verifone

https://verifone.cloud/docs/uk-gateway/threeds_js
Updated: 15-Apr-2022

3-D Secure JS

1.0 Setting up the 3DS Javascript

The JavaScript used in the browser flow performs most of the heavy lifting on behalf of the merchants. The JavaScript collects all the device data of the user's browser, communicates directly with the 3DS Server and handles the user experience of the cardholder during the challenge.

Merchant's back-end	Merchant's front-end	Payment Brand
1. Create Request JWT		
	2. Payments.setupComplete	
	3. Start payment	4. Interaction with the Payment Brand
	5. Payments.validated	
6. Validate Response JWT		

1.1 Add the JavaScript on website

The JavaScript can be added to your site as any other client side script, through a script tag. It is suggested to add the script after all your content before closing the HTML body tag.

Include the script:

```
<script src="https://songbirdstag.cardinalcommerce.com/edge/v1/songbird.js"></script>
```

The JavaScript URLs depend on the environment used:

Environment	URL
Sandbox	https://songbirdstag.cardinalcommerce.com/edge/v1/songbird.js
Production	https://you.will.receive.this.url.during.onboarding.js

1.2 Configure the JavaScript

Cardinal.configure is an optional function that allows to pass configuration object into the JavaScript. Not using this function to your integration will result to use the default configuration options. It is advised to call this function only once per page load and should be called before Cardinal.setup.

Root Level Configuration

Field	Type	Default	Description
timeout	int	8000	The time in milliseconds to wait before a request to Centinel API is considered a timeout
maxRequestRetries	int	1	How many times a request should be retried before giving up as a failure.
logging	object		
button	object		
payment	object		

Logging

Field	Type	Default	Description
level	string	off	The level of logging to the browser console. Enable this feature to help debug and implement Songbird. Possible Values: **off** - No logging to console enabled. This is the setting to use for production systems. **on** - Similar to info level logging, this value will provide some information about whats occurring during a transaction. This is recommended setting for merchants implementing Songbird **verbose** - All logs are output to console. This method can be thought of as debug level logging and will be very loud when implementing Songbird, but is the level needed when getting support from the Cardinal team.

Button

Field	Type	Default	Description
containerId	string	Cardinal-Payments	The HTML Id value of the container to inject all payment buttons into.

Payment

Field	Type	Default	Description
view	string	modal	What type of UI experience to use when Songbird injects payment brand UI elements into the page. Possible Values: · modal - Render as a modal window. This view type renders the payment brand over your page, making it feel separate from your page. · inline - Render inline to the page. This view type embeds the payment brand into the page making it feel like its a part of your website.

Field	Type	Default	Description
framework	string	cardinal	What kind of view framework should be used to render the payment brand. If your site is using a supported framework and you have custom styles applied to it, we will use that framework to make keep the consistent look and feel of your site. When using any other frameworks than 'cardinal' your site is responsible for including the framework assets including CSS, JavaScript, and any other additional files needed. Possible Values: **cardinal** - Use the custom Cardinal view framework built and maintained by CardinalCommerce. Songbird will handle all UI rendering and styles, no additional work is needed. **bootstrap3** - Use bootstrap 3 modal to render the UI elements.
displayLoading	boolean	false	A flag to enable / disable a loading screen while requests are being made to 3DS Server API services. This can provide feedback to the end user that processing is taking place and they should not try to reload the page, or navigate away.

For example, to control the logging volume from the library, use the Cardinal.configure function, as seen below:

Cardinal.configure example

```
Cardinal.configure({
    logging: {
        level: "on"
    }
});
```

Example of using all configuration option possible

```
{
    timeout: 8000,
    maxRequestRetries: 2,
    button:{
        containerId: 'Cardinal-Payments'
    },
    logging:{
        level: 'on'
    },
    payment:{
        view: 'modal',
        framework: 'bootstrap3',
        displayLoading: false
    }
}
```

1.3 Listen for events

This function sets up an event subscription with the JavaScript to trigger a callback function when the event is triggered by the JavaScript. A valid event subscription requires a namespace and a callback function to be run

when the event is triggered. Calling this function with the same namespace multiple times will result in callback being triggered multiple times.

The syntax of the function is:

```
Cardinal.on(EVENT_NAME_SPACE, CALLBACK_FUNCTION);
```

The following sections discuss the events that a merchant can listen to.

1.3.1 payments.setupComplete

payments.setupComplete() is an optional event which should be called after the Cardinal.setup() function.

To listen the payments.setupComplete event:

```
Cardinal.on('payments.setupComplete', function(setupCompleteData){
    // Do something
});
```

If an error has happened during the Cardinal.setup() function, then the payments.setupComplete() will not be triggered. If your callback gets executed, you know that the JavaScript is available to run transactions. This function will receive 2 arguments that describe the loaded state of theJavascript and the current session identifier.

The following object is returned back to the merchant on the payments.setupComplete event as the first argument.

Key	Type	Description
sessionId	String	Merchant Consumer Session Id - This is the consumer's session id assigned to this user by 3DS Server API.
modules	Array of Module State Objects	An array of modules that were attempted to be loaded and their status. You can determine which payment brands were loaded successfully and which may have been configured on the merchant account but failed to load properly. For 3DS ‘cca’ will be returned.

Module State

Key	Type	Description
loaded	String	Merchant Consumer Session Id - This is the consumer's session id assigned to this user by 3DS Server API.
modules	Array of Module State Objects	An array of modules that were attempted to be loaded and their status. You can determine which payment brands were loaded successfully and which may have been configured on the merchant account but failed to load properly. For 3DS ‘cca’ will be returned.

Example of payments.setupComplete data object

```
{
```

```
  "sessionId": "0_4f85c155-6604-4056-8957-7090412af179",
  "modules": [{
    "module": "CCA",
    "loaded": true
  }]
}
```

1.3.2 payments.validated

payments.validated event is triggered when the transaction has been finished and the control is given back to the merchant page. It includes data on how the transaction attempt ended that should be used in the logic for reviewing the results and decision making, how to proceed with the transaction.

If the payments.validated is successful (`"ActionCode": "SUCCESS"`) then the data needed to proceed with the payment Authorisation (CAVV, ECIFlag, XID, Enrolled, PAResStatus, SignatureVerification) will be included in the "Payment" object.

To listen the payments.validated event:

```
Cardinal.on('payments.validated', function
(decodedResponseData, responseJWT){
    // Do something
});
```

The payments.validated event consists of the Response Data and the Response JWT.

Field	Type	Required	Desc
Response Data	JSON object	R	The decoded Payload claim from the response JWT. This is a convenience value that is passed back to the merchant for client side logic decision making. This object should not be used to send data to third parties, as its validity cannot be confirmed.
Response JWT	String	O	Response JWT from 3DS Server API service. This is where the data field came from except in edge cases where a JWT wasn't returned due to an error. The merchant should use the data within this value when sending any data to third parties, since the validity of this data can be confirmed server side by verifying the JWT signature.

The payments.validated can result into three different cases:

Type	Response Data	Response JWT	Description
Normal Processing	Present	Present	No issues encountered
Api Error	Present	Present	An error occurred but 3DS Server API was able to generate a response JWT. You can validate these error responses by validating the JWT as you would in a successful transaction

Type	Response Data	Response JWT	Description
Service Error	Present	Absent	An error was encountered but a response JWT was not generated. This could be many things including: · Request to 3DS Server API timed out. · Request JWT failed authentication at 3DS Server API. · 3DS Server API is unavailable to receive transactions. · JavaScript encountered an unrecoverable error

Response Data

At minimum the response data will include a base object as seen below. However, depending on what occurred in the response additional fields may be present.

Type	Description
ActionCode	The resulting state of the transaction. Possible values: **SUCCESS** - The transaction resulted in success for the payment type used. This would indicate the user has successfully completed authentication. **NOACTION** - The transaction was successful but requires in no additional action. This would indicate that the user is not currently enrolled in 3-D Secure, but the API calls were successful. **FAILURE** - The transaction resulted in an error. For example, with a 3DS transaction this would indicate that the user failed authentication or an error was encountered while processing the transaction. **ERROR** - A service level error was encountered. These are generally reserved for connectivity or API authentication issues. For example, if your JWT was incorrectly signed, or Cardinal services are currently unreachable.
Validated	This value represents whether transaction was successfully or not.
ErrorNumber	Application error number. A non-zero value represents the error encountered while attempting the process the message request.
ErrorDescription	Application error description for the associated error number.
Payment	Payment Object

The payment object for the 3DS transactions is:

Field Name	Description	Required/ Optional/ Conditional	Field Definition

Enrolled	Status of Authentication eligibility. Possible Values: **Y** = Yes- Bank is participating in 3-D Secure protocol and will return the ACSUrl **N** = No - Bank is not participating in 3-D Secure protocol **U** = Unavailable - The DS or ACS is not available for authentication at the time of the request **B** = Bypass- Merchant authentication rule is triggered to bypass authentication in this use case **NOTE**: If the Enrolled value is NOT Y, then the Consumer is NOT eligible for Authentication.		String (1)
CAVV	Cardholder Authentication Verification Value (CAVV) Authentication Verification Value (AVV) Universal Cardholder Authentication Field (UCAF). This value should be appended to the authorization message signifying that the transaction has been successfully authenticated. This value will be encoded according to the merchants configuration in either Base64 encoding or Hex encoding. A Base64 encoding merchant configuration will produce values of 28 or 32 characters. A Hex encoding merchant configuration will produce values of 40 or 48 characters. The value when decoded will either be 20 bytes for CAVV or 20 or 24 bytes if the value is AAV (MasterCard UCAF).	O	String (40)
ECIFlag	Electronic Commerce Indicator (ECI). The ECI value is part of the 2 data elements that indicate the transaction was processed electronically. This should be passed on the authorization transaction to the gateway/processor.	O	String (40)
PAResStatus	Transaction status result identifier. Possible Values: **Y** – Successful Authentication **N** – Failed Authentication **U** – Unable to Complete Authentication **A** – Successful Attempts Transaction	O	String (1)

SignatureVerification	Transaction Signature status identifier. Possible Values: **Y** - Indicates that the signature of the PARes has been validated successfully and the message contents can be trusted. **N** - Indicates that the PARes could not be validated. This result could be for a variety of reasons; tampering, certificate expiration, etc., and the result should not be trusted.	O	String (1)
XID	Transaction identifier resulting from authentication processing. **NOTE:** Gateway/Processor API specification may require this value to be appended to the authorization message. This value will be encoded according to the merchants configuration in either Base64 encoding or Hex encoding. A Base64 encoding merchant configuration will produce values of 28 characters. A Hex encoding merchant configuration will produce values of 40 characters.	O	String (40)
UCAFIndicator	Universal Cardholder Authentication Field (UCAF) Indicator value provided by the issuer. Possible Values: 0 - Non-SecureCode transaction, bypassed by the Merchant 1 - Merchant-Only SecureCode transaction 2 - Fully authenticated SecureCode transaction NOTE: This field is only returned for MasterCard transactions		String (1)
ACSTransactionId	Unique transaction identifier assigned by the ACS to identify a single transaction.	C	String (36)
ThreeDSServerTransactionId	Unique transaction identifier assigned by the 3DS Server to identify a single transaction.	C	String (36)
DSTransactionId	Unique transaction identifier assigned by the Directory Server (DS) to identify a single transaction. **NOTE**: Required for Mastercard Identity Check transaction in Authorization	C	String (36)

Below you some samples of different values returned to the payments.validated event are presented. These JSON objects would be the first argument and the Payload claim of the response JWT where a response JWT

was returned.

Successful Response example:

```
{
   "Validated": true,
    "Payment": {
        "Type": "CCA",
        "ProcessorTransactionId": "uAthLfEYg83iEverTlk0",
        "ExtendedData": {
            "CAVV": "AAABAWFlmQAAAABjRWWZEEFgFz+=",
            "ECIFlag": "05",
            "XID": "dUF0aExmRVlnODNpRXZlclRsazA=",
            "Enrolled": "Y",
            "PAResStatus": "Y",
            "SignatureVerification": "Y"
        }
    },
    "ActionCode": "SUCCESS",
    "ErrorNumber": 0,
    "ErrorDescription": "Success"
}
```

API Level Error (will include a Response JWT to be validated)

```
{
    "Validated": false,
    "ErrorNumber": 4000,
    "ErrorDescription":
"Validation Error A valid merchant consumer session ID is required.",
    "ActionCode": "ERROR",
    "Payment": {}
}
```

Service Level Error (will not include a Response JWT)

```
{
    "Validated": false,
    "ErrorNumber": 1000,
    "ErrorDescription":
"Error processing request. We have encountered an unexpected error.",
    "ActionCode": "ERROR",
    "Payment": {}
}
```

1.4 Initialise JavaScript

To initiate the communication with the server, call the Cardinal.setup() function. All the necessary pre-processing steps should have been completed by the time that the consumer is ready to checkout. Listen for the

payments.setupComplete event to get notified when the JavaScript has finished initializing (Section 5.4.1).

1.4.1 Set up the JavaScript

Cardinal.setup function informs the JavaScript what type of event you are planning to complete on the page it is running on and what files it needs to bootstrap to facilitate that event.

Field	Type	Required/ Optional	Description
Initialization Type	String	R	Tells Songbird.js which flow you're setting up
Initialization Data	JSON object	R	An object used to pass any additional required data to properly complete Songbird initialization. This object will differ from initialization type to initialization type.

Initialization types

Key	Description
init	Setup the necessary files to run the authentication. You should use this initialization type anytime you want to complete payer authentication flows. This type would typically be used on a cart page, or payment details collection page.
complete	Setup the necessary files to return the authorization result to Cardinal. You should use this initialization type if you only plan on returning the authorization / authentication results to Cardinal. This type would typically be used on an order complete page that renders an 'Order was successfully submitted' message.

Example of Cardinal.setup

```
Cardinal.setup("init", {
    jwt: document.getElementById("JWTContainer").value
});
```

A common way to pass your JWT into the JavaScript is to place its value into a hidden input on page load. Using Cardinal.setup() function you can look for that element and select its value.

Example of placing a JWT into a hidden input

```
<input type="hidden" id="JWTContainer" value="[Insert your JWT here]" />
```

1.5 Cardinal Continue

After the Lookup Response is returned, pass the ACSUrl (acs_url), Payload (payload), and TransactionId (transaction_id) and include them in the Cardinal.continue function in order to proceed with the authentication session. The Cardinal.continue will display a modal window and automatically post the consumer's session over to the Issuer’s URL (acs_url) for authentication.

The syntax for Cardinal.continue:

```
Cardinal.continue(PAYMENT_BRAND, CONTINUE_DATA, ORDER_OBJECT, NEW_JWT)
```

Field	Type	Required/ Optional	Description
Payment Brand	String	R	The payment brand to continue. For 3DS the value ‘cca’ should be passed.
Continue Object	JSON object	R	A JSON object containing all the necessary data to complete a 3DS post to an ACS to complete a 3DS transaction.
Order Object	JSON object	O	As Order Object pass the following object replacing the ‘authentication_id’ with the value received in the lookup response. Example: {"OrderDetails":{ "TransactionId" :"*transaction_id*"}}
JWT	String	O	A updated JWT to use while processing the transaction. This allows the merchant to switch JWT's between init and continue events.

Continue Object

Field	Type	Required/ Optional	Description
AcsUrl	String	R	The acs_url returned in the lookup response
Payload	String	R	The ‘payload’ field returned on the lookup response

Cardinal.continue will only work after the payments.setupComplete event has been triggered. Cardinal.continue is suggested to be run later in the flow if payments.setupComplete is not triggered yet.

Example of Cardinal.continue:

```
Cardinal.continue('cca',
{
    "AcsUrl":
"https://testcustomer34.cardinalcommerce.com/merchantacsfrontend/pareq.jsp?vaa=b&
,
    "Payload":
"eNpVUk1zgjAQvedXME7PJEFBVdKt1CECeDkVCk2PcfcnNjv8Kr+7tx4nlbGOcz/se6G1uMENPTPeeIz

 },
 {
    "OrderDetails":{
         "TransactionId" :"123456abc"

                        }
       }
);
```

1.6 BIN Detection

To successfully complete the 3DS Method, the Issuing bank should be contacted to receive the browser information before the authentication is started. Therefore, the BIN is required to be communicated to the JavaScript before sending the lookup request.

There are two ways to implement the BIN Detection to a merchant's web application:

1) Field Decorator This implementation is the simplest and recommended approach when the full PAN is available. A merchant may directly start the JavaScript, provide the PAN and allow for payments.setupComplete event to complete. A new attribute to the input field to identify which field it maps to within the Order Object needs to be added. The credit card number is mapped to the AccountNumber field, therefore for the BIN Detection the AccountNumber will be passed to the attribute 'data-cardinal-field'.

Field Decorator Example

```
<input type="text" data-cardinal-field="AccountNumber" id=
"creditCardNumber" name="creditCardNumber" />
```

The field decorator will attach an event listener to the element that will update the BIN as the cardholder types it in. The BIN value will be updated automatically if the cardholder changes cards or needs to correct an entry.

2) Event Based

The bin.process event is the recommended event base profiling the merchant uses an card that is stored on file. The merchant will need to provide a minimum of the first 6 (e.g. BIN) up to the full card number of the consumer (e.g. max of 19 digits). The more digits of the card number provided the better chances of matching if there is a corresponding EMV 3DS Method URL.

Bin.process example

```
Cardinal.trigger("bin.process", '1234567894561237');
```

1.7 Cardinal.trigger

Cardinal.trigger function triggers an event within Songbird. This is a way to actively send Songbird data instead of waiting passively for events to occur.

Cardinal.trigger syntax

```
Cardinal.trigger("EVENT_NAME_SPACE", 'DATA');
```

1.7.1 bin.process

For bin.process event described in BIN Detection.

Cardinal.trigger implementation example

```
Cardinal.trigger("bin.process", '1234567894561237')
  .then(function(results){
    if(results.Status) {

// Bin profiling was successful. Some merchants may want to only move forward wi

    } else {
        // Bin profiling failed
    }


// Bin profiling, if this is the card the end user is paying with you may start t

    Cardinal.start('cca', myOrderObject);
  })
  .catch(function(error){
    // An error occurred during profiling
  })
```

1.7.2 jwt.update

jwt.update is an event to allow the merchant to change the JWT at any point. This event will update the local cached order object within the JavaScript but it will not push anything to the Cardinal infrastructure. This removed the need to pass in a new JWT into an event such as Cardinal.start or Cardinal.continue.

Cardinal.trigger implementation example

```
Cardinal.trigger('jwt.update', 'my_new_jwt_value');
```