Tapestry5PreventClientSideChanges

How to protect server side generated values against client side changes using a HMAC

(an attempt to fix https://issues.apache.org/jira/browse/TAPESTRY-2482)

Tapestry 5 stores some server side generated state at the client side. Example of this is t:formdata and t:state:client. Client side state however can be changed by the client (which is pretty obvious 😉. Some, including me, see this as a possible security vulnerability. Even though sensitive data (which should not accesible to the user) should never be stored client side, allowing changes to the client side state can still be problematic. One way to prevent client side changes is the addition of a secure checksum that is derived from the client side data. If the data is changed the checksum no longer matches and the server can detect that the state has been changed. The checksum need to be generated in such a way that the client is unable to generate a checksum him/her self. The standard choice for such a checksum is the "keyed-Hash Message Authentication Code" (HMAC) (See: http://en.wikipedia.org/wiki/HMAC).

The following part will briefly explain how you can add an automatic checksum to sensitive elements.

Application State Object (ASO)

The HMAC requires a secret for the calculation of the checksum. In principle you can use one key for all clients (client is defined as having a session). But, in order to make a 'reply attack' less likely (will come to that later) I have chosen to give each use his/her own secret key. It depends on you requirements whether this is acceptable (memory wise).

```
/**
 * Application state object which will hold the randomly generated key used for the
 * calculation of a HMAC checksum.
 *
 * Note: This won't protect you against a 'reply attack'.
 *
 * @author Martijn Brinkers
 *
 */
public class HMAC
{
        private final String algorithm;
        private final SecretKey key;

        public HMAC(@Inject @Value("${hmac.algorithm}") String algorithm)
        throws NoSuchAlgorithmException
        {
                Check.notNull(algorithm, "algorithm");

                this.algorithm = algorithm;

                KeyGenerator kg = KeyGenerator.getInstance(algorithm);

                key = kg.generateKey();
        }

        public SecretKey getKey() {
                return key;
        }

        public String calculateHMAC(String input)
        throws NoSuchAlgorithmException, InvalidKeyException
        {
                Mac mac = Mac.getInstance(algorithm);

                mac.init(key);

                byte[] hmac = mac.doFinal(MiscStringUtils.getAsciiBytes(input));

                return toMaxRadix(hmac);
            }

        public static String toMaxRadix(byte bytes[])
        {
                BigInteger bigInt = new BigInteger(1, bytes);
                return bigInt.toString(36);
        }
}
```

Adding the HMAC hidden field and checking the HMAC

The HMAC need to be calculated after the page has been rendered. The calculated HMAC will need to be added as a hidden field to the page. Adding the hidden field requires a MarkupRendererFilter and checking the HMAC requires a ComponentEventRequestFilter. The following interface and implementation shows you how HMAC is 'injected' and how the HMAC is checked when a request comes in. The HMACFilterImpl requires an array of Strings with the element names that need to be protected with a HMAC. When an element that requires protection is found in the generated page the HMAC of the elements value is calculated and added as a hidden field. When the request comes in, the MHAC of the value is again calculated and compared to a list of HMACs from the request.

```
public interface HMACFilter extends ComponentEventRequestFilter, MarkupRendererFilter {
        /*
         * For now we do not need any extra methods.
         */
}
```

```
/**
 * Filter that calculates the HMAC of a elements value and add the HMAC to the generated page
 * as a hidden element. When a page is activated (for example by a post) the HMAC from the request
 * is checked against a (newly) calculated HMAC. If the HMACs differ it means that the value
 * has been changed and the user is redirected to another page that to report this.
 *
 * @author Martijn Brinkers
 *
 */
public class HMACFilterImpl implements HMACFilter
{
        private final static Logger logger = LoggerFactory.getLogger(HMACFilterImpl.class);

        /*
         * The set of all element names that need to be protected by a checksum
         */
        private Set<String> protectedElements = new HashSet<String>();

        public static String HMAC_PARAMETER = "hmac-checksum";

        private final ApplicationStateManager asm;
        private final Request request;
        private final Response response;
        private final LinkFactory linkFactory;
        private final RequestPageCache requestPageCache;

        /*
         * The page to redirect to when secureID is incorrect
         */
        private final String redirectTo;

        /*
         * Exception thrown when HMAC is incorrect
         */
        private static class IncorrectHMACException extends Exception
        {
                private static final long serialVersionUID = -8133828090623176301L;

            public IncorrectHMACException(String message) {
                super(message);
            }
        }

        public HMACFilterImpl(ApplicationStateManager asm, Request request, Response response,
                        LinkFactory linkFactory, RequestPageCache requestPageCache, String redirectTo,
                        String... protectedElements)
        {
                Check.notNull(asm, "asm");
                Check.notNull(request, "request");
                Check.notNull(response, "response");
                Check.notNull(linkFactory, "linkFactory");
                Check.notNull(requestPageCache, "requestPageCache");
                Check.notNull(redirectTo, "redirectTo");

                this.asm = asm;
                this.request = request;
                this.response = response;
                this.linkFactory = linkFactory;
                this.requestPageCache = requestPageCache;
                this.redirectTo = redirectTo;

                for (String protectedElement : protectedElements)
                {
                        if (protectedElement == null) {
                                continue;
                        }

                        protectedElement = protectedElement.trim().toLowerCase();

                        this.protectedElements.add(protectedElement);
                }
```

```
    }

public void renderMarkup(MarkupWriter writer, MarkupRenderer renderer)
{
    renderer.renderMarkup(writer);

    Document document = writer.getDocument();

    if (document != null)
    {
            Element root = document.getRootElement();

            if (root != null)
            {
                    LinkedList<Element> queue = new LinkedList<Element>();

                    queue.add(root);

                    while (!queue.isEmpty())
                    {
                        Element element = queue.removeFirst();

                        if (element == null) {
                                continue;
                        }

                        String elementName = element.getAttribute("name");

                        if (elementName != null) {
                                elementName = elementName.trim().toLowerCase();
                        }

                        if (protectedElements.contains(elementName))
                        {
                                /*
                                 * It's a protected item so we should calculate the HMAC of the value
                                 */
                                String value = element.getAttribute("value");

                                String hmac;

                                        try {
                                                hmac = calculateHMAC(value);
                                        }
                                        catch (InvalidKeyException e) {
                                                throw new MimesecureRuntimeException(e);
                                        }
                                        catch (NoSuchAlgorithmException e) {
                                                throw new MimesecureRuntimeException(e);
                                        }

                                /*
                                 * Add the HMAC checksum as a hidden element
                                 */
                                        element.element("input",
                                           "type", "hidden",
                                           "name", HMAC_PARAMETER,
                                           "value", hmac);
                        }

                        for (Node n : element.getChildren())
                        {
                                Element child = null;

                                if (n instanceof Element) {
                                        child = (Element) n;
                                }

                            if (child != null) queue.addLast(child);
                        }
                    }
```

```
                }
        }
    }

        public void handle(ComponentEventRequestParameters parameters,
                        ComponentEventRequestHandler handler)
        throws IOException
        {
        Page page = requestPageCache.get(parameters.getActivePageName());

        try {
                if (isHMACProtected(page))
                {
                        /*
                         * We will build a set of all the HMACS we can find in the request
                         */
                        String[] hmacParameters = request.getParameters(HMAC_PARAMETER);

                        Set<String> hmacs = new HashSet<String>();

                        if (hmacParameters != null)
                        {
                                for (String hmac : hmacParameters) {
                                        hmacs.add(hmac);
                                }
                        }

                        for (String protectedElement : protectedElements)
                        {
                                /*
                                 * There can be more than one protected value per element
                                 */
                                String[] protectedValues = request.getParameters(protectedElement);

                                if (protectedValues != null)
                                {
                                        for (String protectedValue : protectedValues)
                                        {
                                                String hmac = calculateHMAC(protectedValue);

                                                if (!hmacs.contains(hmac)) {
                                                        throw new IncorrectHMACException("The hmac " + hmac + "
is incorrect");
                                                }
                                        }
                                }
                        }
                }

                /*
                 * Not protected or checksum is correct so continue
                 */
                handler.handle(parameters);
        }
        catch(IncorrectHMACException e)
        {
                logger.warn(e.getMessage());

                Link link = linkFactory.createPageLink(redirectTo, false);

                response.sendRedirect(link);
        }
        catch (InvalidKeyException e) {
                throw new IOException(e);
                }
        catch (NoSuchAlgorithmException e) {
                throw new IOException(e);
                }
        }

        private String calculateHMAC(String value)
```

```
        throws InvalidKeyException, NoSuchAlgorithmException
        {
                if (value == null) {
                        value = "";
                }

            return asm.get(HMAC.class).calculateHMAC(value);
        }

        private boolean isHMACProtected(Page page)
        {
                /*
                 * For now all actions are protected. We can always create a special Annotation when
                 * we need to specify which pages/actions are protected.
                 */
                return true;
        }
}
```

Application Module

The following items need to be added to your application module (or to a new module)

```
    public void contributeComponentEventRequestHandler(OrderedConfiguration<ComponentEventRequestFilter>
configuration,
                    HMACFilter hMACFilter)
    {
            configuration.add("HMACFilter", hMACFilter, "after:*");
    }

    public static HMACFilter buildHMACFilter(ApplicationStateManager asm,
                    Request request, Response response, LinkFactory linkFactory,
                    RequestPageCache requestPageCache, @Inject @Value("${hmac.redirectTo}") String redirectTo)
    {
            String[] protectedElements = {Form.FORM_DATA, "t:state:client"};

            HMACFilter filter = new HMACFilterImpl(asm, request, response, linkFactory, requestPageCache,
                            redirectTo, protectedElements);

            return filter;
    }

    public void contributeMarkupRenderer(OrderedConfiguration<MarkupRendererFilter> configuration,
                    HMACFilter hMACFilter)
    {
        configuration.add("HMACFilter", hMACFilter, "after:UploadException", "before:Ajax");
    }

    public static void contributeFactoryDefaults(
            MappedConfiguration<String, String> configuration)
    {
        configuration.add("hmac.algorithm", "HmacSHA1");
        configuration.add("hmac.redirectTo", "accessdenied");
    }
```

The default algorithm for the HMAC is HmacSHA1. The hmac.redirectTo parameter sets the page to which is redirected when the HMACs do not match.

Replay Attack

The HMAC checksum protects you against changes done by the client but not against a replay attack (see http://en.wikipedia.org/wiki/Replay_attack) because the checksum calculation is kind of static. If you want to prevent replay-attacks you can for example add a counter that is increased for each request to the HMAC calculation process. The example above uses a random key stored in the user session. After the user session has ended all of the HMACs previously generated become invalid. A reply attack is therefore only possible within one session (and thus won't protect you against the current user)

Martijn Brinkers (m.brinkers@pobox.com)