Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-100241-103018

Jakub Kuska

Operačný systém ako web CMS (WCM) – Volanie funkcie operačného systému cez web

Bakalárska práca

Vedúci práce: Ing. Gabriel Szabó

Bratislava 2023

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

FIIT-100241-103018

Jakub Kuska

Operačný systém ako web CMS (WCM) – Volanie funkcie operačného systému cez web

Bakalárska práca

Študijný program: Informatika

Študijný odbor: Informatika

Miesto vypracovania: Ústav počítačového inžinierstva a aplikovanej informatiky

Vedúci práce: Ing. Gabriel Szabó

Bratislava 2023 **Jakub Kuska**

Annotation

Slovak University of Technology Bratislava

Faculty of Informatics and Information Technologies

Degree Course: Informatika

Author: Jakub Kuska

Diploma Thesis: Operačný systém ako web CMS (WCM) – Volanie funkcie operačného systému cez web

Supervisor: Ing. Gabriel Szabó

Bratislava 2023

Anotácia

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Študijný program: Informatika

Autor: Jakub Kuska
Bakalárska práca: Operačný systém ako web CMS (WCM) – Volanie funkcie operačného systému cez web
Vedúci bakalárskej práce: Ing. Gabriel Szabó
Bratislava 2023

Znenie zadania bakalárskej práce

Operačné systémy sú komplexné systémy, ktoré obsahujú hotové a funkčné riešenia pre rôzne typy úloh, ktoré sú bežne riešené aj v rámci WCM systémov. Typickými príkladmi je manažment používateľov, alebo zdielaný prístup k dátam. Cieľom tohto zadania je vystaviť určité funkcie operačného systému na báze Linuxu cez webové rozhranie. Porovnajte rôzne spôsoby orchestrácie a hostovania dynamických webových služieb na operačných systémoch Linux. V analýze sa sústreďte aj na rozhrania medzi webovým serverom a samotnou webovou aplikáciou (CGI, FastCGI, SCGI, WSGI) aj na webové frameworky s už integrovaným webovým serverom (napr. typycké Java technológie napr. cez servlety, EJB, Spring, Quarkus. Javascriptové knižnice ako napr. Node.js., Groovy on Grails a pod.). Navrhnite a implementujte demo aplikáciu z vybratej podmnožiny jednotlivých technológií, ktorá sprístupní základné informácie o stave počítača, na ktorom daná aplikácia bola spustená. Na opis stavu použite základné ukazovatele vyťaženosti počítača, ako napr. aktuálna vyťaženosť pamäti, procesora a volné miesto na disku. Vytvorené aplikácie porovnajte z pohladu výkonnosti, záťaže na systém, bezpečnosti, schopnosti vykonávať príkazy na danom operačnom systéme. Na základe zistených výsledkov vyhodnoťte optimálny spôsob pre orchestráciu systémových volaní cez webové rozhranie.

Obsah

Kapitola 1

Analýza problematiky

1.1 WCM/CMS systémy vo všeobecnosti

V tejto časti si priblížime čo si môžeme predstaviť pod pojmom WCM/CMS. WCM/CMS pomáhajú používateľom spravovať, kontrolovať, meniť a teda vo všeobecnosti nejakým spôsobom manažovať obsah webstránky, prípadne aplikácie.[6]

Dáta používateľov ako text a metadáta k napríklad obrázkom, súborom a podobne sú vo väčšine prípadov ukladané do databáz, z ktorých je následne pomocou programovacích jazykov na to určených generovaný obsah webu.

Hlavou výhodou WCMS systémov má byť hlavne jednoduchá interakcia t.j. aj používatelia bez znalosti programovacích jazykov by mali vedieť s takýmto systémom jednoducho interagovať a dosiahnuť ich cieľ, či už sa jedná o napísanie článku pri blogu alebo zdieľanie súboru s kamarátom pri banke súborov.

1.1.1 Schopnosti a možnosti WCMS systémov

Je dôležité si zadefinovať čo by mal taký WCMS systém vedieť, v našom prípade sa bude jednať o interakciu s operačným systémom Linux presnejšie preklad systémových volaní, správa súborov a ich opravnení v kombinácií so správou používateľov. Keď máme tieto požiadavky zadefinované, je potrebné sa na jednotlivé požiadavky pozrieť bližšie.

Mimo iného je nutné sa zamyslieť aj nad aplikačnou vrstvou samotného systému t.j. vykreslovanie / aktualizovanie dát na prezentačnej vrstve

1.1.1.1 Cenová dostupnosť

Asi ako v každom aspekte života aj pre nás je pri práci s WCMS systémom dôležitá jeho cena respektíve náklady na prevádzku. Pozitívom je, že aktuálne existuje mnoho open source implementácií WCMS riešení, ktoré majú obrovskú komunitu, čo z týchto riešení vytvára veľmi vyzretý a použiteľný produkt aj pre enteprise riešenia. Ako najznámejšie open source riešenia by sme si dovolili spomenúť WordPress, Typo3, Joomla a podobne.[0]

1.1.1.2 Riadenie prístupu

Väčšina WCM systémov v dnešnej dobe podporuje niektorý z druhov riadenia prístupu. Riadenie prístupu je potrebné na vytvorenie akejsi abstrakcie nad systémom opravnení - v skratke povedané, ponúka systém, ktorý je schopný vytvárať pravidlá a oprávnenia pre používateľov, ktorý WCM systém používajú. Príklad môže byť práve naša aplikácia, kde používateľ A chce zazdieľať súbor prípadne priečinok používateľovi B avšak nikomu inému.

Tu prichádza systém riadenia prístupu, ktorý požiadavku preloží a na pozadí nastaví priečinku alebo súboru, ktorý sa snaží používateľ A zdielať s používa-

teľom B správne oprávnenia, pričom obaja používatelia sú schopní tieto oprávnenia manažovať pomocou používateľského rozhrania.

1.1.1.3 Škálovateľnosť

Škálovateľnosť je veľmi dôležitým aspektom WCM systémov nakoľko ako vývojári by sme chceli aby bol náš systém jednoduche distribuovateľný koncovým zákaznikom. V praxi to teda znamená, že náš systém obsahuje určitý druh inštalátora, ktorý nastaví jednotlivé parametre na základe požiadaviek koncového používateľa, do parametrov môže patriť napríklad IP adresa administrácie, backend servery, názov aplikácia a podobne.[0]

1.1.1.4 Používateľská prívetivosť

Opäť si uvedieme príklad na našom systéme a síce, predstavme si, že chceme zistiť aktuálne využitite zdrojov servera, na ktorom náš WCM systém beží.

Za normálnych okolností by sme sa museli pracne na server pripájať pomocou napríklad protokolu SSH alebo RDP a využitie zdrojov preveriť pomocou zabudovaných nástrojov na to určených.

Avšak ak pomocou rozhrania a skriptov využitie systému vystavíme napríklad vo formáte JSON cez našu API, môžeme veľmi jednoducho vizualizovať využitie zdrojov priamo v používateľskom rozhraní našej aplikácie, čo značne zjednodušuje prácu so serverom.[0]

1.1.1.5 Bezpečnosť

Nakoľko interekacia so serverom prebieha cez vyvedenú API je možné veľmi pekne implementovať rôzne druhy WAF, čo nám značne pomôže koncový server zabezpečiť. Ideálne nechceme používateľom dávať priamy prístup k systémovým

volania OS ale skôr je cestou vytvoriť öbal"medzi identickými príkazmi v WCM a serverom.

Malo by to znamenať asi len toľko, že napríklad príkaz **mv** vo WCM prejde cez prekladovú vrstvu na backende, ktorá overí, či je používateľ autorizovaný príkaz spúšťať v aktuálnom kontexte ak áno, správa prejde ešte cez jednu prekladovú vrstvu, na ktorej beží WAF - tento overí, či správa resp. príkaz neobsahuje žiadne potencionálne nebezpečné dáta.[0]

1.1.2 Typy WCM/CMS systémov

Systémy WCM môžu byť implementované troma základnými spôsobmi, ktoré opisujú kedy sa deje vykresľovanie vygenerovaného obsahy koncovému používateľovi zo štruktúrovaného obsahu, ktorý bol vygenerovaný na základe vzťahov z databázy.

Jedná sa o tieto tri:

- **Offline processing**
- **Online processing**
- **Hybrid processing**

1.1.2.1 Offline processing

Offline processing systémy sú založené na predkreslení obsahu, ktorý bude následne prezentovaný koncovému používateľovi, sú známe aj ako "SSG"(static site generators)[4]

1.1.2.2 Online processing

Podľa názvu je zrejmé, že tento druh WCMS systémov vykonáva takzvané ön demand"vykresľovanie obsahu a teda prekresľuje len také časti webu, ktoré reálne prekreslenie potrebujú, zvyšok obsahu je zväčša prezentovaný z vyrovnávacej pamäte cache.

1.1.2.3 Hybrid processing

Hybridné WCMS systému su kombináciou offline a online systémov, kedy sú určité druhy predgenerované ešte pred prezentáciou, napríklad spustiteľný kód PHP, Java, ASP .NET.

1.1.3 Výhody WCM/CMS systémov

1.1.3.1 Cenová dostupnosť

V dnešnej dobe existuje mnoho skvelých a hlavne schopných riešení WCM/CMS systémov, ktoré su open-source a teda ich prevádzka nestojí ani euro. Tento fakt vie byť kľúčový, hlavne ak chceme pre náš projekt využiť už existujúce riešenie, ktoré bolo vyvinuté podľa metód vývoja softvéru a teda báza zdrojového kódu je väčšinou prehľadne koncipovaná a taktiež je vo väčšine prípadov existujúcich open source riešení systém navrhnutý tak, že je preň veľmi jednoduché vyvíjať nové moduly a rozšírenia, či už sa týkaju funkčných alebo dizajnových aspektov systému.

1.1.3.2 Jednoduchá správa a použiteľnosť

WCM/CMS systémy majú vo všeobecnosti vytvorený univerzálnu "šablónu", čo zabezpečuje, že systém má uniformný vzhľad aj na rôznych stránkach a podstránkach.

V súčasnosti si do sveta WCM systémov prerážajú aj nové progresívne webové frameworky ako Vue.JS, React a iné, ktoré sú ako stvorené pre systémy, ktoré sú založené na komponentoch, ktoré ako celok tvoria finálny obsah webu.

1.1.3.3 Prehľadná správa obsahu

Hlavnou ideou pri systémoch, ktoré majú slúžiť na správu obsahu webu alebo teda v našom prípade na správu serveru OS Linux je hlavná požiadavka kladená na práve použiteľnosť a správny dizajn ovládania správy webového obsahu. Čo to však znamená?

Znamená to asi len toľko, že UI a UX dizajn by mal byť koncipovaný tak, aby používateľ, ktorý bude systém ovládať mal najdôležitejšie prvky jasne viditeľné a aby bolo zrejmé čo tieto vykonávajú.

Čo sa týka rozšírených funkcií systému, tieto by mali byť prehľadne uložené do napríklad rozkladacieho zoznamu a v ňom rozdelené do kategórií

1.1.4 Nevýhody WCM/CMS systémov

1.1.4.1 Latencia systému

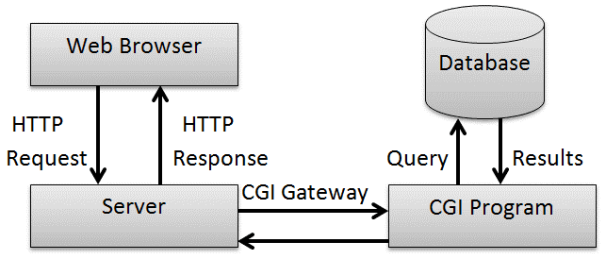
Väčšie inštalácie WCM systémov môžu trpieť veľkou latenciou medzi napríklad preklikmi medzi podstránkami.

Tento jav je v 90 percentách prípadov spôsobený odozvou databázového servera, kde systémová databáza nie je používaná správnym spôsobom, čo spôsbouje pomalé dopyty a teda celý systém čaká kým databázový systém pripraví odpoveď. Je preto veľmi dôležité, aby sme sa pri návrhu databázového systému vyhýbali zbytočným prepojom medzi tabuľkami, využívali pohľady a indexy, ktoré, ak su používané správne vedia tieto problémy eliminovať.

1.2 Rozhrania CGI (Common Gateway Interface)

CGI alebo teda Comman Gateway Interface popisuje štandard pre rozhranie, ktoré by malo prepájať externé aplikácie (CGI programy) s webservermi. Mimo iného taktiež definuje protokol, pomocou ktorého by táto komunikácia medzi externou CGI aplikáciou a webserverom malo priebiehať.

CGI protokol umožňuje webovým serverom spúšťať externé programy a následne odpoveď spracovať a podľa potreby zobraziť na webe. Protokol CGI má výborný medzi platformový výkon a je ho možné implementovať takmer na každom OS.



Obr. 1.1: Diagram pozície protokolu CGI v komunikácií externej aplikácie a webového serveru [2]

1.2.0.1 Princíp fungovania protokolu CGI

Protkol CGI pri detekovaní spojenia vytvorí subprocess CGI, následne sa aktivuje samotný proces, ktorý spracuje požiadavku a v poslednom rade ukončí vytvorený subproces. Tento dizajn sa nazýva **"Fork-And-Execute"** mód (požadový skript je spúštaný ako samostatný proces, nie ako vlákno existujúceho procesu)

Táto implementácia však má jednú zásadnú chybu respektíve nedostatok a síce, každá požiadavka vyustí vo vytvorenie subprocesu pomocou systémového

volania fork, čo môže v kombinácií s veľkým množstvom požiadaviek spôsobiť veľké spomalenia.

Workflow CGI aplikácie:

- **Cez prehliadač si klient vyžiada spustenie CGI aplikácie**
- **Server obdrží požiadavku**
- **Server spustí požadovaný CGI skript / aplikáciu**
- **CGI aplikácia vykoná požadované úkony na základe vstupu používateľa a odpoveď naformátuje do formátu, ktorému webserver rozumie**
- **Server vráti odpoveď používateľovi vo forme webovej prezentácie**

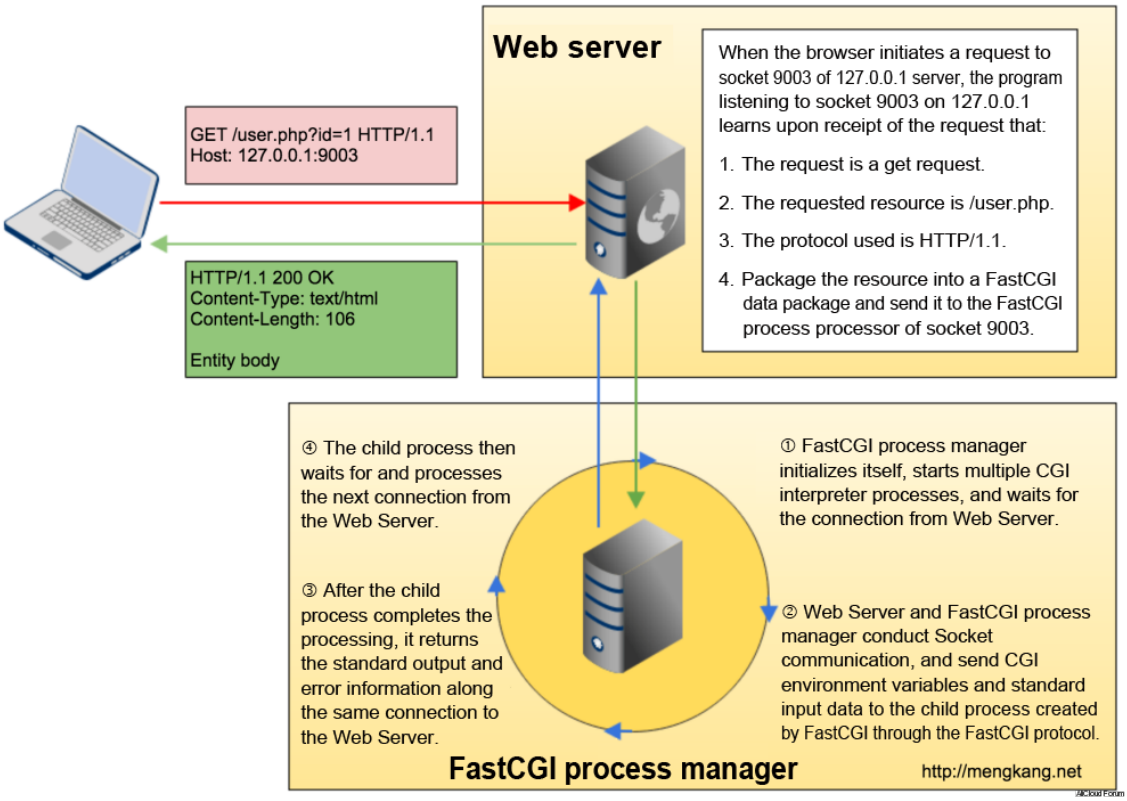
1.2.0.2 Protokol FastCGI

Ako zo samotného názvu vyplýva protokol alebo rozhranie FastCGI vylepšuje to, čo pôvodnému rozhraniu CGI chýbalo a čo boli jeho slabiny. FastCGI teda predstavuje vysoko-výkonné, škalovateľné rozhranie pre komunikáciu serverov HTTP s dynamickými skriptovacími jazykmi.

Ako sme spomenuli vyššie, hlavným problémom klasického rozhrania CGI bol hlavne jeho výkon, ktorý pri vyšších záťažách podstatne trpel a nebol schopný doručiť taký výkon, aký by sme si priali. Dôvodom je, že pri každom HTTP requeste na dynamický program CGI musel reštartovať skript parser a vyparsovať celý skript odznova a až následne bola odpoveď vrátená webserveru na prezentáciu.

Tento problém FastCGI rieši tak, že eliminuje potrebu vytvárať zakaždým nový proces, ktorý by spúštal predmetný CGI skript. Keďze si FastCGI pred-

pripravuje takzvaných workerov, ktorým prenencháva vykonávanie CGI skriptov, v reálnej aplikácií vieme vidieť až päťnásobné zlepšenie výkonu oproti klasickému CGI.



Obr. 1.2: Diagram principálneho fungovania protokolu FastCGI [2]

Ako opisuje aj obrázok vyššie, pricipiálne fungovanie protokolu FastCGI vieme opísať následovnými krokmi:

- **FastCGI modul je načítaný pri štarte webservera**
- **Manažér FastCGI je inicializovaný, pričom sa po inicializácií spustí niekoľko CGI interpreter procesov, ktoré čakajú na požiadavky, ktoré sú prerozdeľované manažérom**
- **Po obdržaní požiadvky je request manažérom zaslaný na vybra-**

ného workera, pričom sú spoločne s požiadavkou workerovi zaslané aj premenné prostredia

- **Všetky tieto dáta sú preposlané cgi procesu, ktorý ich spracuje, pričom samotný worker v tento moment môže spracovať ďalšiu požiadavku**
- **Po dokončení vykonávania skriptu cgi proces vráti odpoveď webserveru, ktorý ju spracuje a následne prezentuje používateľovi**

Keďže FastCGI funguje v zásade inak, než klasický CGI, umožňuje nám dokonca spúšťať CGI aplikácie mimo iného servera, čo pri klasickom CGI nebolo možné a bolo vyslovene nutné mať webserver a CGI aplikáciu na rovnakom serveri.

Tento fakt prináša značnú flexibilitu v nasadzovaní aplikácií, nakoľko vieme spraviť abstrakciu a to tak, že vytvoríme dva servery, kde jeden bude webserver, ktorý bude prijímať a delegovať požiadavky a druhý bude samotný compute, ktorý bude spúšťať CGI skripty a odpovedať s výsledkom webserveru.

1.2.0.3 WSGI (The Web Server Gateway Interface)

The Web Server Gateway Interface je jednoduché rozhranie medzi webserverom a webovou aplikáciou alebo frameworkom definovaným v jazyku Python.

WSGI je nadizajnovaný ako low-level rozhranie medzi webovým serverom a webovou aplikáciou resp. frameworkom. WSGI je založený na existujúcich CGI štandardoch.

Rozhranie WSGI má dve strany:

- **Server/Gateway**

- **Application/Framework**

[3]

Aplikačná strana je

1.3 Operačný systém Linux

1.3.0.1 Krátka história

Akonáhle sa vôbec pustíme do problematiky OS Linux, nebolo by zlé si priblížiť aspoň krátku históriu o tom, ako OS Linux ako taký uzrel svetlo sveta a čo ho urobilo takým, akým je teraz a prečo je taký rozšírený v enteprise prostredí.

Linux prvýkrát uzrel svetlo sveta v roku 1991 ako osobný projekt fínskeho študenta Linusa Torvaldsa, ktorý mal za cieľ vytvoriť nový open source kernel pre operačné systémy.

Keďže bol tento projekt od začiatku koncipovaný ako otvorený okamžite si našiel svojich priaznivcov a je udržiavaný dodnes. Linuxový kernel ako taký je zapísaný v programovacom jazyku C, avšak dnes sú už niektoré z jeho modulov a časť jadra zapísaná aj v jazykoch C++ a Rust.[5]

1.3.0.2 Distribúcie

Ako sme spomínali v histórií OS Linux, Linux ako taký je vlastne len kernel pre OS, čo vlastne znamená, že koncový používateľ dostáva len tej najhrubší základ pre fungovanie finálneho operačného systému.

Tu prichádzajú na radu distribúcie. Pod pojmom distribúcia si v kontexte OS Linux vieme predstaviť niečo ako "fork", kde používateľ A zoberie zdrojový kód linuxového kernelu, spraví úpravy podľa jeho potreby, vytvorí grafickú nadstavbu,

pripraví balíkovací nástroj a systém nazve napríklad "SuperOS", pričom podľa licencie linuxového kernelu aj taká distribúcia musí ostať open source.

To isté teda urobí aj používateľ B, avšak urobí úplne diametrálne zmeny oproti používateľovi A, a tým pádom vytvorí akúsi diverzitu a tým aj obrovskú slobodu koncového používateľa, ktorý si bude vyberať operačný systém, ktorý zodpovedá jeho požiadavkam.

Príkladom nám môže byť napr. OS Ubuntu, ktorý je s jeho grafickým rozhraním a jednoduchou správou vhodný aj pre začiatočníkov v OS Linux verzus napr. RHEL, ktorý je vyslovene koncipovaný na použitie v enterprise prostredí ako prostredník pri prevádzke produkčných systémov.

1.3.0.3 Systémové volania a ich implementácia

Systémovými volaniami v systéme Linux môže rozumieť akýsi vstupný bod do kernelu OS Linux. Systémove volania sú zvyčajne volané pomocou príslušných "wrapperov", ktoré sú zapísane v jazyku C a sú súčasťou základnej knižnice Glibc. Tieto funkcie, ktoré zaobalujú systémové volania zvyčajne nič naviac nerobia, avšak zabezpečujú správny sled udalosti, ktoré sa musia udiať na to, aby sme z user space priestoru mohli zavolať nejaké systémové volanie do kernelu.

V drvivej väčšine prípadov tieto obaľovacie funkcie robia len následovné úkony:

- Kopírujú argument a unikátne číslo systémového volania do registrov, v ktorých tieto argumenty kernel očakáva [1]
- Prepínajú sa do kernel módu, v ktorom kernel vykonáva reálne úkonu súvisiace z aktuálne vykonávaným systémovým volaním [1]
- Nastavujú premennú **errno**, ktorá obsahuje návratovú hodnotu posledného

vykonávaného programu [1]

Avšak v niektorých prípadoch tieto obaľovacie funkcie robia podstatne viac, než základne posielanie argumentov do registrov, napríklad predspracovanie a filtrovanie vstupných argumentov ako aj predspracovanie a formátovanie výstupných hodnôt.[1]

V niektorých prípadoch sa nám môže stať, že štandardná knižnica C neposkytuje peknú obaľovaciu funkciu pre systémové volanie, ktoré sa snažíme volať - v taktomto prípade môže systémové volanie volať aj priamo bez nutnosti využiť spomínané obaľovacie funkcie.

Ak by sme teda chceli zavolať systémové volanie priamo, vieme tak spraviť pomocou intergrovanej funkcie **syscall()**, ktorá pomocou číselného argumentu, ktorý označuje číslo systémového volania zavolá správne rozhranie v assembly. Tento assembly kód následne vykoná systémové volanie a vráti návratovú hodnotu.[1].

Podstatnou vecou, na ktorú netreba zabúdať je architektúra CPU, na ktorej sa snažíme volať systémové volania, rôzne architektúry majú implementované rozhranie na volanie systémových volaní inak. Spomenieme si teda tie najznámejšie a ako v nich systémové volania možno vykonávať.

1.3.0.4 Zabezpečenie v OS Linux

Je faktom, že popularita OS Linux v posledných rokoch začala stúpať enormným spôosobom. Tento fakt prináša do sveta linuxu pochopiteľne aj hrozby, nakoľko pre hackerov niekedy nezaujímavý terč sa dnes stáva čoraz zaujímavejším.

Je preto dôležité sa zamerať na bezpečnosť ako prioritu číslo jedna. Dôvodom je hlavne prevencia do budúcna, kde ak napríklad náš projekt začne byť považovaný za enterprise riešenie, nechceli by sme v ňom mať bezpečnostné diery ešte z čias prvotného vývoja, čo je, bohužial v dnešnej dobe čoraz častejším javom.

Ako sme už raz spomenuli, Linux je v dnešnej dobe terčom mnohých druhov škodlivého softvéru, či už sa jedná o malware, ransomware a podobne. Na druhej strane však existuje aj veľa možností ako sa týmto hrozbám ubrániť.

1. **Zabezpečenie pomocou repozitárov** - Jednou z najdôležitejších vecí pre používateľov OS Linux je byť obozretný aký softvér sťahujú. V Linuxových distribúciách existujú zvyčajne balíkovacie nástroje, pomocou ktorých vieme inštalovať nové balíčky z ich príslušných repozitárov. Tieto repozitáre sú výhodné v tom, že sa jedná o oficiálne repozitáre spravované komunitou vývojárov a overované správcami - môžeme si byť teda istí, že takéto repozitáre by nemali obsahovať žiaden malware a podobne. Nie je odporúčané sťahovať balíčky priamo z internetu (týka sa z časti aj zdrojových kódov)

2. **Používanie antivírusového softvéru** - Ďaľšou možnosťou ako vieme svoj linuxový systém ochrániť, je použitie antivírusového softvéru tak, ako je to napríklad na platforme Windows. Najznámejšie AV pre Linux sú napríklad ESET, ClamAV a iné. Väčšinou sú tie AV koncipované ako démony - bežia na pozadí ako binárka bez GUI. To je však možné v prípade ClamaAV zmeniť a nainštalovať si tento balíček aj s podporou GUI. Odtlačky hrozieb je potom možné manažovať priamo cez rozhranie.

3. **Aktualizácie systému a balíčkov** - Veľmi dôležitým aspektom bezpečnosti pri Linuxových distribúciach je mať vždy systém v čo najaktuálnejšom stave. Je to hlavne z dôvodu, že 70 až 90% [7] softvéru vyvinutého pre Linux

je open-source, čo okrem slobody pre používateľov prináša aj možnosť nahliadnuť hackerom do zdrojového kódu a nájsť potencionálne zraniteľnosti.

4. **Brány firewall** - aby sme mohli zamedziť útokom z rovnakej siete, vieme náš stroj ochrániť pomocou pravidiel brány firewall. Pod pojdom firewall si vieme predstaviť akúsi sadu pravidiel, na základe ktorých systém vykonáva filtrovanie premávky či už smerom dovnútra (ingress) alebo smerom von (egress). Firewall ako taký nás však pochopiteľne nevie ochrániť pred útokmi na jeden konkrétny port, ktorý chceme mať otvorený, príkladom je útok typu DDoS, kedy na napr. port 80 prichádza mnoho škodlivej premávky z rôznych zdrojov za cieľom zahltiť službu, ktorá na danom porte počúva. V taktomto prípade je vhodné nasadiť behaviorálny Anti-DDoS alebo iné podobné riešenie

5. **Správne oprávnenia na priečinky a súbory** - Za cieľom zamedziť ostatným používateľom serveru prístup k súborom, ku ktorým by nemali mať prístup ako sú privátne SSH kľúče, konfiguračné súbory ale aj dočasné súbory kritických programov, je potrebné správne nastaviť jednotlivé oprávnenia. OS Linux ako taký podporuje systém oprávnení kde ako používateľ, ktorý vlastní daný súbor alebo priečinok vie nastaviť kto môže súbor upravovať, spúštať, čítať alebo do neho vykonávať úpravy.

[8]

Keď máme tieto jednotlivé možnosti zabezpečenia zadefinované, je potrebné si uvedomiť fakt, že nikdy nám nebude stačiť len jedná metóda na dosiahnutie plného a dokonalého zabezpečenia systému.

Vždy je potrebné sa zamyslieť nad prípadom použitia stroja, ktorý sa snažíme zabezpečovať a hlavne jeho dostupnými zdrojmi. Na základe týchto ukazo-

vateľov vieme presne nakombinovať najvhodnejšie metódy zabezpečenia pre daný prípad použitia.

Kapitola 2

Opis riešenia

2.0.1 Špecifikácia projektu

Úlohou projektu bolo navrhnúť rozhranie na volanie systémových voláni OS Linux pomocou webového rozhrania a to tak, aby mal koncový používateľ možnosť pomocou jednoduchého ovládania cez používateľské rozhranie možnosť bezpečnym spôsobom vykonávať úpravy priamo na serveri bez nutnosti sa server pripájať cez protokol SSH, RDP a podobne.

V prvom rade je potrebné si zvoliť distribúciu OS Linux, na ktorú budeme náš projekt vyvíjať, máme viacero možností, pokiaľ chceme ísť cestou Debian based distribúcií máme na výber samotný Debian, Ubuntu alebo MX Linux. Taktiež máme na výber viac enterprise based RHEL systémov ako napríklad Rocky Linux, RHEL (nie je zdarma), Oracle Linux alebo Fedora. V našom prípade by sme zvolili Debian based distribúciu Ubuntu 22.04 LTS, ktorá je najnovším prírastkom do rodiny LTS rodiny Ubuntu, dôvodom je hlavne osobná preferencia v kombinácií s nie príliš zložitým či už user manažmentom alebo bezpečnostnými politikami.

Cieľom je využiť protokoly CGI, FastCGI alebo uWSGI, ktoré su priamo šité na tieto účely. Z týchto protokolov respektíve nadstavieb protokolu CGI je úlohou vybrať pre nás ten správny a implementovať spomínané rozhranie, ktoré bude vykonávať preklad príkazov z webu na systémové volania systému OS Linux.

V našom prípade by sme radi išli cestou uWSGI nakoľko tento je priamo vyvinutý pre skriptovací jazyk Python, ktorý by sme vďaka jeho flexibilite využili pre backend resp. prekladovú vrstvu.

Taktiež je potrebné si správne zvoliť webovú technológiu, ktorá bude použitá na doimplementáciu webovú časť našeho rozhrania. Na medziprocesnú komunikáciu existuje viacero rozšírených protokolov ako napríklad XML, JSON a iné, pomocou ktorých vieme prenášať metadáta a telo príkazov, ktoré by sme chceli preložiť.

Na medziprocesnú komunikáciu, teda web - backend, by sme ostali pri JSON, ktorý je dnes už štandardom a či už Python alebo PHP / JavaScript poskytuje öut-of-the-box"knižnice na parsovanie dát vo formáte JSON.

Čo sa webových technológií týka, je to hlavne vec preferencie a taktiež existuje veľké množstvo technológií v rôznych programovacích jazykoch, ktoré vieme použiť. Ak by sme sa rozhodli ísť cestou progreísvnych webových aplikácií, vieme si zvoliť stack React, Vue, Angular + NodeJS, prípadne vieme využiť tradičnejšie a viac production ready riešenia ako Laravel, WordPress a podobne, ktoré však vyžadujú podstatnú znalosť skriptovacieho jazyka PHP. Radi by sme v našom prípade išli cestou Laravel-u, v prípadnej kombinácií PWA frontendu pomocou technológie VueJS.

Rozhodli sme sa tak hlavne kvôli flexibilite a úplnosti Laravelu ako riešenia

pre vytváranie moderných webových stránok, poskytuje veľmi schopný manažment nástroj **Artisan**, pomocou ktorého vieme cez CLI vytvárať **Modely, Controllery, Views, Autentifikačný scaffolding a mnoho iného**, čo by sme za normálnych okolností museli vyvíjať ako sa hovorí "from scratch"

Kapitola 3

Návrh riešenia

3.0.1 Príprava prostredia OS Ubuntu

Ubuntu bude pre nás slúžiť ako náš Linuxový systém, ktorého systémové volania budeme cez naše rozhranie prekladať. Keďže naše WCM bude self-hosted priamo na serveri, potrebujeme nainštalovať pár balíčkov:

- **Aktualizované balíčky na najnovšie verzie vrátane verzie kernelu**
- **NGINX webový server**
- **Python 3.11** s nainštalovaným rozšírením **virtual environment, PAM a podobne**
- **PHP 8.2** s podporou **PHP-FPM** a nainštalovaným **Composer-om**

3.0.2 Návrh architektúry systému

Systém je možné rozdeliť na dve časti, z ktorých každá bude mať na starosti iný aspekt nášho systému. Takýmto spôsobom dosiahneme dostatočnú abstrakciu a akúsi nezávislosť medzi jednotlivými službami.

- **Aplikačnú / Frontend**
- **Serverovú / Backend**

Aplikačná vrstva bude riadiť webovú časť, čize sem vieme zaradiť všetko čo sa webovej prezentácie týka. Patrí sem teda Laravel, PHP, Nginx,...

Naopak **Serverová vrstva** bude zodpovedná za preklad systémových volaní z webu na "raw"linuxové príkazy, pričom bude slúžiť ako middleware, na ktorom bude prebiehať aj overovanie, autentizácia a autorizácia.

Serverová vrstva bude riadená API serverom Python Flask, ktorý bude spracúvať HTTP requesty vo formáte JSON, ktorý v payloade bude obsahovať príkazy a ich argumenty na preklad. Tento API server následne request vyparsuje a pripraví jedntolivé parametre pre zaregistrovaný modul, ktorý bude obstarávať jednotlivé systémove volania pomocou modulu **subprocess**.

Po úspešnom ukončení výkonu systémového volania modul odovzdá API serveru odpoveď, ktorý výsledok naformátuje opäť do formátu JSON a vráti aplikačnej vrstve na vytvorenie webovej prezentácie a vykreslenie / aktualizáciu nových dát.

Každý takýto request je samozrejme autorizovaný pomocou dekorátoru, ktorý vykoná PAM autentifikáciu používateľa a ak je autentifikácia úspešná, používateľovi je vygenerovaný **OAuth token** s platnosťou 60 minút, ktorý bude použitý na každý ďalší request bez nutnosti opakovanej autentifikácie.

Celá architektúra by mala byť koncipovaná s dôrazom na modularitu t.j. jednoduchý vývoj rozšírení bez nutnosti zásahov do jadra programu. Na by sme radi využili návrhový dizajn bootstrapper, ktorý si prejde zaregistrované moduly a na základe ich názvu ich načíta a inicializuje. Následne je volanie veľmi jednoduché a nasleduje konvenciu ***nazov_modulu.funkcia_x***

3.0.3 Prenositeľnosť a inštalácia

Ideálne by sme chceli aby náš systém bol jednoducho prenositeľný na iné servery bez nutnosti zložitej konfigurácie, preto by sme radi adoptovali konfiguráciu pomocou premenných prostredia a .env súborov.

Tento prístup nám dovolí jednoduchý prenos aplikácie na iný server a konfigurácia bude prebiehať iba jednoduchou zmenou údajov v súbore .env, prípadne premenných prostredia.

Ďaľšou metódou zlepšia prenositeľnosti a personalizácia je takzvaná dokerizácia, kedy celú aplikáciu spoločne s frontendom a backendom pripravíme pre platformu **Docker**, ktorá nám dovolí definovať celý aplikačný stack do popisovacieho súboru **Dockerfile** prípadne **docker-compose**, ktorý vieme následne použiť na rozbitie aplikácie do mikroslužieb (rozbitie aplikácie do obrazov)

Literatúra

[1] Linux Community. *System Calls In Linux*. 2023. URL: https://man7.org/linux/man-pages/man2/syscall.2.html.

[2] *Difference between gateway protocols CGI, Fastcgi, WSGI*. 2021. URL: https://www.sobyte.net/post/2021-11/cgi-fastcgi-wsgi/.

[3] Phillip J. Eby. *Python enhancement proposals*. 2003. URL: https://peps.python.org/pep-0333/.

[0] Demetra Edwards et al. *What is a web content management system (WCMS)?* 2021. URL: https://www.techtarget.com/searchcontentmanagement/definition/web-content-management-WCM.

[4] Chris Hall. *The updated big list of static website generators for your site, blog or wiki*. 2014. URL: https://iwantmyname.com/blog/the-updated-big-list-of-static-website-generators-for-your-site-blog-or-wiki.

[5] Caswell Hennig. *A History and its Modern-Day Applications*. 2020. URL: https://commons.marymount.edu/it423casshennig/wp-content/uploads/sites/5421/2020/11/Linux-Distro-Paper.pdf.

[6] Mike Johnston. *What is a CMS?* 2010. URL: https://www.cmscritic.com/what-is-a-cms/.

[7] *State of Open Source Security 2022*. 2022. URL: https://snyk.io/reports/open-source-security/.

[8] Matthew Yaswinski, Md Chowdhury a Mike Jochen. “Linux Security: A Survey”. In: máj 2019, s. 357–362. DOI: 10.1109/EIT.2019.8834112.