# COMP SCI 7412 Secure Software Engineering Assignment 2

Tinson Lai
a1812422

# 1 Part 1

## 1.1 WWBN/AVideo

Repository: WWBN/AVideo
Commit: 7d90140d8a31f00085c257ca8b62af12d15ba26e
Stars: 1179
Contributors: 45
File: plugin/LiveChat/Objects/LiveChatObj.php
Language: PHP
CVE ID: CVE-2019-18662
CWE ID: 89 (SQL Injection)
Lines: 52-53
Fix: e5fd2c6cad309ea5a308f4395625786000fcf0da
Comments: As shown in the list, the vulnerability has already been patched by applying strict character filters on the input. The original code directly concatenate the string received from the client with an SQL query without any neutralisation techniques, thus malicious SQL injection payload can be sent to and executed on the SQL server. However, I don't think this code needs such a strict filter which only keeps ASCII alphanumeric characters. This will be a pain in circumstances where localisation and internationalisation need to be introduced (i.e., using Unicode characters outside the ASCII range). It should use the technique to escape or filter all special characters instead. There is no need to escape or filter those SQL keywords, as it will usually be a malformed SQL query if special characters like semi-colon were escaped or filtered in the SQL injection payload. By this mechanism, the worst circumstance is a fatal error might occur if the code contains improper error handlings but SQL injections will never happen.

## 1.2 haraka/Haraka

Repository: haraka/Haraka
Commit: 4ae5bab6612a1a56a5463b2b93901f99a4f239f4
Stars: 3681
Contributors: 45
File: plugins/attachment.js
Language: JavaScript
CVE ID: CVE-2016-1000282

CWE ID: 77 (Command Injection)
Lines: 105
Fix: 2998b8b0455b8cc2c640344328439e10e685aad9
Comments: This is a quite old commit merged into the repository in 2016, thus the patch and a demonstration of attack has already been released formally by the people initially identified this vulnerability. The original code append the value of a field received from the client to a string representing an OS command preparing for execution. Therefore, it is easy to exploit this system as arbitrary number of Linux commands can be written in one line separated by semi-colon, and there is no neutralisation will be performed before concatenating the string. The solution is using a technique similar to escaping the characters, though it doesn't perform any neutralisation in the updated code. But it changed the way to spawn the process, and everything received from the client will be explicitly passed to the process as a program argument surrounded by single quotes. In this way, every character will now be treated as plain character without any special meanings. This is, in fact, a builtin shell functionality available on all Unix-like systems. I can see that the author of the patch has also added sufficient error handlings to deal with illegal program arguments, and this is a perfect fix in my perspective as it's straightforward without reinventing code to escape the input.

### 1.3 bludit/bludit

Repository: bludit/bludit
Commit: 3ab8c4c0a630b86eedd79020c34d1758f0d2726d
Stars: 741
Contributors: 79
File: bl-kernel/ajax/upload-profile-picture.php
Language: PHP
CVE ID: CVE-2019-12548
CWE ID: 94 (Code Injection)
Lines: 19
Initial Fix: d0843a4070c7d7fa596a7eb2130be15383013487
Improved Fix: 86e0f69030440c82ae82e3b2f3bc11c1f54022f1
Comments: The bug in the original version can be triggered by uploading a malicious PHP code as the payload when changing the user profile avatar. This is due to lacking a way to filter out files based on the file extensions, even though it eventually converts all files to PNG the format. The fix is quite easy, as it now filters out files according to their suffix. However, I don't think this is enough as attacker might be able to change the suffix of a file from `.php` to `.png` or something similar before uploading the file to bypass the checking process. It should perform extra check to determine whether an uploaded file is indeed an image or not. Fortunately, the issue I mentioned has already been fixed in a future commit by further checking its actual MIME type. However, this is just an improvement based on the original fix, and the newer fix is not linked to the CVE ID anymore.

## 2 Part 2

For task 3 and 5 and part of 7, I didn't attempt to do these tasks manually as I don't want to trace those commit histories, so I wrote the script from the start of this part and there is no screenshot can be provided as the steps. However, the generated raw data is available in the file `output.yaml` in the

GitHub repository[1]. The link to the GitHub repository is also in the comment of the submission. The data summarised is in the Excel file submitted alongside with this report.
For task 3, the steps are incorrect, since `git blame` will not be able to show any modification as clearly indicated on git official document[2]. The correct step should be:

1. `git show` the revision and it will output the differences between the given commit and its parent commit.

2. Following the paper (*VCCFinder: Finding Potential Vulnerabilities in Open-Source Projects to Assist Code Audits*[3]), it can generate a list of the lines to be blamed.

3. `git blame` those lines with respects to the parent commits. This can correctly blame those lines which were deleted in the fix commit.

4. Summarise the result and select the commit with the highest number of blames.

For task 4, from the official git document[4]:

- `-w` will ignore all changed in whitespaces, thus adding indentations or blank lines will not be considered as an author of a line of code.

- `-M` will try to identify the deleted blocks of lines and retain their original blames, and this differs from the original behaviour.

- `-C` will try to include changes copied or moved from other files in the same commit in addition to `-M`. Repeating it twice will also consider the initial commits which creates those files. When it is given three times, it will search over all commits in the current repository.

In the script, I only use the `-M` option as chages in whitespaces are negligible, and fortunately, this option did not give a different result. Another important reason I use this option is that, unlike Python, spacing in Java (the language used in both two reposiories) is just for formatting and it doesn't have any special meanings to the syntax. For both `-M` and `-C` options, I don't think it's appropriate for identifying the VCC. The commit which did the moving or copying should, at least, be blamed to be a VCC if the moving or copying indeed causes the vulnerability.
An interesting side note is that due to the nature of Python, it is impossible to specify repeating terminal options by using the git command delegate in GitPython. When I did test those parameters with native console, I was surprised by the output of `git blame` as it output so many irrelevant commits which didn't even modify the specified files.
For task 7, it is easy to get the answer from the generated raw data. However, an interesting point is for both repositories, though I wrote that the author of the fixing commit and VCC are not the same (because the author emails are different), I suspect that they are essentially the same person with different commiting configurations. I can't further confirm this as they didn't sign the commit, though GPG signature is just another guideline anyway.
By tracing the commit on GitHub, it is obvious that VCC and the commit are not in the same day for both repositories.

[1] `https://github.com/laitingsheng/Automated-VCC-Finding-Script`
[2] `https://git-scm.com/docs/git-blame`
[3] `https://dl.acm.org/doi/10.1145/2810103.2813604`
[4] `https://git-scm.com/docs/git-blame`

The first repository fix the issue as someone submit and create a pull request to the project's own Gerrit instance, and the project team has quickly responded to the merge request (in the same day). This issue was not identified before as this problem relates to the default behaviour of JavaX XML parser, and there isn't any reports with respect to the same problems before. Interestingly, I can't find any demonstration of attack, which means this is just a potential vulnerability idnetified by the developers. Therefore, I think it's reasonable that the fixing commit comes more than two years later than the VCC.
The second repository has a quite similar circumstance. The project team indeed quickly respond to the vulnerability report. There is no previous reports on the vulnerability prior to this formal vulnerability report.
I've verified the VCCs manually to ensure that they are correctly identified by the heuristic.