# Straylight manual: Alpha release 10/9/2014

## What is Straylight?

Straylight is a secure and extensible machine interface for ImpressCMS. It allows you to authorise individual client devices, such as a mobile phone or Arduino micro controller, to issue remote commands to your site **without** “logging in”. An authorised device can alter the configuration of your site or ask it to do something, for example, close the site, harden the security posture or turn on debug mode. You can add a new command with a few minutes work.

## How does it work?

1. Remote devices are individually registered within the module with a unique 256-bit pre-shared key (64 hexadecimal characters provides this, ie. 0-9 and a-f. To get such a key visit [www.grc.com/passwords](http://www.grc.com/passwords) and copy the output from the first box). GRC is a reputable, trusted source on security issues.
2. A toggle is available on the client administration page to ‘authorise’ or ‘de-authorise’ registered devices to access the Straylight interface. A device that is registered but de-authorised cannot access the interface.
3. Remote devices send commands to a handler file (target.php) via http POST request.
4. The integrity and authenticity of a command is assured using a hash-based message authentication code (HMAC, SHA256). The remote device supplies a HMAC of the concatenated command parameters in the request, which is calculated using the pre-shared key. Straylight makes the same calculation on the submitted request parameters using its own copy of the pre-shared key. If the HMACs match then the request is authentic and has not been modified.
5. To prevent replay attacks, commands contain the following:
   1. **Timestamp**: Commands expire after an admin-configurable period of time (default 10 minutes, see module preferences), and will also be discarded if they precede current server time.
   2. **Counter**: The client should increment the counter field with each command it issues. Straylight tracks client counters and will reject a request if the counter is lower than the stored (last seen good) value.
   3. **Random text chunk**: Since commands contain a small and fairly repetitive or predictable set of data, a random chunk of text is included to ensure that the HMAC or key are kept difficult to forge. The text also helps pad out the request, as for optimum security the minimum length should be 256 characters.

## Available remote commands

Only a few commands are presently available for demo purposes. They are:

* **checkPulse**: Indicates the site is up by returning an ‘ok’ http response code (200).
* **closeSite**: Turns the site off. (There is no openSite yet).
* **clearCache**: Deletes the contents of /cache and /templates\_c, but leaves index files intact.
* **debugOn**: Activates developer dashboard (inline).
* **debugOff**: Deactivates developer dashboard.
* **lockdown**: Alters the configuration of your site to help it resist casual abuse by troublemakers:
  + New user registrations require admin approval.
  + Admins advised of new registrations.
  + Minimum password length set to 13 characters.
  + Password security requirements set to strong.
  + Minimum username length set to 5 characters.
  + Multiple login disallowed.
  + Remember me login feature disabled.
  + Self-deletion of user accounts disabled.
  + Change of display name disabled.
  + Change of email address disabled.
  + Display of external images / html in user signatures disabled.
  + Upload of custom avatars disabled.
  + IP bans enabled.
  + Comments closed for all modules.
  + CAPTCHA enabled on forms, including registration form.
  + Minimum search keyword length set to 5 characters.
  + GZIP compression disabled to reduce processor load.

## Extending Straylight - adding more remote commands

Adding a new command to Straylight is very easy. Simply modify ClientHandler.php as follows:

1. Add the new command to the list specified in the $valid\_commands array. The command should only be alphabetic characters only or validation will fail.
2. Add a corresponding case statement to the switch in execute\_remote\_command(), containing the logic for implementing your new command.

That’s it!

## Implementing Straylight in a remote device

1. Remote clients submit commands as http POST requests to http://yoursite.com/modules/straylight/target.php.
2. Each request must contain the field names and values described in the table below.

|  |  |
| --- | --- |
| **Field name** | **Value** |
| client\_id | Integer. This is the ID generated for the client by the Straylight module (available from the client admin page). |
| command | String. The value must be alphabetical characters only and be one of the following:   * **checkPulse**: Indicates the site is up by returning an ‘ok’ http response code (200). * **closeSite**: Turns the site off. (There is no openSite yet). * **clearCache**: Deletes the contents of /cache and /templates\_c, but leaves index files intact. * **debugOn**: Activates developer dashboard (inline). * **debugOff:** Deactivates developer dashboard. * **lockdown** |
| counter | Integer. The client must store the value of the counter and increment it each time a new request is sent. The Straylight module tracks this value for each client, and will reject a request if the counter is lower than the last value it saw. This is to prevent a replay attack. |
| timestamp | Integer. Straylight will reject requests that exceed a certain age (default is 10 minutes, set in module preferences). Both the server and client clocks must be accurately set. |
| random | String of random text. The only requirements are that it is i) RANDOMLY GENERATED and ii) has a minimum length of 64 hex or printable ASCII characters. |
| hmac | Alphanumeric string (output by the SHA256 HMAC algorithm). To calculate the HMAC, you must:   1. Concatenate the other request field values **in the order** **specified below** and pass them as input to the algorithm, using the pre-shared key. 2. Truncate the 256 bit output to 128 bits. This means: Keep the leftmost 32 characters of the output and discard the rest. This is part of the HMAC spec and is done because it represents the birthday attack bound for the HMAC-SHA-256+ algorithms (so the real world security is 128 bits). The first 32 characters of the HMAC are used as the authenticator.   As an example, in PHP this can be done:  $hmac = $data ='';  $data = $client\_id . $command . $counter . $timestamp . $random;  $hmac = substr(hash\_hmac('sha256', $data, $key, FALSE), 0, 32);  Where $key is the pre-shared key registered against the client in the Straylight module. |

## Security issues

The main security issues are:

1. The security of the system is only as good as the key. You **must** use a RANDOM 256 bit pre-shared key for each device. Nothing else is acceptable. If you can’t be bothered creating a high-quality random key, and I do mean RANDOM (not your dog’s name or some semi-random rubbish you typed in yourself), then **do not use this module**.
2. Pre-shared key length should be 64 hexadecimal characters (ie. in the range 0-9, a-f). Each hexadecimal character encodes 4 bits of binary data (64 \* 4 = 256 bits). A shorter key will weaken the security of the HMAC and there is no evidence that a longer key is stronger. You can generate a good random 256 bit hexadecimal key easily by visiting <https://www.grc.com/passwords> and copying the output from the first box. Done!
3. The pre-shared key must be kept secret. You should consider the security of the remote device the key is stored on: How likely is it to be lost, stolen, or accessed by someone nasty? (Note that the presently available commands do not contain anything that could permanently damage or exploit your site, but that might change in future).
4. The random chunk of text is important to prevent certain types of attacks against the HMAC, which is otherwise calculated from a very small and repetitive (ie. predictable) data set. However, you should be aware that many small devices can’t generate acceptable pseudo-random output for this purpose. For example, one Arduino micro controller I looked at generates “random” numbers from a fixed internal string only about 3,000 characters long. The distribution of characters *in the string* is random but the sequence is fixed and reused over and over, so if you know the starting position you can predict what the next output will be! So it is best to look into the pseudo-random number generation capabilities of client devices in some detail. If they are inadequate you might consider generating a huge blob of quality random text elsewhere and storing it on the client for later use, discarding the “used” random data as you go (NEVER reuse such data).

## Future development

In the short term I intend to add some data logging features to store arbitrary data sent in by authorised devices (eg. environmental monitoring data or telemetry). In the long term I will add a permissions system to control which commands an individual device has access to, so that different devices can be assigned different roles.

## Further reading

Krawczyk, H., Bellare, M., Canetti, R. (1997). RFC 2104. HMAC: Keyed-Hashing for Message Authentication. http://tools.ietf.org/html/rfc2104.

Kelly, S., Frankel, S. (2007). RFC 4868. Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec: http://tools.ietf.org/html/rfc4868.