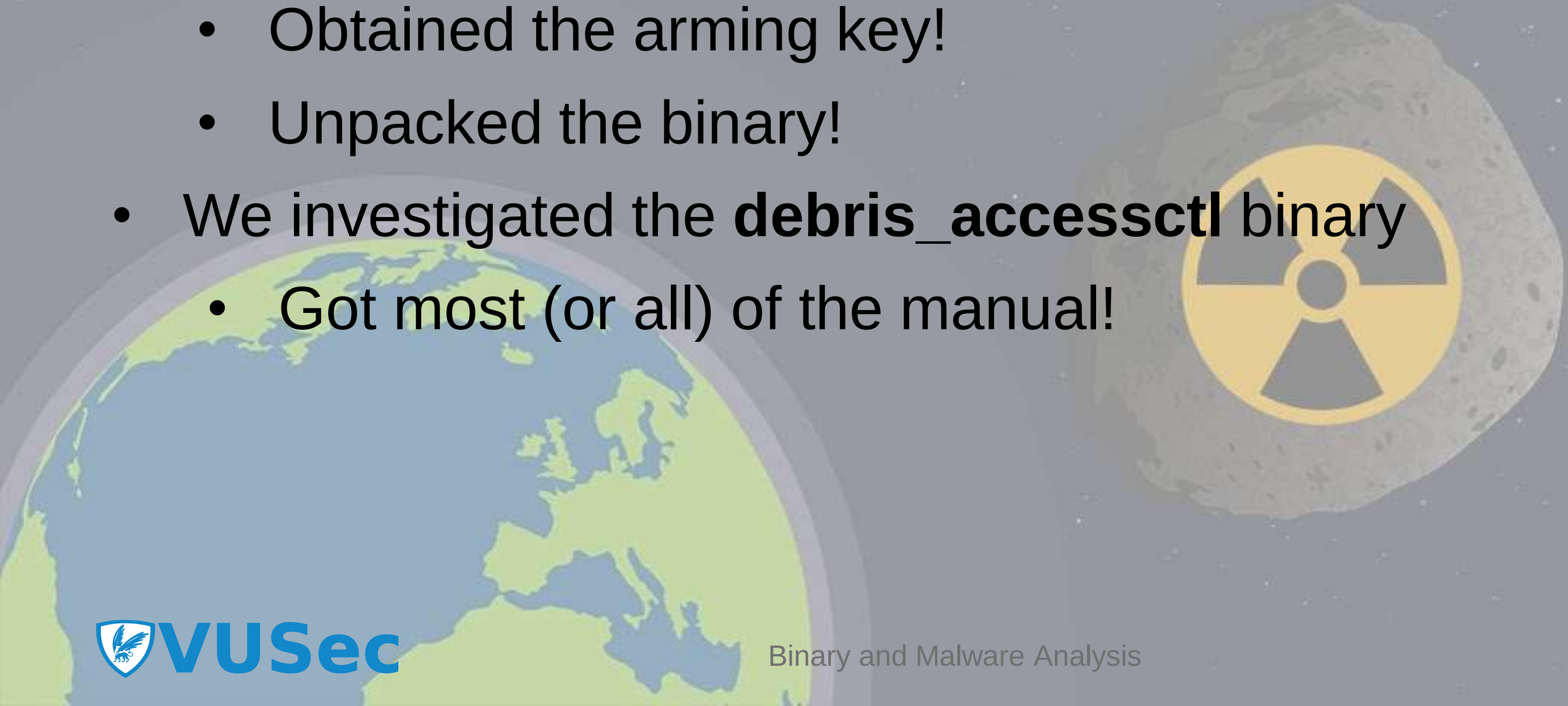
Challenge 4

Thus far

- We have dived into the **armatronic** binary
  - Obtained the arming key!
  - Unpacked the binary!
- We investigated the **debris_accessctl** binary
  - Got most (or all) of the manual!



Bad news

- Manual doesn't look too helpful
- Where are our access codes??

Next steps

- Is there only one manual?
- What is all this other stuff in the binary?
- Can we trust anything in here?

Task 1

Patch out the obfuscation in the binary:

- Call address obfuscation
    - Indirect function calls
- Return address obfuscation
    - Offsets added to return addresses

Task 1a

First approach: **dynamic** deobfuscation

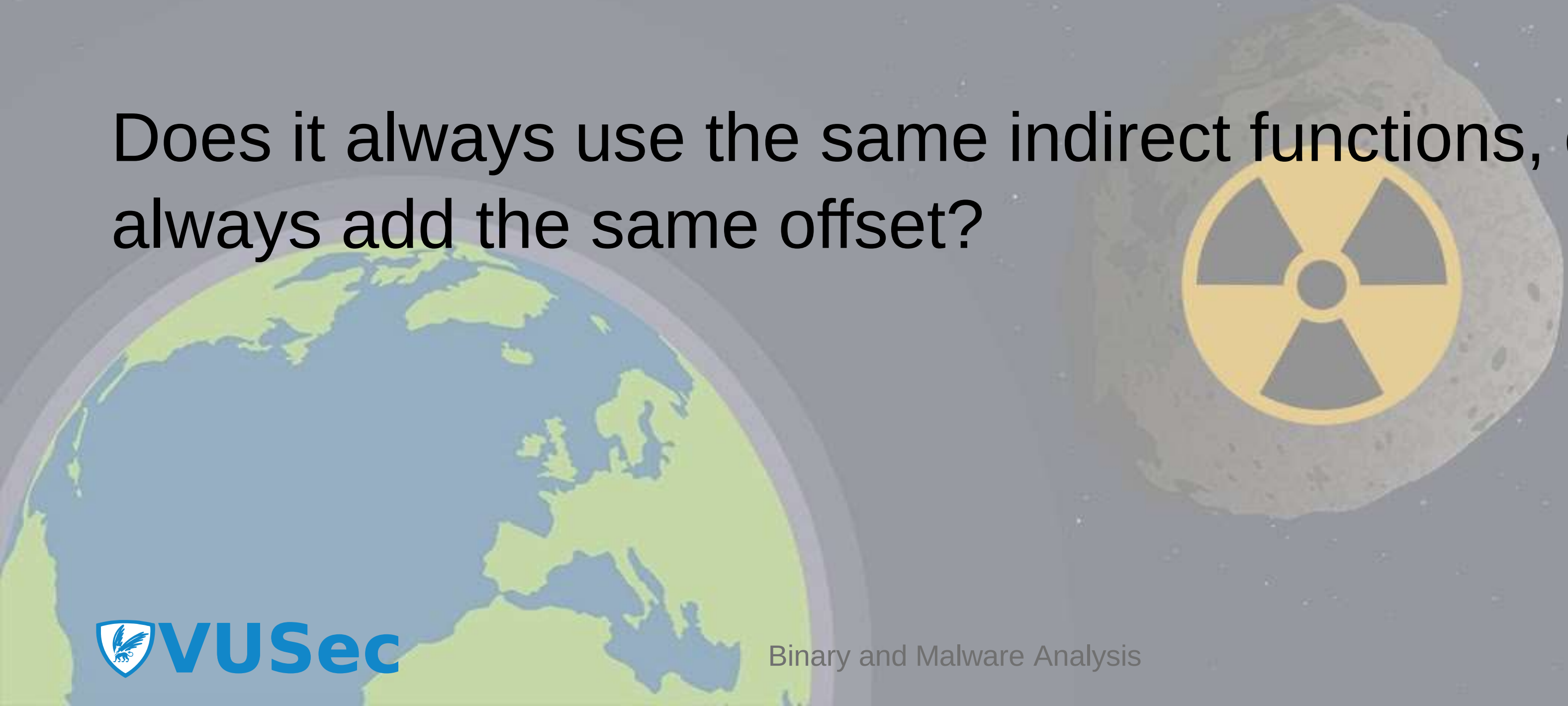
Write a PIN tool to determine addresses used at runtime, and write a script/tool to patch the binary.

- Determine which functions are called and patch the code to call them directly (new CALL?)
- Determine which return addreses are used and patch out unused/misleading code (NOP?)

But wait…

We don’t trust the binary!

Does it always use the same indirect functions, or always add the same offset?



Task 1b

Use automated **static** analysis – deobfuscate the binary without using any dynamic analysis results!

- Determine how call targets are computed, and write a script to determine and patch all static call targets
- Write a script to find any instructions modifying the return address offset, calculate the offset, and patch the callers

Task 2

There seems to be a **second manual** hidden in the binary! Your goal: Extend your taint tool from assignment 2 to recover the new manual.. (or is it a manual?)

You can enable it by using the passcode:

`debris_tracking_emergency_enable`

Task 2

Bad news: it looks like they added protection against dynamic tainting analysis!

There are a huge number of **taint-washing** functions in the binary, and the new manual is passed through many of them! Your goal: investigate them and work out what’s going on!

Task 2

- Find the array with hundreds of taint-washing functions
    - Looks like you should be able to find it with manual inspection
- Two classes of functions used for each byte:
    - Taint washing: Real manual contents?
    - Not taint washing: Fake message?

Task 2

Goal: classify the taint-washing functions, and use that classification to recover the message!

- CFG-based detection
  - Identify differences in the CFGs of the taint-washing and non-taint-washing functions
- Data flow detection
  - Implement some kind of data flow analysis, and build a more *reliable* detection method

CFG classification

You can do this in several ways:

- Ghidra scripting
- angr (lecture next week!)
- Write CFG construction yourself (w/Capstone!)

There are several ways to tell the difference based on the CFG; implement more than one!

Data flow classification

Goal: implement classification based on data flow, which might be helpful for *future* binaries as well as your own. Take a look at the data flow slides/lecture for inspiration..

You should detect whether there is data flow from the input parameter to the return value, so you should **not** depend on any CFG properties or specific instructions/ordering.

Data flow classification

Lots of ways to do this: we suggest using Ghidra scripting. Other methods (Capstone, IDA, etc) are also OK, as long as you don’t just use an existing library/framework to do **all** the work!

Should work for all the instructions present in your binary’s taint-washing functions, and a reasonable number of other (arithmetic) instructions – but doesn’t have to work on the entire binary.

Intended to be a (fun) **challenge** to apply knowledge in a less directed context!

- Doesn’t necessarily have to be perfect or even correct.
- For full points: do something smart and justify why your approach is the best!

Grading

- Deobfuscation and patching -> 4 points
  - Dynamic+static script for return addresses -> 1+1pt
  - Dynamic+static script for calls -> 1+1pt
- Classification of taint-washing functions -> 5 points
  - Script to do CFG-based classification -> 2pt (1pt basic + 1pt multiple properties)
  - Script to do data flow classification -> 3pt (2pt basic + 1pt for ‘resilient’ approach)
- Scripts/code to recover second manual -> 1 point
  - You can do this however you want; build on your chal3 code (or ask for help)
- Bonus points:
  - The usual bonus points for the first students to submit.
  - Bonus this time: build your own CFG!

VUSec

Submission Guidelines

You need to deliver a **zip file** containing:

- A plain text file ‘output’ with your newly-recovered output, if any.
- Patched binaries generated by running your scripts.
- A plain text file ‘**README**’ describing what you did and how to run your code.
- Don’t forget: submit all your code+scripts!

Submission Guidelines

- Submission will be through **Canvas**.
- **Deadline: Thursday, 14th May 2020, 23:59 CEST**
- Delay penalties: 1pt/24h delayed

Good luck!

VUSec