INFO.SEC.C00K3R

CASE STUDY 2

HUGO SERENO & JP DIAS

GOALS

(PREPARE YOURSELF)

To develop an application that supports the creation and execution of **Information Security Recipes**.

The base **unit of computation** in this application is a **Task**. Tasks are comprised of (1) a set of **inputs**, (b) a set of **outputs**, and (c) an **inherent behaviour**. One can compose tasks by **arbitrarily connecting** their inputs and outputs to form a **Recipe**. There's a **standard collection** of tasks that we can already foresee, but the application should be **extensible** enough to support new tasks **via plugins** without altering the base code. Recipes can be **saved** and re**used as tasks** in other recipes.

FUNCTIONALITIES

(WHAT IS THIS SUPPOSED TO DO?)

- **Sources.** Open File, Fetch URL, HTTP Inbound, etc…
- **Sinks.** Send to file, Post via HTTP, WebSockets, etc…
- **Handlers:**

Conversion	To/from Hex, Bin Base64, URL(De)code, CSV2JSON2YML	Text	Uppercase, Lowercase, Encoding
Arithmetic/Logic	Sum, Product, And, Or, Xor, Reduce, Average	Hashing/Encryption	MD5/SHA1/PGP
Networking	Parse URI, Parse Header, DNS Lookup, Ping, Traceroute	Compression	(G)zip, Bz2, Tar
Collections	Sort, Split, Unique, Filter, Head, Take, Reverse, Window, (Un)zip, Map	Selectors	Regex, XPath, JPath, CSS, File Path, CSV Columns
Utils	Beautify, TabsToSpaces, Colorize	Flow Control	Cycles, Functions, Errors

EXAMPLE

(CAN YOU MAKE A DRAWING, PLEASE?)

```
const in1 = FromFile(‘cenas.csv’)
const pwds = Unique(SelectColumn(in1, ‘passwords’))
const pool = Zip(pwds, MD5Rainbow(‘rainbowtable.txt’, pwds))
const usernames = SelectColumn(in1, ‘username’)
const pair = Zip(usernames, pwds)
const final = Map(pair, (u, p) ⇒ OnError(Find(pool, p), ‘?’))
const out1 = ToFile(‘passwords.txt’, final)
```

This is just a **pseudo-code example** to understand that there are things that just **receive** information, that just **send** information, and that **transform** information. They are also **configurable**, and when bundled together they establish a **Direct Acyclic Graph** (DAG).

THINGS TO THINK ABOUT

(DON'T SHOOT BEFORE ASKING QUESTIONS)

- Who **initiates** the flow? The **senders**? The **receivers**?
- Where does the **(temporary) information live** while **waiting** to be processed?
- Can we execute tasks in **parallel**? Are there **dependencies**? Is the communication **blocking**?
- Are all tasks **connectable**? Can we avoid **mistakes**?
- How do we handle **errors**? **Logging**? **Debugging**?
- Input as **lists of things** v.s. **element by element**?
- When does it **stop**?

APPROACH

(WHAT NOW?)

- Identify the main problems… identify the main patterns.
- Every person think differently. How do you convey the solution to your colleagues? Why is yours better?
- Are you prepared to scratch your whole solution every time you start tackling a new problem?
- How will you retain this knowledge so everyone will remember it later?

Therefore, come up with a minimalistic way to reason about your system without actually implementing it, that allows quick feedback from everyone, that evidences potential problems, and that allows you to retain its essence for the coming weeks.

ASSO