BOLETIM DE SEGURANÇA

Vulnerabilidade crítica em biblioteca do Rust tem como alvo usuários do Windows

heimdall
security research
A DIVISION OF ISH

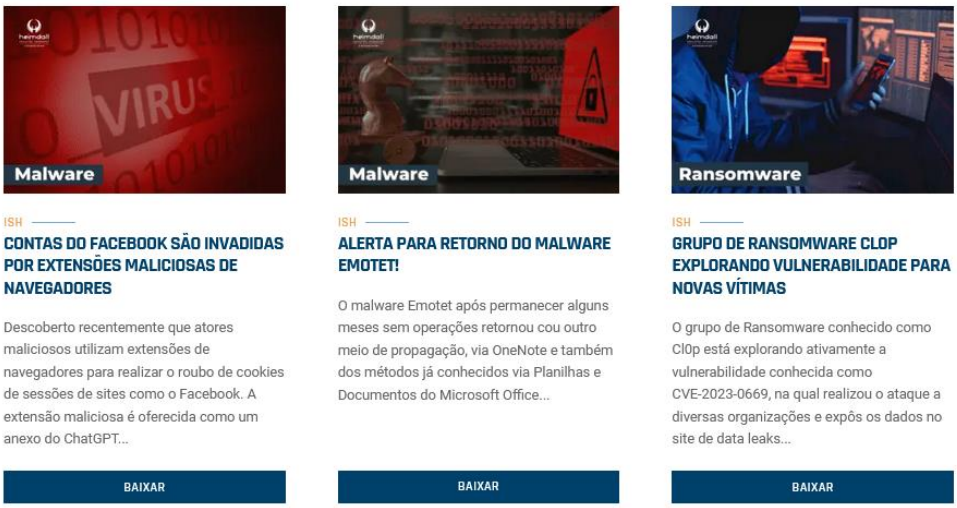
TLP: CLEAR

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

LISTA DE FIGURAS

1 SUMÁRIO EXECUTIVO

Um engenheiro de segurança da Flatt Security, relatou sobre uma falha na biblioteca padrão do Rust que afeta todas as versões anteriores a 1.77.2, podendo ser exploradas atingindo os usuários do Windows e realizando ataques de command injection. A vulnerabilidade conhecida como **BatBadBut**, identificada como CVE-2024-24576, recebeu a pontuação máxima, sinalizando sua alta gravidade. Contudo, ela só se manifesta em situações onde arquivos batch são executados no Windows com argumentos que não são confiáveis.

2 INFORMAÇÕES SOBRE A VULNERABILIDADE

BatBadBut é uma falha de segurança que possibilita a um invasor realizar uma injeção de comando em aplicações Windows que dependem, de forma indireta, da função **CreateProcess**, quando certas condições são satisfeitas. A função CreateProcess() gera, de maneira implícita, o cmd.exe ao executar arquivos em lote (**.bat, .cmd, etc.**), mesmo que o aplicativo não tenha especificado isso na linha de comando. O cmd.exe possui regras complexas para a análise dos argumentos do comando e os tempos de execução da linguagem de programação não conseguem escapar corretamente dos argumentos do comando. Isso resulta na possibilidade de injeção de comandos se alguém conseguir controlar a parte dos argumentos do comando do arquivo em lote.

Na imagem abaixo, o código Node.js simples, por exemplo, pode aparecer calc.exe na máquina do servidor:

```
const { spawn } = require('child_process');
const child = spawn('./test.bat', ['<your-input-here>']);
```

Figura 1 – Exemplo código Node.js.

No entanto, como nem todas as linguagens de programação lidam com essa questão, aconselha-se aos programadores que sejam cuidadosos ao executar comandos no Windows.

Project	Status
Erlang	Documentation update
Go	Documentation update
Haskell	Patch available
Java	Won't fix
Node.js	Patch will be available
PHP	Patch will be available
Python	Documentation update
Ruby	Documentation update
Rust	Patch available

Figura 2 – Linguagens afetadas.

Esse problema ocorre somente se um arquivo em lote for especificado de maneira explícita na linha de comando passada para a função CreateProcess(), e não ocorre quando um arquivo .exe é especificado. Contudo, como o Windows inclui arquivos .bat e .cmd na variável de ambiente PATHEXT por padrão, alguns tempos de execução executam arquivos em lote, contrariando a intenção dos desenvolvedores, se existir um arquivo em lote com o mesmo nome do comando que o desenvolvedor pretendia executar. Assim, até mesmo o trecho de código abaixo pode resultar em execuções arbitrárias de comandos, mesmo que não inclua .bat ou .cmd de forma explícita.

```
cmd := exec.Command("test", "<your-input-here>")
cmd.Run()
```

Figura 3 – Execuções arbitrárias de comandos.

A exploração desses comportamentos pode ocorrer quando um aplicativo executa um comando no Windows, a extensão do arquivo do comando não é especificada pelo aplicativo, ou é .bat ou .cmd. O comando em execução inclui entradas controladas pelo usuário como argumentos. O ambiente de execução da linguagem de programação falha ao escapar corretamente dos argumentos do comando cmd.exe. Essas condições podem permitir a execução arbitrária de comandos.

A origem do BatBadBut reside no comportamento negligenciado da função CreateProcess no Windows. Quando arquivos em lote são executados com a função CreateProcess, o Windows implicitamente gera cmd.exe, pois não pode executar arquivos em lote sem ela. Por exemplo, para executar o arquivo em lote test.bat, o seguinte código é gerado: **C:\Windows\System32\cmd.exe /c .\test.bat**.

```
wchar_t arguments[] = L".\\test.bat";
STARTUPINFO si{};
PROCESS_INFORMATION pi{};
CreateProcessW(nullptr, arguments, nullptr, nullptr, false, 0, nullptr, nullptr, &si, &pi);
```

Figura 4 – Execução do arquivo test.bat.

As linguagens de programação geralmente disponibilizam uma função que permite a execução de um comando, encapsulando a função CreateProcess para proporcionar uma interface mais intuitiva ao usuário. Um exemplo disso é o módulo child_process no Node.js 3, que encapsula a função CreateProcess e oferece um método para a execução de um comando com argumentos específicos.

```
const { spawn } = require('child_process');
const child = spawn('echo', ['hello', 'world']);
```

Figura 5 – Módulo do child process no Node.js.

Nos sistemas Unix, a maioria dos shells segue regras de escape parecidas, onde o caractere de escape é a barra invertida (\). Assim, para usar aspas duplas (“) dentro de uma string que já está entre aspas duplas, a barra invertida pode ser empregada.

```
echo "Hello \"World\""
```

Figura 6 – Caractere de escape na string.

A barra invertida é comumente usada como caractere de escape, sendo um padrão amplamente aceito, inclusive em formatos como JSON e YAML. Contudo, ao executar o comando a seguir no prompt de comando, o programa **calc.exe** será iniciado.

```
echo "\"&calc.exe"
```

Figura 7 – Programa calc.exe sendo iniciado.

O prompt de comando difere de outros, pois não utiliza a barra invertida como caractere de escape, optando pelo sinal “^”. No exemplo do `child_process`, as aspas duplas (“) nos argumentos do comando são escapadas com uma barra invertida (\). Contudo, ao executar o arquivo em lote, essa forma de escape não é suficiente devido às regras de escape do `cmd.exe`. Assim, mesmo com o argumento corretamente separado e a opção do `shell` desativada, o trecho `calc.exe` é gerado.

```
const { spawn } = require('child_process');
const child = spawn('./test.bat', ['"&calc.exe']);
```

Figura 8 – calc.exe gerado.

Esse comportamento possibilita que um argumento mal-intencionado na linha de comando execute uma injeção de comando.

3 RECOMENDAÇÕES

Conforme o Rust Security Response, devido à complexidade do cmd.exe, não foi identificada uma solução que escapasse corretamente dos argumentos em todos os casos. Para manter as garantias da API, foi melhorado a robustez do código de escape e alterado a CommandAPI para retornar um InvalidInput erro quando não puder escapar de um argumento com segurança. Este erro será emitido ao gerar o processo.

A correção foi incluída no Rust 1.77.2.

Se você mesmo implementar o escape ou apenas manipular entradas confiáveis, no Windows também poderá usar o método CommandExt::raw_arg para ignorar a lógica de escape da biblioteca padrão.

4 REFERÊNCIAS

- **Heimdall *by* ISH Tecnologia**
- Flatt
- Rust
- Thehackernews

heimdall
security research
A DIVISION OF ISH