Софийски университет

„Св. Климент Охридски" гр. София

Факултет по математика и информатика

ДОКУМЕНТАЦИЯ

към проект по
„Основи на сигурното уеб програмиране"

Тема: „Blind XPath Injection"

Изготвил:
Лилия Иванова Божанина, ФН: 71873, ИС 3 курс
Преподавател:
гл. ас. Филип Петров

Съдържание

1. Въведение

XPath Injection [8] е атака, насочена към уеб сайтове, които създават XPath заявки от предоставени от потребителя данни. Ако приложението вгражда незащитени данни в заявка за XPath, заявката може да бъде променена, така че да не се анализира повече по първоначално предвидения начин. Това може да се направи, като се заобиколи системата за удостоверяване на уеб сайта и се извлече структурата на един или повече XML документи в сайта.

XPath 1.0 [1] е език, използван за препратка към части от XML [4] документ. Може да се използва директно за заявка на XML документ от приложение или като част от по-голяма операция, като например прилагане на XSLT [3] трансформация към XML документ или прилагане на XQuery [2] към XML документ.

Синтаксисът на XPath прилича на SQL заявка и наистина е възможно да се формират подобни на SQL заявки в XML документ, използвайки XPath. Да приемем например, че XML документ съдържа елементи с името „потребител", всеки от които съдържа 3 под елемента - „име", „парола" и „акаунт". Следният израз на XPath дава номера на акаунта на потребителя, чието име е "jsmith" и чиято парола е "Demo1234" (или празен низ, ако такъв потребител не съществува):

```
string(//user[name/text()='jsmith' and
password/text()='Demo1234']/account/text())
```

2. Последици за сигурността

XPath се използва, наред с други неща, за запитване на XML бази данни в стил, подобен на примера по-горе. XML документ служи като база данни, а заявката XPath е аналогична на SQL заявка. Има ползи от използването на XML база данни; например: преносимост, съвместимост, повторна употреба на документа, изчистеност на документа и резултатите от заявката, както и наличието на структура за документа и резултатите от заявката.

Различни продукти предлагат собствени XML бази данни с вградено XPath средство за заявки. [6] Сред тези продукти са Tamino на Software AG, Apache Software Foundation XIndice, Sleepycat Software Berkeley DbXML, dbXML Group dbXML. Приложение, което използва такъв продукт, може да бъде уязвимо за Blind XPath Injection в зависимост от това как заявката XPath е формулирана от приложението. В други контексти XPath може да се използва директно (т.е. да не се използва софтуер за база данни) и се поддържа изобщо - напр. в .NET framework на Microsoft и в ColdFusion / MX на Macromedia.

XPath се използват за заявки за търсене, за обработка на вход, за извличане на данни и накратко за гъвкави, леки задачи на базата данни.

След като забележи уязвимостта на XPath Injection в приложение, базирано на XPath, нападателят не трябва да разбира напълно заявката XPath. В рамките на няколко опита атакуващият обикновено може да генерира данни за заявка „шаблон", които могат да се използват за Blind XPath Injection. От този момент автоматичен скрипт може да се използва за извличане на пълния XML документ, който е основната база данни.

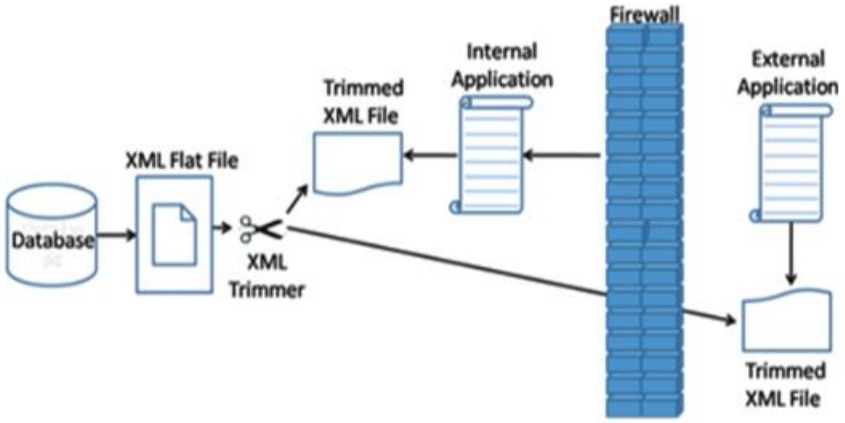
Трябва да се подчертае, че хакерът не трябва да знае точната структура на заявката XPath и освен това, тъй като XML документ няма свързана с него система за контрол на достъпа/привилегии, хакерът е в състояние да извлече документа (базата данни) в неговата пълнота - за разлика от SQL инжектирането, където нападателят е ограничен до привилегиите на акаунта на базата данни, използван от приложението.

3. Управление на XML документи

XML документите могат да се управляват чрез прилагане на следните принципи:

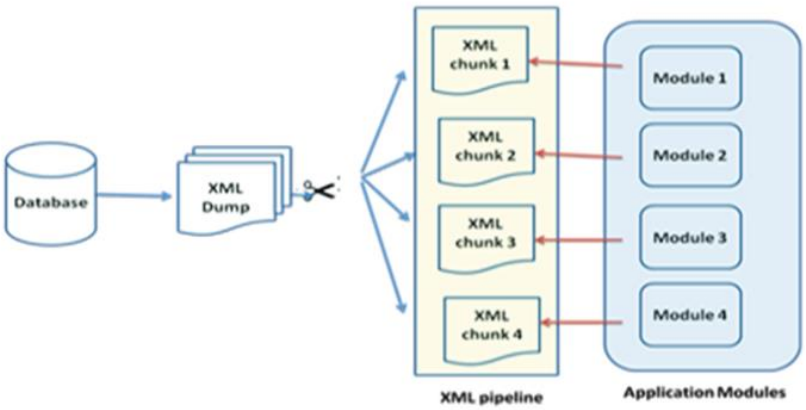
Ако някое приложение има достъп до част от XML документ, имайте предвид, че целият този XML документ е косвено изложен на приложението и неговите потребители.

XML Sorce файлът за приложение не съдържа цялата XML база данни. Той трябва да включва само информацията, от която приложението се нуждае за своята работа. Това е възможно чрез предоставяне на XML базирани данни на базата на необходимост да се знае чрез подрязване на XML файла.



Схемата показва процедурата за изрязване на XML файл.

В този сценарий потребителите все още използват XML дъмп, но нито едно приложение не осъществява пряк достъп до този пълен документ. По-скоро за всеки се получава отделно подмножество на XML файла заявление на базата. Това може да стане чрез автоматизиран процес, наречен „XML тример". Ето как неразрешеният достъп до информация може да се контролира чрез изолиране на приложения от оригиналната база данни.



Приложението трябва да бъде проектирано по такъв начин, че да има достъп до XML данни чрез XML тръбопроводи. Приложението трябва да получава данни само при необходимост. Архитектурата на XML конвейер разделя цялата заявка на по-малки части. Архитектурата на Pipeline поддържа яснотата на потока на всички данни в XML приложения. Тази яснота позволява да се оценят потенциалните заплахи.

4. XPath Syntax

4.1 Избор на възли [7]

nodename - Избира всички възли с името "nodename"

/ - Избира от основния възел

// - Избира възли в документа от текущия възел, които съответстват на избора, независимо къде се намират

. - Избира текущия възел

.. - Избира родителя на текущия възел

@ - Избира атрибути

4.2 Избиране на неизвестни възли

*- Съответства на всеки елемент възел

@* - Съответства на всеки възел на атрибут

node() - Съвпада с всеки възел от всякакъв вид

4.3 Предикати

/ bookstore / book [1] - Избира първия елемент на книгата, който е дъщеря на елемента на книжарницата.

/ bookstore / book [last ()] - Избира последния елемент на книгата, който е дъщеря на елемента на книжарницата

/ bookstore / book [last () - 1] - Избира последния, но един елемент на книгата, който е дъщеря на елемента на книжарницата

/ bookstore / book [position () <3] - Избира първите два елемента на книгата, които са деца на елемента на книжарницата

// title [@lang] - Избира всички елементи на заглавието, които имат атрибут с име lang

// title [@ lang = 'en'] - Избира всички елементи на заглавието, които имат атрибут "lang" със стойност "en"

/bookstore/book[price>35.00] - Избира всички книжни елементи на елемента на книжарницата, които имат елемент на цена със стойност по-голяма от 35,00

/bookstore/book[price>35.00]/title - Избира всички заглавни елементи на елементите на книгата на елемента на книжарницата, които имат елемент на цена със стойност по-голяма от 35,00

4.4 Избиране на няколко пътя

// book / title | // book / price - Избира всички заглавни И ценови елементи на всички елементи на книгата

//title | //price - Избира всички заглавни и ценови елементи в документа

/bookstore/book/title | //price - Избира всички заглавни елементи на елемента книга на елемента на книжарницата И всички елементи на цената в документа

5. XPath Injection

Помислете за уеб приложение, което използва XPath, за да поиска XML документ и да извлече номера на акаунт на потребител, чието име и парола са получени от клиента. Такива приложения могат да вграждат тези стойности директно в заявката XPath, като по този начин създават дупка в сигурността.

Ето пример (Microsoft ASP.NET и C #):

```
...

XmlDocument XmlDoc = new XmlDocument();

XmlDoc.Load("...");

... XPathNavigator nav = XmlDoc.CreateNavigator();
XPathExpression expr =
nav.Compile("string(//user[name/text()='"+TextBox1.Text+ "' and
password/text()='"+TextBox2.Text+ "']/account/text())");

String account=Convert.ToString(nav.Evaluate(expr));

if (account=="")

{

// name+password pair is not found in the XML document –
```

```
// login failed.

...

}

else

{

// account found -> Login succeeded.

// Proceed into the application.

...

}
```

Когато се използва такъв код, нападателят може да инжектира XPath изрази (подобно на SQL инжектиране), напр. предоставете следната стойност като потребителско име:

```
' or 1=1 or ''='
```

Тези данни водят до промяна на семантиката на оригиналния XPath, така че винаги връща първия номер на акаунта в XML документа. Такава атака се нарича „Xpath Injection", аналогия с атаките „SQL инжекция" води до това, че нападателят е влязъл в системата (като първия потребител, изброен в XML документа), въпреки че нападателят не е предоставил валидно потребителско име или парола.

Въпреки че тази атака предоставя на атакуващия достъп до приложението, не е задължително да им предоставя достъп като най-привилегирован акаунт. Всъщност, с изключение на влизането, нападателят не е получил информация за XML „базата данни на акаунта". В някои случаи може да е възможно да се получи информация от системата, ако изразът XPath връща данни от XML документа, който по-късно се показва на потребителя (атакуващ). Например горният код може да показва номера на акаунта на влезлия акаунт в HTML отговора. В този случай нападателят може допълнително да манипулира заявката XPath, за да принуди сървъра да върне различни части от документа. Например, следното впръскване (за параметъра на потребителското име, като същевременно се предоставя стойността „NoSuchPass" като парола) ще върне първия елемент от XPath възела, определен от възела P:

```
NoSuchUser'] | P | //user[name/text()='NoSuchUser
```

Това ще формира следната заявка за XPath:

```
string(//user[name/text()='Foobar'] | p |
//user[name/text()='NoSuchUser' and
password/text()='NoSuchPass']/account/text())
```

Тъй като първият и третият предикат винаги са фалшиви, заявката връща stringvalue [1] на възела-набор P.

Имайте предвид обаче, че в този случай, макар че е възможно да се извлекат повечето данни от базата данни, са необходими някои предварителни познания за заявката XPath. Без да знаем точния формат на заявката, е много трудно да разберем как точно да формираме инжекционния низ. Това е особено вярно, ако съобщенията за грешки се потискат. Обърнете внимание, че по този начин също е невъзможно да се извлекат невъзстановени изрази (например count () и name ()).

Има много случаи, в които не се изпращат XML данни директно на потребителя. В този случай горната атака (и подобни атаки) не е приложима.

6. Blind XPath Injection

Възможно е да се възприеме по-систематичен подход към проблема с инжектирането на XPath. Този подход се нарича „сляпо инжектиране" (основите на което са заложени в контекста на SQL инжектирането [5]). Той предполага повече или по-малко в структурата на заявката, освен че потребителските данни се инжектират в булев контекст на израза. Позволява на атакуващия да извлече един бит информация за една инжекция на заявка. Този бит се реализира, например, като „Успешно влизане" или „Неуспешно влизане".

Този подход е дори по-мощен при XPath, отколкото при SQL, поради следните характеристики на XPath:

- XPath 1.0 е стандартен език. SQL има много диалекти, всички базирани на общ, относително слаб синтаксис.
- XPath 1.0 позволява да се направи запитване към всички елементи от „базата данни" (XML обект). В някои SQL диалекти е невъзможно да се направи заявка за някои обекти от базата данни, като се използва SQL SELECT заявка (напр. MySQL не предоставя „таблица на таблици").
- XPath 1.0 няма контрол на достъпа за „базата данни", докато в SQL някои части от базата данни може да са недостъпни поради липса на привилегии за приложението.

Техниката, която използваме, е следната:

Първо показваме как да обходим документ XPath, като използваме само скаларни заявки (т.е. заявки, чийто тип на връщане е „низ", „числов" или „булев")). Процедурата на обхождане не предполага познаване на структурата на документа; но в края документът в своята цялост е реконструиран.

След това показваме как скаларна XPath заявка може да бъде заменена от поредица от логически заявки. Тази процедура се нарича „булеанизация" на заявката. Булева заявка е заявка, чийто резултат е булева стойност (вярно / невярно). Така че в процес на булеанизация, заявка, чийто тип резултат е низ или цифров, се заменя с поредица от заявки, чийто резултат е булев, и от която можем да възстановим резултата от оригиналния низ или числова заявка.

И накрая, всяка булева заявка може да бъде разрешена чрез еднократно „сляпо" инжектиране. Тоест, показваме как е възможно да се формира низ за инжектиране, включително булева заявка, която, когато се инжектира в XPath заявка, кара приложението да се държи по един начин, ако булевата заявка се разреши в „true", и по друг начин ако заявката се превърне във „false". По този начин нападателят може да определи един бит - булевия резултат от заявката.

Новостта в този подход към XPath Injection е, че той не изисква много предварителни познания за формата на заявка XPath, за разлика от „традиционния" подход, описан по-горе. Не изисква данните от XML документа да бъдат вградени в отговора и, че целият XML документ в крайна сметка се извлича, независимо от формата на заявката XPath, използвана от приложението. Той използва само разлика в поведението на приложението, произтичаща от разлика във връщаната стойност на заявката XPath, за да извлече един бит информация.

7. XPath Crawling

При условие, че има път за текущия елемент (path), можем лесно да продължим.

- Рекурсията започва с път = ''.
- Името на елемента (включително име на пространство от имена) е дадено като име (path), а стойността на пространството от имена е namespace-uri (path).
- Броят на атрибутите на елемента е брой (път / атрибут :: *), името на N-тия атрибут е име (път / атрибут :: * [позиция () = N]), стойността на пространството от имена на N-ия атрибут е namespaceuri (път / attribute ::

* [position () = N]) и стойността на N-тия атрибут е path / attribute :: * [position () = N].

Има четири типа подвъзли: текст, инструкция за обработка (съкратено като „PI"), елемент и коментар. Дразнеща странност на XPath 1.0 е, че е възможно да се изброяват над подвъзлията, но е невъзможно директно да се извлече типа на възела.

Има обаче едно просто решение.

Първо, трябва да знаем броя на различните подвъзли:

count (path / child :: node ()) - броят на всички възли за дадения път.

count (path / child :: text ()) - брой текстови полета деца (до 1 ...).

count (path / child :: comment ()) - брой възли на коментари.

count (path / child :: *) - броят на дъщерните елементи.

count (път / дете :: обработка-инструкция ()) - броят на PI възлите.

Номерът е винаги да поддържате текущия брой възли от всеки срещан досега тип. Това ни позволява да знаем какъв индекс от всеки тип е кандидат за следващия възел. Следователно имаме до четирима кандидати. За тази техника е задължително да не се изброява кандидат от тип, който е изчерпан - оттук и необходимостта първо да се разбере колко подвъзла се очакват от всеки тип. Да предположим, че имаме броячи i, j, k, l за текстови подвъзли, подвъзли за коментари, съответно подвъзли на елементи и PI подвъзли. И нека приемем, за простота, че нито един от типовете не е изчерпан, т.е.

```
i < count(path/child::text()) and j <
count(path/child::comment()) and k < count(path/child::*) and l <
count(path/child::processing-instruction())
```

Сега разглеждаме следния израз на възел:

```
path/child::node()[position()=((i+j+k+l+1)] |
path/child::text()[position()=(i+1)]
```

Ако следващият подвъзел (номер i + j + k + l + 1) наистина е следващият текстов подвъзел (номер i + 1), тогава двата набора от възли, които са обединени, всъщност са един и същ набор от възли , който съдържа точно един възел. Ако обаче следващият подвъзел не е текстов подвъзел, тогава обединението ще създаде два възела. Следователно можем да знаем дали следващият подвъзел е текстов подвъзел, като попитаме:

```
count(path/child::node()[position()=((i+j+k+l+1)] |
path/child::text()[position()=(i+1)])=1
```

Истинска стойност показва, че следващият подвъзел е текстов подвъзел, а фалшива стойност показва, че не е. По същия начин следващите три заявки могат да се използват, за да се определи дали подвъзелът е подвъзел за коментар, подвъзел на елемент или подвъзел PI:

```
count(path/child::node()[position()=((i+j+k+l+1)] |
path/child::comment ()=(j+1)])=1
count(path/child::node()[position()=((i+j+k+l+1)] |
path/child::*()[position()=(k+1)])=1
count(path/child::node()[position()=((i+j+k+l+1)] |
path/child::processing-instruction()[position()=(l+1)])=1
```

Стойността на текстов подвъзел е:

```
path/child::node()[position()=N]
```

Подвъзел за коментар има следните данни:

```
path/child::node()[position()=N]
```

За да се обработи подвъзел на елемент, е необходима рекурсия:

```
path/child::node()[position()= N]
```

И подвъзелът с инструкции за обработка има име:

```
name(path/child::node()[position()=N]),
path/child::node()[position()=N]
```

8. Булеанизация на скаларни заявки на XPath

Сега ще опишем как заявка, чийто тип на връщане е низ или число, може да бъде заменена с последователност от XPath заявки, чийто тип на връщане е булев. Нека приемем, че Q е числова XPath заявка. Нека освен това приемем, че обработваме 32-битови подписани цели числа (което е достатъчно в 32-битова архитектура). Първо извличаме знаковия бит на Q:

```
(Q>=0)
```

Истинската стойност показва, че Q е положителна (или нула), а фалшивата стойност показва, че Q е отрицателна. След това използваме –Q (вместо Q), ако е отрицателно, и продължаваме. Ако приемем, че Q е положителен, извличаме

неговите 31 бита, като също така приемаме, че вече знаем най-значимите N бита (от 31 бита), тогава можем да намерим следващия бит: Нека K е числото, образувано от известните N високи бита, тогава N + 1 битът е настроен на 1, след това останалите, 30-N бита, зададени на 0.

```
((Q-K)>=0)
```

Дава true, ако N + 1 битът (отляво) е 1, и false, ако този бит е 0.

По този начин можем да възстановим положително Q с 31 булеви заявки. Започваме с N = 0 и итеративно извличаме следващия бит, докато стигнем до N = 30 включително.

Заявка за низ S първо се разчита на байтове (или по-точно символи на Unicode), както следва:

Първо, ние правим заявка за дължината на низа, като използваме функцията XPath за дължина на низа, която е числова заявка:

```
(string-length(S))
```

След това можем да прегледаме символите, като намалим заявката в серия от един байт (символ) заявки:

```
(substring(S,N,1))
```

Сега един байт / символна заявка B от своя страна се редуцира в булеви заявки, както следва: Нека приемем, че списъкът с възможни символи (с изключение на двойната кавичка) в документа е известен (обозначете го със C) и, че дължината на списъка е L. Ако е известно, че XML документът всъщност се състои от ASCII символи за печат, включително CR, LF и HT, с изключение на двойни кавички, тогава L е 97. Ние индексираме всеки възможен символ, започвайки от 0 и отивайки до (L-1) . Обозначава се с K = таван (log2 (L)) - това е броят на битовете, които са необходими за определяне на символа. Сега подготвяме K низове с дължина L. N-тият низ е списък с битове в позиция N на символите. Нека определим N-тия низ като CN.

Първо, гарантираме, че байтът не е двойна кавичка:

```
(B='"')
```

Ако изразът връща вярно, тогава байтът е просто двойната кавичка. Ако изразът е невярен, действаме както следва: N-тият бит се извлича по следния начин

```
(number(translate(B,"C","CN"))=1)
```

Ако това даде истина, тогава N-тият бит е 1, а ако даде false, N-тият бит е 0. Забележка, трябва да изключим двойната кавичка от C, иначе синтаксисът XPath ще бъде нарушен. По този начин можем да извличаме низови заявки, използвайки булеви заявки.

9. Mounting Blind XPath Injection

Помислете за следния низ на заявка XPath (от горния код C#):

```
"string(//user[name/text()='"+TextBox1.Text+ "' and
password/text()='"+TextBox2.Text+"']/account/text())"
```

Сега, инжектиране на потребителско име на

```
NoSuchUser' or E or 'foobar'='
```

принуждава предиката да даде true, ако E е булев израз, който оценява на true, и да даде false, ако E се оценява като false. Следователно приложението влиза успешно, ако E е вярно, и отхвърля опита за влизане, ако E е невярно. Сега имаме механизъм, който извлича един-единствен бит от системата - Boolean XPath израз E. Както видяхме по-горе, това е достатъчно за извличане на целия основен XML документ.

Всъщност, скрипт на Perl, прилагащ алгоритъм, който демонстрира тази атака, е написан от Ory Segal и Ronen Heled. Този алгоритъм успешно е извлякъл кратък XML документ (от приложение XPath, предоставено от Chaim Linhart от Sanctum Inc.), като рекурсивно DFS обхожда документа и изпраща булеанизирани заявки под формата на Blind XPath Injection. Както може да се анализира, алгоритъмът има сложност по време на изпълнение (брой заявки) от O (размер на документа). Размерът на заявката е ограничен (до постоянни режийни разходи) от размера на ефективната азбука на документа, L (напр. Само за ASCII документи - 97).

Интересно е да се отбележи, че има много малко варианти на горното инжектиране, които покриват по-голямата част от възможното инжекционно „пространство". По-конкретно, не се изисква (в повечето случаи) да се знае точно структурата на заявката XPath. В това се крие силата на атаката за Blind XPath Injection.

10. Недостатъци на настоящия алгоритъм

- Не ASCII символи и специални символи при булеанизацията на една символна заявка, трябва да изпратим на сървъра XPath заявка, състояща се от всички възможни символи, очаквани в XML документа. Това може да е проблематично в някои случаи - сървърът може да отхвърли такива заявки поради специфични ограничения за символи (например „<", CR, LF, HT). Когато се очакват ASCII документи, това не е проблем (поне не в ASP.NET 1.0).
- Размер на заявката
  Когато размерът на азбуката на документа (L) е голям, заявките за булеанизация на символа (чийто размер е малко по-голям от L) могат да бъдат отхвърлени поради ограничения на дължината (за да се обхване общ документ на Unicode, ще бъдат изпратени около 65000+ символа ...).
  Възможно е да замените дългите низове от възможни символи със сегменти от възможни символи, като по този начин се търгува брой заявки за размера на заявката. В краен случай е възможно да се изпратят множество прости булеви изрази, като (приемем, че s е единичен символ):
  `(B='s')`
  Това, разбира се, се отразява на сложността на изпълнението - в краен случай, умножавайки го по L.
- Несъвършена реконструкция на XML документа
  XML документ може да съдържа атрибути xmlns (обозначаващи пространства от имена), атрибути xml: space (управляващи начина, по който се обработват интервалите) и атрибути xml: lang (определящи езици). Също така, XML декларацията не е достъпна. Освен това, това, което е достъпно, е логическият документ, където DTD, атрибутите и стойностите по подразбиране, XML обектите и CDATA вече са разширени.
- Производителност
  Описаният по-горе алгоритъм не е оптимизиран за скорост. Има някои области, в които може да се приложи оптимизация, например прогнозиране на „разумен" брой битове на цяло число (обикновено много по-малко от 31) и паралелизиране на някои заявки. Булеанизацията на числовата заявка може да се извърши по начин, който позволява паралелно изпращане на всички заявки, като се използват числовите оператори за разделяне и "mod", а байтовото булеанизиране и факторингът от низове до байтове определено могат

да се паралелизират. Паралелното изпращане на няколко запитвания прави много по-добро използване на ресурсите на нападателя.

11. Защита срещу XPath Injection

Защитата срещу XPath Injection е по същество подобна на защитата срещу SQL инжекция. Приложението трябва да дезинфекцира потребителския вход. По-конкретно, символите с единични и двойни кавички трябва да бъдат забранени. Това може да се направи в кода на приложението или външно чрез разполагане на защитна стена за уеб приложения, пред приложението. AppShield може да се използва като решение за предотвратяване на атаки на ниво приложение, като XPath Injection, SQL инжекция и скриптове между сайтове. Разбирайки логиката на приложението и налагайки я, AppShield може да предпазва уебсайтовете от атаки, като същевременно безпроблемно разрешава валидни взаимодействия със сайта. В случай на XPath Injection, AppShield ще гарантира, че скритите полета, които могат да бъдат използвани в контекста на XPath заявки, не се променят и че въвеждането от потребителя (напр. Текстови полета и полета за парола) не съдържа опасни символи или комбинации от знаци, които могат да допринесат до промяна на семантиката на заявката XPath.

Тестване на податливостта на приложението към XPath Injection може лесно да се извърши чрез инжектиране на единична или двойна кавичка и проверка на отговора. Ако е възникнала грешка, вероятно е възможно инжектиране на XPath. Този процес на тестване може да бъде автоматизиран с помощта на инструмент за автоматизирано тестване на сигурността, който тества точно тази уязвимост и точно проверява резултатите от теста.

12. Заключение

Видяхме, че е възможно да се извлече пълният XML документ, използван в заявка за XPath, с много малко познания за заявката XPath и без да е необходимо да се получават данни от отговора, като се използва метод, който наричаме „Сляпо инжектиране на XPath.“ Тази атака е осъществима от гледна точка на време на изпълнение и е внедрена в Perl и успешно монтирана на някои примери за XPath / XML.

Следователно приложенията, които вграждат потребителски данни в XPath заявки по несигурен начин, са податливи на този вид атака и XML документите, използвани от XPath заявките, могат да бъдат компрометирани.

Възможни са мерки за защита и тестване за защита на приложенията срещу XPath Injection.

13. Използвани източници

[1] "XML Path Language (XPath) Version 1.0 - W3C Recommendation", http://www.w3.org/TR/xpath

[2] "XQuery 1.0: An XML Query Language - W3C Working Draft", http://www.w3.org/TR/xquery

[3] "XSL Transformations (XSLT) Version 1.0 - W3C Recommendation", http://www.w3.org/TR/xslt

[4] "Extensible Markup Language (XML) 1.0 (Second Edition) - W3C Recommendation", http://www.w3.org/TR/REC-xml

[5] "Using Binary Search with SQL Injection", Sverre H. Huseby, http://shh.thathost.com/text/binary-search-sql-injection.txt

[6] "XML Database Products – Native XML Databases", Ronald Bourret, http://www.rpbourret.com/xml/XMLDatabaseProds.htm#native

[7] "XPath Syntax" https://www.w3schools.com/xml/xpath_syntax.asp

[8] "XPath Injection", Tech Target,

https://searchsoftwarequality.techtarget.com/definition/XPath-injection