Payment Gateway Integration for Post-Purchase Offer Charge

1. **Introduction**

   For charging offers, we tokenize the user's card and use the token for charging accepted offers. Currently, we support 10 gateways built into the UpStroke plugins, and more than 15 additional gateways are supported via a compatibility addon. We also have an inbuilt Test Gateway by WooFunnels for testing our upsell flow. It can be enabled from global settings and only shows on the checkout page for Admin users.

---

2. **Process of Making a Payment Gateway Compatible with Upsells for Charging Offers**

   2.1 **Feasibility Testing**

   First, we perform feasibility testing of the payment gateway to check its charging mechanism and whether charging through a saved token is supported. If the gateway supports tokenization and/or subscriptions, we attempt to tokenize the payment method during checkout. If tokenization is successful, we proceed with the funnel initiation and show offers to the user.

   2.2 **Payment Methods**

   Some payment methods complete the payment immediately after processing the card and the transaction is successful. Examples include Stripe, Square, Braintree Credit Card, Authorize.NET CIM, NMI gateways, and Ebanx. Other gateways, like PayPal, Mollie, and Omise (in the case of 3Ds), use IPN (Instant Payment Notification) to mark the payment as complete.

---

3. **Building Base Classes**

   3.1 **WFOCU_Gateways**

   The main class, `WFOCU_Gateways`, provides gateway integration functions. Below are some of the key functions:

- `maybe_add_test_payment_gateway()`: Hooked onto `woocommerce_payment_gateways`. This function registers our Test Gateway by WooFunnels to the WooCommerce payment gateways to make it available on the checkout page for admin users.

- `get_integration()`: Retrieves the gateway integration object by providing its gateway key.

- `get_supported_gateways()`: Lists all the payment gateway keys supported by FunnelKit upsells. It also provides the filter hook `wfocu_wc_get_supported_gateways` to add external gateway integrations with upsells.

- `integration_autoload()`: Triggered by the native PHP function `spl_autoload_register()`. This function loads the payment gateway integration classes available in the gateways folder of upsells.

- `load_gateway_integrations()`: Hooked on `wp_loaded`. It loads all the supported payment gateway integration objects.

4. 3.2 **WFOCU_Gateway**

   This is an abstract base class that provides common functions for gateway integrations. All gateway integration classes must inherit from this base class.

---

4. **Integration Process Flow**

   4.1 **Gateway Integration Class Setup**

   To integrate a gateway with upsells for charging offers, we create a class for the gateway integrations and extend the `WFOCU_Gateways` class to start the integration. We define the gateway key in this file and add functions for tokenizing the payment method and saving created tokens in order meta for offer charging.

   4.2 **Tokenization**

   The first requirement for a gateway to be integrated with upsells is to obtain a token, which will be used to charge the offers. Most payment gateways that support tokenization provide a checkbox on the checkout page below the

Payment Card Fields. For example, Stripe offers the "Save payment information to my account for future purchases" option.

In the case of subscription products in the cart, most gateways hide this checkbox and enable force tokenization to save the token in order meta, user meta, or WooCommerce token tables. These tokens are then used to charge subscription renewal orders.

4.3 **Tokenization Filters**

We also use filter hooks to enable force tokenization in some cases. For example, in Stripe, we use the `wc_stripe_force_save_source` filter to enable tokenization and `wc_stripe_display_save_payment_method_checkbox` to hide the checkbox.

In some other cases, we modify the primary payment arguments sent to the gateway API to return a token for charging. For instance, with PayPal Standard, we use the `woocommerce_paypal_args` filter and add a return URL to our upsell page URL.

4.4 **has_token()**

Before redirecting to the first valid offer page of the decided funnel, we use this function to check if the payment method was successfully tokenized and if we have a valid token for charging the offer. We search for the token in its saved location in the order meta.

4.5 **process_charge()**

When the user accepts an offer, we call this function to charge the offer. In this function, we prepare the final payment arguments to send to the gateway API for charging using the created token. We collect most of the information from the parent order, like user billing and shipping details, and add the new order total as the offer package total along with the payment token for charging.

For Stripe, we prepare the source using the `prepare_order_source()` function, which collects the token from the order to prepare a source for charging. After getting a chargeable source, we call `create_intent()` to create an intent for charge according to the new SCA transaction rules. After creating the intent, we call `confirm_intent()` to capture the authorized charge.

4.6 **handle_result()**

After completing the `process_charge()` function, we typically receive a transaction ID if the transaction was successful. We then call `handle_result()` with a parameter of `true` for success, or `false` if the transaction fails.

4.7 **process_refund_offer()**

If the gateway integration allows for refund or void transactions for offer charges, we define this function to handle the refund process. In this function, we make API calls to the gateway to perform a refund transaction. If the refund is successful and we receive a refund transaction ID, we return the transaction. If an error occurs, we return `false`.

4.8 **get_transaction_link()**

If the gateway supports displaying transaction links on the order edit page, we add transaction links to the offer's refund metabox for each offer row. This function creates transaction links to show on the order edit page, with the transaction ID and order ID included.