1st security audit FileSender 2.0, FileSender project response

Security audit:

Auditor:	Daan Keuper, Pine Digital Security, the Netherlands
Audit start:	12 January 2015
Date report:	3 February 2015
Code version:	2.0 development, r3390.
Funding:	provided by HEAnet

Responsible from FileSender project: Jan Meijer

This document:

Author:	Jan Meijer, FileSender project lead
Date:	11 March 2015
Status:	final
Version:	1.0
Classification:	Public

Introduction

FileSender software is entrusted with user's files and hence needs to be secure. To ensure a proper level of security is achieved each major release of FileSender is subject to at least one code security audit.

The FileSender project hired Pine Digital Security to execute a code security audit of the FileSender 2.0 development code. The necessary funds were supplied by HEAnet. The audit was executed on revision 3390 of the SVN branche branches/filesender-2.0 and done in the timeframe 12 January 2015 – 3 February 2015. Pine sent the report with its findings on 3 February 2015. A number of issues were identified.

The report was discussed on 4 February in a meeting between Jan Meijer (FileSender project lead), Etienne Meleard (FileSender development lead) and in a conference call between the two aforementioned and Daan Keuper from Pine Digital Security.

Based on these discussions an assessment was made of each of the identified issues and the appropriate response from the project decided. This document describes both the assessment and the response.

General feedback by Pine Security

Daan Keuper also executed the previous code security audits on the 1.6 and the 2.0-prototype code. He's pleased with the progress we've made and mentioned the code has improved a lot and is much cleaner now with a clear separation in model/view/controller. *He found no structural security issues in the current 2.0 code base* which is an improvement from version 1.6.

2.6.13: Generated passwords and tokens must not be predictable

Issue: *vulnerability:* FileSender uses random numbers and identifiers in various places, for example in the download URL. The reviewd 2.0 codebase uses *uniq_id* and *mt_rand*. Both don't return a sufficiently secure random number which may result in an attacker guessing the state of the random generator. An attacker might thus be able to guess download URLs and CSRF tokens. Recommend to use secure random generator.

Assessment: Issue confirmed and needs to be addressed before the 2.0 release. Preferably without using openssl as it would add a dependency and be much slower (ballpark figure: way more than 10 times slower) than using the other random generator.

The unique identifiers used in FileSender need to have two characteristics:

- the ID must be unique
- the ID must be non-guessable. As long as you can't guess the random number in the ID you can't guess the random generator state.

For the FileUID we check uniqueness, if a collision occurs we generate a new one. To make the ID non-guessable Pine recommends a HMAC. The recommendation is to use a secure random number generator like openssl but using for example /dev/urandom or mtrand and hash that combined with a salt value to an HMAC is sufficiently secure. FileSender would need to auto-generate the salt value.

Pine comments that wile an attacker in this way won't be able to directly know the random number you have to be very careful you don't leak random number to user somewhere.

Response: Etienne will add HMAC with a random generated salt and look at making it configurable to read urandom or use mt_rand.

Follow-up: this was implemented on 4 February, details in ticket #1171

2.7.1: User input must be encoded in the context where it is used

Issue: *defence in depth:* when FileSender receives user input the function sanitizeInput is used to ensure the input will not contain anything that will be executed. This function currently filters the input for a number of dangerous characters and adds a space to them. Pine recommends to use escaping, the current code uses filtering.

Filtering changes the input and might lead to unexpected side-effects.

Assessment: Etienne's collegues in the Sympa mailinglist software project had a security audit executed where an issue was found using variable encoded string injection. The offending string had to be machine-generated. This string defeated perls regexp-based entity encoding mechanism. The php mechanism behaved the same. Sympa introduced a filtering fix which was reported in this FileSender audit.

The filtering fix was based on a blog-article published on OWASP as a recommendation. Unfortunately the blog article is now no longer accessible.

Daan indicated he's unaware of any technique to bypass html_entities. He tested by removing the line of code and injection of code was not possible. The strong recommendation is to not filter on input; you don't know where the data will be used.

Response: there is not yet consensus in the FileSender core team on the best approach. Based on the experience of the Sympa mailinglist software team escaping alone might not be enough. The issue found in the Sympa security audit might now have been fixed and the filtering fix no longer needed. We will await the results of the 2nd security audit and then decide what to do. We will address this before the 2.0 release.

Follow-up:
The issue is currently open and documented in ticket #1172

2.7.3: User input must be encoded before it is used in path/filenames

Issue: *defence in depth:* filenames were not escaped when used in a certain piece of code. This was not an actual vulnerability as in the offending code it wasn't user supplied input but the overall security of the software would improve by ensuring escaped filenames, just in case the piece of code might be called in the future *with* user-supplied input.

As a general recommendation: don't rely on input, and when handling input don't rely on the difference between user input and application input as roles might changes in future code.

Assessment: the recommendation is sound. In this particular case the offending code no longer exists. We were building a file path using data that was given to a function. At the time of the audit this was not a flow, a user could not get data to this point. File names provided by users as part of uploads are treated as other input: sanitised before storage or use. File names in general are not used on the server's file system. They are only stored in the database and given to a downloader to ensure the correct filename is used when downloading a file.

Response: offending code to be removed.

Follow-up: after the code was reported it was discovered the code was not used and therefor removed in changeset 3447. This issue is documented in ticket #1173. A related possible issue concerns using template code in a file name. This is not part of the security audit results and is being investigated and documented in ticket #1181.

2.7.4 User input must be encoded before it is used in SQL commands

Issue: *defence in depth, no vulnerability*. Nearly all SQL commands are done using prepared statements but Daan found a number of places where variables were used in SQL commands. The end user currently has no control over the contents of those variables but this might happen in future services. As a general rule we should use prepared statements, don't trust input. If we don't use prepared statements use some kind of escaping. Cast it to an int or add escaping before printing it to a variable.

Assessment: we use prepared statements where we can but prepared statements don't support "in" statements.

Response: Etienne will investigate whether he can construct a function that prepares a prepared statement.

Follow-up: The response has been implemented in changeset [3481] and is documented in ticket #1174

2.7.7: User input must be encoded before it is used in HTML output

Issue: *vulnerability without accessible attack surface:* there is one single case of javascript injection where not all outputs are escaped; the code allows for specification of a REST API callback handler for iframe integration. Currently the user doesn't have any control over all inputs but *might* have this in future versions.

Assessment: this is considered part of a feature which enables an application integrator to give return data to a handler of your choice idea is you give return data to handler of your choice. Being able to point to an existing function or object method from a parent window is to be expected.

Response: during the discussion of the issue Etienne suggested to limit what you can give as callback. Letters, digits, dash and underscores. This addresses the issue. Daan confirms.

Follow-up: this was implemented in changeset [3453] and documented in ticket #1175

2.11.8: Anti-clickjacking measures must be implemented

Issue: *vulnerability*: there is no protection against click-jacking nor do we mention clickjacking as a risk anywhere in the installation documentation. Recommendation is to set the x-frame-header in each request or same-origin, allow-from in the webserver config and/or document in the manual what to do if iframes are allowed. Either we document the risks with using iframes or ensure they can't be used.

Assessment: this is both a risk and a feature. Integrating FileSender using an iframe is a feature. Should you want to limit inclusion of FileSender in iframes, preliminary investigation suggests W3C's candidate recommendation Content Security Policy (CSP, http://www.w3.org/TR/CSP2/) is the standards-based way to do this. According to http://caniuse.com/#feat=contentsecuritypolicy this is supported on most currrent browsers. IE11 is supported by adding X-Content-Security-Policy. Daan suggests to set the same-origin header in the Apache config.

Response: as this is both a feature and a vulnerability we will not enforce this in the FileSender application. We will use CSP to allow for fine-grained protection. Rather than implement a swath of config options in FileSender we will document how to configure CSP in Apache for use with FileSender to prevent inclusion in iframes.

Follow-up: this is part of the documentation effort for 2.0. It has not yet been completed. Ticket #1176

2.14.2: PHP object injection

Issue: *defence in depth:* when an exception occurs a user is redirected to a page with a serialised array containing information about the exception. This array is unserialised and used to generate an exception object. User input can be used when serialising and unserialising arrays. A user can also give the application an object in this way; the application will happily unserialise object upon which the PHP interpreter will automatically call several functions on that object, e.g. wakeup, destruct, etc. This has an unknown impact on application. At the moment this can not be exploited but it might be exploitable in future FileSender versions. For defense-in-depth this should be addressed.

Assessment: issue confirmed. This can not be exploited now as there is no code like wakeup or destruct that does anything when called but this might become a problem in the future. This will be addressed by using JSON. Daan agrees this is a good solution.

Response: json serialisation will be used to avoid potential code injection when using native serialise/unserialise.

Follow-up: implemented on 4 February in changeset [3454] and documented in ticket #1177

2.14.3: Template injection

Issue: *exploitable vulnerablity*: possible to use template tags on user input which would be displayed in user output. Could enter tag for database password on input and user would then get the password nicely displayed in output.

Assessment: oops!

Response: will fix by replacing template tags *before* inserting any user data.

Follow-up: this was fixed in changeset [3457] and documented in ticket #1178.

2.14.4: Usage of variable variables

Issue: *defence in depth:* the "variable variables" technique is used in the template parser where variables are dynamically created. This may lead to a situation where a user can overwrite any variable in the current symbol table like a path variable and include a php script controlled by the user like a php script included in a URL. At the moment this is not exploitable as the user doesn't control the keys that are provided with the vars parameter. In future versions this might change.

Pine recommends to switch to a different mechanism like a key/value store.

Assessment: This is currently not a problem as there is no flow that allows a user to call the template method. Implementing a key/value store approach makes writing templates heavier. For the time being we will keep this as it is and gather practical experience with the template mechanism.

Details: In the template class there's a method that when called renders a template. This method looks for a template, resolves the template path (is it in the config directory because it's a customised template or is it in the default template directory?). Then it includes that template file if found while creating on the fly a set of varilables that the template will use. The method is given an associated array of key-value pairs (foo=bar) and creates variables for use in the template with name 'foo' and value 'bar'. . It could become

issue confirmed and needs to be addressed. Using a key/value store approach makes writing templates heavier. Hesitating between rewrite using some kind of keystore or filtering given variables before creating variables so they don't contain unexpected stuff like globals etc., path variable. The latter approach is error-prone, every time we introduce variable we also need to update the filter.

When rendering for example a report at end of upload, the template process method with template name "report" can have a variable report=array of report objects. In template code this then allows for a construct like "foreach $report as report do ....".

This mechanism is to make for easier template writing. It is currently not directly exploitable and needs significant other bad programming practice to make this exploitable.

A possible future exploit: if at some point user input is fed directly to the template processing function, it could give a set of variables that could have post data.

Response: this is not exploitable and needs significant bad programming practice to make it exploitable. We will gain practical experience with the template mechanism to see if the balance ease of use vs. security in depth needs to be different in future versions.

Follow-up: none. Documented in ticket #1179

2.14.5: Usage of PHP weak comparison

Issue: *vulnerability, not directly exploitable:* in most parts of the cost weak comparison is used. PHP will try to guess the type of variable on both sides and when possible will make an integer comparison (php default) which in some situations could lead to bypassing certain filter or check. If comparing two strings where first string is very large number and second string is also very large number then php will think they are alike. Lets convert it to integer and make comparison. Recommendation: always use strict comparison.

Assessment: oops.

Response: will fix.

Follow-up: fixed in changeset [3455], documented in ticket #1180 All non-applicables have been double-checked.