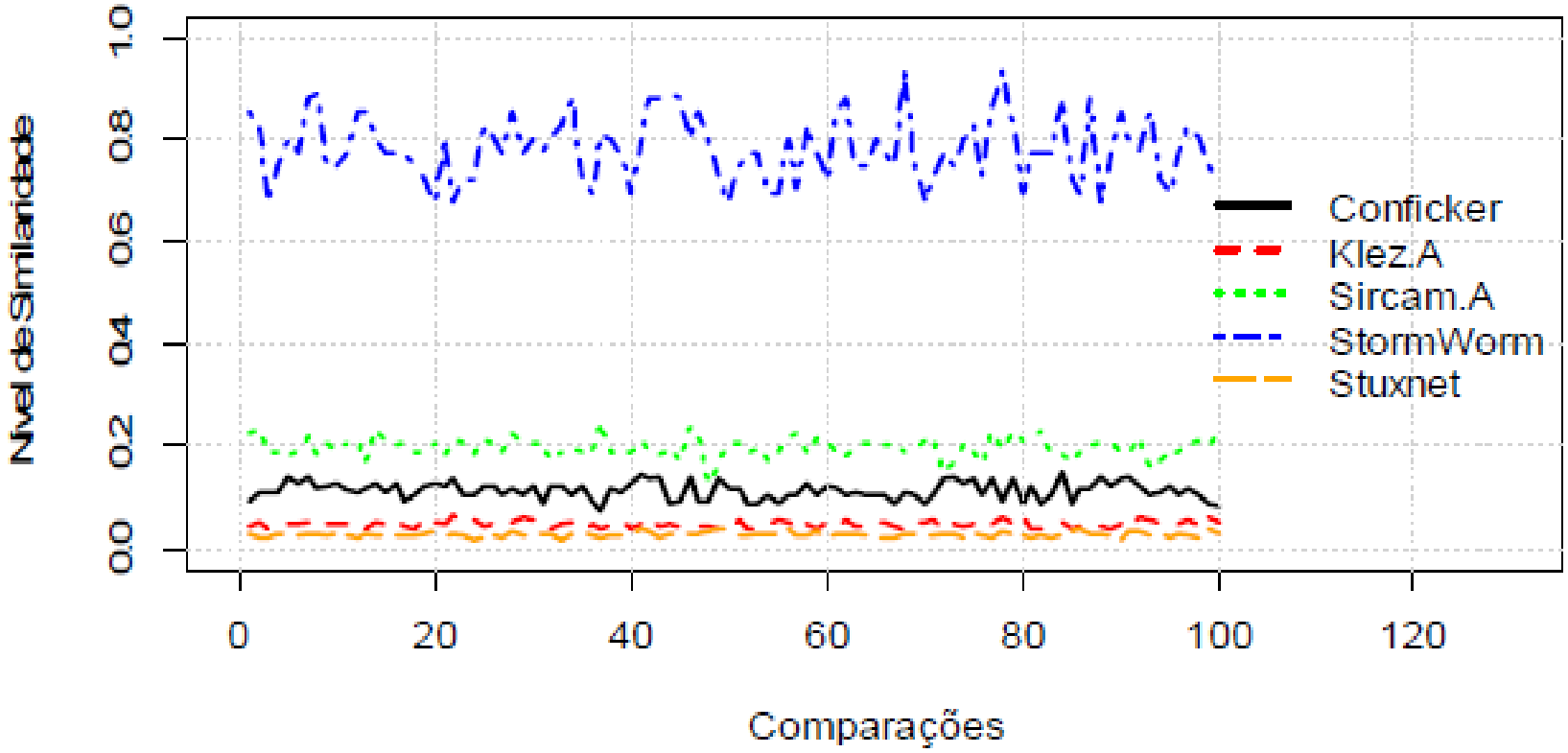


Figura 9: Resultados da comparação dos grafos de referência com suas versões não reduzidas.

Resultados

- Submissão de algumas amostras metamórficas à ferramenta Vírus Total.
  - Analisa vários tipos de *malwares* possível.

Malware Base	Vértices no Grafo de Referência	Média de Vértices nas Amostras Metamórficas	Taxa de Identificações com Sucesso
Conficker	42	60,51	98%
Klez.A	120	99,97	100%
Sircam.A	28	33	99%
StormWorm	23	24	100%
Stuxnet	184	332,93	97%

Tabela 2: Resumo dos resultados de testes de identificação de versões metamórficas de *malware*.

Malware Base	Amostras	Referência	Média de Vértices nas Amostras	Taxa de identificações Errôneas
Conficker	Klez.A	42	99,97	0%
Conficker	Stuxnet	42	332,93	0%
Sircam.A	Klez.A	28	99,97	0%
Sircam.A	Stuxnet	28	332,93	0%
StormWorm	Conficker	23	60,51	0%
StormWorm	Klez.A	23	99,97	0%
StormWorm	Sircam.A	23	33	0%
StormWorm	Stuxnet	23	332,93	0%

Tabela 3: Resumo dos resultados de testes de falsos positivos.

Malware									
Conficker		Klez.A		Sircam.A		StormWorm		Stuxnet	
Identificações	Testes	Identificações	Testes	Identificações	Testes	Identificações	Testes	Identificações	Testes
37	56	33	55	37	56	38	55	40	56
36	55	31	55	37	56	40	55	41	56
34	55	28	53	35	56	38	56	38	55
37	55	27	55	36	55	40	56	37	55

Tabela 4: Resumo dos testes com a ferramenta Vírus Total.

Referências

Martins, G., Santos, P., Danrley, V., Souto, E., De and Freitas, R. (2016). Identificação de Código Maliciosos Metamórficos pela Medição do Nível se Similaridade de Grafos de Dependência. XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais – SBSeg, 2016.

Martins, G., Souto, E., Freitas, R. De and Feitosa, E. (2014). Estruturas Virtuais e Diferenciação de Vértices em Grafos de Dependência para detecção de Malwares Metamórfico. XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais – SBSeg, 2014.

Martins, G., Souto, E., De and Freitas, R.. Identificação de Malware Metamórfico baseado em Grafos de Dependência. Tese (Doutorado), Universidade Federal do Amazonas – 2017.