Phish Feeding: An Active Response to Phishing Campaigns

John Brozycki, CISSP
phishfeeder@trueinsecurity.com
July, 2006

Section I: Phishing and Phish Feeding

The Phishing Problem

Most financial institutions can count themselves as victims of Internet phishing schemes. In a typical Internet-based phishing scheme, either an existing Internet web server is compromised or a new web server is fraudulently established. A web site is set up to appear to be the legitimate website of the target institution. Pages and graphics are often copied directly from the targeted institution's real web site. Emails are then sent to a list of email addresses. The emails falsely appear to come from the targeted institution. Phish emails may be targeted or may be sent indiscriminately to a list of email addresses. Phishers may employ data mining techniques to make their targeting more effective. They may use web crawlers to harvest addresses from lists, groups, and web postings. They may also utilize geographical targeting, using only email domains for entities that are geographically close to the targeted institution. They can buy lists that are organized by geography, domain, and other criteria. Sometimes the targeting appears so accurate that the institution may feel that the email list must have been taken directly from the institution's own systems. Other times, the phish emails may be so poorly targeted that few if any actual customers receive it.

An effective phish email is typically comprised of three components. The first component is the hook. The hook tells you why you should trust what the email is saying. For example, "We detected fraudulent attempts to access your account." The second component is the required action. The required action is what the phishers want you to do. For example, the phish may ask you to email personal account information or click on a link purporting to take you to the targeted institution's web site to provide personal account information. The third component is the push. The push provides incentive to carry out the required action before you either forget or the phish is taken down. An example of a push is "If you do not update your account information within 24 hours, we will lock access to your account." Phishers continually evolve their pitches, fooling people into disclosing personal financial information. These changes are often enough to fool people even if they have been phished before.

Institutions often feel helpless when dealing with a phishing attack. The phish site, which is often run from a legitimate web server compromised by the phishers, may take hours, days, or longer to get taken down. Customers receiving phish emails often believe they are sent by the institution (especially since phish use forged sender and reply to fields to make it appear to come from the target institution), but the institution probably doesn't even learn of the phish until a customer forwards it to them or calls to complain. "Blow back" including out of office replies, invalid destination messages from mail servers, and inquiry replies from recipients of the phish may overwhelm the targeted institution's mail systems.

Phish Feeding

What can the targeted institution do? They can try to contact the web master, the ISP, perhaps the appropriate CERT, and law enforcement, and then wait for the fake site to be removed. Meanwhile, the phish site is active. During this time customers that fall victim to

the phish may suffer identity loss and the target institution likely incurs financial loss. The desire to take an active response may be strong but black hat techniques, such as hacking the hosting server or creating a denial of service for the hosting server can violate laws, portray the company in a negative light, and may impact other victims (such as any other sites hosted on the server where the phish site is up.) Can anything be done? One possibility is to send (or feed) realistic but fake data to the phish site. Phish feeding tries to achieve one or more of the following outcomes:

- Reduce the value of the data if the phishers plan to sell it by diluting real responses with fake ones that look real.
- Provide additional time for customers to realize that they've been phished and contact the institution so accounts may be blocked before being exploited. (Credit cards have been exploited at ATM machines less than 15 minutes after the customer entered the information in the phish website.)
- Provide fake values that the targeted institution may be able to monitor to track malicious access to financial websites and potentially obtain their source IP addresses.
- Frustrate the phishers so that they may move on to easier targets.
- Create a reputation of being difficult to phish so that phishing groups will not try to phish you again and new groups may avoid you.

Phish feeding is something that many may have thought about and perhaps wanted to do. Covertly, some institutions and security companies may be exploring this or even actually doing it. The first part of this paper will discuss how and when this can be done. The second part of this paper will present a framework and sample tools so that others may test this in their own labs and perhaps develop their own tools and techniques to phish feed.

Phishing Challenges

Phishers are looking to obtain financial information that they can either quickly convert into cash or sell to someone. A primary target is any financial institution that doesn't check the Card Validation Value code embedded in the magnetic stripe when a credit or debit card is used at an ATM machine. Credit cards usually carry two Card Validation Value codes. (These are Visa's terms, but MasterCard and the other major cards work the same way only they each call it something slightly different.) One CVV, the CVV2, is printed on the back of the card. Its purpose is to provide an additional piece of information that only the card holder should know and it is primarily used for online transactions. Unfortunately, people responding to phishing schemes often readily disclose their CVV2. The other CVV is embedded in the magnetic stripe. It is known only by the card issuers and processors, not the end user. When an ATM system is not configured to check the CVV value, criminals need only get the basic card information and PIN, perhaps by asking for it in a phish, to be able to create a fake card and use it at an ATM for a cash withdrawal. If a financial institution is known by the phishers to not check the CVV during ATM transactions, it can expect to be a desirable target for phishing. However, while an ATM processing system that verifies the CVV can greatly reduce fraud as well as reduce the ease with which an institution may be exploited, it will not completely remove the threat of phishing.

To use a favorite security analogy, criminals generally go after the low hanging fruit, or easiest victims. At some point, the security bar gets raised, through regulations or improved security measures, and the "fruit" gets raised with it. This forces the phishers to also take a step forward, which they undoubtedly will do. One thing phishers may do is to continue to seek the same information, even though they know that they can't make usable fake plastic cards without the CVV code. The reason they will do this is because in some cases they will already have the CVV code and merely be looking for the PIN. Fans of the "Sopranos" HBO series may recall Artie Bucco's American Express problems when his hostess swiped credit cards. Using a small skimming device, a credit card can easily be swiped and the information stored along with many other cards for uploading later. Depending on how the information is to be exploited, the criminals can sit on the information and wait for the complete financial picture to get painted as information comes in from additional sources. If the desire is to withdraw cash using the card at an ATM machine, then the PIN will be needed. You may be wondering why the PIN would be needed if the entire magnetic stripe has been captured. The reason is because the PIN isn't stored in the magnetic stripe. The PIN and card validation values are generated by an algorithm that uses a secret key for each institution. What *does* appear in the magnetic stripe is an offset. The natural PIN is derived from an algorithm and may not be to the customer's liking. The customer's institution may provide the option of changing the PIN. The offset allows the customer to use his or her own personalized PIN, which is then transformed by the offset back to the natural PIN. So, swiping a credit or debit card will not give criminals the ability to use the card for an ATM withdrawal. The ability to do ATM withdrawals is desirable because the large number of ATM locations and instant cash make them difficult to track.

Phishers and internet criminals are often very organized and connected. They may go after one piece of information, such as the PIN, to correlate it later with existing information, such as the card number and CVV. The first six digits of a credit card number, the BIN (or Bank Information Number,) are uniquely assigned to financial institutions. Given a credit card number, criminals can tell which financial institution issued it. Conversely, given a financial institution, criminals know the BIN numbers to target. Phish kits have been recovered that contained BIN databases containing thousands of financial institutions, so phishers definitely have this capability. You can also look this up for yourself using a website tool such as All-Nettools (http://all-nettools.com/toolbox,financial)

Even if a quick cash out isn't likely due to CVV verification or other security measures, a target institution can still have value to phishers. Many institutions offer additional online services. One example is bill pay. Getting access to a user's online account and his or her bill pay could allow the phishers to create a new payee and send money to that payee. Even without bill pay, phishers can transfer money into other accounts, internal or external, depending upon the limitations of the target institution's online services. Criminals have created impressive looking websites and company front ends and posted online job listings on popular job search sites. These jobs often list requirements that include having an account with particular institutions as well as the willingness to accept and transfer out large quantities of money. Naïve people often fall for these job listings and find themselves in trouble with the law after wiring this money abroad. So, even if an institution is checking CVV and isn't vulnerable to easy card exploitation, there is valuable personal information phishers will continue to pursue.

Phishing continues to evolve. As current vectors of exploitation get shut down, the phishers will continue to find and exploit new ones. People are too easy to fool and the rewards are far too great to stop. This is where phish feeding might come in to play. The goal in phish feeding is to submit fake yet real looking entries into a phishing site. If the fake entries are realistic enough, the phishers will need to expend a great deal of effort to determine which entries were submitted by real customers and which were generated by a phish feeder. This could make you a less desirable target for future attacks. It could also buy customers more time to contact their financial institution before the phishers can exploit their information. Phish feeding could reduce phishers' ability to quickly exploit credit cards at ATMs and reduce the value of the information if they are collecting it to sell.

There are a number of challenges in feeding data that looks real to a phish site. The first challenge is that almost every phish is different. Some ask for names, others for addresses and social security numbers, and others a combination or even all of the above. Each phish must be dealt with on an individual case basis. Although you may start with a template or previously used phish feed code, you will need to modify it for each phish.

Another problem is in creating realistic datasets to match the data requested by the phish. Credit card numbers include a valid BIN code and must adhere to a checksum code. Some phish sites perform checks to validate the card number checksum and may even validate the BIN as well. If the card number is not realistic, it will be easy to spot and may not even be accepted by the phish site's checking routines, preventing feeding the phish altogether. Credit card number generators can be written to create card number lists. The chances are virtually nonexistent that you would generate a real card number as well as the correct expiration date, PIN, and CVV codes. There are web sites that provide sample code for generating card numbers. (Graham King's credit card generator can be found at http://www.darkcoding.net/projects/credit-card-generator/.) Another option is to use real cards that have been deactivated. If your institution has deactivated cards due to theft or compromise, you may be able to obtain a database of these card numbers which are real but cannot be exploited. Names are fairly simple to generate by randomly combining a list of first names with last names. Addresses and phone numbers can be more difficult. There are online services, including one offered by the USPS, which can check the validity of an address. Still, this would require significantly more work by the phishers to check.

The source IP address of a phish feeding system must also be considered. Many phish kits include the victim's IP address as part of the information sent to the phisher. If you were feeding a phish site from a single IP it wouldn't be too difficult for the phishers to identify a single, repeating IP address and start filtering it out. Even from behind a single broadband connection, you can make your source IP address "change" by sending it through anonymous proxy servers or anonymous networks such as The Onion Router (TOR.)

By considering the creation of your data, how authentic it appears, and altering your source IP address, you can make it much more difficult for phishers to extract useful information. It is not possible to prevent phishers from obtaining real data, but it is possible to make it costly in sifting it out. The cost and effort required by the phishers to verify the data might be enough to cause them to move on to more desirable targets. As you examine the phish (the

quality of the phish email that was sent, how well targeted the phish was to the intended audience, how advanced the phish site and its data checking capabilities are) you will start to get an impression of how advanced a phish scheme you are dealing with is, which will help determine how much effort is necessary to create the fake data.

In considering phish feeding you must consider the possibility of retaliation. If you feed a phish site will the phishers seek retribution against your institution? There is always that possibility. However, here are a few reasons why this is unlikely. First, phishers are all about making easy money. Retribution doesn't make them any more money while moving on to easier targets does. Phishing groups are often composed of individuals recruited for specific skills such as spamming, site compromising, money laundering, etc. These loosely organized groups, often formed or recruited over Internet Relay Chat, are held together by the desire to make money. Should one individual want to retaliate he will likely need to do so by himself and with his own resources as the others will want to move on to another target where money can be made. This creates a kind of economic peer pressure against retaliation. Another reason is that phishers are methodical opportunists. Their targeting is generally less random than you might think. Phishers employ organized schemes that target based on geographical locations, industry, and other factors. It's not uncommon for multiple institutions to be simultaneously targeted and even other institutions to be targeted a short time later. This keeps the phishers busy and means that taking time to retaliate against an unprofitable institution will cost them lost time and money from other institutions. Finally, the phishers may not know the source of the feeding (provided you don't feed a phish from source IP addresses that can be directly attributed to your institution.) It is highly possible that security vendors, volunteer groups, or even technically savvy phish recipients could utilize phish feeding techniques. While a response is most likely to come from the targeted institution, many institutions will lack the resources or desire to do so. The possibility of a community based approach would leave the phishers unable to determine who is performing the feeding. For these reasons, although possible, retribution is unlikely.

Law and Ethics

I am not a lawyer, nor do I play one on TV. You should consult with your own legal council and management before embarking on a phish feeding campaign. Let's discuss ethics. You or your security vendor should be contacting the website owner and ISP via phone, fax, or email with a notification asking them to cease and desist hosting a web site which is infringing upon your trademarks and name. If the web site owner or ISP isn't responsive, then really what you are doing is protecting your interests and the interests of your customers by accessing a website, openly available on the Internet, many times. You must, however, be careful not to invoke a denial of service condition when feeding a phish site. Phishers often like to compromise web servers that host multiple sites. Keep in mind that not only is the original website owner likely a victim, but the other websites hosted on the compromised system would become *your* victims if you caused a denial of service.

Section II: A Sample Framework for Phish Feeding

Overview

The purpose of this section is to describe a framework and methodology for accomplishing phish feeding. Specific tools are described that you can obtain to create a phish feeding system. All of the tools are either free or come with free evaluation periods.

WARNING!: You should perform this set up in a private test lab that has no Internet access and gain familiarity with the process and tools. Before taking this out of a lab environment consider the following:

- If testing with a real phish kit, did you examine all of its communication methods and ensure it can't send email or other data?
- Are you doing this from a company network? Do you have permission to do this?
- Have you discussed this with your legal team and do you have management's approval?
- Are your test systems free of any confidential information, fully patched, and behind a firewall or NAT router?

To give a brief overview of phish feeding, a template of the phish site's http form is created by submitting fake data to the phish site and recording the transaction. Static values in the template are then substituted with variable names. A program is written to output a batch file containing realistic data and that batch file then feeds the data to the phish site. Anonymous proxies are used to make the feeds appear to be coming from different IP addresses.

Resources Used

- **VMware Workstation** (www.vmware.com) on a sufficiently powerful desktop or laptop computer. You can use real PCs, but virtual machines can provide some advantages that will be discussed later. VMware Workstation costs about $300 and a trial version is available on VMware's website.

- **Windows XP** PC or Virtual Machine (VM) with Internet Explorer. This represents the most common configuration found on the Internet and so is a good sample environment to start creating a phish feeder.

- **Fedora Core** web server PC or Virtual Machine (VM) with Apache, PHP, and Perl. This is the system you will install a phish kit or sample forms pages to test feeding from the XP system.

- **MacroScheduler Pro** and **WebRecorder** from mjtnet (www.mjtnet.com) which is a great automation tool. (The support from mjtnet is also amazing!) Both products are available for about $235 and trial versions are available on the mjtnet website.

- **Perl** under Windows (running in the XP PCs or VMs.) ActiveState Perl can be downloaded for free from the Active Perl website. (www.activeperl.com/ASPN/Perl)

(You can use any programming or scripting language that you're comfortable using. However, this paper will only discuss Perl.)

- **ConTEXT** text editor. This is an excellent, freely available text editor for Windows that supports language syntax highlighting for many languages, including Perl. ConTEXT is a free download (www.context.cx.)

- **Broadband connection**. Bandwidth usage is not really a concern, but it's better to do these activities from an IP address *not* directly associated with your company. (Not needed when only testing in a lab environment.)

- To support changing the source IP addresses, you can use **ProxyWay**. ProxyWay (www.proxyway.com) is available in a free version as well as a Pro version for $60. If you purchase the Pro version you can request a version that allows a new proxy to be used each time a new connection is started. The reason for wanting this functionality will be revealed later. (Not needed when only testing in a lab environment.)

Methodology

1. Create a Windows XP PC or Virtual Machine to serve as the phish feeder. On it install MacroScheduler Pro and WebRecorder, Perl, and ProxyWay.

2. Create a test web server. This can be a Virtual Machine, communicating with the virtual XP system through VMware's host only networking, or a physical PC on the same network as the physical XP system. Install Fedora Core, configured to run as a web server with JavaScript, PHP, and PERL installed.

3. If possible, obtain a phish kit and install it on the Linux server. A phish kit is a ZIP or TAR archive file which phishers use to transport the entire source of the phish to the server that will host the phish. Phish kits can sometimes be obtained directly from a poorly configured server that's hosting a phish. If you don't have access to a phish kit, there are alternatives. If you're aware of an active phish, such as one that you've received in your own mailbox, you can copy down as much of its source as you can access. Tools such as HTTrack (Windows) or wget (Linux) can be useful for pulling content down. A little manual effort is generally also required if there are several successive forms involved. For instance, you may have to save the source HTML for the first form, enter fake data to get to a second form, and then save the second form's source. You won't get the back end processing code doing this, but you'll get enough to play with for creating a test phish system. An advantage of phish feeding is that to feed a phish you don't have to have any knowledge of the back end processing. It is enough just to be able to interact with the forms pages. If you are unable to find anything to use as a test phish site source, you can create or use sample forms pages on your test server.

4. Copy the URL of the location of the test phish or forms web page on your server and paste it into WebRecorder, as illustrated in figure 1. Begin manually entering the data within the fields. Use the mouse pointer to select the next field rather than using the tab key to allow

WebRecorder to track the field. When you are done, save the WebRecorder script. In the example shown here, the form has only one page and two fields; userid and password.

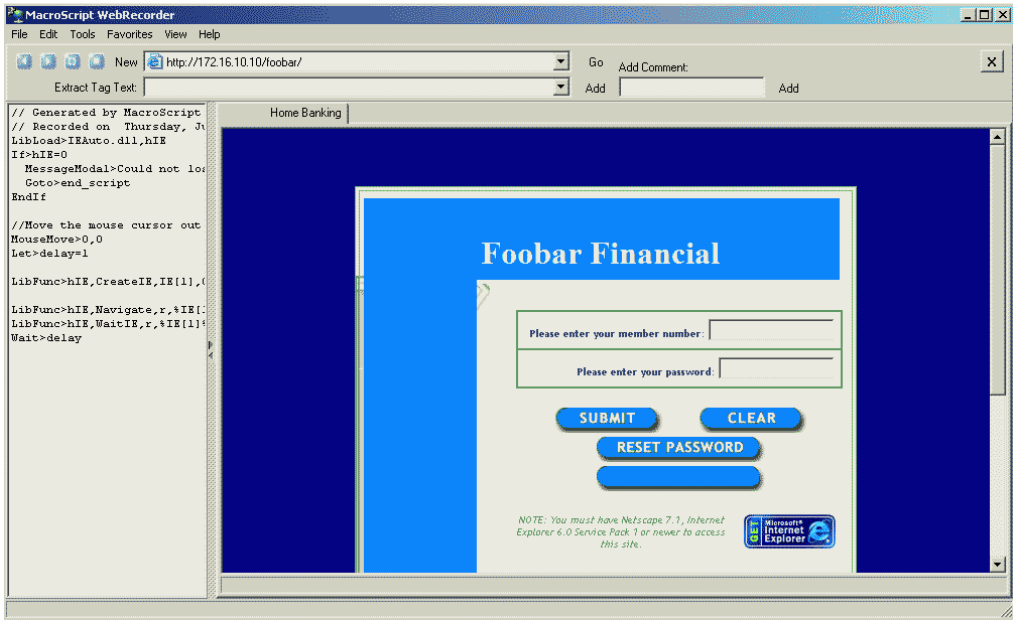

Figure 1: Using WebRecorder to create a template for entering information into a phish form

5. Edit the source of the template you created with WebRecorder. In the example in figure 2 look for the line **`Let>FieldName={"UserID"}`** that identifies the field name from the HTML form, which is immediately followed by the line **`Let>FieldValue={"6788"}`** which statically fills in the value 6788 in the field. Edit the FieldValue line so that it now says **`Let>FieldValue=USERID`** . You have now substituted the static value with a variable named "USERID". You will use this same variable name from a calling script, which will assign a value to USERID later. The same process is repeated to replace the static value for Password with PASS. For a real phish or for your test lab repeat this process for each field present. If there are multiple pages, click the Submit or equivalent button and continue until you've completed entry for the phish. Figure 3 shows the template source after the variable names have been substituted. Also note in figure 3 that two lines were added at the end of the script, a wait>delay (which waits for the value of the variable delay number of seconds) and Exit (which exits the script and browser.) Add these lines to your script.

```
// Generated by MacroScript WebRecorder 1.70
// Recorded on Saturday, June 10, 2006, at 11:19 PM
LibLoad>IEAuto.dll,hIE
If>hIE=0
  MessageModal>Could not load IEAuto.dll, make sure it is in the path or edit the
LibLoad line.
  Goto>end_script
EndIf

//Move the mouse cursor out of harm's way to avoid causing mouseover events to
interrupt
MouseMove>0,0
```

```
Let>delay=1

LibFunc>hIE,CreateIE,IE[0],0

LibFunc>hIE,Navigate,r,%IE[0]%,http://172.16.10.10/foobar/
LibFunc>hIE,WaitIE,r,%IE[0]%
Wait>delay
// Fills in userid field
Let>FrameName={""}
Let>FormName={"signon"}
Let>FieldName={"UserID"}
Let>FieldValue={"6788"}
LibFunc>hIE,FormFill,r,%IE[0]%,str:FrameName,str:FormName,str:FieldName,str:FieldVa
lue,0

// Fills in password field
Let>FrameName={""}
Let>FormName={"signon"}
Let>FieldName={"Password"}
Let>FieldValue={"1234"}
LibFunc>hIE,FormFill,r,%IE[0]%,str:FrameName,str:FormName,str:FieldName,str:FieldVa
lue,0

// Clicks on submit button
Let>FrameName={""}
Let>FormName={"signon"}
Let>TagValue={"imageField"}
LibFunc>hIE,ClickTag,r,%IE[0]%,str:FrameName,str:FormName,INPUT,NAME,str:TagValue

LibFree>hIE
Label>end_script
LibFunc>hIE,KillIE,r,%IE[2]%
```

Figure 2: Sample template created with WebRecorder. Static values entered in the site are in red italics.

```
// Generated by MacroScript WebRecorder 1.70
// Recorded on Saturday, June 10, 2006, at 11:19 PM
LibLoad>IEAuto.dll,hIE
If>hIE=0
  MessageModal>Could not load IEAuto.dll, make sure it is in the path or edit the
LibLoad line.
  Goto>end_script
EndIf

//Move the mouse cursor out of harm's way to avoid causing mouseover events to
interrupt
MouseMove>0,0
Let>delay=1

LibFunc>hIE,CreateIE,IE[0],0

LibFunc>hIE,Navigate,r,%IE[0]%,http://172.16.10.10/foobar/
LibFunc>hIE,WaitIE,r,%IE[0]%
Wait>delay

// Fills in userid field
Let>FrameName={""}
Let>FormName={"signon"}
Let>FieldName={"UserID"}
Let>FieldValue=USERID
LibFunc>hIE,FormFill,r,%IE[0]%,str:FrameName,str:FormName,str:FieldName,str:FieldVa
lue,0

// Fills in password field
```

```
Let>FormName={"signon"}
Let>FieldName={"Password"}
Let>FieldValue=PASS
LibFunc>hIE,FormFill,r,%IE[0]%,str:FrameName,str:FormName,str:FieldName,str:FieldVa
lue,0

// Clicks on submit button
Let>FrameName={""}
Let>FormName={"signon"}
Let>TagValue={"imageField"}
LibFunc>hIE,ClickTag,r,%IE[0]%,str:FrameName,str:FormName,INPUT,NAME,str:TagValue

Label>end_script
LibFunc>hIE,KillIE,r,IE[0]
LibFree>hIE

Wait>SLEEP
Exit
```

Figure 3: Editing the template from WebRecorder. (Grayed lines represent changes to original. Script.)

6. Next create the Perl script which will create a batch file to invoke MacroScheduler, passing it values for the variables you just created. In the example used here, you will randomly generate values for userid and password. The example script in Figure 4 can serve as a framework which can be used for other phish feeder scripts.

7. Run the Perl script by typing "perl phish.pl" at a command prompt. A screen shot is shown in Figure 5. The script contains a prompt to allow you select the number of entries to generate. An example of the script's output, which is a batch file, is shown in Figure 6.

```
# ====================================================================
# NAME: phish.pl
#
# AUTHOR: John Brozycki
#
# PURPOSE: Creates a batch file to invoke MacroScheduler.  Each
#          entry in the batch file is an individual invocation with
#          its own variables.
# ====================================================================
#
$count = 0;
$macroline = 0;
$line = "";
$newname = "";
$command = "\"C:\\Program Files\\MJT Net Ltd\\Macro Scheduler\\msched.exe\"
  c:\\perl\\phish.scp ";   #This line is part of the previous line
$DefaultCount = 25;        #Change this to change default quantity of responses
$namefile = "PHISH.BAT";   #Change this to change default outfile name
$outstring = "";
$InvalidCount = "Y";
open (OUTFILE, ">".$namefile) or die "Cannot open output file: $!";
# ====================================================================
# Print instructions
# ====================================================================
print ("--------------------------------------------------------------\n");
print (" This utility will generate a batch file with multiple responses.\n");
print ("Output is to a local file named:".$namefile."\n");
print ("--------------------------------------------------------------\n\n");
# ====================================================================
# Get the User's input for the number of entries to generate
# ====================================================================
while ($InvalidCount eq "Y")
   {
     print ("\nPlease enter how many entries to generate, 1-10000
[".$DefaultCount."=default]: ");  #This line is part of the previous line
```

```
     $_ = <STDIN>;
     $count = $_;
     chomp($count);
     if ($count > 0 and $count < 10001)
       {
        $InvalidCount = "N";
       }
     if ($count eq "" and $InvalidCount eq "Y")
       {
        print ("\nUsing the default count of ".$DefaultCount."\n");
        $count = $DefaultCount;
        $InvalidCount = "N";
       }
     if ($InvalidCount eq "Y")
       {
        print ("\nERROR: Count needs to be a numeric value between 1 and 10,000.\n");
        print ("You entered: ".$count."  Please try again.\n\n");
       }
     }
# ========================================================================
# Loop through the creation process until the desired number of entries
# has been generated.
# ========================================================================
while ($count > 0)
   {
    $userid = int(rand(9000))+ 1000; #Userids in range of 1000 to 10000
    $pass = int(rand(8000))+ 1235;  #Passwords in range of 1235 to 9235
    $sleep = int(rand(15) + 5;      #Random wait variable, 5 to 20 secs
# Build the command line with variables
    $outstring = $command."/USERID=".$userid;
    $outstring = $outstring." /PASS=".$pass;
    $outstring = $outstring.” /SLEEP=”.$sleep;
    print OUTFILE ($outstring."\n");
     $count = $count - 1;
    }
close OUTFILE;
```

Figure 4: PERL Script to generate batch file

```
C:\Perl>perl samplephish.pl
------------------------------------------------------------------------
 This utility will generate a batch file with multiple responses.
Output is to a local file named:PHISH.BAT
------------------------------------------------------------------------


Please enter how many entries to generate, 1-10000  [25=default]: 5

C:\Perl>_
```

Figure 5: Running the PERL Script to create the batch file

```
C:\Perl>type phish.bat
"C:\Program Files\MJT Net Ltd\Macro Scheduler\msched.exe" c:\perl\phish.scp /USE
RID=5230 /PASS=3015 /SLEEP=8
"C:\Program Files\MJT Net Ltd\Macro Scheduler\msched.exe" c:\perl\phish.scp /USE
RID=4599 /PASS=6005 /SLEEP=13
"C:\Program Files\MJT Net Ltd\Macro Scheduler\msched.exe" c:\perl\phish.scp /USE
RID=1418 /PASS=3027 /SLEEP=19
"C:\Program Files\MJT Net Ltd\Macro Scheduler\msched.exe" c:\perl\phish.scp /USE
RID=3200 /PASS=3646 /SLEEP=16

C:\Perl>
```

Figure 6: The contents of the batch file

8. Invoke the batch file by typing “phish.bat” at a command prompt. The batch file will sequentially run each entry separately, opening Internet Explorer, filling in the data, submitting the data, closing Internet Explorer, and then starting again with the next entry. This is the framework for phish feeding. You have now demonstrated phish feeding

capability in your lab setup. However, there are a number of additional considerations that must be addressed for a successful phish feeding that will comprise the remaining steps in this methodology.

9. It is possible that Macro Scheduler may crash while interacting with Internet Explorer. The feeding process may run for hours or days and you probably won't be watching it every minute of the process. To counter against this, a script named "watchdog.scp" is started prior to running the phish feeding script. Its purpose is to periodically check for a crashed program requestor and automatically click on "OK" to acknowledge the crash. At this point the batch file continues and the next entry is run. The worst case scenario is that you lose submitting the entry where the crash occurred, but it still picks up and continues with the next entry. The script for "watchdog" is contained in Figure 7. Even if you use a different program or utility to feed data to the browser, you should consider something similar to either handle program exceptions or monitor them and generate an alert.

```
// Close any error  dialog boxes if MSCHED.EXE crashes so our script
// continues running. Checks every 2 seconds
Label>top
IfWindowOpen>msched.exe
  WindowAction>3,msched.exe
Endif
Wait>2
Goto>top
```

Figure 7: Watchdog script to acknowledge the requestor if Macro Scheduler crashes.

10. A random time delay is useful in keeping the feed entries from looking too artificial. Within the Windows command line environment, there aren't any native sleep or wait functions. Utilize Macro Scheduler's wait function as illustrated in the sample scripts. In figure 3, the phish template edited from WebRecorder, a variable named SLEEP has been substituted for the WAIT> function. In figure 4, the Perl script, a variable named $SLEEP has been created, assigned a random value from 5 to 20 seconds, and output into the batch script. Set these values to make submissions look realistic. For example, if many phish emails were sent out and there's a large number of potential victims use smaller sleep values. If a smaller amount of phish emails was sent out and there's a small number of potential victims then use larger values. How can you tell? You can get an idea of how many phish emails were sent by looking at how many people contact the targeted institution as well as by examining the "blow back" auto replies and out of office messages.

11. Many phish kits capture and send the victim's IP address. If you cannot obtain the kit or complete source for a phish, you must assume that the victim IP address is included in the email to the phisher. If you run your phish feeder from a single IP address, the phisher will be able to identify a repeating IP address and ignore entries that contain that address. This is perhaps one of the most difficult challenges in phish feeding. One way to handle this is to use an anonymous proxy tool which can relay traffic through one anonymous proxy and then change to another anonymous proxy after a period of time or number of uses. One tool to do this is ProxyWay. The free version supports up to five anonymous proxies. The Pro version does not have a limit. Note that you *will* need to have Internet connectivity to test out anonymous proxy functionality. Also note that anonymous proxies significantly slow down the feeding process.

12. Install ProxyWay on the Windows XP system that will be the phish feeder. Click on the Proxy button, select Proxy List, then Update proxy list. You can seed the list with any websites that list anonymous proxy servers that you want. An example is www.proxy4free.com/page1.html. This site hosts five pages so you can add additional entries, replacing "page1.html" with "page2.html" and so on until you get to five. This is shown in figure 8.

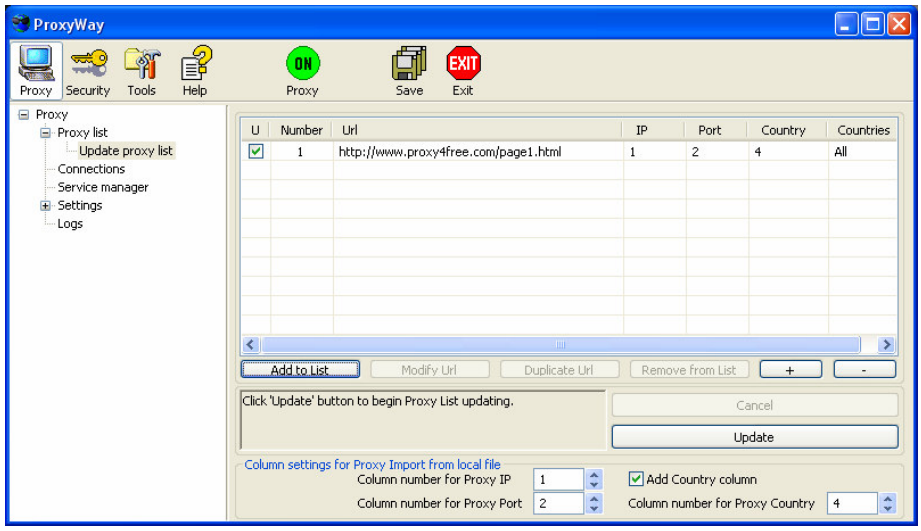

Figure 8: Configuring proxy list sources in ProxyWay

13. Click on Proxy, then on service manager. Click the Add button to add a service. A service represents the anonymous proxy server or proxy cascade (chain) that you can use. Cascades are not necessary. All that you want to do is ensure that the phish server reads a different IP address for each entry. In the "All proxies" part of the Add/Modify service screen select a single proxy server and then click Add.

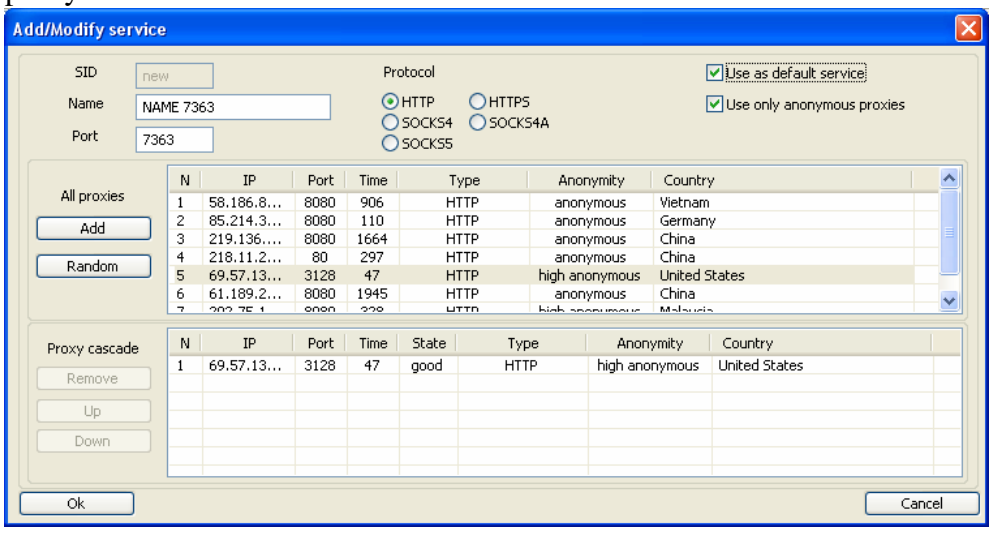

Figure 9: Adding an anonymous proxy server to a ProxyWay service

14. In the free version you can select up to five services. Set up all five. Pick one to be the default and click the checkbox for "use as default service." From the service manager screen select each service and click the Start button. You now have retrieved a list of anonymous proxy servers, selected five, configured those five as individual services, selected one to be the default, and started the service manager.

15. Click Proxy and then select Connections. Select the Internet Explorer tab and click on the LAN connection. ProxyWay can automatically configure proxy settings for IE and other browsers. Go ahead and do this. If you want to do it manually for Internet Explorer go directly into IE, edit the LAN SETTINGS under Connections for IE, enable the checkbox for Use a proxy, and specify the loop back address of 127.0.0.1 and the port value of 81. IE will now send all web requests to ProxyWay running on the same machine.

16. Select Proxy, click on Settings, and select Default service settings. Click the checkbox to enable using another service if current service failed as shown in figure 10. Click on the checkbox to Change Service and set these values to their lowest settings. The lowest settings are 5 minutes for time or 10 connections. These aren't very desirable for phish feeding. Ideally, you want a new anonymous proxy used for each feed. There are two options. If you purchase the Pro version you can request from ProxyWay a version that will change services after each connection. Alternatively, you could configure your script to access the phish site once and then access Google, Yahoo, etc. 9 times before accessing the phish site again. Since the free version only allows 5 services (anonymous proxies) your feeds will only appear to come from 5 different IPs. The free edition is good for testing purposes and as a proof of concept, but for real phish feeding you will want to get the custom Pro version or look for alternative solutions.

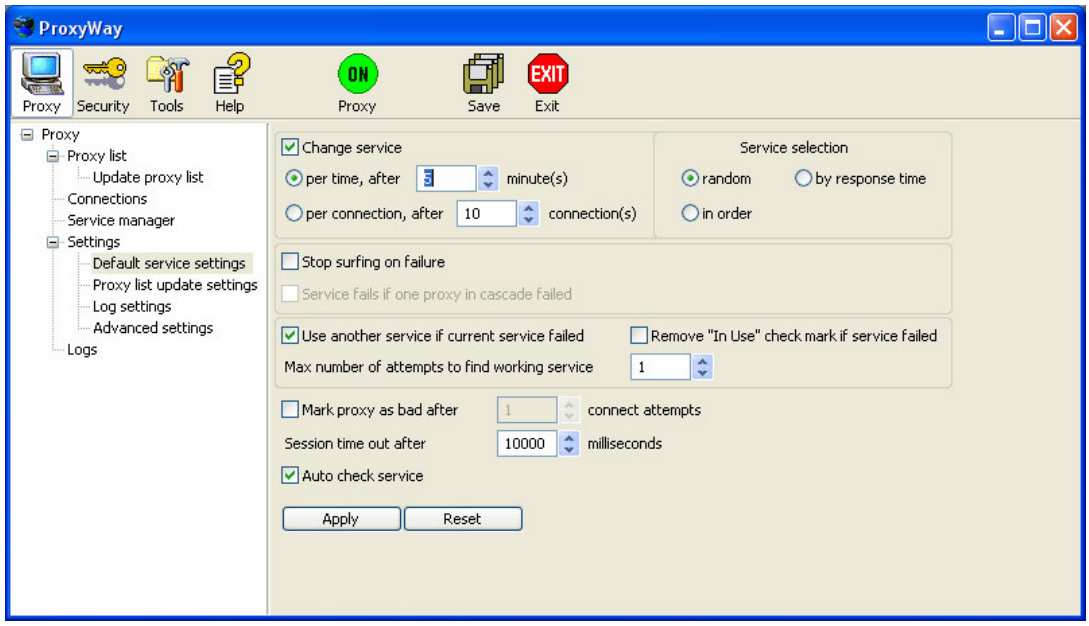

Figure 10: Default service settings for ProxyWay

17. Confirm that ProxyWay is active (the green ON button) and test it by opening IE and accessing a web page. If you go to an IP enumeration page, such as whatismyip.com, you can verify that you are indeed coming from a different IP address. If you configured everything properly, you are now capable of phish feeding from multiple anonymous IP addresses.

18. In the sample scripts used in this paper, only two simple fields were used. What if more complex data is requested such as credit card number, name, address, zip code, or email address? This is the area where you can really become creative. The more realistic the fake data is, the more difficult it will be for the phishers to separate real replies from your fake replies. Below are some tips and ideas for some values that may be requested in a phish.

a. Credit Card number. Never use real, active card values. Use closed out values if available to you. If not, you can create values that will pass the checksum tests and even the BIN tests as mentioned in the previous section. You can find websites and code to generate realistic values.

b. Address/zip code. You can create random entries by stringing together words, such as "Elm" and "Street" or "Elm" and "Court." There are online tools that can be used to validate an address, so expect that phishers will have the ability to check this. You can download lists from the Internet that have addresses. You could even seed your own list with addresses from police departments, town halls, chain stores, etc.

c. Name. Here is a technique that can also be used for other values, such as card numbers, addresses, etc. Create a text file full of first names and a text file full of last names. Load each file into a dynamic array. Now when you want to generate 1,000 entries and you have only 100 entries in your first and last name files, you just pick a random value from each array and combine them together. You may get some repetition, which you can check if you think that's a problem, but duplication amongst thousands of entries is difficult to detect. Figure 11 contains a Perl script that uses two files, firstnames.txt and lastnames.txt, to create a large list of random names by randomly combining the two. Both files are simple text files with a name and carriage return on each line. The more entries you put in each file the better your output will be. This technique can also be used with credit card numbers by loading a set quantity of card numbers into an array and randomly using them when creating entries. Using this method, you can easily create thousands of entries even if your source data only has hundreds of values.

```
# ====================================================================
# NAME: makenames.pl
#
# AUTHOR: John Brozycki
# PURPOSE: Generates a listing of random names.
#
# ====================================================================
#
$count = 0;
$firstnames = 0;
$lastnames = 0;
$line = "";
$newname = "";
$DefaultCount = 10;   #Change this to change default quantity of names
$namefile = "names.txt";  #Change this to change default outfile name
```

```
$InvalidCount = "Y";
open (OUTFILE, ">".$namefile) or die "Cannot open output file: $!";
open (FIRSTNAMES, "firstnames.txt") or die "Cannot open firstname file: $!";
open (LASTNAMES, "lastnames.txt") or die "Cannot open lastname file: $!";
# ================================================================================
# Print instructions
# ================================================================================
print ("-------------------------------------------------------------------------\n");
print (" This utility will generate however many random names based upon two\n");
print ("input files, one holding first names and one holding last names.\n");
print ("Output is to the screen and also to a local file named:".$namefile."\n");
print ("-------------------------------------------------------------------------\n\n");
# ================================================================================
# Get the User's input for the number of names to generate
# ================================================================================
while ($InvalidCount eq "Y")
   {
     print ("\nPlease enter how many names to generate, 1-1000
[".$DefaultCount."=default]: ");  #this is part of the line above
     $_ = <STDIN>;
     $count = $_;
     chomp($count);
     if ($count > 0 and $count < 1001)
       {
        $InvalidCount = "N";
       }
     if ($count eq "" and $InvalidCount eq "Y")
       {
          print ("\nUsing the default count of ".$DefaultCount."\n");
        $count = $DefaultCount;
        $InvalidCount = "N";
       }
     if ($InvalidCount eq "Y")
       {
        print ("\nERROR: Count needs to be a numeric value between 1 and 1000.\n");
        print ("You entered: ".$count."  Please try again.\n\n");
       }
     }
#
# ================================================================================
# Load first name and last name arrays
# ================================================================================
print ("Making ".$count." names.\n");
while (<FIRSTNAMES>)
  {
    $line = $_;
    chomp($line);
    $firstarray[$firstnames] = $line;
    $firstnames = $firstnames + 1;
  }
close FIRSTNAMES;
while (<LASTNAMES>)
  {
    $line = $_;
    chomp($line);
    $lastarray[$lastnames] = $line;
    $lastnames = $lastnames + 1;
  }
close LASTNAMES;
print ("Using ".$firstnames." firstnames and ".$lastnames." lastnames.\n");
#
# ================================================================================
# Loop through the name creation process until the desired number of names
# has been generated.
```

```
# ================================================================================
while ($count > 0)
   {
    $fcount = int(rand($firstnames));
    $lcount = int(rand($lastnames));
       $newname = $firstarray[$fcount]." ".$lastarray[$lcount];
     print ($newname."\n");
     print OUTFILE ($newname."\n");
     $count = $count - 1;
    }
close OUTFILE;
```

Figure 11: Perl script to generate random names

19. Here are some final considerations for the phish feeding process.

    a. Divide and Conquer.
    You can copy a Virtual Machine and have two or more concurrent feeders run. Be careful not to create a Denial of Service condition on the server hosting the phish! Note that Macro Scheduler Pro has the ability to compile scripts into executables which can then be run on other machines without requiring additional licenses. The phish Perl script could be altered to use the created executable instead of the phish.scp file and be distributed to other machines. Additional VMs or physical machines could also be utilized on different broadband connections.

    b. Tracking Success.
    Before starting your feeding process, make a note of some of the first few fake accounts. If possible, set up an alert or IDS for your Internet banking server to key on these fake account numbers. Activity will tell you that the phishers are testing the data you fed and may reveal their IP addresses if they're not behind a proxy or compromised system.

    c. Wild thought.
    Imagine a phish feeding engine that worked like SETI@HOME. PCs all over the world could pull down a signed, encrypted package that would allow the average PC to feed currently active phish sites. The central server would control the number of entries per phish site it doled out to prevent a DoS condition.

Conclusions

This paper has introduced the concept of phish feeding current Internet-based phishing schemes and discussed many considerations that go along with it. It has presented a framework for phish feeding with a set of example tools that can and have been used to implement a phish feeding system. You can set this up in a test lab and try it out. These tools can be directly applied to a real phish site or serve as a proof of concept for higher end tools or systems. Although these tools provide automation for the actual feeding process, it's important to recognize that, at least at the present time, creating a feed for a phish site requires a significant investment of time by a skilled individual. The response needs to be tailored to the dynamics of each individual phish. Whether or not and to what degree efforts to phish feed pay off will depend upon the skills of the feeder, the skills of the phishers, and other factors. Nevertheless, phish feeding might prove a worthwhile tool to actively respond to phishing.