Exploratory testing

Índex

- Introducció
- Consells pràctics

Índex

- **Introducció**
- Consells pràctics

Introducció

- Què és i què no és el ET:
  - **No** és una **tècnica** concreta i específica
  - **Si** és una **actitud**, una manera de fer, una mentalitat
  - Un procés on es **testeja mentre s'explora**
  - Sovint se l'anomena *ad hoc* testing
  - És un procés on, simultaneament (i amb retroalimentació):
    - S'apren
    - Es dissenya el test
    - S'executa el test

      ⇒ L'estrategia del test es va modificant a mesura que coneixem el sistema.

  - Resultats del ET:
    - Un conjunt de notes sobre el producte
    - Errors trobats
    - Un informe concis dels passos realitzats al fer el test

Introducció

- **No** depèn de **instruccions pre-establertes** o programades (*defined* o *scripted*), per tant
  - Com que és un procés creatiu, **depen** de **l'habilitat** del **tester**.
  - El tester **no** es pregunta "quin **test m'han dit** que faci o *em toca fer?*" sinó "quin **bon test** puc fer ara?"
  - Pot ser més productiu que *scripted-tests*
- Com que és una activitat creativa:
  - **ET per parelles**. Mentre un tester pensa i executa nous testos, l'altre pot pensar nous tests, provar el mateix en altres plataformes, buscar documentació.
- **NO** és **incompatible** amb test preestablert o *scripted* (*test cases*, *test scripts*, etc) sinó que és **complementari**.
  - ⇒ ... i de fet, tampoc es pot dir si es fa o no es fa ET, sinó que el ET és un **espectre continu** que va del pur scripted test al ET pur.

Introducció

- **Elements** importants en el ET:
  - **Test design**: Un tester de ET és principalment un test designer, ja que es requereix l'habilitat d'analitzar el producte, avaluar riscos, pensar criticament, ...
  - **Observació critica**: Un tester de ET és un bon observador per veure comportaments foscos o no evidents a simple vista.
  - **Pensament crític**: Necessari per analitzar un sistema.
  - **Produir noves idees**: Són necessaries per definir nous *tests* (també s'anomenen *atacks)*.

Utilitat

- En quines situacions és útil? Quan es necessita
  - **Feedback immediat** d'un nou producte o funcionalitat.
  - **Aprendre ràpid** el producte.
  - Trobar el **bug més important** i es disposa d'un **temps limitat**.
  - **Aillar** i investigar un **defecte** concret.
  - **Investigar l'estat** d'un **risc** concret.
  - Ja s'ha realitzat test **scripting** i es vol una **nova aproximació**.
  - **Improvitzar** test **scripting**.
  - **Analitzar** el **producte** i fer **test planning**.
  - Millorar els tests existents

    ⇒ és majoritariament útil un cop el producte ja s'ha (parcialment) desenvolupat, pero també mentre s'està desenvolupant!

Índex

- Introducció
- **Consells pràctics**

Consells pràctics

- L'experiència demostra que els bons "attacks" a un software quan es fa ET són:

- Attack 1 : **Apply inputs that force all error messages to occur at least once**

WHY

  - Ability to **detect** bad input or to **appropiately respond** to it is an essential characteristic of good software
  - Applicable to any software
  - Error cases require additional error-checking code, outside the main-line functional code, usually done sloppily or missing
  - **Check error messages** design: informative, constructive, offer solutions
  - It's **difficult** to implement **recovery** from exceptions

HOW to determine success ?

- missing error cases programming : **application hangs** or crashes
- **misplaced** error **messages**
- **uninformative** error **messages**

HOW to conduct the attack ? In an **input field** enter an **invalid**

- input **type** (non-numeric, decimal point...)
- input **length** for alphanumeric inputs
- **boundary and out-of-limits** values, once you have guessed the internal data type (short, long, character, float...)

EXAMPLE

- Word 2000, Insert $\rightarrow$ Index and Tables $\rightarrow$ Columns ≥ 5
  Error message shown twice.

- PowerPoint 2000, Insert $\rightarrow$ Object $\rightarrow$ MVSA Button Class
  Force quit.

Attack 2 : **Apply inputs that force the software to establish default values**

WHY

- **Use** of **variables before initialization** can make the software fail
- Force the application to establish or use data that may **not** have been **initialized** with a **suitable default value**
- Developers forgot to use tools or compile settings to detect this situation

HOW to determine success ?

- application **hangs** or crashes
- a **random value** produces a **wrong result**
- GUI shows wrong default values

HOW to conduct the attack ?

- **Look for** option, **configuration**, **set-up** sheets/dialogs/panels/screens
- **Accept default values shown**. Sometimes the application *counts on* the user to enter a value
- If a **value** is **displayed**, **delete** it (enter **null values**)
- Change a default value to another acceptable value and change it back again

TESTING THE UI : INPUTS AND OUTPUTS

Attack 3 : **Explore allowable character sets and potentially special meaning values in string fields**

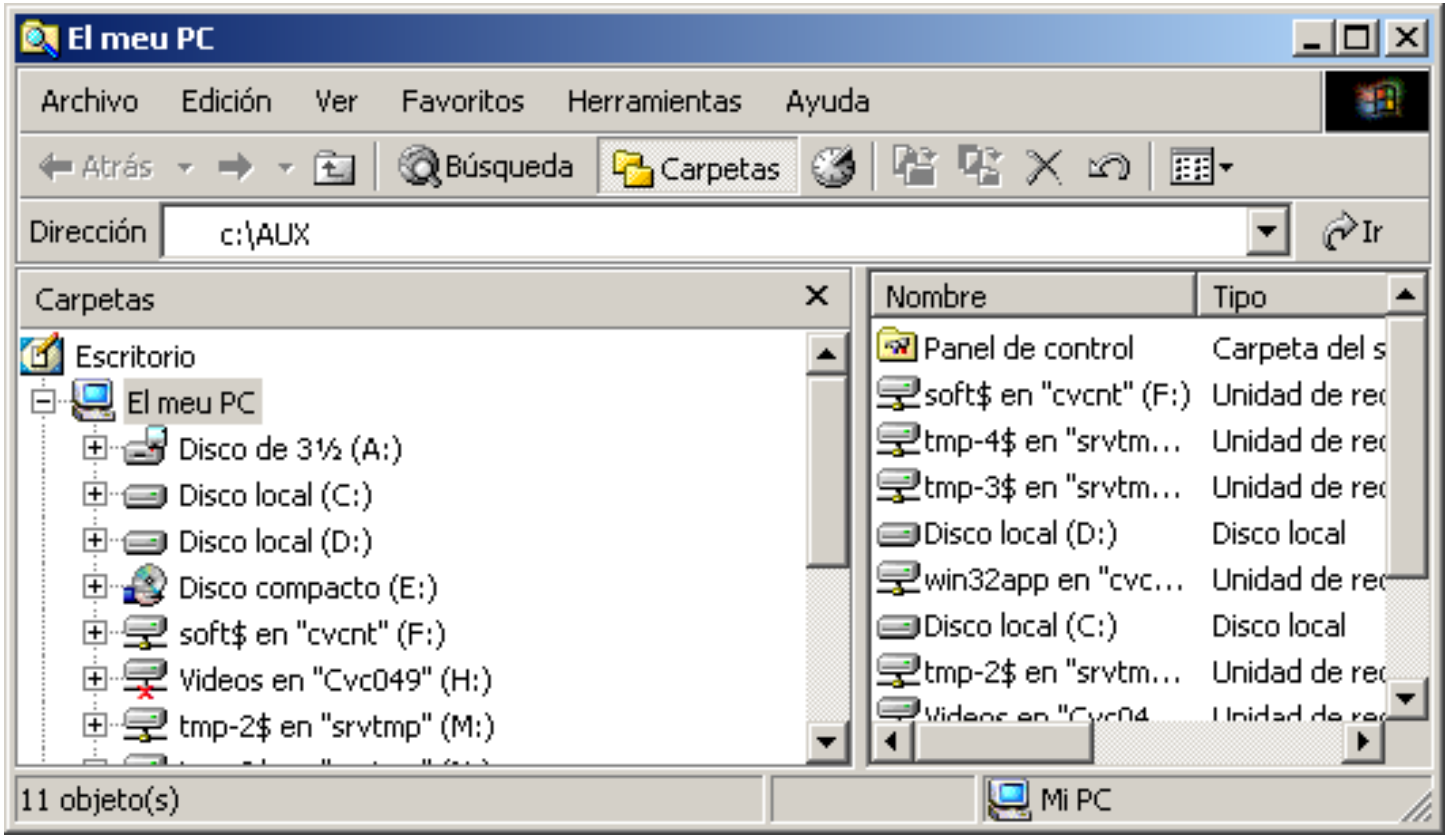
WHY

- Developers fail to write error-handling code for **special characters** like **“**, \, % , NULL (^@), EOF (^Z)
- Same for **reserved keywords** from the operating system or programming language like \n, + in C++ or &, $, | in Unix

TESTING THE UI : INPUTS AND OUTPUTS

EXAMPLE

Enter manually `file://C:\AUX` in the URL field of Internet Explorer 5.5 or `C:\AUX` in Windows 2000 Explorer. The process hangs because AUX is a special device name.



TESTING THE UI : INPUTS AND OUTPUTS

Attack 4 : **Overflow input buffers in string fields or parameters**

WHY

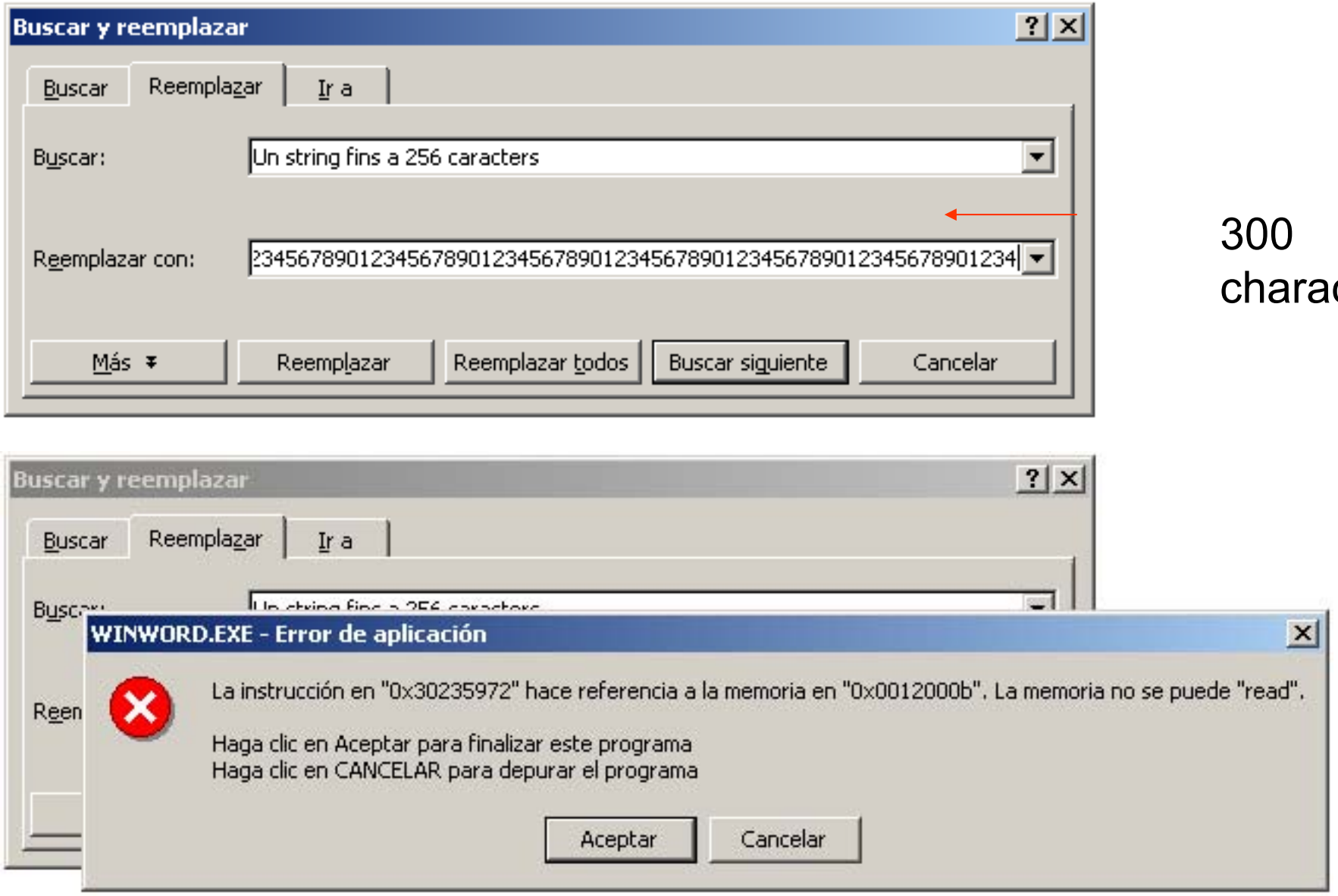
- Enter **long strings** to **overflow** input **buffers** in fields where there are not explict constraints on input length
- If a **hacker** attaches an executable string to the **end of the input** string it may get **executed**

HOW to conduct the attack ?

- type 0123456789 and use Copy/Paste to increase size by 10 each time ; finally, select all, Copy and Paste to a string field

TESTING THE UI : INPUTS AND OUTPUTS

EXAMPLE : Word 2000 Find & Replace



300 characters

TESTING THE UI : INPUTS AND OUTPUTS

Attack 5 : **Find inputs that may interact and test combinations of their values**

WHY

- Test value combinations in which **each value** is **tested** but whose **combination** is **not** and may cause the software to fail
- Developers, particularly developers working on a common code, overlook the **relationship between two input variables**
- Checking a single variable is easy. When multiple variables have to be checked, the control structures (nested `if-then-else`) become more complex

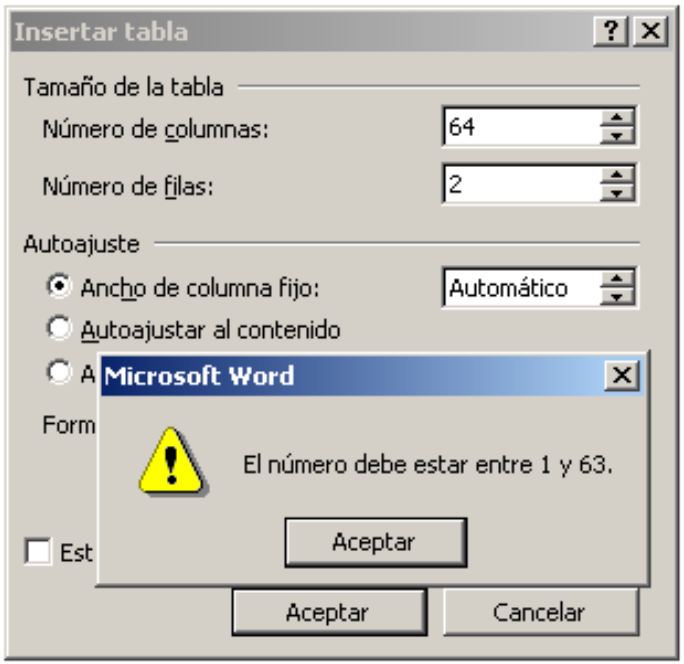
TESTING THE UI : INPUTS AND OUTPUTS

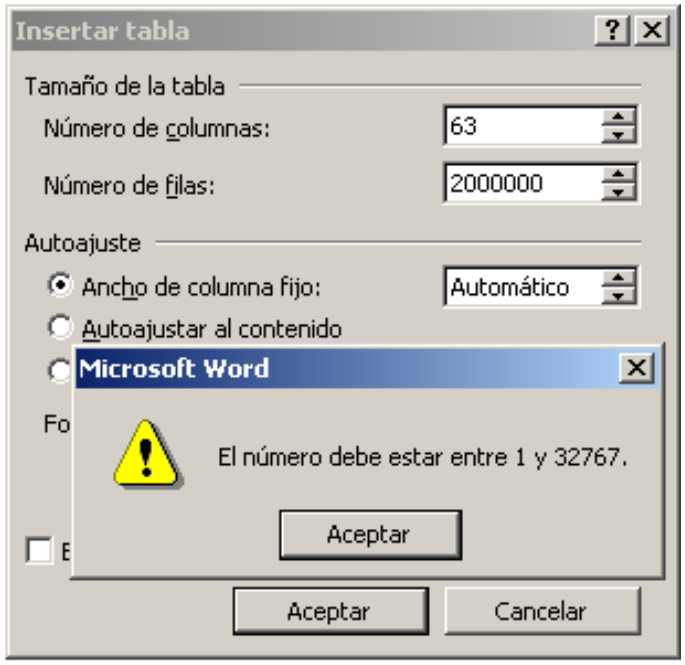
HOW to conduct the attack ?

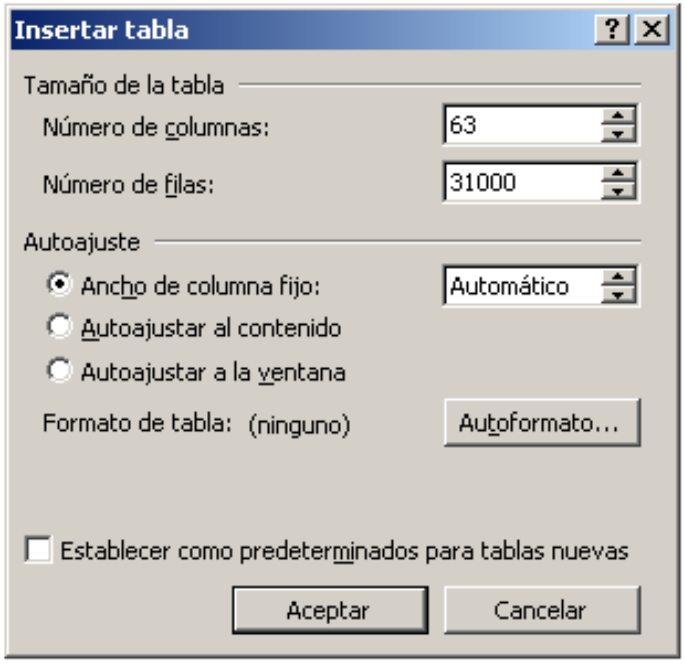
- **Identify** potentially **related variables** as those
  - describing a **common internal data structure**, e.g. number of rows and columns of an image
  - Used toghether in an **internal computation**, e.g. upper, lower, left and right page margins may be used to calculate page size
- Determine a **large** and **small** value for each variable
- From all possible combinations, select a feasible number of test cases

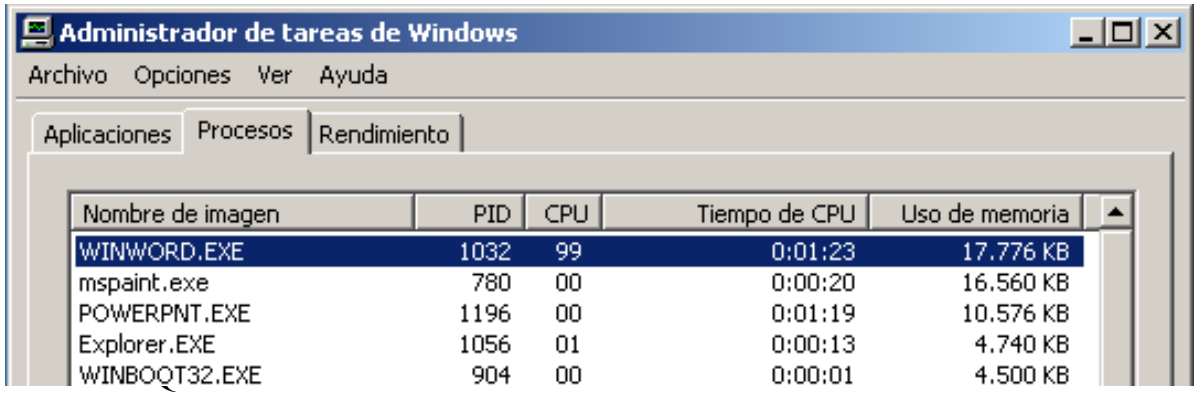
TESTING THE UI : INPUTS AND OUTPUTS

EXAMPLE : Word 2000 Table → Insert dialog









application hangs
if columns ≥ 50 and
rows ≥ 30000

TESTING THE UI : INPUTS AND OUTPUTS

Attack 6 : **Repeat the same input or series of inputs numerous times**

WHY

- Just because something worked once does not mean it will work twice
- If an application accepts **input inside a loop** (receive input + process input + wait for another input)
  Repetition may have the effect of **allocating** increasingly **more and more resources**, therefore stressing the application's data space
- **First iteration** works with internal **default** values for some variables. **Subsequent iterations** with the same input may work with different values for them, yielding a **different result**

TESTING THE UI : INPUTS AND OUTPUTS

HOW to determine if this attack exposes failures

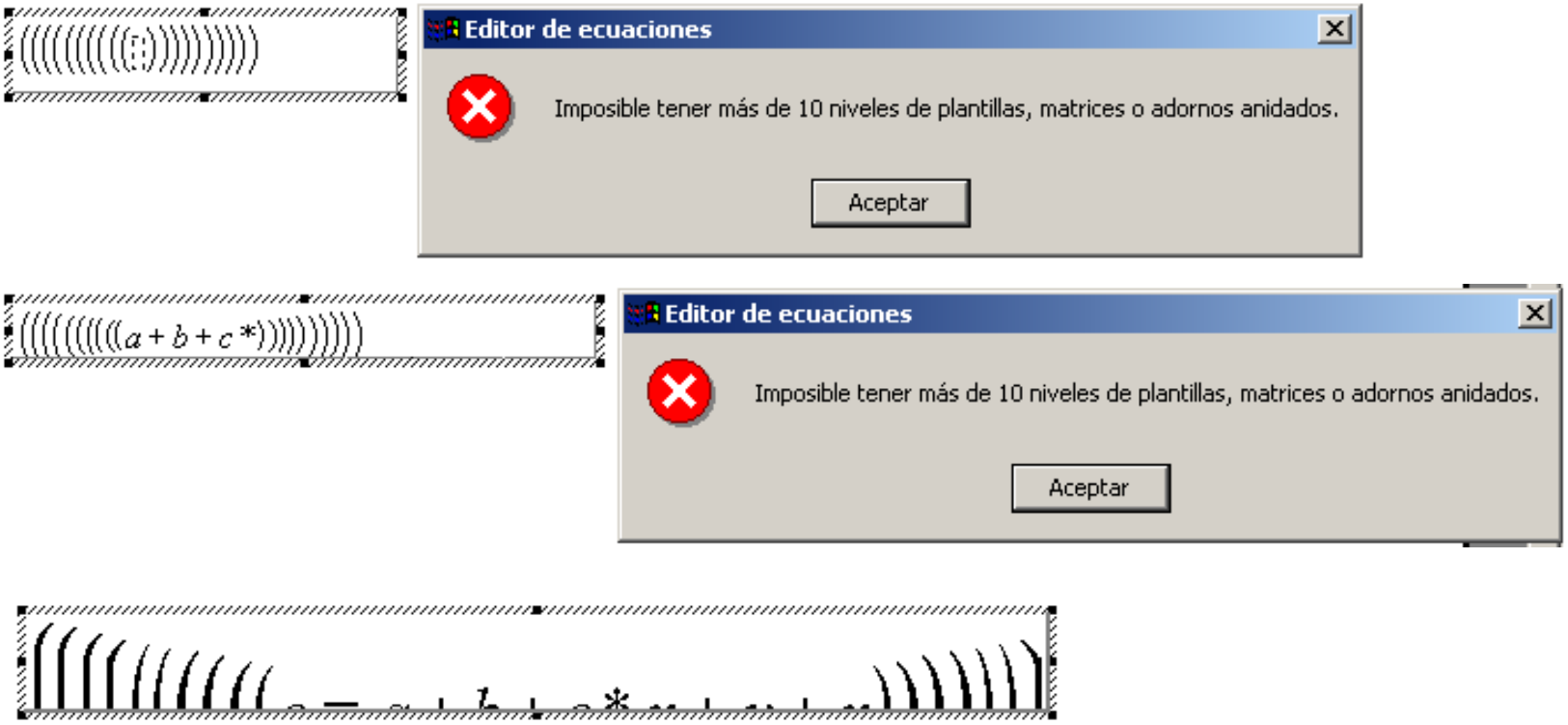
- **Memory run out** may produce **slow execution** and **lack of responsiveness**
- For **GUIs**, **misplaced**, too often or **not often** enough **refreshing**
- Combine with a **memory leak** detector

HOW to conduct this attack

- **Select inputs** that you **expect** the user to **apply numerous times**, e.g. matching parenthesis into an equation editor, formatting paragraphs in a text editor, downloading web pages...
- **Apply** them **over and over** until you reach an internal limit or find out a failure

TESTING THE UI : INPUTS AND OUTPUTS

EXAMPLE : nested parenthesis in Word 2000 + Microsoft Equation Editor 3.0



TESTING THE UI : INPUTS AND OUTPUTS

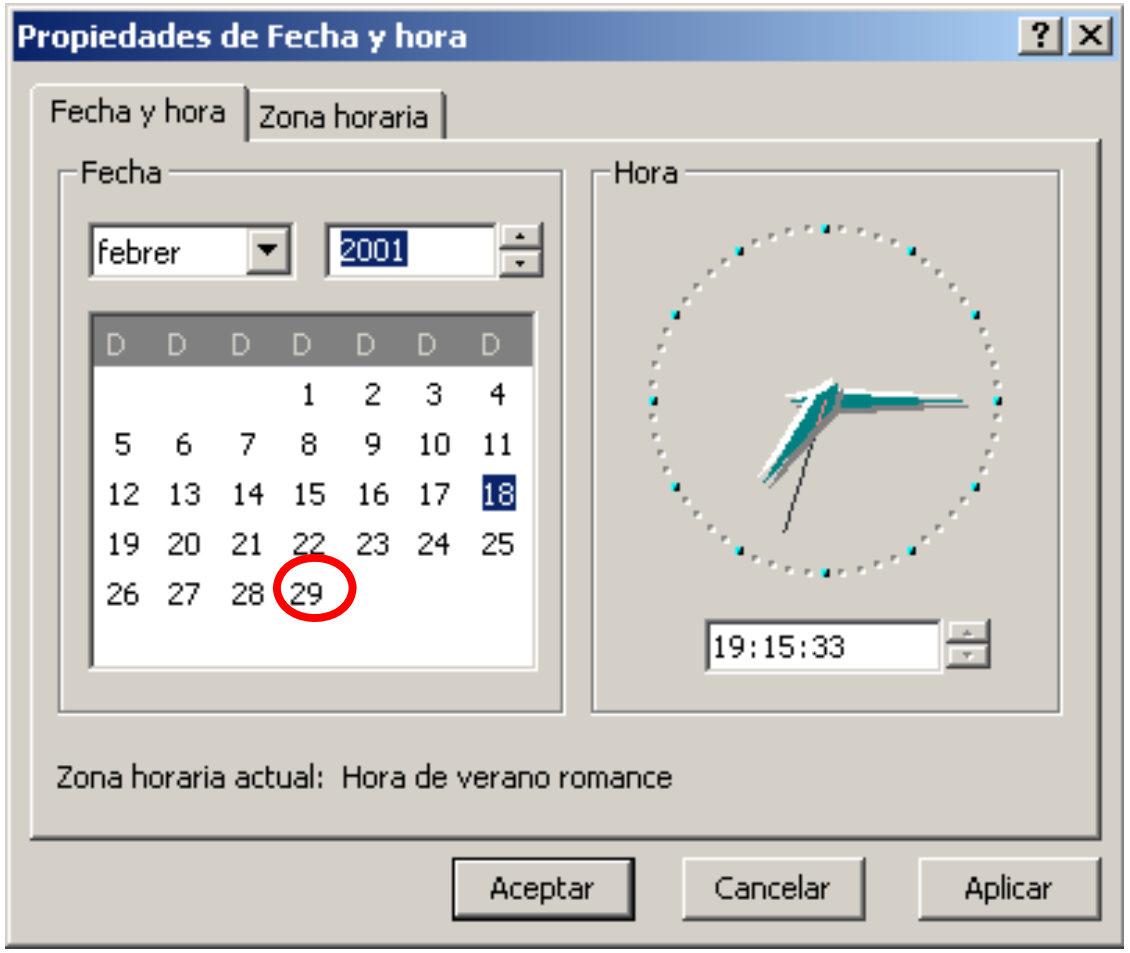
Attack 7 : **Get to know the problem domain and think through special cases of input combinations that force invalid outputs to be generated**

WHY

- Developers with an **uncomplete understanding** of the solution will have a **hard time coding it**
- In most cases, **errors** are **overlooked** special **cases**, inputs that **do not follow the general rule** and must be handled in special ways

TESTING THE UI : INPUTS AND OUTPUTS

EXAMPLE : error fixed in Windows NT service pack 5



TESTING THE UI : INPUTS AND OUTPUTS

Attack 8 : **Force properties of an output to change**

WHY

- Some outputs are just sent to the user and forgotten. Others are editable, have properties: size, color, length... and might come back to the application
- The **developer writes code** that sets **initial** or default properties and then code that allows the **user** to **edit** them. Both are separated and after some change get **inconsistent**.

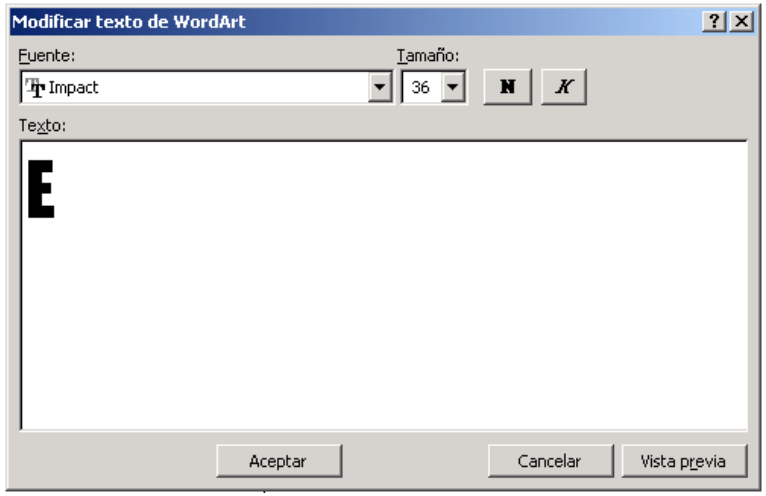
HOW to conduct this attack

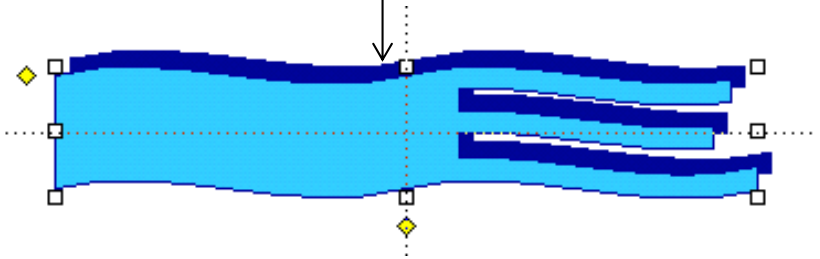
- Review outputs that can be generated with editable properties
- Edit each one

TESTING THE UI : INPUTS AND OUTPUTS

EXAMPLE : Powerpoint 2000 Word art

Editem el texte





but the font has not been changed

TESTING THE UI : INPUTS AND OUTPUTS

Attack 9 : **Force the screen to refresh to find rendering problems in applications with graphical output**

WHY

- **Refreshing is a major problem** of GUIs : refresh too often and the application slows, refreshing not often enough requires the user to ask for refresh or prevents him from doing his task

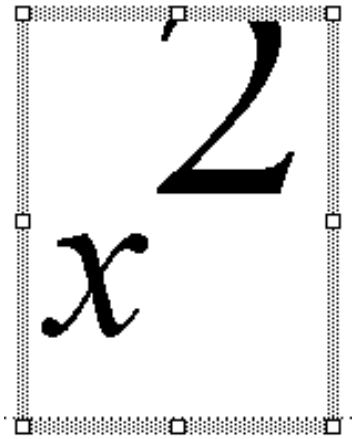
HOW to conduct this attack

- **add, delete** or **move** objects on the screen
- Varying moving **distance**, **overlapping**, mixed objects types

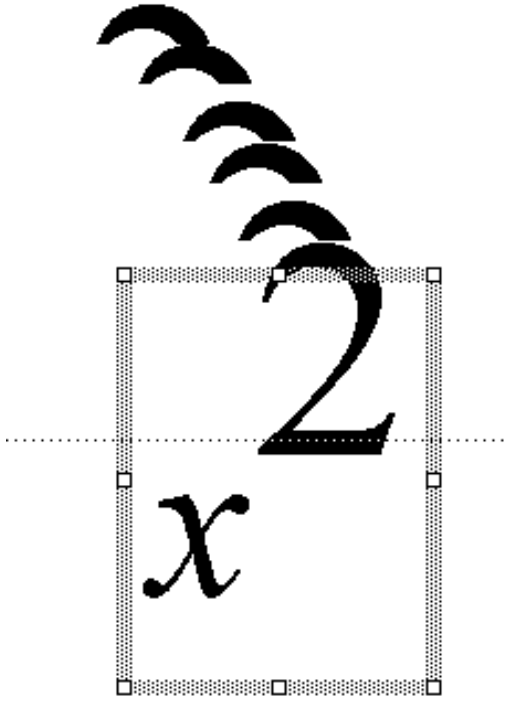
TESTING THE UI : INPUTS AND OUTPUTS

EXAMPLE : PowerPoint 2000 move a paragraph with a superscript of a large font size

x^2



Put PPT in slide view mode and sligtly move downwards the paragraph



TESTING THE UI : DATA AND COMPUTATION

Attack 10 : **Force data structures (known from source code or guessed) to store too many or too few values**

WHY

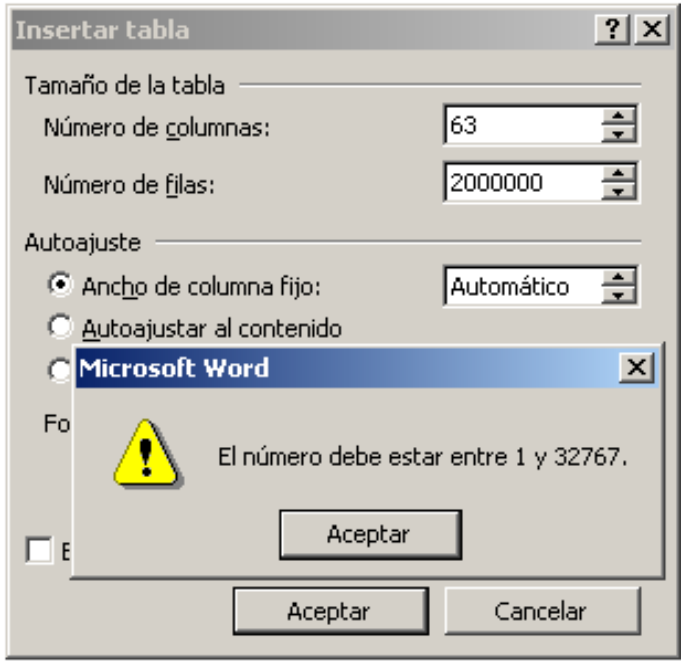
- Incorrect coding of **out of bounds** checking can cause data to be corrupted and improper access
- Typically on **fixed size** data structures like **arrays**

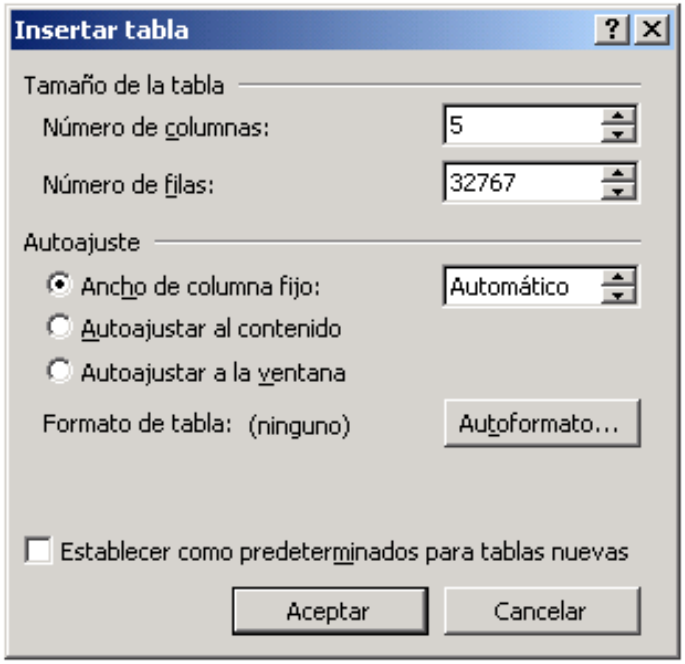
HOW to conduct this attack

- pay special attention to structures whose limits fall on the **boundary** of built-in data types : 256, 1024, 32767
- Force **delete** when the structure is **empty**

TESTING THE UI : DATA AND COMPUTATION

EXAMPLE





Either the application hangs or does it later after inserting a few more columns

Tables become a suspect feature from now on! Following exploratory testing, concetrate attacks on them

TESTING THE UI : DATA AND COMPUTATION

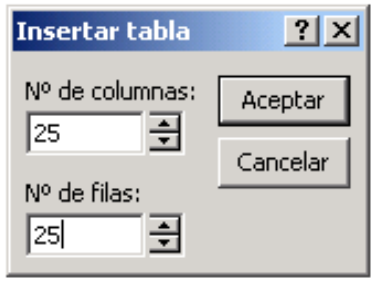
Attack 11 : **Investigate alternative ways to modify internal constraints on data properties, besides size at data structure creation**

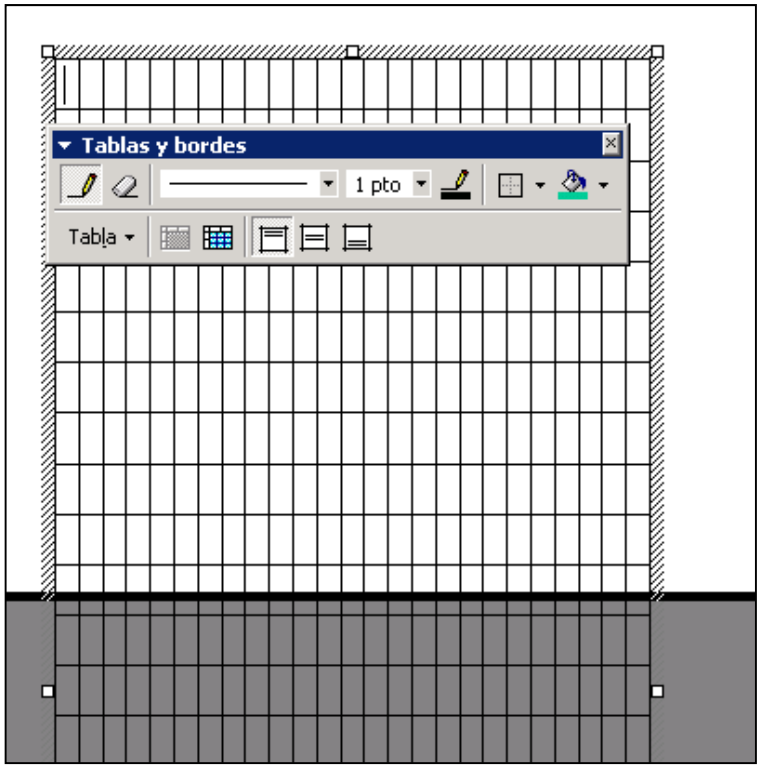
WHY

- Generalized form of previous attack.
- **Constraints** on data **properties** are checked at data **creation** and **modification**, but perhaps at **distant code locations**. The two redundant chunks of code become **inconsistent** after some change

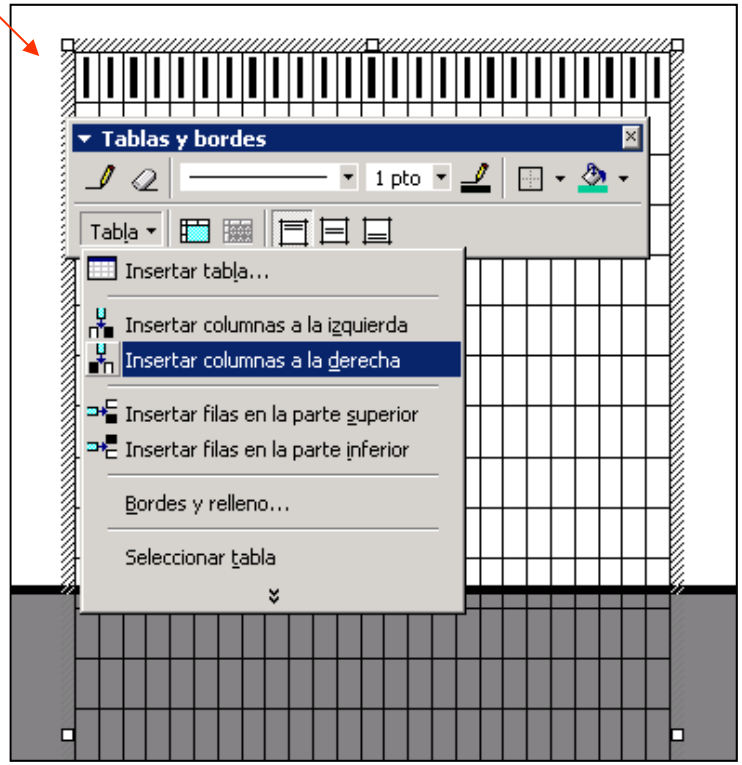
TESTING THE UI : DATA AND COMPUTATION

EXAMPLE : PowerPoint 2000 does not allow tables greater than 25×25 at Insert → Table but once created it can be enlarged and the application hangs





new row



TESTING THE UI : DATA AND COMPUTATION

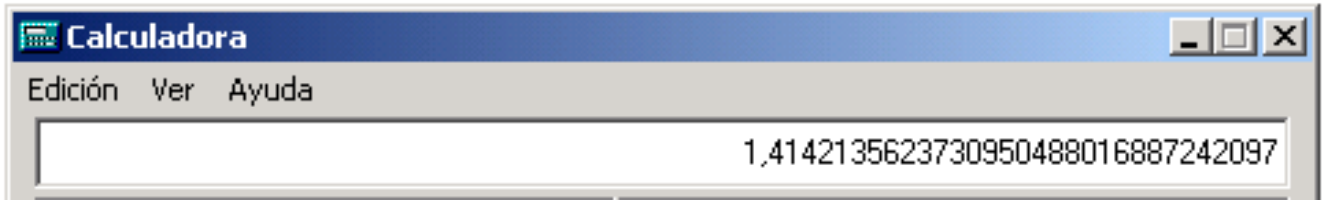
Attack 12 : **Experiment with operand and operator combinations which may cause the software to fail**

WHY

- Computations occur everywhere in a software application: besides mathematical expressions, loops, assignements
- Each **operator**, like division, have a **valid range** of operators. Developers must write significant **error-checking code** but sometimes **fail** to do it.

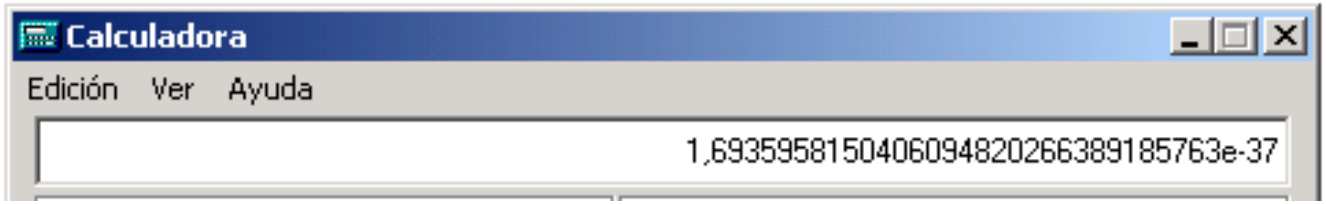
TESTING THE UI : DATA AND COMPUTATION

EXAMPLE



square root of 2.0

x^{2^2}



$4 - x^{2^2}$

As expected, the difference is not zero. The real problem is that the interface concealed the error to the user.

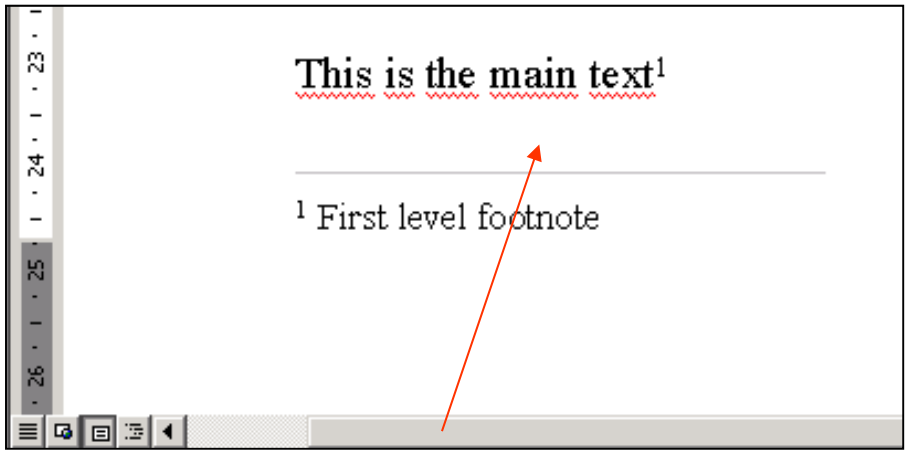
Attack 13 : **Force a function to call itself recursively**

WHY

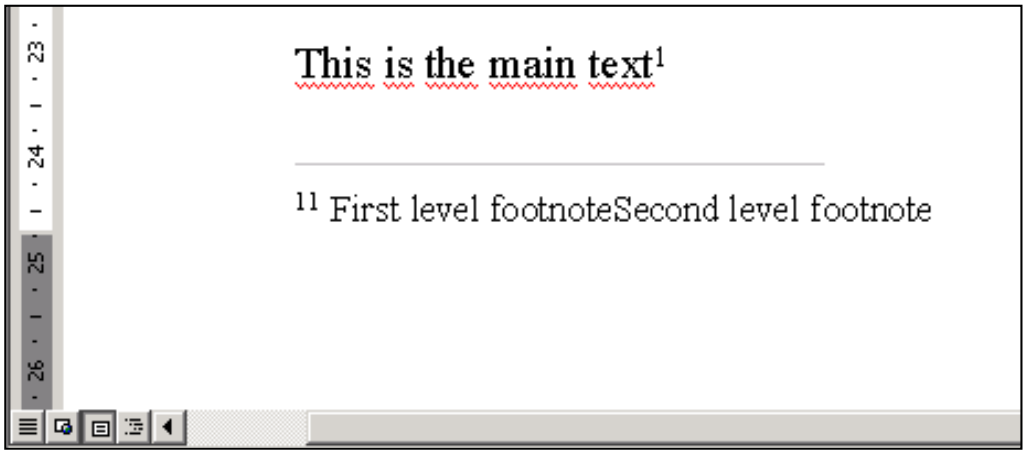
- **Recursion** and loops can be **problematic** if the number of recursion levels / iterations is not bound to a finite number. But potential errors are discovered during unit testing
- Even though an object can interact with other objects, it **might be** that it **can not interact with itself** or a copy of itself. *(e.g. a web page with a script that automatically executes when the page is loaded, and the script reloads the web page)*.
- Developers may fail to write code to handle this special case
- The **error manifests** as a **heap overflow** that causes the application to **crash**

TESTING THE UI : DATA AND COMPUTATION

EXAMPLE : in Word 2000, insert a footnote within a footnote



Insert footnote



Numbering of the footnote changes and no new footnote is inserted

TESTING THE UI : DATA AND COMPUTATION

Attack 14 : **Find features that share data or interact poorly**

WHY

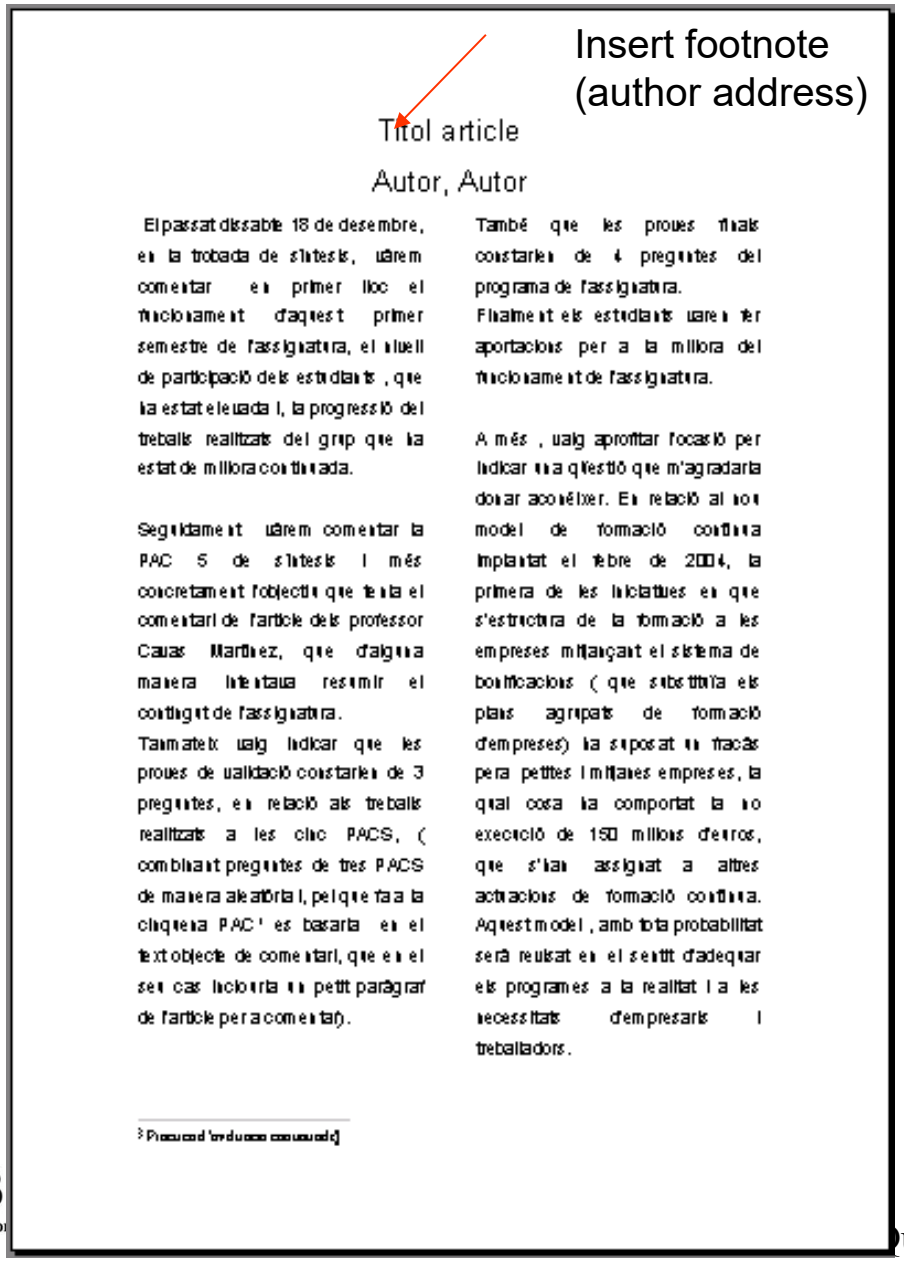
- A **feature** can be tested in isolation. However it could only be **buggy** when it is **used in combination** with oher **features** with which it **shares data**

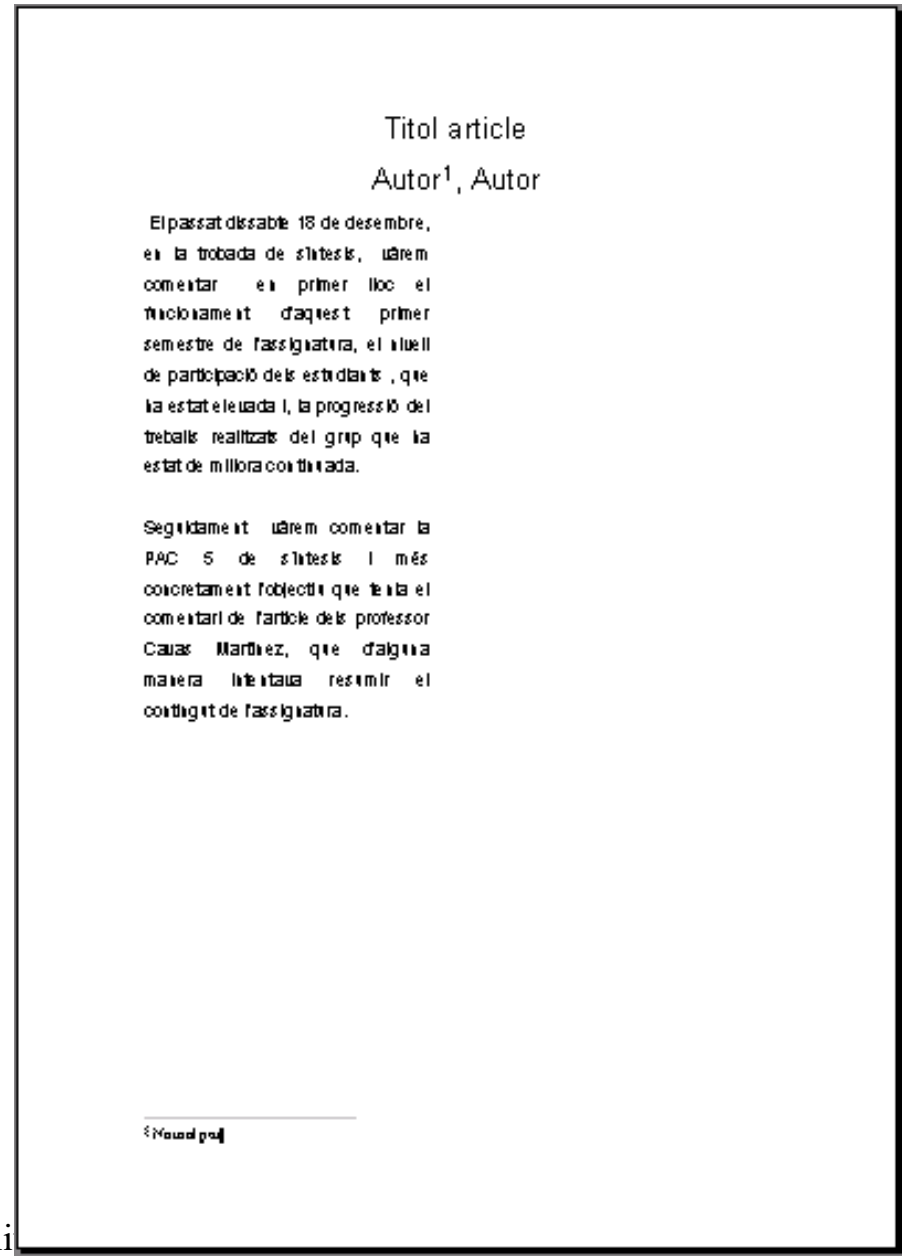
- **Data sharing** *and* **different constraints** on them, e.g. size, of which developers are unaware

- But *which* feature pairs?
  - accept the same inputs
  - produce similar outputs
  - one can get in the way of the other

TESTING THE UI : DATA AND COMPUTATION

EXAMPLE : combining footnotes and two-columns in Word 2000





TESTING OTHER USERS

Testing from the file system interface

1. **Fill** the file system to its **maximum capacity**
2. Force the media to be **busy** or unavailabe
3. Simulate **damaged** storage media
4. Assign an **invalid file name**
5. Vary **file permissions**
6. Vary or **corrupt file contents**

Testing from the software / OS interface

1. **Memory** faults (**insufficient**, lock)
2. **Network** faults (**disconnect**, network **not installed**, network **down**, wrong Winsock version, no ports available...)

Fault injection with HEAT [2] (Hostile Environment Application Tester)

CONCLUSION

1. **Simply going through** the attacks can exercise a great deal of the application functionality

2. A **successful attack** usually means experimenting with **dozens of possibilities** and pursuing a number of dead-ends

3. Attacks provide **concrete goals** in mind from which to **desing test cases**

4. Therefore, every test has a purpose and progress can be monitored

5. Planning on the fly, but *effective* way to find out errors