Vulnerability Advisory

ISEC-V2021-04

Stripe for WooCommerce - WordPress plugin
Lack of capability checks leading to bank identity theft

INTRINSEC

Innovative by design

Site internet
www.intrinsec.com

Blog
www.intrinsec.com/blog

Twitter
@Intrinsec

TOUR CBX, 20ème ETAGE, 1 PASSERELLE DES REFLETS, 92913 PARIS LA DEFENSE CEDEX • TEL +33 1 41 91 77 77
INTRINSEC SECURITE • SAS AU CAPITAL DE 2 760 680 € • SIREN : 812 535 284 • APE : 6203Z

ADVISORY ISEC-V2021-04

Vendor: Payment Plugins (Stripe's partner)

Product: Stripe for WooCommerce

Title: Lack of capability checks leading to bank identity theft

Intrinsec ID: ISEC-V2021-04

Published: 15/07/2021

Last updated: 25/07/2021

Risk: Important

Exploitation: remote with authenticated access

Impact:

- Security feature bypass
- Sensitive data exposure

Description: An authenticated user with access to the */wp-admin/profile.php* page could update their data and set their own Stripe Customer ID's. These identifiers link the user account to the payment methods registered by this user. Thus, an attacker could fix himself his identifier by choosing the one of another user and use the payment methods saved by this user.

Vulnerable versions: 3.3.9 - Older versions are also affected since 3.0.0, and maybe the ones before.

Solutions:

Check user capabilities before executing the action.

Credits:

Vulnerability discovered by Margaux DABERT from Intrinsec.

History:

2021-07-20: Vulnerability identified

2021-08-12: Communication with the plugin editor

2021-09-10: Communication of the advisory to WordPress

2021-10-01: Obtaining the CVE number CVE-2021-39347

DETAILED APPROACH

While reviewing the source code of the *Stripe for WooCommerce* plugin (https://wordpress.org/plugins/woo-stripe-payment/), we identified that the following function in the */includes/admin/class-wc-stripe-admin-user-edit.php* file performs sensitive request without capabilities check.

```
// /includes/admin/class-wc-stripe-admin-user-edit.php:
// init: Lines 13 to 18
// save: Lines 55 to 114

public static function init() {
  […]
  add_action( 'edit_user_profile_update', array( __CLASS__, 'save' ) );
  add_action( 'personal_options_update', array( __CLASS__, 'save' ) );
}

public static function save( $user_id ) {
  $modes = array( 'test', 'live' );
  if ( isset( $_POST['wc_stripe_live_id'] ) ) {
    $old_live_id = wc_stripe_get_customer_id( $user_id, 'live' );
    wc_stripe_delete_customer( $user_id, 'live', true );
    wc_stripe_save_customer( wc_clean( $_POST['wc_stripe_live_id'] ), $user_id, 'live' );
  }
  if ( isset( $_POST['wc_stripe_test_id'] ) ) {
    $old_test_id = wc_stripe_get_customer_id( $user_id, 'test' );
    wc_stripe_delete_customer( $user_id, 'test', true );
    wc_stripe_save_customer( wc_clean( $_POST['wc_stripe_test_id'] ), $user_id, 'test' );
  }

  // check if admin want's to delete any payment methods
  foreach ( $modes as $mode ) {
    if ( isset( $_POST[ $mode . '_payment_method_actions' ] ) ) {
      switch ( wc_clean( $_POST[ $mode . '_payment_method_actions' ] ) ) {
        case 'delete':
          […]
          break;
      }
    }
  }

  $changes = array(
    'live' => $old_live_id !== wc_stripe_get_customer_id( $user_id, 'live' ),
```

```
    'test' => $old_test_id !== wc_stripe_get_customer_id( $user_id, 'test' ),
  );

  // this will prevent the payment method from being deleted in Stripe. We only want to rem
ove the tokens
  // from the WC tables.
  […]
  // if the value has changed, then remove old payment methods and import new ones.
  foreach ( $changes as $mode => $change ) {
    if ( $change ) {
      // Delete all current payment methods in WC then save new ones.
      […]
      // import payment methods from Stripe.
      if ( ( $customer_id = wc_stripe_get_customer_id( $user_id, $mode ) ) ) {
        WC_Stripe_Customer_Manager::sync_payment_methods( $customer_id, $user_id, $mode );
      }
    }
  }
}
```

In the “save” function, user’s capabilities are never checked. This lack of verification can be critical: it enables any authenticated user to trigger the “wc_stripe_save_customer” and “WC_Stripe_Customer_Manager::sync_payment_methods” functions for example. Moreover, the “user_id” is linked to a “customer_id” which is linked to the payment methods and there is no constraints regarding the unicity of the “customer_id” identifier. In this way, several users can have the same “customer_id” identifier and so can have access to the same payment methods saved. Combining this behaviour with the lack of capabilities check, an authenticated attacker could then steal user’s bank identity by fixing his own “customer_id” identifier to that another user and using the payment methods saved by this one.

Any authenticated user with access to */wp-admin/profile.php* page can perform this operation. However, the default format of a “customer_id” identifier is a string that begins with “cus_” followed by 14 characters of an alphabet composed of upper and lower case letters and numbers. There are therefore 62^{14} possible combinations to form identifiers, which reduces the probability of finding a valid identifier. On the other hand, these strings of 14 characters seem to be linked with the time according to our tests. The generated identifiers respect the same ascending order as when they were generated. This can reduce the number of possibilities. As an example, we have generated the following identifiers in order:

- cus_Juh5PHOcDpGDB5
- cus_JuwruEQaFBhyik
- cus_JuwvUm9zlt19wR
- cus_Jux0xM4KwdFgn3
- cus_Juxa7ozJXFcQmi
- cus_Juy1KS97j81jgX

However, the generation of the latter seems to be done on the side of the Stripe interface, outside the context of WordPress.

This vulnerability could enable an attacker to update his profile and retrieve other “id_customer” in order to use the registered payment methods of another user, which would impact the brand image of the company, as well as the confidentiality and integrity of the data.