V-Research

Research & Development for Cybersecurity Engineering

Secure Software Development Lifecycle

Francesco Beltramini, Marco Rocchetto
francesco@v-research.it marco@v-research.it

Dissemination level: Public
Confidentiality level: unencrypted
ECCN: NSR

https://v-research.it

Marco Rocchetto
Co-founder of V-Research
[Research & Development]

Francesco Beltramini
Co-founder of V-Research
[Business Development]

UNI VR.

2009

Doctor Europaeus

Security Administrator

2015

Academic Researcher

Senior Security Engineer

2017

Senior Research Engineer

Head of Security Engineering

V-Research

A **private R&D center** that bridges *foundational challenges* and *engineering needs*.

Agenda

Introduction

- Background
- Software (and System) Development Life Cycle models
- From V-Model to DevOps to DevSecOps
- CI/CD for DevOps and DevSecOps

Techniques and Tools

- Secure Code:
  - Secrets management
  - Access Control
  - Cryptographic Practices

Background

IT / COMMERCIAL SPACE

- Software = customers' first impression of a Company products and services
- Faster release cycles
- Faster features implementation
- New methodologies introduced: Lean, Scrum, DevOps, ...

OPERATIONAL TECHNOLOGY SPACE

- Physical systems changed into Cyber-physical Systems
- Software is now deeply intertwined with physical reality
- Major implications for safety / critical infrastructure / operational efficiency
- Longer cycle Waterfall and V-Model

Waterfall / V-Model

Requirement Gathering: system requirements are captured in this phase and documented in a requirement specification document.

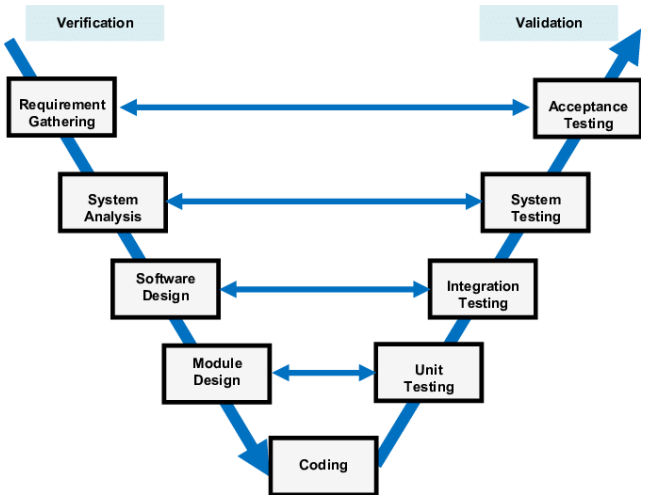
System Design: requirements are studied and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

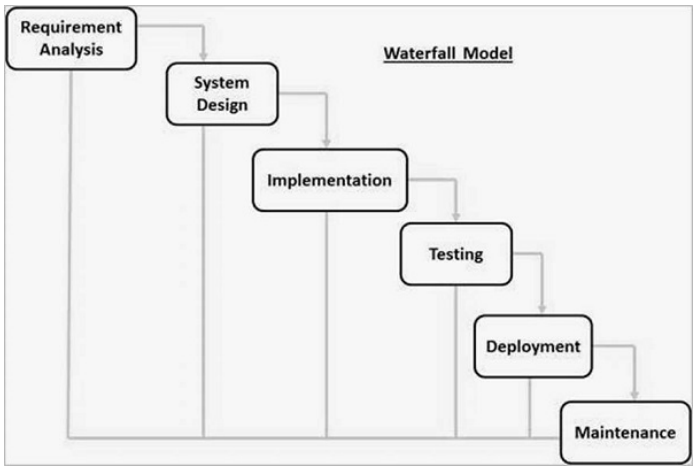
Implementation: the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

Integration and Testing: all the units are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system: once the functional and non-functional testing (e.g. sufficient network bandwidth) is done; the product is deployed in the customer environment or released into the market.

Maintenance: there are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.





Lean

- **Waste Elimination**: anything that doesn't bring value for the product should be eradicated. Producing ahead, Errors, Overusing advanced tools, No unfinished tasks.
- **Fast Delivery**: gives speed to project development through collaborating team members and solving complex issues once they occur, people work independently without being told what to do.
- **Respectful teamwork**: product managers must ensure a consolidated team that communicates appropriately, achieves mutual agreements easily, and can decide how to resolve issues.
- **Optimization**: all project details, wishes, and requirements must be discussed before the development begins. The point of optimization is to dedicate all effort towards features and functionalities to add to the platform.
- **Quality**: prevent waste, yet not sacrificing quality, constant feedback from team members and the project manager.
- **Generate and Implement Knowledge**: conduct constant self-education since technologies do not stay still. Building a vast knowledge base brings new features to the application and increases the speed of development.
- **Delayed Commitment**: fully comprehending business needs and requirements before doing any actions

Scrum

- **Design**: list of features
  - The backlog
  - What still needs to be completed
  - How long it will take
- **Sprint**: hands-on
  - Sprints are periods of time when software development is actually done
  - A sprint usually lasts from one week to one month to complete an item from the backlog
  - The goal of each sprint is to create a saleable product followed by review
  - Then the team chooses another piece of backlog to develop — which starts a new sprint
  - Sprints continue until the project deadline or the project budget is spent
- **Daily scrums**: teams meet to discuss their progress since the previous meeting and make plans for that day
  - The meetings should be brief — no longer than 15 minutes
  - Each team member needs to be present and prepared
  - The ScrumMaster keeps the team focused on the goal

DevOps (V-Model+)

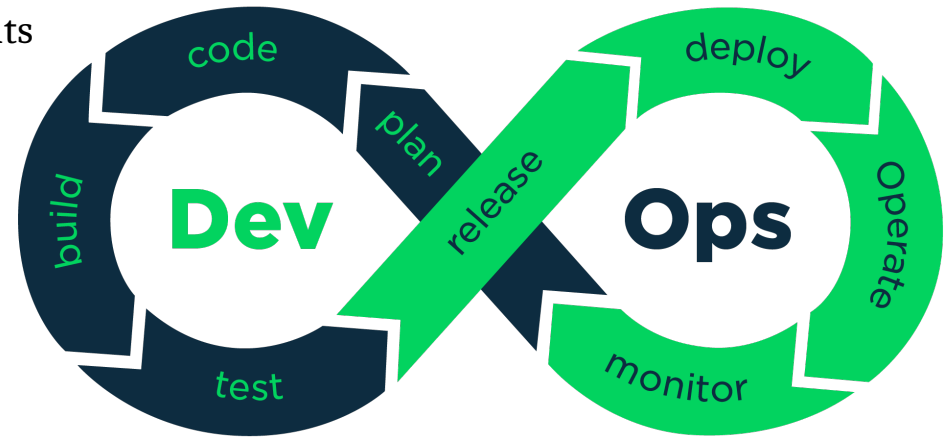
Enables formerly siloed roles—development, IT operations, quality engineering to coordinate and collaborate to produce better, more reliable products.

PLAN: ideate, define, and describe features and capabilities of the applications and systems they are building.

DEVELOP: coding, writing, testing, reviewing, and the integration of code by team members—as well as building that code into build artifacts that can be deployed into various environments

DELIVER: deploying applications into production environments in a consistent and reliable way. Deploying and configuring the fully governed foundational infrastructure that makes up those environments.

OPERATE: maintaining, monitoring, and troubleshooting applications in production environments.



CI / CD practice and pipeline forms the backbone of modern day DevOps operations.

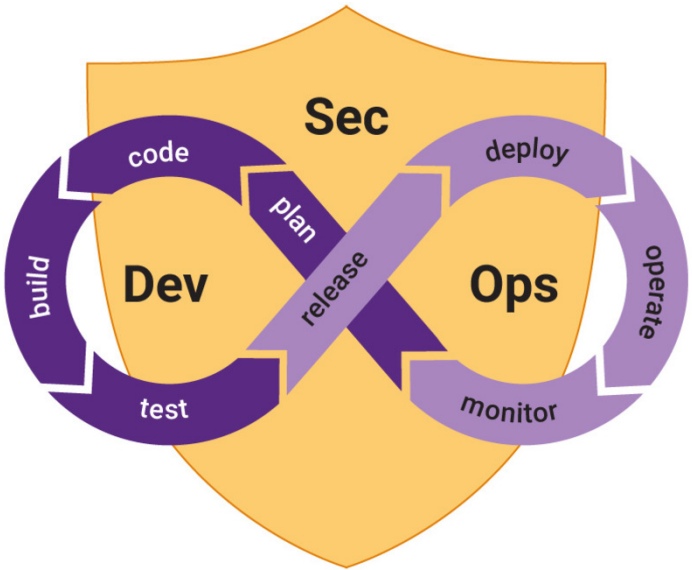
DevSecOps (V-Model++)

Automatically bakes in security at every phase of the software development lifecycle, enabling development of secure software at the speed of DevOps. Everybody is responsible for it!

Throughout the development cycle, the code is reviewed, audited, scanned, and tested for security issues. These issues are addressed as soon as they are identified. Security problems are fixed before additional dependencies are introduced.

Quickly manages newly identified security vulnerabilities:

- **integrating** automated software testing
- **integrating** vulnerability scanning and patching into the release cycle
- **identifying** and **patching** vulnerabilities and exposures
- **limiting** the window a threat actor has to take advantage of vulnerabilities in production systems.



Shift Left: move security from the right (end) to the left (beginning) of the DevOps (delivery) process. Security issues become less expensive to fix.

CI / CD secure practice and pipeline forms the backbone of modern day DevSecOps operations.

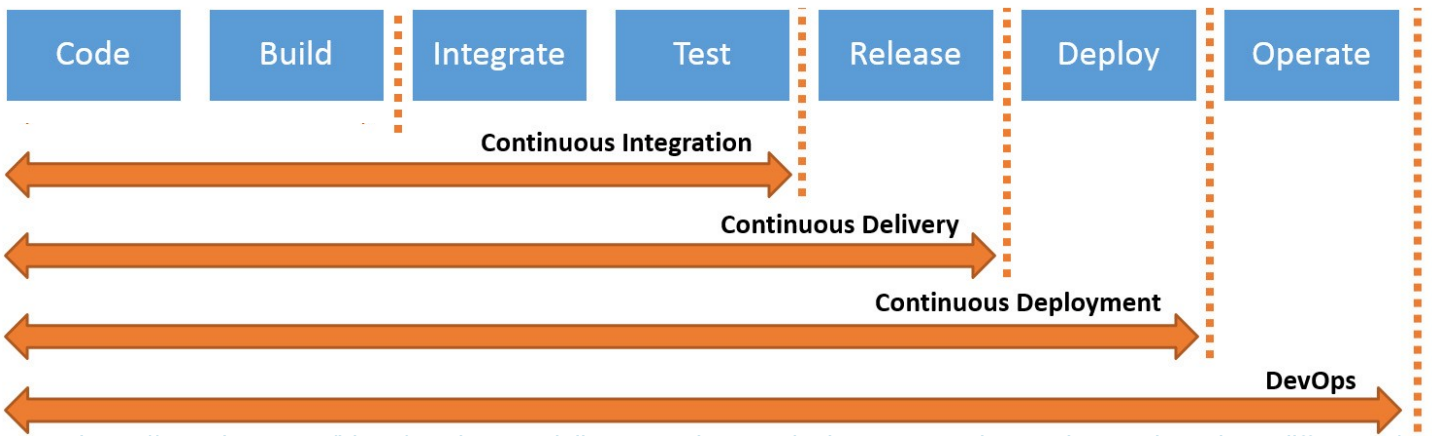
CI / CD in Dev(Sec)Ops

A method to frequently deliver applications (or changes) by introducing **automation** into the stages of the development. Main concepts: **continuous integration, continuous delivery** (or **continuous deployment)**.

Continuous integration: establish a consistent and automated way to build, package, and test applications.

- Maintain a code repository
- Automate the build
- Automate the testing
- Frequent commits to baseline (to detect problems)
- Build after every commit

Continuous delivery: every code change is built, tested, and then pushed to a non-production testing or staging environment. Manual approval for deployment.

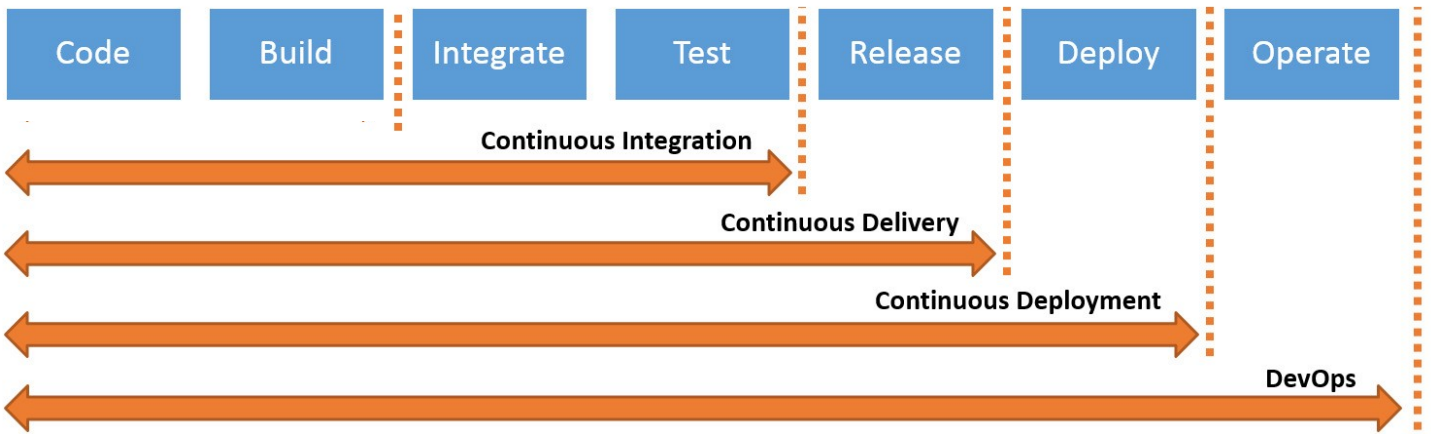


https://www.bmc.com/blogs/continuous-delivery-continuous-deployment-continuous-integration-whats-difference/

CI / CD in Dev(Sec)Ops

Continuous deployment: Production deployment happens automatically without explicit approval

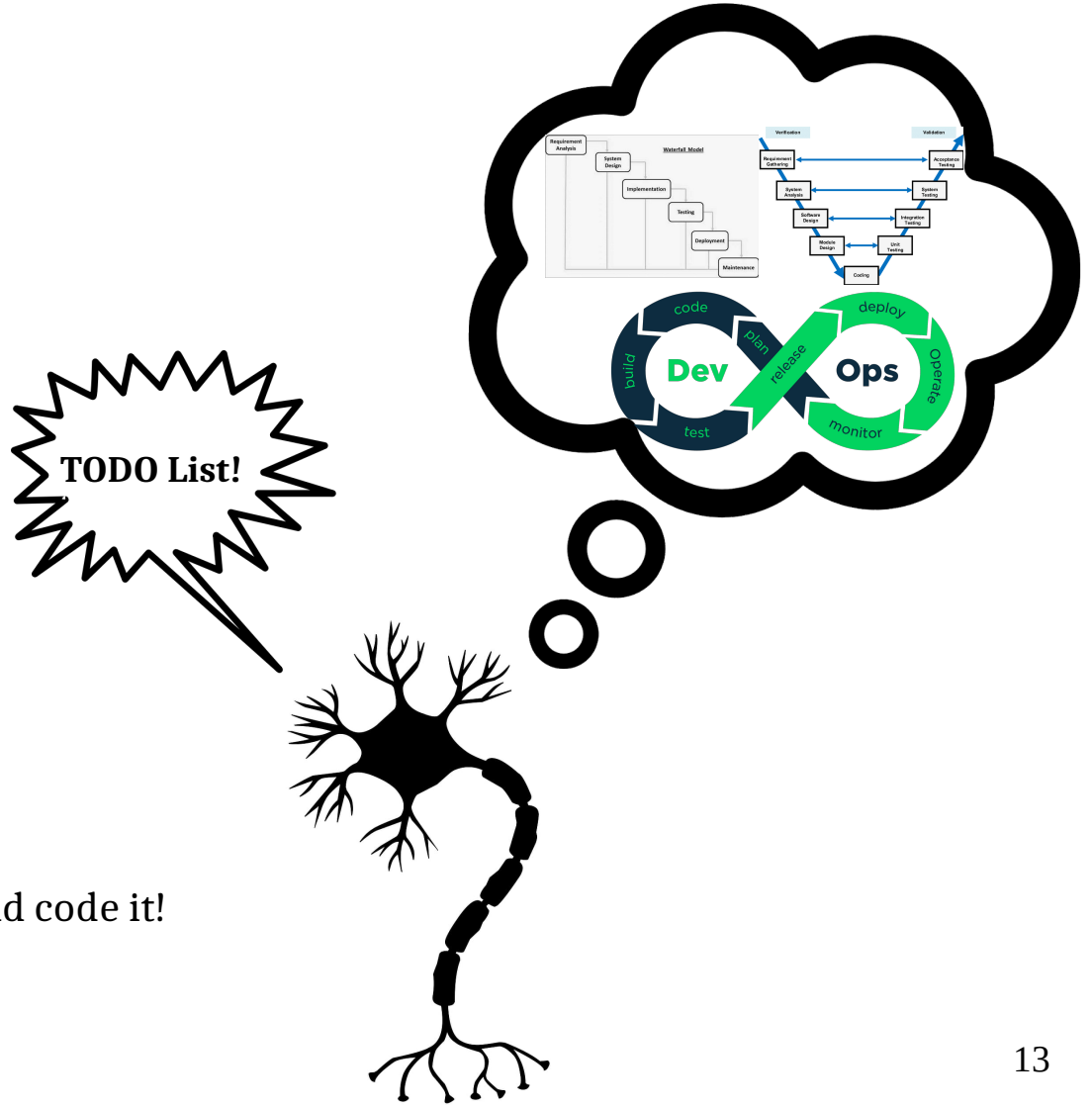
- User acceptance tests (UATs)are automated.
- Heavily relies on well-designed test automation
- Containerization, to ensure same behavior across systems
- Automated roll-backs based on telemetry



https://www.bmc.com/blogs/continuous-delivery-continuous-deployment-continuous-integration-whats-difference/

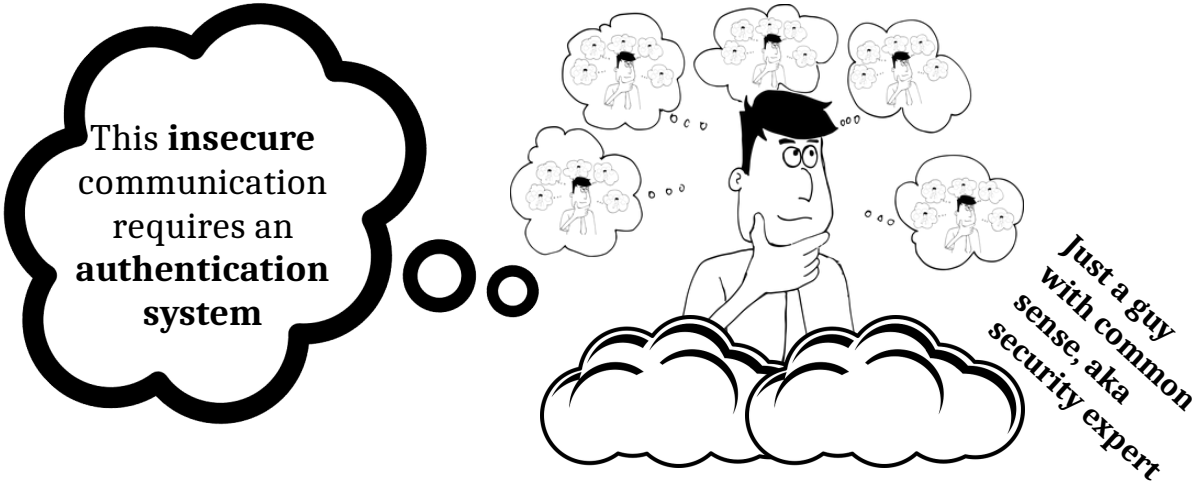
How do we Use These Models

One-Neuron Engineer



1) Requirements: What do you want?
2) Design: Depict it!
3) Implementation: Stop wasting your time and code it!
4) Test: What have you done?

Two-Neurons Engineer



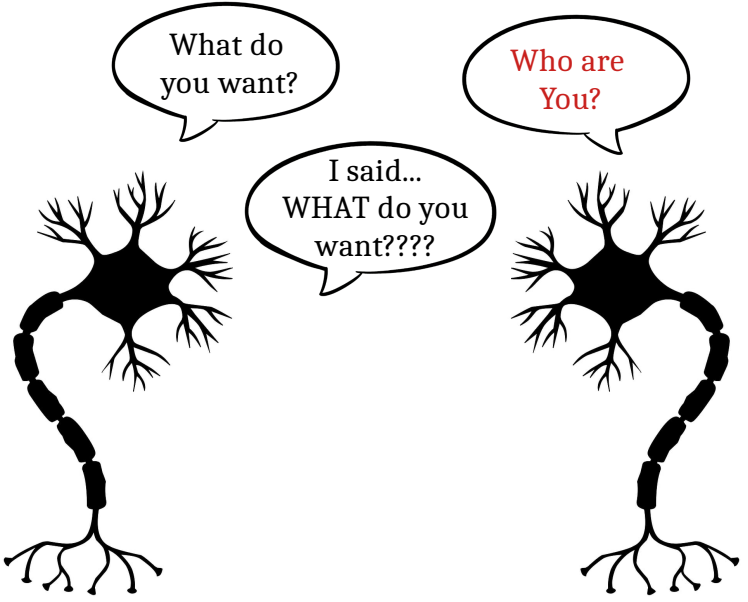
0) Problem Statement

1) Requirements: What do you want?

2) Design: Depict it!

3) Implementatition: Stop wasting your time and code it!

4) Test: What have you done?



V-Reseʌrch

Authentication

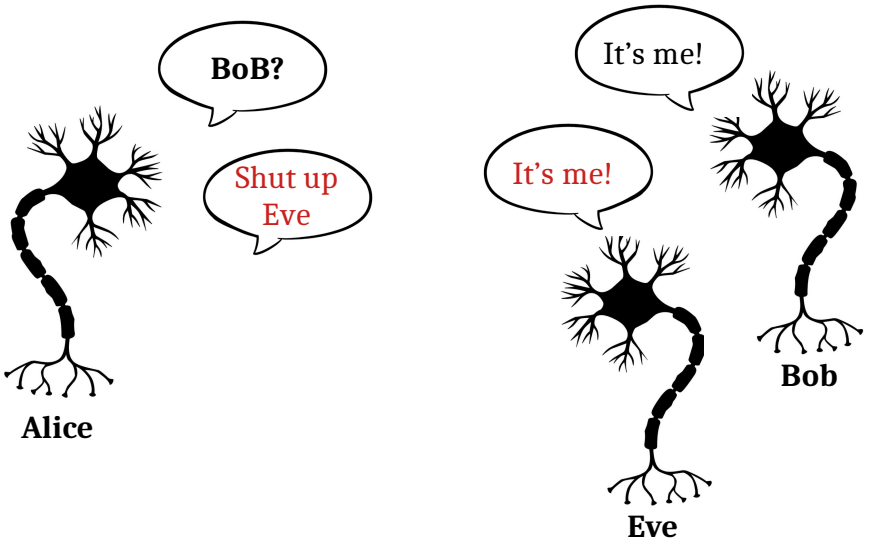
Holy scroll of authentication

Requirements

General Objective

We want to be sure that we are talking with the intended recipient

Three-Neurons Use Case



Holy scroll of authentication

Requirements

General Objective

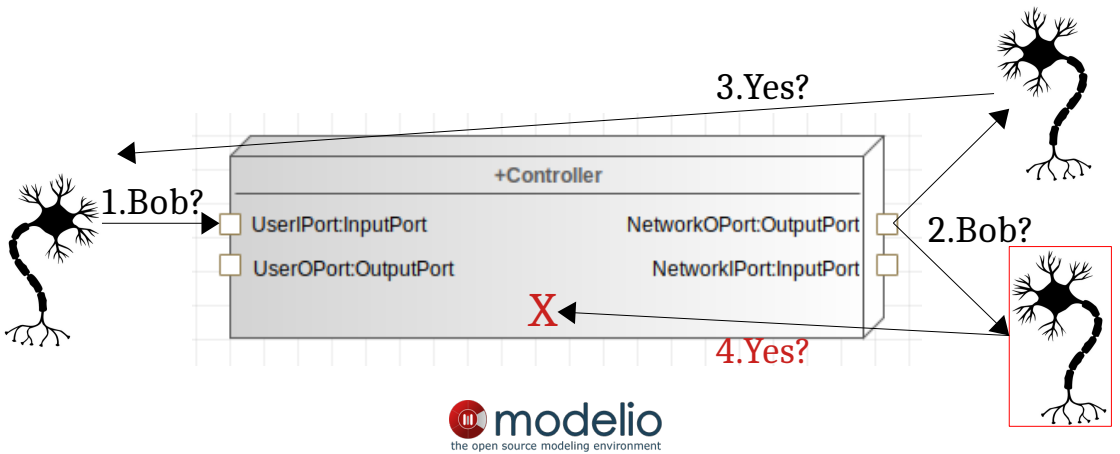
We want to be sure that we are talking with the intended recipient

We want to know when we're not talking with the intended recipient

Use Case (UML Diagram)

Two neurons talks to each other about non-confidential stuff and a third is messing up

Three-Neurons Use Case



UML Deployment Diagram

Holy scroll of authentication Requirements

General Objective

We want to be sure that we are talking with the intended recipient

We want to know when we're not talking with the intended recipient

Use Case (UML Diagram)

Two neurons talks to each other about non-confidential stuff and a third is messing up

(non-)Functional requirements (architecture)

We want a physical device that we can use to authenticate two neurons with each other

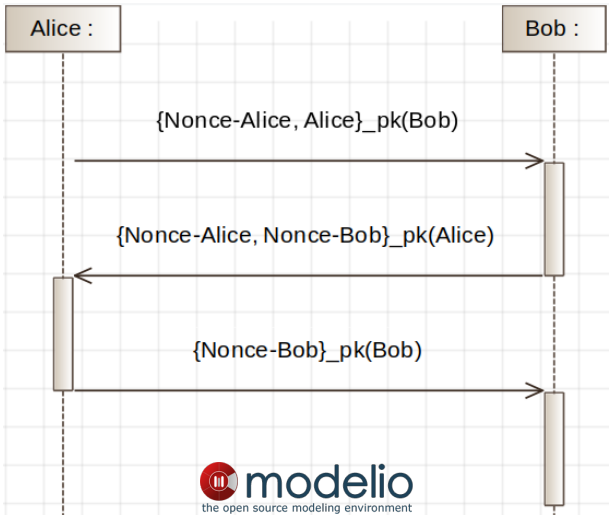
V-Research

Two-Neurons Behavior

R1) Alice has Bob's public key

R2) Secure crypto

...



UML Sequence Diagram

Holy scroll of authentication

Requirements

General Objective

We want to be sure that we are talking with the intended recipient

We want to know when we're not talking with the intended recipient

Use Case (UML Diagram)

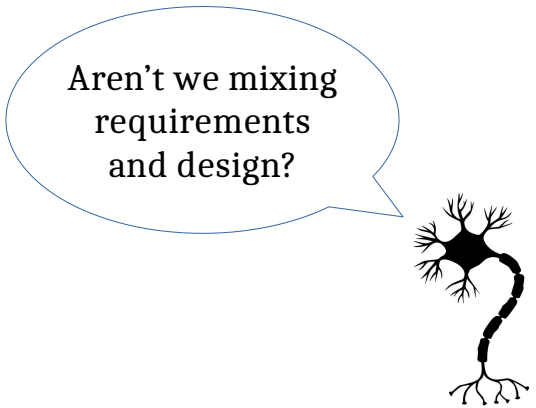
Two neurons talks to each other about non-confidential stuff and a third is messing up

(non-)Functional requirements (architecture)

We want a physical device *that we can use to authenticate two neurons with each other,*

& a functional architecture where ...

Requirements Specification



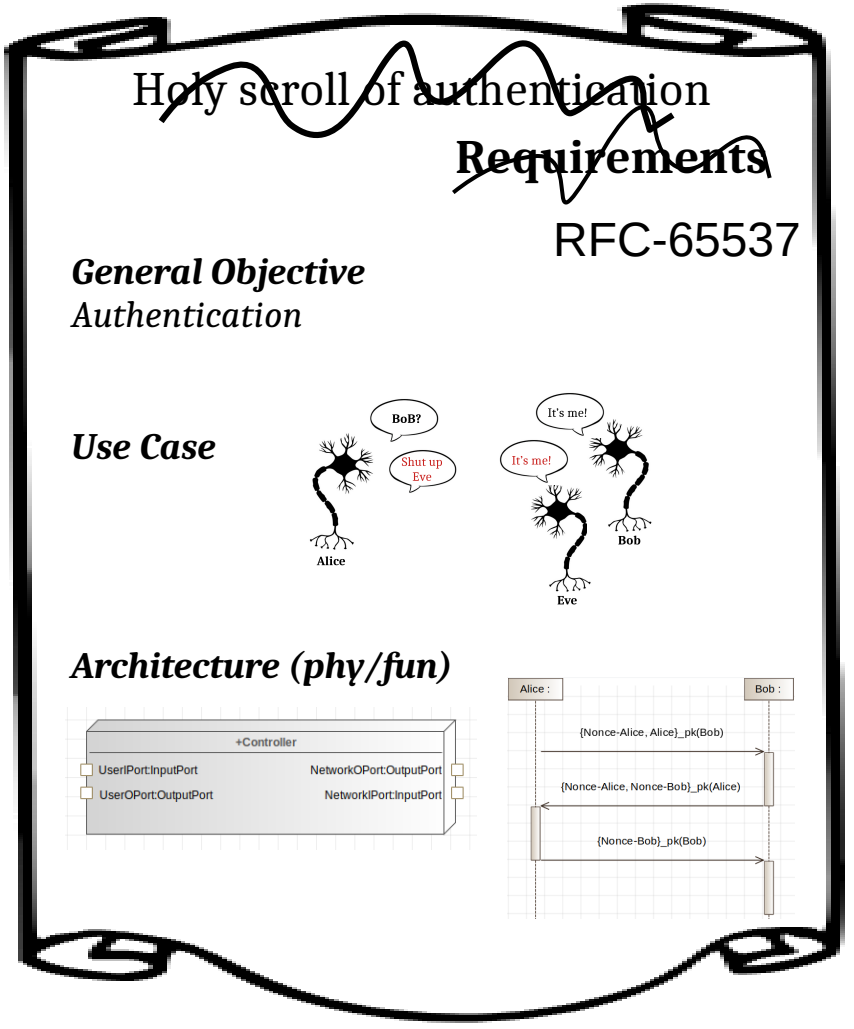
~~0) Problem Statement~~

~~1) Requirements: What do you want?~~

2) **Design**: Depict it!

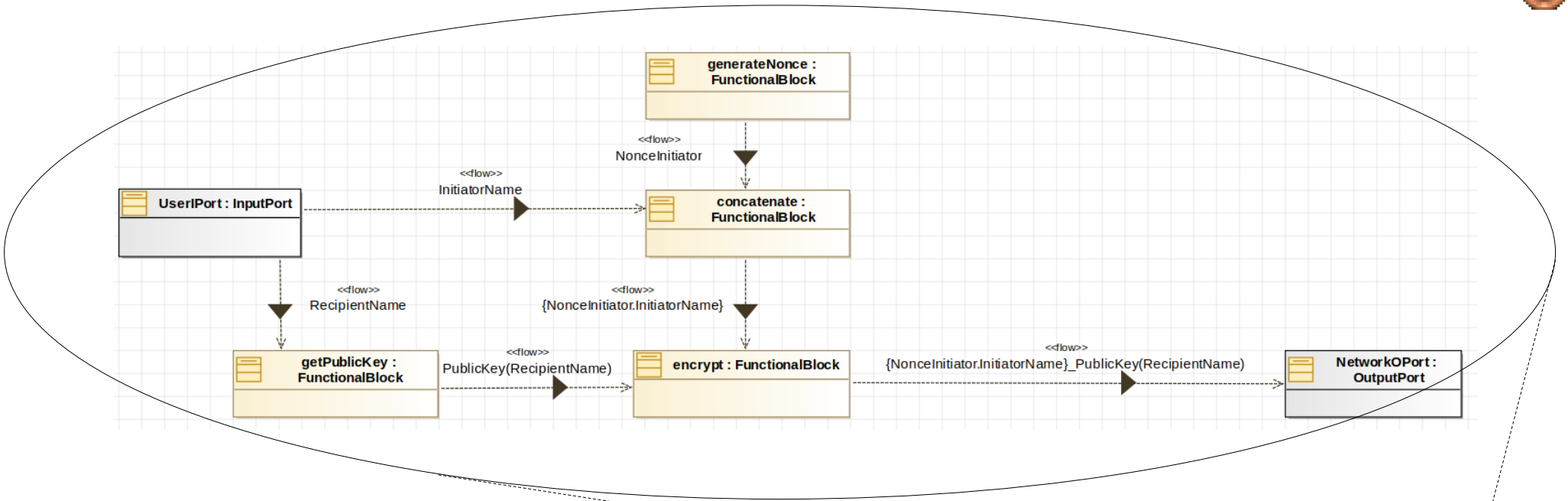
3) Implementation: Stop wasting your time and code it!

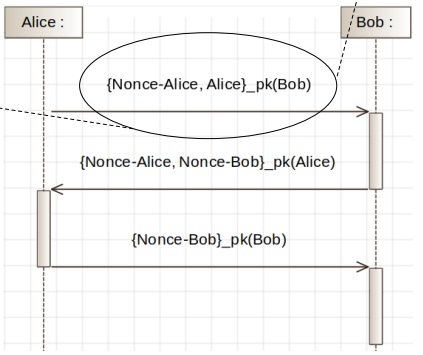
4) Test: What have you done?



V-Reseʌrch

Full-neurons - Detailed Design (Object Diagram)





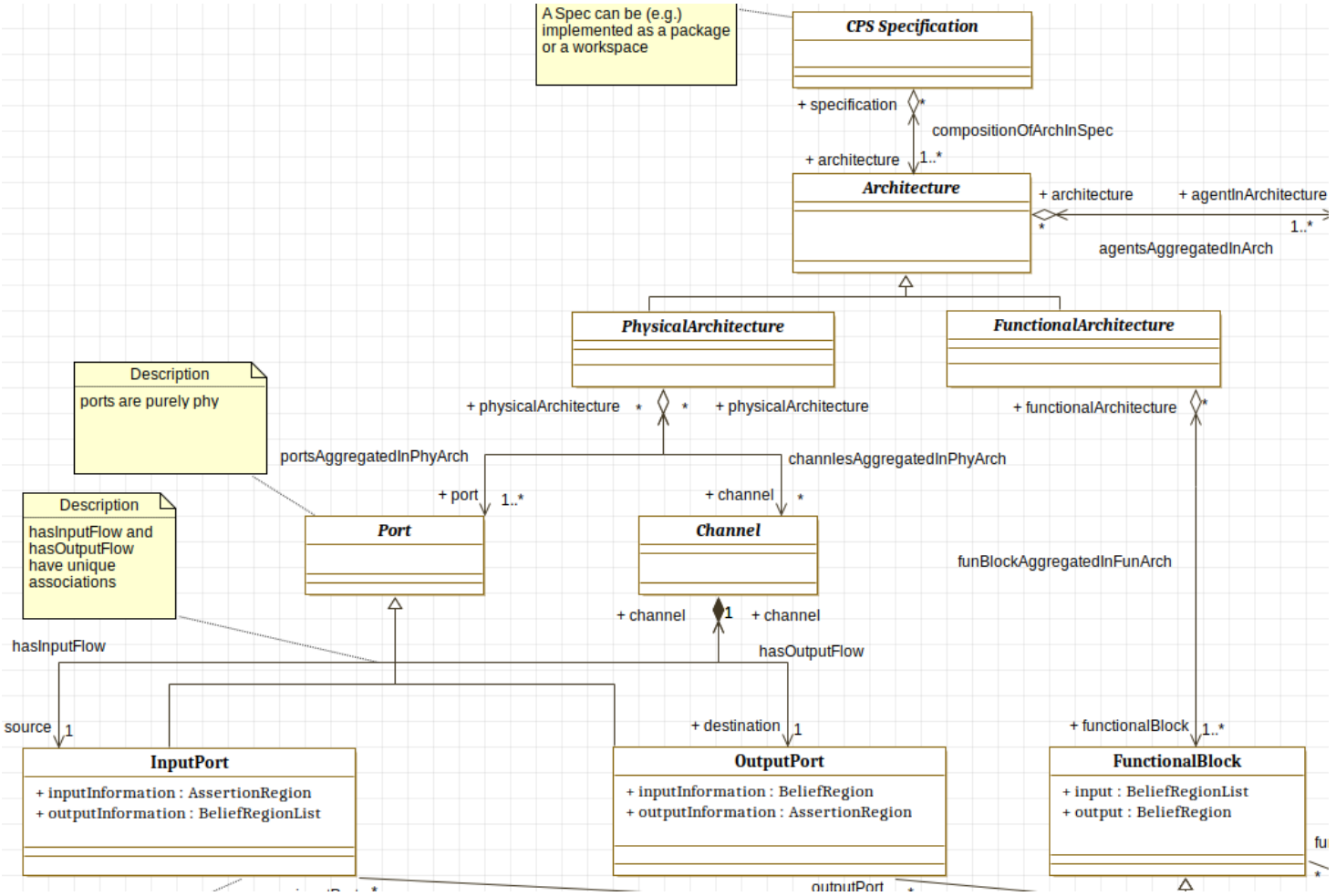
UML is better!

SysML is better!

Pfff....

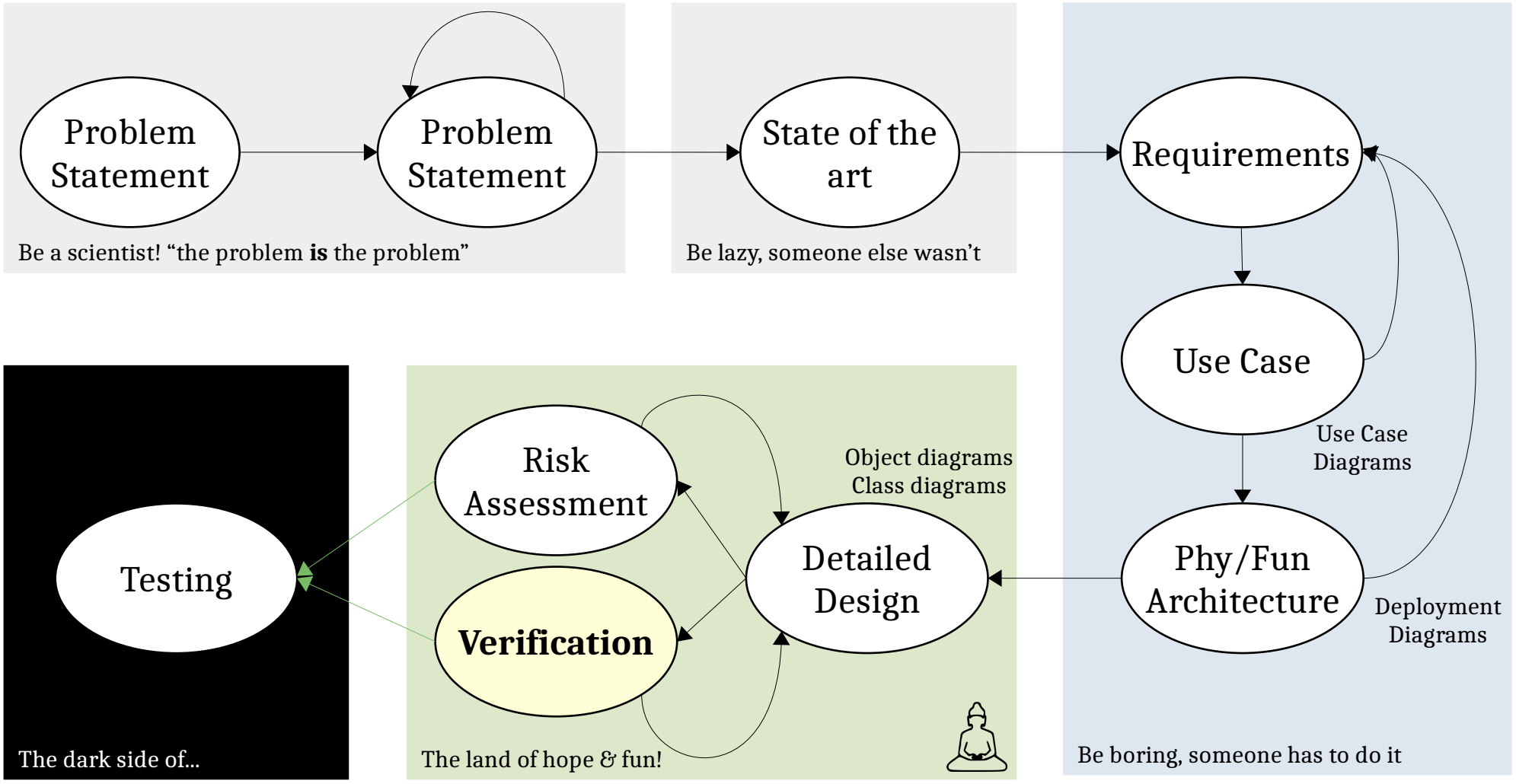
V-Reseʌrch

Full-neurons - Detailed Design (Class Diagram)



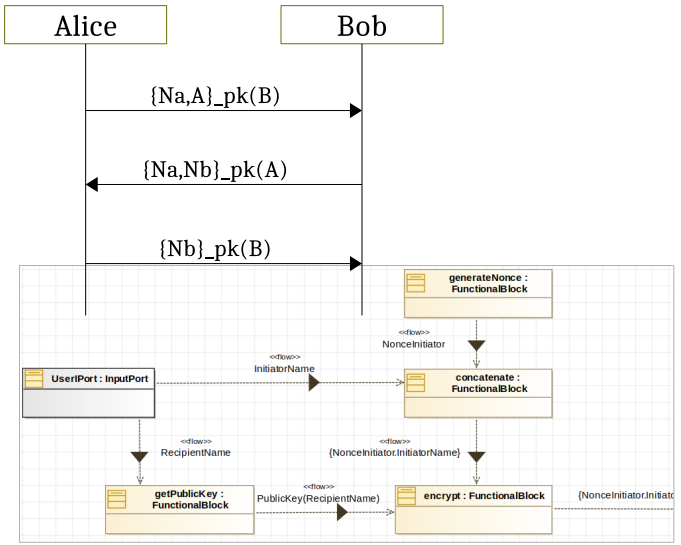
V-Reseʌrch

Keep Calm & Summarize



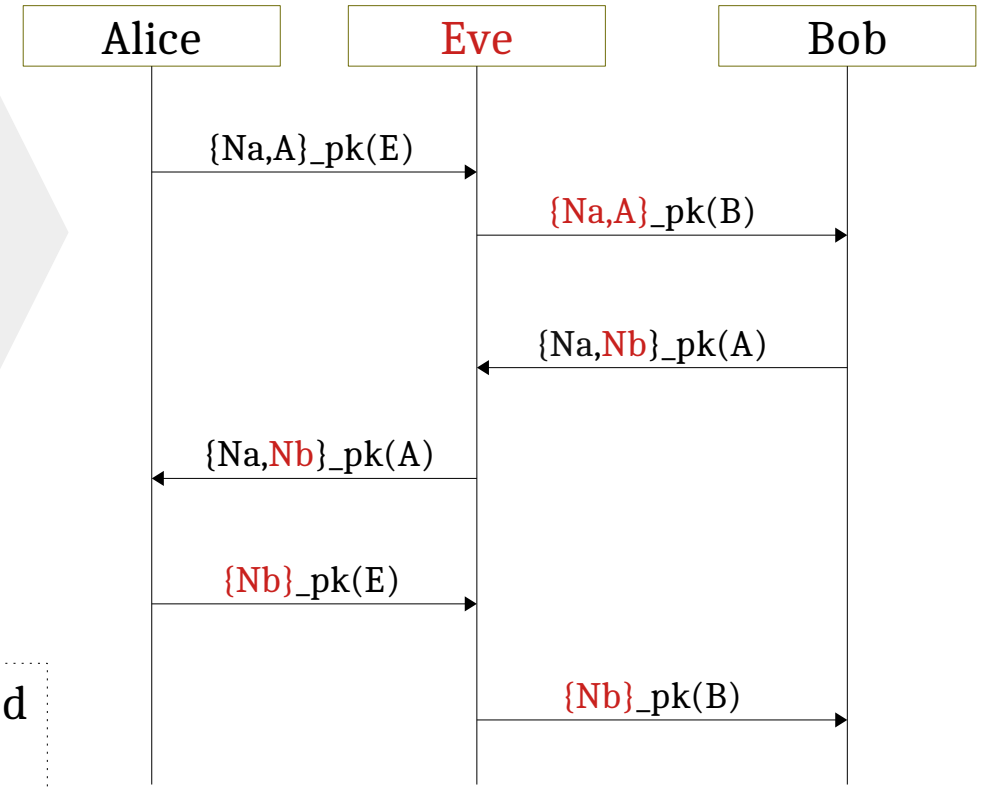
Cybersecurity Verification (Protocol Logic)

Protocol Model → DY Attacker Model → Very Interesting Attacks



Formal Security Verification

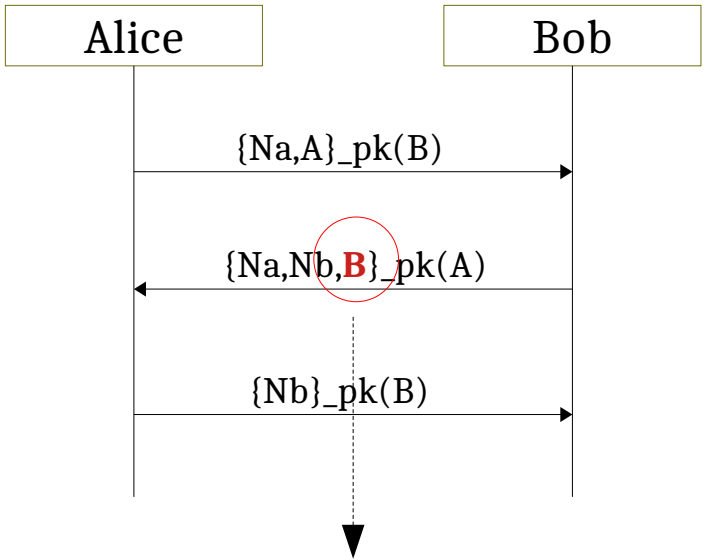
Is **Authentication** valid/preserved by our design?



Man-in-the-middle Attack

V-Reseʌrch

Security – When?



Is it *secure* now?
Nobody knows

We can prove it secure
w.r.t. a definition of security
(DY attacker model)

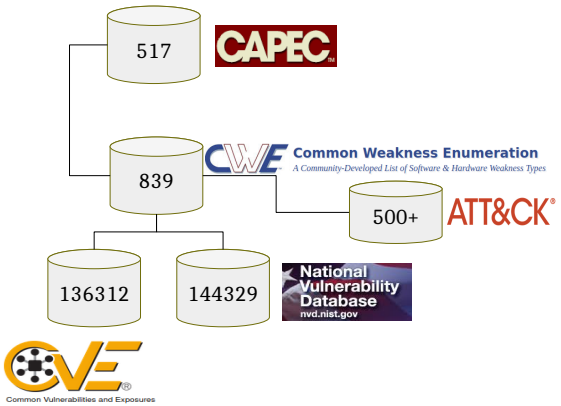
Wat

A lightning talk by Gary Bernhardt from CodeMash 2012

```
failbowl:~(master!?) $ jsc
> [] + []

> [] + {}
[object Object]
> {} + []
0
> {} + {}
NaN
>
```

We code

Static & Dynamic
Code Analysis



More insecurity
We test

Techniques and Tools

Agenda

Introduction

- ~~Background~~
- ~~Software (and System) Development Life Cycle models~~
- ~~From V-Model to DevOps to DevSecOps~~
- ~~CI/CD for DevOps and DevSecOps~~

Techniques and Tools

- Secure Code:
  - Secrets management
  - Access Control
  - Cryptographic Practices

CODE

Set of technologies and best practices for making software as secure and stable as possible:

- **Input Validation**
- **Output Encoding**
- **Authentication and Password Management (includes secure handling of credentials)**
- **Session Management**
- **Access Control**
- **Cryptographic Practices**
- **Error Handling and Logging**
- **Data Protection**
- **Communication Security**
- **System Configuration**
- **Database Security**
- **File Management**
- **Memory Management**
- **General Coding Practices**

Secure Code – Secrets Management

Tools and methods for managing digital authentication credentials (secrets), including passwords, API keys and tokens for use in applications, services, privileged accounts and other sensitive parts

Risks:

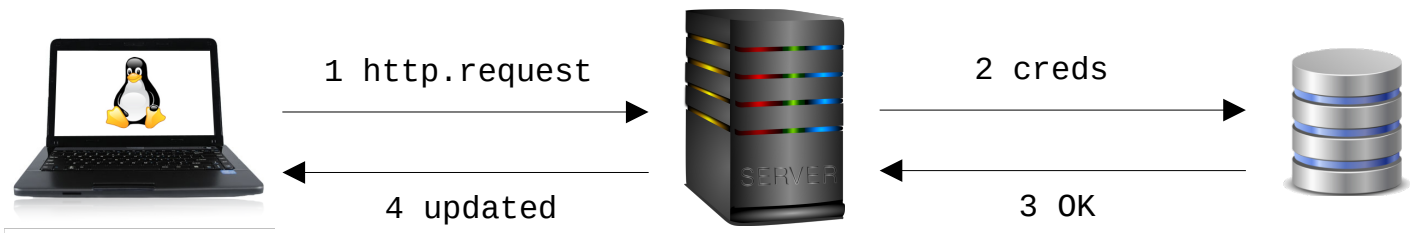
- Incomplete visibility and awareness (involuntary backdoors)
- Hardcoded/embedded credentials (impossible to rotate creds)
- Manual secrets management processes (involuntary backdoors)

Best Practices:

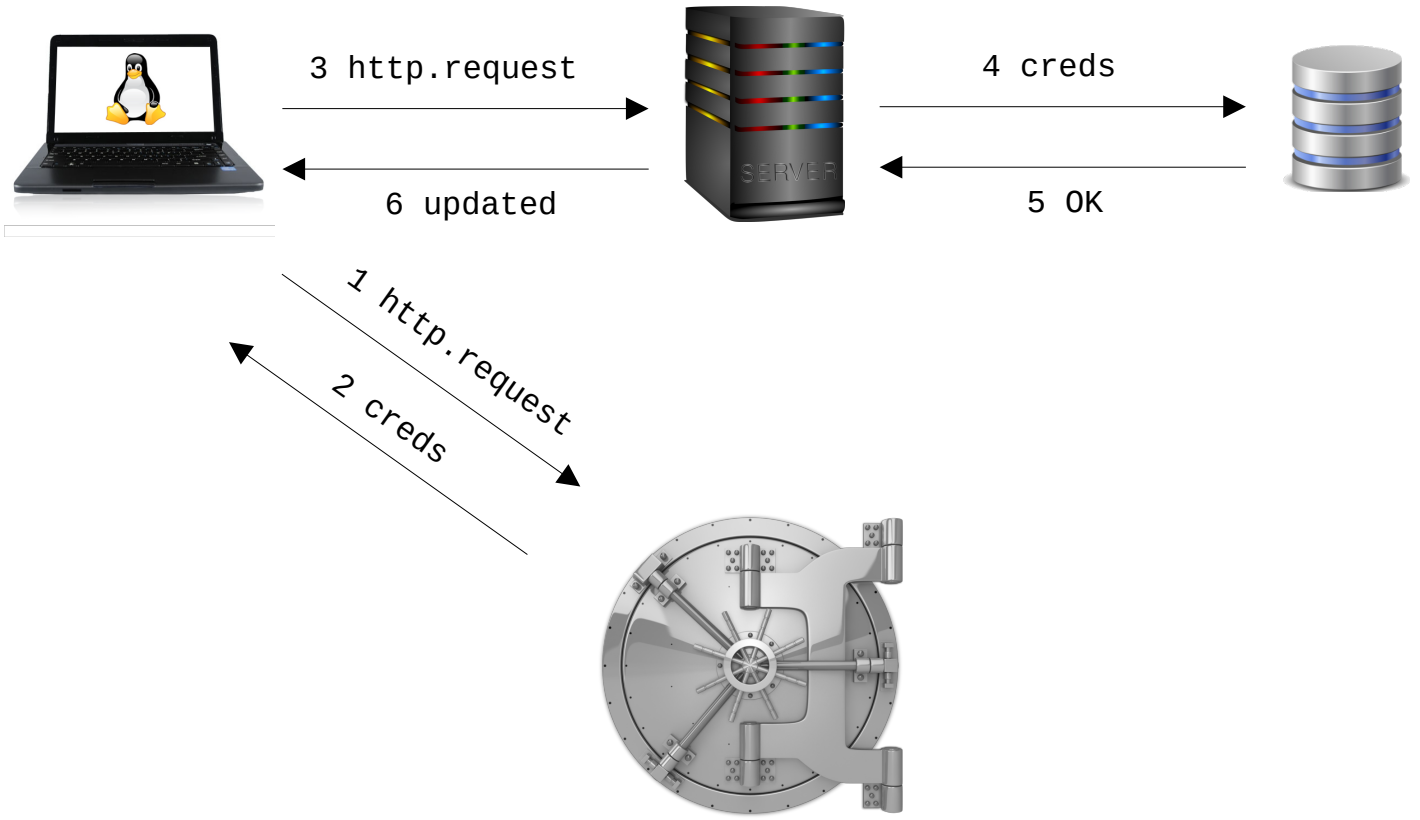
- Eliminate hardcoded/embedded secrets
- Enforce password security best practices
- Centralize secrets management
- Configure access policies to secrets
- Secure the secrets repository

Secure Code – (no) Secrets Management

```
# Simple code snippet to change home address record
class univr:

  def __init__(self):
    # Initialize module
    self.http = urllib3.PoolManager(cert_reqs='NONE')
    self.base_url = 'https://api.univr.it'

  def authentication(self):
    # Student credentials
    username = "db-user"
    password = "hastalavistababy"
    return hmac.new(username.encode(), ',' password.encode())

  def change_home_address(self, address):
    header = self.authentication(self)
    # Use header to authenticate and change address
    self.http.request(method="POST", url=self.base_url, headers=header, data=address)
```



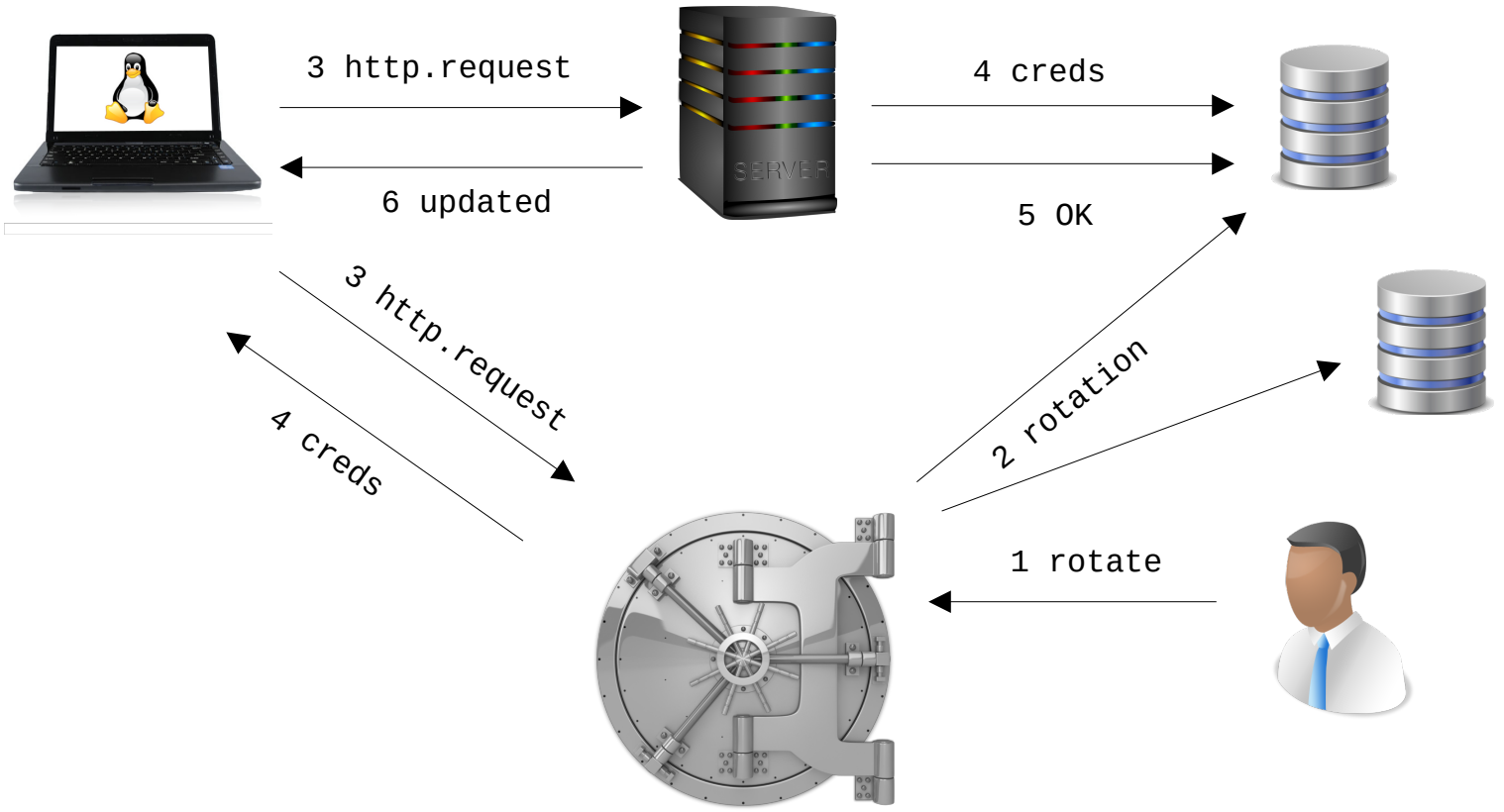
Secure Code – Secrets Management



Secure Code – Secrets Management

```
# Simple code snippet to change home address record
class univr:

  def __init__(self):
    # Initialize module
    self.http = urllib3.PoolManager(cert_reqs='NONE')
    self.base_url = 'https://api.univr.it'
    self.vault_url = 'https://vault.univr.it'
    self.vault_token = 'n0mor3secrets'

  def authentication(self):
    # Student credentials
    creds = json.load(self.http.request(method='GET', url=self.vault_url, headers=token).data)
    username = creds['username']
    password = creds['password']
    return hmac.new(username.encode(), ',' password.encode())

  def change_home_address(self, address):
    header = self.authentication(self)
    # Use header to authenticate and change address
    self.http.request(method="POST", url=self.base_url, headers=header, data=address)
```

Secure Code – Secrets Management (credentials compromise)



Secure Code – Access Control

Zero Trust Architecture

Alternative security model that addresses the fundamental flaws of traditional strategies
1) Assume compromise rather than assuming that data only needs to be protected from outside
2) Do-not-Trust-And-Verify approach rather than Trust-but-Verify approach
3) No perimeter is a secure perimeter, internal users and applications are NOT trusted

Principles
1) Every human connection is authenticated, authorized and audited
2) Every data being transferred is authenticated, authorized and audited
3) Every application flow is authenticated, authorized and audited

Zero Trust Applications should always be designed to check: WHO / WHAT / WHEN / WHERE / WHY / HOW

Secure Code – Access Control

WHO: check the identity of the clients (weak: IP filtering, strong: x509 certs)

WHAT: what is the client trying to access, is it allowed?

WHEN: are there restrictions around the time of day for the connection? Is it an unusual time?

WHERE: where the client is connecting from, where is it connecting to?

WHY: is this client supposed to access this data? What is the reason?

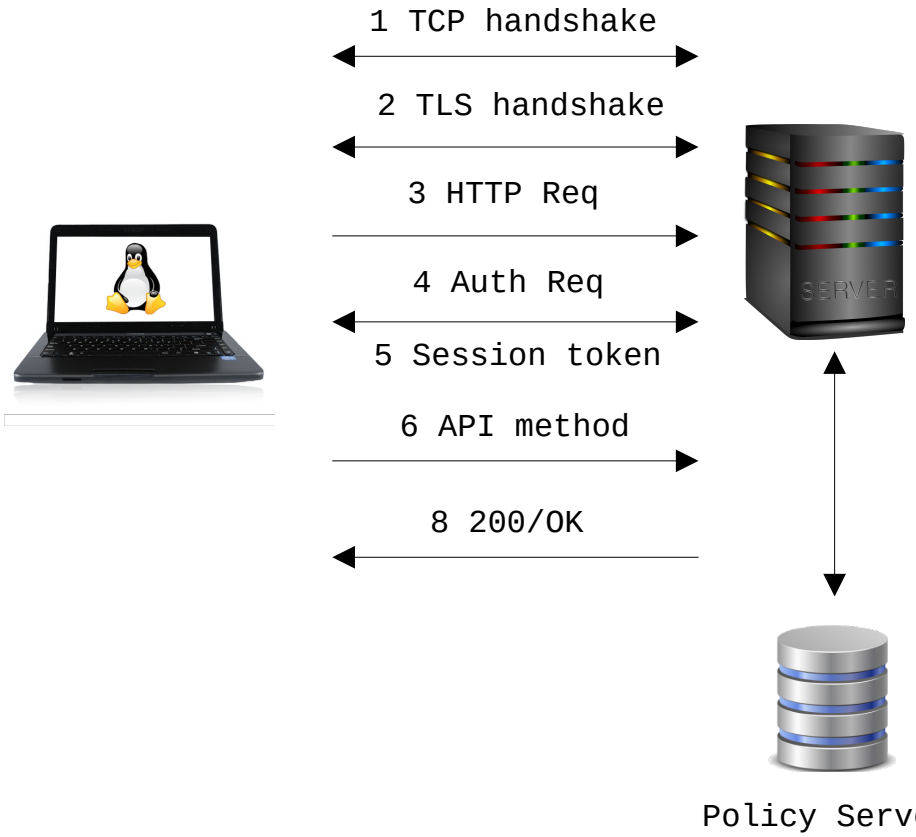
HOW: what methods is the client trying to use to access the data?

… for every client request.

Usually applications are not re-written, but integrated with Application Layer Gateways and Policy Servers.

Secure Code – Access Control

Zero Trust Architecture



L3 / L4 Controls

- Coming from the allowed IP range?
- Right ports / protocols?

L5 / L6 Controls

- Is the x509 client certificate valid?
- Cipher suites OK?

L7 Controls

- Is the Request for a protected resource?
- Is the client authorized to access it?
- Is the method OK to access the resource?
- Are the parameters OK for the client?
- Is the right timeframe?

Anomalies / changes trigger re-auth or additional client challenges.

Secure Code – Cryptographic Practices

Secure your Application Crypto-Ops!

High-assurance software requires secure handling of cryptographic material and running cryptographic operations in a secure enclave.

Hardware Security Modules or **HSM** provide that level of protection for both keys and operations.

- Different levels of assurance (from virtual HSM to physically secured appliances, see FIPS 140-2 L3 and above)
- Separate the application from the crypto-material handling (i.e. developers vs crypto-operators)
- Truly random number generators
- Quality algorithms to generate keys

Challenges:

- VERY expensive
- Admin overhead (especially in lockdown)
- Don't always scale well

Secure Code – Cryptographic Practices

Classic Web Services

2 TCP handshake

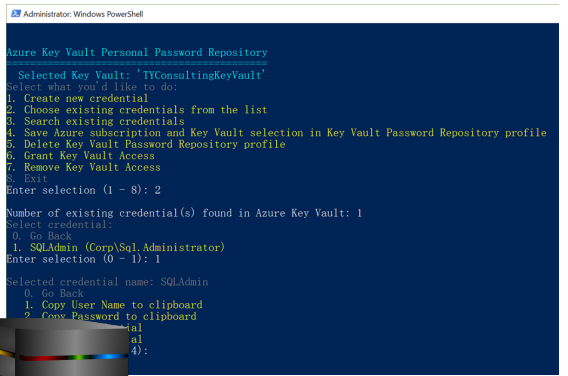
3 TLS handshake (c_random, ..)

5 Server Cert + Data

6 Symmetric key agreed

.....

.....

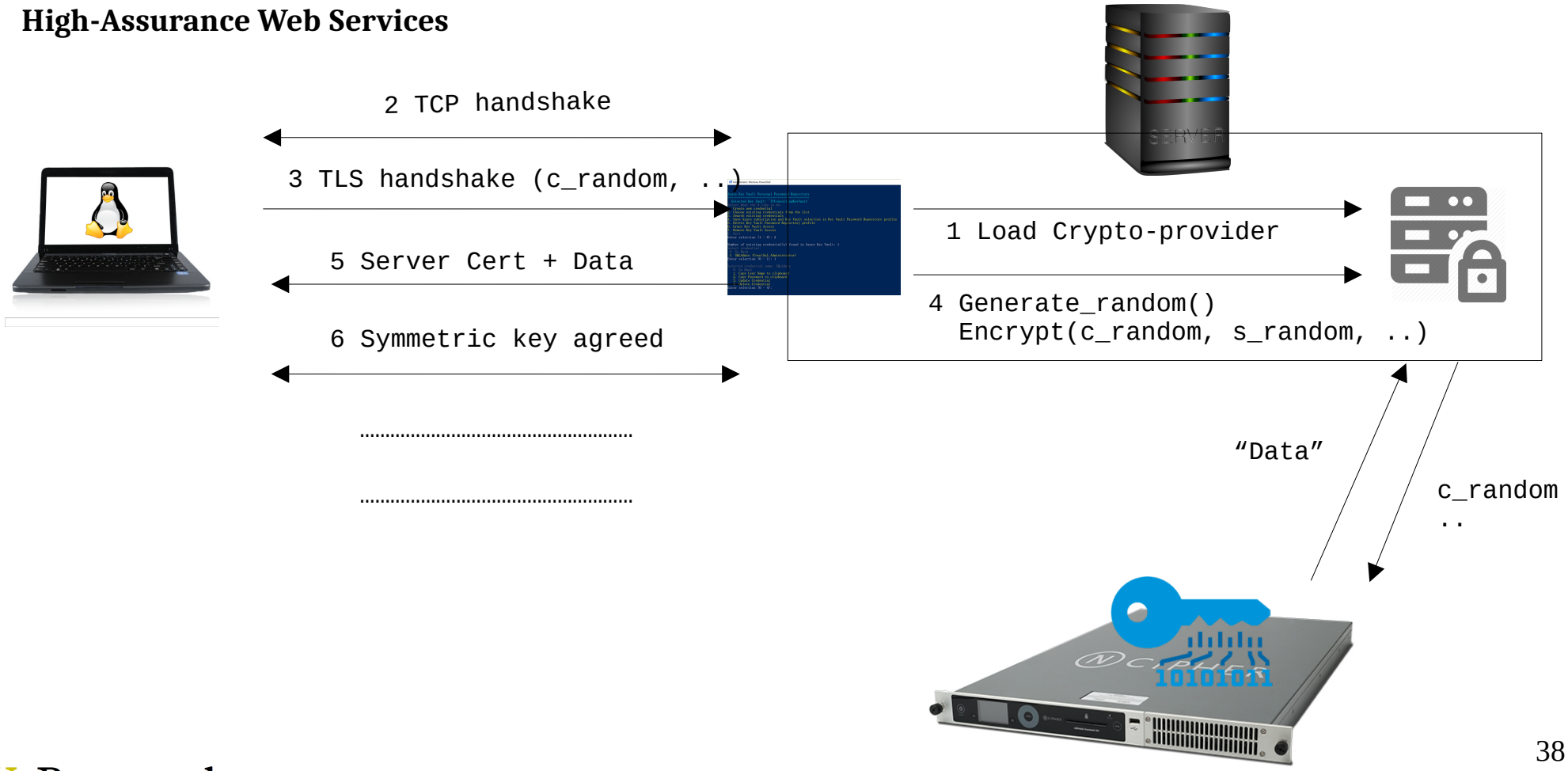


1 Load keys from disk

4 Generate(s_random)
Encrypt(c_random, s_random, ..)

Secure Code – Cryptographic Practices

High-Assurance Web Services



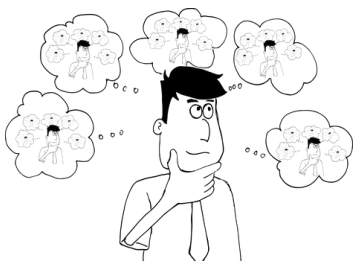
Concluding Remarks

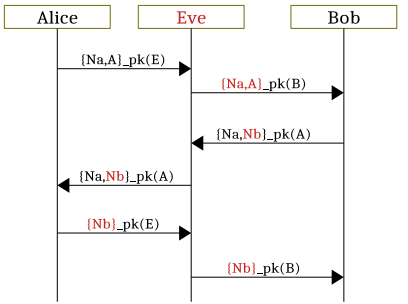
The Problem is How We Reason About Problems

All systems are insecure!

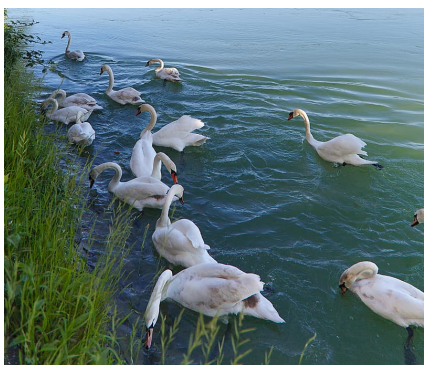
Attacker models

Testing techniques





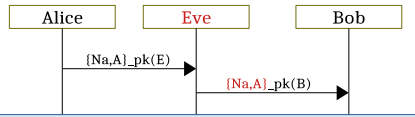
All swans are white!



V-ReseΛrch

The Problem is How We Reason About Problems

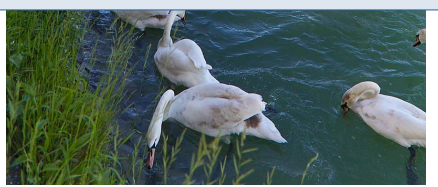
All systems are insecure!



A. Einstein writing to K. Popper (“The logic of scientific discoveries”)

"and I think (like you, by the way) that theory cannot be fabricated out of the results of observation, but that it can only be invented."

All swans are white!



V-ReseΛrch

Cybersecurity Hypothesis

How do we know errors?

A theory of Errors should predict all insecurities

...

Methodology error

Design error

Errors [CWE?]

No use of prepared statements

Causality

Weak System [CWE]

Weak sanitization function

Vulnerable System [CVE]

SQL-injection with payload ‘or 1=1’

System under attack [CAPEC]

Authentication bypass

Join Us!

Collaborations (thesis, internships, or just pass by our office for a coffee) – three main areas:

1) “I’m a believer” or “A quantitative but non-inductive approach to cybersecurity risk assessment”
2) “I’m an engineer” or ”A formal approach to the engineering of security protocols and cyber-physical systems”
3) “Pff... I’m a scientist, give me a challenge!” or “An attacker model beyond the Dolev-Yao one”

V-Reseʌrch

R&D for Cybersecurity Engineering

marco@v-research.it
francesco@v-research.it