OTTO VON GUERICKE
UNIVERSITÄT
MAGDEBURG

INF

FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Institut für Technische und Betriebliche Informationssysteme

Vergleichsbasierte Versionserkennung zur Verbesserung der Schwachstellendetektion bei webbasierten Content-Management-Systemen am Beispiel von TYPO3

Bachelorarbeit

Autor:

Jann-Marten Kias

Matrikelnummer: 218889

Betreuer:

M. Sc. Oliver Keil
Dr.-Ing. Benjamin Behrendt
Prof. Dr.-Ing. Jana Dittmann

Magdeburg, 17. Februar 2022

Inhaltsverzeichnis

Abbildungsverzeichnis

Pseudocodeverzeichnis

Abkürzungsverzeichnis

BSI Bundesamt für Sicherheit in der Informationstechnik

CMS Content-Management-System

CVE Common Vulnerabilities and Exposure

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

URL Uniform Resource Locator

HTML Hypertext Markup Language

CSS Cascading Style Sheet

VCS Version Control System

LTS Long Term Support

ELTS Extended Long Term Support

API Application Programming Interface

Abstract

Das Ziel dieser Arbeit ist die Versionserkennung von Content-Management-Systemen und die mit der Version einhergehende Erkennung von Sicherheitslücken anhand von öffentlich zugänglichen Ressourcen. Um dieses Ziel zu erreichen, muss die Version nahezu exakt bestimmt werden. Die Erkennung einer Version lässt sich dabei am besten durch Fingerprinting umsetzen. Für das Erreichen dieses Ziels werden Versionskontrollsysteme als Grundlage genutzt. Basierend darauf werden zwei Forschungsfragen gestellt. Die Erste befasst sich mit den Versionskontrollsystemen und den aus ihnen zu gewinnenden versionsspezifischen Charakteristiken sowie den Limitierungen. Die zweite Forschungsfrage baut auf die Erste auf und handelt von der Erkennung der exakten Version anhand der ermittelten Versionscharakteristiken. Um die Forschungsragen zu beantworten wurden zwei Konzepte geschaffen und in einer prototypischen Implementierung für TYPO3 umgesetzt. Zusätzlich wird ein Konzept zur Bereitstellung der gewonnenen Informationen vorgestellt. Die prototypische Implementierung wird sowohl an einer Whitebox auf ihre Funktionen und an echten Webseiten getestet. Die Evaluation zeigt unter anderem, dass bei den 849 analysierten Webseiten 453 mindestens eine Sicherheitslücke besitzen. Anhand der beispielhaften Implementierung und der anschließenden Evaluation können beide Forschungsfragen beantwortet werden. Die Arbeit zeigt somit, dass es möglich ist, Versionscharakteristika aus einem Versionskontrollsystem zu ermitteln und diese anschließend für die Erkennung einer Version von einer Content-Management-System Instanz zu verwenden. Obwohl bei der Erkennung keine privilegierte Schnittstellen und nur öffentlich zugängliche Ressourcen verwendet werden, ist es möglich die exakte Version zu bestimmen.

1 Einleitung

Sicherheitslücken in Softwaresystemen können fatale Folgen für die Betreiber der Systeme haben, vor allem wenn diese aus dem Internet erreichbar sind. Ein Beispiel hierfür ist die im Dezember 2021 erkannte Schwachstelle der Softwarekomponente Log4j, die unter anderem viele Firmen betraf und vom Bundesamt für Sicherheit in der Informationstechnik (BSI) als kritisch eingestuft wurde [1, 2]. Angreifer suchen aktiv nach derartigen Sicherheitslücken, um sie anschließend ausnutzen zu können. Alleine in Deutschland wurden im Jahr 2020 rund 108.000 Fälle von Cyberkriminalität von der Polizei erfasst [3]. Zudem wurden 2021 Kosten von 223 Milliarden Euro in Deutschland durch Cyberkriminalität verursacht [4]. Um mit möglichst wenig Aufwand möglichst viele Systeme angreifen zu können, sind im Internet häufig genutzte Softwaresysteme für Angreifer besonders interessant.

Content-Management-Systeme (CMS) werden zur einfachen Erstellung und Pflege von Webseiten genutzt. Im Januar 2022 waren unter den Top-10-Millionen Webseiten nach dem Alexa-Ranking rund 70% mithilfe von CMS erstellt [5]. Zudem sind viele CMS, darunter auch WordPress, das mit Abstand am weitesten verbreitet ist, Open Source und auch für den kommerziellen Nutzen kostenlos, wodurch jeder Nutzer, egal ob Unternehmen oder Privatperson, diese nutzen kann. Viele Hosting-Anbieter ermöglichen die Installation eines CMS mithilfe von One-Click-Lösungen, sodass das Betreiben einer eigenen CMS-Instanz auch für technisch unerfahrene Webseitenbetreiber einfach möglich ist [6, 7, 8]. Dadurch, und durch ihre weite Verbreitung, ist die Sicherheit von CMS besonders kritisch.

Ein häufig auftretendes Problem, welches Angriffe ermöglichen kann, sind veraltete Versionen mit öffentlich bekannten Schwachstellen [9, 10, 11]. Einige Hersteller bieten Tools an, mit denen installierte Instanzen einer Software auf potentielle Sicherheitsprobleme, wie zum Beispiel veraltete Versionen, überprüft werden können. Beispiele hierfür sind „Nextcloud Security Scan“ für die Open Source Software Nextcloud [12] und „WPScan“ für WordPress [13]. Diese Tools unterstützen jedoch nur ihre jeweils dazugehörige Technologie und haben meist mehr Features als die reine Versionserkennung, wodurch sie für nicht technisch versierte Benutzer unübersichtlich werden können.

Aus diesem Grund habe ich im Rahmen des Bachelorpraktikums bei der MÜNSMEDIA GmbH ein Konzept für ein anpassbares, einfach zu bedienendes und für jeden zugängliches Tool, welches Webseitenbetreiber potenziell auf diese Problematik hinweist, entwickelt. Zudem wurde als Teil dieser Arbeit eine prototypische Implementierung dieses Tools angefertigt und evaluiert. Dabei werden Technologien anbieterunabhängig unterstützt, und können mit wenig Aufwand erweitert und selbständig auf die Erkennung neu erscheinender Versionen angepasst werden. Da Software häufig in Versionsverwaltungssystemen organisiert ist, bietet es sich an diese als Ausgangspunkt für das automatisierte Erkennen von Versionen zu nutzen, woraus sich die erste Forschungsfrage ergibt:

Forschungsfrage 1 (F1): Welche Informationen können aus einem Versionsverwaltungssystem extrahiert werden, um automatisiert die exakte Version eines installierten CMS zu bestimmen, und welche Limitierungen existieren dabei?

Für das prototypische Tool wird ein Algorithmus benötigt, der mithilfe der vorher gewonnenen Informationen die Version eines CMS erkennen kann. Dabei ist die Bestimmung der Version eines CMS im Vergleich zu anderen Webapplikationen herausfordernder, da der wesentliche Zweck eines CMS das Anzeigen benutzergenerierter Inhalte ist. Da es unter dem Aspekt der Systemsicherheit unangebracht wäre, von dem jeweiligen Webseitenbetreiber Zugang zu privilegierten Schnittstellen, wie beispielsweise die Daten zum Einloggen in den Administrationsbereich der Webseite, auch Backend genannt, einzufordern, stellt sich aufbauend auf **F1** die folgende zweite Forschungsfrage:

Forschungsfrage 2 (F2): Kann die Version eines CMS bis auf das Patch-Level genau mit den Informationen aus **F1** bestimmt werden, ohne Zugang auf privilegierte Schnittstellen (Administrationsoberflächen, Terminalverbindung, etc.) zu erhalten?

Da der im Rahmen von **F2** entwickelte Prototyp für jeden zugänglich sein und eine einfache Nutzung gewährleistet werden soll, ist zudem die Bereitstellung des Tools ein weiteres Thema, das diese Arbeit behandelt.

1.1 Motivation

Die Information des genutzten Softwaresystems im Zusammenhang mit dessen Versionsnummer ist für einen Angreifer eine wichtige Information. Mit diesen Informationen können in öffentlichen Schwachstellen-Datenbanken bereits bekannten Sicherheitslücken gesucht werden und anschließend diese auf das Zielsystem angewendet werden. Der erste Schritt eines Angreifers ist meist das Ausspähen des späteren Zielsystems [14]. Die Sicherheitslücken werden unter anderem in „Common Vulnerabilities and Exposures“ (CVE) Einträgen festgehalten und sind beispielsweise unter https://cve.mitre.org aufgelistet. Die Nutzung von veralteter Software, in der die entsprechenden Sicherheitslücken noch nicht geschossen sind, ist somit ein großes Risiko, welches dem Nutzer nicht zwangsläufig bewusst ist. Alte Versionen können beispielsweise Schwachstellen enthalten, die es Angreifern erlauben Inhalte auf der Seite zu verändern oder die komplette Kontrolle über das ganze System zu erlangen.

Ein Beispiel dafür, dass bekannte Sicherheitslücken gezielt durch Angreifer gesucht und ausgenutzt werden können, ist eine Sicherheitslücke von Drupal 7, die im Oktober 2014 entdeckt wurde (CVE-2014-3704) [15]. Angreifer nutzten diese Schwachstelle so schnell aus, dass das Drupal-Team warnte, dass alle Drupal Installationen, die nicht innerhalb von sechs Stunden nach Bekanntgabe der Schwachstelle das Update aufgespielt hatten, eventuell kompromittiert wurden [16]. Aber auch andere CMS sind von schweren Sicherheitslücken nicht befreit. So sind in der Vergangenheit von WordPress, dem am häufigsten betriebenen CMS, das in über 40% der laut Alexa-Ranking Top-10-Millionen Webseiten im Januar 2022 verwendet wird [5], ebenfalls mehrere Schwachstellen aufgetreten [17]. Im Februar 2017 wurde in WordPress beispielsweise eine kritische Schwachstelle entdeckt, die es Angreifern ermöglichte Inhalte von Blog-Einträgen zu verändern [18]. Ein weiteres Beispiel ist eine Sicherheitslücke in TYPO3, die es ermöglichte von Version 11.2.0 bis 11.4.0 einen Administrator-Account anzulegen, wodurch das CMS übernommen werden konnte [19]. Die dargelegten Beispiele zeigen, wie wichtig regelmäßige Updates der CMS-Software sind und was es für Folgen haben kann, wenn alte Versionen weiter betrieben werden.

Um Webseitbetreiber für die Gefahren, die aus dem Betrieb veralteter CMS entstehen, zu sensibilisieren, wird in dieser Arbeit eine beispielhafte Implementierung eines Versionserkennungs-Tools für TYPO3 geschaffen. TYPO3 ist ein Open Source CMS, das sich als Beispiel optimal eignet. Es richtet sich an mittelständische Unternehmen und wurde im Januar 2022 von über eine Million Webseiten benutzt [20]. Der große Funktionsumfang von TYPO3 ermöglicht den Einsatz von TYPO3 für kleine, große sowie mehrsprachige Webseiten. Dabei existieren mehrere von den TYPO3-Entwicklern parallel gepflegte Major-Versionen, wodurch der Überblick, vor allem für Laien, schnell verloren gehen kann. Nach der Kenntnis des Autoren gibt es keine ausreichend präzise Implementierung eines Versionserkennungstools für TYPO3 zum Zeitpunkt der Arbeit.

1.2 Struktur der Arbeit

Zunächst werden im Abschnitt 2 die Grundlagen erklärt. Diese werden für die darauf folgenden Abschnitte als Vorwissen benötigt und erläutern die grundlegenden Konzepte von Webanwendungen und CMS am Beispiel von TYPO3. Im weiteren Verlauf werden vorangegangene Arbeiten in Abschnitt 3 beleuchtet und welche Ziele, Ansätze und Erfolge diese zu verzeichnen haben. Durch dieses Vorwissen wird dann im darauf folgenden Abschnitt 4 das Konzept für die automatische Erstellung von Versionscharakteristiken und das Nutzen dieser für eine Versionserkennung dargelegt. Zusätzlich wird diskutiert, wie die gewonnenen Informationen am besten bereitgestellt werden können, ohne die Sicherheit von Webseiten zu gefährden. Anschließend wird in Abschnitt 5 die Implementierung besprochen. Diese beinhaltet nicht nur den Algorithmus zur Versionserkennung, sondern auch die Implementierung eines Git Repository Crawlers für TYPO3 und die Umsetzung eines CVE Parsers. Daraufhin wird die implementierte Software in Abschnitt 6 dargelegt und ausgewertet. Des Weiteren wird in Abschnitt 7 die Arbeit zusammengefasst und die Forschungsfragen beantwortet. Zum Schluss gibt es einen Ausblick für mögliche weitergehende Arbeiten in Abschnitt 8.

2 Theoretische Grundlagen

In diesem Abschnitt werden die Grundlagen erklärt, die für die späteren Vorgehensweisen benötigt werden. Zunächst wird der Ablauf eines Aufrufs einer Webseite in Abschnitt 2.1 beschrieben. Zudem wird das Übertragungsprotokoll HTTP kurz angeschnitten, sowie die Netzwerkkomponente *Reverse Proxy*. Darauf folgt in Abschnitt 2.2 die Erklärung, was eine Webapplikation ist und aus welchen Bestandteilen diese typischerweise besteht. In Abschnitt 2.4 wird die Funktionsweise von CMS am Beispiel von TYPO3 aufgezeigt. Dabei werden auch einige Besonderheiten von TYPO3 beleuchtet. Im Anschluss davon wird der Aufbau von Versionen und Organisierung von Versionen in Versionsverwaltungssystemen erörtert. Zum Schluss werden im Abschnitt 2.5 verschiedene Ansätze für Software-Fingerprinting dargelegt und diskutiert.

2.1 Netztwerkgrundlagen

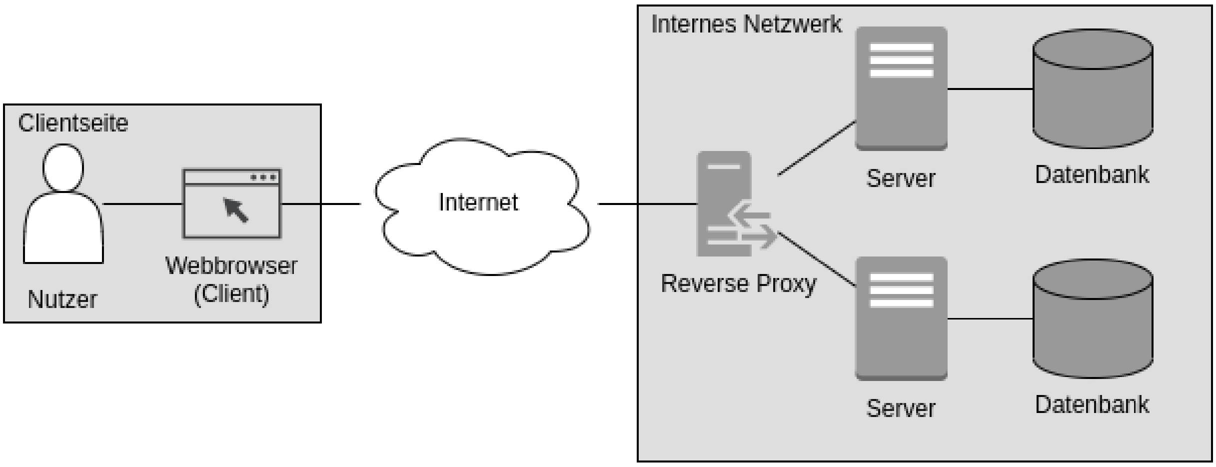


Abb. 1: Vereinfachte Darstellung einer möglichen Netzwerkkonfiguration

Bei einem typischen Aufruf einer Webseite gibt es drei verschiedene Teilbereiche, die bedeutsam für den Weg der Anfrage und der Antwort der Webseite sind (Abb. 1) [21]. Dabei kann jeder einzelne Teilbereich die Antwort beeinflussen und somit das Ergebnis der Anfrage verändern. Um den Umfang und die Gründe dieser Veränderungen zu verstehen, werden im folgenden Abschnitt kurz diese drei Bereiche beleuchtet. Dabei werden einige Punkte des Ablaufs vereinfacht, da diese standardisiert sind und sich nicht auf die Kommunikation auswirken. Anschließend wird das Protokoll für diese Kommunikation erklärt.

2.1.1 Abrufen von Webinhalten

Der erste Bestandteil ist die Clientseite. Dazu gehören zum einen ein Nutzer und ein Webbrowser. Der Nutzer kann ein Mensch, aber auch ein automatisiert handelndes Programm sein. Der Nutzer kann dabei auf diverse Webbrowser zurückgreifen, deren Unterschiede in diesem Fall vernachlässigt werden können. Der Grund dafür ist, dass nur der Aufruf der Seite und der damit einhergehenden Datentransfer in diesem Fall wichtig ist. Der Transfer der Daten kann jedoch durch Browser-Plugins verändert werden. Diese können zum Beispiel das Ausführen von JavaScript-Code der Webseite unterbinden, was zu fehlenden Funktionen bis hin zur nicht aufrufbaren Webseite führt [22].

Das Internet ist dabei der zweite große Bestandteil. Dieses beinhaltet dabei den kompletten Weg vom Client bis zum gewünschten Webserver. Als Erstes wird der Domain-Name der vom Benutzer aufgerufenen URL durch den Internetprovider auf eine IP-Adresse aufgelöst. Sofern die angefragte Domain beim Provider nicht auf einer Blacklist steht und somit blockiert wird, wird die Anfrage an einen Domain-Name-System-Server weitergeleitet, welcher die zu der Domain dazugehörige IP-Adresse heraussucht. Diese wird an den Client zurückgeschickt, worauf dieser eine erneute und direkte Anfrage an den Server mit der gewünschten Webseite stellen kann. Vor dem Server können noch Dienste, wie Load Balancer, Content Distribution Networks oder Schutzsysteme vor Distributed-Denial-of-Service Attacken geschaltet werden, welche die Anfrage gegebenenfalls direkt beantworten oder herausfiltern.

Der dritte und letzte Bestandteil ist das Netzwerk, in dem sich der Webserver befindet. Bei diesem angekommen, verwaltet in vielen Fällen ein Reverse Proxy die Anfrage. Der Client fordert beim Reverse Proxy die benötigten Dateien an, woraufhin der Proxy den dafür zuständigen Server im lokalen Netzwerk anfragt und das Ergebnis weiterleitet. Der Einsatz von Reverse Proxys hat dabei mehrere Vorteile für den Betreiber der Webseite. Ein Vorteil ist, dass im selben Netzwerk mehrere Webseiten oder andere Dienste parallel betrieben werden können und diese auch über verschiedene Domains erreichbar sind. Somit verwaltet der Reverse Proxy die verschiedenen Routen und kann auch Zugriffsrechte definieren, indem er zum Beispiel manche Routen nur für Geräte innerhalb des eigenen Netzwerks freigibt. Ein weiterer Vorteil ist, dass jeder Nutzer nicht direkt mit dem Webserver interagiert und der Server besser gesichert vor Angriffen ist.

2.1.2 HTTP/HTTPS

Die Kommunikation zwischen Client und Webserver erfolgt über das Hypertext Transfer Protocol (HTTP) [23]. Der Informationsaustausch folgt dabei immer dem gleichen Muster. Der Client stellt eine Anfrage an den Server („*Request*“) und der Server beantwortet diese („*Response*“). Sowohl Request als auch Response beinhalten einen *Header* und einen *Body*. Im Header werden Metainformationen wie beispielsweise die genaue angefragte URL oder der Statuscode der HTTP-Antwort mitgesendet. Dieser Statuscode enthält Information, ob die Anfrage erfolgreich war, oder ob ein Fehler aufgetreten ist. Somit weist zum Beispiel ein 404 Code darauf hin, dass die gewünschte Seite nicht gefunden wurde, 403 signalisiert, dass der Zugriff auf die Ressource verweigert wird. Im Body befinden sich die eigentlichen Inhalte der Request bzw. Response. Im Falle einer Request kann der Body beispielsweise eine Datei enthalten, die hochgeladen werden soll. Bei der Response ist im Regelfall der Seiteninhalt der angefragten Webseite enthalten.

Der aktuelle Standard für die Kommunikation im Internet ist das Hypertext Transfer Protocol Secure (HTTPS). Dieses erweitert HTTP, indem es eine TLS/SSL Verschlüsselung hinzufügt [24]. Neben der dadurch sichergestellten Integrität der Kommunikation kann zusätzlich durch ein Zertifikat auch die Authentizität des Servers festgestellt werden. Das Zertifikat wird dabei von einer dritten Partei ausgestellt, die die Echtheit des Servers beglaubigt. Die zusätzlichen Schutzmaßnahmen von HTTPS verbessern die Sicherheit der Datenübertragung zwischen Client und Server und erschweren somit Attacken, wie beispielsweise „Man in the Middle Attacks“.

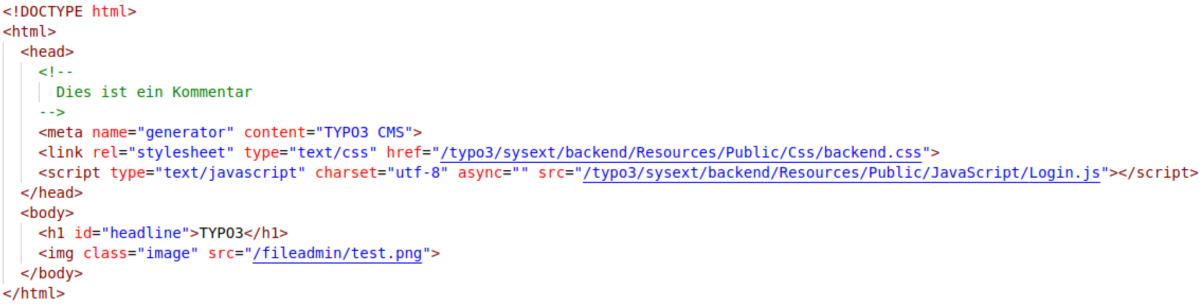
2.1.3 Reverse Proxy

Wie in 2.1.1 beschrieben ist ein Reverse Proxy ein wichtiger Bestandteil einer Netzwerkkonfiguration. Als Reverse Proxy wird ein Server bezeichnet, der in einem Netzwerk eingehende Anfragen entgegennimmt und diese an im internen Netzwerk befindliche Server weiterleitet [25]. Dabei hat er die komplette Kontrolle über die eingehende Request und der herausgehenden Response. Beispielsweise kann er Elemente der Request und Response modifizieren, Elemente hinzufügen oder löschen und deren Zielserver bestimmen. Zudem kann er den Zugriff auf Ressourcen teilweise oder komplett verweigern. So kann er zum Beispiel die Administrationsoberfläche von einem CMS für Zugriffe aus dem Internet sperren und nur lokale bzw. autorisierte Geräte auf diese Route die Verbindung erlauben. Häufig verwendete Reverse Proxys sind der Apache HTTP Server, NGINX und der Microsoft Internet Information Service (IIS). Durch die Nutzung und richtige Konfiguration eines Reverse Proxys kann die Sicherheit eines Netzwerkes verbessert werden, da kein Client mehr eine direkte Verbindung zu den Servern aufbaut.

2.2 Grundlagen von Webapplikationen

Als eine Webapplikation werden Anwendungen bezeichnet, die auf einem Server ausgeführt werden und über ein Netzwerk, meist das Internet, erreichbar sind. Die Nutzer agieren mit Webapplikationen häufig über den Webbrowser als Schnittstelle. Die Nutzung des Internets ist in den letzten Jahren rasant gestiegen. So waren es im Jahr 2015 rund 3 Milliarden Internetnutzer weltweit, während es im Jahr 2020 schon rund 4,6 Milliarden waren [26]. Durch die wachsende Nutzung des Internets werden Webseiten und Webapplikationen immer wichtiger und sind ein stetig wachsender Bereich. Damit die stetig wachsende Anzahl von Webseiten und Webapplikationen auch auf jedem Endgerät läuft, müssen die Kommunikation zwischen Server und Client und die Dateiformate festgelegten Standards entsprechen [27]. Welche Daten typischerweise beim Aufruf einer Webseite übermittelt werden, wird in den kommenden Abschnitten erläutert.

2.2.1 HTML



```html
<!DOCTYPE html>
<html>
  <head>
    <!--
      Dies ist ein Kommentar
    -->
    <meta name="generator" content="TYPO3 CMS">
    <link rel="stylesheet" type="text/css" href="/typo3/sysext/backend/Resources/Public/Css/backend.css">
    <script type="text/javascript" charset="utf-8" async="" src="/typo3/sysext/backend/Resources/Public/JavaScript/Login.js"></script>
  </head>
  <body>
    <h1 id="headline">TYPO3</h1>
    <img class="image" src="/fileadmin/test.png">
  </body>
</html>
```

Abb. 2: Beispiel einer HTML Datei

Wie in Abschnitt 2.1.2 beschrieben, werden via HTTP oder HTTPS Seiteninhalte übertragen. Für die Struktur der angeforderten Seite ist die Hypertext Markup Language verantwortlich, kurz HTML. HTML ist eine Auszeichnungssprache, bei der Inhalte in Form verschachtelter, benannter Tags definiert werden [28]. Ein Beispiel für eine HTML Datei wird in Abbildung 2 gezeigt. Der Name eines Tags bestimmt ebenfalls die Semantik des im Tag befindlichen Inhalts. Ein Beispiel dafür wäre `<h1>` TYPO3 `</h1>`, wodurch eine Überschrift ersten Levels (h1) mit dem Text „TYPO3“ erzeugt wird. Jede HTML-Datei enthält ein `<html>` Element, welches das Wurzelelement ist. Zudem gibt es immer ein `<head>` und ein `<body>` Element. Jeder Tag kann bei HTML zusätzliche Metainformationen in Form von Attributen beinhalten, die das jeweilige Objekt genauer beschreiben.

Für das Aussehen und die Funktionsweise können zusätzlich Bilder, Cascading Style Sheet- und JavaScript-Dateien durch das HTML-Dokument importiert werden. Diese können zum Beispiel mit Tags wie `<img>` für Bilder, `<style>` für CSS, oder `<script>` für JavaScript, eingebunden werden. Obwohl diese Ressourcen theoretisch in das HTML-Dokument direkt eingebettet werden können, werden sie zwecks Modularisierbarkeit und Cache-Optimierung meistens in externe Dateien ausgelagert.

2.2.2 Webassets

In den meisten Fällen referenziert das HTML-Dokument, das der Client vom Server erhalten hat, weitere externe Inhalte. Um diese vom Server abrufen zu können, werden im Browser beim Parsen des HTML-Dokuments die Referenzen auf verknüpfte Ressourcen erkannt und angefragt. Diese Anfragen geschehen iterativ und werden nach dem Laden in die HTML Struktur eingebunden. Dabei sind in der Regel diese Ressourcen öffentlich zugänglich, da sie sonst nicht vom Client angefragt werden könnten.

Ein gängiger Ressourcentyp, den fast jede Webseite einbindet, ist das Cascading Style Sheet, kurz CSS. Diese wird von HTML via eines `<style>` oder `<link>` Tags im Head der HTML-Datei eingebunden. CSS ist für die Darstellung von Markup Languages zuständig und kann verschiedenen HTML-Tags visuelle Eigenschaften zuweisen [29]. Zu diesen Eigenschaften gehören beispielsweise Schriftgröße, Farbe oder Transparenz. Um das Aussehen eines Elementes zu verändern, muss dieses zunächst mit einem Selektor auswählt werden. Diese Selektoren können HTML-Elemente anhand zugewiesener Attribute wie ID oder Class, ihrer absoluten oder relativen Position in der HTML-Baumstruktur oder ihrem aktuellen Zustand auswählen [30]. Dabei können auch mehrere Elemente gleichzeitig ausgewählt werden, wenn sie Eigenschaften teilen. Das Wort „Cascading“ beschreibt die verschiedenen Prioritätslevel innerhalb von CSS. Sollten verschiedene Regeln mit ihren Selektoren dasselbe Element als Ziel haben, werden die Eigenschaften zusammengeführt und gegebenenfalls je nach Priorität der Regel überschrieben.

Neben CSS sind JavaScript-Dateien ebenfalls sehr häufig vorkommende Ressourcen. JavaScript ist im Gegensatz zu CSS und HTML eine Skriptsprache, welche speziell für die Nutzung im Webbrowser optimiert ist [31]. Während CSS der HTML-Baumstruktur lediglich visuelle Eigenschaften anfügt, kann JavaScript unter anderem die Baumstruktur selbst modifizieren [32]. Der geladene JavaScript Code wird im Browser des Webseitenbesuchers ausgeführt. Dabei kann in Echtzeit auf die Eingaben des Nutzers reagiert werden, beispielsweise das Drücken eines Buttons. JavaScript ist somit in der Lage, komplexe Logik von großen Seiten zu modellieren. Sollte die Ausführung von JavaScript blockiert sein, ist die Webseite unter Umständen nicht mehr funktionsfähig.

Neben CSS und JavaScript gibt es noch weitere Ressourcen, die in die Seite eingebunden werden können. Zu diesen Ressourcen zählen Videos, Bilder oder Schriftarten, die auf der eigenen Seite genutzt werden sollen. Diese Komponenten sind jedoch in den meisten Fällen nicht zwingend für die Funktionsweise einer Webseite notwendig.

2.3 Software-Versionierung

In den folgenden Abschnitten geht es um die Versionierung von Software, wie eine Version aufgebaut ist, weshalb die Version wichtig ist und wie sich daraus mit wenig Aufwand Sicherheitslücken ableiten lassen. Zudem wird das Konzept eines Versionsverwaltungssystems erklärt, welches für die Beantwortung von **F1** wichtig ist.

2.3.1 Aufbau von Versionen

In der IT definiert die Versionsnummer einer Software einen spezifischen Entwicklungsstand deren Quellcodes. Wie sich die Versionsnummer zusammensetzt, ist jedem Entwickler selbst überlassen, es gibt jedoch einige häufig verwendete Muster, nach dem sich Entwickler richten können. Die meisten Ansätze der Versionsbezeichnung sind numerisch orientiert, um Vergleiche zu vereinfachen. So setzt sich beispielsweise die Versionen des Linux-Betriebssystems Ubuntu aus der numerischen Version des Monats und den letzten beiden Zahlen des Jahres zusammen, welche durch einen Punkt voneinander getrennt werden [33]. Die Versionsnummer der im Oktober 2021 veröffentlichen Ubuntu-Version lautet 21.10. Eine noch häufiger verwendete Methodik ist das sogenannte *Semantic Versioning*. Dabei besteht die Versionsnummer aus einer Sequenz von drei Nummern mit absteigender Signifikanz, die durch einen Punkt getrennt sind. Die erste Zahl bezeichnet die *Major*-, die zweite die *Minor*- und die dritte die *Patch*-Version. Die Major-Version ändert sich üblicherweise am seltensten, da mit ihr eine grundlegende Veränderung der Software einhergeht. Die Minor-Version verändert sich, wenn Funktionsweisen angepasst oder neue kleinere Funktionen hinzugefügt wurden. In neueren Patch-Versionen werden häufig Bugs behoben oder Sicherheitslücken geschlossen [34].

In einigen Fällen werden mehrere (Major-)Versionen einer Software vom Hersteller gleichzeitig gepflegt und erhalten weiterhin Updates. Dadurch kann es passieren, dass bei Semantic Versioning eine numerisch höhere Version zeitlich gesehen älter sein kann als eine numerisch niedrigere Version. Im Fall von TYPO3 ist beispielsweise die Version 11.0.0 (Veröffentlicht am 22.12.2020) [35] älter als die Version 10.4.21 (Veröffentlicht am 21.09.2021) [36]. Dies kann wiederum dazu führen, dass eine numerisch höhere Version Fehler und Bugs enthält, die in einer „niedrigeren“ Version bereits korrigiert wurden. Das liegt daran, dass mehrere Iterationen der Software vergehen können, bis ein Fehler gefunden und behoben wird. Ein Beispiel dafür ist der TYPO3-Fehler mit der Bezeichnung CVE-2021-32767, der die Versionen 9.0.0 bis 9.5.27, 10.0.0 bis 10.4.17 und 11.0.0 bis 11.3.0 betrifft [37]. Diese Aufteilung von bekannten Sicherheitslücken auf verschiedene Versionen ist für Laien schwer nachvollziehbar und kann zu Verwirrung bezüglich der Sicherheit der eingesetzten Software führen.

2.3.2 Versionsverwaltungssysteme

Um bei der Erstellung einer Software den Überblick zu behalten, mit mehreren Entwicklern zusammenzuarbeiten und gegebenenfalls auf ältere Zustände der Software zurückgreifen zu können, gibt es Versionsverwaltungssysteme (VCS, Version Control System). Der Grundgedanke eines VCS ist dabei die Änderungen von Dateien zusammen mit einem Zeitstempel (Timestamp) sowie einer manuell erstellten und verständlichen Beschreibung der Änderung iterativ abzuspeichern. Eines der am weitesten verbreiteten VCS ist Git, welches durch Plattformen wie GitHub und GitLab unter Entwickler sehr beliebt ist. Git ist dabei darauf ausgelegt, dass mehrere Entwickler gleichzeitig am Quellcode arbeiten können. Obwohl Git dezentral in Form sogenannter *Repositorys* organisiert ist, wird meistens ein *Repository* als zentraler Punkt für diese Zusammenarbeit ausgewählt, im alle Informationen gebündelt werden. Üblicherweise befindet sich dieses zentrale *Repository* auf einem für alle Entwickler erreichbaren Server. Hat ein Entwickler Teile der Software auf seinem lokalen Gerät geändert, kann er diese Änderungsinformationen in einem *Commit* bündeln. Der *Commit* enthält die Art der Änderung an jeder Datei, den Namen des Nutzers und den Timestamp. Anschließend werden diese sowohl in das lokale als auch das zentrale *Repository* übertragen, damit alle anderen Entwickler diese Änderungen ebenfalls abrufen können. In jedem *Repository* können zudem mehrere *Branches* enthalten sein, die parallel nebeneinander betrieben werden können. Ein *Branch* bezeichnet eine Folge von *Commits*, die üblicherweise einem gemeinsamen Zweck, beispielsweise der Implementierung einer spezifischen neuen Funktionalität, dienen. Das gesamte Git *Repository* ist in einer Baumstruktur organisiert, bei dem *Branches* von anderen abstammen und ggf. später wieder zusammen geführt werden. Zusätzlich kann bei jedem *Commit* ein *Tag* erstellt werden, durch den ein bestimmter Zustand (z.B. eine Versionsnummer) an den entsprechenden *Commit* annotiert wird. Da Open Source Projekte häufig durch ihre Open Source Lizenz verpflichtet sind ihren Quellcode öffentlich verfügbar zu machen, stellen diese ihre *Repositorys* meist über Plattformen wie Github oder GitLab zur Verfügung, sodass Informationen über den aktuellen Stand sowie die komplette Projekthistorie öffentlich abrufbar sind [38].

2.4 TYPO3 als CMS

Content-Management-Systeme sind Softwarelösungen zum schnellen Erstellen und effizienten Verwalten von Webseiten. Sie sind dabei meistens in zwei Bereiche unterteilt: das Frontend und das Backend. Das Frontend ist der Bereich einer Webseite, den ein Besucher der Webseite sieht. Der Backendbereich hingegen ist für die Administratoren und Redakteure der Webseite gedacht und durch Zugangsdaten wie Benutzername und Passwort geschützt. In diesem Bereich können durch eine grafische Oberfläche einzelne Seiten für die Webseite erstellt und deren Inhalt verwaltet werden. Diese Benutzeroberfläche ist oftmals intuitiv, sodass diese ohne technisches Vorwissen bedient werden können. Um das Abspeichern der Daten in einer Datenbank und die Erstellung des Quellcodes des Frontends kümmert sich das CMS selbstständig. Durch die einfache Nutzung und die umfangreiche Funktionalität sind CMS sehr weit verbreitet.

TYPO3 ist ein CMS und zudem Open Source Software. Es findet hauptsächlich in der DACH-Region Verwendung, basiert auf der Server-Skriptsprache PHP und bringt zahlreiche Funktionen von sich aus mit [40]. Zu diesen Funktionen gehören Multi Domain Verwaltung, Support für mehrsprachiges Front- und Backend, sowie ein integriertes Caching-System [41]. Mit der von TYPO3 entwickelten Sprache TypoScript kann das Aussehen und die Funktionalität der Webseite umfangreich angepasst werden. Durch diese Funktionen ist TYPO3 für kleinere und mittlere Unternehmen sehr attraktiv, kann aber auch von Privatpersonen verwendet werden. Wie jede Software hat auch TYPO3 mehrere Versionen, die Besonderheit bei TYPO3 ist jedoch, dass die Versionen teilweise parallel gepflegt und mit Updates versorgt werden, wie in Abbildung 3 gezeigt wird. Für jede Major-Version gibt es eine Long Term Support (LTS) Version, die mindestens 36 Monate Updates erhält, wobei ungefähr alle 18 Monate eine neue Major-Version erscheint [42]. In den 36 Monaten einer LTS Version werden die ersten 18 Monate noch Feature-Updates veröffentlicht, während in den letzten 18 Monate hauptsächlich Sicherheits-Updates erscheinen. Nachdem

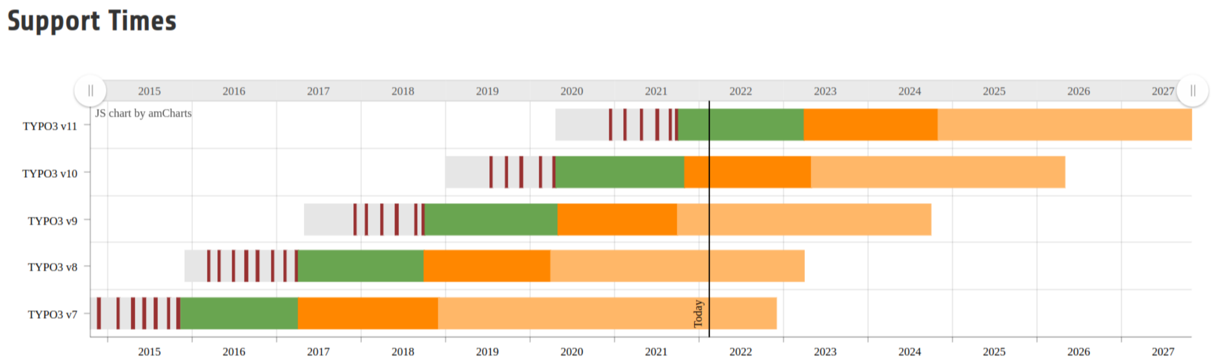


Abb. 3: Übersicht über die parallelen Entwicklungs- und Supportzeiträume verschiedener TYPO3-Versionen: Pre-Release (grau mit rot markierten Entwicklungsmeilensteinen), reguläre Unterstützung (grün), Langzeit-Unterstützung (LTS, orange) und die kostenpflichtige erweiterte Langzeitunterstützung (ELTS, beige). [39]

diese 36 Monate ausgelaufen sind, geht die LTS in eine Extended Long Term Support (ELTS) über, die 12 Monate läuft. Die ELTS Versionen sind nicht mehr Open Source und erfordern den Kauf einer Lizenz. Nach einer ELTS Phase können noch weitere folgen, so ist zum Beispiel Version 7.6 von TYPO3 im ELTS4 (Stand Dezember 2021). Durch die Verfügbarkeit von LTS- und ELTS-Versionen sowie den vergleichsweise hohen Aufwand, auf eine neue Major-Version zu wechseln, kommt es dazu, dass viele Webseiten nach dem erstmaligen Erstellen auf der zu diesem Zeitpunkt aktuellen Major-Version verharren. Dadurch ist das Spektrum von genutzten TYPO3 Versionen sehr breit gefächert. Zudem kommt es durch diese parallele Fortführung der Versionen dazu, dass dieselben Bugs und Sicherheitslücken in mehreren Major-Versionen parallel auftreten können. Diese Bugs und Sicherheitslücken werden in CVEs festgehalten und können öffentlich nachgeschlagen werden.

Eine installierte TYPO3 Instanz besteht aus diversen Ordnern, die unterschiedliche Module beinhalten [43]. Der erste und für diese Arbeit relevanteste Ordner ist der „typo3“ Ordner. In diesem befindet sich das TYPO3-Backend sowie die dazugehörige Login-Seite. Der Login ist im Webbrowser unter der URL `/typo3` erreichbar. In diesem Ordner befinden sich auch alle Systemerweiterungen, die bei der Installation automatisch angelegt werden und für die Funktionsweise von TYPO3 unverzichtbar sind. Diese liegen im Unterordner */typo3/sysext* und beinhalten neben internem PHP-Programmcode auch öffentlich zugängliche Ressourcen, die in einem Ordner namens *Public* liegen. So sind beispielsweise alle Dateien unter der Route `/typo3/sysext/backend/Resources/Public` öffentlich aufrufbar. Hierfür ist es allerdings nötig, die exakte Route sowie den Dateinamen zu kennen, da eine Auflistung aller Ressourcen innerhalb des jeweiligen Public-Ordners unterbunden wird. Es gibt noch weitere öffentlich zugängliche Ordner, die TYPO3 automatisch erstellt. Einer davon heißt *fileadmin* und beinhaltet alle vom Nutzer selbst hinterlegten Dateien. Ein anderer ist der *typo3temp* Ordner. Dieser enthält zum Beispiel vorkomprimierte Dateien, die für die Auslieferung an den Client gedacht sind. Zudem gibt es die Route *typo3conf*, in der sich die Datei `LocalConfiguration.php` befindet, die Systemkonfiguration für TYPO3 enthält. Zusätzlich befinden sich unter *typo3conf/ext* sämtliche nicht zum Hauptsystem gehörenden Erweiterungen, die der Nutzer mithilfe des Extension-Managers von TYPO3 installiert hat.

2.5 Fingerprinting zur Softwareerkennung

Fingerprinting ist ein Konzept, das schon länger bekannt ist. Eine oder mehrere Ausprägungen verschiedener Eigenschaften identifizieren dabei ein Objekt exakt, obwohl es viele Objekte dieser Art gibt [44]. Beispiele dafür lassen sich in vielen Bereichen des Lebens finden, so ist im Fall von technischen Geräten oft eine Seriennummer eine dieser Eigenschaften. Der Begriff Fingerprinting lässt sich dabei von den Fingerabdrücken eines Menschen ableiten, die eindeutig einem einzelnen Menschen zuzuordnen sind. Das Konzept Fingerprinting lässt sich ebenfalls auf diverse Aspekte der IT-Branche übertragen. So ist es für Webseiten beispielsweise möglich, den Benutzer anhand vom genutzten Endgerät oder Browser zu erkennen [45, 46]. Allerdings ist auch der umgekehrte Weg möglich, nämlich als Benutzer die von der Webseite bzw. dem Webserver genutzten Technologien zu identifizieren. In den folgenden Abschnitten werden die verschiedenen Ansätze für eine Webseitenanalyse anhand von Fingerprints erläutert und Beispiele für ihre Implementierung dargelegt.

2.5.1 Pattern-basierte Applikationserkennung

Pattern-basierte Applikationserkennungen wird hauptsächlich zur Identifizierung von vorhandenen Technologien auf einer Webseite genutzt. Jede genutzte Technologie hinterlässt eine gewisse Struktur bzw. Pattern in der Webseite, welche im Nachhinein erkannt werden kann. Diese Pattern, die in diesem Fall die Fingerprints darstellen, können im HTML Code, JavaScript Code, HTTP Headern und weiteren Elementen stecken.

Die Software Wappalyzer ist eine Implementation der Pattern-basierten Applikationserkennung [47] und nutzt reguläre Ausdrücke. Wappalyzer erkennt dabei mehrere genutzte Technologien gleichzeitig, indem der Wappalyzer für jede von ihm unterstützte Technologie eigene reguläre Ausdrücke hat und beispielsweise alle Elemente der Startseite einer Webapplikation vergleicht. Um die jeweiligen regulären Ausdrücke zu konstruieren wird spezifisches Vorwissen über die entsprechende zu erkennende Technologie benötigt. Obwohl dieser Ansatz ressourcenschonend diverse Technologien erkennen kann, ist er nicht immer für die Erkennung der spezifischen Version der Technologie geeignet, da hierfür meist einer tiefgehende Analyse durchgeführt werden muss. Dieser Ansatz wird daher eher genutzt um zu sehen, ob eine Webseite die gesuchte Technologie nutzt, um anschließend eine applikationsspezifische Erkennung zu starten.

2.5.2 Applikationsspezifische Versionserkennung

Die applikationsspezifische Versionserkennung konzentriert sich auf eine einzelne Software, um diese so genau wie möglich zu analysieren. Neben der Erkennung der Version kann dieses Vorgehen auch Sicherheitslücken entdecken. Um die Erkennung überhaupt zu ermöglichen ist ein genaues Verständnis des Aufbaus und Funktionsweise der Software notwendig. Ebenso benötigt dieses Vorgehen mehr Ressourcen als die Pattern-basierte Erkennung, da häufig mehrere Anfragen an den Server gesendet werden müssen, um das System bis ins Detail zu analysieren.

Für die applikationsspezifische Versionserkennung gibt es mehrere grundsätzliche Ansätze. Eine Möglichkeit ist das Verhalten der Software zu analysieren, indem unterschiedliche Verhaltensweisen der Software zwischen Versionen, beispielsweise bei der Behandlung von Grenzfällen, provoziert werden. Anschließend ist es möglich Rückschlüsse auf die Version zu ziehen [48]. Ein anderer Ansatz ist das Abgleichen von Webassets. Für diesen

Ansatz ist jedoch umfangreiches Detailwissen über die zu erkennende Software erforderlich, nämlich welches Webasset zu welcher Version existiert, welcher Inhalt zu welcher Version gehört und unter welcher Route das Asset erreichbar ist. Eine Implementierung von diesem Vorgehen sind WPScan für WordPress [13] und JoomScan für Joomla! [49]. Es gibt momentan noch kein Tool, dass mit vergleichsweise wenig Aufwand die Versionen der meisten CMS erkennen kann, weshalb die Beantwortung der zweiten Forschungsfrage (**F2**) für die Sicherheit von CMS relevant ist.

Das Fingerprinten von CMS ist zudem herausfordernder als bei anderen Arten von Webapplikationen, da ein CMS im Gegensatz zu einer normalen Webapplikation kein fest vordefiniertes Frontend für den Nutzer hat, sondern sehr stark modifizierbar durch den Webseitenbetreiber ist. Somit können bei einem CMS meistens keine Daten aus dem Frontend zur Versionserkennung genutzt werden. Durch diese Umstände wird das Vorwissen zu einem CMS um so wichtiger, damit man schlussendlich die vom Benutzer eingepflegten Daten von denen des CMS unterscheiden kann.

3 Vorangegangene Arbeiten

Das Konzept von Fingerprinting ist bereits länger im IT Bereich bekannt und wird aus diversen Gründen genutzt. Beispielsweise benutzen Unternehmen oder Organisationen Fingerprinting, um Statistiken über die Nutzung verschiedenster Technologien zu erstellen und Trends abzeichnen zu können. Ein Beispiel für solch ein Unternehmen ist Netcraft, die bereits seit mehreren Jahrzehnten verschiedenste Webseiten überwachen [50]. Es gibt aber auch Open-Source-Projekte, die das Überprüfen von Webseiten und Systeme möglich machen. Als Beispiel dafür können die Tools NMAP und Wappalyzer genannt werden [47, 51].

Im Paper „Detecting and defending against Web-server fingerprinting“ von Lee et al. werden diverse Möglichkeiten des Webserver-Fingerprintings erläutert [48]. Die Betrachtung beschränkt sich dabei auf das HTTP Protokoll und das Applikations-Layer. Im Rahmen der Arbeit wurde das Tool HMAP erstellt, welches durch NMAP inspiriert ist. HMAP untersucht dabei diverse Charakteristika eines Webservers, im es zum Beispiel Fehlercodes provoziert und aus diesen Rückschlüssen zieht. Somit ist es in der Lage mit hoher Genauigkeit zu bestimmen, welcher Webserver in welcher Version betrieben wird.

Nicht nur Webserver können durch Fingerprints erkannt werden, sondern auch Clients, die durch ihre genutzten Geräte Spuren hinterlassen. Im Paper „Web-based Fingerprinting Techniques“ von Bernardo und Domingos werden die Möglichkeiten dafür aufgezeigt [52]. Es wird beschrieben, wie mithilfe des Browsers, dessen Plugins, sowie dem genutzten Betriebssystem und Hardwaredaten sehr genaue Nutzerprofile erzeugen werden können. Das Sammeln von Daten über hardwarenahe Schnittstellen wird ebenfalls in der Bachelorarbeit von Lamshöft behandelt [53]. Wird zusätzlich zu den Gerätedaten das Verhalten des Clients einbezogen, kann ein Nutzer einer Webseite identifiziert werden, ohne dass dieser es merkt [52]. Am Schluss werden einige Möglichkeiten, das Nachverfolgen zu unterbinden, aufgezeigt.

Die Bachelorarbeit von Pascal Wichmann mit dem Titel „Automated Inference of Web Software Packages and Their Versions“ handelt von automatisierter Versionserkennung von Webapplikationen [54]. Das Ziel dieser Arbeit ist die Erarbeitung einer Methode um mit möglichst wenig Anfragen an einen Webserver die genutzten Webtechnologien und deren Versionen zu erfassen. Im Rahmen davon wurde das Tool VersionInferrer geschaffen, welches öffentlich zugänglich ist [55]. Mit diesem Tool wurden die ersten 500.000 Seiten von der Alexa Top 1 Million Liste auf 16 vorher definierte Technologien geprüft. Bei 19.5% dieser Seiten wurde mindestens eine der gesuchten Technologien gefunden. Bei diesen wiederum wurde in 67.9% eine eindeutige Version erkannt. In einer manuellen Überprüfung von 50 Seiten zeigte sich, dass bei 14% der Ergebnisse fehlerhaft sind.

Der Artikel von Gorke und Armknecht mit dem Titel „Reverse Fingerprinting“ verfolgt einen anderen Ansatz [56]. Hier wird beschrieben, wie ein Kunde eines Service Providers mittels Fingerprinting die exakte Version der zur Verfügung gestellten Software des Providers ermitteln könnte. Die Autoren gehen davon aus, dass der Service Provider eventuell über die Version einer zur Verfügung gestellten Software nicht die Wahrheit sagt. Es wird daher ein hauptsächlich theoretisches Modell einer endlichen State Machine entwickelt. Zum Bestimmen der exakten Version einer Software wird diese einigen Testaufgaben unterzogen. Dabei wird die Ausgabe des Programmes bei speziellen Eingaben geprüft um Rückschlüsse ziehen zu können.

Bei einem weiteren Ansatz, Technologien dank Fingerprinting zu erkennen, wird Machine Learning genutzt. Eine mögliche Umsetzung davon wird in der Arbeit „Automatic fingerprinting of websites“ von Berg und Lamberg diskutiert [57]. Für viele existierende Fingerprinting-Methoden ist ein umfangreiches Detailwissen über die zu analysierenden Technologien erforderlich. Um diese Einschränkung zu umgehen und neu erscheinende Technologien schneller erkennen zu können, wird hier ein unüberwachter Machine Learning Algorithmus gewählt. Dabei werden durch verschiedene Charakteristika mehrdimensionale Feature-Vektoren erzeugt, die durch anschließendes Clustering in Gruppen zusammengefasst werden können. Anschließend kann jeder Gruppe eine Technologie zuweisen werden.

Nicht nur öffentliche Server können gefingerprinted werden. Das Paper „CORSICA: Cross-Origin Web Service Identification“ von Dresen et al. beschreibt, wie durch das Besuchen einer Webseite das Intranet des Clients analysiert werden kann [58]. Dabei wird JavaScript Code von der Webseite im Browser des Clients ausgeführt und kann Netzwerkscans innerhalb des Intranets des Clients ausführen. CORSICA erkennt nicht nur andere Geräte im Netzwerk, sondern auch deren genutzte Software und deren Version. Dies realisiert CORSICA, indem verschiedene Ressourcen von den Netzwerkgeräten geprüft werden. Zu diesen Ressourcen gehören JavaScript-, CSS- und Bilddateien. Vor allem die Versionserkennung von CMS funktioniert durch dieses Vorgehen gut. In der eigenen Auswertung des Papers erkennt CORSICA bei Joomla!, WordPress und TYPO3 alle Major- und Minor-Versionen zuverlässig. Jedoch ist die Ermittlung des Patch-Levels bei diesen drei CMS schwieriger und mit einer Genauigkeit von rund 15% deutlich unzuverlässiger.

Von CORSICA inspiriert stellt das Paper „Déjà Vu? Client-Side Fingerprinting and Version Detection of Web Application Software“ von Marquardt und Buhl ein ressourcenschonendes Konzept zur Versionsermittlung vor [59]. Dabei wird versucht, möglichst wenig Anfragen an den zu identifizierenden Webserver zu stellen, um hohe Skalierbarkeit zu ermöglichen. Das aus dem Konzept hervorgehende Tool scannt die Webseite der gegebenen URL und ermittelt aus dem Aufbau und die von der Hauptseite eingebundenen Ressourcen die genutzte Technologie und deren Version. In der Evaluation des erstellten Tools wurden 95% der getesteten Joomla, Drupal und TYPO3 Minor-Versionen erkannt.

Nicht unerwähnt darf die Webseite www.t3versions.com von Torben Hansen bleiben, die verspricht die Version einer TYPO3 Webseite zu ermitteln. Dabei wird jedoch nur die Major- und Minor-Version als Ergebnis angezeigt. In den gesammelten Statistiken der Webseite werden zwar auch Patch-Versionen angeblich ermittelt, wie diese Ergebnisse zustande kommen wird jedoch nicht dargelegt. Durch manuelle Überprüfung der Ergebnisse wurde jedoch festgestellt, dass diese Implementierung nicht ausreichend ist. Es wurden beispielsweise Webseiten, deren Version bekannt war, falsch ermittelt und bei anderen, die anhand ihrer Version CVEs beinhalten, nicht als Gefahr dargestellt.

Obwohl einige besprochene Ansätze bereits gute Ergebnisse beim Ermitteln der Versionsnummer gezeigt haben, so ist die Genauigkeit bei der Bestimmung des Patch-Levels noch nicht ausreichend, da dieses essenziell für die Beurteilung der Sicherheit der eingesetzten Software ist. Das Ziel dieser Arbeit ist das Erstellen eines Konzepts für die bestmögliche Erkennung der Versionsnummer, um auf bestehende Sicherheitslücken hinzuweisen. Der Fokus liegt hierbei auf der Genauigkeit der Bestimmung, weshalb zu Ermittlung Ressourcen-aufwändigere Abfragen in Kauf genommen werden.

4 Vergleichsbasierte Versionserkennung bei CMS

In den folgenden Abschnitten werden die Konzepte für die Lösung der Forschungsfragen thematisiert. Der erste Abschnitt befasst sich mit dem Konzept der Extraktion der Versionscharakteristiken aus einem VCS. Anschließend wird erklärt, wie daraus die Version einer CMS Instanz ermittelt werden kann. Weiterhin wird ein Konzept zur sicheren Übermittlung der erkannten Version in Abschnitt 4.3 dargelegt, die trotz des öffentlichen Zugangs des Tools die Betreiber des CMS schützen sollen.

4.1 Erstellen der Versionscharakteristiken

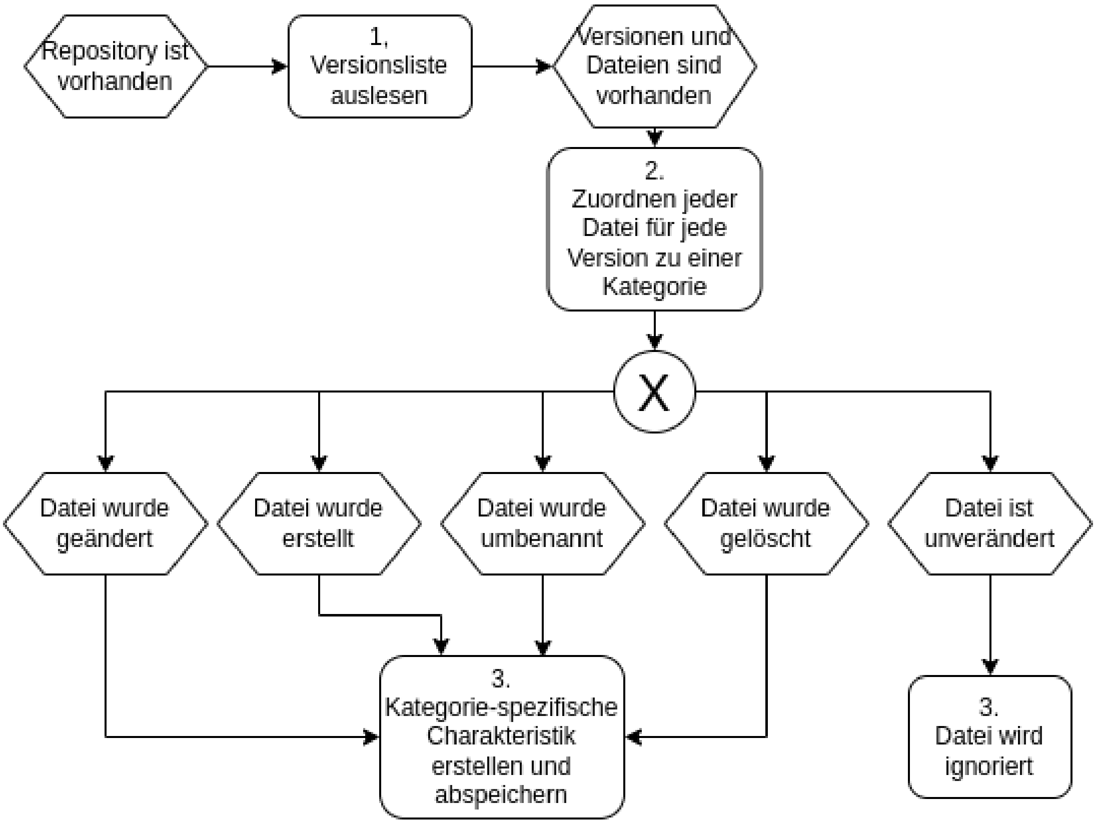


Abb. 4: Konzept der automatisierten Erstellung der Versionscharakteristika

Das automatisierte Generieren von Versionscharakteristika ist unerlässlich, um sicherzustellen, dass die Versionserkennung jederzeit in der Lage ist, die zum jeweiligen Zeitpunkt aktuellsten Versionen der Software zu erkennen. Die Versionscharakteristiken stellen dabei Fingerprints dar. Der Ablauf des dafür entwickelten Konzepts ist in Abbildung 4 in Form einer ereignisgesteuerten Prozesskette modelliert. Als Ausgangspunkt für die Erstellung der Charakteristika wird ein VCS genutzt, beispielsweise Git. Durch das VCS können Informationen, wie eine Liste aller existierenden Versionen sowie die Zustände einer Datei während einer spezifischen Version, abgefragt werden. Die Liste der Versionen wird benötigt, um über alle Versionen numerisch aufsteigend iterieren zu können.

Im zweiten Schritt wird der Zustand jeder Datei mit dem der letzten Version verglichen und darauf basierend in verschiedene Kategorien eingeteilt:

- **Erstellt:** Die Datei existiert in der Aktuellen, jedoch nicht in der vorherigen Version.
- **Gelöscht:** Die Datei existiert in der Vorherigen, jedoch nicht mehr in der aktuellen Version.
- **Umbenannt:** Die Datei existiert in beiden Versionen, ändert jedoch ihren Namen oder Pfad.
- **Geändert:** Die Datei existiert in beiden Versionen, hat jedoch einen geänderten Inhalt.
- **Unverändert:** Die Datei existiert unverändert in beiden Versionen.

Im dritten Schritt werden diese Kategorien genutzt, um weitere Charakteristiken der Dateien zu erheben. Diese sind beispielsweise der Pfad oder der Inhalt der Datei. Nachdem die Charakteristiken erstellt wurden, werden sie zur Version assoziiert gespeichert.

Durch dieses Konzept werden alle Dateien, die sich im VCS befinden, erfasst und mit den ermittelten Charakteristika abgespeichert. Dieses Vorgehen hat jedoch auch seine Grenzen. So können nur im VCS versionierte Dateien erfasst werden, nicht jedoch dynamisch zur Laufzeit generierte oder temporäre Dateien.

4.2 Versionserkennung anhand der Versionscharakteristiken

Basierend auf den Versionscharakteristiken wird die Versionserkennung von einem installierten CMS durchgeführt. Dabei werden jedoch nicht alle erfassten Dateien benötigt, da nicht alle öffentlich über den Webserver erreichbar sind. Im Beispiel von TYPO3 sind nur Dateien relevant, die sich in einem „Public“-Order befinden, da nur diese bei direkten HTTP-Anfragen ausgeliefert werden.

Das Konzept für den Ablauf des Algorithmus zum Erkennen der genutzten CMS Version wird in Abbildung 5 dargestellt. Es basiert auf den Versionscharakteristiken, die durch das Konzept aus Abschnitt 4.1 gesammelt und abgespeichert wurden, wodurch es ohne diese Charakteristiken nicht funktioniert. Um den Algorithmus zur Erkennung der Version zu starten, muss der Nutzer des Tools eine URL für die Webseite, die er analysieren möchte, angeben. Der erste Schritt ist das Initialisieren einer Liste mit allen Versionen anhand der vorher gespeicherten Versionscharakteristika mit den dazugehörigen Versionen. Um die Version des CMS mit möglichst wenigen Anfragen zu ermitteln, muss die Anzahl der gewonnenen Informationen pro Anfrage maximiert werden. Dies kann durch die Auswahl einer Datei mit dem höchstmöglichen Informationspotenzial basierend auf der aktuellen Versionsliste gewährleistet werden. Eine Auswahlmöglichkeit dafür ist, die sich im aktuellen Versionsbereich am häufigsten geändert Datei zu wählen. Nachdem eine Datei ausgewählt wurde, wird sie vom Server abgefragt und die Antwort ausgewertet. Die Charakteristiken der Antwort und der ausgewählten Datei werden verglichen, um die Liste der möglichen Versionen einzugrenzen. Nach diesem Schritt wird ermittelt, ob die Version noch genauer bestimmt werden kann. Dafür wird geprüft, ob es in dem aktuellen Versionsbereich weitere Dateien gibt, welche den Versionsbereich potenziell weiter einschränken könnten. Sollten solche Dateien noch vorhanden sein, so wird der Vorgang von Schritt zwei und drei wiederholt. Ist hingegen die Version eindeutig oder kann nicht genauer

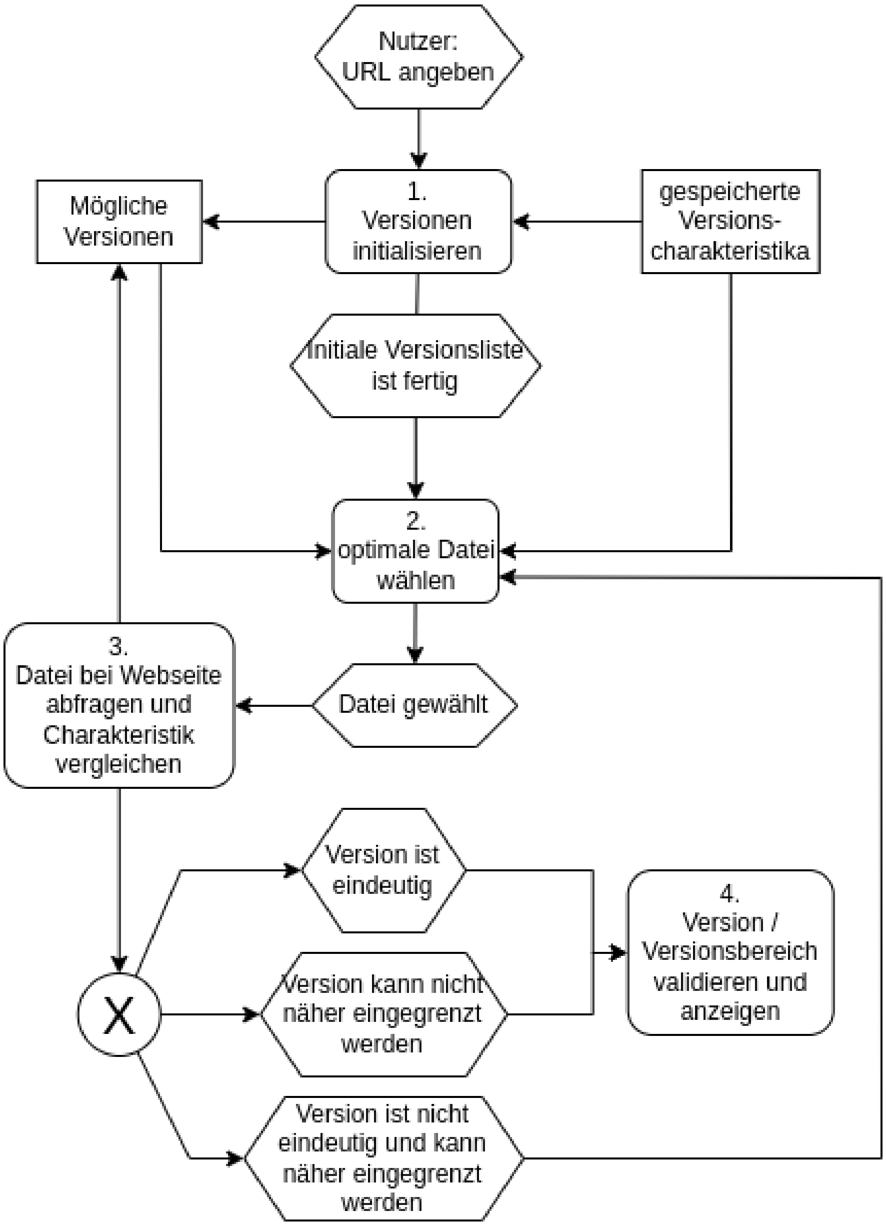


Abb. 5: Webseiten Analyse Konzept

bestimmt werden, so wird im nächsten Schritt das Ergebnis validiert. Dabei wird durch noch nicht verwendete Charakteristiken überprüft, ob das Ergebnis valide ist, oder ob es Charakteristiken gibt, die gegen das Ergebnis sprechen. Zum Schluss wird das Ergebnis zusammen mit der Aussage, ob dieses validiert werden konnte, ausgegeben.

4.3 Bereitstellung der sicherheitskritischen Informationen

Das aus Absatz 4.2 folgende Tool ist als sicherheitskritisch einzustufen, da Angreifer damit die Version einer CMS Instanz ermitteln und daraus Sicherheitslücken ableiten können. Aus diesem Grund ist eine lokale Installation des Tools nicht praktikabel, da in diesem Fall nicht sichergestellt werden kann, ob es für einen Angriff auf ein CMS genutzt wird. Deshalb soll das Tool in Form einer Webapplikation zur Verfügung gestellt werden. Zudem ist das Interagieren durch eine Webseite, besonders für technisch unerfahrene Nutzer, wesentlich einfacher, als das Ausführen eines Programms über die Kommandozeile.

Die Form einer Webapplikation löst jedoch nicht alle Probleme, da sichergestellt werden muss, dass nur Betreiber des zu analysierenden CMS die sicherheitsrelevanten Informationen erhalten. Dies wird durch eine E-Mail-basierte Validierung sichergestellt. Der Nutzer gibt in der Webapplikation die zu überprüfende URL ein. Nachdem durch das Tool ein Ergebnis generiert wurde, werden ausschließlich ausgewählte, oberflächliche Informationen angezeigt. Beispielsweise wird nicht die exakte Versionsnummer ausgegeben, sondern nur, ob eine Versionsnummer ermittelt wurde und ob diese aktuell ist. Während die Ergebnisse angezeigt werden, kann der Nutzer aus einer Liste von E-Mail-Adressen eine auswählen, um einen Link für die genauen Informationen zu erhalten. Diese E-Mail-Adresse muss zu der analysierten Domain gehören und auf einen Admin-Account hinweisen. So kann beispielsweise bei der analysierten Domain „muensmedia.de“ aus „webmaster@muensmedia.de“, „admin@muensmedia.de“, etc. ausgesucht werden. An die ausgewählte E-Mail-Adresse wird anschließend eine Mail mit einem einzigartigen und schwer zu erratenden Link geschickt, unter dem das exakte Ergebnis zeitlich begrenzt zu finden ist. So wird beispielsweise die gefundene Version und alle dazu assoziierten CVEs angezeigt, sowie eine Handlungsempfehlung, falls ein Update erforderlich ist.

5 Implementierung am Beispiel von TYPO3

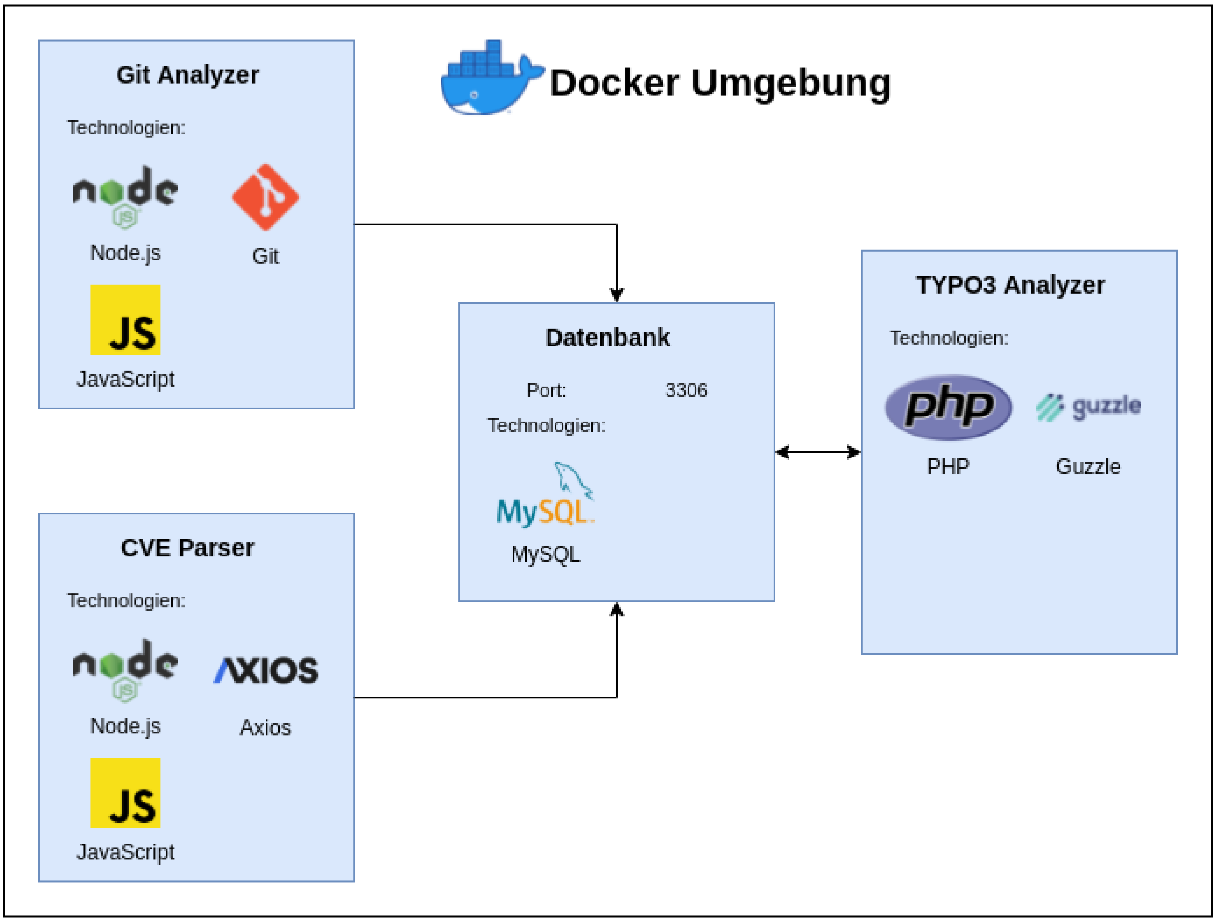


Abb. 6: Struktur der Implementierung [60, 61, 62, 63, 64, 65, 66, 67]

In diesem Abschnitt wird die Implementierung und Funktionsweise der Komponenten des Tools erklärt. Wie in Abbildung 6 dargestellt, gibt es vier verschiedene Instanzen, die alle unterschiedliche Funktionen ausführen. Für die komplette Umgebung wird Docker genutzt. Docker ist ein Programm, das auf Betriebssystemebene Software in sogenannten Containern virtualisiert ausführen kann. Jeder Container ist von Anderen isoliert und kann nur über vordefinierte Kanäle kommunizieren. Docker verbraucht dabei weniger Ressourcen als eine komplette virtuelle Maschine, wodurch es besser möglich ist mehrere Applikationen parallel auf einem Hostsystem laufen zu lassen.

Zentraler Punkt in der Struktur ist die Datenbank, in der alle für die TYPO3 Analyse wichtigen Daten, CVEs und bereits bekannte Ergebnisse liegen. Die für die Analyse von TYPO3 Instanzen wichtigen Daten kommen vom Git Analyzer, der das Git Repository von TYPO3 durchsucht und die vorher als öffentlich, mittel HTTP-Anfragen abrufbaren Dateien abspeichert. Die CVEs in der Datenbank kommen vom CVE Parser, der eine öffentliche Application Programming Interface (API) anfragt und die für TYPO3 relevanten Daten herausfiltert. Anschließend werden die CVEs zu den dazugehörigen Versionen annotiert und in der Datenbank abgelegt. Sowohl der Git Analyzer als auch der CVE Parser müssen nicht permanent laufen, sondern nur in regelmäßigen Zeitabständen erneut gestartet werden, um die Informationen in der Datenbank auf den neusten Stand zu bringen. Beide Tools sind in JavaScript geschrieben und laufen in einer Node.js Umgebung, welche das Ausführen von JavaScript außerhalb des Browsers möglich macht.

In Abbildung 6 rechts neben der Datenbank wird der Container des Algorithmus dargestellt. Dieser und die Datenbank werden beide für die aktive Analyse einer TYPO3 Instanz benötigt. Der Algorithmus muss über die Kommandozeile gestartet werden und ermittelt automatisch die Version der beim Start angegebenen TYPO3 Instanz. Bei der Analyse steht der Algorithmus permanent mit der Datenbank in Verbindung, um die abzufragenden Charakteristiken zu ermitteln und die Antworten des Webservers mit ihnen zu vergleichen. Das Ergebnis der Analyse wird im Anschluss auf der Kommandozeile ausgegeben und in der Datenbank gespeichert, um gegebenenfalls später erneut genutzt werden zu können oder Statistiken zu erheben. Der Algorithmus basiert auf PHP und benutzt die Bibliothek Guzzle für die HTTP Anfragen. Da für das Ermitteln von Ergebnissen eine grafische Benutzeroberfläche wie beispielsweise eine Webseite nicht benötigt wird, wurde diese im Rahmen der Bachelorarbeit nicht implementiert. Für eine mögliche Webseite wird in Abschnitt 4.3 ein Konzept vorgestellt.

5.1 Datenbank

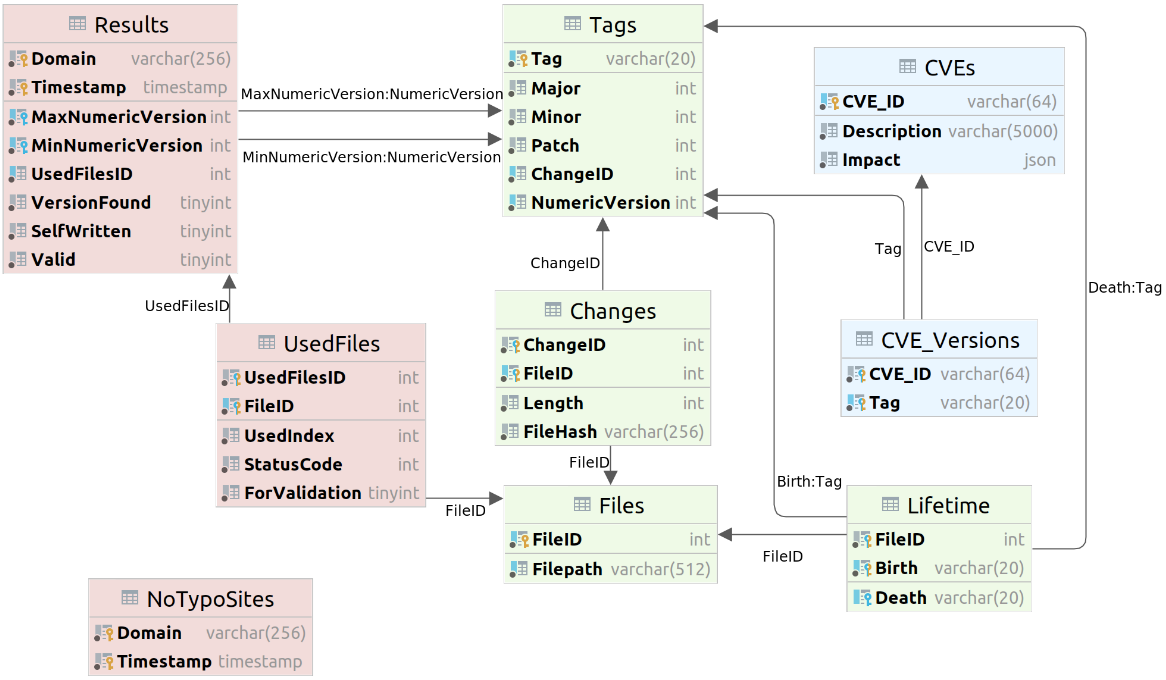


Abb. 7: Entity-Relationship-Modell der Datenbank

Die Datenbank ist der wichtigste Bestandteil der Implementierung. In ihr werden nicht nur Daten für die Erkennung von TYPO3 Instanzen gespeichert, sondern auch Informationen zu CVEs und den bereits analysierten Webseiten. Abbildung 7 illustriert den Aufbau der Datenbank mit ihren neun verschiedenen Tabellen. Im folgenden Abschnitt werden die einzelnen Tabellen und ihr Inhalt erklärt. Die Tabellen können in drei verschiedene Kategorien eingeordnet werden: CVEs (in blau), Fingerprinting-Informationen (in grün) und Analyseergebnisse (in rot).

Die Fingerprinting-Informationen sind die Grundlage auf dem der TYPO3 Analyzer basiert und beinhaltet die Tabellen *Tags*, *Changes*, *Files* und *Lifetime*. Diese Tabellen werden durch das Tool Git Analyzer erstellt und gepflegt. Zu den Inhalten gehören unter anderem die von dem TYPO3 Git Repository annotierten Tags, welche im Fall von

TYPO3 die Versionsnummern angeben. Diese Tags werden im weiteren Text als Versions-Tags bezeichnet. In der Tabelle *Tags* werden alle von TYPO3 bekannten Versions-Tags zusammen mit den entsprechenden Major-, Minor- und Patch-Version sowie den NumericVersion Eintrag gespeichert. Der NumericVersion Eintrag wird für numerische Vergleiche, zum Beispiel ob Version 9.5.20 größer als 10.4.1 ist, benötigt, da dies mit Stringvergleichen schwierig zu lösen ist. Der Eintrag für NumericVersion errechnet sich anhand folgender Formel:

$$NumericVersion = Major * 1.000.000 + Minior * 1.000 + Patch \tag{1}$$

Da es von keiner Minor- oder Patch-Version bei TYPO3 mehr als 1000 gibt, entstehen durch diese Umwandlung keine Ambivalenzen. Als Letztes wird jedem Versions-Tag eine einzigartige ChangeID zugeordnet. Diese ChangeID ist zusammen mit der FileID in der Tabelle *Changes* der Hauptschlüssel. Somit stellt jeder Eintrag in der Tabelle *Changes* eine Datei in einer spezifischen Version dar. Jede dieser Dateien hat zudem noch die Einträge Length, welches die Länge des Inhaltes der Datei in Byte widerspiegelt, sowie den Hash des Inhaltes dieser Datei. Jede FileID zeigt auf den Pfad der Datei. Dieser Zusammenhang wird mit der Tabelle *Files* abgebildet. Zuletzt wird zu jeder Datei die Dauer der Existenz in Bezug auf die Versions-Tags in *Lifetime* definiert. Da eine Datei öfters erscheinen und verschwinden kann, ist neben der FileID auch der Erscheinungszeitpunkt Birth, welcher der erste Versions-Tag ist, in dem die Datei erscheint oder gegebenenfalls erneut erscheint, teil des Primärschlüssels.

Neben den Fingerprinting-Informationen assoziierten Tabellen gibt es noch solche mit den Analyseergebnissen. Zu diesen gehören die Tabellen *Results*, *Usedfiles* und *NoTypoSites*. In der Tabelle *NoTypoSites* werden alle Seiten gespeichert, bei denen während der Analyse die Verwendung von TYPO3 nicht nachgewiesen werden konnte. Zudem wird die Domain mit einem Zeitstempel zusammen abgespeichert, damit später nachvollzogen werden kann, wie alt das Ergebnis ist. In der Tabelle *Results* sind hingegen Domains hinterlegt, bei denen die Nutzung von TYPO3 erkannt wurde. Diese werden ebenfalls mit Zeitstempel sowie zusätzlichen Informationen abgespeichert. Diese sind die maximale und minimale Versionsnummer, eine ID für die genutzten Dateien, ein Wahrheitswert, ob die Version gefunden wurde, ein Wahrheitswert, ob das Ergebnis selbst in die Datenbank geschrieben wurde und ein Wahrheitswert, ob das Ergebnis vom Algorithmus validiert wurde. Direkt zu der UsedFilesID gehört die Tabelle *UsedFiles*. In ihr wird die zu der Suche gehörige ID mit einer Datei verknüpft. Zudem wird mit angegeben, die wievielte überprüfte Datei diese bei der Abfrage ist, welchen Statuscode die Abfrage zurückgegeben hat und ob sie zum Validieren oder für die Analyse genutzt wurde. Die Inhalte für die ganzen Tabellen werden durch den Algorithmus zur Versionserkennung während der Analyse direkt erzeugt und abgespeichert.

Die letzte Kategorie sind die CVEs zu den die Tabellen *CVEs* und *CVE_Versions* gehören. In *CVEs* befinden sich alle für TYPO3 wichtigen CVEs samt Beschreibung und Auswirkungen. Die Auswirkungen sind dabei in einem JSON Format gespeichert, welches von der API übernommen wurde. Die wichtigsten Informationen können somit situativ ausgewählt werden. Zu jedem CVE gibt es meist mehrere Einträge in *CVE_Versions*, die jeweils auf einen Versions-Tag verweisen. Somit können zu jedem CVE die betroffenen Versionen bestimmen werden oder jeder CVE für eine definierte TYPO3 Version gefunden werden.

5.2 Git Repository Crawling

Algorithmus 1 Git Crawling

```
Git Repository klonen
Abspeichern aller Versionstags
Versions-Tags sortieren und Duplikate entfernen
für jeden Versions-Tag mache
    Check-out zu gewünschtem Versions-Tag
    Liste der Änderungen zu letztem Versions-Tag abrufen
    für jede Änderung mache
        Zuordnung zu unverändert, gelöscht, geändert oder hinzugefügt
        wenn nicht gelöscht dann
            Hash und Länge vom Inhalt der Datei bestimmen
            Hash, Länge und Inhalt zum Versions-Tag abspeichern
        ende wenn
        wenn Hinzugefügt dann
            Versions-Tag als Erscheinungszeitraum speichern
        ende wenn
        wenn gelöscht dann
            letzten Versions-Tag als Löschzeitpunkt speichern
        ende wenn
    ende für
ende für
```

Wie in Abschnitt 5 bereits erwähnt, lädt dieses Tool das TYPO3 Git Repository herunter, analysiert es und legt die gewonnenen Informationen in der Datenbank ab. Der Pseudocode in 1 beschreibt das grobe Vorgehen des Algorithmus. Dabei hält sich der Algorithmus an das in Abschnitt 4.1 beschriebene Konzept. Als Erstes wird das Git Repository durch das Git Kommandozeilenwerkzeug geklont und die Versionen abgespeichert. Anschließend werden durch einen regulären Ausdruck die Release-Versionen, also keine Alpha- oder Beta-Versionen, identifiziert. Nachträglich werden Dopplungen entfernt, die beim Beispiel von TYPO3 durch die Änderung der Versionsbezeichnungen auftreten können.

Sind nun alle relevanten Versionen in aufsteigender Reihenfolge geordnet, beginnt der Algorithmus damit über die Versions-Tags zu iterieren. Anschließend wird jede Datei, die im aktuellen Versions-Tag existiert in eine der vier Kategorien „Unverändert“, „Hinzugefügt“, „Geändert“ oder „Gelöscht“ eingeteilt. Ein Spezialfall dabei ist das Umbenennen von Dateien, welche durch Git als separate Kategorie geführt werden. Um dies zu umgehen, werden umbenannte Dateien unter den alten Namen als gelöscht angesehen und unter den neuen Namen als hinzugefügt. Der erste Versions-Tag ist ebenfalls ein Spezialfall, da vor ihm schon andere herausgefilterte Versionen existieren können. Dementsprechend können sich Dateien geändert haben oder gelöscht worden sein. Da diese Informationen allerdings in diesem Fall nicht relevant sind, werden alle Dateien, die in dem ersten Versions-Tag existieren, als hinzugefügt betrachtet. Ist nun die Zuordnung jeder Datei zu einer dieser vier Kategorien abgeschlossen, werden die Inhalte von ausgewählten hinzugefügten und veränderten Dateien eingelesen und anschließend der Hash mit dem *SHA256* Algorithmus gebildet. Zusätzlich werden auch ausgewählte nicht veränderte Dateien eingelesen und abgespeichert, um später einfacher auf diese zurückgreifen zu können. Die

Auswahl der Dateien wird dabei durch einen regulären Ausdruck gesteuert und wählt im Fall von TYPO3 nur Dateien aus, deren Dateipfad den Ordner „Public“ beinhalten, da nur dieser von einem Client erreicht werden kann. Der Hash, eine zum Dateipfad gehörige ID und die Länge des Inhaltes der Datei werden anschließend zum aktuellen Versions-Tag assoziiert in der Datenbank abgespeichert. Wenn die Datei hinzugefügt wurde, wird der Versions-Tag zusammen mit der zum Dateipfad gehörigen ID in einer zusätzlichen Tabelle gespeichert. Das Gleiche geschieht analog, wenn eine Datei gelöscht wird, wobei hier der vorherige Versions-Tag gespeichert wird. Hierdurch sind die Versions-Tags bekannt, in dem diese Datei existiert hat. Wenn dieser Vorgang mit allen Versions-Tags und Dateien abgeschlossen wurde, ist die Datenbank für die Nutzung durch den TYPO3 Analyzer bereit. Der Crawler ist zudem dazu ausgelegt, dass dieser bei einer neu erschienen Version des zu analysierenden Git Repositorys erneut gestartet werden kann, um eine existierende Datenbank zu aktualisieren.

5.3 CVE Parser

Der CVE Parser benutzt eine API der National Vulnerability Database (NVD). Die Anfragen an die API können durch verschiedene Suchfilter eingegrenzt werden. Als Antwort wird eine Datei im JSON Format mit Inhalten aus der Datenbank zurückgeschickt. Die gewollten Informationen, ob beispielsweise ein Eintrag zum TYPO3 CMS dazugehörig ist, sind jedoch nicht einfach zugänglich und unter anderem in einem Beschreibungstext enthalten. Zusätzlich gibt es neben den gewollten CVEs auch viele, die sich auf optionale Erweiterungen von TYPO3 beziehen oder gegebenenfalls nichts mit dem TYPO3 CMS zu tun haben. Um die ungewollten Ergebnisse herauszufiltern, werden die Beschreibungen jedes Eintrags mithilfe von regulären Ausdrücken durchsucht und danach sortiert. Die Ergebnisse werden in drei verschiedene Kategorien gruppiert. Die erste Kategorie sind die CVEs, die sicher zu TYPO3 gehören. Diese müssen in ihrer Beschreibung das Wort „typo3“ enthalten, dürfen dafür jedoch keinen der folgenden Ausdrücke enthalten: „extension“, „package“, „neos“, „(for typo3)“, „joomla!“. Diese Ausdrücke sind häufige Hinweise auf Erweiterungen von TYPO3, oder dass TYPO3 als Vergleich für andere CMS genutzt wurde. Hat ein CVE in der Beschreibung das Wort „typo3“ sowie mindestens einen der eben genannten Ausdrücke, so wird es in die zweite Kategorie eingeordnet. Diese Kategorie beinhaltet alle potenziellen CVEs, die einer manuellen Sichtung bedürfen. Sollte ein CVE in keine der vorherigen Kategorien aufgenommen werden, so wird es automatisch zur dritten Kategorie, die für das TYPO3 CMS unwichtigen CVEs, zugeordnet.

Sind alle CVEs kategorisiert, müssen die betroffenen Versionen zu jedem CVE ermittelt werden. Diese stehen ebenfalls in der erhaltenen JSON, jedoch in unterschiedlichen Formaten. Mithilfe einer Liste von allen möglichen Tags können diese entsprechend zugeordnet werden. Anschließend werden alle CVEs mit ihren Versionen in die Datenbanktabelle *CVEs* und *CVE_Versions* gespeichert. In der Tabelle *CVEs* werden alle CVEs mit deren Beschreibung und dem Impact, ein von der API mitgelieferter Wert, der angibt, wie schlimm dieses CVE ist, gespeichert. Die Verbindung zwischen Versionen und den CVEs wird in der Tabelle *CVE_Versions* hergestellt.

5.4 TYPO3 Fingerprinting

Algorithmus 2 TYPO3 Analyzer

```
URL vom Nutzer erhalten
wenn URL in Datenbank und Ergebnis nicht zu alt dann
    Gib altes Ergebnis zurück
ende wenn
Seite auf Nutzung von TYPO3 prüfen
wenn Wenn es kein TYPO3 nutzt dann
    URL mit Timestamp in NoTypoSites speichern
    Zurück geben, dass es kein TYPO3 nutzt
ende wenn
Schnelle Suche
wenn Schnelle Suche hat kein Ergebnis dann
    Gründliche Suche
ende wenn
wenn Ergebnis vorhanden dann
    Validieren des Ergebnisses
    Ergebnis und Auswertung des Validieren zurückgeben
ende wenn
```

Um das vorgestellte Konzept aus Abschnitt 4.2 umzusetzen, wurde ein Algorithmus zur Versionserkennung für TYPO3 geschaffen. Das Ziel des Algorithmus ist es, die exakt Version einer TYPO3 Instanz bis auf das Patch Level genau zu bestimmen. Der Pseudocode in 2 beschreibt den groben Ablauf. In dem folgenden Abschnitt wird jeder einzelne Bestandteil des Algorithmus genau erklärt.

5.4.1 Erkennen von TYPO3 Seiten

Zu Beginn prüft der Algorithmus, ob die vom Anwender übermittelte URL bereits in der Datenbank vorhanden ist. Dies soll die zu analysierende Seite vor unnötigen Anfragen schützen und Rechenleistung des gesamten Tools schonen. Um zu überprüfen, ob ein Ergebnis bereits vorhanden ist, wird jeweils in der Tabelle *Results* und *NoTypoSites* nach der URL gesucht. Wenn ein oder mehrere Ergebnisse in der *Results*-Tabelle gespeichert sind, die nicht älter als ein Tag sind, wird das jeweils neuste Ergebnis zurückgegeben. Andernfalls wird der Algorithmus erneut ausgeführt. Dasselbe Vorgehen wird mit Ergebnissen aus *NoTypoSites* getan, wobei hier das Ergebnis nicht älter als sieben Tage sein darf. Dieser Unterschied lässt sich damit erklären, dass es sehr unwahrscheinlich ist, dass eine Webseite, die kein TYPO3 nutzt, plötzlich anfängt dies zu nutzen, wohin gegen das Updaten einer bestehenden TYPO3 Instanz wahrscheinlicher ist.

Im nächsten Schritt wird die Seite auf Merkmale einer TYPO3 Installation überprüft. Sollte eine Seite keine typischen Merkmale aufweisen, ist es sehr unwahrscheinlich, dass TYPO3 genutzt wird, was die Analyse der genutzten TYPO3 Version unnötig macht. Um diese Informationen über die zu analysierende Webseite zu erhalten wird Guzzle, ein HTTP Client für PHP, der HTTP Anfragen und deren Verarbeitung vereinfacht, benutzt. Sollte dabei die URL nicht auflösbar sein, oder keine gültige HTML Datei als Antwort auf die HTTP Anfrage zurückkommen, so gilt die Prüfung für die Nutzung von TYPO3 als fehlgeschlagen und der Algorithmus terminiert. Sofern eine gültige HTML

Datei erfolgreich angefragt wurde, wird diese auf drei Merkmale überprüft, die TYPO3 üblicherweise beim Erstellen der Seite automatisch hinzufügt. Das erst und auffälligste Merkmal befindet sich im `<head>` des HTML Dokuments. Es handelt sich um einen von TYPO3 hinzugefügte Kommentar, dass die Seite mithilfe von TYPO3 betrieben wird. Dieser Kommentar kann jedoch durch den Webseitenbetreiber verändert oder komplett ausgeblendet werden. Das zweite Merkmal ist der Generator-Meta-Tag, der sich ebenfalls im `<head>` des HTML Dokuments befindet. Dieser Tag wird von vielen CMS und Frameworks benutzt um zu zeigen, welche Technologien auf dieser Seite benutzt werden. Der Tag hat im Fall von TYPO3 typischerweise die folgende Form: `<meta name="generator" content="TYPO3 CMS">`. Bis zur Version 6.2 enthielt dieser Meta-Tag ebenfalls die Major- und Minor-Version [68]. Das dritte Merkmal sind die eingebundenen Ressourcen. Wie bereits in Abschnitt 2.4 erwähnt, besitzt TYPO3 einige öffentlich zugängliche Routen, aus denen auch die auf der Seite eingebundenen Ressourcen kommen müssen, sofern diese vom selben Server eingebunden werden. Der Alorithmus überprüft dabei die Routen von eingebundenen CSS und JavaScript Dateien. Dafür sucht der Algorithmus für die CSS Dateien nach allen `<link rel="stylesheet">` Tags und analysiert den *href* Eintrag, der angibt, woher die CSS Datei geladen werden soll. Bei den JavaScript Dateien wird nach allen `<script>` Tags gesucht, die einen *src* Eintrag gesetzt haben, welcher die Herkunft der Datei angibt. Sind nun alle Routen zu den Dateien bekannt, kann durch einen regulären Ausdruck überprüft werden, ob die Datei von derselben Domain geladen wird und ob die Route einen öffentlich zugänglichen Pfad von TYPO3 beinhaltet. Neben den kontrollierten Ressourcen könnten zusätzlich dazu noch Bilder-, Video- und andere Dateien überprüft werden. Da jedoch CSS und JavaScript Dateien meist ein eindeutiges Ergebnis liefern, werden diese Art von Ressourcen nicht genauer überprüft. Wenn mindestens eines dieser Merkmale zutrifft, so geht der Algorithmus davon aus, dass die Webseite hinter der URL TYPO3 nutzt und der Algorithmus läuft weiter. Andernfalls ordnet der Algorithmus die URL mit aktuellem Timestamp in die *NoTypoSites* Tabelle ein und gibt dies als Ergebnis an den Nutzer zurück.

5.4.2 TYPO3 Analyse

Um bei einer vorhandenen TYPO3 Installation mit möglichst wenig Anfragen an den Webserver die Version zu erkennen, sind im Algorithmus zwei unterschiedliche Erkennungsstrategien implementiert. Beide beruhen auf dem Vergleichsprinzip wie in Abschnitt 2.5.2 beschrieben, sind jedoch in der Funktionsweise etwas unterschiedlich.

Die erste Methode wird im folgenden als fastSearch bezeichnet und versucht, die Version mit so wenig Anfragen wie möglich zu identifizieren. Diese Suchmethode initialisiert den aktuellen Versionsbereich mit allen aus der Datenbank bekannten Versionen. Um die maximale Menge an Informationen durch eine Anfrage zu gewinnen, wird durch eine spezielle SQL Abfrage die Datei ermittelt, die sich in dem bereits bekannten Versionsbereich am häufigsten verändert. Zur Auswahl könnte ebenfalls noch die Existenzdauer jeder Datei mit einbezogen werden, allerdings zeigte sich durch diese zusätzliche Einschränkung in empirischen Tests keine merkliche Verbesserung der Effizienz. Ist die sich am häufigsten ändernde Datei bekannt, so wird mit Guzzle eine HTTP Anfrage an dessen dazugehörige Route gesendet. Zudem wird die Datei zu einer Liste der bereits abgefragten Routen hinzugefügt, um diese später nicht erneut zu benutzen. Liefert der Server die angefragte Datei aus, wird ein Hash aus deren Dateiinhalt gebildet. Der Versionsbereich zu dem entsprechenden Hash und der Route wird bei der Datenbank angefragt und anschließend die Schnittmenge aus dem alten Versionsbereich und des für die Datei erkannten Bereiches gebildet und als neuer Versionsbereich gesetzt. Antwortet hingegen der Server auf die HTTP Anfrage nicht oder mit einem Fehlercode, so wird bei dieser Suchmethode angenommen, dass diese Datei nicht auf dem Server existiert. Dementsprechend werden alle Versionen, in denen diese Datei existiert, aus dem aktuellen Versionsbereich entfernt. Sollte die Version nicht genauer bestimmbar sein, da es keine weiteren Dateien gibt, die sich im aktuellen Versionsbereich ändern, terminiert fastSearch und liefert den erkannten Versionsbereich als Ergebnis zurück. fastSearch terminiert ebenfalls, wenn sich im aktuellen Versionsbereich keine Version befindet, die Länge des Versionsbereiches also null ist. Das bedeutet, dass fastSearch nicht in der Lage war ein Ergebnis zu ermitteln und die Suche fehlgeschlagen ist. Sollte das der Fall sein wird die Liste der bereits abgefragten Dateien verworfen und die zweite Suchmethode wird gestartet.

Das Ziel der im folgenden als carefulSearch bezeichneten Methode ist die Versionserkennung ohne Rücksicht auf die Anzahl der Anfragen. Damit diese Methode nicht alle möglichen Dateien abfragt, wird zunächst überprüft, ob die Route /typo3 überhaupt erreichbar ist. Neben dieser Route wird zusätzlich eine Liste von Dateien überprüft, die dynamisch aus den Einträgen in der Datenbank generiert werden. Diese Dateien repräsentieren ein minimales Set an Dateien, welches sämtliche TYPO3 Versionen abdeckt, ähnlich eines Minimum Spanning Tree. Sollte eine dieser Dateien gefunden werden, so wird die Suche begonnen, wobei der Bereich, in dem diese Datei existiert, als anfänglicher Versionsbereich genutzt wird. Sollte /typo3 erreichbar sein, jedoch keine Datei gefunden werden, so wird die Suche abgebrochen und ein entsprechender Versionsbereich, der nicht überprüfbar ist, als Hinweis zurückgegeben. Die Suche wird ebenfalls abgebrochen, falls weder /typo3 erreichbar ist noch eine Datei gefunden wurde. Falls eine Suche nach beiden Überprüfungen anfängt, verläuft sie ähnlich der Suche bei fastSearch. Der Unterschied ist jedoch, dass der Versionsbereich nicht verkleinert wird, falls angefragte Dateien nicht erreichbar sind. Die Suche endet nur, wenn ein eindeutiger Versionsbereich fest steht, oder ein Widerspruch in den Dateien gefunden wurde. Ein solcher Widerspruch entsteht beispielsweise, wenn der alte Versionsbereich keine Schnittmenge mit dem Versionsbereich einer gefundenen Datei hat.

5.4.3 Validierung des Ergebnisses

Nachdem durch eine der beiden Suchmethoden die Version ermittelt wurde, wird zum Schluss das Ergebnis validiert, um die Möglichkeit eines Fehlers so gering wie möglich zu halten. Da der Algorithmus zur Analyse der Version einer TYPO3 Instanz nur geschlossene Versionsbereiche erkennt, können in dem zu validierenden Ergebnis keine Lücken auftreten, was für spätere Annahmen wichtig ist. Neben dem zu validierenden Ergebnis sind die von der Suche benutzten Dateien ebenfalls bekannt. Diese sind wichtig, da das Validieren keine bereits genutzten Dateien erneut abfragt, um die Aussagekraft des Validierens zu bewahren. Insgesamt benutzt der Algorithmus drei verschiedene Ansätze um das Ergebnis zu validieren. Dabei ist die Ausführung der Ansätze nach steigender Komplexität und Anzahl der Abfragen geordnet. Für den logischen Aufbau werden diese in umgekehrter Reihenfolge erklärt, da der erste und zweite Ansatz Spezialfälle des dritten sind. Der Zustand von Dateien, der in den folgenden Absätzen erwähnt wird, beschreibt dabei den Hash einer Datei. Dieser kann in mehreren Versionen auftreten, ändert sich jedoch auch, wenn sich die Datei ändert (siehe Abschnitt 5.2).

Der dritte Ansatz, der erst getestet wird, wenn beide vorherigen Ansätze fehlgeschlagen sind, besteht aus mehreren Teilschritten. Der erste Schritt ist es alle Dateien zu finden, die in dem Versionsbereich vorkommen. Dabei kann dieselbe Datei mehrere Male vorkommen, da der Hash der Datei mitberücksichtigt wird. Nachdem diese ermittelt wurden, werden sie nacheinander bei der TYPO3 Instanz angefragt. Wird die angefragte Datei ausgeliefert, so beginnt Schritt zwei. Durch die vorangegangenen Ansätze ist die Information bekannt, dass die überprüfte Datei in mindestens einer Version außerhalb des zu überprüfenden Versionsbereich liegt. Zudem ist durch den Aufbau des Analysealgorithmus für TYPO3 bekannt, dass der gefundene Versionsbereich keine Lücken hat. Somit teilt der vom Algorithmus gefundene Versionsbereich die Menge aller Versionen in drei Gruppen, eine mit allen Versionen, die kleiner sind, eine mit allen Versionen, die größer sind, und eine mit allen Versionen, die sich innerhalb des zu validierenden Bereiches befinden. Die Versionen, die zu der vom Server zurückgegebenen Datei assoziiert werden können, können ebenfalls in diese Gruppen aufgeteilt werden. Um den vom Algorithmus gegebenen Versionsbereich zu validieren, müssen nun die beiden Gruppen mit den außerhalb liegenden Versionen widerlegt werden. Dies wird realisiert, indem für beide außerhalb liegenden Gruppen jeweils eine Datei gefunden wird, die in ihrem Zustand in allen Versionen innerhalb der Gruppe vorkommt, jedoch nicht in den Versionen des zu validierenden Bereiches. Falls für beide Gruppen mindestens eine Datei gefunden wurde, folgt nun der nächste Schritt. In diesem werden die Dateien vom Server angefragt und von der Antwort der Hash gebildet. Dieser Hash sollte ungleich zu dem Hash sein, den die Datei in dem zu widerlegenden Versionsbereich hat. Sollte dieser unerwarteterweise doch gleich sein, so ist der erkannte Versionsbereich falsch und die Validierung schlägt fehl. Wenn aus beiden Gruppen eine Datei überprüft wurde, dessen Hash unterschiedlich ist, so gilt der zu validierende Versionsbereich als bestätigt, da dadurch die beiden außerhalb liegenden Gruppen von Versionen widerlegt wurden. Da der erste Schritt dieses Ansatzes sehr viele Dateien, die überprüft werden können, hervorbringt, werden die für diesen Ansatz erlaubten Anfragen auf zehn Stück limitiert. Nachdem diese zehn Anfragen erreicht wurden, wird der zweite Schritt noch gegebenenfalls durchgeführt. Sollte innerhalb der Limitierung der ermittelte Versionsbereich nicht validiert werden können, so das Ergebnis als nicht validierbar gewertet.

Der zweite Ansatz ist ein speziellerer Fall des dritten und benötigt weniger HTTP Anfragen. Hierbei wird in der Datenbank nach zwei Dateien gesucht, die gemeinsam den Versionsbereich validieren. Eine der beiden muss dabei vor und in dem Versionsbereich denselben Zustand haben, jedoch nicht nach dem Versionsbereich. Bei der anderen Datei muss derselbe Zustand während und nach, jedoch nicht vor dem Versionsbereich bestehen. Sollte für eine der beiden Dateien kein Eintrag in der Datenbank existieren, so gilt dieser Ansatz als fehlgeschlagen. Nachdem beide Dateien ermittelt wurden, wird wieder mithilfe von Guzzle die jeweilige Datei angefragt und der Hash von der Antwort gebildet. Sollte bei beiden Dateien der gebildete Hash mit dem erwarteten, aus der Datenbank stammenden Hash übereinstimmen, ist die Version der TYPO3 Instanz innerhalb der Schnittmenge der Versionen von den beiden zuvor überprüften Dateien. Diese Schnittmenge ist eine Teilmenge des zu validierenden Versionsbereiches, wodurch die Existenz der beiden Dateien das Ergebnis des Algorithmus validiert.

Der erste Ansatz ist der simpelste von allen. Er wird als erstes ausgeführt, da er am wenigsten Anfragen benötigt, jedoch liefert dieser Ansatz am seltensten ein Ergebnis. Das liegt daran, dass dieser Ansatz eine Datei sucht, die in ihrem Zustand nur in dem zu validierenden Versionsbereich existiert. Da nicht für alle Versionsbereiche eine solche Datei existiert, schlägt dies in vielen Fällen fehl. Sollte das der Fall sein, so wird sofort mit dem nächsten Ansatz begonnen. Wurde hingegen eine solche Datei gefunden, wird diese beim Server angefragt. Sofern eine Datei als Response vom Server zurückgeschickt wird, muss der Hash mit dem aus der Datenbank übereinstimmen. Wenn der Hash sich unterscheidet und in einer anderen Version existiert, terminiert der Algorithmus und merkt an, dass der gefundene Versionsbereich nicht richtig ist.

6 Evaluation

In diesem Abschnitt werden die Ergebnisse des Tools aus Abschnitt 5 dargelegt. Zunächst wird im Abschnitt 6.1 die Testumgebung für die Whitebox Tests beschrieben, die im Anschluss in Abschnitt 6.2 genutzt wird, um die Funktionalität des Algorithmus zu überprüfen. Danach wird der Algorithmus an echten Webseiten in Abschnitt 6.3 angewandt und evaluiert. Im Anschluss werden die Ergebnisse diskutiert.

Bei allen Ergebnissen ist zu beachten, dass in diesem Fall nur die LTS Versionen von TYPO3 betrachtet werden können, da nur diese frei zugänglich sind. Bei der Auswertung der Ergebnisse von öffentlichen Webseiten wurden immer die neusten Versionen angenommen, sofern ein Versionsbereich erkannt wurde. Sollte die erkannte Version die letzte LTS Version einer Major-Version sein, die sich bereits im ELTS befindet, so kann es sein, dass die erkannte Version sich von der tatsächlichen Version im Patch-Level unterscheidet, da ELTS Versionen nicht erkannt werden können. Es wird für die Auswertung angenommen, dass die erkannte Version korrekt ist und es sich um keine ELTS Version handelt, vor allem, weil die ELTS Versionen im Gegensatz zu den LTS Versionen käuflich erworben werden müssen, wodurch die Nutzung dieser unwahrscheinlicher wird.

6.1 Testumgebung

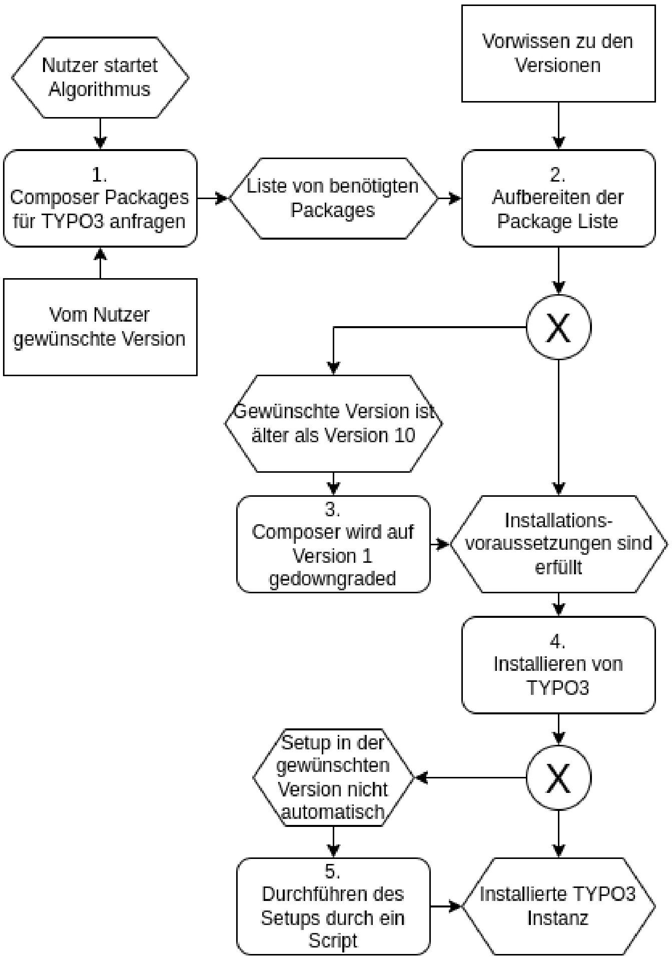


Abb. 8: Ereignisgesteuerte Prozesskette zum Installationsablauf von TYPO3 für Whitebox Tests

Für die Ausführung der Whitebox Tests und um die Umsetzung der Implementierung am Beispiel von TYPO3 zu testen, wird eine einheitliche Testumgebung benötigt. Das Aufsetzen der Testumgebung ist dabei automatisiert, um eine schnelle und zuverlässige Installation der gewünschten Version zu gewährleisten. Da für die Installation von TYPO3 Composer genutzt wird, entstehen einige Probleme. Beispielsweise wird bei einer Composer basierten Installation von TYPO3 nur die Major- und Minor-Version angegeben, die Patch-Version wird automatisch auf die neuste Version gesetzt. Diese und andere Probleme werden von einem für die automatische Installation der Testumgebung geschriebenen Algorithmus beachtet und automatisch gelöst, wodurch die Installation von gewünschten Versionen von TYPO3 bis aufs Patch-Level genau selbständig und einheitlich funktionieren.

Bevor der Algorithmus zum automatischen Installieren der Testumgebung gestartet wird, muss der Nutzer die gewünschte TYPO3 Version angeben. Der exakte Ablauf wird in Abbildung 8 beschrieben. Wenn nun der Algorithmus gestartet wird, wird eine Liste aller für die Installation benötigten Pakete anhand der Major- und Minor-Version von Composer heruntergeladen, jedoch noch nicht installiert. Anschließend wird die Liste automatisch überarbeitet, um die Versionen der Pakete auf die vom Nutzer gewünschte Version anzupassen. In einigen Versionen werden auch zusätzliche Pakete installiert, ohne die die Installation nicht funktionsfähig wäre. Diese wurden durch empirisches Testen ermittelt. Im dritten Schritt wird überprüft, ob die zu installierende Major-Version kleiner als Version 10 ist, da alle Versionen unter 10.0.0 mit Composer v1 installiert werden müssen. Da in der Testumgebung standardmäßig Composer v2 genutzt wird, muss gegebenenfalls ein Downgrade durchgeführt werden. Nachdem nun alle Versionen der Pakete angepasst und die richtige Version von Composer genutzt wird, wird im vierten Schritt TYPO3 installiert. Sollte bei der gewünschten TYPO3 Version während der Installation das automatische Setup nicht möglich gewesen sein, so wird in diesem Fall das Setup durch ein selbst geschriebenes Script durchgeführt. Dieses Script geht den eigentlich manuellen Weg des Setups automatisch durch. Das Setup ist deshalb wichtig, da es die Verbindung zwischen TYPO3 und der Datenbank einrichtet, den Admin Account erstellt und erst danach die TYPO3 Instanz richtig genutzt werden kann. Dabei beachtet das Script die installierte Version von TYPO3, um das Setup korrekt ausführen zu können.

Nach der Installation wird bei jeder TYPO3 Instanz eine vorher erstellte HTML Seite mit eingebundener CSS und JavaScript Datei eingerichtet, um eine normale Webseite nachzustellen und die Funktionalität der TYPO3 Instanz zu überprüfen. Die komplette Testumgebung wird in Docker-Containern organisiert, um im Hostsystem so geringe Spuren wie nur möglich zu hinterlassen, so wenig Performance wie möglich zu beanspruchen und schnell auf andere Systeme übertragen werden zu können. Zur Zeit der Arbeit gibt es 401 verschiedene, öffentlich erhältliche Release-Versionen. Da für eine vereinfachte Installation Composer genutzt wird, sind nicht alle Versionen installierbar, da bei Composer erst ab Version 8.7.10 die neueren Pakete genutzt werden und Version 11.5.5 zum Zeitpunkt der Arbeit die neuste Version ist. Der fokussierte Versionsbereich umfasst somit 109 Versionen. Von diesen 109 Versionen konnten insgesamt 99 Versionen erfolgreich installiert werden. Die zehn Versionen, die nicht installiert werden konnten, konnten aufgrund diverser technischen Probleme bei dem Installations- oder Setup-Schritt nicht durchgeführt werden. Diese wenigen Versionen sind jedoch für die Aussagekraft der Testumgebung nicht von Belang, da die Aufgabe der Testumgebung durch das Abdecken von verschiedenen Major-, Minor- und Patch-Versionen erfüllt ist. Die 99 erfolgreich installierten TYPO3 Versionen dienen als Grundlage für die darauf folgenden Whitebox Tests.

6.2 Versionserkennung in der Testumgebung

Da der implementierte Algorithmus aus Abschnitt 5.4 auf seine Funktionalität geprüft werden muss, bevor dieser an echten Webseiten angewandt wird, werden vorher sogenannte Whitebox Test anhand der Testumgebung aus Abschnitt 6.1 durchgeführt. Eine Whitebox beschreibt dabei ein System, bei dem der Zustand des Systems zu jeder Zeit bekannt ist. Somit ist im Fall von einem Softwaresystem beispielsweise der Quellcode frei lesbar und es kann anhand von diesem vorausgesagt werden, wie das System auf gewisse Anfragen reagieren wird. Beim Algorithmus für die Versionserkennung wird beispielsweise erwartet, dass eine selbst installierte TYPO3 Instanz in Version 9.5.25 exakt als solche erkannt wird. Um zu überprüfen, ob genau dieses Verhalten eintritt, wurden alle 99 installierbaren Versionen von der Testumgebung manuell mithilfe des Algorithmus überprüft, die Ergebnisse zusammengetragen und in mehreren Graphen veranschaulicht.

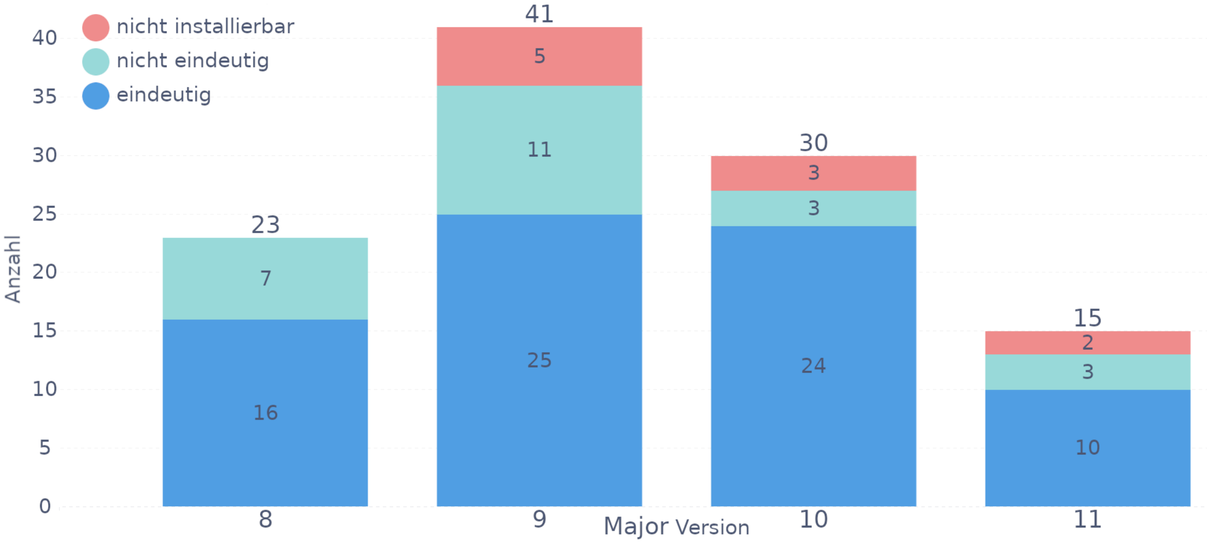


Abb. 9: Genauigkeit der Zuordnung zu Major-Versionen im Whitebox Test

In Abbildung 9 sind die ersten Ergebnisse der Whitebox Tests visualisiert. Dabei sind auch die zehn Versionen mit inkludiert, die nicht installiert werden konnten. Die restlichen 99 Versionen können vom Algorithmus überprüft werden. Der Algorithmus erkennt dabei alle Versionsbereiche richtig, jedoch sind nicht alle Versionen eindeutig. So wird beispielsweise Version 9.5.25 nicht exakt, sondern als Versionsbereich von 9.5.25 bis 9.5.27 erkannt. Insgesamt wird bei 24 Versionen ein Versionsbereich erkannt. Bei allen Versionsbereichen war die eigentliche Version mit enthalten, zudem unterschieden sich die Versionen innerhalb des Versionsbereiches nur im Patch-Level. Der erkannte Versionsbereich wird ebenfalls bei allen in ihm befindlichen Versionen erkannt. So wird bei dem eben genannten Beispiel nicht nur in Version 9.5.25 der Versionsbereich von 9.5.25 bis 9.5.27 erkannt, sondern auch in den Versionen 9.5.26 und 9.5.27. Die größten Versionsbereiche, die in den Whitebox Tests aufgetreten sind, umfassen drei verschiedene Versionen. Neben den 24 Versionen, bei denen ein Versionsbereich erkannt wurde, konnte bei 75 Versionen die exakte Version bestimmt werden.

Die 99 installierbaren Versionen wurden nach der Erkennung, wie in Abschnitt 5.4.3 beschrieben, validiert. Abbildung 10 visualisiert die Anzahl an validierbaren Versionen pro Major-Version. Dabei fällt auf, dass einige Versionen nach dem Erkennungsprozess nicht validiert werden können. Das liegt daran, dass beim Validieren des Versionsbereiches

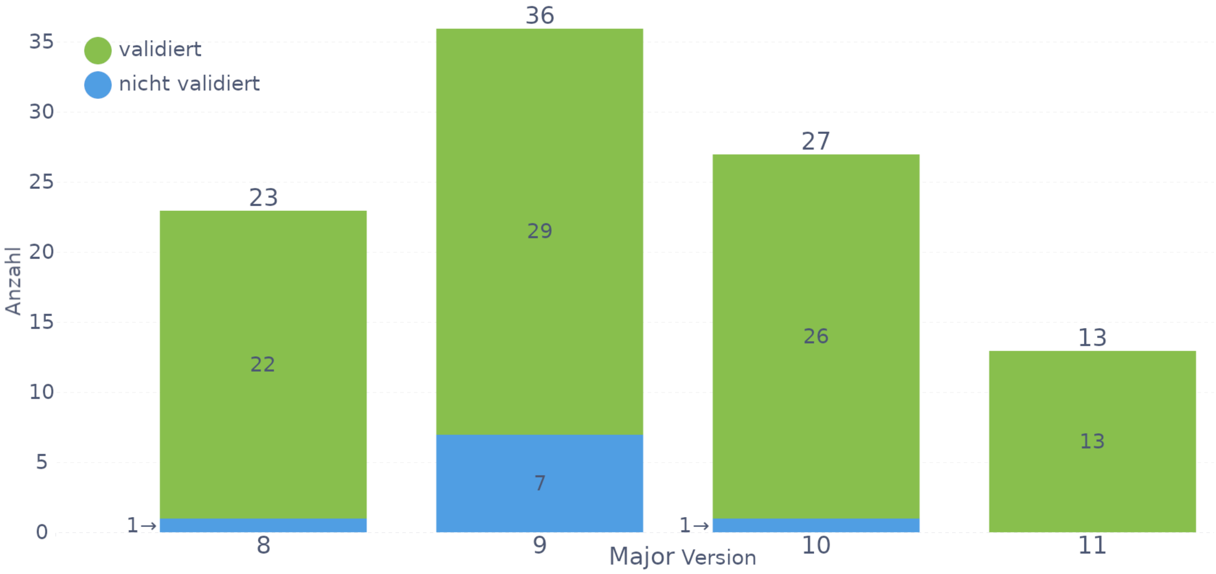


Abb. 10: Anzahl der validierten Instanzen pro Major-Version

keine Dateien genutzt werden, die bereits zum Ermitteln der Version genutzt wurden. Da in wenigen Fällen nur vereinzelte Versionscharakteristiken den Unterschied zwischen zwei Versionen darstellt, kann ohne diese Charakteristiken die Version nicht validiert werden. In der Major-Version 9 tritt dieses Problem häufig auf, weshalb in dieser sieben Versionen nicht validiert werden können. Die Major-Version 11 ist die Einzige, bei der alle installierten Versionen nach dem Erkennen auch validiert werden konnten.

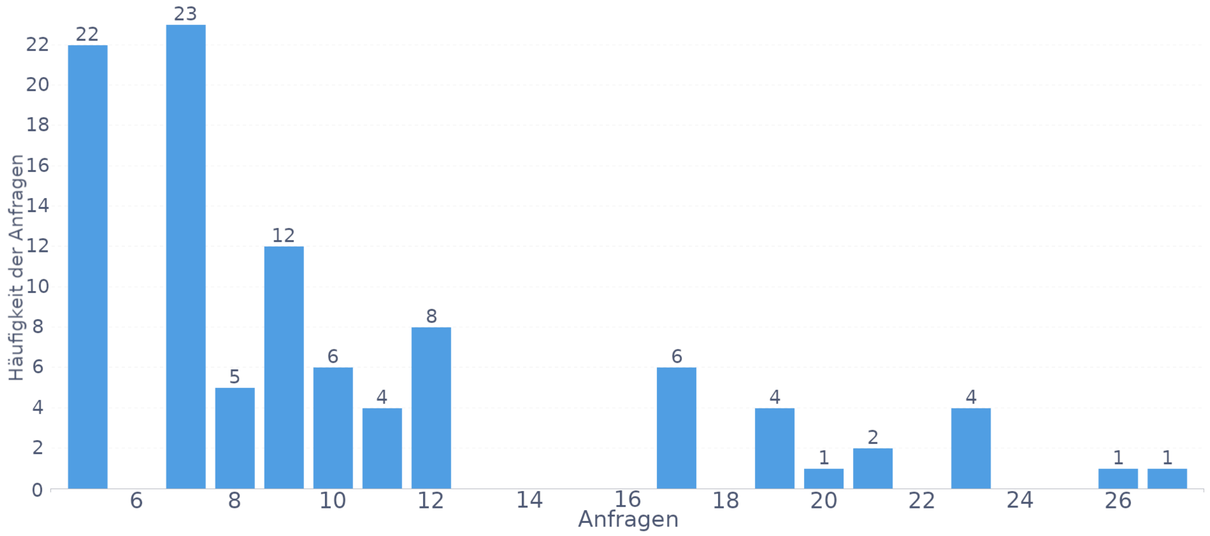


Abb. 11: Häufigkeitsverteilung über die Anzahl der Anfragen, welche für eine Erkennung der Version erforderlich waren

In Abbildung 11 wird die Anzahl der benötigten HTTP-Anfragen pro Version abgebildet. Dabei beinhalten diese Anfragen sowohl den Schritt zum Erkennen, ob eine Webseite TYPO3 nutzt, als auch das Analysieren der Version der Instanz, sowie den Prozess zum Validieren. Für alle drei Schritte werden dabei mindestens fünf Anfragen gebraucht und maximal 27. Im Durchschnitt werden rund zehn Anfragen benötigt. Die Anzahl der ausgeführten Anfragen ist somit in der Regel nicht höher als bei einem normalen Aufruf der

Webseite durch den Webbrowser und stellt keine erhöhte Belastung des Webservers dar.

Die eben dargelegten Ergebnisse der Whitebox Tests zeigen, dass die Implementierung des Konzepts erfolgreich ist. Die Major- und Minor-Version wird in jedem Fall korrekt erkannt. Patch-Versionen werden ebenfalls in einem Großteil der Fälle eindeutig zugeordnet, nur in wenigen Fällen ist die Erkennung ungenau. Eine fehlerhafte Erkennung, bei der die tatsächliche Version sich nicht innerhalb des ermittelten Bereichs befand, ist in keinem Fall aufgetreten. Zusätzlich wurde gezeigt, dass der Algorithmus dabei einen zu analysierenden Server während der Ermittlung der Version nicht zu stark auslastet. Mit diesem Wissen können die Tests an echten Webseiten ohne Bedenken durchgeführt werden.

6.3 Versionserkennung von Internetseiten

Neben den kontrollierten Tests in einer dafür angelegten Testumgebung muss der Algorithmus ebenfalls an öffentlichen Webseiten überprüft werden, da diese Webseiten das Ziel des Tools zur Versionserkennung sind. Für diese Tests wird eine Liste von TYPO3 Webseiten benötigt. Für die Erstellung der Liste wurden zwei verschiedene Ansätze verfolgt. Die Erste Ansatz befasst sich mit Webagenturen, die unter anderem die Erstellung und Wartung von TYPO3 Webseiten anbieten. Auf der jeweiligen Webseite wird bei den meisten Webagenturen ein Portfolio geführt, bei dem alle von dieser Agentur erstellten TYPO3 Webseiten aufgelistet sind. Die aufgelisteten Webseiten wurden alle manuell auf TYPO3 überprüft, um sicherzustellen, dass diese nach wie vor existieren und TYPO3 verwenden. War dies der Fall, so wurde die Webseite zur Liste hinzugefügt. Als zweite Möglichkeit für das Auffinden von TYPO3 Webseiten wurde Shodan genutzt. Shodan ist eine Suchmaschine, die alle mit dem Internet verbundenen Geräte durchsucht. Über eine API wurde eine Liste von diversen TYPO3 Webseiten bei Shodan angefragt und die Ergebnisse ebenfalls manuell überprüft und der bestehenden Liste von TYPO3 Installationen hinzugefügt. Hierbei wurde darauf geachtet, dass die Webseite tatsächlich Inhalt hat und keine leere, frisch installierte Instanz ist.

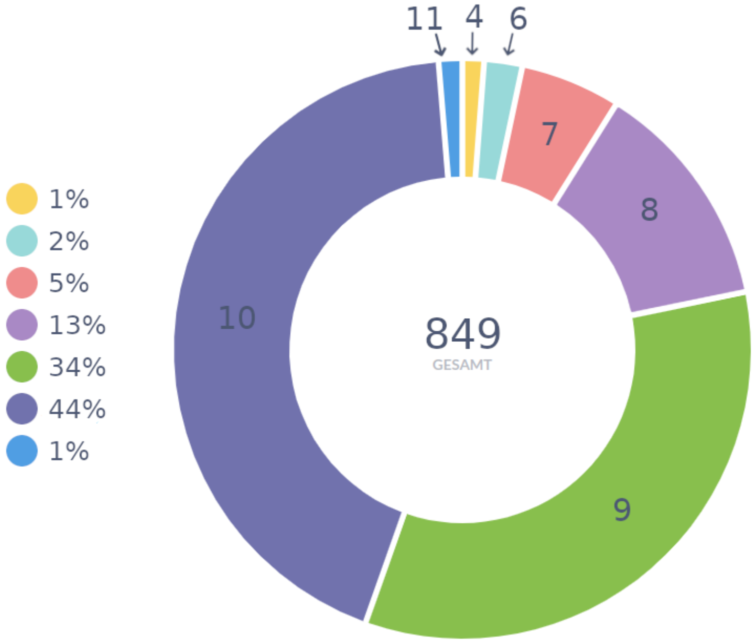


Abb. 12: Verteilung von Major-Versionen bei den analysierten Webseiten

Nach der Erstellung der Liste von TYPO3 Webseiten wurden erfolgreich 849 Webseiten analysiert. Alle Ergebnisse wurden am 21.01.2022 gesammelt und anschließend ausgewertet. Die dabei erkannten Versionen und deren Anzahl ist in der Grafik 15 im Anhang abgebildet. In Abbildung 12 wird die Verteilung der Major-Versionen der 849 Webseiten dargestellt. Dabei ist zu beachten, dass die erste Major-Version von TYPO3, die Version 3 ist, diese jedoch bei keiner Webseite gefunden wurde, und es keine Major-Version 5 gibt. In der Abbildung ist zu sehen, dass ein Großteil der Webseiten mit rund 78% sich auf Major-Version 9 oder 10 befindet. Der geringe Anteil der Version 11 lässt sich dabei damit erklären, dass diese zum Zeitpunkt der Tests gerade erst erschienen ist.

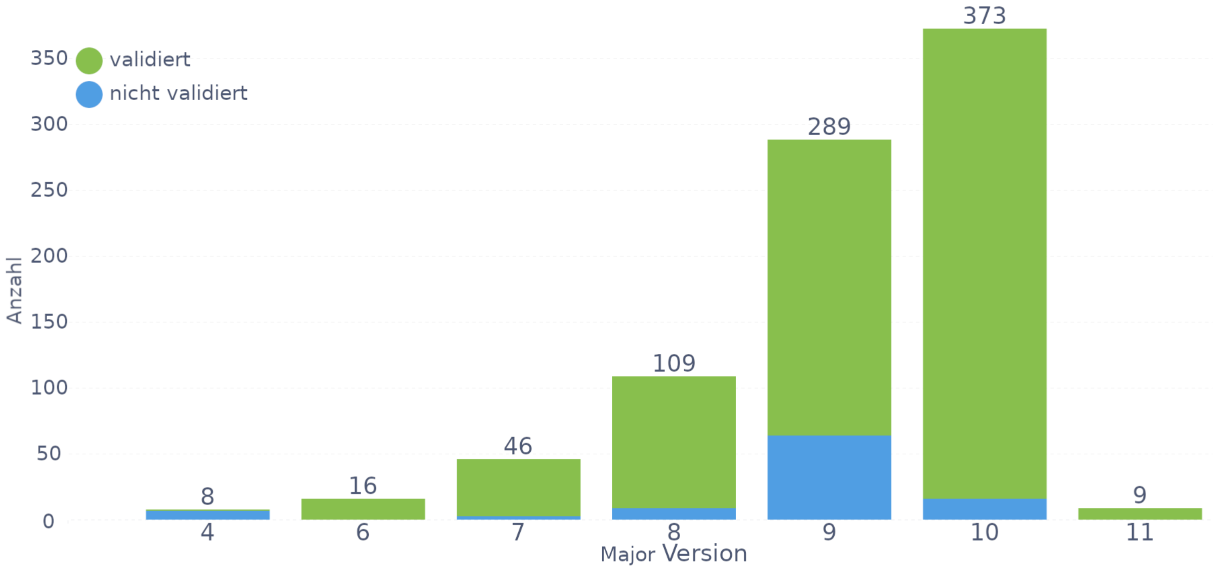


Abb. 13: Verteilung von validierten online Instanzen nach Major-Versionen (TYPO3 beginnt mit Version 3, wobei keine erkannte Webseite diese hatte, und Version 5 existiert nicht)

Neben dem Ermitteln der Version wurde bei den Tests auch die gefundene Version validiert. Abbildung 13 zeigt, dass in der Major-Version 9 viele Webseiten nicht validiert werden konnten, was jedoch nicht bedeutet, dass die erkannte Version fehlerhaft ist. Werden die Ergebnisse aus den Whitebox Tests hinzugezogen, zeigt sich, wieso eine fehlgeschlagene Validierung nicht mit einer fehlerhaften Versionserkennung gleichzusetzen ist. Bei genauerer Betrachtung der Ergebnisse aus den Whitebox Tests, fällt auf, dass alle nicht validierbaren Versionen der Major-Version 9 auch nicht in der Testumgebung validiert werden konnten. Dadurch hat ein fehlgeschlagener Versuch die Version zu validieren keine Aussagekraft für die Korrektheit der Version. Werden die Betrachtungen auf die Major-Versionen 8 und 10 erweitert, gehören auch dort ein Großteil der nicht validierten Versionen zu den in der Testumgebung nicht validierbaren Versionen. Von den 99 nicht validierten Webseiten liegen 89 in dem von der Whitebox abgedeckten Versionsbereich (8.7.10 bis 11.5.5). Werden die Versionen herausgefiltert, die bereits in der Testumgebung nicht validiert werden konnten, bleiben nur drei Webseiten. Unter den drei Webseiten wurde bei einer die Version 8.7.32 erkannt, die die letzte LTS Version er Major-Version 8 darstellt. Wie bereits erwähnt können mit den verwendeten Daten nur LTS Versionen erkannt werden und keine ELTS, weshalb die fehlgeschlagene Validierung ein Anzeichen dafür ist, dass diese TYPO3 Instanz wahrscheinlich eine ELTS Version verwendet. Bei den restlichen beiden Webseiten zeigt die fehlgeschlagene Validierung, dass die erkannte

Version wahrscheinlich einen Fehler enthält. In der Testumgebung konnte jedoch dieses Verhalten nachgestellt werden, indem eine TYPO3 Instanz nicht sauber geupdated wurde. Dabei wurden alte und nicht mehr benötigte Dateien nicht gelöscht, was dazu führte, dass zwar die richtige Version erkannt wurde, diese jedoch nicht mehr validiert werden konnte. Bei den anderen Webseiten, deren Version nicht validiert werden konnte, ist dies nicht zwangsläufig ein Indiz, dass die Version fehlerhaft ist. Es ist eher als eine fehlende zusätzliche Überprüfung der erkannten Version zu verstehen.

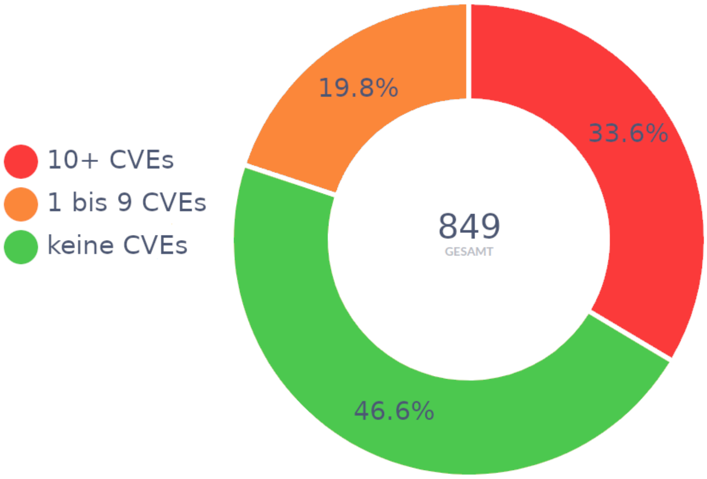


Abb. 14: Prozentualer Anteil der erkannten TYPO3-Installationen in Abhängigkeit von der Anzahl bekannter Schwachstellen

Nachdem die Version einer TYPO3 Instanz ermittelt wurde, kann die Webseite anhand dieser Version mit CVEs in Zusammenhang gebracht werden. Die Abbildung 14 beschreibt dabei die prozentuale Anzahl aus den 849 analysierten Webseiten, die von CVEs betroffen sind. So sind von den 849 Webseiten nur 396 durch ihre Version von keinem CVE betroffen. Im Gegensatz dazu sind jedoch 453 Webseiten von mindestens einem bis maximal 25 CVEs betroffen. Dadurch wird die Relevanz des in dieser Arbeit behandelten Themas gut verdeutlicht, da über die Hälfte der analysierten Webseiten bekannte Sicherheitslücken aufweisen, die potenziell von Angreifern ausgenutzt werden könnten. Wie bereits erwähnt kann der Algorithmus keine ELTS Versionen erkennen, weshalb maximal die letzte LTS Version von einer Major-Version von TYPO3 erkannt wird. Für die Major-Version 7 und 8 ist die Betrachtung besonders wichtig, da dort die jeweilige letzte LTS Version bekannte CVEs hat, diese Major-Versionen jedoch noch ELTS Versionen erhalten, die diese Sicherheitslücken nicht mehr beinhaltet. Da jedoch nicht ausgeschlossen werden kann, dass eine nicht aktualisierte Version verwendet wird, wird für diese Abbildung angenommen, dass die erkannte Version der tatsächlichen entspricht. Geht man hingegen davon aus, dass sämtliche der betroffenen Webseiten auf die aktuellste ELTS-Version aktualisiert wurden, so wären zumindest noch 363 der analysierten Webseiten von mindestens einem CVE betroffen.

Neben den 849 analysierten Webseiten gab es auch 97 Webseiten, die nicht analysiert werden konnten, da unter anderem die für die Analyse benötigten Ressourcen nicht abrufbar waren. Häufig werden die Routen für die benötigten Ressourcen durch einen Reverse Proxy geschützt, der Login Daten bei einem Zugriff auf die Route einfordert oder nur Zugriffe aus einem bekannten Netzwerk erlaubt. Ein weiterer möglicher Grund, weshalb eine Analyse fehlschlagen kann, ist das Verschieben der Routen, das ebenfalls durch einen

Reverse Proxy geschehen kann. Der letzte Grund eines Fehlschlags kann eine sehr stark veraltete TYPO3 Webseite sein. So ist der Algorithmus nicht in der Lage Versionen vor 4.4.0 zu erkennen, was auf zu wenig Versionscharakteristiken in der Datenbank für diese alten Versionen zurückzuführen ist. Da diese Versionen bereits so stark veraltet sind, ist die Erkennung des Patch-Levels nicht von Belang, da alle Versionen in diesem Bereich bekannte Sicherheitslücken aufweisen.

7 Zusammenfassung

In dieser Arbeit werden diverse Aspekte der Versionserkennung von CMS besprochen. Dabei werden unterschiedliche, umfangreiche Konzepte vorgestellt. Eines dieser Konzepte behandelt das automatisierte Erstellen von Versionscharakteristiken, bei dem über alle Versionen eines CMS systematisch anhand eines Versionsverwaltungssystems iteriert wird und Versionscharakteristiken extrahiert und abgespeichert werden. Das Konzept funktioniert mit leichten Anpassungen auch für andere Arten von Versionsrepositories, beispielsweise SVN oder Mercurial, und sogar für Quellcode der in Form von Dateiarchiven vorliegt. Das zweite vorgestellte Konzept baut auf das Erste auf und nutzt die erstellten Versionscharakteristiken, um an einer CMS-Instanz die exakte Version bis auf das Patch-Level genau zu bestimmen. Das letzte in dieser Arbeit vorgestellte Konzept befasst sich mit dem Bereitstellen der aus den anderen Konzepten gewonnenen Informationen, sodass diese sicherheitskritischen Informationen nur Personen erreicht, die die Webseite betreiben, da ansonsten Angreifer die ermittelte Version samt den dazugehörigen CVEs ausnutzen könnten. Die prototypische Implementierung der ersten beiden Konzepte wird am Beispiel von TYPO3 im Abschnitt 5 beschrieben. Das Konzept der automatisierten Versionscharakteristik Erstellung ist durch einen Git Crawler umgesetzt worden, der das Git Repository von TYPO3 automatisch durchsucht und die benötigten Informationen in eine Datenbank ablegt. Anhand dieser Daten analysiert das Versionserkennungstool für TYPO3 eine angegebene Webseite und stellt damit die Umsetzung des Konzepts zur Versionserkennung anhand von Versionscharakteristiken dar. Das letzte Konzept wurde nicht implementiert, da es für die Auswertungen nicht relevant ist. Neben der Implementierung der Konzepte wurde ebenfalls ein Tool geschaffen, dass bekannte CVEs zu TYPO3 in die Datenbank einfügt und mit den dazugehörigen Versionen von TYPO3 in Verbindung bringt. Anschließend wurde das implementierte Tool anhand einer Testumgebung und Webseiten im Abschnitt 6 getestet. Dabei wurde herausgefunden, dass bei 849 analysierten Webseiten über die Hälfte von CVEs betroffen sind.

Die Forschungsfrage **F1** kann dank des Git Crawlers und des Konzepts zur Versionserkennung beantwortet werden. Die Informationen, die aus einem VCS entnommen werden können, sind die öffentlich erreichbaren Ressourcen, die für eine Versionsbestimmung benötigt werden. Dazu gehören hauptsächlich CSS und JavaScript Dateien. Jedoch gibt es dabei einige Limitierungen, wodurch nicht alle Informationen des VCS genutzt werden können. So können beispielsweise keine dynamisch generierten Dateien erfasst werden, ohne den Quellcode sehr genau zu analysieren. Zudem ist unklar, ob diese Dateien gefunden werden können, da häufig die Route auch dynamisch generiert wird und in einigen Fällen von der Zeit und dem Inhalt der Datei abhängt. Eine weitere Limitierung ist, dass nur öffentliche Ressourcen abgefragt werden können, welche bei CMS häufig nur ein geringer Teil der gesamten Ressourcen sind. Des weiteren gibt es die Limitierung, dass aus dem VCS ebenfalls nur durch aufwändige Analyse des Quellcodes das Verhalten von verschiedenen Versionen in Grenzfällen ermittelt werden kann.

Aus den Ergebnissen der Arbeit und durch das Klären der Forschungsfrage **F1** kann nun die Forschungsfrage **F2** beantwortet werden. Wie im Abschnitt 6 gezeigt wurde, ist es möglich die Version einer CMS Instanz, in diesem Fall von TYPO3, bis auf das Patch-Level genau mit den Informationen aus **F1** zu bestimmen. Dies ist jedoch nicht bei jeder Version möglich, da sich in manchen Patch-Versionen nur nicht öffentliche Ressourcen ändern und somit der Unterschied nicht erkannt werden kann. In diesem Fall ist das Ergebnis zumindest für TYPO3 aussagekräftig genug, da häufig nur kleine Versionsbereiche entstehen. Jedoch kann es passieren, dass keine Version erkannt wird, wenn die benötigten öffentlichen Ressourcen von beispielsweise einem Reverse Proxy blockiert werden. Die Arbeit konnte somit beide Forschungsfragen beantworten und zeigt ein mehrstufiges Konzept auf, das die Erkennung von Versionen bei CMS ermöglicht.

8 Ausblick

Das in dieser Arbeit entwickelte Konzept der Versionserkennung kann in der Zukunft erweitert werden oder für andere Arbeiten genutzt werden. Um einen kleinen Ausblick zu bieten werden in diesem Abschnitt einige dieser Möglichkeiten aufgezählt und kurz diskutiert.

Die wohl offensichtlichste Erweiterung der Arbeit ist das Anwenden des erstellten Konzepts auf andere CMS. Prinzipiell ist das Konzept auf alle CMS anwendbar, sofern eine vollständige Versionshistorie des Systems vorhanden ist und darin statische Dateien enthalten sind, die öffentlich abrufbar sind. Da viele CMS solche statischen Assets mit sich bringen, kann das Konzept dort ebenfalls angewendet werden. Open Source CMS eignen sich am besten, da diese häufig alle Versionen öffentlich in ihrem jeweiligen Versionskontrollsystem zur Verfügung stellen. Beispiele für Open Source CMS, die öffentlich aufrufbare Webassets mitbringen, sind Joomla! und WordPress. Die Umsetzung des Konzepts muss dabei für jedes CMS erneut evaluiert werden.

Ein weiterer möglicher Ansatzpunkt für zukünftige Arbeiten ist die Verbesserung der Versionscharakterstika. Wie bereits in der Antwort auf die Forschungsfrage **F1** erwähnt, können dynamisch generierte Dateien bei einem VCS aktuell nicht betrachtet werden. Durch die Erfassung dieser könnte die Ermittlung der Version eines CMS gegebenenfalls schneller und präziser ablaufen. Somit wäre es eventuell möglich Versionen, die bislang nur ungenau erkannt wurden (Versionsbereiche), genauer zu bestimmen.

weiterer interessanter Ansatz ist das automatisierte Informieren der Betreiber veralteter CMS-Instanzen. Dafür müsste ein Webcrawler geschaffen werden, der automatisiert TYPO3 Webseiten im Internet erfasst. Anschließend wäre das in dieser Arbeit geschaffene Tool in der Lage die Version zu erkennen. Sollte die Webseite eine veraltete Version von TYPO3 benutzen, muss automatisch eine Mail an den Betreiber geschickt werden. Neben der Möglichkeit das automatisierte Versenden von E-Mails an die Betreiber selbst zu implementieren, könnten auch bestehende Infrastrukturen genutzt werden. So wäre für diesen Zweck beispielsweise eine Zusammenarbeit mit dem CERT-Bund vom BSI möglich, der anhand der IP-Adresse den Provider automatisch informiert. Der Provider leitet die Informationen anschließend an die Betroffenen weiter [69].

Die letzte Möglichkeit, die in diesem Ausblick betrachtet wird, ist das Erheben von Statistiken. Mithilfe des geschaffenen Tools und durch die Übertragung des Konzepts auf weitere CMS können eine große Menge von Webseiten analysiert werden und deren Updateverhalten protokolliert und ausgewertet werden. Dabei können beispielsweise Studien durchgeführt werden, bei denen Betreiber auf unterschiedliche Arten oder in unterschiedlichen Zeitintervallen über ihre veraltete Version informiert werden. Anschließend kann durch stetiges Testen der Version analysiert werden, wie lange ein Betreiber benötigt, um die veraltete Version zu erneuern oder ob die Webseite überhaupt aktualisiert wird.

9 Literaturverzeichnis

[1] Kritische ’Log4Shell’ Schwachstelle in weit verbreiteter Protokollierungsbibliothek Log4j (CVE-2021-44228). 22. Dezember 2021. URL: https://www.bsi.bund.de/SharedDocs/Cybersicherheitswarnungen/DE/2021/2021-549177-1032.pdf (besucht am 11. 01. 2022).

[2] Mirko Dölle. Sicherheitslücke Log4Shell: Internet in Flammen. 31. Dezember 2021. URL: https://www.heise.de/news/Sicherheitsluecke-Log4Shell-Internet-in-Flammen-6304730.html (besucht am 11. 01. 2022).

[3] Bundeskriminalamt. Cybercrime Bundeslagebild 2020, 2021.

[4] Angriffsziel deutsche Wirtschaft: mehr als 220 Milliarden Euro Schaden pro Jahr. 5. August 2021. URL: https://www.bitkom.org/Presse/Presseinformation/Angriffsziel-deutsche-Wirtschaft-mehr-als-220-Milliarden-Euro-Schaden-pro-Jahr (besucht am 15. 02. 2022).

[5] Usage statistics of content management systems. URL: https://w3techs.com/technologies/overview/content_management (besucht am 12. 01. 2022).

[6] TYPO3 Hosting für Agenturen, Freelancer & Unternehmen. URL: https://www.mittwald.de/cms-hosting/typo3-hosting (besucht am 15. 02. 2022).

[7] AWS Marketplace: t3rrific.com. URL: https://aws.amazon.com/marketplace/seller-profile?id=3c5e5f3c-d60e-4405-a9ca-aae8abfa3e2b\&ref=dtl_B07W2ZPW5P (besucht am 15. 02. 2022).

[8] TYPO3 Cloud Hosting, TYPO3 Installer, Docker Container and VM. URL: https://bitnami.com/stack/typo3 (besucht am 15. 02. 2022).

[9] Torben Hansen. TYPO3 Version Check - Statistic details. 8. Januar 2021. URL: https://www.t3versions.com/statistics-detail/15 (besucht am 11. 01. 2022).

[10] Joomla! Developer Network™: Usage Statistics. URL: https://developer.joomla.org/about/stats.html (besucht am 11. 01. 2022).

[11] Statistiken | WordPress.org Deutsch. 11. Januar 2021. URL: https://de.wordpress.org/about/stats/ (besucht am 11. 01. 2022).

[12] Nextcloud Security Scan. URL: https://scan.nextcloud.com/ (besucht am 12. 01. 2022).

[13] WPScan WordPress Security Scanner. URL: https://wpscan.com/wordpress-security-scanner (besucht am 12. 01. 2022).

[14] Hyeob Kim, HyukJun Kwon und Kyung Kyu Kim. Modified cyber kill chain model for multimedia service environments. *Multimedia Tools and Applications*, 78(3):3153–3170, 2019. DOI: 10.1007/s11042-018-5897-5.

[15] Jörg Thoma. Schwere Sicherheitslücke in Drupal 7. 16. Oktober 2014. URL: https://www.golem.de/news/security-schwere-sicherheitsluecke-in-drupal-7-1410-109890.html (besucht am 12. 01. 2022).

[16] Jörg Thoma. Drupal-Team warnt erneut vor Folgen. 30. Oktober 2014. URL: https://www.golem.de/news/sicherheitsluecke-drupal-team-warnt-erneut-vor-folgen-1410-110209.html (besucht am 12. 01. 2022).

[17] CVE - Search Results. URL: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=WordPress (besucht am 15. 02. 2022).

[18] Hanno Böck. Wordpress-Sicherheitslücke ermöglicht Änderung von Inhalten. 2. Februar 2017. URL: https://www.golem.de/news/sicherheitsluecke-wordpress-sicherheitsluecke-ermoeglicht-aenderung-von-inhalten-1702-125961.html (besucht am 12. 01. 2022).

[19] Oliver Hader. TYPO3-CORE-SA-2021-014: Cross-Site-Request-Forgery in Backend URI Handling. 5. Oktober 2021. URL: https://typo3.org/security/advisory/typo3-core-sa-2021-014 (besucht am 15. 02. 2022).

[20] L. Rabe. Anzahl der mit TYPO3 erstellten Internetseiten weltweit bis Oktober 2021. 27. Oktober 2021. URL: https://de.statista.com/statistik/daten/studie/321009/umfrage/weltweite-anzahl-der-mit-typo3-erstellten-internetseiten/ (besucht am 12. 01. 2022).

[21] How the Web works - Learn web development | MDN. 25. Dezember 2021. URL: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/How_the_Web_works (besucht am 04. 01. 2022).

[22] NoScript. URL: https://noscript.net/ (besucht am 13. 01. 2022).

[23] B.A. Mah. An empirical model of HTTP network traffic. In *Proceedings of INFOCOM '97*, Band 2, Seiten 592–600. IEEE Comput. Soc. Press, 1997. DOI: 10.1109/INFCOM.1997.644510.

[24] Eric Rescorla. HTTP Over TLS, 2000. DOI: 10.17487/RFC2818. URL: https://www.rfc-editor.org/info/rfc2818.

[25] Proxy servers and tunneling. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Proxy_servers_and_tunneling (besucht am 04. 02. 2022).

[26] L. Rabe. Anzahl der Internetnutzer weltweit in den Jahren 2005 bis 2020 sowie eine Schätzung für 2021. 3. Dezember 2021. URL: https://de.statista.com/statistik/daten/studie/805920/umfrage/anzahl-der-internetnutzer-weltweit/ (besucht am 19. 01. 2022).

[27] Web Design and Applications - W3C. URL: https://www.w3.org/standards/webdesign/ (besucht am 07. 02. 2022).

[28] What is HTML? URL: https://www.w3schools.com/whatis/whatis_html.asp (besucht am 04. 02. 2022).

[29] HTML & CSS. URL: https://www.w3.org/standards/webdesign/htmlcss (besucht am 04. 02. 2022).

[30] Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification. URL: https://www.w3.org/TR/CSS22/selector.html (besucht am 04. 02. 2022).

[31] JavaScript Web APIs - W3C. URL: https://www.w3.org/standards/webdesign/script (besucht am 10. 02. 2022).

[32] Philip Ackermann. *JavaScript - Das umfassende Handbuch*, Band 2. Rheinwerk Computing, Bonn, 2020, Seiten 339–396.

[33] Brian Murray. Releases - Ubuntu Wiki. 21. Oktober 2021. URL: https://wiki.ubuntu.com/Releases (besucht am 13. 01. 2022).

[34] Semantic Versioning. URL: https://semver.org/ (besucht am 03. 01. 2022).

[35] Benni Mack. Release v11.0.0 · TYPO3/typo3 · GitHub. 22. Dezember 2020. URL: https://github.com/TYPO3/typo3/releases/tag/v11.0.0 (besucht am 13. 01. 2022).

[36] Oliver Hader. Release v10.4.21 · TYPO3/typo3 · GitHub. 21. September 2021. URL: https://github.com/TYPO3/typo3/releases/tag/v10.4.21 (besucht am 13. 01. 2022).

[37] CVE-2021-32767. 12. Mai 2021. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-32767 (besucht am 13. 01. 2022).

[38] git(1) - Linux manual page. URL: https://man7.org/linux/man-pages/man1/git.1.html (besucht am 15. 02. 2022).

[39] Development Roadmap for TYPO3 CMS. URL: https://typo3.org/cms/roadmap (besucht am 14. 02. 2022).

[40] Companies using TYPO3 and its marketshare. URL: https://enlyft.com/tech/products/typo3 (besucht am 16. 02. 2022).

[41] TYPO3 Is an Enterprise CMS That Helps to Connect With You. URL: https://typo3.org/cms/features (besucht am 16. 02. 2022).

[42] jweiland.net. TYPO3 Versionen im Überblick. 5. Oktober 2021. URL: https://jweiland.net/typo3/versionen-und-updates.html (besucht am 30. 11. 2021).

[43] TYPO3 Documentation Team. Directory structure — TYPO3 Explained main documentation. 24. November 2021. URL: https://docs.typo3.org/m/typo3/reference-coreapi/main/en-us/ApiOverview/DirectoryStructure/Index.html (besucht am 30. 11. 2021).

[44] Neal R. Wagner. Fingerprinting. In *1983 IEEE Symposium on Security and Privacy*, Seiten 18–18. IEEE, 1983. DOI: 10.1109/SP.1983.10018.

[45] Peter Eckersley. How Unique Is Your Web Browser? In *Privacy Enhancing Technologies*, Seiten 1–18. Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-14527-8.

[46] Naoki Takei, Takamichi Saito, Ko Takasu und Tomotaka Yamada. Web Browser Fingerprinting Using Only Cascading Style Sheets. In *10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*, Seiten 57–63. IEEE, 2015. DOI: 10.1109/BWCCA.2015.105.

[47] Elbert Alias. GitHub - AliasIO/wappalyzer: Identify technology on websites. 8. Dezember 2021. URL: https://github.com/AliasIO/wappalyzer (besucht am 08. 12. 2021).

[48] Dustin Lee, Jeff Rowe, Calvin Ko und Karl Levitt. Detecting and defending against Web-server fingerprinting. *Proceedings - Annual Computer Security Applications Conference, ACSAC*:321–330, 2002. DOI: 10.1109/CSAC.2002.1176304.

[49] OWASP JoomScan Project. URL: https://github.com/OWASP/joomscan (besucht am 14. 01. 2022).

[50] Netcraft | Internet Research, Cybercrime Disruption and PCI Security Services. URL: https://www.netcraft.com/ (besucht am 08. 12. 2021).

[51] Nmap: the Network Mapper - Free Security Scanner. URL: https://nmap.org/ (besucht am 08. 12. 2021).

[52] Vítor Bernardo und Dulce Domingos. Web-based Fingerprinting Techniques. In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications*, Band 4, Seiten 271–282. SCITEPRESS - Science und Technology Publications, 2016. DOI: 10.5220/0005965602710282.

[53] Kevin Lamshöft. *Customer Tracking - Grade von Anonymität im Internet: Canvas Fingerprinting und intrinsische Hardwaresignaturen*. Bachelor Thesis, 2015.

[54] Pascal Wichmann. *Automated Inference of Web Software Packages and Their Versions*. Bachelor Thesis, 2018.

[55] Pascal Wichmann. VersionInferrer. 13. April 2021. URL: https://github.com/wichmannpas/VersionInferrer (besucht am 01. 12. 2021).

[56] Christian A. Gorke und Frederik Armknecht. Reverse Fingerprinting. *CoRR*, abs/1912.09734, 2019. arXiv: 1912.09734.

[57] Alfred Berg und Norton Lamberg. Automatic fingerprinting of websites, Degree Project, 2020.

[58] Christian Dresen, Fabian Ising, Damian Poddebniak, Tobias Kappert, Thorsten Holz und Sebastian Schinzel. CORSICA: Cross-Origin Web Service Identification. *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*:409–419, 2020. DOI: 10.1145/3320269.3372196.

[59] Fabian Marquardt und Lennart Buhl. Déjà Vu? Client-Side Fingerprinting and Version Detection of Web Application Software. In *IEEE 46th Conference on Local Computer Networks (LCN)*, Seiten 81–89, 2021. DOI: 10.1109/LCN52139.2021.9524885.

[60] File:Axios logo (2020).svg - Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:Axios_logo_(2020).svg (besucht am 11. 02. 2022).

[61] Docker Logos | Docker. URL: https://www.docker.com/company/newsroom/media-resources (besucht am 11. 02. 2022).

[62] File:Git icon.svg - Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:Git_icon.svg (besucht am 11. 02. 2022).

[63] GitHub - guzzle/guzzle: Guzzle, an extensible PHP HTTP client. URL: https://github.com/guzzle/guzzle (besucht am 11. 02. 2022).

[64] File:Unofficial JavaScript logo 2.svg - Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:Unofficial_JavaScript_logo_2.svg (besucht am 11. 02. 2022).

[65] Datei:MySQL logo.svg – Wikipedia. URL: https://de.wikipedia.org/wiki/Datei:MySQL_logo.svg (besucht am 11. 02. 2022).

[66] File:Node.js logo.svg - Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:Node.js_logo.svg (besucht am 11. 02. 2022).

[67] Datei:PHP-logo.svg – Wikipedia. URL: https://de.wikipedia.org/wiki/Datei:PHP-logo.svg (besucht am 11. 02. 2022).

[68] JavaScript Web APIs - W3C. 15. Juni 2015. URL: https://github.com/TYPO3/typo3/commit/d73a05d2bbdf54bc5294b6e41360e21e6d7929c3 (besucht am 10. 02. 2022).

[69] BSI - CERT Bund Reports. URL: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Cyber-Sicherheitslage/Reaktion/CERT-Bund/CERT-Bund-Reports/cert-bund-reports_node.html (besucht am 08. 02. 2022).

Selbständigkeitserklärung

Hiermit erkläre ich, Jann-Marten Kias, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Jann-Marten Kias
Magdeburg, den 17. Februar 2022

Anhang

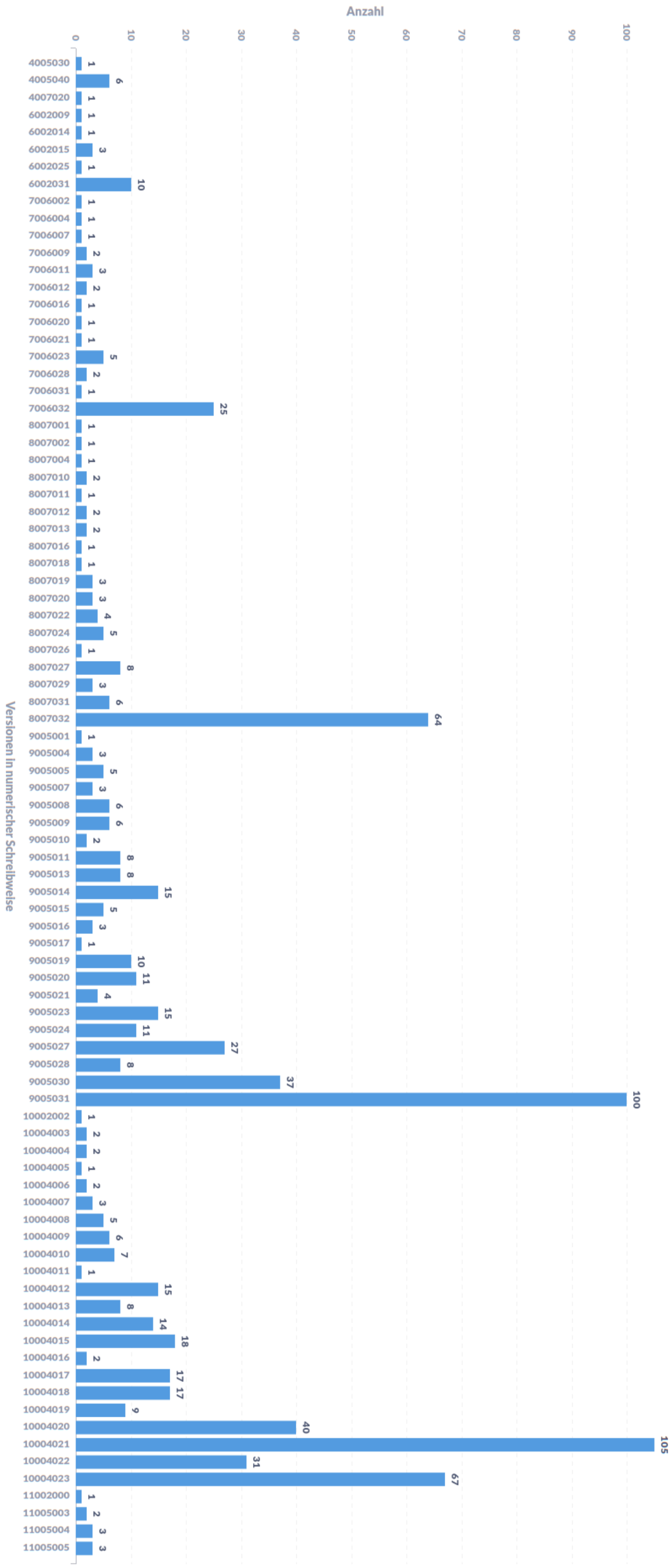


Abb. 15: Anzahl der gefundenen TYPO3-Versionen der 849 Webseiten in numerischer Version