Computing 101 Tutorials

Complete each section with activity

Contents

[1: Bash memory example 1](#_Toc111537085)

[2: Bash multi-threading gone wrong example 3](#_Toc111537086)

[3: Character encoding attacks 4](#_Toc111537087)

# 1: Bash memory example

**Required Knowledge:**Read sections  
<https://en.wikipedia.org/wiki/Memory_paging#Thrashing>

<https://en.wikipedia.org/wiki/Memory_leak>

**Activities:**  
  
1) Save all work on your computer  
  
2) No really, save everything on the computer, as if you expect it to crash, soon…

3) Create a bash script “never-waste-ram.sh” with the following code  
RAM\_BOMB="YARGH"

while true

do

RAM\_BOMB="${RAM\_BOMB}${RAM\_BOMB}"

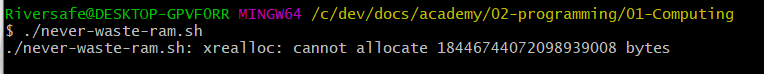
done

4) run ./never-waste-ram.sh

Graphical user interface, text, application, email

Description automatically generated

5) let it run in the background for a minute, try to do things

The slowness is called “thrashing” learn to recognise it, close the program, and know that if you have “thrashing” going on, it’s a RAM issue   
  
If you’re lucky your OS will recognise the leak and stop it  
<foreshadowing>that doesn’t always happen</foreshadowing>  
  
  
***lesson****: RAM management is hard, learn to spot thrashing, and catch memory leaks  
next lesson, what if there are multiple small leaks in several thousand programs*

# 2: Bash multi-threading gone wrong example

**Required Knowledge:**Read sections  
<https://en.wikipedia.org/wiki/Fork_bomb>

**Activities:**  
  
1) Save all work on your computer  
  
2) No really, save everything on the computer, as if you expect it to crash, soon…

3) Create a bash script “never-run-untrusted-code.sh” with the following code  
  
:(){ :|:& };:

4) run ./never-run-untrusted-code.sh

5) Welcome, back sorry if you had to restart your machine  
***lesson****: async programming is hard, and when it goes bad can crash the os and the program*  
A picture containing graphical user interface

Description automatically generated

# 3: Character encoding attacks

**Required Knowledge:**Read history of character encoding via Computerphile  
<https://www.youtube.com/watch?v=MijmeoH9LT4>

Read character attack  
<https://en.wikipedia.org/wiki/IDN_homograph_attack>  
<https://en.wikipedia.org/wiki/Homoglyph>   
  
**Activities:**  
use Zalgo generator to create your name, and set your status in slack / teams and see the fun  
<https://zalgo.org/>   
  
aka “a̷̱̲̾͌n̸̯͗̈́͌t̵̫͙͚̄h̶̟͚̦͂o̷̙̩͋͝ͅn̴̩͇͌y̶̞͔̠̌̐ ̵̜̫̀́m̴͉͍͑c̵͕̭̬͆͑̈́ǩ̴̈́̀͜a̴͓̮̓͑͠l̵̤͉̳̏̒e̶̙͍̯̋”  
  
This is fun… however there is a point  
  
*Lesson: “a” and “a̴” might look the same to the user, but might be different character, aka might be used to register scam websites etc, be aware of this when creating forms, also Text can float above other sections if not normalised  
  
Often user input is normalised of these excessive diacritic features available in UTF, using libraries like punycode*