BOLETIM DE SEGURANÇA

Vulnerabilidade crítica no plugin LiteSpeed Cache do WordPress permite privilégios de administrador

heimdall
security research
A DIVISION OF ISH

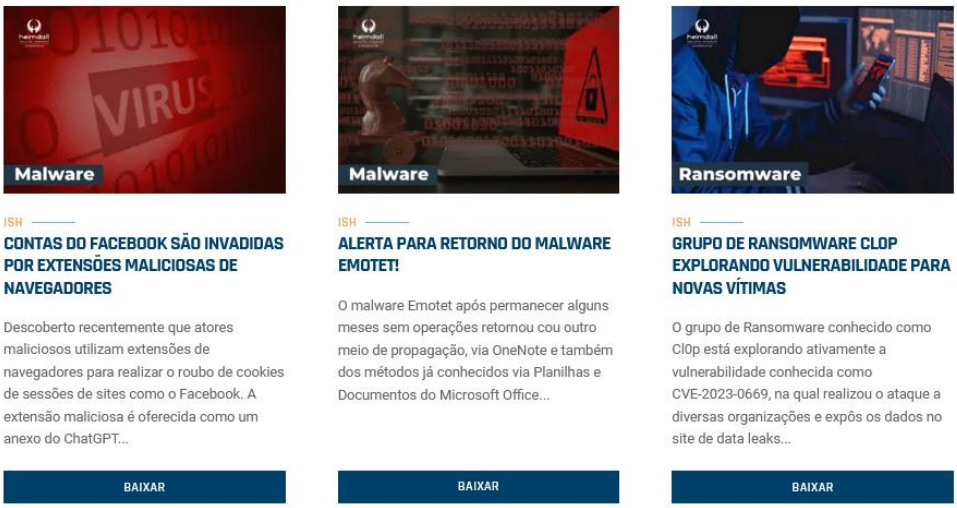
TLP: CLEAR

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

Especialistas em cibersegurança identificaram uma vulnerabilidade crítica CVE-2024-28000 no plugin LiteSpeed Cache do WordPress, possibilitando que usuários não autenticados adquiram acesso de administrador e realizem quaisquer atividades.

Figura 1 – Logo do LiteSpeed Cache.

2 INFORMAÇÕES SOBRE A VULNERABILIDADE

A vulnerabilidade CVE-2024-28000 no plugin LiteSpeed Cache, trata-se de uma falha de escalonamento de privilégios não autenticada que permite atacantes não autenticados escalem seus privilégios e obtenham acesso administrativo, possibilitando o controle total dos sites afetados. Estima-se que mais de **5 milhões** de sites WordPress estejam em risco.

Esta falha está relacionada à implementação inadequada da função de "***function simulation***" dentro do LiteSpeed Cache. Atacantes podem manipular um valor hash armazenado no banco de dados para se passar por um administrador. A função ***async_litespeed_handler()*** gera o valor hash crítico sem verificações adequadas, permitindo que usuários não autorizados a acionem. A função ***is_role_simulation()*** compara o cookie ***litespeed_hash*** com o hash armazenado no banco de dados e, se corresponderem, define a função do usuário atual como administrador. Ele também inclui um recurso de rastreador que rastreia seu site em uma programação para pré-popular os caches das páginas do seu site. Um rastreador pode simular um usuário logado de um determinado ID, portanto, o plugin inclui um recurso de simulação de usuário que verifica os valores dos cookies ***litespeed_role*** e ***litespeed_hash*** para um ID de usuário e hash de segurança e define o ID do usuário atual usando a função ***wp_set_current_user*** se um hash de segurança válido for usado.

O hash de segurança pretende proteger o recurso de simulação de usuário de ser usado por qualquer coisa que não seja uma solicitação de rastreador validada. O trecho de código abaixo é uma ilustração de como o hash de segurança é gerado no método **Str::rrand()** antes de ser usado pela funcionalidade do rastreador.

src/str.cls.php, function rrand()

```php
public static function rrand($len, $type = 7)
{
    mt_srand((int) ((float) microtime() * 1000000));

    switch ($type) {
        case 0:
            $charlist = '012';
            break;
        case 1:
            $charlist = '0123456789';
            break;
        case 2:
            $charlist = 'abcdefghijklmnopqrstuvwxyz';
            break;
        case 3:
            $charlist = '0123456789abcdefghijklmnopqrstuvwxyz';
            break;
        case 4:
            $charlist = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
            break;
        case 5:
            $charlist = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
            break;
        case 6:
            $charlist = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
            break;
        case 7:
            $charlist = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
            break;
    }

    $str = '';

    $max = strlen($charlist) - 1;
    for ($i = 0; $i < $len; $i++) {
        $str .= $charlist[mt_rand(0, $max)];
    }

    return $str;
}
```

Figura 2 – Método Str::rrand().

Infelizmente, a geração desse hash de segurança apresenta vários problemas que tornam seus valores possíveis conhecidos:

- O gerador de números aleatórios que cria o hash de segurança é inicializado com a porção de microssegundos do tempo atual. Isso significa que os únicos valores possíveis para a semente são inteiros de 0 a 999.999.
- O gerador de números aleatórios não é criptograficamente seguro, o que significa que os valores "aleatórios" que ele gera são totalmente determináveis se a semente for conhecida.
- O hash de segurança é gerado uma vez e salvo na opção litespeed.router.hash no banco de dados. Ele não é salgado com um segredo, nem está conectado a uma solicitação específica ou a um usuário específico. Uma vez gerado, o valor nunca muda.
- Devido a todos os pontos acima, existem apenas 1 milhão de valores possíveis para o hash de segurança. Os valores são conhecidos e idênticos em todos os ambientes e sites.

O trecho de código abaixo ilustra como o valor do cookie litespeed_hash é validado como uma condição de guarda para a chamada da função wp_set_current_user, que usa o valor do cookie litespeed_role. Esse código é chamado no gancho init no WordPress e é executado para todas as solicitações, exceto para aquelas na área de administração.

src/router.cls.php, function is_role_simulation()

```
public function is_role_simulation()
{
    if (is_admin()) {
        return;
    }

    if (empty($_COOKIE['litespeed_role']) || empty($_COOKIE['litespeed_hash'])) {
        return;
    }

    Debug2::debug('[Router] starting role validation');

    // Check if is from crawler
    // if ( empty( $_SERVER[ 'HTTP_USER_AGENT' ] ) || strpos( $_SERVER[ 'HTTP_USER_AGENT' ], Crawler::F
AST_USER_AGENT ) !== 0 ) {
    //  Debug2::debug( '[Router] user agent not match' );
    //  return;
    // }

    // Hash validation
    $hash = self::get_option(self::ITEM_HASH);
    if (!$hash || $_COOKIE['litespeed_hash'] != $hash) {
        Debug2::debug('[Router] hash not match ' . $_COOKIE['litespeed_hash'] . ' != ' . $hash);
        return;
    }

    $role_uid = $_COOKIE['litespeed_role'];
    Debug2::debug('[Router] role simulate litespeed_role uid ' . $role_uid);

    wp_set_current_user($role_uid);
}
```

Figura 3 – Validação do cookie litespeed.

O valor válido do hash de segurança é obtido da opção litespeed.router.hash e comparado ao valor no cookie litespeed_hash. A comparação não é estrita nem em tempo constante, o que pode tornar o hash vulnerável a ataques de coerção ou de tempo. No entanto, na prática, seria mais trabalhoso explorar isso do que iterar os valores conhecidos do hash. Sabemos que existem apenas 1 milhão de valores possíveis e conhecidos para o hash de segurança armazenado (e que seu valor nunca muda após ser definido). Isso abre a possibilidade de iterar todos os valores possíveis e enviá-los no cookie litespeed_hash para descobrir o valor válido do hash. Estabelecemos que esse hash de segurança é fraco e, portanto, abre uma vulnerabilidade que levará à chamada da função wp_set_current_user.

No entanto, há uma consideração adicional que impede sua exploração: a funcionalidade de crawler no LiteSpeed Cache está desativada por padrão. A menos que a funcionalidade de crawler tenha sido ativada e pelo menos uma varredura tenha sido realizada, o caminho do código para gerar o hash de segurança não será chamado. Se o valor do hash estiver vazio, a fraqueza não pode ser explorada e o número total de sites vulneráveis seria apenas uma parte dos que usam o plugin. No entanto, há uma fraqueza adicional no plugin que permite que o valor do hash de segurança seja gerado e salvo mesmo quando a funcionalidade de crawler está desativada. Isso significa que todos os sites que usam o LiteSpeed Cache — não apenas aqueles com a funcionalidade de crawler ativada — são vulneráveis. O hash é gerado e salvo no método Router::get_hash. Esse método é chamado por alguns métodos relacionados à configuração que são chamados durante as varreduras, e, surpreendentemente, esses métodos podem ser acionados por um manipulador Ajax não protegido. O método Task::async_litespeed_handler é conectado à ação wp_ajax_nopriv_async_litespeed.

src/router.cls.php, function get_hash()

```php
public static function get_hash()
{
    // Reuse previous hash if existed
    $hash = self::get_option(self::ITEM_HASH);
    if ($hash) {
        return $hash;
    }

    $hash = Str::rrand(6);
    self::update_option(self::ITEM_HASH, $hash);
    return $hash;
}
```

Figura 4 – Hash gerado e salvo no método Router::get_hash.

Uma chamada para */wp-admin/admin-ajax.php?action=async_litespeed&litespeed_type=crawler* por qualquer usuário não autenticado acionará o método com o parâmetro de roteamento necessário

para, em última instância, chamar Router::get_hash e gerar o hash de segurança. Abaixo estão as chamadas em cadeia da solicitação de ação AJAX:

- *Task::async_litespeed_handler()*
- *Crawler::async_handler()*
- *Crawler::start()*
- *Crawler::_crawl_data()*
- *Crawler::_engine_start()*
- *Crawler::_do_running()*
- *Crawler::_get_curl_options()*
- *Router::get_hash()*
- *Str::rrand()*

Esse endpoint AJAX pode ser chamado como o primeiro passo em um ataque ao site para garantir que o hash de segurança exista.

Realizar esse ataque na interface do site é possível, mas seria necessário inspecionar a resposta HTML para determinar se o hash é válido e o usuário está logado. Em vez disso, foi possível escalar a fraqueza via a API REST, que responderá com um código de status HTTP mais útil se o hash for válido, enviando solicitações POST para o endpoint /wp/v2/users da API REST junto com nossos valores iterados no cookie litespeed_hash e o ID do usuário alvo no cookie litespeed_role, uma nova conta de usuário de nível Administrador será criada quando um valor de hash válido for usado. Este ataque tem sucesso sem a necessidade de um nonce da API REST devido à chamada da função wp_set_current_user, que define o usuário atual para nós no contexto da solicitação da API REST quando o hash de segurança é válido. Quando o ataque tem sucesso, ele responderá com um código de status HTTP 201, caso contrário, responderá com um 401.

Se a configuração “Debug Log” do plugin Litespeed Cache estiver definida como “ON”, o hash de segurança também será vazado para o arquivo debug.log na pasta /wp-content/ em qualquer solicitação com um hash de segurança incompatível.

3 RECOMENDAÇÕES

Como essa vulnerabilidade existe porque o código usa uma verificação de hash fraca, a equipe do LiteSpeed decidiu aplicar essas proteções adicionais:

- Adicionada validação de hash do valor da opção async_call-hash na função Router::async_litespeed_handler().
- Adicionado valor litespeed_flash_hash de uso único, uma verificação de hash adicional que será limpa logo após a validação e com TTL definido para apenas 120 segundos.
- Usando 32 caracteres aleatórios para os valores async_call-hash, litespeed_flash_hash e litespeed_hash.
- Para simulação de função de crawler, o código gerará um hash cada vez que o crawler for executado novamente e, uma vez que o hash for validado, armazenará o IP da solicitação atual para a próxima validação.

A equipe que realizou a pesquisa inicialmente recomendou o uso da função hash_equals para o processo de comparação de valores de hash para evitar possíveis ataques de tempo. Também recomendaram o uso de um gerador de valores aleatórios mais seguro, como a função random_bytes. Isso não foi implementado devido à necessidade de suporte ao PHP legado. Portanto, espera-se que a equipe do Litespeed ainda considere implementar isso usando uma biblioteca polyfill.

4 REFERÊNCIAS

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

5 AUTORES

- **Leonardo Oliveira Silva**

heimdall
security research
A DIVISION OF ISH