RansomEXX — Análise do Ransomware Utilizado no Ataque ao STJ

By Gustavo Palazolo

Published: 2020-11-16 · Archived: 2026-04-05 22:01:42 UTC

Acreditamos que a grande maioria dos brasileiros souberam de um ataque cibernético que infelizmente ocorreu no Supremo Tribunal de Justiça recentemente. De acordo com a nota oficial:

> "O Superior Tribunal de Justiça (STJ) comunica que a rede de tecnologia da informação do tribunal sofreu um ataque hacker, nessa terça-feira (3), durante o período da tarde, quando aconteciam as sessões de julgamento dos colegiados das seis turmas. A presidência do tribunal já acionou a Polícia Federal para a investigação do ataque cibernético."

Além da nota oficial, o Departamento de Segurança da Informação (DSI), publicou duas notas com informações técnicas sobre o ataque, incluindo o hash (MD5/SHA1) de dois arquivos:

> (notepad.exe) — 9df15f471083698b818575c381e49c914dee69de
> (svc-new/svc-new) — 3bf79cc3ed82edd6bfe1950b7612a20853e28b09

Apesar dos detalhes técnicos da nota, pouco sabe-se sobre o ataque e sobre o funcionamento dos artefatos acima.

O primeiro deles (notepad.exe) é um loader conhecido por Vatet, que deixaremos de fora do artigo, pois ele está sendo analisado no canal Papo Binário, confere lá!

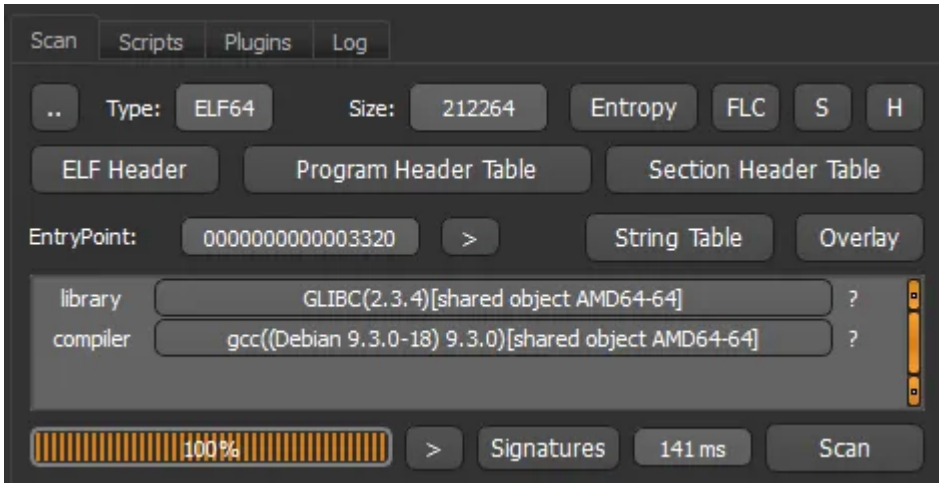
Ainda sobre a nota, o incidente teve proporções críticas, a área de TI do STJ recomendou aos usuários — ministros, servidores, estagiários e terceirizados — que não utilizem computadores, ainda que os pessoais, que estejam conectados com algum dos sistemas informatizados da Corte, até que seja garantida a segurança do procedimento.

Com isso em mente, eu e mais dois amigos, Ialle Teixeira e Felipe Duarte, fizemos uma análise do segundo arquivo e resolvemos publicar aqui, com o único intuito de mostrar a engenharia reversa do malware e talvez ajudar as pessoas a entenderem seu funcionamento e de alguma forma, contribuir com mais informações úteis para os órgãos de interesse.

RansomEXX

O arquivo pertence a uma família de ransomware conhecida por RansomEXX, ou Defray777. Ao longo do ano de 2020, este ransomware ficou conhecido devido a grandes ataques, como o que ocorreu no Departamento de Transporte do Texas (TxDOT).

Até o começo de novembro de 2020, somente versões Windows do RansomEXX haviam sido detectadas, contudo, uma variante para Linux foi recentemente descoberta e publicada pela Kaspersky.

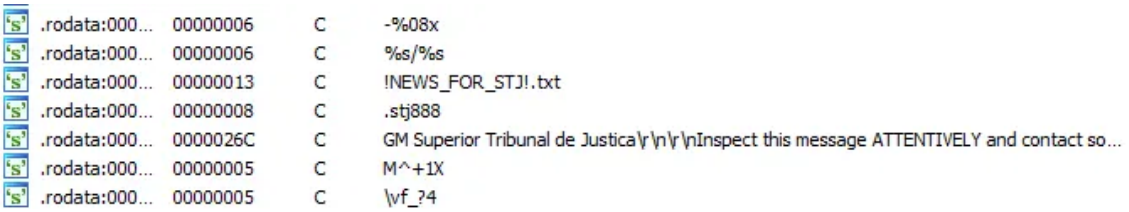


Ferramenta DIE mostrando detalhes sobre o arquivo.

O arquivo publicado pelo DSI é justamente uma versão do RansomEXX para Linux, sendo um binário ELF de 64-bits compilado com GCC, como indica a imagem acima.

A primeira coisa a se notar é que o malware não possui técnicas anti-análise e nem ofuscação no código ou nas strings.

Press enter or click to view image in full size



Parte das strings do RansomEXX.

Outro ponto é que o arquivo foi compilado com as informações de debug, ou seja, através de engenharia reversa, podemos obter detalhes interessantes como nome de variáveis e métodos utilizado pelo atacante.

Nas próximas seções, iremos mostrar os detalhes das principais funções do RansomEXX.

main()

A função “main” do malware é responsável por:

1. Chamar a função “**GeneratePreData**”, que cria o contexto de criptografia;
2. Criar uma thread que executa a mesma função a cada 0.18 segundos, gerando **novas chaves de criptografia**;
3. Encriptar os arquivos de um determinado path, que foi passado via argumento na **linha de comando** de execução.

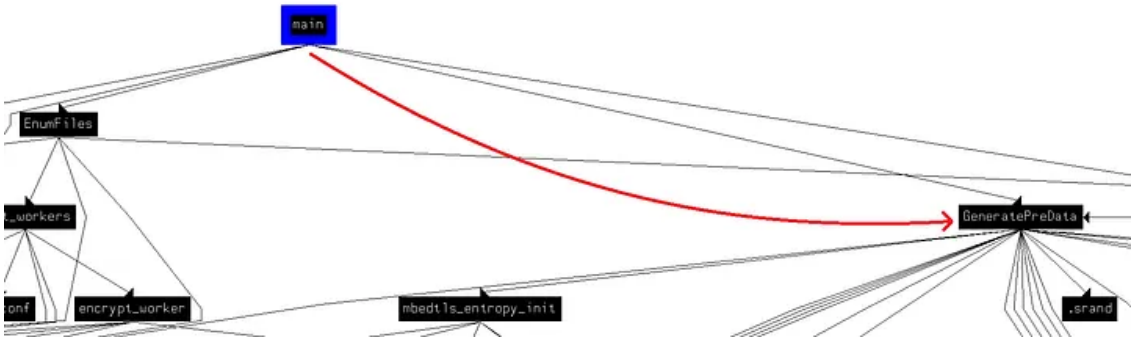
```
int main(int argc, const char **argv, const char **envp)
{
  pthread_t newthread;
  int i;

  GeneratePreData();
  pthread_create(&newthread, 0, regenerate_pre_data, 0);
  for ( i = 1; i < argc; ++i )
  {
    puts(argv[i]);
    EnumFiles(argv[i]);
  }
  return 0;
}
```

Pseudocode em C da função “main()” do RansomEXX.

GeneratePreData

Essa função é responsável por gerar a chave de criptografia que será utilizada pelo malware. Para isso, ele utiliza uma biblioteca open-source chamada mbedtls.

Press enter or click to view image in full size



Representação gráfica das funções, geradas pelo IDA.

Na primeira etapa da função, o malware gera um valor que é utilizado no parâmetro “custom” da função “mbedtls_ctr_drbg_seed”.

```
mov     edi, 0          ; timer
call     time
mov     edi, eax        ; seed
call    _srand
call    _rand
mov     r13d, eax
call    _rand
mov     r12d, eax
call    _rand
mov     ebx, eax
call    _rand
mov     edx, eax
lea     rax, [rbp+custom]
mov     r9, r13
mov     r8, r12
mov     rcx, rbx
lea     rsi, format     ; "%08x%08x%08x%08x"
mov     rdi, rax        ; s
```

Primeira parte da função “GeneratePreData”.

Para isso, ele chama a função “time”, que retorna a data atual representada em um valor de 4 bytes, por exemplo:

```
>>>
>>> valor = 0x5FB14F9B
>>>
>>> print(datetime.fromtimestamp(valor))
2020-11-15 12:56:11
>>>
```

Valor hexadecimal retornado pela função “time”.

Este valor é então passado para a função srand, que alimentará as quatro chamadas subsequentes da função rand. O resultado destas chamadas são então concatenados em uma string de 32 caracteres, parecido com um hash MD5.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   32 63 66 62 32 66 37 65 34 37 36 30 35 35 32 62   2cfb2f7e4760552b
00000010   30 66 33 63 64 30 38 65 34 66 30 64 31 62 38 62   0f3cd08e4f0d1b8b
```

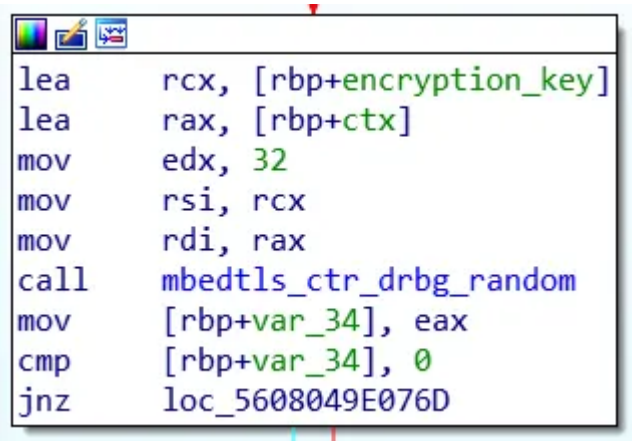
String gerada pela concatenação dos valores srand + rand.

Este valor é utilizado para garantir que a inicialização da chave de criptografia sempre tenha um ponto de partida diferente, a própria biblioteca possui um tutorial mostrando a utilização deste valor na função “mbedtls_ctr_drbg_seed”.

```
mbedtls_ctr_drbg_context ctr_drbg;
char *personalization = "my_app_specific_string";

mbedtls_ctr_drbg_init( &ctr_drbg );

ret = mbedtls_ctr_drbg_seed( &ctr_drbg , mbedtls_entropy_func, &entropy,
                 (const unsigned char *) personalization,
                 strlen( personalization ) );
```

Exemplo de utilização do método “mbedtls_ctr_drbg_seed”.

Para finalizar, o autor do malware utilizou a função “mbedtls_ctr_drbg_random” para gerar uma chave de criptografia aleatória de 256 bits.



Ultima etapa da geração da chave de criptografia.

A criação da chave de criptografia pode ser resumida com a seguinte representação em C:

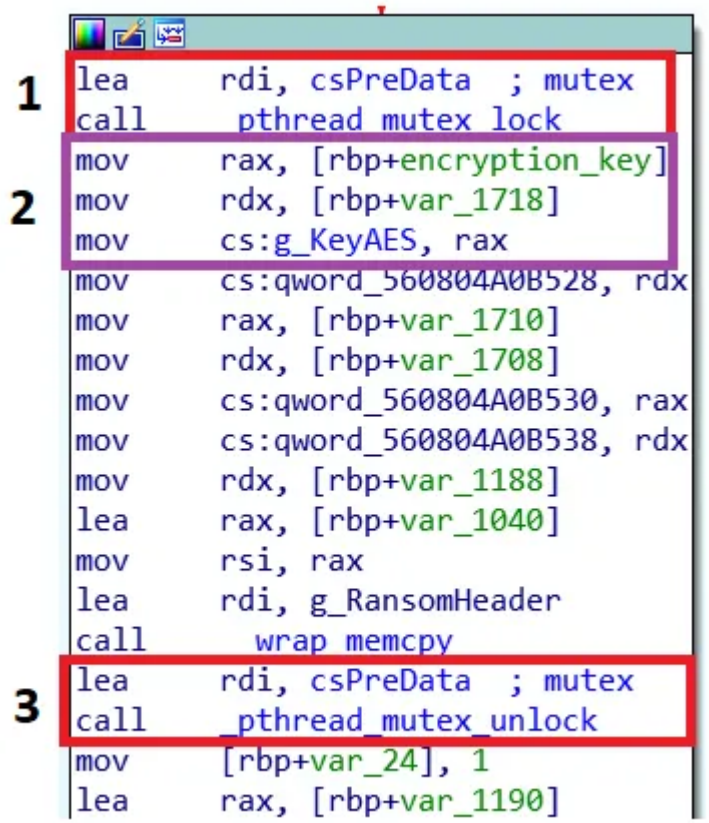
Press enter or click to view image in full size

```
timestamp = time(0);
srand(timestamp);
v1 = rand();
v2 = rand();
v3 = rand();
v4 = rand();
sprintf(&custom, "%08x%08x%08x%08x", v4, v3, v2, v1);
mbedtls_ctr_drbg_init(&ctx);
mbedtls_entropy_init(&p_entropy);
mbedtls_ctr_drbg_seed(&ctx, mbedtls_entropy_func, &p_entropy, &custom, strlen(&custom));
mbedtls_ctr_drbg_random(&ctx, &encryption_key, 32);
```

Pseudocode em C, representando a criação da chave de criptografia.

Como vamos mostrar mais pra frente, este ransomware faz a utilização de múltiplas threads para acelerar o processo de criptografia. Enquanto uma thread gera a chave, a outra é responsável por encriptar os arquivos. Para ler a chave, o malware faz a utilização de mutex, através da função pthread_mutex_lock e pthread_mutex_unlock.

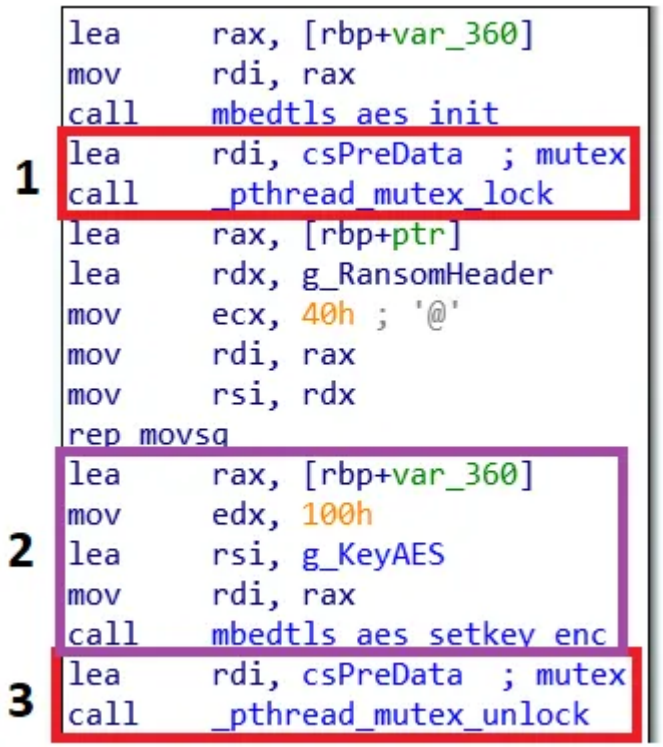
Press enter or click to view image in full size



Malware copiando a chave de criptografia, que será lida por outra thread.

1. O mutex é travado, indicando para as outras threads que a chave de criptografia não pode ser acessada ainda;
2. A chave de criptografia nova é copiada para uma variável, que será lida por outras threads;
3. O mutex é destravado, indicando que o valor já pode ser acessado.

Já na função responsável por encriptar os arquivos, que mostraremos ao longo do post, a mesma lógica é utilizada:



1. O mutex é travado, indicando que a chave está sendo lida pela função;
2. A chave de criptografia é utilizada;
3. O mutex é destravado, indicando que o endereço está livre para receber novas chaves.

Esta lógica permite que tanto na hora de gerar a chave quanto na hora de utilizá-la, nenhum valor seja corrompido, pois ela só poderá ser lida/alterada quando o mutex estiver liberado.

Em resumo, para encriptar os arquivos, o malware gera uma chave aleatória de 256-bit a cada 0.18 segundos, que é lida por uma outra thread, que está encriptando os arquivos.

Ok, mas se a chave de criptografia é gerada dinamicamente e possui um valor novo a cada 0.18 segundos, como o atacante sabe qual chave utilizar na hora de descriptografar os arquivos?

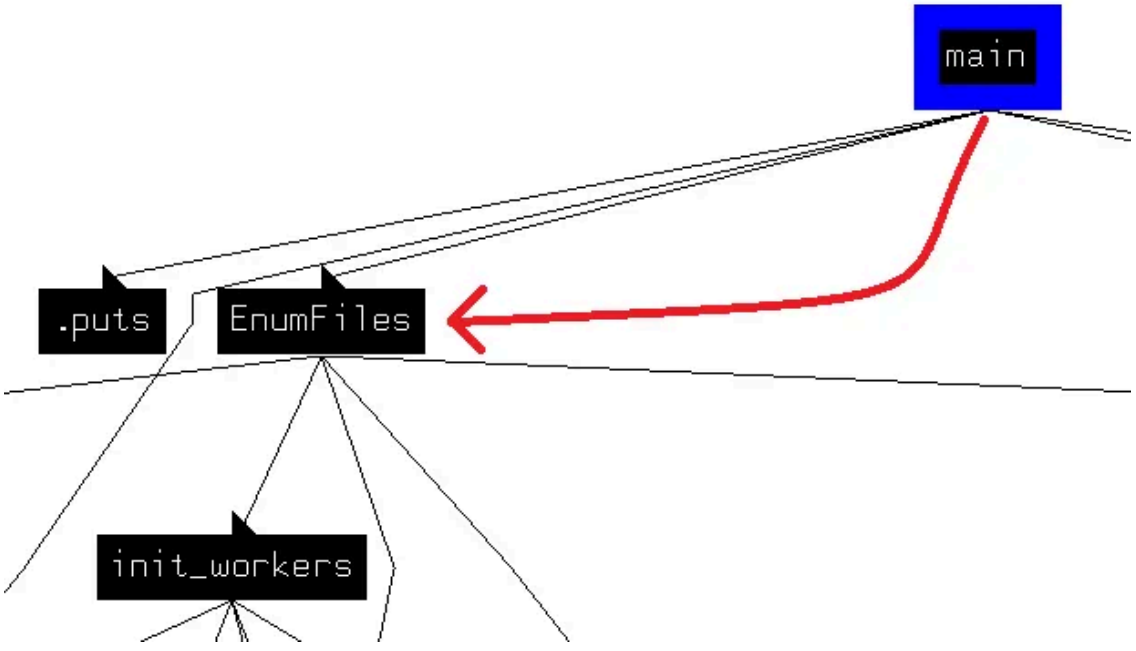
Para recuperar a chave utilizada, o malware encripta o valor utilizando uma chave RSA-4096 pública que está presente no binário:

Chave RSA pública do atacante.

Após encriptar a chave, o valor é salvo junto com o arquivo encriptado, garantindo que somente o atacante possa saber qual foi o valor utilizado na criptografia, para então decifrá-lo.

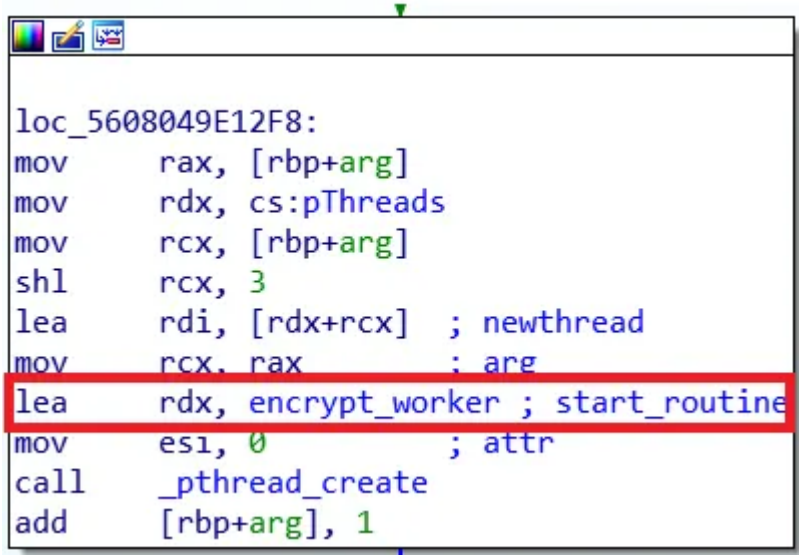
EnumFiles

Conforme mostramos na imagem do pseudocode da “main”, esta função recebe como parâmetro o caminho de onde estão os arquivos a serem encriptados.



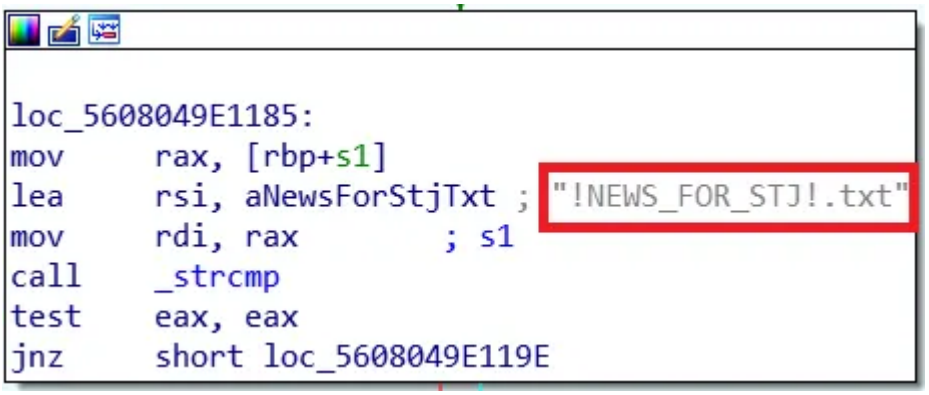
Representação gráfica das funções, geradas pelo IDA.

Em primeiro lugar, a função cria múltiplas threads que chamam a função “**encrypt_worker**”, que contém o código responsável por encriptar os arquivos.



Malware criando threads para executar a função “encrypt_worker”.

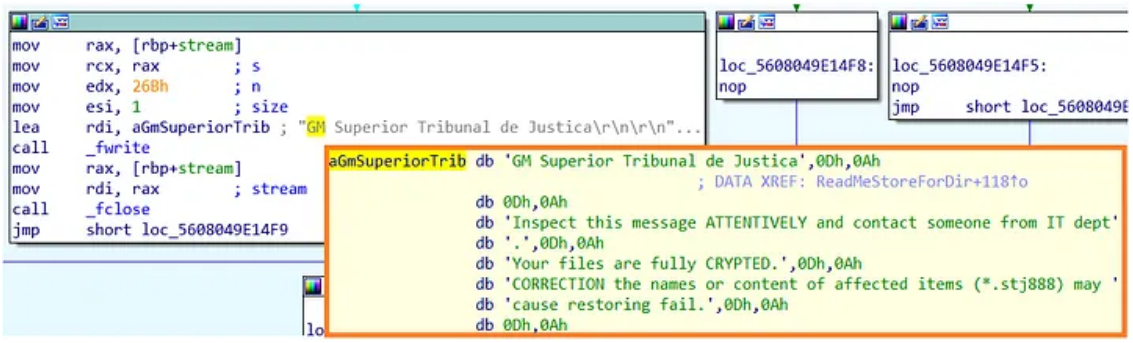
Além disso, ele também chama a função “**list_dir**”, que lista todo o diretório e executa a função “**ReadMeStoreForDir**”, que cria a nota de resgate no mesmo local, com o nome de “**!NEWS_FOR_STJ!.txt**”.



Nome da nota de resgate.

Após criar o arquivo, o conteúdo é escrito usando a função “fwrite”.

Press enter or click to view image in full size



Conteúdo da nota de resgate.

A nota de resgate possui informações para que o STJ entre em contato com o atacante. Vale ressaltar que o email deixado para contato foi reportado como abuse e removido pela ProtonMail, o que possivelmente pode dificultar o

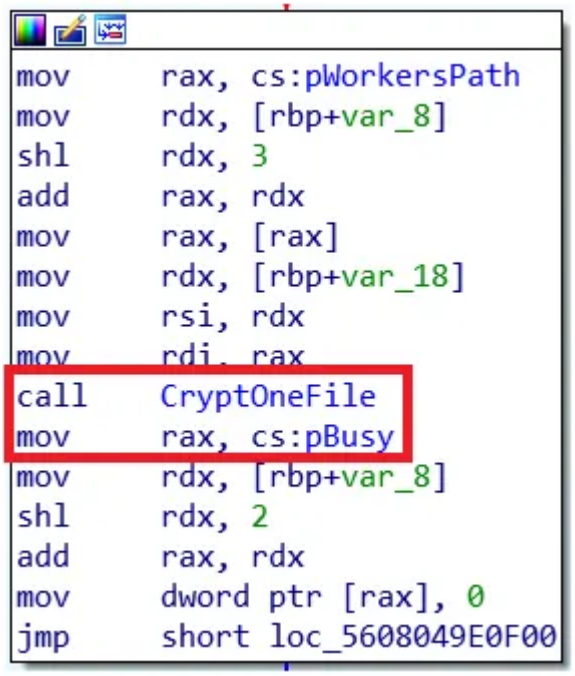
contato com o autor do ataque, como podemos ver no teste abaixo:

Press enter or click to view image in full size

Email da nota de resgate inválido.

CryptOneFile / CryptOneBlock

Conforme mencionado acima, a função **“encrypt_worker”** será executada em uma thread separada e será responsável por executar o código que irá efetivamente encriptar o arquivo. Para cada arquivo, a função **“CryptOneFile”** é chamada.



Chamada para o CryptOneFile.

Dentro da “**CryptOneFile”**, a chave gerada pela outra thread é acessada conforme mostramos nas etapas acima, inicializando uma criptografia AES, com as funções mbedtls_aes_init e mbedtls_aes_setkey_enc.

Uma vez inicializada, o arquivo é encriptado em blocos pela função “**CryptOneBlock**”, que encripta o arquivo utilizando **AES** no modo **ECB**.

Press enter or click to view image in full size

Chamada para a função que encripta os dados, utilizando AES+ECB.

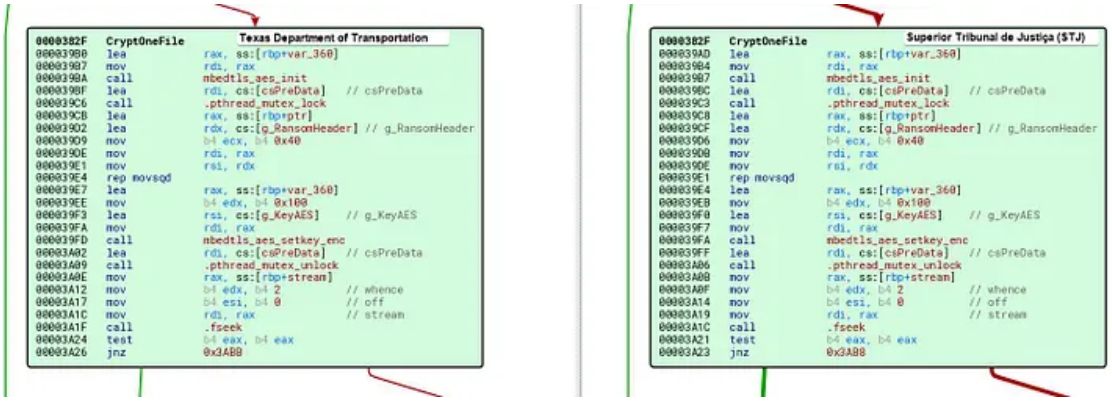
STJ x Departamento de Transportes do Texas

Vale ressaltar também a comparação entre o ransomware utilizado no ataque ao Departamento de Transporte do Texas (TxDOT), pois existe uma grande similaridade entre os binários, que são da mesma família:

Press enter or click to view image in full size

Função que escreve a nota de resgate nos dois executáveis, tendo somente o texto alterado.

Press enter or click to view image in full size



Função que inicializa e lê a chave de criptografia, nos dois executáveis.

Além de levantar diversas hipóteses, também podemos perceber a importância do uso da inteligência gerada a partir de outros incidentes, como possibilidade e melhoria de novos mecanismos de proteção, como uma simples regra via YARA ou qualquer outra tecnologia.

Como podemos ver abaixo, os dois executáveis possuem 99% de similaridade:

Name	Value
basicBlock matches (library)	0
basicBlock matches (non-library)	3990
basicBlocks primary (library)	0
basicBlocks primary (non-library)	3990
basicBlocks secondary (library)	0
basicBlocks secondary (non-library)	3990
flowGraph edge matches (library)	0
flowGraph edge matches (non-library)	5245
flowGraph edges primary (library)	0
flowGraph edges primary (non-library)	5245
flowGraph edges secondary (library)	0
flowGraph edges secondary (non-library)	5245
function matches (library)	49
function matches (non-library)	426
functions primary (library)	49
functions primary (non-library)	426
functions secondary (library)	49
functions secondary (non-library)	426
instruction matches (library)	0
instruction matches (non-library)	36613
instructions primary (library)	0
instructions primary (non-library)	36614
instructions secondary (library)	0
instructions secondary (non-library)	36613
basicBlock: MD index matching (top down)	52
basicBlock: call reference matching	4
basicBlock: edges MD index (top down)	41
basicBlock: edges prime product	3336
basicBlock: hash matching (4 instructions minimum)	505
basicBlock: jump sequence matching	12
basicBlock: prime matching (4 instructions minimum)	40
function: MD index matching (flowgraph MD index, top down)	1
function: call reference matching	3
function: name hash matching	471
Confidence	0.992594
Similarity	0.992592

Resultado da ferramenta BinDiff, comparando os dois executáveis.

Demonstração

Para demonstrar o funcionamento do ransomware, gravamos um pequeno vídeo executando-o em uma VM Linux (Ubuntu).

Demonstração do funcionamento do RansomEXX.

Conclusão

Neste post mostramos os principais detalhes técnicos do ransomware utilizado no ataque ao STJ. O principal objetivo do malware é somente a criptografia dos arquivos, o que é uma operação incomum se compararmos com as famílias de ransomware mais recentes, onde há comunicação com C2 e funcionalidades adicionais.

Lamentamos muito o ocorrido com o STJ e esperamos ter contribuído com alguma informação útil, vale ressaltar que estamos à disposição do governo para auxiliar em qualquer análise adicional.

Vale lembrar que tanto a análise do arquivo e o artigo foram produzidos no dia de hoje, 15 de Novembro de 2020, caso o artigo possua algum erro, por gentileza, agradecemos qualquer feedback.

IOCs

Arquivo:

> *Nome: svc-new/svc-new*
> *SHA256: 08113ca015468d6c29af4e4e4754c003dacc194ce4a254e15f38060854f18867*

MITRE:

> Data Encrypted for Impact https://attack.mitre.org/techniques/T1486/
> ID: T1486

Fontes

http://www.stf.jus.br/portal/cms/verNoticiaDetalhe.asp?idConteudo=454634
https://www.ctir.gov.br/arquivos/alertas/2020/alerta_especial_2020_07_atualizacao_ataques_de_ransomware.pdf
https://attack.mitre.org/techniques/T1486/
https://attack.mitre.org/mitigations/M1053/
https://www.ctir.gov.br/arquivos/alertas/2020/alerta_2020_03_ataques_de_ransomware.pdf

Texto revisado pela Jornalista Aline Palazolo Eiras.
