Software Security

INPUT problems

Erik Poll

Digital Security group

Radboud University Nijmegen

Master in Cyber Security

Problems with INPUT

Insecure input handling is *the* most common security problem

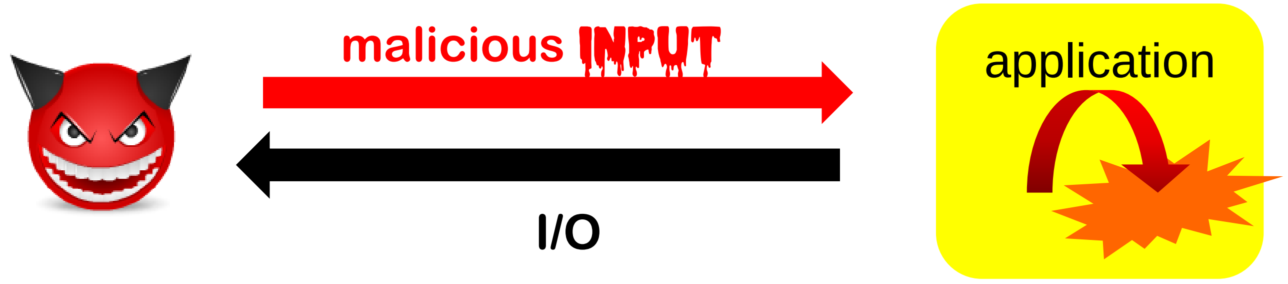
- aka lack of input validation, but that terminology is misleading (as we'll see later)

All input is dangerous & evil

- *All* input should be treated as highly poisonous & contagious

- Beware of the terms untrusted input or untrusted user input : by default, any input & all users should be untrusted

The I/O attacker model (‘hacking’)



- Aka end point attacker, as opposed to MitM attacker
- *Attacker goals?*
  - DoS, information leakage, remote code execution (RCE), or anything in between
  - ie. compromising integrity & availability of the application’s behaviour in *any* way
- *Input flaws we already saw?*

  buffer overflows, integer overflows & format string attacks.

  TOCTOU is also an input problem, but an odd one out.

Dangers of INPUT

Faced with an I/O attacker

Garbage In, Garbage Out

becomes

Malicious Garbage In, *Security Incident* Out

or

Malicious Garbage In, *Evil* Out

Input is dangerous:

- *Any line of code* that handles user input is at risk
- *Any resources (CPU cycles, memory, …) used* in processing are a risk

So ideally, these are kept to a minimum

Abusing bugs or features?

Two types of input security flaws:

1. Some input attacks exploit *bugs*
   - Bugs in code can provide *weird behaviour* that is *accidentally* introduced in the code by programmer; Attackers try to trigger & exploit such weird behaviour
   - Classic example: buffer overflows
2. Other input attacks abuse *features*
   - Some flaws *accidently expose* functionality that was *deliberately* introduced in the code, but which was not meant to be accessible by attackers.
   - Classic example: command or SQL injection

The line between 1 & 2 can be blurry, and a matter of opinion

Root causes of input problems

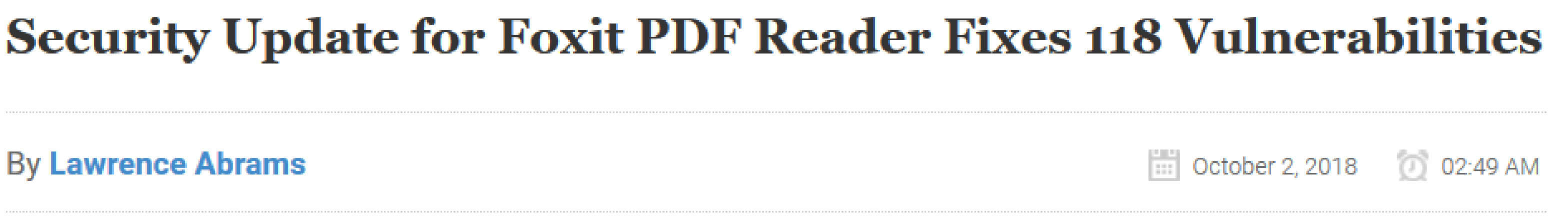
The input formats and languages involved play a central role:

1. Complexity of input formats & languages
   - making bugs in input processing likely
2. Sloppily & unclear specifications of input formats
   - making bugs even more likely
3. Expressivity of input languages
   - giving lots of power to the attackers (for flaws exploiting features)
   - worst thing to do: including a programming language in your input format
4. (Too?) *many* input formats & languages
   - often combined, stacked or nested,
     aggregating all the problems above

Exploiting bugs

(caused by complexity)

Security update of the week



Security Update for Foxit PDF Reader Fixes 118 Vulnerabilities

By Lawrence Abrams October 2, 2018 02:49 AM

https://www.foxitsoftware.com/support/security-bulletins.php

Mainly memory corruption bugs, many allowing Remote Code Execution (RCE), so high impact, and easy to exploit with email attachments.

Why are there so many vulnerabilities in a PDF reader?

PDF is a very complex data format, and Foxit is a "feature-rich" PDF viewer, and also support JavaScript in PDF.

Other PDF viewers also suffer from this

https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=PDF

Root cause of many exploitable memory errors: *parsing*

- Input need to be parsed before it can be processed
  - as IP packet, PDF document, HTML, JPG, mp3 …
- Complex languages make for many parser bugs

  https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=parsing

  Notorious example: Flash, the union of all audio/graphics/video formats you ever heard of & more
  JPG+GIF+PNG+H.264/MPEG4+VP6+MP3+AAC+Speex+PCM+ADPCM +Nellymoser+G7.11+..

  https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Flash
- If these parsers are memory-unsafe C(++) code, then such bugs cause security vulnerabilities with high impact, incl. RCE
- Bugs in input parsers are easy to trigger by attacker,
  with malicious input in an email attachment, on a webpage, ..

Example problem with complex format

Microsoft Security Bulletin MS04-028

- **Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution**
- **Impact of Vulnerability: Remote Code Execution**
- **Maximum Severity Rating: Critical**

Problem occurs with a zero length comment field, without content.

Buffer overflow in image processing is an ideal attack vector!

The victim only has to view an image (in email or on webpage) to get infected, and impact is high (namely remote code execution)

Even in ‘safe’ programming languages

OpenJDK: JPEG decoder input stream handling [CVE-2014-2421]

“A vulnerability in Oracle Java allows an unauthenticated, remote attacker to execute arbitrary code on a targeted system.
The vulnerability is due to improper bounds checks when the affected software parses certain JPEG images. An attacker could exploit this vulnerability by persuading a user to open a malicious web page or crafted malicious file that contains a crafted JPEG image. An exploit could allow the attacker to conduct a buffer overflow attack and execute arbitrary code on the system.

The following Oracle products are vulnerable: Java SE 8, SE 7u51, SE 6u71, SE 5.0u61, Java JavaFX 2.2.51, Java SE Embedded 7u51”

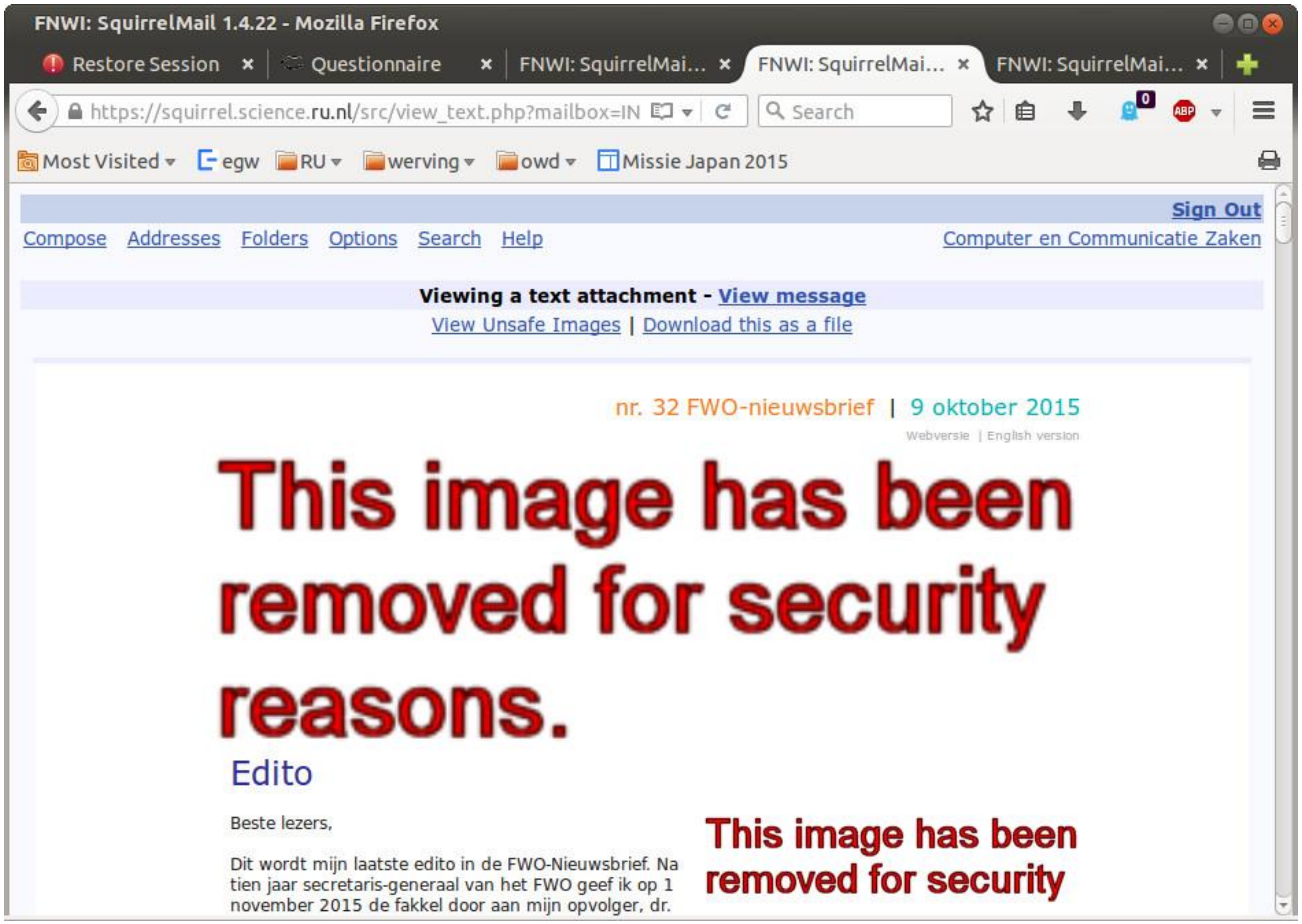
How is this possible in a library of a safe programming language like Java?

Native code in graphics library

Why do people use native code here?

Efficiency…

Countermeasure



FNWI: SquirrelMail 1.4.22 - Mozilla Firefox

https://squirrel.science.ru.nl/src/view_text.php?mailbox=IN

Sign Out

Compose Addresses Folders Options Search Help Computer en Communicatie Zaken

Viewing a text attachment - View message

View Unsafe Images | Download this as a file

nr. 32 FWO-nieuwsbrief | 9 oktober 2015

Webversie | English version

This image has been removed for security reasons.

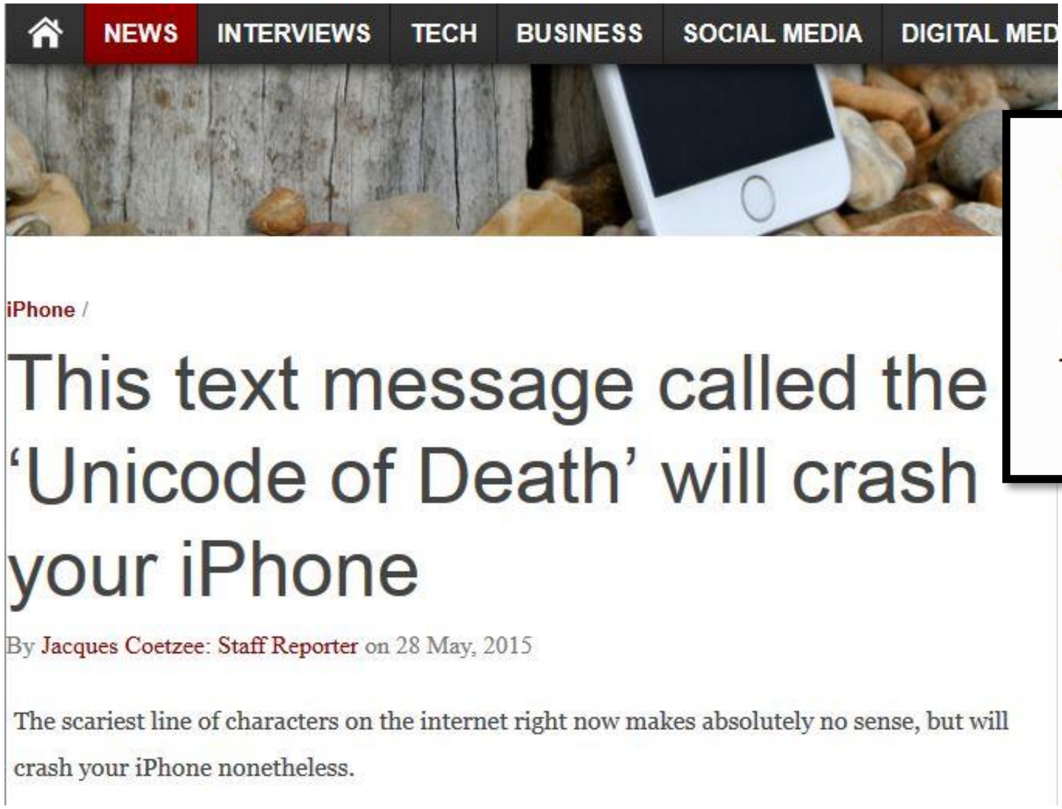
Edito

Beste lezers,

Dit wordt mijn laatste edito in de FWO-Nieuwsbrief. Na tien jaar secretaris-generaal van het FWO geef ik op 1 november 2015 de fakkel door aan mijn opvolger, dr.

This image has been removed for security

Complex input formats

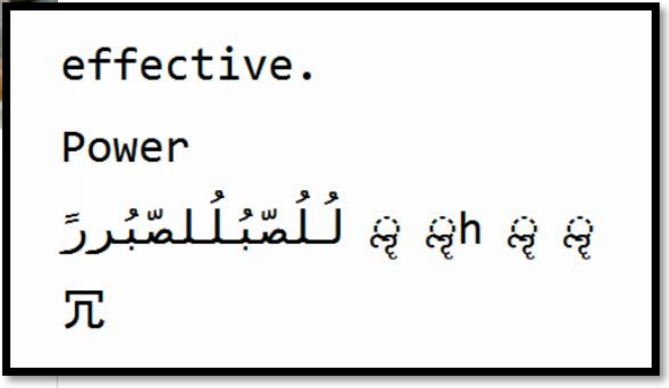


iPhone /

This text message called the 'Unicode of Death' will crash your iPhone

By Jacques Coetzee: Staff Reporter on 28 May, 2015

The scariest line of characters on the internet right now makes absolutely no sense, but will crash your iPhone nonetheless.



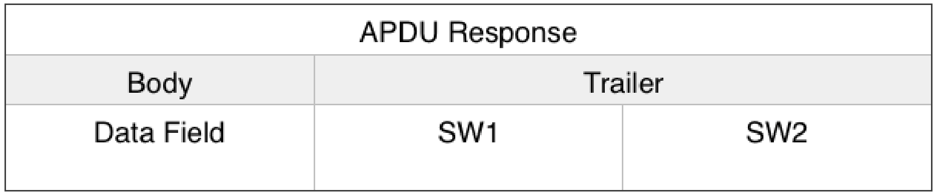
effective.
Power
لُلُصّبُلُلصّبُررً ॣ ॣh ॣ ॣ
冗

Example dangerous SMS text message

Different characters sets or characters encoding, are a constant source of problems.
Many input formats rely on underlying notion of characters.

Even processing simple input languages can go wrong

Sending an extended length APDU can crash a contactless payment terminal.



APDU Response		
Body	Trailer	
Data Field	SW1	SW2

[Jordi van den Breekel, A security evaluation and proof-of-concept relay attack on Dutch EMV contactless transactions, MSc thesis, 2014]

Exploiting features

(caused by expressivity)

Word & Excel & …

Favourite attack vector for attackers:

- **Powershell macros in Word & Excel document!**
- ***Why?*** **No need to craft complex shell code to exploit bugs, simply write a macro to exploit features!**

- **Also without macros using Windows DDE (Dynamic Data Exchange)**

Macro-less Code Exec in MSWord

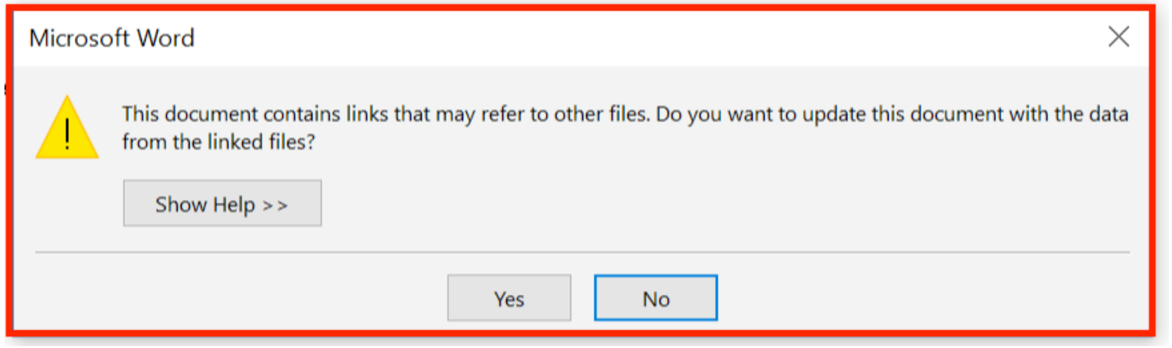
Reading time ~5 min

Posted by saif on 09 October 2017

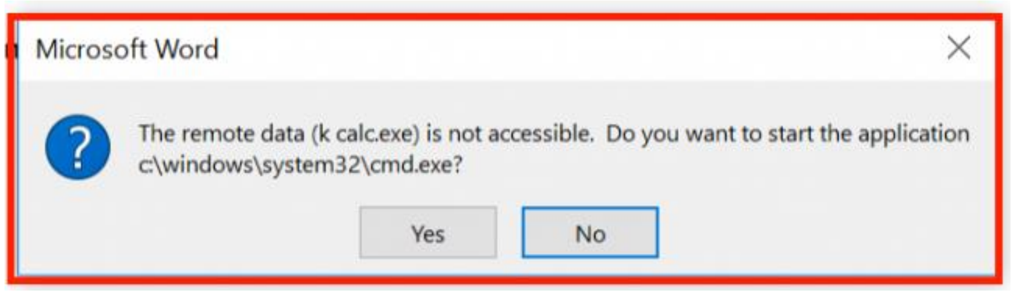
- **Also possible using emails in Outlook Rich Text Format (RTF)**

https://sensepost.com/blog/2017/macro-less-code-exec-in-msword/

DDE warnings


Microsoft Word

This document contains links that may refer to other files. Do you want to update this document with the data from the linked files?


Microsoft Word

The remote data (k calc.exe) is not accessible. Do you want to start the application c:\windows\system32\cmd.exe?

Microsoft considers DDE a feature, and not a bug, but did file a security advisory data autumn 2017

XML & zip bombs

Some input formats enable Denial-of-Service attacks

- a zip bomb (aka zip-of-death) is a small zip file of 40 KB that explodes to 4 GB when unzipped
- an XML bomb is a small XML file of 1 KB that explodes to 3 GB when XML parser expands recursive definitions (as part of canonicalisation)
  - aka billions laughs attack, as the original attack used the string LOL

Moral: any CPU cycles spent or any memory used in processing input (before the input has been validated) pose a security risk!

Injection attacks

Exploiting functionality of some back-end services

- OS command injection
- Path traversal
- SQL injection
- HTML injection (incl. XSS)
- Format string attacks
- LDAP injection
- Xpath injection
- ...

Tell-tale signs

- special characters or keywords that have a special meaning for the input language for this back-end service
  - This is a sign that data will be parsed & processed

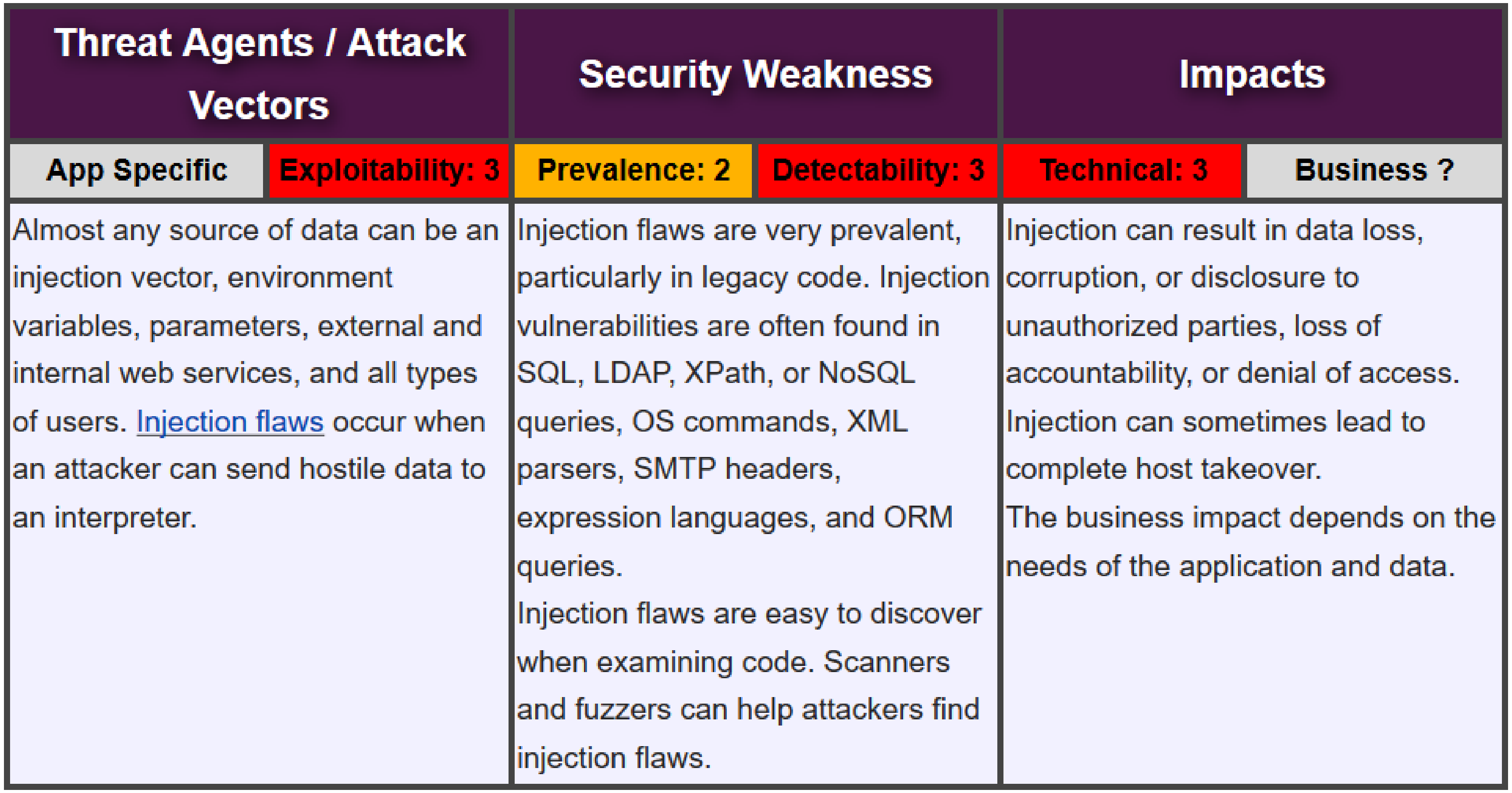
Injection attacks

These attacks abuse expressive power of some input language, eg the language of

- OS commands `erik@ru.nl; rm -fr /`
- Path expressions `../../../etc/passwd`
- SQL statements `'OR '1'='1';` `;DROP TABLES`
- HTML <script>...document.cookie...</script>
  - HTML5 includes JavaScript, DOM, CSS
- Format strings `%x%x%x%x%n`
- XML (//student[username/text..
- LDAP `admin)(&)`
- ....

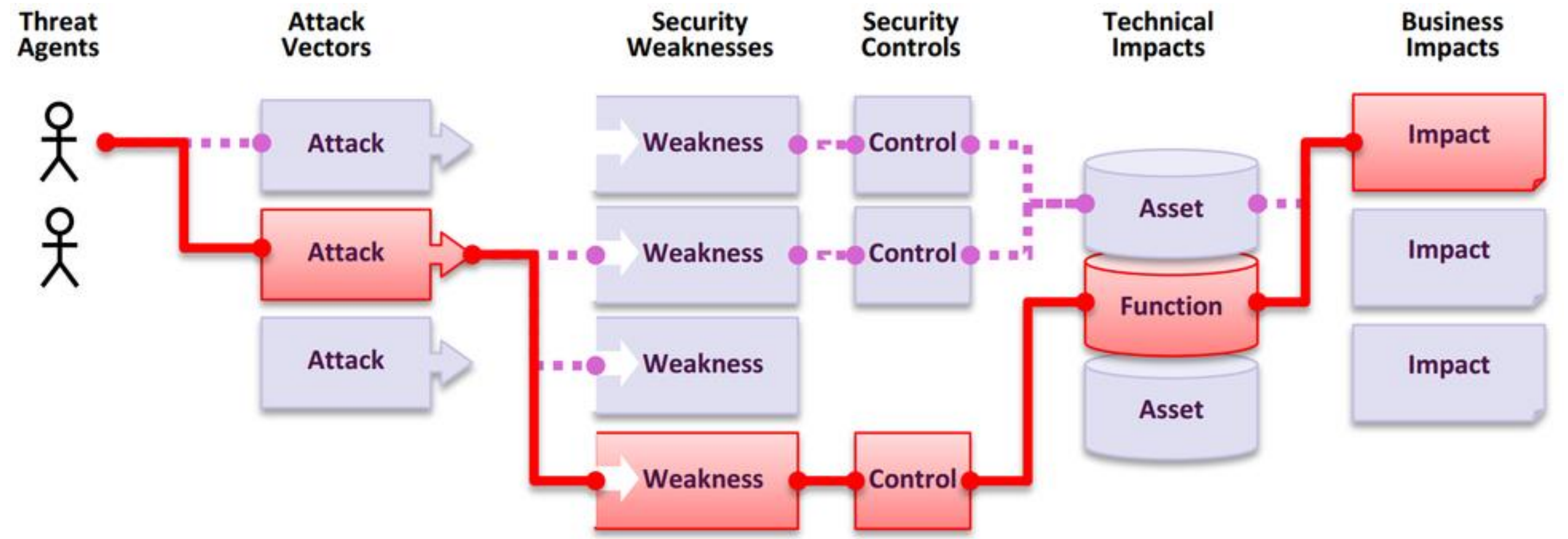
Injection Attacks : no. 1 in Top Ten

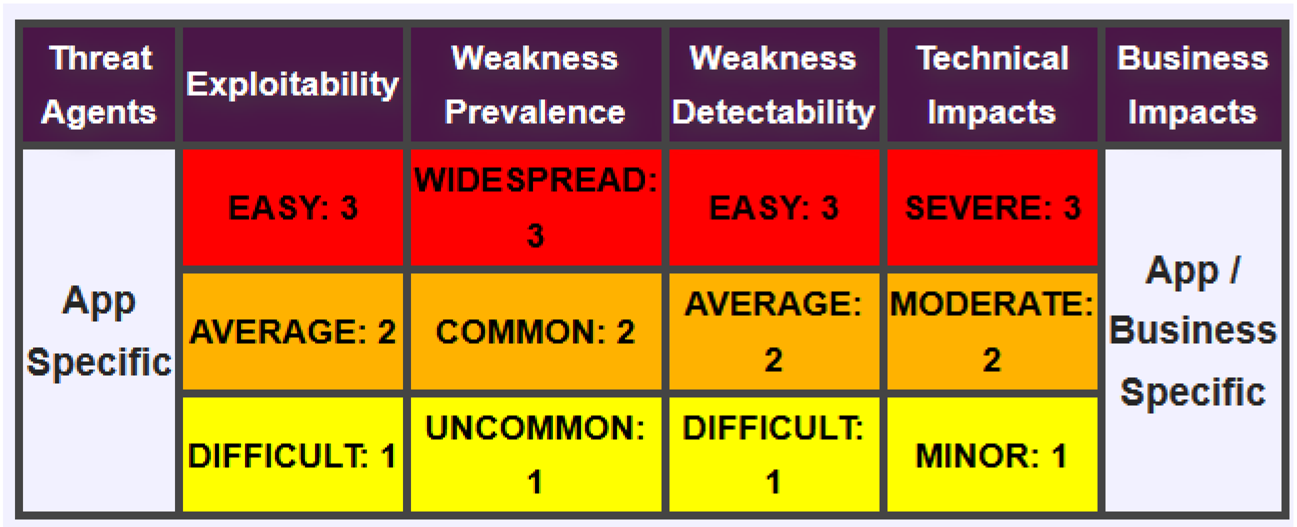
https://www.owasp.org/index.php/Top_10-2017_A1-Injection



Threat Agents / Attack Vectors		Security Weakness		Impacts	
App Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

OWASP Top 10 - Risk Rating

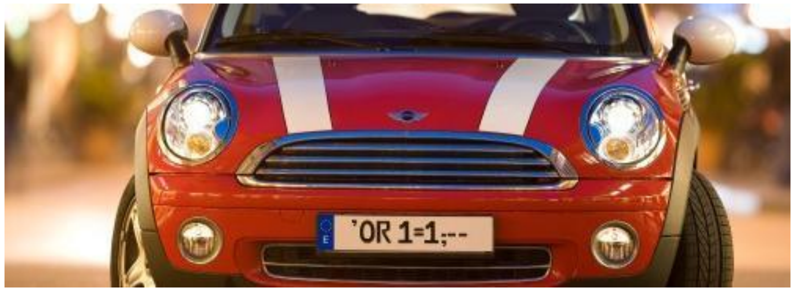




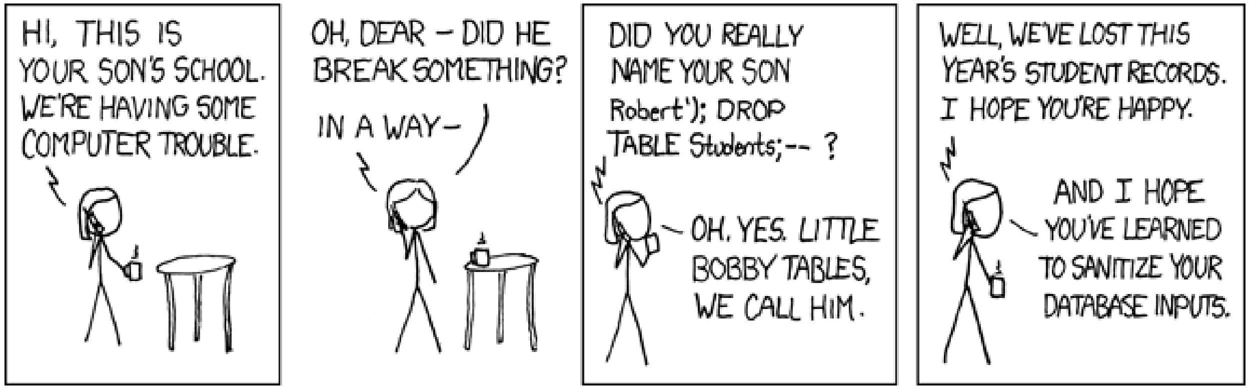
Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	SEVERE: 3	App / Business Specific
	AVERAGE: 2	COMMON: 2	AVERAGE: 2	MODERATE: 2	
	DIFFICULT: 1	UNCOMMON: 1	DIFFICULT: 1	MINOR: 1	

SQL injection

Exploiting the language of SQL queries



Or the language of SQL commands


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

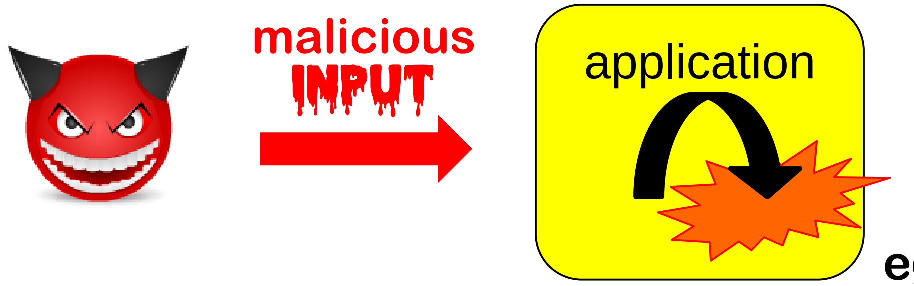
There are more interesting commands than `DROP TABLE`

for example `exec master.dbo.xp_cmdshell`

Processing vs injection/forwarding attacks

Processing Flaws

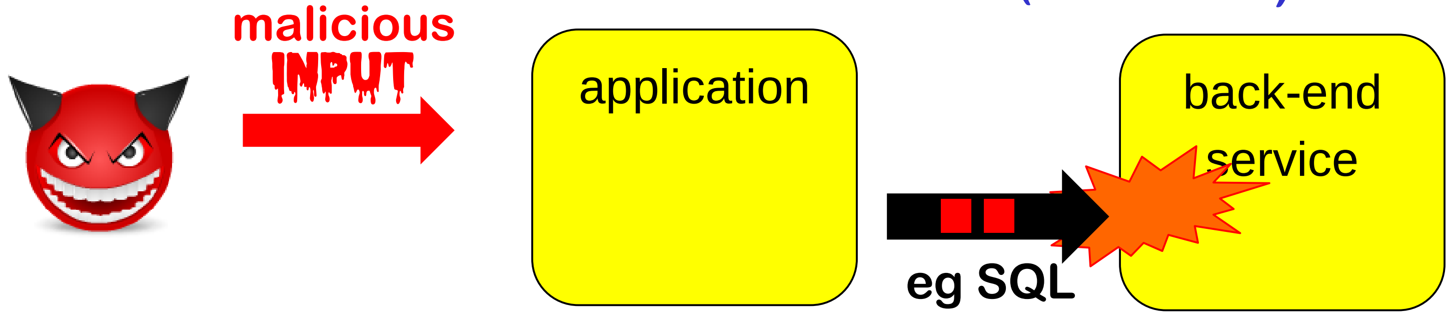
a bug !



eg buffer overflow
in PDF viewer

Injection aka Forwarding Flaws

(abuse of) a feature !



eg SQL
query

LDAP injection

An LDAP query sent to the LDAP server to authenticate a user

```
(&(USER=jan)(PASSWD=abcd1234))
```

can be corrupted by giving as username

```
admin)(&)
```

which results in

```
(&(USER=name)(&))(PASSWD=pwd)
```

where only first part is used, and `(&)` is LDAP notation for `TRUE`

There are also blind LDAP injection attacks.

XPath injection in XML

XML data, eg

```
<student_database>
 <student><username>jan</username><passwd>abcd1234</passwd>
 </student>
 <student><username>kees</nameuser><passwd>geheim</passwd>
 <student>
</student_database>
```

can be accessed by XPath queries, eg

```
(//student[username/text()='jan' and
         passwd/text()='abcd123']/account/text()) _database>
```

which can be corrupted by malicious input such as

```
' or '1'='1'
```

Path traversal aka directory traversal

File names constructed from user input – by string concatenation – can cause problems.

Eg suppose a program uses the paths

```
1.   "/usr/local/client-info/" ++ username
2.   "/usr/local/profilepictures/" ++ username ++ ".jpg"
```

Malicious usernames for attacker to inject:

```
1.      ../../../etc/passwd
2.      ../../../etc/passwd%00
```

null terminator `%00` means suffix `.jpg` will be ignored

Impact of path traversal

- Information leakage

  `../../../etc/passwd`

- Denial-of-Service (DoS)

  `../../dev/random` (is very long to read)

  `../../var/spool/lpr` (is impossible to read)

- Abitrary code execution?
  - If attacker can trick systems in *executing* the wrong file, ideally a file that the attacker can upload
  - Eg put javascript code in your Brightspace profile picture, and try to link to it somewhere in Brightspace

Beyond simple path traversal

Windows supports *many notations* for path names

– classic MS-DOS notation	C:\MyData\file.txt
– file URLs	file:///C|/MyData/file.txt
– UNC (Uniform Naming Convention)	\\192.1.1.1\MyData\file.txt
which can be combined in fun ways, eg	file://///192.1.1.1/MyData/file.txt

Some notations trigger *unexpected behaviour* , eg

- UNC paths to remote servers handled by SMB protocol aka Samba
  - SMB sends password hash to authenticate aka pass the hash
  - This can be exploited by SMB relay attacks on applications handling file names
    - CVE-2000-0834 in Windows telnet,
    - CVE-2008-4037 in Windows XP/Server/Vista, …
    - CVE-2016-5166 in Chromium
    - CVE-2017-3085 & CVE-2016-4271 in Adobe Flash,
    - ZDI-16-395 in Foxit PDF viewer

[Example thanks to Björn Ruytenberg, https://blog.bjornweb.nl]

More injection problems: OWASP list

- Blind SQL Injection
- Blind XPath Injection
- Code Injection
- Command Injection
- Comment Injection Attack
- Content Spoofing
- CORS RequestPreflightScrutiny
- Cross-site Scripting (XSS)
- Custom Special Character Injection
- Direct Dynamic Code Evaluation ('Eval Injection')
- Format string attack
- Full Path Disclosure
- Function Injection
- LDAP injection
- Parameter Delimiter
- PHP Object Injection
- Regular expression Denial of Service - ReDoS
- Resource Injection
- Server-Side Includes (SSI) Injection
- Special Element Injection
- SQL Injection
- SQL Injection Bypassing WAF
- Web Parameter Tampering
- XPATH Injection

[https://www.owasp.org/index.php/Category:Injection]

More obscure example: SSI Injection

Server-Side Includes (SSI) are instructions for a web server written inside HTML. Eg to include some file

```
<!--#include file="header.html" -->
```

If attacker can inject HTML into a webpage, then he can try to inject a SSI directive that will be executed on the server

Of course, there is a directive to execute programs & scripts

```
<!--#exec cmd="rm -fr /" -->
```

NB: with SSI injected code is executed *server-side*, with XSS injected code (javascript) is executed *client-side* in browser

Deserialisation attacks

Serialisation aka marshalling aka flattening aka pickling

- The process of turning some data structure into a binary representation
- Why?

  To transfer it over network

  or store it on disk (ie for persistence)
- Inverse operation of deserialisation, unmarshalling, … used later to reconstruct the object from the raw data

Deserialisation of malicious input can trigger strange behaviour…

- affects Java, PHP, python, Ruby, …

Deserialisation attacks [for Java]

Sample code to read in Student **objects from a file**

```
FileInputStream fileIn = new FileInputStream("/tmp/students.ser");
ObjectInputStream objectIn = new ObjectInputStream(fileIn);
s = (Student) objectIn.readObject(); // deserialise and cast
```

- **If file contains serialised** Student **objects,** readObject **will execute the deserialization code from** Student.java
- **If file contains other objects,** readObject **will execute the deserialisation code for that class**
  - **So: attacker can execute deserialisation code for any class on the CLASSPATH**
  - **Subtle issue: the cast is only performed *after* the deserialization**
- **If this object is later discarded as garbage, eg because the cast fails, the garbage collector will invoke its** finalize **methods**
  - **So: attacker can execute** finalize **method for any class on CLASSPATH**
- **Countermeasure: Look-Ahead Java Deserialisation to white-list which classes are allowed to be deserialised**

How to exploit deserialisation ?

- DoS
  - Attacker serialises a recursive object structure, and deserialization unwinds the recursion and never terminates
  - Attacker edits a serialised object to set an array length to MAX_INT

How to exploit deserialisation ?

- Arbitrary code execution
  - Possible by abusing rich functionality offered by commonly used libraries (eg. WebLogic, IBM WebSphere, JBoss, Jenkins, OpenNMS,Adobe Coldfusion…)
  - May even be possible from scratch, eg in python

```
DEFAULT_COMMAND = "netcat -c '/bin/bash -i' -l -p 4444"
COMMAND = sys.argv[1] if len(sys.argv) > 1 else DEFAULT_COMMAND
class PickleRCE(object):
        def __reduce__(self):
 import os
return (os.system,(COMMAND,))
```

More input problems: CWE classification

Some clusters in the CWE classification, esp.

CWE-990 Tainted Input

collect dozens of variants of input attacks

See http://cwe.mitre.org/data/definitions/896.html

CWE/SANS Top 25 (out of 732!) [Version 3.0]

- Improper Neutralization of Special Elements used in an SQL Command
- … ('OS Command Injection')
- Buffer Overflow
- .. ('Cross-site Scripting')
- Missing Authentication for Critical Function
- Missing Authorization
- Use of Hard-coded Credentials
- Missing Encryption of Sensitive Data
- Unrestricted Upload of File with Dangerous Type
- Reliance on Untrusted Inputs in a Security Decision
- Execution with Unnecessary Privileges
- Cross-Site Request Forgery (CSRF)
- …('Path Traversal')
- Download of Code Without Integrity Check
- Incorrect Authorization
- Inclusion of Functionality from Untrusted Control Sphere
- Incorrect Permission Assignment
- Use of Potentially Dangerous Function
- Use of a Broken or Risky Cryptographic Algorithm
- Incorrect Calculation of Buffer Size
- Improper Restriction of Excessive Authentication Attempts
- URL Redirection to Untrusted Site ('Open Redirect')
- Uncontrolled Format String
- Integer Overflow or Wraparound
- Use of a One-Way Hash without a Salt

Still to come

- Countermeasures, incl.
  - input validation
  - output encoding
  - sandboxing

  but also LangSec approach to address these root causes

- (Optional) lecture on SQL injection, XSS?