

Relatório técnico de identificação de compactação/proteção

Executável DOS protegido pelo ENCRI 3.30

Caracterização da amostra antes da engenharia reversa do algoritmo de descriptografia

Escrito por: Marcos Velasco, 29/07/2026 – www.velasco.com.br

1. Objetivo do relatório

Este relatório documenta, em ordem cronológica, o processo de investigação que levou à identificação, compreensão e remoção da proteção aplicada a um executável DOS pelo sistema **ENCRI 3.30**.

Antes mesmo de iniciar a análise técnica do executável, foi realizada uma pesquisa bibliográfica e documental com o objetivo de localizar informações públicas sobre o ENCRI 3.30, tais como sua documentação técnica, especificação do algoritmo, manuais de utilização, histórico de desenvolvimento, identificação de seu autor ou empresa responsável, exemplos de implementação ou quaisquer estudos anteriores relacionados ao sistema.

Essa pesquisa não produziu resultados relevantes. Não foram encontradas documentações técnicas, publicações, artigos, registros históricos ou informações confiáveis que descrevessem o funcionamento interno do ENCRI 3.30, seu algoritmo de proteção, sua arquitetura ou mesmo a identidade de seu desenvolvedor. Da mesma forma, não foram localizadas referências que permitissem compreender como os executáveis protegidos eram estruturados ou reconstruídos durante sua execução.

Dessa forma, toda a análise descrita neste relatório foi desenvolvida exclusivamente a partir da observação direta do executável protegido, sem o auxílio de documentação oficial, código-fonte, especificações técnicas ou informações previamente conhecidas sobre o mecanismo de proteção empregado.

O primeiro objetivo da investigação não foi recuperar o código-fonte do programa nem compreender sua lógica funcional. Antes disso, era necessário responder a uma questão fundamental:

O arquivo analisado continha realmente o programa original ou apenas um carregador responsável por reconstruí-lo em memória antes de sua execução?

Responder a essa pergunta era essencial para definir a metodologia de trabalho. Caso o executável já contivesse o programa original, a engenharia reversa poderia ser realizada diretamente sobre sua desmontagem. Entretanto, caso existisse uma camada intermediária de proteção, compactação, empacotamento ou criptografia, qualquer tentativa de reconstrução do código-fonte resultaria apenas na análise do mecanismo de proteção, e não da aplicação propriamente dita.

Assim, a primeira etapa da investigação concentrou-se exclusivamente na caracterização do executável. Foram analisados seu formato, sua organização interna, seu conteúdo hexadecimal, suas assinaturas textuais, o fluxo inicial de execução e o comportamento do código presente no ponto de entrada.

Somente após a reunião dessas evidências foi possível concluir, de forma fundamentada, que o arquivo analisado não continha diretamente o programa original em forma executável normal. Em

seu lugar, existia um pequeno carregador responsável por reconstruir outro bloco de código em memória e, posteriormente, transferir a execução para esse novo código.

Essa conclusão foi obtida pela combinação de diversas evidências técnicas, incluindo:

- análise estrutural do formato DOS .COM;
- inspeção hexadecimal do executável;
- identificação da assinatura **ENCRI 3.30**;
- ausência das estruturas normalmente encontradas em aplicações DOS;
- desmontagem do ponto de entrada do programa;
- análise detalhada do fluxo de execução;
- identificação de uma rotina repetitiva de transformação de bytes;
- observação da reconstrução de código em memória seguida da transferência do fluxo de execução.

Somente após a confirmação da existência dessa camada intermediária foi iniciada uma segunda etapa de investigação, dedicada exclusivamente à engenharia reversa do mecanismo de proteção ENCRI 3.30. O objetivo dessa nova fase passou a ser compreender integralmente seu algoritmo de funcionamento, desenvolver um descritografador compatível e, somente então, recuperar o executável original para permitir a análise do programa efetivamente desenvolvido pelo autor.

2. Escopo e premissas

A análise foi realizada sobre um executável DOS no formato .COM. Esse formato é especialmente simples: não possui o cabeçalho estruturado típico de um executável MZ e é carregado pelo DOS em um único segmento, começando no endereço lógico 0100h.

- Offset 0000h do arquivo corresponde ao endereço 0100h em memória.
- Offset 0100h do arquivo corresponde ao endereço 0200h em memória.
- Offset 0250h do arquivo corresponde ao endereço 0350h em memória.
- O primeiro byte do arquivo normalmente é também a primeira instrução executada pelo programa.

Essas relações foram fundamentais porque permitiram comparar os offsets vistos no editor hexadecimal com os endereços utilizados pelo código Assembly durante a execução.

3. Preservação da amostra original

3.1 Criação de cópias separadas

Antes de abrir o arquivo em qualquer ferramenta de edição, foi criada uma cópia de preservação e uma cópia de trabalho. A cópia de preservação permaneceu intocada; a cópia de trabalho foi usada em desmontagens, comparações e testes.

MONOCITO_ORIGINAL.COM
MONOCITO_TRABALHO.COM

3.2 Registro de integridade

Também é recomendável registrar o tamanho do arquivo e calcular um hash criptográfico, preferencialmente SHA-256. Isso permite demonstrar que a amostra analisada é a mesma ao longo de todo o processo.

- Nome do arquivo
- Tamanho em bytes
- Data e origem da obtenção
- Hash SHA-256
- Observações sobre o ambiente de execução

Essa etapa não identifica a proteção, mas garante que todas as conclusões posteriores possam ser reproduzidas sobre a mesma amostra.

4. Primeira hipótese: o arquivo poderia estar compactado ou protegido

Ao receber um executável antigo, não se deve partir imediatamente da suposição de que seu conteúdo corresponde ao código real do programa. Arquivos DOS podem ter sido processados por compactadores, criptografadores, protetores contra cópia ou ferramentas que adicionam um carregador na frente do programa original.

Essas ferramentas normalmente produzem um arquivo com duas partes:

1. Um pequeno programa inicial, chamado stub ou loader, que é executado primeiro.
2. Um bloco de dados contendo o programa original compactado ou criptografado.

Durante a execução, o stub reconstrói o segundo bloco em memória e transfere o controle para ele. Por esse motivo, a primeira desmontagem de um arquivo protegido mostra principalmente o código da proteção, não o código da aplicação.

5. Inspeção hexadecimal inicial

5.1 Abertura do arquivo em um editor hexadecimal

A primeira análise prática consistiu em abrir a cópia de trabalho em um editor hexadecimal. Nessa fase, não foi realizada nenhuma alteração. O objetivo era observar a distribuição dos bytes e procurar indícios visíveis de empacotamento ou proteção.

Foram procurados:

- Textos ASCII legíveis próximos ao início do arquivo.
- Assinaturas de ferramentas conhecidas.
- Regiões extensas com aparência aleatória ou alta entropia visual.
- Poucas mensagens de interface em relação ao tamanho do programa.
- Blocos repetidos ou estruturas semelhantes em diferentes posições.
- Código inicial muito pequeno seguido por uma grande massa de dados.

5.2 Descoberta da assinatura ENCRI

Durante essa inspeção foi encontrada, logo no início do arquivo, uma identificação textual associada ao ENCRI versão 3.30. A presença do texto ENCRI em uma posição fixa próxima ao começo do executável foi o primeiro indício concreto de que o arquivo havia sido processado por uma ferramenta externa.

Offset aproximado 0004h: "ENCRI - V3.30"

A assinatura, isoladamente, ainda não permitia afirmar se o arquivo estava compactado, criptografado ou apenas marcado. Entretanto, ela justificava uma investigação mais profunda do código inicial.

6. Sinais de que o conteúdo não era o programa original

6.1 Quantidade reduzida de textos legíveis

Um programa DOS interativo costuma conter mensagens, nomes de opções, textos de erro, rótulos de menu, nomes de arquivos e outras cadeias legíveis. Na amostra protegida, essas informações não apareciam na quantidade esperada ou estavam ausentes da região inicial.

A ausência dos textos esperados indicava que o conteúdo principal poderia estar transformado. Um bloco criptografado ou compactado deixa de apresentar as sequências ASCII originais de forma diretamente reconhecível.

6.2 Estrutura inicial incompatível com a aplicação

Ao observar os primeiros bytes como instruções 8086, o fluxo não apresentava imediatamente rotinas de interface, leitura de teclado, impressão de mensagens ou processamento funcional. Em vez disso, havia inicialização de registradores, preparação de ponteiros, contadores e uma rotina de transformação de memória.

6.3 Grande bloco tratado como dados

O código inicial acessava uma área posterior do arquivo como uma sequência de bytes a ser lida e transformada. Esse comportamento é típico de um stub: uma parte pequena do arquivo contém código executável e a parte maior funciona inicialmente como dados codificados.

7. Primeira desmontagem do arquivo ainda protegido

7.1 Configuração correta do disassembler

O arquivo foi carregado em um disassembler configurado para arquitetura Intel 8086, modo real de 16 bits e formato binário DOS .COM. O endereço de origem foi definido como 0100h, pois é nesse offset que o DOS inicia a execução de programas .COM.

- Arquitetura: Intel 8086/x86 de 16 bits.
- Modo: Real Mode.
- Formato: binário simples ou DOS COM.
- Endereço inicial: 0100h.
- Ponto de entrada: primeiro byte do arquivo.

7.2 Expectativa antes da desmontagem

Se o arquivo contivesse diretamente o programa real, seria esperado encontrar logo no início rotinas de inicialização da aplicação, chamadas ao DOS, referências a mensagens, funções de teclado, manipulação da tela e estruturas relacionadas à lógica do programa.

7.3 Resultado realmente encontrado

A desmontagem revelou uma rotina compacta que:

- Lia valores internos em offsets fixos.
- Configurava registradores como fonte, destino e contador.
- Percorria um bloco byte a byte.
- Aplicava operações aritméticas envolvendo os bytes anteriores.
- Gravava o resultado em outra área da memória.
- No final, transferia a execução para o conteúdo recém-gerado.

Esse comportamento não correspondia à lógica normal da aplicação. Correspondia ao funcionamento de um carregador responsável por restaurar outro programa antes de executá-lo.

8. Análise do fluxo de execução

8.1 Preparação de origem, destino e tamanho

A rotina inicial preparava um endereço de origem para os dados codificados, um endereço de destino para os bytes recuperados e um contador indicando quantos bytes seriam processados. A existência dessa tríade - origem, destino e comprimento - é característica de rotinas de cópia, descompressão oucriptografia.

8.2 Processamento repetitivo byte a byte

Foi identificado um laço que repetia praticamente as mesmas operações para cada byte. A forma lógica observada era equivalente a:

```
AL = [SI]
AH = AL
AL = AL - DH - DL - 0Dh
DH = AL
DL = AH
[DI] = AL
SI = SI + 1
DI = DI + 1
LOOP
```

A instrução lia um byte codificado, preservava seu valor original, calculava o byte recuperado, atualizava o estado interno do algoritmo e gravava o resultado no destino.

8.3 Dependência entre bytes consecutivos

Os registradores DL e DH eram atualizados após cada iteração. Isso significava que o cálculo de um byte dependia dos valores processados na iteração anterior. Esse padrão é compatível com um mecanismo de criptografia ou ofuscação encadeada, e não com uma simples cópia de memória.

8.4 Transferência para outra região

Após reconstruir o bloco, o fluxo do programa deixava a rotina inicial e passava a executar a região produzida. A transferência de controle para o conteúdo recém-escrito foi a evidência decisiva de que o primeiro código era apenas um estágio de inicialização.

9. Por que não se tratava apenas de dados comuns

Uma rotina que processa dados não significa necessariamente que o programa esteja protegido. Muitos programas legítimos manipulam tabelas e arquivos. A conclusão foi possível porque várias evidências apareceram simultaneamente:

Evidência	Observação	Interpretação
Assinatura ENCRI	Texto ENCRI 3.30 próximo ao início	Arquivo processado por ferramenta externa
Código inicial pequeno	Poucas rotinas antes de um bloco grande	Possível stub ou loader
Poucos textos da aplicação	Mensagens esperadas não apareciam	Conteúdo principal transformado
Laço byte a byte	Transformação aritmética sequencial	Descriptografia ou ofuscação
Origem e destino separados	Dados lidos de uma região e gravados em outra	Reconstrução em memória
Salto final	Execução transferida ao bloco reconstruído	Código real só existe após processamento
Nova assinatura após uma camada	Resultado ainda parecia ENCRI	Proteção aplicada repetidamente

10. Formulação da hipótese técnica

Com base nas evidências, foi formulada a seguinte hipótese de trabalho:

O arquivo analisado não contém diretamente o programa original em estado executável normal. Seu ponto de entrada pertence a um stub do ENCRI 3.30, responsável por descriptografar ou reconstruir um bloco em memória e, somente depois, transferir o controle para o programa protegido.

Essa hipótese explicava de forma coerente todas as observações obtidas até aquele momento. Também previa um teste: se o bloco fosse reconstruído corretamente, o resultado deveria apresentar código 8086 coerente, mais textos legíveis e, possivelmente, uma nova camada ENCRI ou o programa original.

11. Confirmação prática da hipótese

11.1 Extração manual de uma camada

A rotina identificada foi traduzida para pseudocódigo. Em seguida, seus parâmetros foram obtidos diretamente dos offsets utilizados pelo stub: comprimento do bloco, chave inicial e endereço de início dos dados codificados.

paragraphCount = palavra de 16 bits no offset 0100h

decryptedLength = paragraphCount * 16

DL = byte no offset 0102h
DH = byte no offset 0103h
source = offset 0250h

11.2 Reprodução do cálculo

Para cada byte, foi reproduzida a operação observada no Assembly:

```
decrypted = (encrypted - DH - DL - 0x0D) & 0xFF  
DH = decrypted  
DL = encrypted
```

O uso de & 0xFF reproduz o comportamento de um registrador de oito bits, descartando qualquer parte do resultado que ultrapasse o intervalo de 00h a FFh.

11.3 Resultado da primeira remoção

Quando uma camada foi removida, o resultado não se tornou imediatamente o programa final. A saída apresentou novamente a estrutura e a assinatura esperadas de uma camada ENCRI. Isso demonstrou que a proteção havia sido aplicada várias vezes sobre o mesmo programa.

A descoberta das camadas sucessivas confirmou definitivamente que o arquivo inicial era um pacote protegido, e não o executável original em forma direta.

12. Decisão de realizar a engenharia reversa do ENCRI 3.30

Somente após a caracterização e confirmação da camada de proteção foi tomada a decisão de realizar engenharia reversa específica do stub ENCRI 3.30.

A decisão foi necessária porque tentar analisar diretamente o arquivo protegido levaria aos seguintes problemas:

- O disassembler interpretaria dados criptografados como se fossem instruções.
- As referências cruzadas não representariam o programa verdadeiro.
- As mensagens e tabelas originais permaneceriam ocultas.
- O fluxo principal visto seria o da proteção, não o da aplicação.
- Qualquer tentativa de reconstrução de código-fonte começaria pelo componente errado.

A estratégia correta passou a ser dividida em duas engenharias reversas distintas:

3. Primeira engenharia reversa: compreender o ENCRI 3.30 e desenvolver um programa capaz de remover suas camadas.
4. Segunda engenharia reversa: desmontar e estudar o programa real depois de totalmente descriptografado.

13. Requisitos definidos para o Encri33Decryptor

A partir da análise, foram definidos os requisitos do utilitário que automatizaria a remoção das camadas:

- Receber um arquivo .COM pela linha de comando.
- Validar se os dados possuem uma camada ENCRI 3.30 reconhecível.

- Ler corretamente palavras de 16 bits em little-endian.
 - Calcular o tamanho do bloco em parágrafos de 16 bytes.
 - Obter os valores iniciais de DL e DH.
 - Reproduzir exatamente o cálculo byte a byte do Assembly.
 - Repetir a remoção enquanto outra camada válida for encontrada.
 - Não sobrescrever o arquivo original.
 - Informar quantas camadas foram removidas.
 - Gravar o executável final com um nome diferente.

14. Critérios utilizados para considerar a descoberta válida

A conclusão de que o arquivo estava compactado ou protegido não foi baseada apenas em aparência. Foram adotados critérios técnicos cumulativos:

1. A assinatura ENCRI estava presente em posição coerente.
2. O ponto de entrada executava uma rotina de reconstrução, não a lógica da aplicação.
3. O laço transformava bytes usando estado interno encadeado.
4. O código gravava uma nova imagem em memória.
5. O fluxo era transferido para essa nova imagem.
6. A reprodução externa da rotina gerava uma nova camada reconhecível.
7. A repetição do processo terminava em um executável diferente e funcional.

O conjunto desses resultados eliminou a possibilidade de a assinatura ser apenas uma mensagem decorativa e confirmou a existência de um mecanismo real de proteção.

15. Diferença entre compactação, criptografia e proteção

No início da investigação, o termo compactado podia ser usado de forma genérica para indicar que o programa original não estava diretamente disponível. Após a análise, tornou-se possível fazer uma distinção mais precisa:

- Compactação reduz o tamanho e exige descompressão antes da execução.
- Criptografia transforma os bytes por meio de uma chave ou estado, ocultando o conteúdo.
- Empacotamento combina um stub com dados transformados e gera um novo executável.
- Proteção é o objetivo geral de dificultar cópia, análise ou modificação.

O comportamento observado no ENCRI 3.30 corresponde a um empacotador/protetor que aplica uma transformação criptográfica byte a byte. Portanto, tecnicamente, o arquivo não estava apenas compactado: ele estava encapsulado em sucessivas camadas de proteção e criptografia.

16. Resultado dessa fase da investigação

Ao final da fase de caracterização, já era possível afirmar com segurança que:

- O arquivo de entrada era um DOS .COM válido.
- O primeiro código executado não era o programa original.
- A assinatura ENCRI identificava um stub de proteção versão 3.30.
- O stub possuía uma rotina de descritografia encadeada.
- O programa real era reconstruído em memória.
- A proteção havia sido aplicada em múltiplas camadas.
- Uma análise funcional confiável exigia remover todas essas camadas antes da segunda desmontagem.

17. Conclusão

A decisão de desenvolver o Encri33Decryptor não foi tomada apenas porque o texto ENCRI aparecia no arquivo. Ela foi consequência de uma sequência verificável de observações: inspeção hexadecimal, análise do formato .COM, desmontagem do ponto de entrada, identificação de um laço de transformação, acompanhamento do fluxo de memória, observação da transferência de controle e reprodução prática de uma camada.

Essa metodologia evitou o erro de tentar reconstruir o programa a partir do stub de proteção. Primeiro foi necessário compreender e neutralizar o mecanismo que escondia o executável. Somente depois seria tecnicamente válido desmontar novamente o arquivo e iniciar a engenharia reversa da aplicação propriamente dita.

18. Resumo cronológico do processo

Etapa	Procedimento
1	Preservar o arquivo original e criar uma cópia de trabalho.
2	Confirmar que o arquivo era um executável DOS no formato .COM.
3	Abrir o arquivo em um editor hexadecimal, sem modificá-lo.
4	Localizar a assinatura textual ENCRI 3.30 próxima ao início.
5	Observar a ausência de textos e estruturas esperadas da aplicação.
6	Carregar o arquivo em um disassembler 8086 com origem em 0100h.
7	Verificar que o ponto de entrada preparava ponteiros e processava memória.
8	Identificar o laço aritmético byte a byte e o estado mantido em DL/DH.
9	Constar que os resultados eram gravados em outra área da memória.
10	Observar a transferência de execução para o bloco reconstruído.
11	Traduzir a rotina para pseudocódigo e reproduzir uma camada externamente.
12	Encontrar outra camada ENCRI no resultado e confirmar a aplicação repetida.
13	Concluir que o arquivo estava protegido e não continha diretamente o programa real.
14	Definir a criação do Encri33Decryptor para automatizar a remoção completa.
15	Somente depois, planejar uma nova desmontagem do executável limpo.