# State management

Many applications involve passing state between pages or (RPC) requests. A developer needs to be aware when state is passed in such a way that the client can see or modify it (eg. URL variables, cookies, REST parameters) and take proper precautions to prevent leakage of data or unauthorized access to a website or system.

WebHare offers some APIs to help encrypt these values when they are passed 'through' a client but even with encryption you should be careful of 'replay' attacks where the client reuses an encrypted value without having to actually understand it - you might not be able to read the session cookie you stole from a sysop, but it may still provide you with the same access if you can use it.

## Cookies and (URL) variables

Cookies should have the httponly and secure flags whenever possible (%UpdateWebCookie already sets 'httponly' by default). Avoid Javascript-readable cookies even if they do not contain any sensitive data at all - it will save you from having to explain their harmlessness during a security audit.

You can ask UpdateWebCookie for an encrypted cookie (which can be read using %GetDecryptedWebCookie) but you will still need to be careful about replay attacks by whoever holds their value.

## Further tips

- Do not consider a WRD_GUID by itself to be proof of anything - they are too easily leaked. If you need a secure way to identify a WRD entity, encrypt the guid
- Scopes for %EncryptForThisServer must be unique for each different use, attackers may try to inject an encrypted token into a different scope to see if anything can be learned from it.
- Do not worry about variable being passed between screens inside a Tollium application - unlike RPC calls, data passed in function calls inside a tollium (eg the data passed to a LoadScreen/RunScreen) is not available to a client.

# Injection and cross-site scripting

WebHare does a lot to prevent injections and XSS by design: HareScript's SQL integration isolates you from the raw database, the Witty template language has sane defaults to prevent most common cases of incorrect encoding. But, you still need to be aware of potential issues and avoid dangerous patterns:

## SQL Injections

Any use of INSERT, SELECT, UPDATE and DELETE as part of the HareScript language itself is safe as these expressions are automatically converted to parameterized SQL statements before being passed to the database.

However, if you try to directly interface with databases by sending raw statements, you will still need to take the proper precautions and use parameter or escaping if you accept external input. We recommend just using HareScript SQL wherever possible, and using %DynQuery if you need to build queries at runtime.

## Path injections

Do not generate file- or pathnames based on user input - use wrapped blob records (members/attributes of file/image type which store filenames inside the record) where possible, or generate names based on generic UUIDs or time-of-day.

You should especially avoid writing anything to the filesystem that was originally supplied by a user, as you would generally also lose transactional integrity and backup coverage.

## Encoding in Witty

If you find yourself needing to specify an explicit encoding in witty (eg `[title:html]` or worse, `[data:none]`) doublecheck if there isn't a way to avoid it. Be especially careful when using the :url encoding - it's rarely the right thing to do. URLs should generally be constructed in HareScript, not directly in Witty.

The most common exception to this is the textarea, which generally needs a :value encoding for its content (ie: <textarea>[currentdata:value]</textarea>)

## Encoding in HareScript

The text EncodeXXX functions (eg EncodeHTML, EncodeURL, EncodeValue) in HareScript are easy to confuse or forget. You can often avoid these:

- If you're building a URL, consider %UpdateURLVariables (and the other url.whlib functions)
- If you find yourself needing EncodeHTML or EncodeValue, consider using Witty
- Avoid generating HTML directly from HareScript. Again, consider a Witty template

## Generating HTML in JavaScript

Avoid innerHTML in JavaScript. Use textContent where possible. Consider using JSX or other templating solutions if you're building complex DOM nodes in JavaScript

# Sensitive data

WebHare offers some tooling to deal more safely with various sensitive or personal (PII) data you may process.

## Auditing

WebHare ships with a few reporting mechanisms to help you find potentially sensitive data and audit who has access to this data.

- Use `wh gdprscan` on the commandline to get an overview of WRD schemas and database tables that could potentially contain personal data.
- Walk through the "Objects and Rights" view in User and rights management to see users with (implicit) access to sensitive data.
- Publisher Search can search for forms by retention period. Watch for forms with suspiciously long retention periods

## Automatic deletion

You can mark files for automatic deletion at a certain date by setting the deletion property in the lifecycle metadata. Files such as cached import data or WRD schema backups that may contain sensitive data should be marked for automatich deletion to ensure they can't easily linger around even when copied or synced between WebHare servers (the lifecycle metadata is cloned too when copied).

You can find the file's planned deletion date in its object properties on the *Tasks* tab (sysop and supervisor only) and set it using SetInstanceData in code:

```
fileobj->SetInstanceData("http://www.webhare.net/xmlns/publisher/lifecycle",
  [ deletion := AddDaysToDate(14, GetCurrentDatetime())
  ]);
```