Enhancing Teaching Methods in FreeBSD Education through Chaos Engineering and Gamification

Andreas Kirchner, cand. M.Sc. <andreas.kirchner@h-da.de>;
Benedict Reuschling, M.Sc. <benedict.reuschling@h-da.de>

Abstract

Introduction

The increasing misuse of Artificial Intelligence (AI) in education, particularly in student work assessments, poses significant challenges for educators in evaluating students' individual contributions. To address this, the Department of Computer Science at Hochschule Darmstadt (h_da) developed and evaluated a new teaching and learning concept for the *Unix for Software Development* lecture.

Traditionally, this module's assessment relied on theoretical assignments delivered as PDF documents, focusing on the Unix distribution FreeBSD. While these exercises fostered basic proficiency, they lacked practical relevance and often did not align with real-world job requirements, such as troubleshooting, system recovery, and backup management. These essential skills were superficially covered, leaving students underprepared for professional challenges.

Aligned with h_da's mission to equip students with applicable skills for their careers, this thesis explores how Chaos Engineering and Gamification can enhance Unix education. Gamification engages students through interactive and real-world tasks, fostering deeper interest and sustained learning. Chaos Engineering introduces unpredictability to mimic real-world scenarios, compelling students to develop critical thinking and adaptive problem-solving skills.

Another focal point of this study is mitigating AI misuse in assignments. By designing tasks that are complex and context-specific, this approach discourages reliance on AI tools, encouraging students to independently identify errors and devise solutions. This not only promotes academic integrity but also nurtures original problem-solving strategies and practical knowledge application. [1]

The findings of this thesis aim to provide a teaching framework that better prepares students for professional demands while reducing fraudulent use of AI in academia. The proposed approach represents a significant step towards modernizing Unix education and aligning it with the evolving landscape of technology and workplace requirements.`

Existing Approaches

Bomb Lab at the University of Virginia

The Bomb Lab teaches debugging and system analysis by challenging students to "defuse" faulty code before it "explodes." A highscore tracks mistakes, fostering engagement while enhancing skills in assembly language, debugging, and critical thinking. Integrating Chaos Engineering could further improve adaptability to complex errors.[2]

Chaos Monkey System by SadServers

SadServers simulates realistic failure scenarios in AWS server environments, testing system resilience and problem-solving. Participants face dynamic issues, such as service outages and configuration corruption, unknown until activation. A "Check My Solution" feature verifies task completion, encouraging practical learning under pressure. [3]

Solution Description

The proposed Chaos-Education System utilizes FreeBSD jails provided to students at the beginning of the lab exercise. Students are tasked with familiarizing themselves with the system and understanding the operating system environment (login, permissions, etc).

Once all participants are ready, the instructor initiates a scenario from the controller in which intentional faults are introduced into the student-managed jails. Students must identify, analyze, and resolve the issue using standard system administration tools and even root permissions. Depending on the assignment, they may also be required to devise and implement preventive measures to avoid similar failures in the future.

This hands-on approach not only enhances students' troubleshooting and system recovery skills but also fosters a deeper understanding of proactive system management. The instructor can see how many groups have finished the scenario. A global highscore list displays the overall score each student group got for solving the assignment. The instructor can then do a group discussion of the solutions the students came up with, giving everyone the chance to see different solutions for the same problem. That increases overall knowledge of possible solutions and critical thinking.

System Design and Architecture of the Chaos Education System

The Chaos-Education system is designed to be robust, scalable, and follows the Infrastructure as Code (IaC) approach, enabling the setup and deployment of the entire system via scripts. The chaos education system is built on FreeBSD, leveraging its lightweight containerization technology, jails. jails were chosen over Docker and a Linux-based solution, as Docker was not functional on FreeBSD at the time of this work.

Overall System Architecture

The architecture employs a modular and distributed design using multiple isolated jails on a FreeBSD host system. Unlike traditional virtual machines, jails share the host kernel, reducing overhead while maintaining strong resource and process isolation. This setup supports efficient, resource-conserving operation of multiple independent environments on a single host.

jails provide an optimal foundation for scenarios requiring independent environments, such as the Chaos-Learning system. Each student or group is assigned a independent, isolated sandbox, allowing experimentation without risk to other systems or the host.[4]

jail Management and Orchestration

Management and orchestration of jails are handled via Bastille, a robust tool for FreeBSD jail administration. Bastille simplifies the creation, configuration, and resource allocation of jails, making it comparable to Docker Compose. Each jail is equipped with the necessary software to simulate specific failure scenarios. [5]

Snapshot functionality plays a critical role in the system, capturing the state of a jail at a given moment. This allows rollback of changes or restoration of original configurations after errors, which is particularly valuable in a learning environment. Also, after solving a scenario, the instructor can fall back to a clean-state system to start another scenario without having to re-deploy the jails.

Chaos Monkey Controller

At the core of the system is the Chaos Monkey Controller, which orchestrates failure scenarios such as network interruptions, CPU throttling, process crashes, and file corruption. It also hosts the text-based UI, serving as the instructor interface described earlier.[6]

The Chaos Monkey Controller operates as the host system for all jails, managing the orchestration of scenarios and providing an

intuitive interface for controlling and monitoring fault injections. This is not a hard and fast rule and future implementations may separate controller and jails to different systems.

Security Architecture

The security of the Chaos-Education system relies on the isolation capabilities of FreeBSD jails, complemented by additional measures such as firewall configurations and port control. Each jail is secured with a dedicated network interface (VNET) and custom pf firewall rules, ensuring that faults or security breaches within one jail do not affect others or the host system. It also separates the students from each other, preventing them from attacking each other's jails during a scenario to gain an unfair advantage. Regular updates and patches managed via Bastille further maintain system integrity.[7]

Fault Injection and Deployment

Fault scenarios in the Chaos-education system are injected into the jails using Bastille templates combined with shell scripts. These templates define the desired faults by the instructor, such as stopping a service, resource overloading, or configuration file manipulation. Fault templates are distributed automatically across all student jails within the student system pool.

A scripting process ensures the consistency and reproducibility of the fault configurations, keeping the injection process invisible to students. This preserves a realistic and unaltered learning environment.[8]

Additionally, a timing mechanism using Cron enables faults to be triggered either at predefined times or randomly within a specified timeframe. When needed, a one-time Cron job can deploy fault templates across all student systems with a pre-defined delay. This staggered activation introduces realistic challenges, as students must troubleshoot and resolve issues under varying and unexpected conditions, mimicking real-world scenarios.[9]

Highscore Leaderboard as a Motivational Element

Student performance is displayed on a public highscore leaderboard, designed to encourage competition and motivate participants to improve their scores over time. The leaderboard updates in real-time during scenarios, fostering a competitive environment and a sense of urgency that drives students to work efficiently and maximize their points.

This dynamic increases student engagement, prompting deeper involvement with the tasks and a stronger understanding of the material. Additionally, the leaderboard allows instructors to track individual progress, provide targeted feedback, and highlight exceptional achievements, creating further incentives for active participation.

jail setup

1) Create the controller

```
bastille create -T
"${GROUP_NAME}"
"${BOOTSTRAP_VERSION}"
"${SUBNET}100/24"
```

2) Applying the controller template:

```
bastille template
"${GROUP_NAME}"
"${ROOT_DIR}/src/bastille/te
mplates/base/student/"
```

3) Generate SSH keys

```
bastille cmd "${GROUP_NAME}"
sh -c "mkdir -p
/home/controller/.ssh &&
chmod 700
/home/controller/.ssh &&
echo \"${pubkey}\" >>
/root/.ssh/authorized_keys
&& chmod 600
/root/.ssh/authorized_keys"
```

4) Start the SSH daemon:

```
bastille cmd "${GROUP_NAME}"
service sshd restart
```

5) Create initial ZFS snapshot for scenario rollbacks:

```
zfs snapshot
"sys/bastille/jails/${GROUP_
NAME}/root@initial"
```

Overview of DNS Failure Scenario

The DNS failure scenario confronts students with realistic network scenarios requiring them to identify and resolve issues autonomously. The implementation includes key components for time tracking, scenario supervision, and the secure execution of the scenario.

Tracking Student Login Times

To calculate the time spent on each task for the highscore leaderboard, a script leveraging the `last` command tracks login and logout times for individual student users into the jail. Using the `-s` parameter, the script limits the tracking to a specified user and records time in seconds.[11]

The script distinguishes previous login times from the current session using key-value pairs, ensuring accurate highscore calculations. This is used for calculating the scores in the highscore list.

Secure Monitoring Using SSH

Initially, a file-based communication system between student systems and the controller was implemented but was abandoned due to potential security risks. Students, granted admin rights for realism, could manipulate files and gain an unfair advantage.

The revised approach centralizes monitoring through the controller, which connects to student jails every 60 seconds via SSH. The controller retrieves login times and evaluates progress using custom scripts. The tool `sshpass` enables automated password-based SSH connections which benefits from the SSH keys for enhanced security. [12]

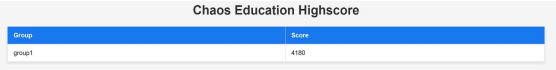
Scenario Initialization and Management

A script, `chaos_run_scenario.sh`, initializes the fault scenario by loading prepared templates into student jails. Each scenario is assigned a unique identifier based on a timestamp (e.g., `DDMMYYYYHHMM`), ensuring reproducibility and traceability. [13]

The template-based approach enables consistent fault injection across all jails, while the timestamp allows for accurate analysis and comparison of results.

Highscore Calculation and Cron Jobs

A sequential approach involving two Cron jobs ensures proper time tracking and highscore calculation. The first job executes a script to collect login times and scenario progress. The second job calculates highscores based on the collected data. This design prevents race conditions caused by simultaneous access to data files.[14]



Scoring Formula

The scoring system uses a mathematical formula to calculate points based on task difficulty, maximum achievable points, time taken, and bonuses:

$$Points = Difficulty \times Maximum\ Points \times \left(1 - \frac{Time\ Taken}{Maximum\ Time}\right) + Bonus\ \%$$

To enhance motivation, the maximum score is set at 1000 points. Studies suggest that higher

numbers in scoring systems have a stronger psychological impact, as they are perceived as more significant—a phenomenon known as the "anchoring effect." A 1000-point system also facilitates intuitive percentage-based grading, improving transparency and acceptance of the evaluation system. This contributes positively to student motivation and engagement.[10]

Scenario Termination and Restoration

When ending a scenario, the system calculates the number of students who successfully completed the task. If discrepancies exist, the instructor can extend the scenario's duration or confirm its termination. Upon confirmation, the system deletes the cron job and offers two options: a) repeat the scenario or b) restore student systems to their initial state.

For restoration, the `zfs` rollback feature is used to revert jails to their original snapshots. This ensures a consistent starting point for the next scenario and maintains system integrity.[15]

Outlook & Future Work

References

[1] Mieczyslaw Owoc, Agnieszka Sawicka und Paweł Weichbroth. “Artificial Intelligence Technologies in Education: Benefits, Challenges and Strategies of Implementation”. In: Aug. 2021, S. 37–58. isbn: 978-3-030-85000-5. doi: 10.1007/978-3-030-85001-2_4.
[2] CS3330: Bomb Lab. url: https://www.cs.virginia.edu/~cr4bd/3330/F2022/bomblab.html (besucht am 27. 08. 2024).
[3] Docs. en. url: https://docs.sadservers.com/docs/
[4] Poul-Henning Kamp und Robert N M Watson. “jails: Confining the omnipotent root.”.
[5] FreeBSD Handbook Chapter 17. jails and Containers. https://docs.freebsd.org/en/books/handbook/jails/
[6] Casey Rosenthal und Nora Jones. Chaos Engineering: System Resiliency in Practice. en. Google-Books-ID: iVjbDwAAQBAJ. O’Reilly Media, Inc.", Apr. 2020. isbn: 978-1-4920-4383-6.
[7] FreeBSD Security Information. https://www.freebsd.org/security/
[8] Mokhtar Ebrahim und Andrew Mallett. Mastering Linux Shell Scripting,: A practical guide to Linux command-line, Bash scripting, and Shell programming, 2nd Edition. en. Google-Books-ID: F99YDwAAQBAJ. Packt Publishing Ltd, Apr. 2018. isbn: 978-1-78899-015-8
[9] FreeBSD Handbook Chapter 14. Configuration, Services, Logging and Power Management. https://docs.freebsd.org/en/books/handbook/config/
[10] https://www.science.org/doi/epdf/10.1126/science.185.4157.1124.
[11] https://man.freebsd.org/cgi/man.cgi?last(1).
[12] https://ieeexplore.ieee.org/abstract/document/8763773
[13] https://books.google.de/books?hl=de&lr=&id=jUvf7wMUGcUC&oi=fnd&pg=PR11&dq=Gamma,+E.,+Helm,+R.,+Johnson,+R.,+%26+Vlissides,+J.+(1994).+Design+Patterns:+Elements+of+Reusable+Object-Oriented+Software.&ots=VP19-hTJJX&sig=NmoefVNGnjZUrBnznplgFnWTaVE#v=onepage&

q&f=false
[14]
https://books.google.de/books?hl=de&lr=&id=kCTMFpEcIOwC&oi=fnd&pg=PR9&dq=Stevens,+%2BW.%2BR.,%2BRago,%2BS.%2BA.%2B(2013).%2BAdvanced%2BProgramming%2Bin%2Bthe%2BUNIX%2BEnvironment&ots=zyGBVL_tsI&sig=Pd0j6y7DOb3WsU1lKByLY8HIDdo#v=onepage&q=Stevens%2C%20%2BW.%2BR.%2C%2BRago%2C%2BS.%2BA.%2B(2013).%2BAdvanced%2BProgramming%2Bin%2Bthe%2BUNIX%2BEnvironment&f=false
[15] zfs snapshots

Bastille Template:

```
CMD ASSUME_ALWAYS_YES=yes pkg
upgrade
PKG nano curl sudo prometheus
grafana
CMD hostname=$(hostname) && echo
$hostname | pw useradd $hostname
-m -s /bin/sh -G wheel
CMD hostname=$(hostname) && echo
$hostname | pw usermod $hostname
-h 0

CP ../../../../ssh/sshd_config
/etc/ssh/
SYSRC sshd_enable=YES
SERVICE sshd start

CMD rm
/usr/local/etc/prometheus.yml

CP
../../../../monitoring/prometheus/
targets.yaml /usr/local/etc/
CP
../../../../monitoring/prometheus/
prometheus.yml /usr/local/etc/
SYSRC prometheus_enable=YES
SERVICE prometheus start
```

```
CP
../../../../monitoring/grafana/pro
metheus-datasource.yaml
/usr/local/etc/grafana/provisionin
g/datasources/
SYSRC grafana_enable=YES
SERVICE grafana start

FSTAB
/mnt/chaos_education/shared_logs
/mnt/shared_logs nullfs ro 0 0

RDR tcp 9090 9090
RDR tcp 3000 3000
```