CHAPTER 1

OS X, Privacy, and Online Safety

Technical Classes: Basic, Standard, Advanced, and Advanced+

SECTION ONE: LOGICAL PRIVACY AND SECURITY

I feel confident in saying privacy and safety will be the most important concerns you will have (or should have) in your online life. And if they aren't now, they will be as time passes on.

For purposes of this chapter, I'm defining "privacy" as the level of control one has over their own personal information as well as the level of control one has as it regards personal information *other people own*. "Online safety" is primarily the ability to prevent unauthorized code from being executed on a system, including the specific controls one has in place to prevent code execution. That extends to preventing information disclosure, unauthorized access to files, application permissions, and so forth.

In actuality, privacy and security are fibers of the same cloth. They can be distinct concepts on their own, or they can be intimately entwined with each other. As such, I'm not going to try to classify every risk we discuss as one or the other; you are smart enough to switch to OS X, so you are smart enough to figure that part out.

I'll be discussing techniques and procedures specific to OS X, those distantly related to OS X, and in a case or two, processes that stand on their own irrespective of the OS one may be using. It's all part of the Big Privacy Picture, and though it may deviate a bit, I consider it required reading material. I'm calling this "logical security" as it does not apply to any particular technical security control, but rather behavioral changes you may wish to make in order to protect your data. So let's get started.

Internet advertising is the bane of the internet, and the core driver of the deep, vast violation of your personal privacy. These days, ad "impressions" don't mean anything. An ad "impression" is where there is some ad on a page somewhere, where the host assumes you looked at it; then bills the advertiser for aggregated impressions. Today, the "conversion" is the golden egg. You are the goose who wasn't aware you laid it. The conversion is where you

CONTENTS

Thor's OS Xodus. DOI: http://dx.doi.org/10.1016/B978-0-12-410463-1.00001-X

are on a site, see an ad, click it, and end up buying whatever the advertiser is selling. Those are big money. It's such big money that the advertising hosts (those who produce the ads for the host site) have technology where they collect and analyze your personal information and browsing history to not only provide an ad, but to provide an ad specially selected for you, based on your browsing patterns and purchase history. The way they can track your movements to sites where you purchase products are via cookies and other bits of shared information.

So, how comprehensive is this data, you ask? It is **so** comprehensive that government agencies and law enforcement routinely ask folks like Google for your individual profile history and any other personal information you may have given them by virtue of the EULA (End User License Agreement) you agree to by using their service.

Think about that for a moment. Here we have the NSA building, a 1.5 million square foot data capture facility, to harvest phone calls, emails, searches, and anything else you may do where a signal is emitted. We have 37,000 FBI agents running about and who knows how many CIA agents.

Even with all of this brainpower, manpower, and the 65 megawatts of power at the new NSA facility, government agencies get their "personal profile" information from a public advertising engine service. That should tell you how much of your life Google stores, and sells.

You now might be asking, "How many requests are made by government entities for Google users?" Well, I'll tell you. Insofar as the data requests for a particular user, there were 21,389 in the six-month period ending on 12/31/12. That's all the data requested by that user for an undetermined amount of time.

Even worse, agencies requested specific, personal information from the actual Google account held by the user 33,634 times in the same 6-month time frame.

It doesn't take a genius to ascertain that the volumes of data Google has on you and me is far more than we may have considered, to the point Law Enforcement uses it to take some manner of legal action. That's scary stuff. I could go into the legal ramifications of a judge actually thinking that data has any evidentiary value, but we'll have to wait until later for that.

Before we tackle the problem of protecting that information, let's see exactly what data Google collects and what data they give away (or sell). According to Google's own privacy statement, they collect:

a. User account information like name, address, credit card numbers (where applicable), pictures, and might even create a Public Profile you don't even know about.

b. What Google services you use, what web sites you view, and everything you do when looking at or clicking ads, including what specific ad it is. Cookies regarding your habits are also shared with any number of third parties. And obviously the gmail traffic you create including sending to and received from data.
c. Phone logs like your phone number, phone numbers you call, forwarded calls, duration, where and when the call was made, SMS "routing information" (whatever that means), and finally, once they figure out it is you by cross-referencing data, they will link your phone number to your Google profile.
d. Full set of information about the computer you are using, such as your hardware make and model, your OS, browser information, unique IDs of hardware, etc. This data alone can easily and uniquely identify you as a specific user. This data is then linked to your profile.
e. Many applications use Google APIs. Map location is one, music streaming another. Google logs things like your GPS location, other information from a mobile device, what WiFi areas you are in (again, including GPS location).
f. They know what applications you install or uninstall, what applications you have, how and when you use them under the auspices of "auto-update" checks in the order of four or six times per day.

You know, little stuff like that. Google does, however, say they have strict policies in place regarding the disbursement of your data. These include the provision to share all of your data with:

a. Law enforcement, government entities like the IRS or Homeland Security, or whatever agency asks and they see fit to comply with.
b. To "affiliates," businesses or people they "trust" or who say they will access the data in "good-faith," Google employees, partner companies, and that guy from Burger King who sings "ding fries are done." And my favorite (directly quoted) where they produce data, apparently to anyone, to "detect, prevent, or otherwise address fraud, security or technical issues." So if your video won't go to 1900×1200, that's a technical issue, so someone can ask for your data.
c. Other sarcasm aside, this I take quite seriously. Buried in their "we use SSL to protect you" bits, they say they also "restrict access" to "employees, contractors, and agents."

What that means is the data you thought was encrypted from end-point to end-point really isn't and they decrypt (or simple redirect an SSL end-point to standard HTTP traffic) your data and store it. Yes, that would be the data you thought was secure.

It's a "death by a thousand cuts" thing – a little bit of data here and there isn't that big of a deal. But when there are so many different sources of data for you, the accumulation of it all creates a real issue. And obviously a huge monetary stream.

I don't want to make it look like I'm singling Google out (even though I am) because there are other, albeit smaller, offenders as well. If you were not aware of it, Microsoft has been trying for a long time to make headway into the advertising industry. In my opinion it's a failed endeavor, as they have already had to write off over 6 billion dollars for the purchase of a single company to support the Ads Platform. Regardless, since they couple with Bing and other Microsoft "owned and operated" sites, their data-mining is also a source of significant concern, given you may stay logged into your Windows Live ID (WLID), or "Microsoft Account" (or whatever they may call it now), in perpetuity for mail, with third-party sites using WLID to authenticate you.

I'll give you an example of the reach this type of tracking can give. Let's say while at work you logon to a Microsoft service such as Windows Live Mail and leave that page up while doing other things. Then you go to Bing to search for something – that data is stored based on your WLID. You then search for "stereo systems" or some such and select a link to Best Buy. They store that too, as does Best Buy. Oh, all other data is stored as well, such as what work research you are doing, and the contents of any email you may send out or receive. At quitting time, you close out of everything and go home from work. After dinner, you go down to your XBox to play Forza Motorsport or something of the like. You have to log onto XBox Live to play the game, and when you do, your profile data is made available to whatever processes XBox decides they can send out. There used to be a company called Massive, which delivered targeted ads to video games. Microsoft purchased that company, so now you've got your data all tied up in a nice little bow. As you drive around the track, you see various billboards and such. As you do so, the video game makes a request to the ad tracking system for an ad to put on the billboard in the game. Your WLID is transferred to the ad delivery mechanism along with identifying information about your profile. Based on that connection, a behavioral targeting call is made and before you can even start into a turn you see a billboard ad for Kenwood Stereo Systems based on your search earlier at work. Massive actually went to the trouble of determining how much Kenwood should pay on the ad delivery, based on how long it was visible, what angle you were at when you saw it, and how much of the full ad you could have viewed.

Scary, huh? This happens billions and billions of times a day, all day, everyday, to countless numbers of other websites and data harvesters.

There are other, and in some ways greater, evils playing this game. If you were wondering, this is where I mention Facebook. Facebook is a massive "in service" ad engine, but also has a web of affiliates giving and taking your personal data. The reason Facebook has that "keep me logged in" checkbox is so they can stick to what they say their privacy policy is while also keeping that cookie alive so that all the affiliate sites you go to can get ad data from Facebook while passing back as much information as they have on you. In fact, even if you are not logged in, sites will actively create objects redirected to Facebook to contribute to the Global Fleecing.

The Emperor Has No Clothes. And Neither Do You!

Now that we're all feeling exposed by these corporate wolves, the real question is "what do we do about it?" Well, remember the previous bit about me not going into the legal ramifications? I lied. One thing we *can* do about it is to pay attention to these legal cases where Facebook or Google data is used as part of the investigation or prosecution. The data shouldn't be allowed. There is absolutely no way whatsoever the integrity of such data can be ensured. Think about the sweeping access Google can give to your information. Think about how many global outsourced contractors they have (10,000+) such as GenPact Ltd. in Bermuda and other outsourcers in other countries. Who has access to your data then? Do you trust the 30,000+ employees world-wide? You and I have no idea, and never will, how many of these people could change, add, or delete the information Google stores on us. For instance, what if one of them dumped some child pornography into your email account and then turned you in to the feds? The courts would consider this to be "solid" evidence against you because Google said it was your information. This should be brought to everyone's attention. If we allow this data to be acceptable in court, we are doomed. DOOMED, I say! OK, I'm done with that bit.

Our goal in the rest of this chapter is to limit the overall amount of data we make available on the internet and then, to the best of our ability, limit how much of that data is available for harvesters. The first step, limiting what we give out, can be applied anywhere and on any OS, but is something I consider very important.

With sites like Facebook, since more of this information is shared than we know, and even more capable of being generated, it is really important to think through what your intent of being on Facebook is. If you wish to keep in touch with friends, then make sure you make your profile private. Friends (and Facebook) will have full access, but keep it out of the public domain.

Never put your real information on Facebook if you can help it, including your name if you can. My Facebook name was a little vulgar, but since it

sounded oriental (my last name was "Tang") it wasn't flagged. I said I lived in a different country, went to a different school, and was fluent in Scottish.

Your friends will know who you are, or you can tell them. It's far easier than you would think. Regarding friends, only "friend" people you actually know. If you wish to treat the number of friends you have on Facebook as a metric by which to measure your popularity or self-worth, you will do so at the cost of exposing your personal information to potentially anyone in the world. Your "friends," once you post something, can copy that data and do whatever they want with it and there is absolutely nothing you can do about it. As such, your data could be (indeed, *will* be) forever preserved on the internet for all time. So when your son or daughter (or you, for that matter) posts some picture with a blow-up doll in one hand and a bottle of whisky in the other, that image could turn up 10 years later when a prospective employer does a bit of research on you before giving an interview. Your ex-spouse could be spying on you to find out if an alimony increase is due, particularly if you post pictures of you in Jamaica with your new "friend" on a shopping spree. I once allowed myself to get into a chat-fight on Bill Maher's page with someone who was clearly wrong, and where I was obviously right.

I went to his page, and not only was it publicly available, but he had pictures of his kids with their names, and a list of cousins, aunts, and other relatives. Within a few minutes, I knew where he and his +1 lived, where they worked, what they looked like, and who their friends were. In just a few minutes, I had all manner of other information, which would have taken me significant effort to gather back in the day. Luckily for him I'm not some whack-job, but I must say the flowers I sent to him from his "Midnight Lover" probably twisted up his girlfriend a bit.

There is another process I want to highly recommend you adopt, and it regards the overall account data you use when purchasing items on the internet. I have done this myself and can't tell you how many times it has saved me considerable time while protecting my "identity" and money. While this has nothing to do with any specific operating system or application, I have to say that if everyone did this, identity theft and exposure to unauthorized transactions would drop dramatically.

There are two things I suggest you do: go get a P.O. box, and go open a debit card account at your bank that is an entirely separate account from any others you may have. Get a debit card for this account – NOT a "credit card." There is no reason to use a credit card to purchase something on the internet unless you don't have the money to pay for it and wish to make payments on items. I humbly submit that from an economic standpoint, people should not buy things they can't afford. If you can't buy a new monitor or your Macbook Pro

without paying cash for them, then don't buy them online. Drive down to Best Buy or phone in the order in cases where you must use a credit card, but don't buy online with one.

I have two accounts at Chase – one is "Production" and the other "Internet." The internet account has a single debit card associated with it, and the only thing I use that account for is internet purchases. I never, ever, use my production account or any other credit card for internet purchases. The internet account was created using my P.O. box account, and I only keep about $100 in it at any given time. Right now there's $25 or so in it. It's important for you to do as I did and ensure there is NO overdraft protection on the account. I've specifically configured the account so that if there is not enough money in the account the transaction will be denied just as if you were at the ATM. In this way, you can't be charged overdraft fees.

If I wish to purchase something on the internet and don't have enough in my internet account, I simply go to Chase online banking and transfer from one account to the other. The funds are immediately available and I can make my purchase without waiting for anything.

This setup buys me a tremendous amount of protection. For one, the worst that can happen if a vendor's database is compromised and my bank information disclosed is that I lose $25 or so. They can't make any credit purchases, and they can't purchase something for more than I have in the bank. Nor can I be charged overdraft fees.

The only personal information they can possibly get from me is my special P.O. box number and not my actual address. The best thing is that I don't care in the least if my account details are released. If they are, and I see fraudulent activity, I just report it, get my money back, cancel that particular debit card and get a new one. I'm never at any level of exposure beyond what I have in that special account.

In fact, literally while I was writing this chapter, I got an email from Adobe saying they were compromised and my password information and bank account information could have been disclosed. I have a recurring payment to Adobe for Creative Cloud, so they have my internet account debit card number on file. If I were using a credit card instead, I wouldn't be writing this right now. I'd be on the phone with the bank canceling the credit card and then going through and trying to figure out where I used that card, where it may be on file, and where reoccurring transactions may be at the risk of failing and my losing service (such as Netflix and Adobe Creative Cloud) and, more importantly, I'd be worried and anxious about what exposure I may have knowing it is really outside of my control.

I honestly didn't care if that account got compromised so I just kept on writing. It's actually not even worth me canceling the account since I'm not at any financial exposure and I know every transaction on that account. That's the other benefit – the accounting on that account is crazy simple. I know there won't be any non-internet transactions on it, and know I only need look at that account for transaction details. In other words, I don't have to scour through a hundred other transactions looking for one that may have been sourced from the internet.

Now, millions of people use PayPal, but I don't anymore. At first, it was great. I just used my internet account to associate with my PayPal account. But then PayPal wanted me to get some other debit card to use just for them which would allow me to go to the ATM and withdraw funds deposited via donations at my website. I thought I'd give it a shot, but they immediately sent me an email asking for my SSN, proof of current address by way of a utility bill, and a copy of my driver's license. I wrote back saying "in that case, no thanks." But they still wanted it to keep my regular PayPal account open. I literally emailed them about 5 times saying I just wanted my regular account but they completely ignored me. So I cancelled my account.

PayPal is a risk-management company, not a bank. When companies like this start asking for people's Social Security numbers, driver's license and copies of utility bills, something very, very wrong is happening with the way we make online transactions.

This is why it is extremely important for you to take your own measures to protect your information. If you actually trust a company like PayPal to protect your core identity information, then you're simply asking for your identity to be stolen. I know that may sound harsh, but PayPal *will* be breached, and your data *will* be exposed. It's simply a matter of time.

Don't think about damage control – think about damage prevention.

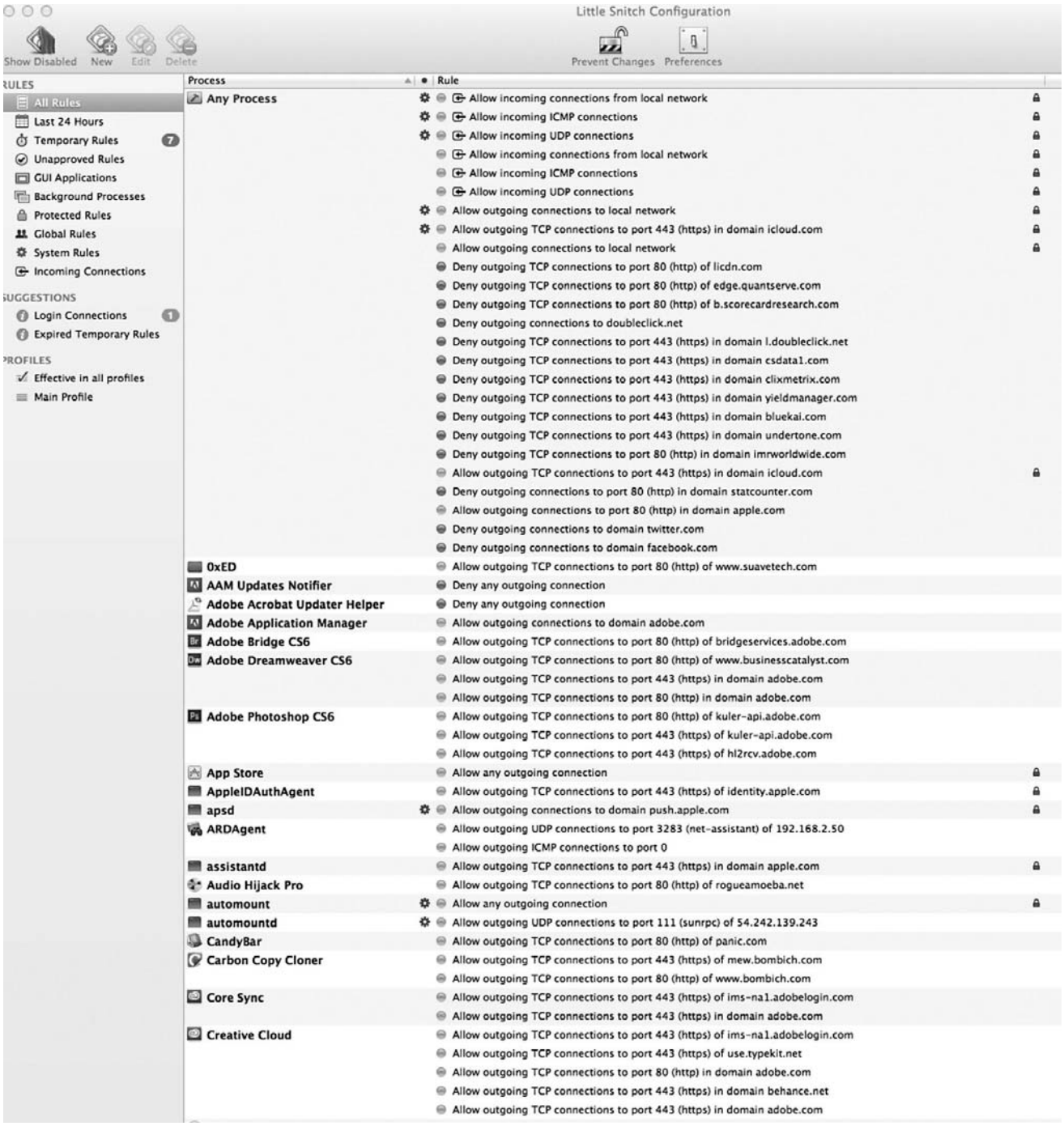
SECTION TWO: TECHNICAL PRIVACY AND SECURITY – LIMITING ACCESS TO SITES

Mac OS X ships with an Apple-developed browser called *Safari*. Safari is a perfectly capable browser, and many people (I presume) are happy with everything it does. I, however, choose to use Firefox as I find it to be a superior stand-alone browser with far more configuration options available to ensure your safety and privacy. In this section, I'll be using Firefox in conjunction with an application called Little Snitch, a third-party application protocol firewall. Normally a third-party application would be part of a separate chapter regarding third party software, but I discuss it here as it directly

impacts your ability to secure and control what protocols, applications, and destinations your data is bound for.

First, let's talk a little bit about Little Snitch, a for-pay firewall application developed by a company called Objective Development. While OS X does indeed come with its own firewall capabilities (covered in a different chapter) – in fact some extremely powerful and granular capabilities – I consider Little Snitch to be a requirement for any OS X installation, as its usability and power is incredible in its own right. In its most basic form, Little Snitch is an application that runs in the background, watching every outbound packet to see if its protocol type is allowed, if it is allowed to a specific destination port, and if it is allowed to any particular host, or domain. You can tell it to have "static" rules such as *deny all outbound HTTPS (port 443) traffic to doubleclick.net for all time* or *deny all outbound HTTPS (port 443) traffic from Firefox to doubleclick.net until I close Firefox. If I open it again, ask me what I want at that point.* And of course, you can have global rules such as *allow all traffic from all applications to hammerofgod.com* or *deny all traffic from any application to facebook.com* (I actually have this as a rule).

There are any number of other configurations, profiles, and rules you can leverage, but for the purposes of this chapter, I'm going to concentrate on Little Snitch's capability of asking you what behavior you wish it to take each time an unknown connection is made. Another way of saying "unknown connection" would be to say "ask each time a connection is attempted where an allow or deny rule does not already exist." Here's a shot of the Little Snitch rules interface.



In normal browsing scenarios this can actually be a bit tedious to manage for rules where you validate connections each time an application is opened, but in the following examples you see why this is important.

I'm sure most of you know this, but for those who do not, browsers offer different functionality via small files called "cookies" that each site you visit can

(by default in most cases) create and store data relevant to your connection. Many, many sites use cookies to maintain persistent logon information as you move around within a site, and others are used to exchange information between sites by way of "redirects" or calls one site makes to another. Here's an example. Say you are a frequent user of Facebook. If so, you will have a cookie on your system in which facebook.com can store any manner of identifying information about you. So let's say you go to foo.com, and foo.com wants to deliver an ad to you, and is using Facebook to do so. Foo.com can't read the Facebook cookie because it is encrypted and only facebook.com can read it. But what it can do is have a small piece of code in the web-page make a separate connection to facebook.com, which can then extract its cookie, read what information it needs to deliver an advertisement to you, and then pass that data back to foo.com along with whatever data foo.com wants to collect on you, provided Facebook supplies it.

When cookies are enabled by default, this will happen in the background and you'll never see it. However, if you disable cookies for particular sites (or even better, only *enable* cookies for particular sites) it can actually affect functionality of the site if the web developers are not conscious enough to check for cookies being enabled or not. I'll give you an example of that in a bit.

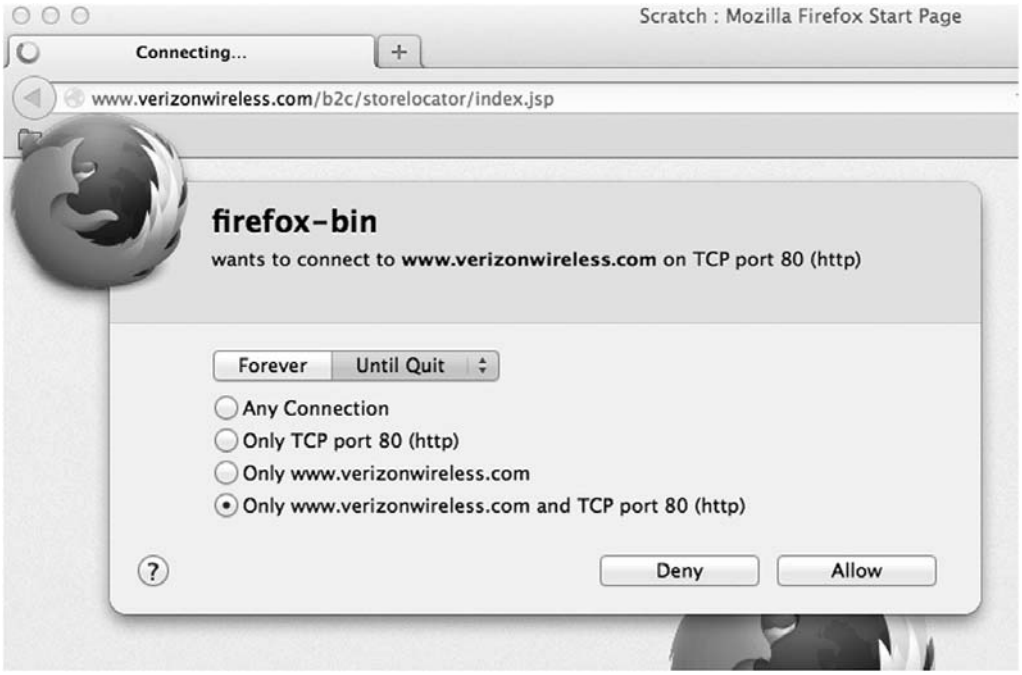
In this example, we'll be using a default installation of Firefox, with Little Snitches network monitor disabled. I don't want to clutter up the book with a mass of trite screenshots for every little configuration, so I'll leave it up to you to install Little Snitch and figure that part out.

Let's fire up Firefox and go find a local store from the Verizon Wireless site at http://www.verizonwireless.com/b2c/storelocator/index.jsp. Easy enough: of course, we enter that URL and we get the full Store Locator page where I can look up a store by zip code.

However, what other connections are being made that we don't know about? It's just a simple store locator, so it can't really be all that bad, right?

Now let's turn Little Snitch's network monitor on and fire up Firefox again and see what is going on in the background when we visit http://www.verizonwireless.com/b2c/storelocator/index.jsp.

A Little Snitch confirmation window now pops up asking us if we want to allow the connection to the www.verizonwireless.com host. This is expected, of course. The default option is to apply whatever action you choose (deny or allow) to the application requesting the connection until that application quits. So in this case, if I clicked "allow," then the connection would be made, but the next time I started up Firefox and went to www.verizonwirelss.com it would ask me again. You can see the other options available in the dialog box.



Note: in this case, the application referenced is "firefox-bin" which is the actual binary application running and not "Firefox" as you would normally see. That is because I started Firefox a bit differently, which I cover below.

I'm going to allow this connection Until Quit. Immediately after allowing this connection, I get another dialog box asking me to confirm a connection to verizonwireless.ugc.bazaarvoice.com.

And then a connection to cache.vzw.com…

And it keeps on going. And after this connection, we get a request to investor.google.com.

There is a "more information" option in Little Snitch's dialog which gives us the following text copied from it. As I continue, I'll just be providing the text and not screenshots of dialog boxes:

> **firefox-bin**
>
> wants to connect to **investor.google.com** on TCP port 80 (http)
>
> | | |
> |---|---|
> | **IP Address** | 74.125.239.98 |
> | **Reverse DNS Name** | nuq05s01-in-f2.1e100.net |
> | **Established by** | /Applications/Firefox.app/Contents/MacOS/firefox-bin |
> | **Process ID** | 32791 |
> | **User** | thor (UID: 501) |

Did we ask to be directed to investor.google.com? No, we didn't. Why would we be directed to investor.google.com if all we are doing is looking for a Verizon store? Because they are all exchanging data in order to build profiles on us, or more accurately, to continue building profiles on us to target "behavioral ads" and any number of other reasons. Where else does verizonwireless.com send us? Here's a full list of redirections, in order, which we would otherwise not have known were executing in background processes:

```
firefox-bin wants to connect to seal.verisign.com on TCP port 443
(https)
firefox-bin wants to connect to b.monetate.net on TCP port 80
(http)
firefox-bin wants to connect to verizonwireless.tt.omtrdc.net on
TCP port 80 (http)
firefox-bin wants to connect to es.verizonwireless.com on TCP port
80 (http)
firefox-bin wants to connect to safebrowsing.cache.l.google.com on
TCP port 80 (http)
firefox-bin wants to connect to crl3.digicert.com on TCP port 80
(http)
firefox-bin wants to connect to crl4.digicert.com on TCP port 80
(http)
firefox-bin wants to connect to akamai.mathtag.com on TCP port 80
(http)
firefox-bin wants to connect to log.invodo.com on TCP port 80 (http
firefox-bin wants to connect to t.acxiom-online.com on TCP port 80
(http)
firefox-bin wants to connect to analytics.verizonwireless.com on
TCP port 80 (http)
firefox-bin wants to connect to tags.bkrtx.com on TCP port 80
(http)
firefox-bin wants to connect to tags.bluekai.com on TCP port 80
(http)
firefox-bin wants to connect to view.atdmt.com.nsatc.net on TCP
port 80 (http)
firefox-bin wants to connect to ads.adrdgt.com on TCP port 80
(http)
firefox-bin wants to connect to adclick.g.doubleclick.net on TCP
port 80 (http)
firefox-bin wants to connect to ads.bluelithium.com on TCP port 80
(http)
firefox-bin wants to connect to www.google.com on TCP port 80
(http)
firefox-bin wants to connect to sales.liveperson.net on TCP port 80
(http)
firefox-bin wants to connect to p.acxiom-online.com on TCP port 80
(http)
firefox-bin wants to connect to d.monetate.net on TCP port 80 (http)
```

```
firefox-bin wants to connect to gtm01.nexac.com on TCP port 80
(http)
firefox-bin wants to connect to scache.vzw.com on TCP port 443
(https)
firefox-bin wants to connect to dpm.demdex.net on TCP port 80
(http)
```

Finally, after all these connections are made, we can then look for a store. However, after we submit our request, even more connections are attempted:

```
firefox-bin wants to connect to blip.tv on TCP port 80 (http)
firefox-bin wants to connect to s.amazon-adsystem.com on TCP port
80 (http)
firefox-bin wants to connect to 20505771p.rfihub.com on TCP port 80
(http)
firefox-bin wants to connect to insight.adsrvr.org on TCP port 80
(http)
firefox-bin wants to connect to d.agkn.com on TCP port 80 (http)
firefox-bin wants to connect to action.media6degrees.com on TCP
port 80 (http)
firefox-bin wants to connect to b.collective-media.net on TCP port
80 (http)
firefox-bin wants to connect to t.brand-server.com on TCP port 80
(http)
firefox-bin wants to connect to ingest.fwmrm.net on TCP port 80
(http)
firefox-bin wants to connect to sales.liveperson.net on TCP port
443 (https)
```

What is rfinhub.com? What is media6degrees.com? And what about demdex.com? Who knows?

I'm not suggesting all of these connections are "evil," but in many (if not most) cases we have no idea what data is actually being exchanged between our browser and any given host, or what data is being exchanged between these third-party hosts by way of cookie data.

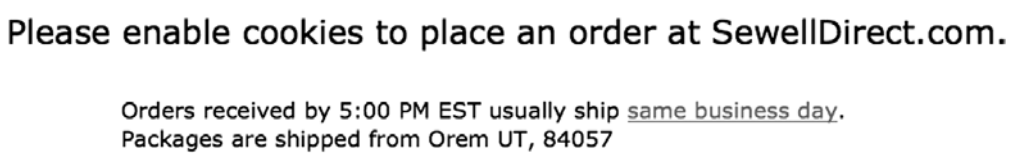
Again, this was just one simple visit to what we would have thought was a single website where we just wanted to look up any given Verizon store's location. I hope this one example will give you some level of insight into how our personal information is being violated.

To be sure, we certainly could have decided to "deny" any or all of the connection requests, but in some cases doing so breaks the website functionality. If you don't allow cookies to the Verizon site, the lookup function simply doesn't work and you won't know why. This isn't always the case, and there actually may not be any issue with a Verizon cookie in itself, but you just

don't know. I've successfully blocked most ad harvesting sites altogether, but in the following Google example you can't.

When you make a submission to Google and choose a result from your search, that result is encrypted and you can only be redirected to the site you clicked if you allow your submission to be parsed by static.google.com. So you can't even use the site if you don't allow them to collect your data. I've been able to "work around" this with some copy and paste foo, but for the most part it just won't work.

This is why simply blocking all cookies isn't a very feasible option. Web sites just won't work and developers are too lazy to check if cookies are required or not. You'll find exceptions as I did with SewellDirect.com, a provider of electronic components. As you'll see in the next section, by default, browser behavior is to block all cookies. When you block cookies at SewellDirect.com, however, you'll see this:



I wish other companies would provide this level of detail in your browser experience.

What we'll discuss next are some ways to better control what data is sent, and to where. We'll do this with a combination of only allowing cookies to approved sites (sites you specifically identify as trusted), as well as the complete blocking of any connection to specific sites.

Firefox "Profiles"

There's just no getting around the fact that you will have to allow connections to some sites and allow cookies to get anything done. This will be the case for your banking sites, sites you logon to for services (such as Facebook), and any other number of sites. I personally don't use Facebook in any capacity whatsoever because I know what they are doing with my data. Most people are not willing to do that though.

So let's start with sites where we need to get work done. In this section we're going to create a "profile" in Firefox allowing us to configure specific browser settings which are distinct from other profiles. For instance, my "work" profile is called `ThorProfessional` while my "don't care" profile is called `Scratch`.

The beauty of profiles is that Firefox allows you to select which one you want to use when you execute the browser binary. By default, Firefox loads with a default profile without asking you if you want to choose, create, or delete any given profile. As such, you're going to have to launch Firefox in a way that tells it to ask for a profile. This is done by directly executing the binary in Terminal (I use iTerm as discussed in Chapter 3) with a `-p` flag.

Rather than clicking your Firefox icon, open up a session in Terminal/iTerm and type the following:

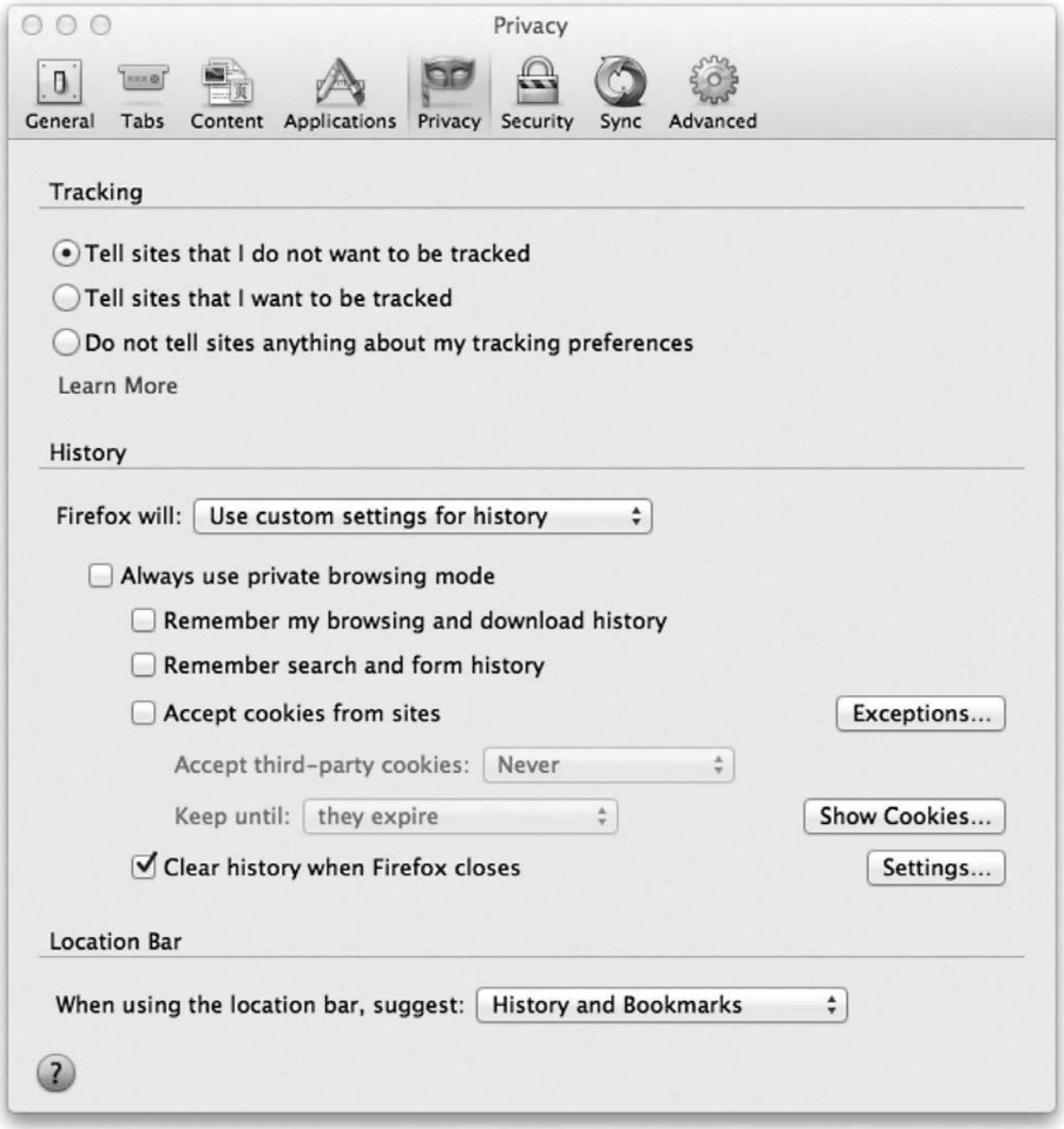
```
/Applications/Firefox.app/Contents/MacOS/firefox-bin -p
```

This will start a new instance of Firefox with the profile manager enabled showing your "default" profile.

We're going to create a new profile – in my case, `ThorProfessional` – by clicking the "Create Profile" button which shows an introduction dialog box and then presents you with the create form.

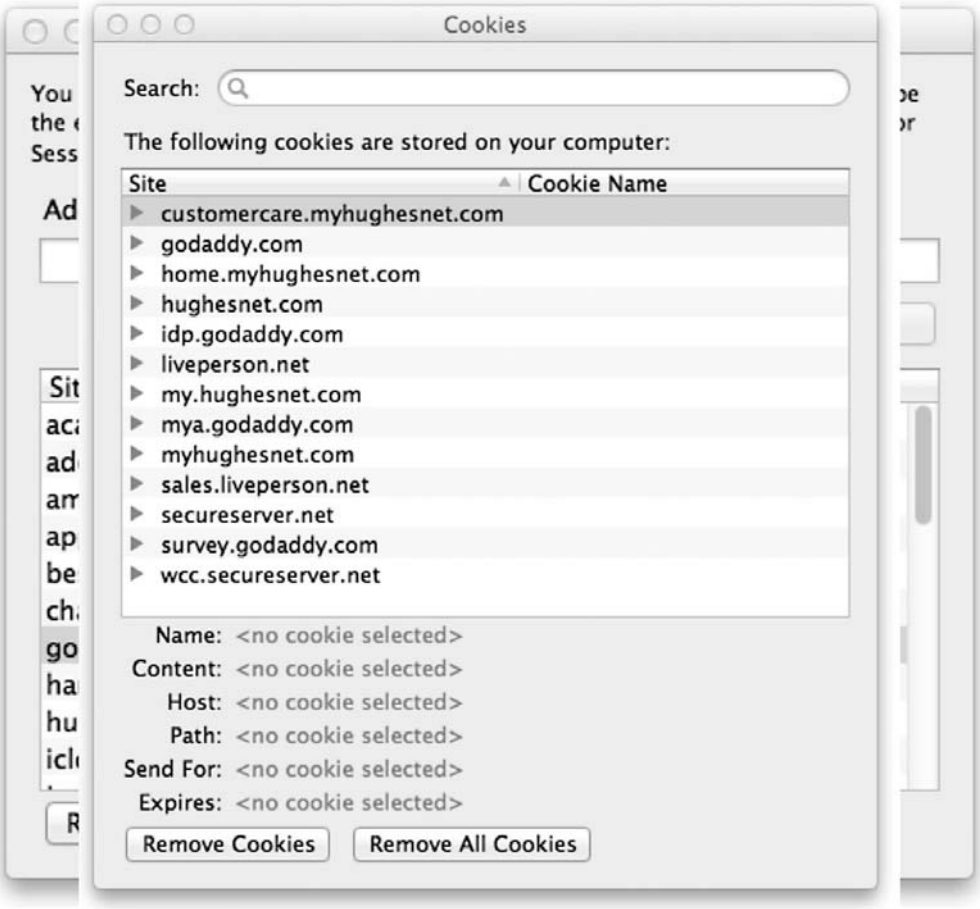
Once that is created, you will see the "Choose User Profile" dialog box with the newly created profile selected. Go ahead and Start Firefox with this profile so we can begin customizing our preferences. Once you've created the profile, you can start Firefox normally and you can choose which profile to use.

Pull up the Firefox Preferences dialog box and select the Privacy icon. This is where we'll set who we will allow cookies from and other behaviors.



By default I tell all sites I do not wish to be tracked (some, of course, couldn't care less what you want) and in History I've deselected "Remember my browser and download history," "Remember search and form history," and "Accept cookies from sites." The last one is what prevents my browser from accepting any cookies from anyone. As mentioned before, this won't work with our work-related and other cookies-required sites. You'll note the "Exceptions…" button – here is a clip from the sites that I will accept cookies from.

I've got several sites in this collection, but the main point is that only these sites can ever place a cookie for my browser. This alone can be a very, very powerful way for you to prevent other sites from placing cookies and tracking your data.



You should be aware that maintaining an "allow only" process can be a bit tedious at first, but since I spend most of my time in `ThorProfessional` all day, it's totally worth it for me. And actually, the sites I visit in `ThorProfessional` almost always work with cookies turned off; it's only

when I want to logon somewhere or order something that I need to add the exception for any given site. After months of using this particular profile those are the only sites with cookies on my system for `ThorProfessional`.

That's it – really! Each site can have any number of actual "cookie" files, but these are the only ones I've allowed to do so. Other places in my Exceptions list don't create persistent cookies, as they don't need to.

Understand there is a difference between what cookies you allow/deny in Firefox and the functionality provided in Little Snitch. Firefox will prevent a site you visit from dropping a cookie via your browser. Little Snitch will allow/deny access to the site in the first place.

Now that we've covered a "professional" or "limited" configuration for enabling cookies per site, let's go back and talk about the default profile. It is not feasible to only have the professional profile where you select sites you want to visit. You can certainly try it as I do, but you might want to at least give yourself the option of having a "standard" or "default" profile you can launch when you have enough work to do on sites that require cookies that you don't want to go through the hassle of setting up in your professional profile; nor should you.

You can either change the default profile to suit your requirements or of course create others. I actually have a few different profiles. In fact, in one case where I was contracted to design a security curriculum for Microsoft Azure (which I affectionately call "OhSure") I created one specific profile for work on that project because of the wide requirements Microsoft had for cookies being enabled in order for their overall service offering to function correctly. Oh, one quick bit o' trivia for you. Microsoft has named their cloud services "Azure," even though azure is the color of a cloudless sky. Go figure.

Anyway, as an example of exactly what can go on in the background on the default cookie configuration of Firefox (or any other browser for that matter) I've added a process to the Advanced Configuration section of this chapter called "Low Level Firefox Profile and Configuration Editing" which will allow you to access the database where Firefox stores cookies and configuration information. This is a *far* superior design to the Windows solution, where Internet Explorer just dumps cookies as actual files to the file system, which then must implement convoluted access controls and permissions to them and which also depends upon the System Registry (or as I call it, *The Bloated Single Point of Failure* solution) to function at all.

Regardless, the functionality illustrated here should give you rich opportunity to learn and play as you master your browser configurations to enhance your security and protect your privacy.

Alternate Search Engines

The easiest way to keep Google and Bing from harvesting your personal information is to simply not use them to search for things. I have a few alternate search engines I use which are not commercially engaged in selling your information, or even tracking it for that matter. One such search engine I use when I can is "DuckDuckGo," a privately funded search engine product which relies upon a number of different sources to provide search results. This is via API (Application Programming Interface) calls to other search engines as well as "crowdsourced" resources like Wikipedia. DuckDuckGo will make these searches for you, and return the results without tracking your information. Though it may be counterintuitive, this model ensures that all results are "equal" among users. If you go to Google and search for "Hooch Dog Diggity," your result set will be different than if I search for the same thing. To be sure, I've never searched for "Hooch Dog Diggity" before in my life because the term just popped into my head while I wrote this. But that's not the point – the point is that Google *filters* your results based on what Google thinks you want, or more accurately, based on what results Google wants you to have. The same search criteria entered into DuckDuckGo will return the same results irrespective of who requests it.

This is important to me. Honestly, I don't see how Google can get away with filtering results to specific individuals when we, for the most part, are under the assumption they actually return results based on what we asked for, not what they want to give us. What is even more concerning to me is that Google not only filters results, but that over time, the results have become purposefully reduced in relevance. This means they are "padding" results with slightly off-kilter content, which makes you visit places you wouldn't normally visit. This of course drives up ad hits as you are sent to places you thought contained the content you were looking for.

This can't happen with DuckDuckGo, and that's a good thing. However, this comes at a cost. DuckDuckGo only has a few employees and is funded by an entrepreneur who made millions on the sale of a previous venture. I like seeing people "give back" to the community that supported their endeavors. As such, they simply can't afford to spend the billions and billions Google puts into research and development, and this is reflected in the result sets. It's not that they are convoluted or inaccurate, it's just that they are not quite what we are used to with spoon-fed results (force-fed?) from Google so you have to do a bit more exploring to get the things you may want. That said, you'll also get results Google would never have given you so it's a bit of a trade-off. I use DuckDuckGo when I can, but even I must sometimes capitulate to the technical capabilities Google has and use their service. But at least we have a choice and can make decisions for ourselves. Another alternative is Privatelee which I also use, and is "Tor" friendly. I suggest you explore other options if you are

concerned about your privacy. And, speaking of Tor, let's discuss what that is and what it can do for you.

TOR Proxy

Let's now talk about your IP address, which is probably the single most exploited element in the attack on your privacy. Being true to my "anti-bloat" policy, I won't go into details about what an IP address is as you can DuckDuckGo it for yourself, but I will cover it a bit.

Your IP is supposed to be a unique numerical identifier as to the Internet Protocol address you are coming from, similar to what physical address you live in. However, it doesn't actually play out that way. Your IP address and the IP address reported to web sites and lookup services can be completely different. It all depends on your provider, the time of connection you have, and the way you connect up to the internet. By way of example, I used to have a static business circuit from Comcast in my home – this means the IP address reported was always the same. This is (for the most part) required when you host your own services.

However, if you were to look up my IP address and map it back to a geographical location, it could return any router address Comcast supported as their infrastructure routing is completely different than the egress point your actual "external" IP is sourced from. So while I lived in the Seattle area, my lookup could tell you I was in Denver somewhere.

You'll see shows and movies where the almighty IP address is used to immediately identify the physical location of a bad guy, but for the most part that is simply ca-ca. If the police wanted my physical address, they would have to get it from Comcast, not some sexed-up stripper on CSI.

All that said, though your IP address won't immediately give up your physical location, it is indeed maintained as your identifying address for the purposes of logical association. It's like your cell phone – you can call someone from Seattle or Denver and the other side of the call won't know where you are, but it's still your cellphone number.

During social engineering engagements I always used a "caller ID spoofing" service to make my phone number show up differently than what it really was. I also used it to screw with my friends by calling them from their mother's phone number, which causes their cell phone to display "Mom" or whatever they've logged her number in as (assuming it is in their cell phone, of course). They'd answer and I'd proceed to tell them about my new, hot girlfriend whose house I was at right then. Good times.

With this knowledge in hand, you can see how an IP spoofing service can be valuable. Hence the introduction of a networking infrastructure called

Tor. "Tor" brings back some nice old memories of an oriental girl I used to date, as that's what she called me, but in this case it's a very powerful privacy tool.

The term "spoof" may be accurate for caller ID in that you are still calling from your phone but the displayed phone number is different, but it's actually not technically correct for Tor. I previously used the term "IP spoofing" incorrectly on purpose to support my example.

When you use Tor, you're not actually "spoofing" anything – your egress traffic actually *is* coming from a completely different system with a different IP address. Here's how it works.

Tor is an international network of privately supported "exit relays" literally located all over the world. These exit relays are supported by thousands of internal relays that route your traffic through them to an actual exit relay from which your internet connection is then sourced. The OS X Tor application is an extremely easy tool to use. When you fire it up and ask to connect to the Tor network, a list of relays is enumerated and different bandwidth, speed, and traffic algorithms are used to connect you to one of them with a secure channel. The relay you connect to will then establish a completely separate connection to another relay. The 2nd Tor relay connection is also made with a secure channel; however, the 2nd relay has no idea who you as a client are. The 2nd relay only knows about the 1st relay. At a minimum, a 3rd relay connection is made from the 2nd relay to yet another Tor relay. Again, the 3rd relay only knows about the 2nd relay. The 3rd relay does not, and cannot, even know what the 1st relay is. This can actually happen a few more times, but at least three internal relay connections will always be made. Then, finally, the last internal relay makes a secure connection to the Exit Relay from which the actual connection to the internet is made. Note that when I say the "relays" don't know anything about the traffic, that means the operators of these relays as well. Only the exit relay could monitor your traffic, and that's only if you used HTTP. Of course, this is no different than your provider or any other connection you currently make over HTTP.

As such, when you are connected to the Tor relay and wish to visit Facebook, for instance, your request is sent to the internal relay you are connected to, which is relayed to the 2nd, which is relayed to the 3rd, which is relayed to the Exit Relay that makes the actual connection to Facebook.

Now let's say you post video evidence of some cops in Kentucky beating up a pregnant woman. Of course, those officers and the agency they work for will say you have illegally recorded them without permission and will seek to charge you with a crime with a more severe sentence than rape in most jurisdictions. I'm not making that up, by the way. It's true.

Anyway, the cops go to Facebook and request full connection information on the client that posted the video in order to put the pinch on you. Fortunately,

thanks to the power of Tor, the only information they can get is from the Exit Relay, which is most likely physically located in Russia, Germany, or any other country in the world.

Even if law enforcement gets some manner of international warrant for the Exit Relay to disclose what information it has, the only possible information it can give is what connection it was made from. It does not know and cannot know anything else about the connection. In this way, it is impossible to trace that post back to you as the locations of the "blind" internal relays can't be traced back anywhere. It is a thing of beauty, and a powerful tool to protect your identity.

Running Tor on your individual Mac is almost trivial. Just download the Tor OS X application and run it. You'll see this window:

Simply click "Start Tor" and the application will do the rest. As described previously the relay-to-relay connection process will execute, and before you know it you'll be exiting the Tor network and onto the internet from the other side of the world (or wherever). Once the connection is complete, Tor will launch the TorBrowser application (a Firefox instance) for you, showing you your connection information.

Congratulations. Your browser is configured to use Tor.

Please refer to the Tor website for further information about using Tor safely. You are now free to browse the Internet anonymously.

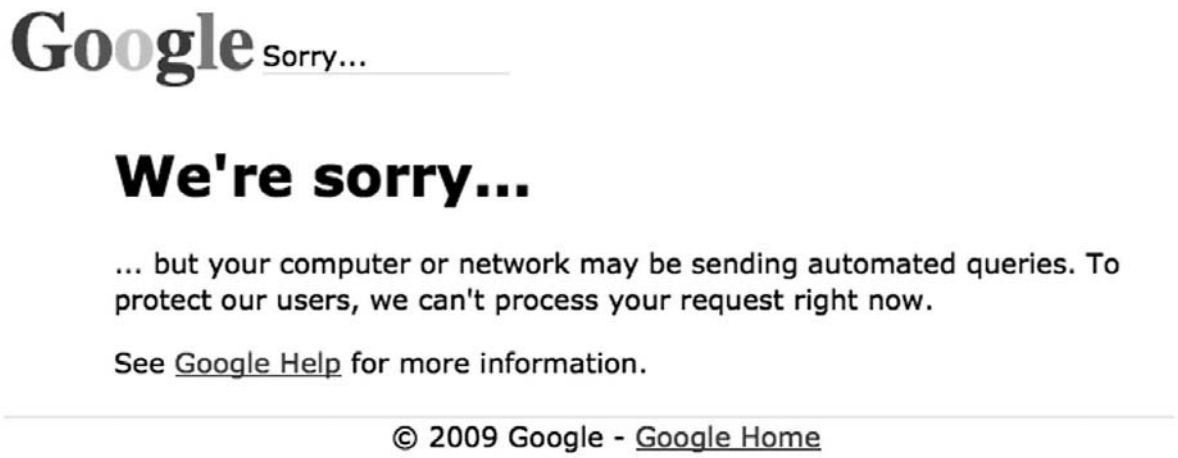
Your IP address appears to be: **212.96.58.189**

In this case, we've received the IP address of 212.96.58.189 which just happens to be in one of my favorite places:

Budapest is lovely this time of year. The identification of "No Proxy Detected" is significant – the IP lookup site I used has no idea I actually *am* using a proxy per se, but since my traffic is exiting directly from that physical relay, it thinks I'm not. And it's their business to provide accurate information.

You're ready to browse knowing your location and IP are absolutely private. Now, before you establish a permanent connection to Tor, there are a couple of things to know. First and foremost, since you are being relayed through a number of volunteer-owned systems, each with their own internet connections, the bandwidth you operate within will normally be a good bit slower than any "direct" connection through your ISP. But that's a small price

to pay for ultimate anonymity. Secondly, some sites may not provide functionality if they know you are coming from the Tor network. An example of this is, you guessed it, Google. While on the Tor network, a visit to Google yields this:



You see, Google monitors the Tor network for exit relay IPs. They make the totally bogus claim of your network "sending automated queries" but they explicitly know you're on the Tor network and block you accordingly. They are a commercial entity, and can certainly make their own decisions about what clients they wish to serve, but in this case they figure that if they can't track you, or if they can't verify your actual IP address, that they won't let you use their service. Again, I completely respect their decision, and if I were a company who made money by selling IP profile information I very well might make the same choice, but the important takeaway is that your IP identification as part of your permanent profile is so valuable that if they can't make money off of you, they block you from their service. To be fair, this doesn't always happen. I don't think they can keep up with the Tor network in real-time, and it honestly can't be that much of a priority for them, but just be aware it could happen. If it does happen, I've been successful in bypassing the block by directly visiting the localized Google service, in this case www.google.hu. When you do this, you'll obviously have to change the local language to English.

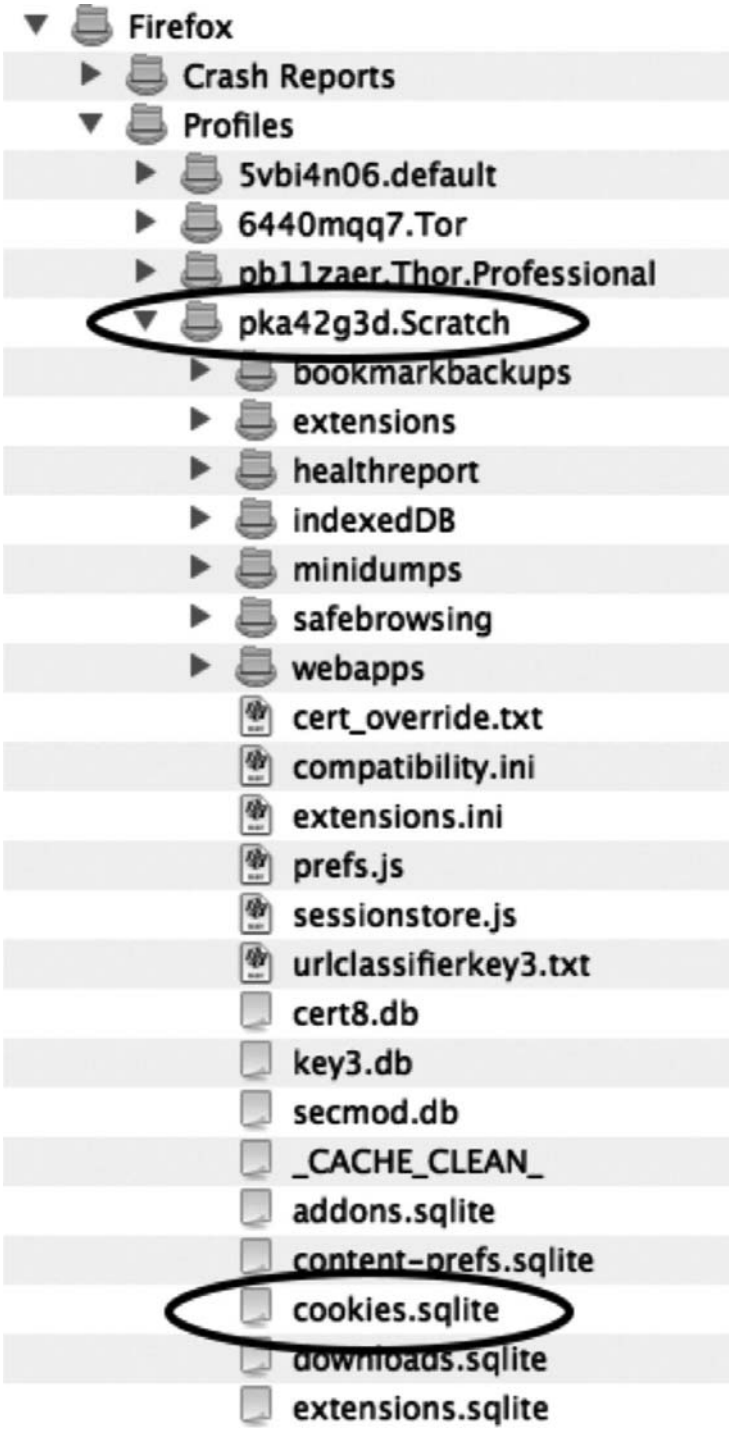
Advanced Configuration Example: Low Level Firefox Profile and Configuration Editing

As mentioned before, Windows applications of any consequence rely and are dependent upon the use of the System Registry to store application data. The System Registry is a shared repository, or as they call it a "database" of extremely complex and convoluted objects and elements. As a shared resource, if the registry gets corrupted or otherwise damaged, all programs dependent upon it will cease to function. The fact than any program you install reads and writes to the Registry means that program could potentially render the Registry inoperable. Microsoft plays down the risk of this happening, but it's easy to do. Trust me, I know – I've done it many, many times (for fun, of course).

OS X applications are actually "packages," which are a physical collection of binaries and support files contained within a directory structure and protected by default restrictions and whatever additional access controls you decide to place upon them.

In the case of Firefox, the applications required files exist in two places: the main Firefox "package," and supporting files in your home directory's ~/Library/Application Support/Firefox directory. The bits we're looking for are the files contained in the directories specifically created for each profile we've created in Firefox.

I've been using a profile called "Scratch" for my general profile used for dirty browsing. No, not "dirty" like that, dirty like "unclean" and "cesspool-eo" browsing. When I created the Scratch profile, Firefox appended the name with a randomized nomenclature and built the necessary support files as follows:

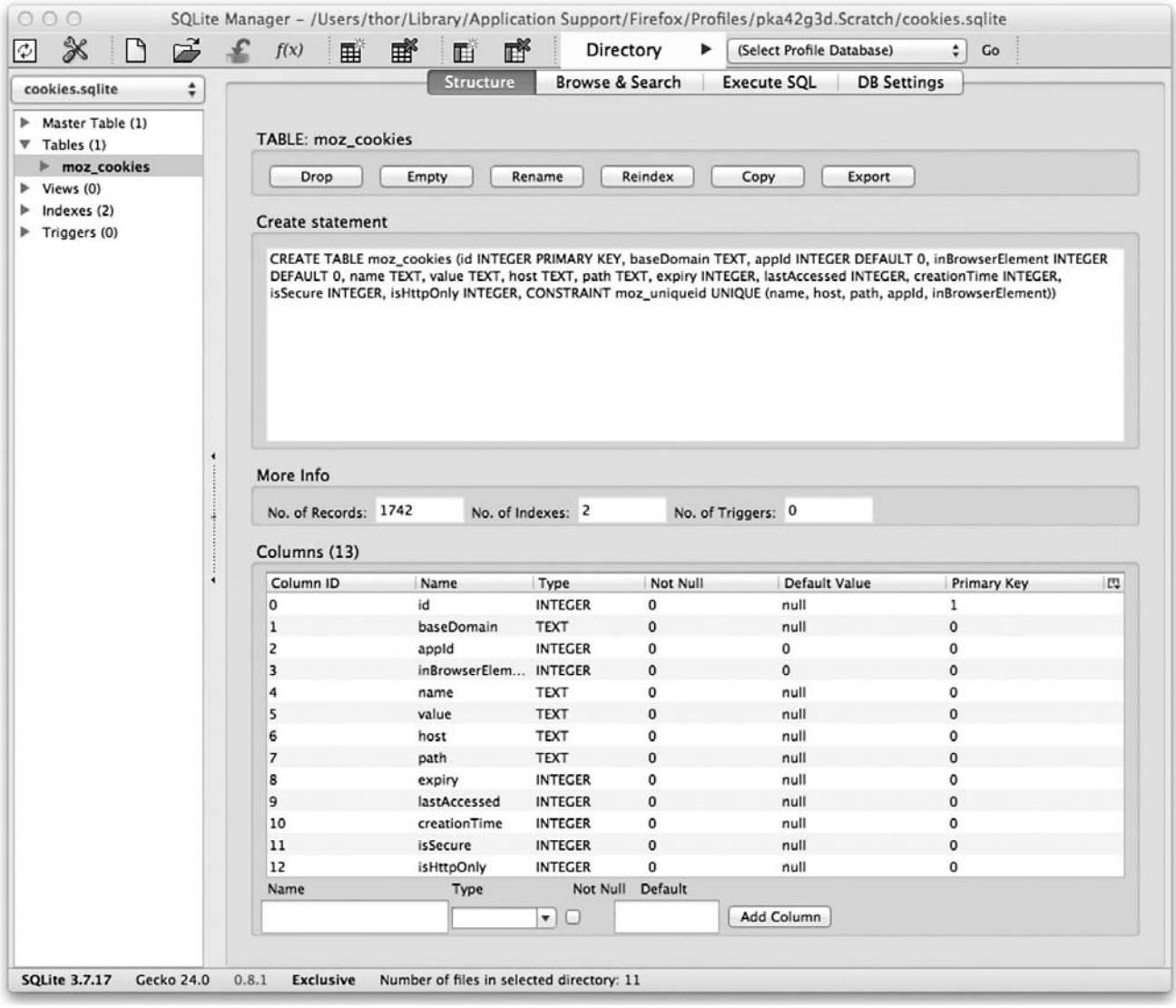

Firefox
Crash Reports
Profiles
5vbi4n06.default
6440mqq7.Tor
pb11zaer.Thor.Professional
pka42g3d.Scratch
bookmarkbackups
extensions
healthreport
indexedDB
minidumps
safebrowsing
webapps
cert_override.txt
compatibility.ini
extensions.ini
prefs.js
sessionstore.js
urlclassifierkey3.txt
cert8.db
key3.db
secmod.db
_CACHE_CLEAN_
addons.sqlite
content-prefs.sqlite
cookies.sqlite
downloads.sqlite
extensions.sqlite

Earlier in the general Firefox section of this chapter I discussed the minimal cookies afforded to us by a customized profile. For the purposes of this section, I want to show you what a dirty profile yields insofar as cookies are concerned. In addition to the pka42g3d.Scratch direction highlighted, you note the "cookies.sqlite" file. SQLite is a "server-less," in-process Structured Query Language (SQL) process allowing any program to store and manage data in a self-contained environment. Most SQL engines require a separate "server" process, which supports a client's requirements. However, SQLite is a standalone process, which doesn't require any other process to support its functionality. The great thing about SQLite is that it is free for use in any way you wish to use it; you may include it in private applications, public applications, commercial applications, or whatever else you want to do with it. It is also "open-source" in that the source code is readily available for download. This ensures there are no secrets in the code and that everyone is freely able to examine, first hand, the exact operational components of the application.

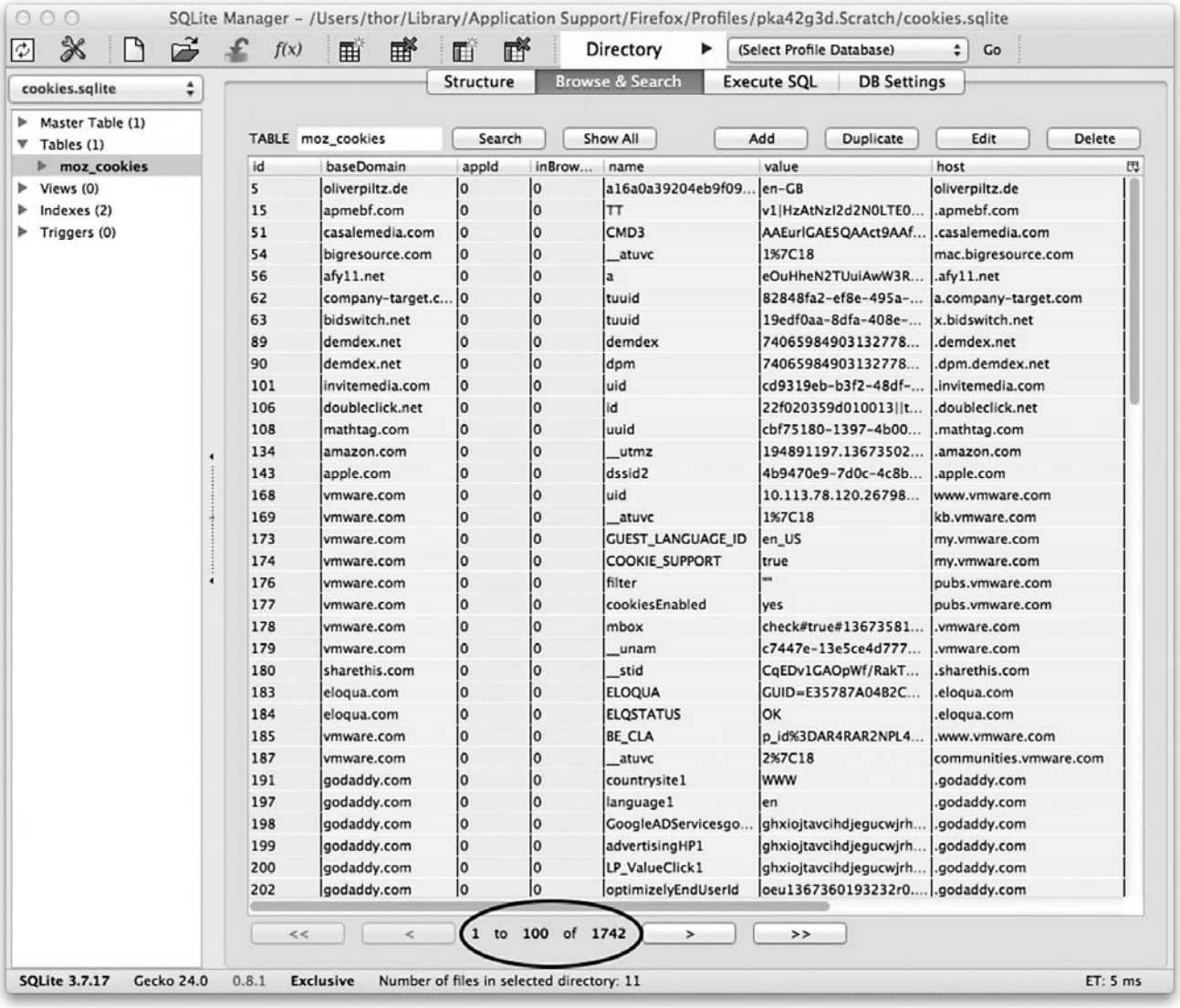
Firefox utilizes SQLite to store and retrieve data elements in order to provide its functionality, including the creation and access of cookie data. This section is two-fold: one, to show you the power you have in accessing Firefox data elements via the SQLite, and secondly to show you exactly how much crap is dumped on you when dirty browsing.

First you'll need to download the SQLite Manager, which is not included in the default Firefox installation. This is super-easy; just go to `Tools->Add-ons`. Look for SQLite in the search field and install it. There's a "restart" button you need to click, and upon doing so you'll see the `SQLite Manager` is now available under the `Tools` menu.

What we'll do now is launch the SQLite Manager and open the cookies.sqlite file and see what manner of nastiness we'll find lurking there. The quickest and most efficient way to do this is to click the "open file" icon and then use the OS X "Go To Folder" feature shortcut ⇧⌘G and directly access the ~/Library/Application Support/Firefox/Profiles directory. Drill down to the profile directory you wish to examine and select the cookies.sqlite file. The Structure tab is selected by default, showing the Master Table structure. Select moz_cookies under Tables.



We now see the entire cookies database structure, which is really quite cool. You can't do anything remotely close to this in Internet Explorer, by the way. Now, let's select Browse & Search and see what we've got.



Wow. We now see the actual contents of each cookie record. I cleared my cookies on this profile about a month ago, and I've already got 1742 cookies dropped. If you take a look at the `value` column, you'll see the contents of the cookie itself. Many records down in my cookie table is a record from hubpages.com. I've never gone to hubpages.com, but I did look about for some information on organic cereals in order to research the chemical composition of various products. I'm starting up an organic farm called Thorganics and this is of great interest to me (seriously). Somehow or another this hubpages.

com site was transferred as part of a Bing search for "organic" and "popular breakfast cereals." The really interesting thing here is that I specifically used Google to search for research sites. I purposefully never use Bing in the profile because I want to see exactly the extent of needle-sharing these sites engage in. Some site picked up my Google search by way of a referral, and that site then decided to fire off that search to Bing. hubpages.com apparently decided to transfer search authority over to Bing instead, and then dropped a cookie on my system so that any other site who wants to use hubpages.com to extract available information can do so via a redirect. The actual contents are:

```
209211014.1368496131.1.1.utmcsr=bing|utmccn=(organic)|utmcmd=organic
|utmctr=popular%20breakfast%20cereals
```

Oddly enough, further down is a cookie from ranker.com, which has this as the contents of the cookie:

```
51811754.1368496147.1.1.utmcsr=bing|utmccn=(organic)|utmcmd=organic
|utmctr=popular%20breakfast%20cereals
```

WOW! Talk about a coincidence! As you can see, both companies have their own identifier for me, yet both have the exact same information for the search criteria. I guess it's a good thing I didn't search for "Christina Hendricks' Bra Size," huh? Oh, wait...

When surfing dirty you have no idea what information is being encoded, such as lifehacker.com's `f5ptmd6egfelt9r1.1367875279554.1367875279554.00000000000001` cookie value. Only they know what those characters all mean. Hopefully this will give you some idea of what's going on when you browse, and the importance of leveraging the procedures covered in the general Firefox Profiles section.

Advanced+ Configuration Example: Shared Tor Proxy in Virtual DMZ Environment

I'm calling this section "Advanced +" because, well, it is. There are professional engineers who can't do this, and some who don't even know it's possible. Regardless, the reason I'm in this business is because I am continually fascinated by what I learn on a daily basis, and it is one of the true, authentic joys I have. As such, I'm not going to assume that since this book is targeted at a much wider audience than I normally write for, I should further assume you won't have any interest in what we're about to talk about. If only 1% of the people reading this have a creative spark struck by this chapter, then it was worth my time. That said, even if you don't plan on implementing this, I think you should give it a read just to see what your possibilities are.

We've previously discussed installing Tor on a client-by-client basis for anonymizing various protocol traffic. What that means is each client we

install Tor on (such as your laptop, your work machine, your spouse's system, etc.) will make their own connection directly to the Tor network, with each system's internet egress exiting out of a different exit node. Now, let's engineer a more advanced configuration where we will build a DMZ segment (meaning "demilitarized zone" though I am loathe to continue supporting the term) using a free and remarkable product from VMware called ESXi. Let me qualify what "free" means in this context. The ESXi server product is indeed free. However, the enterprise tools and server components to run ESXi in a production environment are actually quite expensive, but totally worth it for any given company relying on virtualization technologies.

This chapter isn't just about a Tor proxy – in fact, that just happens to be the service we will be deploying. The *real* value of this chapter is that it will show you how to create a far more secure DMZ segment for you to deploy whatever services you want to deploy. So as you read on, keep in the back of your mind the fact the functionality described herein extends far, far beyond the provisioning of a proxy.

Once we've built the DMZ, we will launch and configure an OS X virtual instance to act as our Tor proxy server *and* Tor relay. When we installed Tor on individual machines, the Tor Browser was configured to connect to the machine's local loopback address of 127.0.0.1, in order to then pipe SOCKS requests out of the outbound Tor connection from the IP address bound to your ethernet card.

In this case, we will start up Tor the same way as we did on clients on our DMZ OS X box, but change its configuration from listening on the loopback 127.0.0.1, where only *it* could connect, to instead configure Tor to listen on the IP address bound to the NIC itself (which in the example will be 192.168.2.50). Diagrams follow, but I want to make sure I properly cover the concepts in text first. In this way, all of your client machines can connect directly to the DMZ Tor box for outbound Tor browsing simply by changing the SOCKS proxy in Firefox and not installing Tor on them all. This does two things: one, you don't have to install Tor on your clients at all in favor of a common proxy, and two, you aren't making multiple connections to the Tor network but rather sharing just one.

Secondly, we will configure the DMZ OS X Tor machine to be a Bridge Relay (also described as an "internal relay") so we can become part of the overall community protecting freedom and privacy by allowing the DMX OS X Tor box to act as an intermediary between other Tor boxes as the traffic makes its way to the Exit Relay. I've configured my Tor installation as an actual Exit Relay. That's where the traffic finally exits to make the direct connection to the web site or mail server. As such, some consider running an Exit Relay can be

"risky" in a sense, because certain law enforcement agencies could be obtuse and think you are up to something shady if you support Tor exit relays. I'll leave that for you to research and get on with this configuration.

ESXi is a virtualization platform where you create a dedicated host machine capable of running as many virtual machines as your hardware configuration allows, and then some. It is quite amazing what products VMware has produced, and ESXi is at the core. If you use HyperV at the moment, as far as migrating to ESXi VMs is concerned, there's really not much to talk about. You simply can't run OS X VMs on Hyper-V. That said, if you hack OS X up enough until it looks like POS X, then you might be able to, but that's true with anything. That said, if you do have existing Microsoft Hyper-V machines, VMware will easily convert them to either VMware Desktop or the ESXi VM format where you can simply add them to your ESXi inventory and be on your way. Microsoft Hyper-V is nice for a few VMs here and there – I used it myself for years. But even Microsoft doesn't use it in production where it "counts." However, they do market it as a production-class platform, but I don't understand why. There's really no comparison to ESXi, which is indeed a "true" enterprise product. That's why we're using it here. But enough of that…

So let's get started. I'm glad you've read this far in the Advanced section of the chapter as this will be quite cool. While I'd love to start at configuring the ESXi server for this bit, I must be true to the book and push through to the part where OS X comes into the picture. Actually, even if you don't plan to do this, I think you might enjoy the processes covered herein.

We will need to cover some rather extensive ESXi configuration options first, so expect some priming here. Again, as this is the only host-based virtualization product that allows you to run OS X, it's very good for you to learn about it, though it will be somewhat complicated.

To give you some context, the soon-to-follow Diagram 1.1 is a quick look at my overall ESXi deployment, consisting of 2 (two) ESXi servers, each with their own set of virtual machines.

One quick note though – this is a screenshot from the vCenter product which is a for-purchase application (and not a cheap one either). However, for the most part the view is the same as the free client software you'll use to connect, named vSphere. The difference is vCenter shows all servers and offers advanced configurations while the use of the vSphere product, being directly connected to the ESXi host (as opposed to connecting to the vCenter host/guest) only allows you to connect to one host at a time; but this should be more than enough for your network. It is important to note this setup is running on Mac Minis which make great ESXi servers. The real added bonus is

you are licensed to run as many OS X images as you wish on Mac equipment. Why have one OS X box when you can have five??

The topology I'm describing is my actual production setup. If we were using the vendor-speak so prevalent in today's market, we would call this my "Private Cloud." A Private Cloud is a collection of virtual machines you already have set up. It's what we called our "LAN with Virtual Machines" before marketing people got hold of it. However, in order for you to buy what you already have, they call it "Private Cloud" so you feel like you're on the cutting edge of stuff you've had for years.

You've probably noticed other sources and sites showing screenshots where they've obfuscated computer names and internal IP addresses. This is a silly practice. It's unfortunate because it shows these authors don't know much about security. Knowing the names of my servers or their internal IP addresses gives no valuable information to an attacker. Redacting portions of the data by way of blocks or blurs only serves to confuse people. And in most examples and test setups this sort of thing hides important components. So what you see here is all live. How's that for trust?

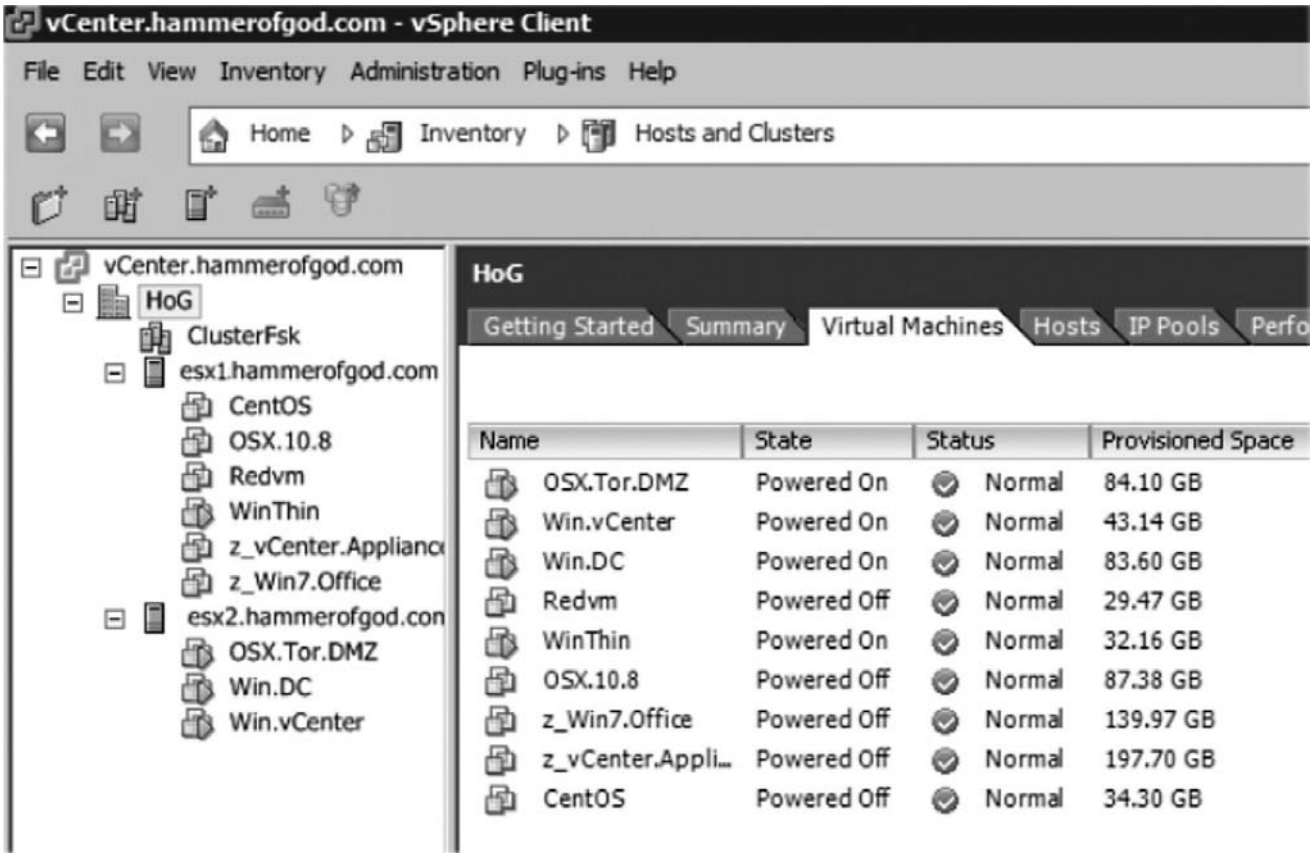


DIAGRAM 1.1

The two ESXi servers are named, originally and creatively enough, ESX.1 and ESX.2. I know. It's a gift. My testing and utility VMs are on ESX.1, and the OS X VM we'll be using for a Tor relay and proxy is on ESX.2. You'll notice I also have my vCenter vm and my Windows domain controller system on ESX.2 as well. The first thing that may pop into your mind is "You, sir, are crazy! How can you deploy of all things a Tor server on the same ESXi host where your domain controller and vCenter server lives? It's madness!!" Excellent question and concern, and I'm here to show you how I can do so and still sleep at night, albeit with assistance from Trazodone.

Diagram 1.1 shows a datacenter named "HoG," a cluster named "ClusterFsk," the ESX.1 host and its VMs, and finally the ESX.2 host and its VMs. This will be the main interface you will use to manage your VMs. This image reflects a "flat network" configuration, meaning all IPs assigned to hosts, guests, and management interfaces are on the same network; in this case, it is 192.168.1.0/24 (or 255.255.255.0, if that is your preferred nomenclature). These machines all live in the same network, and all have only outbound access to the internet.

There are no existing inbound connections being made to this network, or what one may call "port forwarding" from your ISP's "modem" or router. After we go over this diagram we'll no longer need to refer to ESX.1, as all ESX.1 assets in this example will stay on the 192.168.1.0/24 network without changing configuration. ESX.2, however, will indeed be changed.

The network topology in Diagram 1.2 represents where we will be at the *conclusion* of this chapter as it regards the ESXi configuration.

The typical home or small business network consists of workstations plugged into some manner of switch, with that switch being subsequently plugged into a router or what some vendors call a "modem." In many cases the term is inaccurate as digital-only transfers are very common. If one has tendencies toward pedanticism, the term "routing modem" may be appropriate for you. Additionally, your router probably has switch components built into it, typically designed with a small number of ethernet ports (4 or so) for the "LAN" and 1 port for "WAN" or "Internet." I've a Comcast Business Router which actually has the RG58 cable port directly integrated. Lastly, most recent routers support either a dedicated DMZ port, or allow for such functionality via configuration. My Netgear actually has both.

Your internal network's LAN IP addresses will be RFC1918 addresses, meaning they are considered "private" and nonroutable outside your network (e.g. the Internet). You'll have at least one "external" IP address provisioned by your provider.

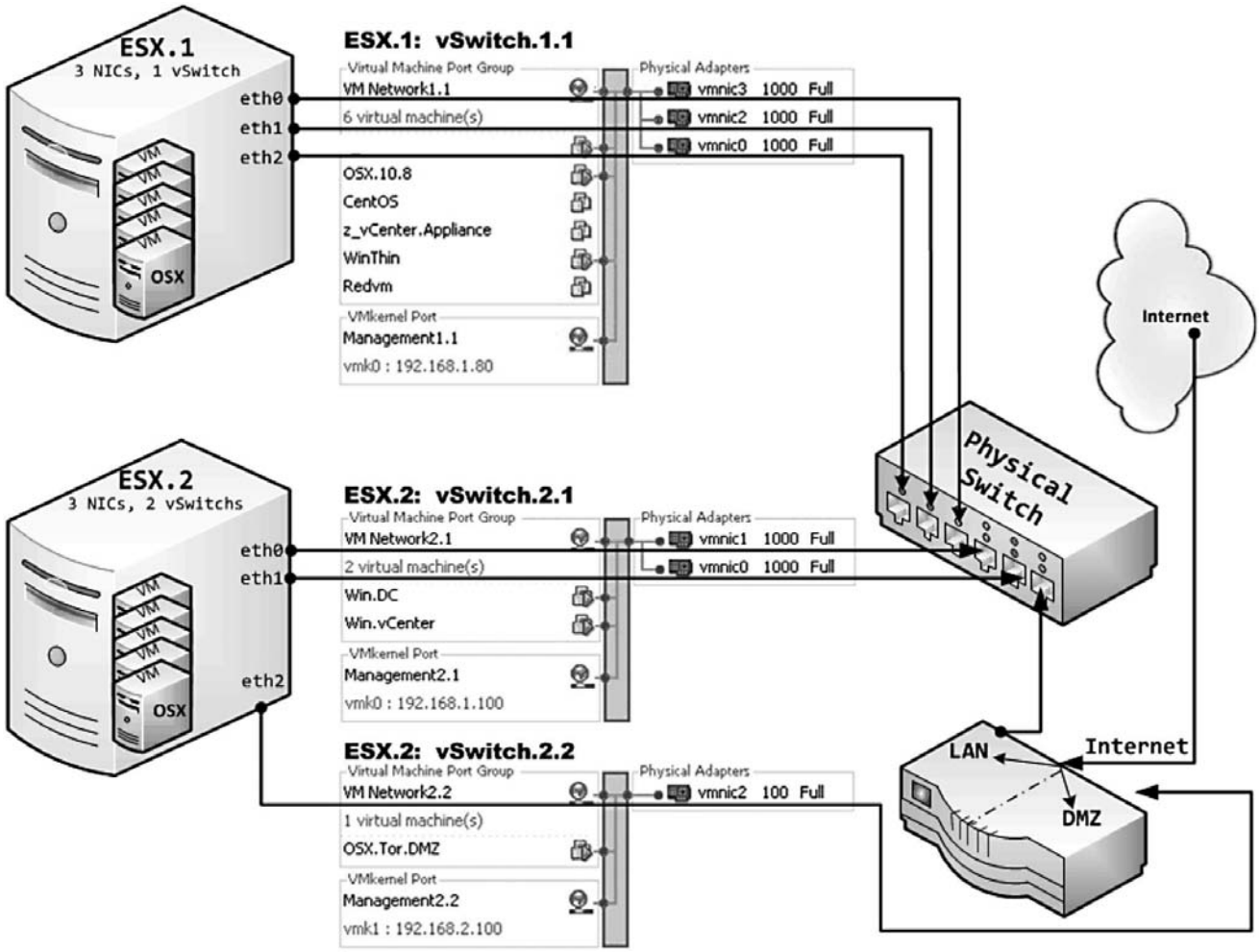


DIAGRAM 1.2

My network is probably a bit more complex and scaled higher than most home networks (and even some businesses) but my configuration is essentially the same as described. The www.hammerofgod.com external address is 173.190.165.173. I've other static IPs available but that's my primary one. All of my internal machines, other than what we're about to change, are in the 192.168.1.0/24 network and exit via the 173.190.165.173 address.

At this point, as we continue with configuring ESXi, things may get more confusing, as what are normally hardware components will now become virtualized components. Regarding Diagram 1.2, consider the initial vCenter Diagram 1.1 where we showed the "flat network," primarily the fact that all switches, NICs, hosts, and VMs were internal on 192.168.1.0 /24. With that in mind, let's focus on the network portion of our goal.

Diagram 1.2 shows two physical servers, each with five virtual OS X machines. You can already feel the power at your command!! Each host has three physical NICs. The first concept for us to cover is that of the "virtual switch." The *physical* NICs are plugged directly into the *physical* switch. However, the NICs themselves are *not* assigned IP addresses as one would typically do in a nonvirtualized environment. Well, to be specific, in a non-ESXi virtualized environment. It's counterintuitive, I know, particularly if coming from a MSFT background, but for now let's consider them "raw" physical ethernet connections with no protocols bound to them at all. In fact, wholly distinct from Windows virtualization products, ESXi *never* actually assigns IPs to NICs. This actually provides unparalleled functionality in redundancy, teaming, and failover. All communication is handled by an object called a "virtual switch" or vSwitch. The only time an IP is bound to an object is when what is called a "vKernel" port group is configured. Don't let that part throw you – I'll explain it in a bit.

Those three NICs on ESX.1 are part of a single vSwitch which manages all traffic for the VMs as well as connectivity requirements for managing different aspects of the ESX.1 host. Potentially as important, the vSwitch uses those three physical NICs to manage failover or load balancing. All IP traffic which has to do with ESX.1 and its hosts is done through vSwitch.1.1, shown in Diagram 1.3.

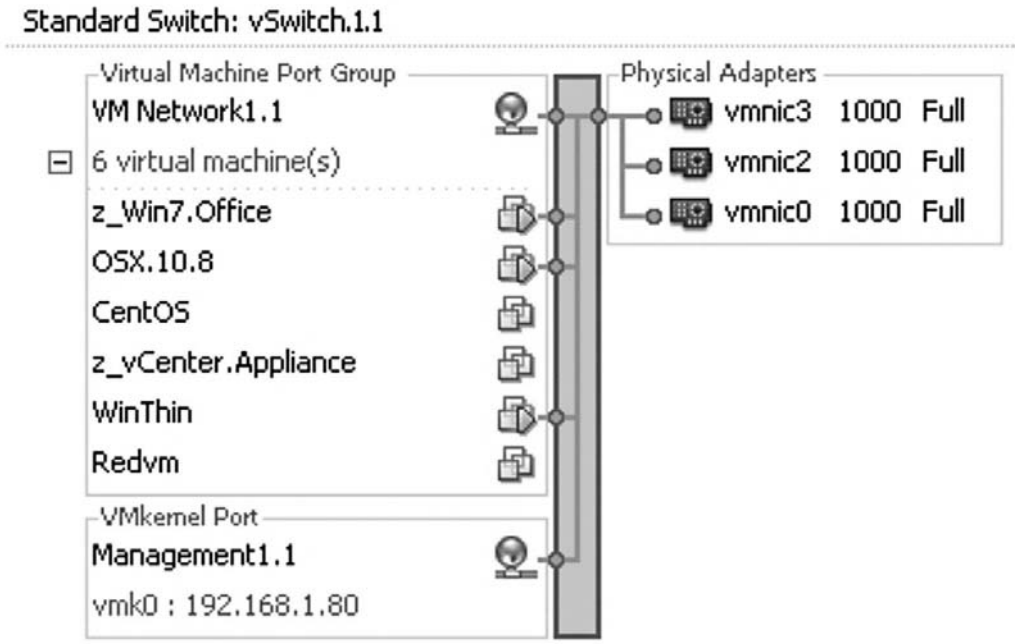


DIAGRAM 1.3

This functionality is provided by out-of-the box ESXi features without dependencies on drivers or "supported NICs" for performance or redundancy functionality. In contrast, Hyper-V is 100% dependent on your NIC manufacturer's 3rd party drivers to function in Windows. Hyper-V is also 100% dependent on the 3rd party driver's binding of the IP address directly to the NIC or team. As such, MSFT will never have these ESXi features because Hyper-V is dependent upon how the base OS works with respect to IPs and binding. Hyper-V simply has no control over it. The Windows development group will never, ever change the way the Windows kernel works in this regard. They can't, technically or otherwise. And this is what will keep Hyper-V out of the "enterprise" market (among many other things) in my ever so humble opinion. So congratulate yourself on already being ahead of the game!

OK – this is how the overall vSwitch works, and it's important for our success that we all understand it. First, Diagram 1.3 shows a preview of what a vSwitch object looks like – we'll get into much more detail in a bit:

When we create a vSwitch, which we will walk through, it is basically a "container" for other networking objects. It is told how many virtual ports exist to support a given number of virtual machines, what MTU we wish to use (typically 1500, but vSwitches support jumbo-frames as well), and it is told which physical NICs are committed to it. However, the switch knows nothing of IP address or other machine traits, other than MAC address in the same way a physical switch wouldn't. Once it knows how many NICs are at its disposal, you can configure it to use those NICs for load balancing, failover, or both at the actual *switch* level, not at the software driver level as Windows requires. Remarkably enough, you can use all three NICs for either feature, or both features *at the same time*! You can even mix and match at will: two NICs for failover, three NICs for performance. So you can literally stuff the host with NICs (not on the MacMini though), and tell the switch all are available for failover and load balancing at the same time! All Windows drivers I've seen only allow one or the other. And I hate having a perfectly good NIC just sitting there doing nothing waiting on the other to fail. But wait! There's more!

In order to use the switch for communication among other hosts, guests, and network objects about your LAN, you must create a group of ports the Virtual Machines will "virtually" plug into called a Virtual Machine Port Group. The vSwitch automatically provisions the needed "ports" from the number available via its configuration.

But check this out – you also do not give the Virtual Machine port group any IP information. That's right: though all the VMs on this port group communicate by merit of their assignment to the port group over TCP/IP, the port group itself doesn't care about IP addresses; this is much in the same way that a group of ports on a physical switch doesn't recognize IP addresses, a

counterintuitive configuration for actual virtual ports. The way the NICs are provided their Observed IP Ranges is simply by tagging traffic on the wire. No muss, no fuss. Again, more on that in a moment.

As with the vSwitch, you can configure individual failover and load balancing options specifically for the VM port group. Meaning as the vSwitch has teaming, so does the Virtual Machine port group using the *same* NICs if desired. That's *major* flexibility.

The other type of port group the vSwitch supports is called a "vKernel" port group, as shown back in Diagram 1.3 we saw earlier. You will obviously need to connect up to the ESXi box to manage it, and that is via the vKernel port group. This *is* where you give the vKernel port group an IP address. The default group is set up during installation, so when you fire up the vSphere client and connect to the IP address of your ESX box, you'll connect to the IP address(s) given to vKernel ports.

The vKernel port group is also where one may configure advanced options within ESXi such as direct iSCSI support, a service called "vMotion" which moves VMs about hosts for you based on utilization and availability, and even other file system support such as NFS.

So to create our DMZ segment, we'll make a vSwitch, create a Virtual Machine port group and a vKernel port group on ESX.2 and we'll be good to go!

We'll move on by way of comparison of our persistent ESX.1 configuration and our ESX.2 final configuration, which looks like Diagram 1.4:

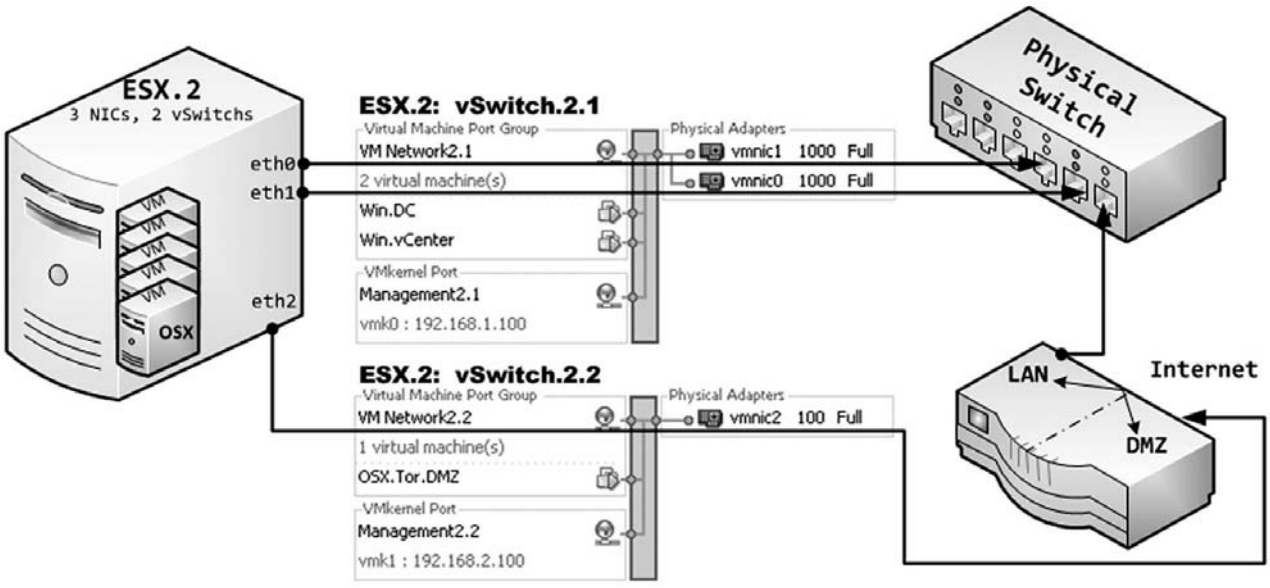


DIAGRAM 1.4

Looking at Diagram 1.2 again, we'll see ESX.1 has three NICs. The physical NICs are represented as *eth0*, *eth1*, and *eth2* though ESXi grabs them and names them *vmnic0*, *vmnic2*, and *vmnic3* as seen in Diagram 1.3. What Diagram 1.3 is actually meant to show is an "out of the box" configuration with the NICs assigned to ESX.1's vSwitch.1.1. The "observed IP ranges" shows the IP of my personal, custom built OS X rig.

To reiterate, I never had to install a single driver – ESXi bound to them natively just by plugging them in. Oh – now is probably a good time to tell you the virtual switch names are *automatically created* with very boring appellations such as vSwitch0 and vSwitch1. I explicitly renamed mine as you see in the illustrations. I like my nomenclature better as the name "vSwitch.1.1" tells me it's the first switch on the ESX.1 server; vSwitch.2.2 tells me it's the second switch on the ESX.2 host. I use this convention with "Virtual Machine" and "vKernel" port group names as well. Unfortunately, vSwitches can't be renamed in the GUI so you'll have to directly SSH into the ESXi host and edit the "/etc/vmware/esx.conf" file with "vi." I'd love to get into that, but it's way outside scope and my editor will yell at me. It's easily found on the Internet, but email me if you can't find it. So as you follow along at home, please keep this in mind.

All three NICs are assigned to vSwitch1.1. The three NICs are plugged into the *hardware* switch, and the vSwitch instance has a "Virtual Machine" port group named "VM Network1.1," and a vKernel port group named "Management1.1." As you see (still referencing Diagram 1.3) Management 1.1 has been given the IP address of 192.168.1.80; this is the IP we use to manage the host itself. Again, this IP has nothing to do with the IP ranges our VM switch observes – it's just to manage the ESXi box itself.

The "VM Network1.1" port group is where the VMs are assigned. Again, the port group itself doesn't have an IP address. I call the vKernel port "Management1.1" because it's actually how I connect to the ESXi machine to *manage* it. That's what that port group is for – to manage the host and configured vMotion, iSCSI, etc. I know I repeat this a good bit, but this was a bit chewy for me to get a grip on initially, so I thought I would explain it to you the way I had to explain it to myself – because repetition is the key to learning! Repetition is the key to learning!

With that background we'll start getting into the meat of things. Later we'll concentrate on the logic represented in the ESX.2 portion of the diagram, which is what you've been waiting for, but first we'll engage in a top-to-bottom overview with what we know so far.

Having a "flat" network doesn't prevent us from building a Tor OS X VM and making it externally available as a relay. We can always port-forward from our router to the 192.168.1.0/24 network, and we'd be done by now. However, we don't know how secure Tor is from an engineering standpoint. We have no idea under what circumstances someone might be able to breach Tor and gain access to our server or network. The same is true of Exchange, IIS, SQL Server, SharePoint, etc. etc. But we'd like to think as business-level applications they have been deeply and extensively scrutinized by many more eyes than an app like Tor. Raise your hand if you laughed.

We also don't know what sort of traffic could be routed through our relay or what manner of twisted and depraved perversions Windows Security MVPs may be engaged in while using our relay.

As such, we're going to take what we already have and, with some reconfiguration, create an isolated network segment earlier called a "DMZ" where compromise of a machine will be strictly limited in its scope of attack options to only those machines contained within that segment. Unfortunately, many have used the term DMZ to describe a network segment where sacrificial lambs and the organization's network scourge are sent to die a slow and lonely death within a maelstrom of filthy internet packets. One shudders.

Unfortunately for those businesses, a DMZ is actually the opposite. We won't dump whatever servers we want willy-nilly without a care for security thinking we've somehow protected ourselves by just putting them there. Rather, our DMZ will be forged with intention – a vision of a pristine, gleaming network paradise where every bit of traffic can be accounted for. Where no packet will be left behind!

The DMZ is where sensitive servers, generally those accepting some manner of anonymous traffic (such as web and mail) are located so that a breach of said machine – given its exposure to external and unqualified traffic – will not afford the attacker a launching pad from which attacks against the internal network can be easily launched. DMZ assets should be on a separate segment and IP space so other targets are not immediately made available in such circumstances.

This is why the DMZ should be an environment of specifically located services where any traffic can be immediately verified as valid. Internal networks are a veritable sandstorm of traffic where one packet cannot effectively be identified from another. While some folks think Intrusion Detection Systems (IDS) have some value within an internal network, I rate them at a zero insofar as use and value are concerned. That's why we will have a shiny DMZ so

that we will always know if something sketchy is afoot and bad traffic will stand out like an Emu at the symphony.

There are several ways to deploy a DMZ, from routers with multiple interfaces to virtual deployments, to firewalls, to barbed-wire ethernet cables. For simplicity's stake, we'll be moving forward with the assumption our "cable-modem" or router offers this functionality. I've not seen a recent router without this feature. That part's easy enough – so let's assume you've set up your cable modem for a DMZ port. With that said, I certainly don't want to leave others in the cold if you don't have the functionality to set up a DMZ port – there will be another configuration for you to use your normal router and ESXi to make your DMZ.

With my network, the DMZ segment will have the 192.168.*2*.0/24 IP address space (remembering our internal network is 192.168.*1*.0/24). Now, it's time for you to do some work. Again, assuming you started with the flat network configuration with your base ESXi installation (which was hopefully easy), the ESX.2 diagram is where it starts for you. Oh, if you are indeed having trouble with ESXi, there are many references on the Internet. However, seeing as how VMware documentation can be a bit fragmented, if you really need assistance just fire off an email to me and I'll see if I can help. I'll do my best to respond in a valuable way.

To move forward, you are going to need at least two NICs in your ESXi box. In fact, as we move forward consider the two ESX.2 vSwitch.2.1 adapters to be one as they will be configured for load balancing that switch. If you are using a MacMini for your ESXi box, then you can buy little USB Ethernet adapters or the Thunderbolt Ethernet adapters; they'll work perfectly. An external USB/Thunderbolt will be the one we use for the DMZ as it will be plenty fast enough as we utilize the gigabit interface(s) for internal traffic.

I guess I should say you can also build your own ESXi box with commodity hardware and *still run OS X VMs*, but that would require me telling you secrets you can already get on the internet. It would also violate your Apple license agreement. And none of us would ever consider doing that, right? Hey, I don't judge, I just make sure you know what technical options you have.

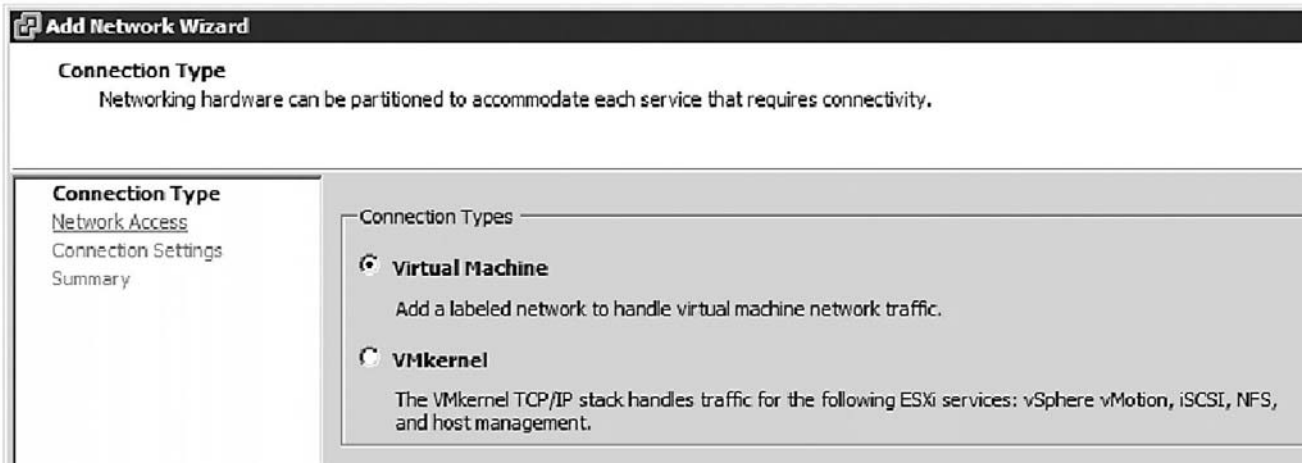
A good way to look at this is that both ESX.1 and ESX.2 were identically configured before these changes. So feel free to use ESX.1 as a base reference as its configuration will not change.

At this point, all three NICs in ESX.2 are physically plugged into the physical switch. Now is probably a good time to go back and study the diagrams

provided thus far. We'll leave it this way until we complete the first step, which is to change our virtual switch configuration. I prefer to leave the NIC we're about to seize plugged into the internal network while this happens, just so any misconfiguration won't lead to a problem within the DMZ.

Leaving ESX.2 vSwitch.2.1 alone, we are going to create a new virtual switch and call it, oh, I don't know, vSwitch.2.2! How original! The purpose of vSwitch.2.2 is to configure the requirements for our *physically* different DMZ network segment. This will give us ESX.2 vSwitch.2.1 serving our main 192.168.1.0/24 network and our vSwitch.2.2 switch serving resources in the DMZ. For now, we'll only have our OS X Tor machine in the DMZ.

This is done by using the Add Network Wizard as described shortly.



Now is where I tell you how all this work you are doing is going to pay off "x" fold in that you'll be able to spin-up virtual instances in the DMZ in a matter of minutes when you want to locate a server isolated from the internal network. The DMZ is a perfect place for you to put your new Apache web server as well by way of OS X Server. Of course, you'll have to save up for that as it costs a whopping $19.99. But it is still just a *tad* cheaper than the thousands you'll pay for a Windows Server license. Bonus!

It may be counterintuitive, but when creating a switch, the first object required is a network access port group. When that is selected, you can then get on with the rest of the process.

The "Add Networking" function brings up the previous dialog box. I'll move on and create a "Virtual Machine" port group, but you just as easily could have created a vKernel group. As previously mentioned, ESXi will now prompt me to either create a new unconfigured vSwitch or bring this network object to a different, already configured *vSwitch* as distinct from an already existing NIC. You can't do that here – you select an entire switch, or create a new one.

And now the new vSwitch configuration where we choose "create" at which point we'll go back and steal the NIC we need for it. The Add Network wizard displays this:

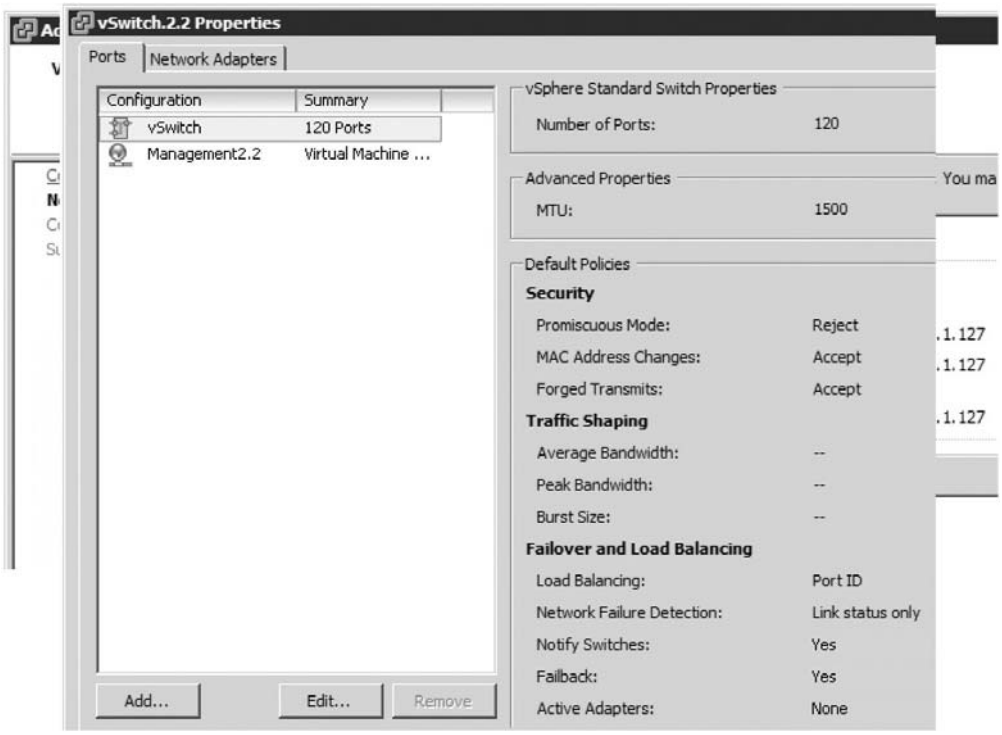
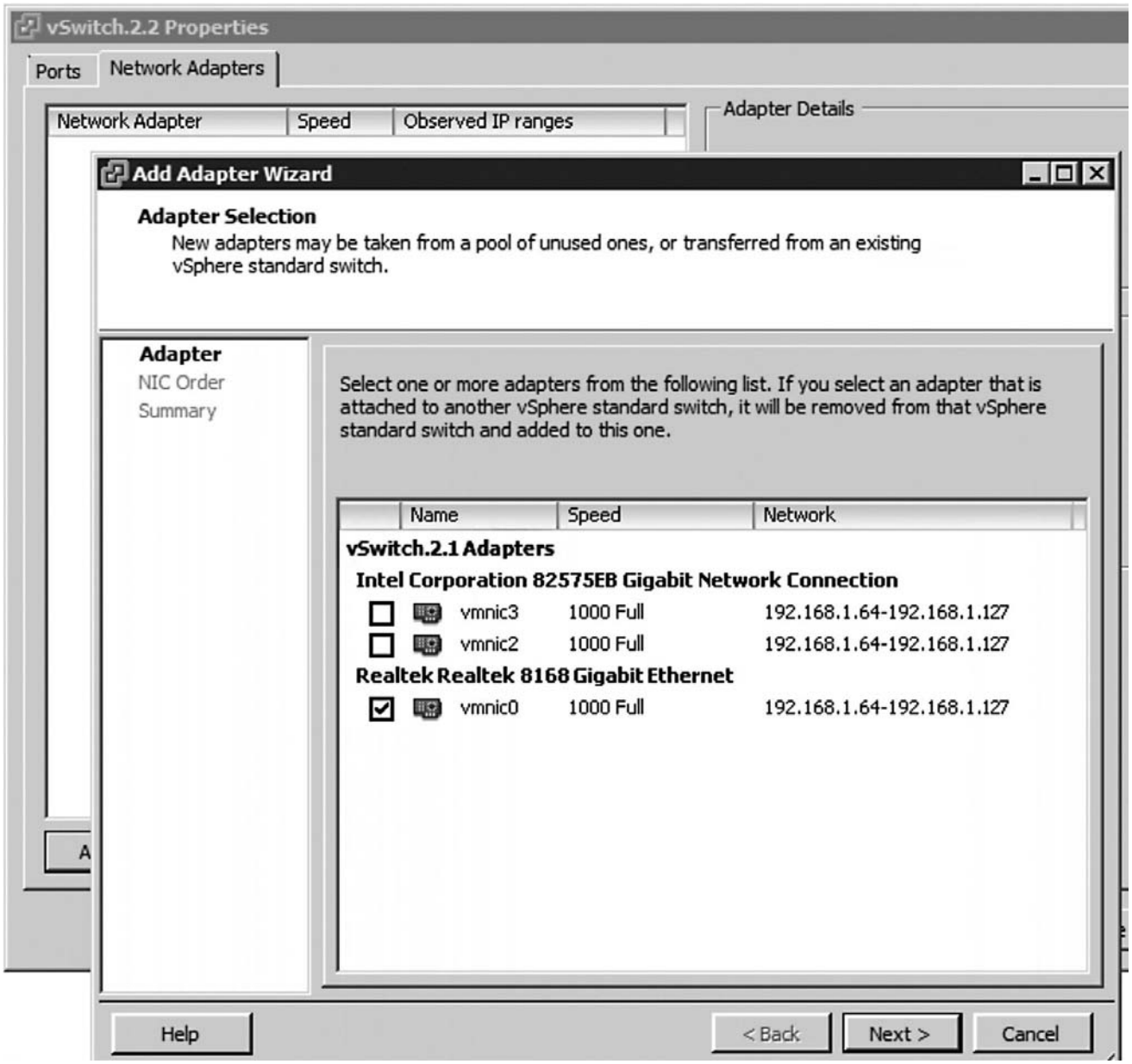


DIAGRAM 1.5

Both "Virtual Machine" and "vKernel" port group names follow the nomenclature of the ESXi hosts and vSwitches. All "vKernel" port groups on all ESXi hosts are named Management1.1, Management1.2, 2.1, and now 2.2.

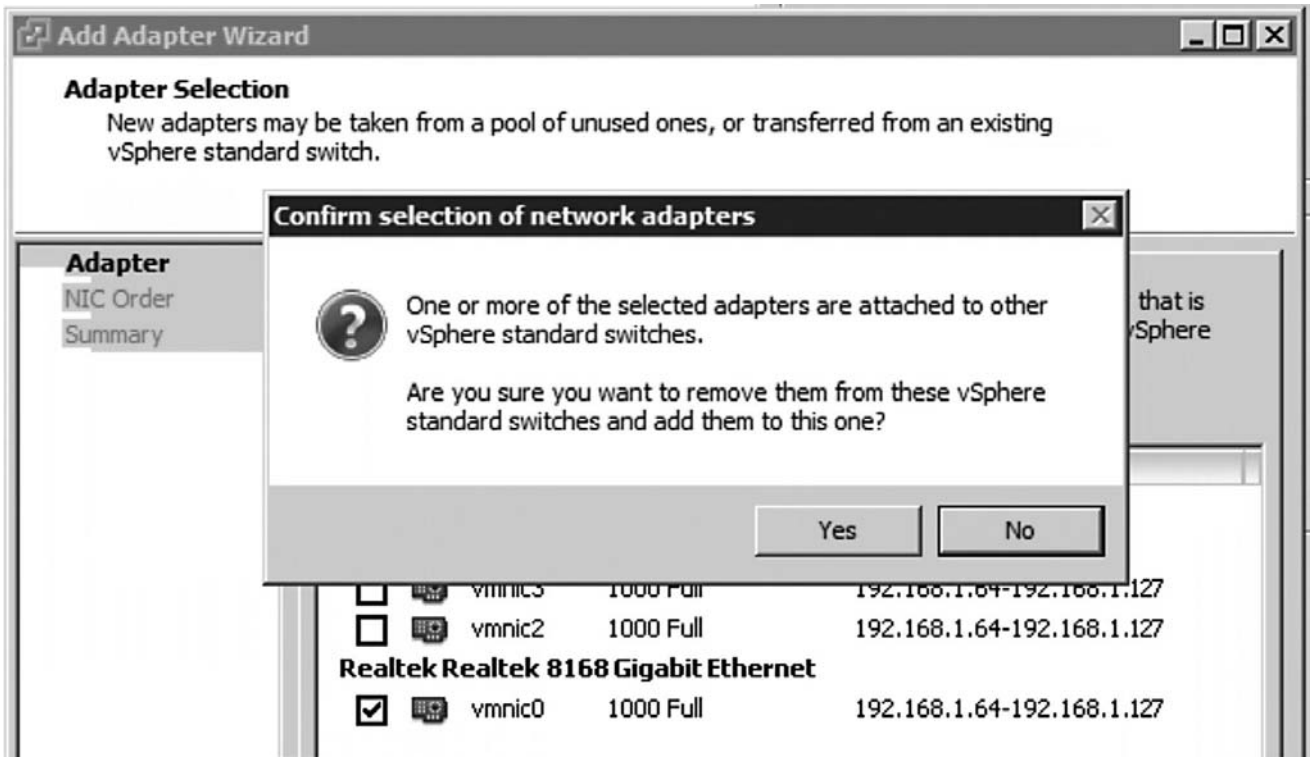


It seems a bit odd to me that the "vSwitch" configuration comes in-between the port group-type and the port group-settings, but, hey, what can you do? So you'll notice in the preview you're about to create a Virtual Machine Port Group assigned to nothing, as we've not seized an adapter yet.

I left the default number of ports to 120, but I went back and changed it to 24 because that's a bunch of unused, lonely ports to worry about. And now, for the moment you've been waiting for, we're going to steal a NIC in broad daylight!

Pull up the new (what is probably) vSwitch0 properties. As I said, the illustration shows my vSwitch after I renamed it, as do the subsequent diagrams. Select the "Network Adapters" tab, and then click "Add." This is where you can select individual NICs as opposed to vSwitches at the beginning of the Add Networking Wizard.

Though it really doesn't matter which card I choose, I selected the built-in Realtek NIC for the DMZ virtual switch vSwitch.2.2.



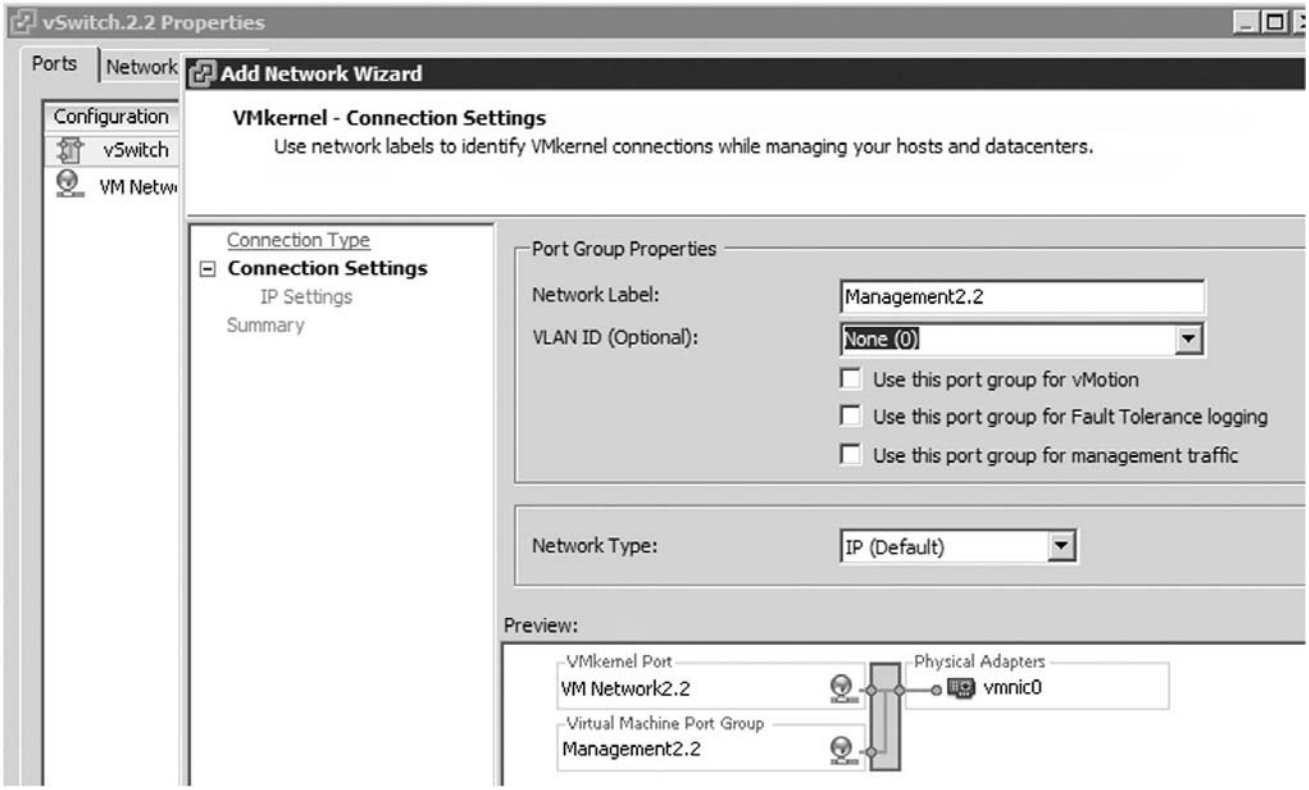
There's no particular reason to select the "odd" one as opposed any either of the other two from the dual-NIC, but after years of working with Windows trying to have some manner of decent fail-over for NICs, I'm used to binding multi-port NIC interfaces together and keeping them that way. Only the same-vendor, same-model, same-card config worked with any semblance of planned design. So I'm just going to keep the dual-port ethernet NICs together, so they stay happy. Besides, if I stole one of those guys, the other one would pine and cry, whining with Broadcast Packets, so it's best to keep the LAN quiet.

Pursuant to that point, here's the confirmation request asking if we are willing to break up the happy home of NICs currently gathered up unto ESX.2 vSwitch.2.1. Are we willing? Oh yeah, we're willing.

Let's do a quick review, and make sure we're all caught up to this point. ESX.2 used to have a single vSwitch consisting of a "Virtual Machine" port group and a "vKernel" port group. In order to reduce confusion between the Virtual Machine port group and the vKernel port group, I named all vKernel port groups "Management" with the appropriate suffix to represent the server and vSwitch number. The vKernel port group used to live in the 192.168.1.0/24 network. Three of three NICs were bound to this virtual switch, all physically connected to our main internal switch. Everything worked, and life was good. Now, in order to create the DMZ segment, we've created a new vSwitch on ESX.2 called vSwitch.2.2. We created a "Virtual Network" port group called "DMZ," and took the Realtek NIC away from vSwitch.2.1 and bound it to vSwitch.2.2 instead. We're almost through with this part of the work!

It's rather important you are up to speed, as the next section might be a bit confusing. So go ahead and review on your own. I'll wait.

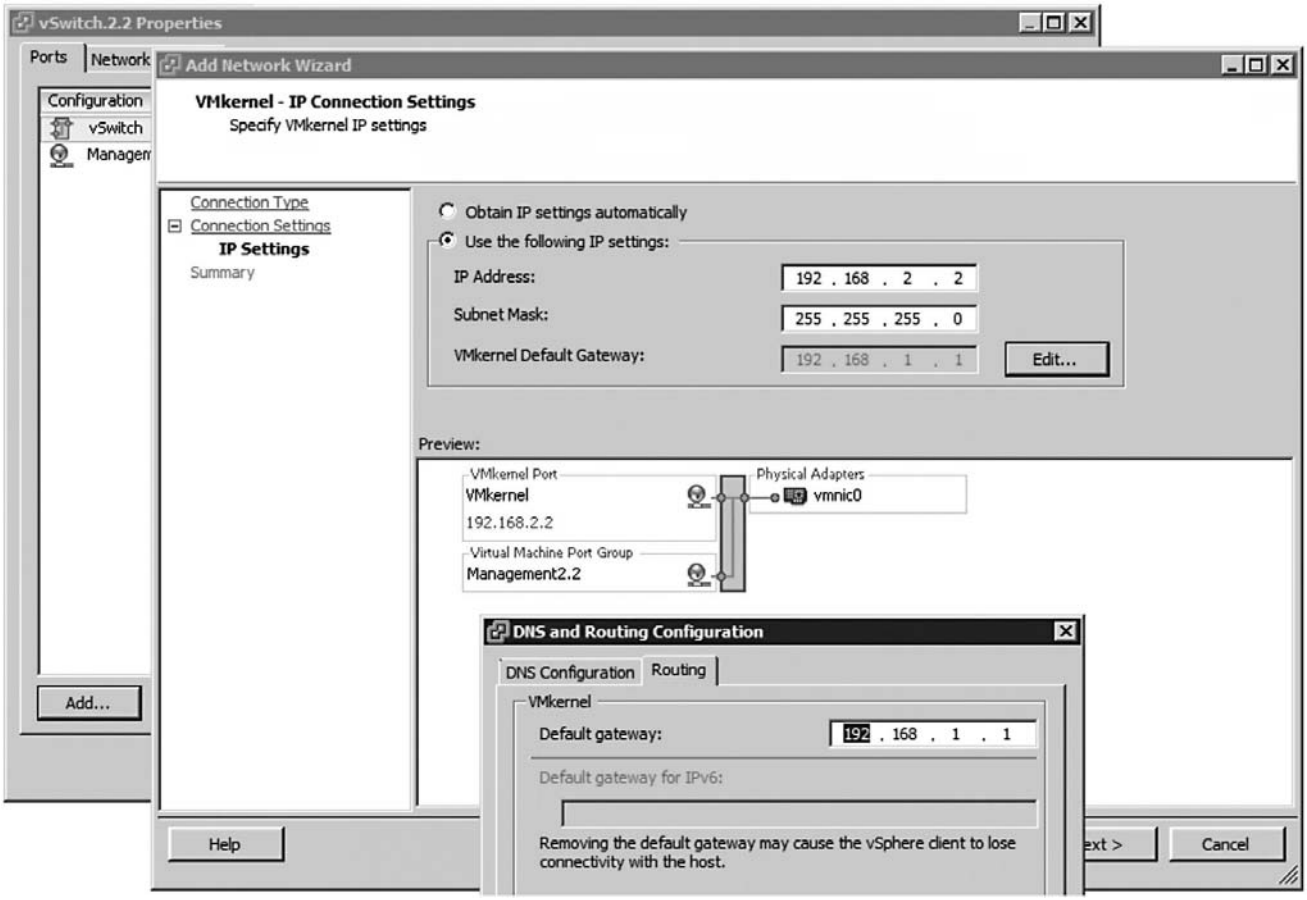
Oh good! You're back!



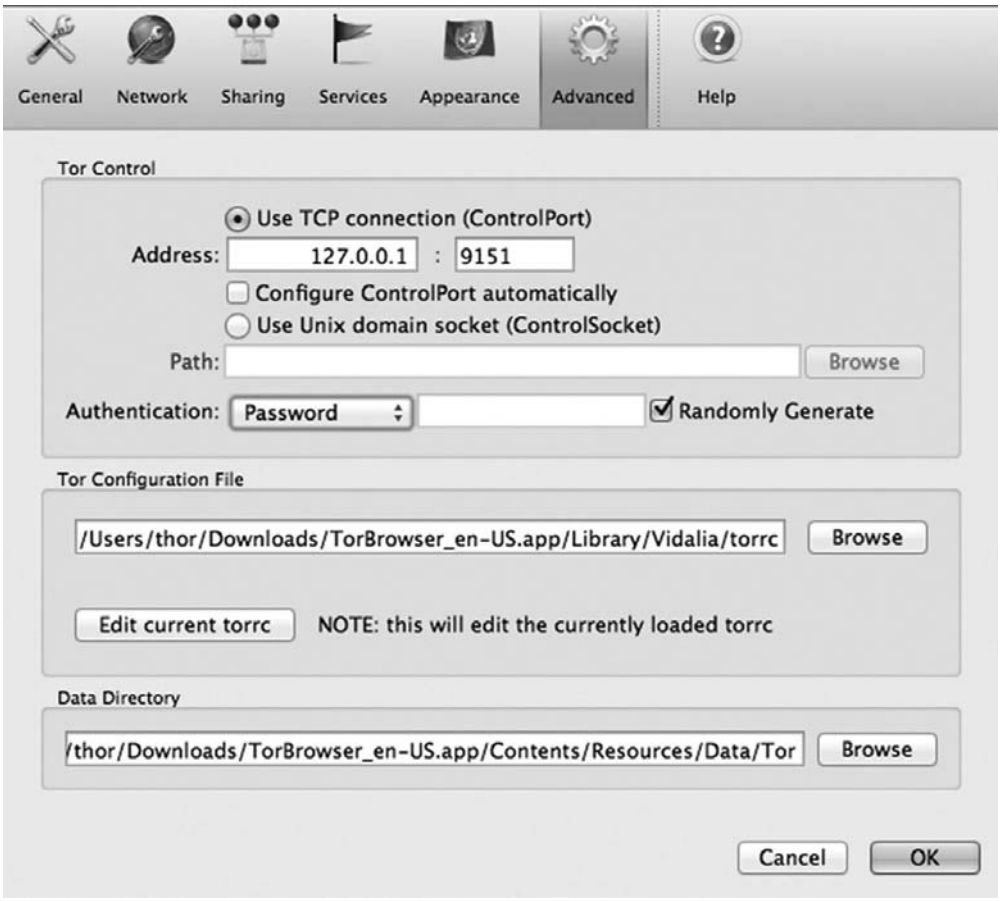
On the vSwitch2.2 I created both a Virtual Machine port group and a vKernel port group. Creating vKernel network for the DMZ is not required, and best practices would dictate you don't create one as there should be no need to perform actual management functions on ESX.2 from the DMZ. That should be done on the more secure, isolated internal network segment.

However, in my case I have a need for it (given the number of VMs I have in the DMZ) so I'll go ahead and create a Management group without actually binding management protocols to it. In this way, I can bind protocols as needed without having to interrupt operations by reconfiguring a switch while my VMs are running.

This basically creates a vKernel port group that doesn't really do anything insofar as any advanced ESXi protocols or services are concerned. Again, this is on purpose – this is our DMZ segment, and if any DMZ assets are compromised, various and sundry traffic and services could be leveraged to strengthen an attack. Given how rich management protocols are on ESXi, I'm disabling them to minimize my exposure.



What the vKernel port group *does* do is to provide me with TCP/IP access to ESXi host-level functionality such as SSH and SCP. So even though I'm leaving my switch a "shell" as far as ESXi is concerned, I still retain the functionality I may need when I'm in a pinch. So, with Management.2.2 created, we'll assign it the IP address we'll use for SSH, SCP or whatever BSD-like TCP/IP services we wish to use.



Note the inclusion of a default gateway of 192.168.1.1. This is important as it provides the route needed from the 192.168.1.0/24 network over to the 192.168.2.0/24 network. Without this, we wouldn't be able to connect to our Tor proxy installed on the OS X VM assigned the 192.168.2.50 address.

Whew! We're finished with the ESXi configuration and have created our DMZ segment! And now we are finally ready to deploy VM instances in the DMZ!!

If you feel like you are going to miss out on the Tor Proxy configuration, fear not. If you don't feel like going through the hassle or don't have any need for a DMZ, you can still follow these instructions from here on out to configure a Tor proxy even if you keep it on your internal network. I wouldn't leave you out in the cold like that!

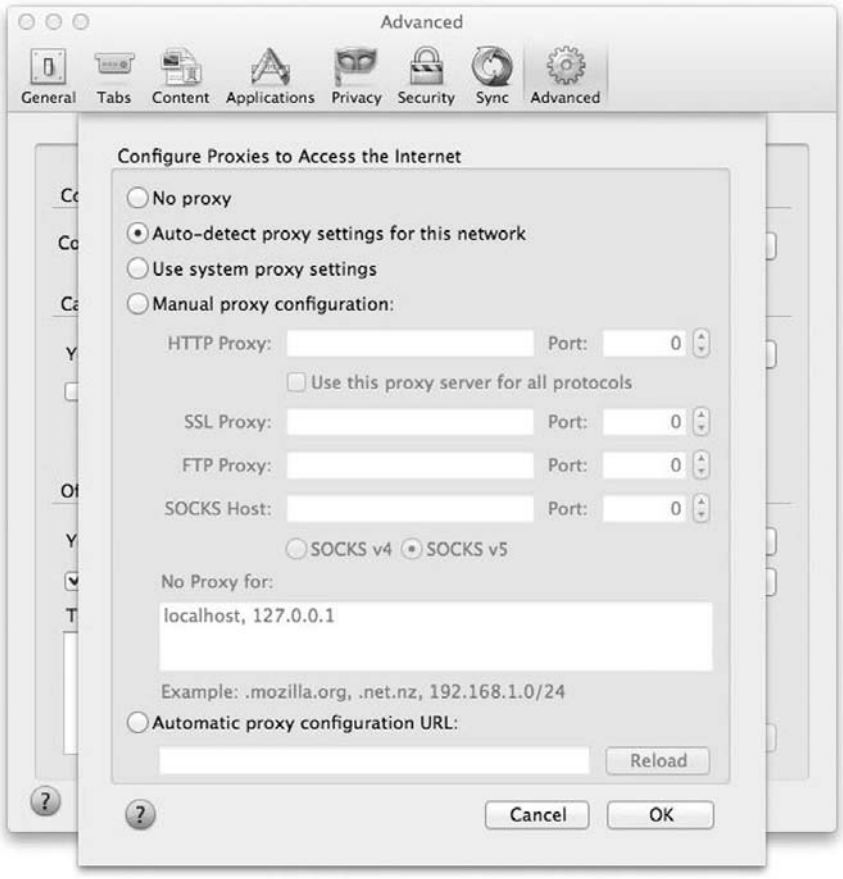
Continuing with the presumption you've got your OS X installation for the Tor Proxy configured with the default Tor installation in place, we'll go through the following steps to provide the proxy functionality we've been working towards.

Remember, in this model we are only installing the Tor client on the OS X box we want to use as the proxy – there is no need to install Tor on all your clients. It's actually quite simple from here. Launch Tor, and select Settings and then the Advanced icon.

```
# This file was generated by Tor; if you edit it, comments will not be preserved
# The old torrc file was renamed to torrc.orig.1 or similar, and Tor will ignore it

ContactInfo tor at hammerofgod dot com
ControlPort 9051
DataDirectory /Users/thor/.tor
DirPort 9030
DirReqStatistics 0
ExitPolicy reject *:*
HashedControlPassword 16:382D366E6E02F15160D172691B67F12D34AFCD3E04BEFF8A62BBE73DBF
Log notice stdout
Nickname HammerofGod
ORPort 9001
RelayBandwidthBurst 10485760
RelayBandwidthRate 5242880
SocksListenAddress 192.168.2.50
SocksPort 9050
```

This is the default configuration with the Control Port being set at port 9151 on IP 127.0.0.1 (the loopback). Note the "Control Port" is not the same as the port we need to set as the "proxy port" in our browser profile. More specifically, this is called the "SocksPort" and has a default value of 9050. You don't see options for the Socks Port in the dialog box, but you will if you select the Edit Current torrc button. Note, your configuration will look different insofar as the paths to the configuration file and data directory are different, which is based on your installation. The full default configuration is found in the torrc file which looks like this:



You may not have all the elements shown, but the ones you will have should be at least:

```
ControlPort 9051
DataDirectory [Your data directory]
DirReqStatistics 0
HashedControlPassword [Your Password Hash]
Log notice stdout
SocksPort 9050
```

We will leave the `ControlPort` and `SocksPort` values alone, but we do need to add a new value called SocksListenAddress.

You'll see additional configuration options and values in my configuration which may or may not be present in your configuration file, again, based on which type (if any) actual Tor relay-type options you've selected. The important key/value pair for our purposes is the line:

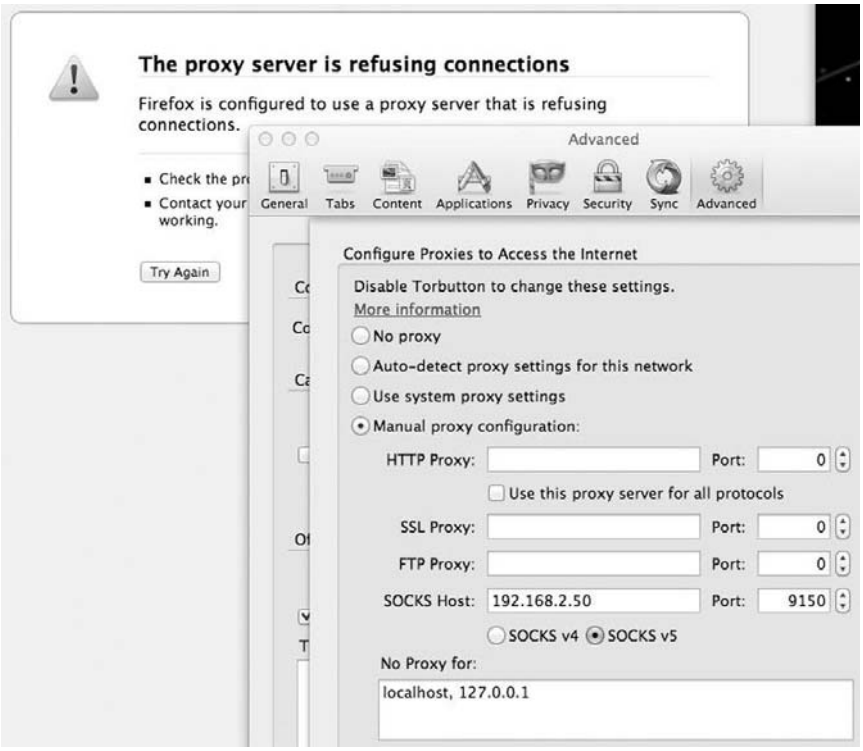
```
SocksListenAddress 192.168.2.50
```

If you followed the previous ESXi configuration instructions to the letter, your OS X Tor Proxy IP will be 192.168.2.50. If not, then you will replace this value with the IP address of your OS X's TCP/IP network address. Without the addition of this key/value, which you will have to type in yourself in the torrc editor window, the default listen port will be 127.0.0.1 even though you don't actually see it explicitly specified.

Save this configuration, exit Tor and relaunch Tor (just to be sure your changes are applied). If Tor doesn't automatically bring up the Start Tor process, just click Start Tor yourself. And we're done as far as the server proxy service is concerned!

All we have to do now is change the SOCKS port we want to proxy connections to in Firefox (or whatever browser you wish). This is where multiple profiles will make your life much simpler. I have a specific profile on my client explicitly configured to use my OS X Tor Proxy. That way I just select my Tor profile in Firefox when I want my connections to be secure without having to muck about with changing configurations and such.

Launching Firefox with a new profile or an existing profile you wish to use, navigate to your Firefox properties, select the Advanced icon, then select the Network tab, and finally click the Settings button under the Connection heading next to the "Configure how Firefox connects to the Internet" text. The default dialog is as follows:



These default settings basically mean that Firefox will automatically detect if you have a proxy server on your network, and if not, to just directly connect to the Internet via your current TCP/IP configuration settings. For our Tor profile, we want to select a Manual proxy configuration and set our SOCKS Host to 192.168.2.50 on port 9050. In this way, when we launch this profile, Firefox will automatically route all outbound SOCKS traffic to our DMZ OS X system. The way your internal address in the 192.168.1.0/24 network will be able to reach the DMZ segment of 192.168.2.0/24 is by way of the default gateway we set for the IP of our Management2.2 vKernel group on vSwitch2.2.

The reason we are selecting the SOCKS protocol (which stands for Sockets Secure) as opposed to what you may normally be familiar with as a HTTP Proxy is because the Tor proxy does not accept actual HTTP Proxy requests – it accepts TCP/IP requests. What this really means for you is that you can actually configure other protocols and applications to use Tor such as mail, remote protocols such as ARD, RDP, SSH, Telnet, etc. or any other IP-based applications. SOCKS also supports UDP forwarding as well if you wish for DNS lookups or the like to also have the source protected by the Tor network! That's way cool, and it's a simple change.

So, to recap our Advanced section, we've learned how to set up an ESXi virtual machine host, configure vSwitches and port groups to create a DMZ segment, populate that DMZ segment with an OS X virtual machine, and configure that VM to act as a proxy for Tor SOCKS connections out to the internet. Finally, we leveraged custom proxy settings in Firefox to make the process of allowing clients on your LAN to easily protect their privacy. I think we've all earned a few hours off to play God of War!