BELA KNJIGA

SMART COM

Kako preprečiti 'Cross-site-scripting' napade

Vladimir Ban

Kazalo vsebine

Uvod

Velika večina današnjih storitev v Internet omrežju se izvaja preko spletnih aplikacij. Tehnologija spletnih aplikacij je postala sestavni del našega življenja. Brez spletnih aplikacij si praktično ne znamo več predstavljati Internet omrežja.

Znano pa je, da spletne aplikacije prinašajo tudi veliko varnostnih groženj, med katerimi so nekatere lahko zelo nevarne - kraja podatkov, vdor v strežnik, zloraba identitet ipd.

Grožnje mnogokrat izhajajo iz tehnično slabo zasnovanih spletnih strani in aplikacij. Mnogo groženj se nanaša na tehnične podrobnosti posameznih skript oz. programčkov, ki sestavljajo posamezno spletno aplikacijo. Lastnik aplikacije se tovrstnih ranljivosti velikokrat niti ne zaveda, saj kot prvo spletne aplikacije verjetno ni izdelal sam, kot drugo, pa je programski jezik skript velikokrat nerazumljiv oz. nepoznan lastnikom.

Skripte in napake v programski kodi niso edini izvor varnostnih ranljivosti spletnih aplikacij. Spletne aplikacije so lahko ranljive tudi zaradi napačnih varnostnih nastavitev strežnikov in operacijskih sistemov, na katerih tečejo, oziroma slabe zasnove same logike aplikacije.

Dodatno so tu še grožnje, ki izhajajo iz slabih avtentikacijskih mehanizmov za uporabnike in/ali skrbnike aplikacij. Pozabiti ne smemo niti na grožnje, ki izhajajo iz prisluškovanja prometu in slabih SSL implementacij.

Spletne aplikacije tako skrivajo celo vrsto tehnično in vsebinsko zelo različnih ranljivosti. Ker predstavljajo pomembno vez med uporabniki in podatki, v resnici velikokrat predstavljajo glavno grožnjo informacijskim sistemom v podjetjih.

Pri tem moramo poudariti, da med ogrožene aplikacije ne sodijo zgolj tiste, ki so že na prvi pogled vsebinsko potencialno problematične (npr. spletna banka, spletna trgovina ipd.), ampak tudi vse ostale, vključno s (splošnimi) spletnimi mesti podjetij. **Marsikatera tehnična ranljivosti in posledično grožnja se enako nanaša na vse tipe spletnih aplikacij, ne glede na njihovo vsebino.**

Res je, da lahko enostavna spletna stran, v primerjavi s kompleksnejšo spletno aplikacijo, prinaša manj točk, kjer lahko napadalec izkoristi razne tehnične ranljivosti. Vendar pa so enostavne spletne strani po drugi strani lahko za napadalce še bolj privlačne, saj varnostni mehanizmi znotraj posameznih skript niso tako zapleteni in močni, kot to (na srečo) velja za bolj zapletene spletne aplikacije. Tehnična grožnja je lahko enako nevarna, ne glede na to ali se ranljivost nahaja znotraj kompleksne spletne aplikacije, ali pa na neki drugi enostavni spletni strani. V dokumentu tako enačimo pojma spletnih aplikacij in spletnih strani, saj z vidika groženj praktično ni razlik.

Nekatere ranljivosti so v praksi pogoste, nekatere pa se pojavljajo redkeje. Nekatere predstavljajo večjo grožnjo, druge pa predstavljajo zgolj manjšo ogroženost. Ena izmed ranljivosti spletnih strani, ki smo jo pri izvajanju varnostnih pregledov različnih spletnih aplikacij pogosto zaznali, hkrati pa ta ranljivost prinaša nevarnejše grožnje, je 'Cross-site-scripting' (kratica XSS).

V nadaljevanju dokumenta bomo podrobneje obrazložili ozadje te ranljivosti, kakšna tveganjaprinaša, kje in kdaj se najbolj pogosto pojavlja in nenazadnje, kako se pred takšno ranljivostjo obvarujemo.

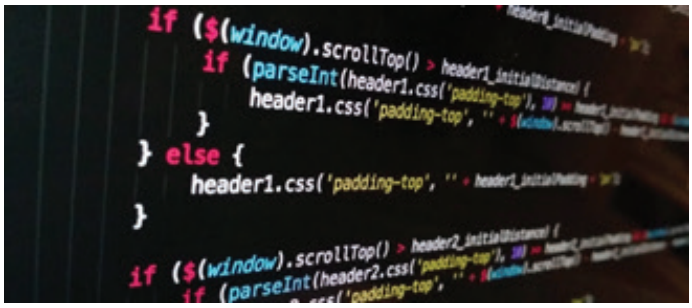
1. poglavje

Kaj je Cross-site-scripting – osnovna razlaga

Spletne aplikacije delujejo tako, da se uporabniki preko svojih spletnih brskalnikov pogovarjajo z aplikacijo, ki teče na strežniku. Brskalnik aplikaciji posreduje določene pakete (podatke, klice, vprašanja ipd.), aplikacija pa nato vrača vsebino, ki se prikaže v brskalniku.

Aplikacija brskalniku vsebino pošilja v obliki HTML zapisov. HTML jezik je zelo enostaven in posledično tudi zelo omejen. Namenjen je predvsem statičnim predstavitvam različne vsebine. V sodobnih aplikacijah na interaktivnih spletnih straneh je takšna omejitev prevelika, zato se je v HTML jeziku že dokaj zgodaj pričel uveljavljati 'javascript' programski jezik. 'Javascript' programski jezik daje HTML stranem bistveno več vsebine in interaktivnosti.

Ko se brskalnik pogovarja s spletno aplikacijo, ta brskalniku poleg klasične HTML vsebine, pošilja tudi 'javascript' ukaze. 'Javascript' ukazi se izvršujejo v brskalniku in tako dajejo uporabniku dodatno kvaliteto pri prikazovanju spletnih strani.



Ker pa je 'javascript' jezik v svojih zmogljivostih bistveno manj omejen, njegova uporaba prinaša določene varnostne grožnje. 'Javascript' ukaze, ki se med delom s spletno aplikacijo izvajajo v brskalniku, določa aplikacija.

Če je aplikacija že v osnovi zlonamerna, lahko brskalnikom naloži izvajanje ukazov, ki bodo povzročili zlonamerno dejanje.

Pri 'Cross-site-scripting' ranljivosti pa imamo v resnici dobronamerno aplikacijo, ki sama po sebi nima namena brskalnikom nalagati zlonamernih ukazov. Vendar ima tehnično ranljivost, ki napadalcem omogoča vplivati na to, kateri 'javascript' ukazi se posredujejo brskalnikom.

Če aplikacija to dovoljuje (v nadaljevanju bomo podrobneje pogledali, na kakšen način), potem takšno ranljivost imenujemo 'Cross-site-scripting' oz. krajše XSS.

Pri XSS gre torej za tehnično ranljivost aplikacije, ki jo napadalec tako ali drugače izkoristi, z namenom škodovati uporabnikom aplikacije in ne aplikaciji sami. Omenjena specifika včasih povzroča »zmedo« pri razumevanju ranljivosti, hkrati pa ranljivost neupravičeno postavlja v manj pomemben položaj. Kar je lahko zelo zavajajoče. XSS ranljivost lahko pomeni zelo hude grožnje, ki jih bomo podrobneje tudi opredelili. Na tem mestu omenimo glavne tri:

- nepooblaščena prijava v aplikacijo,
- prevzem skrbniških pravic nad aplikacijo in
- vdor v notranjost omrežja podjetja.

Ravno v tem pogledu so (splošne) spletne strani podjetij pomembne. Četudi gostujejo in ne vsebujejo omembe vrednih podatkov in funkcij, še vedno velja, da zaposleni v podjetju svoji spletni strani zaupajo. V kolikor vsebuje XSS ranljivost, lahko napadalec zelo učinkovito napade uporabnike in preko njih in XSS izvede napad v notranjost podjetja.

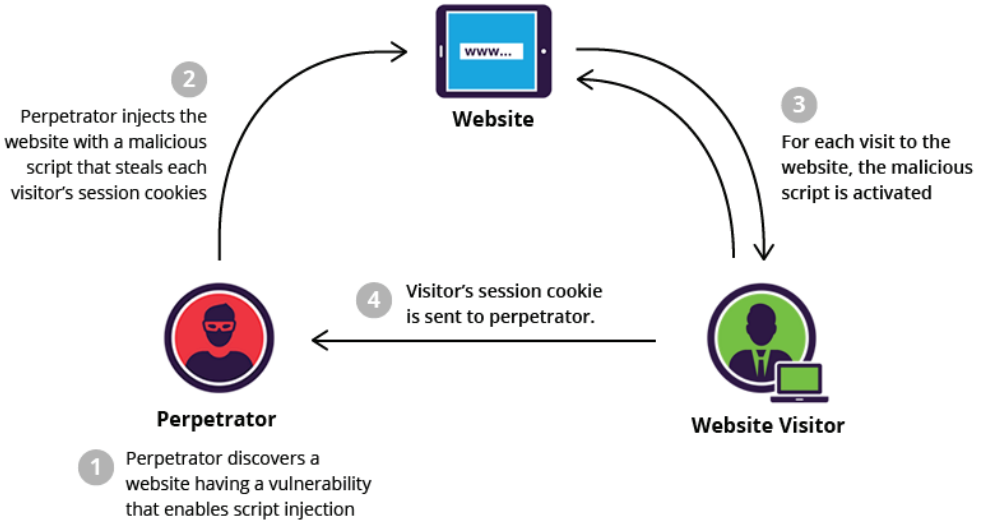
2.
poglavje

Kaj je Cross-site-scripting – tehnična razlaga

Na dveh poenostavljenih primerih si bomo XSS ranljivost ogledali bolj podrobno.

Primer 1:

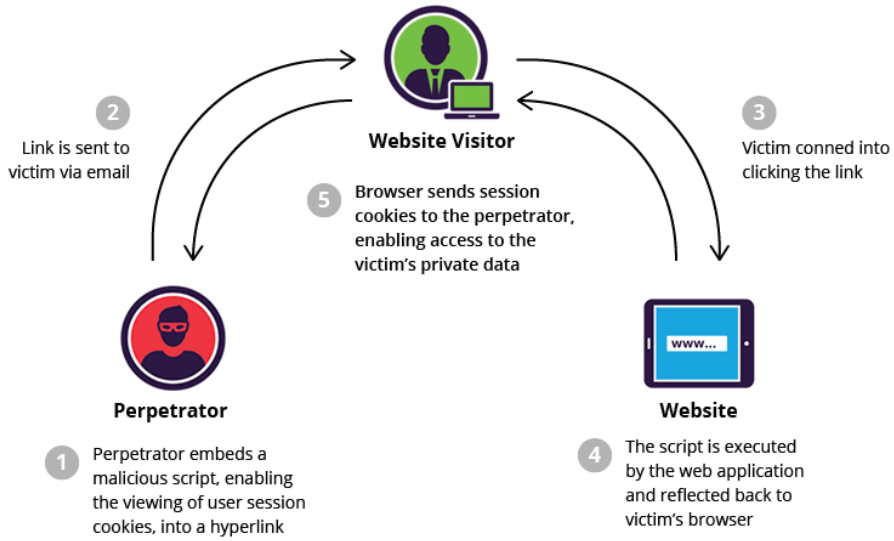
Imamo spletno stran, kjer objavljamo novice, hkrati pa uporabnikom omogoča funkcionalnost komentiranja novic. Vpisani komentarji postanejo del spletne strani in se ob vsakem obisku novice izpišejo obiskovalcem.



V normalni situaciji uporabnik pod komentar vpiše »Dober članek!«. Ko pride drug obiskovalec na to stran, aplikacija njegovemu brskalniku vrne HTML kodo, ki na koncu vsebuje ta zapis. Napadalec lahko namesto komentarja vpiše 'javascript' zlonamerno kodo. Ko v tem primeru tretji obiskovalec pride na stran, aplikacija »komentar« napadalca doda v HTML kodo in jo pošlje brskalniku obiskovalca. Če je koda napisana pravilno, brskalnik uporabnika meni, da gre pri tem delu zapisa dejansko za 'javascript' kodo, ki jo tudi izvede in ne samo izpiše.

Primer 2:

Imamo spletno stran, ki vsebuje funkcionalnost iskalnika.



V normalni situaciji uporabnik v polje iskalnika vpiše »SmartCom«. Aplikacija nato vrne uporabniku HTML stran, ki med drugim vsebuje stavek »Rezultati iskanja besede SmartCom so naslednji...« in potem doda izpisek najdenih besed.

Če pa namesto besede »*SmartCom*« nekdo v iskalno polje vpiše 'javascript kodo', aplikacija v HTML odgovoru vrne »*Rezultati iskanja besede javascript koda so ...*«. Ko HTML odgovor pride do brskalnika, se del »javascript koda« izvede in ne zgolj izpiše kot besedilo.

Navedena primera sta razmeroma poenostavljena in v resnici bi se za boljše razumevanje morali poglobiti še v nekaj tehničnih podrobnosti (o tem bolj podrobno v naslednjih poglavjih), vendar dokaj jasno in točno prikazujeta dva dejanska tipa XSS ranljivosti.

- Prvi tip je 'Stored Cross-site-scripting'.
- Drugi tip je 'Reflected Cross-site-scripting'.

'Stored Cross-site-scripting'

Ta tip ranljivosti izkorišča način delovanja aplikacije, ko ta v bazo (ali drugam) zapiše vhodni podatek uporabnika. Ta podatek pa v prihodnosti uporabi pri generiranju HTML strani pri drugih uporabnikih. 'Stored' XSS ranljivost v primerjavi z 'Reflected' tipom XSS ranljivosti ne potrebuje neposredne interakcije s svojo žrtvijo. Napadalec 'nastavi past' v dobronamerni (a ranljivi) aplikaciji in čaka, da se bo uporabnik/žrtev v to past ujela.

Napadalec lahko na ta način napada žrtve, ki jih sploh ne pozna. Če napadalec pri svojem napadu cilja na točno določeno žrtev, pa je odvisen od tega, ali žrtev dejansko obišče del aplikacije, kjer je nastavljena past. Če ni jasno razvidno, da bo konkretna žrtev prej ali slej obiskala stran, kjer je past postavljena, si lahko napadalec na primer pomaga s 'social-engineering' trikom. Napadalec postavi past, nato pa žrtvi pošlje e-mail sporočilo, kjer jo poskuša zavesti k obisku strani. Pri tem napadalec nima vnaprej zagotovljenega uspeha, da bo žrtev res obiskala stran. Zavedati se moramo, da URL naslov, kamor napadalec želi preusmeriti žrtev, ni sam po sebi zlonameren. Kot prvo kaže na zaupanja vredno spletno stran, kot drugo URL naslov ne vsebuje zlonamernih klicev (ti »čakajo« na spletni strani). Verjetnost, da bo žrtev tak klik izvedla, je lahko zato zelo velika.

'Reflected Cross-site-scripting'

Ta tip ranljivosti je kot zrcalo. Aplikacija res dopušča, da ji nekdo posreduje vhodne podatke tako, da bo ta vrnila zlonamerne 'javascript' klice, a ti klici se vedno vrnejo pošiljatelju, torej tistemu, ki je zlonamerne vhodne podatke posredoval.

Ker napadalec nima namena napadati samega sebe, je odvisen od dodatnih trikov, ki jih mora v napadu uporabiti. Očiten primer takšnega dodatnega trika je 'phishing' e-mail sporočilo, v katerega vstavi URL povezavo. V primerjavi s 'phishing' e-mail sporočilom, ki smo ga omenjali pri 'Stored' tipu ranljivosti, je tu za napadalca zadeva bolj zapletena, saj mora URL link v sporočilu dejansko vsebovati tudi zlonamerno 'javascript' kodo. Zlonamerno kodo napadalec torej ne »shrani« v aplikacijo, ampak jo mora vstaviti v URL naslov, na katerega mora žrtev tudi (nehote) klikniti.

Na prvi pogled se zdi, da je XSS ranljivost 'Reflected' tipa manj nevarna, saj je napadalec neposredno odvisen od odziva žrtve. Vendar se v praksi izkaže, da so načini zavajanja lahko zelo učinkoviti. Načinov, kako napadalec vstavljeno 'javascript' zlonamerno kodo skrije/zakrije v URL naslovu, je več.

Še več, tovrstne ranljivosti so bolj pogoste na spletni strani, tako da napadalec nima prevelikih težav z identifikacijo ranljivih točk v aplikaciji, ki jih nato tudi uporabi.

Razlika med ranljivostma je v načinu, na katerega aplikacija dopušča napadalcu, da v odgovorih dobronamerna aplikacija uporabnikom posreduje zlonamerne 'javascript' ukaze, ter kako napadalec ranljivost dejansko izrabi v praksi. Ne razlikujeta pa se v posledicah. V obeh primerih zlonamerna 'javascript' koda pride do brskalnika uporabnika in se tam izvede in v obeh primerih lahko napadalec uporablja enake zlonamerne 'javascript' klice.

3.
poglavje

Kdaj in kako pride do ranljivosti?

Aplikacija je pri generiranju HTML odgovora uporabila podatek, ki je prišel od uporabnika (ali s predhodnim klicem – če gre za 'Reflected' tip ranljivosti, oziroma iz baze – kadar gre za 'Stored' tip ranljivosti) in s tem nehote v HTML kodo vstavila zlonamerne 'javascript' ukaze.

Na dveh primerih poglejmo zadevo bolj podrobno.

Primer 1:

Aplikacija deluje tako, da v danem trenutku vzame podatek o URL naslovu, ki ga je nazadnje uporabil uporabnik, in ga vstavi v svojo HTML kodo, ki jo v naslednjem odgovoru posreduje brskalniku.

```
5px;">Clean</span> button to clean your source code.</p>
<h2 style="color: #2e6c80;">Some useful features:</h2>
<ol style="list-style: none; font-size: 14px; line-height: 32px; font-weight: bold;">
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img/01-interactive-connection.png" alt="interactive connection" width="45" />
Interactive source editor</li>
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img/02-html-clean.png" alt="html cleaner" width="45" /> HTML Cleaning</li>
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img/03-docs-to-html.png" alt="Word to html" width="45" /> Word to HTML
conversion</li>
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img/04-replace.png" alt="replace text" width="45" /> Find and Replace</li>
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img/05-gibberish.png" alt="gibberish" width="45" /> Lorem-Ipsum generator</li>
<li style="clear: both;"><img style="float: left;" src="http://ranljiva.aplikacija/.../dodan_zlonamerni_del" alt="html table div" width="45" /> Table to DIV
conversion</li>
</ol>
<p>             </p>
<h2 style="color: #2e6c80;">Dodana zlonamerna koda</h2>
<table class="editorDemoTable">
<thead>
<tr>
<td>Name of the feature</td>
```

Primer 2:

Aplikacija deluje tako, da iz nekega vpisanega parametra, ki ga je prejela, vzame vsebino in to vsebino kot besedilo vstavi v HTML kodo, ki jo v naslednjem odgovoru posreduje brskalniku.

```
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img
Interactive source editor</li>
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img
conversion</li>
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img
<li style="clear: both;"><img style="float: left;" src="https://html-online.com/img
<li style="clear: both;"><img style="float: left;" src="http://ranljiva.aplikacija/.../
conversion</li>
</ol>
<p>             </p>
<h2 style="color: #2e6c80;">Dodana zlonamerna koda</h2>
<table class="editorDemoTable">
<thead>
<tr>
<td>Name of the feature</td>
<td>Example</td>
<td>Default</td>
</tr>
</thead>
```

Navedena primera sta zelo realna, a je takoj jasno, da se 'javascript' koda ne bo izvedla sama od sebe, kljub temu, da je aplikacija vsebino z zlonamerno 'javascript' kodo vstavila v HTML odgovor. Zlonamerna vsebina je v HTML odgovor vstavljena tako, da brskalnik tega ne dojame kot ukaz. Vstavljena vsebina je namreč del stavkov/vrstic, ki s predhodnimi narekovaji in ostalimi znaki, vstavljeno 'javascript' kodo »pokvarijo«.

Napadalci morajo zato zlonamerni kodi dodati še nekaj znakov, s katerimi poskrbijo, da se v HTML odgovoru najprej pravilno zaključi nek drug začeti stavek. In vrinjena 'javascript' koda pride do izraza na ta način, da jo brskalnik prepozna. Napadalec lahko v navedenih primerih izvede naslednje.

Primer, ko napadalec vrine zgolj 'javascript' kodo.

```
<li style="clear: both;"><img style="float: left;" src="http://ranljiva.aplikacija/.../dodan_zlonamerni_del" alt="html table div" width="45" />
```

Primer, ko napadalec 'javascript' kodi na začetku vsebine doda znake »/>, na koncu pa znake <!-.

```
<li style="clear: both;"><img style="float: left;" src="http://ranljiva.aplikacija/.../"/>dodan_zlonamerni_del<!--' alt="html table div" width="45" /> Table to DIV
```

Z dodatki se situacija v HTML odgovoru spremeni v prid napadalca. Znaki na začetku vsebine povzročijo, da se začeta vsebina HTML strani ustrezno zaključi. Znaki na koncu vsebine pa sicer niso nujno potrebni za zagon 'javascript' ukazov. Dodani so zato, da onemogočijo, da vsebina HTML strani kodo vizualno ali kako drugače »pokvari«.

Kaj točno mora napadalec v danem trenutku dodati zlonamerni 'javascript' kodi je odvisno od primera do primera. Znani so klasični triki, točno delujoča kombinacija »trikov« pa je odvisna od konkretne situacije. Potrebno se je zavedati, da za napadalca ta del ni problematičen. Na voljo je kar nekaj učinkovitih orodij, ki jih napadalec lahko uporabi, hkrati pa lahko z osnovnim znanjem HTML kode to naredi tudi sam.

Izvirni »greh« aplikacije je, da podatkov pri prevzemu ni ustrezno preverila – če vsebujejo 'javascript' ukaze ali druge posebne znake, ki so tipični pri zlorabi takšnih ranljivosti. Vsi vhodni podatki, ki se nato tako ali drugače uporabijo pri generiranju HTML strani, se morajo preveriti, preden se dejansko uporabijo.

To velja tako za podatke, kjer je takoj jasno, da jih je vpisal uporabnik, kot tudi za podatke, ki jih uporabnik ni neposredno vnesel, lahko pa vpliva na njihovo vsebino. Recimo podatek o tipu brskalnika, ki ga uporabnik uporablja – gre za »sistemski«

podatek, ki ga brskalnik sam doda http paketom pri komunikaciji z aplikacijo, vendar napadalec lahko s posebnimi skriptami ali programi tudi v takšen podatek vstavi zlonamerne 'javascript' klice.

Kako točno mora aplikacija izvajati preverjanje, bomo podrobneje opredelili v ločenem poglavju, tu navajamo nekaj izhodišč, ki so se nakazala v zgornjih primerih:

- Aplikacija lahko preverja vsebino podatkov, ki jih želi uporabiti, če morebiti vsebujejo posebne znake, ki se tipično uporabljajo pri zaključevanju HTML stavkov.
- Aplikacija lahko preverja vsebino podatkov, ki jih želi uporabiti, če morebiti vsebujejo konkretne 'javascript' ukaze.
- Aplikacija lahko preverja vsebino podatkov, ki jih želi uporabiti, če morebiti vsebujejo posebne znake, ki se tipično uporabljajo v 'javascript' sintaksi.

V teoriji je vse jasno. Spoznali pa bomo, da v praksi niso vsi načini preverjanja tudi nujno učinkoviti ali možni.

Dodatno je za aplikacijo pomembno, da zagotovi tudi druge obrambne mehanizme, ki (ne)posredno vplivajo na uspešnost XSS napadov.

4. poglavje

Zakaj je Cross-site-scripting nevaren?

Preden nadaljujemo s tehnično razlago okoliščin XSS ranljivosti je čas, da pogledamo, kakšne so lahko možne posledice tovrstnih napadov. V besedilu uporabljamo izraz »zlonamerna« koda. Za kaj v resnici pravzaprav gre? Kaj lahko povzroči? Kaj 'javascript' jezik zmore in česa se lahko (upravičeno) bojimo?

Grožnje, ki jih programski jezik prinaša, lahko razdelimo v štiri skupine:

- kraja 'Session Cookie' podatka,
- zavajanje uporabnika v brskalniku,
- pridobivanje informacij iz brskalnika in
- ostali napadi.

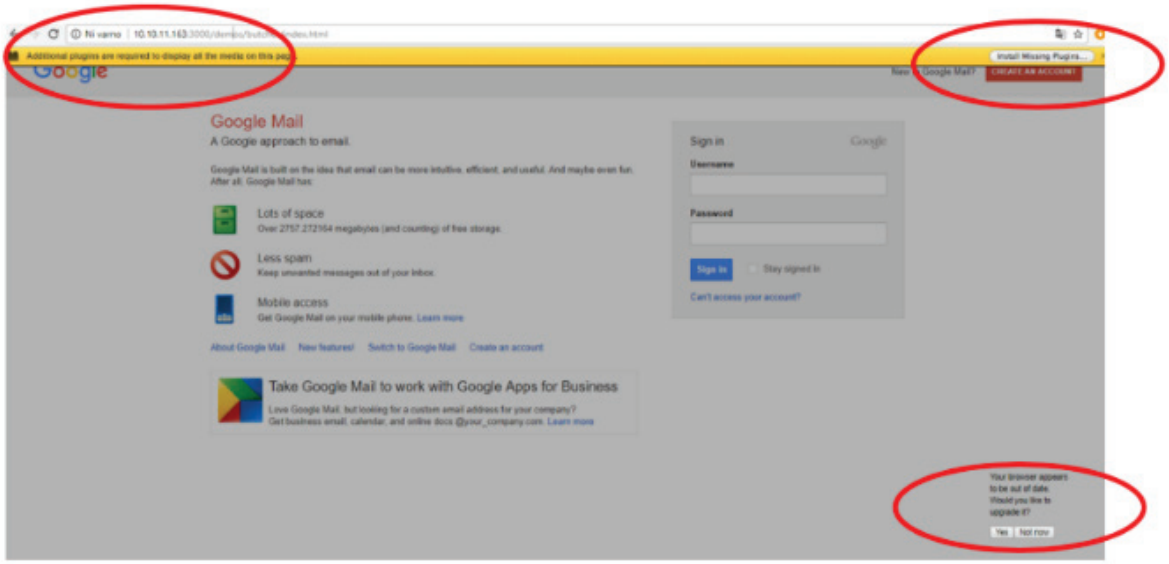
Kraja 'Session Cookie' podatka

Princip avtentikacije v spletnih aplikacijah je poseben. O tem bi lahko napisali ločeno poglavje, a na kratko razloženo, je izziv sledeč. S stališča uporabnikov je zadeva enostavna. Uporabnik se prijavi v aplikacijo in od tam dalje so mu na voljo razne funkcionalnosti. Če se ne prijavi, dostopa do funkcionalnosti nima.

S stališča aplikacije pa temu ni tako. Aplikacija sprejema klice iz Internet omrežja, sestavljeni iz množice http paketov, ki prihajajo z različnimi zahtevki. Nekatere pakete pošiljajo uporabniki, ki se v aplikacijo še niso prijavili, nekatere pa prijavljeni uporabniki. Aplikacija lahko brez posebnih mehanizmov spozna, kateri paket pripada kateri TCP seji, a to s samo avtentikacijo v aplikaciji nima povezave. Delo uporabnika v aplikaciji namreč sproža večje število TCP sej in nepraktično bi bilo, da bi se moral uporabnik na začetku vsake seje posebej in ponovno avtenticirati. Aplikacija zato uporabi 'Cookie' podatke. Po uspešni avtentikaciji uporabnika aplikacija generira vrednost, ki ni uganljiva. Vrednost zapiše v bazo in jo hkrati v obliki 'Cookie' vrednosti pošlje brskalniku uporabnika. Brskalnik nato poskrbi, da je vsak paket, ki je posredovan aplikaciji, opremljen s tem podatkom in tako aplikacija ve ali določen paket dejansko pripada nekemu uporabniku in kateremu pripada. To je poenostavljen primer uporabe 'Cookie' vrednosti. V praksi lahko aplikacija mehanizem še dodatno »zakomplicira«, a večino aplikacij deluje na ta način.

Če napadalec uporabniku uspe ukrasti 'Cookie' podatek, lahko v njegovem imenu nepooblaščeno vstopa v aplikacijo. Eden izmed klasičnih načinov kraje tega podatka je preko 'javascript' ukazov. 'Javascript' ukazi so zmožni v brskalniku prebrati 'Cookie' podatek in ga posredovati napadalcu. Pri XSS ranljivosti torej napadalec lahko poskuša doseči, da pri uporabniku ranljiva aplikacija sproži ukaze, ki ukradejo 'Cookie' vrednost in jo pošlje uporabniku.

Zavajanje uporabnika v brskalniku



Napadalci si za cilj velikokrat postavijo prevzem kontrole nad delovno postajo žrtve. To lahko dosežejo z raznimi zlonamernimi datotekami, ki vsebujejo sistemske klice in posegajo neposredno v operacijski sistem. Vendar 'javascript' jezik takšnih klicev ne pozna. Napadalec torej ne more sestaviti 'javascript' ukaznega zaporedja, ki bi mu neposredno omogočalo dostop do operacijskega sistema. V tem smislu je 'javascript' za napadalca močno omejen. V pomoč pa je lahko. Če želijo prevzeti kontrolo nad delovno postajo svoje žrtve, jo morajo zavesi k zagonu zlonamerne datoteke.

'Javascript' lahko pri takšnem zavajanju zelo pomaga. Ko žrtev obišče zaupanja vredno aplikacijo, lahko napadalec vnese 'javascript' ukaze, ki na različne načine preuredijo spletno stran.

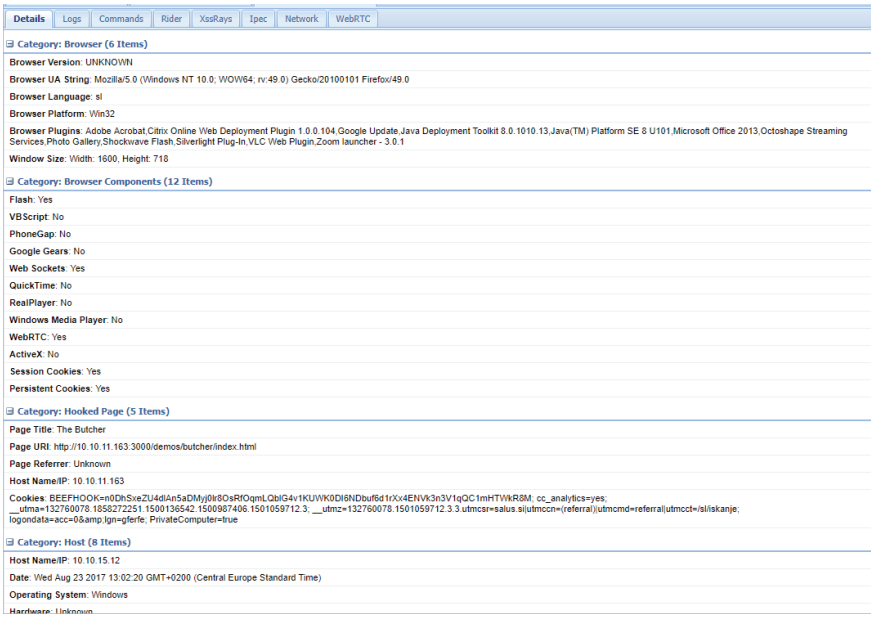
- Ko uporabnik obišče ranljivo aplikacijo, se v brskalniku pojavi okno, ki uporabnika pozove k namestitvi posodobitev za ogled vsebine. Ker se uporabnik nahaja na zaupanja vredni aplikaciji, je verjetnost, da bo to naredil dokaj velika. Napadalec seveda poskrbi, da se v tem primeru na delovno postajo uporabnika namesti prava zlonamerna »exe« datoteka.

- Ko uporabnik obišče ranljivo aplikacijo in klikne na neko funkcijo, se izvede »tiha« 'redirekcija' na lažni strežnik, ki je poln zlonamernih datotek (na primer v aplikaciji klikne na prenos PDF, 'javascript' ukazi pa poskrbijo, da se na delovno postajo uporabnika prenese okužen PDF iz lažnega strežnika in ne iz pravega).

Napadalec lahko z 'javascript' ukazi uporabniku pri delu z ranljivo aplikacijo prikaže lažna vstopna okna. Uporabnik obišče zaupanja vredno aplikacijo. 'Javascript' nato uporabniku med delom na brskalniku prikaže prijavno okno v Gmail ali Facebook sistem. Uporabnik misli, da se je brskalnik odjavil iz sistemov, zato vpiše svoja gesla in nadaljuje delo s pravo aplikacijo. S tem ima napadalec dostop do Gmail ali Facebook računa žrtve. Že to je zelo nerodno, dodatno lahko napadalec preko teh kanalov žrtev še enostavneje prepriča v namestitev zares škodljive datoteke.

Pridobivanje informacij iz brskalnika

Naslavljanje »bližnjic«, ki jih CPU-ji uporabljajo in ki so se sedaj izkazale kot nevarne, je na videz enostavnejše. Takšne popravke lahko izvedemo na nivoju operacijskih sistemov (ali s popravki delovanja BIOS-a), oziroma celo na nivoju programske opreme (ko govorimo o konkretnih napadih na brskalnike preko Spectre ranljivosti). Vendar je tudi tukaj težava. Neizbežno bodo ti popravki vplivali na performančno zmogljivost CPU, saj nenazadnje direktno naslavljajo (torej preprečujejo ali pa vsaj močno omejujejo) uporabo »bližnjic«, ki so jih do sedaj CPU pridno izkoriščale. Vpliv na performance CPU je torej neizbežen, res pa je, da še ni čisto jasno, kako se bo ta vpliv kazal na dejanske hitrosti izvajanja posameznih aplikacij. Pravih podatkov, analiz in nenazadnje izkušenj s tem v praksi seveda še ni na voljo.



'Javascript' ukazi ne morejo posegati v operacijski sistem delovne postaje, lahko pa dostopajo do določenih sistemskih podatkov. To so na primer IP naslov delovne postaje, nameščene verzije brskalnika in dodatkov ipd. Gre za tehnične podatke, ki neposredno ne predstavljajo vdora v omrežje, jih pa napadalec lahko dokaj učinkovito uporabi pri nadaljnjih fazah napada, saj sedaj točno ve, kakšne ranljivosti lahko znotraj pravih zlonamernih kod izkoristi. S tem je napadalcu razkrita marsikatera neznanka, brez katere bi bila priprava napada daljša ali pa bi bil napad celo neuspešen.

Ostali napadi

V to skupino sodijo vse ostale potencialne in možne zlorabe. Nikoli ne smemo podcenjevati zmožnosti in domišljije napadalcev.

Izpostavimo 'javascript' kodo, ki se omenja v aktualnih ranljivostih 'Specter', kjer naj bi bilo možno preko 'javascript' ukazov izkoriščati najnovejše ranljivosti na Intel procesorjih. Možno naj bi bilo posegati v integriteto brskalnikov, kjer zavihek, ki dostopa do zlonamerne strani, posega v ostale zavihke, kjer uporabnik dostopa do varnih aplikacij. Do sedaj je tak »preskok« med zavihki veljal za nemogočega.

5. poglavje

Kakšni so nivoji »jakosti« Cross-site-scripting ranljivosti?

V dosedanjih poglavjih smo spoznali, kaj ranljivost je, kako nastane oz. se pojavi in kakšne so možne posledice. V povezavi z ranljivostjo pa je potrebno razumeti še naslednje.

Ranljivost ni »črno-bela«. Obstajajo mesta v aplikaciji, ki so brez tovrstnih ranljivosti, obstajajo mesta, kjer lahko ranljivost v polnosti izkoristimo, obstajajo pa lahko tudi mesta, kjer ranljivost tehnično obstaja, a je napadalec soočen z omejitvami, ki so rezultat varnostnih mehanizmov ali pa (za napadalca) »splet nesrečnih okoliščin«.

> Smiselno je razumeti, kakšna so lahko takšna »vmesna« stanja, saj lahko tako skrbniki lažje določijo prioritete potrebnih popravkov.

Definirajmo najprej, kdaj je aplikacija sploh ni ranljiva in kaj pomeni, da je aplikacija v določeni točki polno ranljiva. Predpostavimo, da imamo mesto v aplikaciji, kjer dejansko pride do tega, da aplikacija prevzame določen vhodni podatek in ga prepiše v HTML odgovor brskalniku.

Aplikacijo bomo smatrali za polno ranljivo, če lahko na tej točki brskalniku v HTML odgovor vrinemo navodilo (v obliki 'javascript' ukaza), da ob prikazu strani iz napadalčevega strežnika pobere in zažene poljubno 'javascript' datoteko.

Ko govorimo o zlonamerni 'javascript' kodi velja. Bolj je koda zlonamerna, bolj je kompleksna. Četudi aplikacija nima varnostnih mehanizmov proti XSS ranljivosti, si lahko predstavljamo, da bi napadalci v določenih primerih imeli vsaj praktično težavo, kako v nek podatek vriniti kompleksno 'javascript' kodo, dolžine tudi nekaj 100 kB. Zato napadalci radi izkoriščajo naslednjo lastnost HTML strani. HTML stran lahko neposredno vsebuje vse 'javascript' ukaze, ki so del nekega odgovora, lahko pa vsebuje zgolj navodilo brskalnikom, da ob prikazu te HTML strani iz tretjega strežnika poberejo 'javascript' datoteko in jo zaženejo. Če je takšno »navodilo« možno vriniti, pravimo, da je aplikacija na tej točki polno ranljiva, saj napadalec nima težav glede tega, katere 'javascript' ukaze lahko uporabi, katere pa ne.

Če na drugi strani aplikacija v neki točki »prepisuje« vhodne podatke v HTML odgovor, vendar hkrati ne obstaja nobena možnost za vnos zlonamernih ukazov, pravimo, da aplikacija v tej točki ni ranljiva.

Vmesna stanja so tista, kjer po eni strani ne drži, da napadalec ne more aplikaciji vriniti posebnih znakov ali 'javascript' ukazov, vendar hkrati tudi ne more v polnosti uporabiti vseh ukazov, ki bi si jih želel.

Oglejmo si ta vmesna stanja na štirih primerih.

1 Aplikacija posreduje brskalnikom ne-HTML odgovore

V dokumentu smo večkrat omenili, da aplikacija brskalnikom posreduje HTML odgovore. V resnici pa lahko aplikacija brskalnikom posreduje odgovore tudi v drugačni obliki (recimo tekstovni, json obliki ipd.). Brskalniki bodo 'javascript' ukaze izvajali le, če se nahajajo v odgovoru, ki je deklariran kot HTML.

Lahko se torej zgodi, da aplikacija v neki točki dejansko prevzame vhodni podatek in ga brez omejitev prepiše v svoj odgovor brskalniku, vendar bo tak odgovor označen kot ne-HTML format. Napadalec lahko v takšnih primerih res posreduje brskalnikom svojih žrtev odgovore, ki vsebujejo polno zlonamerno 'javascript' kodo. Vendar napadalec brskalnikov neposredno ne bo mogel prepričati, da kodo sploh upoštevajo kot 'javascript' kodo.

Takšne točke torej niso neposredno kritične. Je pa potrebno v teh primerih vzeti v obzir celotno delovanje aplikacije. Če lahko aplikacija pod določenimi pogoji in okoliščinami takšne odgovore vseeno nekje drugje interpretira v HTML obliki, je zloraba teoretično še vedno možna. Ali je temu tako ali ne, pa je včasih težko oceniti.

Odprava tovrstnih točk ni nujna, še vedno pa je priporočljiva, razen, če dejansko lahko ocenimo, da naknadni prenos v HTML obliko ni mogoč.

2 Aplikacija preveč neposredno prenaša vhodne podatke

Druga skrajnost so primeri, ko aplikacija »preveč« dobesedno »prepisuje« vhodne podatke v HTML odgovore. Ta primer je pomemben predvsem za 'Reflected' tip ranljivosti.

Ko uporabnik vpiše URL naslov (ali pa na njega klikne), brskalnik na URL naslovu izvede URL kodiranje. To pomeni, da se določeni posebni znaki pretvorijo v nek drug zapis (tipičen primer je, da brskalnik preslek prevede v %20). To brskalniki ne storijo zaradi varnosti, ampak zato, da URL naslovi morebiti ne bi povzročili kakšnih tehničnih težav na prenosu. A posredno s tem lahko močno otežijo delo napadalcem pri XSS 'Reflected' ranljivostih.

V primerih, ko aplikacija dejansko pričakuje vhodni podatek oz. se zaveda, da gre za podatek, ki je del URL zapisa, aplikacija najprej izvede URL dekodiranje oz. vse posebne znake zapiše nazaj v prvotno obliko. Brskalnik žrtve bo res »pokvaril« 'javascript' kodo, ki jo je napadalec podtaknil v URL naslov, a bo aplikacija to popravila in 'javascript' koda bo ob predpisu v HTML odgovor ponovno takšna, kot mora biti.

Lahko se zgodi, da aplikacija bolj slučajno kot namerno vhodni zapis prepiše v HTML odgovor (tipičen primer je, ko v glavo HTML odgovora enostavno s 'copy-paste' vstavi URL naslov, ki ga je klical uporabnik). V takšnih primerih ni nujno, da aplikacija izvede URL dekodiranje in 'javascript' koda se v HTML odgovor zapiše v »pokvarjeni« obliki, v takšni kot jo je poslal brskalnik.

Če napadalec teste izvaja ročno brez brskalnika, izgleda kot, da ranljivost obstaja, a v praksi jo bo težko izrabiti, saj bo žrtev URL do aplikacije vedno poslala preko brskalnika.

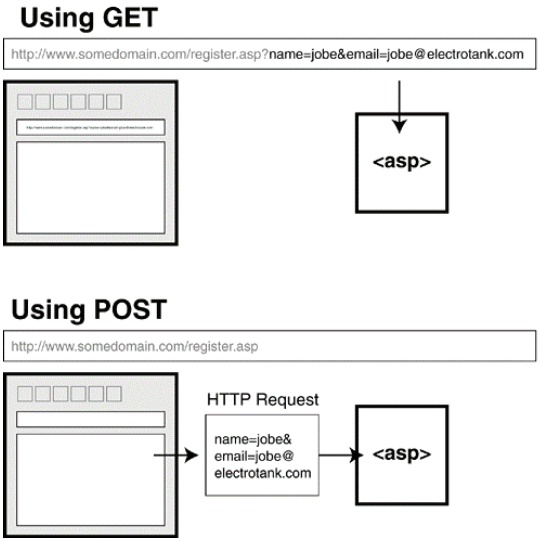
Napadalcu tukaj preostane dvoje. Teoretično je možno, da napadalec pred napadom žrtvi podtakne skripto ali programček, ki prepreči, da brskalnik izvede URL kodiranje. Vendar je to težko in hkrati se postavlja vprašanje, zakaj napadalec sploh išče XSS ranljivost, če pa je svoji žrtvi tako ali tako že zmožen podtikati razne programčke in skripte.

Druga možnost, ki pa je v »zamiranju«, izhaja iz dejstva, da ni nujno, da brskalnik zares izvede URL kodiranje. Določeni brskalniki (npr. Internet Explorer) do nedavnega tega niso počeli. Torej obstaja vsaj hipotetična možnost, da ima uporabnik brskalnik, ki tega ne bo izvedel in je napad potemtakem pri njem izvedljiv. Vsekakor takšne točke niso kritične na enak način, kot tiste, kjer je izraba ranljivosti polna, zgolj iz previdnosti pa svetujemo, da se takšne točke odpravijo.

3 'Reflected' ranljivost znotraj vsebine paketa ali zaglavja

Izvedeli smo že, da mora napadalec pri 'Reflected' tipu ranljivosti žrtvi podtakniti URL naslov, ki vsebuje zlonamerne znake in kodo.

Vendar aplikacija ne sprejema podatkov zgolj iz URL klica. Aplikacije lahko podatke sprejemajo tudi iz vsebine prejetih paketov (brskalnik pošlje podatke v POST obliki in ne v GET obliki), hkrati pa lahko aplikacija vhodne podatke prevzema iz zaglavja prejetih paketov (podatek, ki govori o tipu brskalnika uporabnika). Tehnično gledano lahko aplikacija tudi v takšnih primerih te vhodne podatke na nevaren način uporabi pri generiranju HTML odgovora.



Imamo torej situacijo (ki je v resnici dokaj pogosta pri odkritih XSS ranljivostih), kjer lahko s testom pokažemo, da znamo v podatek znotraj vsebine paketa ali v zaglavje vriniti zlonamerno kodo, ki se potem v polni obliki pojavi v HTML podatku. Tehnično gledano je to klasična XSS 'Reflected' ranljivost.

V praksi pa bo napadalec težko pripravil napad, kjer bo žrtev enak paket poslala aplikaciji. Napad, ko napadalec vso zlonamerno kodo vpiše v URL in ta URL naslov podtakne žrtvi v 'phishing' e-mail sporočilu je zelo nevaren in povsem realen. Napad v katerem napadalec doseže, da žrtev aplikaciji pošlje zlonamerno kodo v notranjost http paketa, pa zelo neverjeten in prej hipotetičen kot realen.

Preden takšno točko opredelimo kot nekritično še pozor! Včasih se zgodi, da aplikacija v neki točki pošlje aplikaciji vnesene podatke v POST obliki. Test pokaže, da aplikacija nato te podatke ne preveri in posledično imamo XSS 'Reflected' ranljivost, ki pa jo ne moremo izrabiti. A marsikdaj aplikacije hkrati podpirajo GET in POST klice in napadalec lahko v tem primeru malce spremeni aplikacijo. Napadalec enostavno podatke iz POST klica premakne v GET klic in takšen klic podtakne žrtvi. Kot rečeno smo v praksi že videli kar nekaj primerov, kjer je takšen napad deloval.

Če test ni uspešen, lahko zaključimo, da ranljivost ni kritična. Vseeno pa priporočamo, da se jo odpravi, saj ni nujno, da napadalec ne najde načina izrabe.

4 Tehnične specifike brskalnikov ali aplikacij, ki nehote omejujejo 'Cross-site-scripting' napade

Zadnji primer je redek, a smo ga pri varnostnih pregledih spletnih aplikacij v praksi že zaznali. Včasih se zgodi, da aplikacija uporabi le del podatkov pri generiranju novega HTML odgovora. Aplikacija pričakuje vnos nekega besedila in nato v odgovoru prikaže le prvih 50 znakov tega besedila. Če aplikacija pri tem prenosu ne izvaja preverjanj glede 'javascript' ukazov, je aplikacija še vedno ranljiva. Napadalec je sicer omejen na prvih 50 znakov, kar v danih okoliščinah ni nujno dovolj za izvedbo napada.

Tovrstni primeri so glede ocene ogroženosti nehvaležni. Pregledovalec oz. skrbnik lahko oceni, da znotraj omejitve ne zna ali ne more sestaviti pravega napada, a to še ne pomeni, da tega ne zmore tudi napadalec. **Nikoli ne smemo podcenjevati znanja in domišljije napadalcev**.

Takšne točke v aplikaciji praviloma označimo kot kritične, za takojšno odpravo, četudi neposredno prave zlorabe nismo dokazali.

6. poglavje

Kako poteka napad preko Cross-site-scripting ranljivosti?

S stališča napadalca mora v aplikaciji najprej identificirati možne točke zlorabe, nato mora pripraviti ustrezne vnose s pravilnimi dodanimi znaki, da se bo zlonamerna 'javascript' koda zares izvedla. Na koncu pa mora pripraviti dejansko zlonamerno skripto, ki pa je lahko zelo kompleksna.

O predpripravi v tem dokumentu ne bomo veliko govorili. Opozarjamo pa, da je mišljenje, da je to zelo težko narediti, napačno. Obstaja namreč cela vrsta orodij, ki bolj ali manj uspešno identificirajo potencialne ranljive točke, obstaja pa tudi cela knjižnica že pripravljenih 'javascript' modelov, ki jih napadalec lahko uporabi za različne napade. Vsekakor napadalec potrebuje nekaj znanja o 'javascript' ukazih in HTML kodi, a sklepanje, da so napadi omejeni zgolj na peščico 'javascript' veleumov je daleč od resnice.

Na štirih primerih si podrobneje oglejmo potek napada v praksi, od točke, ko je napadalec že identificiral ranljivo mesto v aplikaciji in so zlonamerne skripte že pripravljene za napad.

Napad št. 1 – 'Stored' XSS napad preko vnosnih polj – napad na druge uporabnike ali skrbnika

Aplikacija omogoča vpisovanje komentarjev na straneh. Napadalec lahko pripravi besedilo za komentar, ki v resnici na ustrezen način vsebuje zlonamerne 'javascript' kode. Če aplikacija pri vnosu komentarjev ne preverja vsebine, je vnos zlonamernih skript dokaj enostaven. Verjetno napadalec še največ časa porabi za dodatke v besedilu, ker želi 'javascript' ukaze vnesti tako, da se pri žrtvi izvedejo, brez, da ta postane sumničava.

Ko napadalec v spletno stran vrine svoje zlonamerne komentarje, je dovolj, da počaka, da stran obiščejo drugi uporabniki. Če je spletna stran manj obiskana (marsikdaj se zgodi, da so XSS ranljivosti bolj prisotne na »starejšem« delu spletne aplikacije, ki se ne uporablja več) ali pa napadalec cilja na točno določeno osebo, bo morebiti moral še dodatno izzvati osebo, da obišče točno določeno stran. To lahko stori z e-mail 'phishing' sporočilom. Takšno sporočilo bo zelo verjetno uspešno, saj sam URL naslov ne vsebuje nič zlonamernega, zato ga napadalcu ni potrebno skriti. V komentarje novice http://spletna.stran/novica_dneva.html vpiše zlonamerno kodo in nato pošlje e-mail sporočilo »Hej, poglej to novico http://spletna.stran/novica_dneva.html!«.

Če napadom cilja na skrbnike, jih lahko enako uspešno »prepriča«, da obiščejo stran z nastavljeno pastjo. Na določeno stran podtakne zlonamerno kodo, nato pa na strani izvede celo vrsto provokativnih komentarjev, ali pa stran zasuje s komentarji. Marsikateri skrbnik bo pogledal, kaj se dogaja.

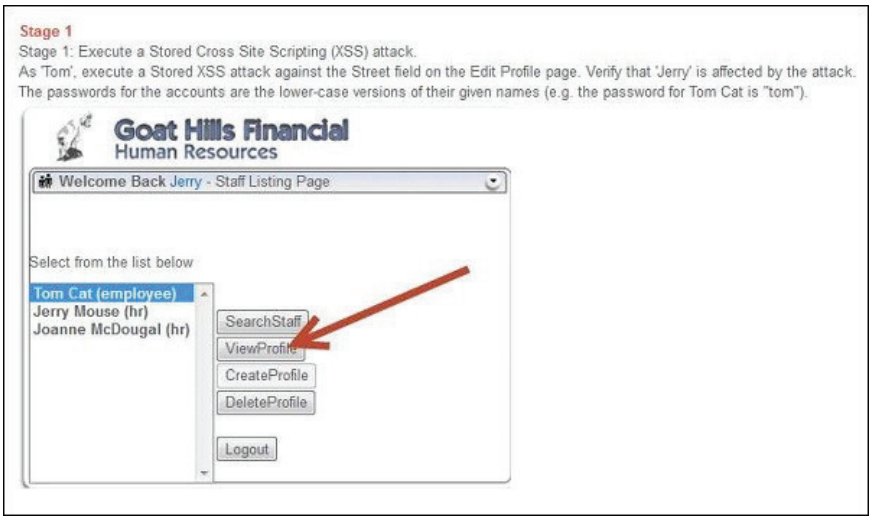
Konkretni primeri:

- Imamo spletno trgovino in spletna stran omogoča komentarje pri artiklih. Napadalec v te komentarje vstavi zlonamerno kodo. Preko kode dobi na primer 'Cookie' vrednost vseh žrtev, ki artikel obiščejo in tako (preko 'Cookie' vrednosti) izvede prijavo v aplikacijo. Lahko pa z 'javascript' kodo povzroči, da se uporabniku pojavi lažno prijavno okno v spletno trgovino. Uporabnik misli, da se je brskalnik odjavil in se ponovno prijavi, s tem pa napadalcu razkrije svoje geslo.

- Imamo spletno stran, ki omogoča komentarje. Napadalec cilja na skrbnika. Vnese provokativen komentar (bodisi veliko njih, ali pa vsebinsko provokativen). Morebiti nato celo izvede funkcijo »prijavi žaljiv komentar«. Zelo verjetno bo skrbnik preveril na strani, kaj se dogaja, napadalec pa bo tako dobil bodisi 'Cookie' vrednost skrbnika, bodisi njegovo geslo.

Napad št. 2 – 'Stored' XSS napad preko vnosov podatkov v profil uporabnika

Prvi primer je povsem realen, a ga v praksi redko srečamo. Razvijalci vedo, da je komentar zelo očitna možna točka vnosa zlonamernih podatkov, zato so tam varovalke v veliki meri urejene.



Bolj premeten način je vnos zlonamernih podatkov v profil uporabnika. Napadalec si ustvari svoj profil (če gre za spletno stran, ki to omogoča) in v polje (recimo svoj naslov) vnese zlonamerno 'javascript' kodo.

Takšen način je predvsem praktičen pri napadu na skrbnika. Uporabniki redko obiskujejo profile drugih uporabnikov (aplikacija verjetno tega sploh ne dopušča). Skrbnik pa ima praviloma vpogled v vse profile.

Pri tem je lahko napadalec potrpežljiv, zgolj vnese kodo v svoj profil ter čaka, da skrbnik pogleda v svoj profil. Lahko pa je napadalec malce bolj provokativen. Na slovenski spletni strani ustvari profil uporabnika, ki na videz prihaja iz Japonske. Ali pa v spletni trgovini ustvari uporabnika, izvede veliko število naročil, ki jih nato takoj prekliče. Možna je tudi situacija, kjer hkrati generira večje število uporabnikov s provokativnimi imeni. Skratka, načinov kako izzivati skrbnika, da odpre stran z našim profilom, je več, in marsikateri je zelo učinkovit. Če je napadalec uspel v stran s profilom vnesti zlonamerno kodo, so 'Cookie' vrednost skrbnika ali pa njegova gesla zopet v nevarnosti.

Napad št. 3 – 'Stored' XSS napad preko seznama zadnjih prijav

Prejšnja dva primera sta se nanašala na situacije, kjer je dejansko obstajalo vnosno polje, ki ga aplikacija ni dovolj zaščitila. Vnosna polja so točke v aplikaciji, kjer je jasno, da uporabnik neposredno vpliva na besedilo, zato so dobre varovalke na takšnih točkah pogostejše.

Bistveno manj pogoste so varovalke na točkah, kjer ni takoj jasno, da gre za vnosni podatek uporabnika. Na primer aplikacija, ki beleži zadnje prijave. Aplikacija ob prijavi samodejno pobere iz http paketov podatke kot so IP naslov, iz katere spletne strani je uporabnik prišel ter tip brskalnika. Vse te podatke nato aplikacija izpisuje v funkciji »pregled zadnjih prijav«.

Vse lepo in prav, a tudi v tem primeru bi morala aplikacija pred beleženjem teh podatkov natančno preveriti vsebino. Čeprav ne gre za podatke, ki jih je neposredno vpisal uporabnik, lahko napadalec na njih vpliva. Podatek o tipu brskalnika je podatek, ki ga brskalnik samodejno vpiše v paket, preden ga pošlje aplikaciji. Napadalec lahko brez večjih težav pripravi skripto, kjer podatek spremeni tako, da mu doda zlonamerne 'javascript' ukaze.

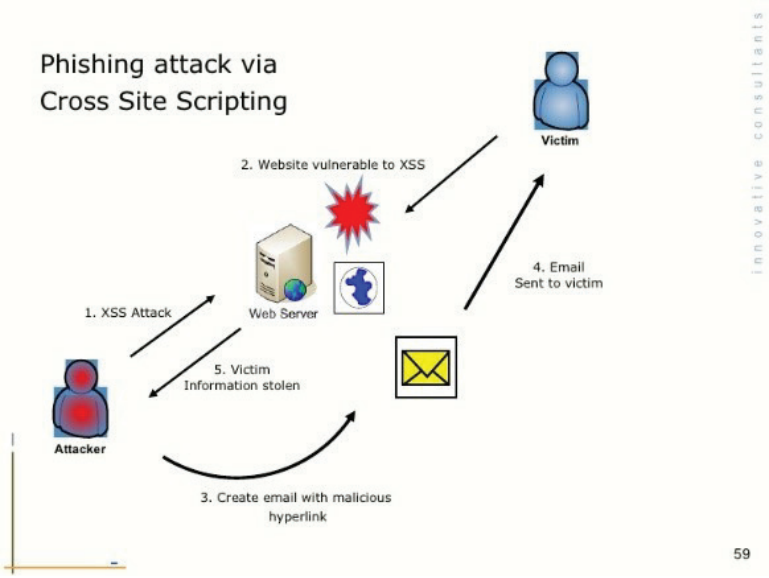
Gre za tipičen napad na skrbnika, saj ostali uporabniki praviloma ne obiskujejo pregleda drugih uporabnikov. Pri tem lahko napadalec izbere taktiko »potrpežljivega čakanja«, lahko pa izbere taktiko »provokacije«. Napadalec izvede mnogokratno prijavo ali pa stori nekaj nenavadnega z namenom, da skrbnik preveri, kdo je ta uporabnik in od kje se prijavlja.

Napad št. 4 – 'Reflected' XSS pri URL klicih

Prvi trije primeri so se nanašali na 'Stored' tip ranljivosti, ta primer pa se nanaša na 'Reflected' tip ranljivosti.

Zelo pogost primer je primer neobstoječih strani. Spletna stran dobi URL zahtevek. Če konkretna podstran obstaja, jo prikaže, če pa ne obstaja, pa uporabnika preusmeri na prvo stran. Pri tem pa se v glavo HTML strani zapiše nazadnje klicani URL.

Če v takšnem primeru napadalec pošlje URL naslov http://spletna.stran/...javascipt_koda..., bo aplikacija vrnila nazaj prvo stran (saj točno takšna podstran s takšnim naslovom ne obstaja),v glavo HTML strani pa bo brez preverjanja vpisala klicani URL naslov in pri prikazu bo brskalnik izvedel 'javascript' kodo.



Povedali smo že, da mora pri 'Reflected' tipu ranljivosti napadalec sedaj ta URL naslov nekako podtakniti žrtvi, da sama klikne na njega.

V osnovi je to relativno enostavno. Dejstvo je, da so 'phishing' e-mail napadi velikokrat uspešni, v tem primeru pa je podtaknjen URL naslov vsaj delno legalen. Začne se namreč z http://spletna.stran/... kar je nekaj, čemur uporabnik zaupa.

Vseeno pa ima napadalec manjšo težavo. Napadalec v takem URL linku ne more skriti drugega dela linka, kjer je napisana 'javascript' koda. Marsikateri uporabnik na ta del sicer ni pozoren, a je 'javascript' koda vseeno »govoreča« in čeprav uporabnik ne pozna 'javascript' ukazov, se mu določene besede hitro lahko zdijo sumljive.

Jasno je tudi, da v URL naslov ne moremo podtikati dolge kode, saj je URL naslov, ki se razteza »čez 3 strani«, že sam po sebi sumljiv.

Napadalec lahko naredi dvoje. Kot prvo bo pri 'Reflected' ranljivostih skoraj vedno uporabil podtikanje 'javascript' ukazov, ki bodo brskalniku velevali, da si 'javascript' kodo prenese iz drugih strežnikov. Hkrati pa bo tudi ta del 'javascript' kode skril v obliki ...document.write(String.CharCode(1,2,3,...))... Tu napadalec izkorišča zmožnost v 'javascript' jezika, kjer lahko na mestu ukaza navede string, sestavljen iz posameznih znakov, kjer znakov sploh ne navede, ampak navede zgolj 'char' kodo tega znaka. Tako je zlonameren link, ki ga podtakne svojim žrtvam, naslednji http://zapianja_vredna_stran/.../..document.write(String.CharCode(1,2,3,...))... Kar pa ni več tako sumljivo in je uspeh zavajanja žrtev realnejši.

Kaj natančno je namen napadalca, je nato odvisno od same aplikacije. Napadalec lahko tako uporabnikom krade 'Cookie' vrednosti, lahko poizkuša izvesti prevaro, s katero si bodo uporabniki naložili »exe« zlonamerne datoteke... Skratka, od tu dalje lahko napadalec vsebinsko izvaja vse tiste napade, ki smo jih že našteli v tem dokumentu.

7. poglavje

Kako se ubraniti pred Cross-site-scripting grožnjami?

Za konec poglejmo še, kako se lahko ubranimo pred XSS grožnjami. Načinov je več in marsikdaj je smiselna kombinacija uporabe več različnih načinov.

Filtriranje znotraj skript spletne aplikacije

Prvi del obrambe se nanaša na same spletne aplikacije. Izvirni greh pri aplikacijah izhaja iz dejstva, da aplikacija privzame vsebino, ki jo je posredno ali neposredno vnesel uporabnik. To vsebino nato uporabi pri pripravi HTML odgovorov za uporabnike, pri tem pa ne izvede preverjanja ali morebiti ta vsebina vsebuje kaj zlonamernega.

Skripte v aplikacijah morajo torej poskrbeti, da se vsi vhodni podatki ustrezno preverijo, vsebina, ki bi lahko prinašala nevarnost pa se mora bodisi odstraniti, bodisi preurediti, da ne predstavlja nevarnosti.

Class	Method	Results	
HTMLHelper	HTMLEncode(...)	In	<script>alert('XSS')</script>
		Out	<script>alert('XSS')</script>
	HTMLDecode(...)	In	<script>alert('XSS')</script>
		Out	<script>alert('XSS')</script>
	EncodeForHtmlAttribute(...)	In	"onmouseover='alert(document.cookie)'
		Out	"onmouseover='alert(document.cookie)'

Pri 'javascript' obrambi imamo na voljo to olajševalno okoliščino, da je 'javascript' jezik zelo odvisen od posebnih znakov. Tako je marsikdaj povsem dovolj, da na podatkih izvedemo kodiranje in vsi posebni znaki se bodo spremenili na ta način, da bo 'javascript' koda neveljavna, hkrati pa se bodo ti znaki še vedno pravilno prikazovali, če so slučajno sestavni del legalnih podatkov.

Kako točno to znotraj kode izvedemo, je za razvijalca zelo enostavno. Na voljo pa so tudi različne skripte za različna okolja; najdete jih na povezavah, ki jih navajamo v zadnjem poglavju.

Ne gre toliko za vprašanje, kako točno izvesti preverjanje podatkov, ampak bolj za vprašanje, ali se je razvijalec sploh spomnil, da bi bilo to nujno potrebno in ali je to preverjanje dejansko izvedel na vseh točkah in ne samo na nekaterih.
Napadalec bo preveril vse teoretično možne točke vnosa (nenazadnje mu pri tem pomagajo različni učinkoviti programi), tako da je za njega dovolj, da razvijalec pozabi na varovalko v eni sami točki.

Kot rečeno, vprašanje »kako narediti dober filter« ne bi smelo biti prvotnega pomena, saj je na voljo dovolj narejenih in mnogokrat preizkušenih filtrov. Vseeno pa še vedno srečamo aplikacije, kjer filtri obstajajo, a je kmalu jasno, da gre za unikatne filtre, ki jih je sprogramiral razvijalec. Sicer to še ne pomeni, da niso varni, a praksa se v praksi velikokrat izkaže nasprotno. Med dejanskimi pregledi smo zaznali primere kjer

- je aplikacija na vseh vnosnih podatkih iskala in brisala zapise <script>, a razvijalec je pri tem spregledal, da je to sicer najbolj tipičen način klicanja 'javascript' ukazov, ni pa edini.
- je aplikacije določene posebne znake spreminjala v neke druge znake in s tem kvarila 'javascript' sintakso. A napad je bil z uporabe »Replace« funkcije še vedno mogoč.
- je aplikacija omejevala izpis na 50 znakov, hkrati pa je odstranjevala vse zapise v obliki http:... Na videz je aplikacija tako preprečila vnos kompleksnejših 'javascript' zaporedij, hkrati pa je preprečila, da bi se vstavilo navodilo brskalnikom, da si 'javascript' privzamejo z drugih strežnikov. A preko 'String.CharCode' je možno takšno varovalko zaobiti.

Skratka, prvi del obrambe proti XSS napadom je ustrezno preverjanje podatkov v aplikaciji, kjer mora veljati naslednje:

- Preverjanje podatkov mora biti vključeno na vseh točkah, ki so neposredno vezane na vnosne podatke.
- Preverjanje podatkov mora biti vključeno na vseh točkah, ki so zgolj posredno vezane na vnosne podatke.
- Pri preverjanju podatkov priporočamo uporabo že narejenih in preverjenih skript in ne razvoj lastnih skript.

Nastavitev strežnika

Na uspešnost XSS napadov lahko vplivajo tudi nastavitve spletnih strežnikov. Omenimo predvsem dve:

- 'Cookie' mora biti označen kot 'http-only'. 'Http-only' oznaka preprečuje 'javascript' ukazom dostop do vrednosti 'Cookie' podatkov. Zaščita je dokaj enostavna in je zelo učinkovita. Tovrstna nastavitev je na žalost mnogokrat spregledana.
  Vendar pozor. To je zgolj zaščita pred napadi, ki imajo za cilj krajo 'Cookie' vrednosti. Velikokrat se zgodi, da skrbniki poskrbijo za nastavitev, a niso pozorni na XSS ranljivosti. Kar pa ni prav, saj smo v dokumentu pokazali, da kraja 'Cookie' vrednosti ni edina grožnja 'javascript' ukazov.

- TRACE metoda mora biti deaktivirana. To pravilo sicer počasi zamira. Gre namreč za to, da vklopljena TRACE metoda negira 'http-only' nastavitev na strežniku. Tehnično gledano je znan napad, kjer napadalec preko XSS ranljivosti lahko pridobi 'Cookie' vrednost, četudi ima ta vklopljeno 'http-only' zaščito. V praksi pa temu ni več tako, saj sodobni brskalniki že sami po sebi dokaj učinkovito negirajo uporabo TRACE klicev. Situacija, ko ima strežnik vklopljeno TRACE metodo, niti ne več tako kritična. A vseeno še vedno priporočamo deaktivacijo TRACE metode, saj je mnogokrat popolnoma nepotrebna.

Omembe vredna je še ena nastavitev, in sicer vzpostavitev SSL protokola. Vsi vemo, da je SSL primarno namenjen šifriranju podatkov in na prvi pogled ni takoj jasno, kakšna je njegova povezava z XSS ranljivostmi.

Obstajajo primeri, ko aplikacije v HTML odgovore ne vrivajo celotne 'javascript' kode, ampak povsem zadostuje, da brskalniku naložijo nalogo, da si kompleksnejšo skripto pri prikazovanju HTML strani enostavno naloži iz tretjega strežnika. Tu pride do izraza delovanje sodobnih spletnih brskalnikov. V primeru, da naša aplikacija uporablja SSL protokol, ti ne bodo dopustili, da se v prikaz HTML strani vključi skripta s tretjega strežnika, četudi tretji strežnik nima pravilno vzpostavljenega SSL protokola. Napadalec bo moral svoj strežnik vpisati v DNS, moral si bo priskrbeti zaupanja vreden digitalni certifikat, sicer brskalniki žrtev s tega strežnika ne bodo hoteli zaganjati 'javascript' ukazov.

To za napadalca sicer ni nekaj, kar je nemogoče izvesti, a je vseeno to v določenih primerih preveč in pretežko in bo napad raje opustil.

Nastavitve poštnih sistemov

Nastavitve poštnih sistemov so lahko pri obrambi zelo pomembne, saj smo pokazali, da je 'phishing' napad na žrtev marsikdaj sestavni del napada. Posebej to velja za 'Reflected' tip ranljivosti, ko mora napadalec žrtev prepričati, da klikne na zlonamerni link. V praksi se to največkrat izvede s pomočjo 'phishing' e-mail napada.

Glede poštnih sistemov izpostavljamo dvoje:

- Poštni sistemi bi morali zaustavljati pošto, ki prihaja od zunaj, a hkrati navaja, da jo pošilja notranji uporabnik.
- Poštni sistemi bi morali zaustavljati pošto, ki prihaja od zunaj, a hkrati navaja, da jo pošilja uporabnik iz domene, ki v resnici ne obstaja.

S temi nastavitvami seveda ne preprečimo vseh 'phishing' e-mail napadov, vsekakor pa praksa pokaže, da vidno zmanjšamo verjetnost uspeha 'phishing' e-mail napadov.

WAF – Web Application Firewall

Marsikatero zlorabo spletnih aplikacij preprečimo tako, da skripte spletne aplikacije podrobno preverjajo določeno vsebino. Tudi pri XSS ranljivostih je to nujno potrebno.

Takšno preverjanje lahko namesto aplikacije izvaja WAF. WAF je naprava, ki stoji pred spletnimi aplikacijami in je specializirana za zaznavanje in preprečevanje cele vrste znanih in neznanih zlorab spletnih aplikacij.

Pri WAF je potrebno razumeti, da so tovrstni sistemi lahko zelo učinkoviti pri preprečevanju t. i. tehničnih ranljivostih. So pa omejeni pri preprečevanju ranljivosti, ki izhajajo iz »napačne« logike aplikacij. Včasih je spletne aplikacije možno zlorabiti tako, da ne izvajamo tehničnih napadov z nenavadnimi klici in znaki, ampak uporabimo aplikacijo na drugačen način, kot je predvideno.

XSS ranljivosti sodijo v klasične tehnične ranljivosti in WAF sistem aplikacije lahko zelo učinkovito in uspešno zaščiti pred tovrstnimi ranljivostmi.

Vendar pozor. WAF je sistem, ki varuje aplikacije in ne uporabnike. XSS napad pa je usmerjen na uporabnike. Uporabnike v podjetju WAF zavaruje v smislu, da ni mogoč XSS napad preko njihove spletne aplikacije.

Symantec Web Isolation

Če je WAF namenjen varovanju naše spletne strani in aplikacij, pa je Web Isolation rešitev, ki neposredno varuje uporabnike pred različnimi zlorabami, ki izhajajo iz obiska spletnih strani v Internet omrežju, vključno seveda z XSS zlorabami.

Sistem zaščite deluje tako, da uporabniki ne dostopajo neposredno do spletnih aplikacij izven našega omrežja, ampak do njih dostopajo preko tega sistema. Sistem namesto uporabnikov brska po spletnih straneh, uporabnikom pa nato posreduje samo vizualno sliko uporabe. 'Javascript' in podobni ukazi se tako ne izvajajo v brskalnikih uporabnikov, ampak na vmesnem sistemu. Uporabnik ima z aplikacijo tako enako vizualno in vsebinsko izkušnjo, istočasni pa je učinkovito zaščiten pred napadi, tudi XSS napadi.

8. poglavje

Viri in dodatne informacije

Razlage 'Cross-site-scripting' ranljivosti

https://en.wikipedia.org/wiki/Cross-site_scripting
https://www.acunetix.com/websitesecurity/cross-site-scripting/
https://www.acunetix.com/websitesecurity/xss/
https://www.youtube.com/watch?v=r79ozjCL7DA
https://www.youtube.com/watch?v=M_nIIcKTxGk

Zaščita pred 'Cross-site-scripting' ranljivostmi

https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
https://www.techieweblog.com/network-security-techie/how-to-protect-from-xss-attack/
https://www.checkmarx.com/2017/10/09/3-ways-prevent-xss/
https://www.wordfence.com/learn/how-to-prevent-cross-site-scripting-attacks/

Ostale informacije povezave s 'Cross-site-scripting' ranljivostmi

https://en.wikipedia.org/wiki/Web_application_firewall
https://www.symantec.com/products/web-isolation

Preverite, katerim varnostnim tveganjem in ranljivostim je izpostavljen vaš informacijski sistem.

Naši strokovnjaki vam bodo pomagali pri oceni varnostnih tveganj in na kakšen način pristopiti k reševanju XSS ranljivosti.

Vprašajte našega strokovnjaka za kibernetsko varnost Vladimirja Bana.

vladimir.ban@smart-com.si

SMART COM

Smart Com ima 27-letno tradicijo na področju IKT in je eden izmed vodilnih sistemskih integratorjev v Jugovzhodni Evropi. Naši strokovnjaki so specialisti za povezovanje vrhunskih tehnologij v sisteme, ki za stranke predstavljajo poslovne rešitve z visoko dodano vrednostjo. Mrežna infrastruktura, kibernetska varnost in nadzorni sistemi so naše najmočnejše strokovne specializacije. Prizadevamo se za vzpostavljanje partnerstev, ki temeljijo na medsebojnem zaupanju za udejanjanje strateških in poslovnih vizij.

www.smart-com.si

© Smart Com