Descriptografando: Experimento em Análise de Memória Volátil Aplicada à Defesa Contra Ransomware

Ana Heloísa B. Mazur, Paulo Roberto de Oliveira, Luciana Andréia Fondazzi Martimiano,

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
Av. Colombo, 5790 – Bloco C56, Zona 7 – CEP: 87020-900
Fone: (44) 3011-4324/3011-4219
Maringá – PR

anabrav@proton.me, proliveira2@uem.br, lafmartimiano@uem.br

Abstract. *Ransomware is a type of malicious software that blocks access to a system or its data and demands a ransom payment. Volatile memory analysis allows for the observation of the state of a running system in certain moments of its execution, making it an interesting approach for malware analysis. This work aims to explore volatile memory analysis techniques and tools applied to the defense against crypto-ransomware attacks by presenting a practical experiment. Through volatile memory analysis, one can likely identify cryptographic material in memory, which enables the recovery of encrypted files in some circumstances. The experiment demonstrated the application of cryptographic key identification techniques based on static analysis and the virtual memory structure.*

Resumo. Ransomware *é um tipo de software malicioso que restringe acesso a um sistema ou seus dados, exigindo o pagamento do resgate. A análise de memória volátil permite observar o estado de um sistema em determinados momentos de execução, o que a torna uma abordagem interessante para a análise de* malware*. Este trabalho tem como objetivo explorar técnicas e ferramentas de análise de memória volátil aplicadas a defesa contra ataques de* ransomware *criptográfico, através de um experimento prático. Com a análise de memória, é possível identificar material criptográfico na memória, o que possibilita, em certas circunstâncias, a recuperação de arquivos criptografados. O experimento demonstrou a aplicação de técnicas de identificação de chaves criptográficas com base na análise estática e estrutura da memória virtual.*

1. Introdução

Ransomware designa um tipo de software malicioso projetado para restringir ou limitar acesso a sistemas ou dados até que o valor de resgate requisitado pelo atacante seja pago pela vítima [Savage et al. 2015]. Em 2022, 66% de 3000 organizações líderes no setor de tecnologia da informação no mundo todo foram atingidas por *ransomware*, tornando-o um dos maiores riscos para organizações e indivíduos no contexto digital [Sophos 2023]. *Ransomware* criptográfico é uma das ameaças mais alarmantes, uma vez que a recuperação dos dados criptografados sem o pagamento do resgate é inviável [Oz et al. 2022]. É necessário entender o funcionamento de softwares maliciosos como

ransomware, e projetar soluções de defesa efetivas para impedir que informações caiam em mãos erradas.

Existem duas abordagens fundamentais para a análise de *malware*, análise estática e análise dinâmica. A análise estática consiste em examinar a amostra sem executá-la, e a análise dinâmica envolve a execução do *malware* [Sikorski and Honig 2012]. Dentre os métodos de análise dinâmica, a análise de memória volátil destaca-se por ser um método considerado confiável, pois não deixa traços no sistema examinado [Cohen and Nissim 2018].

A aquisição e análise de memória volátil são fases da metodologia forense de memória volátil. Aquisição é o processo de obter uma cópia da memória RAM, e análise é o processo de extrair informações úteis do arquivo de memória [Vömel 2013]. A análise de memória é útil para a análise e detecção de *malware*, em especial *malware* que reside na memória por longos períodos de tempo. Um ataque de *ransomware* geralmente é composto por varias fases, incluindo infecção, comunicação com servidores *C&C* (servidores de controle e comando), e extorsão [Keshavarzi and Ghaffari 2020]. Portanto, o estado da memória volátil pode ser uma característica diferencial na defesa contra *ransomware*.

O presente trabalho tem como objetivo explorar as aplicações da análise de memória volátil na defesa contra *ransomware* criptográfico. Para isso, foi definido e realizado um experimento envolvendo a recuperação de chaves criptográficas durante um ataque de *ransomware* baseado em criptografia híbrida. O artigo está organizado da seguinte forma: a Seção 2 apresenta a fundamentação sobre *ransomware* criptográfico; a Seção 3 apresenta os principais conceitos em análise de memória e uma revisão da literatura sobre identificação de chaves criptográficas; a Seção 4 apresenta os trabalhos correlatos; a Seção 5 descreve o experimento e seus resultados; e a Seção 6 apresenta as considerações finais.

2. Ransomware Criptográfico

Ransomware é um tipo de *software* malicioso projetado para restringir ou limitar acesso a sistemas de computadores ou dados do usuário até que o valor de resgate requisitado pelo atacante seja pago [Oz et al. 2022]. Qualquer *malware* que realiza um ataque de extorsão, geralmente envolvendo negação de recursos, se encaixa em alguma categoria de *ransomware*. O método de negação de recursos determina a severidade do ataque, classificando *ransomware* em *ransomware* criptográfico e *ransomware* de bloqueio.

Ransomware criptográfico é um tipo de *ransomware* que criptografa os arquivos armazenados em um sistema, impedindo o usuário de acessá-los [Savage et al. 2015]. Esse tipo de *ransomware* utiliza a criptografia de maneira ofensiva para a extorsão da vítima, exigindo pagamento em troca da chave para descriptografar os dados [Young and Moti Yung 1996]. *Ransomware* de bloqueio limita o acesso ao próprio sistema, geralmente através da interface de usuário. *Ransomware* de bloqueio geralmente é menos sofisticado e pode ser removido através da restauração do sistema , enquanto para *ransomware* criptográfico, em casos que não existam *backups*, é impossível recuperar arquivos após um ataque sem possuir a chave privada – ou seja, sem o pagamento do resgate.

A técnica de criptografia empregada em um ataque pode ser criptografia ***simétrica***, ***assimétrica*** ou ***híbrida***, e pode ser realizada através de APIs do sistema operacional ou

implementada pelo *ransomware* [Oz et al. 2022]. Entender os modelos de criptografia utilizados em ataques de *ransomware* é importante para explorar vulnerabilidades no gerenciamento de chaves e desenvolver possíveis métodos de mitigação. [Bajpai et al. 2018] apresenta os modelos de criptografia utilizados em diferentes *ransomware* criptográficos, e introduz uma taxonomia baseada em gerenciamento de chaves.

Ransomware geralmente utiliza algoritmos conhecidos, como criptografia simétrica AES (*Advanced Encryption Standard*) [Dworkin et al. 2001] e criptografia assimétrica RSA [Rivest et al. 1978] . Algoritmos de chave simétrica utilizam a mesma chave para criptografar e descriptografar dados. Criptografia simétrica é mais rápida que criptografia assimétrica, o que é uma vantagem para *ransomware*, que busca criptografar o máximo de dados possíveis antes de ser detectado. Uma desvantagem é que, com gerenciamento de chaves impróprio, a chave simétrica pode ser exposta. A criptografia de chave assimétrica, também conhecida como criptografia de chave pública, utiliza um par de chaves, que consiste em uma chave pública e uma chave privada. A chave pública é utilizada para criptografar os dados, mas somente possuindo a chave privada é possível descriptografá-los. Porém, além de ser mais lenta que a criptografia simétrica, os dados criptografados com criptografia assimétrica ocupam mais espaço que os dados originais. Por essa razão, *ransomware* atuais utilizam uma abordagem híbrida, que une as vantagens dos dois tipos de criptografia [Bajpai et al. 2018].

Na abordagem híbrida, dados são criptografados utilizando um algoritmo de chave simétrica, pela sua velocidade, e em seguida a chave simétrica é criptografada com um algoritmo de chave assimétrica, utilizando a chave pública do atacante. A chave pública pode ser incluída no código do *ransomware* ou obtida por comunicação com o servidor de comando e controle. A chave simétrica original é destruída, tornando o atacante o único capaz de obtê-la a partir da chave criptografada [Keshavarzi and Ghaffari 2020].

3. Análise de Memória Volátil

A metodologia forense de memória volátil consiste nas fases de análise e aquisição da memória de um sistema. A análise de memória volátil é o processo de extrair informações úteis do arquivo de memória adquirido, podendo envolver a detecção de *malware* e análise de seu comportamento [Vömel 2013]. Essa metodologia oferece uma alternativa para sistemas em que os métodos tradicionais de análise dinâmica não são viáveis, devido a recursos computacionais ou requisitos de disponibilidade. Os processos de aquisição e análise podem ser realizados em máquinas diferentes, o que reduz os recursos computacionais necessários em comparação com a análise dinâmica tradicional. Análise de memória volátil é efetiva para a análise de *malware* que reside na memória por longos períodos de tempo, pois permite analisar as mudanças no sistema entre uma aquisição e a próxima. O estado da memória volátil oferece uma perspectiva melhor sobre o estado do sistema em um dado instante de tempo do que qualquer outra forma de análise dinâmica [Or-Meir et al. 2019].

A aquisição de memória volátil deve ser confiável e segura, de modo a garantir a validade das evidências e evitar que o *software* analisado interfira no processo de análise [Cohen and Nissim 2018]. Técnicas de aquisição de memória volátil são divididas em técnicas baseadas em *hardware* e *software*. Técnicas baseadas em *software* dependem de funções oferecidas pelo sistema operacional, enquanto técnicas baseadas em *hardware* acessam diretamente a memória de um computador, e são portanto consideradas mais

seguras e confiáveis [Vömel 2013]. Técnicas com base em *hardware* incluem placas de *hardware* dedicadas e barramentos de *hardware* especiais. Técnicas com base em *software* incluem virtualização, *software crash dumps*[1], aplicativos de aquisição de memória e *drivers*, ataques de *cold boot*[2], e arquivos de hibernação.

A técnica explorada neste trabalho é a aquisição de memória volátil com base em *software* através da virtualização, devido a sua disponibilidade em comparação com as outras técnicas. A tecnologia de virtualização facilita a execução de *malware* em um ambiente isolado e seguro [Sikorski and Honig 2012]. Utilizando máquinas virtuais, é possível executar múltiplos sistemas operacionais simultaneamente, e garantir o isolamento entre estes e o sistema hospedeiro [Smith and Nair 2005]. A aquisição de memória volátil através de virtualização é possível devido a funcionalidade de *snapshots* oferecida por máquinas virtuais, em que é possível suspender a execução da máquina virtual e obter uma captura do seu estado da memória, que é armazenada no disco da máquina hospedeira [Vömel 2013]. Essa funcionalidade é essencial para a análise de *malware* na memória volátil.

3.1. Identificação de Chaves Criptográficas

Dentre os principais artefatos que podem ser encontrados na memória principal de um sistema durante a execução do *ransomware* têm-se a chave e outros materiais criptográficos, em especial chaves simétricas em *ransomware* de criptografia simétrica ou híbrida. Obtendo a chave simétrica, que deve estar na memória durante o processo de criptografia, é possível em teoria descriptografar o arquivo criptografado.

Um dos primeiros trabalhos que trata de ataques de identificação de chaves criptográficas na literatura é [Shamir and van Someren 1999], no contexto de ataque a um disco rígido. Segundo os autores, esse tipo de ataque é diferente da criptanálise, que consiste em computar chaves desconhecidas. São apresentados um algoritmo para localizar chaves RSA utilizando o conhecimento da infraestrutura de chave pública; e um algoritmo para encontrar chaves criptográficas arbitrárias em programas grandes. Apenas a segunda abordagem é aplicável à criptografia simétrica, e tem como base a característica de aleatoriedade de chaves criptográficas, o que as diferencia significativamente de outras sequências de *bytes* de dados ou código . Quando dados são aleatórios, eles possuem um valor maior de **entropia** – dessa forma, é possível encontrar chaves localizando seções com altos valores de entropia. Os autores afirmam que, na prática, não é necessária uma medida correta de entropia, considerando que a entropia de código e dados geralmente é baixa – sendo assim, uma estimativa do valor é suficiente, como a contagem de valores únicos de bytes em uma sequência. Utilizar uma estimativa também reduz a complexidade da busca devido a complexidade do cálculo da entropia.

[Kaplan 2007] aborda extração de evidência criminosa de discos rígidos criptografados utilizando análise de memória volátil, no contexto da metodologia de resposta ao incidente em forense digital. Segundo os autores, para uma chave assimétrica, cada tentativa de decodificação é custosa, mas o atacante tem a vantagem de poder gerar um par de texto original e texto criptografado utilizando a chave pública, que pode ser utilizado

[1] Arquivos de despejo de falhas de *software*.

[2] Um tipo de ataque em que explora a permanência de memória em células DRAM para acessar a memória principal de um sistema [Halderman et al. 2009].

para checar cada chave candidata. No caso de chaves simétricas, se os dados originais são o conteúdo do disco, é possível tomar vantagem de características estruturais, como estruturas de sistemas de arquivo e a localização do *bootloader*.

[Kaplan 2007] propõe também um novo método para a extração de chaves simétricas, explorando detalhes de implementação de cifras de bloco. O algoritmo tem como base a computação do cronograma de chaves, uma sequência de sub-chaves utilizadas em cada *round* de uma cifra de bloco. O cronograma de chaves geralmente é pré-computado e armazenado junto com a chave original por razões de performance, portanto, sabendo o algoritmo de criptografia, é possível computar o cronograma de chaves de uma chave candidata e verificar se uma sub-chave aparece em alguma posição na memória. Entretanto, na prática, sistemas de criptografia simétricos consistem em mais elementos do que apenas uma cifra de blocos e uma chave privada [Kaplan 2007]. Ou seja, é necessário entender os detalhes como o modo de operação da cifra, e como os vetores de inicialização são gerados, caso contrário o ataque torna-se um ataque de força bruta contra a implementação da criptografia.

[Maartmann-Moe et al. 2009] propõem utilizar a estrutura da memória para reduzir o espaço de busca, observando apenas regiões mais prováveis da memória, como o espaço de endereço virtual de um processo específico, através da tradução de endereços. Isso pode ser feito automaticamente utilizando ferramentas de análise de memória, como o *framework Volatility*[3]. Dessa forma, a região de memória reconstruída pode ser utilizada em um ataque de força bruta. Outros trabalhos apresentam técnicas de identificação de chaves que dependem do acesso ao código fonte para determinar características ou estruturas da chave, como [Walters and Petroni 2007]. Esse tipo de técnica não é muito útil quando aplicada a código malicioso, em que não se tem acesso ao código fonte [Davies et al. 2020], porém a análise do código binário pode levar a conclusões semelhantes.

As técnicas descritas são efetivas, porém podem ser frustradas. [Shamir and van Someren 1999] sugerem armazenar a chave em partes separadas como medida preventiva. A técnica com base no cronograma de chaves falha se o cronograma está armazenado em outro tipo de estrutura, ou se ele é sempre re-computado [Kaplan 2007]. Outras estratégias de mitigação sugeridas em [Halderman et al. 2009] envolvem descartar ou ofuscar chaves antes que um adversário possa obter acesso físico à máquina; transformar a chave quando ela é gravada na memória e reconstruí-la quando necessário; e impedir que *software* de extração de memória execute na máquina. *Software* criptográfico deve destruir chaves e dados não criptografados na memória assim que não são mais necessários. Também é preciso garantir que as chaves nunca sejam escritas no disco, como resultado do gerenciamento da memória virtual [Maartmann-Moe et al. 2009].

4. Trabalhos Correlatos

Enquanto as pesquisas nas áreas individuais de análise de memória volátil e defesa contra *ransomware* são abundantes atualmente, trabalhos que abordam ambas as áreas são poucos: a maioria dos trabalhos exploram a análise de memória aplicada a *malware* em geral. Por outro lado, os trabalhos que abordam *ransomware* variam na aplicação da análise

[3] `https://github.com/volatilityfoundation/volatility3`

de memória. [Davies et al. 2020] e [Bajpai and Enbody 2020] são dois dos dos primeiros trabalhos na literatura que abordam identificação de chaves de criptografia aplicado especificamente à *ransomware*.

Em [Cohen and Nissim 2018] foi proposta e testada uma solução de detecção de *ransomware* e outros tipos de *malware* com base na aplicação de algoritmos de aprendizagem de máquina a características extraídas de capturas de memória. Os algoritmos aplicados apresentaram resultados altos de detecção. A solução proposta, porém, não é capaz de identificar qual processo do sistema representa um *malware* – ela é projetada para analisar a memória do sistema como um todo. [Davies et al. 2020] investiga a utilidade de técnicas atuais da análise forense de memória para a descoberta de chaves de criptografia utilizadas por *ransomware*. Foi conduzida uma série de experimentos com as diferentes famílias *NotPetya*[4], *BadRabbit*[5] e *Phobos*[6], envolvendo a tentativa de descriptografar os arquivos criptografados, e a criação de uma *timeline* da presença da chave na memória durante a execução do *ransomware*.

Uma das principais diferenças entre *ransomware* e outras aplicações criptográficas, além do comportamento malicioso, é que *ransomware* executa diversas etapas fundamentais antes de iniciar a criptografia dos dados no sistema da vítima. Ou seja, as chaves não estarão presentes na memória do sistema da vítima no início da execução do programa, por isso determinar quando capturar a memória é essencial. Considerando as fases de ataque, idealmente a memória deve ser adquirida durante a fase de criptografia, porém essa pode durar de alguns minutos até horas, e nada garante que o *ransomware* não realize um bom gerenciamento de chaves e apague as chaves da memória imediatamente após seu uso [Davies et al. 2020].

Em [Bajpai and Enbody 2020] é apresentada uma técnica de resposta e recuperação para neutralizar um ataque de *ransomware* após a sua ocorrência, em contraste com soluções focadas na prevenção e detecção. Os autores argumentam que a ameaça de *ransomware* exige uma estratégia de defesa com foco em todos os aspectos de segurança, inclusive na recuperação, e soluções alternativas a *backups* também devem ser consideradas. Segundo os autores, entre as soluções de defesa existentes, existe uma disparidade entre soluções de detecção e de neutralização de *ransomware*. Implementações convencionais de algoritmos criptográficos são altamente suscetíveis a ataques *side-channel*, como a análise de memória volátil, quando a criptografia é realizada em um sistema adversário. Em um ataque de *ransomware*, a aquisição da chave de criptografia através da exploração de erros na implementação do sistema de criptografia ou através de ataques *side-channel* permite a restauração dos dados sem o pagamento do resgate. No caso de *ransomware* de criptografia híbrida, na ausência de falhas de implementação, a criptografia é inquebrável – entretanto, o conhecimento da chave simétrica antes da aplicação da criptografia assimétrica é suficiente para descriptografia.

5. Experimento

O experimento conduzido seguiu um conjunto de etapas determinadas a partir da revisão da literatura e análise de trabalhos correlatos. A Figura 1 apresenta um diagrama simpli-

[4]`https://en.wikipedia.org/wiki/Petya_(malware_family)`
[5]`https://attack.mitre.org/software/S0606/`
[6]`https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-060a`

ficado da execução. As etapas são listadas abaixo.

1. **Seleção da amostra de *ransomware* criptográfico**: Pesquisar e escolher uma família de *ransomware* criptográfico, e obter a amostra a ser estudada.
2. **Execução da amostra em um ambiente isolado**: Executar a amostra em uma máquina virtual, em um ambiente seguro e isolado da internet.
3. **Aquisição de memória volátil**: Adquirir uma imagem da memória volátil da máquina virtual durante o período de execução do *ransomware*.
4. **Análise da imagem de memória**: Aplicar técnicas e ferramentas de análise de memória volátil para obter informação sobre o *ransomware* e o estado da máquina.
5. **Recuperação**: Partindo dos resultados da análise, explorar as possibilidades de recuperação de dados criptografados.

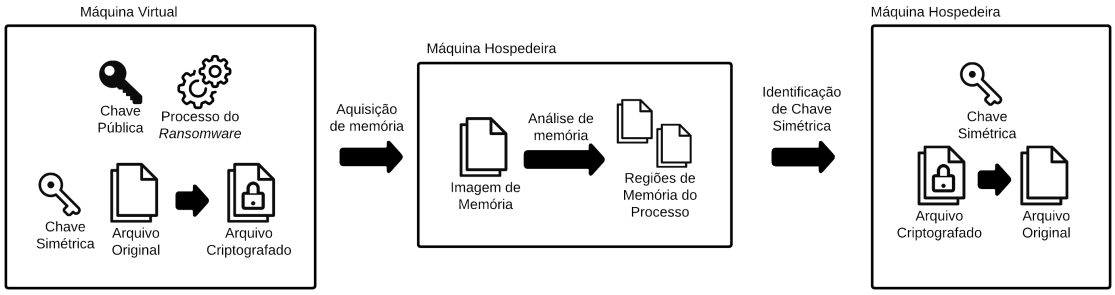

Figura 1. Execução do experimento

O *ransomware* selecionado como objeto de estudo é denominado *Dearcry*, empregado em ataques operados manualmente no início de 2021. Os ataques foram facilitados através da exploração das vulnerabilidade *CVE-2021-26855* e *CVE-2021-27065* em servidores Microsoft Exchange[7] [Loman 2021]. O *ransomware* fez vítimas nos Estados Unidos, Luxemburgo, Indonesia, Irlanda, India e Alemanha, exigindo quantias de até 16 mil dólares como resgate [Abrams 2021]. Em 2024, não é uma ameaça recente ou particularmente relevante, porém foi escolhida devido a sua baixa complexidade, como atestado na análise relatada em [Loman 2021]. A *hash* MD5 da amostra obtida é `0e55ead3b8fd305d9a54f78c7b56741a`.

Após a escolha do *ransomware*, a amostra foi executada em um ambiente isolado, utilizando a ferramenta *VirtualBox* para o gerenciamento de máquinas virtuais. Foi preparada uma máquina virtual com o sistema operacional Windows 10 de 64 bits, utilizando a mídia de instalação obtida em `https://www.microsoft.com/pt-br/software-download/windows10ISO`. A máquina hospedeira tem 4 gigabytes de memória, processador *Intel Core i3*, executando o sistema operacional *Arch Linux*, e a máquina hóspede tem 2 gigabytes de memória e 80 gigabytes de armazenamento.

Anterior à execução da amostra, foi gerado um arquivo de texto teste de 500 megabytes, chamado `sample.txt`, com uma sequência de caracteres alfabéticos aleatórios. Este é o arquivo a ser criptografado e posteriormente recuperado. Tendo em mente o interesse na extração de chaves criptográficas, foi necessária a aquisição

[7]Aplicação servidora de e-mails e calendário digital. Disponível em `https://www.microsoft.com/pt-br/microsoft-365/exchange/email`.

da memória da máquina durante o processo de criptografia. Sendo assim, o tamanho do arquivo foi determinado de modo a aumentar o período de tempo necessário para sua criptografia completa, facilitando a aquisição. O arquivo foi transferido para a máquina virtual, utilizando a funcionalidade de pastas compartilhadas do pacote de *software Guest Additions* do *VirtualBox* [Oracle 2024].

VirtualBox oferece funcionalidades de *snapshot*, em que é possível salvar o estado de uma máquina virtual. Entretanto, o formato de arquivo de *snapshot* não pode ser analisado diretamente com o *framework Volatility*. Portanto, para adquirir a memória da máquina, foi utilizada a ferramenta *VBoxManage*, a interface de linha de comando para VirtualBox. A etapa de análise de memória foi realizada utilizando o *framework Volatility 3* e o *dissassembler* IDA. O objetivo da análise foi investigar o comportamento do *ransomware* e seu efeito na memória do sistema, e entender como a criptografia de arquivos é realizada. Em seguida, a etapa de recuperação utilizou as informações adquiridas para descriptografar o arquivo teste criptografado, obtendo a versão original do arquivo. As etapas de análise e de recuperação foram realizadas na máquina hospedeira.

5.1. Análise de Memória

A captura foi realizada após ser verificada a criação do arquivo `sample.txt.CRYPT`, a versão criptografada do arquivo original. Utilizando o *framework Volatility*, a imagem de memória obtida foi analisada para obter informações sobre os processos em execução no momento da captura e extrair páginas e artefatos da memória. Esse processo é descrito nesta subseção.

Através do *plugin* `windows.pslist`, foi obtida uma lista dos processos em execução no sistema. A saída apresenta o PID do processo, o PID do processo pai, o nome do arquivo, o endereço de memória virtual, o número de *threads*, o número de *handles*, o horário de criação e terminação, entre outras informações sobre processos. Nesse caso, é fácil identificar o *malware* pelo nome do arquivo correspondente ao processo, `0e55ead3b8fd305d9a54f78c7b56741a.exe`. Em situações reais, a identificação do processo pode ser dificultada por nomes parecidos com processos do sistema, ou pela manipulação dos ponteiros da lista ligada de processos mantida pelo sistema. Neste último caso, pode-se utilizar o *plugin* `windows.psscan`, que varre a memória em busca de estruturas do tipo `_EPROCESS` [Ligh et al. 2014].

Obtendo o PID do processo malicioso, é possível obter também a lista de *handles*[8] através do *plugin* `windows.handles`; e o comando utilizado para executar o processo, sabendo que seu processo pai é o *prompt* de comando do Windows, utilizando o *plugin* `windows.cmdline`. De maior interesse é a possibilidade de extrair arquivos residentes na memória, em especial os arquivos executáveis, utilizando o *plugin* `windows.dumpfiles`. Através desse *plugin*, filtrando os resultados pelo PID do processo, foi obtido o arquivo executável do *ransomware*.

Muitos arquivos PE, especialmente executáveis maliciosos, modificam seu próprio código em tempo de execução, utilizando técnicas como *packing* ou criptografia. Essas técnicas dificultam a análise estática do código quando ele está em disco. Quando carregado na memória, porém, a camada de ofuscação introduzida é removida, sendo possível

[8]Referências a objetos do sistema operacional, como um arquivo, processo, ou fluxo de execução.

extrair o código descomprimido da memória [Ligh et al. 2014]. No caso de *Dearcry*, o executável em disco não apresenta sinais de ofuscação – suas *strings* e bibliotecas importadas são visíveis a análise estática.

5.2. Análise Estática

Após a extração do arquivo executável da memória, é possível realizar a análise estática como qualquer outro executável. Arquivos executáveis e DLLs em sistemas Windows 32 e 64 bits são estruturados no formato PE, ou *Portable Executable*. Esse formato de arquivos contém a informação necessária para que o sistema carregue e execute o programa, incluindo referências de bibliotecas dinâmicas, tabelas de funções importadas e exportadas, etc[9]. [Sikorski and Honig 2012] divide a análise estática em análise estática básica e análise estática avançada, ou engenharia reversa. A primeira consiste em utilizar ferramentas para extrair informações do executável – como *strings*, bibliotecas e funções – sem analisar o código de máquina; a segunda consiste em utilizar ferramentas de *disassembly* para examinar o código de máquina.

Posição	**Conteúdo da *string***
`0xf078c`	OpenSSL
`0x12e388`	.CRYPT
`0x1354a0`	——BEGIN RSA PUBLIC KEY——
`0x13562d`	——END RSA PUBLIC KEY——
`0x135718`	... .EXE .DLL .CAD .AVI .H.CSV .DAT .ISO .PST .PGD .7Z ...

Tabela 1. Saída da ferramena *GNU Strings*

A análise estática básica revelou informações relevantes sobre o provável comportamento do *ransomware*. A tabela 1 apresenta os dados encontrados através da ferramenta *GNU Strings*[10]. A tabela 2 apresenta as DLLs importadas pelo executável, obtidas através da ferramenta *readpe*[11]. Partindo desses dados, é possível supor que:

- A amostra provavelmente utiliza criptografia híbrida, devido a chave pública RSA.
- A amostra criptografa os arquivos com as extensões da lista, incluindo arquivos executáveis e DLLs.
- O executável é estaticamente ligado com a biblioteca de criptografia OpenSSL[12].

A engenharia reversa da amostra foi conduzida utilizando o *disassembler* IDA *Free*[14]. O objetivo principal desta análise foi obter informações detalhadas sobre o processo de criptografia realizado pelo *ransomware*. Essas informações são úteis para expor possíveis falhas de implementação ou vulnerabilidades do sistema de criptografia, e possivelmente para restaurar danos causados pelo *malware*.

O arquivo executável foi carregado no *disassembler* IDA e submetido a análise automática inicial. Como foi verificada a presença da biblioteca *OpenSSL*, fez-se necessário diferenciar os procedimentos definidos pela aplicação daqueles oferecidos pela

[9] `https://en.wikipedia.org/wiki/Portable\_Executable`
[10] `https://www.man7.org/linux/man-pages/man1/strings.1.html`
[11] `https://github.com/mentebinaria/readpe`
[12] `https://www.openssl.org/`
[14] `https://hex-rays.com/ida-free/`

DLL	Descrição
`KERNEL32.dll`	APIs básicas para gerenciamento de memória, operações de entrada e saída, e criação de processos e *threads*.
`ADVAPI32.dll`	Providencia funções para manipulação do registro do *windows*.
`WS2_32.dll`	Funcionalidades de comunicação pela internet.
`USER32.dll`	Componentes de interface de usuário.
`CRYPT32.dll`	Implementa funções de certificação e comunicação criptográfica [13].

Tabela 2. DLLs importadas

biblioteca. Para isso, foi utilizada a tecnologia *IDA FLIRT*, aplicando a assinatura *FLIRT* da biblioteca *OpenSSL* disponível no repositório FLIRTDB[15]. A tecnologia *FLIRT* é um diferencial do *disassembler* IDA, que emprega algoritmos de correspondência de padrões para identificar sequências de código de biblioteca segundo assinaturas conhecidas [Eagle 2008].

Realizando uma busca pela *string* "`.CRYPT`" observada anteriormente durante a análise de *strings*, reconhecida como a extensão adicionada ao nome de arquivos criptografados, foi possível encontrar resultados promissores. Uma ocorrência da *string* foi encontrada no segmento de dados, e uma única referência a essa *string* na sub-rotina `sub_B311C0`. Analisando esta sub-rotina, é possível identificá-la como a sub-rotina responsável pela criptografia de arquivos individuais. A seguir, a operação da sub-rotina é descrita em detalhes.

1. A sub-rotina recebe como argumento a chave pública RSA do *ransomware* e o nome do arquivo a ser criptografado.
2. O nome do novo arquivo é criado concatenando o nome do arquivo original com a extensão "`.CRYPT`".
3. Dois fluxos de arquivos são abertos, o arquivo original no modo de leitura e gravação, e o arquivo de saída no modo de gravação.
4. Um buffer é preenchido com 48 *bytes* aleatórios, identificados como a chave de 32 *bytes* e vetor de inicialização de 16 *bytes* para o algoritmo de criptografia simétrica AES.
5. Os dados de cabeçalho são escritos no arquivo de saída, incluindo a chave e o vetor de inicialização criptografados em RSA[16].
6. Realiza a criptografia do arquivo de entrada em AES no modo CBC, gravando o resultado no arquivo de saída.
7. Sobrescreve o arquivo de entrada para evitar a recuperação do seu conteúdo no disco.

5.3. Recuperação

As informações adquiridas pela análise de memória e análise estática do executável facilitam o processo de extração de chaves, que possibilita a descriptografia do arquivo criptografado. O arquivo em questão é o arquivo `sample.txt.CRYPT` e sua versão original

[15] `https://github.com/Maktm/FLIRTDB`

[16] Ver apendice A.

`sample.txt`, que estava sendo criptografado no momento da aquisição de memória. Como *Dearcry* usa criptografia híbrida, com uma chave simétrica para cada arquivo, ao obter uma chave de criptografia simétrica só é possível descriptografar o arquivo correspondente.

Foi determinado que o algoritmo utilizado pelo *ransomware* é o algoritmo de criptografia simétrica AES com chave de 256 bits, no modo de operação CBC, e que na sub-rotina utilizada para a criptografia de arquivos, é gerada uma chave e vetor de inicialização, alocados de forma contígua na pilha em um *buffer* de 48 *bytes*. A chave tem 32 *bytes* ou 256 bits, e o vetor de inicialização tem 16 *bytes*, como especificado para o algoritmo. Sabendo que o material criptográfico é alocado na pilha, é possível reduzir significativamente o espaço de busca. Para isso, é necessário conhecimento da organização do espaço de memória virtual de um processo no sistema operacional *Windows*, de modo a identificar as regiões de memória que contém dados de pilhas de execução de *threads*.

O posicionamento de regiões de memória específicas em um processo pode variar de acordo com o sistema e arquitetura do processador. É possível obter informações sobre as regiões de memória de um processo de diferentes fontes. Uma das principais fontes são *virtual address descriptors* (VADs), ou descritores de memória virtual [Ligh et al. 2014]. *Virtual Address Descriptors* são estruturas de dados mantidas pelo gerenciador de memória do Windows que mantém os endereços virtuais utilizados por um processo. Para cada processo, é mantida um conjunto de VADs que descrevem o estado de seu espaço de endereço virtual. VADs são organizadas em árvores AVL auto balanceadas, para facilitar a inserção, busca, e remoção de nós. Cada nó da árvore representa uma região de memória virtualmente contígua que apresenta as mesmas características [Yosifovich et al. 2017].

As características de uma região de memória na VAD são descritas por um conjunto de *flags*. Duas das principais *flags* são `Protection` e `PrivateMemory`. `Protection` indica que tipo de acesso deve ser permitido na região de memória. O valor dessa *flag* pode ser útil para determinar que dados podem estar presentes na região. `PrivateMemory` indica que a região de memória é privada e não pode ser compartilhada ou herdada por outros processos. Se essa flag está presente, então a região de memória não contém arquivos mapeados, como executáveis ou DLLs. As regiões de *heap* e pilhas de um processo geralmente são marcadas como privadas [Ligh et al. 2014].

O *plugin* `windows.vadinfo` do *framework Volatility* lista os nós VAD em uma imagem de memória e seus atributos, incluindo endereços iniciais e finais, *flags*, e localização de arquivos mapeados, e permite a extração das regiões apresentadas. O *plugin* foi utilizado na imagem de memória adquirida, e 183 arquivos foram extraídos, totalizando 169 *megabytes*. Dentre estes foram selecionadas as regiões de memória que podem conter páginas da pilha – regiões com nível de proteção `PAGE_READWRITE` (leitura e escrita) e memória privada. Com os 16 arquivos resultantes foi realizada a busca por chaves criptográficas, utilizando um algoritmo de busca por entropia.

O algoritmo obtem uma estimativa do valor de entropia para cada subsequência de n bytes. A estimativa é calculada contando o número de valores únicos *bytes* na subsequência – quanto mais valores únicos, maior a entropia; e quanto mais valores repetidos, menor a entropia. Se essa estimativa exceder o limite estabelecido, a subsequência é con-

siderada uma possível chave criptográfica, e a tentativa de descriptografia é realizada. Nessa aplicação do algoritmo, n é o tamanho do *buffer* que contém ambos a chave e o vetor de inicialização AES, ou seja, $n = 48$ *bytes*. O limite de entropia foi determinado por tentativa e erro como 42 *bytes*.

Para a tentativa de descriptografia, foram considerados somente os 16 primeiros *bytes* do conteúdo do arquivo, isto é, apenas 16 *bytes* após o final do cabeçalho adicionado pelo *ransomware*. Como o algoritmo AES é uma cifra de blocos [Dworkin et al. 2001], o conteúdo pode ser parcialmente descriptografado, reduzindo o tempo de busca.

Para determinar se a chave correta foi encontrada, é necessário diferenciar uma tentativa de descriptografia bem sucedida de uma tentativa falha, o que pode ser difícil se não se tem nenhum conhecimento do arquivo original. Em outras palavras, é necessário saber o conteúdo de alguma parte do arquivo para comparar com o conteúdo obtido – caso contrário, as tentativas não podem ser automatizadas e a busca torna-se inviável. Isso apresenta um desafio um tanto contraditório, pois é necessário conhecer o conteúdo original para obter o conteúdo original. Entretanto, um detalhe é essencial: basta conhecer ao menos o valor de **um *byte*** e **sua posição no arquivo**, e os resultados das tentativas são drasticamente reduzidos, e podem então ser examinados manualmente para encontrar o resultado correto. No caso de formatos de arquivos estruturados, por exemplo, pode ser possível obter um único resultado, conhecendo a estrutura interna do arquivo. É possível comparar os valores de números mágicos, que são valores especiais utilizados para reconhecimento de formatos de arquivos[17]. Por outro lado, para arquivos não estruturados, algum outro tipo de informação deve ser conhecida. No presente experimento, o arquivo em questão é um arquivo de texto simples, porém tem-se conhecimento do conteúdo do arquivo original. O valor utilizado para a comparação foi o primeiro caractere do arquivo. Também é possível descriptografar grande parte do arquivo sem o vetor de inicialização, devido a natureza do modo CBC.

O processo de extração de chaves foi realizado através de um *script* em *Python*[18]. O algoritmo de busca foi aplicado em cada um dos arquivos de memória selecionados, seguindo para o próximo arquivo se nenhuma tentativa de criptografia teve sucesso. Os arquivos de regiões de memória foram ordenados pelo VPN, ou número virtual de página. A chave e o vetor de inicialização utilizados na criptografia do arquivo teste foram encontrados com sucesso, possibilitando a recuperação de todos os 500 megabytes do arquivo original. Foram necessárias 3 122 955 tentativas de descriptografia para encontrar a chave original, que foi encontrada na oitava região de memória analisada, percorrendo um total de 5 646 677 *bytes* (aproximadamente 5.4 *megabytes)* dos 169 *megabytes* de memória.

Embora não seja possível recuperar o restante dos arquivos comprometidos devido ao uso de diferentes chaves simétricas para cada arquivo, esse experimento demonstra que a criptografia simétrica e híbrida é vulnerável quando se tem acesso a memória, e são apresentados alguns métodos interessantes para a redução do espaço de busca das chaves e do tempo de execução do algoritmo, utilizando a análise estática de *malware* e conhecimento da estrutura da memória volátil.

[17] `https://pt.wikipedia.org/wiki/N\%C3\%BAmero\_m\%C3\%A1gico\_(programa\%C3\%A7\%C3\%A3o\_de\_sistemas)`

[18] Disponível em `https://github.com/anabravn/ransom-key-ex`

6. Considerações Finais

O experimento realizado com o *ransomware Dearcry* envolveu o processo de análise de memória de uma máquina virtual vítima de um ataque. Foram descritas as ferramentas e métodos utilizados para obter informações da imagem de memória capturada, utilizando o *framework Volatility* para extrair a memória individual do processo malicioso, e o *disassembler* IDA para analisar o código de criptografia do *ransomware*. O comportamento do *ransomware* e sua abordagem de criptografia foram expostos, analisados, e discutidos. O experimento demonstrou como é possível identificar a chave simétrica de criptografia, apresentando métodos para reduzir o espaço de busca através da estrutura da memória, conhecimento da aplicação maliciosa obtido através da análise estática, e conhecimento do sistema de criptografia. O experimento teve sucesso em recuperar um dos arquivos criptografados, explorando a abordagem de criptografia híbrida. Não foram identificadas falhas na implementação da criptografia, o que impossibilita a recuperação de outros arquivos devido ao uso de diferentes chaves simétricas para cada arquivo, que existem na memória em diferentes períodos de tempo.

A identificação de chaves criptográficas como um contra-ataque para a criptografia de um *ransomware* em execução não é uma solução prática. Primeiramente, a análise de memória não é um processo contínuo no sistema, ou seja, é necessário a aquisição da memória em um momento específico para que esta possa ser analisada – sendo assim, seria difícil obter uma imagem de memória que coincida com o momento em que o *ransomware* esteja em execução e que contenha o material criptográfico necessário para a eventual recuperação de dados. Além disso, se a execução do *ransomware* foi detectada no sistema, a solução racional é terminar o processo imediatamente, minimizando a perda de arquivos. Por essas razões, o experimento aqui apresentado atualmente é inviável em uma situação real. Apesar dessas limitações, os autores acreditam que a situação explorada traz conhecimentos importantes para a área da segurança, e que vale a pena ser examinada, se não pela sua utilidade atual, então pela sua possível utilidade futura. Como afirmado em [Bajpai and Enbody 2020], é importante considerar todos os aspectos de segurança para construir boas estratégias de defesa.

Diferentemente de outros trabalhos que abordam a extração de chaves criptográficas de *ransomware*, este trabalho destaca a análise estática de código como a principal ferramenta para guiar a análise de memória. Possíveis questões a serem exploradas em trabalhos futuros incluem: análise e comparação de famílias de *ransomware* na perspectiva de métodos de criptografia e gerenciamento de chaves; e automação da identificação de chaves com base na estrutura da memória virtual.

Referências

Abrams, L. (2021). DearCry ransomware attacks Microsoft Exchange with ProxyLogon exploits. *BleepingComputer*.

Bajpai, P. and Enbody, R. (2020). Memory Forensics Against Ransomware. *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–8.

Bajpai, P., Sood, A. K., and Enbody, R. (2018). A key-management-based taxonomy for ransomware. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–12. ISSN: 2159-1245.

Cohen, A. and Nissim, N. (2018). Trusted detection of ransomware in a private cloud using machine learning methods leveraging meta-features from volatile memory. *Expert Systems with Applications: An International Journal*, 102(C):158–178.

Davies, S. R., Macfarlane, R., and Buchanan, W. J. (2020). Evaluation of live forensic techniques in ransomware attack mitigation. *Forensic Science International: Digital Investigation*, 33:300979.

Dworkin, M. J., Barker, E., Nechvatal, J. R., Foti, J., Bassham, L. E., Roback, E., and Jr, J. F. D. (2001). Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST).

Eagle, C. (2008). *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*. No Starch Press, USA.

Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J., and Felten, E. W. (2009). Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98.

Kaplan, B. (2007). RAM is Key Extracting Disk Encryption Keys From Volatile Memory.

Keshavarzi, M. and Ghaffari, H. (2020). I2CE3: A dedicated and separated attack chain for ransomware offenses as the most infamous cyber extortion. *Computer Science Review*, 36:100233.

Ligh, M. H., Case, A., Levy, J., and Walters, A. (2014). *The art of memory forensics: detecting malware and threats in Windows, Linux, and Mac memory*. Wiley, Indianapolis, IN. OCLC: ocn885319205.

Loman, M. (2021). DearCry ransomware attacks exploit Exchange server vulnerabilities. *Sophos News*.

Maartmann-Moe, C., Thorkildsen, S. E., and André Årnes (2009). The persistence of memory: Forensic identification and extraction of cryptographic keys. *Digital Investigation*, 6:S132–S140.

Or-Meir, O., Nissim, N., Elovici, Y., and Rokach, L. (2019). Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. *ACM Computing Surveys*, 52(5):88:1–88:48.

Oracle (2024). Oracle VM VirtualBox User Manual. Versão 7.0.14.

Oz, H., Aris, A., Levi, A., and Uluagac, A. S. (2022). A Survey on Ransomware: Evolution, Taxonomy, and Defense Solutions. *ACM Computing Surveys*, 54(11s):238:1–238:37.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

Savage, K., Coogan, P., and Lau, H. (2015). The evolution of ransomware.

Shamir, A. and van Someren, N. (1999). Playing 'Hide and Seek' with Stored Keys. In Franklin, M., editor, *Financial Cryptography*, Lecture Notes in Computer Science, pages 118–124, Berlin, Heidelberg. Springer.

Sikorski, M. and Honig, A. (2012). *Practical malware analysis: the hands-on guide to dissecting malicious software*. No Starch Press, San Francisco.

Smith, J. and Nair, R. (2005). The architecture of virtual machines. *Computer*, 38(5):32–38. Conference Name: Computer.

Sophos (2023). The State of Ransomware. Technical report, Sophos.

Vömel, S. (2013). *Forensic acquisition and analysis of volatile data in memory*. PhD thesis, University of Erlangen-Nuremberg.

Walters, A. and Petroni, N. L. (2007). Volatools: Integrating Volatile Memory Forensics into the Digital Investigation Process.

Yosifovich, P., Ionescu, A., Russinovich, M. E., and Solomon, D. A. (2017). *Windows Internals, Part 1: System architecture, processes, threads, memory management, and more*. Microsoft Press, 7 edition.

Young, A. and Moti Yung (1996). Cryptovirology: extortion-based security threats and countermeasures. *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 129–140.

A. Arquivos, Cabeçalho e Chaves

```
D1 7B 33 8B A5 91 B1 A8 3B D0 89 A0 F0 E1 02 02
79 52 9A 99 11 A4 3D 23 EA A0 70 5D 55 60 D6 15
DE 80 E1 A2 EC E6 A8 73 19 A4 AF 0D 28 9D 3B D6
```

Figura 2. Chave simétrica e vetor de inicialização

```
gdlndeygjcyhvcawlltpnxjooivvlvmcdvyzrhtxvqxtiuyj
gxvgjindxjwhgjfxghudahmrxktgcivwggnikqqhuaubsuba
uavanytodgukenydfqszpriselnvvylhcgzxmvmzubifurmm
```

Figura 3. Início do arquivo original

A Figura 2 apresenta a chave e o vetor de inicialização para o algoritmo AES para o arquivo teste. A Figura 3 apresenta os primeiros 144 *bytes* do arquivo original, e a Figura 4 apresenta os primeiros 432 *bytes* do arquivo criptografado. A Figura 5 apresenta o cabeçalho do arquivo criptografado. Os campos são: o marcador "`DEARCRY!`", o tamanho da chave e vetor criptografados em RSA, o conteúdo da chave e vetor criptografados, os *bytes* `04 00 00 00`, e o tamanho do arquivo original.

```
44 45 41 52 43 52 59 21  00 01 00 00 00 79 17 d9
92 ce 2c 34 49 c3 68 20  b6 4c 8a 29 1c 9a 61 b4
30 88 9a 18 7d 3a 87 7b  11 c9 ac c9 81 36 ae d0
e2 cb f1 1c dc 92 fd 14  68 ed a7 f2 65 74 8b 50
f6 3f 60 79 99 18 19 eb  83 8b 34 92 04 99 84 37
1d 7b de 6c 8c 5e da 37  12 69 d7 5e d8 63 c6 de
4c f2 83 6c 74 18 60 15  7e 67 93 32 30 94 af 31
c3 65 fd 1e 35 79 c2 78  b8 45 eb 5f 16 a3 72 40
4e b1 33 bb 71 20 69 9d  a9 c9 79 d4 4c 91 72 96
da ac 6b 2d 00 c8 1c 8d  28 69 be 43 4f 58 a4 d4
73 70 33 99 f1 42 2f fd  2b 0a eb 0f 90 2c a0 d1
bf ea a6 c1 0e 25 92 8a  0a 7d 27 46 04 fc 14 63
db 03 a2 4f e4 08 f0 6d  6f b3 a4 fc a5 ff f4 25
9a c3 12 bf 7d 0e e4 41  60 d5 db 23 e9 72 3f 3d
e9 19 35 fb 84 10 67 c2  36 88 2f 82 31 d5 17 bb
99 36 10 6b 08 82 3d 58  16 7c 7b f0 b3 a3 db de
76 38 9a ab a7 93 04 36  51 98 83 bb 04 00 00 00
00 00 40 1f 00 00 00 00  c4 ea 0f 47 7f 0c ea bf
89 c5 1c 50 fd 2a 8d 50  68 4e d1 5c 6d 7f 2c 72
ef 10 5a 28 b4 93 49 85  32 7a a8 4b 68 e5 7b 35
e4 7b a9 31 ff d1 31 71  17 9f 0d e9 ac f0 76 65
42 5e 0d d9 86 55 84 42  3f 52 5d c2 b3 2d b0 84
1a e0 c1 b7 f9 fc 6c 2e  2c 2f 64 72 40 d6 17 cb
9c 4f 6a c9 27 ab fc cd  f5 16 f5 77 9d 8a a7 41
08 3f b2 1d ea d1 f7 71  bb 88 4d 6a 37 5d ef 46
74 3e ad 43 6e 23 4d 43  ca 84 26 13 ee dd ac a1
bb 7b 67 68 51 69 eb 24  7c c2 a7 8f 57 80 47 5c
```

Figura 4. Início do arquivo criptografado

```
44 45 41 52 43 52 59 21 00 01 00 00 00 79 17 D9 92 CE 2C 34 49
C3 68 20 B6 4C 8A 29 1C 9A 61 B4 30 88 9A 18 7D 3A 87 7B 11 C9
AC C9 81 36 AE D0 E2 CB F1 1C DC 92 FD 14 68 ED A7 F2 65 74 8B
50 F6 3F 60 79 99 18 19 EB 83 8B 34 92 04 99 84 37 1D 7B DE 6C
8C 5E DA 37 12 69 D7 5E D8 63 C6 DE 4C F2 83 6C 74 18 60 15 7E
67 93 32 30 94 AF 31 C3 65 FD 1E 35 79 C2 78 B8 45 EB 5F 16 A3
72 40 4E B1 33 BB 71 20 69 9D A9 C9 79 D4 4C 91 72 96 DA AC 6B
2D 00 C8 1C 8D 28 69 BE 43 4F 58 A4 D4 73 70 33 99 F1 42 2F FD
2B 0A EB 0F 90 2C A0 D1 BF EA A6 C1 0E 25 92 8A 0A 7D 27 46 04
FC 14 63 DB 03 A2 4F E4 08 F0 6D 6F B3 A4 FC A5 FF F4 25 9A C3
12 BF 7D 0E E4 41 60 D5 DB 23 E9 72 3F 3D E9 19 35 FB 84 10 67
C2 36 88 2F 82 31 D5 17 BB 99 36 10 6B 08 82 3D 58 16 7C 7B F0
B3 A3 DB DE 76 38 9A AB A7 93 04 36 51 98 83 BB 04 00 00 00 00
00 40 1F 00 00 00 00 C4 EA 0F 47 7F 0C EA BF 89 C5 1C 50 FD 2A
```

Figura 5. Cabeçalho adicionado ao arquivo criptografado