# Tutorial - Call UAPI's SSL::install_ssl Function in Custom Code



## Introduction

This tutorial creates custom code to call UAPI's `SSL::install_ssl` function. Due to their inherent complexities, SSL-related functions often present problems for third-party developers. This tutorial provides explanations for the additional steps required in order to successfully call SSL functions.

## Create custom `SSL::install_ssl` code

> **Note:**
> Select the tab below that corresponds to the desired coding language.

Perl PHP

### Set the strict pragma and use warnings.

These declarations instruct Perl to return errors if the file contains potentially-unsafe code.

> **Note:**
> You can omit the `warnings` declaration in production code, but we **strongly** recommend that you use it during development.

```
# Return errors if Perl experiences problems.
use strict;
use warnings;
```

### Set web request module dependencies.

We recommend that you set a dependency for one or more modules that allow you to perform web requests.

- This tutorial uses the `LWP::UserAgent` and `LWP::Protocol::https` modules.
- The `HTTP::Tiny` module also fulfills this need.

```
# Allow my code to perform web requests.
use LWP::UserAgent;
use LWP::Protocol::https;
```

### In this tutorial

### Related documentation

### Use UTF-8 encoding.

You must use UTF-8 encoding when you call UAPI functions. If you do not use the correct encoding, your code may result in wide character warnings.

```
# Use the correct encoding to prevent
wide character warnings.
use Encode;
use utf8;
```

---

### Use JSON formatting.

Declare the use of JSON formatting, to ensure that your code can properly decode JSON.

```
# Properly decode JSON.
use JSON;
```

---

### Use Base64 encoding.

Declare the use of the `MIME::Base64` module, which ensures that your code properly interacts with Base64-encoded authentication headers.

```
# Function properly with Base64
authentication headers.
use MIME::Base64;
```

---

### Provide authentication information.

Declare the variables for your cPanel account username and password in order to allow your code to authenticate with cPanel & WHM.

```
# Authentication information.
my $username = 'username';
my $password = '12345luggage';
```

---

### Set the API call location.

Set the `$request` variable to the URL for the desired API call.

```
# The URL for the SSL::install_ssl UAPI
function.
my $request =
"https://localhost:2083/execute/SSL/ins
tall_ssl";
```

---

### Allow HTTPS connections to unsigned services.

Set the `PERL_LWP_SSL_VERIFY_HOSTNAME` environment variable to `0` to allow HTTPS connections to unsigned services.

> **Warning:**
> You **must** include this step in your code. Services on the local host are **alw ays** unsigned.

```
# Required to allow HTTPS connections to
unsigned services.
# Services on localhost are always
unsigned.
$ENV{PERL_LWP_SSL_VERIFY_HOSTNAME} = 0;
```

---

### Create a new `UserAgent` object.

Use the `LWP::UserAgent` module's `new()` method to create a new `UserAgent` obj ect to use for the API call.

```
# Create a useragent object.
my $ua = LWP::UserAgent->new();
```

---

### Create authentication headers.

Use the new `UserAgent` object to set up authentication headers that use the `$usern ame` and `$password` values from lines 22 and 23.

```
# Add authentication headers.
$ua->default_header(
    'Authorization' => 'Basic ' .
MIME::Base64::encode("$username:$passwo
rd"),
);
```

---

### Read in the SSL certificate and key files.

Declare variables and set values for the SSL certificate and key files.

> **Note:**
> You must URI-encode the `cert` and `key` parameters' values.

```
# Read in the SSL certificate and key
file.
my ( $cert, $key );
{
    local $/;
    open ( my $fh, '<',
'/path/to/certificate.crt' );
    $cert = <$fh>;
    close $fh;

    open ( $fh, '<', '/path/to/key.key'
);
    $key = <$fh>;
    close $key;
}
```

---

### Create a POST request for the call.

Create a POST request that includes the function's required input parameters. The `$response` variable will contain the call's output.

```
# Make the call.
my $response = $ua->post($request,
    Content_Type => 'form-data',
    Content => [
        domain => 'example.com',
        cert   => $cert,
        key    => $key,
    ],
);
```

---

### Create an object to decode the call's output.

Create an object to decode the call's JSON-formatted output. You will use this object to sort and pretty-print the JSON.

```
# Create an object to decode the JSON.
# Sorted by keys and pretty-printed.
my $json_printer =
JSON->new->pretty->canonical(1);
```

---

### UTF-8 encode and then decode the call's output.

Use the `Encode` module to UTF-8 encode the call's output in the `$response` variable . Then, use the `JSON` module to decode the output.

> **Note:**
> If you do not UTF-8 encode this output, you may receive wide character warnings.

```
# UTF-8 encode before decoding to avoid
wide character warnings.
my $content =
JSON::decode_json(Encode::encode_utf8($
response->decoded_content));
```

---

### Print the function's output.

Use the object that line 65 created to pretty-print and sort the JSON output. Then, use the `Encode` module to UTF-8 encode the pretty-printed and sorted JSON, and print

the final result.

> **Note:**
> If you do not UTF-8 encode this output, you may receive wide character warnings.

```
# Print output, UTF-8 encoded to avoid
wide character warnings.
print
Encode::encode_utf8($json_printer->enco
de($content));
```

### Initiate error logging.

Set error reporting to `E_ALL` in order to log all errors and warnings.

> **Note:**
> You can omit this step in production code, but we **strongly** recommend its use during development.

```
// Log everything during development.
// If you run this on the CLI, set
'display_errors = On' in php.ini.
error_reporting(E_ALL);
```

---

### Provide authentication information.

Declare the variables for your cPanel account username and password in order to allow your code to authenticate with cPanel & WHM.

```
// Declare your username and password
for authentication.
$username = 'example';
$password = 'luggage12345';
```

---

### Define the API call.

Define the API host, and the URL for the desired API call.

```
// Define the API call.
$cpanel_host = 'localhost';
$request_uri =
"https://$cpanel_host:2083/execute/SSL/
install_ssl";
```

---

### Specify the locations of the SSL certificate and key files.

Declare variables and set values for the SSL certificate and key files.

> **Note:**
> This example manually defines the filename and destination. You can also use functions or the `$argv` array to pass these values.

```
// Define the SSL certificate and key
files.
$cert_file =
realpath("/path/to/cert.crt");
$key_file =
realpath("/path/to/key.key");
```

---

### Set up the payload to send to the server.

Set up the payload to send to the server. This data includes the function's required input parameters.

> **Note:**
> You must URI-encode the `cert` and `key` parameters' values.

```
// Set up the payload to send to the
server.
$payload = array(
    'domain' => "example.com",
    'cert'   =>
file_get_contents($cert_file),
    'key'    =>
file_get_contents($key_file)
);
```

---

### Set up the cURL request object.

Set up a cURL request object that uses the specified `$username` and `$password` variables.

**Notes:**

- Only use the `CURLOPT_SSL_VERIFYHOST` and `CURLOPT_SSL_VERIFYPEER` options with self-signed or expired certificates.
- `localhost` **always** uses a self-signed certificate.

```
// Set up the cURL request object.
$ch = curl_init( $request_uri );
curl_setopt( $ch, CURLOPT_HTTPAUTH,
CURLAUTH_BASIC );
curl_setopt( $ch, CURLOPT_USERPWD,
$username . ':' . $password );
curl_setopt( $ch,
CURLOPT_SSL_VERIFYHOST, false );
curl_setopt( $ch,
CURLOPT_SSL_VERIFYPEER, false );
```

---

### Create a POST request for the call.

Create a POST request that includes the function's required input parameters and uses the `$payload` variable from line 19.

```
// Set up a POST request with the
payload.
curl_setopt( $ch, CURLOPT_POST, true );
curl_setopt( $ch, CURLOPT_POSTFIELDS,
$payload );
curl_setopt( $ch,
CURLOPT_RETURNTRANSFER, true );
```

---

### Make the call.

Use the `curl_exec()` method to make the API call. Then, use the `curl_close()` method to close the cURL object that you created.

```
// Make the call, and then terminate the
cURL caller object.
$curl_response = curl_exec( $ch );
curl_close( $ch );
```

---

### Decode and validate the output.

Use the `json_decode()` method to decode the `$curl_response` value, which contains the call's output. Then, validate that output.

- Lines 43 through 45 of the example below check to ensure that the call returned output.
- Lines 46 through 48 of the example below ensure that there were no errors.

```
// Decode and validate output.
$response = json_decode( $curl_response
);
if( empty( $response ) ) {
    echo "The cURL call did not return
valid JSON:\n";
    die( $response );
} elseif ( !$response->status ) {
    echo "The cURL call returned valid
JSON, but reported errors:\n";
    die( $response->errors[0] . "\n" );
}
```

---

### Print output.

Print the validated output and exit.

```
// Print and exit.
die( print_r( $response ) );
```

## Completed code

When you finish this tutorial, your code will resemble the following examples:

Click to view the complete Perl code...

```
#!/usr/local/cpanel/3rdparty/bin/perl

# Return errors if Perl experiences
```

```
problems.
use strict;
use warnings;

# Allow my code to perform web
requests.
use LWP::UserAgent;
use LWP::Protocol::https;

# Use the correct encoding to prevent
wide character warnings.
use Encode;
use utf8;

# Properly decode JSON.
use JSON;

# Function properly with Base64
authentication headers.
use MIME::Base64;

# Authentication information.
my $username = 'username';
my $password = '12345luggage';

# The URL for the SSL::install_ssl
UAPI function.
my $request =
"https://localhost:2083/execute/SSL/i
nstall_ssl";

# Required to allow HTTPS connections
to unsigned services.
# Services on localhost are always
unsigned.
$ENV{PERL_LWP_SSL_VERIFY_HOSTNAME} =
0;

# Create a useragent object.
my $ua = LWP::UserAgent->new();

# Add authentication headers.
$ua->default_header(
    'Authorization' => 'Basic ' .
MIME::Base64::encode("$username:$pass
word"),
);

# Read in the SSL certificate and key
file.
my ( $cert, $key );
{
    local $/;
```

```
    open ( my $fh, '<',
'/path/to/certificate.crt' );
    $cert = <$fh>;
    close $fh;

    open ( $fh, '<',
'/path/to/key.key' );
    $key = <$fh>;
    close $key;
}

# Make the call.
my $response = $ua->post($request,
    Content_Type => 'form-data',
    Content => [
        domain => 'example.com',
        cert   => $cert,
        key    => $key,
    ],
);

# Create an object to decode the
JSON.
# Sorted by keys and pretty-printed.
my $json_printer =
JSON->new->pretty->canonical(1);

# UTF-8 encode before decoding to
avoid wide character warnings.
my $content =
JSON::decode_json(Encode::encode_utf8
($response->decoded_content));

# Print output, UTF-8 encoded to
avoid wide character warnings.
```

```
print
Encode::encode_utf8($json_printer->en
code($content));
```

Click to view the complete PHP code...

```
<?php
// Log everything during development.
// If you run this on the CLI, set
'display_errors = On' in php.ini.
error_reporting(E_ALL);

// Declare your username and password
for authentication.
$username = 'example';
$password = 'luggage12345';

// Define the API call.
$cpanel_host = 'localhost';
$request_uri =
"https://$cpanel_host:2083/execute/SS
L/install_ssl";

// Define the SSL certificate and key
files.
$cert_file =
realpath("/path/to/cert.crt");
$key_file =
realpath("/path/to/key.key");

// Set up the payload to send to the
server.
$payload = array(
    'domain' => "example.com",
    'cert'   =>
file_get_contents($cert_file),
    'key'    =>
file_get_contents($key_file)
);

// Set up the cURL request object.
$ch = curl_init( $request_uri );
curl_setopt( $ch, CURLOPT_HTTPAUTH,
CURLAUTH_BASIC );
curl_setopt( $ch, CURLOPT_USERPWD,
$username . ':' . $password );
curl_setopt( $ch,
CURLOPT_SSL_VERIFYHOST, false );
curl_setopt( $ch,
CURLOPT_SSL_VERIFYPEER, false );
```

```
// Set up a POST request with the
payload.
curl_setopt( $ch, CURLOPT_POST, true
);
curl_setopt( $ch, CURLOPT_POSTFIELDS,
$payload );
curl_setopt( $ch,
CURLOPT_RETURNTRANSFER, true );

// Make the call, and then terminate
the cURL caller object.
$curl_response = curl_exec( $ch );
curl_close( $ch );

// Decode and validate output.
$response = json_decode(
$curl_response );
if( empty( $response ) ) {
    echo "The cURL call did not
return valid JSON:\n";
    die( $response );
} elseif ( !$response->status ) {
    echo "The cURL call returned
valid JSON, but reported errors:\n";
    die( $response->errors[0] . "\n"
);
}
```

```
// Print and exit.
die( print_r( $response ) );
?>
```