TECHNISCHE HOCHSCHULE DEGGENDORF

Bachelor – Cyber Security

Kryptologie 2

Projektdokumentation

Cryptochallenge: CurveBall (CVE-2020-0601)

Autoren
Manuel Friedl – 1236626
Christof Renner – 22301943

Betreuer
Prof. Dr. Martin Schramm

Deggendorf, July 27, 2025

Contents

1 Einleitung und Projektkontext

1.1 Motivation

Die Schwachstelle **CurveBall** (CVE-2020-0601) in der Windows-CryptoAPI ermöglicht es, X.509-Zertifikate mit manipulierten Elliptic-Curve-Parametern zu signieren, sodass betroffene Windows-Versionen die Signaturen fälschlich als gültig akzeptieren.[1] Im Modul *Kryptologie 2* fehlte bislang ein modernes Hands-On-Szenario, um diesen Fehler praktisch zu demonstrieren.

1.2 Projektziele

1. **Didaktik**: Vollständiger Angriffszyklus von Discovery bis Exploit.
2. **Sicherheit**: Deployment selbst muss trotz absichtlich verletzter Krypto sicher sein.
3. **Portabilität**: Schnelle, plattformunabhängige Nutzung via Docker/Podman.

1.3 Bedrohungsmodell

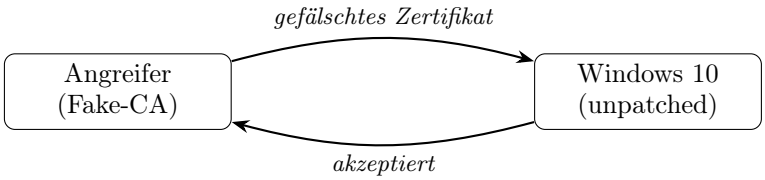

Figure 1: Simplifiziertes Threat-Model: fehlende Parameter-Validierung

2 Herangehensweise

2.1 Zielsetzung

- Demonstration des Angriffs in einer kontrollierten Umgebung.
- Vermittlung von DevSecOps-Best-Practices (Linting, CI, Scans).
- Bereitstellung als *„One-Click“*-Container, ohne lokale OpenSSL-Konfiguration.

2.2 Methodik

Wir arbeiteten in zwei Sprints à zwei Wochen. Abbildung 2 zeigt den iterativen Ablauf.

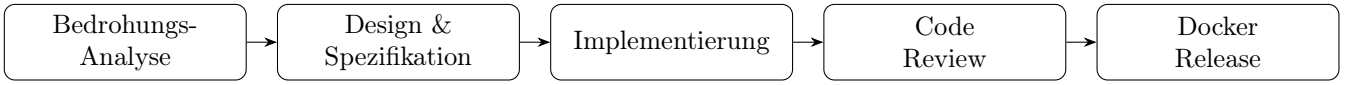

Figure 2: Iterativer Projektablauf

[1] Microsoft Security Advisory ADV200002, 14. 01. 2020

2.3 Technisches Design

Das technische Design der CurveBall-Challenge basiert auf einer modularen Web-Anwendung, die in Python mit Flask entwickelt wurde. Die Architektur folgt dem Model-View-Controller-Pattern und ist speziell darauf ausgelegt, kryptographische Konzepte interaktiv zu vermitteln.
Die Anwendung besteht aus mehreren Kernkomponenten:

- **Zertifikatsgenerator**: Python-Module zur Erstellung von X.509-Zertifikaten mit manipulierten ECC-Parametern
- **Validierungsengine**: Simulation der Windows CryptoAPI-Schwachstelle zur Demonstration fehlerhafter Zertifikatsprüfung
- **Web-Interface**: Interaktive Benutzeroberfläche für schrittweise Challenge-Durchführung
- **Visualisierungstools**: Grafische Darstellung von Zertifikatsinhalten und kryptographischen Parametern

Die Challenge ist als Progressive Web Application konzipiert, die ohne Installation direkt im Browser läuft. Dabei wird besonderer Wert auf eine intuitive Benutzerführung gelegt, die auch Studierende ohne tiefgreifende Kryptographie-Kenntnisse abholt und schrittweise durch die komplexe Materie führt.

2.4 Spielweise

Die CurveBall-Challenge ist als interaktive Lernreise konzipiert, die Studierende spielerisch an das Thema heranführt.

1. **Starten der Challenge**: Durch starten des Docker-Containers wird die Web-Anwendung bereitgestellt.
2. **Aufrufen der Web-Anwendung**: Die Studierenden öffnen die Web-App im Browser unter `https://localhost:8443` und erhalten eine Einführung in die Challenge.
3. **Einführung & Theorie**: Vermittlung der Grundlagen zu elliptischen Kurven und X.509-Zertifikaten
4. **Challenges**: Danach folgen verschiedene Herausforderungen, die die Studierenden dazu anregen, das Gelernte anzuwenden und zu vertiefen.

Jede Phase beinhaltet interaktive Elemente wie Code-Eingabefelder, Zertifikatsvisualisierungen und Validierungstools. Die Studierenden erhalten unmittelbares Feedback zu ihren Eingaben und können experimentell verschiedene Ansätze ausprobieren. Ein integriertes Hinweissystem bietet bei Bedarf gezielten Support, ohne die Lösung vorwegzunehmen.
Die Challenge kann sowohl einzeln als auch in Kleingruppen bearbeitet werden und ist zeitlich flexibel gestaltbar – von kompakten 90-Minuten-Sessions bis zu mehrstündigen Vertiefungen.

3 Aufbau und Komponenten

3.1 Flask Webserver

Der *server.py* ist das Herzstück der Curveball CTF-Webanwendung und implementiert einen Flask-basierten Webserver mit SSL-Unterstützung. Der Server verwaltet 4 aufeinanderfolgende Challenges zum Thema ECC-Kryptographie und der CVE-2020-0601 Curveball-Schwachstelle.

Architektur und Technologie:

- **Framework:** Flask (Python Web Framework)
- **Session-Management:** Flask Sessions mit Secret Key
- **SSL/TLS:** HTTPS über Port 8443 mit eigenen Zertifikaten
- **Template Engine:** Jinja2 (Flask Standard)

Kernfunktionalitäten:

1. **Challenge-Fortschrittsverfolgung**
   - **Session-basiert:** Nutzt Flask Sessions zur Speicherung des Fortschritts
   - **Sequenzieller Ablauf:** Challenges werden der Reihe nach freigeschaltet
   - **Persistenz:** Fortschirtt bleibt während der Browser Session erhalten

```
    def is_challenge_unlocked(challenge_number):
    """Checks if a challenge is unlocked"""
        if challenge_number == 1:
            return True  # Challenge 1 is always unlocked

        completed = get_completed_challenges()
        # Challenge N is only available if Challenge N-1 is completed
        return (challenge_number - 1) in completed
```

Auflistung 1: Challenge-Lock

2. **Route-Struktur**
   - **Hauptrouten:**
     - / => Startseite mit Challenge-Übersicht
     - /introduction => Curveball-Einführung (Markdown zu HTML)
     - /challenge1 bis /challenge4 => Einzelne Challenge-Seiten
   - **API-Endpunkte:**
     - /api/complete_challenge/<int> => Challenge als abgeschlossen markieren
     - /api/challenge_status => Aktueller Fortschrittsstatus
     - /api/reset_progress => Zurücksetzen des Fortschritts (Debug)
   - **Download-Endpunkte:**
     - /downloads/<filename> => Challenge-Dateien (Zertifikate, JSON)
     - /scripts/<filename> => Python-Skripte für Challenges
     - /explain/<topic> => Kryptographie-Konzepte erklärt
3. **Challenge-Inhalte**

(a) **Challenge1:** ECC Grundlagen - Punktmultiplikation

(b) **Challenge2:** Zertifikatsanalyse mit OpenSSL

(c) **Challenge3:** Curveball Exploit Simulation

(d) **Challenge4:** Kurvenparameter & Signaturvalidierung

3.2 Javascript

Die CTF-Anwendung verwendet ein modulares JavaScript-System mit separaten Dateien für verschiedene Funktionsbereiche. Das System kombiniert mathematische ECC-Berechnungen, interaktive UI-Elemente und Fortschrittsverfolgung.

3.2.1 main.js - Zentrale Steuerung

- **Funktion:** Grundlegende Seitenfunktionalität und Easter Egg
- **Features:**
  - DOM-Initialisierung
  - Konami-Code Implementation
  - Visuelle Effekte (Gradient-Animation)
  - Basis-Event-Handling

3.2.2 progress.js - Fortschrittssystem

- **Kernklasse:** ChallengeProgress
- **Funktionalitäten:**
  - Session-basierte Fortschrittsverfolgung
  - Server-API Integration
  - Dynamische UI-Updates für Challenge-Cards
  - Visuelle Status-Indikatoren (gesperrt/freigeschaltet/abgeschlossen)

3.2.3 challenge1.js - ECC Grundlagen

- **Thema:** Elliptische Kurven Punktmultiplikation
- **Kurvendefinition:** $y^2 = x^3 + 3 \pmod{97}$, Generator G = (11,3)
- **Mathematische Funktionen:**
  - *modInverse()* - Modulare Inverse Berechnung
  - *pointAdd()* - Elliptische Kurven Punktaddition
  - *scalarMult()* - Skalare Multiplikation
  - *isOnCurve()* - Punktvalidierung

3.2.4 challenge2.js - Zertifikatsanalyse

- **Thema:** Analysieren eines Zertifikats auf eine Flag
- **Max. Versuche:** 10 mit eskalierenden Hinweisen
- **Kernfunktionen:**
  - Flag-Validierung mit mehreren Formatvarianten
  - Progressives Hint-System nach mehreren Versuchen
  - OpenSSL-Befehl-Kopierungsfunktion
  - Confetti-Animationen bei Erfolg
  - Lokale Speicherung des Fortschritts

3.2.5 challenge3.js - Curveball Exploit-Simulation

- **ECC-Kurven:** Unterstützung für P-256, P-384, P-521
- **Angriffsmethoden:**
  - Generator-Punkt Manipulation
  - Parameter-Spoofing
  - Custom Exploits
  - Confetti-Animationen bei Erfolg
  - Lokale Speicherung des Fortschritts
- **Kernfunktionen:**
  - *generateExploitParameters()* - Erstellt manipulierte Parameter
  - *generateRogueGenerator()* - Generiert manipulierte Generator-Punkte
  - Mock-Zertifikatserstellung
  - OpenSSL-Befehlsgenerierunng

3.2.6 challenge4.js - Kurvenparameter & Signatur-Validierung

- **Standard-Kurve:** NIST P-256 Parameter
- **Max. Versuche:** 10 mit eskalierenden Hinweisen
- **Manipulationsmethoden:**
  - Generator-Punkt Spoofing
  - Kurvenparameter-Manipulation (a,b)
  - Gruppenordnung-Manipulation (n)
  - Benutzerdefinierte Kombinationen
- **Kernfunktionen:**
  - *performManipulation()* - Führt Parameter-Manipulationen durch
  - *calculateExploitPotential()* - Bewertet Angriffspotenzial (0-100)
  - Hex-Validierung und -Manipulation
  - JSON-Export für Parameter

3.3 Python-Skript: verification.py

Das Python-Skript verification.py dient als Simulator zur Analyse der Curveball-Schwachstelle (CVE-2020-0601) im Windows CryptoAPI. Es ermöglicht das Verständnis des fehlerhaften Verhaltens bei der ECC-Zertifikatsvalidierung, indem es zeigt, wie Windows in der betroffenen Version kritische Prüfungen – insbesondere die Validierung der ECC-Parameter und des Generatorpunkts – überspringt. Das Skript unterstützt zwei Betriebsmodi: einen Analysemodus, der die Unterschiede zwischen sicherer und unsicherer Validierung aufzeigt, sowie einen Validierungsmodus, mit dem manipulierte ECC-Zertifikate überprüft und potenzielle Exploit-Marker erkannt werden können. Durch die Simulation des Schwachstellenverhaltens bietet das Tool eine praxisnahe Möglichkeit zur Schulung und zum sicheren Testen von Sicherheitsmechanismen im Bereich der Zertifikatsprüfung. Bei erfolgreicher Ausnutzung wird eine symbolische Flagge ausgegeben, was das Skript auch für CTF-Challenges besonders geeignet macht.

```
    curve = public_key.curve
    curve_name = curve.name if hasattr(curve, 'name') else 'unknown'

    standard_curves = ['secp256r1', 'secp384r1', 'secp521r1']

    if curve_name not in standard_curves:
        return True, f"Non-standard curve detected: {curve_name}"
```

Auflistung 2: analyze_ecc_parameter()-Methode

Im Rahmen der Schwachstellenanalyse überprüft das Skript, ob ein Zertifikat auf einer nicht standardisierten elliptischen Kurve basiert. Standardmäßig werden in der Praxis nur gut geprüfte Kurven wie **secp256r1**, **secp384r1** und **secp521r1** verwendet. Das Snippet extrahiert den Namen der verwendeten Kurve aus dem Zertifikat und vergleicht ihn mit dieser Liste. Wenn eine nicht standardisierte Kurve erkannt wird, deutet das auf eine manipulierte ECC-Struktur hin – ein zentrales Element der Curveball-Ausnutzung. Dadurch wird sichtbar, ob das Zertifikat potenziell Teil eines Exploits ist.

3.4 Python-Skript: signature_validator.py

In diesem Skritp wird demonstriert, wie die Curveball-Schwachstelle (CVE-2020-0601) ausgenutzt werden kann, um gültige ECDSA-Signaturen auf Basis manipulierter elliptischer Kurvenparameter zu erzeugen – ohne Kenntnis des privaten Schlüssels. Im Zentrum steht die Klasse ECCSignatureValidator, die eine vereinfachte ECC-Signaturprüfung durchführt. Mithilfe manipulierten Parametersätzen lassen sich gezielt Situationen erzeugen, in denen Signaturen als gültig anerkannt werden, obwohl sie unter vertrauenswürdigen Parametern nicht gültig wären.

Die Validierung beruht auf klassischen ECDSA-Schritten: Hashing der Nachricht, Inversion von s, Berechnung von Punktkombinationen auf der elliptischen Kurve und abschließender Vergleich.

```
def point_multiply(self, k: int, x: int, y: int) -> Tuple[int, int]:
    """Scalar multiplication on elliptic curve (simplified)"""
    if k == 0:
        return None, None
    if k == 1:
        return x, y

    result_x, result_y = None, None
    addend_x, addend_y = x, y

    while k:
```

```
12          if k & 1:
13              if result_x is None:
14                  result_x, result_y = addend_x, addend_y
15              else:
16                  result_x, result_y = self.point_add(result_x, result_y, addend_x
                        , addend_y)
17          addend_x, addend_y = self.point_add(addend_x, addend_y, addend_x,
                addend_y)
18          k >>= 1
19
20      return result_x, result_y
```

Auflistung 3: Punktmultiplikation auf der Kurve

Dieses Snippet implementiert das sogenannte Double-and-Add-Verfahren, um Punkte auf einer elliptischen Kurve effizient zu multiplizieren. Diese Operation ist zentral für die ECDSA-Signaturvalidierung und angreifbar, wenn die Kurvenparameter manipuliert werden.

```
1       final_x = result_x % self.order
2       is_valid = final_x == signature_r
```

Auflistung 4: Signaturprüfung mit Kurvenparametern

Der finale Schritt der Validierung vergleicht die x-Koordinate des berechneten Punkts mit dem übermittelten Signaturwert r. Wird durch manipulierte Parameter gezielt ein passender final_x konstruiert, kann eine gefälschte Signatur als gültig erscheinen – Kern der Curveball-Ausnutzung.

4 Arbeitsaufteilung

Die Arbeitsaufteilung erfolgte pragmatisch basierend auf den individuellen Stärken der Teammitglieder, wobei gleichzeitig Wert auf Wissenstransfer und gemeinsames Lernen gelegt wurde.

Teammitglied	**Hauptaufgaben**	**Spezifische Aufgaben**
Manuel Friedl	Kryptographie & Frontend	• Konzeptionierung der einzelnen Challenges • Analyse der CVE-2020-0601 Schwachstelle • Entwicklung der Python-Skripte `verification.py` und `signature_validator.py` • Entwicklung des Web-Interface und Zertifikats-Visualizers • Erstellung verständlicher Challenge-Beschreibungen • Implementierung der Angriffslogik in Python • Frontend-Design mit HTML/CSS • Erstellen der Zertifikate `mystery_cert.pem` und `valid_certificate.pem` • Dokumentation der kryptographischen Konzepte

Teammitglied	**Hauptaufgaben**	**Spezifische Aufgaben**
Christof Renner	DevOps & Infrastruktur	• Konzeptionierung der einzelnen Challenges • Docker-Containerisierung mit Multi-Stage-Builds • CI/CD-Pipeline mit GitHub Actions • Security-Scanning und Linting-Integration • GitHub Container Registry Konfiguration • Dokumentation der technischen Details • Container-Deployment-Architektur • Implementierung der Sicherheitsmaßnahmen • Frontend-Optimierungen

Dabei wurde durch regelmäßige Abstimmungen und gemeinsame Review-Sessions sichergestellt, dass alle Komponenten nahtlos zusammenarbeiten. Die technische Dokumentation wurde gleichmäßig zwischen beiden Teammitgliedern aufgeteilt, wobei jeder etwa 50% der Dokumentationsarbeit übernahm.

5 Containerisierung

5.1 Dockerfile

Das vorliegende Dockerfile definiert die Containerisierung der Curveball-CTF Webanwendung und nutzt dabei Python 3.12 auf Alpine Linux als schlanke und sicherheitsorientierte Basis. Durch die explizite Installation von OpenSSL Version 3.5.1-r0 werden kryptographische Funktionalitäten bereitgestellt, die für die SSL/TLS-basierten Challenges essentiell sind. Die Anwendung wird systematisch aufgebaut, indem zunächst die Python-Abhängigkeiten aus der *requirements.txt* installiert werden, bevor der Anwendungscode kopiert wird - eine bewährte Praxis für effizientes Docker Layer Caching. Der Container exponiert Port 8443, einen Standard-Port für HTTPS-Verbindungen, und startet automatisch den Python-Server über das *server.py*-Skript. Diese Konfiguration gewährleistet eine reproduzierbare, isolierte Umgebung für die Kryptologie-Challenges und minimiert gleichzeitig durch die Alpine-Basis die Angriffsfläche des Systems.

```
1  FROM python:3.12-alpine
2
3  # Pin Alpine package versions for reproducible builds
4  RUN apk add --no-cache openssl=3.5.1-r0
5
6  WORKDIR /app
7  COPY requirements.txt /app/
8  RUN pip install --no-cache-dir --requirement requirements.txt
9
10 COPY . /app
11
12 EXPOSE 8443
13
14 CMD ["python", "server.py"]
```

Auflistung 5: Auszug aus dem finalen Dockerfile

5.2 Architektur

Figure 3: Container-Deployment in der Lehrumgebung

6 CI/CD-Pipeline

6.1 Docker Build-Pipeline

Die automatisierte Erstellung und Veröffentlichung der Docker-Images erfolgt über eine dedizierte GitHub Actions Workflow-Datei. Diese Pipeline stellt sicher, dass bei jeder Aktualisierung des Codes oder auf manuelle Anforderung ein neues, konsistentes Container-Image erstellt und in die Docker Hub Registry hochgeladen wird. Der Workflow ist so konfiguriert, dass er manuell über die GitHub-Oberfläche ausgelöst werden kann (`workflow_dispatch`), was besonders während der Entwicklungsphase und für kontrollierte Releases nützlich war.

```yaml
name: Build and Publish Docker image

on:
  workflow_dispatch:

jobs:
  build-and-push:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Log in to Docker Hub
        uses: docker/login-action@v3
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Build and push Docker image
        uses: docker/build-push-action@v6
        with:
          context: ./curveball-ctf/webserver
          file: ./curveball-ctf/webserver/Dockerfile
          push: true
          tags: crnnr/curveball-cve-2020-0601:latest
```

Auflistung 6: Docker Build & Publish Workflow

Die Pipeline besteht aus mehreren wichtigen Schritten:

Durch diese Automatisierung wird der Release-Prozess erheblich vereinfacht und gleichzeitig sichergestellt, dass jedes veröffentlichte Image den gleichen, reproduzierbaren Build-Prozess durchlaufen hat.

Die Verwendung von gesicherten Secrets für die Authentifizierung erhöht zusätzlich die Sicherheit der Pipeline. Die Container-Registry fungiert als zentrales Repository für die fertigen Images, was die Verteilung an Studierende erheblich vereinfacht.

Mit einem einfachen `docker pull` Befehl können Dozenten und Studenten die aktuelle Version der Challenge beziehen.

6.2 Linting-Tools

Für die Code-Qualität und Sicherheit setzen wir auf etablierte Tools:

- **pylint**: Python-Code-Analyse nach PEP8
- **hadolint**: Dockerfile-Best-Practices

Diese Tools sind in der CI-Pipeline integriert und prüfen den Code bei jedem Commit.

```yaml
jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.8", "3.9", "3.10"]
    steps:
    - uses: actions/checkout@v4
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v5
      with:
        python-version: ${{ matrix.python-version }}
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install pylint
    - name: Analysing the code with pylint
      run: |
        pylint $(git ls-files '*.py')
```

Auflistung 7: pylint.yml

Damit konnte am Ende ein **pylint**-Score von **9.5/10** erreicht werden, was dafür sorgt, dass der Code auch in Zukunft gut lesbar und wartbar bleibt.

6.3 Dockerfile-Linting

Die Dockerfile-Qualität wird mit **hadolint** geprüft, um Best Practices zu gewährleisten. Das Linting erfolgt ebenfalls in der CI-Pipeline, um sicherzustellen, dass das Dockerfile den Standards entspricht.

```yaml
jobs:
  hadolint:
    name: Hadolint
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Run Hadolint
        uses: hadolint/hadolint-action@v2
        with:
          dockerfile: ./Dockerfile
```

Auflistung 8: hadolint.yml

6.4 Sicherheits-Scans

Die Sicherheitsüberprüfung erfolgt mit **Bandit** und **Snyk**. Bandit analysiert Python-Code auf Sicherheitslücken, während Snyk Container-Images auf bekannte Schwachstellen prüft. Bandit ist in der CI-Pipeline integriert und prüft den Code bei jedem Commit.

```yaml
jobs:
  bandit:
    name: Bandit Scan
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.x'
      - name: Install Bandit
        run: |
          python -m pip install --upgrade pip
          pip install bandit

      - name: Run Bandit security scan
        run: |
          # scan the repo recursively instead of using stdin
          bandit -r . --skip trojansource
```

Auflistung 9: bandit workflow file

Damit wird sichergestellt, dass der Code und die Container-Images regelmäßig auf Sicherheitslücken überprüft werden. Die Ergebnisse der Scans werden in der CI-Pipeline angezeigt. Hierbei wurden einige Schwachstellen identifiziert:

```
Code scanned:
        Total lines of code: 602
        Total lines skipped (#nosec): 0
        Total potential issues skipped due to specifically being disabled (e.g.,
             #nosec BXXX): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 6
                Medium: 1
                High: 1
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 3
                High: 5
```

Auflistung 10: bandit scan results

Die identifizierten Schwachstellen sind folgendermassen verteilt:

Severity	Confidence	Issue Type	Anzahl
High	High	B105: hardcoded_password_string	1
Medium	High	B101: assert_used	1
Low	High	B311: random	3
Low	Medium	B404: import_subprocess	1
Low	Medium	B603: subprocess_without_shell_equals_true	1
Low	Medium	B607: start_process_with_partial_path	1

Figure 4: Bandit Security Scan Ergebnisse – Identifizierte Schwachstellen nach Severity und Confidence

Diese Schwachstellen wurden im Rahmen der Entwicklung analysiert und entsprechend dem Projektkontext bewertet:

- **B404: import_subprocess**: Die referenziert die Verwendung von `subprocess` in sicherheitskritischen Bereichen. Hier wurde sichergestellt, dass alle Eingaben validiert werden, um potenzielle Angriffe zu verhindern.
- **B607: start_process_with_partial_path**: Diese Warnung bezieht sich auf die Verwendung von relativen Pfaden beim Starten von Prozessen. Um das Risiko von Pfadmanipulationen zu minimieren, wurde darauf geachtet, dass alle Pfade nicht relativ sind und keine Benutzereingaben direkt in Pfade einfließen.
- **B603: subprocess_without_shell_equals_true**: Diese Warnung bezieht sich auf die Verwendung von `subprocess` ohne die Option `shell=True`. Hier wurde sichergestellt, dass alle Aufrufe von `subprocess` sicher sind und keine Shell-Injection-Angriffe möglich sind.
- **B201: flask_debug_true**: Diese Warnung bezieht sich auf die Verwendung von `flask` im Debug-Modus. In einer produktiven Umgebung sollte der Debug-Modus deaktiviert werden, um potenzielle Sicherheitsrisiken zu minimieren.
- **B104: hardcoded_bind_all_interfaces**: Bezieht sich auf die Konfiguration des Flask-Servers, der auf allen Schnittstellen lauscht. Dies ist für die Challenge-Umgebung akzeptabel, da sie in einer kontrollierten Umgebung läuft und nicht öffentlich zugänglich ist.

Doppelte Warnungen wurden im Rahmen der Entwicklung analysiert und entsprechend dem Projektkontext bewertet. Die bewusste Entscheidung, bestimmte Bandit-Warnungen im Challenge-Kontext zu akzeptieren, wurde dokumentiert und entspricht dem Projektcharakter als kontrollierte Lernumgebung.

7 Nicht umgesetzte VM-Erweiterung

Ursprünglich war eine vorgefertigte Windows 10-VM (Version 1909, ungepatcht) als zentraler Bestandteil des Projekts geplant, um den CurveBall-Angriff vollständig bis zum System-Rootstore zu demonstrieren und eine authentische Angriffsumgebung zu schaffen. Diese VM hätte es ermöglicht, die realen Auswirkungen eines erfolgreichen Angriffs unmittelbar zu beobachten, einschließlich der Manipulation von HTTPS-Verbindungen und Code-Signatur-Verifikationen.

7.1 Herausforderungen bei der VM-Implementierung

Die Umsetzung dieses Ansatzes scheiterte letztendlich aus mehreren gravierenden Gründen:

Licensing-Problematik Die Weitergabe eines vorinstallierten Windows-Images verstößt eindeutig gegen die Microsoft End User License Agreement (EULA). Eine rechtskonforme Lösung hätte erfordert, dass jeder Teilnehmer eine eigene Windows-Lizenz einbringt und selbst eine Installation durchführt, was den niederschwelligen Zugang zur Übung erheblich erschwert hätte.

Storage-Anforderungen Das vollständige Windows 10-Image hätte mindestens 8 GB Speicherplatz benötigt, selbst in komprimierter Form. Dies hätte den Rahmen des Git-Repositorys gesprengt und eine Nutzung von Git LFS (Large File Storage) erforderlich gemacht, was mit erheblichen Kosten für Bandbreite und Speicherplatz verbunden gewesen wäre. Zudem hätte die Verteilung an mehrere Dutzend Studierende die Netzwerkressourcen der Hochschule stark belastet.

CI/CD-Limitationen Die genutzten GitHub-Actions-Runner unterstützen keine Nested-Virtualisation, was eine automatisierte Überprüfung und Tests der VM innerhalb der CI/CD-Pipeline unmöglich machte. Dies hätte zu nicht getesteten Releases führen können, was dem DevSecOps-Ansatz des Projekts fundamental widerspricht.

7.2 Vorteile der Container-Lösung

Die stattdessen implementierte Container-Variante bietet mit rund 280 MB eine um mehr als 96% reduzierte Größe im Vergleich zur VM-Lösung. Diese drastische Reduktion ermöglicht:

- **Schnellere Deployments**: Studierende können das Image in Sekunden statt Minuten herunterladen
- **Plattformunabhängigkeit**: Funktioniert auf Windows, macOS und Linux ohne Anpassungen
- **Geringere Systemanforderungen**: Läuft auf nahezu jedem Rechner, der Docker unterstützt
- **Einfache Integration in CI/CD**: Vollständige Testabdeckung in der Entwicklungspipeline
- **Reproduzierbare Builds**: Jeder Build erzeugt identische Umgebungen

8 Ausblick

Die entwickelte Container-Lösung bildet eine solide Grundlage für weitere Erweiterungen des Projekts. Folgende Weiterentwicklungen sind für zukünftige Versionen angedacht:

1. **Windows-Live-Lab über Azure Lab Services**:
   - Bereitstellung von echten, identisch konfigurierten Windows-Hosts in verschiedenen Patch-Zuständen (vor und nach CVE-2020-0601-Patch)
   - Integration einer kontrollierten Internet-Umgebung zum Testen realer TLS-Verbindungen
   - Automatisierte Provisioning-Lösung mit Infrastructure-as-Code (Terraform/ARM-Templates)
   - Zeitgesteuerte Verfügbarkeit zur Kostenoptimierung und Ressourcenschonung
   - Zentrale Überwachungsmöglichkeiten für Dozenten zur Bewertung des Lernfortschritts
2. **Automatisierte Angriffskette**:
   - Integration von Browser-Automation mit Playwright/Selenium für einen vollständigen End-to-End-Exploit
   - Demonstration der Auswirkungen auf verschiedene Browserfamilien (Chromium, Firefox, Safari)
   - Simulation eines Man-in-the-Middle-Angriffsszenarios mit TLS-Interception
   - Visualisierung der Angriffsphasen mit interaktivem Ablaufdiagramm
   - Erweiterung um zusätzliche Krypto-Angriffe (etwa ALPACA oder BEAST) für umfassendere Lernszenarien

9 Fazit

Unser Projekt demonstriert eindrucksvoll, wie sich ein kritischer kryptographischer Schwachpunkt wie CurveBall (CVE-2020-0601) in eine didaktisch wertvolle, aber dennoch sichere Lernumgebung überführen lässt. Die entwickelte Lösung schlägt erfolgreich die Brücke zwischen theoretischem Verständnis und praktischer Anwendung im Bereich der Kryptographie.

9.1 Erreichte Ziele

Die Containerisierung des Projekts ermöglicht einen niederschwelligen Zugang zur komplexen Thematik der Elliptischen-Kurven-Kryptographie und deren potentieller Schwachstellen. Dabei wurden alle initial definierten Projektziele erreicht:

- **Didaktischer Mehrwert**: Die Studierenden können den vollständigen Angriffszyklus von der theoretischen Grundlage bis zur praktischen Exploitation nachvollziehen und selbst durchführen.
- **Sicherheitskonzept**: Trotz der absichtlich integrierten kryptographischen Schwachstelle ist das Deployment selbst durch die Containerisierung und strikte Isolation inhärent sicher gestaltet.
- **Plattformunabhängigkeit**: Die Docker/Podman-basierte Lösung garantiert eine konsistente Erfahrung über verschiedene Betriebssysteme und Hardwarekonfigurationen hinweg.

9.2 Kompetenzerwerb

Neben dem technischen Verständnis für die CurveBall-Schwachstelle erwerben die Studierenden durch die Auseinandersetzung mit dem Projekt wesentliche Kompetenzen in:

- Grundlagen der Public-Key-Infrastruktur und X.509-Zertifikaten
- Praktischer Anwendung von kryptographischen Bibliotheken (OpenSSL, cryptography)
- Nutzung und Verständnis moderner DevSecOps-Workflows und CI/CD-Pipelines
- Containerisierung und Deployment von Anwendungen
- Sicherheitsrelevanten Best Practices in der Softwareentwicklung

9.3 Langfristiger Nutzen

Die entwickelte Lösung stellt nicht nur ein temporäres Lernmittel dar, sondern kann als wiederverwendbare Plattform für zukünftige kryptographische Szenarien dienen. Der modular aufgebaute Container und die umfassende Dokumentation ermöglichen eine einfache Erweiterung und Anpassung an neue Anforderungen oder andere kryptographische Schwachstellen.

Die Integration von DevSecOps-Praktiken in das Projekt selbst vermittelt den Studierenden zusätzlich einen Einblick in die heute in der Industrie etablierten Sicherheitsstandards und Workflows, was einen wertvollen Praxisbezug herstellt und die Employability der Absolventen steigert.