Breve Comparação de Ferramentas para Análise Estática de Código Malicioso

William Akihiro Alves Aisawa[1],
Pedro Henrique Lara Campos[1],
Cesar Augusto Cavalheiro Marcondes[2]
Paulo Matias[1]

[1] Departamento de Computação
Universidade Federal de São Carlos (UFSCar) – São Carlos, SP – Brasil

alveswill@hotmail.com,root@pedrohlc.com,matias@ufscar.br

[2]Divisão de Ciência da Computação
Instituto Tecnológico de Aeronáutica (ITA) – São José dos Campos, SP – Brasil

cmarcondes@ita.br

Abstract. *This article briefly compares IDA Pro and Ghidra, tools for static analysis of software through reverse engineering, focusing on their utility for preventing and counteracting malware, analyzing the code, understanding its operating logic, algorithms, and specifications without risking exposure to the execution of malicious code. Both tools are excellent for static analysis of malware, even Ghidra being a newly released tool.*

Resumo. *Neste artigo, fazemos uma breve comparação entre o IDA Pro e o Ghidra, ferramentas de análise estática por engenharia reversa, focando em sua utilidade para prevenir e contra-atacar malware, analisando os códigos, entendendo sua lógica de funcionamento, os algoritmos e especificações, sem arriscar-se à exposição da execução do código malicioso. Ambas as ferramentas são excelentes para análise estática de malware, mesmo Ghidra sendo uma ferramenta recentemente lançada.*

1. Introdução

Existem grandes indústrias comerciais que giram em torno da criação de *malwares* e *anti-malwares* [Gutmann 2007] e, atualmente, *malwares* formam a parcela mais abundante de diversas categorias de ameças ao ambiente computacional [Zolkipli and Jantan 2010]. Esse fato deve-se ao aprimoramento das técnicas empregadas pelos cibercriminosos, dando origem a diversas categorias de *malware*, que utilizam variados vetores de infecção e apresentam diferentes formas de comportamento errático.

Um dos grandes desafios na análise de *malware* está relacionado à coleta de dados úteis sem arriscar o ambiente computacional usado na análise [Burji et al. 2010]. Precisa-se entender a lógica de funcionamento, algoritmos e especificações sem que o código seja acionado e prejudique o analista. Visando esse tipo de tarefa, existem ferramentas que auxiliam na engenharia reversa, análise e compreensão. Este artigo realiza um breve

estudo comparando duas dessas ferramentas: o **IDA Pro**[1], desenvolvido pela Hex-Rays, e o **Ghidra**[2], recentemente disponibilizado pela *National Security Agency* (NSA).

O artigo está estruturado em três seções. Esta introdução é seguida da Seção 2, que apresenta um resumo de cada uma das ferramentas comparadas e os resultados obtidos na execução de um experimento. Por fim, a Seção 3 apresenta a conclusão do trabalho.

2. Estudo Comparativo e Conceitual das Ferramentas

2.1. Principais Características

As principais características das ferramentas propostas são comparadas na Tabela 1 e introduzidas nos parágrafos seguintes.

	IDA Pro + Hex-Rays	Ghidra
Disassembler	✓	✓
Descompilador	✓	✓
Sistemas Operacionais	Linux, Mac OS X, Microsoft Windows	Linux, Mac OS X, Microsoft Windows
Código Aberto	✗	✓
Grafo de Controle de Fluxo	✓	✓
Extensões	✓	✓
Depurador	✓	✗
Depurador Remoto	✓	✗
Trabalho Colaborativo	✗	✓
Suporte nativo a arquiteturas	60	20

Tabela 1. Comparação entre IDA Pro e Ghidra

O IDA Pro é uma das ferramentas de engenharia reversa mais conhecidas no mercado. Trata-se de um *disassembler* interativo, amplamente utilizado para a reversão de software. Escrito totalmente em C++, é multiplataforma (Linux, Mac OS X e Microsoft Windows) e também possui um grande número de extensões que permitem ampliar suas funcionalidades, como o Hex-Rays Decompiler [Hex-Rays 2019], fornecido pelo seu próprio fabricante.

Ghidra, por sua vez, é um software *open source* de engenharia reversa desenvolvido em Java pelo departamento de pesquisas da NSA. Essa ferramenta também é multiplataforma (oficialmente, suporta Linux, Mac OS X e Microsoft Windows) e ajuda a analisar

[1] `https://www.hex-rays.com/products/ida/index.shtml`
[2] `https://www.nsa.gov/resources/everyone/ghidra`

códigos maliciosos diversos, podendo fornecer aos profissionais uma melhor compreensão das vulnerabilidades potenciais em suas redes e sistemas [National Security Agency 2019].

Disassembler: O IDA conta com suporte a cerca de 60 tipos de arquitetura de microprocessadores, embora sua versão *starter* possua suporte somente a endereçamento de 32-bits. O Ghidra suporta nativamente apenas 20 tipos de arquiteturas. No IDA, pode-se acrescentar suporte a novos conjuntos de instruções por meio de extensões escritas em C ou Python. Já o Ghidra facilita essa tarefa oferecendo uma linguagem de propósito específico denominada Sleigh.

Descompilador: O Ghidra é capaz de descompilar código de qualquer arquitetura para a qual tenha sido adicionado suporte ao *disassembler* por meio da linguagem Sleigh. Já o Hex-Rays Decompiler do IDA só descompila código de algumas arquiteturas explicitamente suportadas pelo fabricante (atualmente, x86, x64, ARM32, ARM64 e PowerPC) e não torna possível que o usuário adicione suporte a uma nova arquitetura.

Extensões: Ambos permitem que extensões sejam desenvolvidas em Python. A implementação de Python utilizada pelo Ghidra é o Jython, que fornece acesso à hierarquia de classes completa do Ghidra, exceto pelo descompilador, que é desenvolvido em código nativo. Apesar disso, pode-se interagir com o descompilador por meio da API.

Depurador: Ao passo que o Ghidra não funciona como depurador, o IDA conta com um depurador versátil, com suporte a múltiplos alvos de depuração e que pode manipular arquivos remotamente por meio de um servidor remoto de depuração. O servidor pode ser executado em uma plataforma diferente da plataforma em que o IDA está sendo executado, tornando possível, por exemplo, depurar o *kernel* Linux de um dispositivo embarcado a partir de um computador com Windows. O depurador do IDA disponibiliza uma API que pode ser manipulada por extensões e *scripts*.

Versionamento e trabalho colaborativo: O Ghidra conta com recursos de versionamento de projeto e trabalho colaborativo, funcionalidades não disponíveis no IDA.

Custo: O Ghidra está disponível para *download* sem custo algum em seu site oficial e seu código fonte está disponível em um repositório no GitHub[3]. Já a versão mais barata do IDA custa em torno de 979 dólares, podendo chegar perto dos 20.000 dólares, uma vez que os descompiladores são vendidos separadamente. A Hex-Rays disponibiliza gratuitamente para uso não-comercial uma versão antiga (atualmente 7.0) do IDA, porém sem descompilador, sem suporte a diversas arquiteturas e formatos de executável e sem suporte a extensões em Python.

Outros recursos: O Ghidra sincroniza o cursor posicionado sobre a saída do descompilador com o cursor na saída do *disassembler*. No entanto, uma funcionalidade similar pode ser obtida no IDA Pro com a extensão HexRaysCodeXplorer[4].

2.2. Breve Comparação Prática

Para comparar a qualidade do código descompilado, uma mesma função de um *malware* foi fornecida a ambas as ferramentas (Figura 1). O malware utilizado possui o resumo (*hash*) `551b48e425dcf4337ee023ad65a871123d172e43fabbc965252f5a2e69d0bd4a`.

[3] `https://github.com/NationalSecurityAgency/ghidra`
[4] `https://github.com/REhints/HexRaysCodeXplorer`

```
/*Código descompilado pelo IDA Pro*/
_BOOL4 __cdecl
CUtility::IpIsLan(CUtility *this)
{
  unsigned int v1; //eax
  unsigned int v2; //eax
  unsigned int v3; //eax
  unsigned int v4; //eax
  unsigned int v5; //eax
  unsigned int v6; //eax
  unsigned int v7; //eax
  unsigned int v8; //eax
  //[esp-8h][ebp-10h]
  unsigned int v10;
  //[esp-8h][ebp-10h]
  unsigned int v11;
  //[esp-8h][ebp-10h]
  unsigned int v12;
  //[esp-8h][ebp-10h]
  unsigned int v13;
  //[esp-8h][ebp-10h]
  unsigned int v14;
  bool v16;//[esp+4h][ebp-4h]
  bool v17;//[esp+5h][ebp-3h]
  bool v18;//[esp+6h][ebp-2h]
  bool v19;//[esp+7h][ebp-1h]

  if ( !this )
    return 1;
  v1 = inet_addr("10.0.0.0");
  v16 =
  CUtility::IpCompare(this, v1, v10) < 0
  && (v2 = inet_addr("10.255.255.255"),
  CUtility::IpCompare(this, v2, v11)>0)
  if ( v16 )
    return 1;
}
```

```
/*Código descompilado pelo Ghidra*/
uint IpIsLan(uint param_1)
{
  byte bVar1;
  bool bVar2;
  in_addr_t iVar3;
  int iVar4;
  uint local_c;

  if (param_1 == 0) {
    return 1;
  }
  iVar3 = inet_addr("10.0.0.0");
  iVar4 = IpCompare(param_1,iVar3);
  if (iVar4 < 0) {
    iVar3 = inet_addr("10.255.255.255");
    iVar4 = IpCompare(param_1,iVar3);
    if (iVar4 < 1) goto LAB_0808ec2a;
    bVar2 = true;
  }
  else {
LAB_0808ec2a:
    bVar2 = false;
  }
  if (bVar2) {
    return 1;
  }
}
```

Figura 1. Códigos descompilados pelo IDA Pro com Hex-Rays Decompiler e pelo Ghidra

O Ghidra reconheceu corretamente o tipo da variável `iVar3`, ao passo que no Hex-Rays seria necessário corrigir a assinatura do método `IpCompare` para obter um código igualmente legível. Esse comportamento ajuda na análise de *malware* pois desofusca estruturas como a `in_addr_t`, que armazena endereços resolvidos em conexões de rede, facilitando assim a localização e a compreensão de como as conexões são formadas e utilizadas.

O Hex-Rays conseguiu expressar o fluxo de execução inteiramente com construções de programação estruturada, ao passo que o Ghidra gerou código que faz uso de `gotos`. O primeiro é mais eficiente para uso em análises, já que a instrução `goto` é muito evitada e quando aparece demonstra menor fidelidade ao código original.

As ferramentas constroem condições de formas diferentes: o Hex-Rays tenta produzir o menor código possível. No exemplo, a diferença por zero foi substituída por uma simples negação, alguns blocos de condição com atribuições de valores por condições lógicas e, quando o efeito da condição foi de apenas uma linha, os colchetes foram omitidos. Neste caso, considerações a respeito da melhor legibilidade são mais subjetivas, variando de acordo com a pessoa que fará a analise e com o código a ser analisado.

O Ghidra permite que mais de um arquivo seja analisado no mesmo projeto, assim bibliotecas dinâmicas e até dependências puderam ser adicionadas para compreender

melhor as chamadas à funções externas e sua participação final no objetivo do código malicioso.

A visualização do código lado-a-lado com o *assembly* do Ghidra permite uma consulta prática em casos que o descompilador não produziu um resultado claro, útil em casos de ofuscação que tenha como alvo esse tipo de análise.

O Hex-Rays possui um grafo de chamadas entre as funções num escopo do aplicativo inteiro enquanto no Ghidra o grafo fica restrito à função selecionada. Esses grafos auxiliam de forma diferentes na compreensão do funcionamento de um *malware*.

Ferramentas básicas como busca por chamadas de funções, símbolos e *strings*, manipulação dos tipos de dados, comentários, rótulos, manuais e informações de ajuda estão presentes em ambas as ferramentas.

Verificou-se que o tempo para abertura e análise do *malware* no IDA Pro foi menor que no Ghidra. Para análise manual por parte de um analista, a diferença pode ser pouco impactante, uma vez que essa espera só é necessária quando um binário é adicionado ao projeto. No entanto, a diferença no tempo de análise pode ser significativa para pesquisas que envolvam a automação de análise de uma grande amostragem de *malwares*.

3. Conclusões

Apesar de recentemente lançado, o Ghidra apresentou-se como ferramenta competitiva para análise estática de binários com relação ao IDA Pro. Em termos de extensibilidade, o Ghidra já apresenta vantagens, por ser de código aberto e por possibilitar acrescentar suporte a novas arquiteturas inclusive em seu descompilador. O presente estudo limitou-se a apresentar um resumo das características divulgadas pelos próprios fabricantes das ferramentas, além de algumas evidências anedóticas de diferenças entre as funcionalidades de cada uma. Espera-se que estudos futuros abordem de maneira mais detalhada a qualidade da saída dos descompiladores, colaborando para a evolução das ferramentas e para seu potencial de acelerar o trabalho do analista de *malware*.

Referências

Burji, S., Liszka, K. J., and Chan, C.-C. (2010). Malware analysis using reverse engineering and data mining tools. In *2010 International Conference on System Science and Engineering*, pages 619–624. IEEE.

Gutmann, P. (2007). The commercial malware industry. In *DEFCON Conference*.

Hex-Rays (2019). Hex-rays. `https://www.hex-rays.com/files/hexrays_info.pdf`.

National Security Agency (2019). Ghidra. `https://www.nsa.gov/resources/everyone/ghidra`.

Saxe, J. and Sanders, H. (2018). *Malware Data Science: Attack Detection and Attribution*. No Starch Press.

Zolkipli, M. F. and Jantan, A. (2010). Malware behavior analysis: Learning and understanding current malware threats. In *2010 Second International Conference on Network Applications, Protocols and Services*, pages 218–221. IEEE.