Extração Automatizada de Chaves Criptográficas em Ransomwares: Uma Abordagem Forense Baseada em AOB

Gabriel Alves Gama

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Monte Carmelo - MG
2025

Gabriel Alves Gama

Extração Automatizada de Chaves Criptográficas em Ransomwares: Uma Abordagem Forense Baseada em AOB

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Área de concentração: Sistemas de Informação

Orientador: Dr. Diego Nunes Molinos

Monte Carmelo - MG
2025

Este trabalho é dedicado a minha mãe que sempre esteve comigo a cada momento da minha vida me apoiando e fazendo com que fosse possível chegar até aqui, sem ela nada aconteceria.

Para o meu pai e avô que não viu este dia, mas que esteve presente também em meu coração.

Agradecimentos

Agradecer primeiramente a Deus, Jesus Cristo e Nossa Senhora, que me deram forças para chegar onde cheguei, pois tudo só foi possível graças as inúmeras formas de que Deus abençoou a minha vida para que fosse possível chegar até aqui e concluir este objetivo.
À minha querida mãe Juliana, que a amo de todo o coração, que me apoiou em cada etapa da minha vida e me ajudou a chegar até aqui, com todo o seu amor, força e dedicação.
A minha avó e tios e a todos os meus familiares que me apoiaram e estiveram presentes comigo nesta jornada.
Para aqueles que já partiram, meu pai e meu avô, fica aqui meu agradecimento e uma homenagem especial, às suas memórias que permanecem vivas em mim.
E aos amigos e amigas, que também estiveram comigo, e aqueles novos que o curso me proporcionou a conhecer, obrigado pelas risadas, momentos de tranquilidade e brincadeiras, vocês sempre estarão em meu coração. E também para aquelas pessoas especiais que estiveram na minha vida e que também contribuíram para chegar até aqui.
Ao meu orientador, pela paciência, orientação, ensinamentos e apoio que me deu.
Aos professores e professoras pelas oportunidades, ensinamentos e dedicação para me tornar e crescer como profissional.
Também meus agradecimentos a todos os colaboradores da UFU.
Para concluir, obrigado a cada pessoa que se fez presente em minha vida, e de alguma forma contribuiu para chegar até aqui e concluir está etapa em minha vida, muito obrigado a todos!!

"Entrega o teu caminho ao Senhor; confia nele, e ele o fará."
(Salmos 37:5)

Resumo

Ameaças do tipo *ransomware* têm-se consolidado como um dos principais vetores de ataques cibernéticos, apresentando crescimento contínuo tanto em incidência quanto na complexidade de suas técnicas. Relatórios recentes apontam impactos financeiros expressivos, com prejuízos que atingem milhões de dólares por incidente. Esses ataques empregam mecanismos criptográficos para cifrar arquivos em sistemas comprometidos, condicionando a recuperação dos dados ao pagamento de resgate, geralmente associado à disponibilização da chave ou do mecanismo de decifragem. Nesse contexto, este trabalho propõe um mecanismo de extração automatizada de chaves criptográficas de *ransomwares* por meio da análise de memória volátil durante a execução do *malware*. Para garantir controle sobre o comportamento interno do *malware* e viabilizar experimentos reprodutíveis, foi desenvolvida uma amostra própria de *ransomware*, inspirada conceitualmente em amostras tais como *WannaCry* e *LockBit*. A abordagem fundamenta-se no uso de assinaturas AOB (*Array of Bytes*) para identificar, em tempo de execução, rotinas associadas à geração e ao armazenamento temporário da chave criptográfica na memória volátil. O processo experimental envolveu análise estática com a ferramenta IDA Free, inspeção dinâmica de memória com *Cheat Engine* e a implementação de uma ferramenta automatizada, denominada *AOBTool*. Os resultados indicam que a chave pode ser localizada e extraída da memória antes de seu descarte, em tempo inferior ao necessário para a conclusão do processo de criptografia dos arquivos, evidenciando o potencial da técnica como suporte à computação forense e à mitigação de incidentes envolvendo *ransomware*.

Palavras-chave: Ransomware, Engenharia Reversa, Extração de Chaves, AOB, Análise de Memória.

Abstract

Ransomware threats have emerged as a primary vector for cyberattacks, with incidence and the complexity of their techniques continuing to grow. Recent reports highlight the significant financial impacts of these attacks, with losses reaching millions of dollars per incident. Ransomware typically employs cryptographic mechanisms to encrypt files on compromised systems, making data recovery contingent on the payment of a ransom, generally associated with the receipt of the cryptographic key or a decryption tool. In this context, this work proposes an automated mechanism for extracting cryptographic keys from ransomware by analyzing volatile memory during malware execution. To gain greater control over the malware's internal behavior and to enable reproducible experiments, a custom ransomware sample was developed, inspired by notable strains such as WannaCry and LockBit. The proposed approach leverages AOB (Array of Bytes) signatures to identify, in real-time, routines related to the generation and temporary storage of cryptographic keys in volatile memory. The experimental process involved static analysis with IDA Free, dynamic memory inspection with Cheat Engine, and the development of an automated tool, AOBTool. The results indicate that the cryptographic key can be located and extracted from memory before it is disposed of, within a timeframe shorter than that required to complete the file encryption process. This highlights the potential of the proposed technique as a valuable support tool for digital forensics and for mitigating ransomware-related incidents.

Keywords: Reverse Engineering; Key Extraction; Memory Forensics; AOB Signature; Ransomware.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AOB	Array of Bytes – Matriz de Bytes
API	Application Programming Interface – Interface de Programação de Aplicações
ASLR	Address Space Layout Randomization – Randomização do Layout do Espaço de Endereçamento
CPU	Central Processing Unit – Unidade Central de Processamento
DLL	Dynamic-Link Library – Biblioteca de Vínculo Dinâmico
FAT32	File Allocation Table 32 – Tabela de Alocação de Arquivos 32
FS	File System – Sistema de Arquivos
GUI	Graphical User Interface – Interface Gráfica do Usuário
HEX	Hexadecimal – Hexadecimal
IDA	Interactive Disassembler – Desassemblador Interativo
JSON	JavaScript Object Notation – Notação de Objetos JavaScript
MBR	Master Boot Record – Registro Mestre de Inicialização
MSVC	Microsoft Visual C++ – Compilador Microsoft Visual C++
NIST	National Institute of Standards and Technology – Instituto Nacional de Padrões e Tecnologia
OpenSSL	Open Secure Sockets Layer – Biblioteca Criptográfica de Código Aberto
RAM	Random Access Memory – Memória de Acesso Aleatório

REST	Representational State Transfer – Transferência Representacional de Estado
SHA	Secure Hash Algorithm – Algoritmo de Hash Seguro
SO	Operating System – Sistema Operacional
VM	Virtual Machine – Máquina Virtual
VMM	Virtual Machine Monitor – Monitor de Máquina Virtual
VS	Visual Studio – Microsoft Visual Studio
PE	Portable Executable – Formato de Arquivo Executável do Windows
Xref	Cross Reference – Referência Cruzada
ASCII	American Standard Code for Information Interchange – Código Padrão Americano para Intercâmbio de Informação
PID	Process Identifier – Identificador de Processo
RDP	Remote Desktop Protocol – Protocolo de Área de Trabalho Remota
AES	Advanced Encryption Standard – Padrão Avançado de Criptografia
RSA	Rivest–Shamir–Adleman – Algoritmo Criptográfico de Chave Pública
CryptoAPI	Cryptographic Application Programming Interface – API Criptográfica do Windows
RaaS	Ransomware as a Service – Ransomware como Serviço

Sumário

Capítulo 1

Introdução

O *ransomware* consolidou-se como uma das ameaças cibernéticas mais devastadoras e financeiramente mais danosas do cenário global. Segundo dados divulgados em abril de 2025 pela TechTarget, os ataques de *ransomware* continuam a crescer em frequência e sofisticação, gerando prejuízos significativos para organizações públicas e privadas em todo o mundo, e representou nos últimos anos mais de 20% de todas as violações de segurança reportadas globalmente, com destaque para o aumento no uso de técnicas avançadas de evasão e criptografia. (KERNER, 2025).

O crescimento dos ataques de *ransomware* tem sido acompanhado por impactos relevantes de ordem social e econômica. Em 2021, diversos ataques ganharam destaque internacional devido à sua escala e às consequências geradas fora do ambiente corporativo. De acordo com Splashtop Team (2025), o grupo *DarkSide* comprometeu os sistemas da *Colonial Pipeline*, o que resultou em interrupções no abastecimento de combustível na costa leste dos Estados Unidos e gerou preocupação entre consumidores e autoridades. De forma semelhante, a JBS USA, maior fornecedora de carne bovina do mundo, teve suas operações temporariamente interrompidas após um ataque do grupo REvil, o que afetou diretamente a cadeia de suprimentos alimentares. Esses casos evidenciam que ataques de *ransomware* podem provocar impactos que ultrapassam as organizações afetadas, atingindo setores essenciais da sociedade.

Diante do exposto, observa-se que ataques de *ransomware* não se limitam ao ambiente digital, produzindo impactos sociais relevantes e colocando em risco infraestruturas críticas. Além de causar paralisações operacionais e prejuízos financeiros significativos, esses ataques afetam a confiança de usuários e instituições nos sistemas de informação. A gravidade desse tipo de ameaça é agravada pela dificuldade de recuperar dados criptografados, especialmente na ausência de políticas de *backup* adequadas ou quando o pagamento do resgate não resulta na restauração completa das informações.

Neste contexto, evidencia-se a necessidade de desenvolver soluções capazes de atuar não apenas na detecção de *ransomware*, mas também na extração automatizada das chaves criptográficas utilizadas no processo de cifragem. Essa abordagem é essencial para

reduzir os impactos dos ataques, acelerar a recuperação dos sistemas comprometidos e fortalecer os processos de resposta a incidentes em segurança cibernética. Ante o exposto, este trabalho propõe um mecanismo de recuperação de chaves criptográficas baseado na análise de memória volátil e na utilização de padrões (AOB) para a identificação e extração de chaves utilizadas no processo de cifragem de *ransomware*.

1.1 Motivação

Os ataques de *ransomware* tornaram-se uma das ameaças cibernéticas mais prevalentes e lucrativas das últimas décadas, afetando desde usuários domésticos até grandes corporações e instituições governamentais. Relatórios de segurança apontam um aumento contínuo tanto na frequência quanto na sofisticação dessas campanhas, impulsionado pelo uso de plataformas de *Ransomware-as-a-Service* (RaaS) e pela automação dos vetores de ataque (Veeam Software, 2025; DeepStrike, 2025). Em 2025, foram observados crescimentos expressivos no número de incidentes e uma tendência de ataques cada vez mais rápidos e complexos, capazes de explorar vulnerabilidades em sistemas operacionais, redes corporativas e serviços em nuvem.

Apesar dos esforços e avanços nas defesas cibernéticas, a recuperação de arquivos cifrados por *ransomware* sem o pagamento do resgate permanece um dos maiores desafios para profissionais de TI e peritos forenses, especialmente quando as chaves criptográficas não podem ser obtidas por métodos tradicionais de análise ou quando backups não estão disponíveis ou são insuficientes (Veeam Software, 2025; DeepStrike, 2025).

Além da frequência crescente, os ataques de *ransomware* têm se tornado cada vez mais sofisticados. A IBM (IBM Security, 2023) destaca que os cibercriminosos passaram a adotar táticas de *dupla extorsão*, em que os dados são não apenas cifrados, mas também exfiltrados sob ameaça de vazamento, e `tripla extorsão`, que envolve ainda a extorsão de clientes e parceiros comerciais das vítimas. Mesmo organizações que possuem backups robustos ou pagam o resgate continuam vulneráveis a esses ataques mais agressivos.

Segundo o IBM Security (2024), o cenário de ameaças cibernéticas tem se caracterizado pelo aumento da velocidade e da complexidade dos ataques, com destaque para campanhas que exploram credenciais comprometidas e falhas em sistemas corporativos. O relatório evidencia que ataques de *ransomware* continuam figurando entre as ameaças mais relevantes observadas globalmente, exigindo respostas cada vez mais rápidas por parte das equipes de segurança.

Diante deste contexto, fica evidente a necessidade de mecanismos eficazes de prevenção e mitigação, uma vez que o curto intervalo entre a invasão inicial e a execução das ações maliciosas reduz significativamente a capacidade de reação das organizações. Além disso, estudos complementares da própria IBM indicam que incidentes envolvendo *ransomware* estão associados a custos elevados de recuperação, o que ressalta o impacto financeiro

expressivo desse tipo de ataque (IBM Security, 2024).

Diversos trabalhos na literatura propõem estratégias para mitigar os efeitos de ataques de *ransomware*, incluindo mecanismos de detecção antecipada, prevenção da execução maliciosa e análise comportamental em tempo real (KOLBITSCH et al., 2009; FRANCO; SOARES; SANTOS, 2024; DAVIES; MACFARLANE; BUCHANAN, 2020). Entretanto, poucos trabalhos dedicam-se à extração direta das chaves criptográficas utilizadas no processo de cifragem.

Diante desse contexto, este trabalho propõe uma abordagem forense baseada na análise da memória volátil de sistemas comprometidos, com o objetivo de recuperar tais chaves antes que os dados cifrados se tornem irrecuperáveis. A investigação dessa estratégia busca oferecer uma alternativa eficaz para responder a incidentes envolvendo *ransomware*.

1.2 Percepção da Problemática

A principal dificuldade imposta pelos *ransomwares* é a criptografia dos arquivos, que inviabiliza o acesso às informações sem a posse da chave utilizada para a criptografia. Mesmo com estratégias de mitigação, como *backups* regulares ou o bloqueio de execução, uma parte significativa das vítimas permanece suscetível a perdas severas de dados e financeiras.

Estudos recentes indicam que, durante a execução, *ransomwares* podem armazenar temporariamente suas chaves criptográficas na memória volátil do sistema. Em Mazur, Oliveira e Martimiano (2024), os autores identificaram e extraíram chaves AES-256 e vetores de inicialização diretamente da memória de um sistema infectado pelo *ransomware* DearCry, por meio de técnicas de engenharia reversa e de ferramentas como Volatility 3 e IDA. Os resultados obtidos reforçam o potencial da análise de memória volátil como abordagem forense viável para mitigar ataques de *ransomware*, especialmente no contexto da recuperação de dados criptografados.

Corroborando esses achados, Lee (2023) demonstra que grande parte dos *ransomwares* modernos utiliza APIs criptográficas amplamente difundidas, como CryptoAPI e OpenSSL, no processo de cifragem dos dados. O emprego recorrente dessas bibliotecas resulta em padrões previsíveis de alocação e manipulação de chaves criptográficas na memória volátil, o que amplia a viabilidade de técnicas forenses voltadas à identificação dessas bibliotecas. Esse comportamento reforça a necessidade de abordagens automatizadas capazes de explorar tais padrões de forma sistemática, complementando estratégias baseadas em análise manual ou específicas de famílias de *ransomwares*.

Dessa forma, este trabalho é guiado pelas seguintes questões de pesquisa:

QP1: É viável a extração de chaves criptográficas utilizadas por ransomwares a partir da análise da memória volátil, por meio da identificação de padrões recorrentes baseados em AOB?

QP2: Em que medida a utilização de padrões AOB permite automatizar o processo de identificação e extração de chaves criptográficas na memória volátil, reduzindo a dependência de análise manual e específica por família de ransomwares?

1.3 Objetivos

1.3.1 Objetivo Geral

Investigar a viabilidade da extração automatizada de chaves criptográficas de *ransomwares* por meio da análise de memória volátil e de técnicas de engenharia reversa, utilizando padrões binários baseados em *AOB* para identificar com precisão as estruturas responsáveis pela geração e armazenamento da chave.

Objetivos Específicos

Os objetivos específicos deste trabalho visam detalhar as etapas necessárias para a investigação e validação da abordagem proposta para a extração automatizada de chaves criptográficas em *ransomwares*. Para isso, busca-se fundamentar teoricamente os conceitos envolvidos, analisar o comportamento do *malware* em ambiente controlado, identificar padrões relevantes na memória volátil e desenvolver uma ferramenta capaz de automatizar o processo de extração. Por fim, pretende-se avaliar a eficiência da solução proposta por meio de experimentos controlados, comparando o tempo de extração da chave com o tempo de cifragem do ransomware.

- ❏ Revisar os principais conceitos relacionados a ataques de *ransomwares*, criptografia simétrica, ciclo de execução de *malware* e análise de memória, com base em estudos de caso reais;
- ❏ Analisar, por meio de ferramentas de engenharia reversa *(IDA Free, Scylla, Cheat Engine)*, a dinâmica de geração e de manipulação de chaves criptográficas em ambiente controlado;
- ❏ Identificar padrões recorrentes na memória que permitam localizar *buffers* ou instruções associadas à chave criptográfica;
- ❏ Implementar uma ferramenta automatizada capaz de realizar a varredura da memória, localizar o AOB definido e extrair a chave criptográfica em tempo de execução;
- ❏ Avaliar a eficácia da solução proposta com uma amostra de *ransomware* desenvolvida especificamente para este trabalho, garantindo controle total sobre o fluxo de execução, sem comprometer a validade dos resultados;

- ❏ Comparar os tempos de extração da chave com os de cifragem do *ransomware*, demonstrando se a técnica é suficientemente rápida e eficiente;

Com esses objetivos, busca-se contribuir para o avanço da computação forense e da segurança cibernética, oferecendo um método capaz de auxiliar na recuperação de dados e na análise de incidentes envolvendo *ransomwares*.

1.4 Contribuições

As principais contribuições deste trabalho podem ser resumidas da seguinte forma:

- ❏ Definição de um método sistemático de identificação de padrões binários (AOB) relacionados à geração e ao armazenamento da chave criptográfica.
- ❏ Implementação da ferramenta *AOBTool*, capaz de realizar varredura de memória, localizar instruções críticas e extrair a chave criptográfica antes que o *ransomware* a apague.
- ❏ Demonstração experimental de que a abordagem é eficiente e extrai a chave significativamente mais rápido do que o próprio *ransomware* executa sua rotina de cifragem.
- ❏ Fornecimento de um guia replicável de análise forense de memória com potencial de aplicação em pesquisas acadêmicas e no desenvolvimento de ferramentas de resposta a incidentes.

1.5 Organização da Monografia

Este trabalho está estruturado da seguinte forma:

- ❏ **Capítulo 1 – Introdução:** apresenta o problema, a motivação, os objetivos e o contexto no qual a pesquisa está inserida.
- ❏ **Capítulo 2 – Fundamentação Teórica:** descreve os conceitos essenciais para a compreensão da pesquisa, incluindo o funcionamento de *ransomwares*, técnicas de criptografia, análise de memória, engenharia reversa e estudo de casos representativos.
- ❏ **Capítulo 3 – Método:** detalha o ambiente experimental, a construção da amostra de *ransomware* utilizada, as ferramentas empregadas (IDA Free, Visual Studio, entre outras) e o processo de extração e validação dos padrões AOB e

- **Capítulo 4 – Desenvolvimento AOBTool:** apresenta a implementação completa da ferramenta *AOBTool*, explicando suas funções, arquitetura, fluxo lógico e justificando cada etapa do processo de detecção e extração da chave em memória.
- **Capítulo 5 – Experimentos e Análise dos Resultados:** apresenta a implementação completa da ferramenta *AOBTool*, explicando suas funções, arquitetura, fluxo lógico e justificando cada etapa do processo de detecção e extração da chave em memória.
- **Capítulo 6 – Conclusão:** apresenta as conclusões finais do trabalho, sintetizando os principais resultados alcançados, respondendo às questões de pesquisa, destacando as contribuições científicas da proposta.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica que sustenta o desenvolvimento deste trabalho de conclusão de curso, reunindo os principais conceitos, definições e trabalhos da literatura relacionados ao *ransomware* e à análise forense de memória volátil. Inicialmente, são discutidos os aspectos fundamentais dos *ransomwares*, incluindo seu funcionamento, variações, estágios de ataque e as principais famílias historicamente relevantes, de modo a contextualizar a ameaça e seus impactos no cenário da segurança cibernética.

Em seguida, são abordados os conceitos relacionados à memória volátil e às técnicas forenses aplicadas à sua análise, com ênfase na extração de artefatos temporários relevantes, como chaves criptográficas. Também são apresentadas técnicas de engenharia reversa e de varredura por padrões binários, em especial o uso de AOB, que fundamentam a metodologia proposta neste trabalho.

Por fim, o capítulo discute os principais trabalhos similares a esta proposta na literatura, destacando suas abordagens, limitações e contribuições, bem como as lacunas existentes que motivam a proposta deste estudo. Essa organização visa fornecer o embasamento conceitual e técnico necessário à compreensão da metodologia adotada e da contribuição científica apresentada nos capítulos subsequentes.

2.1 Ransomwares

Ransomware é um tipo de software malicioso (*malware*) que impede o acesso a sistemas ou arquivos por meio de criptografia, exigindo o pagamento de um resgate para restaurar o acesso. Conforme destacado por Microsoft (2024a), esse tipo de ataque geralmente é realizado por meio de campanhas de *phishing* ou de exploração de vulnerabilidades, sendo uma das formas mais comuns e lucrativas de crime cibernético na atualidade.

Ao infectar dispositivos, o *ransomware* pode bloquear completamente o sistema ou criptografar arquivos essenciais, deixando a vítima dependente de uma chave de descriptografia que só seria fornecida após o pagamento do resgate, geralmente em criptomoedas.

No entanto, mesmo com o pagamento, não há garantia de que o acesso seja restabelecido, o que torna o impacto potencialmente irreversível.

Além do impacto financeiro, esses ataques comprometem a integridade de dados sensíveis, afetando operações empresariais e até mesmo serviços públicos essenciais.

Como referência histórica, o primeiro caso documentado de *ransomware* ocorreu em 1989, com o *Trojan AIDS*, criado por Joseph Popp. Segundo Young e Yung (1996), o *malware* foi distribuído por meio de disquetes enviados a participantes de uma conferência da OMS, contendo:

- ❏ Um adesivo com a inscrição "Disquete introdutório de informações sobre AIDS";
- ❏ Instruções detalhadas de instalação;
- ❏ Um contrato de licença exigindo US$378 pelo "software";

Esse episódio marcou o surgimento inicial de uma prática que, ao longo das décadas, consolidou-se como uma das maiores ameaças cibernéticas globais.

De acordo com o guia da IBM Security sobre *ransomware* (IBM Security, 2023), um ataque típico passa por cinco estágios distintos:

1. **Acesso inicial**: O acesso inicialmente ocorre por meio de *phishing*, de exploração de vulnerabilidades conhecidas ou de comprometimento de protocolos de acesso remoto, como o *Remote Desktop Protocol* (RDP).

2. **Pós-exploração**: Após o acesso inicial, os invasores podem instalar ferramentas como *Remote Access Tools* (RAT) ou outros tipos de *malware* para garantir persistência e ampliar sua presença no sistema.

3. **Entendimento e expansão**: Nessa fase, os atacantes analisam o ambiente local, identificando alvos e oportunidades de movimentação lateral, técnica utilizada para acessar outros sistemas e domínios conectados à rede.

4. **Coleta e exfiltração de dados**: Os dados valiosos são identificados e exfiltrados, incluindo credenciais, informações pessoais e propriedade intelectual. Isso pode ser utilizado para aplicar a chamada "extorsão dupla", combinando criptografia com ameaça de vazamento.

5. **Implementação e extorsão**: Os arquivos são criptografados (ou o dispositivo é bloqueado, no caso de *ransomware* de bloqueio) e uma nota de resgate é apresentada à vítima. Essa notificação pode aparecer como um arquivo de texto ou uma janela pop-up, instruindo sobre o pagamento — geralmente em criptomoedas , para obtenção da chave de descriptografia ou restauração do sistema.

2.1.1 Variações de *Ransomwares*

As variações de *ransomware* são categorizadas de acordo com os métodos utilizados para a extorsão ou o impacto nas vítimas. A seguir, são descritas as principais formas identificadas segundo a (IBM Security, 2023).

Leakware/Doxware

Combina criptografia com a ameaça de vazamento de dados sensíveis. Diferentemente das versões iniciais, as atuais realizam ambas as ações simultaneamente.

Ransomware Móvel

Focado em dispositivos móveis, geralmente via:

- ❏ Aplicativos maliciosos;
- ❏ Downloads automáticos;

Segundo IBM Security (2023), como existe o backup via nuvem, muitos dos atacantes preferem utilizar o bloqueio de tela para bloquear o acesso ao dispositivo móvel.

Wipers

Variante que:

- ❏ Pode apagar dados permanentemente;
- ❏ Frequentemente associada a ataques hacktivistas de estados-nação;

Em alguns casos, mesmo após a vítima pagar o resgate aos atacantes, esse *ransomware* continua a destruir os dados.

Scareware

Utiliza engenharia social para:

- ❏ Simular alertas de entidades oficiais;
- ❏ Falsas acusações criminais;
- ❏ Alertas de vírus fictícios;

Esse tipo pode atuar como porta de entrada para outros *malwares*, pois esse tipo de *ransomware* tem como objetivo assustar a vítima com falsas alegações, fazendo a mesma até comprar um *ransomware* que se passa por software de antivírus.

***Ransomware* como Serviço (RaaS)**

O *Ransomware as a Service* (RaaS) é um modelo de operação no qual desenvolvedores disponibilizam infraestruturas e códigos de *ransomware* para terceiros, denominados afiliados, que executam os ataques. Esse modelo contribui para a disseminação de campanhas de *ransomware*, ao reduzir a barreira técnica necessária para a realização de ataques e permitir a reutilização de códigos, técnicas e infraestruturas de extorsão (SECURITY, 2024).

Variantes de *Ransomwares* Notáveis

Diversas famílias de *ransomware* se destacaram ao longo dos anos por sua sofisticação, impacto ou inovação nos métodos de ataque. Abaixo, destacam-se algumas das mais relevantes de acordo com a (IBM Security, 2023).

CryptoLocker

Lançado em setembro de 2013, o CryptoLocker é considerado um marco no surgimento do *ransomware* moderno. Utilizando uma botnet para sua disseminação, essa versão foi uma das primeiras a adotar criptografia robusta para sequestrar arquivos de usuários. Estima-se que tenha arrecadado cerca de 3 milhões de dólares antes de ser neutralizado em 2014 por esforços internacionais. Seu sucesso inspirou uma nova geração de *ransomware*s, como o *WannaCry* e o *Petya* (IBM Security, 2023).

WannaCry

O *WannaCry* explorava a vulnerabilidade *EternalBlue* no Windows. Em 2017, ele infectou mais de 200 mil sistemas em mais de 150 países, exigindo pagamento em criptomoeda sob ameaça de exclusão permanente dos dados. O prejuízo global é estimado em até 4 bilhões de dólares (IBM Security, 2023).

Petya e NotPetya

Diferente de *ransomware*s convencionais, o Petya danificava diretamente a tabela mestra de arquivos (MFT), impossibilitando a inicialização do sistema operacional. O NotPetya, uma variação ainda mais destrutiva lançada em 2017, foi utilizado em um ataque cibernético de grande escala, sendo incapaz de restaurar os dados mesmo após pagamento, caracterizando-se como um *wiper* (IBM Security, 2023).

Ryuk

Detectado em 2018, o *Ryuk* é uma família de *ransomware* conhecida por ataques altamente direcionados contra grandes organizações, geralmente associados a pedidos de resgate elevados. Diferentemente de campanhas automatizadas, o Ryuk caracteriza-se por exigir esforço manual significativo, incluindo reconhecimento prévio da infraestrutura da vítima e movimentação lateral antes da execução da carga maliciosa.(Cloudflare, 2024).

DarkSide

O *DarkSide* é uma família de *ransomware* identificada a partir de 2020, conhecida por ataques direcionados contra grandes corporações e infraestruturas críticas. O grupo opera sob o modelo de RaaS, disponibilizando o *malware* e a infraestrutura de ataque para afiliados em troca de uma participação nos lucros obtidos com os resgates.

Os ataques associados ao DarkSide exploram, em geral, acessos remotos inadequadamente protegidos, como serviços RDP expostos, credenciais fracas e configurações incorretas de segurança em sistemas Windows e Linux. O grupo ganhou ampla notoriedade em 2021 após o comprometimento da empresa Colonial Pipeline, evidenciando o potencial impacto desse tipo de *ransomware* em serviços essenciais e reforçando sua relevância no cenário de ameaças cibernéticas contemporâneas (Akamai, 2024).

Locky

O *Locky* é uma família de *ransomware* identificada a partir de 2016, conhecida por campanhas de disseminação em larga escala baseadas em engenharia social. Seu principal vetor de infecção consistia no envio de e-mails contendo documentos do Microsoft Word com macros maliciosas, que, quando ativadas pela vítima, realizavam o download e a execução do *ransomware*. Uma vez em execução, o Locky criptografava uma ampla variedade de arquivos do sistema, renomeando-os com extensões específicas e exibindo uma nota de resgate com instruções para pagamento, caracterizando um dos primeiros exemplos de campanhas massivas e automatizadas de *ransomware* (BELCIC, 2019).

LockBit

O LockBit é uma família de *ransomware* voltada principalmente a ataques direcionados contra empresas e organizações governamentais, caracterizando-se pela alta automação, rápida propagação em redes corporativas e uso de técnicas de dupla extorsão. Operando sob o modelo de RaaS, o LockBit emprega criptografia forte para bloquear os dados das vítimas e exige pagamento de resgate em troca da chave de decifragem, mantendo-se como uma das ameaças mais relevantes e ativas no cenário de *ransomware* moderno, apesar de ações de repressão contra seus operadores (Kaspersky, 2024).

2.2 Memória Volátil - RAM

A memória volátil (RAM) armazena informações temporárias essenciais durante a execução de programas, incluindo *ransomwares*, sendo perdidas após o desligamento do sistema. De acordo com as diretrizes do NIST, a RAM pode conter dados forenses relevantes, tais como processos em execução, conexões de rede ativas, comandos recentemente executados, estruturas internas de programas e dados sensíveis manipulados em tempo de execução National Institute of Standards and Technology (2006), Ligh et al. (2011).

A análise de memória volátil permite identificar, entre outros artefatos (National Institute of Standards and Technology, 2006; CASEY, 2011; LIGH et al., 2011):

- ❏ Processos maliciosos em execução e módulos carregados na memória;
- ❏ Chaves criptográficas temporárias utilizadas durante a cifragem;
- ❏ Estruturas de dados internas empregadas pelo *malware*;
- ❏ Sessões de rede ativas e comunicações suspeitas;
- ❏ Histórico recente de comandos e comportamento do usuário.

Esse tipo de análise é fundamental para compreender a atuação de *ransomware*s em tempo de execução, muitas vezes revelando artefatos que não estão disponíveis no sistema de arquivos persistente.

2.2.1 Extração de Informação da Memória

A escolha pelo uso de assinaturas do tipo *Array of Bytes* (AOB), fundamenta-se em vantagens técnicas relevantes para o contexto forense. Diferentemente de abordagens baseadas em símbolos, strings ou informações de depuração, as assinaturas AOB operam diretamente sobre padrões binários, mantendo sua aplicabilidade mesmo em executáveis desprovidos de símbolos ou submetidos a processos de ofuscação leve. Além disso, o uso de máscaras com curingas permite abstrair variações introduzidas por mecanismos como *Address Space Layout Randomization* (ASLR) e realocação de módulos, garantindo maior robustez da técnica entre diferentes execuções do mesmo binário.

Neste trabalho, as assinaturas AOB não são tratadas como regras estáticas de detecção, mas como descritores binários de comportamento criptográfico, capazes de capturar invariantes estruturais observados durante a execução do ransomware.

2.2.1.1 *Array of Bytes* (AOB)

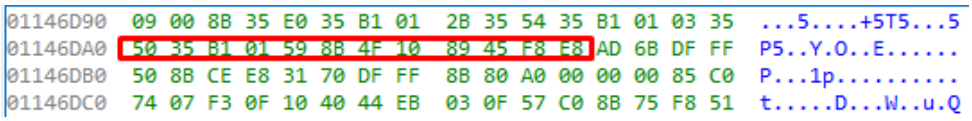
A técnica conhecida como *Array of Bytes* (AOB) consiste na identificação de padrões específicos de bytes na memória que podem indicar a presença de estruturas relevantes, como chaves criptográficas, bibliotecas de compressão ou trechos de código malicioso.

Essa abordagem é amplamente utilizada em análises forenses de memória e engenharia reversa, sendo aplicada com o auxílio de ferramentas como o *YARA*, que permite a definição de regras baseadas em assinaturas binárias, incluindo instruções, *strings* e padrões hexadecimais, para a detecção de artefatos maliciosos em memória (LIGH et al., 2011; ÁLVAREZ, 2013).

Por exemplo, ao buscar uma chave criptográfica AES-256 em um processo, é possível identificar padrões recorrentes de inicialização de algoritmos ou sequências de bytes associadas a bibliotecas como OpenSSL. Mesmo em diferentes execuções ou sistemas, certas assinaturas permanecem consistentes devido à reutilização de rotinas criptográficas padronizadas (FRANCO; SOARES; SANTOS, 2024).

Ferramentas de engenharia reversa, como o IDA Free, com o auxílio de plugins especializados, permitem extrair essas sequências diretamente do código compilado, gerando assinaturas AOB que podem ser utilizadas em varreduras de memória. A Figura 1 ilustra um exemplo de padrão AOB extraído de um binário analisado.

Figura 1 – Representação ilustrativa de um padrão AOB na memória



Fonte: elaborado pelo autor.

2.2.2 Engenharia Reversa

A engenharia reversa é essencial para entender o funcionamento interno de softwares maliciosos, como *ransomware*s, especialmente quando não se dispõe do código-fonte original. Essa técnica envolve a análise de binários compilados com o objetivo de reconstruir, compreender ou documentar seu comportamento. No contexto de segurança cibernética, ela é amplamente empregada para examinar vulnerabilidades, identificar padrões de funcionamento e desenvolver estratégias de defesa (AREMO, 2021; FRANCO; SOARES; SANTOS, 2024).

Entre as ferramentas mais utilizadas nesse processo de engenharia reversa está o IDA, na sua versão Free ou PRO, um *disassembler* interativo que permite inspecionar o código de baixo nível de executáveis. Com ele, é possível aplicar técnicas como a análise de fluxo de controle, a inspeção de chamadas de função e a identificação de trechos responsáveis por operações criptográficas ou maliciosas (Hex-Rays, 2025). A importância do IDA Free é reforçada pelo seu uso por agências governamentais, como a *CISA*, o *FBI* e o *DoD*, que o empregaram na análise do *malware* Taidoor, associado a campanhas cibernéticas patrocinadas por atores estatais (CISA, 2020).

2.3 Trabalhos Relacionados

Em (MAZUR; OLIVEIRA; MARTIMIANO, 2024), os autores realizaram um experimento prático de análise de memória volátil com o objetivo de identificar e extrair chaves criptográficas utilizadas por *ransomwares*, especificamente da família *Dearcry*. O estudo explorou técnicas de análise estática e dinâmica aplicadas à memória de sistemas Windows, utilizando ferramentas como o Volatility 3 e o disassembler IDA. A abordagem adotada permitiu identificar buffers contendo chaves e vetores de inicialização alocados na pilha, explorando o conhecimento da estrutura da memória virtual e o uso de algoritmos baseados em entropia para localizar regiões suspeitas. A chave AES de 256 bits e o IV utilizados na criptografia foram localizados com sucesso, permitindo a recuperação de um arquivo de 500 MB originalmente criptografado. Embora a metodologia se mostre inviável em ambientes reais, por exigir a aquisição precisa da memória no momento da execução do *ransomware*, o trabalho demonstra o potencial da análise de memória aliada à engenharia reversa como ferramenta de mitigação de danos em ataques de *ransomware*.

Em (LEE, 2023), os autores propuseram um mecanismo para detectar e identificar automaticamente módulos criptográficos utilizados em *ransomwares*, especialmente os desenvolvidos com CryptoAPI e OpenSSL. Por meio de engenharia reversa de binários executáveis em ambientes Windows, o estudo emprega técnicas como análise de padrões de código, além de regras *YARA* (ÁLVAREZ, 2013) para automatizar o processo. A abordagem demonstrou melhorias na precisão da detecção e na redução de falsos positivos, destacando-se como uma contribuição relevante e complementar à proposta deste estudo, que também busca identificar métodos criptográficos por meio de análise de padrões em memória.

Já em (PLOSZEK; ŠVEC; DEBNÁR, 2021), os autores realizaram uma análise detalhada de esquemas de criptografia empregados por *ransomwares* modernos, avaliando 10 amostras distintas de famílias como Ryuk, GandCrab, LockBit e Nemty. O estudo identificou quatro esquemas criptográficos principais, combinando criptografia simétrica (geralmente AES) com assimétrica (RSA), além de diferentes estratégias de geração e armazenamento de chaves. Os resultados reforçam a importância de compreender os mecanismos criptográficos utilizados por *ransomwares*. E entender como a maioria destes *ransomwares* agem é de suma importância para o prosseguimento deste estudo.

Em (JONES; SHASHIDHAR, 2017), os autores exploraram mais a fundo o *ransomware WannaCry* por meio de técnicas estáticas e dinâmicas, com foco na engenharia reversa de seu comportamento em ambientes Win32. O estudo identificou o uso extenso da API criptográfica do Windows (CryptAPI), como as funções `CryptGenKey` e `CryptEncrypt`, utilizadas para gerar e aplicar chaves de cifragem. Além disso, foi desenvolvida uma ferramenta protótipo de defesa baseada na verificação de hashes e no bloqueio de processos suspeitos, destacando a importância de abordagens automatizadas para mitigar esse tipo de ameaça. Embora com objetivos distintos, a proposta dos autores

alinha-se a este estudo no uso de engenharia reversa para a compreensão e neutralização de *ransomwares*.

Uma abordagem extensiva sobre a análise de códigos maliciosos no ambiente Windows foi desenvolvida por (MARINHO et al., 2022), que propõe uma metodologia estruturada que abrange desde técnicas automatizadas até engenharia reversa avançada. O estudo apresenta uma investigação utilizando ferramentas como Process Monitor, Wireshark e Noriben, e ressalta a importância de ambientes isolados para garantir a segurança da análise. Apesar de o foco principal estar no *malware* bancário, diversas práticas e ferramentas discutidas têm aplicação direta na investigação de *ransomwares*, especialmente na identificação de indicadores de comprometimento (IOC) e de *payloads* carregados em memória. Essa contribuição reforça o valor da análise de comportamento associada à extração em memória na mitigação de ameaças.

2.4 Síntese dos Trabalhos Correlatos

A Tabela 1 apresenta um resumo comparativo dos trabalhos relacionados mais relevantes considerados nesta pesquisa, destacando características técnicas e de avaliação que são pertinentes à proposta deste trabalho.

A análise dos trabalhos relacionados evidencia que, embora existam pesquisas consolidadas sobre engenharia reversa e análise de memória aplicada a *ransomwares*, há lacunas relevantes que este estudo busca preencher. Como mostrado na Tabela 1, a maioria das abordagens anteriores foca em análise estática ou em extração pontual de informações criptográficas sem integrar técnicas de detecção em tempo real e utilização conjunta de múltiplas ferramentas de análise neste contexto, a proposta deste trabalho se diferencia por:

- ❏ Implementar a extração de informações criptográficas em tempo real diretamente da memória do sistema infectado;
- ❏ Utilizar padrões de código (AOB) e análise de comportamento para identificar métodos criptográficos de maneira automatizada;
- ❏ Integrar ferramentas como IDA e memória volátil em um fluxo contínuo de análise e mitigação;

Tabela 1 – Comparativo sintético dos trabalhos relacionados e da proposta deste trabalho

Trabalho	**Extração reportada**	**Extração em tempo real**	**AOB / Padrões**	**Análise estática**	**Análise dinâmica**	**Ferramenta criada**
Mazur; Oliveira; Martimiano (2024)	✓	✗	✗	✓	✓	✗
Lee (2023)	✗	✗	✓	✓	✗	✓
Ploszek; Švec; Debnár (2021)	✗	✗	✗	✓	✓	✗
Jones; Shashidhar (2017)	✗	✗	✗	✓	✓	✓
Marinho et al. (2022)	✗	✗	✗	✓	✓	✗
Esta Proposta	✓	✓	✓	✓	✓	✓

Capítulo 3

Método

O método adotado neste trabalho combina técnicas de análise forense de memória volátil e engenharia reversa com o uso de varredura por AOB, permitindo a identificação e extração de chaves criptográficas utilizadas por *ransomware* durante sua execução. A abordagem experimental proposta permite avaliar, de forma controlada e reprodutível, a viabilidade dessa técnica como mecanismo de apoio à resposta a incidentes, contribuindo para a mitigação de danos e para o fortalecimento das práticas de segurança cibernética.

Esta pesquisa está estruturada em etapas sucessivas: (a) revisão dos principais métodos de análise de *ransomwares* e de extração de chaves; (b) construção de um ambiente virtual isolado, em máquinas Windows 10 sob Oracle VirtualBox, desligadas da rede de internet; (c) execução controlada de amostra de *ransomware* (amostra sintética), captura da memória de processo e análise estática/dinâmica para identificação de rotinas e geração de assinaturas AOB; (d) desenvolvimento de uma ferramenta de varredura (*AOBTool*) para localizar, em dumps ou processos em execução, os padrões associados às chaves; e (e) avaliação da abordagem. A seguir, as etapas serão detalhadas.

3.1 Ambiente Experimental

O ambiente foi preparado com *snapshots* limpos e com mecanismos de contenção de rede, de modo a evitar qualquer propagação indevida do código malicioso. Essa configuração garante segurança, reprodutibilidade dos experimentos e controle total do estado do sistema durante a execução das amostras.

Os experimentos foram executados em um laboratório virtual isolado, composto por máquinas virtuais gerenciadas pelo Oracle VirtualBox. Cada máquina virtual foi configurada com Windows 10 22H2 (instalado a partir de ISO), 4 096 MB de RAM, 1 vCPU e 50 GB de disco. As VMs foram restauradas a partir de snapshots limpos antes de cada execução, para garantir a reprodutibilidade e a contenção de amostras maliciosas. Todo o tráfego de rede foi bloqueado ou roteado para uma rede isolada e não havia conexão com a internet pública durante as execuções.

3.1.1 Instrumentação e Ferramentas

As ferramentas utilizadas no experimento e suas respectivas funções foram:

- **VirtualBox 7.2.2** (Oracle Corporation, 2025) plataforma de virtualização para criação e snapshot das máquinas de teste.
- **Windows 10 22H2** (Microsoft, 2024b) sistema operacional alvo das execuções experimentais.
- **IDA Free 9.2** (Hex-Rays, 2025) disassembler/depurador usado para análise estática e identificação de rotinas criptográficas no binário; será citado como *IDA* após a primeira menção.
- **Scylla v0.9.8** (NtQuery, 2015) ferramenta para dumping/reconstrução de PE em memória.
- **IDA-Fusion (plugin)** (senator715, 2024) plugin auxiliar para geração de array of byte (AOB).
- **Cheat Engine 7.6** (Cheat Engine Team, 2024) utilizada para inspeção dinâmica e varredura de padrões em memória durante testes de validação e comportamento de memória do *ransomware*.
- **Visual Studio Community 2022** (Microsoft, 2022) ambiente de desenvolvimento para compilação de códigos auxiliares e da ferramenta própria.
- **Ferramenta própria de varredura AOB** (implementada em C++) desenvolvida pelo autor para localizar padrões AOB e extrair chaves em processos alvo (descrita na Seção 3.2.2) *AOBTool*.

3.2 Extração da Chave Criptográfica da Amostra Sintética

3.2.1 Amostra Sintética Desenvolvida

Para a validação experimental da técnica proposta, optou-se pela utilização de uma amostra de *ransomware* desenvolvida especificamente no contexto deste trabalho. A amostra foi implementada em linguagem *C++*, direcionada à arquitetura *x86*, e concebida a partir da análise conceitual e técnica de mecanismos amplamente documentados na literatura e observados em famílias reais de *ransomware*, com destaque para o *WannaCry* e o *LockBit*, ambos discutidos no capítulo de fundamentação teórica.

Embora os experimentos tenham sido conduzidos com uma amostra sintética, o foco desta pesquisa não reside em características específicas de uma família de *ransomware*, mas sim no comportamento intrínseco ao uso de bibliotecas criptográficas e à manipulação temporária de chaves na memória volátil. Esse comportamento decorre de propriedades fundamentais dos mecanismos criptográficos empregados e independe da origem exata do binário analisado.

A amostra sintética desenvolvida preserva comportamentos essenciais observados em *ransomwares* reais, tais como: geração dinâmica de chaves criptográficas em tempo de execução, utilização dessas chaves em rotinas de cifragem de arquivos e descarte do material sensível após a conclusão da operação, e metódos de ofuscação. Dessa forma, foi possível reproduzir de maneira fidedigna o comportamento da classe de *malware* analisada, mantendo o foco no fenômeno investigado a permanência temporária da chave criptográfica na memória volátil sem incorrer em riscos éticos, legais ou operacionais associados ao uso de binários maliciosos autênticos.

A implementação da amostra sintética proporcionou uma maior previsibilidade e controle sobre os experimentos. Ela permitiu a análise de pontos críticos, como a geração e o descarte da chave, que são altamente variáveis. Por exemplo, a implementação do descarte da chave é algo que, em *ransomwares* reais, pode ocorrer de maneira não documentada e sem um padrão claro. A partir de nossa amostra, foi possível observar e quantificar com precisão os efeitos da intervenção forense antes do término da operação do ransomware.

Ressalta-se que a utilização de uma amostra controlada não compromete a validade nem a generalidade dos resultados obtidos. A técnica de varredura baseada em assinaturas AOB atua diretamente sobre padrões binários presentes na memória do processo, independentemente da origem do código analisado. Assim, o método proposto é aplicável tanto a amostras sintéticas quanto a *ransomwares* reais, desde que os padrões identificados estejam presentes durante a execução. O uso da amostra própria garante, portanto, segurança experimental, reprodutibilidade dos testes e precisão na análise, atendendo plenamente aos objetivos da pesquisa.

3.2.2 Processo de Extração de Chaves

O processo proposto para a extração de chaves criptográficas em *ransomwares* segue as seguintes etapas:

1. Identificação do processo correspondente ao *ransomware* em execução;
2. Captura da memória volátil (RAM) do sistema infectado;
3. Análise da memória do processo utilizando a técnica de varredura por AOB.
4. Extração e validação da chave criptográfica identificada na memória.

Essa abordagem é fundamentada em técnicas forenses, que demonstram a eficácia da análise de memória em tempo de execução para a identificação de artefatos, como chaves e estruturas criptográficas (LIGH et al., 2011).

3.3 Desenvolvimento da Ferramenta AOBTool

A partir das assinaturas AOB identificadas durante a análise estática e dinâmica da amostra de *ransomware*, foi desenvolvida a ferramenta denominada *AOBTool*. O objetivo da ferramenta é automatizar a etapa de varredura de memória volátil, reduzindo a dependência de inspeções manuais e aumentando a precisão, a repetibilidade e a confiabilidade do processo de extração da chave criptográfica.

A *AOBTool* foi projetada para operar diretamente sobre processos em execução, realizando a identificação do módulo alvo, a varredura de regiões específicas da memória do processo alvo e a leitura controlada de *buffers* associados às assinaturas previamente validadas. A ferramenta adota uma abordagem não intrusiva, limitando-se à leitura da memória, sem modificar o código ou o estado do processo analisado.

O desenvolvimento priorizou simplicidade arquitetural, eficiência temporal e clareza operacional, características essenciais para aplicações em contextos forenses e experimentais. A estratégia baseada em assinaturas permite explorar janelas temporais nas quais informações sensíveis, como chaves criptográficas temporárias, ainda permanecem na memória volátil durante o ciclo de cifragem do *ransomware*. Uma vez identificado o padrão correspondente, a ferramenta realiza a extração dos dados associados, registrando-os para análise posterior.

Cabe destacar que, embora a ferramenta seja apresentada neste capítulo em nível conceitual, os detalhes de implementação, a arquitetura interna e as decisões de projeto são discutidos de forma aprofundada no capítulo dedicado ao desenvolvimento da solução, conforme descrito na Seção 4. Nesta seção, o foco concentra-se na contextualização da ferramenta dentro do fluxo experimental e na sua integração com a metodologia proposta.

3.4 Avaliação da Abordagem Proposta

Por fim, a abordagem foi avaliada quanto à sua eficácia na identificação e extração das chaves criptográficas em uma amostra sintética de *ransomware*, considerando critérios como a taxa de sucesso, a precisão na localização dos padrões, o impacto no tempo de resposta e a viabilidade de aplicação em cenários forenses. Os resultados obtidos foram analisados de forma qualitativa e quantitativa, o que permitiu discutir as contribuições, as limitações e as possibilidades de extensão da técnica proposta.

Embora os experimentos tenham sido repetidos múltiplas vezes para reduzir efeitos pontuais, não foram aplicadas análises estatísticas avançadas, uma vez que o objetivo prin-

cipal consistiu em verificar a viabilidade e a temporalidade relativa da extração da chave. Estudos futuros podem ampliar o número de execuções e empregar métricas estatísticas mais robustas para aprofundar a análise quantitativa.

Capítulo 4

Desenvolvimento AOBTool

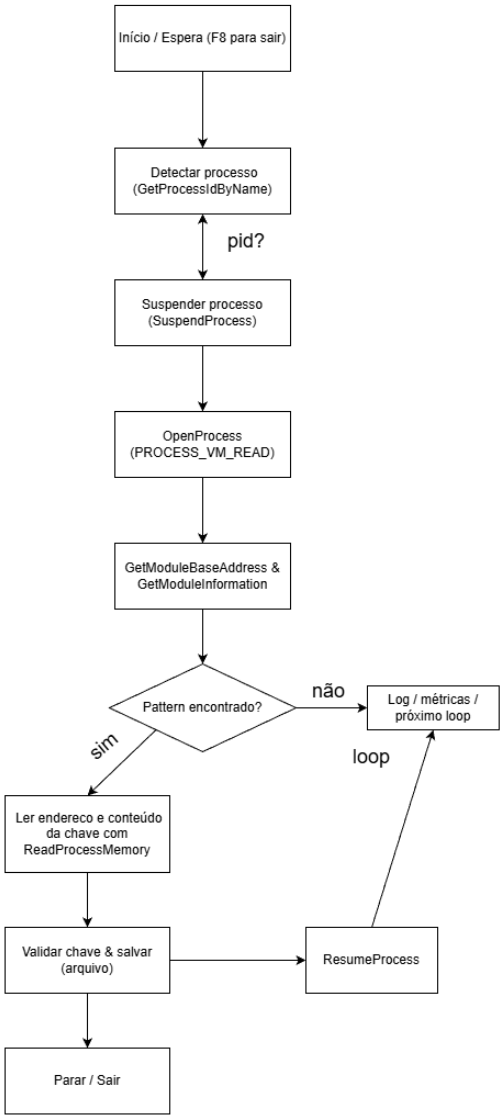
Nesta seção, é apresentada a implementação da ferramenta *AOBTool*, desenvolvida e utilizada nos experimentos deste trabalho para a varredura de memória e a extração de chaves criptográficas. A Figura 2 ilustra o fluxo geral de funcionamento da ferramenta, evidenciando as etapas de identificação do processo-alvo, varredura por padrões *Array of Bytes* (AOB) e materialização da chave criptográfica extraída.

A *AOBTool* foi projetada com foco em simplicidade, eficiência temporal e controle forense do processo analisado. Diferentemente de ferramentas genéricas de análise de memória, sua arquitetura prioriza a extração rápida de artefatos críticos antes que o *ransomware* finalize sua rotina de criptografia e elimine a chave da memória. Para isso, optou-se por uma implementação de baixo nível, baseada diretamente nas APIs do sistema operacional Windows, minimizando sobrecarga e dependências externas.

Embora existam algoritmos mais sofisticados para a busca de padrões, como o descrito em Boyer e Moore (1977) ou técnicas baseadas em paralelismo, optou-se por uma abordagem linear devido à previsibilidade, facilidade de depuração e baixo impacto computacional no contexto experimental adotado. Em ambientes forenses, a transparência e a confiabilidade do método podem ser mais relevantes do que a otimização extrema de desempenho.

Novamente, torna-se importante ressaltar que, neste trabalho, as assinaturas AOB não são tratadas como regras estáticas de detecção, mas como descritores binários de comportamento criptográfico, capazes de capturar invariantes estruturais observados durante a execução do *ransomware*.

Figura 2 – Fluxo geral da ferramenta de varredura e extração (visão lógica)



Fonte: elaborado pelo autor.

4.1 FindPatternInProcess

A função `FindPatternInProcess` é responsável por percorrer as regiões de memória acessíveis de um processo em execução, realizando a comparação sequencial entre os bytes lidos da memória e o padrão AOB previamente definido. O objetivo dessa varredura é localizar a ocorrência exata do padrão associado a estruturas relevantes do *ransomware*, como *buffers* ou instruções relacionadas à chave criptográfica. Quando o padrão é identificado, a função retorna o endereço de memória correspondente, permitindo a extração

controlada dos dados de interesse.

Listagem 4.1 – Função FindPatternInProcess

```
DWORD FindPatternInProcess(HANDLE hProcess, DWORD base, DWORD size,
    const char* pattern, const char* mask) {
    SIZE_T patternLength = strlen(mask);
    std::vector<char> buffer(size);
    SIZE_T bytesRead;

    if (!ReadProcessMemory(hProcess, (LPCVOID)base, buffer.data(), size,
        &bytesRead)) {
        return 0xBEEE;
    }

    for (DWORD i = 0; i < size - patternLength; i++) {
        bool found = true;
        for (DWORD j = 0; j < patternLength; j++) {
            if (mask[j] != '?' && pattern[j] != buffer[i + j]) {
                found = false;
                break;
            }
        }
        if (found) return base + i;
    }

    return 0xBEEE;
}
```

A estratégia de varredura adotada apresenta complexidade linear em função do tamanho da região de memória analisada e do comprimento do padrão AOB. Embora essa abordagem não seja otimizada para grandes volumes de dados, ela se mostra adequada ao contexto forense considerado neste trabalho, no qual a análise é direcionada a regiões específicas de memória associadas a módulos carregados pelo ransomware, reduzindo significativamente o espaço de busca efetivo.

4.1.1 Descrição linha a linha

- Declara as variáveis de controle: o comprimento do padrão é obtido via `strlen(mask)`;
- Cria um vetor `buffer` com o tamanho total do módulo a ser lido;
- Usa `ReadProcessMemory` para copiar o conteúdo de memória do processo remoto;
- Caso a leitura falhe, retorna o valor especial `0xBEEE`, usado para indicar erro;
- Percorre byte a byte a memória lida, comparando com o padrão e respeitando os coringas da máscara;

- ❏ Quando encontra coincidência total, retorna o endereço absoluto (`base + i`);
- ❏ Se nada for encontrado, retorna novamente `0xBEEE`;

Essa função é essencial para identificar o ponto exato do código em memória onde a chave criptográfica foi referenciada.

4.1.2 Considerações técnicas

A estratégia adotada consiste em uma varredura linear sobre o espaço de memória do módulo alvo. Apesar de sua simplicidade, essa abordagem apresenta complexidade $O(n \cdot m)$, onde n representa o tamanho do módulo e m o tamanho do padrão. Considerando que os módulos analisados possuem tamanho limitado e que a execução ocorre uma única vez por instância do *ransomware*, o impacto computacional mostrou-se desprezível nos experimentos realizados.

Optou-se por esse método devido à facilidade de implementação, previsibilidade de comportamento e compatibilidade com máscaras contendo coringas, característica essencial para lidar com variações introduzidas por otimizações do compilador e técnicas de ofuscação.

4.2 GetModuleBaseAddress

Essa função obtém o endereço base de um módulo carregado em um processo.

Listagem 4.2 – Função GetModuleBaseAddress

```
1  DWORD GetModuleBaseAddress(DWORD pid, const std::wstring& moduleName) {
2      DWORD baseAddress = 0;
3      HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPMODULE |
           TH32CS_SNAPMODULE32, pid);
4      if (hSnap != INVALID_HANDLE_VALUE) {
5          MODULEENTRY32W me32 = { sizeof(me32) };
6          if (Module32FirstW(hSnap, &me32)) {
7              do {
8                  if (moduleName == me32.szModule) {
9                      baseAddress = (DWORD)me32.modBaseAddr;
10                     break;
11                 }
12             } while (Module32NextW(hSnap, &me32));
13         }
14         CloseHandle(hSnap);
15     }
16     return baseAddress;
17 }
```

- Usa a função `CreateToolhelp32Snapshot` com as flags `TH32CS_SNAPMODULE` e `TH32CS_SNAPMODULE32` para capturar os módulos do processo.
- Itera sobre os módulos com `Module32FirstW` e `Module32NextW`.
- Quando o nome do módulo (`me32.szModule`) coincide com o argumento informado, o endereço base (`modBaseAddr`) é retornado.

4.2.1 Relação com ASLR

A obtenção dinâmica do endereço base do módulo é essencial devido à presença do mecanismo de ASLR nos sistemas Windows modernos. Esse mecanismo faz com que o endereço de carregamento do executável varie a cada execução, inviabilizando abordagens baseadas em endereços fixos. Dessa forma, a combinação entre endereço base dinâmico e varredura por AOB torna a técnica robusta frente à aleatorização de memória.

4.3 GetProcessIdByName

Essa função busca na lista de processos em execução o identificador (*PID*) associado a um executável específico.

Listagem 4.3 – Função GetProcessIdByName

```
DWORD GetProcessIdByName(const std::wstring& processName) {
    DWORD pid = 0;
    HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (hSnap != INVALID_HANDLE_VALUE) {
        PROCESSENTRY32W pe32 = { sizeof(pe32) };
        if (Process32FirstW(hSnap, &pe32)) {
            do {
                if (processName == pe32.szExeFile) {
                    pid = pe32.th32ProcessID;
                    break;
                }
            } while (Process32NextW(hSnap, &pe32));
        }
        CloseHandle(hSnap);
    }
    return pid;
}
```

- Cria um *snapshot* de todos os processos ativos com `CreateToolhelp32Snapshot`;
- Itera os resultados com `Process32FirstW` e `Process32NextW`;
- Quando o nome do executável coincide com o desejado, retorna o `ProcessID`;

4.3.1 Limitações conhecidas

A identificação do processo pelo nome do executável é adequada ao ambiente experimental controlado deste trabalho. Em cenários reais, *ransomwares* podem empregar técnicas como *process hollowing*, nomes aleatórios ou injeção em processos legítimos. No entanto, como o objetivo do estudo é validar a viabilidade da extração de chaves em memória, a simplificação adotada não compromete os resultados nem a generalidade da técnica.

4.4 Suspensão e Retomada do Processo

Essas duas funções trabalham em conjunto para suspender e retomar todas as threads de um processo durante a leitura de memória.

Listagem 4.4 – Função SuspendProcess

```
1  void SuspendProcess(DWORD pid) {
2      HANDLE hSnap = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
3      if (hSnap != INVALID_HANDLE_VALUE) {
4          THREADENTRY32 te32 = { sizeof(te32) };
5          if (Thread32First(hSnap, &te32)) {
6              do {
7                  if (te32.th32OwnerProcessID == pid) {
8                      HANDLE hThread = OpenThread(
                           THREAD_SUSPEND_RESUME, FALSE, te32.
                           th32ThreadID);
9                      if (hThread) {
10                         SuspendThread(hThread);
11                         CloseHandle(hThread);
12                     }
13                 }
14             } while (Thread32Next(hSnap, &te32));
15         }
16         CloseHandle(hSnap);
17     }
18 }
```

Listagem 4.5 – Função ResumeProcess

```
1  void ResumeProcess(DWORD pid) {
2      HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
3      if (hSnapshot == INVALID_HANDLE_VALUE) return;
4
5      THREADENTRY32 te = { 0 };
6      te.dwSize = sizeof(te);
7
8      if (Thread32First(hSnapshot, &te)) {
```

```
        do {
            if (te.th32OwnerProcessID == pid) {
                HANDLE hThread = OpenThread(THREAD_SUSPEND_RESUME, FALSE
                    , te.th32ThreadID);
                if (hThread) {
                    ResumeThread(hThread);
                    CloseHandle(hThread);
                }
            }
        } while (Thread32Next(hSnapshot, &te));
    }

    CloseHandle(hSnapshot);
}
```

- A função `SuspendProcess` itera por todas as threads do sistema e, quando identifica uma thread pertencente ao processo de interesse, usa `SuspendThread` para pausá-la;
- A função `ResumeProcess` faz o processo inverso, chamando `ResumeThread`;
- Essa suspensão garante consistência da memória durante a leitura da chave;

A suspensão temporária do processo garante a consistência do estado da memória durante a varredura e a leitura da chave criptográfica. Sem essa etapa, alterações concorrentes realizadas pelas threads do *ransomware* poderiam sobrescrever ou invalidar os dados no exato momento da leitura, resultando em inconsistências ou falhas na extração.

Do ponto de vista forense, essa abordagem simula uma intervenção controlada, permitindo capturar artefatos voláteis críticos antes que sejam eliminados, prática alinhada com técnicas de resposta a incidentes em tempo real.

4.5 Função de Entrada do Programa

Listagem 4.6 – Função de Entrada do Programa

```
int main() {
    std::wstring targetExe = L"Ransomware TCC.exe";
    std::cout << "Esperando processo: " << std::string(targetExe.begin()
        , targetExe.end()) << "...\n";


    auto inicio_total = std::chrono::high_resolution_clock::now();
    int tentativas = 0;

    while (true) {
        if (GetAsyncKeyState(VK_F8) & 0x8000) {
```

```
11              std::cout << "Encerrando...\n";
12              break;
13          }
14
15          DWORD pid = GetProcessIdByName(targetExe);
16          if (pid != 0) {
17              tentativas++;
18
19
20              auto inicio_varredura = std::chrono::high_resolution_clock::
                    now();
21
22              SuspendProcess(pid);
23
24              HANDLE hProcess = OpenProcess(PROCESS_VM_READ |
                    PROCESS_QUERY_INFORMATION, FALSE, pid);
25              if (hProcess) {
26                  DWORD base = GetModuleBaseAddress(pid, targetExe);
27                  if (base) {
28                      MODULEINFO modInfo = { 0 };
29                      GetModuleInformation(hProcess, (HMODULE)base, &
                            modInfo, sizeof(modInfo));
30                      DWORD size = (DWORD)modInfo.SizeOfImage;
31
32                      const char* pattern = "\xBA\x00\x00\x00\x00\x8B\xC8\
                            xE8\x00\x00\x00\x00\xBA";
33                      const char* mask = "x????xxx????x";
34
35                      auto inicio_pattern = std::chrono::
                            high_resolution_clock::now();
36                      DWORD instr_addr = FindPatternInProcess(hProcess,
                            base, size, pattern, mask);
37                      auto fim_pattern = std::chrono::
                            high_resolution_clock::now();
38
39                      if (instr_addr != 0xBEEE) {
40                          DWORD key_addr;
41                          if (ReadProcessMemory(hProcess, (LPCVOID)(
                                instr_addr + 1), &key_addr, sizeof(DWORD),
                                nullptr)) {
42                              char key[33] = { 0 };
43
44                              auto inicio_leitura = std::chrono::
                                    high_resolution_clock::now();
45
46
```

```
47                          if (ReadProcessMemory(hProcess, (LPCVOID)
                                key_addr, key, 32, nullptr) &&
48                          key[0] != 0 && key[1] != 0 && key[2] != 0) {
49
50                              auto fim_leitura = std::chrono::
                                    high_resolution_clock::now();
51                              auto fim_varredura = std::chrono::
                                    high_resolution_clock::now();
52                              auto fim_total = std::chrono::
                                    high_resolution_clock::now();
53
54                              std::chrono::duration<double, std::milli
                                    > tempo_pattern = fim_pattern -
                                    inicio_pattern;
55                              std::chrono::duration<double, std::milli
                                    > tempo_leitura = fim_leitura -
                                    inicio_leitura;
56                              std::chrono::duration<double, std::milli
                                    > tempo_varredura = fim_varredura -
                                    inicio_varredura;
57                              std::chrono::duration<double>
                                    tempo_total = fim_total -
                                    inicio_total;
58
59                              std::cout << "\n
                                    ================================\n";
60                              std::cout << "CHAVE ENCONTRADA: " << key
                                     << std::endl;
61
62                              std::ofstream ofs("C:\\Users\\Gama\\
                                    Desktop\\Dump\\chave_lida.bin", std::
                                    ios::binary);
63                              ofs.write(key, 32);
64                              ofs.close();
65
66                              std::cout << "Chave salva em: C:\\Users
                                    \\Gama\\Desktop\\Dump\\chave_lida.bin
                                    \n";
67                              std::cout << "METRICAS DE TEMPO:\n";
68                              std::cout << "Busca do pattern: " <<
                                    tempo_pattern.count() << " ms\n";
69                              std::cout << "Leitura da chave: " <<
                                    tempo_leitura.count() << " ms\n";
70                              std::cout << "
                                    ================================\n";
71
72                              CloseHandle(hProcess);
```

```
73                                  ResumeProcess(pid);
74                                  break;
75                              }
76                          }
77                      }
78                      else {
79                          auto fim_varredura = std::chrono::
                               high_resolution_clock::now();
80                          std::chrono::duration<double, std::milli>
                               tempo_varredura = fim_varredura -
                               inicio_varredura;
81                          std::cout << "Varredura " << tentativas << ":
                               Pattern nao encontrado ("
82                          << tempo_varredura.count() << " ms)\n";
83                      }
84                  }
85                  CloseHandle(hProcess);
86              }
87              ResumeProcess(pid);
88          }
89      }
90
91      std::cout << "Pressione qualquer tecla para sair...";
92      std::cin.get();
93      return 0;
94  }
```

4.6 Visão Geral do Fluxo de Execução

A função principal centraliza a lógica de coordenação da ferramenta, integrando detecção do processo alvo, sincronização temporal, varredura por padrões e extração do artefato criptográfico. O fluxo foi estruturado de modo a minimizar o intervalo entre a detecção do processo e a leitura da chave, aspecto crítico para o sucesso da abordagem.

4.6.1 Descrição Passo a Passo

1. Define o nome do executável alvo: `"Ransomware TCC.exe"`;
2. Exibe uma mensagem informando que está aguardando o processo;
3. Entra em um laço infinito que é interrompido apenas com a tecla `F8`;
4. Quando o processo é detectado, o programa o suspende e abre com permissão de leitura;

5. Obtém o endereço base e tamanho do módulo carregado;
6. As variáveis `pattern` e `mask` armazenam,respectivamente, o padrão de bytes e a máscara gerados pelo plugin. O conteúdo dessas variáveis representa a assinatura binária previamente identificada durante a análise, sendo utilizada pela ferramenta para realizar a varredura da memória do processo e localizar a região correspondente ao trecho de código ou estrutura de interesse;
7. Executa a função `FindPatternInProcess` para localizar o padrão na memória;
8. Caso o padrão seja encontrado, lê o ponteiro da chave e, em seguida, lê a chave real de 32 bytes;
9. Exibe a chave encontrada e salva-a em arquivo binário para análise posterior;
10. Retoma o processo e volta ao estado de espera;

4.6.2 Controle Temporal e Métricas

A instrumentação do código com medições de tempo permite avaliar empiricamente a eficiência da ferramenta, possibilitando a comparação direta entre o tempo de extração da chave e o tempo de execução do *ransomware*. Essas métricas subsidiam a análise apresentada no capítulo de resultados, demonstrando que a intervenção ocorre dentro da janela temporal viável.

4.6.3 Síntese do Desenvolvimento

Este capítulo apresentou o desenvolvimento e a implementação da ferramenta *AOB-Tool*, detalhando suas principais funções, e mecanismos de operação. A integração entre varredura por assinaturas AOB, controle de execução do processo e leitura direta da memória demonstrou-se eficaz para a extração de chaves criptográficas em tempo de execução. A ferramenta materializa, em nível prático, a abordagem proposta neste trabalho, servindo como base experimental para a validação dos resultados discutidos nos capítulos subsequentes.

Capítulo 5

Experimentos e Síntese dos Resultados

Nesta seção serão apresentados os experimentos realizados para validar a proposta. Na etapa inicial, executou-se a captura de memória (*dump*) do *ransomware* em execução. Os arquivos gerados foram posteriormente analisados com ferramentas de engenharia reversa (Hex-Rays, 2025) e com a própria ferramenta de varredura AOB (*AOBTool*).

5.1 Execução e Captura

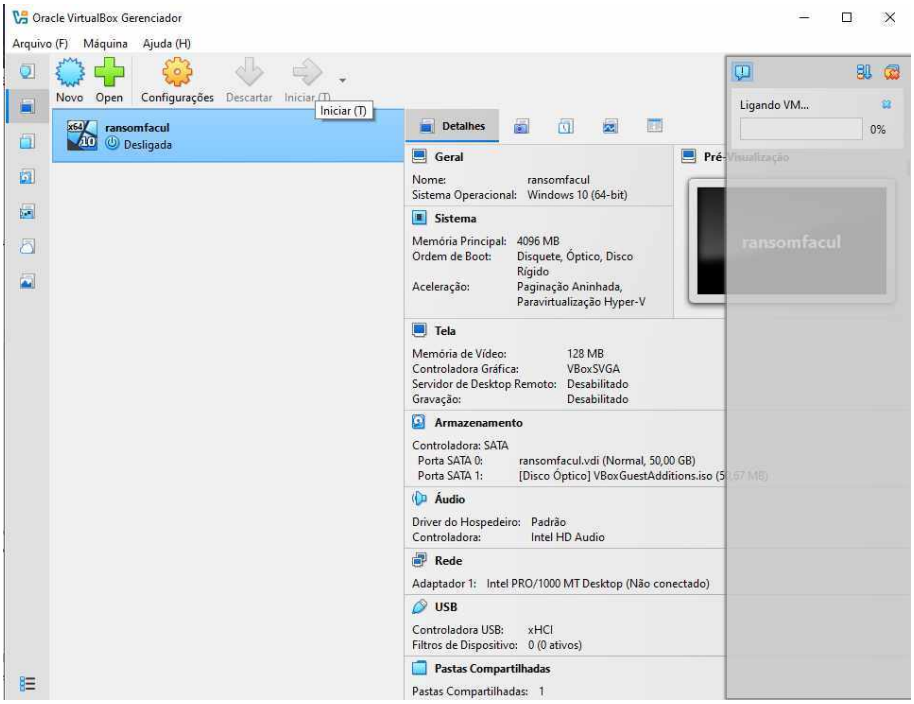
Para cada execução experimental procedeu-se da seguinte forma:

- Restaurou-se uma versão limpa da VM e iniciou-se o sistema alvo.
- Executou-se a amostra controlada do *ransomware* na VM.
- Quando o *ransomware* iniciou o procedimento de criptografia, procedeu-se à interrupção controlada de sua execução para permitir a captura do estado em memória. A seguir, descrevem-se os passos executados com o *Cheat Engine* para anexar e pausar o processo-alvo; os dumps foram realizados posteriormente com o *Scylla* (NtQuery, 2015).

5.1.1 Pausa do Processo e Preparação para Dump

- **Abrir o Cheat Engine e selecionar o processo:** Abrir o aplicativo *Cheat Engine* na VM alvo e clicar em Select a process to open (ícone de computador). Na janela exibida, localizar o processo correspondente ao *ransomware* pelo nome ou PID e clicar em `Open` para anexar-se ao processo.
- **Acessar opções avançadas (Advanced Options):** Com o processo anexado, abrir o menu de opções/advanced para pausar o processo de execução do *ransomware*.

Figura 3 – Iniciação da máquina virtual utilizada nos experimentos



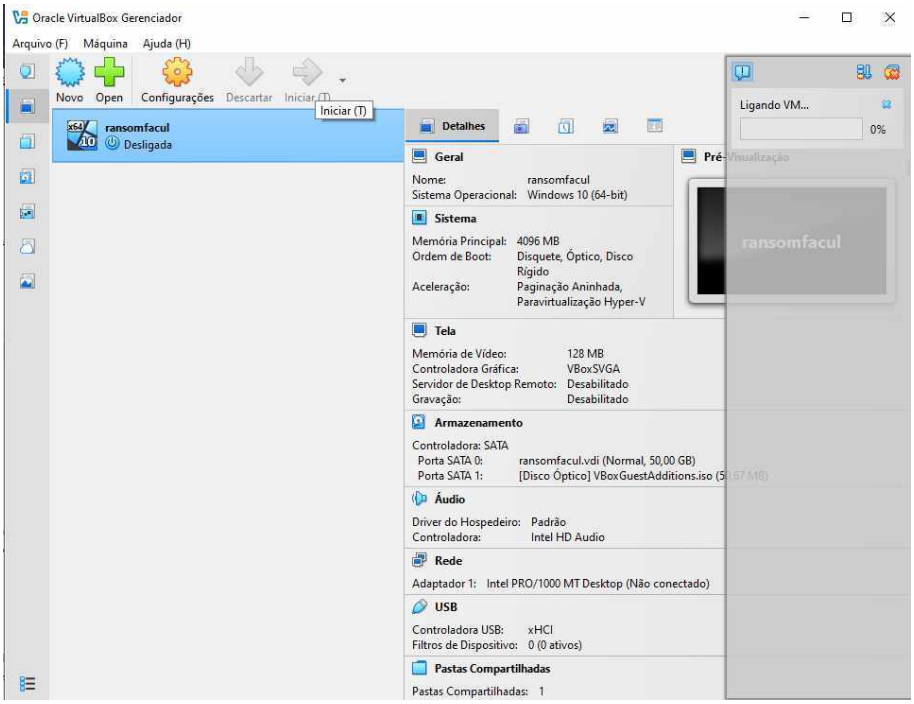
Fonte: elaborado pelo autor.

5.1.2 Despejo *(Dump)* com Scylla

Após pausar o processo-alvo com o *Cheat Engine* (Seção 5.1.1), procedeu-se ao despejo de memória do processo utilizando a ferramenta *Scylla* (NtQuery, 2015). Os passos realizados foram os seguintes:

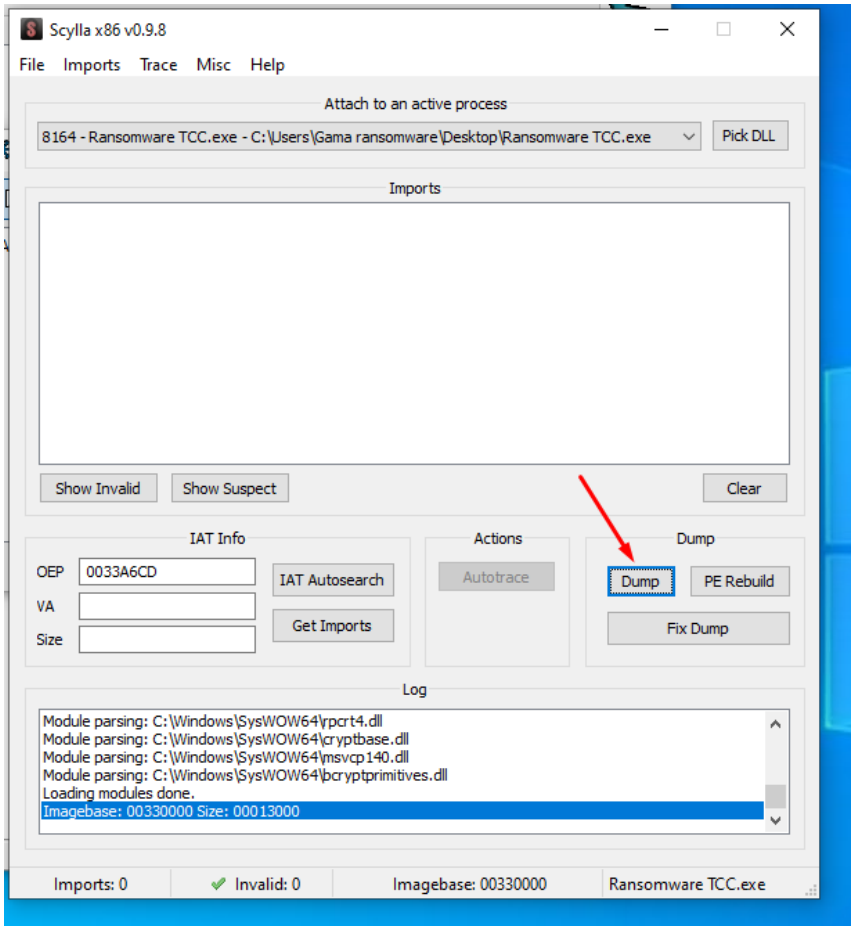
- **Abrir o Scylla e anexar ao processo ativo:** Iniciar o aplicativo *Scylla* na VM de análise. Selecionar a opção *Attach to an active process* e localizar o processo do *ransomware* na lista por nome ou *PID*; clicar e selecioná-lo para anexar.
- **Executar a operação de Dump:** Com o processo anexado, navegar até a aba/-controle *Dump* e clicar em *Dump* para iniciar a extração do espaço de memória/-processo para um arquivo local.

Figura 4 – Iniciação da máquina virtual utilizada nos experimentos



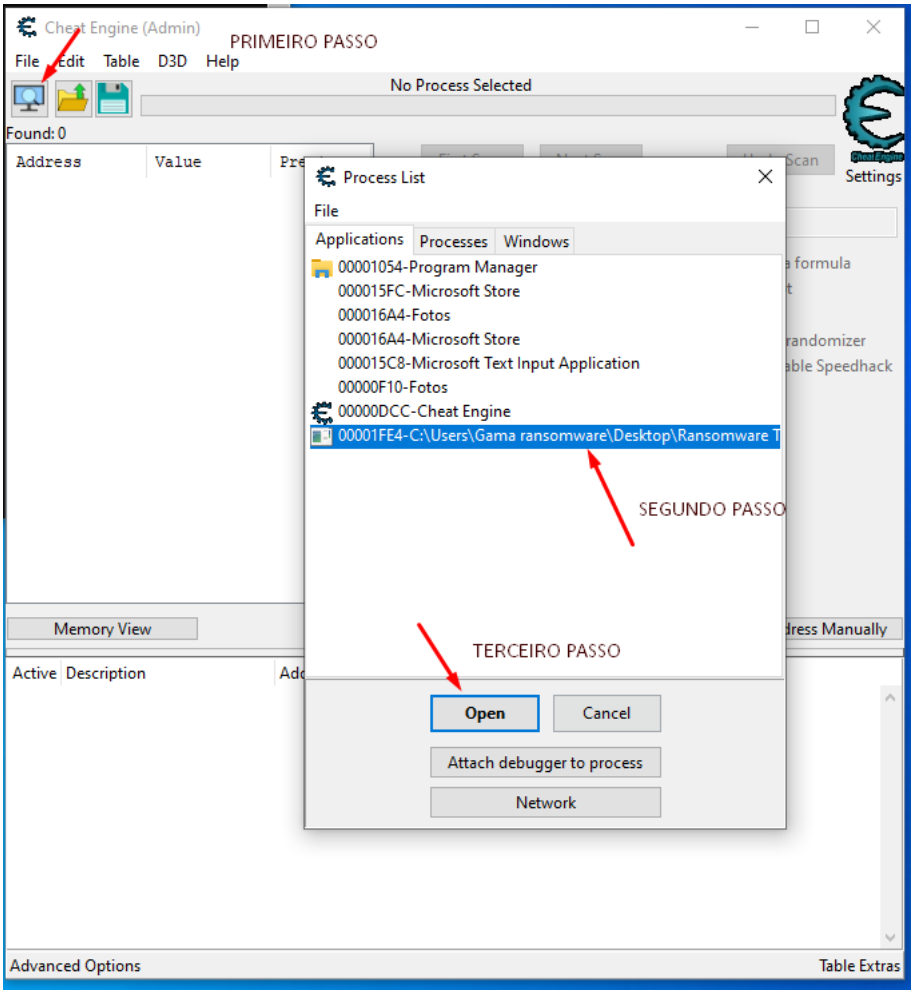
Fonte: elaborado pelo autor.

Figura 8 – Scylla: controle *"Dump"* utilizado para iniciar a extração do processo em memória



Fonte: elaborado pelo autor.

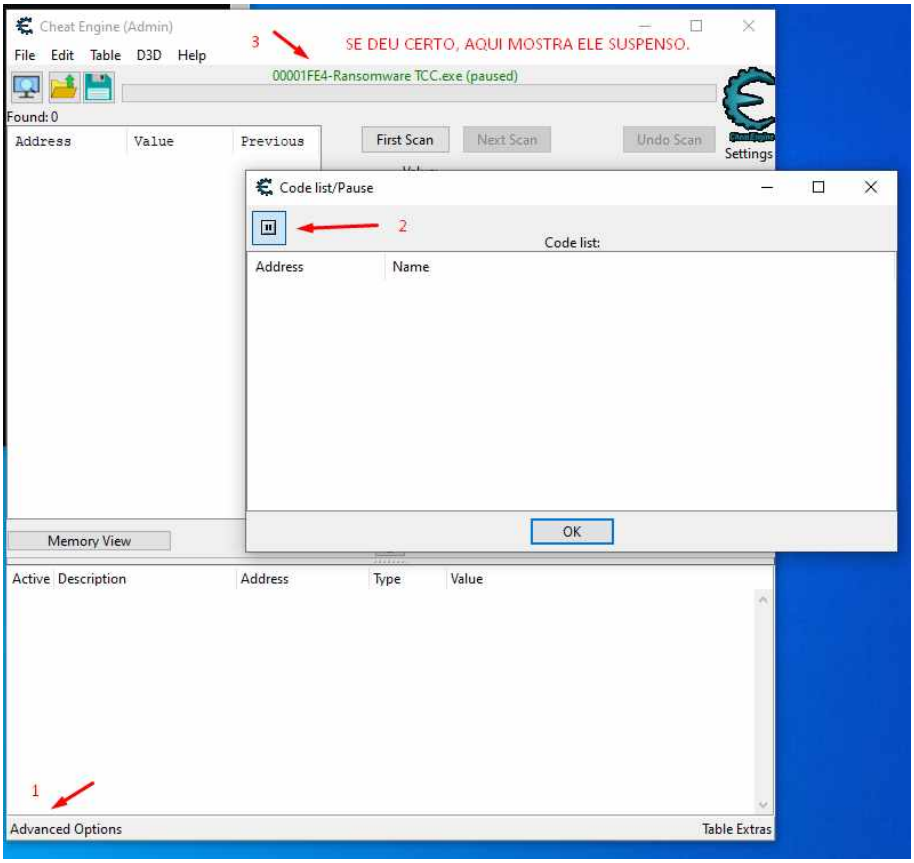
Figura 5 – Diálogo *"Select a process"* do Cheat Engine para seleção do processo alvo



Fonte: elaborado pelo autor.

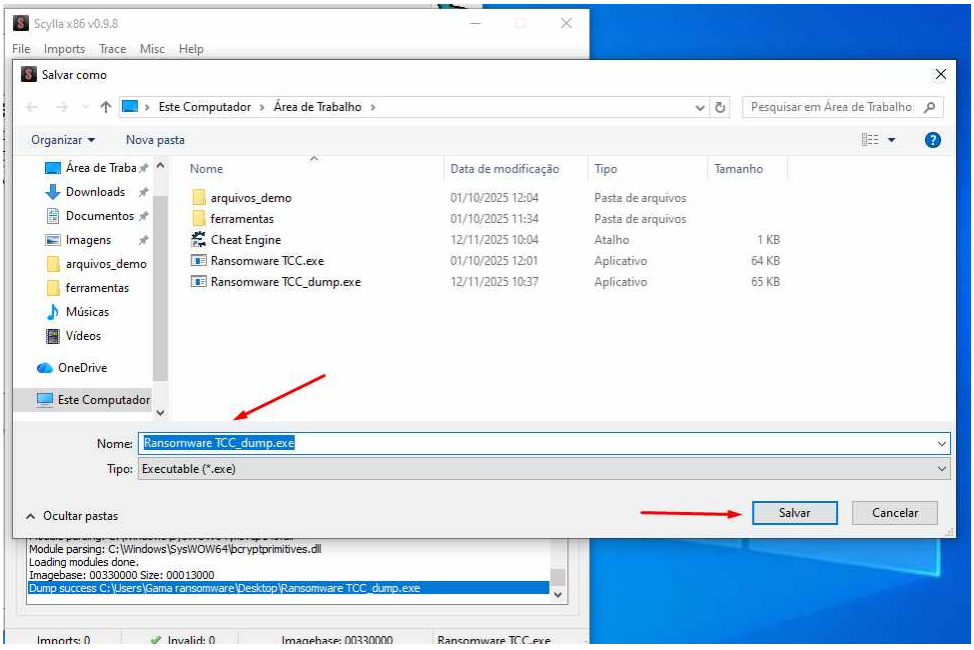
- **Escolher local de salvamento do dump:** Após acionar `Dump`, selecionar a pasta destino e nome do arquivo para salvar o arquivo de dump e confirmar a operação.

Figura 6 – Menu *"Advanced Options"* do Cheat Engine exibindo controles avançados



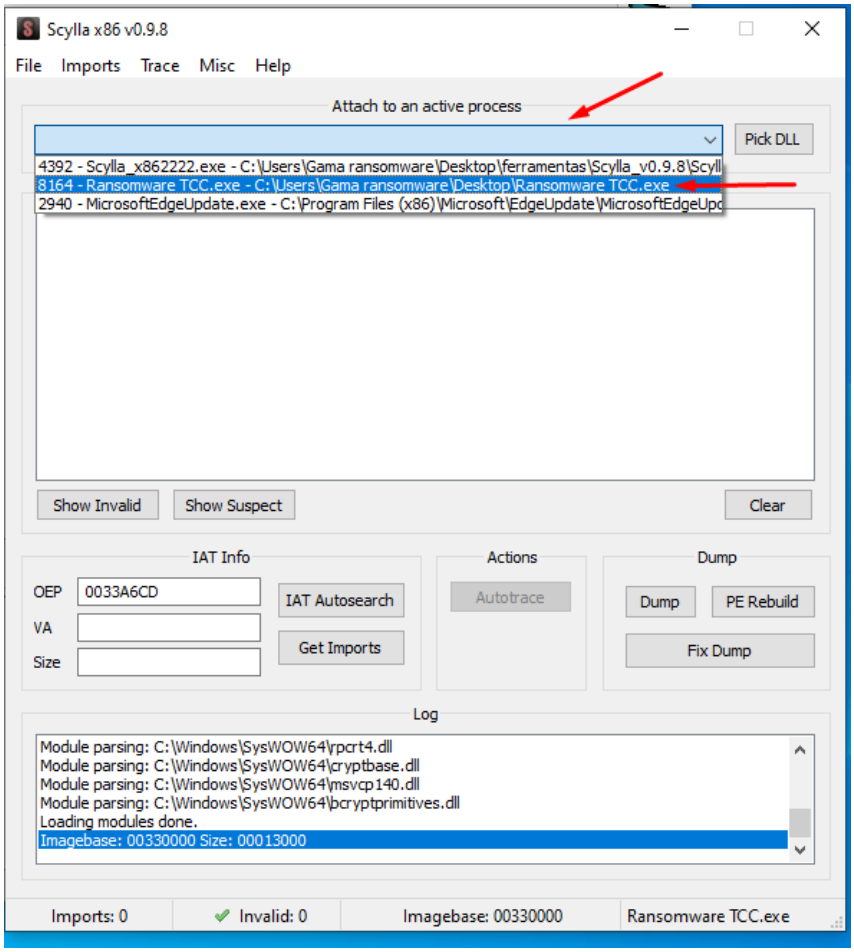
Fonte: elaborado pelo autor.

Figura 9 – Seleção do caminho e nome do arquivo para salvamento do dump gerado pelo Scylla



Fonte: elaborado pelo autor.

Figura 7 – Scylla: diálogo *"Attach to an active process"* com seleção do processo alvo



Fonte: elaborado pelo autor.

5.2 Análise do *Dump* no *IDA Free*

Após a geração e transferência do arquivo de dump pelo Scylla (Seção 5.1.2), realizou-se a análise estática inicial com o *IDA Free* (versão 9.2) (Hex-Rays, 2025) para identificar rotinas de criptografia e gerar assinaturas AOB. Esta etapa constitui frequentemente a porção mais trabalhosa e exigente de uma investigação de *malware*. Esta etapa é a que o analista dedica tempo, atenção e paciência para dissecar o código, compreender os fluxos de execução e inferir intenções a partir de instruções de baixo nível.

A análise no IDA começa pela observação geral e revisão das seções carregadas, imports, strings e funções automaticamente identificadas e evolui para um exame aprofundado das rotinas suspeitas. A leitura de assembly requer cautela e prática; instruções isoladas raramente dizem tudo, sendo necessário analisar o contexto (cross-references), os usos de registradores, as chamadas de função e os acessos a memória. Ferramentas e visualizações do IDA, como o gráfico de fluxo de funções, a janela de *Strings* e a listagem de *Imports*, ajudam a localizar pontos de interesse, como inicializações de bibliotecas criptográficas, chamadas a APIs de sistema e rotinas de geração de chaves.

O trabalho prático envolve várias atividades iterativas: renomear funções e variáveis locais à medida que se compreende seu papel, inserir comentários e anotações para facilitar retornos futuros, reconstruir protótipos de função (tipagem de parâmetros e retorno) para obter desassemblagens mais legíveis; e seguir cadeias de referências de dados para encontrar buffers e estruturas que contenham chaves. Plugins e utilitários adicionais (por exemplo, detectores de rotinas criptográficas ou geradores de assinaturas) podem acelerar a identificação de padrões, mas a confirmação humana é essencial, isto é, testar hipóteses, executar trechos em ambiente controlado (quando possível) e validar interpretações.

Outras tarefas importantes na fase de análise incluem: tratar obfuscação (desempacotar trechos quando aplicável), identificar rotinas de inicialização que configuram algoritmos, mapear pontos onde a chave é alocada ou copiada e gerar assinaturas AOB a partir de sequências de instruções/dados estáveis. A geração de uma AOB confiável demanda escolher padrões que sejam suficientemente específicos para evitar falsos positivos, mas suficientemente genéricos para resistir a pequenas mudanças de compilação ou empacotamento. Isto costuma requerer experimentação e ajuste fino.

Além das análises manuais, é fundamental documentar tudo: capturas de tela das vistas relevantes (disassembly, graph, strings), logs de renomeações e comentários, versões dos binários/commits, e hashes dos dumps analisados. Essas evidências tornam o trabalho reprodutível e verificável. Finalmente, os artefatos produzidos na IDA (como AOBs, offsets e anotações) servem como insumo para a etapa seguinte, que é a validação automática com a *AOBTool* que testa se os padrões identificados realmente permitem localizar e extrair as chaves criptográficas de um *ransomware* em execução

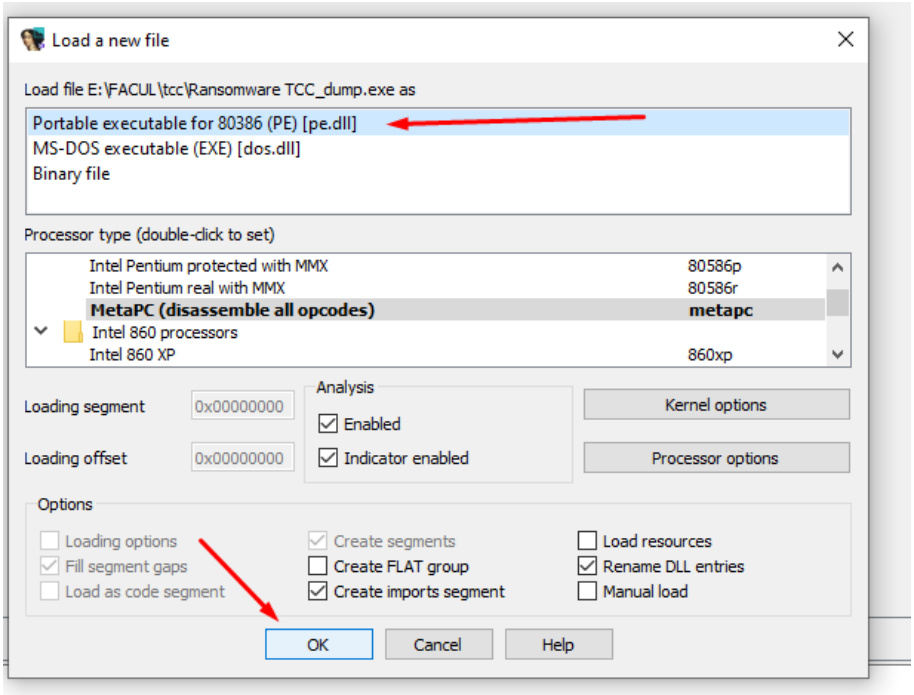
- ❑ **Iniciar o IDA Free e iniciar nova desmontagem:** Clicar em *New / Open* para selecionar um novo arquivo a ser analisado. Navegar até o diretório onde está o dump gerado e selecionar o arquivo do *ransomware*.

Figura 10 – Diálogo de seleção de arquivo no *IDA Free*: escolha do *dump* a ser analisado

Fonte: elaborado pelo autor.

- ❑ **Configurar o tipo de arquivo carregado:** Na janela “Load a new file” (Load new file), confirmar o tipo detectado pelo IDA. Selecionar (quando aplicável) a opção *Portable executable for 80386 (PE) [pe.dll]* ou o tipo correspondente ao binário/*dump* em análise, e clicar em `OK` para prosseguir com a análise automática.

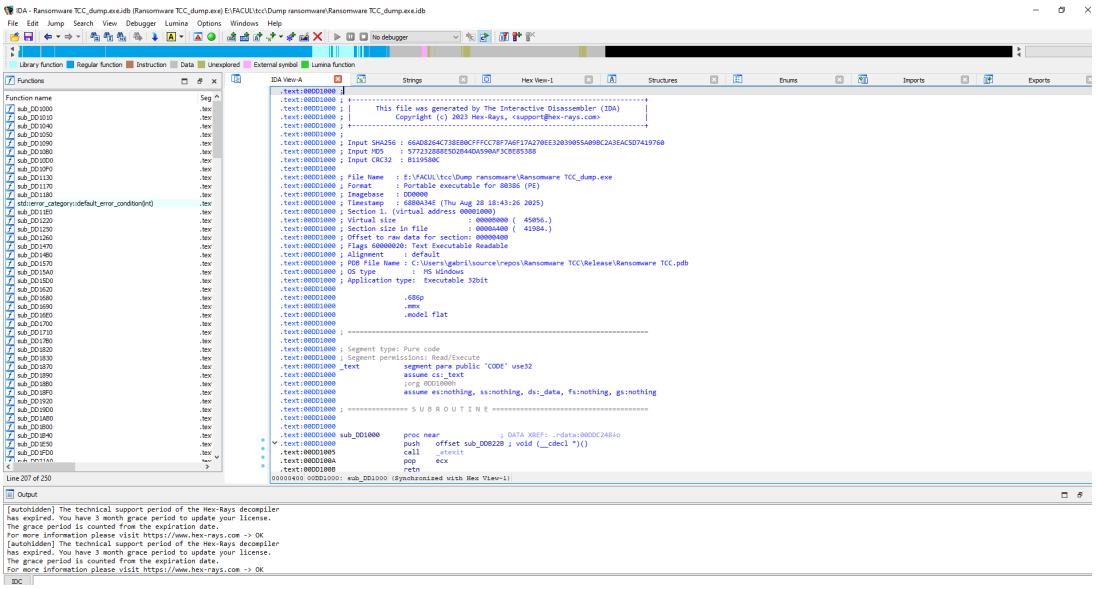
Figura 11 – Janela *"Load a new file"*: seleção do formato PE antes da análise



Fonte: elaborado pelo autor.

- **Aguardar análise automática e visualizar as janelas principais:** Após confirmar, aguardar o IDA realizar o parsing inicial e análise automática de funções; em seguida visualizar as janelas de *Disassembly*, *Functions*, *Strings* e *Imports* para localizar rotinas relacionadas à criptografia.

Figura 12 – Visão principal do *IDA Free* após o carregamento do arquivo



Fonte: elaborado pelo autor.

5.2.1 Inspeção de *Strings* e Identificação de Trechos Relevantes

Durante a fase inicial de exploração do binário no IDA Free, listaram-se as strings presentes no arquivo (atalho `Shift+F12`) para obter rapidamente indicadores textuais que pudessem apontar para rotinas criptográficas, nomes simbólicos ou mensagens internas. Paralelamente, realizou-se uma varredura dos *imports* e da lista de funções geradas automaticamente pelo IDA, buscando chamadas a APIs relacionadas à criptografia (por exemplo, chamadas a OpenSSL ou CryptoAPI) e ou outras funções geradas para criptografia e padrões de fluxo típicos de inicialização de chaves.

Figura 13 – Lista de *strings* obtida por meio do atalho `Shift+F12` no *IDA Free*

Fonte: elaborado pelo autor.

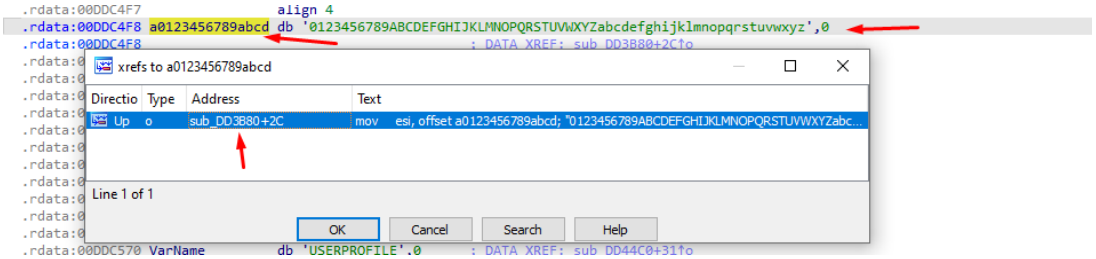
5.2.2 Identificação da String Suspeita e Navegação por Xrefs

Ao analisar a lista de *strings* (Figura 14), chamou a atenção a entrada localizada em `.rdata:00DDC4F8`:

```
.rdata:00DDC4F8 a0123456789abcd db '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
      abcdefghijklmnopqrstuvwxyz',0
```

Trata-se de uma sequência alfanumérica longa (valores ASCII consecutivos) que poderia estar associada a um identificador ou, potencialmente, a material criptográfico. Para investigar o contexto de uso dessa string, selecionou-se a linha correspondente e pressionou-se a tecla `x` (Xrefs). Na janela de cross-references foi clicado em `OK` para saltar para a função que referencia essa string, o que permitiu localizar a rotina provável responsável por sua manipulação.

Figura 14 – Trecho da seção `.rdata` exibindo a sequência alfanumérica identificada na lista de *strings*



Fonte: elaborado pelo autor.

5.2.3 Análise da Função Suspeita e Decompilação

Ao saltar para a função que contém o Xref, utilizou-se a decompilação (atalho `F5`) para obter uma visão de nível mais alto do fluxo lógico. A decompilação evidenciou pontos onde buffers eram alocados e onde ocorriam cópias/atribuições de valores constantes e locais compatíveis com operações de preparação ou armazenamento de chaves. Na visualização do decompiler notou-se, nas proximidades das instruções de leitura/escrita do buffer, uma sequência de bytes/valores que exibiam alta entropia (letras e números aparentemente aleatórios), reforçando a hipótese de que aquele bloco poderia conter material cripto (chave/nonce).

Figura 15 – Visão de decompilação (F5) da função que referencia a string identificada

```
char sub_DD3B80()
{
  unsigned int v0; // eax
  unsigned int v1; // ecx
  unsigned int v2; // esi
  __int64 v3; // rax
  char result; // al
  int v5; // [esp+8h] [ebp-13D0h] BYREF
  int v6[1250]; // [esp+Ch] [ebp-13CCh]
  char v7[64]; // [esp+1394h] [ebp-44h] BYREF

  strcpy(v7, "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz");
  v0 = std::_Random_device();
  v6[1248] = -1;
  v1 = 1;
  v6[0] = v0;
  do
  {
    v0 = v1 + 1812433253 * (v0 ^ (v0 >> 30));
    v6[v1++] = v0;
  }
  while ( v1 < 0x270 );
  v2 = 0;
  v5 = 624;
  do
  {
    v3 = 62i64 * (unsigned int)sub_DD8D90(&v5);
    if ( (unsigned int)v3 <= 3 )
    {
      do
        v3 = 62i64 * (unsigned int)sub_DD8D90(&v5);
      while ( (unsigned int)v3 < 4 );
    }
    result = v7[HIDWORD(v3)];
    aPuytqb8b560s6y[v2++] = result;
  }
  while ( v2 < 0x20 );
  aPuytqb8b560s6y[32] = 0;
  return result;
}
```

Fonte: elaborado pelo autor.

5.2.4 Análise Detalhada da Função `sub_DD3B80`

A função `sub_DD3B80` (Figura 15) destacou-se por conter operações típicas de geração pseudoaleatória de caracteres, com estrutura compatível à criação de chaves temporárias ou vetores de inicialização. Seu corpo decompilado é apresentado abaixo para contextualização da análise.

Listagem 5.1 – Função extraída do IDA

```
1   char sub_DD3B80()
2   {
3       unsigned int v0; // eax
```

```
4         unsigned int v1; // ecx
5         unsigned int v2; // esi
6         __int64 v3; // rax
7         char result; // al
8         int v5; // [esp+8h] [ebp-13D0h] BYREF
9         int v6[1250]; // [esp+Ch] [ebp-13CCh]
10        char v7[64]; // [esp+1394h] [ebp-44h] BYREF
11
12        strcpy(v7, "0123456789
              ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz");
13        v0 = std::_Random_device();
14        v6[1248] = -1;
15        v1 = 1;
16        v6[0] = v0;
17        do {
18            v0 = v1 + 1812433253 * (v0 ^ (v0 >> 30));
19            v6[v1++] = v0;
20        } while (v1 < 0x270);
21        v2 = 0;
22        v5 = 624;
23        do {
24            v3 = 62i64 * (unsigned int)sub_DD8D90(&v5);
25            if ((unsigned int)v3 <= 3) {
26                do
27                v3 = 62i64 * (unsigned int)sub_DD8D90(&v5);
28                while ((unsigned int)v3 < 4);
29            }
30            result = v7[HIDWORD(v3)];
31            aPuytqb8b560s6y[v2++] = result;
32        } while (v2 < 0x20);
33        aPuytqb8b560s6y[32] = 0;
34        return result;
35    }
```

A análise detalhada permitiu compreender que a rotina define um vetor de caracteres `v7` com todos os símbolos alfanuméricos (maiúsculos e minúsculos), que é utilizado como base para a composição da chave. A função invoca um gerador pseudoaleatório (`std::_Random_device`) e utiliza uma sequência de cálculos multiplicativos (`1812433253 * (v0 ^ (v0 >> 30))`) com várias interações.

Em cada iteração, é selecionado um índice dentro do vetor `v7`, concatenando 32 caracteres (0x20 em hexadecimal) para formar a string final armazenada em `aPuytqb8b560s6y`. Essa string, portanto, representa possivelmente a chave gerada pelo *ransomware* em tempo de execução.

Em suma, a combinação das técnicas de listagem de *strings* (`Shift+F12`), análise de imports/funções e decompilação (F5) possibilitou localizar um trecho suspeito e identificar

sequências hexadecimais compatíveis com uma chave.

5.3 Geração de Assinaturas AOB a partir da Função Identificada

Com a função `sub_DD3B80` identificada e decompilada, procedeu-se à busca direta nas views do IDA para localizar ocorrências reais da chave gerada em tempo de execução. A seguir descrevem-se os passos executados.

- ❑ **Buscar prefixo da chave no Text view / Strings:** No *Text view / Strings view* do *IDA Free* procurou-se pelo prefixo identificado no buffer (por exemplo: `aPUYTqb8b560S6`) para localizar referências. Essa busca permitiu observar que a sequência aparecia em múltiplos pontos do binário, indicando uso recorrente.

Figura 16 – Busca pelo prefixo da chave (`aPUYTqb8b560S6`) na visão de texto do *IDA Free*

Fonte: elaborado pelo autor.

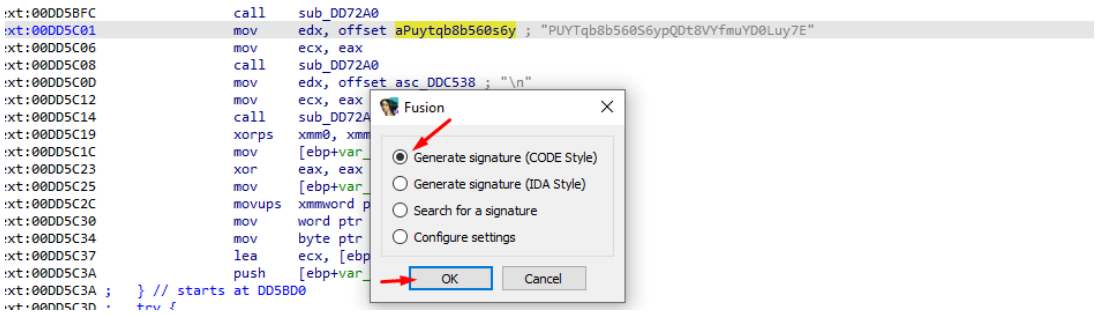
- ❑ **Inspecionar a instrução:** A partir da string localizada, seguiu-se para os *xrefs* e encontrou-se o trecho de código onde a string é carregada:

Listagem 5.2 – Trecho de assembly onde a string é referenciada

```
1 .text:00DD5BFC     call     sub_DD72A0
2 .text:00DD5C01     mov      edx, offset aPuytqb8b560s6y ; "
    PUYTqb8b560S6ypQDt8VYfmuYD0Luy7E"
3 .text:00DD5C06     mov      ecx, eax
4 .text:00DD5C08     call     sub_DD72A0
5 .text:00DD5C0D     mov      edx, offset asc_DDC538 ; "\n"
6 .text:00DD5C12     mov      ecx, eax
```

- ❑ **Invocar o plugin IDA-Fusion para gerar assinatura:** Após selecionar a instrução de interesse (por exemplo na linha com `mov edx, offset aPuytqb8b560s6y`), acionou-se o plugin IDAFusion (senator715, 2024) (atalho `Ctrl+Alt+S`). Na janela do plugin selecionou-se a opção *Generate signature* com o estilo *CODE Style* para gerar uma assinatura AOB baseada no código em volta da instrução.

Figura 17 – Janela do plugin *IDA-Fusion*: opção *Generate signature* (estilo *CODE*)



Fonte: elaborado pelo autor.

❑ **Assinatura AOB gerada (padrão + máscara):** O plugin retornou uma assinatura no formato padrão em bytes com a máscara correspondente:

Listagem 5.3 – Assinatura AOB gerada pelo IDA-Fusion (exemplo)

```
1 \xBA\x00\x00\x00\x00\x8B\xC8\xE8\x00\x00\x00\x00\xBA
2 x????xxx????x
```

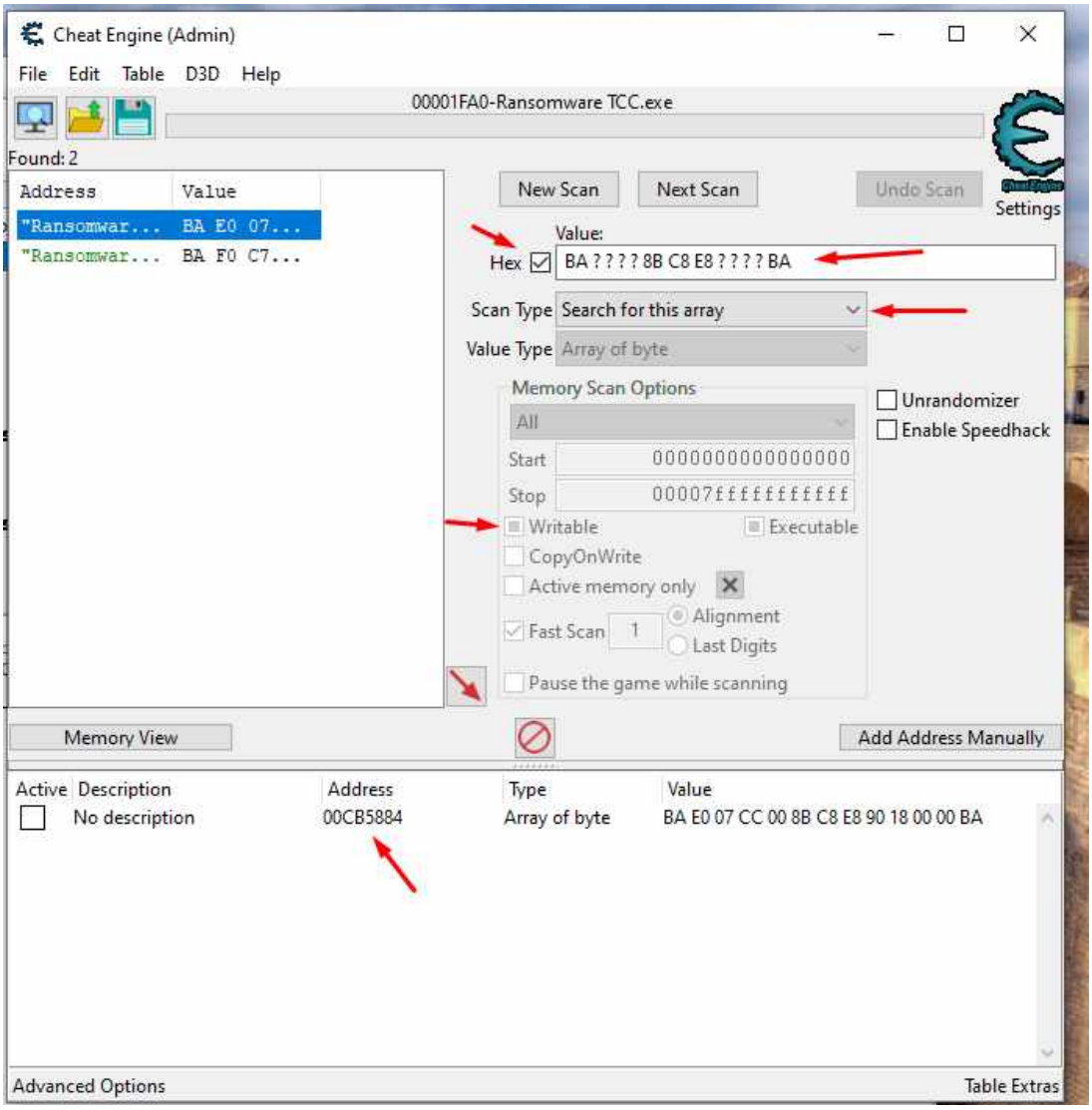
Após gerar a assinatura, procedeu-se à validação em dumps anteriores (e em novas execuções) para verificar se o AOB encontra a mesma sequência em memória e se o offset relativo leva ao buffer contendo a chave. Caso a assinatura apresente falsos positivos, ela é refinada (ajustando bytes constantes e máscaras) até atingir equilíbrio entre sensibilidade e especificidade, como foi feito outras análises ficou comprovado que este padrão seria estável e o que vai ser usado.

Além da validação feita em dumps estáticos, realizou-se também uma verificação dinâmica utilizando o *Cheat Engine* (Cheat Engine Team, 2024) com o *ransomware* em execução. O objetivo foi confirmar, em tempo real, que a AOB gerada correspondia de fato à região de memória onde a chave criptográfica era armazenada durante a rotina de cifragem. Para isso, a assinatura obtida via IDAFusion (senator715, 2024) foi pesquisada diretamente na memória do processo, permitindo observar se o endereço retornado pelo scan coincidia com o buffer identificado previamente no disassembly.

Uma vez localizado o padrão, abriu-se o *Memory View* sobre o endereço encontrado, confirmando visualmente que os 32 bytes exibidos correspondiam exatamente à chave pseudoaleatória gerada pelo *ransomware*. Esse procedimento reforçou a consistência da assinatura desenhada e comprovou empiricamente que o método proposto identifica a chave em tempo real, ainda durante a execução do *ransomware*.

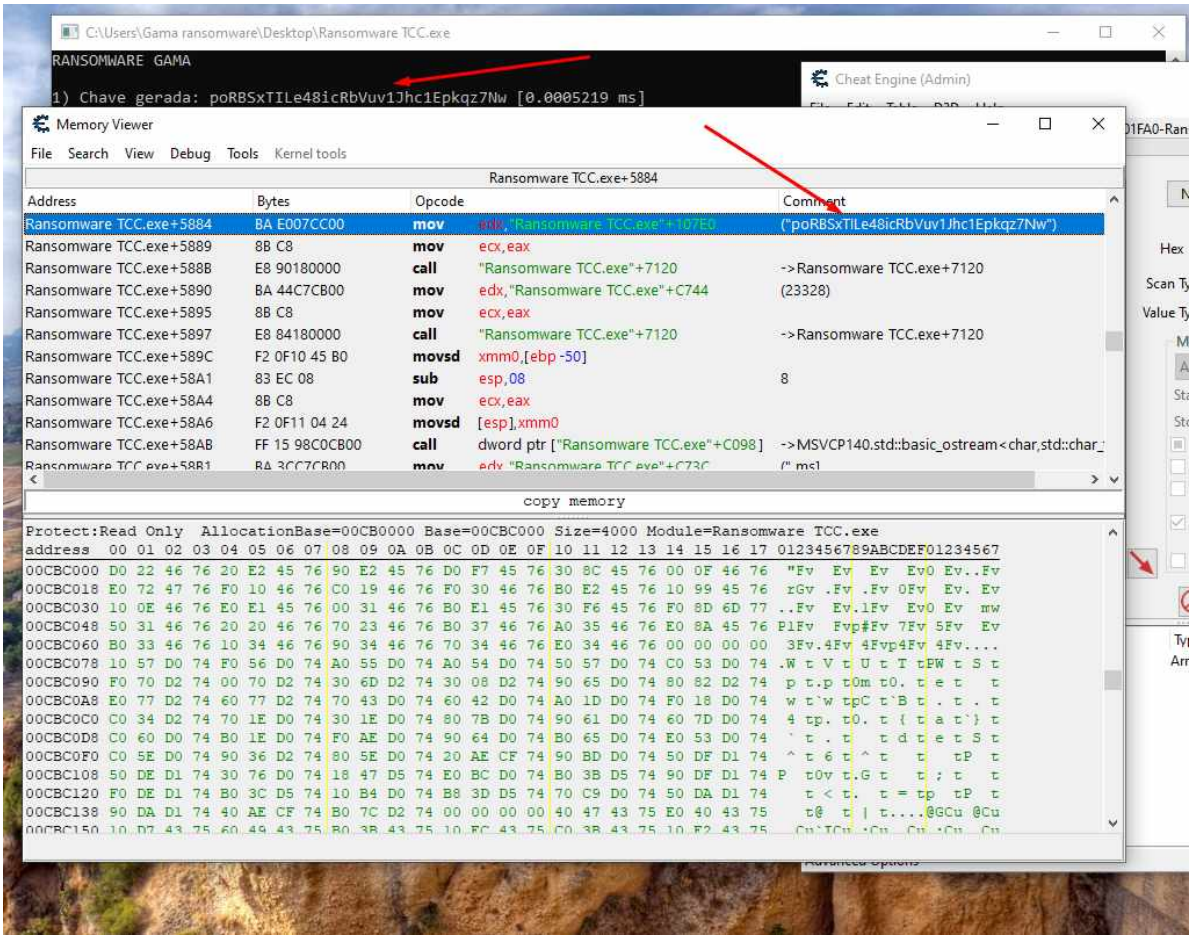
As Figuras **??** e **??** apresentam: (i) a busca da AOB no *Cheat Engine* e (ii) a visualização direta da chave na região de memória correspondente.

Figura 18 – Busca da assinatura AOB utilizando o *Cheat Engine* para fins de validação



Fonte: elaborado pelo autor.

Figura 19 – Visualização da memória em tempo de execução por meio do *Cheat Engine*



Fonte: elaborado pelo autor.

O presente trabalho propõe um método prático para identificação e extração de chaves criptográficas de *ransomwares* em tempo de execução baseado em assinaturas AOB geradas por análise estática. A partir da assinatura gerada (padrão + máscara) para trechos de código identificados no binário por exemplo, a sequência de instruções que referencia o buffer contendo a chave, a ferramenta desenvolvida efetua fazendo a varredura diretamente no espaço do *ransomware* em execução, localiza o padrão e deriva o offset relativo até o buffer candidato. Esse fluxo combina engenharia reversa (para criar assinaturas AOB) com análise forense de memória e automação (AOBTool) para viabilizar detecção rápida e reprodutível de chaves efêmeras geradas por *malware*.

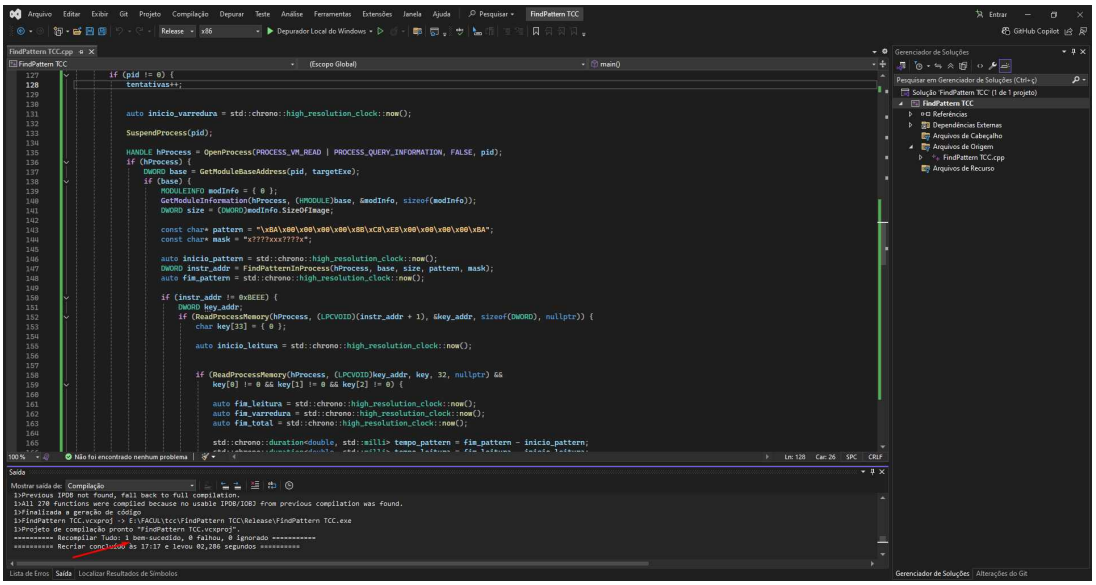
5.4 Experimento para Análise de Tempo

A ferramenta desenvolvida (*AOBTool*) foi implementada e compilada no ambiente Microsoft Visual Studio (Microsoft, 2022). Após a geração do executável, a ferramenta foi executada em segundo plano na máquina virtual (ambiente experimental controlado descrito na subseção (5.1).

Em seguida, o *ransomware* foi iniciado dentro da máquina virtual alvo e, durante sua execução, o processo foi inspecionado para comprovar que a chave gerada em tempo de execução poderia ser localizada e extraída diretamente da memória.

Conforme mostra a figura 20, é o projeto sendo compilado e gerando o executável proposto.

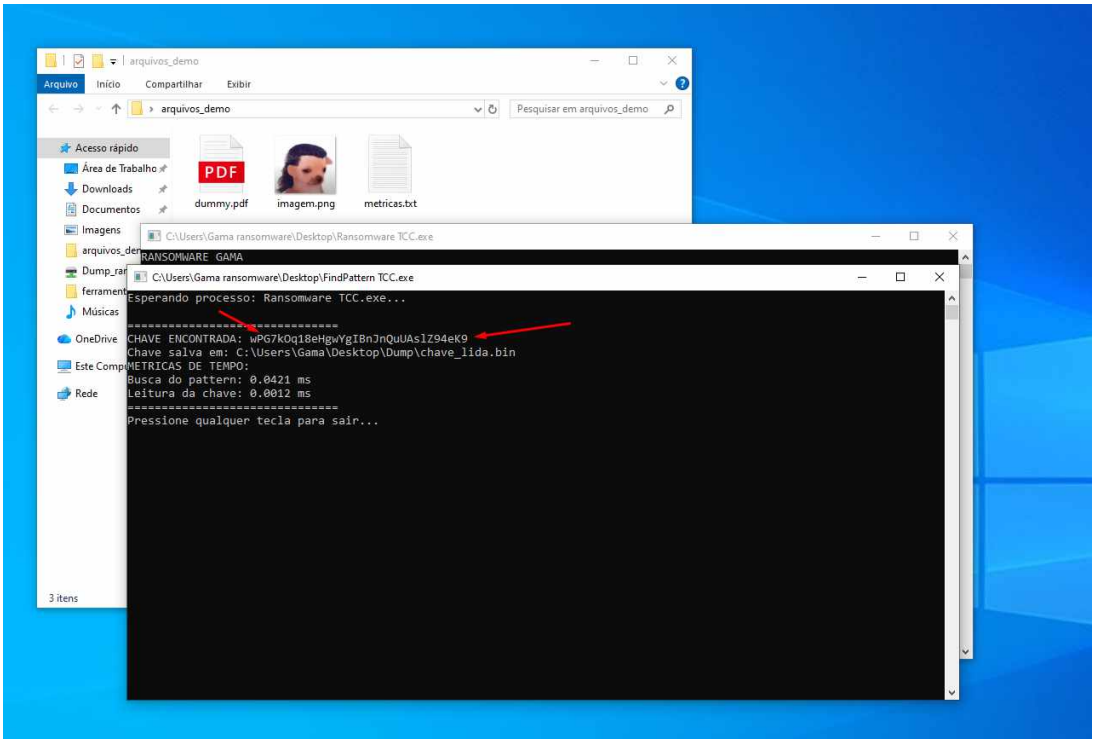
Figura 20 – Compilação do projeto e geração do executável no *Visual Studio*



Fonte: elaborado pelo autor.

Nesta figura 21 mostra a ferramenta *AOBTool* em execução juntamente com a amostra sintética, onde mostra a ferramenta extraindo em tempo de execução a chave criptográfica antes do *ransomware* crifar os arquivos.

Figura 21 – Execução da ferramenta *AOBTool* após a compilação



Fonte: elaborado pelo autor.

5.4.1 Tempo de Execução X Tempo de Extração

Foram realizados 10 experimentos independentes (execuções completas do *ransomware* de teste e validação com a *AOBTool*). Para cada execução foram registradas as métricas relevantes: tempo de geração da chave pelo *ransomware*, tempo total de criptografia (medida do tempo que o *ransomware* demorou para completar a rotina de cifração dos arquivos), tempo de busca do pattern pela *AOBTool* e tempo de leitura da chave. A Tabela 2 apresenta os valores (unidade: ms).

Tabela 2 – Resultados das 10 execuções (tempo em milissegundos).

Execução	Chave gerada	Tempo criptografia	Busca do pattern	Leitura da chave
1	0.0006170	0.0821518	0.0447	0.0012
2	0.0008672	0.0712812	0.0444	0.0012
3	0.0006281	0.0613382	0.0444	0.0012
4	0.0005972	0.0681346	0.0448	0.0012
5	0.0006065	0.0707624	0.0449	0.0012
6	0.0004499	0.0637167	0.0589	0.0013
7	0.0004331	0.0514660	0.0434	0.0013
8	0.0005269	0.0562920	0.0430	0.0012
9	0.0016015	0.0913421	0.0462	0.0012
10	0.0004250	0.0833063	0.0429	0.0012

A avaliação experimental concentrou-se na comparação direta entre o tempo de extração da chave e o tempo necessário para a cifragem dos arquivos, por se tratar do critério mais relevante para verificar a viabilidade da abordagem proposta. Não foram exploradas variações adicionais, como diferentes volumes de dados, cargas do sistema ou múltiplos ciclos de cifragem, uma vez que tais análises extrapolam o objetivo principal deste estudo, que consiste em verificar a existência de uma janela temporal explorável para a extração da chave criptográfica.

5.5 Síntese dos Resultados

A Tabela 2 apresenta os tempos observados em cada execução: (i) o intervalo até a geração da chave pelo *ransomware* (*Chave gerada*), (ii) o tempo observado até a conclusão da rotina de criptografia dos arquivos (*Tempo criptografia*), (iii) o tempo de busca do padrão AOB pela ferramenta AOBTool e (iv) o tempo de leitura do buffer contendo a chave (*Leitura da chave*).

A análise dos resultados evidencia diferenças claras entre o tempo necessário para a execução das rotinas críticas do *ransomware* e o tempo gasto pela *AOBTool* para localizar e extrair a chave criptográfica diretamente da memória volátil. Observa-se que o tempo de geração da chave é extremamente reduzido em todas as execuções, ocorrendo em frações de milissegundo, o que é esperado, uma vez que a geração consiste majoritariamente em operações aritméticas e pseudoaleatórias em memória.

Em contrapartida, o tempo total de criptografia apresenta maior variação entre as execuções. Essa variação pode ser explicada por fatores como operações de entrada e saída em disco, chamadas ao sistema operacional para acesso a arquivos, alocação e liberação de buffers e eventuais variações de escalonamento de CPU durante a execução. Tais fatores são inerentes a processos de cifragem de arquivos e contribuem para que essa etapa seja significativamente mais custosa em termos temporais do que a simples geração da chave.

No que se refere à *AOBTool*, os tempos de busca do padrão AOB mantiveram-se relativamente estáveis entre as execuções, situando-se majoritariamente na ordem de dezenas de microssegundos. Esse comportamento é consistente com a abordagem adotada, uma vez que a varredura ocorre sobre uma região delimitada da memória do módulo alvo, utilizando uma assinatura previamente validada, o que reduz significativamente o espaço de busca e o custo computacional. A leitura do buffer contendo a chave, por sua vez, apresentou tempos praticamente constantes e desprezíveis quando comparados às demais métricas, pois consiste em uma única operação de leitura direta de memória.

De forma geral, observa-se que, em todas as execuções, o tempo total gasto pela *AOBTool* (busca do padrão somada à leitura da chave) foi inferior ao tempo necessário ao *ransomware* para concluir sua rotina de criptografia e proceder à remoção da chave da memória. Isso indica que a ferramenta foi capaz de atuar dentro da janela temporal

crítica em que a chave ainda se encontra acessível na memória volátil do processo.

Em termos práticos, esses resultados indicam que:

- **Sucesso experimental:** a *AOBTool* foi capaz de localizar e extrair a chave em todas as execuções reportadas.
- **Viabilidade:** em um número significativo de execuções o processo de extração ocorre em tempo menor do que aquele necessário ao *ransomware* para finalizar a criptografia/remover a chave, isso demonstra que a abordagem é tecnicamente viável para mitigar o dano (recuperação da chave antes da perda irreversível dos dados).

Capítulo 6

Conclusão

Os resultados obtidos demonstram que a análise da memória volátil, aliada ao uso de padrões binários baseados em *Array of Bytes*, pode ser uma estratégia eficaz para a extração de chaves criptográficas de *ransomwares* em tempo de execução. A abordagem proposta mostrou-se funcional, reproduzível e eficiente no escopo experimental avaliado, indicando seu potencial como apoio a ações de mitigação e de resposta a incidentes de segurança da informação.

Embora os experimentos tenham sido conduzidos a partir de uma amostra sintética de *ransomware*, esta foi desenvolvida com base em comportamentos amplamente documentados em famílias reais, preservando características essenciais, como a geração dinâmica de chaves, o uso de rotinas criptográficas e o descarte do material sensível após a cifragem. Dessa forma, o uso de uma amostra controlada não compromete a validade dos resultados, ao mesmo tempo em que garante segurança, controle e reprodutibilidade experimental.

Conclui-se, portanto, que a extração automatizada de chaves criptográficas diretamente da memória volátil constitui uma contribuição relevante no contexto da computação forense, ao demonstrar a viabilidade técnica de recuperar informações críticas durante a execução de ransomwares. A abordagem apresentada reforça o papel da análise de memória como instrumento complementar às estratégias tradicionais de resposta a incidentes, oferecendo subsídios técnicos para a investigação, a recuperação de dados e a mitigação de danos em ambientes comprometidos.

6.1 Resposta às Questões de Pesquisa

A primeira questão de pesquisa (**QP1**) buscou verificar se é viável a extração de chaves criptográficas utilizadas por *ransomwares* a partir da análise da memória volátil, por meio da identificação de padrões recorrentes baseados em AOB.

Com base nos experimentos realizados, neste momento é importante reforçar que, embora os experimentos tenham utilizado uma amostra sintética, o foco deste trabalho recai sobre o comportamento intrínseco à utilização de bibliotecas criptográficas e à perma-

nência temporária de chaves na memória volátil, e não sobre características específicas de uma família de *ransomware*, ou seja, trata-se de um comportamento decorrente de propriedades fundamentais dos mecanismos criptográficos, independentemente da origem do binário analisado.

Neste contexto, compreende-se que essa abordagem é tecnicamente viável. A análise estática e dinâmica do binário da amostra sintética desenvolvida permitiu identificar com precisão as rotinas responsáveis pela geração, uso e armazenamento temporário da chave criptográfica em memória. A partir da análise dessas rotinas contidas no *ransomware*, foi possível derivar uma AOB estável, capaz de localizar o ponto exato do código e o *buffer* que continha a chave criptográfica durante a execução do processo malicioso.

Os resultados experimentais demonstraram que, em todas as execuções avaliadas, a chave criptográfica permaneceu acessível na memória volátil por um intervalo suficiente para permitir sua extração, validando empiricamente a hipótese central deste trabalho.

A segunda questão de pesquisa (**QP2**) investigou em que medida o uso de padrões AOB permite automatizar o processo de identificação e extração de chaves criptográficas na memória, reduzindo a dependência de análises manuais e de análises específicas para cada família de *ransomware*. Os resultados obtidos indicam que o uso de assinaturas AOB contribui significativamente para a automação do processo. A ferramenta desenvolvida, denominada *AOBTool*, foi capaz de realizar a varredura da memória do processo-alvo, identificar o padrão binário previamente definido e extrair automaticamente a chave criptográfica, sem a necessidade de intervenção manual durante a execução.

6.2 Principais Contribuições

A partir dos resultados alcançados, as principais contribuições deste trabalho podem ser destacadas da seguinte forma:

- ❏ Proposição de uma metodologia que integra engenharia reversa, análise de memória volátil e varredura por assinaturas AOB para a extração de chaves criptográficas utilizadas por *ransomwares*;
- ❏ Desenvolvimento e validação da ferramenta *AOBTool*, capaz de suspender processos, varrer regiões específicas da memória e extrair automaticamente chaves criptográficas em tempo de execução;
- ❏ Evidência experimental de que o tempo de extração da chave é inferior ao tempo necessário para a conclusão da rotina de cifragem do *ransomware* sintético desenvolvido, demonstrando a viabilidade técnica da abordagem proposta;

- ❑ Contribuição prática para a área de computação forense, oferecendo encaminhamentos sobre um método replicável que pode auxiliar investigações acadêmicas e estratégias de resposta a incidentes envolvendo *ransomwares*.

Os objetivos gerais e específicos definidos na Introdução foram alcançados. A análise estática e dinâmica do binário, realizada com o auxílio do *IDA Free* (Hex-Rays, 2025), do *Scylla* (NtQuery, 2015), possibilitou a identificação precisa das rotinas responsáveis pela geração, utilização e descarte da chave criptográfica durante a execução do *ransomware*. A partir dessas rotinas, foi possível derivar assinaturas *Array of Bytes* (AOB) robustas e estáveis, capazes de localizar, em memória, tanto a instrução crítica quanto o buffer que continha a chave criptográfica.

Embora a técnica baseada em assinaturas do tipo AOB apresente limitações frente a mecanismos avançados de empacotamento e polimorfismo, ela oferece vantagens técnicas relevantes no contexto de análises forenses conduzidas em ambientes controlados, como o baixo custo computacional associado à varredura de memória e a previsibilidade de seu comportamento durante a execução.

Além disso, a abordagem baseada em AOB apresenta elevada capacidade de integração com técnicas complementares, como heurísticas baseadas em entropia, identificação de chamadas a APIs criptográficas e análise comportamental. Dessa forma, o uso de assinaturas AOB deve ser compreendido não como uma solução isolada, mas como uma camada eficaz e coerente em uma estratégia mais ampla de investigação e de resposta a incidentes.

A ferramenta desenvolvida, denominada *AOBTool*, demonstrou precisão nas execuções realizadas. Em todos os experimentos reportados, a chave criptográfica foi localizada e extraída com sucesso, conforme apresentado na Tabela 2 e discutido na Seção 5.4.

Cabe destacar que a abordagem apresentada não se propõe a substituir mecanismos de defesa corporativos nem foi avaliada em ambientes de produção com controles de segurança ativos. O objetivo central consistiu em demonstrar a viabilidade técnica da extração de chaves criptográficas durante a execução do *ransomware*, em um contexto forense controlado. A avaliação em ambientes corporativos reais, sujeitos a mecanismos adicionais de proteção, constitui etapa posterior e extrapola o escopo deste trabalho.

6.3 Trabalhos Futuros

Como trabalhos futuros, destaca-se a realização de experimentos com amostras reais de *ransomware*, coletadas de repositórios controlados e amplamente utilizados na literatura, como *VirusShare, MalwareBazaar*, bem como de conjuntos disponibilizados por laboratórios acadêmicos e instituições de pesquisa. A aplicação da metodologia proposta em amostras reais permitirá avaliar a robustez das assinaturas *Array of Bytes* diante de

técnicas mais avançadas de ofuscação, empacotamento (*packing*) e diversidade de implementações criptográficas presentes em famílias de *ransomware* em circulação.

A análise desses aspectos contribuirá para refinar a metodologia, seja por meio do aprimoramento das assinaturas AOB, seja pela incorporação de estratégias adaptativas de sincronização e de monitoramento da execução do processo malicioso.

Referências

Akamai. **What is DarkSide Ransomware?** 2024. <https://www.akamai.com/pt/glossary/what-is-darkside-ransomware>. Citado na página 24.

AREMO, A. A. **Reverse Engineering**. 2021. ResearchGate. Acesso em: 14 dez. 2025. Disponível em: <https://www.researchgate.net/publication/356784389_Reverse_Engineering_Research>. Citado na página 26.

BELCIC, I. **What is Locky Ransomware?** 2019. <https://www.avast.com/pt-br/c-locky>. Citado na página 24.

BOYER, R. S.; MOORE, J. S. A fast string searching algorithm. **Communications of the ACM**, ACM New York, NY, USA, v. 20, n. 10, p. 762–772, 1977. Disponível em: <https://dl.acm.org/doi/abs/10.1145/359842.359859>. Citado na página 35.

CASEY, E. **Digital Evidence and Computer Crime**. 2011. Elsevier Inc. Citado na página 25.

Cheat Engine Team. **Cheat Engine**. 2024. <https://www.cheatengine.org/downloads.php>. Versão 7.6. Citado 2 vezes nas páginas 31 e 58.

CISA. **MAR-AR20-216A: Chinese Malware Taidoor**. 2020. <https://hex-rays.com/case-studies/digital-forensics>. Relatório de Análise de Malware publicado pela Agência de Segurança Cibernética e Infraestrutura (CISA). Citado na página 26.

Cloudflare. **Ryuk Ransomware**. 2024. <https://www.cloudflare.com/pt-br/learning/security/ransomware/ryuk-ransomware/>. Citado na página 24.

DAVIES, S. R.; MACFARLANE, R.; BUCHANAN, W. J. Evaluation of live forensic techniques in ransomware attack mitigation. **Forensic Science International: Digital Investigation**, Elsevier, v. 33, p. 300979, 2020. Citado na página 16.

DeepStrike. **Ransomware Statistics 2025**. 2025. Disponível em: <https://deepstrike.io/blog/ransomware-statistics-2025>. Citado na página 15.

FRANCO, D. P.; SOARES, C.; SANTOS, J. Análise de ransomwares para identificação e extração binária de chaves criptográficas. **Revista Brasileira de Criminalística**, v. 13, n. 1, p. 143–155, 2024. Citado 2 vezes nas páginas 16 e 26.

Hex-Rays. **IDA Free - Disassembler and Debugger**. 2025. <https://hex-rays.com/ida-pro>. Versão 9.2. Citado 5 vezes nas páginas 26, 31, 46, 51 e 67.

IBM Security. **O que é ransomware**. 2023. Disponível em: <https://www.ibm.com/br-pt/topics/ransomware>. Citado 4 vezes nas páginas 15, 21, 22 e 23.

_____. **X-Force Threat Intelligence Index**. 2024. Disponível em: <https://www.ibm.com/reports/threat-intelligence>. Citado 2 vezes nas páginas 15 e 16.

JONES, J.; SHASHIDHAR, N. Ransomware analysis and defense - wannacry and the win32 environment. **International Journal of Information Security Science (IJISS)**, v. 6, n. 4, p. 57–69, 2017. Disponível em: <https://dergipark.org.tr/en/download/article-file/2160203>. Citado na página 27.

Kaspersky. **LockBit Ransomware**. 2024. <https://www.kaspersky.com.br/resource-center/threats/lockbit-ransomware>. Citado na página 24.

KERNER, S. M. **Ransomware trends, statistics and facts for 2025**. 2025. Disponível em: <https://www.techtarget.com/searchsecurity/feature/Ransomware-trends-statistics-and-facts>. Citado na página 14.

KOLBITSCH, C. et al. Effective and efficient malware detection at the end host. In: **USENIX security symposium**. N.A: USENIX Association, 2009. p. 351–366. Disponível em: <https://www.researchgate.net/publication/221260426_Effective_and_Efficient_Malware_Detection_at_the_End_Host>. Citado na página 16.

LEE, H.-W. Cryptography module detection and identification mechanism on malicious ransomware software. **Journal of Convergence on Internet of Things**, Korea Internet of Things Society, v. 9, n. 1, p. 1–7, 2023. Disponível em: <https://koreascience.or.kr/article/JAKO202317956807358.page>. Citado 2 vezes nas páginas 16 e 27.

LIGH, M. H. et al. **Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code**. Hoboken, Nova Jersey, EUA: Wiley Publishing, Inc., 2011. Citado 3 vezes nas páginas 25, 26 e 33.

MARINHO, R. et al. Introdução à análise de códigos maliciosos para ambiente windows. In: **Anais do XXII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais (SBSeg 2022)**. Porto Alegre: Sociedade Brasileira de Computação, 2022. Disponível em: ResearchGate. Acesso em: 14 dez. 2025. Disponível em: <https://www.researchgate.net/publication/364582051_Introducao_a_Analise_de_Codigos_Maliciosos_para_ambiente_Windows>. Citado na página 28.

MAZUR, A. H. B.; OLIVEIRA, P. R. de; MARTIMIANO, L. A. F. Descriptografando: Experimento em análise de memória volátil aplicada à defesa contra ransomware. In: **XXIV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg 2024)**. Sociedade Brasileira de Computação, 2024. Disponível em: <https://sol.sbc.org.br/index.php/sbseg/article/view/30025>. Citado 2 vezes nas páginas 16 e 27.

Microsoft. **Visual Studio Community 2022**. 2022. <https://visualstudio.microsoft.com/pt-br/vs/community/>. Citado 2 vezes nas páginas 31 e 60.

_____. **What is ransomware?** 2024. Disponível em: <https://www.microsoft.com/security/business/security-101/what-is-ransomware>. Citado na página 20.

_____. **Windows 10, versão 22H2 (ISO)**. 2024. <https://www.microsoft.com/pt-br/software-download/windows10>. Citado na página 31.

National Institute of Standards and Technology. **Guide to Integrating Forensic Techniques into Incident Response**. 2006. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-86.pdf>. Citado na página 25.

NtQuery. **Scylla - Process Dumping Tool**. 2015. <https://github.com/NtQuery/Scylla>. Versão 0.9.8. Citado 4 vezes nas páginas 31, 46, 47 e 67.

Oracle Corporation. **Oracle VM VirtualBox**. 2025. <https://www.virtualbox.org/>. Versão 7.2.2. Citado na página 31.

PLOSZEK, R.; ŠVEC, P.; DEBNÁR, P. Analysis of encryption schemes in modern ransomware. **Journal of Information Security and Applications**, v. 25, 2021. Disponível em: <https://hrcak.srce.hr/clanak/380443>. Citado na página 27.

SECURITY, I. **O que é o ransomware como serviço (RaaS)**. 2024. <https://www.ibm.com/br-pt/topics/ransomware-as-a-service>. Citado na página 23.

senator715. **IDA-Fusion (plugin)**. 2024. <https://github.com/senator715/IDA-Fusion>. Citado 3 vezes nas páginas 31, 57 e 58.

Splashtop Team. **Os 5 ataques de resgate mais devastadores de 2021... Até agora**. 2025. Disponível em: <https://www.splashtop.com/pt/blog/the-5-most-devastating-ransomware-attacks-of-2021so-far?srsltid=AfmBOooSDvaRqSYx02oB0MKH4zSTWFAoY8-TaaVBcQ3Mk5WK8yvAU6UU>. Citado na página 14.

Veeam Software. **Tendências de Ransomware 2025**. 2025. Disponível em: <https://go.veeam.com/ransomware-trends-pt>. Citado na página 15.

YOUNG, A.; YUNG, M. Cryptovirology: Extortion-based security threats and countermeasures. In: IEEE. **Proceedings 1996 IEEE Symposium on Security and Privacy**. [S.l.], 1996. p. 129–140. Citado na página 21.

ÁLVAREZ, V. M. **YARA: The Pattern Matching Swiss Army Knife for Malware Researchers**. 2013. <https://virustotal.github.io/yara/>. Citado 2 vezes nas páginas 26 e 27.