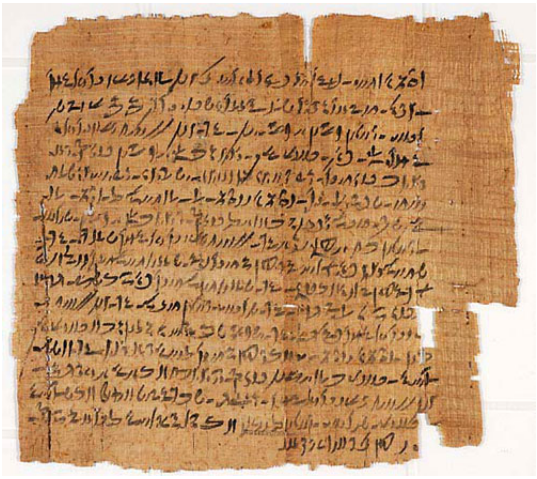
université de BORDEAUX

Manuel utilisateur

Application Web de Visualisation et de Manipulation de Fragments de Documents Anciens



GUEUX Elisabeth
MIALON Salomé
PIET Quentin
POLIN Axel

13 Mai 2019

Table des matières

Chapitre 1

Interface graphique

1.1 Les différents éléments de l'interface

1.1.1 Entête

1- Canvas : renvoie à l'interface de visualistaion et manipulation des fragments.

2- Créer un répertoire : permet de créer un nouvel utilisateur/projet. Une fois créé, renvoie à la page d'authentification Rentrer les nouveaux identifiants pour se connecter à la nouvelle session.

3- Sauvegarder l'assemblage : permet de sauvegarder l'assemblage créé dans le canvas à partir des fragments et de l'importer sous la forme d'une entité unique.

4- Log out : permet de quitter la session actuelle.

1.1.2 Panneau latéral

Le panneau latéral comporte les miniatures et les références des papyrus utilisables pour une session donnée.

importation des fragments dans le canvas

Pour afficher un fragment dans le canvas :

1- effectuer un glisser-déposer de la miniature du fragment d'intérêt vers le canvas. L'image affichée est la version recto-couleur (RCL) du fragment.

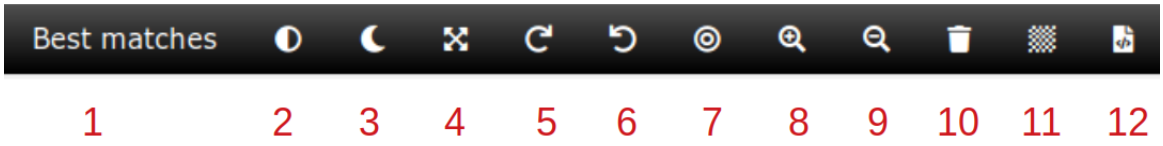
Nb : Cette fonctionnalité n'est pas utilisable sur les appareils tactiles.

2- Cliquer sur cette miniature affiche une liste de 4 versions disponible pour le fragment désiré (R/V : IR/CL). Cliquer sur une des 4 options permet d'importer la version correspondante du fragment.

modifier la version affichée (fragment par fragment)

Cette option permet de visualiser différentes version d'un fragment, sans modifier la version de l'ensemble des fragments affichés dans le canvas.

1.1.3 Barre de fonctionnalités



1- Best matches : retourne les fragments les plus susceptibles d'être associés avec le fragment sélectionné ainsi que les scores de correspondance associés aux fragments retournés. Cliquer sur une miniature afin d'importer l'image correspondante dans le canvas. Les scores sont obtenus via l'exécution d'algorithmes de traitement d'images.

2- Mode de visualisation : permet de modifier la version affichée pour l'ensemble des fragments présents dans le canvas.

3- Dark/light mode : permet de modifier la couleur du canvas (bleu sombre/blanc).

4- Désassemblage : permet de désassembler un assemblage préalablement généré à partir de plusieurs fragments.

5- Rotation vers la droite : permet de faire pivoter le fragment sélectionné vers la droite.

6- Rotation vers la gauche : permet de faire pivoter le fragment sélectionné vers la gauche.

7- Remise à l'échelle : permet de retrouver la taille que présentaient les fragment au moment de leur importation dans le canvas.

8- Zoom avant : effectue un zoom avant sur l'ensemble des fragments.

9- Zoom arrière : effectue un zoom arrière sur l'ensemble des fragments.

10- Supprimer une image : supprime le fragment sélectionné du canvas. Il est toujours possible de le ré-importer à partir du panneau latéral.

11- Treshold : effectue une opération de seuillage sur le fragment sélectionné et retourne l'image obtenue.

12- Affichage des méta-données : Affiche les informations sur la langue et les dimensions du fragment sélectionné.

1.2 fonctionnalités clavier

Chapitre 2

Serveur

2.1 Avant-propos

Cette section est à destination des développeurs. Sera abordé ici : les considérations de sécurité côté serveur, le protocole d'intégration de scripts côté serveur, le protocole d'ajout de fonctionnalités dans les composants d'Area côté client. Enfin, vous trouverez une liste de liens vers les références des technologies utilisées.

Pour toute question, réclamation ou exploit découvert veuillez poster une issue sur Papyrus_UI ou contacter directement Axel Polin à univ_apolin@protonmail.com

2.2 Discussion concernant la sécurité

Avertissement : les développeurs initiaux ont sécurisé le serveur à l'aide de paquets et de configuration de base. Cette note de sécurité concerne uniquement la partie NodeJS. Aucune configuration n'a été construite pour la partie hardware du serveur, ni pour la partie firewall ou tout autre logiciel extérieur à NodeJS. Cette note est informative. Cela signifie qu'elle ne couvre pas tout les aspects de sécurité de l'information et qu'elle ne remplace pas une politique de cyberdéfense approprié.

2.2.1 Utilisation exclusive du protocole HTTPS

Le serveur est configuré pour fonctionner exclusivement en HTTPS ce comportement ne devrait pas être modifié.

2.2.2 Certificats de sécurité

Dans un soucis d'harmonisation entre les différents outils de sécurité utilisés dans cette application l'algorithme de chiffrement des données utilisé est *Elliptic curve digital signature algorithm (ECDSA)*. La courbe utilisée dans algorithme ECDSA est secp256k1. Cette courbe devrait être robuste jusqu'en 2050 environ (voir Référentiel Général de Sécurité version 2.0 Annexe B1 Mécanismes cryptographiques Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques Version 2.03 du 21 février 2014) Ces mesures devront être respecté à l'avenir.

Voici le protocole permettant de générer des certificats (ainsi que leur clé correspondante) :
openssl ecparam -name secp256k1 -out secp256k1.pem
puis

openssl ecparam -in secp256k1.pem -genkey -noout -out secp256k1-key.pem

2.2.3 Sécurisation des en-tête HTTP

Voici les paramètres de requête fixés par défaut dans le serveur :

- Cache-Control : no-store, no-cache, must-revalidate, proxy-revalidate
- Pragma : no-cache
- Expires : 0
- Surrogate-Control : no-store
- X-DNS-Prefetch-Control : off
- X-Frame-Options : SAMEORIGIN
- X-Download-Options : noopen
- X-Content-Type-Options : nosniff
- X-XSS-Protection : 1 ; mode=block
- X-CSRF-Token : $argon2i$v=19$m=4096,t=3,p=1$<HASH>
- Strict-Transport-Security : max-age=15552000 ; includeSubDomains
- Accept-Ranges : bytes
- ETag : W/"XXXXXXXX"
- Public-Key-Pins : 'pin-sha256="<KEY1BASE64>" ; pin-sha256="<KEY2BASE64>" ; pin-sha256="<KEY3BASE64>" ; max-age=36500'

Il s'agit d'un en-tête classique avec les éléments de sécurité de base : protection contre les attaques de type XSS [1], CSRF [2]. Le dernier paramètre permet de changer de certificat si l'un d'entre eux est compromit (remplacer <KEY1BASE64> par la clé codé en Base64). L'en-tête X-CSRF-Token permet de fournir une clé unique au client depuis le serveur. Ceci permet de protéger le client des attaques CSRF. Les paramètres de cet en-tête ne devrait pas être supprimé.

2.2.4 Protection contre les attaques CSRF

La protection contre les attaques CSRF est construite de la manière suivante : le serveur et le client partage une clé secrète présente dans chaque échange risqué. Ici la phrase secrète est intégrée aux en-têtes HTTP dans le paramètre X-CSRF-Token. Le serveur enregistre la clé dans le fichier de session de l'utilisateur. Il compare ensuite la clé enregistré à la clé fournit par le client dans ses requêtes. Les requêtes sensibles protégées par ce procédé sont toutes les requête POST sauf celle du login et du prototype pour les métadonnées.

Construction côté serveur (requête POST) mise en place :

```
// Make CSRF protection token;
var secret = crypto.randomBytes(32);
var salt = crypto.randomBytes(32);
req.session.csrfToken = await argon2i.hash(secret, salt);
// END CSRF TOKEN GENERATION;
```

Construction côté serveur (requête POST) verification :

```
if (!req.body.csrf) return res.sendStatus(400);
var csrf = req.body.csrf;
if (csrf !== req.session.csrfToken) return res.sendStatus(400);
```

1. https ://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.md
2. https ://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

Côté client (exemple pour suppression de repertoire utilisateur) :

```
$http({
    type: "GET",
    url: '/csrf',
  }).then(function(response) {
    var dataToSend = JSON.stringify({"csrf" : response.headers('X-CSRF-Token')});
    // ^- Create a JSON string with concatenation of img txt and the Area.images array (see Area);

    $http({ // Post all datas to server;
      method : "POST", // Method accepted by the server is POST;
      url : '/secure/rd', // The URL where the server accept this type of POST;
      data : dataToSend, // Put the data to send here;
      dataType: 'json', // The type of data is JSON;
      contentType: 'application/json; charset=utf-8' // Content-Type for the server and the communication;
    }).then(function(response) {
      alert("Votre repertoire a ete supprime !");
       window.location.reload();
    }, function(response) {
      alert("Error while deleting working directory !!"); // If RD fail alert user;
    });

});
```

Login CSRF : avertissement pour le futur

Un utilisateur non enregistré (login) n'est pas protégé contre les attaques CSRF tant qu'il n'est pas logué. Ceci concerne tout les utilisateurs même ceux qui ont un projet enregistré dans le serveur. Le protocole de protection CSRF respecte les recommandations de sécurité actuelle cependant il faudra peut-être corriger cela dans de futures versions selon les prochaines recommandations de sécurité.

2.2.5 Protection des mots de passes

Les mots de passe sont hashés dans le serveur à l'aide de l'algorithme Argon2i [3]. Cet algorithme est la nouvelle référence des fonctions de hashage. Sauf choix délibéré celui-ci devrait être conservé.

2.2.6 Choix de développement : non utilisation de base de données

Les utilisateurs et les projets sont gérés à l'aide de fichier JSON. Étant donné que le trafic prévu ne sera pas très élevé il n'a pas été nécessaire de paramétrer des bases de données. De plus, l'absence de base de données supprime le risque de SQLi [4]. Les futures développeur pourront modifier ce comportement s'il le souhaite il n'y a pas de contre-indication dans la mesure où les règles contenus dans ces fichiers sont respectés : SQLi prévention Cheat Sheet , Paramétrage Requête

3. https ://password-hashing.net/
4. https ://www.owasp.org/index.php/SQL_Injection

2.2.7 Avertissements d'OWASP ZAP et BurpSuite Professional

ZAP

Zap détecte un seul problème de sécurité : *Inclusion de fichier source JavaScript inter-domaine* son statut est priorité basse. En effet, l'application importe des fichiers JavaScript depuis d'autres domaine que celui du serveur. Cela ne pose pas de problème tant que les conditions suivantes sont respéctées : *S'assurer que les fichiers source JavaScript soient chargés uniquement à partir de sources fiables, et que les sources ne peuvent être contrôlés par les utilisateurs de l'application.*

BurpSuite

BurpSuite détecte la même erreur que ZAP. Il retourne en plus une information de type *Input returned in response (reflected).* Cependant il semble s'agir d'un faux positif, le problème existe mais ne semble pas exploitable. Si un future développeur remarque un problème de sécurité lié à cette issue merci de se référer aux détails suivant :

Issue detail

The value of the URL path filename is copied into the application's response.

Issue background

Reflection of input arises when data is copied from a request and echoed into the application's immediate response.

Input being returned in application responses is not a vulnerability in its own right. However, it is a prerequisite for many client-side vulnerabilities, including cross-site scripting, open redirection, content spoofing, and response header injection. Additionally, some server-side vulnerabilities such as SQL injection are often easier to identify and exploit when input is returned in responses. In applications where input retrieval is rare and the environment is resistant to automated testing (for example, due to a web application firewall), it might be worth subjecting instances of it to focused manual testing.

Vulnerability classifications

CWE-20: Improper Input Validation
CWE-116: Improper Encoding or Escaping of Output

Request/Response (filtered)

Request : GET /secure/interface.htmlajijp5w4f7 HTTP/1.1
Response : <pre>Cannot GET /secure/interface.htmlajijp5w4f7</pre>

2.2.8 Quelques avertissements

Utilisation du serveur

N'utilisez pas ***node debug [...]***, utilisez plutôt ***node inspect [...]***. Cette recommandation fait suite à l'exploit suivant : NodeJS Debugger - Command Injection (Metasploit)

Code HTML côté client

Pour des raisons de facilité de maintenance du serveur les pages web appellent des scripts hors domaine. Les domaines utilisés sont sécurisés mais ne dépendent pas du domaine de l'application. Ce comportement peut-être modifié en téléchargeant directement les scripts correspondant dans l'arborescence du serveur. Ainsi le client ne dépendra que du domaine de l'application.

Liens utiles

Voici une liste non-exhaustive de sites web ou ressources utiles :
Site web de l'agence nationale de la sécurité des systèmes d'information
Site web de OWASP Zed Attack Proxy Project
Site web de portswigger web security
Site web d'exploit database

2.3 Intégration de scripts côté serveur

Deux types de scripts peuvent être utilisés : JavaScript et Python.

2.3.1 Intégration JavaScript

Après avoir crée votre script il faut le charger de la manière suivante : *require(/path/to/JavaScript.js)*. Il vaut ne pas polluer le Global Scope. Utilisez *module.exports.<FONCTION>*.

Ensuite voici le déroulement :

1/ Importer le script.

```
var ScriptPath = require(/path/to/<script>.js);
```

2/ Préparer le bloc de requête de la manière suivante.

La requête est post ou get. Si POST, les variables d'entrée se trouve dans req.body ; si GET les variables d'entrée se trouve dans res.params ou req.query cf Express API

Voici un exemple avec POST.

```
router.post(tresholdPath,function(req,res){ // Route : when POST treshold (body contains img ref) make some action and send 200 OK;
    if(req.session.isAuthenticated === "Success"){ // If the user is login;
    \\ [...] INSTRUCTIONS

      var img = req.body.img; // The body contain the image reference.

     \\ lancer des fonctions, issuent du script importer, sur les donnees de la requete
    \\ gerer les erreurs et renvoie au serveur
  });
```

2.3.2 Intégration Python

L'intégration des scripts python est indirect. Le module python-shell est déjà installé et intégré au serveur.

Voici les étapes pour intégrer le script :

1/ Créer la variable de path du script.

```
var ScriptPath = __dirname + '/Scripts/<script>.py';
```

2/ Préparer le bloc de requête de la manière suivante.

La requête est post ou get. Si POST, les variables d'entrée se trouvent dans req.body ; si GET les variables d'entrée se trouvent dans res.params ou req.query cf Express API

Voici un exemple avec POST.

```
router.post(tresholdPath,function(req,res){ // Route : when POST treshold (body contains img ref) make some action and send 200 OK;
    if(req.session.isAuthenticated === "Success"){ // If the user is login;
    \\ [...] INSTRUCTIONS
```

Récupération des éléments de la requête et paramétrage des options (voir Github python-shell)

```
      var img = req.body.img; // The body contain the image reference.

      var options = {
        mode: 'text',
        pythonPath: <path to python>,
        args: [`-i ${imgToScript}`, `-o ${datasPath}`]
      };
```

Démarrer le script : en cas d'erreur retourner l'erreur, sinon traiter les résultats et renvoyer au client 200 OK

```
        PythonShell.run(<your script path>, options, function (err, results)
    {
            if (err) throw err;
            // [...] INSTRUCTIONS
            res.sendStatus(200);
        });
     }
     else {
            console.log('WARNING [TRESHOLD] : Access Denied ('+req.session.user
    +')');
            res.redirect(mainPath);
     }
  });
```

Voila pour le protocole de base.

2.4 Ajout de fonctionnalités dans les composants d'Area côté clients

Le 'canvas' possède des composants (les images). Il existe deux manière d'altérer ces composants. La première sont les fonctions intégré directement dans les composants : contains, remove, disass. La seconde sont les fonctions présente dans Crtl.js et notamment dans le contrôleur 'ToolsCommand'.

2.4.1 Ajout de fonctionnalités dans ToolsCommand

L'altération d'un composant à travers ToolsCommand passe par Area.selection qui contient le composant séléctionné. Vous pouvez suivre l'exemple suivant :

```
$scope.RotateLeft = function(){
  /*
    * name: RotateLeft;
    * @param : nothing;
    * @return : nothing:
    * This function reduce the Area.selection.angle property used to rotate
      the selected image (counter clock-wise rotation).
    */
      if (Area.selection != null) {
        Area.selection.angle -= 5 * Math.PI / 180;
      }
};
```

2.4.2 Ajout de fonctionnalité dans un composant

L'ajout de fonctionnalité directement dans un composant passe par des prototype de fonctions.

Voici un exemple :

```
component.prototype.disass = function () {

  var index = Area.images.indexOf(this); // Search the object in Area.images;
  if (index > -1) { // If the object is found...
    var refImg = Area.images[index].ref;
    if (refImg.substr(0,8) === "Compound") {
      var tmpImg = Area.images[index];
      $.get('Datas/'+refImg+'.json', function(data, status){
        $.get('/secure/ref', function(tableJSON, status){

          var table = JSON.parse(tableJSON);
          Area.images = [];
          var l=data.length;
          var l2 = table.length;

          for (var i=0;i<l;i++) {
            for (var j=0;j<l2;j++){
              if (table[j].Ref === data[i].ref){
                Area.images.push(new component(table[j].RCL,table[j].Ref));
                Area.images[i].x = data[i].x;
                Area.images[i].y = data[i].y;
                Area.images[i].angle = data[i].angle;
                Area.scale = data[i].scaleFactorW;
                Area.images[i].image.width = Area.images[i].image.width*Area.scale;
                Area.images[i].image.height = Area.images[i].image.height*Area.scale;
              }
            }
          }

        });
      });
      $.get("/csrf", function(data, status, xhr){
        $.post("/secure/DestroyCMP",{compound: refImg, csrf: xhr.getResponseHeader('X-CSRF-Token')},
          function(data, status){
            console.log("Compound destroyed !");
          });
      });
    }
  }
}
```

2.5 Liens vers des références

Express API
NodeJS API
NPM pour les dépendances
OWASP index, voir Reference à gauche
Common Weakness Enumeration