Enowars 6 post mortem

Underleaf

Lenard Mollenkopf

Service Idee

- Echten Service nachbauen
- Service nachbauen den man danach noch verwenden könnte
- Overleaf als Vorbild

Was ist Overleaf?

- Kollaborativer LaTeX-Editor
- Workflow: Account erstellen → Projekt erstellen → Dokument schreiben → Dokument kompilieren → repeat
- Payed Features:
  - Synchronisation von Projekt und einem git-repo (nur in der cloud Version)
  - Kompilieren in isolierten Containern (nur in cloud und enterprise Version)

Was ist Overleaf?

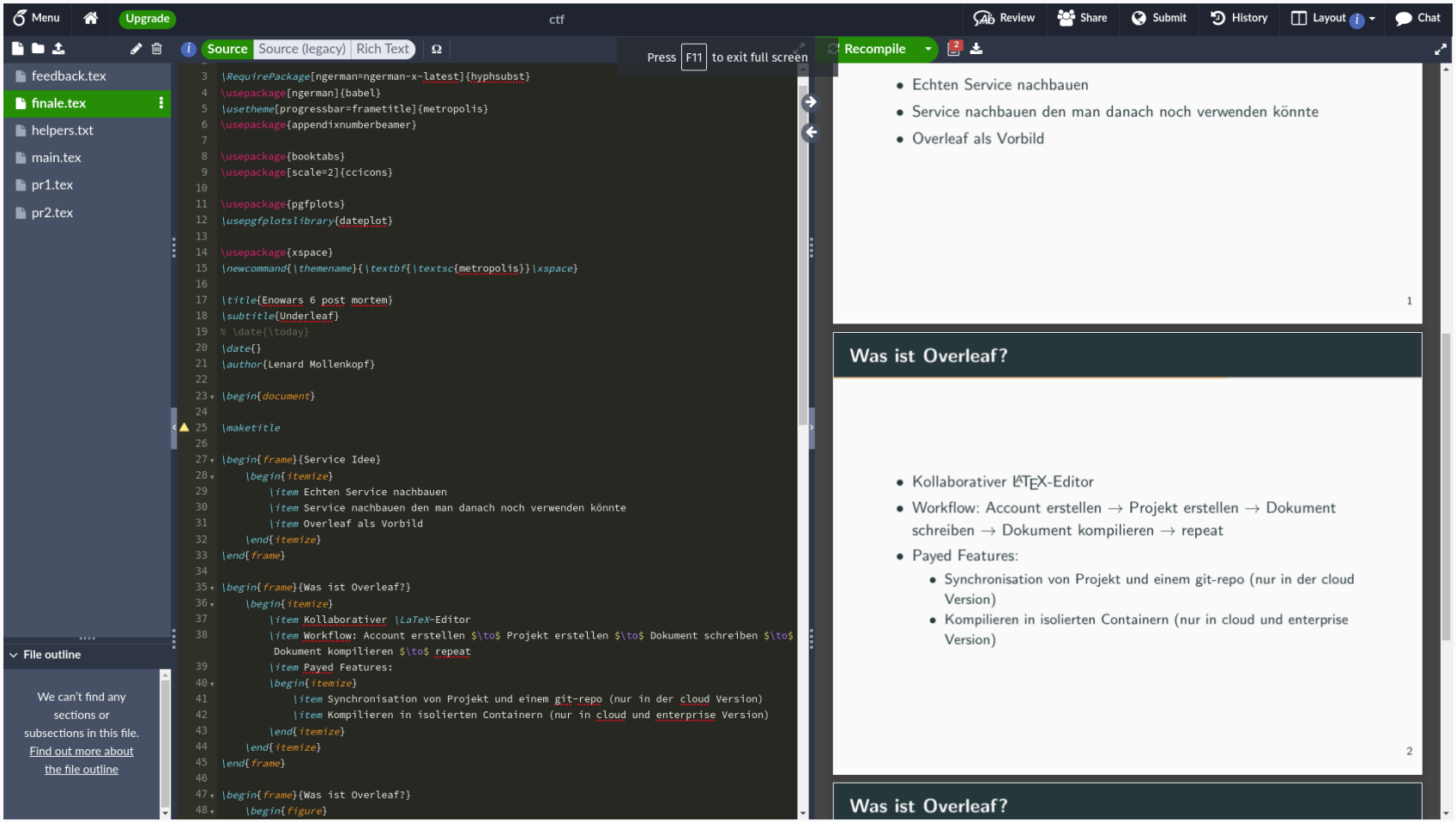

Abbildung 1: Screenshot von https://overleaf.com

Die Kopie - underleaf

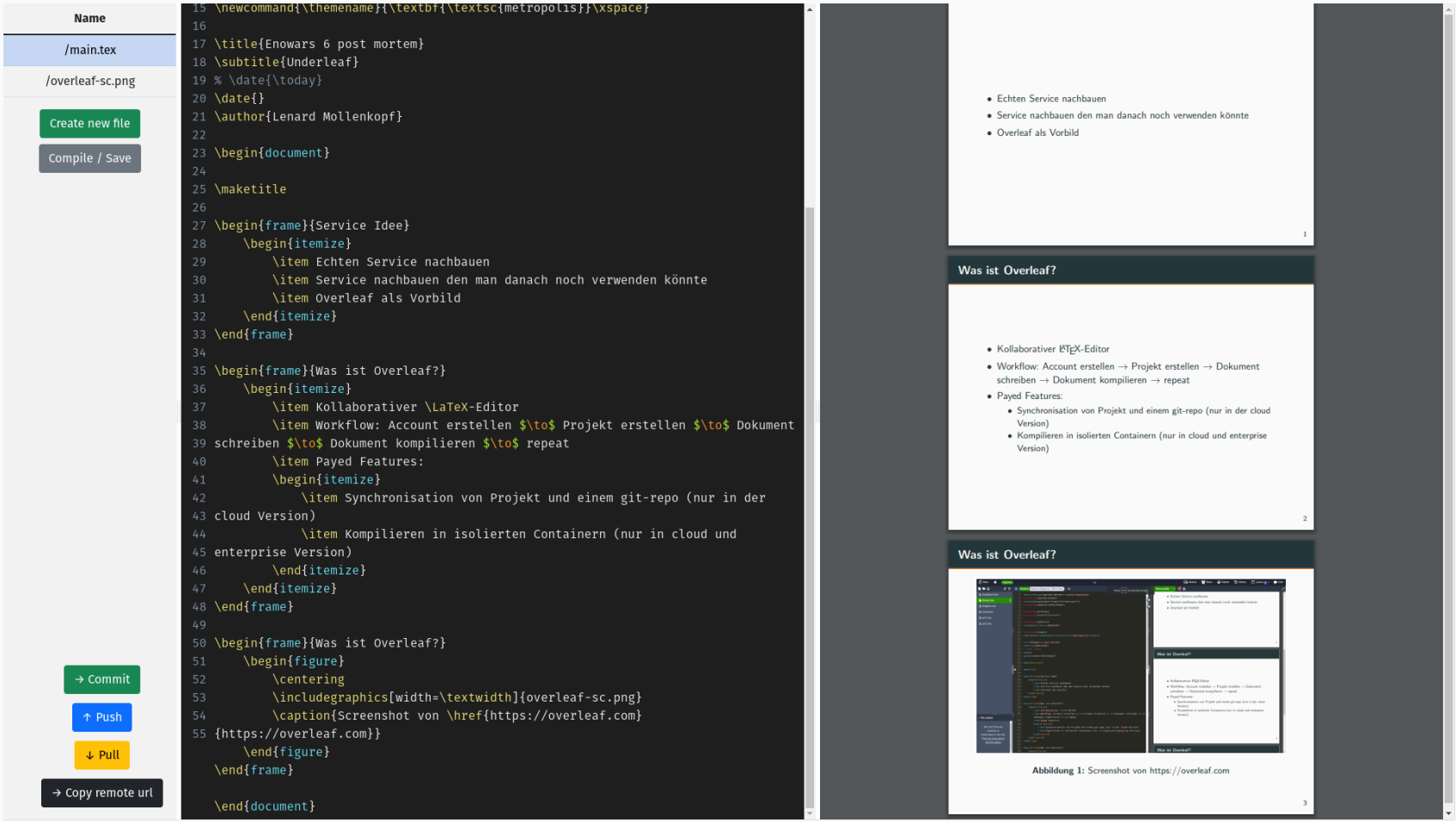

Abbildung 2: Screenshot von `underleaf`

Demo

Aufbau

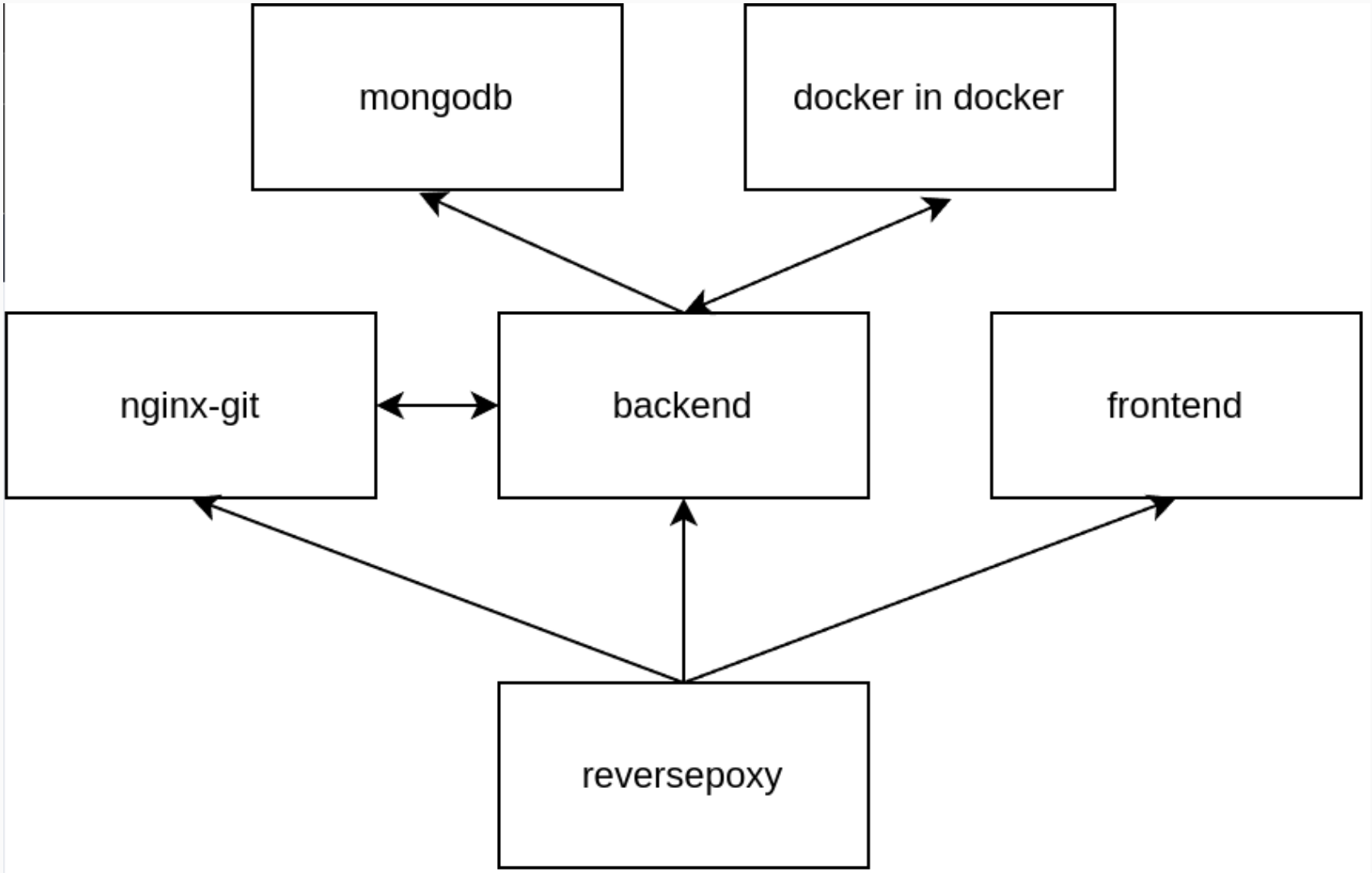

Abbildung 3: Screenshot von `underleaf`

Aufbau

- `reverseporxy`: nginx der Anfragen sortiert
- `frontend`: Vuejs interface
- `nginx-git`: Docker image von *catks*[1] basierend auf `git-http-server`[2], bzw. `git-http-backend`[3]
- `backend`: typescript + express App
- `mongodb`: Datenbank für `backend`
- `docker in docker`: dockerd für das Kompilieren von LaTeX

[1] https://github.com/catks/gitserver-http
[2] https://github.com/bahamas10/node-git-http-server
[3] https://github.com/substack/git-http-backend

Erste Schwachstelle

- Git kann `symlinks`
- express `res.download(path)` folgt symlinks
- kaputte Prüfung, ob Pfad erlaubt ist

Erste Schwachstelle (Code)

```
 9  export const downloadFile: RequestHandler = async function (req, res, next) {
10    try {
11      const projPath = resolve(getProjectPath(req.params.id));
12      const reqPath = req.params[0];
13      const path = resolve(projPath, reqPath);
14
15      if (await symlinkPathResolvesTo(path, getProjectPath(""))) {
16        if (!(await exists(path))) {
17          res.status(404).send("404 file not found");
18          return;
19        }
20
21        if ((await fs.lstat(path)).isDirectory()) {
22          res.status(403).send({ status: "path is a directory" });
23          return;
24        }
25
26        if (path.startsWith(resolve(projPath, ".git"))) {
27          res.status(403).send({ status: "path is in .git" });
28          return;
29        }
30
31        res.download(path);
32      } else {
33        res.status(403).json({ status: "File not accessible" });
34      }
35    } catch (e) {
36      next(e);
37    }
```

Abbildung 4: Ausschnitt von `downloadFile.ts`

Zweite Schwachstelle

- LaTeX kann RCE, da `-shell-escape` an ist[4]
- Docker soll Schutz bieten
- Docker isoliert Netzwerk nicht stark genug
- `db` ist über IP und ,,`db`" erreichbar

[4] https://0day.work/hacking-with-latex/

Zweite Schwachstelle (Exploit)

- LaTeX macht Verbindung zu `db` und Exploiter auf
- Exploiter ließt Daten aus der Datenbank aus
- Profit

Dritte Schwachstelle

- `reverseproxy` prüft Zugang zu `/git` per `auth_request` an `/api/auth/basic`
- `httpBasic.ts` ließt Projekt Id aus „x-original-url”
- Aber FALSCH!

Dritte Schwachstelle (Code)

```
7   try {
8     const url =
9       typeof req.headers["x-original-url"] === "string"
10        ? req.headers["x-original-url"]
11        : "in://valid";
12    const originalUrl = new URL(url);
13
14    if (
15      originalUrl.pathname.startsWith("/git/") &&
16      typeof originalUrl.pathname.split("/")[2] === "string"
17    ) {
18      const header = req.headers["authorization"];
19      if (header && header.startsWith("Basic ")) {
20        const base64 = header.split(" ")[1];
21        const [username, password] = Buffer.from(base64, "base64")
22          .toString()
23          .split(":");
24
25        if (await checkLogin(username, password)) {
26          if (
27            await canModifyProject(username, originalUrl.pathname.split("/")[2])
28          ) {
29            res.json(status_ok);
30            return;
31          }
32        }
33      }
34    }
35
36    res.status(401).json({ status: "unauthorized" });
```

Abbildung 5: Ausschnitt von `httpBasic.ts`

Vierte Schwachstelle + Exploit (Bonus)

Demo

Wertung

- *,,A bit above mid"*
- *,,Had some issues. Rendering didnt work at atl. "TypeError: crypto.subtle is undefined". Didn't look at it any further."*
- *,,The dev instructions on the readme did not work. The bug that was being exploited took me two hours to triage properly after I had reproduced it from pcaps, so props for that."*
- *,,fun challenge. restriction to container was engaging"*
- *,,interesting one! but when I tried to find a fix, our vm started crashing constantly."*
- *,,interesting unintended exploits (git path traversal) I don't like config bugs (mongodb)"*
- *,,I couldn't render the documents"*

Wertung

- I had fun with this challenge, probably in parts because we first-blooded it :D
- It felt overwhelming when first looking at it (so many containers, sub-folders, code). In hindsight I still feel like it could have been simpler and still include the same bugs. Having to read more code does not make a challenge more fun, it usually only adds pain.
- In our exploit we deleted more files that we needed to (out of laziness, rm * is short :P). We were told to stop it because it created a race condition between exploits. While I agree that it could remove the fun, I think should 100% be prevented by the service. Please implement the services in a way that does not allow stuff that you don't want to happen. Of course, no software is flawless and unintended bugs also happen in CTF challenges.

Wertung

- I liked the symlink/download bug more than the DB default credentials bug. The second one required far more exploit engineering (find the DB IP, talk to mongo without a mongo client present) while it was a lame bug (default creds) that was easy to fix (change password or add network separation).
- The PoW was annoying for exploitation. I get why you had to put it there, but it first increased load on our exploit thrower and then distracted us because we had to come up with a more clever way of computing and reusing the PoW between exploits.
- Having Docker and docker-compose files is great to reproduce a local setup. Some things were a little counter-intuitive, e.g. having to restart the reverse proxy after changing the backend, and that changing the db password env var only influences the db password during first creating of the container.

What worked?

- Mittlere Wertung im SLA
- Meisten Vulns gut angekommen
- Vuln und Fix nicht offensichtlich

What did not work?

- `Crypto.subtle` nur über `https` oder `localhost`.
- Viel Code / Complexität
- Probleme mit erstem Start / dev setup
- Unintended Vulns

Lessons Learned

- Service sollte *gutes* Logging haben
- Defense in Depth ist auch wichtig, wenn sich schon viele den Code angeguckt haben
- *Perfomance Matters* geht nicht im Nachhinein