

Insper
Engenharia da Computação
Relatório Final de Iniciação Tecnológica

Matheus Silva Melo de Oliveira

Orientador(a): Prof. Rodolfo da Silva Avelino

**Estudo Analítico de Crypto Ransomwares utilizados em ataques cibernéticos
no Brasil em 2021**

São Paulo

2022

Resumo

Com o crescimento exponencial de dispositivos conectados à redes, ataques utilizando softwares maliciosos de encriptação se tornaram uma comunidade, atingindo diferentes escopos da sociedade, seja em seu aspecto público ou privado. Dessa forma, analisar comportamento destas amostras, com objetivo principal de se propagar estratégias de mitigação se tornam uma necessidade. Neste relatório, são propostas análises históricas e técnicas que dissequem softwares maliciosos, com objetivo de amplificar análises mitigatórias e informativas dessas ameaças, além de prototipar uma solução que seja eficiente para mitigação e redução de danos contra as ameaças analisadas, gerando assim uma ferramenta interessante para o meio cibernético no certame de análise de *Malware*.

Palavras-chave: Cibersegurança. Ransomware. Detecção de Ransomware. Antivírus. Resposta a incidentes.

Abstract

With the exponential growth of devices connected to networks, attacks using malicious encryption software became a commonality, affecting different spheres of society, whether in their public or private aspects. Thus, analyzing the behavior of these samples, with the main objective of propagating mitigation strategies, becomes a necessity. In this report, historical and technical analyses are proposed to identify malicious software, with the objective of amplifying mitigatory and informative analyses of these threats, in addition to prototyping a solution that is efficient for mitigating and reducing damage against the analyzed threats, thus generating an interesting tool for the cyber environment in the Malware analysis contest.

Keywords: Cibersecurity. Ransomware. Ransomware Detection. Antivirus. Incident response.

Sumário

1 Introdução	5
2 Desenvolvimento	9
2.1 Comportamentos e padrões de ataques Ransomware	9
2.2 Amostras e subconjuntos de ransomwares utilizados em 2021	12
2.3 Configuração do ambiente de testagem	13
2.4 <i>Ransomware Ryuk</i>	14
2.4.1 Contextualização	14
2.4.2 Análise estática	15
2.4.3 Análise dinâmica	23
2.4.4 Debugging	38
2.4.5 Matriz de ataque da ameaça	53
2.5 Criando contramedidas	54
2.5.1 Efetividade de prevenção de execuções maliciosas	61
3 Considerações finais	65
REFERÊNCIAS	66

1 Introdução

A segurança cibernética se tornou nos últimos tempos um dos principais desafios enfrentados pelas empresas e pela sociedade. Com a crescente dependência da tecnologia, as ameaças cibernéticas se tornaram mais sofisticadas e frequentes, causando danos significativos a empresas e indivíduos. Ataques como roubo de dados pessoais, extorsão por meio de *ransomware*, espionagem industrial e sabotagem de infraestruturas críticas são apenas alguns exemplos destas ameaças que podem causar prejuízos financeiros e danos à reputação das empresas. Além disso, a sociedade como um todo também é afetada pelos ataques cibernéticos, que podem resultar em perda de privacidade, interrupção de serviços públicos e violações de direitos fundamentais.

Certamente, a maioria dos incidentes cibernéticos são causados por *malwares*, que podem ser distribuídos por meio de *phishing*, exploração de vulnerabilidades em sistemas e redes, downloads de arquivos infectados e outros métodos. Nesse contexto, *malwares* são definidos como softwares maliciosos com intuito principal de infiltração em dispositivos computacionais. Assim, estes se tornaram uma ameaça crescente durante os últimos anos devido a maior facilidade e obtenção de recursos computacionais que permitem conexão e exposição de endereços de maneira muito mais acessível que no passado. Outro fator catalisador para maior ocorrência destes ataques cibernéticos, decorre da maior aderência ao *home office*, que apesar de trazer muitos benefícios no tangente à qualidade de vida ao trabalhador e flexibilidade para empresa, acarretou maiores vulnerabilidades e exposições geradas por conta da maior quantidade de alvos agora, sem a proteção de ambientes totalmente encriptados, o que facilita a ação destes softwares.

Neste contexto, o subconjunto de *malwares* conhecidos como *ransomwares*, estes caracterizados por ações relacionadas a encriptação de dados da máquina infectada seguida por extorsão ao solicitar que a vítima pague uma quantia para ter seus dados resgatados e não vazados para domínio público, se tornaram os principais vilões para corporações governamentais e privadas, atingindo significativamente a credibilidade e confiabilidade destes, além de gerar imensos prejuízos financeiros e operacionais decorrentes do ataque. Em termos práticos, ataques cibernéticos famosos nesse escopo foram registrados por grandes corporações como Renner,

CVC, Porto Seguro, Atento e Serasa [1], e corroboram a latência alarmante desse tipo de cibercrime, visto que cerca de 55% das organizações brasileiras participantes da pesquisa “anual The State of *Ransomware* 2022”, realizada pela empresa de segurança Sophos, sofreram algum tipo de ataque cibernético envolvendo *ransomwares* [2] e que são reconhecidos mais de 1540 ataques por semana nos mais diversos segmentos [3].

Entretanto, esses tipos de ataques não são exclusividade dos dias atuais. A primeira ameaça envolvendo encriptação de dados seguido de extorsão ocorreu no fim da década de 80, e ficou conhecido como *Pc Cyborg Trojan* [4]. Tal ataque era decorrido de encriptação e ocultação de arquivos no disco da máquina, seguido de uma mensagem solicitando o pagamento de 189 dólares para a devolução dos arquivos. Com a evolução constante do escopo da informática, as formas modernas e mais conhecidas de *ransomware*, começaram a emergir nos anos 2000, com o surgimento do *GPCoder* [5], que fora o primeiro tipo de *ransomware* espalhado via e-mail. Em decorrência da maior disponibilidade de recursos computacionais, muitos hackers e entusiastas enxergavam na confecção desses tipos de softwares a oportunidade de aumentarem seus conhecimentos e habilidades, consolidando diversas variantes de *ransomwares*, como por exemplo o *Locker ransomware*, *Citadel* e *Lyposit* [5], que no começo da década passada atacaram diversas instituições financeiras e governamentais com intuito de vantagens financeiras. Dessa forma, diversas variantes e subconjuntos destes softwares surgiram e se disseminaram pelo globo cibernético, gerando prejuízos milionários à diversas companhias e instituições, consolidando o ataque de *ransomwares*, um dos maiores temores modernos, visto que na maioria dos casos, devido ao conceito de criptografia assimétrica e híbrida, a única de maneira de se obter os dados sequestrados de volta, é mediante ao pagamento de extorsão a atacantes, ou obtenção de backups (que nem sempre é possível).

Assim, prevenir ou impedir que tais ataques ocorram de maneira prévia se tornam uma necessidade para minimizar o impacto causado no escopo financeiro e operacional causada por tais ataques, conforme ilustrado na figura 1, de acordo com o levantamento feito pelo Ibraspd [6], os mais diversos setores como varejo, tecnologia, educação, indústria e instituições governamentais estão suscetíveis a ataques de *ransomware*, consolidando assim a necessidade de técnicas de prevenção e mitigação a esses eventos deletérios. O retorno financeiro obtido a partir dos

ataques cibernéticos se tornou tão significativo que se instituiu um modelo de negócios baseado na produção e distribuição de modelos e amostras de softwares maliciosos, conhecido como RaaS (*Ransomware as a Service*). Esse modelo de negócios promove o compartilhamento de modelos entre cibercriminosos, resultando em uma maior evolução dessas amostras e um aumento na quantidade e no impacto desses ataques em todo o mundo.

Figura 1 – Ataques de Ransomware divulgados na mídia em 2021

Contexto Atual – RANSOMWARE com repercussão na mídia*



Fonte: Ibraspd (2021) [6]

Nesse sentido, torna-se necessária a realização de análises técnicas que possibilitem um maior entendimento do funcionamento desses softwares, com foco em compreender seu comportamento, tecnologias envolvidas, algoritmos de criptografia utilizados assim como a construção de medidas protetivas se tornam necessárias para a proteção de dados sensíveis tal qual para gerar maior resiliência operacional para corporações que tem seus prosseguimentos totalmente atrelados a integridade de tais dados.

Dessa forma, é fundamental buscar o entendimento e compreensão do funcionamento desses softwares maliciosos, utilizando conceitos de engenharia reversa de software. Essa técnica consiste na análise das características intrínsecas ao arquivo malicioso, seja este um executável, biblioteca dinâmica linkável (DLL) ou outro tipo de arquivo binário (ELF), por meio de análises estáticas e dinâmicas.

Análises estáticas tem como objetivo o entendimento de características superficiais e aparentes do arquivo, como entendimento de quais API's do sistema operacional foram utilizadas ou requisitadas (em geral atreladas a DLL's do Windows

ou SO's do Unix), *strings* armazenadas na região de dados (*.data* e *.rodata*), verificação de arquivos comprimidos ou ofuscados (nomes e funções ocultados para dificultar análises) e modificação de assinaturas de execução correlatas a aplicativos confiáveis ao sistema operacional e seus sistemas de segurança, enganando varreduras de antivírus.

Por outro lado, análises dinâmicas primam pelo monitoramento e “*debugging*” (dissecção) do arquivo em tempo de execução, analisando chamadas de funções, *DLL*'s executadas, algoritmos de encriptação, conexão à servidores online e chamadas de sistema operacional que possam vir a auxiliar na continuação e persistência do ataque. Dessa forma, são executados em tempo real, softwares e mecanismos que rastreiam ou modificam a execução do *malware*, permitindo um entendimento mais detalhado de seu respectivo comportamento, tal qual a o exame de determinadas execuções e alterações que só se tornam identificáveis em tempo real, como por exemplo *buffers* que fazem utilização de alocação dinâmica de memória.

Doravante, a realização de análises documentais destes processos, tal qual a criação de ferramentas que auxiliem na identificação e mitigação destes softwares se torna uma prática necessária e utilitária para a sociedade que cada vez mais tem seu funcionamento atrelado a computadores e dispositivos que precisam ter condições íntegras e seguras para operação. Portanto, o principal objetivo desta iniciativa tecnológica é fornecer conhecimento, técnicas e ferramentas de forma pública para a sociedade, visando melhorar a divulgação de uma área ainda muito restrita às empresas de segurança da informação, bem como munir e incentivar interessados na área de análise de *malware*, resposta a incidentes e engenharia reversa de software. Com isso, busca-se melhorar diversos segmentos desse escopo e, consequentemente, beneficiar a sociedade como um todo.

2 Desenvolvimento

2.1 Comportamentos e padrões de ataques *Ransomware*

Por terem como característica principal a encriptação de arquivos, *ransomwares* são categorizados em dois tipos de grupos [5]. O primeiro e mais comum, é denominado como *Crypto*, que tem sua funcionalidade baseada na encriptação de arquivos e dados no desktop ou disco, não influenciando no uso de funções do computador, mas impedindo acesso aos arquivos atingidos mediante à extorsão. O segundo é denominado *Locker*, onde bloqueia o computador ou dispositivo mobile impedindo o usuário de fazer uso deste, não realizando diretamente encriptação de arquivos na sessão de armazenamento deste.

Em geral, os tipos mais encontrados e letais atualmente se consolidam como *Crypto ransomware*, tanto pela maior dificuldade de recuperação dos arquivos, tal qual pela maior facilidade de extorsão aos alvos. Destarte, um ciclo de *Crypto ransomware* portável para plataforma Windows segue um padrão de operações comuns que elucidam o ataque [8].

Figura 2 –Estágios comuns de ataque de Crypto Ransomwares. Fonte: Elaborado pelo autor



Fonte: Elaborado pelo autor

Destaca-se no ciclo apresentado na figura 2, muitas outras ações intermediárias não foram elucidadas, sejam pelas particularidades de cada amostra ao interagir com o alvo infectado, seja pela finalidade da ação ser redundante com outra descrita no diagrama. Destaca-se também que nem todos os softwares maliciosos disponíveis na rede seguem de maneira procedural todos os passos descritos, novamente dependendo da característica intrínseca do software analisado e seus respectivos fins.

É importante, em prosseguimento, entender-se como em termos gerais cada uma dessas fases se apresenta em termos de comportamento e interação com a

máquina infectada. O primeiro estágio, no qual ocorre a fase de contaminação, ocorre pela interação da vítima com algum fator de entrada para a amostra maligna. Mais de 59% dos casos entre 2015 e 2016, vieram de links ou arquivos disponibilizados via e-mail, tanto por meio de *phishing* ou *spam*. Cerca de 24% das infecções vieram por meio de contato com web aplicações ou sites, 8% de redes sociais, dispositivos USB ou aplicações desktop e 9% de origens desconhecidas [5]. Nesse concernente, nota-se que aplicações web utilizando-se de protocolos e-mail (SMTP) ou HTML, se protagonizam como a principal forma de transmissão de *ransomwares*, o que eleva a preocupações com a mitigação de links e URLs maliciosas, tal qual como um maior cuidado para interação com links externos.

Após o *ransomware* infectar a máquina da vítima, inicia-se, em alguns casos, a obtenção da chave de encriptação, segundo estágio, que consiste em uma sequência de caracteres aleatória que irá pautar o padrão de criptografia utilizada, geralmente conhecido como função de *hash*. Essa chave é essencial para continuar a encriptação em progresso para manutenção de um padrão utilizado, e, portanto, é estritamente necessária para desencriptar esses arquivos se assim necessários [9]. Essa chave pode ser gerada em um ambiente externo como um servidor ou o próprio servidor do atacante e enviado ao computador infectado via conexão socket com a web em uma faixa de IP (Protocolo de Internet), consolidando o conceito de servidor de *Comando e Controle* [7] (normalmente referenciado como C&C em literaturas sobre cibersegurança). Esse contato com o servidor C&C pode promover o envio da chave de um ambiente externo ao do computador infectado, diminuindo as chances de recuperação dos arquivos sem a extorsão a ser realizada. Nota-se também que esse contato com servidor externo pode ocorrer em outros estágios, seja para obtenção de informações, tal qual para envio de dados se assim necessário.

Com a chave de encriptação portável no dispositivo, inicia-se o terceiro estágio de infecção, conhecido pela encriptação dos dados e arquivos no disco da máquina (caso se trate de um *Crypto Ransomware*). Esta etapa consiste na enumeração de arquivos alvos e modificação de seus conteúdos com intuito mor de inutilizá-los e torná-los meros *placeholders* para os arquivos atacados. Nota-se que a modificação desses arquivos gera uma diferenciação não somente em seu conteúdo e extensão que são modificados, mas também em seu provável tamanho, influenciando no conceito de entropia de um arquivo (nível de aleatoriedade deste). Em geral, os arquivos modificados são modificados sequencialmente em mesmos diretórios ou em

processos/threads diferentes para otimizar o trabalho. Em alguns tipos de *ransomwares*, antes de um arquivo ser efetivamente encriptado, este pode ser copiado para um diretório criado pelo software malicioso com objetivo de “sequestrar” o arquivo, enviando este em um estágio posterior para um servidor de posse do atacante, dando ao cibercriminoso a posse de dados íntegros e a ameaça de expô-los na internet caso não seja realizado o pagamento solicitado.

Após a encriptação, boa parte dos subgrupos de *ransomwares* implementam técnicas de persistência [7], que findam por achar maneiras de manter as ações maliciosas realizadas na máquina atuantes até que seja realizado o pagamento solicitado. Dessa forma, o software invasor tende por, em plataformas Windows, alterar registros de valores do Sistema operacional. Um registro de valor consiste em um banco de dados central que gerencia configurações e recursos como drivers, interfaces, seguranças e logs são armazenados nessa central [10], e logo, realizando modificações que impeçam restauração de estados anteriores ou que apaguem determinados rastros em geral registrados intrinsecamente pelo sistema operacional.

Doravante, o software prima pela criação do chamado *Ransom Note*, que pode variar de um arquivo de texto (extensão *.txt*) disposto no desktop ou porção de memória *C://* do Windows; até uma mudança no próprio Wallpaper do computador. Nessa mensagem, o atacante informa à vítima sobre o ataque, meios de contato para obtenção dos arquivos e meios para realização do pagamento, geralmente gateways de *blockchain* como *Bitcoin* ou *Ethereum*, visto que são mais difíceis de rastreio por conta da descentralização adotadas pela tecnologia em questão.

Outra etapa geralmente seguida por essas ameaças consiste na limpeza de rastros. Esse procedimento pode ser mesclado ao estágio quatro em alguns tipos de ataque, mas em geral tendem por limpar, excluir e modificar arquivos que possam fornecer qualquer tipo de identificação ou informações do software malicioso que não as informações dispostas no *Ransom Note*. Em geral são excluídos quaisquer tipos de logs gerados por softwares de monitoramento, ou modificações externas aos arquivos encriptados.

Por fim, o *ransomware* em certos casos contata o servidor C&C novamente para obtenção e envio de dados se assim necessário. Em geral, o software invasor envia ao servidor informações como o IP público da máquina alvo, características de softwares e arquivos atingidos, arquivos íntegros para possibilidade de exposição pública e afins [7]. Dessa forma, o software malicioso pode receber ainda

determinadas e pequenas ações que possam ser executadas por contato posterior com o servidor, como uma nova chave pública, por exemplo.

2.2 Amostras e subconjuntos de *ransomwares* utilizados em 2021

Segundo um relatório divulgado em agosto de 2022 pela empresa norte-americana *Fortinet* [11], o Brasil sofreu cerca de 31.5 bilhões de ataques cibernéticos no primeiro semestre de 2022, representando um aumento de 94% em relação ao ano anterior. Tais dados solidificam a ameaça do que se denomina como *Ransom as a Service* (RaaS), visto que grande parte das amostras listadas pelo relatório advém de “famílias de *Ransomwares*”, ou foram feitos com base em outras amostras já conhecidas do cibercrime, o que elucida um crescente aumento na utilização e criação destas amostras para ataques em território brasileiro.

Dentre as famílias de amostras mostradas pela *Fortinet* [11], foram priorizadas para análise estática e dinâmica amostras de arquivos com assinaturas *Ryuk* por essencialmente se caracterizarem como *Crypto Ransomware* e estarem em utilização durante todo ano de 2021, e mantidos em utilização para 2022. Todas os arquivos e amostras analisados foram obtidas no site *Malware Bazaar*, que tem como proposição a disponibilização de amostras encontradas por pesquisadores de segurança e analistas de *malware* para estudos e criação de ferramentas de mitigação e propagar entendimento para prevenção de possíveis incidentes.

Figura 3 –Ataques notificados por amostra na América Latina em 2021-22

ReVIL	THANOS	LOCKBIT	RYUK
3,179	1,430	1,996	648
LOCKY	CONTI	BlackMatter	HIVE
1,049	778	1,142	1,813

Fonte: Bnamicas (2022) [11]

Desse modo, entender e disponibilizar informações acerca do funcionamento destes softwares maliciosos, se torna uma prática necessária para construção de *patches* de correção em vulnerabilidades em sistemas operacionais ou sistemas de detecção, como antivírus; tal qual permite que usuários se tornem capazes de

identificar sinais de ataques em progresso podendo utilizar ferramentas ou métodos de mitigação com maior velocidade, podendo em alguns casos evitar maiores danos.

2.3 Configuração do ambiente de testagem

Para realizar a análise dos arquivos base, deve-se inicialmente, configurar um ambiente de testagem que seja seguro para execução do *malware* em geral sendo definido como *Sandbox*, permitindo a visualização completa de sua atuação e processos decorrentes, sem alterar o ambiente hospedeiro ou afetar a rede local.

Dessa maneira, utiliza-se uma máquina virtual com sistema operacional Windows, conforme estabelece-se o escopo da análise. Recomenda-se que o sistema hospedeiro seja outro que não o virtualizado, como algum baseado em distribuições Unix, para que em eventual disseminação involuntária da amostra maliciosa para o sistema hospedeiro não incorra em encriptação severa dos dados, visto que na maioria dos casos, uma amostra é específica para um tipo de sistema operacional, principalmente para simplificar desafios correlatos a portabilidade do arquivo para diferentes sistemas.

Destarte, utiliza-se uma imagem ISO de Windows 10 (licença gratuita disponibilizada no site da Microsoft), para configuração de uma máquina virtual no software Virtual Box. Este último foi escolhido pela facilidade de manejo entre *snapshots* (uma cópia de um volume ligado a uma instância computacional), um passo essencial para se recuperar estágios anteriores à execução da amostra malicioso, permitindo a reexecução de análises de forma mais prática e eficaz.

Para a análise foram alocados 60GB de armazenamento dinâmico de disco e 2GB de memória RAM, permitindo uma performance eficaz dos softwares necessários para análise estática e dinâmica das amostras escolhidas, tal qual a eficiência de IDE's para geração de código personalizado se assim necessário. Recomenda-se, também, a desativação do antivírus padrão do sistema operacional para não interferência e possível impedimento das análises a serem realizadas. Nota-se também a importância de se utilizar versões atualizadas dos softwares de virtualização, tal qual a tomada de procedimentos que garantam a segurança do sistema hospedeiro como a rede em questão, como:

- Ao se executar o software malicioso, desconecta-se a placa de rede ou configura-se está como *host-only* para conexões privadas com outra instância, sem contato com as redes públicas;
- Desativa-se sistemas de compartilhamento de pastas ou configura-se estas como *read-only* para o sistema virtualizado, dificultando possíveis invasões do *ransomware*;
- Mantenha-se snapshots regulares de cada procedimento analisado para criação de pontos de segurança para retorno em caso de testes malsucedidos ou inadequados.

Dentro do sistema virtualizado, serão utilizados os seguintes softwares, cujo objetivo primário será abordado de acordo com sua utilização posterior:

- API Monitor V2;
- Detect It Easy;
- Floss;
- Hex Workshop;
- IDA Freeware 7.7;
- Pestudio;
- X32dbg;
- X64dbg;
- Yara64;

2.4 Ransomware Ryuk

2.4.1 Contextualização

A variante *Ryuk* é um software malicioso com operação atribuída a cibercriminosos russos, o grupo WIZARD SPIDER, atacando setores governamentais, de manufatura, saúde e tecnologia [12]. Segundo estimativa da Trend Micro [12], estima-se que o grupo tenha faturado cerca de 150 milhões de dólares em 2020.

Segundo a empresa de segurança norte americana, Trend Micro, o *Ryuk* utiliza uma tática de *Download as a service* como vetor de ataque, sendo distribuído por outras famílias de *malware*, e também podendo infectar vítimas com outros softwares maliciosos. Todavia, aponta-se *phishing* como maior causa de infecção pela amostra,

conforme relata a *AdvIntel* [12], 91% dos ataques têm seu início em links ou e-mails suspeitos.

Outro fator preocupante relacionado a ameaça do Ryuk é sua notoriedade como RaaS, visto que é frequentemente oferecido para outros atacantes, conseguindo obter porcentagens de ataques realizados com sucesso, incorrendo em uma utilização maior do que a operação gerenciada somente pelos seus desenvolvedores originais.

A maneira de contágio da amostra ocorre principalmente via e-mails de spam, advindos sobretudo de endereços de e-mails alterados, com objetivo de enganar a vítima via engenharia social. O contágio é iniciado, em geral, quando usuários abrem documentos do pacote Office em anexos dos e-mails fraudados. Quando aberto, o documento pode executar em backdoor um comando PowerShell que tende a baixar os Trojan TrickBot e Emotet que gerenciam o download do Ryuk e de outras possíveis ameaças. Além disso, caso haja vulnerabilidades já encontradas no alvo, atacantes podem se conectar via RDP, baixando e executando o Ryuk diretamente no host [42].

2.4.2 Análise estática

Após o download da amostra Ryuk via *Malware Bazaar* na máquina virtual, inicia-se a análise estática do arquivo. Uma análise estática consiste em um conjunto de procedimentos de dissecação do arquivo sem realizar execução parcial ou total deste, sendo assim uma análise inicial para extração de informações básicas acerca do binário analisado [13]. Esse tipo de procedimento permite, na maioria dos casos, a classificação do tipo de ameaça, API's utilizadas, assinaturas e certificados utilizados para camuflar atividade, *strings* contidas na região de memória que possam fornecer indícios do comportamento da ameaça e metadados do arquivo que fornecem informações correlatas a sua entropia (aleatoriedade de bytes que permite concluir se um arquivo é considerado *packed*, um passo necessário em análises de *Malware*, visto que estes softwares em geral sofrem este tipo de processo), ponto de entrada na memória (endereço da sessão de código/texto), compilador utilizado (Visual Studio, GCC ou Delph), tipo do arquivo (executável ou DLL) e assinaturas *hash* do arquivo (essas sendo importantes para verificação se a ameaça analisada tem registros por empresas de segurança, o que pode fornecer análises paralelas que melhorem a análise realizada).

Dessa forma, utiliza-se três softwares para análise estática da amostra *Ryuk*. O primeiro foi utilizando o editor hexadecimal *Hex Workshop*, que permite o exame manual byte a byte do executável, permitindo assim a identificação de assinaturas ou comportamentos que possam concluir suspeitas acerca da amostra. Ao utilizar o software para leitura da amostra temos os primeiros bytes, e sua respectiva transcrição para ASCII, que nos permitem dados relevantes sobre a amostra.

Figura 4 – Primeiros 240 bytes da amostra e sua respectiva transcrição ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	0123456789ABCDEF01234567
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	B8	00	00	00	00	00	00	MZ
00000018	40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	F8	00	00	00	00	00	00	00	00	00
00000048	21	B8	01	4C	CD	21	54	68	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	!..L.!This program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	6D	6F	64	65	2E	0D	0D	0A	t be run in DOS mode...
00000078	24	00	00	00	00	00	00	00	35	6B	51	5E	71	0A	3F	0D	71	0A	3F	0D	71	0A	3F	0D	\$.....5kQ^q.?.q.?.q.?
00000090	C5	96	CE	0D	78	0A	3F	0D	C5	96	CC	0D	08	0A	3F	0D	C5	96	CD	0D	69	0A	3F	0Dx.?.....?....i.?
000000A8	12	57	3C	0C	60	0A	3F	0D	12	57	3A	0C	6F	0A	3F	0D	12	57	3B	0C	60	0A	3F	0D	W<.`?.W;.o.?.W;.`?.
000000C0	78	72	AC	0D	76	0A	3F	0D	71	0A	3E	0D	72	0A	3F	0D	1F	57	36	0C	73	0A	3F	0D	xr..v.?..q.>.+?.W6.s.?
000000D8	1F	57	3D	0C	70	0A	3F	0D	52	69	63	68	71	0A	3F	0D	00	00	00	00	00	00	00	00	.W=.p.?..Richq.?.....
000000F0	00	00	00	00	00	00	00	00	50	45	00	00	4C	01	05	00	EB	5A	77	5B	00	00	00	00PE..L....Zw[...]

Fonte: Elaborado pelo autor

Conforme averígua-se na *figura 4*, nota-se a presença dos dois primeiros bytes **0x4D 0x5A**, que transcritos para ASCII significam, respectivamente os caracteres **MZ**, que ilustram o início de todo arquivo PE, o *stub* do MS-DOS utilizado. Já na posição **0xF8**, inicia-se um conjunto de 4 bytes, **{0x50 0x45 0x00 0x00}**, que se constituem como a própria assinatura de arquivo de PE, o que permite assim sua execução em ambientes Windows, e caso usuários tentem rodar o programa em antigos sistemas *MS-DOS*, irá ser mostrada a mensagem iniciada no byte **0x4E**, que afirma que o programa não pode ser executado nestes sistemas. Esse tipo de análise manual é interessante para identificação do tipo de arquivo consolidado pela ameaça e seu ambiente de execução padrão como ambientes Windows, Linux, Mac ou Android.

Em seguida, para obtenção de dados mais elaborados da amostra, utiliza-se o *Pestudio 9.37*, para obtenção de informações e metadados mais detalhados da amostra, como sua assinatura *hash*, compilador utilizado, sistema de compilação e funções ou API's ilustradas no arquivo antes de sua execução.

Figura 5 – Metadados da amostra disponibilizados via Pestudio 9.37

property	value
md5	5AC0E050F93F36E69025FAEA1FBBA450
sha1	070974FD4E6E7C4DAD96FEDBD7D903296CA90571
sha256	238A0A94FFBFC0A6735FE7E7E799880A8F32CE1D55EC49A13A3E853120B2
first-bytes-hex	4D 5A 90 00 03 00 00 04 00 00 FF FF 00 B8 00 00 00 00 00 40 00 00 00 00 00 00 00 00
first-bytes-text	M Z
file-size	393216 bytes
entropy	6.249
imphash	DC373C013370FA1B013773F3BF6F1
signature	Microsoft Visual C++
tooling	Visual Studio 2013
entry-point	E8 0D 02 00 00 E9 80 FE FF F5 BB EC A1 18 20 41 00 E3 E0 1F 6A 20 39 2B C8 8B 45 08 D3 C8 33 05
file-relocation	n/a
description	
file-type	32-bit
cpu	GUI
subsystem	0x5B775AE8 (Fri Aug 17 23:31:55 2018 UTC)
compiler-stamp	0x5B775AE8 (Fri Aug 17 23:31:55 2018 UTC)
debugger-stamp	n/a
resources-stamp	n/a
import-stamp	0x00000000 (Thu Jan 01 00:00:1970 UTC)
export-stamp	n/a
executable	

Fonte: Elaborado pelo autor

Conforme visualiza-se da *figura 5*, a amostra apresenta como impressão *hash sha256* o valor:

- **23F8AA94FFB3C08A62735FE7FEE5799880A8F322CE1D55EC49A13A3F8
5312DB2**

Esta amostra pode ser encontrada em outros repositórios ou sites de análise como *Virus Total*, um fato condizente, dado que esta derivou de um repositório online e provavelmente, fora fruto de análise de outros analistas. Na imagem, é possível constatar-se que a assinatura do arquivo é *Microsoft Visual C++*, um possível indício de seu compilador (Visual Studio 2013) e sua linguagem fonte (C++), sendo compilado em meados de agosto de 2018, fato que ilustra a utilização desta amostra três anos após seu surgimento, e o possível surgimento de outras amostras. Nota-se, conforme constatado anteriormente, o fato de o arquivo ser executável, e com arquitetura 32-bit, um pouco mais simplória que a de 64-bit, adotada pela maioria de softwares atualmente, mas uma informação de suma importância para análises posteriores principalmente correlatas a execução da amostra.

Posteriormente, analisa-se dentro do campo de cabeçalhos do arquivo (*dos-stub*), a sessão de *optional-header*, que apesar do que o nome indica é opcional apenas para arquivos objetos, mas obrigatória para executáveis, como a amostra em questão. Nela existem informações importantes acerca da amostra como tipo do executável, endereço de entrada, *imageBase* (executável quando vai à memória), diretório de dados e *import table* (tabela que resolve as bibliotecas e DLL's importadas pelo programa quando em execução).

Figura 6 – Optional Header da amostra *Ryuk*

property	value	detail
characteristics	0x140	0x0000000000000000
address-space-layout-randomization (ASLR)	0x0040	true
data-execution-prevention (DEP)	0x0100	false
code-integrity	0x0000	false
structured-exection-handling (SEH)	0x0000	false
windows-driver-model (WDM)	0x0000	false
terminal-server-aware (TSA)	0x0000	true
control-flow-guard (CFG)	0x0000	false
image-bound	0x0000	false
image-solution	0x0000	false
High-Entropy	0x0000	false
AppContainer	0x0000	false
general		
subsystem	0x0002	GUI
memsize	0x0100	0x0000000000000000
file-checksum	0x0000000000000000	0x0000000000000000
entry-point	0x00001738	0x00001738 section:text
base-of-code	0x00001000	section:text
base-of-data	0x00000000	section:data
size-of-code	0x00000010	4544 bytes
size-of-data	0x00000000	30000 bytes
size-of-uninitialised-data	0x00000000	0 bytes
size-of-uninitialised-data	0x00000000	405504 bytes
size-of-image	0x00000000	1024 bytes
size-of-headers	0x00000000	1048576 bytes
size-of-stack-reserve	0x00000000	1048576 bytes
size-of-stack-commit	0x00000000	4096 bytes
size-of-heap-reserve	0x00000000	1048576 bytes
size-of-heap-commit	0x00000000	4096 bytes
section-alignment	0x00000000	4096 bytes
file-alignment	0x00000000	512 bytes
directories-number	0x00000010	16
LoaderFlags	0x00000000	0x00000000
WindowsVersionValue	0x00000000	0x00000000
Image-Type	0x00400000	0x00400000
linker-version	14.0	14.0
os-version	5.1	Windows XP
image-version	0.0	0.0
subsystem-version	5.1	5.1

Fonte: Elaborado pelo autor

Dentre as informações contidas na struct *Optional Header* [14], algumas são essenciais para informações correlatas a análise de amostras maliciosas, como por exemplo a verificação de que o arquivo analisado possui o campo *address-space-layout-randomization* (ASLR) ativado. Esse campo surgira com objetivo primário de ser um mecanismo de segurança principalmente contra *exploits* de *buffer overflow*, visto que uma vez que o atacante conhecia endereços fixos de memória e seus respectivos offsets, era possível criar scripts que exploravam tais vulnerabilidades assim que o processo fosse aderido na memória. Com a implementação de ASLR, ocorre-se uma randomização a localização de endereços que são carregados na memória, impossibilitando a localização de pontos de entrada ou endereços específicos [15]. No contexto de análise de *malware*, este artifício é utilizado por muitos desenvolvedores maliciosos para dificultar o trabalho de analistas em dissecar a amostra, dificultando a procura de endereços de execução e até mesmo blocos específicos a serem examinados em análises dinâmicas e debug. Visualiza-se também a versão do sistema operacional em que se realizou a compilação do arquivo como *Windows XP*. Em geral, essa distribuição Windows é mais utilizada por desenvolvedor maliciosos por possuírem maiores vulnerabilidades e menor chance de rastreio acerca do sistema de origem.

Em seguida analisa-se as sessões do arquivo, como suas permissões, tamanhos e endereços virtuais.

Figura 7 – Características das sessões do arquivo malicioso

property	value	value	value	value	value
name	.text	.rdata	.data	.glibs	.reloc
md5	1A4FE3E34DCE7601BC763D...	60C19940D6E79800395956FA...	6A2501AE7AA166A10E40DC...	7A6634CF1ABB9C1BBBF0C...	957D2B700E31B00F9B89EDB...
entropy	6.614	4.891	6.202	1.461	6.161
file-ratio (99,74%)	11.33 %	5.86 %	81.38 %	0.13 %	1.04 %
raw-address	0x00000400	0x00000B200	0x00010C00	0x0005EE00	0x0005F000
raw-size (392192 bytes)	0x0000A0E00 (44544 bytes)	0x00005A00 (23040 bytes)	0x0004E200 (320000 bytes)	0x00000200 (512 bytes)	0x00001000 (4096 bytes)
virtual-address	0x00001000	0x0000C000	0x00012000	0x00061000	0x00062000
virtual-size (93397 bytes)	0x0000AC7 (44231 bytes)	0x0000594A (22858 bytes)	0x0004EBDC (322524 bytes)	0x000000B4 (180 bytes)	0x00000E14 (3604 bytes)
entry-point	0x00000173B	-	-	-	-
characteristics	0x60000020	0x40000040	0xC0000040	0x40000040	0x42000040
writable	-	-	x	-	-
executable	x	-	-	-	-
shareable	-	-	-	-	-
discardable	-	-	-	-	x
initialized-data	-	x	x	x	x
uninitialized-data	-	-	-	-	-
unreadable	-	-	-	-	-
self-modifying	-	-	-	-	-
virtualized	-	-	-	-	-

Fonte: Elaborado pelo autor

É notório na figura 7, que mais de 80% dos bytes do executável estejam alocados na sessão *.data*, onde se armazenam valores inicializados previamente e que podem ser utilizados durante a execução como *buffers de payload* (códigos executáveis imersos dentro de arquivos), *strings* contendo informações como nomes de diretórios, caminhos do sistema operacional, extensões ou processos, e variáveis

e constantes correlatas a enumeração de arquivos. Ademais, nota-se que esta sessão permite ser escrita (*writable*), o que pode significar a modificação de dados durante execução e dificultando o mapeamento destes durante análises estáticas.

Posteriormente, constata-se que o arquivo realiza a importação de três *DLL*'s do Windows:

- *Kernel32*, um dos módulos do kernel do Windows utilizado principalmente para gerenciamento de memória e operações de E/S.
- *Shell32*, é um *DLL* do Windows que permite acesso ao console do sistema operacional, levando a escalação de privilégios ou execuções de processos.
- *Advapi32*, é a biblioteca desenvolvida pela própria Microsoft para fornecer procedimentos e funções de programas padrões no sistema operacional.

Desse modo, das 70 funções importadas pelo executável, 11 chamam a atenção por sua interação com o sistema operacional e sua capacidade de modificação, execução ou mapeamento de arquivos e processos, conforme ilustrado na figura 8.

Figura 8 – Funções elucidadas como potenciais perigos na execução da amostra

functions (70)	flag (11)	ordinal (0)	library (3)
WriteFile	x	-	kernel32.dll
RaiseException	x	-	kernel32.dll
GetCurrentProcessId	x	-	kernel32.dll
GetCurrentThreadId	x	-	kernel32.dll
TerminateProcess	x	-	kernel32.dll
GetModuleHandleExW	x	-	kernel32.dll
FindFirstFileExA	x	-	kernel32.dll
FindNextFileA	x	-	kernel32.dll
GetEnvironmentStringsW	x	-	kernel32.dll
ShellExecuteW	x	-	shell32.dll
SystemFunction036	x	-	advapi32.dll

Fonte: Elaborado pelo autor

Dentre as funções ilustradas acima, vê-se que estas se subdividem nos seguintes grupos de atuação:

- Enumeração e modificação de arquivos (*WriteFile*, *FindFirstFileExA*, *FindNextFileA*);
- Enumeração e gerenciamento de processos (*GetCurrentProcessId*, *GetCurrentThreadId*, *TerminateProcess*, *GetModuleHandleExW*);
- Enumeração de strings no ambiente executado (*GetEnvironmentStringsW*);
- Execução de comandos no sistema (*ShellExecuteW*);
- Geração de números e sementes (seeds) pseudoaleatórias (*SystemFunction036*, definida na documentação da Microsoft como *RtlGenRandom* [16]).

Apesar da geração de números pseudoaleatórios, visualiza-se que pelas *DLL*'s importadas, nenhuma é correlata a funções de criptografia ou geração de chaves criptográficas que são em geral utilizadas em *Ransomwares* [13]. Desse modo, duas hipóteses se mostram possíveis, a primeira correlata ao fato de que essas funções são executadas apenas em memória por execução dinâmica, provavelmente advinda de *payloads* ou por execuções internas, e a segunda de que o arquivo analisado seja classificado como *Dropper*, isto é, seja apenas um arquivo intermediário que irá gerar o executável principal em sua execução. A segunda abordagem se torna mais plausível visto que, na atualidade, este tipo de procedimento se tornou muito eficiente para cibercriminosos, visto que disseminar arquivos *Droppers* pela internet é menos suspeito que *ransomwares* propriamente, e permitem também menor possibilidade de mitigação, visto que é mais difícil detectar-se a amostra devido à ausência de indicadores fortes da presença de softwares maliciosos de criptografia. De todo modo, somente será possível averiguar tais hipóteses em posterior análise dinâmica do executável.

Buscando dados acerca da compactação do arquivo, utiliza-se o software Detect It Easy 2.05, no qual se averíga a entropia relativa do arquivo, isto é, a aleatoriedade dos bytes dispostos no executável, podendo facilitar a análise dinâmica do arquivo em momento posterior, visto que em arquivos com alta entropia, geralmente significam que este está compactado ou criptografado, dificultando sua análise e *debugging* [17]. Após abertura da amostra no software supracitado, este gera uma curva de *byte analisado x grau de entropia associado*, sendo essa escala entre (1 e 8). Além disso, provém a entropia total do arquivo e porcentagem de compactação do arquivo analisado, permitindo assim a conclusão se a amostra em questão é considerada compactada.

Figura 9 – Análise de entropia para a amostra Ryuk



Fonte: Elaborado pelo autor

Conforme consta a *figura 9*, o arquivo não possui indícios de compactação total, tal qual ofuscação de seus conteúdos. Todavia, esses fatos podem não se confirmar conforme hipótese anterior, devido a suspeita da amostra em questão ser possivelmente um *Dropper* e não o *Ransomware* propriamente.

Analogamente, conforme constatado nas análises anteriores, inspeciona-se as *strings* alocadas estaticamente no arquivo, visto que podem revelar detalhes do comportamento do software, seja pela evocação de bibliotecas ou funções importadas em execuções, dificultando o mapeamento destas em análises estáticas, seja para encontrar diretórios, domínios e afins, que podem revelar informações importantes sobre o ataque e possíveis impactos. Para tal, com apoio do software de linha de comando *Floss*, mapeia-se todas as *strings* armazenadas nas sessões de dados (*.data* e *.rodata*) e as disponibiliza para análises. No arquivo foram detectadas cerca de 5128 *buffers de chars*, sendo duas destas *strings* decodificadas. Dentre as mais de cinco mil, algumas mostram informações úteis, como funções de registro, alocação de memória, gerenciamento de sessão crítica, execução de Shell e ajuste de privilégios:

- *EventRegister*
- *EventSetInformation*
- *EventUnregister*
- *EventWriteTransfer*
- *FlsAlloc*
- *FlsFree*
- *FlsGetValue*
- *FlsSetValue*
- *InitializeCriticalSectionEx*
- *GetCurrentPackageId*
- *LCMapStringEx*
- *LocaleNameToLCID*
- *ShellExecuteW*
- *OpenThreadToken*
- *OpenProcessToken*
- *ImpersonateSelf*
- *AdjustTokenPrivileges*

- *LookupPrivilegeValueW*

Nota-se que algumas destas já constavam na tabela de *imports* cabeçalho opcional do DOS-STUB, sendo provavelmente correlatos a estas alocações. Outras, como *AdjustTokenPrivileges* e *LookupPrivilegeValueW*, não constavam na tabela de importações, e se mostram eficazes para ação do arquivo em sua execução, principalmente pelos ajustes necessários para posterior encriptação e persistência.

Foram notadas muitas linhas de comando relativas a desabilitação de serviços importantes, como antivírus famosos, serviços de backup e base de dados, visando principalmente não dar alternativas de mitigação à vítima.

- *stop "Sophos System Protection Service" /y*
- *stop "Sophos Web Control Service" /y*
- *stop "SQLsafe Backup Service" /y*
- *stop "SQLsafe Filter Service" /y*
- *stop "Symantec System Recovery" /y*
- *stop "Veeam Backup Catalog Data Service" /y*
- *stop AcronisAgent /y*
- *stop AcrSch2Svc /y*
- *stop Antivirus /y*
- *stop ARSM /y*
- *stop BackupExecAgentAccelerator /y*
- *stop BackupExecAgentBrowser /y*
- *stop BackupExecDeviceMediaService /y*
- *stop BackupExecJobEngine /y*

Analogamente, Ryuk tem comandos de deleção silenciosa de todas as cópias de volumes (Shadow copies) localizadas no host local (drivers de memória) ou em servidores de arquivos conectados, além de excluir arquivos padrões de backup, possivelmente com intuito de dificultar a recuperação de arquivos do alvo, facilitando a ação de estelionato.

- *vssadmin Delete Shadows /all /quiet*
- *vssadmin resize shadowstorage /for=c: /on=c: /maxsize=401MB*
- *vssadmin resize shadowstorage /for=c: /on=c: /maxsize=unbounded*
- *vssadmin resize shadowstorage /for=d: /on=d: /maxsize=401MB*
- *vssadmin resize shadowstorage /for=d: /on=d: /maxsize=unbounded*

- `del /s /f /q c:*.VHD c:*.bac c:*.bak c:*.wbcat c:*.bkf c:\Backup*.* c:\backup*.* c:*.set c:*.win c:*.dsk`
- `del /s /f /q d:*.VHD d:*.bac d:*.bak d:*.wbcat d:*.bkf d:\Backup*.* d:\backup*.* d:*.set d:*.win d:*.dsk`

Em seguida, nota-se que o arquivo procura por diretórios específicos, que provavelmente devem ser estratégicos para procura de arquivos ou utilização de serviços para a atividade maliciosa. Alguns destes são listados a seguir:

- `\Documents and Settings\Default User\`
- `\users\Public\`
- `\Documents and Settings\Default User\finish`
- `\users\Public\finish`
- `\Documents and Settings\Default User\sys`
- `\users\Public\sys`
- `\System32\cmd.exe`
- `\Documents and Settings\Default User\`
- `\users\Public\`
- `C:\Users\flare\program.exe`

Por fim, a mais interessante *string* localizada no arquivo, trata-se de um comando de persistência utilizado para mudar o arquivo de registro de valores do Windows, permitindo que o software resista a reboots e tentativas de remoção advindas do sistema:

• <code>/C</code>	<code>REG</code>	<code>ADD</code>
<code>"HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /v "svchos" /t REG_SZ /d " " /f.</code>		

Tal comando, provavelmente executado via função `ShellExecuteW` é uma maneira efetiva do atacante conseguir evitar a mitigação do ataque via *reboot* instantâneo do sistema devido a persistência gerada.

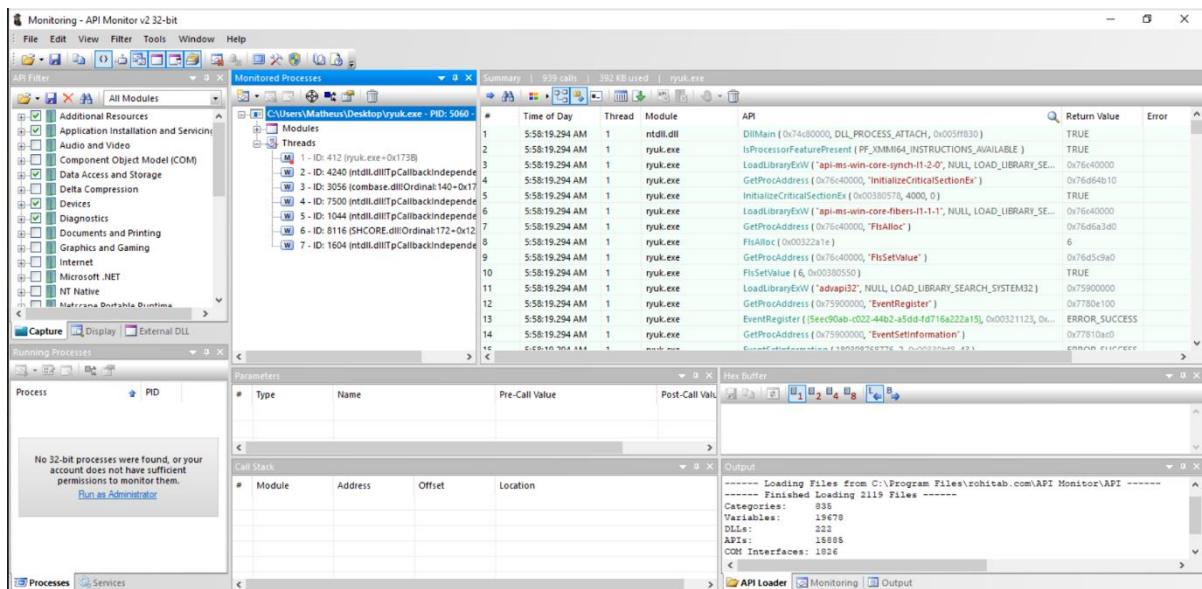
2.4.3 Análise dinâmica

Posterior a análise estática do arquivo, é necessário a execução experimental e segura do arquivo malicioso para se entender na prática o comportamento do ataque com objetivo de amplificar ainda mais o escopo da análise. Dessa forma, para a

execução prática do *Ransomware*, desconecta-se a placa de rede da máquina virtual para evitar conexão deste com a C&C e enumeração de hosts na mesma rede, tal qual utiliza-se o software de monitoramento *Api Monitor V2*, que permite em tempo real o monitoramento de interações do programa investigado com o sistema operacional, principalmente com chamadas de API's e funções do Windows, que aqui serão essenciais para entender-se o comportamento do arquivo durante a atividade de encriptação. Desse modo, nota-se que a detecção e análise de movimentos da ameaça que deveriam ocorrer utilizando a rede da *virtual machine*, provavelmente será superficial e não detectada, todavia, trata-se de uma maneira de garantir a segurança do sistema hospedado, tal qual informações das redes locais utilizadas.

Para monitoramento e rastreamento das ações tomadas pelo arquivo, marca-se no *Api Monitor V2* todos os filtros de API's evocadas que são correlacionadas aos módulos importados pelo arquivo (Kernel32, Shell32 e Advapi32). Com os filtros de rastreamento selecionados, escolhe-se o executável a se monitorar (*ryuk.exe*) e inicia-se a execução do arquivo, conforme exemplifica-se na figura 10.

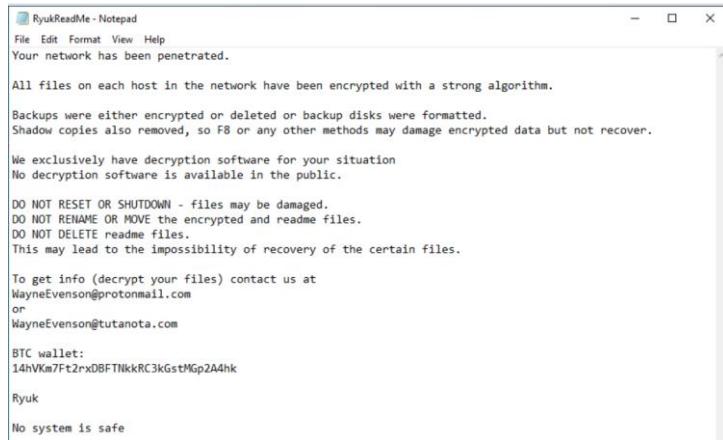
Figura 10 – Monitor Api V2 rastreando ações do executável *ryuk.exe*



Fonte: Elaborado pelo autor

Terminada a execução, pode-se averiguar que na área de trabalho, assim como outros diretórios de comum acesso, fora criado um arquivo nomeado *RyukReadMe.txt*, que se trata do *Ransom Note* da amostra, conforme ilustrado na figura 11.

Figura 11 – Ransom note do ransomware analisado



Fonte: Elaborado pelo autor

Na figura acima, pode-se ver jargões utilizados por atacantes como o reforço da potência do algoritmo de encriptação utilizado, a disponibilização destes mediante ao pagamento de uma quantia em Bitcoin, e meios de contato para entender o procedimento de obtenção dos arquivos. Nota-se na nota a importância que o atacante dá a não modificação dos arquivos encriptados e respectivas pastas modificadas, visto que estes provavelmente são necessários para o processo de desencriptação, que tem em geral processos reversos ao da criptografia, mas que precisam de pontos de entrada, que provavelmente foram alocados pelo *ransomware*.

Retornando ao software de monitoramento, podemos analisar o comportamento da ameaça *ryuk*. Conclui-se que a amostra realmente é um *dropper*, isto é, um arquivo intermediário que cria e executa o software malicioso real, conforme averígua-se na figura 12.

Figura 12 – Dropper cria e executa o ransomware

306	S:58:21.166 AM	1	ryuk.exe	CreateFileW ("C:\users\Public\qjah.exe", GENERIC_WRITE, 0, NULL, CREAT...	0x0000018c	
307	S:58:21.179 AM	1	ryuk.exe	LoadLibraryA ("kernel32.dll")	0x76550000	
308	S:58:21.179 AM	1	ryuk.exe	GetProcAddress (0x76550000, "IsWow64Process")	0x765706e0	
309	S:58:21.179 AM	1	ryuk.exe	GetCurrentProcess ()	GetCurrentProc...	
310	S:58:21.179 AM	1	ryuk.exe	IsWow64Process (GetCurrentProcess(), 0x005ffa2c)	TRUE	
311	S:58:21.179 AM	1	ryuk.exe	FreeLibrary (0x76550000)	TRUE	
312	S:58:21.179 AM	1	ryuk.exe	WriteFile (0x0000018c, 0x00327b0, 174592, 0x005ffa28, NULL)	TRUE	
313	S:58:21.179 AM	1	ryuk.exe	CloseHandle (0x0000018c)	TRUE	
314	S:58:21.179 AM	1	ryuk.exe	ShellExecuteW (NULL, NULL, "C:\users\Public\qjah.exe", "C:\Users\Mathe...	0x0000002a	
315	S:58:21.183 AM	1	ntdll.dll	->DllMain (0x77240000, DLL_PROCESS_ATTACH, NULL)	TRUE	
316	S:58:21.183 AM	1	ntdll.dll	->DllMain (0x77710000, DLL_PROCESS_ATTACH, NULL)	TRUE	
317	S:58:21.183 AM	1	RPCRT4.dll	->ResolveDelayLoadedAPI (0x76300000, 0x763a7204, NULL, 0x76df3160, 0...	0x7643b410	
318	S:58:21.183 AM	1	ntdll.dll	->NtWaitForSingleObject (0x7643b400, 0x00000001, NULL, NULL, NULL)	TDUE	

Fonte: Elaborado pelo autor

Nesse concernente, nota-se que o nome do *ransomware* é aleatório e deriva-se provavelmente do uso da *RtlGenRandom*, que gera números aleatórios que se convertidos em ASCII, podem gerar um nome randômico, o que dificulta possíveis rastreamentos deste arquivo por sistemas de detecção. Ademais, analisa-se que posterior a criação do executável, carrega-se o módulo *Kernel32*, e executa-se no

endereço do processo atual (*ryuk.exe*) a função *IsWow64Process*, que retorna um booleano para verificação do sistema atual possui arquitetura de 64 bits, o que é verdade no caso da máquina virtual utilizada, e que, em caso contrário, significaria arquitetura de 32 bits. Em seguida, com o *handler* de modificação do arquivo ainda aberto, escreve-se neste seu conteúdo de acordo com a arquitetura da máquina invadida via função *WriteFile*. Por fim, fecha-se a referência ao ponteiro do arquivo, e executa-se a função *ShellExecuteW*, com os seguintes parâmetros:

Figura 13 – Função *ShellExecuteW* e seus parâmetros

API	<i>ShellExecuteW</i>
Module	<i>Shell32.dll</i>
Category	Windows Shell
<hr/>	
ShellExecuteW (
NULL,	
NULL,	
"C:\users\Public\qkjah.exe",	
"C:\Users\Matheus\Desktop\ryuk.exe",	
NULL,	
SW_HIDE	
);	

Fonte: JWMSFT(2023) [18]

Os dois primeiros parâmetros apresentados pela figura 13, trata-se de parâmetros opcionais, respectivos ao gerenciamento de erros e a ação performada. Em seguida, referencia-se no terceiro parâmetro o arquivo a ser executado, no caso trata-se do *ransomware* especificamente, seguido pelo parâmetro de execução, no caso analisado sendo o caminho do *dropper*, um indício de que uma das funções iniciais do amostra é justamente excluir o *dropper*, com o fim de dificultar ações de resposta a incidente. O quinto e o sexto parâmetro são respectivos para o diretório da ação a se inicializar, no caso em específico não sendo necessária a especificação, e um macro acerca de se ocultar ou não o terminal de execução da ação. Na construção do *ransomware Ryuk*, esta ação foi ocultada via macro *SW_HIDE*, não permitindo ao usuário notar a execução do shell [18]. Posterior à essa execução, o arquivo *Ryuk.exe* encerra sua execução e é excluído, todavia a ação maliciosa continua via arquivo criado. Dessa forma, devido a rápida execução dessa ação não é permitido a análise do arquivo recém-criado, criando-se a necessidade de uma forma de análise deste antes que seja executado. Dessa forma, deve-se realizar um *patch*, no arquivo *dropper*, que impeça a execução do *ransomware*, e o deixe no diretório *C:\Users\Public*, para possibilidade de análise. Para tal alçada, restaura-se a snapshot da máquina virtual para um estado prévio a execução do *dropper*, e com o apoio do software de análise *IDA Freeware 7.7*, inspeciona-se a sessão de código do

arquivo para encontrar a função *ShellExecuteW*, e por meio de *patch*, isto é, modificação de seu código original para um customizado, impedir a execução desta. Desse modo, após *Ryuk.exe*, criar o arquivo original e preencher seu conteúdo, ele não o irá executar via Shell, permitindo assim a análise do arquivo gerado antes da ação maliciosa.

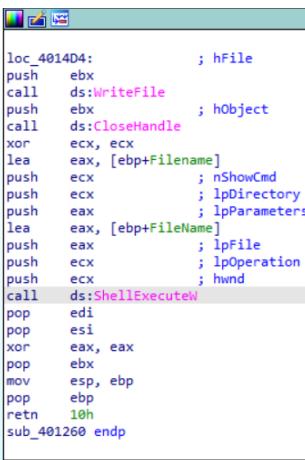
Figura 14 – Função *ShellExecuteW* encontrada no IDA em referência da sessão de dados e código

```
.idata:0040C11C ; HINSTANCE (_stdcall *ShellExecuteW)(HWND hwnd, LPCWSTR lpOperation, LPCWSTR lpFile, LPCWSTR lpParameters
idata:0040C11C           extrn ShellExecuteW:dword
idata:0040C11C           ; CODE XREF: sub_401260+296tp
idata:0040C11C           ; DATA XREF: sub_401260+296tr ...
idata:0040C120
idata:0040C120
```

Fonte: Elaborado pelo autor

Conforme mostra a figura 14, ao procurar por referências a função de execução de *Shell*, encontra-se duas chamadas a esta, uma na sessão de dados, relativa a *string* utilizada durante sua evocação em execução, e na sessão de código, que é onde efetivamente deve-se analisar para realização do patch.

Figura 15– Código Assembly com a chamada a função *ShellExecuteW*



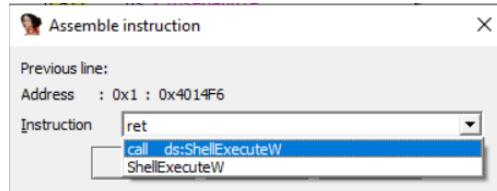
```
loc_4014D4:          ; hFile
push    ebx
call    ds:WriteFile
push    ebx          ; hObject
call    ds:CloseHandle
xor     ecx, ecx
lea     eax, [ebp+Filename]
push    ecx          ; nShowCmd
push    ecx          ; lpDirectory
push    eax          ; lpParameters
lea     eax, [ebp+FileName]
push    eax          ; lpFile
push    ecx          ; lpOperation
push    ecx          ; hwnd
call    ds:ShellExecuteW
pop     edi
pop     esi
xor     eax, eax
pop     ebx
mov     esp, ebp
pop     ebp
retn   10h
sub_401260 endp
```

Fonte: Elaborado pelo autor

Seguindo para a referência de código ilustrada na figura 14, averíguase o bloco de código ilustrado na figura 15, que mostra sequencialmente os passos tomados pelo *dropper* para criação da amostra principal e sua execução, conforme anteriormente mostrado na figura 12. Assim, para evitar a execução do *ransomware* via *dropper*, e assim permitir a análise do software de encriptação propriamente, deve-se retirar a referência a função *ShellExecuteW*, realizando um *patch*.

Dessa forma, ao invés da evocação a supracitada função, troca-se para a instrução *ret*, gerando um retorno do bloco de código ao invés de chamar a função em questão, evitando a execução do arquivo.

Figura 16 – Patch do *dropper*, evitando a chamada da ShellExecuteW



Fonte: Elaborado pelo autor

Após o *patch* do programa, salva-se as mudanças realizadas, conforme mostra a *figura 16*, e cria-se um arquivo de backup (com a instrução original), com o fim de segurança. Após essa mudança, torna-se eficiente a execução do *dropper*, sem que este engatilhe durante processamento, a execução do *ransomware*, permitindo assim a análise do executável de encriptação com maiores detalhes. Executa-se o executável *Ryuk.exe* novamente com o monitoramento do *Monitor API V2*, e constata-se com sucesso a criação do *ransomware*, mas ao invés do caso anterior, este não fora executado, permitindo assim análise concisa dessa amostra.

Figura 17 – Execução do *dropper* com *patch* comprovando a não execução do *ransomware*

285	9:26:10.973 AM	1	ryuk.exe	CreateFileW ("C:\users\Public\vzNOe.exe", GENERIC_WRITE, 0, NULL, CREA...	0x00000018c
286	9:26:10.973 AM	1	ryuk.exe	LoadLibraryA ("kernel32.dll")	0x76550000
287	9:26:10.973 AM	1	ryuk.exe	GetProcAddress (0x76550000, "IsWow64Process")	0x765706e0
288	9:26:10.973 AM	1	ryuk.exe	GetCurrentProcess ()	GetCurrentProc...
289	9:26:10.973 AM	1	ryuk.exe	IsWow64Process (GetCurrentProcess(), 0x0aaef944)	TRUE
290	9:26:10.973 AM	1	ryuk.exe	FreeLibrary (0x76550000)	TRUE
291	9:26:10.973 AM	1	ryuk.exe	WriteFile (0x00000018c, 0x00b527b0, 174592, 0x0aaef940, NULL)	TRUE
292	9:26:10.988 AM	1	ryuk.exe	CloseHandle (0x00000018c)	TRUE
293	9:26:10.988 AM	1	ryuk.exe	GetLastError ()	ERROR_SUCCESS
294	9:26:10.988 AM	1	ryuk.exe	FIsGetValue (7)	0x010b0d70
295	9:26:10.988 AM	1	ryuk.exe	SetLastError (ERROR_SUCCESS)	

Fonte: Elaborado pelo autor

Conforme averíga-se via *figura 17*, dessa vez não há chamada da função de execução de Shell, e por conseguinte, nenhuma encriptação de arquivos nos diretórios da máquina virtual. Destarte, verifica-se também no caminho especificado pela função *CreateFileW*, a presença de um executável que possui nome gerado aleatoriamente (no caso atual denominado *vzNOe.exe*), conforme elucidado anteriormente. Dessa forma, analisa-se este executável para então entender-se de maneira prática como a criptografia nos arquivos locais ocorre. Faz-se uso de análise estática para compreensão inicial acerca desse executável e possíveis indícios de seu comportamento. Utiliza-se novamente o *Pestudio 9.37* para captação de dados intrínsecos a amostra, como sua *hash SHA256*, e outros indicadores, conforme ilustra-se na *figura 18*.

Figura 18 – Dados e metadados estáticos do arquivo do executável gerado

Fonte: Elaborado pelo autor

Nota-se como fator interessante que a arquitetura do executável em questão é de 64-bit, diferentemente do anterior (32-bit), e que comprova a interpretação acerca do funcionamento do *dropper Ryuk.exe*, que funciona em 32-bit para adaptabilidade em ambas as arquiteturas, e utilizando a chamada *IsWow64Process* para conferência da arquitetura da máquina invadida para compilação adequada e funcional do *ransomware* propriamente, evitando possíveis erros de compatibilidade e execução. Tal qual o *dropper*, o *ransomware* em questão também tem compilação datada para meados de 2018, o que aparenta ser uma das primeiras amostras encontradas relacionadas ao grupo WIZARD SPIDER. Notou-se ainda que na *DOS-STUB* do arquivo, mas especificamente em seu cabeçalho opcional, conforme mencionado anteriormente, o presente executável também apresentava randomização de seu layout de memória (ASLR), o que indica a presença de técnicas de proteção a *debugging*. Tal suspeita é confirmada dado o fato da presença da string “*anti-debug: IsDebuggerPresent*”, estar presente no arquivo, mediante detecção com apoio do software *Detect it Easy*, ilustrando a preocupação dos atacantes para evitar-se análises do software utilizado e a possibilidade de criação de técnicas de mitigação e proteção à tal ameaça.

Posteriormente, ao analisar-se as bibliotecas e funções importadas da API do Windows, nota-se que a amostra importa as mesmas três *DLL*'s, que seu respectivo *dropper*: *Kernel32*, *Shell32* e *Advapi32*. Todavia, as funções importadas com aparente risco de utilização maliciosa, apresentam-se com maior amplitude em termos de utilização, revelando a natureza *multi-thread* do executável. Conforme constatado pela figura 19, a presença das seguintes funções, elucidam a característica de criação e enumeração de *Threads* no arquivo, o que pode ser uma forma de aumentar a

velocidade de ações executadas de maneira semiparalela, além de dificultar a ação de softwares de detecção devido a divisão da ação em múltiplos Threads:

- *GetCurrentThread;*
- *CreateRemoteThread;*
- *GetCurrentThreadId;*
- *OpenThreadToken.*

Figura 19 – Funções importadas pelo executável com risco eminente

functions (94)	flag (25)	ordinal (0)	library (3)
OpenProcess	x	-	kernel32.dll
CreateToolhelp32Snapshot	x	-	kernel32.dll
Process32NextW	x	-	kernel32.dll
GetCurrentThread	x	-	kernel32.dll
DeleteFileW	x	-	kernel32.dll
Process32FirstW	x	-	kernel32.dll
CreateRemoteThread	x	-	kernel32.dll
RtlLookupFunctionEntry	x	-	kernel32.dll
TerminateProcess	x	-	kernel32.dll
GetCurrentProcessId	x	-	kernel32.dll
GetCurrentThreadId	x	-	kernel32.dll
RaiseException	x	-	kernel32.dll
GetModuleHandleExW	x	-	kernel32.dll
WriteFile	x	-	kernel32.dll
FindFirstFileExA	x	-	kernel32.dll
FindNextFileA	x	-	kernel32.dll
GetEnvironmentStringsW	x	-	kernel32.dll
WriteProcessMemory	x	-	kernel32.dll
SystemFunction036	x	-	advapi32.dll
LookupPrivilegeValueW	x	-	advapi32.dll
AdjustTokenPrivileges	x	-	advapi32.dll
OpenProcessToken	x	-	advapi32.dll
OpenThreadToken	x	-	advapi32.dll
LookupAccountSidW	x	-	advapi32.dll
ShellExecuteW	x	-	shell32.dll

Fonte: Elaborado pelo autor

Constata-se pelas funções apontadas na *figura 19*, que a ação do programa não se restringe ao mapeamento e criação de *Threads*, mas também com o gerenciamento e enumeração de processos ativos. Devido à ausência de uma função correlata a criação de processos (como *CreateProcessA* ou *CreateProcessAsUserA*), o gerenciamento de processos aparenta revelar a utilização de uma técnica denominada *Process Injection* [13], que consiste enumerar os processos em execução e injetar código malicioso nestes, em geral utilizando *Threads* auxiliares, dificultando o mapeamento das atividades realizadas, além de poder obter-se permissões e privilégios mais elevados de processos já em execução. As funções relacionadas a manipulação de processos elucidadas na *figura 19* são as seguintes:

- *OpenProcess;*
- *CreateToolhelp32Snapshot;*
- *Process32NextW;*
- *Process32FirstW;*
- *TerminateProcess;*
- *GetCurrentProcessId;*
- *WriteProcessMemory;*
- *OpenProcessToken;*

Dentre as citadas, as que mais elucidam a utilização de *Process Injection* são, *OpenProcess*, que dado o PID (identificador de um processo), consegue abrir o processo; *CreateToolhelp32Snapshot*, que cria um snapshot do processo, tal qual seu gerenciamento de memória, *threads* e *heaps*; e *WriteProcessMemory*, que permite escrever ou modificar na memória do processo aberto, permitindo assim a injeção de código malicioso durante a execução em endereços específicos mapeados anteriormente pela *CreateToolhelp32Snapshot*. Todavia, chama a atenção a presença não explícita de funções de criptografia da *API* do Windows, visto que estas são cruciais, na maioria dos casos para encriptação de arquivos nos ambientes Microsoft. Dessa forma, necessita-se de buscas efetivas acerca da importação desses tipos de funções, que podem ser verificadas em posteriores análises de *debugging* ou execução monitorada do executável.

Ao analisar-se as *strings* presentes no arquivo, percebe-se que muitas das encontradas, são as mesmas presentes no arquivo *dropper*, levando a conclusão de que a presença destes no primeiro, tinham como único objetivo de serem repassadas estática e dinamicamente ao executável do *ransomware* propriamente. Dentre estas, ressalta-se aquelas respectivas a importação de funções maliciosas, comandos executados com objetivo de persistência conforme elucidado anteriormente e o texto base utilizado para o *Ransom Note* da amostra.

Concomitantemente, verificou-se as sessões do arquivo, e nota-se uma busca diferença quando comparado com o arquivo *dropper* (este último que possuía uma sessão de dados muito maior que as demais, essa provavelmente contendo material para transcrição de código e comandos para o executável a ser gerado posteriormente). Conforme averígua-se pela *figura 20*, mais de 50% dos bytes do executável são correlatos à sessão de código, explicitando a densidade da execução esperada, advinda de enumeração de arquivos e processos, criação de *threads*, injeção de processos, criação de arquivos de texto, encriptação de arquivos e outros esperados desta ação maliciosa.

Figura 20 – Sessões de arquivo do executável de encriptação

property	value	value	value	value	value	value
name	.text	.rdata	.data	.pdata	.gfps	.rsrc
md5	E92E1CE20AB2CCC86021B8...	8EB0AD9FF18AA268125F88...	D2FC8C8CC817C49ED6524...	99FC798789F3970E8E62681C...	C46634C37FD4CBB0C5D33...	AA46FB8F7A37BDE9C5429...
entropy	6,445	5,543	3,853	5,095	1,415	4,714
file-ratio (99,41%)	50,15 %	29,03 %	15,84 %	2,64 %	0,29 %	0,29 %
raw-address	-	-	-	-	-	-
raw-size (173568 bytes)	0x000000400	0x000015A00	0x000022000	0x000028C00	0x000029E00	0x00002A000
virtual-address	0x00015600 (87552 bytes)	0x0000C600 (50688 bytes)	0x0000E00 (27648 bytes)	0x00001200 (4608 bytes)	0x00000200 (512 bytes)	0x00000200 (512 bytes)
virtual-size	0x00001000	0x000017000	0x000024000	0x0000389000	0x000080000	0x0038C0000
virtual-size (3700864 bytes)	0x000154F0 (87280 bytes)	0x0000C428 (50216 bytes)	0x00364538 (355664 bytes)	0x00000194 (4500 bytes)	0x00000048 (168 bytes)	0x000001E0 (480 bytes)
entry-point	0x00007934	-	-	-	-	-
characteristics	0x60000020	0x40000040	0xC0000040	0x40000040	0x40000040	0x40000040
writable	-	-	x	-	-	-
executable	x	-	-	-	-	-
shareable	-	-	-	-	-	-
discardable	-	-	-	-	-	-
initialized-data	-	x	x	x	x	x
uninitialized-data	-	-	-	-	-	-
unreadable	-	-	-	-	-	-
self-modifying	-	-	-	-	-	-
virtualized	-	-	-	-	-	-
file	n/a	n/a	n/a	n/a	n/a	n/a

Fonte: Elaborado pelo autor

Após as constatações estáticas, executa-se o arquivo disponibilizado pelo *dropper* (na execução em questão o arquivo tinha nome randômico de *vzNOe.exe*), com o auxílio da aplicação *API Monitor*, com objetivo de se rastrear os passos executados pelo arquivo e outros aspectos somente percebidos durante a execução, como a encriptação. Dessa forma, configura-se o rastreamento de funções das bibliotecas importadas pelo software em sua *import table*. De tal forma após a execução, confirma-se que o software, de fato é o responsável pela encriptação dos arquivos e diretórios da máquina virtual, e permite-se, por conseguinte o entendimento mais profundo de seu comportamento. Inicialmente, nota-se que após realizar o carregamento de bibliotecas e funções necessárias para seu funcionamento, a primeira ação do executável é processar a linha de comando de sua execução (em teoria advinda da função *ShellExecuteW* do arquivo *dropper*), na qual um dos argumentos é o caminho do arquivo *dropper*, e logo em seguida realiza a exclusão deste, com objetivo de ocultação de vestígios do ataque, conforme apresenta a figura 21.

Figura 21 – Exclusão do arquivo *dropper* realizada para ocultar vestígios

214	1:47:31.947 PM	1	vzNOe.exe	GetCommandLineW ()	0x00000271bf4...
215	1:47:31.947 PM	1	vzNOe.exe	CommandLineToArgvW ("C:\Users\Public\vzNOe.exe" C:\Users\Matheus\...)	0x00000271bf4...
216	1:47:31.947 PM	1	ntdll.dll	DllMain (0x00007ffe5d4ae000, 1, 0x0000000000000000)	1
217	1:47:31.947 PM	1	ntdll.dll	DllMain (0x00007ffe5c3c0000, 1, 0x0000000000000000)	1
218	1:47:31.947 PM	1	vzNOe.exe	DeleteFileW ("C:\Users\Matheus\Desktop\ryuk.exe")	1
219	1:47:31.964 PM	1	vzNOe.exe	LocalFree (0x00000271bf4264a0)	0x000000000000...

Fonte: Elaborado pelo autor

Em seguida, o arquivo inicia o estabelecimento de persistência no sistema. Conforme averiguado anteriormente tal fato ocorre por meio da modificação da chave de registro, dando ao executável malicioso a capacidade de ser resiliente a táticas de mitigação mais simples. Tal estratégia é consolidada novamente pela utilização da função de execução *ShellExecuteW*, que recebe os parâmetros de modificação do

registro e aplicação das propriedades de persistência ao executável, conforme é explicitado pela *figura 22*.

Figura 22 – Execução de shell de persistência na máquina alvo

API	ShellExecuteW
Module	Shell32.dll
Category	Windows Shell
<pre>ShellExecuteW (0x0000000000000000, 0x0000000000000000, "C:\Windows\System32\cmd.exe", "/C REG ADD \"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\" /t REG_SZ /d \"C:\Users\Public\vzNOe.exe\" /f", 0x0000000000000000, 0);</pre>	

Fonte: Elaborado pelo autor

Posteriormente, o executável inicia uma escalada de obtenção de privilégios com objetivo de garantir níveis de execução privilegiados para modificação livre de arquivos e diretórios, além da possibilidade de enumeração e gerenciamento de processos, este último sendo necessário para concretização da técnica de *Process Injection* mencionada anteriormente. Nesse sentido, a *figura 23* ilustra os passos utilizados pelo software para a desejada elevação, na qual, primeiramente obtém-se um *handler* para o atual *thread* em execução no processo e com este em posse, realiza-se a obtenção do *token* de acesso deste, que será essencial para o ajuste de privilégios necessários para o gerenciamento dos processos em execução; em seguida é evocada a função *LookupPrivilegeValueW*, na qual passada o segundo parâmetro como *SeDebugPrivilege*, constitui-se como necessário para o ajuste do token de acesso de segurança a outros processos [19]. Caso tal função conclua sua execução de maneira adequada, retornando assim um valor booleano verdadeiro, é chamada em sequência a função *AdjustTokenPrivileges*, que recebendo o *token* do *thread* atual, e com valor correto para ajuste dos privilégios, consegue consolidar os acessos necessários do atual processo para conseguir modificar e injetar código maliciosamente em outros processos ativos na máquina invadida, dificultando detecção durante execução deste.

Figura 23 – Escalada de privilégios para injeção em processos

477	1:47:32.057 PM	1	vzNOe.exe	GetCurrentThread ()	0xfffffffffffffffffe
478	1:47:32.057 PM	1	vzNOe.exe	OpenThreadToken (0xfffffffffffffe, 40, 0, 0x000000fc29c3df90)	1
479	1:47:32.057 PM	1	vzNOe.exe	LookupPrivilegeValueW (0x0000000000000000, "SeDebugPrivilege", 0x000...)	1
480	1:47:32.057 PM	1	ADVAPI32.dll	LocalAlloc (64, 56)	0x000000271bf4...
481	1:47:32.057 PM	1	ADVAPI32.dll	LocalFree (0x0000000000000000)	0x000000000000...
482	1:47:32.057 PM	1	ADVAPI32.dll	LocalFree (0x0000000000000000)	0x000000000000...
483	1:47:32.057 PM	1	ADVAPI32.dll	LocalFree (0x00000271bf47f120)	0x000000000000...
484	1:47:32.057 PM	1	vzNOe.exe	AdjustTokenPrivileges (0x000000000000002a0, 0, 0x000000fc29c3df48, 16, 0...)	1

Fonte: Elaborado pelo autor

Com as permissões necessárias para gerenciamento dos processos, o executável inicia a percorrer todos os processos em execução na máquina, seguindo

um procedimento iterativo, ilustrado pelas *figuras 24 e 25*. Antes de iniciar a busca, o executável armazena informações do processo atual via função *CreateToolHelp32Snapshot*, conforme elucidado anteriormente, e utilizando a chamada *Process32NextW*, torna-se possível procurar todos os processos subjacentes no sistema operacional, visto que em distribuições DOS do Windows, por todos os processos ativos são duplamente ligados sequencialmente em uma lista ligada [20], ao se possuir acesso a um nó desta lista (o processo atual) em conjunto a privilégios elevados de acesso e gerenciamento de processos, se torna possível listar, abrir e gerenciar toda a cadeia de processos na máquina, sendo essa estratégia utilizada pelo *ransomware*, para injeção de código malicioso em diversos processos, acelerando o processo de encriptação de arquivos. Nota-se que no decorrer da iteração, são disparadas exceções ao abrir e guardar *handlers* para alguns destes, sobretudo os primeiros a serem percorridos (elucidados pela cor amarela nas chamadas de funções rastreadas pelo *API Monitor*, conforme ilustra a *figura 24*), que decorre, provavelmente, destes não permitirem gerenciamento externo, independente de privilégios utilizados no *user space*, por serem primordiais para funcionamento e gerenciamento do sistema operacional.

Figura 24 – Ryuk inicia iteração de processos ativos na máquina

498	1:47:32.057 PM	1	vzNOe.exe	<i>CreateToolhelp32Snapshot</i> (2, 0)	0x000000000000...
499	1:47:32.057 PM	1	vzNOe.exe	<i>Process32FirstW</i> (0x0000000000000029c, 0x000000fc29c3dcf0)	1
500	1:47:32.057 PM	1	vzNOe.exe	<i>Process32NextW</i> (0x0000000000000029c, 0x000000fc29c3dcf0)	1
501	1:47:32.057 PM	1	vzNOe.exe	<i>SetLastError</i> (0)	
502	1:47:32.057 PM	1	vzNOe.exe	<i>OpenProcess</i> (2097151, 0, 4)	0x000000000000...
503	1:47:32.057 PM	1	vzNOe.exe	<i>CloseHandle</i> (0x0000000000000000)	0
504	1:47:32.057 PM	1	vzNOe.exe	<i>Process32NextW</i> (0x0000000000000029c, 0x000000fc29c3dcf0)	1
505	1:47:32.057 PM	1	vzNOe.exe	<i>SetLastError</i> (0)	
506	1:47:32.057 PM	1	vzNOe.exe	<i>OpenProcess</i> (2097151, 0, 72)	0x000000000000...
507	1:47:32.057 PM	1	vzNOe.exe	<i>CloseHandle</i> (0x0000000000000000)	0
508	1:47:32.057 PM	1	vzNOe.exe	<i>Process32NextW</i> (0x0000000000000029c, 0x000000fc29c3dcf0)	1
509	1:47:32.057 PM	1	vzNOe.exe	<i>SetLastError</i> (0)	
510	1:47:32.057 PM	1	vzNOe.exe	<i>OpenProcess</i> (2097151, 0, 332)	0x000000000000...
511	1:47:32.057 PM	1	vzNOe.exe	<i>CloseHandle</i> (0x0000000000000000)	0
512	1:47:32.057 PM	1	vzNOe.exe	<i>Process32NextW</i> (0x0000000000000029c, 0x000000fc29c3dcf0)	1
513	1:47:32.057 PM	1	vzNOe.exe	<i>SetLastError</i> (0)	

Fonte: Elaborado pelo autor

Todavia, conforme o executável encontra processos que sejam suscetíveis a abertura (realizada inicialmente pela chamada *OpenProcess*), consegue a obtenção do *token* do processo, essencial para gerenciamento, seguida pela obtenção do nome, grupos e contas disponíveis na máquina, mediante o fornecimento do *security identifier* [21] do processo (disponibilizado via *token*) a função *LookupAccountSidW*, que retorna tais informações. Após a obtenção de tais dados de contas, usuários e *token* do processo, fecha-se a referência ao *handler*, e inicia-se a busca pelo próximo processo encadeado. É interessante notar-se que nessa iteração, não há qualquer

indício, ainda, de criação de threads paralelos para execução, o que gera indícios que tais dados são armazenados inicialmente em uma estrutura de dados para uso futuro na criação de threads que irão injetar código malicioso na referência de memória dos processos percorridos, e então realizar a criptografia de arquivos.

Figura 25 – Etapas tomadas para captura de informações do processo iterado

1:47:32.120 PM	1	vzNOe.exe	Process32NextW (0x0000000000000029c, 0x000000fc29c3dcf0)	1
1:47:32.120 PM	1	vzNOe.exe	SetLastError (0)	
1:47:32.120 PM	1	vzNOe.exe	OpenProcess (2097151, 0, 3816)	0x000000000000...
1:47:32.120 PM	1	vzNOe.exe	OpenProcessToken (0x000000000000002a4, 131080, 0x000000fc29c3dce0)	1
1:47:32.120 PM	1	vzNOe.exe	GetTokenInformation (0x0000000000000045c, 1, 0x00000271bf47f120, 0, 0x...)	0
1:47:32.120 PM	1	vzNOe.exe	GetProcessHeap ()	0x00000271bf4...
1:47:32.120 PM	1	vzNOe.exe	HeapAlloc (0x00000271bf410000, 8, 44)	0x00000271bf4...
1:47:32.120 PM	1	vzNOe.exe	GetTokenInformation (0x0000000000000045c, 1, 0x00000271bf47eaa0, 44, 0...)	1
1:47:32.120 PM	1	vzNOe.exe	LookupAccountSidW (0x0000000000000000, 0x00000271bf47eab0, 0x0000...)	0
1:47:32.120 PM	1	vzNOe.exe	GlobalAlloc (0, 16)	0x00000271bf4...
1:47:32.120 PM	1	vzNOe.exe	GlobalAlloc (0, 32)	0x00000271bf4...
1:47:32.120 PM	1	vzNOe.exe	LookupAccountSidW (0x0000000000000000, 0x00000271bf47eab0, 0x0000...)	1
1:47:32.120 PM	1	vzNOe.exe	GlobalFree (0x00000271bf4850c0)	0x000000000000...
1:47:32.120 PM	1	vzNOe.exe	GlobalFree (0x00000271bf48a5f0)	0x000000000000...
1:47:32.120 PM	1	vzNOe.exe	GetProcessHeap ()	0x00000271bf4...
1:47:32.120 PM	1	vzNOe.exe	HeapFree (0x00000271bf410000, 1, 0x00000271bf47eaa0)	1
1:47:32.120 PM	1	vzNOe.exe	CloseHandle (0x000000000000002a4)	1

Fonte: Elaborado pelo autor

Ao chegar ao fim da lista ligada, fato constatado devido a exceção gerada pela função *Process32NextW* conforme apresentado na figura 26, inicia-se a criação de *threads* que irão, por fim, injetar código malicioso nos processos que tiveram informações armazenadas. Dessa forma, conforme visualiza-se na figura 26, inicia-se um novo processo iterativo, no qual, realiza-se novamente a abertura de processos com informações armazenadas anteriormente com a função *OpenProcess*, e em seguida, aloca-se espaço de memória neste com a invocação de *VirtualAllocEx*, permitindo assim a injeção de código no processo invadido com a chamada *WriteProcessMemory* que, provavelmente, deposita instruções e carregamentos necessários para a encriptação dos arquivos. Finalizada a escrita de código de malicioso, *Ryuk* cria um novo *thread* com a função *CreateRemoteThread*, permitindo a execução remota do recém-criado *thread* no processo invadido, garantindo simultaneamente, maior velocidade na encriptação dos arquivos devido a execução semiparalela destes, e encobrimento de rastros, visto que ao se monitorar somente o *thread* original do processo do *ransomware*, não são encontradas quaisquer chamadas à funções criptográficas, o que dificulta a mitigação de danos ou identificação de atividade maliciosa.

Figura 26 – Criação de *threads* remotos para execução de código no processo invadido

1501	1:47:32.183 PM	1	vzNOe.exe	Process32NextW (0x0000000000000029c, 0x000000fc29c3dcf0)	0
1502	1:47:32.183 PM	1	vzNOe.exe	CloseHandle (0x0000000000000029c)	1
1503	1:47:32.183 PM	1	vzNOe.exe	GetModuleFileNameW (0x0000000000000000, 0x000000fc29cff710, 100)	25
1504	1:47:32.183 PM	1	vzNOe.exe	SetLastError (0)	
1505	1:47:32.183 PM	1	vzNOe.exe	OpenProcess (2097151, 0, 3040)	0x000000000000...
1506	1:47:32.183 PM	1	vzNOe.exe	GetModuleHandleA (0x0000000000000000)	0x00000ff6d91...
1507	1:47:32.183 PM	1	vzNOe.exe	SetLastError (0)	
1508	1:47:32.183 PM	1	vzNOe.exe	VirtualAllocEx (0x0000000000000029c, 0x00007ff6d9120000, 3727360, 12288, f)	0x00007ff6d91...
1509	1:47:32.183 PM	1	vzNOe.exe	WriteProcessMemory (0x000000000000000029c, 0x00007ff6d9120000, 0x0000000000000000, 1)	1
1510	1:47:32.183 PM	1	vzNOe.exe	CreateRemoteThread (0x0000000000000029c, 0x0000000000000000, 0, 0x0000000000000000, 0, 0x0000000000000000)	0x000000000000...
1511	1:47:32.183 PM	1	vzNOe.exe	Sleep (300)	
1512	1:47:32.495 PM	1	vzNOe.exe	SetLastError (0)	

Fonte: Elaborado pelo autor

Ulteriormente, ao se visualizar um dos recém-criados *threads* pelo *Ryuk* que executa código injetado em um processo arbitrariamente invadido, constata-se, finalmente a presença de funções criptográficas, que ao enumerar um dado arquivo em um dos diretórios, inicia-se o processo de encriptação. Constatou-se que como tais funções importadas não constavam na tabela de *imports* do executável de acordo com o apurado na anterior análise estática, conclui-se que o *ransomware* importa tais funções de maneira dinâmica durante a execução, provavelmente utilizando funções de carregamento, com objetivo de se evitar suspeitas de softwares de detecção, que na maior parte dos casos, encontram ameaças deste tipo devido à presença destas funções, que, de maneira clara, realizam a principal ação conhecida destes tipos de ataques.

Figura 27 – Processo de encriptação realizado pelo *Ryuk*

26421	1:47:55.322 PM	356	vzNOe.exe	SetFileAttributesW ("C:\MinGW\include\sys\bsdtypes.h", 128)	1
26422	1:47:55.322 PM	356	vzNOe.exe	CreateFileW ("C:\MinGW\include\sys\bsdtypes.h", 3221225472, 0, 0x00000... 0x000000000000...	
26423	1:47:55.322 PM	356	vzNOe.exe	GetFileSizeEx (0x0000000000000000f48, 0x000000fc353ffce8)	1
26424	1:47:55.322 PM	356	vzNOe.exe	GetFileSizeEx (0x0000000000000000f48, 0x000000fc353ffd00)	1
26425	1:47:55.322 PM	356	vzNOe.exe	SetFilePointerEx (0x0000000000000000f48, { u = { LowPart = 2298, HighPart = ... } })	1
26426	1:47:55.322 PM	356	vzNOe.exe	ReadFile (0x0000000000000000f48, 0x000000fc353ffd50, 25, 0x000000fc353ffd...)	1
26670	1:47:55.369 PM	356	vzNOe.exe	SetFilePointer (0x0000000000000000f48, 0, 0x0000000000000000, 0)	0
26671	1:47:55.369 PM	356	vzNOe.exe	CryptGenKey (0x00000271bf476a50, 26128, 1, 0x000000fc353ffcd0)	1
26672	1:47:55.369 PM	356	rsaenh.dll	-BCryptGenerateSymmetricKey (0x000000271bf497190, 0x000000fc353ff7c0, 0x0000000000000000)	
26673	1:47:55.369 PM	356	vzNOe.exe	-VirtualAlloc (0x0000000000000000, 1000090, 4096, 4)	0x000000271c23...
26674	1:47:55.369 PM	356	vzNOe.exe	-SetFilePointer (0x0000000000000000f48, 0, 0x0000000000000000, 0)	0
26675	1:47:55.369 PM	356	vzNOe.exe	-ReadFile (0x0000000000000000f48, 0x000000271c23d0000, 2588, 0x000000fc35...)	1
26676	1:47:55.369 PM	356	vzNOe.exe	-CryptEncrypt (0x00000271bf48f7a0, 0x0000000000000000, 1, 0, 0x000000...)	1
26677	1:47:55.369 PM	356	rsaenh.dll	-BCryptDestroyKey (0x000000271bf486eb0)	0x0000000000000000
26678	1:47:55.369 PM	356	rsaenh.dll	-BCryptGenerateSymmetricKey (0x000000271bf497190, 0x000000fc353ff700, 0x0000000000000000)	
26679	1:47:55.369 PM	356	rsaenh.dll	-BCryptSetProperty (0x000000271bf486eb0, "IV", 0x000000fc353ff7c8, 16, 0)	0x0000000000000000
26680	1:47:55.369 PM	356	rsaenh.dll	-BCryptSetProperty (0x000000271bf486eb0, "ChainingMode", 0x000007ff...)	0x0000000000000000
26681	1:47:55.369 PM	356	vzNOe.exe	-CryptEncrypt (0x000000271bf48f7a0, 0x0000000000000000, 1, 0, 0x00000027...)	1
26682	1:47:55.369 PM	356	rsaenh.dll	-BCryptSetProperty (0x000000271bf486eb0, "IV", 0x000000271bf466c4, 16, 0)	0x0000000000000000
26683	1:47:55.369 PM	356	rsaenh.dll	-BCryptEncrypt (0x000000271bf486eb0, 0x000000271c23d0000, 2588, 0x00...)	0x0000000000000000
26684	1:47:55.369 PM	356	vzNOe.exe	-SetFilePointer (0x0000000000000000f48, 0, 0x0000000000000000, 0)	0
26685	1:47:55.369 PM	356	vzNOe.exe	-WriteFile (0x0000000000000000f48, 0x000000fc353ffd18, 6, 0x000000fc353ffce4, 1)	1
26686	1:47:55.369 PM	356	vzNOe.exe	-CryptExportKey (0x000000271bf48f7a0, 0x000000271bf48ed90, 1, 0, 0x00000...)	1
26687	1:47:55.369 PM	356	vzNOe.exe	-CryptExportKey (0x000000271bf48f7a0, 0x000000271bf48ed90, 1, 0, 0x00000...)	1
26688	1:47:55.369 PM	356	vzNOe.exe	-BCryptEncrypt (0x000000271bf48f7a0, 0x000000271bf48ed90, 1, 0, 0x00000...)	1
26689	1:47:55.369 PM	356	rsaenh.dll	-BCryptEncrypt (0x000000271bf485b10, 0x000000271bf466cb8, 32, 0x0000...)	0x0000000000000000
26690	1:47:55.369 PM	356	vzNOe.exe	-WriteFile (0x0000000000000000f48, 0x000000fc353ffd70, 268, 0x000000fc353ff...)	1
26691	1:47:55.369 PM	356	vzNOe.exe	-CloseHandle (0x0000000000000000f48)	1
26692	1:47:55.369 PM	356	vzNOe.exe	-VirtualFree (0x000000271c23d0000, 0, 32768)	1
26693	1:47:55.369 PM	356	vzNOe.exe	-CryptDestroyKey (0x000000271bf48f7a0)	1
26694	1:47:55.369 PM	356	rsaenh.dll	-BCryptDestroyKey (0x000000271bf486eb0)	0x0000000000000000

Fonte: Elaborado pelo autor

Pela figura 27, nota-se que as principais funções de finalidade criptográfica são:

- *CryptGenKey*, responsável por gerar um par de chaves criptográficas (privada e pública). O segundo argumento desta função é de suma importância, pois nele é especificado o algoritmo padrão gerador da chave criptográfica a ser gerada [22], e pela figura, constata-se que tem valor decimal 26128 (hexadecimal 0x66610), que conforme apresenta a documentação da Microsoft [23], corresponde ao algoritmo *256 bit AES* (Advanced Encryption Standard), algoritmo padrão de encriptação de dados mais comum utilizado atualmente em softwares e redes, visto que é impenetrável e resistente a ataques de força-bruta [24];
- *CryptEncrypt*, função encarregada por realizar a encriptação propriamente, que recebe dentre seus parâmetros, um *handler* para a chave gerada anteriormente e um *pointer* para o *buffer* do conteúdo a ser criptografado (neste caso, do arquivo recém-aberto pela função *ReadFile*), visto que seu conteúdo original será sobreescrito pela encriptação deste;
- *CryptExportKey*, que tem objetivo de exportar a chave recém gerada para uma localização especificada em seus parâmetros, utilizada para reutilização da chave de maneira posterior;
- *CryptDestroyKey*, que apaga a referência do *handler* a chave criptográfica gerada, com objetivo principal de se apagar referências à chave utilizada durante a execução.

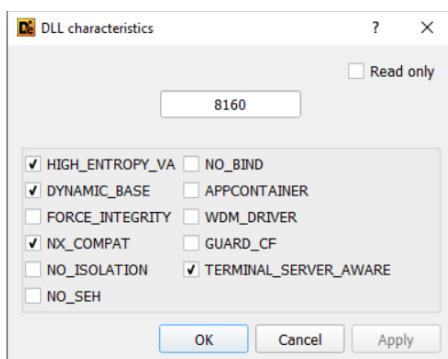
Após a encriptação do arquivo o *ransomware*, em seu *thread* principal, continua a enumerar processos e iniciar remotamente a execução de código malicioso que irá progressivamente encriptar arquivos nos diretórios invadidos. Devido à grande complexidade de arquivos a serem encriptados e diversas peculiaridades destes (extensões diferentes, cabeçalhos ou tamanhos) o software demonstra outras ações que aparentemente não são captadas em sua completude pela execução dinâmica rastreada pelo *API Monitor*, fazendo-se necessária uma análise com maior segmentação, determinada pela técnica de *debugging* do arquivo, na qual consiste em uma dissecação do alvo, analisando valores de registradores em certas instruções, *breakpoints* em chamadas específicas e lógica utilizada nas ações realizadas, que permitem elucidar com maior detalhamento comportamentos do ataque, tais quais como tratamento de exceções, a importação das funções de criptografia e a execução remota do código malicioso em outros processos.

2.4.4 Debuging

Destarte a necessidade de entendimento pormenorizado de determinadas ações durante a execução, utiliza-se o software de *debugging Xdbg64*, que permitirá tal alcada. Nota-se novamente, que devido a não conexão de um host de internet na *sandbox* utilizada (por fins de segurança), é provável que a detecção de comportamentos relativos a contatos com a C&C do ransomware ou encriptação de drivers de rede não sejam adequadamente detectados e analisados.

Inicialmente, devido ao arquivo possuir *address-space-layout-randomization* (ASLR) em seu cabeçalho opcional de seu *NT Header*, se faz necessário a customização desse campo com intuito de verificação de endereços de memória fixos, que permitam assim o exame de endereços com maior precisão e possibilitando reexecuções, podendo-se utilizar bases de endereços já capturados anteriormente. Para realizar essa modificação no cabeçalho do arquivo, utiliza-se novamente o *Detect it Easy 2.05*, para advir o *patch* no executável.

Figura 28 – Desativando propriedade de ASLR (Dynamic Base) no *ransomware*



Fonte: Elaborado pelo autor

Com a randomização do endereço de memória do arquivo desativada, é possível capturar endereços padrões de chamadas de funções e modificação de valores e configurar *breakpoints* nestes, permitindo visualização dos valores contidos em registradores e sessão de dados do executável, mostrando maiores detalhes durante a execução. Desse modo, utilizando o software *IDA Freeware 7.7*, realiza a busca por endereços de chamadas de funções tidas como importantes que ocorram no *thread* principal do processo, permitindo a análise posterior de outros *threads* e processos criados pela diretiva do executável. Conforme já averiguado em análise estática, as funções de criptografia não aparecem como sua importação definida no arquivo, sendo feita, provavelmente por execução dinâmica durante a execução. Desse modo,

procurar endereços dessas funções via *IDA*, não é uma hipótese factível. Destarte, são procurados endereços respectivos a funções de modificação da memória de processos, vide *Process Injection*, e criação de *threads* remotos, que pela execução lógica estão relacionadas a utilização e invocação das funções requisitadas e podem apresentar assim mais detalhes sobre o comportamento dessa ação principal do executável. São capturados então, endereços de execução das seguintes funções:

- *OpenProcess*;
- *WriteProcessMemory*;
- *CreateRemoteThread*.

Figura 29 – Chamadas e endereços das funções supracitadas acima via *IDA Freeware*

```
.text:00000001400025B3      mov    r8d, ebx      ; dwProcessId
.text:00000001400025B6      xor    edx, edx      ; bInheritHandle
.text:00000001400025B8      mov    ecx, 1FFFFFh   ; dwDesiredAccess
.text:00000001400025BD      call   cs:OpenProcess

.text:000000014000265C      mov    r9, rbp      ; nSize
.text:000000014000265F      mov    qword ptr [rsp+68h+f1Protect], rax ;
.text:0000000140002664      mov    r8, rsi      ; lpBuffer
.text:0000000140002667      mov    rdx, rbx      ; lpBaseAddress
.text:000000014000266A      mov    rcx, rdi      ; hProcess
.text:000000014000266D      call   cs:WriteProcessMemory

.text:000000014000269C      mov    [rsp+68h+lpThreadId], 0 ; lpThreadId
.text:00000001400026A5      lea    r9, StartAddress ; lpStartAddress
.text:00000001400026AC      mov    [rsp+68h+dwCreationFlags], 0 ; dwCrea
.text:00000001400026B4      xor    r8d, r8d      ; dwStackSize
.text:00000001400026B7      xor    edx, edx      ; lpThreadAttributes
.text:00000001400026B9      mov    qword ptr [rsp+68h+f1Protect], rbx ;
.text:00000001400026BE      call   cs>CreateRemoteThread
```

Fonte: Elaborado pelo autor

De posse dos endereços de chamadas na *figura 29*, pode-se inicializar a examinação da amostra *Ryuk* no debugger *Xdbg64*. Ao abrir o executável no debugger, segue-se os endereços de chamadas (respectivamente *0x1400025BD*, *0x14000266D* e *0x1400026BE*) ilustrados na *figura* acima, e estabelecendo *breakpoints* nestes, permitindo com maiores detalhes o exame do valor contido em registradores no momento das chamadas, permitindo o rastreamento o processo invadido e outras nuances utilizadas durante o ataque. Posterior a alocação destas paradas de software, inicia-se a execução do *ransomware* via *Xdbg64*. Anterior a primeiro breakpoint de invasão de processos, nota-se que o processo armazena o nome de três processos, que posteriormente serão utilizados com o fim de evitar de a rotina de injeção de código não os atinja, dado que os três são processos essenciais de gerenciamento de arquivos, sistema e permissões do Windows, e assim, uma modificação na rotina desses pode gerar erros de sessão do sistema operacional, e prejudicar o posterior processo de encriptação.

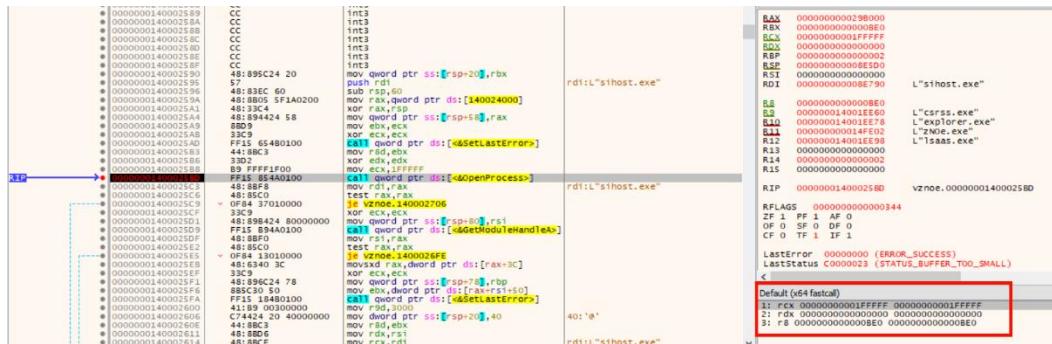
Figura 30 – Processos que o Ryuk evita injetar código

4C:8D00 EECF0100	lea r9,qword ptr ds:[14001EE60]	0000000014001EE60:L:"csrss.exe"
4C:8D15 FFCF0100	lea r10,qword ptr ds:[14001EE78]	0000000014001EE78:L:"explorer.exe"
4C:8D25 18D00100	lea r12,qword ptr ds:[14001EE98]	0000000014001EE98:L:"lsas.exe"

Fonte: Elaborado pelo autor

Após a primeira parada na chamada de *OpenProcess*, já se torna possível a identificação do processo a ser invadido.

Figura 31 – Breakpoint alcançada em *OpenProcess* e seus parâmetros na convenção *fastcall*



Fonte: Elaborado pelo autor

Conforme ilustra a figura 31, é possível averiguar os parâmetros passados à chamada *OpenProcess* (respectivamente valores armazenados nos registradores *rcx*, *rdx* e *r8*, via convenção x86-64 bits), que conforme documentação da Microsoft [25] explicita, são correlatos ao desejado acesso ao processo (garantido pela chamada *AdjustTokenPrivileges* com a macro *SeDebugPrivilege* como um de seus parâmetros, conforme analisado anteriormente), booleano que permite capturar processos filhos a partir do original invadido, e o *PID* (identificador único) do processo apropriado pelo executável malicioso. Este último parâmetro se torna o mais interessante de se observar, visto que revela de maneira objetiva o processo a ter seu endereço de memória modificado, e nesse contexto, será aquele de *ID 0xBE0* (3040 em decimal). Com essa informação em mãos, torna-se possível a análise simultânea do processo invadido via *debugger*, antes de sua execução propriamente dita com a função *CreateRemoteThread*. Conferindo no gerenciador de tarefas, o processo com tal *PID* trata-se do *SIHost.exe* (ilustrado pela figura 32), que significa *Shell Infrastructure Host*, sendo uma das componentes próprias do Windows e que tem funções de auxiliar na manutenção da interface gráfica do sistema operacional (GUI) [26], sendo assim uma das razões de se tornar um dos primeiros alvos do *Ryuk*, visto que este percorre uma lista encadeada dos processos ativos no sistema, conforme apresentado anteriormente.

Figura 32 – *S/Host.exe* que deverá ter sua memória sobreescrita por código malicioso

Name	PID	Status	User name	CPU	Memory (a...)	UAC virtualizat...
svchost.exe	2380	Running	Matheus	00	4,460 K	Disabled
svchost.exe	2448	Running	SYSTEM	00	564 K	Not allowed
svchost.exe	2480	Running	LOCAL SE...	00	420 K	Not allowed
svchost.exe	2528	Running	LOCAL SE...	00	60 K	Not allowed
svchost.exe	2544	Running	SYSTEM	00	708 K	Not allowed
svchost.exe	2772	Running	SYSTEM	00	2,467 K	Not allowed
svchost.exe	2784	Running	LOCAL SE...	00	420 K	Not allowed
taskhost.exe	2808	Running	Matheus	00	1,796 K	Disabled
taskhost.exe	2816	Running	Matheus	00	4,020 K	Disabled
ApplicationFrameHo...	2836	Running	Matheus	00	1,796 K	Disabled
shost.exe	3040	Running	Matheus	00	3,684 K	Disabled
svchost.exe	3128	Running	SYSTEM	00	1,832 K	Not allowed
spinmonitor-08.exe	3224	Running	Matheus	00	6,120 K	Disabled
svchost.exe	3244	Running	SYSTEM	00	508 K	Not allowed
svchost.exe	3280	Running	SYSTEM	00	2,080 K	Not allowed
svchost.exe	3368	Suspended	Matheus	00	0 K	Disabled
systemSettings.exe	3482	Running	LOCAL SE...	00	1,188 K	Not allowed
svchost.exe	3684	Running	Matheus	00	10,213 K	Disabled

Fonte: Elaborado pelo autor

Subsequentemente, reexecuta-se o *software* malicioso, até a parada na chamada *WriteProcessMemory*. Esta chamada apresenta detalhes importantes da memória modificada do processo *S/Host*. Conforme se apresentam os parâmetros da função, segundo a *Microsoft* [27] apresenta protótipo descrito pela figura 33, onde são interessantes as entradas *lpBaseAddress*, *lpBuffer* e *nSize*, que representam respectivamente o endereço base a ser escrito no processo invadido, o buffer de dados que serão escritos em memória do processo alvo, e a quantidade de dados em bytes que serão escritos.

Figura 33 – Protótipo da função *WriteProcessMemory*

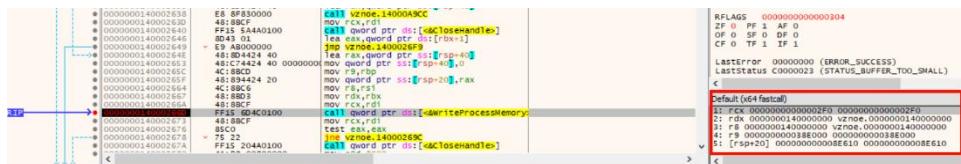
```
BOOL WriteProcessMemory(
    [in] HANDLE hProcess,
    [in] LPVOID lpBaseAddress,
    [in] LPCVOID lpBuffer,
    [in] SIZE_T nSize,
    [out] SIZE_T *lpNumberOfBytesWritten
);
```

Fonte: KARL-BRIDGE(2022) [27]

Conhecendo tais detalhes, ao observar-se a evocação desta função durante execução, são revelados detalhes interessantes acerca do funcionamento *multi-thread* do *Ryuk*. Pelos parâmetros 2 e 3 evocados conforme ilustrado pela figura 34, elucida-se o fato que *lpBaseAddress* e *lpBuffer* tem os mesmos valores, representando assim a característica de que o *ransomware* escreve seu próprio conteúdo (dado pelo endereço do *buffer* ser respectivo ao início de seu cabeçalho) na base do processo invadido, permitindo que este, em sua execução, além da ação de encriptação, possa realizar outros comportamentos como invasão a outros processos e respectiva ação de *threads* remotos, o que explicam o fato da ação maliciosa ocorrer em menos de 5 minutos, devido a capacidade de espalhamento da amostra. Analisando o parâmetro 4, respectiva a quantidade *bytes* a serem sobreescritos no processo alvo, nota-se o valor *0x38E000* (equivalente a 3728 kB), que se comparado a memória alocada para

o *sihost.exe* na figura 32 de 3684 kB, o que leva ao entendimento de que toda a memória do processo invadido é sobreescrita, corroborando o entendimento de espalhamento do código malicioso nos processos invadidos em toda sua extensão de memória, se tornando uma cópia do processo original do *Ryuk*.

Figura 34 – Breakpoint em *WriteProcessMemory* durante processo de debugging

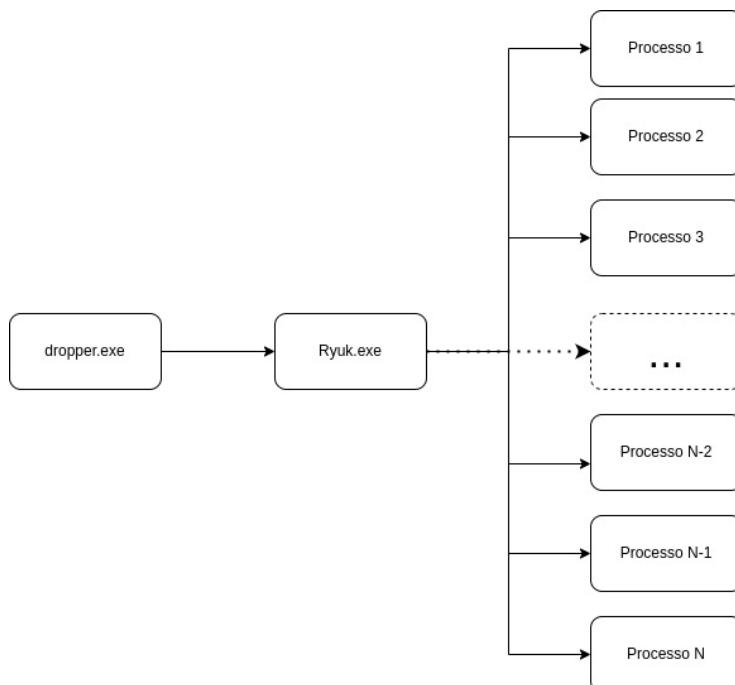


Fonte: Elaborado pelo autor

A constatação anterior explicita a velocidade de atuação do *ransomware*, que durante experimentação apresentou duração de 3 minutos, encriptando todos os arquivos alvos (excluindo de seus objetivos arquivos executáveis e de sistema operacional) nos drives lógicos da máquina virtual.

Dessa forma, constata-se que o *Ryuk* injeta remotamente em processos alvos parte de seu próprio código com objetivo executar remotamente nestes simultaneamente a rotina de encriptação. Esse comportamento, representado graficamente pela figura 35, implica na alta taxa de modificação de arquivos, dificultando assim a identificação por medidas convencionais e dificultando a mitigação deste, devido a divisão da ação em diversos nós remotos na máquina.

Figura 35 – Ilustração gráfica do procedimento *multithread* utilizado pelo *Ryuk*. Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor

Em seguida na análise da ameaça, executa-se novamente o *ransomware* até a *breakpoint* da chamada *CreateRemoteThread*. Ao se observar o escopo desta função [28], vide figura 36, o parâmetro *lpStartAddress* representa o endereço de início de execução do *thread* no processo invadido, isto é, o endereço onde o *thread* criado será iniciado remotamente. Com essa informação torna-se possível também a inspeção do executável *sihost.exe*, que conforme averiguado anteriormente, teve sua porção de memória sobrescrita pelo conteúdo do próprio *Ryuk*.

Figura 36 – Protótipo da função CreateRemoteThread

```
HANDLE CreateRemoteThread(
    [in] HANDLE           hProcess,
    [in] LPSECURITY_ATTRIBUTES lpThreadAttributes,
    [in] SIZE_T           dwStackSize,
    [in] LPTHREAD_START_ROUTINE lpStartAddress,
    [in] LPVOID            lpParameter,
    [in] DWORD             dwCreationFlags,
    [out] LPDWORD          lpThreadId
);
```

Fonte: KARL-BRIDGE(2022) [28]

Doravante, ao captar o valor deste endereço, é possível entender-se as ações que serão tomadas dentro da alocação de memória sobrescrita do processo *sihost.exe*.

Conforme averiguável pela figura 37, nota-se que o parâmetro *lpStartAddress* (o quarto ordinalmente), tem valor *0x140001AD0* armazenado no registrador *r9*, e, portanto, torna-se possível ao se abrir o processo invadido supracitado, o monitoramento paralelo deste, e assim principalmente se entender quais ações são tomadas de forma primária, seja a injeção em outros processos ou a encriptação propriamente.

Figura 37 – Chamada em execução da função CreateRemoteThread

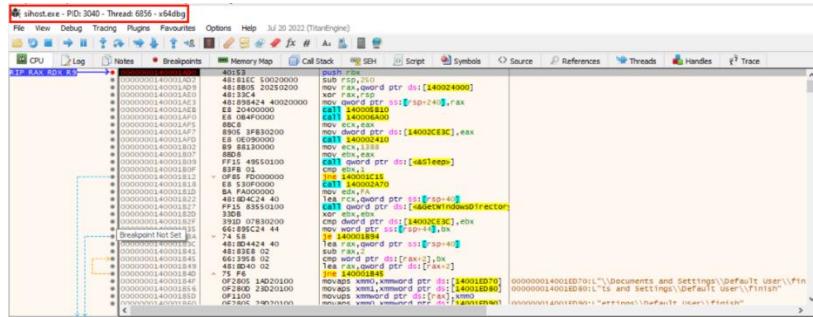


Fonte: Elaborado pelo autor

Buscando acompanhar o monitoramento do *thread* executado remotamente, abre-se uma outra instância do *Xdbg64* e vincula-se sua execução ao do *sihost.exe* de *PID* 3040. Em seguida, estabelece-se um *breakpoint* no endereço de início de execução evidenciado pela figura 36, e voltando a instância do *debugger* vinculada ao processo principal do *Ryuk*, executa-se novamente o processo. Após tal execução, nota-se que a *breakpoint* estabelecida no processo injetado (*Sihost.exe*) foi atingida, ilustrando o início do processo de execução remota, que se caracteriza pela realização de configurações iniciais já realizadas pelo processo original, ilustrando

assim, a mimetização das ações de replicação tomadas pelo *ransomware* e sua característica de replicação de código base em todos os processos invadidos, conforme elucida a figura 38.

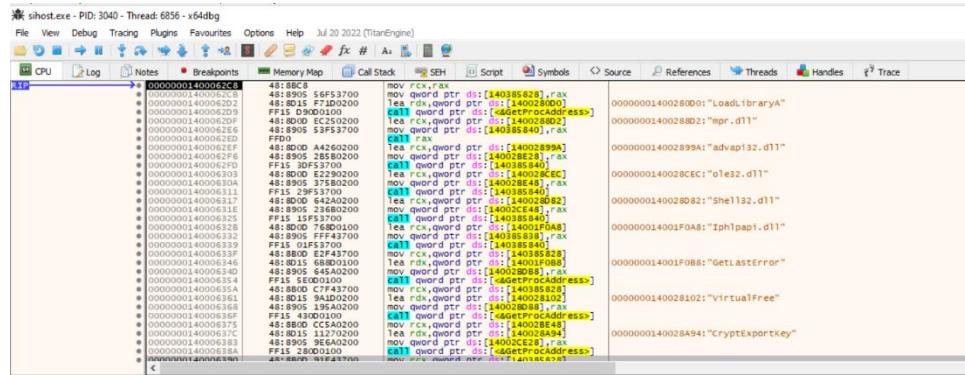
Figura 38 – Outra instância do debugger ligado ao processo invadido, Sihost



Fonte: Elaborado pelo autor

Com intuito de uma análise mais cuidadosa nesta instância, executa-se, inicialmente, instrução a instrução com a opção *step-over*. Após 5 instruções, é chamada a primeira função dentro do processo invadido, localizada no endereço *0x140001AEB*. Esta, após retornar, indica um salto para um bloco de código no qual são realizados os carregamentos dinâmicos de bibliotecas que foram identificadas durante o monitoramento dinâmico (como as relacionadas a encriptação), mas que não eram identificadas dentro do cabeçalho de *imports* do arquivo, em análise estática. Tais informações são valiosas, pois revelam muito do comportamento possivelmente ainda não detectado do ataque, devido a funções importadas que não se tornam detectáveis antes da execução.

Figura 39 – Bloco de instruções onde são realizados carregamentos dinâmicos de funções exportadas



Fonte: Elaborado pelo autor

Estas funções e módulos são alocadas em memória no processo em questão utilizando duas funções principalmente:

- *LoadLibraryA*, responsável pelo carregamento de módulos (*DLL's*) no processo chamado [29];
- *GetProcAddress*, retorna o endereço de funções exportadas de *DLL's* carregadas no processo [30].

A utilização conjunta dessas duas chamadas garante a importação dinâmica dos seguintes módulos:

- mpr.dll
- ole32.dll
- Iphlpapi.dll

Dadas estas *DLL's*, em conjunto as já importadas *Shell32.dll*, *Kernel32.dll* e *Advapi.dll*, funções anteriormente não mapeadas são incluídas no escopo de importação do processo, permitindo chamadas posteriores neste. Estas têm por funcionalidades principais a encriptação de arquivos, execução de comandos remotamente, alocação de memória e gerenciamento de arquivos. As principais chamadas são destacadas abaixo:

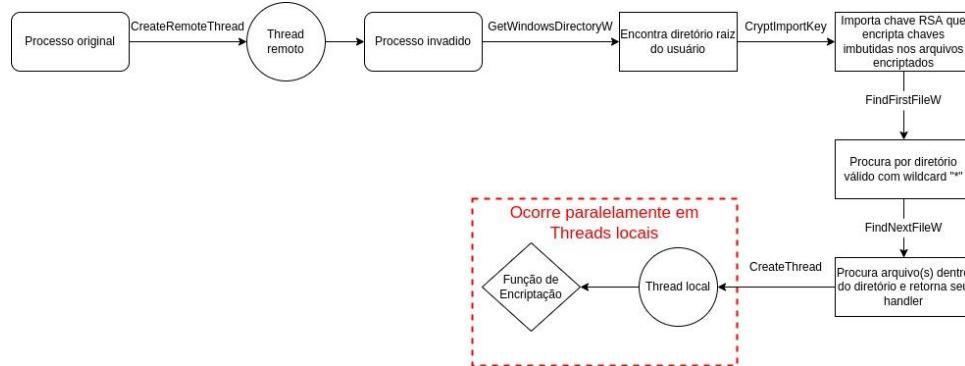
- GetLastError
- VirtualFree
- CryptExportKey
- DeleteFileW
- GetDriveTypeW
- GetCommandLineW
- GetStartupInfoW
- FindNextFileW
- VirtualAlloc
- GetUserNameA
- ExitProcess
- Wow64RevertWow64FsRedirection
- CreateProcessA
- GetVersionExW
- Wow64DisableWow64FsRedirection
- GetSystemDefaultLangID
- GetUserNameW
- ReadFile

- RegQueryValueExA
- CloseHandle
- RegSetValueExW
- RegCloseKey
- CopyFileA
- SetFileAttributesW
- WinExec
- CryptDeriveKey
- CryptGenKey
- Sleep
- GetCurrentProcess
- ShellExecuteW
- GetFileSize
- GlobalAlloc
- FindClose
- GetModuleFileNameA
- ShellExecuteA
- GetModuleHandleA
- GetModuleFileNameW
- CreateFileA
- GetFileSizeEx
- WriteFile
- GetLogicalDrives
- WNetEnumResourceW
- RegOpenKeyExW
- WNetCloseEnum
- GetWindowsDirectoryW
- SetFileAttributesA
- RegOpenKeyExA
- SetFilePointer
- GetTickCount
- GetFileAttributesW

- FindFirstFileW
- CryptAcquireContextW
- WNetOpenEnumW
- CryptDecrypt
- CryptImportKey
- SetFilePointerEx
- CopyFileW
- CreateProcessW
- CreateDirectoryW
- CreateThread
- CryptDestroyKey
- CoCreateInstance
- CreateFileW
- GetFileAttributesA
- CryptEncrypt
- RegDeleteValueW

Dentre as importadas acima, nota-se a presença das já detectadas funções do módulo *Bcrypt.dll* (Windows Crypto API), essencial para tarefas criptográficas do sistema operacional em questão; funções de enumeração de arquivos e diretórios (*FindFirstFileW*, *FindNextFileW* e *GetWindowsDirectoryW*) e funções de criação de threads locais (*CreateThread*) e remotos (*CreateRemoteThread*). Essas importações são deveras importantes pois são o esqueleto principal do processo de enumeração de arquivos e encriptação decorrido, podendo ser visualizado esquematicamente pela figura 40:

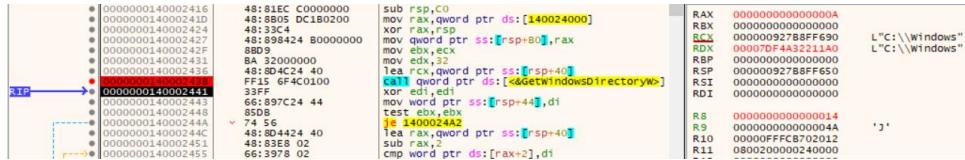
Figura 40 – Esquematização do processo de enumeração e encriptação de arquivos.



Fonte: Elaborado pelo autor

Conforme a figura esquematiza, após a invasão do processo original injetando um *thread* remoto no processo invadido, este encontra o diretório raiz do Windows.

Figura 41 – Função de obtenção do diretório raiz do Windows na máquina invadida



Fonte: Elaborado pelo autor

Em seguida, utilizando a função *CryptImportKey*, é importada uma chave RSA que aparenta estar embutida estaticamente no executável. Esta chave será importante posteriormente para o *Ryuk* encriptar chaves AES geradas dinamicamente para cada arquivo embutidas também nestes, sendo assim, uma possível estratégia utilizada pelos cibercriminosos para descriptografiação (mediante processo reverso) dos arquivos, caso a vítima realize o pagamento.

Figura 42 – Chamada da função CryptImportKey



Fonte: Elaborado pelo autor

Figura 43– Arquétipo da função *CryptImportKey* e seus parâmetros

```

BOOL CryptImportKey(
    [in] HCRYPTPROV hProv,
    [in] const BYTE *pbData,
    [in] DWORD dwDataLen,
    [in] HCRYPTKEY hPubKey,
    [in] DWORD dwFlags,
    [out] HCRYPTKEY *phKey
);
  
```

Fonte: ALVINASHCRAFT (2021) [31]

Figura 44– Chave RSA importada pela função

Address	Hex	ASCII
00000001400293D0	06 02 00 00 00 A4 00 00 52 53 41 31 00 08 00 00	...RSA1...;7U ...kôAVÀ...;7U
00000001400293E0	01 00 01 00 90 68 F2 C1 56 C1 BF 3E 00 60 37 DA	...@.0.01...\$
00000001400293F0	02 88 B3 A6 A9 51 5F D6 8E 6F EC 02 97 93 18 24	030ca.O..ôææSER,
0000000140029400	F4 33 30 63 61 00 4F 82 0D F0 E6 E0 33 C8 52 9D	ôWéçU'W'V'.P'
0000000140029410	14 77 C9 BF C6 D6 B9 77 B3 76 8E 2A 89 6C 50 B4	â_P_.U._S.eõI.
0000000140029420	E2 36 5F D0 A6 AD 06 89 5E 73 1E 65 F5 CF 01	E5 F8 29 EO
0000000140029430	0C C9 29 78 4C F1 F9 C8 2F 9F D9 40 E5 F8 29 EO	(LauÉ/.Uéæo)a
0000000140029440	C5 27 EC DF 90 F9 43 11 C2 98 89 00 11 5F 13 1B	Á'18.Uç.A'
0000000140029450	25 55 51 5E EF EA 04 5D FB C4 C1 26 3F FA 2F	%U.Aijj.JüAA&ü/
0000000140029460	91 6C 15 6C 00 F4 A4 EB E5 B9 49 B2 04 47 75 FA	.1.1.ôkå'K.GuÙ
0000000140029470	A4 2E 11 9A 58 EA 93 49 41 7D 6E D7 16 77 23 A0	...Xê.I}nx.w#
0000000140029480	D9 67 B9 83 98 72 A3 13 CA BE 8A BE F4 D8 CE 6F	Ug...r£.E...ôôôto
0000000140029490	F0 22 B8 78 FE 4E 91 87 79 A8 73 C2 91 F3 C4 0B	û'xpñ..y.sA.oA.
00000001400294A0	08 12 28 16 62 77 91 D9 19 21 4A F3 61 FD 08 3C	..+Dw.Y.!Joy.<
00000001400294B0	3A 25 70 88 71 45 78 17 51 25 19 04 0B A5 2D 95	:Sp.QEx.Q%..¥.
00000001400294C0	3B FB A1 B2 E2 E7 96 3B 49 C0 C2 38 4D B7	8ùi...äc.;;IAÄSM-
00000001400294D0	67 31 17 86 88 C4 65 18 7B 87 37 D6 8A 2A 14 2C	gi...Äe.{.70.%
00000001400294E0	B0 C1 4A A4 00 00 00 00 73 74 61 72 74 20 22 22	'Aj...start ""

Fonte: Elaborado pelo autor

Conforme apresenta o arquétipo da função *CryptImportKey*, ilustrado pela figura 43, o segundo parâmetro (ponteiro *phData*) desta trata-se de uma array de Bytes contendo um header BLOB chamado PUBLICKEYSTRUC (“RSA1” na figura

43), seguido pela chave criptográfica, no caso desta em específico, ela não se encontra criptografada, por isso o quarto parâmetro (uma chave para descriptografar a chave importada) é nula [31]. O endereço da chave é indicado no segundo parâmetro da função evocada (guardada no registrador RDX), *figura 42*, com valor de hexadecimal 0x1400293D0, que elucidado pela *figura 44*, mostra o arquivo texto *BLOB* da chave.

Em seguida, o software malicioso inicia a busca por arquivos a partir do diretório base do volume **C://** do Windows, com wildcard “*” para procurar pela grande totalidade dos arquivos. Essa função irá iterar sobre os diretórios desde o volume **C://** até encontrar o primeiro arquivo válido dentro de algum subdiretório, e a partir deste, procurar por outros análogos no mesmo subdiretório para o processo de encriptação.

Figura 45– Chamada de função *FindFirstFileW* para achar primeiro arquivo dentro do caminho C://*



Fonte: Elaborado pelo autor

Posteriormente, ao encontrar um diretório válido com arquivos com a qual o processo invadido possui permissões para abrir e modificar, este iniciará o procedimento de enumerar os outros arquivos dentro deste mesmo diretório e criar um *Thread* local para cada arquivo. Dessa forma, nota-se que o executável tende a criar um loop de iteração sobre os arquivos em cada diretório válido encontrado, e dado um arquivo alvo válido para a encriptação, inicia-se a execução do procedimento criptográfico deste, explicando a enorme velocidade de encriptação de arquivos que o *ransomware* possui, além de tornar inviável que aplicações antivírus consigam detectar e parar, em tempo de execução, esta ameaça.

Figura 46– Chamada de função *CreateThread* para iniciar processo de encriptação do arquivo *aclapi.h*



Fonte: Elaborado pelo autor

Conforme perceptível pela *figura 46*, ao encontrar um arquivo dentro de um diretório válido, inicia-se a criação de um *Thread* local no processo para realização do procedimento criptográfico neste, e no *Thread* principal, segue-se com a enumeração de arquivos e diretórios válidos, conforme visto pela chamada da função

FindNextFileW, que dado um arquivo dentro de um diretório, procurará referências para outros arquivos dentro deste mesmo diretório ou subdiretórios deste [32]. Ademais, ao analisar melhor o arquétipo da função *CreateThread* (figura 47), dois parâmetros são importantes para entendermos sua utilização neste contexto:

- *lpStartAddress*, que indica o endereço da função (rotina) que será executada pelo thread, em específico será a rotina de encriptação do arquivo;
- *lpParameter*, que indica os parâmetros a serem passados à rotina, em específico será um ponteiro para o arquivo a ser encriptado [33].

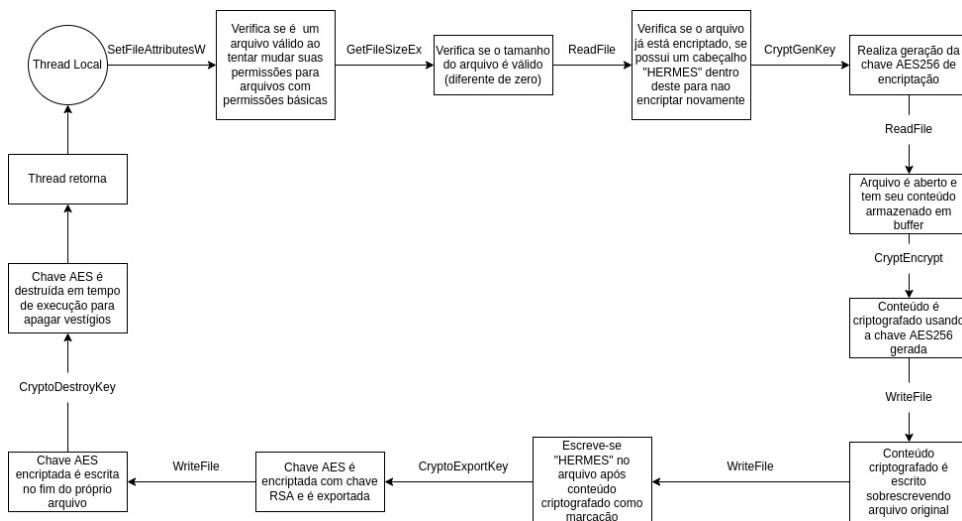
Figura 47– Arquétipo da função *CreateThread*

```
HANDLE CreateThread(
    [in, optional] LPSECURITY_ATTRIBUTES lpThreadAttributes,
    [in]           SIZE_T             dwStackSize,
    [in]           LPTHREAD_START_ROUTINE lpStartAddress,
    [in, optional] _drv_aliasesMem LPVOID lpParameter,
    [in]           DWORD            dwCreationFlags,
    [out, optional] LPDWORD          lpThreadId
);
```

Fonte: KARL-BRIDGE (2022) [32]

Dessa forma, já sabemos que o parâmetro *lpParameter* será preenchido pelo arquivo iterado pelas funções *FindFirstFileW* ou *FindNextFileW*, assim, nota-se que ao se seguir o endereço especificado no parâmetro *lpStartAddress*, será indicada a rotina onde a encriptação propriamente dita ocorre. Tal rotina, em específico, tem seu comportamento indicado pela figura 48.

Figura 48– Esquematização da rotina de encriptação. Fonte: Elaborado pelo autor.



Fonte: Elaborado pelo autor

O primeiro procedimento realizado pelo *ransomware* na rotina de encriptação é a verificação de permissão do arquivo iterado, no qual é realizada uma alteração forçada de permissões do arquivo, alocando como permissão padrão do sistema operacional. Caso o arquivo seja protegido, a função *SetFileAttributesW* retornará

erro, e o Thread retornará sem sucesso. Todavia, caso seja um arquivo sem restrições de permissões, como a maioria de arquivos de diretório comuns, a rotina segue para seu próximo procedimento [34].

Em seguida, é realizada a verificação do tamanho do arquivo com a chamda *GetFileSizeEx*, inicialmente como verificação de arquivo nulo, na qual é encerrada a rotina, tal qual para alocação de memória para o *buffer* que conterá o conteúdo do arquivo em execução. Realizada a verificação de tamanho do arquivo em bytes, o handler do arquivo é finalmente aberto via função *ReadFile*, onde posteriormente é realizada uma verificação se a string “HERMES” está embutida. Tal verificação se faz necessária pois este é um dos passos de finalização de encriptação utilizada pelo ransomware Hermes, do qual o Ryuk descende [35], onde tal string é embutida ao final de arquivos encriptados para evitar uma nova encriptação por parte do *Ryuk*, otimizando a operação do software malicioso em termos de tempo e eficiência.

Figura 49 – Verificação da presença da string “HERMES” no arquivo

```

0000000140003C9A 48:8BCB FF1E 96910200
0000000140003CA0 85C0
0000000140003CA1 75 0A
0000000140003CA2 B8 04000000
0000000140003CA3 E9 62070000
0000000140003CB0 8BEC
0000000140003CB1 8B3D0
0000000140003CB2 8BCF
0000000140003CB3 48:8D45 C1
0000000140003CB4 48:85D2
0000000140003CC1 74 23
0000000140003CC2 8B45 48
0000000140003CC3 75 1D
0000000140003CC4 8038 45
0000000140003CC5 75 18
0000000140003CD1 8078 01 52
0000000140003CD2 75 12
0000000140003CD3 75 07 02 4D
0000000140003CD4 75 0C
0000000140003CD5 8078 03 45
0000000140003CD6 75 06
0000000140003CE1 8078 04 53
0000000140003CE2 75 2A
0000000140003CE3 FFC1
0000000140003CE4 48:FFC0
0000000140003CE5 83F9 14

    mov rcx,rbx
    test eax,eax
    jne 140003C8B
    mov eax,4
    jmp 14000441A
    mov rdx,rcx
    mov ecx,ed1
    lea rax,qword ptr ss:[rbp-3F]
    test rdx,rdx
    je 140003C99
    cmp byte ptr ds:[rax-1],48
    jne 140003C99
    cmp byte ptr ds:[rax],45
    jne 140003C99
    cmp byte ptr ds:[rax+1],52
    jne 140003C99
    cmp byte ptr ds:[rax+2],4D
    jne 140003C99
    cmp byte ptr ds:[rax+3],45
    jne 140003C99
    cmp byte ptr ds:[rax+4],53
    jne 140003D13
    inc rcx
    inc rax
    cmp ecx,14

48: 'H'
45: 'E'
52: 'R'
4D: 'M'
45: 'E'
53: 'S'

Default (x64 fastcall)
1: rcx 00000000000000000000000000000000
2: rdx 00000000000000000000000000000000
3: r8 0000000000000001 0000000000000001
4: r9 000000920387FB90 000000920387FB90

```

Fonte: Elaborado pelo autor

Posterior as verificações do conteúdo do arquivo, é gerada a chave de criptografia a ser utilizada para encriptação do arquivo em questão. É gerada então uma chave que utiliza o algoritmo de encriptação AES 256, conforme o segundo parâmetro da chamada *CryptoGenKey* [36], *AlgId* (identificador único do algoritmo utilizado), indicado pelo valor de *0x6610*, conforme documentação oficial da Microsoft [37]. Este algoritmo, denominado Advanced Encryption Standard de 256 bits é muito utilizado para fins maliciosos visto que é do tipo simétrico, isto é, pode ser reversível caso se possua a chave original de encriptação (facilitando fins de chantagem), além deste ser inquebrável via força bruta por computadores atuais [38].

Figura 50 – Chamada função *CryptGenKey*

```

0000000140003D59 48:8BC5 FF1E 59800200
0000000140003D5A 85C0
0000000140003D5B 75 1E
0000000140003D5C 48:8BCB
0000000140003D5D FF15 D4900200
0000000140003D5E 48:8B4C24 40
0000000140003D5F FF15 B1479800

    mov rcx,r14
    qword ptr ds:[<>CryptGenKey>]
    test eax,eax
    jne 140003D5E
    mov rcx,rdx
    qword ptr ds:[<>CloseHandle>]
    mov rcx,qword ptr ss:[rsp+4]
    qword ptr ds:[<>CryptDestroyKey>]

Default (x64 fastcall)
1: rcx 00000000000000000000000000000000
2: rdx 00000000000000000000000000000000
3: r8 0000000000000001 0000000000000001
4: r9 000000920387FB90 000000920387FB90

```

Fonte: Elaborado pelo autor

Após a geração da chave, é realizada novamente a abertura do conteúdo do arquivo e armazenamento deste em um *buffer* via *ReadFile*. Com o conteúdo carregado em memória é realizada prontamente a encriptação desta via função *CryptEncrypt*, que utilizando a chave gerada pelo passo anterior cifra o conteúdo conforme algoritmo base [39]. Após conteúdo ser cifrado, este sobrescreve o conteúdo original do arquivo com a chamada *WriteFile*, que com o handler do arquivo original, escreve o conteúdo do *buffer* preenchido pela função *CryptEncrypt*.

Figura 51 – Chamada função *CryptEncrypt*

Fonte: Elaborado pelo autor

Com o arquivo encriptado, são realizados procedimentos de finalização do procedimento, com intuitos aparentes de facilitar uma recuperação dos arquivos, mediante extorsão, e limpeza de rastros. Após o arquivo ter seu conteúdo cifrado, é escrito logo em seguida ao conteúdo escrito anteriormente, a string “HERMES” via *WriteFile*, que conforme averiguado em passos anteriores, tende a indicar que um arquivo já está encriptado, evitando desta forma cifragens sucessivas sobre o mesmo arquivo.

Figura 52 – Escrita da string “HERMES” dentro arquivo criptografado

```
b 00000010400041A9 48:8D55 88 lea    rdx,[word ptr ss:[rdp-78]]  
b 00000010400041B0 48:8CCE mov    rdx,rax  
b 00000010400041B0 FF15 D23D3800 call   dword ptr ds:[<#WriteToFile>]  
b 00000010400041B6 85C0 test   eax,eax  
b 00000010400041B8 75 2F je    14000041E9  
b 00000010400041B8 3D2D xor    edx,edx  
b 00000010400041B8 41:B8 00800000 mov    r8d,8000  
b 00000010400041C2 48:8CCE mov    rcx,rs1  
  
Default [x64 fastcall]  
1: rcx 00000000000000E58 00000000000000E58  
2: rdx 0000009203B7F968 0000009203B7F968 "HERMES"  
3: r8 0000000000000000 0000000000000000  
4: r9 0000009203B7F934 0000009203B7F934  
5: [rsp+20] 0000000000000000 0000000000000000
```

Fonte: Elaborado pelo autor

Em seguida, é realizado o procedimento de exportação da chave gerada com a chamada *CryptExportKey*, que conforme explicitado pela documentação da Microsoft [40], exporta uma chave criptográfica encriptada, primeiro argumento, encriptada por uma outra chave, segundo argumento. Nota-se então o uso da chave RSA, importada anteriormente, realizando o processo conhecido por Encriptação envelope [41], procedimento muito utilizado para proteção de dados e conhecido por ser uma segunda barreira contra quebra de cifragem. Após a exportação desta para um *buffer* em memória, a chave AES cifrada é também escrita no arquivo (*WriteFile*), sendo utilizada possivelmente para recuperação deste.

Figura 53 – Chamada da função *CryptExportKey*

```
00000000140004203 48:897C4 20 mov qword ptr ss:[rsp+20],rdi  
00000000140004204 48:BD41 01 lea r8d,qword ptr ds:[r9+1]  
00000000140004205 F8E0 DE8E0200 test rax,rax  
00000000140004212 85C0 xor rax,rax  
00000000140004214 75 2F jne 140004245  
00000000140004216 33D2 xor edx,edx  
00000000140004218 41:BB 08000000 mov r8d,8000  
0000000014000421E 48:B8E0 mov rcx,rsi
```

Default (x64 Fastcall)

```
1: rcx 0000023CC044E9B0 <_CPGenKey> (0000023CC044E9B0)  
2: rdx 00023CC017B5B0 <_CPGenKey> (0000023CC017B5B0)  
3: r8 0000000000000001 0000000000000001  
4: r9 0000000000000000 0000000000000000  
5: [rsp+20] 0000000000000000 0000000000000000  
6: [rsp+28] 000000000203BF940 000000000203BF940
```

Fonte: Elaborado pelo autor

Por fim, após escrita da chave no arquivo, é realizada a destruição dessa chave durante *run-time* com a função *CryptoDestroyKey*, que conforme nome explicita realiza a destruição para o handler da chave criptográfica utilizada, neste contexto, referente a chave AES 256 gerada. Esse procedimento ocorre principalmente para esconder rastros do *ransomware* e dificultar o processo de quebra de cifra por softwares de recuperação. Após a destruição da chave o *Thread* local retorna, encerrando assim a encriptação do arquivo. Esta rotina é repetida para todos os arquivos dentro de diretórios válidos, e ao ser replicado em diversos processos, cada qual utilizando múltiplos threads para cada arquivo, ação maliciosa da amostra *Ryuk* tem comportamento extremamente acelerado no alvo atingido.

Figura 54 – Retorno do Thread local

```

00007FFE5BFE7034 8BC8
00007FFE5BFE7036 48:FF15 6BB90600 mov    ecx,eax
00007FFE5BFE703D 0F1F4400 00 call   qword ptr ds:[<&RtlExitUserThread>]
00007FFE5BFE7042 CC                nop    dword ptr ds:[rax+rax],eax
00007FFE5BFE7043 48:FF15 E6C00600 int3
00007FFE5BFE7044 0F1F4400 00 call   qword ptr ds:[<&RtlGetSuiteMask>]
00007FFE5BFE704A 0040              nop    dword ptr ds:[rax+rax],eax

```

Fonte: Elaborado pelo autor

2.4.5 Matriz de ataque da ameaça

Destarte, após diferentes crivos de análise do ataque, pode-se utilizar de maneira sumária como este ocorre seguindo o padrão de ataque representado pela *tabela 1*, que segundo o padrão Mitre [42], consegue elucidar os pontos principais utilizado pelo *Ryuk* durante o ataque.

Tabela 1 – Matriz Mitre de ataque do Ryuk

Acesso inicial	Execução	Persistência	Escalação de privilégios	Evasão de defesa	Discovery	Impacto
Phishing (emails maliciosos)	API nativa	Chaves de registro (regedit)	Injeção em processos	Correspondência a nomes e localizações reais	Discovery de configuração de sistema da rede	Serviços desativados
Engenharia social	CMD do Windows	Contas válidas	Chaves de registro (regedit)	Injeção em processos	Enumeração de arquivos e diretórios	Encriptação de dados
-	-	-	Contas válidas	Modificar ou desativar ferramentas	Discovery de processos	Desativar recuperação de sistema
-	-	-	Manipulação de token de acesso	Contas válidas	-	-
-	-	-	-	Manipulação de token de acesso	-	-

Fonte: MANOCHA (2022) [42]

Esta representação é essencial para um entendimento amplo e conciso da ameaça, entendendo seus padrões de atuação, podendo assim facilitar na mitigação e detecção desta em outros ambientes, facilitando a resposta a incidentes dentro do ambiente invadido.

2.5 Criando contramedidas

Após entendimento estático e dinâmico da amostra, gerando discernimento acerca dos padrões dos padrões de ataque comuns, é possível o início da criação de contramedidas, isto é, a prototipação de soluções que visem utilizar as informações captadas em análises de *Malware* (no caso em contexto tratando-se de um *Ransomware*) para detecção e mitigação destas ameaças de maneira automatizada.

Neste concernente, foi idealizada a criação de um software, ilustrativamente chamado “Atreus”, que munido com as informações do ataque *Ryuk*, pode identificar e suspender o ataque, reduzindo significativamente os danos causados pelo amostra.

Destarte, baseado nas principais características da amostra captadas durante as análises estáticas e dinâmicas, a estrutura de defesa do software é apresentada nos seguintes pilares:

- Monitoramento de chamadas da função *CreateRemoteThread*, que em geral tendem a indicar a tentativa de *Process Injection* por meio de ameaças, excluindo da busca processos que em geral tendem a utilizar essa chamada por padrão, evitando a detecção de falsos positivos.
- Escaneamento de amostras executáveis no computador que apresentem strings como as detectadas na análise estática, arquiteturas de sistema parecidos (32 bits para o *dropper* e 64 bits para o *payload*) e funções (advindas de DLLs) importadas similares às utilizadas pelo *Ryuk*;
- Monitoramento de alterações de registro do Windows que indiquem a criação de meios de persistência da ameaça;

Ao rastrear estes comportamentos torna-se possível mitigar danos da ação proposta pelo *ransomware*, facilitando a tomada de resposta ao incidente na máquina invadida. Para este fim de rastreamento, sobretudo em relação a chamadas da função *CreateRemoteThread* (*API Tracing*), é necessário a utilização de ferramentas que integrem adequadamente no sistema operacional Windows, possibilitando com maior confiabilidade e velocidade o monitoramento de processos que realizem chamadas a esta função. Para a alçada, já existem bibliotecas *open-source* disponibilizadas pela

Microsoft, que facilitam o monitoramento em tempo real, visto que são integrados ao sistema operacional. Dentre as mais famosas soluções disponíveis, destacam-se o *Microsoft Detours* [43] e *Sysinternals* [44]. A primeira trata-se de uma biblioteca com fins de interceptação de código, buscando principalmente pelo rastreamento (*tracing*) de funções da Win32API em tempo de *runtime*, enquanto a segunda trata-se de um conjunto de ferramentas que oferecem funcionalidades de rastreio conjunto de funções e interações com o sistema operacional, inspecionamento de processos e arquivos.

Neste concernente como *Sysinternals* oferece um conjunto maior de interações com o sistema operacional em diversas frentes, utiliza-se algumas de suas funcionalidades para rastreamento da função *CreateRemoteThread*, via módulo *Sysmon* [45], excluindo neste caso, processos que comumente utilizam esta função, tal qual como browsers ou gerenciadores do host do sistema operacional, reduzindo a quantidade de falsos positivos detectados.

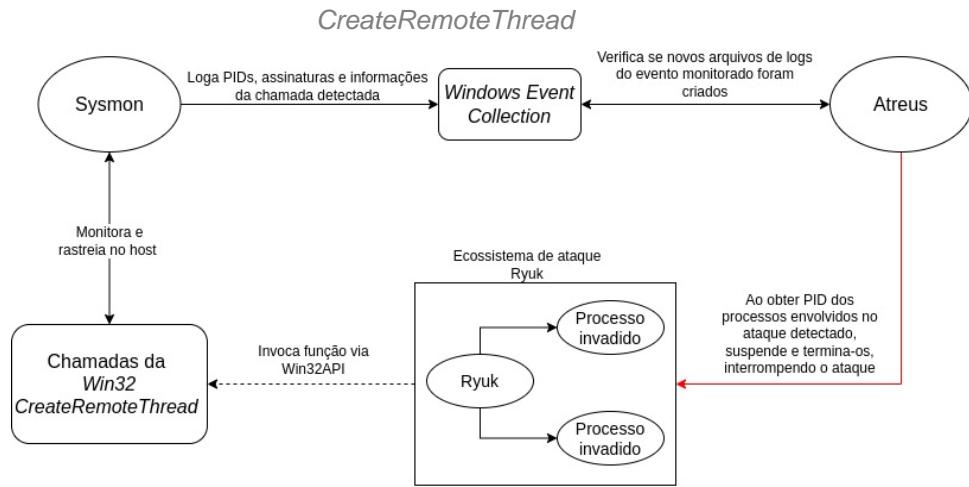
Figura 55 – Arquivo de configuração do Sysmon para detecção de *CreateRemoteThread* e evitando processos comuns

```
<CreateRemoteThread onmatch="exclude">
    <SourceImage condition="is">C:\Windows\system32\wbem\WmiPrvSE.exe</SourceImage>
    <SourceImage condition="is">C:\Windows\system32\svchost.exe</SourceImage>
    <SourceImage condition="is">C:\Windows\system32\wininit.exe</SourceImage>
    <SourceImage condition="is">C:\Windows\system32\cssrss.exe</SourceImage>
    <SourceImage condition="is">C:\Windows\system32\services.exe</SourceImage>
    <SourceImage condition="is">C:\Windows\system32\winlogon.exe</SourceImage>
    <SourceImage condition="is">C:\Windows\system32\audiiodg.exe</SourceImage>
    <StartModule condition="is">C:\Windows\system32\kernel32.dll</StartModule>
    <TargetImage condition="is">C:\Program Files (x86)\Google\Chrome\Application\chrome.exe</TargetImage>
</CreateRemoteThread>
```

Fonte: Elaborado pelo autor

Conforme a figura 55 apresenta, muitos processos e drivers comuns de utilização do sistema operacional são excluídos para não afetar performance e funcionamento adequado do host, tal qual de softwares comuns. Após criação do arquivo de configuração da ferramenta, é necessário criar um mecanismo de verificação de detecção, visto que o *Sysmon* ao detectar eventos não é capaz de criar contramedidas por padrão, se restringindo a apenas criar logs por meio do *Windows Event Collection*. Destarte, para utilização destes logs como um artefato de mitigação, cria-se um *daemon*, cuja função seja fazer queries constantes no *namespace* '*Microsoft-Windows-Sysmon/Operational*', verificando novos arquivos detectados, e com as informações coletadas tomar ação sobre os processos envolvidos, tanto o processo fonte, tal qual os processos alvos, possibilitando assim a diminuição do processo de encriptação realizado pelo Ryuk.

Figura 56 – Arquétipo de funcionamento do *daemon* Atreus no rastreamento de chamadas



Fonte: Elaborado pelo autor

Conforme ilustrado pela figura 56, o software Atreus ao detectar qualquer novo log suspeito realizado pelo Sysmon, poderá executar em tempo real medidas de controle contra os processos suspeitos, interrompendo simultaneamente todos os processos envolvidos, tal qual o processo original a partir do executável *Ryuk* tanto pelos invadidos na iniciativa multi-thread do ataque.

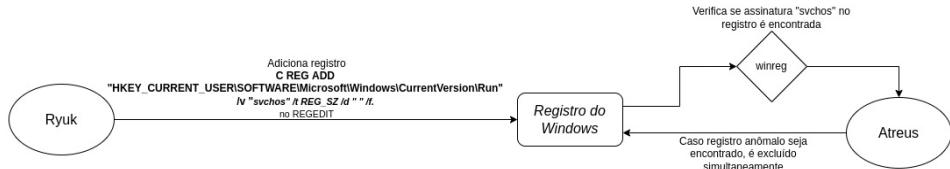
Concomitantemente a suspensão dos processos envolvidos, pode-se realizar a captura de informações destes processos como arquivos abertos por estes, DLL's importadas, informações de *runtime* e até realizar *dumps* de processos, gerando detalhamento sobre estes que podem ser utilizados para auditorias e iniciativas de resposta ao incidente posteriores. Neste interim, utiliza-se as seguintes ferramentas do conjunto *Sysinternals*:

- *Procdump*, para realização de dumps de um processo desejado [46];
- *Handle*, para verificar handles de arquivos abertos pelo processo investigado [47];
- *ListDLLs*, com intuito de verificar DLL's importadas pelo processo [48];
- *PSlist*, para detalhamento de características operacionais do processo investigado, como número de threads, handles abertos, tempo de CPU e tempo de execução [49];

Com estas utilidades torna-se eficaz também a coleta de informações sobre o ataque realizado, possibilitando melhor detalhamento do ataque pela obtenção de logs. Analogamente, é também necessário o rastreamento de mudanças realizadas no registro do Windows, realizado pelo *Ryuk* com fins de persistência da ameaça. Para

tal, pode-se utilizar a biblioteca “*winreg*” do Windows disponibilizada para linguagem de programação Python, para realizar buscas no registro. Desse modo, caso seja detectado alguma adição no registro que siga o seguinte padrão já observado em ataques *Ryuk*, na sessão de análise estática, o *daemon* Atreus deve ser capaz de remover imediatamente tal registro, impedindo a persistência da amostra na máquina, impedindo uma vertente de ataque comum da amostra.

Figura 57 – Ilustração de rastreamento de registro realizado pelo Atreus



Fonte: Elaborado pelo autor

Similarmente, é necessária uma busca automatizada por arquivos que ainda não executados, que por vezes podem ser encontrados em máquinas vítimas, esperando algum gatilho de execução. Desse modo, encontrar arquivos executáveis suspeitos antes de sua execução é essencial para também realizar uma proteção adequada contra ataques de *Ransomware*. Deste modo, utiliza-se duas ferramentas para detecção de arquivos estáticos na máquina alvo:

- Regras Yara, que de acordo com padrões, strings estáticas, funções, DLL's e composições de bytes, pode reconhecer amostras de *Malware* [50], além de possuir integração nativa com linguagem de programação Python;
- *Sigcheck*, ferramenta do arsenal *Sysinternals*, que pode ser integrado ao portal web *Virustotal*, para rastreamento de arquivos em tempo real, procurando por amostras reportadas no portal [51].

Utilizando ambas as ferramentas em conjunto, iterando diretórios comumente utilizados pelo *Ryuk* para alocação de seu arquivo *dropper*, como desktop ou pasta de downloads, tal qual para seu executável padrão, que conforme análise dinâmica, se localiza no caminho **C:/users/public**.

Com caminhos de pastas frequentemente utilizados pela ameaça, é necessário construir regras Yara que sejam adequadas para detecção dos arquivos *Ryuk*, ou seja, utilizar informações advindas da análise estática de ambos os arquivos para detecção fidedigna destes. Deste modo, pode-se reunir informações importantes para detecção, em primeiro plano, do arquivo *dropper* tais como:

- Estrutura de arquivo PE;
- Arquitetura 32 bits;
- Ter como funções importadas *IsDebuggerPresent* e *ShellExecuteW*;
- Strings estáticas utilizadas, tais como as utilizadas na construção do ransom note, adição do registro de persistência e exclusão de backups.

De posse destas informações, torna-se possível a construção de uma regra Yara com intuito mor de detecção do arquivo *dropper*, conforme ilustra a figura 58.

Figura 58 – Regra Yara usada para detecção do arquivo *dropper* do Ryuk

```
rule ryuk_dropper{
    meta:
        description = "Aim to detect Ryuk dropper"
    strings:
        $s1 = "REG ADD" wide
        $s2 = "\"HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\" wide
        $s3 = "/v \\svchost\" wide
        $s4 = "/t REG_SZ" wide
        $s5 = "/d \\\" wide
        $s6 = "No system is safe" ascii wide
        $s7 = "vssadmin resize shadowstorage" ascii wide
        $s8 = "UNIQUE_ID_DO_NOT_REMOVE" ascii wide
    condition:
        pe.is_pe and
        all of them and
        pe.is_32bit() and
        pe.imports("Kernel32.dll", "IsDebuggerPresent") and
        pe.imports("Shell32.dll", "ShellExecuteW")
}
```

Fonte: Elaborado pelo autor

Concomitantemente, é necessário estabelecer comportamentos comuns encontrados no arquivo principal do *ransomware*, tais quais:

- Estrutura de arquivo PE;
- Importação de funções *CreateRemoteThread*, *WriteProcessMemory*, *GetProcAddress* e *LoadLibraryA*;
- Strings estáticas utilizadas, tais quais como início da chave de encriptação RSA1, nome do arquivo ransom note, comandos utilizados para interromper serviços e adição do registro de persistência.

Novamente, com as informações obtidas na análise estática torna-se capaz a criação de uma regra Yara para a detecção do executável principal do *Ryuk*, conforme elucidado pela figura 59:

Figura 59 – Regra Yara usada para detecção do arquivo *principal* do Ryuk

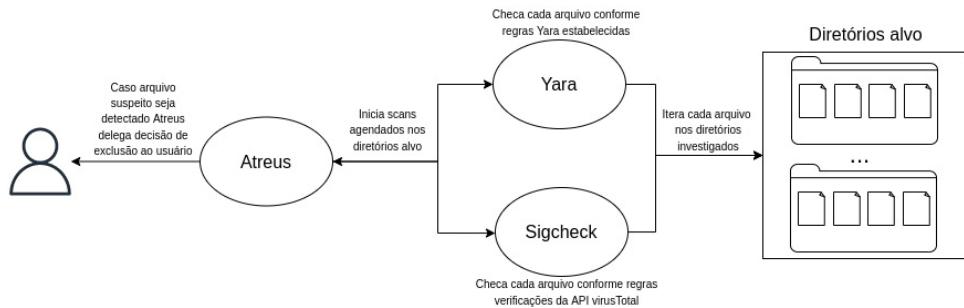
```
rule ryuk_exe{
    meta:
        description = "Aim to detect Ryuk executable"
    strings:
        $s1 = "REG ADD" wide
        $s2 = "\\"HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run\\\" wide
        $s3 = "/v \"svchost\" wide
        $s4 = "/t REG_SZ" wide
        $s5 = "/d \\" wide
        $s6 = "RyukReadMe.txt" wide
        $s7 = "SeDebugPrivilege" wide
        $s8 = "stop Antivirus /y" wide ascii
        $s9 = "RSA1" wide ascii

    condition:
        pe.is_pe and
        all of them and
        pe.imports("Kernel32.dll", "CreateRemoteThread") and
        pe.imports("Kernel32.dll", "WriteProcessMemory") and
        pe.imports("Kernel32.dll", "GetProcAddress") and |
        pe.imports("Kernel32.dll", "LoadLibraryA")
}
```

Fonte: Elaborado pelo autor

Com as regras Yara devidamente ajustadas, torna-se necessário o ajuste do *daemon* para a iteração dos diretórios já descritos como possíveis alvos de alocação do *Ransomware*, e utilizar tanto o *Sigcheck* quanto Yara, para verificar e enumerar arquivos suspeitos, possibilitando a exclusão destes de acordo com a tomada de decisão do usuário, diminuindo a exclusão por engano de arquivos falsos-positivos.

Figura 60 – Fluxo de detecção de arquivos suspeitos via Atreus

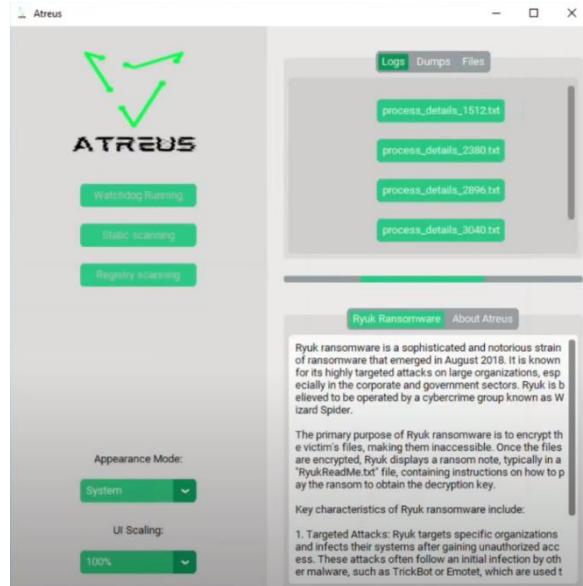


Fonte: Elaborado pelo autor

Com três frentes de detecção de ataques, o protótipo *Atreus* tende a se consolidar adequado na detecção e mitigação de ataques advindos da amostra *Ryuk*. Para facilitar o gerenciamento das informações coletadas e status dos três daemons elucidados anteriormente, cria-se uma interface desktop que centralize todo o processamento do protótipo, podendo permitir a visualização de logs coletados, arquivos suspeitos e ativação de algum *daemon* não ativo. Destarte é ideal que esta contenha elementos gráficos que facilitem a tomada de decisão dos usuários, assim como gerem informações adequadas sobre o processo de monitoramento.

Desse modo, utiliza-se as bibliotecas nativas da linguagem de programação Python, Tkinter e Pyinstaller. A primeira para prototipação de uma interface desktop com fácil visualização dos widgets desejados, tal qual monitoramento adequado dos daemons, gerando avisos e alertas acerca de ações tomadas ou interação com usuário. A segunda, visa a criação do código fonte Python em um executável nativo Windows, o que facilita a portabilidade e redução de possíveis erros de performance ou dependências. A interface principal do protótipo pode ser visualizada pela *figura 61*, na qual apresenta-se o status de execução dos daemons, informações contextuais sobre *Ryuk Ransomware* e o protótipo Atreus, além da possibilidade de visualização de logs, dumps e executáveis suspeitos integrados a interface, facilitando a busca por detalhes de um ataque ou teste realizado.

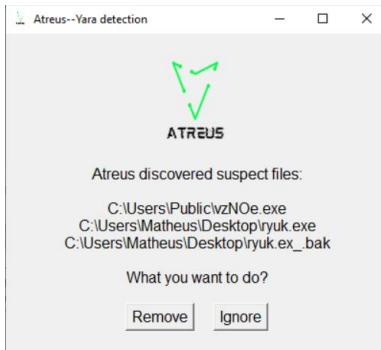
Figura 61 – Interface central do protótipo Atreus



Fonte: Elaborado pelo autor

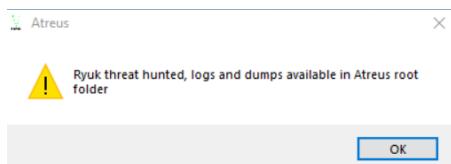
Concomitantemente cria-se *Message Boxes*, com intuito de formalizar a atuação dos daemons após a detecção de comportamentos suspeitos, gerando feedbacks ao usuário. A que permite maior interação com o usuário é justamente a disparada pelo processo de detecção de executáveis suspeitos, na qual lista os arquivos suspeitos detectados e delega ao usuário a decisão de excluir ou manter os arquivos no host, conforme elucida a *figura 62*. As geradas, respectivamente, pelo daemon de detecção de processos que evocam a chamada *CreateRemoteThread* e de mudanças no registro do Windows, disparam informes sobre ações tomadas, como a suspensão de processos suspeitos ou a remoção de registros maliciosos no registro, conforme ilustram as *figuras 63* e *64*.

Figura 62 – MessageBox de detecção de executáveis suspeitos



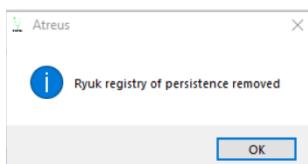
Fonte: Elaborado pelo autor

Figura 63 – MessageBox de informe de suspensão de processos suspeitos



Fonte: Elaborado pelo autor

Figura 64 – MessageBox de informe de exclusão de registro malicioso detectado



Fonte: Elaborado pelo autor

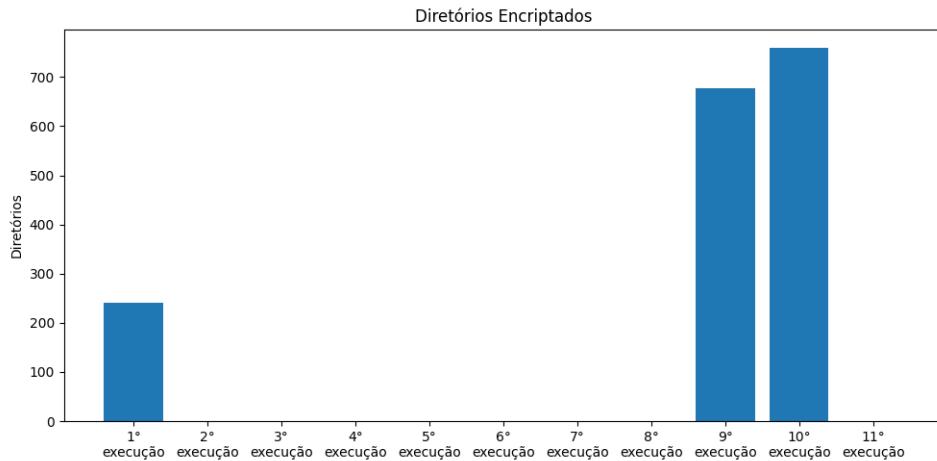
2.5.1 Efetividade de prevenção de execuções maliciosas

Após a concepção do arquétipo estrutural do software, realiza-se testes para averiguação de sua eficácia de impedimento a execuções tal qual o entendimento decorrente do impedimento de seu ataque em estágios iniciais.

Foram executados 11 testagens em ambientes similares de máquinas virtuais com sistema operacional Windows, com o protótipo Atreus já instalado e em execução, confrontando pela amostra de ransomware *Ryuk* analisada. A primeira medida de controle válida para análise é a quantidade de diretórios encriptados e atingidos pelo amostra. É possível obter tal medida mediante a procura da quantidade de *Ransom Notes* na máquina alvo. Em um alvo sem qualquer proteção à execução da ameaça, cerca de 3040 diretórios e subdiretórios são afetados. Com o Atreus em execução, considerando todas as testagens realizadas, fora constatado uma média

de apenas 165 subdiretórios afetados, todos dentro de um mesmo diretório central (MinGW, relativa a versão portada de ferramentas GNU para Windows). Destaca-se que em 8 destes testes, nenhum diretório fora afetado, representando uma significativa melhora na redução de danos causados.

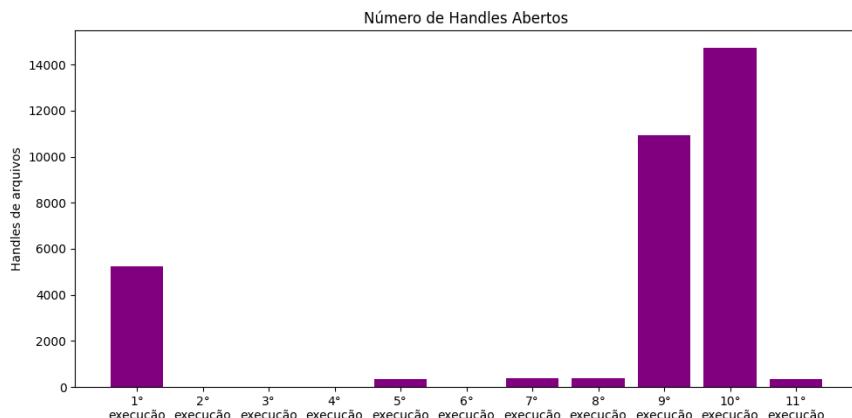
Gráfico 1 – Número de diretórios encriptados durante testes realizados



Fonte: Elaborado pelo autor

Analogamente, ao se observar nessas testagens a quantidade de *handles* de arquivos abertos por cada processo suspeito de pertencer a uma cadeia de ataque *Ryuk*, é possível notar-se que, conforme esperado, existe uma relação entre o número de diretórios afetados pela quantidade de arquivos abertos com o fim de encriptação. Conforme ilustra a figura 66, nas execuções nas quais existiram danos em diretórios, nota-se notoriamente mais *handles* de arquivos abertos, indicando que a encriptação de uma certa quantidade de arquivos ocorreu ainda com sucesso no ataque. Todavia, nota-se que os arquivos afetados pertencem ainda ao mesmo diretório, indicando uma redução de ataque amplo pelo alvo.

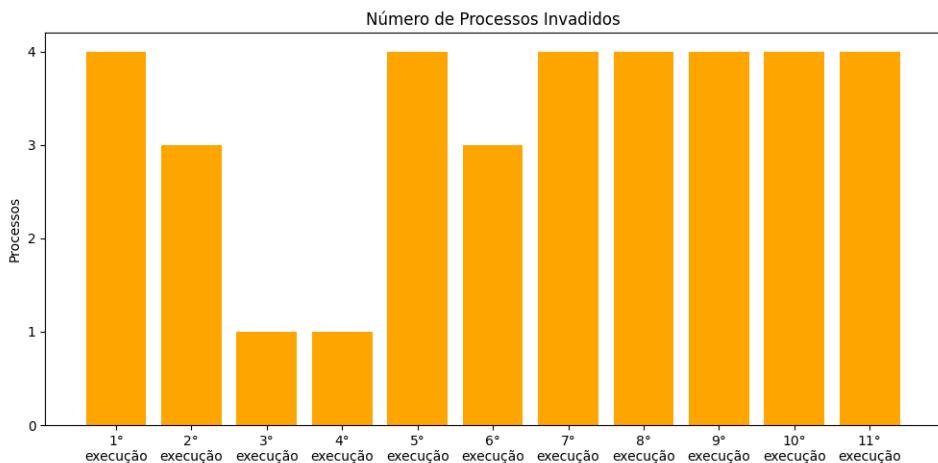
Gráfico 2 – Números de handles abertos para arquivos nos alvos



Fonte: Elaborado pelo autor

Simultaneamente, ao verificar a quantidade de processos invadidos pelo executável principal do *Ransomware*, nota-se que o protótipo consegue reduzir o ecossistema de dezenas de processos invadidos para no máximo quatro. Em geral, tais processos tem funções de gerenciamentos de recursos do host do Windows, como Svchost, Sihost e Taskhost. Apesar de notoriamente notar-se que quanto menos processos invadidos menor a chance de prosseguimento do ataque, evidencia-se que a execução remota dos *threads* nestes é o ponto chave para danos mais significativos, visto que em testes nos quais não foram detectados nenhum diretório atingido (como a 7^a execução) tiveram a mesma quantidade de processos invadidos que testagens nas quais foram constatadas amostras de encriptações (como na 10^a execução).

Gráfico 3 – Quantidade de processos invadidos em cada execução



Fonte: Elaborado pelo autor

Por fim, analisa-se o tempo em memória e em CPU dos processos decorrentes do executável principal do *Ransomware*, com intuito mor de entendimento do tempo de resposta do protótipo Atreus, tal qual para avaliação de uma possível correlação entre o tempo de execução deste para com a quantidade de diretórios atingidos. Conforme ilustra a figura 68, a taxa média de tempo em memória do executável em memória fora de 6.6 segundos, o que também representa, por consequência o tempo de detecção e tomada de medidas do protótipo. Averigua-se que é um tempo adequado para redução considerável de danos, visto que conforme perceptível pela análise anterior do *Ransomware*, este tem capacidade de encriptar milhares de diretórios em minutos.

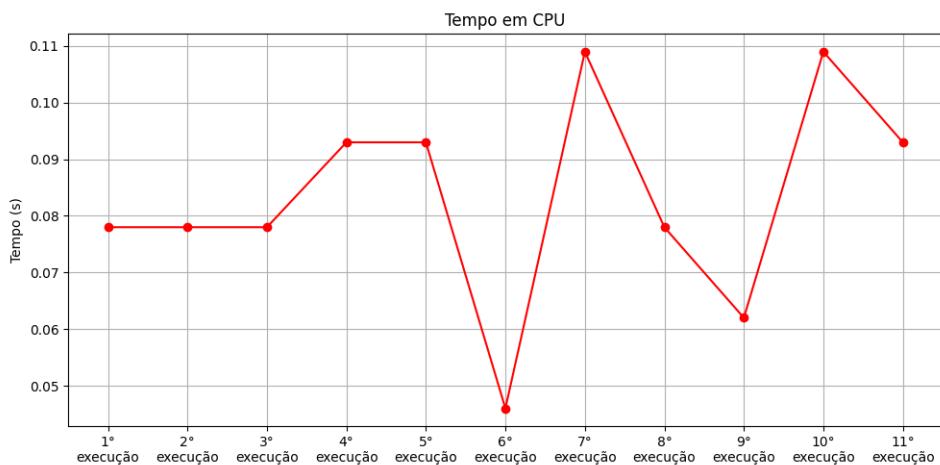
Gráfico 4 – Tempo em memória do processo advindo do executável principal do Ryuk



Fonte: Elaborado pelo autor

Ulteriormente, capta-se também o tempo em CPU dos processos principais do *Ransomware*. Interessante averiguar que não existe uma correlação clara entre o tempo de CPU do processo principal e a quantidade de alvos atingidos, fato possivelmente explicado pelo comportamento multi-thread da ameaça, que consegue desta forma acelerar o tempo de ataque, distribuindo-o em processos invadidos, que conseguem dessa forma mais tempo em CPU.

Gráfico 5 – Tempo em CPU do processo advindo do executável principal do Ryuk



Fonte: Elaborado pelo autor

3 Considerações finais

Doravante ao apresentado no estudo, é notório que o campo da segurança cibernética correlato a análise de *Ransomware* revela muitas perspectivas sobre a mitigação de danos causados por tais ameaças, tal qual em ambientes de resposta a incidentes. Conforme apresentado a maioria dos estudos da área concentram-se no ramo privado, não sendo totalmente acessíveis ao público, dificultando a disseminação de conhecimentos básicos e complexos do meio.

Ademais, durante esta iniciação tecnológica, fora possível entender-se de maneira clara e analítica comportamentos padrões da amostra analisada, tal qual seu processo de persistência, encriptação, invasão de processos e ocultação de rastros. Com plena compreensão destes, fora possível a construção de um protótipo de mitigação e prevenção ao *Ransomware* em questão, gerando dessa forma uma alternativa *open-source* de ramo defensivo a estes softwares maliciosos, sendo este também um estímulo a outros pesquisadores ou entusiastas da área a colaborarem e compartilharem conhecimentos em alternativas abertas, possibilitando maior difusão de informações gratuitamente.

Analogamente, estabelecem-se como passos iterativos desta pesquisa os seguintes tópicos:

- Criação de um mecanismo de decriptação de arquivos encriptados pelo *Ryuk*, após o porte da chave RSA;
- Adição de comportamento de detecção e prevenção do protótipo a outras amostras de *Ransomware*, aumentando sua robustez;
- Portabilidade do protótipo para ambientes Linux e Mac, melhorando a interoperabilidade deste e combate a outros tipos de ameaça.

Por fim, ressalta-se a importância de iniciativas de pesquisa acadêmica no âmbito de análise de *Malware*, aumentando a disseminação de conhecimento sobre o tópico, reduzir a médio prazo a carência de ferramentas e protótipos *open-source* e gratuitos de prevenção e mitigação e também gerar estímulos para maior controle de ameaças localizadas e detectadas.

Referências

1. **5 ataques cibernéticos no Brasil em 2021 que geraram alerta.** Disponível em: <<https://forbes.com.br/forbes-tech/2021/12/5-ataques-ciberneticos-no-brasil-em-2021-que-geraram-alerta/>>. Acesso em: 2 set. 2022.
2. **Brasil: ransomware atingiu 55% das organizações em 2021.** Disponível em: <<https://www.cisoadvisor.com.br/brasil-ransomware-atingiu-55-das-organizacoes-em-2021/>>. Acesso em: 5 out. 2022.
3. **Brasil tem 1.540 ataques de ransomware por semana.** Disponível em: <<https://br.financas.yahoo.com/news/brasil-tem-1540-ataques-de-ransomware-por-semana-195630062.html>>. Acesso em: 5 out. 2022.
4. TAILOR, J.; PATEL, A. A Comprehensive Survey: Ransomware Attacks Prevention, Monitoring and Damage Control. **International Journal of Research and Scientific Innovation (IJRSI)** |, v. IV, 2017.
5. Richardson, Ronny and North, Max M., "Ransomware: Evolution, Mitigation and Prevention" (2017). Faculty Publications. 4276. <https://digitalcommons.kennesaw.edu/facpubs/4276>
6. **Incidentes Relevantes.** Disponível em: <<https://www.ibraspd.org/incidentes>>. Acesso em: 25 dez. 2022.
7. BEAMAN, C. et al. Ransomware: Recent advances, analysis, challenges and future research directions. **Computers & Security**, v. 111, p. 102490, dez. 2021.
8. MONIKA; ZAVARSKY, P.; LINDSKOG, D. Experimental Analysis of Ransomware on Windows and Android Platforms: Evolution and Characterization. **Procedia Computer Science**, v. 94, p. 465–472, 2016.
9. **Chave de encriptação – Definirtec.** Disponível em: <<https://definirtec.com/chave-de-encriptacao>>. Acesso em: 25 dez. 2022.
10. **O que é e como funciona o registro do Windows?** Disponível em: <<https://www.avast.com/pt-br/c-windows-registry#topic-1>>. Acesso em: 26 dez. 2022.

11. BNamericas - Brazil is the second country that suffers th... Disponível em: <<https://www.bnamicas.com/en/news/brazil-is-the-second-country-that-suffers-the-most-cyber-attacks-in-latin-america>>. Acesso em: 5 out. 2022.
12. Ransomware RYUK. Disponível em: <https://www.trendmicro.com/pt_br/what-is/ransomware/ryuk-ransomware.html>. Acesso em: 31 dez. 2022.
13. A, M. K. **Learning Malware Analysis : Explore the Concepts, Tools, and Techniques to Analyze and Investigate Windows Malware**. Birmingham: Packt Publishing Ltd, 2018.
14. KARL-BRIDGE-MICROSOFT. **PE Format - Win32 apps**. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#optional-header-standard-fields-image-only>>.
15. TEAM, C. **ASLR - Address Space Layout Randomization**. Disponível em: <<https://blog.convisoappsec.com/aslr-address-space-layout-randomization/>>. Acesso em: 1 jan. 2023.
16. LASTNAMEHOLIU. **RtlGenRandom function (ntsecapi.h) - Win32 apps**. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/ntsecapi/nf-ntsecapi-rtlgenrandom>>. Acesso em: 1 jan. 2023.
17. **Analizando arquivos para conteúdo integrado e atividade maliciosa**. Disponível em: <<https://www.ibm.com/docs/pt-br/qsip/7.4?topic=content-analyzing-files-embedded-malicious-activity>>. Acesso em: 1 jan. 2023.
18. JWMSFT. **ShellExecuteW function (shellapi.h) - Win32 apps**. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecutew>>.
19. ALVINASHCRAFT. **Privilege Constants (Winnt.h) - Win32 apps**. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/secauthz/privilege-constants>>.

20. DUC, H. N. **Windows Process Internals : A few Concepts to know before jumping on Memory Forensics | By Kirtar Oza - eForensics**. Disponível em: <<https://imphash.medium.com/windows-process-internals-a-few-concepts-to-know-before-jumping-on-memory-forensics-823d72d4d7b8>>. Acesso em: 21 jan. 2023.
21. GRANTMESTRENGTH. **LookupAccountSidW function (winbase.h) - Win32 apps**. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-lookupaccountsidw>>. Acesso em: 21 jan. 2023.
22. ALVINASHCRAFT. **CryptGenKey function (wincrypt.h) - Win32 apps**. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptgenkey>>. Acesso em: 22 jan. 2023.
23. ALVINASHCRAFT. **ALG_ID (Wincrypt.h) - Win32 apps**. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/seccrypto/alg-id>>. Acesso em: 22 jan. 2023.
24. TEAM, M. A., WSR. **O que é criptografia AES-256 e como ela funciona?** Disponível em: <<https://www.websiterating.com/pt/cloud-storage/what-is-aes-256-encryption/>>. Acesso em: 22 jan. 2023.
25. KARL-BRIDGE-MICROSOFT. **OpenProcess function (processthreadsapi.h) - Win32 apps**. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-openprocess>>. Acesso em: 29 jan. 2023.
26. **sihost.exe Windows process - What is it?** Disponível em: <<https://www.file.net/process/sihost.exe.html>>. Acesso em: 29 jan. 2023.
27. KARL-BRIDGE-MICROSOFT. **WriteProcessMemory function (memoryapi.h) - Win32 apps**. Disponível em: <<https://learn.microsoft.com/en>>.

us/windows/win32/api/memoryapi/nf-memoryapi-writeprocessmemory>. Acesso em: 29 jan. 2023.

28. KARL-BRIDGE-MICROSOFT. **CreateRemoteThread** function (**processthreadsapi.h**) - Win32 apps. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createremotethread>>.

29. KARL-BRIDGE-MICROSOFT. **LoadLibraryA** function (**libloaderapi.h**) - Win32 apps. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-loadlibrarya>>.

30. KARL-BRIDGE-MICROSOFT. **GetProcAddress** function (**libloaderapi.h**) - Win32 apps. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getProcAddress>>.

31. ALVINASHCRAFT. **CryptImportKey** function (**wincrypt.h**) - Win32 apps. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptimportkey>>.

32. ALVINASHCRAFT. **FindNextFileW** function (**fileapi.h**) - Win32 apps. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-findnextfilew>>.

33. KARL-BRIDGE-MICROSOFT. **CreateThread** function (**processthreadsapi.h**) - Win32 apps. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread>>.

34. ALVINASHCRAFT. **SetFileAttributesW** function (**fileapi.h**) - Win32 apps. Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-setfileattributesw>>.

35. **Ransomware RYUK**. Disponível em: <https://www.trendmicro.com/pt_br/what-is/ransomware/ryuk-ransomware.html>.

36. ALVINASHCRAFT. **CryptGenKey function (wincrypt.h) - Win32 apps.** Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptgenkey>>.
37. ALVINASHCRAFT. **ALG_ID (Wincrypt.h) - Win32 apps.** Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/seccrypto/alg-id>>.
38. KARKI, A. A Review on Advanced Encryption Standard (AES). **International Journal of Computer Sciences and Engineering**, v. 6, n. 8, p. 551–556, 31 ago. 2018.
39. ALVINASHCRAFT. **CryptEncrypt function (wincrypt.h) - Win32 apps.** Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptencrypt>>. Acesso em: 25 jun. 2023.
40. ALVINASHCRAFT. **CryptExportKey function (wincrypt.h) - Win32 apps.** Disponível em: <<https://learn.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptexportkey>>.
41. **IBM Cloud Docs.** Disponível em: <<https://cloud.ibm.com/docs/key-protect?topic=key-protect-envelope-encryption#:~:text=Envelope%20encryption%20works%20by%20using>>. Acesso em: 26 jun. 2023.
42. MANOCHA, H. **Ryuk Ransomware: History, Timeline, and Adversary Simulation.** Disponível em: <<https://fourcore.io/blogs/ryuk-ransomware-simulation-mitre-ttp>>.
43. **Detours.** Disponível em: <<https://www.microsoft.com/en-us/research/project/detours/>>. Acesso em: 6 ago. 2023.
44. MARKRUSS. **Sysinternals - Sysinternals.** Disponível em: <<https://learn.microsoft.com/en-us/sysinternals/>>.

45. MARKRUSS. **Sysmon** - **Windows Sysinternals**. Disponível em: <<https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon>>.
46. MARKRUSS. **ProcDump** - **Sysinternals**. Disponível em: <<https://learn.microsoft.com/en-us/sysinternals/downloads/procdump>>.
47. MARKRUSS. **Handle** - **Sysinternals**. Disponível em: <<https://learn.microsoft.com/en-us/sysinternals/downloads/handle>>. Acesso em: 6 ago. 2023.
48. MARKRUSS. **ListDLLs** - **Sysinternals**. Disponível em: <<https://learn.microsoft.com/en-us/sysinternals/downloads/listdlls>>.
49. MARKRUSS. **PsList** - **Sysinternals**. Disponível em: <<https://learn.microsoft.com/en-us/sysinternals/downloads/pslist>>.
50. **YARA - The pattern matching swiss knife for malware researchers**. Disponível em: <<https://virustotal.github.io/yara/>>.
51. MARKRUSS. **Sigcheck** - **Sysinternals**. Disponível em: <<https://learn.microsoft.com/en-us/sysinternals/downloads/sigcheck>>.