University of Tartu
Faculty of Science and Technology
Institute of Computer Science

CVE-2020-8819
An essay for Computer Security (LTAT.06.002)

Writer: Anett-Kristin Palmar
Supervisor: Meelis Roos, MSc

Tartu 2020
Introduction

With the following I allow The University of Tartu to publically exhibit this essay five (5) years, starting from the publishing date (01.05.2020).

The website builder WordPress has a partner called WooCommerce; it allows WordPress users to add the WooCommerce plugin to their site to create an online store. The WooCommerce plugin can include CardGate payment methods, if the user wants them. For this vulnerability, the user needed to have chosen to use any version of the CardGate Payment Gateway.

The vulnerability was discovered on the 24th of February 2020 by Vladislav Svolsky, an ethical hacker based in Saint Petersburg. As of March 2, 2020, CVE-2020-8819 has been fixed on all versions of the CardGate payment gateway (3.1.15 and earlier).

The gist of the vulnerability is that the payment gateway didn’t check that the information it received was from a valid source in the instant payment notification (IPN) callback function. This could have let unauthorized people replace critical information and even receive payments meant for the store.

The vulnerability

Payment gateway is a term that refers to the communication between a payment portal (in this case the WooCommerce plugin) and the user’s payment method’s backend (ie PayPal; CardGate has several merchants). Essentially it’s the user’s information being sent to the payment method and the corresponding information being sent back.

As mentioned in the introduction, the payment gateway didn’t check that the information it received was from a valid source in the instant payment notification (IPN) callback function.

An IPN is a method for online retailers to automatically send and/or receive messages based on the events that pertain to a transaction. These messages are sent between the two sides and gateway listeners, which read the messages and then react to them based on what the backend tells them to.

This in context of the vulnerability means that anyone could send a message to the CardGate gateway and the gateway wouldn’t check if the message was from someone they should trust.

This means that anyone could, in the words of Vladislav Svolsky, “remotely replace critical plugin settings (merchant id, secret key etc) with known to him and therefore bypass payment process (eg. spoof order status by manually sending IPN callback request with a valid signature but without real payment) and/or receive all subsequent payments (on behalf of the store)”. As you can imagine, this kind of vulnerability is critical for businesses to fix so no one can exploit it and trust in their company and their cybersecurity doesn’t wane.

The code

The code is question is in the CardGate GitHub in woocommerce/cardgate/cardgate.php and cardgate/cardgate-clientlib-php/src/Client.php, in that order:

```php
if (! empty($_REQUEST['cgp_sitesetup']) && ! empty($_REQUEST['token'])) {
    try {
```
As you can see, first the code checks whether the site is set up and that there isn’t an empty request token. If both of those prove to be true, it goes into the try{} function. The anything proves to cause an error then it moves onto the catch{} section and sends out an API exception message and kills the current PHP script.

The code uses the function pullConfig( $sToken_) to get $aResults and $aConfigData using $aResults. And without checking the authorization of this request token, it moves onto update_option function using data from $aConfigData. It then uses the die() function to write a message and kill the current PHP script.

The corrected code looks like this:

```php
require_once WP_PLUGIN_DIR . '/cardgate/cardgate-clientlib-php/init.php';
 baisTest = ($_REQUEST['testmode'] == 1 ? true : false);
$aResult = cardgate\api\Client::pullConfig($_REQUEST['token'], $bIsTest);
$aConfigData = $aResult['pullconfig']['content'];
update_option('cgp_mode', $aConfigData['testmode']);
update_option('cgp_siteid', $aConfigData['site_id']);
update_option('cgp_hashkey', $aConfigData['site_key']);
update_option('cgp_merchant_id', $aConfigData['merchant_id']);
update_option('cgp_merchant_api_key', $aConfigData['api_key']);
die($aConfigData['merchant'] . '.' . get_option('cgp_siteid') . '.200');
} catch (cardgate\api\Exception $oException_) {

die(htmlspecialchars($oException_->getMessage()));
}

As you can see, first the code checks whether the site is set up and that there isn’t an empty request token. If both of those prove to be true, it goes into the try{} function. The anything proves to cause an error then it moves onto the catch{} section and sends out an API exception message and kills the current PHP script.

The code uses the function pullConfig( $sToken_) to get $aResults and $aConfigData using $aResults. And without checking the authorization of this request token, it moves onto update_option function using data from $aConfigData. It then uses the die() function to write a message and kill the current PHP script.

The corrected code looks like this:

```php
if (! empty($_REQUEST['cgp_sitesetup']) && ! empty($_REQUEST['token'])) {
    try {
        require_once WP_PLUGIN_DIR . '/cardgate/cardgate-clientlib-php/init.php';
        $sVersion = ( $this->get_woocommerce_version() == '' ? 'unknown' : $this->get_woocommerce_version() );
        $sLanguage = substr( get_locale(), 0, 2 );
        $bIsTest = ($_REQUEST['testmode'] == 1 ? true : false);
        $iMerchantId = (int)(get_option('cgp_merchant_id') !== false ? 0 : get_option('cgp_merchant_id'));
        $sMerchantApiKey = (get_option('cgp_merchant_api_key') !== false ? 'initconfig' : get_option('cgp_merchant_api_key'));
        $oCardGate = new cardgate\api\Client( $iMerchantId, $sMerchantApiKey, $bIsTest );
        $oCardGate->setIp( $_SERVER['REMOTE_ADDR'] );
        $oCardGate->setLanguage( $sLanguage );
        $oCardGate->version()->setPlatformName( 'WooCommerce' );
        $oCardGate->version()->setPlatformVersion( $sVersion );
        $oCardGate->version()->setPluginName( 'CardGate' );
        $oCardGate->version()->setPluginVersion( get_option('cardgate_version') );
        $aResult = $oCardGate->pullConfig($_REQUEST['token']);
        if (isset($aResult['success']) && $aResult['success'] == 1){
```
```php
$aConfigData = $aResult['pullconfig']['content'];
update_option('cgp_mode', $aConfigData['testmode']);
update_option('cgp_siteid', $aConfigData['site_id']);
update_option('cgp_hashkey', $aConfigData['site_key']);
update_option('cgp_merchant_id', $aConfigData['merchant_id']);
update_option('cgp_merchant_api_key', $aConfigData['api_key']);
die ($aConfigData['merchant'] . '.' . get_option('cgp_siteid') . '.200');

} else {
    die('Token retrieval failed.');
}
} catch (cardgate\api\Exception $oException_)
{
die(htmlspecialchars($oException_->getMessage()));
}

public function pullConfig( $sToken_ ) {
    /* commented out a section for the sake of this essay’s length */
    return $this->doRequest($sResource);
}
```

As before, the code checks whether the site is set up and that there isn’t an empty request token. This part remains the same.

This code checks the version of WooCommerce. This is due to an automatically configuring option that only exists in the latest version of the software. It also checks the client language.

Instead of the merchant id and api key being through $aConfigData, it finds them outside of it.

After that is where the control happens. It creates the correct environment (if the request was from WooCommerce) to check against the current token's environment. If $aResults is ['success'] and equals one (1) then the environment was correct and the sender was an authorized user. This means that update_options() can go through, as an unauthorized token cannot send a request. If $aResults isn’t ['success'] or doesn’t equal one (1), then PHP sends the message “Token retrieval failed” and kills the PHP script.

The amendment works due to the fact that it now checks whether the token came from an authorized source (WooCommerce). It isn’t a particularly complicated fix, but a necessary one.

**Sources**

The vulnerability. [https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-8819](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-8819) Last opened 01.05.2020

CardGate Github. [https://github.com/cardgate/woocommerce/issues/18](https://github.com/cardgate/woocommerce/issues/18) Last opened 01.05.2020

Payment Gateway. [https://www.investopedia.com/terms/p/payment-gateway.asp](https://www.investopedia.com/terms/p/payment-gateway.asp) Last opened 01.05.2020


Code excerpts. [https://github.com/cardgate/woocommerce/pull/17/commits/0b83588d604c8c56c7fded43144fced96b2ada9](https://github.com/cardgate/woocommerce/pull/17/commits/0b83588d604c8c56c7fded43144fced96b2ada9) Last opened 01.05.2020