Network and Computer Security Report
Group 26: Secure DokuWiki

64712 Alexandre Almeida, 64764 Gonçalo Avelar, and 56960 João Loff

Instituto Superior Técnico - Campus Alameda

1 Introduction

DokuWiki is a simple to use and highly versatile open source wiki software that does not require a database. Built in access controls and authentication connectors make DokuWiki especially useful in the enterprise context and the large number of plugins contributed by its vibrant community allow for a broad range of use cases beyond a traditional wiki. Wikis are quick to update and new pages are easily added. Designed for collaboration while maintaining a history of every change. Therefore, it is essential to provide high level security.

In this report we describe our project, including our solution. In the end of the document the reader is presented with the appendix A, B, and C, where we present our architecture, security guidelines of the DokuWiki, and some examples of the libraries that we used.

2 Objectives

Our project goal was to extend DokuWiki in a way that we could provide higher levels of security. We started by assuring that the saved wiki pages were properly ciphered to ensure confidentiality in a way that an attacker with access to the files would not see its contents. We also assure the pages backups, sending every page to the Amazon AWS Cloud Storage. The next step were to force the users to digitally sign their pages updates, assuring integrity and authenticity. We have ended up developing a Certificate Authority prototype, which simply receives certificate signing requests and properly generates trustworthy certificates, and also provides a certificate repository to the users. Although, our CA lacks of certificates revocation.

3 DokuWiki Security

3.1 Login System

Registered users can login using the "Login"[1] functionality. After supplying the correct credentials in the login form the credentials are verified against a hash of the credentials which were generated in the registration. The wiki either auto-generates a password by mail (default) or the user is allowed to supply his own password.

[1] `https://www.dokuwiki.org/login`

3.2 Access Control List

In a default wiki everyone is allowed to create, edit and delete pages. However sometimes it makes sense to restrict access to certain or all pages. This is when Access Control List[2] (ACL) comes into play. DokuWiki access restrictions can be bound to pages and namespaces.

4 Solution Architecture

DokuWiki is implemented using HTML, PHP, and Javascript languages and was developed in a way that allow users or developers to extend its functionality. It is easily modifiable due to the existence of an event system and a plugin management. The event system allows custom handling in addition to or instead of the standard processing for any part of DokuWiki which signals its activity via the event system. The custom handlers, or hooks, can be included in any plugin facilitating the process of extend the functionality. It is also possible for custom DokuWiki content to create and signal events of their own. Also, it is possible to develop our own event.

In this section we describe what we have implemented in this project and how have we done it. In the appendix A the reader is presented with a brief architectural picture of what is described in this section.

4.1 Content Encryption Plugin

In order to make our desired extensions we have developed a plugin for content encryption, which properly cipher the wiki pages and make backups of them using Amazon AWS Cloud Storage. To do this, we took advantage of the event system to intercept the update and retrieval of every single page. Thus, our plugin behavior is based in the following:

1. First we subscribe the events that we are interested in, which are page_write[3] and page_read[4].
2. Every time the event page_write was triggered, the plugin manager gave the opportunity to execute all the plugins that had subscribed that event. Therefore, our plugin is executed every time this event is raised, intercepting the page before it were wrote to disk. Then, we cipher the page content with a symmetric secret key which belongs only to the DokuWiki entity. This way, the content of the page that would be written to disk is ciphered, assuring confidentiality. After this we send the ciphered page to the Amazon AWS Cloud Storage[5].

[2] `https://www.dokuwiki.org/acl`
[3] `https://www.dokuwiki.org/devel:event:io_wikipage_write`
[4] `https://www.dokuwiki.org/devel:event:io_wikipage_read`
[5] `http://aws.amazon.com/s3/`

3. Every time the event page_read raises, the same thing happens: our plugin executes. In this case, we do the exact opposite: we intercept the page after it were read. After this, we decipher the page content with the same secret key, guaranteeing that the proper content is presented to the authorized users.

The cipher and decipher operations were performed using Advanced Encryption Standard (AES) in Cipher-block chaining (CBC) mode, using blocks of 128bits. To do this we used the Mcrypt[6] php library.

Regarding the symmetric secret key we have developed a function to generate a 32 character key, which is called when the DokuWiki is installed. This key is assumed to be stored in a secret secure location. We simulated this by storing the key in a file inside a folder named securelocation. Of course, in a real world system this would not be done. As in real world, the key would be kept with some person, in a vault, or in any other secure location.

This key is also regenerated time to time to avoid cryptanalysis. The approach that we used was to regenerate the key after a given time period. So, every time the key is used to decrypt something, we check the time period. If this time period is exceeded the key is regenerated. This raised a problem, which were: the already ciphered pages with the previous key. If we were talking about message transmission this would not be such a problem because we could always resend the messages, however, in this case, we need to provide the pages content to the users. So, when the key is regenerated we recursively iterate through the pages directory using the old key to decipher the pages and afterwards use the new key to cipher. Thereby, providing the pages content to the users.

4.2 Forcing users to digitally sign wiki pages

The other extension that we have implemented was forcing the user pages updates to be digitally signed, to assure another level of authenticity, integrity of the page content, and non-repudiation. In order to perform this extension, we have employed asymmetric cryptography using RSA algorithm, hashing with SHA1, and digital certificates provided by our CA, which we will briefly discuss in the next subsection. The signatures were according to the Public-Key Cryptography Standards (PKCS#1).

Conceptually, what happens is: the user edits the page, generates the hash of the page content, and signs it with his private key. Then, the page update is send along with the signature, and a tag that represents the origin, destination and time, avoiding attacks such as Surreptitious Forwarding and Message Stealing. When the DokuWiki receives the pages update, requests the user certificate (based on his username), verifies if the received certificate is valid, and if so, uses the correspondent public key to verify the signature received. If the signature is valid the request to update the page is accept and the page is saved. If the user certificate or the signature is invalid the request is rejected.

In order to perform this we had to do the following: a) when the user hits the save button, it is forced to provide his private key, and consequently sign the page

[6] `http://php.net/mcrypt`

content. This signature is generated using the client side language Javascript. To do this, we used forge[7] library, that is a native implementation of TLS and have tools to write crypto-based and network-heavy webapps. b) when the save request arrives in DokuWiki, the signature is verified along with the tag in the request. To verify the signature we have used the phpseclib[8] library which is a pure implementation in PHP, designed to be fully interoperable with OpenSSL[9] and other standardized cryptography programs and protocols.

The generation of the RSA key-pair was performed also using phpseclib on the user side. The key-pair is stored in the same fashion as we store the symmetric key for DokuWiki. We assume to have a secure location where the private key is confined.

4.3 Certificate Authority

We have implement a brief Certificate Authority which generates certificates to the users, and stores them in its repository. To perform this we used again phpseclib, using certificates according to X.509 standard. The certificate generation is based in the following:

1. The CA generates a self-signed certificate and stores it.
2. The user requests a the CA's certificate.
3. The CA sends the certificate to the user.
4. The user then generates a certificate signing request (CSR) with his public key, and sends it to the CA.
5. The CA receives the CSR and generates a X.509 certificate for that public key, signing it with its private key.
6. The certificate is then stored in the repository and is also sent to the user.

Of course, as in other public key infrastructure (PKI), it is assumed that the user trusts the CA to generate certificates. This is an Flat Trust Relation with a single CA on top, having a single point of management.

5 Conclusion

In this project we have been faced with some real world situations, which go along with the theoretical classes. We have to take into account a lot of details and be 100% aware of the problems that we have or might. In this report we have described what are the main points in our implementation, and some problems that we have faced. As the context of the course, we have focused on the security part rather than on the functionality.

After the project, we are more sensible to the problems that this area brings: problems related to the compatibility of the language libraries, even when using the same algorithm and standards; encoding of ciphered and signed text when sending messages to another endpoint, and others.

[7] `https://github.com/digitalbazaar/forge`
[8] `http://phpseclib.sourceforge.net`
[9] `http://www.openssl.org`

Appendix

A Architecture

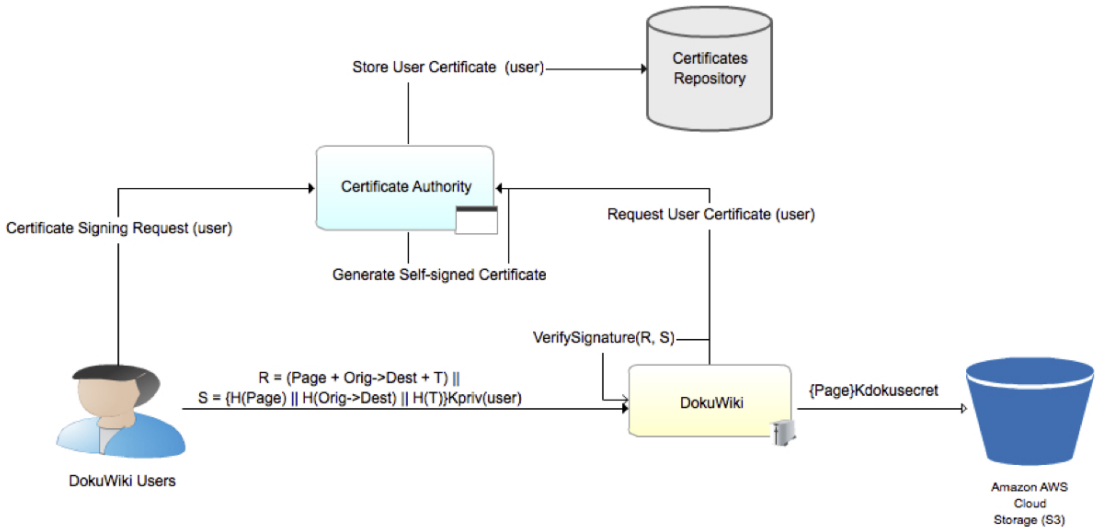

Fig. 1. DokuWiki's Extension Solution Architecture.

B Security Guidelines for Plugin Authors

To make sure that the developed plugin does not compromise the security of the whole wiki it is installed on, developers should follow the guidelines outlined on this appendix.

B.1 Cross Site Scripting (XSS)

This is probably the most common vulnerability to be found in DokuWiki plugins. Cross Site Scripting refers to an attack where malicious JavaScript code is introduced into a website. This can be used to redirect innocent users to malicious websites or to steal authentication cookies. DokuWiki's plugin mechanism gives plugin developers a great deal of flexibility.

Escaping output At an absolute minimum the plugin should ensure any raw data output has all HTML special characters converted to HTML entities using the htmlspecialchars function. URLs values should be escaped using rawurlencode. Also any wiki data extracted and used internally (eg. user names) should be treated with suspicion.

Input Checking Check always all the input. Use filters, conversions to the exact data type and ensure that it have only allowed data.

B.2 Cross Site Request Forgery (CSRF)

This vulnerability often appears into plugins due to the lack of understanding of this issue, often confused with the XSS. Cross Site Request Forgery refers to an attack where the victim's browser is tricked by a malicious site to ask for a page on a vulnerable site to do an unwanted action. The attack assumes the victim's browser have credentials to change something on the vulnerable site.

Adding Security Token DokuWiki offers functions to help dealing against CSRF attacks: getSecurityToken() and checkSecurityToken().

B.3 Remote Code Inclusion

This attack allows an attacker to inject (PHP) code into the application. This may occur on including files, or using unsafe operations functions like eval or system. Said so, always filter any input that will be used to load files or that is passed as an argument to external commands.

B.4 Information leaks

This attack may lead to the exposure of files that should usually be protected by DokuWiki's ACL or it might expose files on the server (like /etc/passwd). Said so, always filter any input that will be used to load files or that is passed as an argument to external commands; always use DokuWiki's ACL check functions when accessing page data.

C Libraries Examples

C.1 Phpseclib

Listing 1.1. Generation of RSA key pair.

```
<?php
include('Crypt/RSA.php');
```

```
$rsa = new Crypt_RSA();
$rsa->setPrivateKeyFormat(CRYPT_RSA_PRIVATE_FORMAT_PKCS1);
$rsa->setPublicKeyFormat(CRYPT_RSA_PUBLIC_FORMAT_PKCS1);
extract($rsa->createKey());
?>
```

Listing 1.2. Signing a text with private key and signature validation with public key.

```
<?php
include('Crypt/RSA.php');
$rsa = new Crypt_RSA();
$rsa->loadKey('...'); // private key
$plaintext = '...';
$rsa->setSignatureMode(CRYPT_RSA_SIGNATURE_PKCS1);
$signature = $rsa->sign($plaintext);
$rsa->loadKey('...'); // public key
echo $rsa->verify($plaintext, $signature) ? 'verified' : 'unverified';
?>
```

Listing 1.3. Certificate signing request generation.

```
<?php
include('File/X509.php');
include('Crypt/RSA.php');
$privKey = new Crypt_RSA();
extract($privKey->createKey());
$privKey->loadKey($privatekey);
$x509 = new File_X509();
$x509->setPrivateKey($privKey);
$x509->setDNProp('id-at-organizationName', 'phpseclib demo cert');
$csr = $x509->signCSR();
echo $x509->saveCSR($csr);
?>
```

Listing 1.4. Validation of certificate signature.

```
<?php
include('File/X509.php');
$x509 = new File_X509();
$x509->loadCA('...'); // CA cert
$cert = $x509->loadX509('...'); // USER cert
echo $x509->validateSignature() ? 'valid' : 'invalid';
?>
```

C.2 Mcrypt

Listing 1.5. Encrypting and decrypting with mcrypt library.

```
<?php
    # --- ENCRYPTION ---
    $key = pack('H*',
        "bcb04b7e103a0cd8b54763051cef08bc55abe029fdebae5e1d417e2ffb2a00a3");
    $key_size = strlen($key);
    $plaintext = "This string was AES-256 / CBC / ZeroBytePadding
        encrypted.";
    # create a random IV to use with CBC encoding
    $iv_size = mcrypt_get_iv_size(MCRYPT_RIJNDAEL_128, MCRYPT_MODE_CBC);
    $iv = mcrypt_create_iv($iv_size, MCRYPT_RAND);
    $ciphertext = mcrypt_encrypt(MCRYPT_RIJNDAEL_128, $key,
                                 $plaintext, MCRYPT_MODE_CBC, $iv);
    $ciphertext = $iv . $ciphertext;
    # encode the resulting cipher text so it can be represented by a
        string
    $ciphertext_base64 = base64_encode($ciphertext);

    # --- DECRYPTION ---
    $ciphertext_dec = base64_decode($ciphertext_base64);
    # retrieves the IV, iv_size should be created using
        mcrypt_get_iv_size()
    $iv_dec = substr($ciphertext_dec, 0, $iv_size);
    # retrieves the cipher text (everything except the $iv_size in the
        front)
    $ciphertext_dec = substr($ciphertext_dec, $iv_size);
    $plaintext_dec = mcrypt_decrypt(MCRYPT_RIJNDAEL_128, $key,
                                    $ciphertext_dec, MCRYPT_MODE_CBC,
                                        $iv_dec);

?>
```

C.3 Forge

Listing 1.6. RSA Examples with forge library.

```
<script>
var rsa = forge.pki.rsa;
var pki = forge.pki;
// convert a PEM-formatted private key to a Forge private key
var privateKey = pki.privateKeyFromPem(pem);
// sign data with a private key and output DigestInfo DER-encoded bytes
var md = forge.md.sha1.create();
md.update('sign this', 'utf8');
var signature = privateKey.sign(md);
// verify data with a public key
var verified = publicKey.verify(md.digest().bytes(), signature);
</script>
```