GymInf

Individuelles Projekt

Ein Linux-Desktop im Web: linuxcloud.ch

Alexandre & Olivier Warin

Abbildung 1: Logo [Ope21]

Abgabedatum: Dienstag, 10. Oktober 2023
Betreuungsperson: Prof. Dr. Ronny Standtke

Vorwort

In dieser Arbeit haben wir versucht, einen Linux-Desktop in einen Webbrowser zu packen. Genauer gesagt wollen wir mit diesem Projekt eine Art ≪Lernstick im Browser≫ generieren. Dieser soll im Unterricht bei Kompatibilitätsproblemen mit der heterogenen Hard- und Software in einem BYOD (Bring your own Device) Setting eingesetzt werden können. Nach diesem Prinzip soll die Idee des Lernsticks erweitert werden.

Es war uns eine grosse Ehre, von Prof. Dr. Ronny Standtke, dem Leiter der Entwicklung des Lernsticks, persönlich betreut zu werden. An dieser Stelle möchten wir uns herzlich für seine Betreuung und Unterstützung bedanken.

Ferner möchten wir uns bei den unzähligen Personen bedanken, die an der Entwicklung der in dieser Arbeit verwendeten freien Software beteiligt waren oder sind. Durch ihre engagierte Arbeit ermöglichen sie nicht nur die Realisierung eines Linux-Desktops im Web, sondern auch den Zugang zu leistungsfähigen Werkzeugen und Ressourcen für jedermann. Ihr Engagement für die Philosophie freier Software fördert die Zusammenarbeit und den Wissensaustausch auf globaler Ebene. Ohne diese enorme Vorarbeit wäre diese Arbeit bei Weitem nicht möglich gewesen.

Schliesslich hoffen wir, dass diese Arbeit dazu beitragen kann, das Verständnis und die Akzeptanz von freier Software und insbesondere von Linux-Desktops weiter zu verbessern. Wir sind davon überzeugt, dass diese Technologie das Potenzial hat, unsere Arbeitsweise zu revolutionieren und neue Horizonte zu eröffnen. Möge sie dazu beitragen, eine Welt zu gestalten, in der Freie Software und offene Standards eine zentrale Rolle spielen und jedem den Zugang zu leistungsfähigen digitalen Werkzeugen ermöglichen. Vielen Dank!

Alexandre und Olivier Warin

Bemerkung zur vorliegenden Ausgabe der Arbeit

Es handelt sich bei dieser Ausgabe um eine gegenüber der abgegebenen und bewerteten Version leicht modifizierte Variante. Das Original enthielt ein paar wenige Stellen, die nicht für eine Veröffentlichung geeignet waren.

Inhaltsverzeichnis

Kapitel 1

Die Ausgangslage

1.1 Bring Your Own Device (BYOD)

An den Schulen der Autoren (Kanton Basel-Landschaft) und an den Schulen der mit den Autoren befreundeten Lehrpersonen (Kanton Basel-Stadt) ist ein BYOD-Konzept (*Bring Your Own Device*) vorgesehen. Dabei ist zu beachten, dass BYOD in der Regel eine Sparmassnahme darstellt, indem die Kosten für die im Unterricht benötigten Geräte vom Kanton auf die Schülerinnen und Schüler bzw. deren Erziehungsberechtigte abgewälzt werden.

Natürlich hat ein BYOD-Konzept Vorteile. Vor allem können alle Beteiligten immer in ihrer gewohnten Umgebung arbeiten.

Leider gibt es auch einige Probleme im Unterrichtsalltag. So stehen in einigen Klassenzimmern so wenige Steckdosen für die Schülerinnen und Schüler zur Verfügung, dass es nach einiger Zeit zu Problemen bei der Energieversorgung der Geräte kommt. Zu empfehlen, morgens immer mit ≪vollgeladenem Akku≫ in die Schule zu kommen, wird hier nur bedingt helfen, da Akkus bekanntlich Verschleissteile sind, was sich nach bisherigen Beobachtungen bei vielen Schülergeräten schon früh bemerkbar macht. Viele dieser Geräte halten nach wenigen Monaten keinen ganzen Schultag mehr durch. Verlängerungskabel und Multisteckdosen sind inzwischen fester Bestandteil des Inhaltes vieler Rucksäcke der Schülerinnen und Schüler. Aber auch so reichen bei mittelgrossen bis grossen Klassen die verfügbaren Steckdosen oft nicht aus.

Während diese Arbeit nicht zur Lösung des Stromversorgungsproblems beitragen kann, soll ein zweites Problem angegangen werden: Die Schülerinnen und Schüler kommen mit so unterschiedlichen Geräten in den Unterricht, dass es sich in der Praxis als unmöglich erwiesen hat, von selbst minimalen Softwareanforderungen auszugehen. Dabei geht es nicht nur um bestimmte, meist unfreie Softwarepakete, die nicht auf allen Geräten installiert werden können, aber von einigen Leuten als zwingend zu installieren gefordert werden, sondern auch um Kleinigkeiten: Einige Windowsgeräte scheinen Verbindungen auf externe Server über das Secure-Shell-Protokoll (SSH) auf der installierten Powershell nicht ohne Weiteres zu unterstützen. Andere Geräte haben sogar Schwierigkeiten, eine im Browser heruntergeladene HTML-Datei in einem

anderen Programm zu bearbeiten und dann lokal im Browser anzuzeigen.

1.2 Freie Software

Der Begriff «Freie Software», bezieht sich auf Software, die Benutzern bestimmte Freiheiten gewährt. Diese Freiheiten beinhalten in der Regel das Recht, die Software auszuführen, zu untersuchen, zu modifizieren und zu verteilen. Im Wesentlichen geht es bei freier Software darum, den Benutzern die Kontrolle über ihre eigenen Computer und die von ihnen verwendete Software zu ermöglichen.

Abbildung 1.1: Logo der Free Software Foundation [Jia19]

Ein wichtiger Aspekt freier Software ist das Konzept des öffentlich verfügbaren Quellcodes. Im Gegensatz zu proprietärer Software, bei der der Quellcode meistens geheim gehalten wird, erlaubt freie Software den Benutzern, den Quellcode einzusehen und zu modifizieren. Dies ermöglicht es den Benutzern, die Software an ihre individuellen Bedürfnisse anzupassen und Fehler zu beheben. Ferner können Benutzer freie Software kopieren und weitergeben, was eine breite Verfügbarkeit und eine ständige Weiterentwicklung fördern kann.

Die Free Software Foundation (FSF) definiert vier Grundfreiheiten für Freie Software [Fou23c]:

0. Die Freiheit, das Programm für jeden Zweck auszuführen.
1. Die Freiheit, den Quellcode zu untersuchen und zu verändern.
2. Die Freiheit, Kopien der Software zu verbreiten.
3. Die Freiheit, modifizierte Versionen zu veröffentlichen und weiterzugeben.

1.2.1 Freie Software an Schulen

Richard Stallman [Fou23b] ist der Gründer der Free Software Foundation [Fou23a] und ein bekannter Aktivist. Seine Ideen im Bezug auf den Einsatz freier Software an Schulen sind eng mit seiner Philosophie der Freien Softwarebewegung verbunden [Fou22].

Abbildung 1.2: Richard Stallman (∗1953) [Com19]

Freie Software als Bildungsressource Stallman glaubt, dass der Einsatz von Freier Software in Schulen es Schülerinnen und Schülern ermöglicht, nicht nur technische Fähigkeiten zu erlernen, sondern auch ein Verständnis für die zugrundeliegenden Prinzipien von Freiheit, Zusammenarbeit und Transparenz zu entwickeln. Er argumentiert, dass der Zugang zum Quellcode und die Möglichkeit, Software an die eigenen Bedürfnisse anzupassen, den Lernprozess verbessern und kritisches Denken fördern können.

Unabhängigkeit von proprietärer Software Stallman betont die Wichtigkeit, Schulen nicht von proprietärer Software abhängig zu machen. Er argumentiert, dass der Einsatz proprietärer Software den Schulen die Kontrolle über ihre eigenen Systeme nimmt und sie von kommerziellen Anbietern abhängig macht. Durch den Einsatz freier Software können Schulen ihre eigene technologische Infrastruktur kontrollieren und anpassen, ohne von externen Unternehmen abhängig zu sein.

Zusammenarbeit und Gemeinschaft Stallman betont, dass Freie Software auf den Prinzipien der Zusammenarbeit und des Teilens basiert. Durch den Einsatz freier Software in Schulen können Lehrnende und Lehrpersonen gemeinsam an Projekten arbeiten, Wissen austauschen und voneinander lernen. Freie Software fördert eine Kultur der Zusammenarbeit und stärkt die Gemeinschaft innerhalb der Schule.

Kostenersparnis und Nachhaltigkeit Stallman argumentiert, dass der Einsatz Freier Software in Schulen zu erheblichen Kostenersparnissen führen kann. Da solche Software frei verfügbar ist, entfallen die Lizenzkosten für proprietäre Software. Ferner führt die Möglichkeit, Software auf älterer Hardware einzusetzen, zu einer längeren Nutzungsdauer vorhandenen Geräte, was mit einer nachhaltigeren Nutzung von Ressourcen einhergeht.

Es ist wichtig anzumerken, dass die Ideen von Richard Stallman nicht unumstritten sind und es verschiedene Ansichten zu diesem Thema gibt. Zum Beispiel argumentieren einige Kritiker, dass der Einsatz von freier Software in Schulen technische Einschränkungen mit sich bringen könnte oder dass der Fokus auf freie Software andere wichtige Aspekte der Bildung vernachlässigen könnte. Es ist ratsam, verschiedene Perspektiven zu berücksichtigen und fundierte Entscheidungen zu treffen, die den spezifischen Bedürfnissen

einer Schule gerecht werden.

Die Autoren dieser Arbeit sind überzeugt, dass es sinnvoll ist, immer dann auf Freie Software zurückzugreifen, wenn dies mit vertretbarem Aufwand möglich ist.

Mit einer rein proprietären Lösung wäre diese Arbeit auf jeden Fall in der vorliegenden Form nicht umsetzbar gewesen.

Kapitel 2

Zielsetzungen

2.1 Idee

Die bei einem *BYOD*-Konzept zwangsläufig auftretende Heterogenität der Hardware, aber auch der Software auf Betriebssystemebene, führt in einem schulischen Umfeld, in dem viel kollaborativ gearbeitet wird, zwangsläufig zu den bereits genannten Problemen. Es sollte daher nach einer Alternative gesucht werden, die es ermöglicht, unabhängig von Hardware und Betriebssystem auf allen Geräten die gleiche Arbeitsumgebung zu nutzen.

So entstand die Projektidee, den Schülerinnen und Schülern einen auf einem externen Server installierten Linux-Desktop im Browser zur Verfügung zu stellen. Auf diesem Linux-System können dann alle von den Fachschaften gewünschten Programme installiert und entsprechend im Unterricht eingesetzt werden — unabhängig davon, ob es sich bei den konkreten Geräten der Schülerinnen und Schüler um Windows-Notebooks, Linux-Notebooks, MacBooks, iPads oder Android-Tablets handelt.

2.2 Features

Folgende Features wurden während der Konzeptionierungsphase im Sinne von Maximalzielen als wünschenswert identifiziert:

Systemvoraussetzungen Die Lösung sollte mit einem Standard-Browser benutzbar sein.

Persistence Persönliche Arbeitsdokumente und Einstellungen sollen nach dem Abmelden erhalten bleiben.

Easy Deployability Der Aufwand, die Lösung auf einem anderen Server für eine weitere Schule zu installieren, soll möglichst gering sein.

Privacy by Design Personenbezogene Daten werden ausschliesslich verschlüsselt gespeichert. Niemand ausser den jeweiligen Benutzern soll darauf Zugriff haben.

Security Die Lösung soll sicher sein: Weder soll der Server über diese Cloud-Lösung kompromittierbar sein, noch sollen einzelne Nutzer auf die Accounts anderer Nutzer zugreifen können.

Skalierbarkeit Die Lösung soll so ausgelegt sein, dass viele Nutzerinnen und Nutzer das System gleichzeitig nutzen können. Gegebenenfalls ist ein Loadbalancing vorzusehen, so dass nach dem dem Prinzip der Easy Deployability einfach weitere Server installiert und direkt genutzt werden können.

FOSS Es soll möglichst ausschliesslich freie Software zum Einsatz kommen.

Im Folgenden soll nun zu den einzelnen Features etwas detaillierter eingegangen werden.

2.3 Systemvoraussetzungen

Jede Lösung, die es erforderlich macht, dass die Schülerinnen und Schüler irgendein Client Tool (wie zum Beispiel einen VNC Viewer) auf ihren Geräten installieren müssen, führt die ganze Idee dieser Arbeit ad absurdum. Es geht ja gerade darum, dass nicht jedes Gerät jede Software unterstützt. Daher ist dieses Ziel absolut wichtig. Am Ende muss eine Variante stehen, bei der eine wirklich überall vorinstallierte Software — in diesem Fall ein HTML5-fähiger Browser — die einzige Systemvoraussetzung auf Clientseite darstellt.

Deswegen wird *Guacamole* [Fou23d] verwendet. Guacamole ist ein freier Remote Desktop Gateway, der es ermöglicht, Remote Desktop Sessions im Browser anzuzeigen und zu bedienen.

Einschätzung bei Projektbeginn: Dieses Ziel ist mit hoher Wahrscheinlichkeit erreichbar.

2.4 Persistenz

Eigentlich ist die Persistenz eine Selbstverständlichkeit, die ein sinnvolles Arbeiten über viele Schulstunden hinweg überhaupt erst ermöglicht. Andere Funktionen können jedoch zu Problemen führen: Beispielsweise müssen die Daten beim Ausloggen irgendwie nur verschlüsselt vorliegen und erst beim erneuten Einloggen wieder entschlüsselt werden.

Hier ergibt sich eine weitere technische Schwierigkeit: Der verfügbare Speicherplatz auf bezahlbaren Servern ist eher begrenzt. Es kann also sein, dass die Userhomes irgendwie auf externe Cloud-Speicher ausgelagert werden müssen, natürlich in verschlüsselter Form.

Einschätzung zu Projektbeginn: Dieses Ziel ist wahrscheinlich erreichbar, wobei möglicherweise Kompromisse notwendig sind.

2.5 Easy Deployability

Dieser Aspekt hat eigentlich keine hohe Priorität, würde aber zu einer höheren Akzeptanz führen, so dass die hier vorgestellte Lösung etwas breiter eingesetzt werden könnte, mit dem Nebeneffekt, dass Linux-Desktops etwas bekannter werden und so Hemmschwellen gegenüber einem freien Betriebssystem abgebaut werden können.

Dabei sind zwei Aspekte zu berücksichtigen:

Serverinstallation Es sollte keine komplizierte Installation notwendig sein. Eine mögliche Lösungsvariante wäre, einen grossen Container (Docker, Podman oder auch LXC) anzubieten, den man nur auf einem Server importieren muss und dann mit minimalem Konfigurationsaufwand direkt loslegen kann.

Account-Einrichtung Ideal wäre es, wenn die bestehenden Schul-Accounts direkt weiter verwendet werden könnten. LDAP könnte hier eine Möglichkeit bieten. Allerdings ist das Active Directory des Kantons, in dem die Autoren arbeiten, nur schwer durchschaubar, so dass es kaum verwendet werden kann. Zudem verwenden andere Schulen möglicherweise ganz andere Systeme, um ihre Zugänge zu verwalten. Es ist also eher eine Lösung geplant, bei der die Einrichtung der Konten so einfach wie möglich ist.

Einschätzung zu Beginn des Projekts: Dieses Ziel scheint mit gewissen Kompromissen bei der Kontenverwaltung erreichbar zu sein.

2.6 Privacy by Design

In Hinblick des heute vorherrschenden Zwangs, in dieser Hinsicht fragwürdige Software wie zum Beispiel Microsoft Teams [Mic] zu verwenden, ist ≪Privacy by Design≫ sicherlich ein erstrebenswertes Ziel. Selbstverständlich sollen Bemühungen in diese Richtung unternommen werden: Linux Homeverzeichnisse lassen sich eigentlich recht gut gegen fremde Augen schützen und es gibt auch einige Optionen im Bereich der Kryptographie.

Auf der anderen Seite ist es leider so, dass zumindest die berechtigten Serveradministratoren ohnehin immer alles sehen können. Auch Verschlüsselungen können, zum Beispiel durch den Einsatz eines Keyloggers in einem PAM-Modul, letztlich für den root-Benutzer leicht umgangen werden. Hier ist man also letztlich auf die Diskretion des Systemadministrators angewiesen. Ausserdem ist es immer denkbar, dass ein Benutzer irgendeine Form von Privilege Escalation durchführt und sich dann selbst in der Rolle des Root-Users wiederfindet.

Insgesamt werden also sicherlich einige, wahrscheinlich schmerzhafte Kompromisse eingegangen werden müssen. Die Autoren dieser Arbeit werden sich bemühen, diese Kompromisse, soweit möglich und sinnvoll, nicht zu sehr zu Lasten des Datenschutzes reifen zu lassen. Letztlich wird man aber den Schulen, die diese Lösung einsetzen, raten müssen, am besten von vornherein keine sensiblen Daten auf dem System speichern zu lassen.

Einschätzung zu Projektbeginn: Die Umsetzung ist wohl nur zur Hälfte möglich. Kompromisse sind unumgänglich.

2.7 Sicherheit

Der Wunsch, dass der Server dieser Cloud-Lösung nicht kompromittiert werden kann, ist wohl genau das: Wunschdenken. Grundsätzlich sollen auf einem Server Linux-Desktops laufen, was natürlich impliziert, dass die Benutzerinnen und Benutzer zwangsläufig auch Code auf dem Server ausführen können.

Diesem Problem soll durch eine möglichst gute Isolierung zum Beispiel durch virtuelle Maschinen oder durch Container entgegengewirkt werden. Dazu muss recherchiert werden, inwieweit ein bezahlbarer Server überhaupt virtuelle Maschinen zulässt (denn wenn keine Hardwarebeschleunigung möglich ist, ist dieser Weg eigentlich zum Scheitern verurteilt) und welche Container unsicher sind oder gar bekannte Escape-Exploits besitzen. So ist ein Podman-Container, der grundsätzlich unprivilegiert läuft, einem klassischen Docker-Container, für den bereits Escape-Exploits [Sec21] bekannt sind, sicherlich vorzuziehen. Und im Falle von LXC sollten natürlich nur unprivilegierte Container verwendet werden.

Es besteht auch der Wunsch, gegen Ende des Projektes einen kleinen ≪Penetrationstest≫ durchzuführen. Dazu müssen aber zu diesem Zeitpunkt entsprechende Angriffsideen vorhanden sein. Dies ist mental nicht ganz einfach, da bis zu diesem Zeitpunkt viel Denkarbeit in Richtung Absicherung (≪Blue Team≫) geleistet worden sein wird. Und das Budget lässt es leider nicht zu, einen professionellen Pentester zu engagieren.

Einschätzung zu Projektbeginn: Dieses Ziel ist im Sinne von Good Practice bzw. Best Effort mit dem einen oder anderen Kompromiss erreichbar. Offenbar ist das Ziel aber im Sinne von perfekter Sicherheit unmöglich zu erfüllen.

2.8 Skalierbarkeit

Im Grunde sollte es über einen entsprechenden Load-Balancing-Proxy gut möglich sein, das System zu skalieren. Eine Voraussetzung dazu ist es aber, dass die Nutzerkonten und Nutzerdaten auf irgendeine Weise über alle Instanzen synchronisiert werden können. Das kann durch gegenseitiges Updaten passieren oder indem diese Dinge auf einen externen Server ausgelagert werden.

Im Falle der Accounts scheint ein gegenseitiges Update derzeit einfacher umsetzbar, da Accountmanagementsysteme wie Active Directory doch recht aufwendig in der Einrichtung und Pflege sind.

Im Falle der Benutzerdaten (Homeverzeichnisse) ist aufgrund der Speicherbegrenzung ohnehin eine Auslagerung auf bezahlbare Server geplant.

Grundsätzlich erscheint es möglich, Skalierbarkeit zu gewährleisten. Es fehlen jedoch die notwendigen Ressourcen: Um die Skalierbarkeit (und deren Grenzen) innerhalb eines einzelnen Servers zu testen, müsste das System über einen längeren Zeitraum an mindestens einer grossen Schule getestet werden können, wofür die Zeit vermutlich zu knapp bemessen ist. Und um die Skalierbarkeit über mehrere Server zu testen, müssten mehrere Server gemietet werden. Hier besteht ein Finanzierungsproblem.

Einschätzung zu Projektbeginn: Dieses Ziel ist wahrscheinlich zumindest theoretisch erreichbar. Allerdings ist die praktische Umsetzung im Rahmen dieser Arbeit nicht testbar.

2.9 FOSS

Es ist sogar die Überzeugung der Autoren, dass eine Umsetzung mit unfreier Software wesentlich komplizierter und schwieriger wäre. Freie Software bietet hier deutliche Vorteile zum Beispiel in der Anpassbarkeit.

Eine Ausnahme bleibt das proprietäre RDP-Protokoll, das aus Performancegründen wahrscheinlich zum Einsatz kommen wird. Freie Alternativen wie SPICE [spib] oder x2go [ste23] sehen zwar interessant aus, scheinen aber zum einen nicht so effizient und zum anderen nicht so einfach im Browser einsetzbar zu sein.

Einschätzung zu Projektbeginn: Dies ist mit wenigen Ausnahmen, wie RDP, wahrscheindlich vollständig realisierbar.

Kapitel 3

Bisherige Lösungen

In diesem Kapitel sollen kurz einige Lösungsansätze und Ideen vorgestellt werden, auf die die Autoren dieser Arbeit in der Zwischenzeit gestossen sind.

3.1 Verbindliche Standardgeräte

Einige Kolleginnen und Kollegen sprechen sich für die Einführung eines obligatorischen Gerätes aus. Für den Einsatz im BYOD-Umfeld soll also genau ein Modell eines bestimmten Herstellers angeschafft werden und keine alternativen Geräte und auch keine alternativen Betriebssysteme zugelassen werden. Es wird erwartet, dass der Umfang der installierbaren Software dann überall gleich ist — schliesslich handle es sich ja dann um die jeweils identische Hardware.

Den Autoren ist keine Schule bekannt, die diesen Ansatz verfolgt. Im Kanton Basel-Stadt gibt es allerdings recht strenge Vorgaben[Kan22] an die zu verwendenden Geräte, die in der Praxis einem verbindlichen Standardgerät recht nahe kommen, da es nur wenige Modelle gibt, die alle Anforderungen in der festgeschriebenen Form erfüllen können.

Die Autoren dieser Arbeit lehnen die Idee, ein verbindliches Standardgerät einzusetzen, als untauglichen Ansatz ab. Dies aus folgenden Gründen:

BOMD Ein solches Konzept widerspricht dem BYOD-Konzept — *Bring Your Own Device* — und ersetzt es durch ein BOMD-Konzept: *Buy Our Mandatory Device*. Der positive Aspekt von BYOD, dass jeder in seiner gewohnten Umgebung arbeiten kann, geht dabei völlig verloren. Zudem setzt ein BOMD-Konzept insbesondere finanzschwache Familien unter Druck, da die Möglichkeit, ein etwas einfacheres, aber dafür günstigeres (oder allenfalls gebrauchtes) Gerät zu kaufen, nicht mehr oder nur noch sehr eingeschränkt gegeben ist.

Pädagogische Aspekte Eine der Aufgaben der Schule und insbesondere der Lehrerinnen und Lehrer ist es, die Schülerinnen und Schüler zum selbstständigen Denken anzuregen, damit sie in ihrem späteren Leben freie und informierte Entscheidungen treffen können. Viele Menschen scheinen im IT-Bereich keine freien Entscheidungen treffen zu können, weil sie zum Beispiel immer nur die gleiche Software

gesehen und benutzt haben. Solche Zusammenhänge führen, so die Vermutung der Autoren dieser Arbeit, zu einem Teufelskreis: Bestimmte Software wird als ≪Standard≫ angepriesen, weil man nur diese Software kennt und dieses Halbwissen auch so weitergibt, so dass die nächste Generation wiederum nur diese eine Software kennt und daher wiederum als ≪Standard≫ anpreist. Das Ergebnis ist ein erheblicher Freiheitsverlust. Diesen Teufelskreis zu durchbrechen sollte nach Ansicht der Autoren ein zentraler Bestandteil der gymnasialen Ausbildung sein. In einem BOMD-Szenario agieren Lehrpersonen jedoch zwangsläufig in mehr oder weniger ausgeprägtem Masse als ≪Werbeverkäufer für eine bestimmte Soft- oder Hardwarefirma≫. Dies ist ein moralisches Problem, das die Autoren dieser Arbeit nicht einfach hinnehmen wollen und können.

Technische Aspekte Die Erwartung, dass jede Software auf ≪baugleichen≫ Geräten überall gleich gut läuft, ist ein Trugschluss. Sobald ein Gerät benutzt wird, unterscheidet es sich vom nächsten, theoretisch ≪baugleichen≫ Gerät. Aspekte wie persönlich gewählte Betriebssystemeinstellungen, frei verfügbarer Speicherplatz, Versionen einzelner Komponenten, Hardwarefehler oder auch unterschiedliche Bedienung führen dazu, dass auch in einem solchen Szenario nicht immer alles überall gleich gut genutzt werden kann. So ist es in der Erfahrung der Autoren schon vorgekommen, dass ein Programm auf dem Gerät von Schüler A problemlos funktioniert hat, auf dem baugleichen Modell von Schüler B aber überhaupt nicht.

Vendor Lock-in Beim Einsatz von ≪Standardgeräten≫ besteht eine erhebliche Gefahr, dass sich die Schule nach kurzer Zeit in einem Vendor Lock-in-Szenario wiederfindet: Es könnte schwierig werden, den ≪Standard≫ zu einem späteren Zeitpunkt wieder zu wechseln, zum Beispiel wenn der Hersteller die Preise zu stark erhöht oder das (Software-)Produkt ganz einstellt. In einem solchen Fall könnten interne Dokumente, in die unter Umständen viele Arbeitsstunden geflossen sind, plötzlich nicht mehr verfügbar sein. Dieses Szenario erscheint durchaus realistisch, da die Strategie des ≪Vendor Lock-in≫ durchaus Teil der Gesamtunternehmensstrategie diverser grosser Softwarekonzerne zu sein scheint.

Dieses Problem wäre weniger gravierend, wenn auf dem gewählten Standardgerät möglichst nur Freie Software eingesetzt würde. Dies ist derzeit leider kaum realistisch. Auch in einem solchen Fall ist es wichtig, Alternativen aufzuzeigen. Denn die Freiheit der Softwarewahl ist ein Kernelement der Freie-Software-Bewegung.

FOSS an Schulen An dieser Stelle sei noch auf weitere Argumente von Richard Stallman (Unterabschnitt 1.2.1) zur Frage verwiesen, warum Schulen möglichst ausschliesslich auf Freie Software setzen sollten.

Ökologische Aspekte Da in einem BOMD-Szenario das ≪Standardgerät≫ mit hoher Wahrscheinlichkeit in regelmässigen Abständen neu definiert werden muss (schliesslich sollen stets aktuelle Versionen verwendeter Software eingesetzt werden können), können Altgeräte nur bedingt an jüngere Geschwister weitergegeben werden, wenn diese später ebenfalls in die Schule kommen. Die bisher verwendeten Geräte sind ab diesem Zeitpunkt ≪wertlos≫ und werden zumindest teilweise entsorgt, obwohl sie eigentlich noch problemlos hätten weiterverwendet werden können.

Abbildung 3.1: VirtualBox Logo [Ora]

Abbildung 3.2: VMware Logo [vmwa]

3.2 Virtuelle Maschinen

Virtuelle Maschinen sind rein softwarebasierte Computer, die innerhalb eines Hostrechners laufen. Auf diese Weise ist es möglich, innerhalb eines Betriebssystems (bzw. innerhalb einer Rechnerarchitektur) einen virtuellen Computer mit komplett eigener (simulierter) Hard- und Software zu betreiben. Typische Anwendungsbeispiele sind Benutzerinnen und Benutzer von Windows-Maschinen, die eine Linux-Distribution ausprobieren möchten, oder Linux-Benutzerinnen und -Benutzer, die kurzzeitig ein ausschliesslich in Windows verfügbares Programm verwenden müssen.

Es gibt verschiedene Programme, die virtuelle Maschinen ermöglichen:

3.2.1 VirtualBox und VMware

VirtualBox [Ora23b] und VMware [vmwb] sind sogenannte Typ-2-Hypervisoren, die die Simulation einer virtuellen Maschine innerhalb eines vollwertigen Betriebssystems (Host) ermöglichen. VirtualBox ist sowohl in einer freien als auch in einer proprietären Version verfügbar, während VMware ausschliesslich unter einer proprietären Lizenz vertrieben wird.

Auch wenn die Installation von VirtualBox in der Regel sehr einfach funktioniert, sind die Autoren dieser Arbeit im Schulbetrieb bereits auf auf grössere Schwierigkeiten im Schulbetrieb gestossen:

- Der Wunsch, VirtualBox auf den Schulgeräten zu installieren, wurde von der Informatik des Arbeitgeberkantons mit dem Hinweis auf zu wenig Speicherplatz abgelehnt.
- Auf neueren Mac-Systemen (seit den M1-Prozessoren) ist VirtualBox derzeit nicht oder nur in einer eingeschränkten Beta-Version lauffähig[1].
- Auf neueren Windows-Geräten der Schülerinnen und Schüler konnte VirtualBox ebenfalls nicht installiert werden. Es gibt nämlich inzwischen auch Windows-Geräte mit einem Prozessor der ARM-Architektur [Mic23], die von VirtualBox nicht unterstützt wird.
- Für Schülerinnen und Schüler mit iPads steht diese Option ohnehin nicht zur Verfügung.

3.2.2 QEMU/KVM

QEMU (Quick Emulator) [qem] ist eine freie Virtualisierungssoftware, die mit Hilfe von KVM (Kernel-based virtual Machine) [lina] einen Typ-1-Hypervisor darstellt. Genau genommen ist KVM formal ein

[1] Developer preview for macOS / Arm64 (M1/M2) hosts [Ora23a]

Typ-2-Hypervisor, aber viele Meinungen gehen in die Richtung, dass KVM als Typ-1-Hypervisor bezeichnet werden kann. Diese Debatte ist jedoch nicht Teil dieser Arbeit.

Ein Typ-1-Hypervisor ist in der Lage, direkt auf der Hardware des Host-Rechners zu laufen, unabhängig vom dort installierten Betriebssystem. Daher ist es hier auch möglich, einzelne (physische) Hardwarekomponenten wie zum Beispiel die Grafikkarte vollständig unter die Kontrolle der virtuellen Maschine zu stellen (pass-through)[Ban21].

QEMU/KVM ist für die Schulpraxis ähnlich interessant wie VirtualBox, leidet aber auch mehr oder weniger unter den gleichen Schwierigkeiten und Problemen. Die theoretische Möglichkeit, QEMU ohne KVM laufen zu lassen, erlaubt es jedoch, einfache virtuelle Maschinen ohne weitreichende Systemberechtigungen (≪Admin-Rechte≫) zum Beispiel auf Schulrechnern einzusetzen. Da dann aber die Hardwarebeschleunigung fehlt, ist dies allenfalls als Proof-of-Concept oder für reine Konsolenanwendungen wirklich brauchbar.

3.2.3 Windows Subsystem für Linux

Das Windows Subsystem für Linux ist ein in Windows eingebauter Kompatibilitätslayer, der Linux-Anwendungen und sogar ganze Linux-Distributionen unter Windows nutzbar werden lässt. Seit Version 2 wird dazu der Hypervisor *Hyper-V* verwendet, wodurch das Windows Subsystem for Linux gewissermassen als virtuelle Linux-Maschine innerhalb von Windows betrachtet werden kann [Mic22b].

Der Ansatz ist interessant und macht Windows-Rechner sicherlich vielseitiger nutzbar. Leider profitieren in einer BYOD-Umgebung nur die Schülerinnen und Schüler mit einem Windows-Rechner davon. Und die IT des Arbeitgeberkantons möchte das System auf den Schulgeräten nicht aktivieren.

3.3 Live-Systeme

Live-Linux-Distributionen sind Betriebssysteme, die von einem externen Speichermedium wie einer DVD oder einem USB-Stick gebootet werden können, ohne dass eine Installation auf der Festplatte erforderlich wird. Sie ermöglichen es, ein voll funktionsfähiges Linux-Betriebssystem zu nutzen, ohne die bestehende Betriebssysteminstallation zu beeinträchtigen. Dazu wird die vorhandene Hardware während des Bootvorgangs erkannt und das System entsprechend konfiguriert.

Live-Distributionen befinden sich dabei ausschliesslich im flüchtigen Arbeitsspeicher des Rechners, so dass keine dauerhaften Änderungen vorgenommen werden. Wird der Datenträger entfernt und der Rechner neu gestartet, findet sich dieser im ursprünglichen Zustand wieder und alle Spuren der Live-Distribution sind wieder verschwunden.

Um dennoch Einstellungen und Arbeitsdokumente im Sinne der Persistenz zu erhalten, bieten die meisten Live-Distributionen die Möglichkeit, auf Wunsch eine spezielle Persistenz-Partition auf einem Datenträger wie der Festplatte des Computers einzurichten. Damit verliert man aber natürlich den Aspekt, dass das System unverändert bleibt.

Live-Linux-Distributionen werden für verschiedene Zwecke eingesetzt:

Systemdiagnose und -wiederherstellung: Live-Linux-Distributionen enthalten häufig eine Vielzahl

von Diagnose- und Wiederherstellungswerkzeugen, mit denen Benutzerinnen und Benutzer beschädigte Dateisysteme überprüfen, Viren entfernen, Daten wiederherstellen und andere Probleme auf dem System beheben können.

Sicherheit und Penetration Testing: Live-Linux-Distributionen wie Kali Linux [Lim23] sind bei Sicherheitsexpertinnen und -experten, ethischen Hackerinnen und Hackern, aber leider auch bei Script-Kiddies beliebt. Sie bieten eine umfangreiche Sammlung von Tools für Netzwerkanalyse, Schwachstellenanalyse und Penetrationstests. Da solche spezialisierten Distributionen aus verschiedenen Gründen nicht zum Einsatz im alltäglichen, regelmässigen Gebrauch geeignet sind, sind Lösungen, bei denen zum Beispiel Kali Linux als Live-System nur bei Bedarf gebootet wird, sehr praktisch und beliebt.

Privatsphäre und Anonymität: Einige Live-Linux-Distributionen wie Tails [tai23] sind darauf ausgelegt, die Privatsphäre und Anonymität der Benutzer zu schützen. Sie leiten den gesamten Internetverkehr über das TOR-Netzwerk [tor] und enthalten Verschlüsselungswerkzeuge zum Schutz sensibler Daten.

Testen von Linux-Distributionen: Live-Linux-Distributionen ermöglichen es Benutzerinnen und Benutzern, verschiedene Linux-Distributionen auszuprobieren, bevor sie sich für eine Installation entscheiden. Auf diese Weise können sie die Benutzeroberfläche und die Software testen. Zwar sind heute Hypervisoren wie VirtualBox weit verbreitet, die ähnliche Möglichkeiten zum risikolosen Ausprobieren einer Linux-Distribution bieten, jedoch haben Live-Distributionen hier einen entscheidenden Vorteil gegenüber virtuellen Maschinen: Es kann auch die Kompatibilität mit der physischen Hardware getestet werden.

Viele moderne Distributionen, wie zum Beispiel Manjaro Linux [KG23] und viele andere, enthalten heute auf ihren Installationsmedien eine Live-Variante ihrer selbst. Damit ist es möglich, die Distribution als Live-System gefahrlos zu testen und nur bei Gefallen direkt aus dem Live-System heraus auf dem Rechner fest zu installieren.

Bildung und Schule: Live-Linux-Distributionen werden auch in Bildungseinrichtungen eingesetzt, um Schülerinnen und Schülern praktische Erfahrungen mit Linux zu ermöglichen, ohne dass sie ihre eigenen Computer modifizieren müssen.

In der Schulpraxis mit dem BYOD-Modell hat sich leider gezeigt, dass Live-Distributionen nicht überall eingesetzt werden können. Mac-Geräte werden in der Regel nur dann unterstützt, wenn sie über Intel-Prozessoren verfügen und auch Windows-Geräte können hier für Enttäuschung sorgen, beispielsweise wenn es sich um eine ARM-Architektur handelt. Und bei iPad-Geräten scheint eine solche Lösung leider völlig undenkbar.

Leider ist es sogar so, dass selbst auf teilweise vorhandenen Schulgeräten solche Live-Systeme in der Regel nicht gebootet werden können, da die nötigen Keys für Secureboot fehlen und zudem das UEFI-Setup passwortgeschützt ist. Daher müsste die kantonale IT die Möglichkeit, von USB zu booten, explizit freischalten. Dies wird aber kaum geschehen, da diese Sperre als Sicherheitsmassnahme begründet wird.

Es gibt viel zu viele Live-Distributionen, um auf alle einzugehen, ohne den Rahmen dieser Arbeit bei Weitem

zu sprengen. Aufgrund dessen wird hier nur exemplarisch auf zwei bekannte Live-Linux-Distributionen eingegangen.

3.3.1 KNOPPIX

KNOPPIX [Kno] ist eine Debian-basierte Live-Distribution, die von Klaus Knopper entwickelt wurde. Auch wenn KNOPPIX historisch gesehen nicht die erste Live-Distribution war, so ist es doch die erste Live-Distribution, die den Autoren dieser Arbeit in ihrer mittlerweile jahrzehntelangen Linux-Erfahrung begegnet ist. In den späten Neunzigern und frühen Nullerjahren war es noch üblich, dass relativ viel Handarbeit nötig war, um alle Hardware-Features des Rechners auszunutzen und das System gut zu konfigurieren. Umso beeindruckender war die schon damals hervorragend funktionierende automatische Hardwareerkennung während des Bootvorgangs von KNOPPIX.

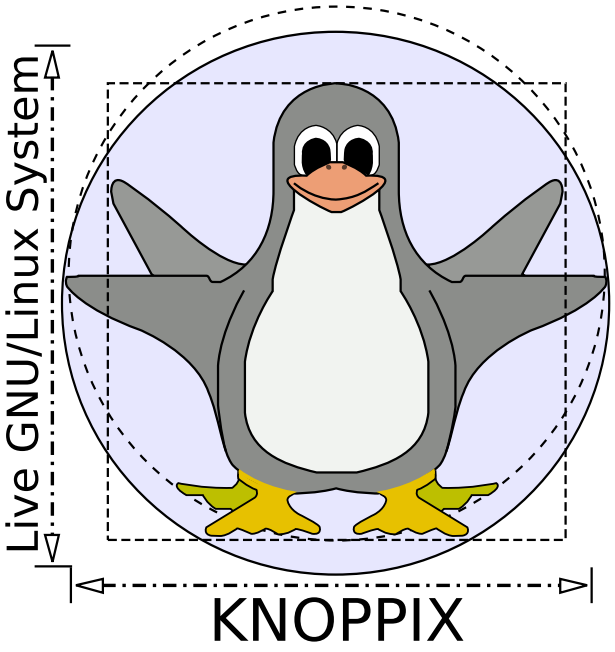

Abbildung 3.3: KNOPPIX Logo [Com07]

Basierend auf KNOPPIX und seiner hervorragenden Hardwareerkennung gibt es mittlerweile eine ganze Reihe von Derivaten. Ein Beispiel ist Desinfec't [Sch11]. Diese Distribution, die früher unter dem Namen «Knoppicillin» bekannt war, erlaubt es, ein mit Viren, aber auch durch Rootkits verseuchtes Windows-System aus einer sicheren, «neutralen» Umgebung heraus zu säubern. Seit der Umbenennung von Knoppicilin in Desinfec't im Jahr 2010 basiert diese Distribution nicht mehr auf KNOPPIX, sondern auf einem Live-System von Fedora. Aus historischen Gründen soll diese sehr nützliche Distribution hier dennoch als KNOPPIX-Derivat erwähnt werden.

3.3.2 Lernstick

Der Lernstick [Fac23] ist eine Live-Distribution, die eine mobile Lern- und Arbeitsumgebung für den schulischen Kontext bietet.

Abbildung 3.4: Logo von Lernstick EDU [Ler]

Die Entwicklung des Lernsticks, der als Orientierungshilfe für das System dieses Projektes dient, wird vom Betreuer dieser Arbeit, Prof. Dr. Ronny Standtke, geleitet[Ber23].

Es gibt mehrere Varianten des Lernsticks. Diese Arbeit konzentriert sich auf die Variante ≪Lernstick EDU≫. Es wird nicht versucht, die Prüfungsumgebung ≪Lernstick EXAM≫ zu implementieren. Die Situation in einem Browser unterscheidet sich diesbezüglich deutlich von der in einem ≪vollwertigen≫ Betriebssystem und müsste daher von Grund auf neu konzipiert werden. Die Erstellung einer solchen Prüfungsumgebung ist nicht Gegenstand dieser Arbeit.

3.4 Browserbasierte Lösungsansätze

Es gibt eine Reihe existierender Lösungen, die direkt im Browser laufen und daher nur geringe Anforderungen an die Client-Geräte stellen.

3.4.1 Moodle und das Virtual Programming Lab

Moodle ist ein freies, weit verbreitetes Lernmanagementsystem, das direkt im Browser genutzt wird [Moo23a][Moo23b]. Während des Lockdowns aufgrund der Covid-19-Pandemie und der damit verbundenen Schulschliessungen und Umstellung auf Fernunterricht waren die Autoren (und auch sehr viele Schülerinnen und Schüler) mit der mehr oder weniger vorgeschriebenen ≪Lernplattform≫ Microsoft Teams nicht zufrieden. Deshalb haben sie zusammen mit einigen gleich gesinnten Kolleginnen und Kollegen Moodle installiert, zunächst auf einem eigenen Server, später auf einem schulinternen Server.

Die Erfahrungen waren sehr positiv, so dass die Lernplattform Moodle auch heute noch regelmässig an den Schulen eingesetzt wird, an denen die Autoren tätig sind.

Moodle scheint, zumindest gemäss der Erfahrung der Autoren dieser Arbeit, leider oft nicht so eingesetzt, wie es gedacht ist. Nicht selten trifft man Installationen an, die mehr oder weniger nur als Dateiablage genutzt werden und vom Design her suboptimal aussehen. Die wahre Stärke von Moodle zeigt sich aber erst, wenn man es für das nutzt, was es eigentlich ist: Eine Lernplattform mit Aufgaben, Quiz, Lernfortschrittsverfolgung, dynamischer Steuerung der Anzeige von Lerninhalten, Feedback und einer sehr grossen Zahl nützlicher Plug-Ins wie interaktive Lernvideos, Diktatfunktionen, Anbindung an die Videokonferenz Bigbluebutton [Inc23a] oder an einen OnlyOffice-Server zum kollaborativen Arbeiten an Office-Dokumenten und vieles mehr. Es kann auch nicht schaden, das optisch veraltet wirkende

Standarddesign zu ändern. Auch dies geschieht leider bei vielen von den Autoren beobachteten Installationen nicht.

Ein besonders erwähnenswertes Plug-In für den Informatikunterricht ist das von J.C. Rodríguez-del-Pino von der Universität Las Palmas de Gran Canaria entwickelte Virtual Programming Lab [Rod23]. Das Virtual Programming Lab ermöglicht es, in Moodle eine komplexe Entwicklungsumgebung für verschiedene Programmiersprachen einzurichten. Debugger, Testfälle, von der Lehrperson bereitgestellter Beispielquellcode und die parallele Verwendung mehrerer Dateien werden unterstützt. Ausserdem bietet das Plug-In natürlich auch alle Moodle-Schnittstellen, um Moodle-Features wie Lernfortschrittsverfolgung, automatisierte Korrektur und Bewertung, Abgabetermine und vieles mehr nutzen zu können.

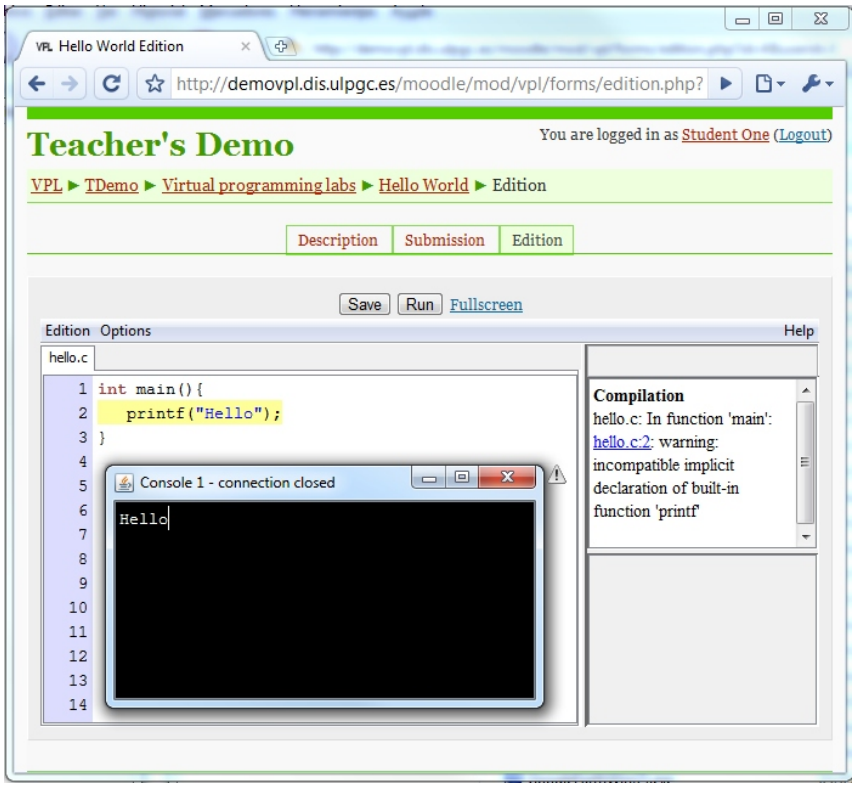

Abbildung 3.5: Screenshot des Virtual Programming Labs [Ler]

Das Virtual Programming Lab benötigt einen separaten Server, auf dem die von den Lernenden geschriebenen Programme in einem Docker-Container kompiliert und ausgeführt werden.

Leider ist die verfügbare Dokumentation sehr spärlich und es ist daher schwierig, den Einsatz des Virtual Progamming Labs aus Sicht des Unterrichtenden richtig zu konfigurieren.

Gleichwohl wurde das Virtual Programming Lab von den Autoren bereits zwei Schuljahre lang erfolgreich im Unterrichtsalltag eingesetzt, wodurch viele Probleme mit nicht installierbarer Software auf BYOD-Geräten gelöst werden konnten.

3.4.2 Simple Protocol for Independent Computing Environments (SPICE)

Das SPICE-Projekt [spib] hat sich zum Ziel gesetzt, eine vollständige Open-Source-Lösung für den nahtlosen Fernzugriff auf virtuelle Maschinen bereitzustellen, so dass man ohne Komplikationen Videos abspielen, Audio aufnehmen, USB-Geräte gemeinsam nutzen und Ordner freigeben kann. Es scheint also ein sehr vielversprechendes Projekt für diese Arbeit zu sein.

Um dieses Projekt erfolgreich umsetzen zu können, ist ein zuverlässiger und voll funktionsfähiger Client, der über einen Browser zugänglich ist, unabdingbar.

Im Zuge einer entsprechenden Recherche wurde ein offizieller HTML5-Client [spi21a] und das Projekt «spice-web-client» [eye19] gefunden werden, weshalb die Autoren SPICE als zumindest potenziell browserbasiert in diese Liste aufgenommen haben.

Leider werden beide Projekte seit Jahren nicht mehr weiterentwickelt und sind nicht vollständig bzw. funktionieren nicht mit Guacamole. Ausserdem ist die gleichzeitige Nutzung mehrerer Clients nur als experimentelles Feature des SPICE-Protokolls [spia] möglich. Dies wäre ein zu grosses Risiko für das Projekt.

Trotzdem können sich die Autoren gut vorstellen, in Zukunft eine Variante des Projektes mit diesem Protokoll zu realisieren.

3.4.3 Remote Desktop Protocol (RDP)

Das Remote Desktop Protocol (RDP) [Mic22a] ist ein proprietäres Netzwerkprotokoll, das von Microsoft entwickelt wurde. Es ermöglicht einem Benutzer, sich mit einem entfernten Computer zu verbinden und diesen fernzusteuern. Das RDP-Protokoll wird hauptsächlich für die Fernverwaltung von Windows-basierten Systemen verwendet.

Zu den wichtigsten Eigenschaften des Remote Desktop Protocols gehören:

Effizienz: Während andere Fernzugriffsprotokolle wie VNC oder SSH mit X-Weiterleitung den kompletten Bildschirm (VNC) oder zumindest einzelne Fenster (`ssh -X`) als Bild übertragen, sendet RDP nur die notwendigen Anweisungen, um den gewünschten Bildschirminhalt lokal darzustellen. Dadurch ist das Protokoll deutlich bandbreitenschonender und reduziert die Serverlast: Die Grafikeinheit des Servers muss hier kaum arbeiten, während sie beispielsweise bei VNC den gesamten Desktop rendern muss und der Server das Ergebnis dann über die TCP-Verbindung verschickt.

Sicherheit: RDP bietet eine Reihe von Sicherheitsmechanismen, um die Verbindung zwischen Client und Server zu schützen. Dazu gehören Verschlüsselung, Authentifizierung und Netzwerkzugriffskontrolle.

Funktionen: RDP ermöglicht es dem Benutzer, den entfernten Computer so zu steuern, als sässe er direkt davor. Der Benutzer kann Tastatureingaben und Mausbewegungen senden und Bildschirmausgaben des entfernten Computers anzeigen. Ausserdem können Dateien und Ressourcen zwischen dem lokalen und dem entfernten Computer ausgetauscht werden.

Multimedia-Unterstützung: RDP bietet auch Unterstützung für die Übertragung von Multimediainhalten über die Verbindung. Dies ermöglicht beispielsweise das Streaming von Audio- und Videodateien

vom entfernten Computer zum lokalen Client.

Skalierbarkeit: Das RDP-Protokoll unterstützt auch die gemeinsame Nutzung von Anwendungen durch mehrere Benutzer. Ein Server kann mehrere RDP-Sitzungen gleichzeitig hosten, wobei jeder Benutzer seine eigene Sitzung mit dem Server hat.

RDP benötigt eine entsprechende Client-Software, um eine Verbindung zum entfernten Desktop herzustellen. Daher gehört RDP eigentlich nicht in die Liste der browserbasierten Lösungen. Mit Apache Guacamole existiert jedoch ein browserbasierter RDP-Client, weshalb RDP dennoch in dieser Liste aufgeführt wird.

3.4.4 X Remote Desktop Protocol

Das X Remote Desktop Protocol (XRDP) ist eine Open-Source-Implementierung von RDP [neu23]. XRDP kann auf verschiedenen Betriebssystemen wie Linux, Windows und macOS eingesetzt werden und ermöglicht die grafische Anmeldung an entfernten Maschinen. Dabei akzeptiert XRDP Verbindungen von verschiedenen RDP-Clients wie xfreerdp oder remmina [Gat23], aber auch Apache Guacamole. Letzteres sorgt dafür, dass XRDP auch als (zumindest potenziell) browserbasierter Ansatz betrachtet werden kann.

Kapitel 4

Schwierigkeiten

In diesem Kapitel werden einige technische Schwierigkeiten beschrieben, auf die die Autoren bei der Umsetzung des Projekts gestossen sind.

4.1 LDAP

Die Schulen im Arbeitgeberkanton der Autoren verwenden Active Directory, um die schulinternen Benutzeraccounts zu verwalten. Um die zukünftigen Benutzerinnen und Benutzer nicht zu einer oft als umständlich empfundenen Account-Erstellung zu zwingen, lag es daher nahe, zu versuchen, das bestehende LDAP zu nutzen, um die schulinternen Benutzer-Accounts direkt in der virtuellen Desktop-Umgebung weiter verwenden zu können.

Guacamole bietet eine Anbindung an LDAP-Server [Fou23d] und für Linux selbst gibt es mit *SSSD* [SSS] einen systemd-Daemon, der es erlaubt, Benutzer über LDAP zu authentifizieren, als Ergänzung zur internen Datei `/etc/passwd`.

Bei den Versuchen, die Kontenverwaltung über LDAP zu steuern, stiessen die Autoren jedoch auf einige Schwierigkeiten, die letztlich dazu führten, dass die Versuche in diese Richtung abgebrochen wurden. Diese Schwierigkeiten werden im Folgenden kurz beschrieben.

4.1.1 Das Active Directory des Kantons Basel-Landschaft

Das Active Directory des Arbeitgeberkantons ist sehr umfangreich. Die Autoren vermuten, dass diese Datenbank über viele Jahre gewachsen ist und immer wieder neue Funktionen hinzugefügt wurden. Für in Active Directory komplett unerfahrene Personen, wie es die Autoren sind, erwies es sich insgeamt als äusserst schwierig, sich in dieser Datenbank soweit zurechtzufinden, um den erwähnten SSSD-Daemon korrekt konfigurieren zu können.

4.1.2 Rechtliche Netzwerkprobleme

Der LDAP-Server des Kantons Basel-Landschaft ist aufgrund verschiedener interner Firewall-Regeln nur in einer demilitarisierten Zone (DMZ) erreichbar. Die Autoren administrieren die schulinternen Webappserver und können von diesen aus auf den LDAP-Server zugreifen, da sich die Webappserver in der gleichen DMZ befinden.

Ein Zugriff von ausserhalb, zum Beispiel vom Projektserver, ist jedoch nicht ohne Weiteres möglich. Es wäre also, um einen externen Server auf die gewünschte Weise betreiben zu können, erforderlich, die Firewallregeln zu umgehen, was offensichtliche rechtliche Schwierigkeiten mit sich brächte.

Die Chancen, dass der Projektserver einmal kantonsintern betrieben wird, sind eher gering. So setzt die Schulinformatik, wie es scheint, seit vielen Jahren auf das Ökosystem rund um Microsoft und deren Serversoftware. Und die erwähnten Webappserver sind als kleine virtuelle Server mit wenig Arbeitsspeicher und Speicherplatz dafür nicht geeignet.

4.1.3 Technische Netzwerkprobleme

Es hat sich auch herausgestellt, dass der kantonsinterne LDAP-Server noch TLS in der Version 1.0 verwendet — zumindest war das zum Zeitpunkt der LDAP-Experimente der Autoren laut den aus den in den Logfiles enthaltenen Meldungen ein Grund für die Weigerung des SSSD-Daemons. Moderne Linux-Distributionen, die mit aktuellen Versionen von OpenSSL ausgeliefert werden, lehnen jedoch die Zusammenarbeit mit derart veralteten TLS-Standards aus Sicherheitsgründen ab.

Als Experiment wurde versucht, OpenSSL so weit zu downgraden, dass ein funktionierender Handshake mit diesem offensichtlich veralteten LDAP-Server möglich war. Dieses Unterfangen stellte sich jedoch nach einiger Zeit als eine immer grösser werdende Kaskade von Abhängigkeitsproblemen heraus, so dass dieses Experiment nach einigen Tagen abgebrochen wurde.

Selbst wenn diese Downgrading-Methode funktioniert hätte, wäre dies aufgrund der berechtigten Sicherheitsbedenken, die dazu geführt haben, dass OpenSSL TLS in der Version 1.0 nicht mehr unterstützt, keine gute Idee im Sinne der ≪best practises≫ gewesen.

4.1.4 Fehlende Erfahrung mit LDAP

Dieses Problem ist sehr subjektiv, soll aber nicht unerwähnt bleiben. Die Autoren leben computertechnisch seit Jahrzehnten ausschliesslich in der Linux-Welt und kommen dort gut zurecht. Gleichzeitig empfinden sie die Windows-Welt inzwischen oft als übermässig kompliziert und schwer durchschaubar.

Windows-Systeme scheinen völlig anders aufgebaut zu sein, als es die Autoren gewohnt sind, und diese Welt folgt offensichtlich diametral anderen technischen (und auch moralischen) Philosophien.

Unter diesem Gesichtspunkt war es eine grosse Herausforderung, sich mit den Interna eines Active Directory zu beschäftigen. Auch sind sich die Autoren bis heute nicht sicher, ob sie diesbezüglich alles Wesentliche richtig verstanden haben.

Aufgrund dieses Umstandes erschien der Zeitaufwand, der exklusiv in die Beschäftigung mit den technischen

Details von LDAP investiert werden müsste, nur um das Account Management komfortabler zu gestalten, irgendwann als verhältnismässig zu gross bei entsprechend geringem Nutzen.

4.1.5 Konflikt mit der ≪Easy deployability≫

Es ist keineswegs gesichert, dass andere Schulen, die dieses System in Zukunft potenziell nutzen wollen, ebenfalls auf ein LDAP-System setzen. Und wenn ein Active Directory vorhanden ist, ist auch nicht garantiert, dass dieses ausreichend analog zum baselbieter LDAP aufgebaut ist.

Ein Serveradministrator einer anderen Schule müsste hier also wahrscheinlich erhebliche Anpassungen vornehmen. Im besten Fall müssten die SSSD-Einstellungen angepasst werden, im schlechtesten Fall (nämlich dann, wenn die betreffende Schule über keinen LDAP-Server verfügt) müsste eine komplette alternative Möglichkeit zur Account-Generierung und -Verwaltung implementiert werden.

Im Hinblick auf die angestrebte ≪easy deployability≫ wurde daher auch aus diesem letzten Grund entgültig entschieden, die Experimente mit LDAP nicht weiter zu verfolgen.

4.2 Verschlüsselung der Homeverzeichnisse

Da davon auszugehen ist, dass der Speicherplatz auf einem einzelnen, finanziell tragbaren Server für eine ausreichend grosse Anzahl von Nutzerinnen und Nutzern nicht ausreichen wird, wurde bereits zu einem frühen Zeitpunkt der Projektentwicklung davon ausgegangen, dass die Homeverzeichnisse auf externe Server ausgelagert werden müssen. Damit verbunden ist die zwingende Notwendigkeit, diese Homeverzeichnisse zu verschlüsseln, so dass allfällige Cloud-Anbieter die persönlichen Dateien der Schülerinnen und Schüler nicht lesen können. Andernfalls würde der kantonale Datenschutzbeauftragte das Projekt zu Recht niemals für den produktiven Einsatz freigeben.

Es ist an dieser Stelle wichtig zu betonen, dass diese Verschlüsselung nur dazu dient, die Daten vor schulhausfremden Augen wie die eines externen Cloud-Anbieters, zu schützen und nicht dazu, die individuellen Home-Verzeichnisse zwischen den Benutzern unzugänglich zu machen. Letzteres soll durch Benutzerrechte auf Betriebssystemebene und nicht durch Verschlüsselung erreicht werden. Die Idee dahinter ist, dass jemand, der sich Root-Rechte verschaffen kann, um diese Einschränkung der Benutzerrechte zu umgehen, mit diesen erhöhten Privilegien sowieso immer alle Daten der Benutzerinnen und Benutzer lesen kann.

Es gibt bereits Erfahrungen aus früheren Jahren, in denen ein auf einem grossen, günstigen, aber im Ausland stehenden Server gespeichertes und verschlüsseltes Festplattenabbild problemlos auf einen viel kleineren schweizer Server eingebunden und lokal entschlüsselt wurde. Dieser technische Trick wurde notwendig, weil der kleine Server viel zu wenig Speicherplatz für den vorgesehenen Nutzen bietet, der externe Server aufgrund seines physischen Standortes aber Datenschutzprobleme mit sich bringt. Seit Sommer 2020 funktioniert diese Methode im täglichen Gebrauch durchgehend problemlos und mit überraschend guter Performance, und zwar so, dass der externe Server niemals das Keyfile oder andere unverschlüsselte Daten zu Gesicht bekam.

Aufgrund dieser Erfahrungen war die Hoffnung gross, dass eine transparente Verschlüsselung der Homeverzeichnisse einfach zu realisieren sein würde. Dies erwies sich als Irrtum. Dies lag vor allem daran, dass

in einem unprivilegierten Container gearbeitet werden musste, der wiederum in einem unprivilegierten Container eingebettet war (Details zum Aufbau der Containerstruktur finden sich in Kapitel 5).

Das Problem konnte am Ende gelöst werden. Das Vorgehen, um die funktionierende Lösung zu erreichen ist in Kapitel 5 beschrieben — in diesem Kapitel sollen hingegen nur die Fehlversuche und Probleme beschrieben werden.

4.2.1 LUKS/cryptsetup

Zunächst wurde derselbe Ansatz verfolgt, der auch bei dem beschriebenen und im täglichen Einsatz befindlichen System zwischen einem kleinen Server auf schweizer Boden und einem grossen externen Server in einem Rechenzentrum im Ausland verwendet wird, wobei der externe Server als eine Art Speichererweiterung dient. Die Vorgehensweise dazu kann wie folgt beschrieben werden:

1. Auf dem externen Server befindet sich eine mit LUKS (Linux Unified Key Setup) [cry23] verschlüsselte Image-Datei. Der externe Server hat die entschlüsselte Version noch nie gesehen und kennt auch das Keyfile nicht.
2. Mit SSHFS [lib22] wird das Verzeichnis des externen Servers, das die verschlüsselte Image-Datei enthält, auf dem internen Server eingebunden.
3. Mit dem auf dem internen Server vorhandenen Keyfile wird die Image-Datei per cryptsetup entschlüsselt. Dabei wird ein neues Block-Device unter `/dev/mapper` erzeugt.
4. Dieses neue Block-Device wird nun an einem weiteren Mountpoint auf dem internen Server gemountet.
5. An diesem Mountpoint steht nun der durch die Image-Datei definierte Speicherplatz zur Verfügung.

Die Idee, die Aufgabe der Verschlüsselung der Homeverzeichnisse auf die gleiche Weise zu lösen, war bestechend. cryptsetup liess sich problemlos auf dem Arbeitscontainer installieren und ein grosses, zufälliges Keyfile war mit Hilfe von OpenSSL schnell erzeugt. Anschliessend wurde ein 2GB grosses, leeres Image `clean.img` in einem neuen Verzeichnis `/images` erzeugt und darauf ein ext4-Dateisystem angelegt.

Die Idee bestand zu diesem Zeitpunkt daraus, dass jeder Benutzer seine eigene Imagedatei erhält. Neue Nutzerinnen und Nutzer erhalten zu Beginn eine Kopie der Datei `clean.img`. Auf diese Weise ist auch gleich eine Quotaregelung eingebaut (derzeit 2 GB pro User, aber in Zukunft durch entsprechendes Ersetzen von `clean.img` leicht anpassbar). Und der Ordner, in dem die fertig verschlüsselten und auf die Entschlüsselung wartenden Images abgelegt werden sollen, `/image/encrypted/`, könnte dann in Zukunft ganz einfach auf einen externen Server ausgelagert werden, zum Beispiel per SSHFS, CIFS [coma] oder NFS (Network File System) [comb]. Der weitere Plan war, für jeden Benutzer ein eigenes Keyfile zu generieren, welches per GPG verschlüsselt in einem gemeinsamen Ordner abgelegt werden sollte. Über ein entsprechendes PAM-Modul sollte dann beim Anmelden der Benutzerinnen und Benutzer jeweils das GPG-Keyfile geöffnet werden (die Passphrase sollte jeweils zusammen mit dem Benutzerpasswort aktualisiert werden), woraufhin das Keyfile entschlüsselt und damit das persönliche Home-Image wieder nutzbar gemacht werden sollte.

Bevor jedoch ein System von On-the-fly-Kopien von `clean.img` und anschliessender Verschlüsselung

dieser Kopien und deren transparenter Einhängung, geschweige denn eine Kaskade von PAM-Modul und gpg-agent aufgebaut wurde, wurde das Prinzip an einer ersten Kopie von `clean.img` manuell ausprobiert.

Diese Kopie konnte problemlos mit LUKS formatiert werden, so dass nun tatsächlich ein mit dem Keyfile verschlüsseltes Image vorlag. Der nächste Schritt, das Image zu «öffnen» (`cryptsetup luksOpen`), schlug jedoch fehl.

Zunächst beschwerte sich cryptsetup über ein fehlendes Kernelmodul. Da der unprivilegierte Arbeitscontainer, der sich zudem noch in einem unprivilegierten «Obercontainer» befand, den Kernel mit dem Hostsystem teilte und innerhalb des Containers mangels echter root-Rechte keine Kernelmodule geladen werden konnten, musste das betreffende Modul direkt im Hostsystem geladen werden. Im Hinblick auf die «easy deployability» nicht optimal, aber noch tolerierbar.

Doch cryptsetup verweigerte weiterhin die Zusammenarbeit: Es stellte sich heraus, dass unter `/dev/mapper` keine Block-Devices innerhalb des Containers angelegt werden dürfen - ein Umstand, der eigentlich von Anfang an hätte klar sein müssen.

Es folgten unzählige Versuche, dieses Problem zu umgehen. Zum Beispiel wurden die folgenden Versuche unternommen, alle ohne Erfolg:

- Manipulation des UID- und GID-Mappings.
- Die in LXC eingebaute Möglichkeit, Verzeichnisse mit dem Host zu teilen [DA17].
- Unzählige Varianten mit speziellen und immer komplizierteren bind-mounts, die teils mit weiteren bind-mounts auf dem Host, teils mit bind-mounts in anderen parallelen Containern, privilegiert und unprivilegiert, auf irgendwann kaum noch nachvollziehbare Weise verknotet wurden.
- Die Images entgegen dem Ziel der «easy deployability» per SSHFS auf das Hostsystem auszulagern, dort zu entschlüsseln und, wiederum per SSHFS, die Ergebnisse innerhalb des Containers wieder zu verwenden.

Am Ende musste eingesehen werden, dass dieser Weg in einen unprivilegierten Container nicht möglich ist. Und in nachträglicher Betrachtung ist das auch gut so, denn eine Manipulationsmöglichkeit von `/dev/mapper` innerhalb eines Containers würde eine grosse Sicherheitslücke in Linux-Container reissen.

4.2.2 rclone

Ein sehr vielversprechender Ansatz war die Verwendung von rclone. rclone ist ein Tool, das es erlaubt, verschiedene Cloud-Anbieter bequem auf der Kommandozeile zu nutzen und dort auch verschlüsselte Verzeichnisse, sogenannte «buckets», anzulegen und lokal zu mounten [Cra23a].

Für diesen Ansatz wurde ein neuer, unprivilegierter Container mit dem Namen *storage* angelegt[1], der allein für die Entschlüsselung eines Verzeichnisses zuständig war, in dem später die Home-Verzeichnisse abgelegt werden sollten.

[1] In Kapitel 5 findet sich mehr zum Aufbau der einzelnen Container.

In *storage* wurde nun ein Verzeichnis `/root/storage` erzeugt und rclone angewiesen, dieses Verzeichnis als lokale Cloud zu verwenden. Künftig, wenn die Homeverzeichnisse auf eine externe Cloud ausgelagert werden würde, müsste man nur die rclone-Konfiguration entsprechend anpassen, während alles andere noch gleich funktionieren sollte — soweit zumindest die Idee in dieser Phase der Projektentwicklung.

Weiter wurde mit rclone in `/root/storage` ein verschlüsseltes Unterverzeichnis `crypt` angelegt, welches anschliessend per `rclone mount` in ein Verzeichnis `/root/storage-decrypted` eingehängt wurde.

Die ersten Tests waren vielversprechend: Alles, was in `/root/storage-decrypted` gespeichert wurde, landete in verschlüsselter Form in `/root/storage/crypt`.

Als Nächstes wurde im Arbeitscontainer versucht, das Verzeichnis `/root/storage-decrypted` von *storage* mit SSHFS lokal einzubinden. Dies funktionierte auch.

Allerdings musste festgestellt werden, dass trotz der Mount-Option `allow_other` das Verzeichnis nur für den Benutzer lesbar war, der den SSHFS-Befehl ausgeführt hatte. Eigentlich hätte man dies sogar als Feature nutzen können, indem alle Benutzer beim Anmelden ihre eigenen SSHFS-mounts durchführen — die Daten wären dann noch besser vor systeminternen Blicken geschützt.

Leider war dies nicht die einzige Einschränkung: Es war überhaupt nicht möglich, die Dateirechte in diesem per SSHFS gemounteten Verzeichnis zu ändern. So wäre es zum Beispiel für Schülerinnen und Schüler nicht möglich, im Informatikunterricht ein Shell-Skript zu programmieren und diesem Ausführungsrechte zu erteilen.

Nach vielen weiteren Versuchen mit UID-Mappings und diversen Mount-Optionen wurde den Autoren klar: rclone kann solche Dateiattribute gar nicht übernehmen, weil die Cloud-Anbieter diese nicht unterstützen.

Damit war auch die Idee, rclone einzusetzen, zum Scheitern verurteilt. Das war sehr schade, denn diese Variante hätte ganz im Sinne der ≪easy deployability≫ dafür gesorgt, dass durch eine einfache Anpassung der rclone-Konfiguration ein nahezu beliebiger Cloud-Anbieter als Speicherort für die Home-Verzeichnisse hätte genutzt werden können - die Liste der von rclone unterstützten Anbieter und Protokolle ist nämlich recht lang [Cra23c].

4.3 Quota

Wie beschrieben, hat die Idee, Image-Dateien für die Home-Verzeichnisse zu verwenden, nicht direkt funktioniert. Selbst das Loop-Mounten von unverschlüsselten Festplattenabbildern scheint in einem unprivilegierten Container nicht möglich zu sein. Damit ist ein einfacher Weg zur Einrichtung von Quotas weggefallen.

Auch hier wurde am Ende eine Lösung gefunden, auch wenn diese mit einem Kompromiss einhergeht. Diese Lösung wird in Kapitel 5 beschrieben. Im aktuellen Kapitel folgen hingegen einige Ansätze, die nicht zum Ziel geführt haben.

4.3.1 Idee: Quota auf Dateisystemebene

Einige Dateisysteme, wie btrfs [kda], das für den Speicherpool der Container verwendet wird, bieten Quota-Optionen an. Diese müssen jedoch in `/etc/fstab` gesetzt werden, damit sie ab dem Mounten aktiv sind. Innerhalb eines Linux-Containers werden jedoch keine Dateisysteme über `/etc/fstab` gemountet.

Daher müsste eine Quotaregel auf Dateisystemebene ausserhalb des Containers auf der Ebene des Hostsystems auf den Containerspeicher angewendet werden. Dies würde jedoch bedeuten, dass sich die Quota auf alle Container bezieht, was nicht praktikabel erscheint.

Zwar wäre es möglich, dem Storage Pool der Linux-Container zusätzliche, grössenbeschränkte Volumes hinzuzufügen, die separat innerhalb der Container gemountet werden. Dies müsste jedoch ebenfalls auf der Ebene des Hostsystems geschehen, was die *easy deployability* verhindert. Ausserdem müssten diese Volumes dann innerhalb der Container an der richtigen Stelle verlinkt werden, was mit der in Kapitel 5 beschriebenen Verschlüsselungsmethode nicht vereinbar ist. Dass am Ende, wie ebenfalls in Kapitel 5 beschrieben, eine Containerstruktur mit ineinander verschachtelten (nested) Containern gewählt wurde, macht diesen Ansatz zudem noch komplexer und kaum verfolgenswert.

4.3.2 Idee: Watchdog

Eine Art Watchdog-Skript könnte die Grösse der Userhomes im Auge behalten und bei Userhomes, die eine definierte Maximalgrösse zu überschreiten drohen, eingreifen.

Die Frage ist jedoch, wie dieser Eingriff genau aussehen soll. Der ≪watchdog≫ könnte der betreffenden Person die Schreibrechte im eigenen Homeverzeichnis entziehen, was aber zur Folge hat, dass sie auch keine Dateien mehr löschen kann, um das Problem zu beheben. Oder er könnte einfach das Home-Verzeichnis löschen und neu anlegen — ein offensichtlich nicht sehr benutzerfreundlicher Ansatz.

Insgesamt scheint die Idee eines solchen Kontrollmechanismus also leider auch in eine Sackgasse zu führen.

4.3.3 Idee: Individuelle Storagecontainer

Es wäre möglich, verschachtelte Container mit entsprechend begrenztem Speicherplatz zu erzeugen und den Inhalt dieser Container zum Beispiel per SSHFS verfügbar zu machen. Im Grunde würde man damit das Loopmounten von Image-Dateien auf komplizierte Weise und mit viel Overhead abbilden.

Während es technisch sicherlich möglich ist, viele Container (einen pro User) parallel zu betreiben und bei der Accounterstellung entsprechend neu zu generieren, erscheint diese Idee doch sehr fehleranfällig und insgesamt schwierig zu warten. Zudem ist, wie in Kapitel 5 beschrieben, die Anzahl der verfügbaren IP-Adressen für diese Container vermutlich stärker begrenzt als die Anzahl der potenziellen Nutzerinnen und Nutzer.

Nicht zuletzt würde dieses System die Auslagerung der Userhomes auf externe Server erheblich erschweren: Die einzelnen Speichercontainer müssten durch völlig andere Strukturen ersetzt werden. Ziel wäre es jedoch, einen solchen Wechsel so einfach wie möglich zu gestalten.

4.3.4 Idee: Mounten eines Verzeichnisses über einen lokalen FTP-Server

Der FTP-Server proftpd verfügt über ein Quota-Modul, mit dem die Quotas beim Schreiben von Dateien über FTP etwas feiner eingestellt werden können, wenn die Quotas auf Dateisystemebene aktiviert sind. Eine Zeit lang wurde daher die Idee verfolgt, einen lokalen proftpd-Server einzurichten, dort die Quotas zu setzen und dann die entsprechenden Verzeichnisse über ein FUSE-Dateisystem per lokalem FTP zu mounten.

Leider muss auch hierfür das Dateisystem auf Host-Ebene angepasst werden, was der «easy deployability» widerspricht. Ausserdem erwies sich das Verfahren als fehleranfällig und schwierig zu warten.

Analoge Versuche mit anderen Netzwerkdateisystemen wie NFS, CIFS, SSHFS und SMB führten ebenfalls in eine Sackgasse, da diese solche Erweiterungen nicht kennen und teilweise auch die Dateirechte nicht korrekt handhaben können.

4.3.5 Idee: Mounten via FUSE

Mit ext4fuse gibt es eine ext4-Implementation für FUSE, die das Mounten von Imagedateien ohne loopmount ermöglicht [Lle20]. Leider wird ext4fuse seit einiger Zeit nicht mehr weiterentwickelt und konnte auch nicht mehr kompiliert werden.

Eine prinzipiell funktionierende Alternative ist fuse2fs, das ebenfalls ext4-Dateisysteme über FUSE auf Images mounten kann, ohne ein loopmount zu benötigen [And23].

Tests mit fuse2fs verliefen vielversprechend. Images konnten tatsächlich gemountet und verwendet werden. Leider scheint fuse2fs kein Journaling zu unterstützen. Korrumpierte Image-Dateien würden daher tendenziell zu Problemen bis hin zum Datenverlust führen. Die Autoren schätzen die Wahrscheinlichkeit solcher Ereignisse im Produktivbetrieb als zu hoch ein. Spätestens dann, wenn diese Images/Userhomes auf getrennten Servern liegen, ist mit gelegentlichen Ausfällen aufgrund von Netzwerkunterbrechungen zu rechnen, die gelegentliche Rebuilds erforderlich machen. Dies ist zumindest die Erfahrung der Autoren dieser Arbeit.

4.3.6 Idee: Erzeugen einer minimalen virtuellen Maschine

In einer virtuellen Maschine könnten die Imagedateien problemlos loopgemountet werden, sogar eine Verschlüsselung mittels LUKS wäre dort möglich. Da aber auf dem Hostsystem keine hardwarebeschleunigten virtuellen Maschinen laufen, müsste diese VM auf Softwareebene allein mit QEMU emuliert werden. Damit dürfte die Performance für den kryptographischen Teil nicht ausreichen, aber als reiner Image-Mounter, der über ein Netzwerkdateisystem erreichbar ist, könnte zum Beispiel ein auf Geschwindigkeit und geringe Systemgrösse getrimmtes Alpine Linux als virtuelle Maschine einen Versuch wert sein.

Da wir uns für ein System mit verschachtelten, unprivilegierten Linux-Container Containern entschieden haben (siehe Kapitel 5), müssten jedoch einige Gerätedateien des Hosts (nämlich `/dev/kvm`, `/dev/vhost-net` und `/dev/vhost-sock`) dem Hauptcontainer zur Verfügung gestellt werden. Diese Geräte öffnen potenziell den Weg zum Rest des Hosts. Dies stellt ein Sicherheitsrisiko dar, das von den Autoren als zu hoch eingeschätzt wird.

Ausserdem hat sich herausgestellt, dass das Gerät `/dev/kvm` ohnehin schon auf Host-Ebene fehlen würde — eine Einschränkung des Server-Anbieters.

4.4 Datenaustausch

Es wäre wünschenswert, wenn die Schülerinnen und Schüler auf einfache Weise Daten mit dem System austauschen könnten.

4.4.1 Idee: Microsoft OneDrive einbinden

Der Kanton Basel-Landschaft stellt den Schülerinnen und Schüler einen Zugang zu OneDrive zur Verfügung. Es liegt daher nahe, sofern man die datenschutzrechtlichen Bedenken vorerst ausser Acht lässt, diesen in das System einzubinden. Solange das Homeverzeichnis nicht ausgelagert wurde, macht es aus Gründen der Speicherkapazität keinen Sinn, Daten mit einem solchen Dienst zu synchronisieren. Er müsste also gemountet werden und dieser Prozess sollte dabei mit möglichst geringem Cache-Speicher auskommen.

Das Projekt onedriver [jst] scheint dafür gut geeignet zu sein. Es bietet die Möglichkeit des Mountens, ist grafisch ansprechend gestaltet und sollte daher auch für die Anwenderinnen und Anwender ohne grossen Hürden zugänglich sein.

Leider hat sich herausgestellt, dass der Zugang zum OneDrive des Kantons eine solche Einbindung nicht oder zumindest nicht ohne Umweg zulässt.

Abbildung 4.1: Fehlermeldung von onedriver

Gegenwärtig bleibt somit nur die Möglichkeit, OneDrive über einen anderen Weg, wie zum Beispiel rclone [Cra23b], zu integrieren, was mit den Schülerinnen und Schülern gemeinsam durchgeführt werden sollte.

Zu einem späteren Zeitpunkt wurde dafür noch ein entsprechendes Skript geschrieben. Dies wird in Abschnitt 5.14 genauer beschrieben.

4.5 Load Balancing

Es ist anzunehmen, dass ein einzelner Server mit steigender Zahl an parallelen Nutzerinnen und Nutzern an seine Kapazitätsgrenzen stösst. Deshalb wurde eine Form des Load-Balancings eingeplant: Eingehende Verbindungen sollen von einem Loadbalancer-Container wahlweise an den lokalen Guacamole-Container oder an eine sekundäre Instanz auf einem separaten Server weitergeleitet werden. Dabei sind ebenfalls einige Probleme aufgetreten, die im Folgenden beschrieben werden.

4.5.1 Loadbalancing über ein nginx Reverse Proxy mit SSL Terminierung

Als Erstes wurde versucht, ein simpler Reverse Proxy mit nginx aufzustellen, um diesen anschliessend zu einem Load Balancer zu erweitern. Dabei sollte im Proxy die SSL-Verbindung terminiert und neu aufgebaut werden, damit über ein technisches Cookie die Usersessions festzuhalten in dem Sinne, dass ein

einmal über Guacamole im Desktop eingeloggter Nutzer nicht mitten in der Session den Server wechselt, eingefügt werden kann.

Grundsätzlich hat das funktioniert, aber es hat sich leider herausgestellt, dass hierbei eine enorme Performance-Einbusse eintritt: Die Desktopsysteme erhalten gemäss den Tests der Autoren so einen Lag von mehreren Sekunden, zeitweise sogar eine halbe Minute. Es konnte nicht ermittelt werden, woran das lag. Scheinbar ist nginx dafür, eher überraschenderweise, nicht geeignet.

4.5.2 Loadbalancing auf TCP-Ebene: Kein Sessionmanagement

Der Loadbalancer HAProxy[hap] bietet Loadbalancing auf TCP-Ebene an. Hier ist jedoch prinzipbedingt fast keine Möglichkeit gegeben, die Sessions der Nutzerinnen und Nutzer konsistent auf demselben Server zu halten. Da der HAProxy im TCP-Modus ohne SSL-Terminierung arbeitet (SSL-Passthrough), ist es nur möglich, Sessions anhand der Quell-IP zuzuordnen. Eine Lösung über ein Cookie, wie es mit einem Reverse Proxy und SSL-Terminierung machbar wäre, ist hier nicht gegeben.

Gleichzeitig werden in der Unterrichtspraxis aber alle Schülerinnen und Schüler von der scheinbar gleichen Quell-IP her kommen. Schliessich sind sie ja alle über denselben Internetanschluss der Schule verbunden.

Somit wird hier das Load-Balancing ad absurdum geführt: Sämtliche Nutzerinnen und Nutzer werden, mit wenigen Ausnahmen (wie Schülerinnen und Schüler, die aus irgendwelchen Gründen von zu Hause aus mitarbeiten) doch wieder auf ein und denselben Server geschickt.

Ohne die Sessions zu berücksichtigen, kann es aber schnell passieren, dass jemand sich auf Server 1 einloggt, das nächste TCP/IP-Paket dann aber an Server 2 geschickt wird, der dann nicht weiss, was er damit anfangen soll.

4.5.3 Loadbalancing auf User-Ebene

Es wurde auch die Idee verfolgt, die Nutzerinnen und Nutzer bei der Accounterstellung auf gleich auf verschiedene Server zu verteilen. Das Problem hierbei ist aber, dass dann immer noch ein einziger Guacamole-Container auf einem einzigen Server für alle Verbindungen verantwortlich ist - lediglich die konkreten Desktop-Container werden auf verschiedene Server verteilt. Die Last auf diesen einzigen Guacamole-Container und seinen Host-Server dürfte aber immer noch stark ansteigen, wenn mit vielen parallelen Nutzerinnen und Nutzern gerechnet wird.

4.5.4 Fehlende Testmöglichkeiten

Die finanzielle Ausgangslage erlaubt es nicht, mehrere Server anzumieten, nur um das Loadbalancing zu testen. Vieles der Umsetzung des Loadbalancings muss während der Entwicklungsphase dieses Projektes also leider eher auf lediglich theoretischer Ebene im Sinne von einer ≪Vorbereitung auf diesen Fall≫ passieren. Es ist den Autoren aus diesem Grund also bedauerlicherweise nicht ohne Weiteres möglich, das Load Balancing detailliert auszuarbeiten.

Dieses letztgenannte Problem wird auch bei der schlussendlich umgesetzten Lösung, die in Kapitel 5 beschrieben wird, verbleiben.

Kapitel 5

Umsetzung eines Prototyps

In diesem Kapitel soll die erste praktische Umsetzung des Projektes zusammen mit den dabei notwendig gewordenen Entscheidungsfindungen beschrieben werden.

Auf einer Metaebene betrachtet haben die Autoren in vergangenen Projekten gute Erfahrungen mit einem zweistufigen Vorgehen gemacht:

Prototyp-Phase: In dieser Phase wird ein Prototyp entwickelt. Dabei geht es in erster Linie darum, etwas Funktionierendes zu erhalten. Es wird nur wenig Fokus auf sauberen Code gelegt. ≪Schnelle Hacks≫ sind in dieser Phase üblich und auch Spaghetti-Code kommt nicht selten vor. Das Ziel ist es, eine vollständig funktionierende Version des Projektes zu erhalten und dabei in kurzer Zeit möglichst viel Erfahrung zu sammeln. Es geht also darum, möglichst alle potenziellen Probleme zu erkennen und Lösungsansätze zu entwickeln, nicht aber darum, diese in allen Details sauber und effizient auszugestalten.

Release-Candidate-Phase: Durch die Erfahrungen aus der Prototyp-Phase ergibt sich nun ein Gesamtbild. Jetzt ist bekannt, auf welche Probleme man achten muss, welche Lösungsansätze wie funktionieren und welche dies nicht tun. Es ergibt sich jetzt also einen besseren Überblick und somit einen konsistenteren Plan. Mit anderen Worten kann jetzt das ganze Projekt mit etwas Abstand betrachtet und entsprechend von Anfang an sauber konzeptioniert und erarbeitet werden. Daher wird jetzt das Projekt von Grund auf neu aufgesetzt, wobei die gesammelte Erfahrung hilft, viel saubereren Code zu schreiben und hoffentlich auch effizientere Konzeptionierungen umzusetzen. Letzteres wäre in der ersten Phase kaum möglich, da noch nicht im Detail bekannt ist, wohin die Reise geht und was dafür wie, wann und wo realisiert und zusammengeführt werden muss.

Entsprechend handelt es sich bei der in diesem Kapitel beschriebenen Version um einen Prototyp, der in einer späteren Phase komplett überarbeitet wurde, um einen Release Candidate zu kreieren. Die bei dieser Neugestaltung gemachten Änderungen werden im Kapitel 6 beschrieben.

Da die Umsetzung des Prototyps wie erwartet ausserordentlich lehrreich und wichtig war, stellt sich dieses Kapitel als sehr umfangreich dar und kann als Hauptkapitel der ganzen Arbeit betrachtet werden.

5.1 Virtuelle Maschinen oder Container?

Eine Installation der zu verwendenden Distribution direkt auf dem Server kam nie in Frage. Diese Variante hätte zu viele Nachteile:

Sicherheit: Die Nutzerinnen und Nutzer des Systems würden so direkt Code auf dem Host ausführen und diesen auf diese Weise leicht übernehmen können.

Ebenso ist geplant, dass die nutzbaren virtuellen Maschinen oder Container einmal pro Nacht von einer Mastermaschine beziehungsweise einem Mastercontaner neu geklont werden. Auf diese Weise bleiben allfällige Manipulationen, unabhängig davon ob diese beabsichtigt sind oder nicht, innerhalb der Nutzermaschinen nicht persistent.

Flexibilität: Jede Schule, die unsere Lösung einsetzen möchte, müsste zusätzlich dafür einen separaten dedizierten Server zur Verfügung stellen. Der Grundsatz der ≪Easy Deployability≫ gebietet es aber, dass das System auf verschiedenen bereits vorhandenen Servern mit möglicherweise bereits vorhandenen Diensten nachinstalliert werden können soll, und das auch möglichst unabhängig von der dort schon installierten Distribution.

Wartung: Ein System von virtuellen Maschinen oder Container erlaubt es, zum Beispiel für Updates zuerst die Mastermaschine beziehungsweise den Mastercontainer zu einer Testingmaschine beziehungsweise Testingcontainer zu klonen, dort alle Änderungen vorzunehmen, zu testen und anschliessend zurückzuklonen. Das macht das System sehr robust und leicht zu warten.

Würde das System dagegen direkt auf dem Server installiert, wäre eine solche Wartungsmöglichkeit nur mit grösserem Aufwand zu realisieren, zum Beispiel durch Bereitstellung eines zusätzlichen modellähnlichen Testservers.

Somit musste eine Entscheidung zwischen einer Containerlösung und einer Lösung mit virtuellen Maschinen (VMs) getroffen werden.

Auch Mischvarianten, zum Beispiel Container innerhalb einer virtuellen Maschine, wurden untersucht. Systeme wie Kata [Foua] setzen solche Lösungen ein, um die Sicherheit zu erhöhen: Ein Angreifer, der den Container verlassen kann, befindet sich immer noch lediglich in einer virtuellen Maschine, also einer weiteren Isolationsschicht, und nicht direkt auf dem Hostsystem. Das schützt ein derartiges Deployement zum Beispiel davor, dass Kernelexploits — ein Container nutzt normalerweise den Kernel des Hostsystems — effektiv genutzt werden können, um sich Zugang zum Host zu verschaffen. Hier würde ja lediglich der Kernel der VM-Ebene angegriffen werden.

Dennoch hat sich herausgestellt, dass auch ein solcher Ansatz keine beliebig hohe Sicherheit bietet. Tatsächlich hat der Sicherheitsforscher Yuval Avrahami von Palo Alto Networks mehrere Techniken entwickelt, um einen Kata-Container vollständig zu verlassen. Diese Techniken hat Herr Avrahami an der Blackhat Konferenz 2020 präsentiert [Avr21].

Somit erkauft man sich mit einem derartigen Ansatz eine gewisse, aber längst nicht umfassende Sicherheit und bezahlt diese gleichzeitig mit deutlich erhöhten Ressourcenanforderungen an den Server.

Wie sich herausgestellt hat, bieten Containerlösungen einige Vorteile gegenüber virtuellen Maschinen. Einige davon sind:

Ressourceneffizienz: Container nutzen die Betriebssystemressourcen effizienter als VMs. Container teilen sich den Host-Kernel mit dem Betriebssystem, während VMs über separate Betriebssysteminstanzen verfügen. Dadurch benötigen Container weniger Speicherplatz, da sie keine vollständigen Betriebssysteminstallationen enthalten. Ausserdem ist der Start und die Ausführung von Containern schneller als bei VMs, da weniger Overhead durch das Starten von Betriebssystemen und Hypervisoren entsteht.

Schnellere Bereitstellung: Container bieten eine schnellere Bereitstellung im Vergleich zu VMs. Container sind in sich geschlossene Einheiten, die alle Abhängigkeiten und Bibliotheken enthalten, um eine Anwendung auszuführen. Dies ermöglicht eine einfachere und schnellere Bereitstellung von Anwendungen ohne die Notwendigkeit, ein vollständiges Betriebssystem bereitzustellen. Im Gegensatz dazu erfordern VMs das Starten eines gesamten Betriebssystems, was mehr Zeit und Ressourcen beansprucht.

Skalierbarkeit: Containersysteme wie Docker [Inc] bieten eine hohe Skalierbarkeit. Container können horizontal skaliert werden, indem mehrere Instanzen einer Anwendung erstellt werden. Durch die Isolierung von Containern können sie unabhängig voneinander skaliert werden, wodurch eine effizientere Nutzung der verfügbaren Ressourcen ermöglicht wird. VMs hingegen erfordern mehr Konfigurations- und Verwaltungsaufwand, um horizontal skaliert zu werden.

Isolation: Container bieten eine gute Isolierung von Anwendungen auf Betriebssystemebene. Sie nutzen die Namespaces und Cgroups des Linux-Kernels, um sicherzustellen, dass Anwendungen in Containern voneinander isoliert sind und ihre Ressourcen nicht beeinflussen.

Diese Isolationsmechanismen gewährleisten bis zu einem gewissen Grad die Sicherheit und Stabilität von Containern. VMs bieten zwar deutlich stärkere Isolationsebenen, indem sie jedes Betriebssystem in einer eigenen virtuellen Umgebung auf virtueller Hardware ausführen, jedoch führt dies zu einem höheren Overhead im Vergleich zu Containern.

Portabilität: Container sind in hohem Masse portabel und können unabhängig von der zugrunde liegenden Infrastruktur konsistent ausgeführt werden. Container enthalten alle notwendigen Abhängigkeiten und Konfigurationen in sich selbst, so dass sie leicht zwischen verschiedenen Umgebungen verschoben werden können. Dies ermöglicht eine reibungslose Entwicklung, Bereitstellung und Skalierung von Anwendungen über verschiedene Systeme hinweg. Im Gegensatz dazu benötigen virtuelle Maschinen spezielle Hypervisoren und zusätzliche Anpassungen, um auf verschiedenen Plattformen ausgeführt werden zu können.

Ökosystem und Orchestrierung: Für Container wurde ein breites Ökosystem von Werkzeugen und Plattformen entwickelt, um die Verwaltung und Orchestrierung von Containern zu erleichtern. Werkzeuge wie Docker und Kubernetes [Aut23b] bieten Funktionen zur Automatisierung, Überwachung und Skalierung von Containern auf Clusterebene. Obwohl auch VMs über Orchestrierungstools verfügen, ist das Container-Ökosystem umfangreicher und bietet mehr Möglichkeiten für ein effizientes Anwendungsmanagement.

Kosten: Wie sich herausgestellt hat, bieten viele preisgünstige Server gar nicht die Möglichkeit, hardwarebeschleunigte Hypervisoren wie KVM zu nutzen. Entsprechende Virtualisierungstechnologien werden nach den Recherchen in der Regel vom Anbieter im UEFI- oder BIOS-Setup deaktiviert und können vom Kunden nicht reaktiviert werden. Damit wären virtuelle Maschinen nur sehr langsam lauffähig, während Containerlösungen sich von dieser Einschränkung völlig unbeeindruckt zeigen und sogar auf virtuellen Servern (vServern) problemlos ihren Dienst verrichten.

5.1.1 Entscheidung: Container

Aufgrund der genannten Vorteile wurde die Entscheidung gegen virtuelle Maschinen und für eine Containerbasierte Lösung getroffen. Ausschlaggebend war hier am Ende hauptsächlich das Kostenkriterium: Die finanziellen Ressourcen sind zu beschränkt, um beliebig teure Bare-Metal-Server, die Virtualisierungen erlauben, zu mieten. Die Autoren dieser Arbeit gehen ausserdem davon aus, dass Schulen, die das System nutzen wollen, ebenfalls finanzielle Überlegungen in ihren Entscheidungsprozess einfliessen lassen werden. Die scheinbar hohe und teure Anforderung an einen Server, virtuelle Maschinen nativ zu unterstützen, wäre also auch gemäss der ≪Easy Deployability≫ kontraproduktiv.

5.2 Container

Containervirtualisierungen bieten eine ressourcenschonende Alternative zu virtuellen Maschinen, um Gastinstanzen eines Betriebssystems voneinander isoliert auf einem Hostsystem zu betreiben. Im Gegensatz zu Virtualisierungslösungen mit Hypervisoren nutzen solche Containersysteme einige Ressourcen des Hostsystems mit. An erster Stelle ist hier der Kernel zu nennen: Containerisierte Betriebssysteme nutzen den Kernel und weitere Ressourcen des Host-Systems und arbeiten daher sehr effizient. Gleichzeitig bieten sie eine etwas bessere Isolation als einfache Chroot-Umgebungen, die in der Regel leicht zu umgehen sind und daher auch nicht als ≪Sicherheitsfeature≫ verwendet werden sollten (≪chroot jail escape≫ [Hac]). Manchmal werden Container daher auch als ≪chroot on steroids≫ bezeichnet.

Ein Nachteil dieser gemeinsamen Nutzung von Host-Ressourcen ist jedoch der Sicherheitsaspekt. Beispielsweise wirkt sich ein Kernel-Exploit im Container direkt auf den Host aus, da es sich um den gleichen Kernel handelt.

Dennoch haben wir uns, wie bereits erwähnt, unter anderem aus Effizienzgründen für eine Containerbasierte Lösung entschieden. Daher folgt nun eine kurze Auflistung der bekanntesten Container-Virtualisierungslösungen.

5.2.1 Docker

Docker [Inc] ist eine freie Container-Virtualisierungslösung, die vorwiegend dazu dient, einzelne Anwendungen mit all ihren Abhängigkeiten als einfach zu installierendes Gesamtpaket (≪Docker-Images≫) zur Verfügung zu stellen. Auf diese Weise können aufwendige Installationsroutinen, die gegebenenfalls diverse Abhängigkeiten und Bibliotheken mitinstallieren müssten, umgangen werden. Gleichzeitig können auf diese Weise installierte Programme und Programmpakete sehr einfach wieder entfernt werden, ohne dass Reste in Form von mitinstallierten Abhängigkeiten auf dem System zurückbleiben.

Aus sicherheitstechnischer Sicht erscheinen Docker-Container den Autoren dieser Arbeit etwas heikel. Der Dienst, der den Dockercontainer startet, benötigte vor Docker Version 19.03 zwingend root-Rechte und auch in späteren Versionen ist eine unprivilegierte Nutzung nur unter bestimmten Umständen möglich [doc].

Es dürfte im für dieses Projekt benötigten Szenario gefährlich sein, einen Container mit privilegierten Rechten auszuführen. Grundsätzlich gilt nämlich, dass Root-Rechte im Container mit Root-Rechten auf dem Host gleichgesetzt werden können. Den Autoren sind mehrere CTFs[1] bekannt, die genau diesen Umstand eingebaut haben: Man verschafft sich in einem ersten Schritt Zugang zu einem Server, befindet sich dann in einem Docker-Container, verschafft sich dort Root-Privilegien und bricht dann aus dem Container aus. Das Ergebnis ist dann, dass man mit root-Rechten auf dem Host landet, weil der Container selbst mit root-Privilegien läuft. Ein solcher CTF ist etwa die ≪Ready≫-Maschine von HackTheBox [Sec21].

Umgekehrt ist es auch möglich, die Root-Rechte des Docker-Daemons zu nutzen, um als relativ unprivilegierter Nutzer, der Docker-Container erzeugen und starten darf, vom Host aus indirekt Root-Rechte auf demselben Host zu erhalten. Dazu wird einfach ein kleiner Container erzeugt, um aus diesem dann wieder auszubrechen. Die ≪Extension≫-Maschine von HackTheBox ist ein Beispiel für ein CTF, in dem diese Methode als Lösungsstrategie der Herausforderung verwendet werden kann[dar23].

Abbildung 5.1: Docker Logo [Inc23b]

Des Weiteren scheint Docker eher für einzelne Anwendungen und weniger für komplette Betriebssysteme geeignet zu sein. Dies liegt auch daran, dass Docker-Images in der Regel immutable, also unveränderbar sind.

Nicht zuletzt neigt Docker nach Erfahrung der Autoren aufgrund seines Netzwerkmodells dazu, bestehende iptables-Regeln (Firewall) auszuhebeln, was oft nicht bemerkt wird. Die Idee dabei ist eigentlich, dass ein Dienst, der in einem Docker-Container installiert ist, im Sinne von ≪just works≫ erreichbar wird. Leider ist es nicht so einfach, dieses Verhalten zu unterbinden, wenn man es nicht möchte [cha].

Aus diesen Gründen scheint Docker keine gute Lösung für dieses Projekt zu sein.

5.2.2 Podman

Podman [Pod] ist ein freies Tool, das, ähnlich wie Docker, zur Verwaltung von Containern verwendet wird. Es gibt einige Unterschiede Verhalten und in den Funktionen zwischen Podman und Docker. Relevant für diese Projektarbeit sind im Wesentlichen dabei:

[1] ≪Capture the Flag≫ (CTF) ist eine spielerische Herausforderung zum Ausprobieren und Trainieren von Methoden und Strategien aus dem Bereich der Cybersicherheit.

Architektur: Podman wurde entwickelt, um ohne einen zentralen Daemon-Prozess zu arbeiten. Im Gegensatz dazu verwendet Docker einen zentralen Daemon, der die Verwaltung der Container übernimmt. Mit Podman können Container direkt als eigenständige Prozesse ausgeführt werden, was eine bessere Integration mit dem Betriebssystem ermöglicht.

Rootless-Modus: Podman unterstützt den sogenannten ≪Rootless≫-Modus, in dem Container ohne Administratorrechte ausgeführt werden können. Im Gegensatz zu Docker ist der Rootless-Modus bei Podman standardmässig aktiviert, während er bei Docker nur mit Einschränkungen nutzbar ist.

Kompatibilität: Podman ist kompatibel zur Docker Kommandozeilenschnittstelle (CLI). Das bedeutet, dass viele der Docker-Befehle auch mit Podman verwendet werden können, was die Migration von Docker zu Podman erleichtert und es ermöglicht, bestehende Docker-Images direkt in Podman zu verwenden. Wenn man in seiner Shell einen entsprechenden Alias von Podman auf Docker gesetzt hat, kann man sogar die von Docker gewohnten Befehle und Befehlsoptionen weiter verwenden, als würde man mit Docker arbeiten. In einer ersten oberflächlichen Betrachtung könnte man also gar zum Schluss kommen, dass es kaum noch einen Grund gibt, Docker zu verwenden und nicht stattdessen alle solchen Container via Podman zu verwalten.

Netzwerkmodell: Podman verwendet für die Netzwerkkonfiguration das CNI (Container Networking Interface), während Docker ein eigenes Netzwerkmodell verwendet. Dies hat unter anderem zur Folge, dass Podman im Gegensatz zu Docker keine Firewall-Regeln ungefragt ausser Kraft setzt.

Unter dem Aspekt der Sicherheit macht Podman daher in der Gesamtbetrachtung einen deutlich ausgereifteren Eindruck. Dennoch scheint auch Podman gerade wegen der hohen Kompatibilität und Ähnlichkeit zu Docker nur bedingt geeignet, komplette Betriebssysteme als eine Art ≪leichtgewichtige virtuelle Maschine≫ zu betreiben.

5.2.3 LXC

Linux-Container [linb], kurz LXC, stellen eine weitere Containervirtualisierungstechnologie dar. Dabei werden ganze Betriebssysteme in einem Container innerhalb eines Host-Betriebssystems erstellt und ausgeführt. Die Linux-Container nutzen dabei, wie alle anderen Containertechnologien auch, den Kernel des Host-Systems, was sie sehr effizient macht.

Es ist auch möglich, innerhalb eines LXC-Containers weitere LXC-Container zu installieren (*nesting*).

Im Gegensatz zu Docker sind bei LXC viele komplette Betriebssysteme als Images verfügbar. Ausserdem sind die Container standardmässig unprivilegiert. Privilegierte Container sind möglich und für bestimmte Anwendungen auch notwendig, aber in der Regel nicht empfehlenswert. Privilegierte Container gelten aus ähnlichen Gründen als unsicher wie Docker mit seinen standardmässig privilegierten Containern.

Während ein Ausbruch aus einem privilegierten Container eine eher triviale Angelegenheit ist[2], konnten auch nach intensiver Recherche keine Informationen über einen möglichen Ausbruch aus einem unprivilegierten Linux-Container gefunden werden. Dies bedeutet natürlich nicht, dass es keine derartigen Sicherheitsprobleme gibt, aber zumindest scheinen die entsprechenden Techniken weniger bekannt zu sein.

[2] In einem privilegierten Container kann das Dateisystem des Hosts eingebunden werden.

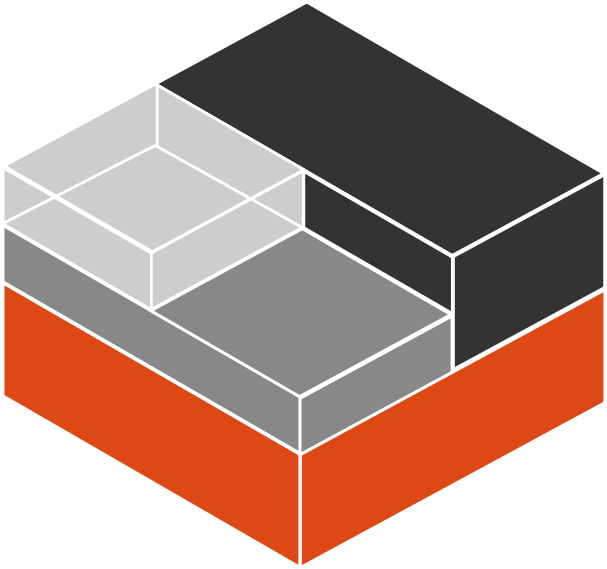

Abbildung 5.2: Linux Containers Logo [Comb]

5.2.4 LXD

LXD [Ltd23b], kurz für ≪Linux Container Deamon≫, ist eine Ergänzung zu LXC. Im Wesentlichen ist LXD ein Management-Toolset für LXC. Linux-Container, die mit Hilfe von LXD erzeugt und verwaltet werden, werden allerdings auch als ≪LXD-Container≫ bezeichnet, was nicht nur wegen des etwas redundanten Namens ≪Linux Container Deamon Container≫ zu einiger Verwirrung führt. In dieser Arbeit wird daher fortan allgemein von LXC gesprochen, auch wenn die Verwendung von LXD gemeint ist. Technisch mag diese Gleichsetzung nicht in allen Details korrekt sein, aber im Kontext dieses Projektes gibt es keinen relevanten Unterschied.

Dank LXD gibt es einige nützliche Werkzeuge für LXC. So ist es zum Beispiel mit `lxd-migrate` möglich, bestehende virtuelle oder auch physische Systeme in einen Linux-Container zu konvertieren [Gra22].

Einen interessanten Sonderfall stellen virtuelle Maschinen dar, die seit LXD Version 4.0 unterstützt werden: Durch Setzen des Flags ≪–vm≫ beim Erzeugen eines Containers wird LXD angewiesen, aus dem Container eine vollständige virtuelle Maschine zu erzeugen, die dann mit Hilfe von QEMU/KVM lauffähig ist [Gra20]. Diese virtuelle Maschine teilt sich dann keinen Kernel mehr mit dem Hostsystem und verhält sich entsprechend anders. Da für diese Arbeit bereits entschieden wurde, aus Performancegründen auf virtuelle Maschinen zu verzichten, hat dieses Feature hier keine weitere Bedeutung. Dennoch ist es interessant zu sehen, wie hier die Grenzen zwischen Containern und virtuellen Maschinen weiter verschwimmen.

5.2.5 Entscheidung: LXC

Linux-Container scheinen für die Zwecke dieses Projektes unter Berücksichtigung von Performance- und Sicherheitsaspekten am besten geeignet zu sein. Es ist vorgesehen, ein System zu schaffen, das aus einem nicht privilegierten Hauptcontainer und mehreren darin enthaltenen, natürlich ebenfalls nicht privilegierten Untercontainern besteht. Auf diese Weise wird einfache Installierbarkeit (lediglich ein grosser

Hauptcontainer muss auf den Server importiert werden) mit Flexibilität und Robustheit (verschiedene Subcontainer für verschiedene Teilaufgaben) kombiniert.

5.3 Server und Hostbetriebssystem

Der Server sollte finanziell tragbar sein, aber genügend Ressourcen bieten, um einer grösseren Anzahl von Schülerinnen und Schülern die gleichzeitige Nutzung des Systems zu ermöglichen.

Da für diese Arbeit der Lernstick als mögliche Zieldistribution ins Auge gefasst wurde, waren die Mindestanforderungen an den Lernstick ausschlaggebend. Leider scheint es in der Dokumentation keine Stelle zu geben, an der die Mindestanforderungen an Arbeitsspeicher beschrieben werden — lediglich bei der Einrichtung des Lernsticks in einer VirtualBox VM ist von ≪deutlich über 1024MB≫ die Rede [Ber21]. Es wurde daher mit etwa dem Zehnfachen dieser Menge, also 10 GB RAM für die im späteren Betrieb aktiv genutzten Desktopumgebungen gerechnet. Dies sollte ausreichen, da das System nicht für zu ressourcenintensive Anwendungen genutzt werden sollte und auch bei vielen parallelen Benutzerinnen und Benutzern stabil bleiben muss.

Zusätzlich wurden nochmals 2 GB RAM für Overhead (Nested Container, Guacamole Server) sowie weitere 2 GB RAM für zusätzliche Container (zum Beispiel Testsysteme) eingeplant. Zusammen mit weiteren 2 GB RAM, die für das Hostsystem sowie als Puffer verbleiben, ergibt sich ein Gesamtbedarf von 16 GB RAM oder mehr. Es bleibt abzuwarten, ob dies richtig eingeschätzt wurde.

Die Anforderungen an den Festplattenspeicher sind im Grunde nicht sehr hoch. Natürlich müssen die Basissysteme mehrerer Betriebssysteme Platz haben, aber Linux-Container scheinen hier sehr schlank zu sein. Der Plan sieht vor, die Homeverzeichnisse auf einen anderen Cloud-Dienst auszulagern, so dass diese lokal nur wenig Speicherplatz benötigen.

Als Host-Betriebssystem soll eine bewährte und für ihre Stabilität bekannte Distribution wie Debian eingesetzt werden.

5.3.1 Entscheidung: Server bei netcup.de

Basierend auf diesen groben Schätzungen wurde entschieden, einen Root-Server vom Typ ≪RS 2000 G9.5≫ bei netcup.de[neta] zu mieten. Das Preis-Leistungs-Verhältnis scheint gut zu sein — ein Besseres konnte zur der Zeit nicht von den Autoren ausfindig gemacht werden. Der Server bietet mit 16 GB RAM für den Anfang ausreichend Arbeitsspeicher und mit einer 320 GB grossen SSD mehr als genug Platz. Zudem ist der Preissprung zum nächstgrösseren Server erheblich.

Als Betriebssystem wurde Debian GNU/Linux in der Version 12 ≪bookworm≫ installiert. Zudem konnte die Domain mit dem eher provisorisch klingenden Namen `linuxcloud.ch` gesichert werden.

Aus Sicherheitsgründen wurde sofort ein Benutzerkonto ≪lxadmin≫ ohne sudo-Rechte, aber mit lxc-Rechten angelegt. Die gesamte Einrichtung, Konfiguration und Administration der Linux-Container auf dieser Ebene wird mit diesem Benutzeraccount durchgeführt. Die Idee dahinter ist, dass jemand, der aus dem Hauptcontainer ausbrechen kann, sich zumindest nur in einer unprivilegierten Rolle befindet. Auch

wenn diese letzte Verteidigungslinie sehr schwach ist[3], ist sie besser als nichts.

5.4 Betriebssysteme in den Containern

Als Betriebssystem innerhalb der Container kommen verschiedene Möglichkeiten in Betracht.

5.4.1 Alpine Linux

Diese spezielle Distribution zeichnet sich durch geringe Grösse und hohe Sicherheitsstandards aus.

Abbildung 5.3: Alpine Linux Logo [Tea23a]

Alpine Linux kommt aufgrund der Sicherheitsfeatures vor allem als Betriebssystem für den *äusseren Container* in Frage. Als Betriebssystem für inneren Container jedoch scheint es zu weit weg vom bekannten Lernstick-System zu sein.

5.4.2 Debian GNU/Linux

Grundsätzlich kann man mit Debian GNU/Linux nicht viel falsch machen, wenn man Stabilität und einen möglichst hohen Anteil an freier Software im Einsatz sucht. Auch für den Einsatz als LXC-Host dürfte Debian GNU/Linux ab Version 12 ≪Bookworm≫ bestens geeignet sein. Gleichzeitig basiert der Lernstick auf Debian, so dass es mit vertretbarem Aufwand möglich sein sollte, dieses System in Richtung einer möglichst starken Ähnlichkeit mit dem ursprünglichen Lernstick-System zu trimmen.

Abbildung 5.4: Debian Logo [Deb20]

[3]Benutzer:innen mit LXC-Berechtigung könnten einen privilegierten Container erstellen und dann das Host-Dateisystem darin mounten, um im Endeffekt Root-Rechte auf dem Host zu erlangen

Somit kommt Debian GNU/Linux sowohl als Betriebssystem für den äusseren, aber auch für die inneren Container in Frage.

5.4.3 Ubuntu

Ubuntu [Ltda] ist eine der am weitesten verbreiteten Distributionen. Ursprünglich basiert Ubuntu auf Debian GNU/Linux und ist daher nicht allzu weit vom Lernstick entfernt.

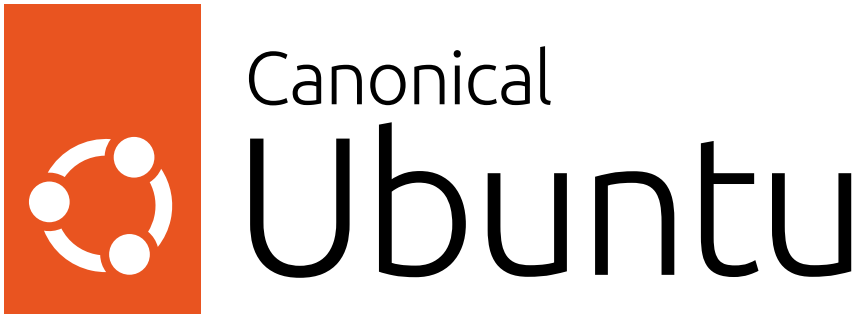

Abbildung 5.5: Ubuntu Logo [Ltdb]

Ubuntu wird von Canonical Ltd. entwickelt — einem kommerziellen Unternehmen. Die Autoren bevorzugen eigentlich Distributionen, die mehr von der Community getrieben werden und nicht von einem Unternehmen abhängig sind.

In der Vergangenheit gab es auch Bedenken hinsichtlich der Datensammlung durch Canonical und der Integration von Amazon-Suchergebnissen in Dash. Canonical hat jedoch auf diese Kritik reagiert und die Datenschutzeinstellungen verbessert [Bro12].

Gleichzeitig ist aber auch festzuhalten, dass LXC/LXD massgeblich bei Canonical Ltd. entwickelt wurde und Ubuntu damit die derzeit wohl «LXC-kompatibelste» Distribution darstellt [lin23c].

Damit kommt Ubuntu auch als Betriebssystem sowohl für die äusseren als auch für die inneren Container in Frage.

5.4.4 Lernstick EDU

Da mit dem Tool `lxd-migrate` eine bestehende virtuelle (oder auch physikalische) Maschine in einen LXC-Container migriert werden kann, sollte auf jeden Fall versucht werden, das ursprüngliche Lernstick-System zumindest als Betriebssystem für die inneren Container zu verwenden. Für den Einsatz als äusserer Container sehen die Autoren hingegen keine Argumente.

5.4.5 Entscheidung äusserer Container: Ubuntu

Für den äusseren Container wurde zunächst Alpine Linux ausprobiert. Die geringe Grösse ist in der Tat beeindruckend. Allerdings gab es dann Schwierigkeiten, LXD/LXC wieder in diesen äusseren Container zu installieren.

Daher wurde als nächstes Debian GNU/Linux in der Version 12 «Bookworm» ausprobiert. Die Installation funktionierte ohne Probleme, allerdings traten anschliessend Schwierigkeiten innerhalb des Containers auf, wenn grössere Pakete nachinstalliert werden sollten: Immer wieder wurde mit der Meldung «Transport endpoint is not connected», darauf aufmerksam gemacht, dass der Dateisystemtreiber von LXC abgestürzt sei, sobald grössere Datenmengen, zum Beispiel im Rahmen eines Updates oder einer Desktop-Installation geschrieben werden mussten. Nach längerer Recherche stellte sich heraus, dass hier eine ACPI-Einstellung Probleme verursacht. Ein Forenbeitrag schlug vor, den ACPI-Dienst abzuschalten [dca22]. Leider bezog sich der Forenbeitrag auf Ubuntu und nicht auf Debian und tatsächlich schien der Dateisystemtreiber bereits beim Abschalten des ACPI-Dienstes abzustürzen. Dieses Vorgehen verlangte nämlich zunächst die Installation der ACPI-Tools und bereits dieser Vorgang hat den Bug schon ausgelöst.

Daher wurde als nächstes Ubuntu 22.04 LTS ausprobiert. Auch hier gab es die erwähnte Fehlermeldung «Transport endpoint is not connected». Allerdings half hier der Tipp aus dem erwähnten Forenbeitrag und das Problem konnte behoben werden.

Daher wurde entschieden, Ubuntu 22.04 LTS als Betriebssystem im äusseren Container zu verwenden. Dieser äussere Container erhielt den Namen «euler» nach dem berühmten Basler Mathematiker Leonhard Euler, auf den sehr viele wichtige mathematische Erkenntnisse und Teilgebiete zurückgehen.

Leonard Euler

Leonhard Euler, geboren am 15. April 1707 in Basel (Schweiz) und gestorben am 18. September 1783 in St. Petersburg (Russland), war einer der bedeutendsten Mathematiker aller Zeiten. Seine Beiträge deckten ein breites Spektrum mathematischer Disziplinen ab und er spielte eine entscheidende Rolle bei der Entwicklung der modernen Mathematik, ihrer Anwendungen und somit auch der Informatik.

Zahlentheorie und Algebra: Euler leistete bahnbrechende Arbeiten auf dem Gebiet der Zahlentheorie und der Algebra. Er bewies den kleinen Satz von Fermat in verallgemeinerter Form und entwickelte die Theorie der quadratischen Residuen. Seine Arbeiten auf diesem Gebiet sind heute auch in der Informatik von zentraler Bedeutung, zum Beispiel für das kryptographische Verfahren RSA.

Analysis und Differentialgleichungen: Euler war ein Pionier der Analysis und legte den Grundstein für die moderne Infinitesimalrechnung. Er entwickelte das Euler-Verfahren zur numerischen Lösung von Differentialgleichungen und trug zur Entwicklung der Taylorreihe bei. Zu seinen Arbeiten in der Analysis gehört auch die Einführung der Eulerschen Zahl e, die eine fundamentale Konstante in vielen Bereichen der Mathematik darstellt, so auch bei Berechnungen in der Informatik.

Graphentheorie: Einer der wichtigsten Beiträge Eulers zur Mathematik waren seine Arbeiten zur Graphentheorie. Er formulierte das erste Ergebnis der Graphentheorie, den so genannten «Eulerschen Polyedersatz». Dieser Satz verknüpft die Anzahl der Ecken, Kanten und Flächen eines Polyeders und legt damit den Grundstein für die moderne Graphentheorie. Heute sind Graphen und ihre Anwendungen in der Informatik allgegenwärtig, insbesondere bei der Modellierung von Netzwerken und Datenstrukturen.

Abbildung 5.6: Leonhard Euler (1707–1783) [Coma]

Viele der Gebiete, auf denen Euler Pionierarbeit geleistet hat — die obige Aufzählung ist bei Weitem nicht vollständig — sind heute für die Informatik von grosser Bedeutung. Seine Beiträge zur Zahlentheorie, Algebra, Analysis, Graphentheorie, Kombinatorik und Wahrscheinlichkeit legten den Grundstein für viele Bereiche der modernen Mathematik und Informatik. Viele Algorithmen und mathematische Konzepte, die in der Informatik verwendet werden, basieren auf den Erkenntnissen und Methoden, die Euler entwickelt hat.

Die Autoren dieser Arbeit erinnern sich mit Freude an die Zeit, in der sie im gleichen Gebäude und in denselben Räumlichkeiten in Basel Mathematik mit der Spezialisierung auf Zahlentheorie studiert haben, wie es Leonard Euler persönlich vor etwa 300 Jahren schon getan hat.

Aufgrund der enormen Vielfalt der Leistungen Leonhard Eulers, der hier als eine Art ≪Universalmathematiker, der alle Teilgebiete überblickte≫ betrachtet werden kann, wird der äussere Container — ein Container, der alle inneren Container überblickt — daher auch *euler* genannt.

Nach der Generierung von *euler* wurde aus Sicherheitsgründen sofort der darin enthaltene Standardbenutzer ≪ubuntu≫, welcher sudo-Rechte besitzt, gelöscht und ein neuer Benutzer ≪lxadmin≫ ohne sudo-Rechte, aber mit lxc-Rechten angelegt. Mit diesem in *euler* enthaltenen Benutzer ≪lxadmin≫ wurde aus den gleichen Sicherheitsgründen wie mit dem Benutzer ≪lxadmin≫ auf dem Hostsystem verfahren.

5.4.6 Entscheidung innere Container: Ubuntu

Zunächst wurde versucht, eine virtuelle Maschine, auf der Lernstick EDU installiert war, mit Hilfe von `lxd-migrate` in einen Container innerhalb von *euler* zu migrieren. Der Prozess löste einen grossen Upload aus, der leider am Ende hängen blieb. Nach insgesamt drei weiteren Versuchen, die alle zum gleichen Ergebnis führten, war die Hoffnung, dass es sich lediglich um ein Problem im Upstream der privaten Internetverbindung handeln könnte, leider unerfüllt.

Es hat sich herausgestellt, dass hier wieder das alte Problem mit dem Dateisystemtreiber aufgetreten ist: In den Logfiles des gerade entstehenden neuen Subcontainers war die bekannte Fehlermeldung ≪Transport

endpoint is not connected» zu lesen.

Damit war leider klar, dass diese Methode vorerst nicht funktionieren würde. Es scheint ratsam zu sein, auf die Behebung dieses Bugs zu warten, bevor ein Einsatz von `lxd-migrate` erneut ausprobiert werden kann.

Aufgrund dieser Erfahrung wurde entschieden, auch für die internen Container auf Ubuntu 22.04 LTS zu setzen.

Auf diese Weise wurden zunächst zwei innere Container generiert und die ACPI-Dienste in diesen sofort deaktiviert:

master Dieser Container enthält das zu verwendende System. Es ist geplant, dass dieser Container regelmässig auf einen oder mehrere im produktiven Einsatz befindliche Container geklont wird und diese dann verwendet werden, während *master* unverändert bleibt, sobald alles einmal fertig installiert ist. So steht dann immer ein «sauberer» Container zur Verfügung.

guacamole Da als Erstes der RDP-Gateway Guacamole ausprobiert werden soll, wurde dafür ein separater Container angelegt. Dies erhöht die Isolation und macht den Hauptcontainer *euler* übersichtlicher und leichter wartbar.

In diesen Containern wurde natürlich aus analogen Sicherheitsüberlegungen wie oben als Erstes das Default-Benutzerkonto «ubuntu» entfernt. Besonders erwähnenswert ist an dieser Stelle, dass es in der verwendeten Ubuntu Variante einen Dienst mit der Bezeichnung «cloud-init»[Ltd23a] gibt, der beim ersten Start unter anderem diesen Standardnutzer «ubuntu» anlegt. Dies betrifft insbesondere auch Klone von bestehenden Containern: Jede neue Kopie eines solchen Containers enthält zwangsläufig wieder diesen Standardaccount inklusive sudo-Rechte, sobald diese Kopie zum ersten mal gestartet wird — unabhängig davon ob man im Originalcontainer das «ubuntu»-Konto schon entfernt hat oder nicht. Daher war es wichtig, diesen Dienst zu deinstallieren und auch aus den sudoers-Einstellungen den Namen «ubuntu» zu entfernen, damit diese potentielle Sicherheitslücke geschlossen wird und auch bleibt. Dieser Umstand war den Autoren bis anhin nicht bekannt.

Während *guacamole* ausser dem durch die Installation von Guacamole ohnehin angelegten «www-data» keinen dedizierten Benutzer-Account benötigt, wurde in *master* für die beiden Autoren jeweils ein Account mit sudo-Rechten und starkem Passwort angelegt, um dort arbeiten zu können. Es ist geplant, diese Accounts wieder zu entfernen, sobald alles fertig konfiguriert ist — ein sudo-berechtigter Account im produktiven System scheint eigentlich nicht notwendig zu sein.

Folgende weitere Container sind geplant:

deployed Dies ist ein oder sind mehrere Container, die am Ende konkret verwendet werden, d.h. mit denen sich die Schülerinnen und Schüler verbinden. Diese werden täglich als Kopie von *master* neu erstellt, um sicherzustellen, dass die eingesetzten Systeme stets sauber sind.

testing Dies ist ein weiterer Klon von *master*. Neue Software oder Systemupdates sollten zunächst in diesem *testing*-Container eingespielt werden, beispielsweise auf Wunsch einer Fachschaft der Schule. Nach erfolgreichem Funktionstest wird *master* als Klon von *testing* neu generiert. Dies verhindert

fehlerhafte Systemupdates und Installationen, die im schlimmsten Fall das System unbenutzbar hinterlassen könnten.

balancer Der Balancer-Container beinhaltet einen Load-Balancing-Proxy-Server, über den sich Besucher der Cloud-Lösung nicht direkt mit Guacamole, sondern mit Balancer verbinden. Je nach Auslastung des Servers leitet Balancer die Anfragen entweder an Guacamole auf dem lokalen Server oder wenn nötig an Guacamole auf einem entfernten Server weiter, auf dem eine identische Installation von *euler* und allen darin enthaltenen Sub-Containern installiert ist.

5.5 Die Anzahl der im Einsatz stehenden Container

Wie schon angedeutet galt es eine weitere Entscheidung zu treffen.

5.5.1 Alle Benutzer im gleichen Container

In dieser Variante gibt es nur einen *deployed* Container, den sich alle Lernenden teilen müssen. Natürlich wären die Homeverzeichnisse mit Hilfe von Linux-Benutzerberechtigungen und Kryptographie strikt voneinander getrennt und alle Benutzerinnen und Benutzer hätten ihre eigenen individuellen Desktops. Das ist unter Linux durchaus möglich: Es gibt bei Weitem noch genügend mögliche User-IDs, selbst innerhalb eines nested Containers. Die gleichzeitige Anmeldung ist auf unseren Rechnern gemäss `/proc/␣sys/kernel/pty/max` auf 4096 Accounts beschränkt. Diese Zahl, die deutlich kleiner ist als die maximale Anzahl möglicher paralleler User-IDs insgesamt, wird von keiner der Gymnasien, an denen die Autoren dieser Arbeit tätig sind, auch nur annähernd erreicht.

Diese Variante hat den Vorteil, dass sie einfacher zu warten ist. Die Benutzerkonten müssen nicht auf vielen Maschinen synchronisiert werden und *guacamole* muss die Benutzer nicht auf viele Maschinen verteilen. Die entsprechenden Konfigurationen bleiben überschaubar.

Der Nachteil ist, dass die Isolation nicht so strikt ist, wie wenn jeder Benutzer seinen eigenen Container hätte.

5.5.2 Jeder Benutzer in einem separaten Container

Technisch scheint es auch möglich zu sein, sehr viele, sogar tausende Linux-Container parallel zu betreiben. Damit wäre es möglich, dass jeder Lernende seinen persönlichen Container zur Verfügung gestellt bekommt.

Offensichtlich sind die Vor- und Nachteile hierbei genau umgekehrt wie bei der vorherigen Variante: Die Isolation ist besser, aber der Wartungsaufwand ist höher.

5.5.3 Entscheidung: Ein Container für alle

Die Autoren sind zuversichtlich, dass eine ausreichende Isolierung mit Hilfe der Bordmittel von Linux erreicht werden kann. Schliesslich ist das Betriebssystem Linux seit jeher als Mehrbenutzersystem konzipiert.

Der Wartungsaufwand für potenziell tausende von Containern, die immer wieder neu geklont werden müssen, steht nach Meinung der Autoren in keinem Verhältnis.

Gegebenenfalls könnte man eines Tages eine Mischvariante einsetzen, wenn sich bestimmte technische Ressourcen als unzureichend erweisen sollten. Beispielsweise könnten dann alle Schülerinnen und Schüler eines Jahrgangs oder mit einem bestimmten Anfangsbuchstaben im Namen in einem gemeinsamen Container untergebracht werden. Solche Mischformen ergeben immer noch eine überschaubare Anzahl von Containern, so dass sich der Pflegeaufwand in Grenzen halten dürfte. Die Benutzerkonten müssen ohnehin zwischen *testing*, *master* und *deployed* synchronisiert werden (zum Beispiel darf eine Passwortänderung innerhalb von *testing* im Betrieb nicht ≪vergessen≫ werden, wenn *testing* wieder von *master* geklont wird). Bei einer nicht allzu grossen Anzahl von Containern ist dies aber mit vertretbarem Aufwand machbar.

Da die Zeit aber noch nicht reif ist, um über solche Mischformen nachzudenken, wurde beschlossen, es zunächst mit einem einzigen Container zu versuchen, der von allen Nutzerinnen und Nutzern gemeinsam genutzt wird.

Einen wichtigen Einfluss auf diese Entscheidung hatte nicht zuletzt die folgende Überlegung: Jeder Untercontainer wird seine eigene IP-Adresse im gleichen privaten Subnet innerhalb von *euler*, der wiederum in einem privaten Subnet innerhalb des Hosts arbeitet, benötigen. Es ist auch unklar, in welchem Subnet der Host wiederum arbeiten muss. Würde nun jeder Nutzer seinen eigenen Container bekommen, dann müsste *guacamole* die Logins später auf entsprechend viele IPv4-Adressen und Ports verteilen. Dies erhöht den Einrichtungsaufand von neuen Accounts und führt mehr potentielle Fehlerquellen ein.

5.6 Desktop-Umgebungen

Es gibt viele mögliche Desktop-Umgebungen in einem Linux-System — zu viele, um sie hier alle aufzulisten. In die engere Auswahl kamen drei davon.

5.6.1 GNOME

GNOME ist eine der bekanntesten und am weitesten verbreiteten Desktop-Umgebungen. GNOME ist auch auf dem Lernstick die Standard-Desktop-Umgebung. Die Oberfläche wirkt aufgeräumt, intuitiv und effizient.

Abbildung 5.7: GNOME Logo [Fouc]

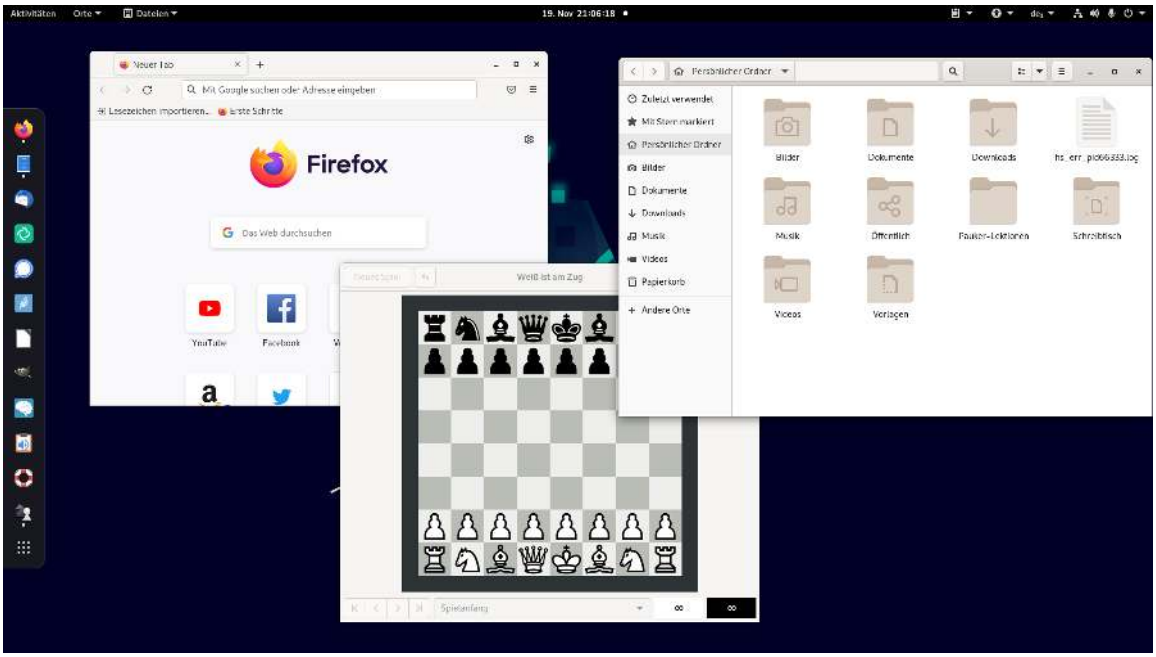

Abbildung 5.8: Screenshot von GNOME unter Lernstick EDU

Das Bedienkonzept unterscheidet sich allerdings etwas von dem von Microsoft Windows. Dies führt zum Nachteil, dass Windows-Benutzerinnen und -Benutzer eine gewisse Eingewöhnungszeit benötigen.

5.6.2 KDE Plasma

Die KDE-Umgebung war in den Versionen 3.x sehr beliebt, verlor aber in den Versionen 4.x aus verschiedenen Gründen viele Anhänger. In den aktuellen 5.x-Versionen ist KDE Plasma jedoch wieder eine erstaunlich ressourcenschonende All-in-One-Lösung, die zudem so gestaltet ist, dass sich Windows-Nutzerinnen und -Nutzer schnell darin zurechtfinden dürften.

Abbildung 5.9: KDE Logo [KDEa]

Abbildung 5.10: Plasma Logo [KDEb]

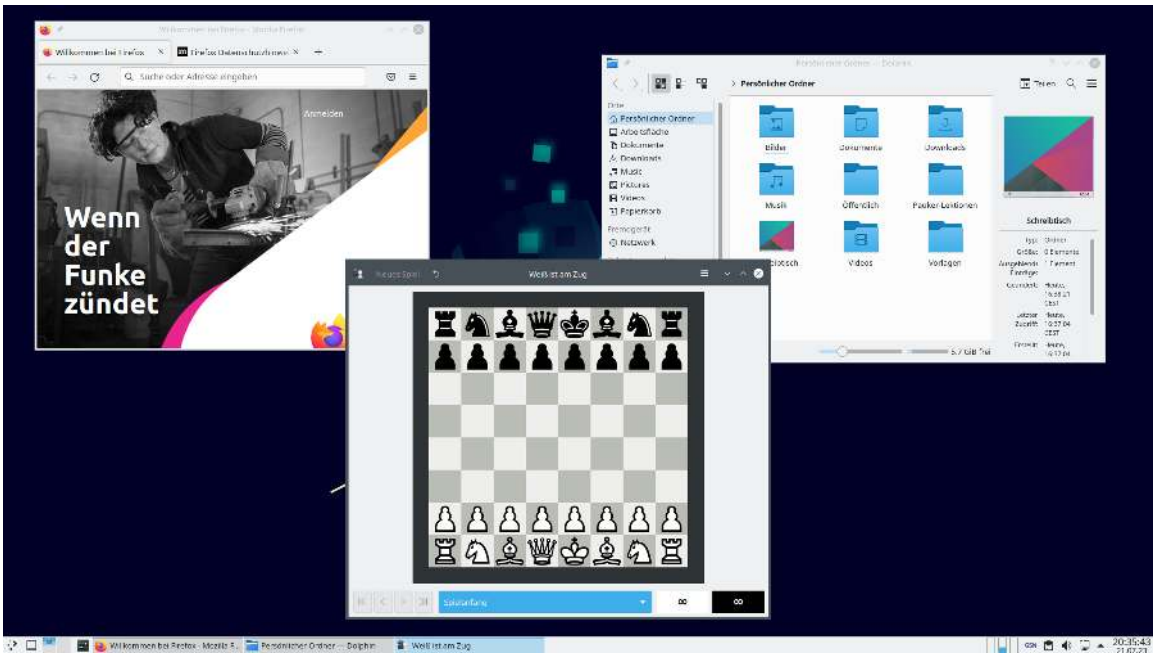

Abbildung 5.11: Screenshot von KDE Plasma unter Lernstick EDU

5.6.3 Xfce

Xfce ist ein sehr schlanker und effizienter Desktop, der auch sehr «windowsartig» konfiguriert werden kann, aber auch viele Features des wesentlich umfangreicheren KDE-Plasma-Desktops nicht mitbringt. So gibt es etwa in Xfce kein E-Mail-Programm, welches das Microsoft Exchange-Protokoll versteht, während KDE mit KMail über ein solches Tool verfügt [KDE20]. An den Schulen des Kantons Basel-Landschaft wurde vor einigen Jahren das IMAP-Protokoll mit Verweis auf «Datenschutz» komplett deaktiviert und seitdem kann nur noch via Exchange mit dem E-Mail-Server kommuniziert werden.

Abbildung 5.12: Xfce Logo [Comc]

Natürlich ist es auch möglich, Programme und Werkzeuge aus KDE in Xfce zu starten, allerdings müssen dann unter Umständen viele KDE-interne Abhängigkeiten mitinstalliert werden.

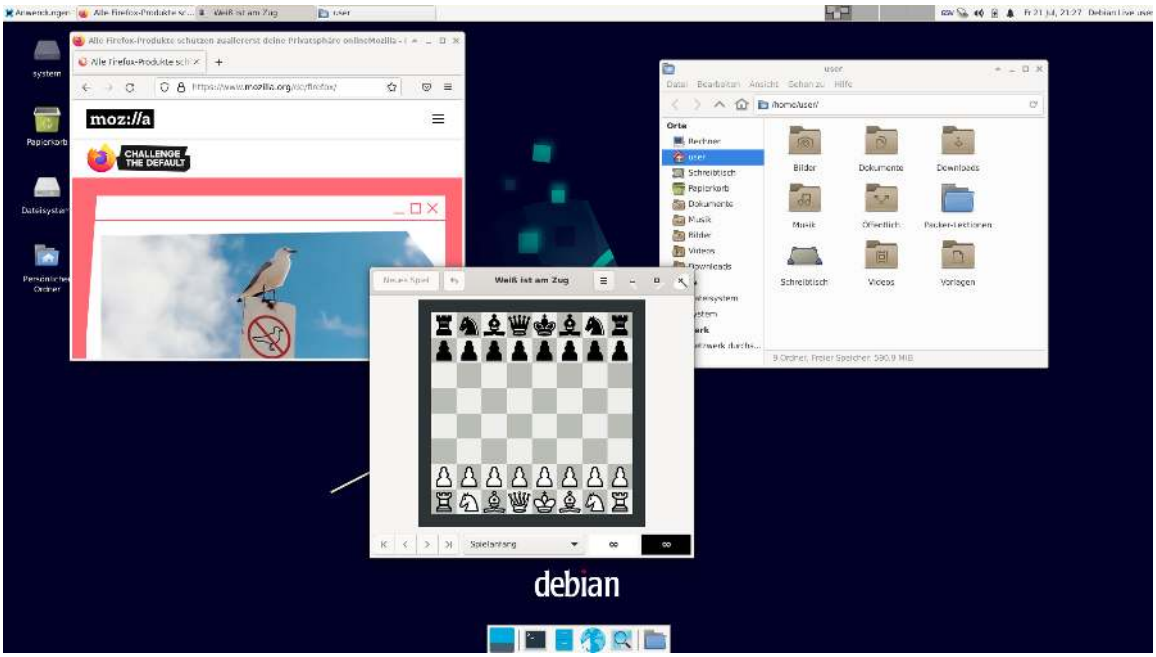

Abbildung 5.13: Screenshot von Xfce unter Lernstick EDU

5.6.4 Entscheidung: KDE Plasma

Obwohl man sich hier etwas vom Look & Feel des Lernsticks entfernt, fiel die Entscheidung zugunsten von KDE Plasma aus. Die Gründe dafür waren im Wesentlichen, dass KDE 5.x für seinen grossen Funktionsumfang erstaunlich effizient ist und sich Windows-Benutzerinnen und -Benutzer schnell darin zurechtfinden dürften. Es wäre in der Tat ein wünschenswertes Nebenziel dieses Projektes, wenn es dazu beitragen könnte, Vorbehalte im Sinne von

> ≪Linux sieht gar nicht so kompliziert aus, wie ich mir das vorgestellt habe≫

abzubauen. Die Autoren des Projekts sehen es daher als sehr hilfreich an, wenn die Schülerinnen und Schüler mit einer Desktop-Umgebung konfrontiert werden, die in Look&Feel und Bedienung nicht allzu weit von dem entfernt ist, was sie sich gewohnt sind.

5.7 Installation von xrdp

Nach der Installation von KDE Plasma auf *master* sollte dieser Desktop über das Internet erreichbar sein.

Wie bereits erwähnt, tendierten die Autoren schon früh dazu, hierfür das Remote Desktop Protocol (RDP) in Betracht zu ziehen. Mit xrdp existiert eine Opensource-Implementierung von RDP, die vereinfacht als grafischer Login-Manager daherkommt, so dass es möglich ist, sich über das Netzwerk an einer grafischen Oberfläche anzumelden. Mehrere gleichzeitige Logins werden unterstützt, indem für jeden neu angemeldeten Benutzer ein neuer Desktop geöffnet wird. Praktisch ist auch, dass umgekehrt die individuelle Session des Benutzers erhalten bleibt: Loggt sich ein Benutzer ein zweites Mal mit dem

gleichen Account ein, sieht er die gleiche Desktop-Session. Es ist auch möglich, das Fenster zu schliessen und sich später wieder anzumelden, um genau an der Stelle weiterzumachen, an der man aufgehört hat.

Die Installation von xrdp kann, wie sich herausgestellt hat, recht aufwendig sein. Es werden von xrdp nämlich auch erweiterte Funktionen wie Audio-Weiterleitung, Drucker-Weiterleitung oder weitergeleitete externe Speichermedien unterstützt, was den Konfigurationsaufwand bei der Installation leider nicht trivial macht.

Glücklicherweise steht ein automatisiertes Installationsskript zur Verfügung, das einem diese Arbeit abnimmt [Gri23a].

Mit Hilfe dieses Installationsskripts wurde xrdp ohne Probleme in «master» installiert.

Die Installation wurde anschliessend von beiden Autoren mit dem RDP-Client «remmina» getestet [Gat23]. Dazu wurde im Rahmen dieser Tests ein SSH-Reverse-Tunnel aufgebaut, um den RDP-Port 3389 noch nicht von *master* zum Host freigeben und dort die Firewall entsprechend lockern zu müssen — dies sollte ohnehin nie nötig sein, da schlussendlich die Verbindung über Guacamole funktionieren soll.

Die Tests verliefen, bis auf die Audio-Weiterleitung (siehe Unterabschnitt 5.7.1), reibungslos und wie erhofft, fand sich jeder Autor in seiner eigenen, frischen KDE-Plasma-Umgebung wieder.

5.7.1 Audio-Weiterleitung

Das oben erwähnte Installationsskript sollte eigentlich auch eine Audio-Weiterleitung implementieren. Leider funktionierte dies nicht auf Anhieb und es wurden keine Audiogeräte angezeigt.

Ein Blick in den Quellcode (Listing 1) des Installationsskripts zeigte, dass dieses (noch) auf den Soundserver PulseAudio [fre23] und nicht auf das installierte PipeWire [Pip22] setzt.

```
750 # step 5 - Create xrdp sound modules
751 cd /tmp
752 /bin/echo -e "\e[1;32m       |-| Compiling and building xRDP Sound modules...     \e[0m"
753 git clone https://github.com/neutrinolabs/pulseaudio-module-xrdp.git
754 cd pulseaudio-module-xrdp
755 ./bootstrap
756 ./configure PULSE_DIR=$PulsePath
```

Listing 1: Ausschnitt aus dem xrdp Installationsskript [Gri23a]

Eine entsprechende Anleitung für PipeWire [Gri23b], die für Ubuntu 22.10 konzipiert wurde, führte zu keinem Ergebnis. Um sie auf Ubuntu 22.04 zu portieren, müssten erhebliche Anpassungen vorgenommen werden. Daher wurde entschieden, PipeWire komplett zu deinstallieren und ausschliesslich auf PulseAudio zu setzen.

Leider wurden danach immer noch keine Audiogeräte angezeigt. Um das Problem zu lösen, wurden daher die Logfiles durchsucht. Dabei fielen die Zeilen von Listing 2 auf.

```
pipewire-pulse[1548]: mod.rt: RTKit error: org.freedesktop.DBus.Error.TimedOut
pipewire-pulse[1548]: mod.rt: could not make thread 1601 realtime using RTKit: Eingabe-/Ausgabefehler
```

Listing 2: Fehlermeldungen von PipeWire beziehungsweise PulseAudio

Es scheint, dass RTKit (RealtimeKit) [hef20] in einem unprivilegierten Container nicht richtig funktioniert (siehe Listing 3). RTKit wird verwendet, um bestimmten Prozessen (insbesondere dem Audioserver) ohne Administratorrechte Echtzeitpriorität zu geben. Dies scheint in diesem System nicht unbedingt notwendig zu sein.

```
rtkit-daemon.service: Failed to set up network namespacing: Permission denied
rtkit-daemon.service: Failed at step NETWORK spawning /usr/libexec/rtkit-daemon: Permission
↪  denied
```

Listing 3: Fehlermeldung von RTKit

Daher wurde beschlossen, RTKit zu deinstallieren. Danach wurden die Audiogeräte endlich wie gewünscht angezeigt und die Tonausgabe funktionierte einwandfrei.

5.8 Installation von Guacamole

Nachdem xrdp sich als soweit funktionsfähig gezeigt hatte, bestand der nächste Schritt darin, diese Verbindung über den Browser zugänglich zu machen. Dazu sollte der gleichnamige Remote-Desktop-Gateway in *guacamole* installiert werden.

Es stellte sich nach einer längeren Fehlersuche heraus, dass die Verwendung des zusätzlichen Containers *guacamole* auch deshalb eine nützliche Entscheidung war, weil so verschiedene Varianten einer Guacamole-Installation mit Hilfe von Container-Snapshots unkompliziert ausprobiert werden konnten. Die Versuche wurden in dieser Reihenfolge durchgeführt:

1. Eine manuelle Installation, die sich als zu kompliziert und fehleranfällig herausstellte, da eine überraschend grosse Anzahl von Abhängigkeiten und zusätzlichen Paketen installiert und konfiguriert werden musste.

2. Eine Installation in einem Docker-Container, die abgebrochen wurde, weil Docker die vorhandenen Firewall- und Portweiterleitungsregeln durcheinanderbrachte.

3. Eine automatisierte manuelle Installation mit einem Installationsskript [iti]. Diese Variante hat sich als gut funktionierend erwiesen.

5.8.1 Konfiguration

Vorerst wurde eine Installation mit selbstsignierten Zertifikaten gewählt. Es ist geplant, zu einem späteren Zeitpunkt ein SSL-Zertifikat von Letsencrypt [Gro] auf dem Hostsystem zu installieren und die entsprechenden Dateien per Softlink an der richtigen Stelle innerhalb des Containers *guacamole* verfügbar zu machen. Auf diese Weise, so die Hoffnung, ist auch für diesen Aspekt eine ≪easy deployability≫ gewährleistet.

Nach erfolgter Installation wurde mit Hilfe des Standard-Administrationskontos ≪guacadmin≫ ein neues Administrationskonto angelegt und danach ≪guacadmin≫ gelöscht.

Anschliessend wurde eine erste RDP-Verbindung zu *master* vorbereitet. Wichtig war hierbei, die Option ≪Ignore Server Certificate≫ zu aktivieren oder besser die entsprechenden xrdp-Zertifikate von *master* in die Guacamole-Konfiguration einzubinden. Da für dieses Vorgehen aber kaum Dokumentation zu finden war und diese Verbindung ohnehin rein Container-intern bleibt, wurde beschlossen, mit dieser Einschränkung leben zu können.

Die RDP-Verbindung zu *master* wurde getestet. Es stellte sich heraus, dass, obwohl *master* direkt von *guacamole* aus erreichbar war, keine Verbindung auf Port 3389 zustande kam. Verschiedene Scans mit dem Netzwerk-Scanner-Tool `nmap` vom Hostsystem, von *euler*, von *guacamole* und auch von *master* selbst haben ergeben, dass der RDP-Port 3389 von innerhalb des *master*-Containers offen ist, aber von allen Instanzen ausserhalb von *master* gefiltert wird.

Eine Überprüfung auf mögliche iptables-Regeln und ufw-Einstellungen ergab, dass keine Firewall-Einstellungen vorhanden waren, die dieses Verhalten erklärt hätten.

Daraufhin wurde OpenSSH testweise auf *master* installiert, und zur Überraschung und weiteren Verwirrung der Autoren war *master* danach von allen Containern aus per SSH erreichbar, insbesondere von auch *guacamole*. Es schien also, als wäre der unprivilegierte RDP-Port 3389 standardmässig blockiert, der privilegierte SSH-Port 22 jedoch nicht. Eine Erklärung für dieses Verhalten konnte bis zum Abgabetermin dieser Arbeit nicht gefunden werden.

Schliesslich konnte das Problem aber gelöst werden, indem die Containereinstellungen von *master* so geändert wurden, dass der RDP-Port 3389 vom Hauptcontainer *euler* an *master* weitergeleitet wird und *guacamole* stattdessen versucht, sich mit *euler* zu verbinden.

Die Verbindungseinstellungen innerhalb von Guacamole erlauben es, die beim Guacamole-Login eingegebenen Anmeldeinformationen an den RDP-Server weiterzureichen. Dies geschieht über zwei Variablen, nämlich `GUAC_USERNAME` und `GUAC_PASSWORD`, die einfach als Default für die jeweiligen RDP-Anmeldeinformationen eingetragen werden müssen. Diese Option kann verwendet werden, um zu vermeiden, dass sich Benutzerinnen und Benutzer zweimal anmelden müssen (einmal an der Guacamole-Oberfläche und ein zweites Mal an der xrdp-Anmeldung), vorausgesetzt, sie haben an beiden Orten immer die gleichen Anmeldeinformationen.

Im folgenden Abschnitt wird beschrieben, wie diese Option genutzt und wie sichergestellt wurde, dass die Passwörter synchronisiert bleiben.

5.9 Accountmanagement

Wie unter Kapitel 4 beschrieben, wurde der Versuch, die Verwaltung der Accounts über das im Arbeitgeberkanton vorhandene Active Directory zu steuern, abgebrochen. Es musste daher ein Verfahren entwickelt werden, das es den Benutzerinnen und Benutzer ermöglicht, auf möglichst einfache Weise Accounts zu erstellen. Dabei waren diverse Aspekte zu berücksichtigen:

1. Es ist sicherzustellen, dass nur berechtigte Benutzerinnen und Benutzer einen Acccount anlegen können.
2. Die Erstellung eines Acccount soll möglichst keine Benutzereingaben verwenden, um einerseits ein sicheres Initialpasswort zu erzwingen und andererseits mögliche Servermanipulationen durch Eingabemasken zu verhindern.
3. Die Login-Informationen sollen auf möglichst sichere Weise zwischen allen Containern, die diese Account-Informationen benötigen, synchronisiert werden.
4. Es soll auch verhindert werden, dass Benutzerinnen und Benutzer für Guacamole und für den Linux-Desktop zwei separate Accounts benötigen. Das wäre zwar etwas sicherer, aber so umständlich, dass die meisten Benutzerinnen und Benutzer wahrscheinlich von diesem System Abstand nehmen würden.

5.9.1 Interface zur Accounterzeugung

Zu diesem Zeitpunkt wurde noch keine für die Nutzerinnen und Nutzer nutzbare Schnittstelle zur Accounterstellung erstellt. Zunächst sollte sichergestellt werden, dass die neuen Accounts korrekt propagiert und synchronisiert werden.

Der Plan sah zu diesem Zeitpunkt vor, dass auf einem beliebigen schulinternen Server eine kleine Website eingerichtet wird, die mit Hilfe eines PHP-Skripts Folgendes ausführt.

1. Die Nutzer der Website müssen sich mit den Anmeldedaten der Schule anmelden. Im Falle der Schulen der Autoren dieser Arbeit würde dies über LDAP geschehen (wie es bereits andere Websites innerhalb des Schulnetzwerks tun), im Falle aller anderen Schulen würde dies auf die jeweils dort übliche Weise geschehen. Dieser Aspekt muss wohl oder übel von Schule zu Schule individuell angepasst werden, sollte aber im Einzelfall keinen grossen Konfigurationsaufwand verursachen.
2. Nach erfolgreichem Login wird ein zufälliges Passwort generiert und dem Benutzer auf der Website angezeigt.
3. Der für die Website verwendete Anmeldename wird zusammen mit dem soeben zufällig generierten Passwort und einem weiteren zufälligen Salt in einer verschlüsselten Datei an einem Ort gespeichert, der von ausserhalb der Schule zugänglich ist. Für die Verschlüsselung der Datei wird dabei ein öffentlicher GPG-Schlüssel von *euler* verwendet.
4. Dieser Ordner wird alle 10 Minuten von einem Cronjob geleert.

5. Auf *euler* schaut alle 5 Minuten (das Intervall sollte kürzer sein als das Löschintervall) ein Cronjob in diesem öffentlichen Ordner vorbei und holt sich eventuell neu hinzugekommene Dateien.

6. Auf *euler* werden die Dateien entschlüsselt und gemäss Unterabschnitt 5.9.4 verarbeitet.

Auf diese Weise kann diese Website sowohl für die Einrichtung eines Kontos als auch für das Zurücksetzen eines vergessenen Passworts verwendet werden.

Sicherheitsüberlegungen

Das GPG-System sollte eine ausreichend starke Verschlüsselung gewährleisten. Da die Benutzer der Website zudem keinen Einfluss auf die Eingabe haben (sie können den Benutzernamen nicht wählen, da sie sich sonst nicht einloggen können; und das Passwort wird für sie generiert), besteht auch keine Gefahr einer möglichen Command-Injection. Für den Fall, dass jemand sein eigenes Passwort setzen möchte, ist das Verfahren so ausgelegt, dass über die Website ein temporäres Passwort gesetzt wird und dieses dann innerhalb der Desktop-Umgebung geändert wird.

Natürlich hat jeder, der root-Zugriff auf *euler* hat, auch Zugriff auf den privaten Schlüssel zum Entschlüsseln dieser Dateien und kann so an die Benutzerkennwörter gelangen. Wie in den Sicherheitsbetrachtungen unter Unterabschnitt 5.9.2 näher erläutert wird, kann aber ohnehin nicht verhindert werden, dass jemand mit root-Rechten auf *euler* an die Benutzerpasswörter (und mehr) gelangt.

Die Autoren, die offensichtlich einen solchen Root-Zugriff haben, finden das schade — sie würden es vorziehen, wenn auch sie keine Möglichkeit dazu hätten, auch nur theoretisch die Initialpasswörter der Nutzerinnen und Nutzer lesen zu können. Ein derart hoher Standard der Datensicherheit scheint aber nicht erreichbar zu sein. Zumindest haben die Autoren auch nach sehr langer Überlegung und Ideensuche keine Möglichkeit dazu gefunden.

Vor der Umsetzung dieses Vorhabens wurde zunächst manuell alles Notwendige auf der Containerseite vorbereitet und getestet. Die folgenden Unterabschnitte befassen sich mit diesen Änderungen.

5.9.2 Modifizierung von `/usr/bin/passwd`

Ändert ein Benutzer oder eine Benutzerin das Passwort mit dem unter Linux üblichen Bordwerkzeug `passwd`, so bekommen weder *guacamole* noch andere Container (wie *master*, wenn die Passwortänderung in *deployed* stattfindet, was normalerweise der Fall sein sollte) diese Änderung mit. Um dieses Problem zu lösen, wurde die Binärdatei `/usr/bin/passwd` auf *master* (von dem später *deployed* ein Klon sein wird) in `/usr/bin/passwd.orig` umbenannt.

In einem nächsten Schritt wurde ein Shell-Skript unter `/usr/bin/passwd`[4] erstellt, das Folgendes macht:

1. Zunächst wird geprüft, ob der aufrufende Benutzer root ist. Ist dies der Fall, wird eine entsprechende Meldung ausgegeben und das Originalprogramm `/usr/bin/passwd.orig` aufgerufen.

2. Ansonsten wird nach dem aktuellen Passwort gefragt, das eingegeben werden muss (silent).

3. Ist dieses Passwort nicht korrekt, wird mit einer Fehlermeldung abgebrochen.

[4]Dieses Skript ist im Ordner `deprecated` im Repo [WW23b] archiviert.

4. War das Passwort korrekt, wird zweimal nach einem neuen Passwort gefragt (ebenfalls silent).

5. Stimmen die beiden Eingaben nicht überein, wird abgebrochen.

6. Ansonsten wird das neue Passwort durch nicht interaktiven Aufruf des Originalprogramms `/usr/bin/passwd.orig` gesetzt und zusätzlich eine Datei im Ordner `/updateguac` angelegt, die den sha256-Hash des gerade eingegebenen Passwortes enthält und die von einem Cronjob auf *euler* weiter verarbeitet und an *guacamole* weitergereicht wird.

Zusammengefasst wurde der `passwd`-Befehl durch ein Skript ersetzt, das dasselbe Ergebnis liefert, aber zusätzlich eine neue Update-Datei erstellt, die später von *guacamole* verwendet werden kann, um die dortigen Benutzerkonten analog zu aktualisieren.

Gedanken zur Sicherheit

Hier wird offensichtlich zumindest kurzfristig mit Klartext-Passwörtern gearbeitet, was auf jeden Fall die Alarmglocken schrillen lassen sollte. Allerdings:

- Die nicht-interaktive Eingabe der Passwörter in das originale `/usr/bin/passwd.orig` erfolgt über Pipes, die nach Recherchen ≪anonym≫ arbeiten, also nur sehr schwer (wenn überhaupt) von anderen Benutzern ausgelesen werden können.

- Gleiches gilt für die Berechnung des sha256-Hashes und die Überprüfung des alten Passwortes. Letzteres wird mit `unix_checkpw` realisiert.

- Die Passwörter werden zwar in Dateien unter `/updateguac` gespeichert (die übrigens innerhalb einer Minute wieder gelöscht werden, siehe Unterabschnitt 5.9.3), aber nur in Form eines sha256-Hashes. Je nach Passwortstärke kann das geknackt werden, das ist den Autoren soweit klar, zumal es sich leider um ungesalzene Hashes handeln muss. Gleichzeitig sind die Dateien aber auch nur von dem jeweiligen Benutzer lesbar, der die Passwortänderung ausgelöst hat (möglicherweise gibt es hier aber eine racing condition, bis die Dateirechte entsprechend gesetzt wurden). Natürlich kann jemand, der root-Rechte erlangt hat, die Dateien immer noch lesen, aber:

- Generell wird es in diesem Projekt nicht als Sicherheitslücke angesehen, wenn der Benutzer root etwas lesen kann, was er nicht sollte. Der root-User kann ohnehin immer alles lesen, und es dürfte nicht möglich sein, dies zu verhindern. Zum Beispiel könnte root über ein neues PAM-Modul einen Keylogger einschleusen, oder für diesen Fall noch einfacher das Shell-Skript unter `/usr/bin/passwd` so modifizieren, dass es ihm auch direkt das Plaintext-Passwort liefert. Man beachte, dass dies unabhängig davon ist, ob `/usr/bin/passwd` wie im vorliegenden Fall bereits durch ein Shell-Skript ersetzt wurde — root könnte auch auf diese Weise das originale Binary durch eine entsprechende Backdoor ersetzen. Und es kann an dieser Stelle gleich angemerkt werden, dass das neue Shell-Skript `/usr/bin/passwd` für normale Benutzer natürlich nicht schreibbar ist.

- Jemand könnte versuchen, unter falschem Namen eine Datei in `/updateguac` abzulegen, um das Passwort eines fremden Benutzers auf dem Guacamole-Login zu ändern. Dies würde aber einerseits nicht bedeuten, dass auch das Passwort auf dem Desktop geändert wird, sondern nur, dass der Guacamole-Login für den betreffenden Zielbenutzer nicht mehr funktioniert, da die Passwörter

zwischen Guacamole und dem Desktopsystem nicht mehr übereinstimmen. Gleichzeitig läuft dieser Angriff aber ohnehin ins Leere, da der Benutzername später über die Besitzverhältnisse und nicht über den Dateinamen erkannt wird. Dass der Dateiname mit dem Benutzernamen identisch ist, hat nur den Grund, dass es in sehr dummen Konstellationen nicht passieren kann, dass ein Benutzer oder eine Benutzerin versehentlich die Passwortänderung eines anderen Benutzers oder einer anderen Benutzerin überschreibt, weil beide fast gleichzeitig eine Passwortänderung ausgelöst haben.

Kurz: Die Autoren dieser Arbeit sehen ein, dass dieses Vorgehen sicherheitstechnisch nicht ganz sauber ist, glauben aber nach langen Überlegungen und Tests nach bestem Wissen und Gewissen, dass sich das Risiko in akzeptablen Grenzen hält.

5.9.3 Propagieren von Passwortänderungen an Guacamole

Auf *euler* schaut ein Cronjob einmal pro Minute im *master*-Container (später wahrscheinlich eher im *deployed*-Container) nach, ob es unter `/updateguac` Dateien gibt. Wenn ja, wird geschaut, welchem Benutzer die Datei gehört und was der Inhalt ist. Es ist zu beachten, dass wenn jemand von ausserhalb des Containers (hier von der ≪Ebene euler≫) so etwas macht, er automatisch root im Container *master* ist und somit den Inhalt der fraglichen Datei lesen kann.

Anschliessend wird `updateguac.sh`[5] im Container *guacamole* aufgerufen, dem als Parameter der gefundene Besitzer (Username) und der ausgelesene Passworthash übergeben wird. Dieses Shell-Skript führt dann folgende Aktionen aus (zu beachten ist, dass dieses Shell-Skript innerhalb des Containers *guacamole* mit root-Rechten ausgeführt wird):

1. Es wird in der Datei `guacamole:/etc/guacamole/user-mapping.xml` nachgesehen, ob der betreffende Benutzer bereits im Guacamole-System existiert. Falls ja, wird sein Passworthash entsprechend aktualisiert.

2. Wenn der betreffende Benutzer nicht existiert, wird ein entsprechender Eintrag mit einer passenden Default-RDP-Verbindung zu diesem Benutzer angelegt und der angegebene Passwort-Hash eingetragen.

Im Anschluss wird, wieder von der ≪Ebene euler≫ aus, die Datei im Ordner `/updateguac` in *master* (später im produktiven Betrieb in *deployed*) gelöscht.

Auf diese Weise ist sichergestellt, dass die Accountinformationen im Guacamolesystem mit denjenigen aus den Desktop-Containern übereinstimmen und auch Passwortaktualisierungen übernommen werden.

Gedanken zur Sicherheit

In dieser Phase wird der SHA256-Hash eines Benutzerpassworts in Umlauf gebracht. Die Sicherheit eines solchen Hashes hängt natürlich von der Stärke des Passwortes ab.

Im Gegensatz zum vorherigen Teil wird der Hash hier sogar als Parameter und nicht als Pipe übergeben. Dies ist prinzipiell von allen Benutzern auf der gleichen Maschine beobachtbar.

[5] Dieses Skript ist im Ordner `deprecated` im Repo [WW23b] archiviert.

Allerdings muss hier berücksichtigt werden, dass dieser Aufruf auf den Ebenen *euler* und *guacamole* stattfindet. Auf diese Ebenen sollte normalerweise nicht zugegriffen werden dürfen. Sollte es jemandem gelingen, aus dem Container auszubrechen, ist ohnehin alles verloren:

- Wer aus dem Desktop-Container ausbricht, landet zwar nicht in der root-Rolle von *euler*, aber in der Rolle des Nutzers lxadmin, der lxc-Befehle ausführen kann. Somit kann dieser Angreifer über

  ```
  lxc exec deployed bash
  ```

  zurück in den Desktop-Container gehen und ist dann dort root, womit, wie im vorigen Abschnitt beschrieben, ohnehin alle Schranken gefallen sind.

- Mit dem gleichen Trick, wie der Ausbruch gelungen ist, könnte der Angreifer dann einfach noch einmal ausbrechen und wäre dann sogar auf dem Host. Immerhin erneut nicht als root, aber er kann dort als lxadmin auch neue, privilegierte, Container erstellen. Darin ist es dann wiederum möglich das Host-Dateisysem zu mounten, womit man sehr einfach an Administratorrechte auf dem Host gelangt.

  Es ist an der Stelle wichtig zu bemerken, dass die Verschachtelung von Containern (nesting) nicht als Sicherheitsfeature gedacht ist, sondern lediglich dazu dient, die ≪Easy Deployability≫ zu gewährleisten.

Ein weiterer Punkt ist, dass ein Angreifer im Prinzip wählen kann, welchen ≪Hash≫ er letztlich an das Shell-Skript in *guacamole* übergibt, indem er seine eigene Datei im Desktop-Container unter `/updateguac` ablegt. Dieser ≪Hash≫ könnte beliebige Befehle enthalten, die in den Aufruf des Shell-Skripts in *guacamole* oder in den darin enthaltenen *sed*-Befehl eingeschmuggelt werden könnten.

Allerdings scheint die Shell so etwas nicht zu interpretieren. Es ist jedenfalls den Autoren auch nach vielen Versuchen nicht gelungen, über diesen Trick eine Command Execution auszulösen.

Analog dazu könnte ein Angreifer auch einen speziell gewählten Dateinamen wählen, der beim Aufruf mit *cat* später innerhalb des Desktop-Containers als root vielleicht eine solche Command Execution auslöst. Aber auch das ist nach vielen Versuchen nicht gelungen.

Gleichzeitig hat ein Rechercheversuch keine Argumente geliefert, warum das grundsätzlich nicht möglich sein sollte.

Insgesamt bleibt also die Einschätzung, dass diese Lösung möglicherweise gefährlich sein könnte — ob und wie das ausgenutzt werden könnte, ist den Autoren aber nicht bekannt.

Somit scheint dieser Schritt im Zweifelsfall einen der erwähnten notwendigen Sicherheitskompromisse im Sinne von ≪best effort≫ darzustellen.

5.9.4 Erstellung neuer Konten: Ein weiterer Hack

Ein Cronjob von root auf *euler* schaut im Verzeichnis `/root/newusers` nach, ob neue Dateien angekommen sind. In Zukunft werden dies die verschlüsselten Dateien sein, die das Accounterstellungs-Tool auf der schulinternen Website erstellt hat. Sie enthalten Benutzernamen und Passwörter.

Falls neue Dateien vorhanden sind, erstellt der Cronjob im Desktop-Container den entsprechenden Benutzer mit dem gewünschten Passwort oder aktualisiert dieses, falls der Benutzer bereits existiert. Hierfür entschlüsselt er die Datei in einer Pipe. Dieses Vorgehen ist möglich, da der Cronjob mit root-Rechten auf den Container zugreifen kann.

Anschliessend wird der SHA256-Hash des Passworts in eine entsprechende Datei, die dem entsprechenden User gehört, unter `/updateguac` im Desktop-Container geschrieben, um diese anschliessend von dem bereits beschriebenen Cronjob, der die Guacamole-Accounts aktualisiert, abholen und verarbeiten zu lassen.

Im Anschluss wird die Datei unter `/root/newusers` auf *euler* gelöscht.

Gedanken zur Sicherheit

Hier werden nicht nur Hashes, sondern auch wieder Klartext-Passwörter in Umlauf gebracht. Die Eingabe des Passwortes in den Desktop-Container erfolgt jedoch sowohl bei der Aktualisierung des Desktop-Accounts als auch bei der Eingabe des Hashes über eine Pipe, die nicht so einfach abgehört werden kann. Auch die Entschlüsselung der mit GPG verschlüsselten Dateien führt nicht zu Klartextdateien, die auf dem Dateisystem abgelegt werden. Festzuhalten ist natürlich, dass hier davon ausgegangen wird, dass kein Coldboot-Angriff erfolgt, bei dem der Inhalt des Arbeitsspeichers ausgelesen werden könnte.

Die Tatsache, dass dieser Cronjob als root läuft und lxadmin auf *euler* somit keinen Zugriff auf diese Dateien hat, bedeutet, dass lxadmin und damit jeder, der aus dem Desktopcontainer ausbricht, die Plaintext-Passwörter nicht sehen kann. Es sollte jedoch nicht vergessen werden, dass, wie oben beschrieben, Containerausbrecher leicht Root-Zugriff erlangen können. Insofern ist diese Hürde nur gering, aber natürlich besser als nichts.

Ansonsten sehen die Autoren hier keine Möglichkeit für eine Nutzerin oder einen Nutzer, dieses Verfahren für böswillige Zwecke zu missbrauchen.

Das heisst natürlich nicht, dass es nicht möglich ist, sondern nur, dass den Autoren auch nach längerem Nachdenken und Ausprobieren keine Möglichkeit eingefallen ist.

Somit wird auch dieser Punkt potenziell als ≪Kompromiss im Sinne von best effort≫ bewertet.

5.9.5 Synchronisation zwischen *master*, *testing* und *deployed*

Zur Erinnerung: Die Container haben folgende Funktionen/Aufgaben:

- *deployed* ist der Desktop-Container, der aktiv genutzt wird.
- *master* ist der ≪saubere Container≫, aus dem *deployed* einmal pro Nacht neu geklont wird.
- *testing* ist ein Container, der von *master* geklont wird, um darin neue Einstellungen, Softwareinstallationen und Updates durchzuführen. Nach erfolgreichem Test wird *master* wieder von *testing* geklont oder, falls das Update fehlschlägt, *testing* einfach verworfen.

Normale Benutzer haben nur Zugang zu *deployed*.

Wenn nun ein Benutzer oder eine Benutzerin innerhalb von *deployed* das Passwort ändert, muss diese Änderung auch an *master* und, falls ein solcher Container bereits existiert, auch an *testing* propagiert werden.

Dies kann auf sehr einfache Weise realisiert werden: Immer kurz bevor *deployed* gelöscht und durch einen neuen Klon von *master* ersetzt wird, werden die aktuellen Dateien `deployed:/etc/passwd`, `deployed:/etc/group` und `deployed:/etc/shadow` aus *deployed* an die entsprechenden Stellen in *master* und gegebenenfalls in *testing* kopiert. Dadurch werden die Benutzerkonten in all diesen Containern entsprechend synchron gehalten.

Gedanken zur Sicherheit

Hier werden keine Klartext-Passwörter ausgetauscht, sondern nur die Hashes in `/etc/shadow`. Da auch hier eine Lösung über Piping gewählt wurde, sollte dieser Vorgang nicht so leicht abzuhören sein. Ausserdem findet er auf der Ebene *euler* statt, und auch hier gilt: Wer schon einmal aus dem Desktop-Container in *euler* ausbrechen konnte, hat in jedem Fall bereits gewonnen.

Die Autoren dieser Arbeit schätzen daher das Risiko dieses Teils des ≪Accountsynchronisations-Hacks≫ als sehr gering ein.

5.10 Verschlüsselte Homeverzeichnisse mit Quota

Der ursprüngliche Plan sah eine recht komplizierte Kaskade von Technologien vor:

1. Jedem Benutzer wird bei der Account-Erstellung eine persönliche Image-Datei mit fester Grösse zugewiesen. Zusätzlich erhält der neue Benutzer eine zufällige Schlüsseldatei und ein GPG-Key-Pair mit dem Benutzerpasswort als Passphrase.
2. Die Passworterneuerungsroutinen werden so erweitert, dass bei jeder Passwortänderung auch die Passphrase des GPG-Schlüsselpaares analog aktualisiert wird.
3. Die zufällig generierte Schlüsseldatei wird verwendet, um das persönliche Festplattenabbild des Benutzers mit LUKS/cryptsetup zu verschlüsseln.
4. Anschliessend wird die Schlüsseldatei mit dem öffentlichen GPG-Schlüssel verschlüsselt und nur in dieser Form gespeichert.
5. Loggt sich nun eine Schülerin oder ein Schüler ein, so wird über ein entsprechendes PAM-Modul der GPG-Keyring dieser Person entsperrt und anschliessend mit der nun entschlüsselbaren Schlüsseldatei das verschlüsselte Image per cryptsetup (≪geöffnet≫), entschlüsselt und als Homeverzeichnis gemountet.
6. Beim Ausloggen des Benutzers wird das Image wieder ausgehängt und geschlossen.

Diese Vorgehensweise hätte eine Reihe von Vorteilen gehabt:

Skalierbarkeit: Die verschlüsselten Images könnten problemlos auf externe Server und Cloud-Anbieter ausgelagert werden, bei Bedarf auch auf mehrere externe Server. Auch die Grösse der Images könnte

mit etwas Aufwand nachträglich verändert werden, zum Beispiel durch einfaches Ersetzen durch Images anderer Grösse.

Sicherheit: Personen mit Zugriff auf den Speicherort der verschlüsselten Images, sei dieser lokal oder extern, können den Inhalt dieser Images nicht lesen. Auch wenn man Zugriff auf die Keyfiles hat (die natürlich wiederum niemals extern gespeichert werden), benötigt man immer noch den privaten GPG-Schlüssel des jeweiligen Benutzers, um dieses Keyfile verwenden zu können. Und auch wenn man in den Besitz dieses GPG-Schlüsselpaares gelangt, benötigt man immer noch das Passwort des Benutzers, um es zu entsperren. Insgesamt sind die Daten damit hervorragend gut vor neugierigen Augen geschützt, sowohl extern als auch intern zwischen den verschiedenen Nutzern. Und selbst ein böswilliger Benutzer mit Root-Rechten muss nun mindestens so lange warten, bis sich der auszuspähende Benutzer eingeloggt hat, und selbst dann sieht er nur während dieser Sitzung die unverschlüsselte Version des betreffenden Userhomes (vorausgesetzt, es wurde nicht bereits vor der Accounterstellung eine entsprechende Hintertür eingebaut).

Quota: Der Ansatz mit einem verschlüsselten Image pro Benutzerin und Benutzer führt gleichzeitig eine Quota-Funktionalität ein: Jede Anwenderin und jeder Anwender kann maximal so viele Daten in seinem Homeverzeichnis ablegen, wie die Grösse des Images zulässt.

Sekundäre Homeverzeichnisse: Sollte das System einmal versagen, weil bei einem der vielen notwendigen Schritte etwas schiefgeht, wird das Homeverzeichnis für die betreffende Benutzerin oder den betreffenden Benutzer nicht gemountet. Das bedeutet aber nur, dass die betreffende Person anschliessend vor einem leeren Home-Verzeichnis steht, nämlich dem ursprünglich bei der Erstellung des Kontos generierten Standard-Home-Verzeichnis (dieses bleibt einfach bestehen und wird durch das Mounten des Images lediglich «unterdrückt»). Die Desktopumgebung ist also prinzipiell weiterhin nutzbar, es fehlen nur die persönlich angelegten Dateien und Einstellungen, bis das Problem behoben ist. Der Einsatz im Unterricht ist aber auch so weiterhin möglich, wenn auch natürlich nicht auf so angenehme Weise.

Aber auch Nachteile sind damit verbunden:

Hohe Ausfallwahrscheinlichkeit: So viele notwendige technische Schritte ergeben zwangsläufig auch viele potenzielle «Points of Failure».

Hoher Overhead: Es ist davon auszugehen, dass insbesondere bei vielen parallelen Nutzern das dann notwendige System der massenhaften transparenten Ver- und Entschlüsselung zu einem deutlich erhöhten Verbrauch an Arbeitsspeicher und anderen Ressourcen des Hostsystems führen wird. Möglicherweise müsste hier also früher auf einen leistungsfähigeren Server gewechselt oder das für später geplante Load-Balancing früher als geplant eingeführt oder erweitert werden.

Schwierigkeiten im Container: Wie in Kapitel 4 beschrieben, hat sich herausgestellt, dass es erhebliche technische Schwierigkeiten gibt beim Versuch, einen solchen Ansatz innerhalb von Linux-Containern zu verfolgen.

Da dieser Lösungsansatz wie beschrieben aufgegeben werden musste, sah die endgültige Lösung anders aus.

5.10.1 gocryptfs

Gocryptfs ist ein in Go geschriebenes, verschlüsseltes Overlay-Dateisystem [rfj23]. Es wurde ausprobiert und hat in ersten Tests zuverlässig funktioniert.

Da sich herausstellte, dass selbst einfache Loop-Mounts innerhalb eines unprivilegierten Containers nicht möglich sind, entpuppte sich die Umsetzung von Quotas in Verbindung mit gocryptfs als eine grössere Herausforderung als zu Beginn gedacht.

Es wurde beschlossen, einen neuen Container *storage* zu erstellen, der sich ausschliesslich um die Bereitstellung der Homeverzeichnisse kümmern soll, indem darin zwei Verzeichnisse angelegt werden:

- Das Verzeichnis `/root/encryptedhomes` wird vom Host zur Verfügung gestellte (und auf Host-Ebene gemountete) Image-Dateien enthalten. Diese Image-Dateien enthalten je ein ext4-Dateisystem und haben eine bestimmte Grösse — zur Zeit 2 GB. Gespeichert werden darin nur per gocryptfs verschlüsselte Daten.
- Das Verzeichnis `/root/decryptedhomes` enthält Mountpoints für die per gocryptfs entschlüsselte Versionen der Inhalte von `/root/encryptedhomes`.

In einem Kompromiss mit der «easy deployability» wurde es notwendig, Image-Dateien auf der Ebene des Hosts zu generieren. Innerhalb der Container fehlen schlicht die Rechte, solche Images zu mounten. Dank gocryptfs konnte aber zumindest die Verschlüsselung innerhalb der Container stattfinden und somit die Voraussetzungen an den Host noch möglichst minimal gehalten werden.

Ein Skript `generate-encrypted-homes.sh` auf dem Host erzeugt eine gewünschte Anzahl solcher Image-Dateien, mountet diese und stellt diese Mountpoints als LXC-disk-Gerät dem Container *euler* zur Verfügung.

Es wäre auch möglich gewesen, per `losetup` aus den Images Loopdevices zu erzeugen und diese direkt als LXC-Disc weiterzuleiten. Dies hätte aber für Probleme bei einem allfälligen Reboot des Hosts geführt: Diese Loopdevices hätten jedes Mal wieder neu erzeugt werden müssen. Die Autoren gehen aber nicht davon aus, dass jeder potenzielle Host auf Systemd setzt, womit eine allgemein funktionierende Lösung für jede mögliche Distribution, die vielleicht auf dem Hostsystem eingesetzt wird, nur sehr schwierig zu erreichen wäre. Mounts können hingegen simpel in der fstab eingetragen werden, so dass sie bei jedem Reboot wieder automatisch durchgeführt werden. Dies ist der Grund, weshalb der Host die Images selbst mountet und nicht als Loopdevices zur Verfügung stellt.

Die Image-Dateien werden mit aufsteigenden Nummern, beginnend bei 1000 benannt. Diese Nummern entsprechend den User-IDs innerhalb der Container.

Der Hauptcontainer *euler* leitet diese Freigaben analog an *storage* weiter, wo sie als Unterverzeichnisse in `/root/encryptedhomes` landen.

Gocryptfs bietet die Option «`-passfile`», die es erlaubt, das Passwort aus einer Datei statt aus der Standardeingabe zu lesen. Diese Option wurde verwendet, um die Ver- und Entschlüsselungsrozesse vollständig nicht-interaktiv zu gestalten. Dazu erzeugt *storage* pro Image-Datei je mit Hilfe von OpenSSL jeweils ein zufälliges, 50 Zeichen langes Passwort und speichert dies im Verzeichnis `/keyfiles` innerhalb

von *storage*. Anschliessend erzeugt *storage* in `/root/decryptedhomes`, wieder pro Image-Datei, einen passenden Mountpoint und verbindet das unter `/root/encryptedhomes` gefundene Image damit. Am Ende liegen in `/root/decryptedhomes` also transparent entschlüsselte Varianten des Inhaltes der Image-Dateien. Was dort gespeichert wird, landet umgekehrt in transparent verschlüsselter Version im jeweiligen Container. Dieser Initialisierungsprozess wird vom Host aus angestossen, sobald dieser neue Images zur Verfügung stellt.

Das Skript zum Erstellen von Accounts wurde entsprechend angepasst, um diese verschlüsselten Quota-Images zu nutzen. Dabei wurde geplant, ein allfällig altes Image, das durch UID-Recycling von einem alten, inzwischen gelöschten User, wiederverwendet wird, in einem Schritt sicher zu löschen. Leider verhindert das Journaling des ext4-Dateisystems in den Images teilweise ein sicheres Löschen durch übliche Tools wie `shred`. Daher wurde versucht, nach regulärem Löschen das Image zumindest per `dd` komplett mit Nullen zu überschreiben. Diese Operation hat sich innerhalb dieser Kaskade an Weiterreichungen vom Host bis in einen inneren Container hinein aber als deutlich zu langsam herausgestellt. Somit blieb es erst einmal bei einem normalen Löschvorgang per `rm`. Immerhin gilt aber auch, dass den Autoren keine ext4-Datenwiederherstellungstools bekannt sind, die ohne root-Rechte auskommen.

Auf *storage* wurde nun ein Systemd-Dienst angelegt, der beim Booten des Containers dafür sorgt, dass alle Verzeichnisse in `/root/ecryptedhomes` und `/root/decryptedhomes` jeweils wieder verknüpft werden. Die jeweiligen individuellen Konfigurationsdateien für gocryptfs werden dazu bei der Initialisierung ebenfalls in `/keyfiles` gespeichert.

Es ist an dieser Stelle wichtig, darauf hinzuweisen, dass das Verzeichnis `/keyfiles` in *storage* an einem anderen Ort gesichert werden sollte. Ohne dieses Verzeichnis lassen sich im schlimmsten Fall die Daten in den Homeverzeichnissen nicht mehr wiederherststellen.

Als Nächstes wurde der OpenSSH-Server in *storage* installiert und gemäss den folgenden Sicherheitsüberlegungen von Abschnitt 5.10.1 konfiguriert.

Weiter wurde ein SSH-Keypaar auf *master* erzeugt, um einen passwortlosen SSH-Login von *master* auf *storage* zu ermöglichen. Anschliessend wurde ein weiterer Systemd-Dienst auf *master* angelegt, der beim Booten des Containers per SSHFS alle Unterverzeichnisse von `/root/decryptedhomes` von *storage* nach `/decryptedhomes` auf *master* mountet, sobald das Netzwerk erreichbar ist.

Nun wurden per rsync die bereits vorhandenen Homeverzeichnisse nach `/decryptedhomes/UID/username` verschoben, wobei hier für *UID* die User-ID und für *username* des jeweiligen Users verwendet wurde. Dann wurden die Verzeichnisse in `/home` durch Softlinks zu den jeweiligen personalisierten Unterverzeichnisse in `/decryptedhomes` ersetzt. Derselbe Prozess wird ausserdem immer dann angestossen, wenn ein neuer Benutzer generiert wird.

Danach funktionierten alle Tests wie erwartet: Erstellt ein Benutzer in seinem Home-Verzeichnis eine neue Datei, so wird diese korrekt gespeichert, ist auch nach einem Neustart des Containers verfügbar und wird auf *storage* mit einer verschlüsselten Version in `/root/encryptedhomes` synchronisiert. Neue Unterverzeichnisse verhalten sich analog und alle Rechteeinstellungen und -änderungen bleiben wie gewünscht erhalten. Ebenso funktioniert die Quota-Regelung: Jede Nutzerin und jeder Nutzer hat nur soviel Platz im jeweiligen Homverzeichnis, wie die Grösse der Image-Dateien auf dem Host ursprünglich

besitzen und dieser Platz wird auch korrekt in den KDE-Tools wie Dolphin angezeigt.

Der Host kann nun auch problemlos die Speicherung der Image-Dateien an einen externen Server auslagern, solange sie nach wie vor lokal gemountet werden können, zum Beispiel über SSHFS. Die Container bekommen von dieser Auslagerung nichts mit und der Inhalt der Images bleibt verschlüsselt.

Insgesamt steht nun also pro Nutzerin und Nutzer ein individuelles Homeverzeichnis zur Verfügung, das verschlüsselt auf einem externen Container (und von dort potenziell wieder auf einem externen Server) gespeichert ist. Damit erhalten alle neuen Container, die von *master* geklont werden, automatisch die gleichen Homeverzeichnisse, so dass hier keine weitere Synchronisierungsarbeit geleistet werden muss.

Allerdings hat sich die Zeit zwischen dem Start des *master*-Containers und seiner Verfügbarkeit für die Benutzerinnen und Benutzer um etwa eine Minute verlängert, da es anscheinend eine Weile dauert, bis das Netzwerk so weit ist, dass der Systemd-Dienst `/encryptedhomes` mounten kann. Das ist nicht weiter problematisch, soll an dieser Stelle aber dennoch erwähnt werden, da es eine deutliche Verlangsamung des Container-Startprozesses darstellt. Linux-Container haben unter normalen Umständen eine beeindruckend hohe Startgeschwindigkeit und stehen praktisch instantan zur Verfügung.

Gedanken zur Sicherheit

Da den Autoren gocryptfs bisher nicht bekannt war, konnte die Robustheit der von diesem Tool verwendeten Kryptographie nur sehr begrenzt eingeschätzt werden. Es existiert jedoch ein Security-Audit, das einige Schwachstellen aufgezeigt hat [Hor17].

Die gefundenen Schwachstellen beziehen sich glücklicherweise nicht auf passive Angreifer, sondern nur auf Angreifer, die zumindest den Chiffretext verändern können. In der Praxis werden solche Angreifer in dem System aber bereits auf einfachere Weise Zugriff auf die unverschlüsselten Daten haben.

Eine offensichtlich gefährliche Sache scheint der passwortlose SSH-Login von *master* auf *storage* zu sein, zumal dieser zu einem root-Login (!) auf *storage* führt. Da aber auf *storage* alle Dateirechte und -besitzverhältnisse in `/home` abgebildet werden müssen, führt kein Weg daran vorbei, dass es der Benutzer root ist, der auf *storage* seinen Dienst verrichtet.

Hier ist allerdings zu beachten, dass der entsprechende private SSH-Schlüssel auf *master* nur von root gelesen werden kann. Wer aber root auf *master* hat, sieht ohnehin auch alle Home-Verzeichnisse direkt und muss dazu nicht in den *storage*-Container wechseln. Insofern wird die effektive Angriffsfläche nicht wirklich vergrössert, wenn man die Praxis betrachtet.

Auch wenn der *storage*-Container in keiner Weise privilegierter ist als der *master*-Container (ausserdem gibt es dort nur den root-Benutzer und sonst keine Accounts), ist nicht auszuschliessen, dass es auf unbekannte Weise möglich ist, als Angreifer über den *storage*-Container mehr zu erreichen, als es über den *master* allein möglich wäre. Sollten, was absehbar ist, die Image-Dateien auf einen externen Storage-Server ausgelagert werden, so braucht allerdings lediglich der Host die entsprechenden Anmeldeinformationen für diesen externen Server zu kennen. Die Container müssen nicht wissen, wo die Images physisch gespeichert sind — sie müssen lediglich die entsprechenden Shares erhalten. Mit anderen Worten: Um an allfällige Anmeldeinformationen für einen externen Storage-Server zu gelangen, müsste ein Angreifer den Container

verlassen können. Es ist nicht auszuschliessen, dass dies möglich ist, aber spätestens dann sind ohnehin alle Dämme geborsten, zumal dann wie erwähnt eine Privilege Escalation auf Host-Ebene mit Hilfe der LXC-Rechte, die ein solcher Ausbrecher dann haben wird, möglich sein wird. Es ist ensprechend sinnlos, darüber zu spekulieren, was dann alles passieren kann, wenn ein Angreifer root auf dem Host (!) hat.

Solche Gefahren lassen sich in dieser Konstellation wahrscheinlich nie ganz ausschliessen. Um sie jedoch zumindest zu minimieren, wurde der sshd-Service auf *storage* wie folgt konfiguriert:

- Die passwortbasierte Anmeldung wurde deaktiviert.
- Die Berechtigung, Port- oder X11-Weiterleitungen einzurichten, wurde entfernt.
- Per chroot wurde die Umgebung auf `/root` beschränkt — die Passwortdateien liegen ausserhalb dieses Verzeichnisses und ist daher nicht einfach per SSH erreichbar.
- Es wurde das Kommando `internal-sftp` erzwungen, wodurch der Dienst nicht mehr für eine Shell, sondern nur noch für SFTP/SSHFS-mounts verwendet werden kann.

Die Autoren gehen davon aus, dass mit diesen Massnahmen die hier aufgetretenen Risiken weitestgehend minimiert werden konnten. Ein potenzieller Ausbruch aus dem Container ist ein generelles Problem und macht dieser SSH-Service nicht per se unsicherer. Zumindest ist den Autoren auch nach viel Recherche keine Technik bekannt geworden, wie man aus einem unprivilegierten Linuxcontainer ausbrechen kann — im Gegensatz zum Beispiel zu einem Dockercontainer. Da LXC Entwicklerteam scheint auch sehr viel Arbeit in das Thema Sicherheit zu stecken.

Ein neuer User, dessen UID rezykliert wurde, benutzt ein Image eines alten Users. Die Daten darauf werden zwar bei der entsprechenden Accounterstellung gelöscht, aber theoretisch wären sie meist noch auslesbar — die Idee, das Image zuerst wenigstens mit Nullen vollzuschreiben, wurde aus Performancegründen aufgegeben. Dennoch gilt, dass man ohne Root-Rechte wohl keine entsprechende Datenwiederherstellung erreichen kann. Natürlich tauchen immer wieder neue Privilege Escalation Techniken auf, aber dieser Aspekt muss wohl unter den zu treffenden Sicherheitskompromissen gelistet werden.

5.11 Die Umsetzung des Accountmanagements

Bereits in einer frühen Projektphase wurde beschlossen, die Erstellung der Konten an einen schulhausinternen Dienst zu delegieren. Auf diese Weise können sehr viele unterschiedliche Szenarien abgedeckt werden.

Entsprechend dem in Unterabschnitt 5.9.1 beschriebenen Plan wurde auf einem internen Server des Gymnasiums Münchenstein eine PHP-Seite aufgeschaltet, die nur über einen Login mit Zugangsdaten der Schulen Basel-Landschaft erreichbar ist. Dieser interne Server trägt den Namen *webapp*, der im Folgenden auch innerhalb dieser Arbeit verwendet wird.

Im vorliegenden Fall erfolgt die Anmeldung über LDAP und der auf dem *webapp*-Server laufende Apache-Webserver kümmert sich die entsprechende Authentifizierung. Auf diese Weise kann dieser Login-Prozess leicht an andere Authentifizierungsmechanismen angepasst werden, bei Bedarf sogar über `.htaccess`, je nach Schulumgebung und vorhandener interner IT-Infrastruktur.

In diesem konkreten Fall lässt der Apache auf dem schulinternen Server Logins von allen Personen zu, die einen Schulaccount im Kanton Basel-Landschaft besitzen — also Lehrpersonen, Schülerinnen und Schüler sowie Mitarbeitende und zwar nicht nur solche des Gymnasium Münchensteins, sondern auch unter anderem jene vom Gymnasium Muttenz, womit unter anderem beide Schulen, an denen die Autoren tätig sind, abgedeckt werden.

Das PHP-Skript `test.php`[6] extrahiert aus den Servervariablen den Benutzernamen des eingeloggten Benutzers und generiert ein zufälliges Passwort. Die Kombination aus Benutzername und Passwort wird in einer verschlüsselten Datei gespeichert, deren Name sich aus dem aktuellen UNIX-Timestamp und einem mit zufällig generierten Teil zusammensetzt. Der Zeitstempel dient dazu, die Reihenfolge solcher Dateien für einen einzelnen Benutzer zu erhalten, falls jemand das PHP-Skript mehrmals in zu kurzer Zeit aufruft.

Entgegen dem ursprünglichen Plan, der unter Unterabschnitt 5.9.1 beschrieben ist, wurde das Kryptosystem von GPG auf RSA umgestellt. Der Grund dafür ist, dass es sich als schwierig erwiesen hat, den öffentlichen GPG-Schlüssel ohne einen Keyring zu verwenden. Der Benutzer ≪www-data≫, unter dem der Apache Webserver auf *webapp* läuft, hat jedoch üblicher- und sinnvollerweise keine Login-Shell. Es hat sich herausgestellt, dass der GPG-Keyring unter solchen Umständen die Arbeit verweigert. Die Generierung und Verwendung eines RSA-Schlüsselpaares mit Hilfe von OpenSSL verursachte hingegen keine Probleme.

Auf diese Weise wurde auf *euler* ein RSA-Schlüsselpaar erzeugt und der öffentliche Schlüssel auf *webapp* kopiert. Das kleine PHP-Skript auf dem *webapp*-Server generiert nun nach erfolgreichem Login ein zufälliges Passwort, legt dieses zusammen mit dem erkannten Login-Namen des lokal eingeloggten Benutzers oder der lokal eingeloggten Benutzerin in einer verschlüsselten Datei ab und speichert diese an einem aus dem Internet öffentlich zugänglichen Ort. Anschliessend wird das Passwort angezeigt und dem Benutzer beziehungsweise der Benutzerin mitgeteilt, dass er oder sie sich in etwa 2 Minuten mit diesem Passwort auf `https://linuxcloud.ch` einloggen kann.

In der Zwischenzeit läuft auf *euler* ein Cronjob, der einmal pro Minute auf diesem öffentlich zugänglichen Ort des *webapp*-Servers prüft, ob eine neue solche Anfrage eingeht. Ist dies der Fall, wird dieser auf *euler* entschlüsselt und der entsprechende Auftrag zur Accounterstellung ausgeführt. Im Falle eines bereits existierenden Accounts wird lediglich das Passwort aktualisiert. Auf diese Weise dient das PHP-Skript, wie bereits im Plan unter Unterabschnitt 5.9.1 beschrieben, gleichzeitig als Passwort-Reset-Tool.

Diese Auftragsbearbeitung hinterlegt auch an einem öffentlich erreichbaren Ort eine kleine Datei, die den aktuellen Status als einen der folgenden Möglichkeiten beschreibt:

- Der Request wurde abgeholt und wird gerade bearbeitet.
- Der Account wurde erstellt und ist bereit.
- Der Account konnte nicht erstellt werden, weil es nicht genügend freien Speicherplatz gibt auf dem Server.
- Die Accounterstellung wurde aufgrund eines unbekannten Fehlers abgebrochen.

[6] Dieses Skript ist im Ordner `deprecated` im Repo [WW23b] archiviert.

Die Nutzerinnen und Nutzer erhalten auf dem *webapp*-Server einen Link zu einer entsprechenden Seite, wo der aktuelle Status abgerufen werden kann. Auf diese Weise werden Nutzerinnen und Nutzer nicht komplett im Dunkeln darüber gelassen, wie weit der Account schon bereit ist. Diese Statusseite muss als zweites PHP-Skript auf dem *webapp*-Server gehostet werden, da auch hier wieder der Benutzername abgefragt werden muss, um den Status korrekt zuzuordnen.

Kurzzeitig wurde überlegt, das PHP-Skript neben dem Benutzernamen auch das schulinterne Passwort, das beim Login auf *webapp* verwendet wird, abfragen und weiterleiten zu lassen. Der Vorteil wäre hier, dass sich die Benutzerinnen und Benutzer auf der linuxcloud-Lösung dann mit dem gleichen Passwort einloggen könnten, das sie auch sonst in der Schule verwenden. Die Nachteile überwiegen jedoch:

1. Wenn das Passwort auf Schulebene geändert wird, müssen die Benutzerinnen und Benutzer diese Änderung auch in der Cloud-Desktop-Umgebung manuell vornehmen. Dies wird kaum jeder tun, was zu Verwirrung führen würde.

2. Auch wenn diese Schulpasswörter nur kurz in einer Variablen des PHP-Skripts sichtbar wären, ist es grundsätzlich ein massiver Vertrauensbruch, wenn Klartext-Passwörter der Nutzerinnen und Nutzer ausgelesen werden. Nur weil man es kann, ist es noch lange nicht moralisch richtig, es auch zu tun. Nicht nur aus rechtlichen Gründen lehnen die Autoren daher ein solches Vorgehen strikte ab.

Da die Accounts auf *master*, *deployed* und gegebenenfalls *testing* synchronisiert werden müssen, wurde in dieser Projektphase beschlossen, einen zusätzlichen kleinen Linux-Container *accounts* zu erstellen, der die zentrale Verwaltung der Accounts auf den Desktop-Containern sowie auf *guacamole* übernimmt. Da dieser *accounts*-Container nur wenig leisten muss, wurde beschlossen, hier auf Alpine Linux zu setzen, um *accounts* eher minimalistisch zu halten.

Es wurde überlegt, einen OpenLDAP-Server auf *accounts* zu installieren. Der Wartungsaufwand erwies sich jedoch als unverhältnismässig hoch — so scheint es zum Beispiel nicht ohne Weiteres möglich zu sein, dass die Benutzerinnen und Benutzer dann selbstständig ihre Passwörter aus der Desktop-Umgebung heraus ändern können.

Im Rahmen dieser Projektarbeit erschien daher der nsswitch-Dienst `libnss-extrausers` geeigneter. Er ermöglicht im Wesentlichen die Verwendung von Zweitversionen von `/etc/passwd`, `/etc/shadow`, `/etc/group` und `/etc/gshadow`, die standardmässig unter `/var/lib/extrausers` abgelegt werden können.

Nachdem die PAM-Dienste so aktualisiert wurden, dass sie auch `libnss-extrausers` berücksichtigen, war der Rest kein Problem mehr: Neue Benutzeraccounts werden im Container *accounts* angelegt und die Desktopcontainer holen sich die benötigten Dateien aus *accounts* wie `/etc/passwd` und `/etc/shadow` per SSHFS nach `/var/lib/extrausers`.

Das Ergebnis funktionierte wie gewünscht: Alle neu angelegten Accounts erschienen in nutzbarer Form in den Desktop-Containern und auch Manipulationen wie Passwortänderungen waren problemlos möglich. Da die Homeverzeichnisse nun bekanntlich in den Quota-Images erzeugt werden, waren auch diese Userhomes korrekt verfügbar.

Bei dieser Gelegenheit wurde auch gleich analog die Datei `/etc/guacamole/user-mapping.xml` von

guacamole nach *accounts* verschoben und in *guacamole* per SSHFS neu verfügbar gemacht. Hintergrund dazu waren erste Überlegungen in Richtung der Skalierbarkeit: Kommen eines Tages sekundäre Server hinzu, so können die dort platzierten Container ebenfalls per SSHFS direkt auf die Container *accounts* und *storage* des primären Servers zugreifen, womit alle Synchronisierungen bezüglich Accounts und Userhomes automatisch gegeben sind. Dies ist auch der Grund, warum hier auf SSHFS zurückgegriffen wurde und nicht auf NFS, das gemäss der Erfahrung der Autoren nur unverschlüsselt problemlos funktioniert. Ebenso wurde aus diesen Überlegungen heraus auf eine Lösung über LXC-intern freigegebene Ordner verzichtet.

5.11.1 Einrichtung des Administrator-Zugangs

Um die Administration des Systems in Zukunft so einfach wie möglich zu gestalten und die Notwendigkeit zu vermeiden, direkt über den Hostserver auf die einzelnen Container zugreifen zu müssen, wurde ein Account mit dem Namen ≪administrator≫ erstellt. Dieser Account hat sudo-Rechte und die User-ID 999. Es wurde bewusst darauf verzichtet, dem ≪administrator≫ ein verschlüsseltes Homeverzeichnis oder ein Quota zuzuweisen. Letzteres wird erst bei User-IDs ab 1000 angenommen. Dieser Account ist nicht für die normale Nutzung vorgesehen, was auch durch ein verändertes, rotes Hintergrundbild signalisiert wird.

Speziell am ≪administrator≫-Account ist, dass dieser Nutzer auf guacamole-Ebene auswählen kann, ob er sich mit *deployed*, *master* oder *testing* verbinden möchte. Je nach Wahl erscheint auf dem Desktop eine andere Meldung, die darauf hinweist, in welchem Container man sich gerade befindet und was der Zweck dieses Containers ist. Ebenfalls wird auf dem Desktop stets die aktuelle Anzahl an freien Images für die Userhome-Quotas angezeigt.

Geplant wurde zu dem Zeitpunkt, dass der ≪administrator≫ eine kleine Reihe von Tools zur Verfügung gestellt bekommt, zum Beispiel um die Erzeugung von Löschanträgen von Accounts (siehe Unterabschnitt 5.11.2 zu erleichtern.

5.11.2 Entfernen von Usern

Es wurde in den Desktopcontainern ein Verzeichnis `/delrequests` angelegt, das dem User ≪administrator≫ gehört, der als einziger Schreibrechte darin besitzt. Legt der ≪administrator≫ eine Datei, deren einziger Inhalt ein Benutzername ist, dort ab, kann er so *euler* signalisieren, dass der betreffende Benutzer gelöscht werden soll.

Auf *euler* läuft ein Cronjob, der einmal nachts den Container *deployed* aus *master* frisch klont. Bevor er das jedoch tut, schaut dieser Cronjob in den Desktopcontainern nach, ob es solche User-Löschanträge gibt. Diese werden abgearbeitet, indem

1. Der Account aus dem Container *accounts* gelöscht wird.
2. Der entsprechende Eintrag aus der `/etc/guacamole/user-mappings.xml` aus dem Container *guacamole* entfernt wird.
3. Der Löschantrag schlussendlich entfernt wird.

Auf diese Weise ist es dem ≪administrator≫ einfach möglich, von den Desktopcontainern aus Accounts zu entfernen. Konkret berücksichtigt werden aber aus Sicherheitsgründen nur im Container *master* platzierte

Löschanträge.

5.11.3 Gedanken zur Sicherheit

Die Passwörter werden auf der Website auf *webapp* generiert und angezeigt. Die Initialpasswörter der Benutzerinnen und Benutzer werden also grundsätzlich, wenn auch nur kurz, im Arbeitsspeicher des *webapp*-Servers im Klartext gehalten und auch so via https an die Benutzerinnen und Benutzer übermittelt. Dies ist einerseits natürlich heikel, da die IT-Infrastruktur der Schulen im Falle des Kantons Basel-Landschaft einen SSL-brechenden Zwangsproxy enthält. Andererseits ist aber auch vorgesehen, die Benutzerinnen und Benutzer beim ersten Login zu einem Passwortwechsel zu ermuntern oder wenn möglich sogar zu zwingen.

Die generierten Passwörter haben eine Länge von 9 Zeichen und bestehen aus Zeichen des base64-Alphabets. Diese Kombination scheint — zumindest für den Einsatz als Initialpasswort — ein akzeptabler Kompromiss zwischen Passwortentropie und Benutzerfreundlichkeit zu sein.

Das RSA-Kryptosystem sollte nach dem derzeitigen Stand der zahlentheoretischen Forschung einen hinreichend guten Schutz gegen unbefugtes Auslesen bieten.

Neu müssen *guacamole* und auch die Desktop-Container passwortlosen SSH-Zugriff als root auf *accounts* haben. Dies scheint zunächst ein offensichtliches Problem zu sein. Die Gefahr relativiert sich aber, wenn man bedenkt, dass in *accounts* nichts steht, was nicht schon direkt in *guacamole* oder in Desktop-Containern lesbar wäre: Der Container *accounts* enthält die Benutzerkonten in Form der Dateien `/etc/passwd`, `/etc/shadow`, `/etc/group` und `/etc/gshadow` sowie `/etc/guacamole/user-mapping.xml`. Ansonsten ist der Container ≪leer≫ und enthält lediglich eine Standard-LXC-Installation von Alpine Linux. Kurzum: Es gibt dort nichts, was einem potenziellen Angreifer oder einer potenziellen Angreiferin mehr Informationen oder Zugriffe verschaffen könnte, als es bereits auf *guacamole* oder den Desktop-Containern selbst gibt. An dieser Stelle darf nicht vergessen werden, dass jemand, der in den Besitz eines gültigen privaten SSH-Schlüssels gelangt, bereits Root-Rechte auf einem Desktop-Container (oder *guacamole*) erlangt haben muss.

Auch das Löschen von Benutzern oder das Ändern von Passwörtern wäre direkt auf den Desktop-Containern möglich und würde keinen Zugriff auf *accounts* erfordern.

Ausserdem verhindert `libnss-extrausers` standardmässig, dass UIDs kleiner als 500 in die extrauser-Dateien übernommen werden. Damit ist auch sichergestellt, dass sich niemand über *accounts* als zweiter Benutzer mit der UID 0 eine Hintertür hinterlassen kann[git].

Sollte es jemand schaffen, sich schreibenden Zugang zu `/delrequests` zu verschaffen, um gezielt Nutzeraccounts zu löschen, so hat diese Angreiferin oder dieser Angreifer bereits weitgehende Rechte erlangt: Entweder bereits root-Rechte oder zumindest Rechte des ≪administrator≫-Nutzers. Da letzterer über sudo-Rechte verfügt, ist von dort aus der root-Account in greifbarer Nähe. Somit gilt, dass in so einem Fall unweigerlich schon sehr viel schiefgelaufen sein muss und dies kein gesondertes Problem dieses Ansatzes mit den Userlöschanfragen darstellt. Dennoch wurde beschlossen, nur Löschanträge, die auf dem Container *master* hinterlegt wurden, zu berücksichtigen. Ein Angreifer oder eine Angreiferin, die root-Rechte im Desktop-Container erlangt, ist, zumindest in erster Linie, gefangen im *deployed*-Container, dessen Löschanträge ignoriert werden. Ausserdem ist es zumindest nicht möglich, eine persistente root-Backdoor in

deployed zu installieren, da *deployed* — aus genau dieser Überlegung heraus — regelmässig frisch generiert wird. Eine allfällige Privilege Escalation müsste also von einer böswilligen Nutzerin oder einem böswilligen Nutzer immerhin jeden Tag erneut durchgeführt werden.

5.12 Load Balancing

Auch wenn es die begrenzten allgemeinen Ressourcen der Autoren nicht erlauben, die Performance des Systems unter der Last vieler paralleler User eingehend zu testen, ist dennoch davon auszugehen, dass die Serverkapazitäten, sowohl auf Hostebene als auch auf der Ebene der Desktop- und Guacamole-Container an Grenzen stossen werden. Daher wurde schon früh in der Konzeptphase überlegt, wie die Skalierbarkeit durch Verteilung der Last auf mehrere Server grundsätzlich umsetzbar sein könnte.

Die Idee war nun die, dass der Hauptcontainer *euler* auf mehreren Servern installiert werden kann. Ein Server wird dabei als ≪Primärserver≫ definiert, alle anderen als ≪Sekundärserver≫. Die Funktionen sollten dabei wie folgt verteilt werden:

Primärserver Dieser Server enthält alle Untercontainer und Konfigurationen um selbstständig zu funktionieren, sowie eine Liste der Sekundärserver und einen zusätzlichen Untercontainer *balancer*, der sich um das Load Balancing kümmert.

Sekundärserver Der Sekundärserver enthält nur die Untercontainer *guacamole* und *deployed* (alle anderen Untercontainer werden nach der Installation von *euler* entfernt). *deployed* wird dabei regelmässig vom Primärserver her kopiert. Der Sekundärserver kennt die Adresse des Primärservers und leitet alle notwendigen SSHFS-Verbindungen an *storage* und *accounts* an die jeweiligen Container im Primärserver weiter. Ein Sekundärserver erlaubt nur eingehende Verbindungen, die vom Loadbalancer des Primärservers her kommen, so dass man diese als Nutzerin und Nutzer nicht ≪direkt≫ verwenden kann.

Wie erwähnt wurde also in *euler* ein weiterer Untercontainer *balancer* erzeugt. Auf diesem wurde HAProxy[hap] installiert und mit folgenden Details konfiguriert:

- Es wurde der Modus ≪http≫ gewählt, damit sticky Sessions überhaupt möglich werden. Dazu muss aufgrund der dabei notwenigen SSL-Terminierung das von Guacamole genutzte (und auf dem Host per letsencrypt erzeugte) SSL-Zertifikat mit *balancer* geteilt werden.
- In eingehenden Verbindungen soll ein technisches Cookie eingepflanzt werden, um die Sessions wiedererkennen zu können.
- Eingehende Verbindungen vom *webapp*-Server der Schule sollen stets nur an *guacamole* des Primärservers weitergeleitet werden und nicht an Sekundärserver. Das hat den Grund, dass für die Status-Seite der *webapp*-Server zuverlässig Dateien, die auf dem Primärserver liegen, abholen können muss.
- Als Balancing-Modus wurde≪leastconn≫ eingestellt. Der Default ≪roundrobin≫ würde lediglich die Verbindungsanfragen in rotierender Weise durch die Liste der Backends reichen — in diesem Projekt kann es aber sein, dass einige User über längere Zeit, zum Beispiel über eine Doppellektion hinweg,

verbunden bleiben müssen, während sich andere nur kurz zum Testen anmelden. Die Variante «roundrobin» könnte in solchen Fällen zu sehr ungleichen Verteilungen führen.

- Als Backends müssen, wohl während dem Bootstraping-Prozess, alle Sekundärserver, welche die Verbindung direkt nach *guacamole* aufnehmen, aufgelistet werden, sowie der eigene, lokale *guacamole*-Container des Primärservers selbst.

So konfiguriert schien soweit alles zu funktionieren. Leider ist hier eine wesentliche Einschränkung der Ressourcen der Autoren festzuhalten: Da es zur Zeit keinen «Sekundärserver» gibt, konnte nur getestet werden, ob *balancer* die eingehenden Verbindungen an *guacamole* des Primärservers weitereleitet — einen weiteren Eintrag in der Liste gibt es bis jetzt nicht. Im Rahmen dieser Projektarbeit kann also nicht getestet werden, ob einer (oder mehrere) allfälliger «Sekundärserver» korrekt berücksichtigt werden. Allerdings gibt es auch wenig Grund zur Annahme, dass dies nicht funktionieren sollte — HAProxy ist schliesslich genau dafür gemacht.

Aufgrund des technisch notwendig gewordenen Sessioncookies wurde recherchiert, ob dieses eine Cookiehinweispflicht impliziert. Dies scheint aber für solche rein technisch notwendigen Cookies nicht der Fall zu sein[Dat23].

5.13 Annäherung an Lernstick EDU

Wie in Unterabschnitt 5.4.6 beschrieben, war es bisher nicht möglich, den Lernstick EDU direkt als inneren Container zu verwenden. Daher wurde versucht, das Ubuntu-System dem Lernstick EDU so weit wie möglich anzugleichen. Im Einzelnen wurden die Liste der im Lernstick EDU installierten Programmpakete und alle Einstellungen, soweit möglich und sinnvoll, in das System übernommen.

Es mussten ohnehin einige Anpassungen am System vorgenommen werden, so dass dieser Unterschied zwischen dem Cloudsystem aus diesem Projekt und dem Lernstick-System nicht mehr so sehr ins Gewicht fallen sollte. Zu beachten ist jedoch, dass unser System nun auf Ubuntu 22.04 aufgebaut ist und nicht mehr auf Debian 11, auf dem der Lernstick in der Version zum Zeitpunkt der Erstellung dieses Projekts basierte.

Die Tatsache, dass unterschiedliche Distributionen im Lernstick und in diesem Projekt verwendet werden, verlangt es jedoch, dass so wenige Pakete — vorrangig systemrelevante Pakete — wie möglich im Ubuntu-System installiert werden, die für Debian 11 ausgelegt sind. Innerhalb der Debian-Welt spricht man bei einer entsprechenden Mischung verschiedener Releases von einem sogenannten FrankenDebian [Don23]. Weiter gibt es im Lernstick einige Pakete, wie Pauker [Sta], die nicht in den Standard-Repositories von Ubuntu enthalten sind, aber bequem über ein Repo von Lernstick installiert werden können.

Um einer möglichen «Vermischung» von Versionen und entsprechenden Abhängigkeiten entgegenzuwirken, wurden Pakete aus den Lernstick-Repositories durch einen Eintrag in `master:/etc/apt/preferences.d/lernstick_pin` jedoch mit einer geringeren Priorität versehen.

```
Package: *
Pin: origin "*lernstick.ch*"
Pin-Priority: 300
```

Listing 4: `master:/etc/apt/preferences.d/lernstick_pin`

5.13.1 Anpassungen an die Liste der installierten Programmpaketen

Einige der Programme aus der langen Liste der auf dem Lernstick installierten Pakete sind im Kontext dieses Projekts nicht sinnvoll und wurden daher entfernt:

- Da die Entscheidung für KDE Plasma bereits in Unterabschnitt 5.6.4 getroffen wurde, konnten die Desktopumgebungen Cinnamon [Teab], GNOME [Pro], LXDE [con23b], MATE [Tea23b], Xfce [Tea23c] aus der Liste entfernt werden. Ein Wechsel der Desktopumgebung wird für die Nutzerinnen und Nutzer nicht ohne Weiteres möglich sein.
- Da die Benutzerinnen und Benutzer keine Software nachinstallieren können, wurden Programme wie Synaptic [Paa04], lernstickwelcome [Ler23] und andere grafische Programme zur Installation von Programmpaketen entfernt.
- Die Speichermedienverwaltung dlcopy vom Lernstick wurde ebenfalls entfernt, da diese in dem Kontext dieses Projektes nicht funktionieren dürfte.

Abbildung 5.14: Screenshot der Speichermedienverwaltung des Lernsticks

- NetworkManager [netb] wurde entfernt, da die Netzwerkverbindung von LXC zur Verfügung gestellt wird und nicht verändert werden kann.
- Das Programm gparted [gpa] wurde entfernt, da es keinen Sinn ergibt, wenn die Benutzerinnen und Benutzer im Container die Partitionen verändern können. Ohnehin hätten sie dazu gar keine

Berechtigung.

- Der Bootloader grub [gnu21] wurde entfernt, da das System damit nicht bootet.
- Das Programm Transmission [tra] wurde entfernt, da das System nicht für BitTorrent-Downloads vorgesehen ist. Einerseits ist die rechtliche Situation hierzu nicht immer klar, andererseits ist der Speicherplatz in den persönlichen Nutzerverzeichnissen sowieso zu begrenzt dafür.
- Das Paket basisschrift wurde vorläufig entfernt, da die darin enthaltene Schrift urheberrechtlich geschützt ist. Genauer ist in den Nutzungsbedingungen [EEB] Folgendes zu lesen:

  > ≪Die Schrift kann für Lehrerinnen und Lehrer für Materialien, die sie in ihrem Unterricht verwenden, frei verwendet werden. Auch Lehrmittelautoren, Lehrmittelproduzenten und Verlage können die Schrift für Produkte, die zur Verwendung im schulischen Kontext in der Schweiz entwickelt werden, kostenfrei verwenden. Sie sind allerdings verpflichtet, die BKZ Geschäftsstelle über ihre Produkte zu informieren.≫

  Da das Projekt noch sehr neu ist, ist noch keine entsprechende Meldung an die BKZ Geschäftsstelle erfolgt. Sollte dies zu einem späteren Zeitpunkt erledigt sein, kann das Paket aber auf einfache Weise nachträglich integriert werden.
- Aus Performancegründen mussten leider das Programm Stellarium [Ste23] und das Spiel Extreme Tux Racer [Bel+23] entfernt werden. Beide waren auf dem virtuellen Desktop nicht sinnvoll nutzbar. Da nicht alle Programme und Spiele getestet werden konnten, ist damit zu rechnen, dass weitere Programme aus dem gleichen Grund entfernt werden müssen.
- Das Programm youtube-dl wird in Ubuntu 22.04 LTS in der veralteten Version 2021.12.17-1 ausgeliefert. Diese Version funktionierte nicht mehr einwandfrei. Deshalb wurde youtube-dl durch eine aktuelle stabile Version (2023.09.24) des Forks yt-dlp [yt-23] ersetzt.

Auf der anderen Seite wurden die folgenden Programme hinzugefügt:

Xcas Dies ist ein freies Computer-Algebra-System, das einer der Autoren bereits erfolgreich in einer schriftlichen Maturaprüfung eingesetzt hat [PG23].

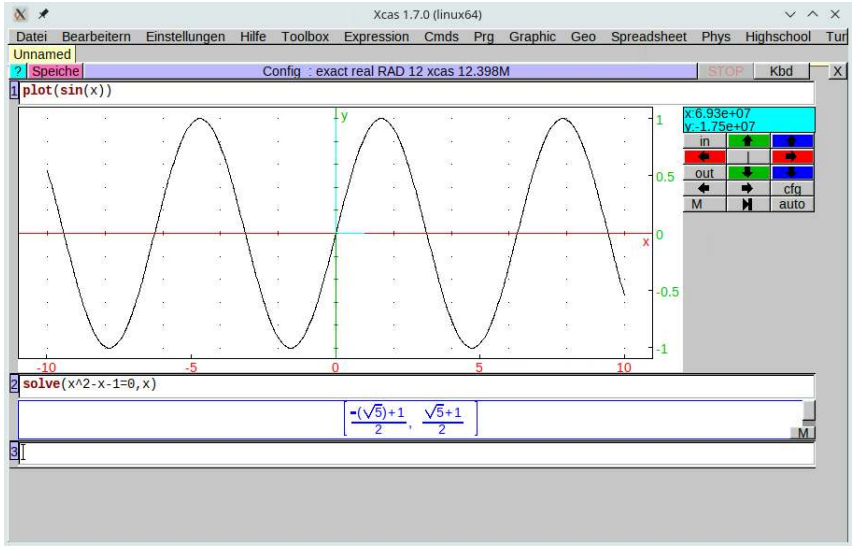

Abbildung 5.15: Screenshot von Xcas

Qalculate! Ein wissenschaftlicher Taschenrechner mit vielen Funktionen [Qua].

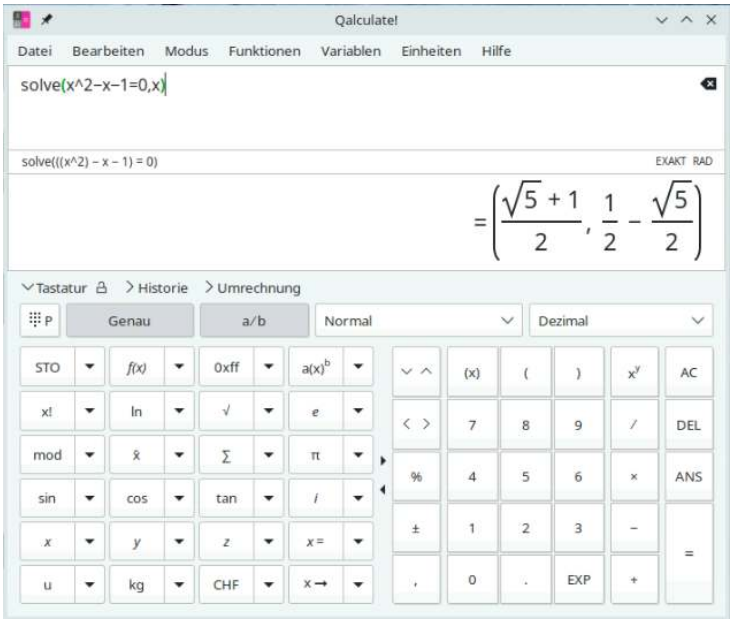

Abbildung 5.16: Screenshot von Qalculate!

VSCodium Die Open Source Version von Visual Studio Code. Diese IDE wird von vielen Lehrpersonnen in der Informatik verwendet [vsc].

Sicherlich gibt es noch weitere Programme, die weggelassen oder hinzugefügt werden könnten oder sollten. Die genaue Softwareauswahl sollte sich aber einerseits stark am Lernstick orientieren, andererseits können durch Erfahrungen in der Praxis weitere Programme wegfallen oder hinzukommen.

5.13.2 Weitere Anpassungen

KDE Plasma Start

Nach der Installation der Pakete aus der genannten Liste konnte die KDE-Plasma-Umgebung nicht mehr korrekt gestartet werden. Stattdessen starteten nun «gleichzeitig» mit KDE Plasma auch Bedienelemente, die zum GNOME Desktop gehören.

Abbildung 5.17: Vermischte Desktop Umgebung: KDE Plasma mit Leiste von GNOME

Es stellte sich heraus, dass während der Installation der Link `master:/etc/alternatives/x-session-manager` auf `master:/usr/bin/gnome-session` gesetzt und beim Login ausgeführt wurde. Nachdem dieser manuell auf `master:/usr/bin/startplasma-x11` gesetzt wurde, funktionierte die Anmeldung wieder problemlos.

Steuerung via Touchscreen und Update auf Plasma 5.25

Ein Ziel des Projektes besteht darin, dass Schülerinnen und Schüler mit Touchscreen-Geräten, wie einem Apple iPad, dieses System nutzen können. Ubuntu 22.04 LTS liefert die Version 5.24 von KDE-Plasma aus [dev22]. Eine Recherche hat ergeben, dass Plasma ab Version 5.25 einen speziellen Touchscreen-Modus anbietet [Web22]. Dieser soll die Benutzung mit Hilfe eines Touchscreens erleichtern, indem gewisse Gesten unterstützt werden und einige Elemente grösser dargestellt werden. Obwohl unklar ist, ob die Gesten über guacamole korrekt funktionieren werden, wurde entschieden, mittels des PPAs «Kubuntu Backports Extra» [Ltd22] Plasma auf Version 5.25 zu aktualisieren.

Danach wurde der Touchscreen-Modus standardmässig aktiviert.

```
[Input]
TabletMode=on
```

Listing 5: `manager:/etc/skel/.config/kwinrc`

Zusätzlich wurde die Höhe der Kontrollleiste auf 60 Pixel erhöht, um auch hier ein besseres Touchscreen-Erlebnis zu ermöglichen.

Die Bedienung des Touchscreens funktioniert trotz des Updates leider nicht ganz reibungslos, da die Gesten teilweise vom Browser oder vom Betriebssystem des Clientgerätes abgefangen werden. So ist zum Beispiel bei manchen Geräten zum Verschieben eines Objekts ein Doppeltipp auf das Objekt notwendig. Um den Zugang für die Schülerinnen und Schüler zu erleichtern, wurde ein kurzes Erklärvideo gedreht, in dem dies erklärt und gezeigt wird [oli23b].

Mit einem Touchscreen lässt sich das guacamole-Menu durch Wischen öffnen. Auch dies wurde in einem kurzen Video festgehalten [oli23a].

Wird das System ohne physische Tastatur genutzt, bietet Guacamole glücklicherweise Lösungen an. Diese werden den Schülerinnen und Schüler ebenfalls in Videoform gezeigt [oli23d].

Es sei darauf hingewiesen, dass die Autoren kein iPad besitzen und diese Einstellungen nur sporadisch mit kurzfristig ausgeliehenen Geräten getestet wurden.

Knöpfe zum Abmelden und Herunterfahren

Es ist nicht sinnvoll, dass die Schülerinnen und Schüler das System herunterfahren oder neu starten können. Die entsprechenden Schaltflächen sind standardmässig aber vorhanden und sogar ziemlich dominant platziert (siehe Abbildung 5.18).

Um Benutzern ausserhalb der sudo-Gruppe die Möglichkeit zu nehmen, das System herunterzufahren, wurde eine polkit-Regel (Listing 6) eingetragen.

```
polkit.addRule( function(action, subject) {
    const power_actions = [
        'org.freedesktop.login1.reboot',
        'org.freedesktop.login1.reboot-multiple-sessions',
        'org.freedesktop.login1.power-off',
        'org.freedesktop.login1.power-off-multiple-sessions',
    ];
    if ( power_actions.includes( action.id ) ) {
        let result = polkit.Result.AUTH_ADMIN;

        if ( subject.isInGroup( 'sudo' ) ) {
            result = polkit.Result.YES;
        }
        return result;
```

```
15      }
16  } );
```

Listing 6: `master:/etc/polkit-1/20-disable-unprivileged-power-control.rules` [Ikr19]

Das Problem entpuppte sich allerdings als grösser als zunächst angenommen. Nicht nur wurde die Schaltfläche zum Herunterfahren trotz polkit-Regel weiterhin angezeigt, sondern führte nun ein Klick auf diesen Button zur vollständigen Terminierung der Plasmasitzung. Ein schwarzer Bildschirm war dann das Resultat, aus dem Nutzerinnen und Nutzer nicht mehr selbst weiterkommen würden. An diesem Punkt angelangt würde nämlich ein manueller Eingriff des Administrators, wie ein Neustart des Containers, nötig werden.

Die Schülerinnen und Schüler sollten also auf keinen Fall den Shutdown-Knopf drücken. Eine solche Anweisung stellt keine akzeptable Lösung dar, da nicht garantiert werden kann, dass die Schülerinnen und Schüler sich daran halten — ob absichtlich oder nicht. Ausserdem wirkt eine derartige Bitte doch eher künstlich. Sie erinnert auch an seltsam wirkende Anweisungen, wie die Instruktion der IT der Kantons Basel-Landschaft keine Teams in Microsoft Teams umzubenennen, weil sonst diverse automatisierte Wartungsarbeiten nicht mehr richtig funktionieren.

Daher wurde versucht, den Shutdown-Button zu entfernen oder seine Funktionsweise zu ändern. Leider wurde in der Dokumentation zum Kiosk-Modus von KDE Plasma[Com23a] nur die Möglichkeit gefunden, das Ausloggen eines Benutzers komplett zu sperren (key `logout`). Da Logouts möglich bleiben sollen, stellt diese Lösung keine gute Option dar.

Auch nach längerer Suche konnte leider keine geeignete Einstellung gefunden werden. Es stellte sich heraus, dass ein Klick auf den Shutdown-Button das Binary `/usr/bin/plasma-shutdown` ausführt. Leider macht ein Klick auf den Logout-Button genau dasselbe, nur mit anderen Parametern. Daher hätte ein einfaches Entfernen dieses Binaries dazu geführt, dass auch hier die Funktionalität zum Abmelden verloren gegangen wäre.

Das erwähnte Binary `/usr/bin/plasma-shutdown` ist Teil des Pakets `plasma-workspace` [KDE23]. Um das Problem zu lösen, wurde der entsprechende Quellcode studiert.

```
23  void Shutdown::logout()
24  {
25      startLogout(KWorkSpace::ShutdownTypeNone);
26  }
27
28  void Shutdown::logoutAndShutdown()
29  {
30      startLogout(KWorkSpace::ShutdownTypeHalt);
31  }
32
33  void Shutdown::logoutAndReboot()
34  {
35      startLogout(KWorkSpace::ShutdownTypeReboot);
36  }
```

Listing 7: Ausschnitt aus `startkde/plasma-shutdown/shutdown.cpp` [KDE23]

Die Funktion `startLogout` wird offenbar mit unterschiedlichen Argumenten aufgerufen. Damit also die Buttons zum Rebooten und Herunterfahren künftig die gleiche Funktion auslösen wie der Logout-Button, wurden die Zeilen 30 und 35 entsprechend angepasst. Neu soll also jedes Mal die Funktion `KWorkSpace::ShutdownTypeNone` übergeben werden.

```
23 void Shutdown::logout()
24 {
25     startLogout(KWorkSpace::ShutdownTypeNone);
26 }
27
28 void Shutdown::logoutAndShutdown()
29 {
30     startLogout(KWorkSpace::ShutdownTypeNone);
31 }
32
33 void Shutdown::logoutAndReboot()
34 {
35     startLogout(KWorkSpace::ShutdownTypeNone);
36 }
```

Listing 8: Modifizierte Version von `startkde/plasma-shutdown/shutdown.cpp` [KDE23]

Nachdem diese modifizierte Version von plasma-workspace neu kompiliert und das Binary `/usr/bin/plasma-shutdown` entsprechend aktualisiert wurde, war das Problem behoben und ein Klick auf «Herunterfahren» führt nun zum einfachen Abmelden.

Diese Lösung des Problems zeigt eindrücklich, wie nützlich und wichtig es ist, dass der Quellcode verfügbar ist und modifiziert werden darf.

Als zusätzliche kosmetische Massnahme wurde der Logout-Button etwas dominanter dargestellt und die wie gewünscht nicht mehr einen Shutdown auslösende Option «Herunterfahren» weiter in den Hintergrund gerückt als in der Standardeinstellung.

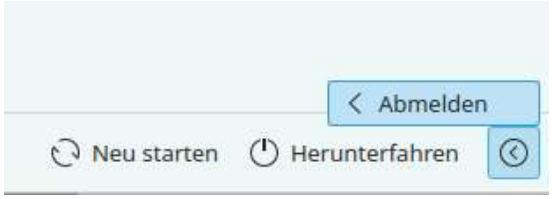

Abbildung 5.18: Logout-Button in der Standardeinstellung

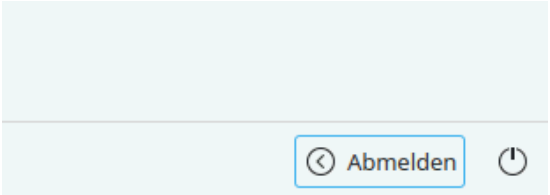

Abbildung 5.19: Logout-Button mit der neuen Einstellung

Nicht sichtbare Dateiauswahl

Bei einem Test wurde festgestellt, dass der Dialog zur Auswahl einer Datei in VSCodium teilweise hinter dem Hauptfenster geöffnet wurde, so dass man ihn gar nicht mehr sehen konnte. Deswegen wurde die Regel aus Listing 9 eingeführt. Diese bewirkt, dass ≪GTK FileChooserDialog≫ [Teaa] immer im Vordergrund geöffnet wird. Das ist deshalb wichtig, da dieses Fenster sonst leicht übersehen werden kann und bei Touchscreen-Geräten das Verschieben von Fenstern nicht optimal funktioniert.

```
[$Version]
update_info=kwinrules.upd:replace-placement-string-to-enum,kwinrules.upd:use-virtual-desktop-ids

[1]
Description=gtkfilechooserdialog-on-top
above=true
aboverule=3
types=32
windowrole=gtkfilechooserdialog
windowrolematch=1

[General]
count=1
rules=1

[file-chooser-dialog]
Description=gtkfilechooserdialog-on-top
above=true
aboverule=3
types=32
windowrole=gtkfilechooserdialog
windowrolematch=1
```

Listing 9: master:/etc/skel/.config/kwinrulesrc

Hintergrundbild

Als kosmetische Massnahme wurde noch ein Hintergrundbild mit Hilfe des Lernstick-Logos (Abbildung 3.4), einem mit DALL-E [Ope21] generiertem Logo (Abbildung 1) und [Mol22] erstellt und als Standard gesetzt.

Abbildung 5.20: Hintergrundbild basierend auf den [Mol22] und den Abbildungen 1 und 3.4

Grafischer Passwortwechsel

Um das Ändern des Passworts für die Benutzerinnen und Benutzer zugänglicher zu machen, wurde mit Hilfe von kdialog [Com23b] ein Skript `change-password.sh` (Listing 14 auf Seite 127) geschrieben, das den Passwortwechselprozess grafisch durchführt.

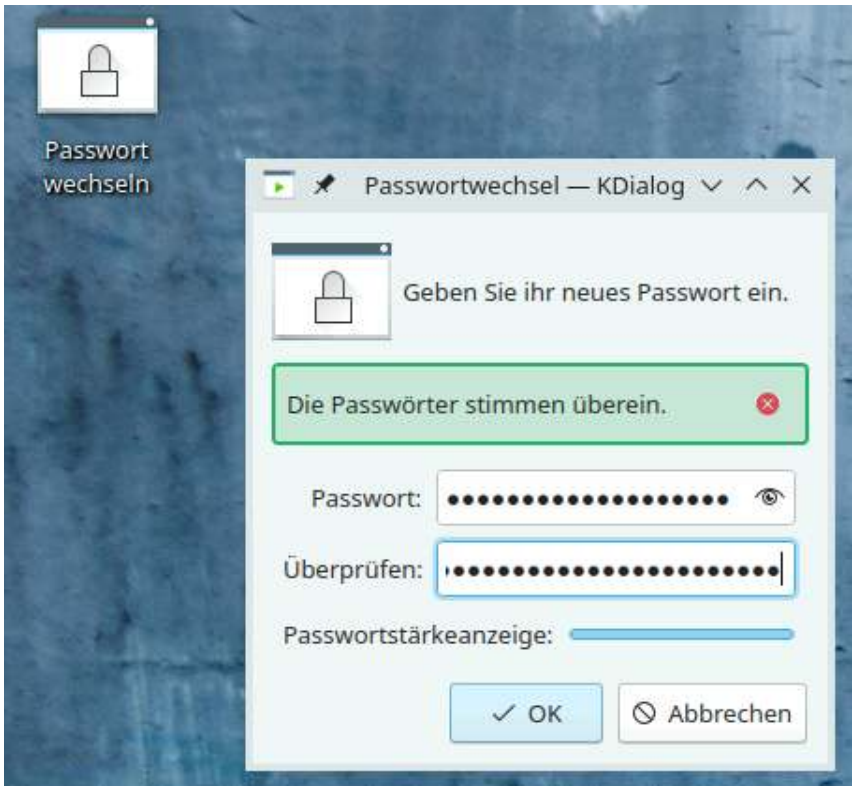

Abbildung 5.21: Graphischer Passwortwechsel

Zusätzlich wird dieses Skript mit einem Icon auf dem Desktop zur Verfügung gestellt (siehe links oben in Abbildung 5.21).

Durch einen passenden Autostart-Eintrag wird dieses Skipt beim ersten Login ausserdem zusammen mit einer eingeblendeten Willkommensbotschaft automatisch aufgerufen und erzwingt einen Passwortwechsel. Erst nachdem dieser Wechsel vollzogen wurde, wird dieser Autostart-Eintrag entfernt.

5.14 Datenaustausch mittels OneDrive

Damit die Schülerinnen und Schüler eine einfache Möglichkeit haben, Daten mit dem System auszutauschen, wurde das Skript `setup-onedrive-mount.sh` (Listing 17) erstellt. Dieses generiert einen rclone remote names «onedrive». Interessanterweise klappte dieses Vorgehen im Gegensatz zu dem zuvor versuchten Ansatz über das Tool onedriver ([jst]).

Der Systemd-Dienst `ondrive-mount.service` wird nach erfolgreicher Konfiguration aktiviert und sorgt dafür, dass das entsprechende OneDrive-Laufwerk beim Login automatisch gemountet wird.

Um die Einrichtung für die Schülerinnen und Schüler noch etwas zugänglicher zu gestalten, wurde zusätzlich das Anleitungsvideo [oli23c] erstellt.

Das Ganze wurde mit einem Schulaccount des Kanton Baselland getestet. Es ist zu erwarten, dass hier trotzdem Support geleistet werden muss. Wie zuverlässig das für alle Schülerinnen und Schüler

funktionieren wird, muss noch erprobt werden.

5.15 Die fertigen Container im Überblick

Da inzwischen einige Container zusammen gekommen sind, folgt hier eine grobe Übersicht, zusammen mit den wichtigsten Eigenschaften und Funktionen der einzelnen Container.

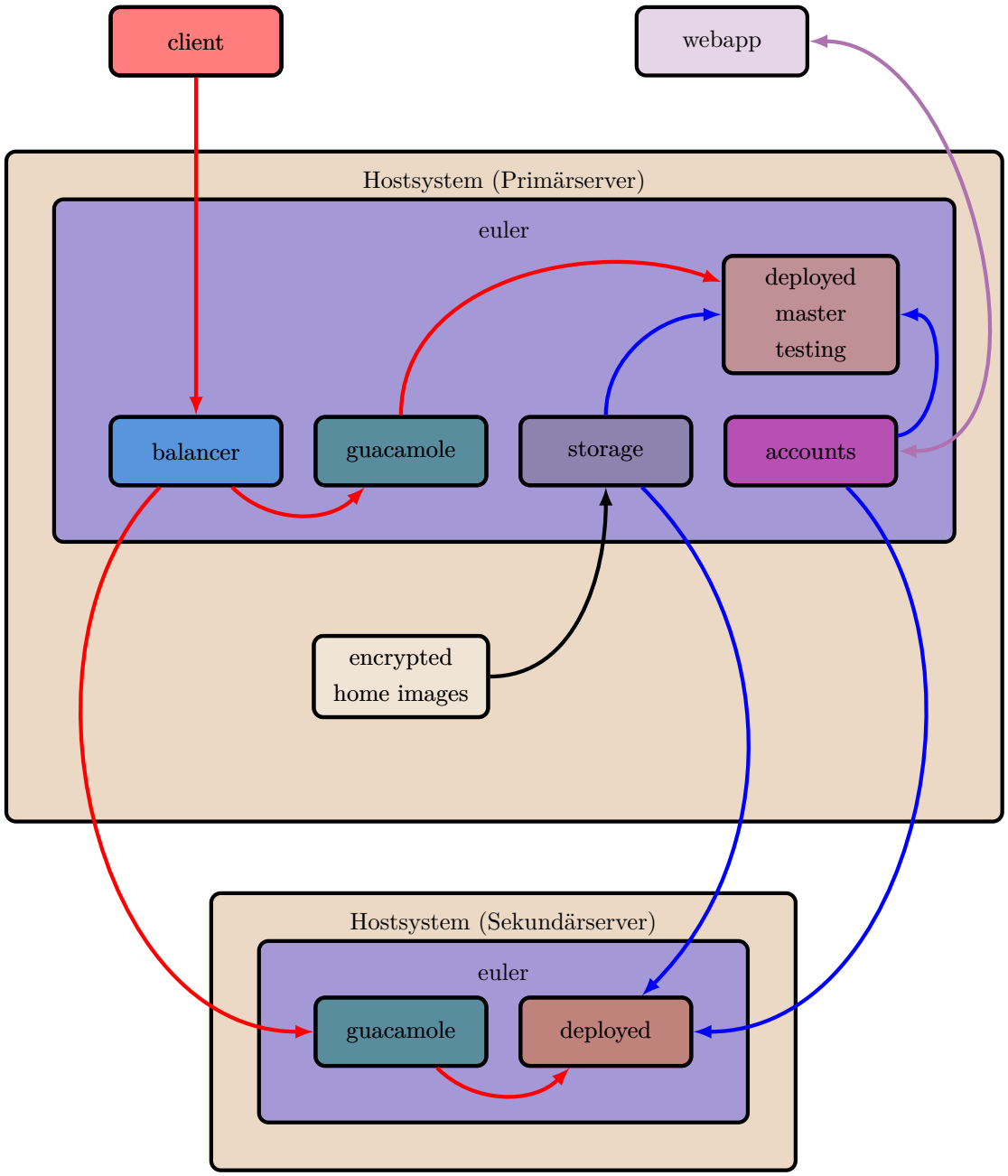

Abbildung 5.22: Vereinfachte schematische Darstellung

5.15.1 euler

Der Hauptcontainer *euler* delegiert die erforderlichen Aufgaben an die jeweiligen Untercontainer. Beispielsweise veranlasst er *storage*, neue User-Home-Images zu verschlüsseln und bereitzustellen oder im Container *accounts* neue Accounts anzulegen, wenn eine entsprechende Anfrage eingeht.

Der Grund für diesen Container ist der Versuch, das System möglichst einfach installierbar zu halten. So muss jemand, der dieses Cloudsystem einsetzen will, mehr oder weniger nur diesen einen Hauptcontainer auf dem jeweiligen Server importieren. Darin sollte wiederum möglichst alles enthalten sein. Leider ist das nicht vollständig gelungen, aber die Idee wurde trotzdem bis zum Ende des Projektes so weit wie möglich weiterverfolgt.

5.15.2 webapp

Der *webapp*-Server ist kein Container, sondern ein schulinterner Server, auf welchem ein Schul-Login durchgeführt und überprüft werden kann. Sollte dieser Login funktionieren, wird an *euler* ein Account-Erzeugungs-Request gesendet.

5.15.3 accounts

Der Container *accounts* verwaltet die Nutzeraccounts. Alle Benutzer bekommen auf diesem Container einen passenden Eintrag in `/etc/passwd`, `/etc/shadow` und `/etc/group`. Diese Dateien werden per SSHFs den Desktopcontainern zur Verfügung gestellt.

Weiter befindet sich in diesem Container auch die Datei `user-mapping.xml`, die ebenfalls per SSHFS mit dem Guacamole-Container geteilt wird. Darin befinden sich die Accountinformationen für Guacamole.

Dieser Hilfscontainer dient dazu, dass die Useraccounts über alle Desktop- und Guacamolecontainer über alle Host-Server hinweg synchronisiert bleiben. Der Hauptcontainer *euler* veranlasst es jeweils, dass in *accounts* bei Bedarf passende neue Accounts erstellt oder gelöscht werden.

5.15.4 storage

In *storage* werden die Userhomes verwaltet. Der Host teilt die zur Verfügung gestellten Image-Dateien mit *euler*, der diese wiederum mit *storage* teilt. Dort werden sie per `gocryptfs` verschlüsselt und in der entschlüsselten Form wiederum mit den Desktopcontainern per SSHFS geteilt.

Es mag hier das Missverständnis geben, dass die einzelnen User allein den passenden Schlüssel besitzen und diese Schlüsseldateien sich somit nicht in einem für die Nutzerinnen und Nutzer gar nicht zugänglichen Container wiederfinden sollten. Dieser Mehraufwand (auch von der Rechenleistung her) lohnt sich aber nicht: Privilegierte Nutzerinnen und Nutzer könnten auch dann noch die Homeverzeichnisse der eingeloggten Userinnen und User anschauen, wenn die jeweiligen Schlüsseldateien nur von den Besitzern selbst gelesen werden könnten. Die Verschlüsselung soll hingegen hauptsächlich die Home-Images schützen, die eines Tages vermutlich gar nicht mehr lokal auf dem Host gespeichert sind, sondern auf einen externen Cloud-Dienst ausgelagert werden.

Der Container *storage* teilt die entschlüsselten Homeimages sowohl mit den lokalen Desktopcontainern als auch mit allfälligen externen Sekundärservern (Load Balancing).

5.15.5 balancer

Der Client verbindet sich immer mit dem Container *balancer* auf dem Primärserver. Auf *balancer* ist eine Instanz von `haproxy` installiert, welche die Verbindung wahlweise an den lokalen *guacamole*-Container oder an einen entsprechenden Container auf einem Sekundärserver weiterleitet. Auf diese Weise werden die Nutzerinnen und Nutzer transparent über mehrere Server verteilt, sofern denn auch mehrere solcher Server vorhanden sind.

5.15.6 guacamole

Auf dem Container *guacamole* läuft eine Instanz von `guacamole`, welche die Nutzerinnen und Nutzer schlussendlich mit einem Desktopcontainer, also für gewöhnlich mit *deployed* über das RDP-Protokoll verbinden. Die notwendigen Informationen über die Nutzeraccounts holt sich *guacamole* per SSHFS vom Container *accounts*.

5.15.7 Die Desktopcontainer

Es gibt insgesamt 3 Desktopcontainer. Alle holen sich ihre Home-Verzeichnisse per SSHFS vom Container *storage* und ihre Nutzeraccounts vom Container über eine Kombination von SSHFS und `libnss_extrau␣sers` von *accounts*:

deployed

Das ist der Container, der aktiv genutzt wird.

master

Dieser Container wird nicht genutzt. Er dient als aktuelle Schablone für *deployed*, welcher jede Nacht einmal frisch von *master* geklont wird.

testing

Dieser Container wird bei Bedarf ebenfalls als Klon von *master* generiert. Anschliessend werden allfällige Einstellungsänderungen, Softwareinstallationen und -Updates zuerst auf Testing ausprobiert und, wie der Name impliziert, getestet. Sollte alles wie gewünscht klappen, wird anschliessend *master* als Klon von *testing* neu generiert. Ansonsten wird *testing* einfach wieder verworfen.

Spezialfall: Administratoraccount

Es gibt auf den Desktop-Containern sowie auf *guacamole* einen speziellen Nutzer mit dem Namen «administrator», welcher sudo-Rechte auf allen Desktopcontainern besitzt. Dieser besitzt kein über *storage* geteiltes Homeverzeichnis, sondern agiert nur rein lokal. Dieser Account hat von *guacamole* aus jedoch die Option, sich in alle drei Desktopcontainer einzuloggen. Ausserdem sind für diesen Account

spezielle Tools, wie zum Löschen von Usern oder zur Anzeige der Anzahl verbleibenden freien Home-Images, vorgesehen.

Ein Login mit diesem Account resultiert in einem anderen Standardhintergrundbild, um gleich deutlich erkennbar zu machen, dass man gerade als Administrator eingeloggt ist.

Abbildung 5.23: Hintergrundbild für den Administrator

Der Administrator-Account kann unter anderem benutzt werden, um normalen Nutzerinnen und Nutzern zu helfen oder Software auf *testing* zu installieren und auszuprobieren.

5.15.8 Primärserver - Sekundärserver

Während auf dem Primärserver alles Notwendige dabei ist, wird auf dem Sekundärserver im Bootstrap-Prozess der Container *euler* auf *guacamole* und *deployed* reduziert. Alles andere, wie *storage* und *accounts* holen sich die Container auf dem Sekundärserver per SSHFS von den jeweiligen Containern im Primärserver.

Es kann grundsätzlich beliebig viele Sekundärserver geben. Die entsprechende Liste in *balancer* auf dem Primärserver muss aber entsprechend aktuell gehalten werden.

Wie erwähnt wurde diese Funktionalität jedoch nur soweit möglich vorbereitet und konnte im Rahmen dieser Projektarbeit nicht getestet werden. Auch der Bootstrap-Prozess selbst muss vorerst ungetestet bleiben, da auch dazu ein zweiter Server nötig wäre, wenn man den vorhandenen Server nicht zuerst löschen möchte — ein Risiko, das die Autoren vorerst nicht eingehen möchten.

Kapitel 6

Von Grund auf neue, verbesserte und aufgeräumte Umsetzung

In dem ganzen Projekt sind Stabilität, Effizienz und Optimierung von entscheidender Bedeutung. In diesem Kapitel wird eine überarbeitete Version des Projekts vorgestellt.

Diese Neuimplementierung profitiert von wertvollen Erkenntnissen, die während der Entwicklungsphase des Prototyps gewonnen wurden. Im Verlauf der Zeit wuchs die Prototyp-Version und verlor an Übersichtlichkeit. Dieser Entwicklungsprozess lieferte jedoch viele Erfahrungen, die nun in der Umsetzung einer überarbeiteten Version genutzt werden konnten. Diese neue Projektversion sollte so nicht nur übersichtlicher, sondern auch erheblich effizienter werden. In diesem Kapitel werden die Schlüsselkomponenten und Vorteile dieser neuen Implementierung im Detail erläutert und aufgezeigt, wie diese den Weg für eine noch erfolgreichere Umsetzung des Projekts ebnet.

Der Beginn der neuen Umsetzung fiel mit technischen Problemen des Host-Servers zusammen. Ein aufgrund vieler Snapshots fast vollgelaufenes btrfs-Dateisystem verursachte Probleme, die nicht sofort durch Rebalancing gelöst werden konnten. Da bereits eine Komplettüberarbeitung des ganzen Projektes in Planung war, wurde nicht weiter versucht, diese Probleme zu lösen und der Server wurde entsprechend neu aufgesetzt.

Da hierbei die ganzen Erfahrungen aus der Protoyp-Version eingeflossen sind, soll hier nicht noch einmal auf sämtliche Details eingegangen werden. Stattdessen sollen in diesem Kapitel die Änderungen beschrieben werden, die sich beim Wechsel von der Prototyp-Version zur Release-Candidate-Version ergeben haben.

6.1 Reduktion der Containerzahl

Da die Container *accounts* und *storage* beide eine vergleichbare Aufgabe, nämlich etwas für alle Desktop-Container gemeinsam bereit zu stellen, teilen, wurden die zwei Container zu einem Container *manager* vereint, der beides tut: Accountverwaltung und die Bereitstellung der verschlüsselten Homeverzeichnisse. Dieser neue Container *manager* enthält auch die Datei `user-mapping.xml`, die per SSHFS an den

Container *guacamole* weitergereicht wird. Dadurch hat *manager* nun die komplette Kontrolle über das Usermanagement und alle entsprechenden Aktionen finden jetzt dort statt und nicht mehr verteilt auf *guacamole* und *manager* (vormals *accounts*). Insbesondere muss also auch *euler* nun keine Anfragen wie Passwortwechselrequests mehr bearbeiten.

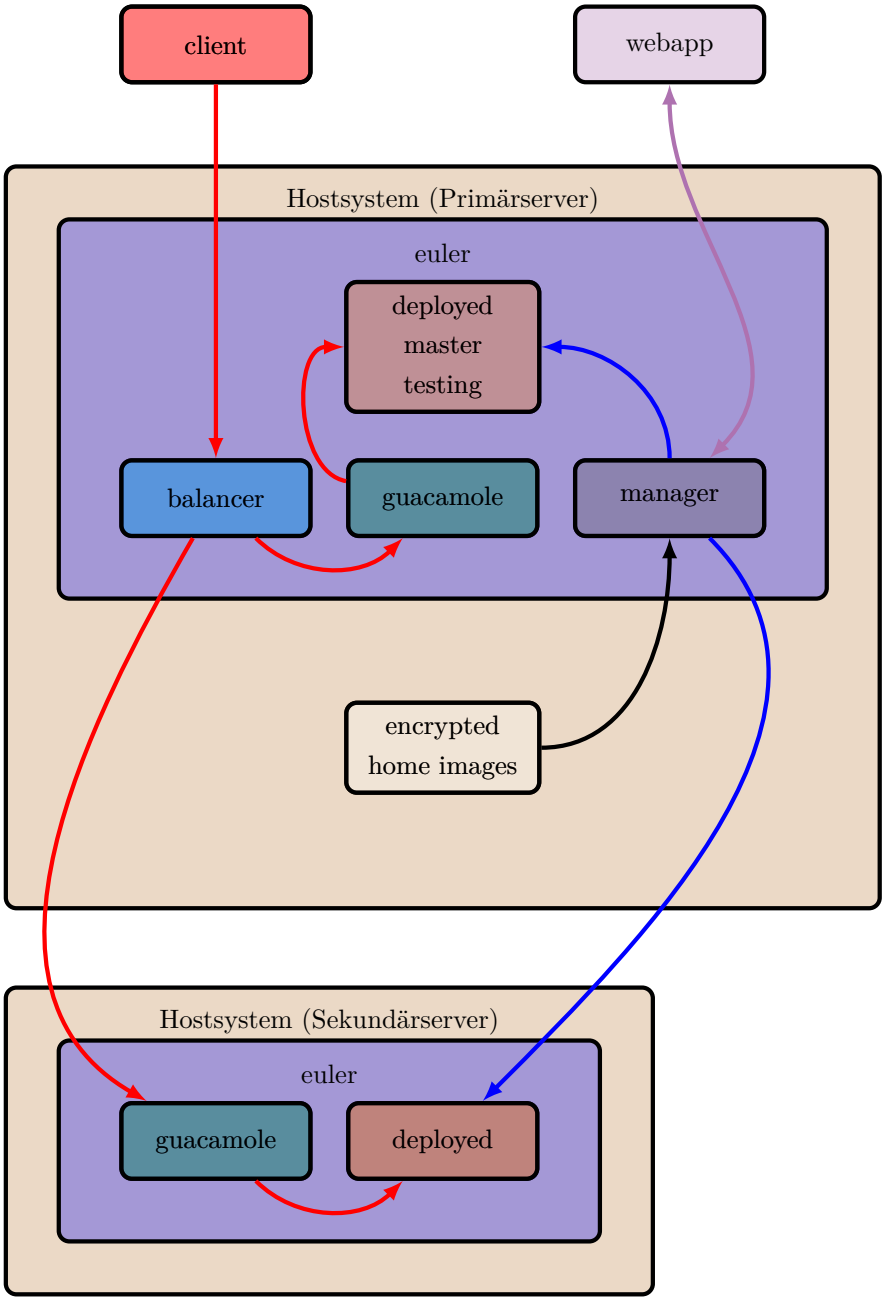

Abbildung 6.1: Vereinfachte schematische Darstellung

6.2 Komplettüberarbeitung des Passwortmanagements

In der Prototyp-Version wurde das Programm `/usr/bin/passwd` durch ein Shell-Skript überschrieben, das, neben dem Aktualisieren des entsprechenden Unix-Passwortes, auf komplizierte Weise Passwortänderungsrequests via *euler* an *guacamole* weiterleiten musste. Das hat zwar funktioniert, aber der Ansatz schien doch zu viele mögliche Fehlerquellen mitzubringen. So war zum Beispiel nicht klar, was passiert, wenn ein Nutzer eine Alternative zu `/usr/bin/passwd` wie `/usr/bin/chpasswd` zu benutzen versucht. Zu versuchen, alle Alternativen zu blockieren, hätte mit hoher Wahrscheinlichkeit damit geendet, dass etwas übersehen geblieben wäre. Auch war unklar, was passiert, wenn ein Benutzer zu schnell hintereinander versuchen würde, sein Passwort zu ändern oder in der Zwischenzeit von *webapp* ein neuer Request für diesen Nutzer ankommen würde. Da es aus offensichtlichen Sicherheitsüberlegungen nie möglich sein darf, dass ein Desktop-Container direkt Befehle auf *euler* ausführen kann, muss nämlich immer bis zu einer Minute gewartet werden, bis ein Cronjob auf *euler* einen entsprechenden Request wahrnimmt und danach handelt. Mögliche Folgen bei zeitlich ungünstig platzierten Request wären, dass ein Nutzer sich zum Beispiel nicht sofort wieder einloggen kann oder gar die Passwörter zwischen dem Desktop und *guacamole* nicht mehr synchronisiert sind.

Aus diesen Gründen wurde beschlossen, für die überarbeitete Version des Projekts das Passwortmanagement komplett neu zu gestalten. Neu wird die Aktualisierung eines Passwortes in der `user-mapping.xml` von *guacamole* direkt über Linux PAM (Linux Pluggable Authentication Moduls) erledigt [lin23a].

Erste Versuche mit `pam_script.so` und `pam_exec.so` schlugen deswegen fehl, weil PAM offenbar nur im ≪auth≫-Modus das eingegebene Passwort weitergeben kann, aber nicht im beim Passwortwechsel benutzten ≪password≫-Modus.

Daher wurde schlussendlich ein eigenes PAM-Modul `pam_guac.so` (Listing 11 auf Seite 123) programmiert. Dieses Modul übernimmt das bei einem Passwortwechsel eingegebene neue Passwort, berechnet davon den SHA256-Hash und fügt diesen an die passende Stelle von `user-mapping.xml` ein. Dadurch erfolgt ein Passwortwechsel jetzt fast in Echtzeit und direkt über die normalen Passwort-Tools.

Ein Nachteil an dieser Lösung ist, dass die Desktop-Container jetzt Lese- und Schreibzugang auf die `user-mapping.xml` haben müssen. Das ist etwas, was die Autoren gerne vermieden hätten. Doch der gefühlte Gewinn bei dieser Lösung war zu gross. Ausserdem wird das entsprechende Verzeichnis von *manager* her per SSHFS nur so gemountet, dass lediglich root Lese- und Schreibrechte hat. Wie in bisherigen Sicherheitsüberlegungen auch schon gilt also ebenfalls hier der Kompromiss: Wenn jemand root-Rechte auf dem Desktop erlangt hat, ist ohnehin bereits (fast) alles verloren.

Ein angenehmer Nebeneffekt dieser Lösung sollte jedoch nicht unerwähnt bleiben: Die C-Kenntnisse der Autoren basierten auf wenigen, über die letzten zwei Jahrzehnte verteilten Projekten und waren entsprechend eingerostet. Dieses Wissen wurde nun auf unterhaltsame Weise aufgefrischt. Insbesondere wurde etwas über libxml2 [Pro22] gelernt, um die Verarbeitung der XML-Datei sinnvoll zu gestalten. Diese wiedererlangten Kompetenzen wurden auch genutzt, um das Programm `delete-guacamole-user` (Listing 42 auf Seite 149), welches Benutzer aus der Datei `user-mapping.xml` löscht, in C zu schreiben.

Ähnlich verhielt es sich mit dem Wissen über PAM selbst: Man musste einiges über das PAM-System lernen, um diese Lösung zu implementieren. Die Autoren haben also sehr profitiert.

6.3 Verbesserung der Accountverwaltung

Da *manager* inzwischen auch die Guacamole-Benutzerliste kontrolliert, können die Aufgaben des Shell-Skripts `newusers.sh`[1] der Prototyp-Version direkt auf *manager* statt auf *euler* ausgeführt werden.

Das Skript wurde komplett überarbeitet und im Sinne der Unix-Philosophie

> ≪*do one thing and do it well*≫ [MP78]

in mehrere Skripts aufgeteilt. Ausserdem wurde der Name `newuser.sh` verworfen, da es sonst leicht zu Verwechslungen mit dem bereits vorhandenen Befehl `newusers` kommen kann.

Für die Accountverwaltung gibt es auf *manager* neu die folgenden Skripte:

`create-user.sh` Dieses Skript erzeugt manuell ein neues Konto durch die Angabe von Nutzername und Passwort. Falls ein Nutzername bereits existiert wird das Passwort entsprechend neu gesetzt. Der gewünschte Benutzername wird als Parameter übergeben und das Passwort wird abgefragt oder via Pipe übernommen.

`fetch-user-creation-requests.sh` Das Skript lädt Anfragen zur Erstellung von Konten oder zur Passwortzurücksetzung von *webapp* herunter. Es wird über einen Systemd-Timer regelmässig ausgeführt.

`process-creation-requests.sh` Dieses Skript arbeitet die vorhandenen Anfragen ab und schickt Sie an `create-user.sh`. Es wird mit Hilfe des Systemd-Dienstes `creation-requests-watcher.service` und `inotifywait` [ino23] automatisch aufgerufen, wenn neue Anfragen vorhanden sind. Die Anfragen befinden sich im Verzeichnis `/etc/create-user-requests`

Es akzeptiert nur via RSA verschlüsselte Anfragen.

`orphan-remover.sh` Dieses Skript löscht alle Einträge der Guacamole-Benutzerliste, die nicht mehr in `/etc/passwd` auftauchen. Solche ≪Waisenkinder≫ können entstehen, wenn ein Nutzer nur auf Systemebene gelöscht wird. Zusätzlich überwacht `orphan-remover.service` (Listing 46 auf Seite 151) über `entr` auf *manager*, ob sich in der Datei `/etc/passwd` etwas geändert hat. In diesem Fall wird `orphan-remover.sh` ausgeführt. So können auf `manager` Konten einfach mittels `userdel` gelöscht werden.

`process-deletion-requests.sh` Analog zu `process-creation-requests.sh` arbeitet dieses Skript, ebenfalls via Systemd und `inotifywait`, Löschanfragen ab. Die Löschanfragen befinden sich im Verzeichnis `/etc/delete-user-requests`.

Der Plan sah eigentlich vor, dass mit Hilfe von ≪extrausers≫ und einer weiteren Implementierung über PAM (über die C-Funktion `pam_sm_delete_user` oder über `pam_exec.so`) Benutzer ganz normal aus jedem Desktop-Container gelöscht werden können. Dadurch wäre dieses Skript nicht nötig gewesen. Leider hat sich herausgestellt, dass das Löschen eines Systembenutzers zunächst ein Lock-File für die entsprechende Passwort-Datei erzeugt. Solche Lock-Files funktionieren aber nicht über SSHFS oder auch über rclone mounts. Es bleibt wohl nur die Möglichkeit, Accounts wieder über Löschanfragen zu entfernen. Deswegen wurde dieses Skript erstellt.

[1] Dieses Skript ist im Ordner `deprecated` im Repo [WW23b] archiviert.

Zusätzlich wurden noch weitere Skripte für die Desktop-Container geschrieben:

`request-user-creation.sh` Dieses Skript erlaubt es dem Benutzer administrator (via sudo) von den Desktop-Containern aus gleiche Anfragen wie von *webapp* aus zu stellen. So ist es für den Administrator möglich, neue Konten zu erstellen oder Passwörter neu zu setzen. Es erstellt also per RSA verschlüsselte Anfragen.

`request-user-deletion.sh` Dieses Skript erstellt Löschanfragen, so dass Konten von den Desktop-Containern aus von administrator gelöscht werden können.

`account-management.sh` Um die Verwaltung der Konten durch den Nutzer `administrator` noch etwas benutzerfreundlicher zu gestalten wurde zusätzlich noch dieses kleines graphisches Administrationstool mit Hilfe von kdialog [Com23b] geschrieben.

Als weitere Verbesserung wird nun in `create-user.sh` beim Anlegen eines Accounts, der eine alte User-ID rezykliert, nicht nur das alte Home-Verzeichnis gelöscht, sondern auch der alte Gocryptfs-Key verworfen und ein Neuer erzeugt. Auf diese Weise sollen eventuell noch vorhandene Daten unbrauchbar gemacht werden.

Die Webseite zur Generierung von Konten wurde bei dieser Gelegenheit auch grundlegend überarbeitet. Zum einen sieht die Website zur Erstellung von Konten dank dem Einsatz von CSS jetzt optisch deutlich ansprechender aus, zum anderen werden die Statusmeldungen nun auch auf derselben Seite angezeigt. Etwas Javascript-Code sorgt jetzt für die automatische Aktualisierung dieser Statusmeldungen, welche neu auch einen Zeitstempel versehen werden und etwas aussagekräftiger geworden sind.

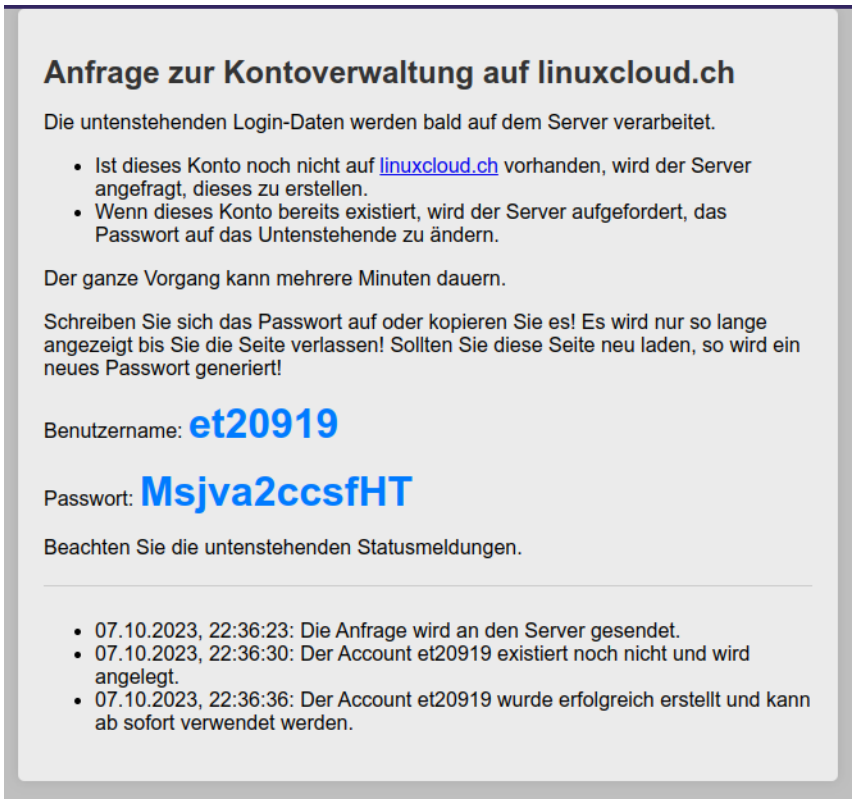

Abbildung 6.2: Screenshot der Webseite zur Generierung von Konten [WW23a]

Auch die entsprechenden PHP Skripte wurden komplett überarbeitet und neu wird jetzt jeweils ein Passwort bestehend aus 12 zufälligen alphanumerischen Zeichen generiert.

6.4 Verzicht auf statische IP-Adressen

Da bei der Neuinstallation festgestellt wurde, dass LXC/LXD bei der Initialisierung nicht immer den gleichen IP-Bereich verwendet und auch nicht bekannt ist, welcher IP-Bereich bei fremden Deployments überhaupt möglich ist, wurde beschlossen, nicht mehr auf feste IPs zu setzen. Stattdessen gibt es jetzt auf *euler* ein Shell-Skript (`update-hosts.sh`), das die IP-Adressen aller Container ausliest und in die Datei `/etc/hosts` aller Container (inklusive *euler* selbst) einträgt beziehungsweise diese Einträge aktualisiert. Das Skript wird nach dem Start der inneren Container ausgeführt und zusätzlich regelmässig als Systemd-Timer aufgerufen.

6.5 Deployement der Systemanpassungen

Damit notwendige Änderungen am System (wie die Angleichung an den Lernstick, die veränderte KDE-Shutdown-Prozesse oder das Passwortmanagementsystem via PAM) bei allfälligen Systemupdates auf Containerebene einfacher erhalten bleiben, wurde ein öffentliches Git-Repository [WW23b] eröffnet mit allen dazu notwendigen Shell-Skripten, C-Programmen und Dateien. Das Repository wird in allen

Containern geklont.

Dazu ist das Skript `deploy.sh` gekommen, welches alle für den Container, aus dem es aufgerufen wurde, relevanten Dateien und Einstellungen aus dem Repository nimmt und an die korrekte Stelle im Container linkt. Die C-Programme müssen aber separat kompiliert werden.

Dieses System hat für die Entwicklungsphase ganz offensichtliche Vorteile: Man muss nicht mehr jedes einzelne Skript und jede einzelne Einstellung von Hand in alle relevanten Container kopieren und die entsprechende Absprache zwischen den Autoren wurde auch viel einfacher.

Gleichzeitig wird dieses Konzept aber auch Vorteile für das System im Einsatz haben: Vermutlich wird eines Tages die zugrundeliegende Ubuntu-Version oder auch die Version von KDE Plasma erneuert. Dann können auf einfache Weise sämtliche notwendigen Änderungen, die an einer Standardinstallation gemacht werden müssen, durchgeführt werden.

Es ist denkbar, dass dieser Prozess in Zukunft sogar vollständig automatisiert werden kann, indem ein eigenes Repository für das Paketsystem von Ubuntu erstellt wird. Dies ist jedoch zum aktuellen Zeitpunkt noch eine reine Zukunftsidee.

6.6 Dienst zum Zurücksetzen der Desktop-Einstellungen und X-Session Reset

Es kann vorkommen, dass ein Schüler die Einstellungen seines Desktops so verändert, dass er sie nicht mehr selbst wiederherstellen kann. Es ist zum Beispiel denkbar, dass jemand das Panel entfernt und nicht weiss, wie man es zurückholen kann.

Für diesen Fall wurde ein weiterer Dienst auf *webapp* erstellt. Wenn sich ein Nutzer auf der Seite `https://webapp-gymms.sbl.ch/linuxcloud/auth/reset.php` anmeldet, wird eine Anfrage zum Zurücksetzen der Einstellungen gestellt.

Diese Anfrage wird von *euler* abgeholt und das entsprechende `.config` Verzeichnis neu aufgesetzt. Im gleichen Zug werden auch alle Prozesse des Nutzers gestoppt, um so auch eine eingefrorene Sitzung zu beenden.

Kapitel 7

Mögliche Verbesserungen

Ein Projekt wie das Vorliegende ist nie wirklich fertig. Daher sollen an dieser Stelle bereits einige Gedanken für künftige mögliche Verbesserungen und Erweiterungen beschrieben werden.

7.1 Accountmanagment

Es wurde erheblicher Aufwand für die Entwicklung des Accountmanagements betrieben. Die Lösung mittels PAM-Extrausers funktioniert zwar, bedarf jedoch weiterer Verbesserungen. Erschwerend kommt hinzu, dass dies lediglich über die xml-Datei `user-mapping.xml` implementiert wurde. Die Passwörter müssen derzeit dieselben sein wie die System-Passwörter der Nutzerinnen und Nutzer und sind als ungesalzene (!) SHA256-Hashes gespeichert. Obwohl die Autoren sich bemüht haben, diese Hashes für normale Benutzer unzugänglich zu machen, widerspricht diese Situation den etablierten ≪Best Practices≫.

Eine zeitaufwändige Verbesserungsmöglichkeit wäre das Schreiben einer eigenen Authentifizierungs-Erweiterung für Guacamole, um die Passwörter besser zu schützen [Fou23e].

Eine offenbar sauberere Lösung wäre es, einen eigenen OpenLDAP-Server in einem Container wie *manager* aufzusetzen und dann sowohl die PAM- als auch die Guacamole-Authentifizierung darüber laufen zu lassen [Foub]. Dieser Ansatz würde den zeitlichen Rahmen dieser Arbeit sprengen, auch angesichts der sehr geringen LDAP-Erfahrung der Autoren. Erste Recherchen haben ergeben, dass es schwierig sein könnte, simple Passwortänderungen über Linux-Bordmittel wie `/usr/bin/passwd` zuzulassen. Ein solches System scheint demnach nicht so einfach realisierbar zu sein. Aufgrund des damit einhergehenden Lernprozesses beabsichtigen die Autoren aber trotzdem, diese Idee zu einem späteren Zeitpunkt weiterzuverfolgen.

Möglicherweise wäre es sinnvoll, die Guacamole-Accounts von den Desktop-Accounts zu trennen, sodass die Passwörter nicht mehr synchronisiert werden müssen. Da jedoch ein wesentlicher Teil der Benutzerfreundlichkeit dadurch geopfert würde — die meisten Nutzer und Nutzerinnen würden vermutlich nicht verstehen, warum sie sich zwei Passwörter merken und sich dann doppelt einloggen müssen — kam die Idee auf, auf Ebene von Guacamole keine Authentifizierung mehr durchzuführen und alle Nutzer und Nutzerinnen direkt zu `xrdp` weiterzuleiten, wo dann die eigentliche Authentifizierung stattfindet. Diese

Idee wurde für ein paar Tage verfolgt, jedoch aus folgenden Gründen für den Moment verworfen:

- Früher bot Guacamole einen Modus an, bei dem die Authentifizierung vollständig deaktiviert wurde. In den aktuellen Versionen ist dieser Modus nicht mehr verfügbar. Versuche, das Modul aus alten Versionen zu portieren, sind bislang erfolglos geblieben [Jum15].
- Die Umsetzung der Idee, einen Dummy-Login in Guacamole zu erstellen, stiess auf Schwierigkeiten. Das Guacamole-Login-Menü ermöglicht kein einfaches HTTP-Login, über das die Dummy-Logindaten per GET übergeben werden könnten. Dadurch scheint es zumindest auf einfache Weise nicht möglich, dass die Nutzerinnen und Nutzer über einen speziellen Link direkt auf das System zugreifen können.
- `xrdp` zeigt zuerst eine irreführende und für die meisten Nutzer vermutlich eher kryptisch wirkende Fehlermeldung, wenn sie von Guacamole aus mit ungültigen Zugangsdaten weitergeleitet werden. Erst nachdem diese Meldung weggeklickt wurde, erscheint das Login-Menü von `xrdp`. Es wäre sicherlich möglich, dieses Verhalten zu ändern, aber aufgrund anderer Probleme wurde diese Idee vorerst nicht weiterverfolgt.

7.2 Übersetzungen

Derzeit sind viele Teile des Systems ausschliesslich in deutscher Sprache verfasst. Auch die Willkommensnachricht, die beim ersten Login angezeigt wird und wo die Nutzungsbedingungen zu finden sind, ist nur auf Deutsch verfügbar. Es wäre wünschenswert, sämtliche Elemente in Zukunft auch in andere Sprachen zu übersetzen. Möglicherweise ergibt sich hierzu eine Zusammenarbeit mit einer Sprach-Fachschaft an einer der Schulen der Autoren.

7.3 Dokumentation

Es wurde entschieden, die Dokumentation primär auf die Endverbraucher auszurichten und sich dabei auf Anleitungsvideos zu konzentrieren. Allerdings ist es auch wichtig, technische Dokumentationen für Installation und Nutzung zu verfassen, spätestens wenn das System auch an weiteren Schulen eingesetzt werden soll. Die aktuellen zeitlichen Einschränkungen erlauben es leider nicht, das jetzt schon zu erledigen, die Sache befindet sich aber definitiv auf der ≪Must-Liste≫. Auch weitere Anleitungsvideos sind wünschenswert.

Die Autoren sind der Ansicht, dass eine solche Dokumentation nur dann gut geschrieben werden kann, wenn das System ausführlich getestet und implementiert wurde. Erst dann wird es klar, worauf beim Einsatz eines Sekundärservers geachtet werden muss. Leider lässt die aktuelle finanzielle Situation keine solchen Szenarien zu, weshalb die Autoren in der aktuellen Phase des Projektes auf Betatester angewiesen sind.

7.4 Quota

Die Autoren sind unzufrieden damit, dass Home-Verzeichnisse als Images auf dem Host erzeugt werden müssen. Das Ziel war es, dass keine Ausführung auf dem Host selbst notwendig ist, nachdem der

Hauptcontainer *euler* installiert ist. Jedoch hat sich diese Option als die einzige umsetzbare Lösung herausgestellt, da unprivilegierte Container weder Loopmounts, noch Quotas auf Dateisystemebene (da es keine `/etc/fstab` gibt) oder Devicemapping erlauben. Ein Wechsel auf privilegierte Container stellt keine vertretbare Option dar, da diese fast per Definition unsicher sind.

Hier bleibt nur die Hoffnung auf zukünftige Entwicklungen seitens LXD/LXC. Es ist möglich, dass in Zukunft geeignete Optionen zur Verfügung stehen. Die Fortschritte bei der Entwicklung von Linux-Containern scheinen rasch voranzuschreiten.

7.5 Penetrationstests

Die Autoren haben innerhalb ihres Erfahrungs- und Wissensstands um Sicherheit und Privatsphäre bemüht. Allerdings mussten sie an vielen Stellen Kompromisse eingehen. Es wird oft davon ausgegangen, dass niemand root-Rechte innerhalb eines Containers erlangen kann, da ab diesem Zeitpunkt nicht mehr viel verhindert werden kann.

Die Geschichte der Privilege Escalation Exploits belehrt uns jedoch eines Besseren. Die Wahrscheinlichkeit, dass es innerhalb des Desktops möglich ist, root-Zugriff zu erlangen, sollte nicht übermässig niedrig eingeschätzt werden.

Fehlende finanzielle Mittel, Zeit und vor allem die richtigen Kontakte haben es bisher leider nicht erlaubt, einen professionellen Penetrationstester damit zu beauftragen, das System einer genauen Prüfung zu unterziehen. Die Autoren planen jedoch, dies in Zukunft nachzuholen, da die Ergebnisse eines Pentesters wertvolle Hinweise zur Milderung von Schwachstellen bieten können. Möglicherweise gibt es Wege, um die Sicherheit und den Schutz persönlicher Nutzerdaten erheblich zu verbessern, ohne Kompromisse eingehen zu müssen, wie es beispielsweise die Aussage ≪Wir hoffen, dass niemand Root-Zugriff erhält, da sonst ohnehin nicht viel getan werden kann≫ ist.

Im Rahmen des bisherigen Kenntnisstandes der Autoren gibt es keine vollständigen Lösungsansätze für die Probleme, die in dieser Arbeit mit Kompromissen gelöst wurden. Es ist jedoch anzumerken, dass es selbstverständlich Personen gibt, die über wesentlich umfangreicheres Wissen und viel mehr praktische Erfahrung verfügen.

Es bleibt zu hoffen, dass das System zu einem späteren Zeitpunkt von einer solchen Person getestet werden kann.

Kapitel 8

Fazit

8.1 Erreichen der Ziele

Projekte dieser Art sind bekanntlich nie wirklich abgeschlossen. Trotzdem soll in diesem Abschnitt grob zusammengetragen werden, welche Ziele aus Kapitel 2 wie weit bis zum Abgabetermin erreicht wurden.

8.1.1 Systemvoraussetzungen

Es ist gelungen, ein System aufzusetzen, welches fast keine Bedingungen an das Client-Gerät voraussetzt. Ein moderner Browser ist eigentlich alles, was benötigt wird.

Leider scheinen Geräte wie iPads noch mehr Schwierigkeiten als ursprünglich erwartet zu machen und die Steuerung über Touchscreens ist nicht überall so intuitiv, wie es gewünscht wäre. Auf einigen Geräten, wie einem von den Autoren getesteten Microsoft Surface, funktioniert der Touchscreen jedoch wie von den meisten Nutzerinnen und Nutzern erwartet. Verschieben und Scrollen kann man mit einer simplen Wischbewegung und das Guacamolemenü scheint auch zuverlässig zu funktionieren. Einige befragte Testpersonen hatten auf anderen Geräten Schwierigkeiten mit der Copy-Paste-Funktionalität und das Verschieben von Fenstern war nur mit der Doppel-Tap-Technik möglich, wie auf Seite 80 beschrieben.

Diese Einschränkungen könnten auf unterschiedliche Weiterleitungen von Touchgesten bzw. der Zwischenablage durch die Client-Systeme zurückzuführen sein - einige scheinen die Gesten explizit an Guacamole zu senden, während andere die Gesten strikt lokal verarbeiten und letztlich nur simulierte Mausklicks weiterleiten.

Mit dieser Einschränkung wird man wohl leben müssen. Es wurde versucht, das Problem abzumildern, indem Videos produziert wurden, die die Steuerung demonstrieren [oli23b], [oli23a] und [oli23d].

Insgesamt wird jedoch empfohlen, ein Desktopsystem besser mit (evtl. externer) Tastatur und Maus als über den Touchscreen zu bedienen.

8.1.2 Persistence

Dieses Ziel wurde vollständig erreicht: Die persönlichen Homeverzeichnisse bleiben erhalten.

Allerdings muss angemerkt werden, dass es nicht möglich war, die Persistence ausführlich zu testen. Daher bleiben etwaige Bugs vorbehalten und die Empfehlung für eine Backup-Strategie bleibt wie folgt formuliert: Es ist zu beachten, dass neben den Home-Image-Dateien auch die Gocryptfs-Keyfiles in einem Backup-Konzept unbedingt gesichert werden müssen.

8.1.3 Easy Deployability

Die einfache Installierbarkeit ist ein angestrebtes Ziel, welches bisher bedauerlicherweise noch nicht erreicht werden konnte.

Zwar funktioniert fast alles innerhalb eines Hauptcontainers, wenn dieser importiert wurde und somit im Wesentlichen alles installiert ist. Dabei sollte der Hauptcontainer gross genug sein, um alle benötigten Komponenten enthalten zu können. Jedoch sind noch einige weitere Schritte zu erledigen:

1. Eine geeignete Portweiterleitung muss im Hauptcontainer eingerichtet werden, abhängig von der Umgebung kann das entweder der standardmässige HTTPS-Port 443 sein oder, falls bereits ein HTTP-Service auf dem Server aktiv ist, ein alternativer Port.

2. Damit die Container sie nutzen können, müssen SSL-Zertifikate an den entsprechenden Ort kopiert werden und zwar so, dass diese, inklusive dem privaten Schlüssel, vom Container-internen root-User gelesen werden können. Falls der Host keine Zertifikate besitzt, müssen sie generiert werden, beispielsweise über Letsencrypt [Gro].

3. Die Passwörter für den Guacamole-Administrator und den Desktop-Administrator müssen festgelegt werden.

4. Im Untercontainer *manager* sollte ein neues RSA-Schlüsselpaar erstellt werden. Der öffentliche Schlüssel muss in den Desktopcontainer *master* sowie auf den schulhausinternen Server kopiert werden, der Account-Erstellungsanfragen verarbeitet.

5. In den Containern *guacamole* und *master* sollten die jeweiligen root-User neue SSH-Schlüsselpaare erstellen und den öffentlichen Teil in den Container *manager* kopieren.

6. Der schulinterne Server, in dieser Arbeit *webapp* genannt, muss eingerichtet werden: Es sollten nur berechtigte Personen in der Lage sein, Accounts zu erstellen und der Server benötigt die korrekte Zieladresse des Primärservers, um Statusmeldungen zu den erstellten Accounts empfangen zu können.

7. Falls der Host als Sekundärserver fungiert, sollten nicht benötigte Untercontainer wie *Manager* und *Balancer* entfernt werden. Des Weiteren müssen die SSHFS-Mounts in *Master* und *Guacamole* so angepasst werden, dass sie auf den Primärserver zeigen und nicht mehr auf einen lokalen Container. Dazu müssen entsprechende Port-Weiterleitungen auf dem Primärserver eingerichtet werden.

8. Es ist empfehlenswert, in den Sekundärservern eine Firewall-Einstellung vorzunehmen, die lediglich die Weiterleitung von Daten vom Primärserver erlaubt. Es wird nicht empfohlen, dass die Nutzerinnen

und Nutzer sich frei für eine Verbindung mit dem Sekundärserver entscheiden können.

9. Das Shell-Skript zur Erzeugung der Home-Images muss auf dem Primärhost installiert sein. Zudem muss die erwartete Verzeichnisstruktur von dem Skript bereits erstellt worden sein. Falls notwendig, sollte diese auf einen externen Ort ausgelagert werden, um ausreichend Speicherkapazität zu gewährleisten.

10. Es ist wichtig, dass auf dem Primärhost im *balancer* die Liste der möglichen Sekundärserver aktuell gehalten wird, damit diese auch genutzt werden können.

11. Vor der Erstellung des ersten Kontos sollten Home-Images erstellt werden.

All diese zusätzlichen Anpassungen erschweren die ≪Easy Deployability≫. Einige Teile könnten zwar durch ein Bootstrap-Skript automatisiert werden, aber es gibt auch dabei einige Einschränkungen, insbesondere in Bezug auf die folgenden Punkte:

- Es ist schwierig vorherzusagen, welche konkrete IT-Infrastruktur an verschiedenen Schulen vorhanden ist. Daher müsste das Bootstrap-Skript in der Lage sein, eine Vielzahl von Faktoren zu erkennen und flexibel darauf zu reagieren. Es scheint fast wie ein eigenständiges Projekt, das möglicherweise von jemandem erweitert werden kann, der im Rahmen einer weiteren Projektarbeit diese Arbeit weiterführen möchte.

- Ein Skript dieser Art hätte im Rahmen dieser Projektarbeit weitgehend ungetestet bleiben müssen. Aufgrund begrenzter finanzieller und zeitlicher Ressourcen der Autoren ist es nicht möglich, zusätzliche Server anzumieten, um Installationsroutinen zu testen. Dies könnte das System häufig so stark beeinträchtigen, dass es von Grund auf neu aufgebaut werden müsste.

Letztendlich gibt es Ansätze für die sogenannte ≪Easy Deployability≫, welche in Zukunft dazu beitragen könnten, dieses Ziel zu erreichen. Leider sind diese Ansätze zum Zeitpunkt der Abgabe nicht ausreichend, um das Ziel als erreicht zu betrachten.

8.1.4 Privacy by Design

Dieses Ziel wurde zum Teil erreicht. Zwar können Personen mit Administrationszugriff prinzipiell alle personenbezogenen Daten lesen — eine Möglichkeit, dies zu verhindern, müsste aber zwangsläufig eine lokale Verschlüsselung der Daten auf den Geräten der Nutzerinnen und Nutzer beinhalten. Das Kernziel des Projekts bestand jedoch darin, die virtuelle Desktopumgebung so geräteunabhängig wie möglich zu gestalten. Hierbei besteht ein grundlegender Konflikt. Im Rahmen der Untersuchung wurde die Nutzbarkeit für alle Geräte höher bewertet als der Schutz vor unethisch handelnden root-Benutzern.

Vorausgesetzt, dass die Ordnerberechtigungen nicht umgangen werden, gibt es tatsächlich keine Möglichkeit für einen Nutzer, auf die Daten eines anderen Nutzers zuzugreifen. Auch die Möglichkeit, dass ein neuer User durch UID-Recycling Zugang zu den Daten eines Vorgängers erhält, wurde berücksichtigt. Und in der Praxis noch wichtiger: Falls die Userhomes, wie es absehbar ist, auf einen externen Server ausgelagert werden, um mehr Speicherplatz zur Verfügung zu haben, sehen die Administratoren dieses externen Servers ausschliesslich verschlüsselte Daten.

Alles in allem kann das Ziel als erfüllt betrachtet werden, wenn auch mit einigen Kompromissen.

8.1.5 Sicherheit

Es bleibt fraglich, inwieweit das Ziel erreicht wurde. Es wurde beträchtliche geistige Arbeit in diesen Aspekt investiert und alle potentiellen Sicherheitslücken, die den Autoren auffielen, so gut wie möglich geschlossen. Es wurden jedoch auch an einigen Stellen Kompromisse eingegangen, wie zum Beispiel bei der Speicherung von Passworthashes für *guacamole* in ungesalzener Form. Ein viel grösseres Problem aber ist, dass die Autoren hier lediglich über Laienwissen verfügen. Es wird vermutet, dass ein professioneller Penetrationstester noch zahlreiche Schwachstellen aufdecken würde. Die Beauftragung eines solchen Experten liegt jedoch ausserhalb des vertretbaren Rahmens dieser Projektarbeit.

Es wurde nach bestem Wissen und Gewissen gearbeitet — ob das ausreicht ist aktuell schwierig zu sagen.

Wir empfehlen allen Schulen, die das Projekt nutzen möchten, das System ausgiebig testen zu lassen. Es ist ratsam, eine entsprechende Testphase durchzuführen, bevor die Anwendung produktiv eingesetzt wird.

Ausserdem kann auch ein regelmässiges erneutes Testen als sinnvoll erachtet werden, da Kernelexploits und weitere Sicherheitslücken schliesslich nicht auf dem heutigen Stand bleiben.

8.1.6 Skalierbarkeit

Theoretisch gesehen wurde das Ziel der Skalierbarkeit erreicht. Jedoch ist es nicht möglich, die Implementierung der Skalierbarkeit zu testen. Aus Sicht der Autoren gibt es jedoch keine Anhaltspunkte dafür, dass der Loadbalancer nicht wie erwartet funktionieren sollte.

Trotz allem kann das Ziel ehrlicherweise nicht als erreicht bezeichnet werden, sondern nur als ≪vermutlich erreicht≫.

8.1.7 FOSS

Das Ziel, ausschliesslich freie Software zu nutzen, wurde nur teilweise erreicht. Die erste wesentliche Ausnahme besteht in der Nutzung des proprietären RDP-Protokolls, dessen Leistungsvorteile gegenüber allen von Guacamole unterstützten Alternativen wie VNC oder SSH mit X-Weiterleitung überzeugend waren. Immerhin wurde mit xrdp eine freie Implementierung genutzt.

Ob das Ziel ansonsten zu 100% erreicht wurde, kann mit hoher Wahrscheinlichkeit verneint werden. Es war nicht möglich, sämtliche automatisch installierten Pakete von Ubuntu, Kubuntu-Desktop und den Lernstick-Quellen auf ihre Lizenz hin zu prüfen. In manchen Fällen sollte es jedoch problemlos möglich sein, bereits entdeckte unfreie Software aus dem System zu entfernen, sofern das Umfeld ausreichend idealistisch geprägt ist, um dies zu wünschen.

Auf andere Aspekte bezogen gestaltet sich die Frage nach der Realisierbarkeit eines reinen FOSS-Systems leider komplizierter. Der verwendete Kernel des Hostsystems ist kein Libre-Kernel [Ame23]. Ob LXC/LXD unter einem solchen Kernel funktionieren würde und ob eine unfreie Software als Abhängigkeit für LXC/LXD vorhanden ist, wurde nicht getestet. Aus Sicht der Container selbst wiederum spielt der Kernel eigentlich keine Rolle, da sie den Kernel des Hosts nutzen und somit nicht dessen Auswahl beeinflussen. Um

die Zielsetzung zu erfüllen, unser System möglichst unabhängig von der konkreten Hostserverkonfiguration zu gestalten, wurde beschlossen, in einer allfälligen Installationsanleitung keine spezifischen Kernel oder Distributionen zu verlangen.

Neben zeitlichen und finanziellen Einschränkungen haben wir uns auch deshalb dazu entschieden, auf das Testen mit verschiedenen Hostdistributionen, -konfigurationen und -kerneln zu verzichten.

8.2 Persönliches Fazit

Wir haben bereits viele ähnliche Projekte durchgeführt, wenn auch nicht in diesem Umfang, seit wir uns Anfang der Neunzigerjahre mit viel Trial&Error, aber noch ohne Internetzugang, das Programmieren mit QBASIC selbst beigebracht haben. Die Linux-Welt war für uns seit unseren Anfängen in den frühen 2000er Jahren immer eine grosse Spielwiese. Wir begannen mit ZipSlack [Vol], bei dem wir uns zuerst deutlich überfordert fühlten, und haben seitdem verschiedene Distributionen wie SuSE, Debian, Gentoo, Arch und weitere ausprobiert. Durch das Einlesen in die Thematik, anfangs noch über gedruckte Handbücher von SuSE Linux, das Konsultieren von Webforen wie `cyberarmy.com`, das Lösen von CTF-Challenges, das Hacken von Videospielkonsolen und das Aufsetzen diverser Computersysteme haben wir mittlerweile eine grosse Verbundenheit mit der Hackerkultur erlangt. Die kommerzielle Windowswelt erscheint uns dagegen immer fremdartiger und umständlicher. Hingegen fühlen wir uns seit vielen Jahren in der Welt der freien Software heimisch.

Von dieser Perspektive heraus betrachtet, ist es nicht überraschend, dass uns die Realisierung dieses Projekts, trotz oder gerade wegen zahlreicher Herausforderungen, viel Freude bereitet hat. Wir haben wie immer viel dazugelernt und unser vorhandenes Wissen erweitert. Das Projekt ist, unabhängig von einer Bewertung, für uns schon jetzt ein voller Erfolg.

Dennoch möchten wir darauf hinweisen, dass wir uns nicht als studierte Informatiker betrachten. Nach der absolvierten Weiterbildung empfänden wir dies als Affront gegenüber all denen, die das Fach über viele Jahre ≪richtig studiert≫ haben. Mit dieser Aussage möchten wir keineswegs die Qualität der Weiterbildung in Frage stellen, denn wir konnten hier definitiv einige neue Erfahrungen sammeln. Jedoch möchten wir nicht vergessen, dass eine derartige Weiterbildung nicht mit jahrelangem intensiven und exklusiven Fokussieren auf die Materie, wie es zum Beispiel in unserem Mathematikstudium über viele Jahre der Fall war, vergleichbar ist. Wirkliche Kompetenz, insbesondere hinsichtlich der Zusammenhänge zwischen den verschiedenen Teilbereichen und dem Warum und Wie, erlangt man nur durch einen entsprechend hohen Zeitaufwand. Wir stellen einen deutlichen (Kompetenz-)Unterschied fest, wenn wir unseren Mathematik- und Informatikunterricht vergleichen.

Wir sind Mathematiker mit Spezialisierung auf Zahlentheorie und keine Informatiker.

Und wir sind Hacker, die viel Spass an solchen Projekten haben.

Kapitel 9

Arbeitsaufteilung

In diesem Kapitel wird versucht, die Arbeitsteilung der Autoren zu dokumentieren. Es muss jedoch darauf hingewiesen werden, dass diese Arbeitsteilung im Laufe des Arbeitsprozesses stark verwischt wurde. Offensichtlich haben beide Autoren intensiv zusammengearbeitet, viele Diskussionen geführt, gemeinsam nach Ideen und Fehlern gesucht und auch gegenseitig ihre Arbeit korrigiert und zum Beispiel Codes korrigiert und von einem ≪ersten Hack≫ zu sauberem Code verbessert. Die folgende Auflistung kann daher nur ein grobes Bild wiedergeben, wobei jeweils die Person genannt wird, die mit der jeweiligen Aufgabe begonnen und den grössten Beitrag geleistet hat.

Für den Text dieser Arbeit selbst gilt, dass derjenige, der etwas getan hat, dies jeweils unmittelbar danach in der Arbeit dokumentiert hat. Dies wird insbesondere im Kapitel 5 deutlich.

Dass es keine direkte Korrelation zwischen der Anzahl der Tasks und dem Gesamtaufwand gibt, bedarf kaum einer weiteren Erläuterung: Manches (wie die Installation von xrdp) war relativ schnell erledigt, während andere Arbeiten (wie die Annäherung an Lernstick EDU) viele Tage und Nerven gekostet haben.

Zusätzlich zu diesem Kapitel wird im Anhang bei jedem Quellcode unter ≪Federführender Autor≫ bzw. Programm angegeben, welcher der beiden Autoren den grösseren Beitrag zu diesem Quellcode geleistet hat.

9.1 Alexandre Warin

- Entwicklung der Projektidee.
- Vorabrecherche zu virtuellen Maschinen und Containerlösungen sowie Sicherheitsaspekten (wie zum Beispiel die Lösung von Kata).
- Vorabrecherche zu SPICE, RDP, VNC und Guacamole.
- Erstellung einer stark vereinfachten ersten Vorabversion für erste Tests von LXC und Guacamole (nicht auf dem Server, sondern auf einer virtuellen Ubuntu-Maschine).

- Entwurf des Containersystems mit *euler* und den Subcontainern.
- Fehlgeschlagene Ausbruchsversuche aus den Containern.
- Installation der Basissysteme *euler*, *guacamole*, *balancer*, *master*, *manager* und anfangs auch *accounts* und *storage*. Dazu die Recherche, wie ein Bug im LXC Storage Driver umgangen werden kann.
- Installation und Einrichtung von kubuntu-desktop, xrdp und guacamole.
- Erarbeiten der Lösung für Quotas und verschlüsselte Homeverzeichnisse.
- Konzeption der Account-Verwaltung:
  - Versuche mit LDAP, SSSD und NIS.
  - Extrauser recherchieren und einrichten.
  - Passwortmanagement über eigenes passwd-Skript.
  - Pushen von Passwortänderungen an Guacamole über `user-mapping.xml`.
  - Experimente mit PAM.
  - Einrichten eines Request-Systems zum Löschen von Benutzern.
  - Entwicklung des Requestsystems für die Accounterstellung via webapp-Server.
- Design und Implementierung des *deployed-master-testing* Subcontainer Trio Systems.
- Einrichtung eines Balancing-Systems zur Nutzung von Sekundärservern (ungetestet).
- Intensive Überlegungen nach bestem Wissen und Gewissen zum Thema Sicherheit in jedem einzelnen Arbeitsschritt. (einige Wege wurden dann aufgrund dieser Überlegungen wegen zu hoher Sicherheitsrisiken verworfen).

9.2 Olivier Warin

- Verbesserung des Account Management Systems über PAM in C statt über Shell-Skripts. Dies beinhaltet die Einarbeitung in sowohl in PAM als auch in libxml2.
- Alle Programme in C.
- Alle Systemd-Dienste und -Timer.
- System mit Anfragen zum Löschen von Benutzer in der zweiten überarbeiteten Version des ganzen Systems.
- System, dass dafür sorgt, dass die Datei `/etc/hosts` in allen Containern aktuell bleibt.
- Anpassungen, um das System näher an Lernstick EDU zu bringen.
- Integration von Microsoft OneDrive.

- Unterstützung der Soundausgabe in den Desktops.
- Anleitungsvideos für die Schülerinnen und Schüler [oli23a], [oli23b], [oli23c]. [oli23d],
- Erstellung der Webseite [WW23a] und komplette Überarbeitung der PHP-Skripte.
- Grobes Konzept der schriftlichen Arbeit.
- LaTeX Layout
- Bereinigung der LaTeX- und Programmcodes.
- Zusammenstellung aller Skripts und Programme im Anhang.
- Willkommensnachricht `welcome.sh` (Listing 26 auf Seite 134)
- Grafisches Passwortwechel-Tool `change-password.sh` (Listing 14 auf Seite 127)
- Grafisches Administrationstool `account-management.sh` (Listing 13 auf Seite 126)
- Patchen von KDE, damit die Benutzer den Container nicht herunterfahren können.
- Entwicklung eines Systems, mit dem Änderungen an den Shell-Skripten und Konfigurationsdateien einfach per git in die entsprechenden Container gepusht werden können.
- Gestaltung aller Bilder: Übersichtsgrafiken, Hintergrundbilder (Administrator und normale Benutzer) und Logo. Letzteres mit Hilfe von DALL-E [Ope21].

Quellenverzeichnis

[Ame23] Free Software Foundation Latin America. *GNU Linux-libre*. 1. September 2023. URL: https://www.fsfla.org/ikiwiki/selibre/linux-libre/ (besucht am 8. Oktober 2023).

[And23] matts1 und Andere. *fuse2fs.c*. 1. Februar 2023. URL: https://github.com/tytso/e2fsprogs/blob/master/misc/fuse2fs.c (besucht am 24. Juli 2023).

[Aut23a] OpenSSL Project Authors. *OpenSSL. Cryptography and SSL/TLS Toolkit*. 2023. URL: https://www.openssl.org/ (besucht am 15. September 2023).

[Aut23b] The Kubernetes Authors. *Production-Grade Container Orchestration*. 2023. URL: https://kubernetes.io/ (besucht am 21. Juli 2023).

[Avr21] Yuval Avrahami. *Escaping virtualized Containers*. 26. Februar 2021. URL: https://youtu.be/jFlqVe11eeM (besucht am 20. Juli 2023).

[Ban21] Avimanyu Bandyopadhyay. *What is a Hypervisor? What's the Difference Between Type 1 and 2?* 27. Dezember 2021. URL: https://linuxhandbook.com/what-is-hypervisor/ (besucht am 19. Juli 2023).

[Bel+23] Steven Bell u. a. *Extreme Tux Racer*. 19. Juni 2023. URL: https://sourceforge.net/projects/extremetuxracer/ (besucht am 11. August 2023).

[Ber21] Universität Bern. *Lernstick Documentation*. 2021. URL: https://lernstick-doc.readthedocs.io/de/latest/index.html (besucht am 19. Juli 2023).

[Ber23] Universität Bern. *Forschungsstelle Digitale Nachhaltigkeit*. 8. Oktober 2023. URL: https://www.digitale-nachhaltigkeit.unibe.ch/dienstleistungen/lernstick/team/index_ger.html (besucht am 2023).

[Bro12] Jon Brodkin. *Ubuntu bakes Amazon search results into OS to raise cash*. 24. September 2012. URL: https://arstechnica.com/information-technology/2012/09/ubuntu-bakes-amazon-search-results-into-os-to-raise-cash/ (besucht am 21. Juli 2023).

[cha] chaifeng. *To Fix The Docker and UFW Security Flaw Without Disabling Iptables*. URL: https://github.com/chaifeng/ufw-docker (besucht am 21. Juli 2023).

[Coma] Wikimedia Commons. *Leonhard Euler*. public domain. URL: https://commons.wikimedia.org/wiki/File:Leonhard_Euler.jpg (besucht am 7. Juli 2020).

[Comb] Wikimedia Commons. *Linux Containers logo*. public domain. URL: https://commons.wikimedia.org/wiki/File:Linux_Containers_logo.svg (besucht am 7. August 2023).

[Comc] Wikimedia Commons. *Logo of the Xfce project*. GNU Lesser General Public License. URL: https://commons.wikimedia.org/wiki/File:Xfce_logo.svg (besucht am 21. Juli 2023).

[coma] The kernel development community. *CIFS*. URL: https://www.kernel.org/doc/html/latest/admin-guide/cifs/index.html (besucht am 28. Juli 2023).

[comb] The kernel development community. *NFS*. URL: https://docs.kernel.org/admin-guide/nfs/index.html (besucht am 28. Juli 2023).

[Com07] Wikimedia Commons. *Knoppix logo*. GNU General Public License. 1. Oktober 2007. URL: https://commons.wikimedia.org/wiki/File:Knoppix_logo.svg (besucht am 26. Juli 2023).

[Com19] Wikimedia Commons. *Richard Stallman*. Creative Commons Attribution 4.0 International. 2019. URL: https://commons.wikimedia.org/wiki/File:Richard_Stallman_at_LibrePlanet_2019.jpg (besucht am 28. Januar 2023).

[Com23a] The KDE Community. *Kiosk Keys*. 2023. URL: https://develop.kde.org/docs/administration/kiosk/keys/ (besucht am 19. September 2023).

[Com23b] The KDE Community. *Shell Scripting with KDE Dialogs*. 2023. URL: https://develop.kde.org/docs/administration/kdialog/ (besucht am 8. August 2023).

[con23a] LXD contributors. *LXD documentation*. 17. Juli 2023. URL: https://documentation.ubuntu.com/lxd/en/latest/ (besucht am 18. Juli 2023).

[con23b] LXDE contributors. *LXDE. Desktop Environment for all*. 2023. URL: https://www.lxde.org/ (besucht am 27. Juli 2023).

[Cra23a] Nick Craig-Wood. *Crypt*. 30. Juni 2023. URL: https://rclone.org/crypt/ (besucht am 23. Juli 2023).

[Cra23b] Nick Craig-Wood. *Rclone. Rclone syncs your files to cloud storage*. 10. September 2023. URL: https://rclone.org/ (besucht am 15. September 2023).

[Cra23c] Nick Craig-Wood. *rclone. Supported providers*. 6. Juli 2023. URL: https://rclone.org/#providers (besucht am 28. Juli 2023).

[cry23] cryptsetup. *LUKS. Linux Unified Key Setup*. 18. Juli 2023. URL: https://gitlab.com/cryptsetup/cryptsetup (besucht am 28. Juli 2023).

[DA17] Virgil Dupras und Morgan Aubert. *Shared folders*. 2017. URL: https://lxdock.readthedocs.io/en/stable/usage/shared_folders.html (besucht am 28. Juli 2023).

[dar23] darknite. *Hack The Box: (Extension) Docker escape on root privileges*. 21. März 2023. URL: https://threatninja.net/2023/03/hack-the-box-extension-docker-escape-on-root-privileges/ (besucht am 20. Juli 2023).

[Dat23] Datenschutz.org. *Technisch notwendige Cookies einfach erklärt*. 12. Juni 2023. URL: https://www.datenschutz.org/technisch-notwendige-cookies/ (besucht am 13. August 2023).

[Dav23] Wayne Davison. *rsync*. 29. April 2023. URL: https://rsync.samba.org/ (besucht am 15. September 2023).

[dca22] HiFiJ und dcato. *Looking to install ubuntu-desktop on Proxmox LXC Template 21.04*. 23. Juni 2022. URL: https://forum.proxmox.com/threads/looking-to-install-ubuntu-desktop-on-proxmox-lxc-template-21-04.88175/ (besucht am 20. Juli 2023).

[Deb20] Debian. *Debian Open Use Logo*. Creative Commons Attribution-ShareAlike 3.0 Unported License. 5. Oktober 2020. URL: https://www.debian.org/logos/openlogo.svg (besucht am 20. Juli 2023).

[dev22] Kubuntu devs. *Kubuntu 22.04 LTS Released*. 21. April 2022. URL: https://kubuntu.org/news/kubuntu-22-04-lts-released/ (besucht am 5. September 2023).

[doc] docker docs. *Run the Docker daemon as a non-root user (Rootless mode)*. URL: https://docs.docker.com/engine/security/rootless/ (besucht am 20. Juli 2023).

[Don23] DontBreakDebian. *Don't make a FrankenDebian*. 24. Mai 2023. URL: https://wiki.debian.org/DontBreakDebian#Don.27t_make_a_FrankenDebian (besucht am 26. Juli 2023).

[EEB] NW EDK, EDK-Ost und BKZ. *Nutzung der Basisschrift*. URL: https://basisschrift.ch/nutzung-der-basisschrift (besucht am 27. Juli 2023).

[era23] eradman. *Event Notify Test Runner (entr)*. 25. August 2023. URL: https://eradman.com/entrproject/ (besucht am 15. September 2023).

[eye19] eyeos. *spice-web-client*. 3. Juni 2019. URL: https://github.com/eyeos/spice-web-client (besucht am 19. Juli 2023).

[Fac23] Berner Fachhochschule. *Lernstick*. 2023. URL: https://www.bfh.ch/de/forschung/forschungsbereiche/lernstick/ (besucht am 19. Juli 2023).

[Foua] OpenInfra Foundation. *The Speed of containers, the security of VMs*. URL: https://katacontainers.io/ (besucht am 20. Juli 2023).

[Foub] OpenLDAP Foundation. *OpenLDAP*. URL: https://www.openldap.org/ (besucht am 8. Oktober 2023).

[Fouc] The GNOME Foundation. *GNOME Logo*. URL: https://foundation.gnome.org/wp-content/uploads/sites/12/2021/03/GnomeLogoHorizontal.svg (besucht am 21. Juli 2023).

[Fou22] Free Software Foundation. *Free Software and Education*. 2022. URL: https://www.gnu.org/education/education.html (besucht am 19. Juli 2023).

[Fou23a] Free Software Foundation. *Free Software Foundation*. 2023. URL: https://www.fsf.org (besucht am 19. Juli 2023).

[Fou23b] Free Software Foundation. *Staff and Board*. 2023. URL: https://www.fsf.org/about/staff-and-board (besucht am 20. Juli 2023).

[Fou23c] Free Software Foundation. *What is Free Software?* 2023. URL: https://www.gnu.org/philosophy/free-sw.html.en (besucht am 19. Juli 2023).

[Fou23d] The Apache Software Foundation. *Apache Guacamole Manual*. 2023. URL: https://guacamole.apache.org/doc/gug/index.html.

[Fou23e] The Apache Software Foundation. *Custom authentication*. 2023. URL: https://guacamole.apache.org/doc/gug/custom-auth.html (besucht am 8. Oktober 2023).

[Fre20] Inc Free Software Foundation. *GNU Stow*. 1. November 2020. URL: https://www.gnu.org/software/stow/ (besucht am 27. September 2023).

[fre23] freedesktop.org. *PulseAudio*. 23. Mai 2023. URL: https://www.freedesktop.org/wiki/Software/PulseAudio/ (besucht am 28. Juli 2023).

[Gat23] Antenore Gatta. *Remote access screen and file sharing to your desktop*. 2023. URL: https://remmina.org/ (besucht am 26. Juli 2023).

[git] git.launchpad.net. *index: ubuntu/+source/libnss-extrausers*. URL: https://git.launchpad.net/ubuntu/+source/libnss-extrausers/tree/s_config.h (besucht am 8. August 2023).

[gnu21] gnu.org. *GNU GRUB*. 31. August 2021. URL: https://www.gnu.org/software/grub/ (besucht am 27. Juli 2023).

[gpa] gparted.org. *GNOME Partition Editor*. URL: https://gparted.org/ (besucht am 27. Juli 2023).

[Gra20] Stéphane Graber. *Running virtual machines with LXD 4.0*. April 2020. URL: https://discuss.linuxcontainers.org/t/running-virtual-machines-with-lxd-4-0/7519 (besucht am 20. Juli 2023).

[Gra22] Stéphane Graber. *Migrating systems to LXD*. YouTube. 3. November 2022. URL: https://youtu.be/F9GALjHtnUU (besucht am 17. Juli 2023).

[Gri23a] Griffon. *xRDP Easy install xRDP on Ubuntu 18.04, 20.04, 22.04, 22.10, 23.04 (Script Version 1.4.7)*. 2. Mai 2023. URL: https://c-nergy.be/blog/?p=18918 (besucht am 21. Juli 2023).

[Gri23b] Griffon. *xRDP Testing Initial Pipewire Sound Rediretion Support in Ubuntu 22.10 (Early Stage)*. 29. Januar 2023. URL: https://c-nergy.be/blog/?p=18772 (besucht am 28. Juli 2023).

[Gro] Internet Security Research Group. *Let's Encrypt*. URL: https://letsencrypt.org/ (besucht am 22. Juli 2023).

[Hac] HackTricks. *Escaping from Jails*. URL: https://book.hacktricks.xyz/linux-hardening/privilege-escalation/escaping-from-limited-bash (besucht am 21. Juli 2023).

[hap] haproxy. *haproxy*. URL: https://www.haproxy.com/ (besucht am 12. August 2023).

[Hat21] Black Hat. *Alcatraz: A Practical Hypervisor Sandbox to Prevent Escapes from the KVM/QEMU and KVM-Based MicroVMs*. YouTube. 6. Dezember 2021. URL: https://youtu.be/8R-gaIFeMfs (besucht am 5. Februar 2023).

[hef20] heftig. *REALTIMEKIT Realtime Policy and Watchdog Daemon*. 5. April 2020. URL: https://github.com/heftig/rtkit (besucht am 28. Juli 2023).

[Hor17] Tylor Hornby. *Gocryptfs Security Audit*. 3. März 2017. URL: https://defuse.ca/audits/gocryptfs.htm (besucht am 23. Juli 2023).

[Ikr19] Ikraav. *What is the correct way to prevent non-root users from issuing shutdowns or reboots*. 26. Mai 2019. URL: https://superuser.com/questions/354678/what-is-the-correct-way-to-prevent-non-root-users-from-issuing-shutdowns-or-rebo (besucht am 31. Juli 2023).

[Inc] Docker Inc. *Develop faster. Run anywhere.* URL: https://www.docker.com/.

[Inc23a] BigBlueButton Inc. *Virtual Classroom Software*. 2023. URL: https://bigbluebutton.org/ (besucht am 26. Juli 2023).

[Inc23b] Docker Inc. *Docker Logo*. 2023. URL: https://www.docker.com/wp-content/uploads/2022/03/horizontal-logo-monochromatic-white.png (besucht am 21. Juli 2023).

[ino23] inotify-tools. *inotify-tools*. 12. September 2023. URL: https://github.com/inotify-tools/inotify-tools (besucht am 21. September 2023).

[iti] itiligent. *Guacamole 1.5.2 Virtual Desktop & Jump Server Appliance*. URL: https://github.com/itiligent/Guacamole-Setup (besucht am 21. Juli 2023).

[Jia19] Li Jiang. *The Official Free Software Foundation Logo*. Creative Commons Zero 1.0 Universal Public Domain Dedication. 2019. URL: https://www.gnu.org/graphics/fsf-logo.html (besucht am 19. Juli 2023).

[jst] jstaf. *onedriver*. URL: https://github.com/jstaf/onedriver (besucht am 10. August 2023).

[Jum15] Michael Jumper. *Chapter 8: Disabling authentication*. 2015. URL: https://guacamole.apache.org/doc/0.9.7/gug/noauth.html (besucht am 9. Oktober 2023).

[Kan22] Erziehungsdepartement des Kantons Basel-Stadt. *AKOM Mindestanforderungen Geräte BYOD*. 19. Dezember 2022. URL: https://wg.edubs.ch/schulprofil/byod-bring-your-own-device/link/akom-mindestanforderungen-gerate-byod-sus-ver06.pdf/download (besucht am 19. Juli 2023).

[kda] kdave. *Welcome to BTRFS documentation!* URL: https://btrfs.readthedocs.io/en/latest/ (besucht am 28. Juli 2023).

[KDEa] Team KDE. *KDE Logo*. URL: https://kde.org/stuff/clipart/logo/kde-logo-white-blue-rounded-source.svg (besucht am 21. Juli 2023).

[KDEb] Team KDE. *Plasma Logo*. URL: https://kde.org/stuff/clipart/logo/plasma-logo-colorful.svg (besucht am 21. Juli 2023).

[KDE20] KDE. *Kmail/Configuring Kmail/Accounts/Office 365*. 15. Juni 2020. URL: https://userbase.kde.org/Kmail/Configuring_Kmail/Accounts/Office_365 (besucht am 21. Juli 2023).

[KDE23] KDE. *Plasma Workspace*. 28. September 2023. URL: https://github.com/KDE/plasma-workspace/tree/Plasma/5.25 (besucht am 17. August 2023).

[KG23] Manjaro GmbH & Co KG. *Manjaro empowering devices and users*. 2023. URL: https://manjaro.org/ (besucht am 26. Juli 2023).

[Kno] Klaus Knopper. *KNOPPIX*. URL: https://www.knopper.net/knoppix/ (besucht am 20. Juli 2023).

[Ler] Lernstick. *Logo von Lernstick EDU*. URL: https://www.bfh.ch/.imaging/mte/bfh-theme/image-and-gallery-xl/dam/bfh.ch/forschung/wirtschaft/Digital-sustainability-lab/lernstick/Lernstick-EDU.jpg/jcr:content/Lernstick-EDU.jpg (besucht am 19. Juli 2023).

[Ler23] Lernstick. *lernstickWelcome*. 16. Februar 2023. URL: https://github.com/Lernstick/lernstickWelcome (besucht am 27. Juli 2023).

[lib22] libfuse. *SSHFS*. 26. Mai 2022. URL: https://github.com/libfuse/sshfs (besucht am 28. Juli 2023).

[Lim23] OffSec Services Limited. *The most advanced Penetration Testing Distribution*. 2023. URL: https://www.kali.org/ (besucht am 26. Juli 2023).

[Lin] Arch Linux. *LDAP Authentication*. URL: https://wiki.archlinux.org/title/LDAP_authentication (besucht am 21. Juli 2023).

[lina] linux-kvm.org. *Kernel Virtual Machine*. URL: https://www.linux-kvm.org/page/Main_Page (besucht am 26. Juli 2023).

[linb] linuxcontainers.org. *Container and virtualization tools*. Content under Creative Commons CC BY NC SA. URL: https://linuxcontainers.org/ (besucht am 21. Juli 2023).

[lin23a] linux-pam. *Linux-PAM*. 25. August 2023. URL: http://www.linux-pam.org/ (besucht am 11. September 2023).

[lin23b] linuxcontainers.org. *LXC Documentation*. 17. Juli 2023. URL: https://linuxcontainers.org/lxc/documentation/ (besucht am 18. Juli 2023).

[lin23c] linuxcontainers.org. *LXD is now under Canonical*. 4. Juli 2023. URL: https://linuxcontainers.org/lxd/ (besucht am 21. Juli 2023).

[Lle20] Gerard Lledó. *ext4fuse*. 29. September 2020. URL: https://github.com/gerard/ext4fuse (besucht am 24. Juli 2023).

[Ltda] Canonical Ltd. *Ubuntu*. URL: https://ubuntu.com/ (besucht am 21. Juli 2023).

[Ltdb] Canonical Ltd. *Ubuntu Logo*. URL: https://assets.ubuntu.com/v1/ff6a9a38-ubuntu-logo-2022.svg (besucht am 21. Juli 2023).

[Ltd22] Canonical Ltd. *Kubuntu Backports Extra*. 13. September 2022. URL: https://launchpad.net/~%20kubuntu-ppa/+archive/ubuntu/backports-extra (besucht am 17. August 2023).

[Ltd23a] Canonical Group Ltd. *Cloud-init*. 2023. URL: https://cloudinit.readthedocs.io/en/latest/index.html (besucht am 8. Oktober 2023).

[Ltd23b] Canonical Ltd. *Run system containers with LXD*. 2023. URL: https://ubuntu.com/lxd (besucht am 21. Juli 2023).

[Mic] Microsoft. *Microsoft Teams*. URL: https://www.microsoft.com/en-us/microsoft-teams/group-chat-software/ (besucht am 26. Juli 2023).

[Mic22a] Microsoft. *Remote Desktop Protocol: Basic Connectivity and Graphics Remoting*. 9. März 2022. URL: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpbcgr (besucht am 19. Juli 2023).

[Mic22b] Microsoft. *Windows Subsystem for Linux Documentation*. 27. Juni 2022. URL: https://learn.microsoft.com/en-us/windows/wsl/ (besucht am 19. Juli 2023).

[Mic23] Microsoft. *ARM-based Surface devices FAQ*. 1. April 2023. URL: https://learn.microsoft.com/en-us/surface/surface-arm-faq (besucht am 19. Juli 2023).

[Mol22] MolnarSzabolcsErdely. *pixabay*. 22. September 2022. URL: https://pixabay.com/photos/blue-background-grunge-abstract-7470781/ (besucht am 24. Juli 2023).

[Moo23a] Moodle. *Die Moodle-Geschichte*. 2023. URL: https://moodle.com/de/uber/die-moodle-geschichte/ (besucht am 23. Juli 2023).

[Moo23b] Moodle. *Statistics*. 2023. URL: https://stats.moodle.org/ (besucht am 23. Juli 2023).

[MP78] MD McIlroy und EN Pinson. «BA Tague Unix Time-Sharing System Forward». In: *The Bell System Technical Journal. Bell Laboratories* (1978).

[Mün21] Gymnasium Münchenstein. *BYOD – Anforderungen an die Geräte*. Februar 2021. URL: https://www.gymmuenchenstein.ch/images/0-downloads/Unsere_Schule/BYOD_Anforderungen.pdf (besucht am 19. Juli 2023).

[Mut23] Gymnasium Muttenz. *BYOD – Anforderungen an die Geräte*. 2023. URL: https://www.gym-muttenz.ch/fileadmin/user_upload/Geraetevorgaben_2023.pdf (besucht am 19. Juli 2023).

[neta] netcup.de. *Root-Server*. URL: https://www.netcup.de/vserver/ (besucht am 20. Juli 2023).

[netb] networkmanager.dev. *NetworkManager*. URL: https://networkmanager.dev/ (besucht am 27. Juli 2023).

[neu23] neutrinolabs. *xrdp*. 2023. URL: https://github.com/neutrinolabs/xrdp.

[oli23a] oli4math. *linuxcloud.ch: guacamole Menu öffnen und schliessen*. YouTube. 3. September 2023. URL: https://youtu.be/6AvnCnVn4ww (besucht am 3. September 2023).

[oli23b] oli4math. *linuxcloud.ch: Objekte mit dem Touchscreen verschieben*. YouTube. 3. September 2023. URL: https://youtu.be/Farncez20XQ (besucht am 3. September 2023).

[oli23c] oli4math. *linuxcloud.ch: OneDrive einbinden*. YouTube. 19. September 2023. URL: https://youtu.be/_llp9Adx348 (besucht am 19. September 2023).

[oli23d] oli4math. *linuxcloud.ch: Verschiedene Eingabemethoden mit einem Touchscreen*. YouTube. 3. September 2023. URL: https://youtu.be/r7lLZ9hQosQ (besucht am 3. September 2023).

[Ope21] OpenAI. *DALL-E: Creating Images with Text*. 2021. URL: https://openai.com/research/dalle/ (besucht am 21. Juli 2023).

[Ora] Oracle. *Logo von VirtualBox*. URL: https://www.virtualbox.org/graphics/vbox_logo2_gradient.png (besucht am 19. Juli 2023).

[Ora23a] Oracle. *Download VirtualBox (Old Builds): VirtualBox 7.0*. 2023. URL: https://www.virtualbox.org/wiki/Download_Old_Builds_7_0 (besucht am 20. Juli 2023).

[Ora23b] Oracle. *VirtualBox*. 2023. URL: https://www.virtualbox.org (besucht am 19. Juli 2023).

[Paa04] Daniel Paarmann. *synaptic*. 2004. URL: https://www.nongnu.org/synaptic/ (besucht am 27. Juli 2023).

[PG23] Bernard Parisse und Renée De Graeve. *Giac/Xcas, a free computer algebra system*. 2023. URL: https://www-fourier.ujf-grenoble.fr/~parisse/giac.html (besucht am 27. Juli 2023).

[Pip22] PipeWire. *pipewire*. 2022. URL: https://pipewire.org/ (besucht am 28. Juli 2023).

[Pod] Podman. *The best free & open source container tools*. URL: https://podman.io/ (besucht am 21. Juli 2023).

[Pro] The GNOME Project. *GNOME. Get things done with ease, comfort, and control*. URL: https://www.gnome.org/ (besucht am 27. Juli 2023).

[Pro22] The GNOME Project. *libxml2: Tht XML C Parser and Toolkit of Gnome*. 8. Dezember 2022. URL: http://xmlsoft.org.

[qem] qemu.org. *QEMU. A generic and open source machine emulator and virtualizer*. URL: https://www.qemu.org (besucht am 19. Juli 2023).

[Qua] Qualculate. *Qalculate! the ultimate desktop calculator*. URL: https://qalculate.github.io/ (besucht am 27. Juli 2023).

[rfj23] rfjakob. *gocryptfs*. 17. Juni 2023. URL: https://github.com/rfjakob/gocryptfs (besucht am 23. Juli 2023).

[Rod23] J.C. Rodríguez-del-Pino. *VPL, the Virtual Programming lab for Moodle*. 8. Juni 2023. URL: https://vpl.dis.ulpgc.es/ (besucht am 23. Juli 2023).

[Sch11] Jürgen Schmidt. *Desinfec't*. 23. März 2011. URL: https://www.heise.de/hintergrund/Desinfec-t-2011-1213110.html (besucht am 20. Juli 2023).

[Sec21] SecNigma. *Hack The Box: Ready*. 15. Mai 2021. URL: https://secnigma.wordpress.com/2021/05/15/hack-the-box-ready/ (besucht am 20. Juli 2023).

[spia] spice.org. *Simultaneous clients connection*. URL: https://www.spice-space.org/multiple-clients.html (besucht am 19. Juli 2023).

[spib] spice.org. *SPICE*. URL: https://www.spice-space.org/index.html (besucht am 19. Juli 2023).

[spi21a] spice. *spice-html5*. 4. Mai 2021. URL: https://gitlab.freedesktop.org/spice/spice-html5 (besucht am 19. Juli 2023).

[spi21b] spice. *spice-html5*. 4. Mai 2021. URL: https://gitlab.freedesktop.org/spice/spice-html5 (besucht am 19. Juli 2023).

[SSS] SSSD. *sssd Open Source Client for Enterprise Identity Managment*. URL: https://sssd.io/ (besucht am 22. Juli 2023).

[Sta] Prof. Dr. Ronny Standtke. *Pauker*. URL: https://pauker.sourceforge.net/ (besucht am 26. Juli 2023).

[Sta02] Richard Stallman. *Free Software, Free Society: Selected Essays of Richard M. Stallman. subtitle*. GNU Press, 2002. ISBN: 1-882114-98-1. URL: https://www.gnu.org/philosophy/fsfs/rms-essays.pdf.

[Sta99] Richard Stallman. *Why Schools Should Exclusively Use Free Software*. 1999. URL: https://www.gnu.org/education/edu-schools.html (besucht am 29. Januar 2023).

[Ste23] Stellarium. *stellarium*. 10. August 2023. URL: https://stellarium.org/ (besucht am 11. August 2023).

[ste23] stefanbaur. *X2Go - everywhere@home*. 4. Juli 2023. URL: https://wiki.x2go.org/doku.php (besucht am 26. Juli 2023).

[tai23] tails.net. *Tails*. 2023. URL: https://tails.net/ (besucht am 26. Juli 2023).

[Teaa] GTK Development Team. *GTK FileChooserDialog*. URL: https://docs.gtk.org/gtk3/class.FileChooserDialog.html (besucht am 16. August 2023).

[Teab] Linux Mint Development Team. *Linux Mint Projects*. URL: https://projects.linuxmint.com/cinnamon/ (besucht am 27. Juli 2023).

[Tea23a] Alpine Linux Development Team. *Logo von Alpine Linux*. 2023. URL: https://alpinelinux.org/alpinelinux-logo.svg (besucht am 20. Juli 2023).

[Tea23b] The MATE Team. *MATE Desktop Environment*. 2023. URL: https://mate-desktop.org/ (besucht am 27. Juli 2023).

[Tea23c] Xfce Development Team. *Xfce Desktop Environment*. 2023. URL: https://www.xfce.org/ (besucht am 27. Juli 2023).

[tor] torproject.org. *Browse Privately. Explore Freely.* URL: https://www.torproject.org/ (besucht am 26. Juli 2023).

[tra] transmission. *Transmission. A Fast, Easy and Free Bittorrent Client*. URL: https://transmissionbt.com/ (besucht am 26. Juli 2023).

[vmwa] vmware. *Logo von vmware*. URL: https://www.vmware.com/content/dam/digitalmarketing/vmware/en/images/company/vmware-logo-grey.svg (besucht am 19. Juli 2023).

[vmwb] vmware.com. *vmware*. URL: https://www.vmware.com (besucht am 19. Juli 2023).

[Vol] Patrick Volkerding. *ZipSlack*. URL: http://www.slackware.com/zipslack/ (besucht am 8. Oktober 2023).

[vsc] vscodium.com. *VSCodium. Free/Libre Open Source Software Binaries of VS Code*. URL: https://vscodium.com/ (besucht am 27. Juli 2023).

[Web22] KDE Webmasters. *Plasma 5.25*. 17. Juni 2022. URL: https://kde.org/announcements/plasma/5/5.25.0/ (besucht am 17. August 2023).

[WW23a] Alexandre Warin und Olivier Warin. *Kontoverwaltung für linuxcloud.ch*. 7. Oktober 2023. URL: https://webapp-gymms.sbl.ch/linuxcloud/auth/ (besucht am 7. Oktober 2023).

[WW23b] Alexandre Warin und Olivier Warin. *linuxcloud-setup*. 9. Oktober 2023. URL: https://gitlab.com/wasto/linuxcloud-setup (besucht am 9. Oktober 2023).

[yt-23] yt-dlp. *yt-dlp*. 24. September 2023. URL: https://github.com/yt-dlp/yt-dlp (besucht am 27. September 2023).

Abbildungsverzeichnis

Anhang

Dieser Anhang enthält eine Übersicht über die Skripte und Programme, die für den Betrieb des Systems verwendet werden. Alle genannten Ressourcen befinden sich in einem öffentlichen Git-Repository[1] [WW23b], das auch zusätzliche Konfigurationsdateien enthält. Ausserdem befinden sich dort im Ordner `deprecated` Skripte bzw. Versionen von Skripten, die für die Prototypversion verwendet wurden.

Ziel dieser Zusammenstellung ist es, eine transparente und zugängliche Quelle und Dokumentation für die relevanten Codes und Dateien bereitzustellen, um Interessierten einen umfassenden Einblick in die Funktionalität des Systems zu ermöglichen.

`{deployed,manager,master,testing}:/lib/x86_64-linux-gnu/pam_guac.so`

Federführender Autor Olivier Warin

Beschreibung Ein PAM-Modul, das bei einer Passwortänderung auch das entsprechende Passwort für Guacamole in der Datei `user-mapping.xml` anpasst. Genauer gesagt wird dort der entsprechende SHA256-Hash eingetragen.

Fehlt der Benutzer in `user-mapping.xml`, wird dieser entsprechend angelegt. Somit kann (auf *manager*) einfach ein normaler Befehl wie `useradd` verwendet werden, um einen neuen Benutzer anzulegen. Sobald für diesen Benutzer ein Passwort gesetzt wird, wird dieses automatisch in `user-mapping.xml` eingetragen.

Verwendung `pam_guac.so <user-mapping-file>`

Das Programm wird automatisch durch PAM ausgeführt.

```
25 password required pam_guac.so /var/guacamoleusers/user-mapping.xml
26 password        [success=1 default=ignore]      pam_unix.so obscure yescrypt use_authtok
27 password        [success=1 default=ignore]      pam_extrausers.so obscure yescrypt use_authtok
```

Listing 10: Eintrag in `{deployed,master,testing}:/etc/pam.d/password-common`

[1] Dieses Repository wurde erst mit Beginn der zweiten Version (siehe Kapitel 6) gestartet. Ein Grossteil der Entwicklungsschritte ist daher dort nicht dokumentiert.

Programmiersprache C

Spezielle Abhängigkeiten

- libxml2 [Pro22]
- PAM [lin23a]
- OpenSSL [Aut23a]

Quellcode

```c
#include <libxml/parser.h>
#include <libxml/tree.h>
#include <openssl/evp.h>
#include <openssl/rand.h>
#include <security/pam_appl.h>
#include <security/pam_modules.h>
#include <security/pam_ext.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>

#define usermappingDefaults "/var/guacamoleusers/user-mapping-defaults.xml"

int sha256(const char * password, char ** hash) {
  EVP_MD_CTX * mdctx;
  const EVP_MD * md;
  unsigned char md_value[EVP_MAX_MD_SIZE];
  unsigned int md_len;

  OpenSSL_add_all_digests();
  md = EVP_sha256();

  mdctx = EVP_MD_CTX_new();
  EVP_DigestInit_ex(mdctx, md, NULL);
  EVP_DigestUpdate(mdctx, password, strlen(password));
  EVP_DigestFinal_ex(mdctx, md_value, & md_len);

  EVP_MD_CTX_free(mdctx);

  * hash = (char * ) malloc((md_len * 2 + 1) * sizeof(char));
  if ( * hash == NULL) {
    return PAM_SYSTEM_ERR;
  }

  for (int i = 0; i < md_len; i++) {
    sprintf( * hash + 2 * i, "%02x", md_value[i]);
  }
  return PAM_SUCCESS;
}

```

```
45  int getUID(const char * userName) {
46    struct passwd * userEntry = getpwnam(userName);
47
48    if (userEntry == NULL) {
49      fprintf(stderr, "Error: User '%s' not found!\n", userName);
50      exit(PAM_SYSTEM_ERR);
51    }
52
53    return userEntry -> pw_uid;
54  }
55
56  xmlDocPtr parseXML(const char * fileName) {
57    xmlDocPtr document = xmlReadFile(fileName, NULL, XML_PARSE_NOBLANKS);
58    if (document == NULL) {
59      fprintf(stderr, "Error parsing XML file.\n");
60      exit(PAM_SYSTEM_ERR);
61    }
62    return document;
63  }
64
65  xmlNodePtr getXMLRoot(xmlDocPtr document) {
66    xmlNodePtr root = xmlDocGetRootElement(document);
67    if (root == NULL) {
68      fprintf(stderr, "Empty XML file.\n");
69      exit(PAM_SYSTEM_ERR);
70    }
71    return root;
72  }
73
74  bool isUserInXML(xmlNodePtr root,
75    const char * userName) {
76    for (root = root -> children; root != NULL; root = root -> next) {
77      if (root -> type == XML_ELEMENT_NODE &&
78        xmlStrcmp(root -> name, (const xmlChar * )
79          "authorize") == 0) {
80        xmlChar * user = xmlGetProp(root, (const xmlChar * )
81          "username");
82        if (user != NULL && xmlStrcmp(user, (const xmlChar * ) userName) == 0) {
83          xmlFree(user);
84          return true;
85        }
86        xmlFree(user);
87      }
88    }
89    return false;
90  }
91
92  xmlNodePtr updateXMLUserPassword(xmlNodePtr root,
93    const char * userName, char * hashedPW) {
94    xmlNodePtr authorize;
95    for (authorize = root -> children; authorize; authorize = authorize -> next) {
96      if (xmlStrcmp(authorize -> name, (const xmlChar * )
```

```
        "authorize") == 0) {
      xmlChar * usernameAttr =
        xmlGetProp(authorize, (const xmlChar * )
          "username");

      if (usernameAttr &&
        xmlStrcmp(usernameAttr, (const xmlChar * ) userName) == 0) {
        xmlSetProp(authorize, (const xmlChar * )
          "password",
          (const xmlChar * ) hashedPW);
        xmlFree(usernameAttr);
        break;
      }
      xmlFree(usernameAttr);
    }
  }
  free(hashedPW);
  return root;
}

xmlNodePtr addXMLUser(xmlNodePtr root,
  const char * userName, char * hashedPW) {
  xmlDocPtr tailDoc = parseXML(usermappingDefaults);
  xmlNodePtr newRoot = getXMLRoot(tailDoc);

  // Create the authorize node and set its attributes
  xmlNodePtr authorizeNode = xmlNewChild(root, NULL, BAD_CAST "authorize", NULL);
  xmlNewProp(authorizeNode, BAD_CAST "username", BAD_CAST userName);
  xmlNewProp(authorizeNode, BAD_CAST "password", BAD_CAST hashedPW);
  xmlNewProp(authorizeNode, BAD_CAST "encoding", BAD_CAST "sha256");

  // Copy the child nodes from the new XML document to the authorize node
  xmlNodePtr child = newRoot -> children;
  while (child != NULL) {
    xmlNodePtr newNode = xmlCopyNode(child, 1);
    xmlAddChild(authorizeNode, newNode);

    child = child -> next;
  }
  xmlFreeDoc(tailDoc);
  return root;
}

int saveXMLFile(const char * xmlFile, xmlDocPtr doc) {
  if (xmlSaveFormatFile(xmlFile, doc, 1) == -1) {
    fprintf(stderr, "Error: Could not save the updated XML file.\n");
    xmlFreeDoc(doc);
    return PAM_SYSTEM_ERR;
  }
  xmlFreeDoc(doc);
  return PAM_SUCCESS;
}
```

```
149
150 PAM_EXTERN int pam_sm_chauthtok(pam_handle_t * pamh, int flags, int argc,
151   const char ** argv) {
152   const char * password;
153   char * hashedPW;
154   const char * userName;
155   const char * usermappingFile;
156   if (argc > 0) {
157     usermappingFile = argv[0];
158   } else {
159     usermappingFile = "/var/guacamoleusers/user-mapping.xml";
160   }
161
162   int pamResult;
163
164   if (flags & PAM_UPDATE_AUTHTOK) {
165     //get the username
166     pamResult = pam_get_user(pamh, & userName, NULL);
167     if (pamResult != PAM_SUCCESS) {
168       return pamResult;
169     }
170     //get the password
171     pamResult = pam_get_authtok(pamh, PAM_AUTHTOK, & password, NULL);
172     if (pamResult != PAM_SUCCESS) {
173       return pamResult;
174     }
175     //hash the password
176     pamResult = sha256(password, & hashedPW);
177     if (pamResult != PAM_SUCCESS) {
178       return pamResult;
179     }
180
181     if (getUID(userName) >= 1000) {
182       xmlDocPtr doc = parseXML(usermappingFile);
183       xmlNodePtr root = getXMLRoot(doc);
184       if (isUserInXML(root, userName)) {
185         root = updateXMLUserPassword(root, userName, hashedPW);
186       } else
187         root = addXMLUser(root, userName, hashedPW);
188       return saveXMLFile(usermappingFile, doc);
189     }
190   }
191   return PAM_SUCCESS;
192 }
```

Listing 11: pam_guac.c

{deployed,master,testing}:/home/administrator/.local/bin/current-container.sh

Federführender Autor Alexandre Warin

Beschreibung Präsentiert dem Administrator-User Informationen über den Container, auf dem er sich gerade befindet.

Verwendung `current-container.sh`

Programmiersprache bash

Quellcode

```
#!/bin/bash
case "$HOSTNAME" in
"testing")
  echo "Sie befinden sich im Testing-Container."
  echo "Änderungen bleiben temporär bis Sie den"
  echo "Master-Container neu generieren!"
  echo "Löschen des Testing-Containers ohne zuvor"
  echo "den Master-Container neu zu generieren,"
  echo "macht alle hier durchgeführten Änderungen"
  echo "rückgängig."
  ;;
"master")
  echo "Sie befinden sich im Master-Container."
  echo "Hier sollten Sie nur in Ausnahmefällen"
  echo "arbeiten!"
  echo "Der normale Workflow besteht darin, zuerst"
  echo "einen Testing-Container zu generieren, darin"
  echo "allfällige Änderungen durchzuführen und zu"
  echo "testen, und anschliessend den Master-Container"
  echo "neu zu generieren."
  ;;
"deployed")
  echo "Sie befinden sich im von den Nutzerinnen und"
  echo "Nutzern aktiv im Gebrauch befindlichen Deployed-"
  echo "Container!"
  echo "Normalerweise sollten Sie hier keine Änderungen"
  echo "durchführen, ausser den Nutzerinnen und Nutzern"
  echo "direkt zu helfen."
  ;;
*)
  echo "Unbekannter Hostname: $HOSTNAME"
  ;;
esac
```

Listing 12: `{deployed,master,testing}:/home/administrator/.local/bin/current-container.sh`

`{deployed,master,testing}:/usr/local/sbin/account-management.sh`

Federführender Autor Olivier Warin

Beschreibung Ein grafisches Werkzeug für den Administrator, um Passwörter zurückzusetzen, Benutzerkonten zu erstellen oder zu löschen.

Verwendung Das Skript steht dem Nutzer administrator an einem prominenten Ort auf dem Desktop zur Verfügung.

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten kdialog [Com23b]

Quellcode

```
#!/bin/sh
userMappingFile="/var/guacamoleusers/user-mapping.xml"
userList=$(grep -o 'username="[^"]*"' "$userMappingFile" | sed 's/username="//;s/"//')
title="Verwaltung von Nutzerkonten"
validUsernamePattern="^[a-z][a-z0-9_-]{0,31}$"

choose() {
  choice=$(kdialog --title "$title" --menu "Bitte wählen Sie eine Option" \
    create-user "Konto erstellen" \
    change-password "Passwort eines Kontos ändern" \
    delete-user "Nutzer löschen") && return 0
  return 1
}

createUser() {
  if echo "$userName" | grep -qE "$validUsernamePattern"; then
    newPassword=$(kdialog --title="$title" \
      --newpassword="Geben Sie das gewünschte Passwort ein.") &&
      echo "$newPassword" | request-user-creation.sh "$userName" &&
      kdialog --title="$title" \
        --msgbox "Es wurde eine Anfrage für das Konto $userName an \
den Server geschickt!" && exit 0
  else
    kdialog --error "Ungültiger Benutzername"
  fi
}

deleteUser() {
  if echo "$userName" | grep -qE "$validUsernamePattern"; then
    kdialog --title "$title" --yesno \
      "Sind Sie sicher, dass sie das Konto von $userName löschen möchten?" &&
      request-user-deletion.sh "$userName" &&
      kdialog --title="$title" --msgbox \
        "Es wurde eine Anfrage zur Löschung des Kontos von $userName \
an den Server geschickt!" && exit 0
  else
```

```
37      kdialog --error "Ungültiger Benutzername"
38    fi
39  }
40
41  while choose; do
42    case "$choice" in
43    "create-user")
44      userName=$(kdialog --title="$title" \
45        --inputbox "Bitte geben Sie den gewünschten Benutzernamen ein.") &&
46        createUser
47      ;;
48    "delete-user")
49      userName=$(kdialog --combobox \
50        "Bitte wählen Sie das zu löschende Konto aus." $userList) &&
51        deleteUser
52      ;;
53    "change-password")
54      userName=$(kdialog --combobox \
55        "Bitte wählen Sie das Konto aus für dass Sie das Passwort \
56  neu setzen möchten." $userList) &&
57        createUser
58      ;;
59    esac
60  done
```

Listing 13: {deployed,master,testing}:/usr/local/sbin/account-management.sh

{deployed,master,testing}:/usr/local/bin/change-password.sh

Federführender Autor Olivier Warin

Beschreibung Ein Tool für den Desktop, mit dem Nutzer ihr Passwort einfach und benutzerfreundlich ändern können.

Verwendung Das Skript steht allen Desktop-Nutzern an einem prominenten Ort auf dem Desktop zur Verfügung.

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten kdialog [Com23b]

Quellcode

```
1   #!/bin/sh
2   title="Passwortwechsel"
3   tmpFifo=$(mktemp -u)
4   mkfifo "$tmpFifo"
5   setNewPassword() {
6     if printf "$1\n$2\n$2" | passwd >/dev/null 2>/dev/null; then
7       kdialog --title="$title" --msgbox="Passwort erfolgreich geändert" &&
```

```
      exit 0
  else
    kdialog --title="$title" --error="Passwortwechsel abgelehnt!"
    return 1
  fi
}
retVal=1
while [ $retVal -ne 0 ]; do
  oldPassword=$(kdialog --title="$title" --password="Bitte geben Sie ihr aktuelles Passwort ein")
↪  || exit 1
  printf '%s\0' "$oldPassword" >"$tmpFifo" &
  unix_chkpwd "$USER" nullok <"$tmpFifo"
  retVal=$?
  if [ $retVal -ne 0 ]; then
    kdialog --title="$title" --warningcontinuecancel="Falsches Passwort" || exit 1
  fi
done

rm "$tmpFifo"
while true; do
  newPassword=$(kdialog --title="$title" \
--newpassword="Geben Sie ihr neues Passwort ein.\n\
Bitte verwenden sie kein Passwort was Sie bei einem anderen \
Dienst bereits verwenden!") || exit 1
  setNewPassword "$oldPassword" "$newPassword"
done
```

Listing 14: `{deployed,master,testing}:/usr/local/bin/change-password.sh`

`{deployed,master,testing}:/usr/local/bin/get-remaining-images.sh`

Federführender Autor Alexandre Warin

Beschreibung Zeigt an, wie viele noch verfügbare Images für verschlüsselte Home-Verzeichnisse vorhanden sind.

Verwendung `get-remaining-images.sh`

Programmiersprache POSIX Shell-Skript

Quellcode

```
#!/bin/sh
(
  awk -F: '($3 >= 1000 && $3 < 65534) {printf "%s\n",$3}' /var/lib/extrausers/passwd
  ls -1 /decryptedhomes/
) | sort | uniq -u | wc -l
```

Listing 15: `{deployed,master,testing}:/usr/local/bin/get-remaining-images.sh`

`{deployed,master,testing}:/usr/local/bin/setup-onedrive-mount.sh`

Federführender Autor Olivier Warin

Beschreibung Das Skript `setup-onedrive-mount.sh` legt ein OneDrive-Laufwerk in der Konfigurationsdatei von rclone an.

Das Skript wurde mit einem OneDrive-Zugang des Kantons Baselland getestet, wie er allen Lernenden und Lehrenden dieses Kantons zur Verfügung steht.

Verwendung Das Skript `setup-onedrive-mount.sh` wird allen Desktopbenutzern prominent auf dem Desktop angezeigt.

Zusätzlich wird nach erfolgreicher Konfiguration der Systemd-Dienst `onedrive-mount.service` aktiviert. Dieser sorgt dafür, dass der entsprechende OneDrive-Ordner bei jedem Login automatisch gemountet wird.

```
[Unit]
Description=OneDrive mount (rclone)
AssertPathIsDirectory=%h/Desktop/OneDrive
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/rclone mount --vfs-cache-mode minimal onedrive: Desktop/OneDrive
ExecStop=/usr/bin/fusermount -u Desktop/OneDrive
Restart=on-failure
RestartSec=10s

[Install]
WantedBy=default.target
```

Listing 16: `manager:/etc/skel/.config/systemd/user/onedrive-mount.service`

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten

- rclone [Cra23b]
- kdialog [Com23b]

Quellcode

```
#!/bin/sh

remoteName="onedrive"

if rclone listremotes | grep -q "^$remoteName:"; then
  kdialog --warningyesno \
    "Es gibt bereits einen Eintrag Namens $remoteName bei rclone. \n
```

```
  Möchten Sie diesen Löschen und neu erstellen?" &&
    rclone config delete "$remoteName" || exit 1
fi

rclone --onedrive-drive-type business config create "$remoteName" onedrive
systemctl --user enable onedrive-mount.service
systemctl --user start onedrive-mount.service &&
  rm "$HOME/Desktop/setup-onedrive-mount.desktop"
```

Listing 17: `master:/usr/local/bin/setup-onedrive-mount.sh`

`{deployed,master,testing}:/usr/local/sbin/request-user-creation.sh`

Federführender Autor Olivier Warin

Beschreibung Erstellt eine Anfrage zur Erstellung eines Benutzerkontos.

Verwendung `request-user-creation.sh <username>`

Das gewünschte Passwort kann entweder interaktiv eingegeben werden, oder es kann per Pipe in das Skript eingegeben werden.

Programmiersprache bash

Quellcode

```
#!/bin/bash

if [ $# -eq 0 ]; then
  echo "Usage: $0 <username>"
  echo "You can also pipe the desired password into the command"
  exit 1
fi

publicKey="/root/.ssh/linuxcloud-public.pem"
requestDir="/var/lib/extrausers/create-user-requests"
userName="$1"
timeStamp=$(date +%s)
requestFile=$(mktemp "${timeStamp}XXXXXX")
read -rsp "Passwort for user $userName?" password
printf "$userName\t$password" | openssl pkeyutl -encrypt -inkey "$publicKey" \
  -pubin -out "$requestFile"
mv "$requestFile" "$requestDir"
```

Listing 18: `{deployed,master,testing}:/usr/local/sbin/request-user-creation.sh`

{deployed,master,testing}:/usr/local/sbin/request-user-deletion.sh

Federführender Autor Olivier Warin

Beschreibung Erstellt eine Anfrage zur Löschung eines Benutzerkontos.

Verwendung `request-user-deletion.sh <username>`

Programmiersprache POSIX Shell-Skript

Quellcode

```
#!/bin/sh

userName="$1"
tempDir=$(mktemp -d)
tempRequestFile="$tempDir/$userName"
requestDir="/var/lib/extrausers/delete-user-requests"
requestFile="$requestDir/$userName"

userID=$(id -u "$userName") || (echo "User $userName does not exist." && exit 1)
pkill -9 -U "$userID"
echo "$userName" >"$tempRequestFile"
chown "$userName" "$tempRequestFile"
mv "$tempRequestFile" "$requestFile"
rm -r "$tempDir"
exit 0
```

Listing 19: {deployed,master,testing}:/usr/local/sbin/request-user-deletion.sh

{deployed,master,testing}:/usr/local/sbin/setup-storage.sh

Federführender Autor Alexandre Warin

Beschreibung Es werden verschiedene Verzeichnisse eingebunden:

- Das Verzeichnis `/var/guacamoleusers` wird eingebunden, damit Root Änderungen an der Datei `user-mapping.xml` vornehmen kann. Dies ist beispielsweise bei Passwortänderungen erforderlich.
- Das Verzeichnis `/var/lib/extrausers` wird eingebunden, damit der Desktop-Container die Benutzer von *manager* verwenden kann.
- Die Home-Verzeichnisse werden mit Hilfe des Skripts `update-storage.sh` eingebunden.

Verwendung Das Skript wird durch den Systemd-Dienst `setup-storage.service` gestartet. Dieser stellt zuerst sicher, dass *manager* erreichbar ist.

```
[Unit]
Description=Mounts all system sshfs-mounts

[Service]
RemainAfterExit=yes
ExecStartPre=/usr/bin/ping -c 1 manager
ExecStart=/usr/local/sbin/setup-storage.sh
Type=oneshot
Restart=on-failure
RestartSec=10s

[Install]
WantedBy=multi-user.target
```

Listing 20: {deployed,master,testing}:/etc/systemd/system/setup-storage.service

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten

- SSHFS [lib22]
- update-storage.sh (Listing 25)

Quellcode

```
#!/bin/sh

remoteHost="manager"
#Set up user-mapping.xml, making sure only root can access it:
sshfs -o StrictHostKeyChecking=no,uid=0,gid=0,allow_other,default_permissions \
  root@"$remoteHost":/var/guacamoleusers/ /var/guacamoleusers/

#Set up extrausers
sshfs -o StrictHostKeyChecking=no -o allow_other -o follow_symlinks -o default_permissions \
  -o IdentityFile=/root/.ssh/id_rsa \
  root@"$remoteHost":/etc /var/lib/extrausers/

update-storage.sh
```

Listing 21: {deployed,master,testing}:/usr/local/sbin/setup-storage.sh

{deployed,master,testing}:/usr/local/sbin/unmount-all-fuse.sh

Federführender Autor Olivier Warin

Beschreibung Das Skript hängt alle Geräte aus, die von einem Benutzer mit fuse gemountet wurden.

Verwendung Das Skript wird automatisch via `pam_exec.so` bei jedem Abmelden ausgeführt [lin23a].

```
13 session optional pam_exec.so /usr/local/sbin/unmount-all-fuse.sh
```

Listing 22: Zeile aus {deployed,master,testing}:/etc/pam.d/common-session

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten `pam_exec.so`

Quellcode

```
1  #!/bin/sh
2  userID=$(id -u "$PAM_USER")
3
4  if [ "$PAM_TYPE" = "close_session" ]; then
5    if who | grep -wq "$PAM_USER"; then
6      echo "user $PAM_USER is still logged in. Aborting"
7      exit 1
8    else
9      if [ "$userID" -lt 1000 ]; then
10       echo "UID $userID of user $PAM_USER is too low. Aborting."
11       exit 1
12     else
13       for fuserMount in \
14         $(mount | grep fuse | grep "user_id=$userID" | cut -d' ' -f3); do
15         sudo -u "$PAM_USER" fusermount -uz "$fuserMount"
16       done
17       exit 0
18     fi
19   fi
20 fi
```

Listing 23: {deployed,master,testing}:/usr/local/sbin/unmount-all-fuse.sh

{deployed,master,testing}:/usr/local/sbin/update-storage.sh

Federführender Autor Alexandre Warin

Beschreibung Bindet alle aktiven Home-Verzeichnisse ein.

Verwendung Das Skript wird gestartet, sobald der Systemd-Dienst `watch-for-new-homes.service` neue Home-Verzeichnisse erkennt, die einzubinden sind.

```
[Unit]
Description=Makes sure all necessary Home Images are mounted
After=setup-storage.service

[Service]
ExecStartPre=/usr/bin/ping -c 1 manager
ExecStart=/bin/sh -c 'echo /var/lib/extrausers/homelist | entr -n -p
↪  /usr/local/sbin/update-storage.sh'
Restart=always
RestartSec=30

[Install]
WantedBy=multi-user.target
```

Listing 24: {deployed,master,testing}:/etc/systemd/system/watch-for-new-homes.service

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten

- SSHFS [lib22]
- entr (für den Systemd-Dienst) [era23]

Quellcode

```
#!/bin/sh
remoteHost="manager"
decryptedHomesDir="/decryptedhomes"
( mount | grep "decryptedhomes/" | cut -f1 -d" " | cut -f4 -d"/"
cat /var/lib/extrausers/homelist ) | sort | uniq -u | while read image; do
  (ls -dq "$decryptedHomesDir/$image" >/dev/null 2>/dev/null) ||
    mkdir -p "$decryptedHomesDir/$image"
  mount | grep "decryptedhomes/$image" >/dev/null 2>/dev/null ||
    nohup sshfs -o StrictHostKeyChecking=no \
      -o allow_other -o default_permissions \
      -o IdentityFile=/root/.ssh/id_rsa \
      root@"$remoteHost":"$decryptedHomesDir/$image" \
      "$decryptedHomesDir/$image" > /dev/null
done
```

Listing 25: {deployed,master,testing}:/usr/local/sbin/update-storage.sh

`{deployed,master,testing}:/usr/local/bin/welcome.sh`

Federführender Autor Olivier Warin

Beschreibung Begrüsst einen neuen Benutzer, zeigt einen Haftungsausschluss an und fordert den Benutzer auf, sein Passwort zu ändern. Dieses Skript ist standardmässig im Autostart und wird von dort entfernt, sobald das Passwort erfolgreich geändert wurde.

Verwendung `welcome.sh`

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten

- kdialog [Com23b]
- `change-password.sh` (Listing 14)

Quellcode

```
#!/bin/sh
title="Willkommen"
disclaimerText="/var/local/disclaimer.txt"

kdialog --title="$title" --textbox "$disclaimerText" 400 400
kdialog --title-"$title" --msgbox="
Herzlich Willkommen auf dem linuxcloud Desktop!

Bitte ändern Sie Ihr Passwort. Sie werden dazu aufgefordert, wenn Sie
dieses Fenster schlieen.
"

change-password.sh && rm "$HOME/.config/autostart/welcome.sh.desktop"
```

Listing 26: `{deployed,master,testing}:/usr/local/bin/welcome.sh`

`deploy.sh`

Federführender Autor Olivier Warin

Beschreibung Linkt alle für den entsprechenden Host relevanten Dateien aus dem Repository [WW23b] an die entsprechende Stelle.

Verwendung `./deploy.sh <host>`

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten GNU Stow [Fre20]

Quellcode

```
#!/bin/sh
sourceDir=${1%/}
ls "$sourceDir" >/dev/null 2>/dev/null || exit 1
stow --target=/ --no-folding --dir="$sourceDir" --adopt -S .
ldconfig
```

Listing 27: `deploy.sh`

`euler:/usr/local/sbin/fetch-reset-requests.sh`

Federführender Autor Alexandre Warin

Beschreibung Ruft von *webapp* nicht bearbeitete Anfragen zum Zurücksetzen der Desktop-Einstellungen ab und führt diese aus.

Verwendung `fetch-reset-requests.sh`

Das Skript wird durch den Systemd-Dienst `fetch-reset-requests.service` gestartet.

```
[Unit]
Description=Fetch User Reset Requests from WebApp

[Service]
Type=simple
ExecStart=/usr/local/sbin/fetch-reset-requests.sh

[Install]
WantedBy=multi-user.target
```

Listing 28: `euler:/etc/systemd/system/fetch-reset-requests.service`

Dieser Dienst wird durch den Systemd-Timer `fetch-reset-requests.timer` regelmässig ausgeführt.

```
[Unit]
Description=Timer to fetch user reset requests

[Timer]
OnBootSec=3min
OnUnitActiveSec=45s

[Install]
WantedBy=timers.target
```

Listing 29: `euler:/etc/systemd/system/fetch-reset-requests.timer`

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten `process-reset-requests.sh`

Quellcode

```
#!/bin/sh

requestDir="/etc/reset-user-requests"
finishedRequests="$requestDir/.meta/finished-requests"
webAppURL="https://webapp-gymms.sbl.ch/linuxcloud/reset-requests"
tempDir=$(mktemp -d)

(
  curl --insecure "$webAppURL/list"
  cat "$finishedRequests"
) | sort |
  uniq -u | xargs -n 1 -I {} -P 5 \
  curl --insecure "$webAppURL/{}" -o "$tempDir/{}"
cp "$tempDir"/* "$requestDir" &&
rm -r "$tempDir"
process-reset-requests.sh
```

Listing 30: `euler:/usr/local/sbin/fetch-reset-requests.sh`

`euler:/usr/local/sbin/process-reset-requests.sh`

Federführender Autor Alexandre Warin

Beschreibung Arbeitet alle Anfragen zum Zurücksetzen der Desktop-Einstellungen in `/etc/reset-user-requests/` ab.

Verwendung Das Skript wird in `fetch-reset-requests.sh` aufgerufen.

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten `reset-user.sh`

Quellcode

```
#!/bin/sh

requestDir="/etc/reset-user-requests"
finishedRequests="$requestDir/.meta/finished-requests"

if [ -z "$(ls $requestDir)" ]; then
  echo "Keine Anfragen gefunden."
  exit 0
fi

```

```
for requestFile in "$requestDir"/*; do
  userName=$(cat "$requestFile")
  userID=$(lxc exec manager -- id -u "$userName")
  if [ "$userID" -lt 1000 ]; then
    echo "UID $userID is too low. Not deleting $userName." >&2
  else
          reset-user.sh "$userName"
  fi
  basename "$requestFile" >>"$finishedRequests"
  rm "$requestFile"
done
```

Listing 31: `euler:/usr/local/sbin/process-reset-requests.sh`

`euler:/usr/local/sbin/reset-user.sh`

Federführender Autor Alexandre Warin

Beschreibung Setzt das Verzeichnis `$HOME/.config` auf die Standardwerte aus `/etc/skel/.config` zurück und beendet alle Prozesse des entsprechenden Nutzers.

Verwendung `reset-user.sh <username>`

Programmiersprache POSIX Shell-Skript

Quellcode

```
#!/bin/sh

userName="$1"
userID=$(lxc exec manager -- id -u "$userName") || exit 1
homeDir="/decryptedhomes/$userID/$userName"

for container in deployed master testing; do
        lxc exec "$container" -- pkill -9 -U "$userID"
done
lxc exec manager -- rsync -aL --del /etc/skel/.config/ "$homeDir/.config/"

```

Listing 32: `euler:/usr/local/sbin/reset-user.sh`

`euler:/usr/local/bin/generate-container.sh`

Federführender Autor Alexandre Warin

Beschreibung Erzeugt einen der Container *deployed*, *master* oder *testing* neu.

- *deployed* wird erzeugt, wobei *master* geklont wird.
- *master* wird erzeugt, wobei *testing* geklont wird.
- *testing* wird erzeugt, in dem *master* geklont wird.

Fünf Minuten vor der Ausführung und eine Minute vor der Ausführung wird eine Nachricht zur Vorwarnung an alle angemeldeten Benutzer des Zielcontainers gesendet.

Verwendung `generate-container.sh <container>`

Dieses Skript wird durch den Systemd-Dienst `reset-deployed.service` so ausgeführt, dass deployed neu von master geklont wird.

```
[Unit]
Description=Clones master to deployed

[Service]
Type=oneshot
ExecStart=/usr/bin/sudo -u lxadmin /usr/local/bin/generate-container.sh deployed
```

Listing 33: `euler:/etc/systemd/system/reset-deployed.service`

Dieser Dienst wiederum wird jeden Tag um 03:00 durch den Systemd-Timer `reset-deployed.timer` gestartet.

```
[Unit]
Description=Resets deployed every night

[Timer]
OnCalendar=*-*-* 03:00:00

[Install]
WantedBy=timers.target
```

Listing 34: `euler:/etc/systemd/system/reset-deployed.timer`

Programmiersprache POSIX Shell-Skript

Quellcode

```
#!/bin/sh

targetContainer="$1"
```

```
sourceContainer="master"
case $targetContainer in
master)
  sourceContainer="testing"
  rdp=3389
  ;;
testing)
  rdp=3388
  ;;
deployed)
  rdp=3387
  ;;
*)
  exit 1
  ;;
esac

lxc info "$sourceContainer" >/dev/null 2>/dev/null || exit 1
lxc exec "$targetContainer" -- wall -n \
  "ACHTUNG: Dieses System wird in 5 Minuten \
(um $(date -d '+5 minutes' '+%H:%M')) neu gestartet. \
Bitte speichern Sie ihre Dokumente!"
sleep 4m
lxc exec "$targetContainer" -- wall -n \
  "ACHTUNG: Dieses System wird einer Minute \
(um $(date -d '+1 minutes' '+%H:%M')) neu gestartet. \
Bitte speichern Sie ihre Dokumente!"
sleep 1m
lxc stop "$targetContainer"
lxc delete "$targetContainer"
lxc copy "$sourceContainer" "$targetContainer"
lxc config device set "$targetContainer" \
  rdp-forward listen=tcp:0.0.0.0:$rdp connect=tcp:127.0.0.1:3389
lxc start "$targetContainer"
```

Listing 35: `euler:/usr/local/bin/generate-container.sh`

`euler:/usr/local/sbin/update-hosts.sh`

Federführender Autor Olivier Warin

Beschreibung Das Skript liest die IPv4-Adresse jedes Containers, einschliesslich des Containers *euler*, und fügt sie auf allen Containern in der Datei `/etc/hosts` hinzu oder bearbeitet den entsprechenden Eintrag.

Verwendung Der Systemd-Dienst `update-hosts.service` startet dieses Skript.

```
[Unit]
Description=Update Hosts File

```

```
[Service]
Type=oneshot
ExecStart=/usr/local/sbin/update-hosts.sh
Restart=on-failure
RestartSec=10s

[Install]
WantedBy=multi-user.target
```

Listing 36: `euler:/etc/systemd/system/update-hosts.service`

Dieser Systemd-Dienst wird durch den Systemd-Timer `update-hosts.timer` 2 Minuten nach dem Aufstarten und dann alle 30 Minuten gestartet.

```
[Unit]
Description=Update hosts File

[Timer]
OnBootSec=2min
OnUnitActiveSec=30min
Unit=update-hosts

[Install]
WantedBy=timers.target
```

Listing 37: `euler:/etc/systemd/system/update-hosts.timer`

Programmiersprache POSIX Shell-Skript

Quellcode

```
#!/bin/sh

hostsFile="/etc/hosts"
containerList=$(lxc list status=running --format csv --columns=n)
ipv4Regex='([0-9]{1,3}[\.]){3}[0-9]{1,3}'

setIP() {
  ipAddress="$1"
  host="$2"
  echo "$ipAddress" | grep -qE "$ipv4Regex" ||
    (
      echo "Host $host not ready. Got an invalid IP Adress: $ipAddress"
      return 1
    )
  grep -wq "$host" "$hostsFile" || echo "$ipAddress $host" >>"$hostsFile"
  sed -E -i "s/($ipv4Regex)[[:space:]]\b$host\b/$ipAddress $host/" \
    "$hostsFile"
  for cnt in $containerList; do
    lxc exec "$cnt" -- sh -c \
```

```
      "grep -wq $host $hostsFile || echo $ipAddress $host >> $hostsFile"
    lxc exec "$cnt" -- \
      sed -E -i "s/($ipv4Regex)[[:space:]]\b$host\b/$ipAddress $host/" \
      "$hostsFile"
  done
  return 0
}

hostName=$(hostname)
defaultNetworkDevice=$(ip route | awk '/default/ {print $5}')
localIP=$(ip addr show "$defaultNetworkDevice" |
  grep -oP '(?<=inet\s)\d+(\.\d+){3}')
setIP "$localIP" "$hostName"

exit_code=0
for container in $containerList; do
  containerIP=$(lxc list "$container" --format=csv --columns=4 | cut -d' ' -f1)
  setIP "$containerIP" "$container" || exit_code=1
done
exit "$exit_code"
```

Listing 38: `euler:/usr/local/sbin/update-hosts.sh`

`host:/usr/local/sbin/generate-encrypted-homes.sh`

Federführender Autor Alexandre Warin

Beschreibung Erstellt eine bestimmte Anzahl von Images für Home-Verzeichnisse, so dass am Ende mindestens die angegebene Anzahl vorhanden ist. Diese Images werden in den *manager* eingebunden und verschlüsselt.

Verwendung `generate-encrypted-homes.sh <total-number-of-images>`

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten

- gocryptfs [rfj23]
- OpenSSL [Aut23a]

Quellcode

```
#!/usr/bin/sh
#Erwarteter Parameter: Gesamtzahl der Userhomes die es nachher geben soll.
#(Nicht Anzahl neu zu erstellender User!)
#
encryptedHomesDir="/encryptedhomes"
decryptedHomesDir="/decryptedhomes"
keyfilesDir="/keyfiles"
keyLength=50
```

```
firstUID=1000
lastUID=$(($1 + firstUID - 1))
imageSize=2048 # in MB

for uid in $(seq $firstUID $lastUID); do
  if grep -e "$uid".img /etc/fstab >/dev/null; then
    continue
  fi
  dd if=/dev/zero of="$encryptedHomesDir/images/$uid".img bs=1M count="$imageSize"
  mkfs.ext4 "$encryptedHomesDir/images/$uid".img
  mkdir "$encryptedHomesDir/mount/$uid"
  echo "$encryptedHomesDir/images/$uid.img $encryptedHomesDir/mount/$uid" \
    ext4 loop,defaults,_netdev 0 2 >>/etc/fstab
  mount "$encryptedHomesDir/mount/$uid"
  mkdir "$encryptedHomesDir/mount/$uid/$uid"
  chmod -R 777 "$encryptedHomesDir/mount/$uid"
  lxc config device add euler "$uid" disk \
    source="$encryptedHomesDir/mount/$uid/$uid" \
    path="$encryptedHomesDir/$uid" \
    readonly=false
  lxc exec euler -- \
    lxc config device add manager "$uid" disk \
    source="$encryptedHomesDir/$uid" \
    path="$encryptedHomesDir/$uid" \
    readonly=false
  lxc exec euler -- \
    lxc exec manager -- \
    openssl rand -out "$keyfilesDir/$uid" -base64 "$keyLength"
  lxc exec euler -- \
    lxc exec manager -- \
    gocryptfs -passfile "$keyfilesDir/$uid" \
    -config "$keyfilesDir/$uid".conf \
    -init "$encryptedHomesDir/$uid"
  lxc exec euler -- \
    lxc exec manager -- \
    mkdir "$decryptedHomesDir/$uid"
  lxc exec euler -- \
    lxc exec manager -- \
    gocryptfs -passfile "$keyfilesDir/$uid" \
    -config "$keyfilesDir/$uid".conf \
    -force_owner "$uid:$uid" \
    "$encryptedHomesDir/$uid" \
    "$decryptedHomesDir/$uid"
  #entr doesn't work over sshfs!
  #We need to update the desktop mounts now!
  lxc exec euler -- \
    lxc exec manager -- sh -c \
     "ls \"$encryptedHomesDir/\" >/etc/homelist"
  for container in deployed master testing; do
    lxc exec euler -- \
      lxc exec "$container" update-storage.sh
  done
```

```
61 done
```

Listing 39: `host:/usr/local/sbin/generate-encrypted-homes.sh`

`manager:/usr/local/sbin/create-user.sh`

Federführender Autor Alexandre Warin

Beschreibung Das Skript dient dazu, neue Benutzerkonten anzulegen oder ein neues Passwort zu setzen. Wird das Skript für ein bestehendes Benutzerkonto aufgerufen, wird dessen Passwort entsprechend neu gesetzt.

Wird das Skript für einen noch nicht existierenden Benutzernamen aufgerufen, wird dieser angelegt.

Verwendung `create-user.sh <username>`

Das gewünschte Passwort kann entweder interaktiv eingegeben werden, oder es kann per Pipe in das Skript eingegeben werden.

Programmiersprache bash

Spezielle Abhängigkeiten

- gocryptfs [rfj23]
- rsync [Dav23]

Quellcode

```
#!/bin/bash

encryptedHomesDir="/encryptedhomes"
decryptedHomesDir="/decryptedhomes"
statusDir="/var/guacamoleusers/status"
passwdFile="/etc/passwd"
desktopHomesDir="/decryptedhomes"
defaultShell="/bin/bash"
keyfilesDir="/keyfiles"
keyLength=50
resetRequestsDir="/etc/reset-user-requests"

statusMsg() {
  #status messages are only shown on webapp if they end with a dot!
  timeStamp=$(date +"%d.%m.%Y, %H:%M:%S")
  echo "$timeStamp: $1" | tee "$statusFile"
  sleep 2
}

bye() {
  sleep 5
  rm "$statusFile"
  exit "$1"
}

abortUser() {
  errorMsg="$1"
  if $isNewUser; then
```

```
    userdel -r "$user"
  fi
  statusMsg "$timeStamp: $errorMsg" >&2
  bye 1
}

if [ $# -eq 0 ]; then
  echo "Usage: $0 <username>"
  echo "You can also pipe the desired password into the command"
  exit 1
fi

user="$1"
read -rsp "Passwort for user $user?" password
id "$user" && isNewUser=false || isNewUser=true
statusFile="$statusDir/$user"

if $isNewUser; then
  statusMsg "Der Account $user existiert noch nicht und wird angelegt."
  nextUID=$(
    (
      cut -d: -f3 "$passwdFile" | grep "^....$"
      seq 1000 9999
    ) |
      sort -n | uniq -u | head -1
  )
else
  statusMsg "Der Account $user existiert bereits. Das Passwort wird \
neu gesetzt."
  nextUID=$(id -u "$user")
fi

mkdir -p "$desktopHomesDir/$nextUID/"
echo \
  "$user:$password:$nextUID:$nextUID::$desktopHomesDir/$nextUID/$user:$defaultShell" |
  newusers || abortUser \
  "Fehler beim Anlegen/Abändern des Benutzers $user auf Systemebene."
userID=$(id -u "$user") ||
  abortUser "Beim Abrufen der UID von $user ist ein Fehler aufgetreten."

if [ -e "$encryptedHomesDir/$userID" ]; then
  #Move the user home into his encrypted container, making sure it's empty
  #beforehand We also delete the old keyfile, remove the old gocryptfs
  #connection and set it up again using a new keyfile. That way any old data
  #we somehow forget to delete become inaccessable.
  if $isNewUser; then
    find "${decryptedHomesDir:?}/${userID:?}/" -mindepth 1 -delete ||
      abortUser "Beim Löschen des alten entschlüsselten persönlichen \
Ordners ist ein Fehler aufgetreten."
    umount -l "$decryptedHomesDir/$userID"
    find "${encryptedHomesDir:?}/${userID:?}/" -mindepth 1 -delete ||
      abortUser "Beim Löschen des alten verschüsselten persönlichen \
```

```
Ordners ist ein Fehler aufgetreten."
    rm "${keyfilesDir:?}/${userID:?}"
    rm "${keyfilesDir:?}/${userID:?}.conf"
    (
      openssl rand -out "$keyfilesDir/$userID" -base64 "$keyLength" &&
        gocryptfs -passfile "$keyfilesDir/$userID" -config \
          "$keyfilesDir/$userID".conf -init "$encryptedHomesDir/$userID" &&
        gocryptfs -passfile "$keyfilesDir/$userID" -config \
          "$keyfilesDir/$userID".conf -force_owner \
          "$userID:$userID" "$encryptedHomesDir/$userID" \
          "$decryptedHomesDir/$userID" &&
        mkdir -p "$decryptedHomesDir/$userID/$user" &&
        rsync -aL "/etc/skel/" "$decryptedHomesDir/$userID/$user/" &&
        chown -R "$user:$user" "$decryptedHomesDir/$userID/$user" &&
        chmod 700 "$decryptedHomesDir/$userID/$user" &&
        #Now is the last possible time to let the desktop containers update
        #their quota mounts if needed. We do that by updating a shared file
        #containing all used images and let entr be triggered on the desktop side
        ls "$encryptedHomesDir/" >/etc/homelist
    ) ||
      abortUser "Beim Erzeugen des persönlichen, verschlüsselten Ordners \
ist ein Fehler aufgetreten."
    echo "$user" >"$resetRequestsDir/$user"
    sleep 45 #Wait until the reset request is done
    statusMsg "Der Account $user wurde erfolgreich erstellt und kann ab sofort \
verwendet werden."
    sleep 5
    bye 0
  else
    statusMsg "Das Passwort wurde erfolgreich zurückgesetzt."
    bye 0
  fi
else
  abortUser "Benutzer $user konnte nicht erstellt werden, weil keine \
freien Home-Images mehr vorhanden sind. D.h. der Speicher reicht nicht
aus."
fi
```

Listing 40: `manager:/usr/local/sbin/create-user.sh`

`manager:/usr/local/sbin/decrypt-storage.sh`

Federführender Autor Alexandre Warin

Beschreibung Entschlüsselt die verschlüsselten Home-Verzeichnisse und stellt sie unter `/decryptedhomes` zur Verfügung.

Verwendung `decrypt-storage.sh`

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten gocryptfs [rfj23]

Quellcode

```
#!/bin/sh

encryptedHomesDir="/encryptedhomes"
decryptedHomesDir="/decryptedhomes"
keyfilesDir="/keyfiles"

for directory in "$encryptedHomesDir"/*; do
  uid=$(basename "$directory")
  gocryptfs -passfile "$keyfilesDir/$uid" -config \
    "$keyfilesDir/$uid.conf" -force_owner "$uid:$uid" "$directory" \
    "$decryptedHomesDir/$uid"
done
```

Listing 41: `manager:/usr/local/sbin/decrypt-storage.sh`

`manager:/usr/local/sbin/delete-guacamole-user`

Federführender Autor Olivier Warin

Beschreibung Entfernt einen Benutzer aus der Datei `user-mapping.xml` und löscht somit seinen Guacamole-Nutzeraccount.

Verwendung `delete-guacamole-user <user>`

Programmiersprache C

Spezielle Abhängigkeiten libxml2 [Pro22]

Quellcode

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <libxml/parser.h>
#include <libxml/tree.h>
```

```
6
7  void deleteEntry(xmlNodePtr node,
8    const char * userName) {
9    xmlNodePtr current = node;
10   while (current != NULL) {
11     if (current -> type == XML_ELEMENT_NODE && xmlStrcmp(current -> name,
12         (const xmlChar * )
13         "authorize") == 0) {
14       xmlAttrPtr attr = current -> properties;
15       while (attr != NULL) {
16         if (xmlStrcmp(attr -> name, (const xmlChar * )
17             "username") == 0) {
18           xmlChar * attrContent =
19             xmlNodeListGetString(current -> doc, attr -> children, 1);
20           if (attrContent != NULL &&
21             xmlStrcmp(attrContent, (const xmlChar * ) userName) == 0) {
22             xmlUnlinkNode(current);
23             xmlFreeNode(current);
24             xmlFree(attrContent); // Free the attribute content
25             return;
26           }
27           xmlFree(attrContent); // Free the attribute content
28         }
29         attr = attr -> next;
30       }
31     }
32     deleteEntry(current -> children, userName);
33     current = current -> next;
34   }
35 }
36
37 int main(int argc, char * argv[]) {
38   if (argc < 2) {
39     printf("Usage: %s <username> [<usermapping-file>]\n", argv[0]);
40     return EXIT_FAILURE;
41   }
42
43   const char * userName = argv[1];
44   const char * usermappingFile;
45
46   if (argc == 3) {
47     usermappingFile = argv[2];
48   } else {
49     usermappingFile = "/var/guacamoleusers/user-mapping.xml";
50   }
51
52   xmlDocPtr doc = xmlReadFile(usermappingFile, NULL, XML_PARSE_NOBLANKS);
53   if (doc == NULL) {
54     fprintf(stderr, "Failed to parse XML file.\n");
55     return EXIT_FAILURE;
56   }
57
```

```
58    xmlNodePtr root = xmlDocGetRootElement(doc);
59    deleteEntry(root, userName);
60
61    if (xmlSaveFormatFile(usermappingFile, doc, 1) == -1) {
62      fprintf(stderr, "Failed to save XML file.\n");
63    }
64
65    xmlFreeDoc(doc);
66    xmlCleanupParser();
67
68    return EXIT_SUCCESS;
69  }
```

Listing 42: `manager:/usr/local/sbin/delete-guacamole-user`

`manager:/usr/local/sbin/fetch-user-creation-requests.sh`

Federführender Autor Olivier Warin

Beschreibung Ruft von *webapp* nicht bearbeitete Benutzeranfragen (oder Passworteinstellungen) ab.

Verwendung `fetch-user-creation-requests.sh`

Das Skript wird durch den Systemd-Dienst `fetch-user-creation-requests.service` gestartet.

```
1  [Unit]
2  Description=Fetch User Creation Requests from WebApp
3
4  [Service]
5  Type=simple
6  ExecStart=/usr/local/sbin/fetch-user-creation-requests.sh
7
8  [Install]
9  WantedBy=multi-user.target
```

Listing 43: `manager:/etc/systemd/system/fetch-user-creation-requests.service`

Dieser Dienst wird durch den Systemd-Timer `fetch-user-creation-requests.timer` regelmässig ausgeführt.

```
1  [Unit]
2  Description=Timer for Fetch User Creation Requests
3
4  [Timer]
5  OnBootSec=3min
6  OnUnitActiveSec=45s
7
8  [Install]
9  WantedBy=timers.target
```

Listing 44: `manager:/etc/systemd/system/fetch-user-creation-requests.timer`

Programmiersprache POSIX Shell-Skript

Quellcode

```
#!/bin/sh

requestDir="/etc/create-user-requests"
finishedRequests="$requestDir/.meta/finished-requests"
webAppURL="https://webapp-gymms.sbl.ch/linuxcloud/requests"
tempDir=$(mktemp -d)

(
  curl --insecure "$webAppURL/list"
  cat "$finishedRequests"
) | sort |
  uniq -u | xargs -n 1 -I {} -P 5 \
  curl --insecure "$webAppURL/{}" -o "$tempDir/{}"
cp "$tempDir"/* "$requestDir"
rm -r "$tempDir"
```

Listing 45: `manager:/usr/local/sbin/fetch-user-creation-requests.sh`

`manager:/usr/local/sbin/orphan-remover.sh`

Federführender Autor Olivier Warin

Beschreibung Wenn ein Nutzer nicht als Systemnutzer in der Datei `/etc/passwd` vorhanden ist, wird er aus der Datei `user-mapping.xml` entfernt.

Verwendung `orphan-remover.sh`

Das Skript wird vom Systemd-Dienst `orphan-remover.service` mit Hilfe von entr immer dann ausgeführt, wenn sich die Datei `/etc/passwd` ändert.

```
[Unit]
Description=Watcher for user deletion requests
After=network.target

[Service]
Type=simple
ExecStart=/bin/sh -c 'echo /etc/passwd | entr -n -p /usr/local/sbin/orphan-remover.sh'
Restart=always
RestartSec=1m

[Install]
WantedBy=multi-user.target
```

Listing 46: `manager:/etc/systemd/system/orphan-remover.service`

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten

- `delete-guacamole-user` (Listing 42)
- Für den Systemd-Dienst: entr [era23]

Quellcode

```
#!/bin/sh

usermappingFile="/var/guacamoleusers/user-mapping.xml"
guacamoleUsers=$(grep -o 'username="[^"]*"' "$usermappingFile" | cut -d '"' -f 2)

for userName in $guacamoleUsers; do
  grep -q "^$userName:" /etc/passwd ||
    delete-guacamole-user "$userName" "$usermappingFile"
done
```

Listing 47: `manager:/usr/local/sbin/orphan-remover.sh`

`manager:/usr/local/sbin/process-creation-requests.sh`

Federführender Autor Olivier Warin

Beschreibung Arbeitet alle Anfragen zur Nutzergenerierung (bzw. Passwortänderungen) in `/etc/create-user-requests/` ab.

Verwendung Das Skript wird durch den Systemd-Dienst `creation-requests-watcher.service` bei neuen Anfragen ausgelöst.

```
[Unit]
Description=Watcher for user creation requests
After=network.target

[Service]
Type=simple
RemainAfterExit=yes
ExecStart=/bin/sh -c 'while true; do inotifywait -e create /etc/create-user-requests &&
↪  process-creation-requests.sh; done'
Restart=always
RestartSec=10ms

[Install]
WantedBy=multi-user.target
```

Listing 48: `manager:/etc/systemd/system/creation-requests-watcher.service`

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten

- OpenSSL [Aut23a]
- Für den Systemd-Dienst: inotifywait aus dem Paket inotify-tools [ino23]

Quellcode

```
#!/bin/sh

statusDir="/var/guacamoleusers/status"
requestDir="/etc/create-user-requests"
privateKey="/root/.ssh/linuxcloud-private.key"
finishedRequests="$requestDir/.meta/finished-requests"

if [ -z "$(ls $requestDir)" ]; then
  echo "Keine Anfragen gefunden."
  exit 0
fi

for request in "$requestDir"/*; do
  sleep 1 #Avoid racing condition
  authTok=$(openssl pkeyutl -decrypt -inkey "$privateKey" -in "$request")
  username=$(echo "$authTok" | cut -f1)
  statusFile="$statusDir/$username"
  password=$(echo "$authTok" | cut -f2)
  timeStamp=$(date +"%d.%m.%Y, %H:%M:%S")
  echo "$timeStamp: Die Anfrage für $username wird vom Server bearbeitet." |
    tee "$statusFile"
  echo "$password" | create-user.sh "$username"
  basename "$request" >>"$finishedRequests"
  rm "$request"
done
```

Listing 49: `manager:/usr/local/sbin/process-creation-requests.sh`

`manager:/usr/local/sbin/process-deletion-requests.sh`

Federführender Autor Olivier Warin

Beschreibung Löscht alle Benutzer und ihre Home-Verzeichnisse, für die eine Löschanfrage gestellt wurde und deren UID 1000 oder höher ist.

Verwendung Das Skript wird durch den Systemd-Dienst `deletion-requests-watcher.service` bei neuen Löschanfragen ausgelöst.

```
[Unit]
Description=Watcher for user deletion requests
After=network.target

[Service]
Type=simple
ExecStartPre=/usr/local/sbin/process-deletion-requests.sh
ExecStart=/bin/sh -c 'inotifywait -e create /etc/delete-user-requests &&
↪  process-deletion-requests.sh'
Restart=always
RestartSec=10s

[Install]
WantedBy=multi-user.target
```

Listing 50: `manager:/etc/systemd/system/deletion-requests-watcher.service`

Programmiersprache POSIX Shell-Skript

Spezielle Abhängigkeiten Für den Systemd-Dienst: inotifywait aus dem Paket inotify-tools [ino23]

Quellcode

```
#!/bin/sh

requestDir="/etc/delete-user-requests"

if [ -z "$(ls $requestDir)" ]; then
  echo "Keine Anfragen gefunden."
  exit 0
fi

for requestFile in "$requestDir"/*; do
  userName=$(stat -c %U "$requestFile")
  userID=$(id -u "$userName")
  if [ "$userID" -lt 1000 ]; then
    echo "UID $userID is too low. Not deleting $userName." >&2
  else
    userdel -r "$userName"
  fi
  rm "$requestFile"
done
```

Listing 51: `manager:/usr/local/sbin/process-deletion-requests.sh`

webapp:/var/www/linuxcloud/auth/authtok.php

Federführender Autor Alexandre Warin

Beschreibung Erzeugt ein Passwort für einen Benutzer mit SBL-Login. Dieses wird zusammen mit dem Benutzernamen regelmässig mit Hilfe von `fetch-user-creation-requests.sh` (Listing 45) und anschliessend von `process-creation-requests.sh` (Listing 49) verarbeitet.

Verwendung Dieses Skript ist in die Webseite [WW23a] eingebettet.

Programmiersprache PHP

Quellcode

```php
<?php

function generateRandomPassword($length) {
   $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
   $password = '';
   $maxIndex = strlen($characters) - 1;
   for ($i = 0; $i < $length; $i++) {
     $password .= $characters[random_int(0, $maxIndex)];
   }
   return $password;
}

$domain = "linuxcloud.ch";
$requestsDir = "../requests";
$listFilePath = $requestsDir."/list";
$username = $_SERVER['PHP_AUTH_USER'];
$password = generateRandomPassword(12);
$requestFilePath = tempnam($requestsDir,time());
$requestFileName = basename($requestFilePath);
$publicKey = file_get_contents('linuxcloud-public.pem');

openssl_public_encrypt("$username\t$password", $encryptedAuthTok, $publicKey);
file_put_contents($requestFilePath, $encryptedAuthTok);
file_put_contents($listFilePath,$requestFileName."\n",FILE_APPEND);

echo "<p>Benutzername: <span id=\"username\">$username</span></p>";
echo "<p>Passwort: <span id=\"password\">$password</span></p>";
?>
```

Listing 52: `webapp:/var/www/linuxcloud/auth/authtok.php`

webapp:/var/www/linuxcloud/auth/{index.html,style.css,script.js}

Federführender Autor Olivier Warin

Verwendung Diese Webseite ist unter [WW23a] verfügbar.

Beschreibung Kleine Webseite, die die beiden PHP Skripte `authtok.php` und `status.php` einbettet, um das Generieren von Nutzerkonten und das Zurücksetzen eines Passwortes benutzerfreundlich zu gestalten. Die Statusmeldungen werden automatisch aktualisiert.

Programmiersprache HTML, CSS und JavaScript

Quellcodes

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Kontoverwaltung f&uuml;r linuxcloud.ch</title>
    <link rel="stylesheet" href="style.css">
    <script src="script.js" defer></script>
  </head>
  <body>
    <div class="container">
      <h1>Anfrage zur Kontoverwaltung auf linuxcloud.ch</h1> Die untenstehenden Login-Daten werden
      ↪  bald auf dem Server verarbeitet. <ul>
        <li> Ist dieses Konto noch nicht auf <a href=\"https://linuxcloud.ch\">linuxcloud.ch</a>
        ↪  vorhanden, wird der Server angefragt, dieses zu erstellen. </li>
        <li> Wenn dieses Konto bereits existiert, wird der Server aufgefordert, das Passwort auf
        ↪  das Untenstehende zu &auml;ndern.</li>
      </ul> Der ganze Vorgang kann mehrere Minuten dauern. <p> Schreiben Sie sich das Passwort auf
      ↪  oder kopieren Sie es! Es wird nur so lange angezeigt bis Sie die Seite verlassen!
      ↪  Sollten Sie diese Seite neu laden, so wird ein neues Passwort generiert! </p>
      <div id="authtok"></div> Beachten Sie die untenstehenden Statusmeldungen. <div
      ↪  id="status"></div>
      <ul id=statusList></ul>
    </div>
  </body>
</html>
```

Listing 53: `webapp:/var/www/linuxcloud/auth/index.html`

```
body {
    font-family: Arial, sans-serif;
    background-color: #8888;
    margin: 0;
    padding: 0;
}

.container {
    max-width: 600px;
    margin: 0 auto;
    padding: 20px;
    background-color: #eeee;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
```

```
}

h1 {
    font-size: 24px;
    color: #333;
}

#username, #password {
    font-size: 32px;
    font-weight: bold;
    color: #007bff;
    margin-bottom: 20px;
}

#status {
    border-top: 1px solid #ccc;
    margin-top: 20px;
    padding-top: 10px;
}
```

Listing 54: webapp:/var/www/linuxcloud/auth/style.css

```
const statusMessages = [];

function fetchStatus() {
  fetch('status.php')
    .then(response => response.text())
    .then(data => {
      data=data.trim();
      if (data.endsWith(".") && statusMessages[statusMessages.length - 1] !==
        data) {
        statusMessages.push(data);
        updateStatusList();
      }
    })
    .catch(error => {
      console.error('Error fetching status:', error);
    });
  setTimeout(fetchStatus, 1000);
}

function updateStatusList() {
  const statusList = document.getElementById('statusList');
  statusList.innerHTML = '';
  for (const message of statusMessages) {
    const listItem = document.createElement('li');
    listItem.textContent = message;
    statusList.appendChild(listItem);
  }
}

```

```
function updateAuthTok() {
  const infoOutput = document.getElementById('authtok');
  fetch('authtok.php')
    .then(response => response.text())
    .then(data => {
      infoOutput.innerHTML = data;
    });
  infoOutput.classList.add("loaded");
}

function initStatus() {
  const options = {
    year: 'numeric',
    month: '2-digit',
    day: '2-digit',
    hour: '2-digit',
    minute: '2-digit',
    second: '2-digit',
    hour12: false,
  };
  const locale = 'de-CH';
  const timeStamp = new Date()
    .toLocaleString(locale, options);
  statusMessages.push(timeStamp + ": Die Anfrage wird an den Server gesendet.");
}
initStatus();
updateStatusList();
updateAuthTok();
fetchStatus();
```

Listing 55: `webapp:/var/www/linuxcloud/auth/script.js`

`webapp:/var/www/linuxcloud/auth/reset.php`

Federführender Autor Olivier Warin

Beschreibung Dieses PHP Skript generiert eine Anfrage zum Zurücksetzen der Desktop-Einstellungen eines Nutzers.

Programmiersprache PHP

Quellcode

```
<?php

$domain = "linuxcloud.ch";
$requestsDir = "../reset-requests";
$listFilePath = $requestsDir."/list";
$username = $_SERVER['PHP_AUTH_USER'];

```

```
$requestFilePath = tempnam($requestsDir,time());
$requestFileName = basename($requestFilePath);


file_put_contents($requestFilePath, $username);
file_put_contents($listFilePath,$requestFileName."\n",FILE_APPEND);

echo "Anfrage gesendet! Ihre Desktop-Einstellungen werden innerhalb einer";
echo "Minute zurückgesetzt. Falls Sie schon eingeloggt sind, werden Sie die";
echo "Verbindung zum Desktop kurzzeitig verlieren.";
echo "Anschliessend können Sie sich wieder neu anmelden. Ihr Desktop";
echo "sollte dann wieder normal funktionieren.";
?>
```

Listing 56: `webapp:/var/www/linuxcloud/auth/reset.php`

`webapp:/var/www/linuxcloud/auth/status.php`

Federführender Autor Alexandre Warin

Beschreibung Dieses PHP Skript holt Statusmeldungen über die Erzeugung von Nutzerkonten oder Zurücksetzungen von Passwörtern hab. Dieses Skript ist in die Webseite [WW23a] eingebunden.

Programmiersprache PHP

Quellcode

```
<?php

$username = $_SERVER['PHP_AUTH_USER'];
$url = "https://status.linuxcloud.ch/status/".$username;

/* Ignoring SSL Certificate */
$context = stream_context_create([
    'ssl' => [
        'verify_peer' => false,
        'verify_peer_name' => false,
    ],
]);

echo file_get_contents($url,false,$context);
?>
```

Listing 57: `webapp:/var/www/linuxcloud/auth/status.php`