Taming the Chaos: Managing Reproducibility in Experiments and Results

ACSAC LASER Workshop 2025 - 09.12.2025 - Hawaii

Sebastian Neef

Security in Telecommunications
TU Berlin, Germany

Who am I?

Sebastian Neef

- PhD candidate @ Chair for Security in Telecommunication, Technical University Berlin, Germany
- Research focuses on web-, network and software security
- Other activities
  - ▷ IT-Security freelancer (consulting, pentests, . . .)
  - ▷ Bugbounty and responsible disclosure programs

Our work: Notifications about SSHFP misconfigurations

- Earlier: **Oh SSH-it, What's My Fingerprint? A Large-Scale Analysis of SSH Host Key Fingerprint Verification Records in the DNS**, Neef et al. @ CANS 2022
  - ▷ `SSHFP` DNS records can improve security by allowing to automatically verify SSH servers' host keys
  - → Large-scale analysis → adoption is low, but high amount of misconfigurations → voids security benefits
  - ▷ `https://dl.acm.org/doi/10.1007/978-3-031-20974-1_4`

- Now: **Fix it - If you can! Towards Understanding the Impact of Tool Support and Domain Owners**, Hennig et al. @ACSAC 2025
  - ▷ Large-scale analysis focused on **notifiying** affected parties
  - ▷ Collected 12.3M (36M sub)domains → validated & scanned 7.97M (26.5M sub)domains
  - ▷ ~10.5k SSHFP instances → 1638 misconfigured → N=518 notifications
  - ▷ `https://s.kit.edu/sshfp-paper`

From Data to Publication

Why is reproducibility important?

- Independent reproduction necessary to validate findings
- Functional prototypes necessary for comparison to new approaches
- Open datasets necessary as basis for future work
- . . .

Why is reproducibility important?

- Independent reproduction necessary to validate findings
- Functional prototypes necessary for comparison to new approaches
- Open datasets necessary as basis for future work
- ...

- Unavailability of code/data/artifacts can be frustrating (anecdotal):
  - ▷ High efforts compiling similar or identical datasets
  - ▷ No understanding of used implementation → Can't identify bugs or hidden assumptions
  - ▷ Reviewers asking to compare to non-existing or non-functional related work prototypes
  - ▷ ...

Why is reproducibility important?

- Independent reproduction necessary to validate findings
- Functional prototypes necessary for comparison to new approaches
- Open datasets necessary as basis for future work
- ...

- Unavailability of code/data/artifacts can be frustrating (anecdotal):
  - ▷ High efforts compiling similar or identical datasets
  - ▷ No understanding of used implementation → Can't identify bugs or hidden assumptions
  - ▷ Reviewers asking to compare to non-existing or non-functional related work prototypes
  - ▷ ...
- Who had similar experiences?

Reproducibility: Best and worst case scenarios

- Let's assume all code and data were published
  - ▷ Code in a repository (i.e. GitHub, GitLab, etc.)
  - ▷ Data in a repository (i.e. GitHub, Zenodo, etc.)

- **Best case:** Successful Artifact Evaluation
  - ▷ Structured repository or prepared virtual machine, documentation of steps, ...
- **Worst case:** ”Trust me bro” or ”Works for me”
  - ▷ Unstructured repositories, missing documentation, ...

The reality of working with code and data in experiments...

... it can become a chaotic mess!

- **Changes to code:**
  - bug fixes, new algorithms, new aspects
  - $\rightarrow$ Multiple versions of a script, tool, prototype
- **Changes to input & output datasets**
  - additional data, supplementary data, more data cleaning, new outputs from updated tools/scripts
  - $\rightarrow$ Multiple versions of a dataset
- **Missing documentation**
  - ”Code is documentation enough”
  - Time pressure or ”will this even work?”
- **Doubts & Fears**
  - Somebody will judge me for my (bad) coding
  - Somebody could find bugs in the implementation

The reality of working with code and data in experiments...

... it can become a chaotic mess!

- **Changes to code:**
  - ▷ bug fixes, new algorithms, new aspects
  - ▷ $\rightarrow$ Multiple versions of a script, tool, prototype
- **Changes to input & output datasets**
  - ▷ additional data, supplementary data, more data cleaning, new outputs from updated tools/scripts
  - ▷ $\rightarrow$ Multiple versions of a dataset
- **Missing documentation**
  - ▷ ”Code is documentation enough”
  - ▷ Time pressure or ”will this even work?”
- **Doubts & Fears**
  - ▷ Somebody will judge me for my (bad) coding
  - ▷ Somebody could find bugs in the implementation

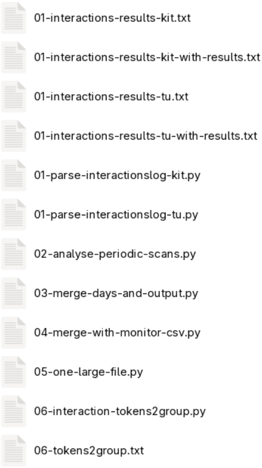
$\rightarrow$ How can we counter and overcome these issues?

Discussion: Status quo

- Have you had or noticed similar issues?
- How are you managing your data and code?
- Are there any environments or tools you can recommend?

Our approach to taming the chaos

1. **Version control: Git**
   - ▷ Allows to track changes to code and data
   - → What changed when, how and by whom?
2. **Naming scheme**
   - ▷ Incrementing folders/scripts/datasets for each step
     - ▷ Step 1: `01-data-collection.py` → `01-data.csv`
     - ▷ Step 2: `02-analysis.py` → `02-results.txt`
     - ▷ ...
   - → Easy to identify which code and data belongs to which processing step
   - → Allows to re-running one or more steps
3. **Containerization with Docker**
   - ▷ Abstraction layer from one own's computer / operating system configuration
   - ▷ One container with shared volumes for each processing step reduces build times
   - → Avoids "Works on my machine"-issues

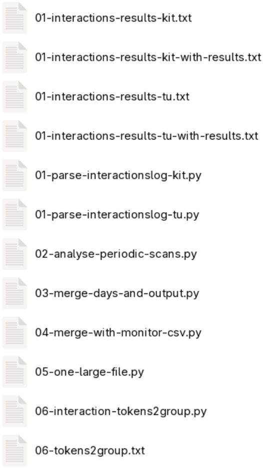


Our approach to taming the chaos

1. **Version control: Git**
   - ▷ Allows to track changes to code and data
   - → What changed when, how and by whom?

2. **Naming scheme**
   - ▷ Incrementing folders/scripts/datasets for each step
     - ▷ Step 1: `01-data-collection.py` → `01-data.csv`
     - ▷ Step 2: `02-analysis.py` → `02-results.txt`
     - ▷ ...
   - → Easy to identify which code and data belongs to which processing step
   - → Allows to re-running one or more steps

3. **Containerization with Docker**
   - ▷ Abstraction layer from one own's computer / operating system configuration
   - ▷ One container with shared volumes for each processing step reduces build times
   - → Avoids "Works on my machine"-issues



- Does this make sense? Would this be applicable to your research / experiments?

It’s not perfect: Further potential issues

1. **Version pinning?** (”Use dependency in version X.Y.Z.” vs. ”Use the latest version”)
   - ▷ Good: Avoids dependency changes that might break the code in the future
   - ▷ Bad: Specific versions might become unavailable
   - → Still pin, update when unavailable?

2. **Containerization vs. Virtualization?**
   - ▷ Containers might not be usable in all cases → Virtual Machines?
   - ▷ Problem: How to collaborate with VMs? How to share VMs? How to track (configuration) changes?
   - ▷ Problem: What about hardware-based or non-virtualizable experiments?

3. **Handling of intermediate/output datasets**
   - ▷ Problem: How to ensure that after a change all ”downstream” scripts are rerun and intermediate outputs updated?
   - → Automation via Makefiles? Versioning of outputs? Symlinks to ”latest” output?

4. **(G)UI tools**
   - ▷ Problem: What if graphical user interface tools are used and settings/input cannot be exported?
   - → Documentation of each interaction/input step? Screenshots of each view?

What do we already have?

- **FAIR principles**[1]
  - **F**indable → (Meta)data has identifier and indexed
  - **A**ccessible → (Meta)data accessible with open protocol
  - **I**nteroperable → (Meta)data has common format for storage/processing
  - **R**eusable → (Meta)data has suitable license and documentation

- **Ten Simple Rules for Reproducible Computational Research**[2]
  1. For Every Result, Keep Track of How It Was Produced
  2. Avoid Manual Data Manipulation Steps
  3. Archive the Exact Versions of All External Programs Used
  4. Version Control All Custom Scripts
  5. Record All Intermediate Results, When Possible in Standardized Formats
  6. For Analyses That Include Randomness, Note Underlying Random Seeds
  7. Always Store Raw Data behind Plots
  8. Generate Hierarchical Analysis Output, Allowing Layers of Increasing Detail to Be Inspected
  9. Connect Textual Statements to Underlying Results
  10. Provide Public Access to Scripts, Runs, and Results

[1] GO FAIR, 2016, `https://www.go-fair.org/fair-principles/`

[2] Sandve et al., 2013, `doi.org/10.1371/journal.pcbi.1003285`

Discussion: What (else) needs to be done?

- ”Solve the problem with technology”: Use package managers with reproducibility guarantees, e.g. Nix[3]?
  - ▷ Each experiment as a (standalone) Nix package?
  - ▷ Problem: Learning curve and adoption? Overhead?

- What are our expectations towards shared code and data?
- What are our expectations towards researchers?
- Is it even possible to come up with a ”one size fits ~~all~~ most” solution?
- How can we motivate (enforce?) such solutions or guidelines?

[3] https://nixos.org/

Call to action & Thank you!

- Please always consider reproducibility and follow the FAIR principles
- Use and submit to Artifact Evaluations as a way to get feedback
- Motivate others to do the same :)

- You can always reach out to me:
  - Email: neef@tu-berlin.de
  - Website: `https://sebastian-neef.de/`

`https://www.sebastian-neef.de/uploads/acsac25-laser.pdf`