BOLETIM DE SEGURANÇA

Vulnerabilidade crítica na linguagem de programação R permite execução arbitrária de

heimdall
security research
A DIVISION OF ISH

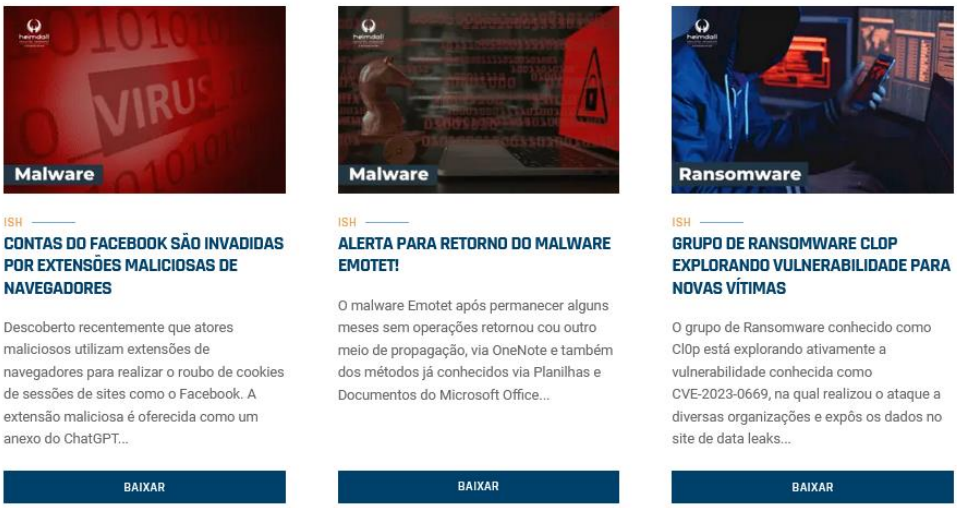
TLP: CLEAR

Receba alertas e informações sobre segurança cibernética e ameaças rapidamente, por meio do nosso **X**.

[Heimdall Security Research]

Acesse boletins diários sobre agentes de ameaças, *malwares*, indicadores de comprometimentos, TTPs e outras informações no *site* da ISH.

[Boletins de Segurança – Heimdall]



SUMÁRIO

1 Sumário Executivo 5
2 Informações sobre a vulnerabilidade 6
3 Recomendações 10
4 Referências 11

LISTA DE FIGURAS

Figura 1 – Função DelayedAssign. 7
Figura 2 – Criação de promise. 7
Figura 3 – Exploração do readRDS. 8
Figura 4 – Biblioteca rbytecode. 9

1 SUMÁRIO EXECUTIVO

Pesquisadores de segurança identificaram uma vulnerabilidade CVE-2024-27322 na linguagem de programação R, que possibilita a execução de código arbitrário durante a desserialização de conteúdo não seguro. Arquivos RDS (**R Data Serialization**) ou pacotes R, comuns entre cientistas de dados e desenvolvedores, podem ser usados para explorar essa brecha. Arquivos RDS corrompidos ou pacotes R com código mal-intencionado, quando carregados, permitem a execução desse código no aparelho da vítima.

2 INFORMAÇÕES SOBRE A VULNERABILIDADE

É possível criar um arquivo RDS malicioso que podem executar comandos arbitrários ao serem carregados no ambiente R. Esse risco está vinculado à manipulação de objetos de promessa e à execução de código em um contexto de avaliação lenta. A função central ‘R_Unserialize’, localizada no arquivo ‘serialize.c’, é responsável por processar o cabeçalho do RDS, que incluem detalhes como o formato do arquivo, a versão do arquivo, a versão do R utilizada na serialização, a versão mínima do R para leitura e, ocasionalmente, a codificação nativa. O formato XDR é o mais usual entre os RDS, mas os formatos ASCII e binário também são possíveis, cada um com seus identificadores únicos. A função ‘R_Unserialize’ omite verificações de cabeçalho para arquivos versão 2, mas para a versão 3, ela confirma um número inteiro adicional e a string ‘native_encoding’. O bytecode do RDS não contém instruções de término, mas suporta a serialização e desserialização de quase todos os objetos em R, graças aos atributos e valores incorporados nas estruturas CAR e CDR. A função ‘ReadItem’ interpreta 36 instruções de bytecode, que se ampliam com comandos adicionais e variam em tamanho conforme sua função específica.

Os formatos de arquivo RDS, que podem ser ASCII, binário ou XDR (este último sendo o mais comum), são identificados por códigos únicos chamados "números mágicos". Normalmente, esses códigos ocupam dois bytes, mas problemas com o formato ASCII podem resultar em números mágicos de três bytes. A função ‘R_Unserialize’ começa lendo esses códigos e prossegue para processar os outros elementos do cabeçalho, armazenados como inteiros – 4 bytes para os formatos XDR e binário, e até 127 bytes para ASCII. Arquivos da versão 2 não passam por verificações de cabeçalho, enquanto na versão 3, um inteiro adicional é lido, seguido pela string ‘native_encoding’, que por padrão é 'UTF-8'. Versões que não são 2 nem 3 exigem a verificação das versões do escritor e do leitor. Após a validação do cabeçalho, a função lê um item do blob.

O formato RDS é notável porque, apesar de consistir em bytecode interpretado e executado pela função ‘ReadItem’, não possui comandos de término. A desserialização resulta em um único objeto, e a leitura se encerra após esse objeto ser processado. Isso significa que qualquer exploração deve se encaixar em um tipo de objeto já existente, sem possibilidade de inserção antes ou depois do objeto retornado. No entanto, quase todos os objetos em R podem ser serializados e desserializados usando RDS, graças aos atributos e valores incorporados nas estruturas CAR e CDR. A função ‘ReadItem’ do interpretador RDS inclui 36 instruções de bytecode, com instruções adicionais disponíveis em combinação com as principais. Cada instrução RDS varia em tamanho de acordo com sua função, mas todas começam com um inteiro que codifica a instrução e sinalizadores através de mascaramento de bits.

Ao analisar o código de desserialização, foi identificado várias funções que inicialmente pareciam suspeitas, mas que não constituíam vulnerabilidades concretas. No entanto, a descoberta de uma instrução específica que gerava um objeto de promise mudou essa percepção. Para compreender o objeto de promise, é essencial entender o conceito de avaliação lazy. Essa técnica adia a avaliação de símbolos até o momento em que são efetivamente necessários, ou seja, no ponto de acesso. Um exemplo prático dessa abordagem é a função 'delayAssign', que permite a atribuição de uma variável somente no momento em que ela é chamada.

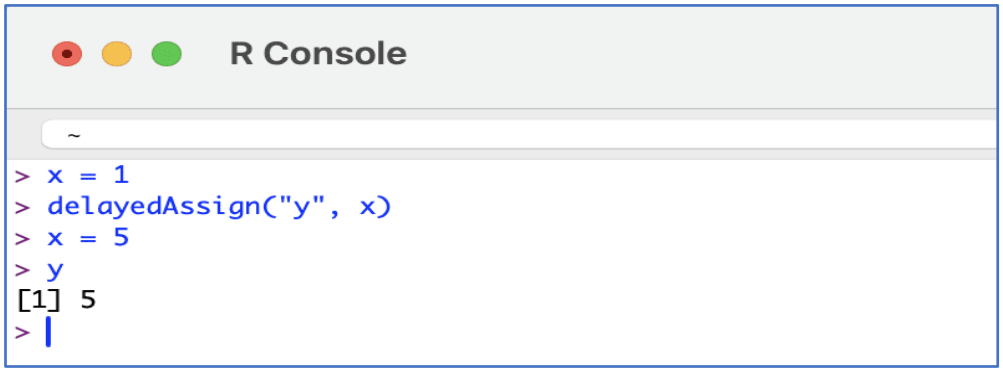


Figura 1 – Função DelayedAssign.

A realização do processo descrito é feita através da criação de um objeto de promise, que contém um símbolo e uma expressão vinculada. Quando o símbolo 'y' é utilizado, a expressão responsável por atribuir o valor de 'x' a 'y' é ativada. O ponto crucial é que 'y' só é definido com o valor 1 no momento em que é utilizado, não antes. Mesmo sem ter alcançado a execução de código durante a desserialização, acredita-se na possibilidade de construir todos os objetos necessários para formar uma promise que, ao ser utilizada posteriormente, teria sua avaliação concretizada.

Ao estabelecer uma promise com um valor indefinido ao invés de um símbolo específico, seria possível gerar uma carga que, no momento do acesso à promise, desencadearia a execução da expressão correspondente.

```
Opcode(TYPES.PROMSXP, 0, False, False, False,None,False),
Opcode(TYPES.UNBOUNDVALUE_SXP, 0, False, False, False,None,False),
Opcode(TYPES.LANGSXP, 0, False, False, False,None,False),
Opcode(TYPES.SYMSXP, 0, False, False, False,None,False),
Opcode(TYPES.CHARSXP, 64, False, False, False,"system",False),
Opcode(TYPES.LISTSXP, 0, False, False, False,None,False),
Opcode(TYPES.STRSXP, 0, False, False, False,1,False),
Opcode(TYPES.CHARSXP, 64, False, False, False,'echo "pwned by HiddenLayer"',False),
Opcode(TYPES.NILVALUE_SXP, 0, False, False, False,None,False),
```

Figura 2 – Criação de promise.

Uma vez que o arquivo malicioso seja gerado e incorporado pelo R, o exploit ocorrerá automaticamente, não importando a maneira pela qual a variável seja invocada.

Figura 3 – Exploração do readRDS.

Após uma análise detalhada no GitHub, foi identificado que a função readRDS, um dos vários métodos que podem ser utilizados para explorar essa falha de segurança, aparece em mais de 135.000 arquivos R. A investigação dos repositórios revelou que a maioria das implementações foi feita com dados de entrada não verificados do usuário, o que representa um risco de comprometimento integral dos sistemas onde os programas são executados. Detectou-se que entre os códigos-fonte que possuem potenciais vulnerabilidades estão inclusos projetos de grandes entidades como R Studio, Facebook, Google, Microsoft, AWS, entre outros fornecedores de software de renome.

Os pacotes R facilitam a distribuição de código R compilado e dados, que são essenciais para tarefas estatísticas de outros usuários. Até a data de redação deste artigo, o CRAN, um repositório de pacotes, hospeda 20.681 pacotes. Qualquer um pode submeter pacotes ao CRAN, contanto que atendam a certos requisitos, como a inclusão de arquivos específicos (por exemplo, um arquivo de descrição) e a aprovação em verificações automatizadas, que, no entanto, não detectam a vulnerabilidade em questão.

Os pacotes R utilizam o formato RDS para armazenar e recuperar dados. Durante a compilação de um pacote, dois arquivos são gerados para auxiliar nesse processo:

- Arquivo .rdb: Serializa os objetos do pacote como blobs binários de dados.

- Arquivo .rdx: Armazena metadados dos objetos serializados no arquivo .rdb, incluindo seus deslocamentos.

Ao carregar um pacote, os metadados no formato RDS do arquivo .rdx são empregados para localizar os objetos no arquivo .rdb. Os objetos são então descomprimidos e desserializados, carregando-os como se fossem arquivos RDS.

Dessa forma, os pacotes R estão sujeitos à vulnerabilidade de desserialização e podem ser explorados em ataques à cadeia de suprimentos através dos repositórios de pacotes. Para comprometer um pacote R, um atacante precisa apenas substituir o arquivo .rdx por um arquivo mal-intencionado. Assim, quando o pacote é carregado, o código malicioso é executado automaticamente.

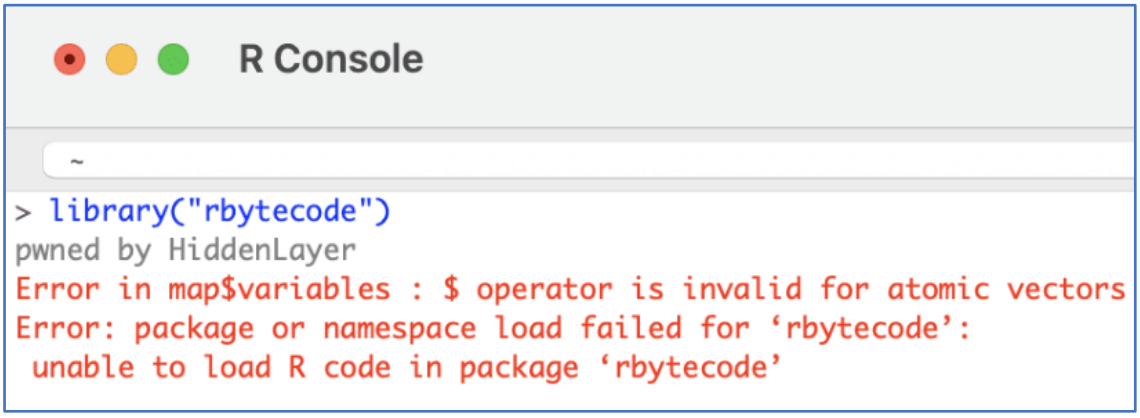


Figura 4 – Biblioteca rbytecode.

3 RECOMENDAÇÕES

Além dos indicadores de comprometimento elencados abaixo pela ISH, poderão ser adotadas medidas visando a mitigação da infecção do referido *malware*, como por exemplo:

Atualize o software

- Se você estiver usando a linguagem de programação R, atualize para a versão 4.4.0 ou superior, pois a vulnerabilidade afeta as versões de 1.4.0 até 4.4.0.

Utilize ativos de segurança

- Mantenha firewalls e anti-malware sempre atualizados para proteger contra ameaças, incluindo de endpoints.

Política de mínimo privilégio

- Implemente uma política de mínimo privilégio, permitindo aos usuários exercerem ações na rede restritas ao desempenho de suas funções organizacionais.

Cuidado com dados não confiáveis

- Evite a desserialização de dados não confiáveis, pois isso pode permitir a execução de código arbitrário em seu sistema.

4 REFERÊNCIAS

- **Heimdall *by* ISH Tecnologia**
- HiddenLayer
- Thehackernews
- NVD

heimdall
security research
A DIVISION OF ISH