

s2-045 分析

1. 漏洞分析

以下分析基于版本 struct 2.3.20 版本

struct 使用java ee中的Filter去拦截请求，并实现自己的功能。也就是说，用户所发出的请求，首先会在

org.apache.struts2.dispatcher.ng.filter 中的

StrutsPrepareAndExecuteFilter类去执行。首先执行类中的doFilter方法。这个方法是自动调用的，在这里可以struct拦截用户的请求，并实现自己的业务代码。于是会执行 prepare.wrapRequest 这个调用。如下图

```
prepare.prepare(request, response);
prepare.createActionContext(request, response);
prepare.assignDispatcherToThread();
//// TODO: 08/03/2017 parse the header
request = prepare.wrapRequest(request);
ActionMapping mapping = prepare.findActionMapping(request, response, true);
if (mapping == null) {
    boolean handled = execute.executeStaticResourceRequest(request, response);
    if (!handled) {
```

我们跟入这个函数

```
public HttpServletRequest wrapRequest(HttpServletRequest oldRequest) throws ServletException {
    HttpServletRequest request = oldRequest;
    try {
        // Wrap request first, just in case it is multipart/form-data
        // parameters might not be accessible through before encoding (ww-1278)
        request = dispatcher.wrapRequest(request);
    } catch (IOException e) {
        throw new ServletException("Could not wrap servlet request with MultipartRequestWrapper!", e);
    }
}
```

会执行 dispatcher.wrapRequest 这个函数。于是继续跟入

```
String content_type = request.getContentType ();
if (content_type != null && content_type.contains ("multipart/form-data")) {
    MultipartRequest mpr = getMultipartRequest ();
    LocaleProvider provider = getContainer ().getInstance (LocaleProvider.class);
    request = new MultipartRequestWrapper (mpr, request, getSaveDir (), provider);
} else {
    //// TODO: 08/03/2017 get method is here
    request = new StrutsRequestWrapper (request, disableRequestAttributeValueStackLookup);
}
```

在这里，将会判断一下是否包含multipart/form-data这个头。注意，在

这里，并没去区分是什么http方法，也就是说，get方法强行包含一个带有恶意代码的content-type头，也会执行到这里的。

在这里，首先判断一下是否为null和是否包含multipart/form-data。如果不包含，也就是get方法和post表单提交，则执行else里面的内容。如果包含，也就是说可能有文件上传，则执行if里面的内容。

在if中，首先获取到默认的解析器

```
*/
protected MultiPartRequest getMultiPartRequest() {
    MultiPartRequest mpr = null;
    //check for alternate implementations of MultiPartRequest
    // // TODO: 08/03/2017 get default multiparse
    Set<String> multiNames = getContainer().getInstanceNames(MultiPartRequest.class);
    for (String multiName : multiNames) {
        if (multiName.equals(multipartHandlerName)) {
            mpr = getContainer().getInstance(MultiPartRequest.class, multiName);
        }
    }
    if (mpr == null ) {
        mpr = getContainer().getInstance(MultiPartRequest.class);
    }
    return mpr;
}
```

可以查看一下multipartHandlerName的默认值

```
public void setMultipartSaveDir(String val) { multipartSaveDir = val; }

@Inject(StrutsConstants.STRUTS_MULTIPART_PARSER)
public void setMultipartHandler(String val) {
    multipartHandlerName = val;
}

@Inject
```

也就是获取默认配置中的值，而在struts的默认配置(org.apache.struts2的default.properties)中，这个值恰好为jakarta，默认配置为

```
#struts.objectTypeDeterminer = notiger

### Parser to handle HTTP POST requests, encoded using the MIME-type multipart/form-data
# struts.multipart.parser=cos
# struts.multipart.parser=pell
# struts.multipart.parser=jakarta-stream
struts.multipart.parser=jakarta
# uses javax.servlet.context.tempdir by default
struts.multipart.saveDir=
struts.multipart.maxSize=2097152

### Load custom property files (does not override struts.properties!)
# struts.custom.property.application.org.apache.struts2.application.custom
```

回到dispatcher.wrapReques中，获取到默认的解析器后，将会把MultiPartRequestWrapper这个类实例化，执行MultiPartRequestWrapper的构造函数。跟入MultiPartRequestWrapper的构造函数

```
public MultiPartRequestWrapper(MultiPartRequest multiPartRequest, HttpServletRequest request) {
    super(request);
    errors = new ArrayList<String>();
    multi = multiPartRequest;
    defaultLocale = provider.getLocale();
    setLocale(request);
    try {
        multi.parse(request, saveDir);
        for (String error : multi.getErrors()) {
            addError(error);
        }
    } catch (IOException e) {
        if (LOG.isWarnEnabled()) {
            LOG.warn(e.getMessage(), e);
        }
        addError(buildErrorMessage(e, new Object[] {e.getMessage()}));
    }
}
```

在这里，将会执行multi.parse函数，因为这里的multipartrequest为JakartaMultiPartRequest类，于是将会执行JakartaMultiPartRequest.parse方法，跟入之

```
public void parse(HttpServletRequest request, String saveDir) throws IOException {
    try {
        setLocale(request);
        processUpload(request, saveDir);
    } catch (FileUploadBase.SizeLimitExceededException e) {
        if (LOG.isWarnEnabled()) {
            LOG.warn("Request exceeded size limit!", e);
        }
        String errorMessage = buildErrorMessage(e, new Object[] {e.getPermittedSize(), e.getActualSize()});
        if (!errors.contains(errorMessage)) {
            errors.add(errorMessage);
        }
    } catch (Exception e) {
        if (LOG.isWarnEnabled()) {
            LOG.warn("Unable to parse request", e);
        }
        String errorMessage = buildErrorMessage(e, new Object[] {});
        if (!errors.contains(errorMessage)) {
            errors.add(errorMessage);
        }
    }
}
```

这里是重点，在这里将会执行java的捕获异常，如果捕获到异常的花，将会执行buildErrorMessage方法，也就是产生漏洞的地方。继续跟入，将会执行processUpload函数

```

        LOG.debug("Preparing error message for key: [#0]", errorKey);
    }
    return LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, e.getMessage(), args);
}

protected void processUpload(HttpServletRequest request, String saveDir) throws FileUploadException, UnsupportedEncodingException {
    for (FileItem item : parseRequest(request, saveDir)) {
        if (LOG.isDebugEnabled()) {
            LOG.debug("Found item " + item.getFieldName());
        }
        if (item.isFormField()) {
            processNormalFormField(item, request.getCharacterEncoding());
        } else {
            processFileField(item);
        }
    }
}

```

在这里，将会调用parseRequest对请求进行解析，跟入

```

protected List<FileItem> parseRequest(HttpServletRequest servletRequest, String saveDir) throws FileUploadException,
DiskFileItemFactory fac = createDiskFileItemFactory(saveDir);
ServletFileUpload upload = createServletFileUpload(fac);
return upload.parseRequest(createRequestContext(servletRequest));
}

```

在这里，将会调用upload.parseRequest函数，跟入

```

/**
 * public List<FileItem> parseRequest(RequestContext ctx)
 *     throws FileUploadException {
 *     List<FileItem> items = new ArrayList<>();
 *     boolean successful = false;
 *     try {
 *         FileItemIterator iter = getItemIterator(ctx);
 *         FileItemFactory fac = getFileItemFactory();
 *         if (fac == null) {
 *             throw new NullPointerException("No FileItemFactory has been set.");
 *         }
 *         while (iter.hasNext()) {
 *             final FileItemStream item = iter.next();
 *             // Don't use getName() here to prevent an InvalidFileNameException.
 *             //todo throw invaild content-type
 *             final String fileName = ((FileItemIteratorImpl.FileItemStreamImpl) item).name;
 *             FileItem fileItem = fac.createItem(item.getFieldName(), item.getContentType(),
 *                 item.isFormField(), fileName);
 *             items.add(fileItem);
 *             try {
 *                 Streams.copy(item.openStream(), fileItem.getOutputStream(), true);
 *             } catch (FileUploadIOException e) {
 *                 throw (FileUploadException) e.getCause();
 *             }
 *         }
 *     } catch (FileUploadException e) {
 *         // ...
 *     }
 *     return items;
 * }

```

现在已经进入org.apache.commons.fileupload的FileUploadBase类，注意在这里，没有对FileItemStreamImpl进行捕获异常，于是跟入FileItemStreamImpl的构造函数中

```

FileItemIteratorImpl(RequestContext ctx)
    throws FileUploadException, IOException {
    if (ctx == null) {
        throw new NullPointerException("ctx parameter");
    }
    //TODO throw exception
    String contentType = ctx.getContentType();
    if ((null == contentType)
        || (!contentType.toLowerCase(Locale.ENGLISH).startsWith(MULTIPART))) {
        throw new InvalidContentTypeException(
            format("the request doesn't contain a %s or %s stream, content type header is %s",
                MULTIPART_FORM_DATA, MULTIPART_MIXED, contentType));
    }
}

```

在这里，将会得到content-type头，并且检查一下是否以multipart开头。所以，exp中并不会以multipart开头。于是在这里将会抛出异常，并将content-type信息放入异常中。

这里并没有任何代码捕获异常，于是向函数调用方继续抛出，到org.apache.commons.fileupload的FileUploadBase类的parseRequestMethod，但是parseRequestMethod方法中，恰好没有对这段代码捕获异常，于是继续向上抛出异常。在JakartaMultiPartRequest.parseRequestMethod方法中，也没有捕获异常，于是到JakartaMultiPartRequest.processUpload方法中。在这里，也没有捕获异常，于是返回到JakartaMultiPartRequest.parse函数中。这里对代码捕获异常，于是在这里，将会执行catch里面的语句，也就会在这里调用buildErrorMessage函数，参数为异常的信息，也就是包含恶意content-type头的那一段字符串。跟入这个函数

```

@protected String buildErrorMessage(Throwable e, Object[] args) {
    String errorKey = "struts.messages.upload.error." + e.getClass().getSimpleName();
    if (LOG.isDebugEnabled()) {
        LOG.debug("Preparing error message for key: [%0]", errorKey);
    }
    return LocalizedTextUtil.findText(this.getClass(), errorKey, defaultLocale, e.getMessage(), args);
}

```

在这里将会执行LocalizedTextUtil.findText。谷歌一下这个函数的信息

When looking for the message, if the key indexes a collection (e.g. user.phone[0]) and a message for that specific key cannot be found, the general form will also be looked up (i.e. user.phone[*]).

If a message is found, it will also be interpolated. Anything within \${...} will be treated as an OGNL expression and evaluated as such.

发现这里可能会执行ognl语句，跟入这个函数

```

// get default
GetDefaultMessageReturnArg result;
if (indexedTextName == null) {
    result = getDefaultMessage(aTextName, locale, valueStack, args, defaultMessage);
} else {
    result = getDefaultMessage(aTextName, locale, valueStack, args, null);
    if (result != null && result.message != null) {
        return result.message;
    }
    result = getDefaultMessage(indexedTextName, locale, valueStack, args, defaultMessage);
}

// could we find the text, if not log a warn
if (unableToFindTextForKey(result) && LOG.isDebugEnabled()) {
    String warn = "Unable to find text for key '" + aTextName + "' ";
    if (indexedTextName != null) {

```

将会执行getDefaultMessage，参数为包含payload的content-type头，跟入

```

static GetDefaultMessageReturnArg getDefaultMessage(String key, Locale locale, ValueStack valueStack, Object[] args,
String defaultMessage) {
    GetDefaultMessageReturnArg result = null;
    boolean found = true;

    if (key != null) {
        String message = findDefaultText(key, locale);

        if (message == null) {
            message = defaultMessage;
            found = false; // not found in bundles
        }

        // defaultMessage may be null
        if (message != null) {
            MessageFormat mf = buildMessageFormat(TextParseUtil.translateVariables(message, valueStack), locale);

            String msg = formatWithNullDetection(mf, args);
            result = new GetDefaultMessageReturnArg(msg, found);
        }
    }
}

```

在这里，将会执行TextParseUtil.translateVariables，谷歌一下这个函数

translateVariables

```
public static Object translateVariables(char open,
                                       String expression,
                                       ValueStack stack,
                                       Class asType,
                                       TextParseUtil.ParsedValueEvaluator evaluator,
                                       int maxLoopCount)
```

Converted object from variable translation.

Parameters:

open -

expression -

stack -

asType -

evaluator -

Returns:

Converted object from variable translation.

到这里，将会把包含payload的content-type头作为ognl语句去执行。再继续跟，就是表达式处理的过程了。于是，漏洞就产生了