

Assessing Internet Security Risk

(as published on Security Focus as a 5 part series – October 2002
by Charl van der Walt @ SensePost)

the introductory stuff

This paper is all about risk; what it is, what it means to us, how we measure it and how we manage it. But mainly, it's about the new Internet world and how we determine what the risks are to the Internet-connected systems that make our businesses function.

The Internet, like the wild west of old, is an un-chartered new world, full of fresh and exciting opportunities. However, like the wild west, the Internet is also fraught with new threats and obstacles; dangers the average businessman hasn't even begun to understand. But I don't have to tell you this. You've heard that exact speech at just about every single security conference or seminar you've ever attended, usually accompanied by a veritable array of slides and graphs demonstrating exactly how serious the threat is and how many millions of dollars your company stands to lose. The "death toll" statistic are then almost always followed by a sales pitch for some or other product that's supposed to make it all go away. Yeah right.

Am I saying the threat isn't real? Am I saying the statistics aren't true? No. What I'm saying is that I fail to see what relevance any of this has to me and my company. Should the fact that e-Bay supposedly spend \$ 120,000 dollars recovering from *Mafia Boy's* DDoS attack really have an impact on my corporate IT policy? I think not.

And yet I can't afford to ignore these facts completely. That would be just plain dumb. What I need to do is to recognize that there are new threats and challenges and, like the other threats and challenges that businesses have always known, these need to be met and managed. No need to panic. No need to spend any money. Yet.

The next thing I need to do is to understand what the specific risks are that my company faces from being connected to the Internet. In the same way that you don't borrow your business strategy from e-Bay, you probably shouldn't borrow your IT security strategy from them either. You need to understand your company's own unique risk profile.

As with so many other things in life, the key to effective information security is to work smart, not hard. And in this case, working hard means investing your valuable time, money and human resources on addressing the specific problems that are the most likely to cause the most damage. The math is really quite simple. But before you can do the sums, you have to populate the variables. Here are some of the questions you'll have to ask yourself:

1. What are the resources I'm actually interested in protecting? (Information & Information Systems).
2. What is the value of those resources? (\$ or relative)
3. What are the all the possible threats that those resources face?
4. What is the likelihood of those threats being realized?
5. What would be the impact of those threats on my business, if they were realized?

Having answered the five questions above, you can then investigate mechanisms (both technical and procedural) that might address those risks, and then weigh up the cost of each possible solution against the potential impact of the threat. Once again, the math is simple: If the cost of the solution is higher than the potential financial impact of the risk (or risks) being addressed,

then one may need to investigate other solutions, consider accepting and living with a part of the risk, or accepting and living with the risk completely.

This paper concentrates on helping you to answer questions 3 & 4 in the context of Internet-connected systems: What are the threats that my Internet-connected systems face and what are the chances of those threats being realized. Over the next few weeks we will explore the thinking around Internet Security Assessments, not only *why* they are done, but also *how* they are done. By the end of this series you should understand how performing an Internet Security Assessment can contribute an effective information security strategy, what you should expect from such an assessment and even how you could go about performing such an assessment yourself.

About Me

My name is Charl van der Walt. I work for a South African company called *SensePost*. We specialize internationally in the provision of information security services, including assessments of the kind I am describing here. My background is in Computer Science, I am a qualified BS7799 "Lead Auditor" and I've been doing this kind of work for about five years now. I have a dog called Fish.

It's OK to get Hacked – the reasoning behind security assessments

Background

An Internet Security Assessment is about understanding the risks that your company faces from being connected to the Internet. As already discussed, we go through this exercise in order to effectively decide how to spend time, money and human resources on information security. In this way our security expenditure can be *requirement driven*, not *technology driven*. In other words, we implement controls because we know that they're needed, not just because the technology is available. Some firms refer to security assessments as *Ethical Hacking* or *Penetration Testing*. Although I also use this terminology, I see it as something completely different and do not see its use as appropriate in this context.

Security Assessments vs Risk Analysis

Later in this paper I'll show you a diagram of what's known as the "security life cycle" – a depiction of the concept that security is a continual cycle with a number of distinct phases being repeated on an ongoing basis. You'll notice that this cycle distinguishes between a risk analysis and a security assessment. You may even have come across both terms before and wondered at the distinction. It's not my intention to argue semantics here. Indeed, I'm not even convinced that there is universal consensus on the precise definition of each term. Here's how I see it, briefly: A risk analysis is typically performed early in the security cycle. It's a business-oriented process that views risk and threats from a *financial* perspective and helps you to determine the best security strategy. Security assessments are performed periodically throughout the cycle. They view risk from a *technical* perspective and help to measure the *efficacy* of your security strategy. The primary focus of this paper is on this kind of assessment.

Internal vs External Assessments

I have further limited this paper to a discussion of *Internet Security Assessments*. Let me point out right from the start that this is only a part of the picture. Our company distinguishes between *Internal Assessments* and *External Assessments* in the following way:

An external assessment is also known as perimeter testing and can be loosely defined as testing that is launched from outside the perimeter of the private network. This kind of testing emulates the threat from hackers and other external parties and is often concerned with breaching firewalls and other forms of perimeter security.

With internal testing the analyst is located somewhere within the perimeter of the private network and emulates the threat experienced from internal staff, consultants, or disgruntled employees, or in the event of unauthorized physical access or a compromise of the perimeter security. These kinds of threats comprise more than 60% of the total threat portfolio.

Although an Internet assessment is attractive because it is finite and answers a direct question, the following should be noted at the outset:

1. An Internet assessment will not identify all the risks to your information resources. Areas that are clearly not addressed include the following:
 - a. Threats from within the trusted environment
 - b. Threats from RAS and other external connections
 - c. Threats from your extranet and connections to 3rd parties
2. There are other ways of assessing risk, without doing a technical assessment.

Although it's beyond the scope of this discussion, the scope of an Internet Assessment can easily be expanded to include areas like RAS and the Extranet (which is why we actually refer to the service as an *external* assessment). However, even with the limited scope, there are a number of strong reasons for performing an Internet Security Assessment.

But first, let's remind ourselves why we want to do an assessment in the first place.

Reasons for performing a Technical Security Assessment

I've often thought, at the end of a security assessment project, that I probably could have advised the customer without having to perform the entire analysis. Internet installations are generally all very similar and one sees the same mistakes being made at different installations all over the world. And yet I haven't quite given up on the idea. There are a number of reasons for my continued faith in technical assessments.

Firstly, a technical assessment allows me to fully familiarize myself with the customer's architecture. By the time the assessment is finished, I usually understand their Internet architecture at least as well they do, often even better. This puts me in a unique position to offer then real and useful advice and ongoing technical support.

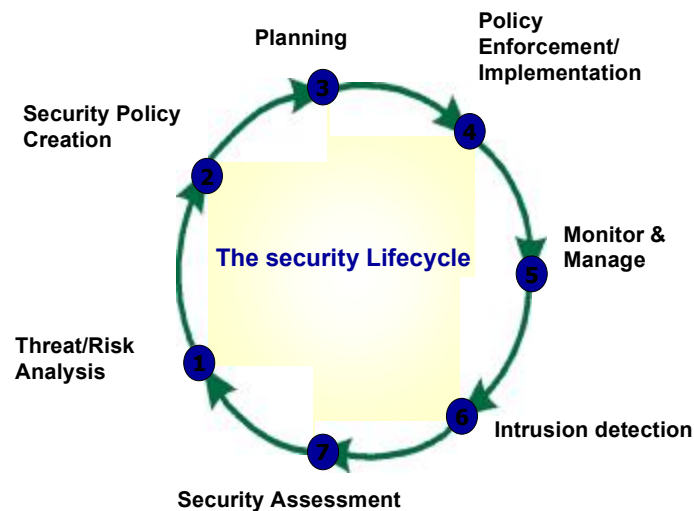
The technical familiarity I've acquired also very often buys me the respect of the customer's technical personnel. That, in turn, puts me in an even better position to advise them. Because our clients themselves are often non-technical people – risk managers and financial managers – it is essential that we also win the trust and respect of the technical team. *Penetration Testing*, a later phase in the assessment methodology during which we actually attempt to breach security and compromise the customer's systems, is particularly effective in this regard. It's hard for someone to argue that their security is sufficient when you've already clearly demonstrated that it can be compromised. The fact that our findings are based on a formal assessment methodology lends weight to the recommendations we make.

Sometimes, an objective assessment from an independent third party is necessary to convince others that you're taking security seriously. This is becoming more of an issue in certain sectors, where government, shareholders and other regulatory authorities are expecting companies to provide proof of proper information security.

Moreover, the fact is that a properly executed assessment may very well identify problems that otherwise may have gone un-noticed. A single small finger-fault in your firewall configuration may be all that's needed by an attacker and a thorough technical assessment may be the only way of determining this.

But most importantly, an assessment introduces *objectivity*. With the overwhelming number of security products and vendors in the market, it's important that spend money for the right reasons. A good assessment should help you to understand and prioritize your security requirements, allowing you to invest resources effectively. Very often, the most serious requirements will not be addressed by the simple acquisition of more technology, and its important for the customer to understand that.

Actually, this last point is nothing new and security assessments have been seen as an important phase in the security *lifecycle* for as long as there's been information security theory. One version of the lifecycle looks like this:



Notice how the assessment phases (*threat / risk analysis* and *security assessment*) are the first and last step in the process. The analysis is used to identify what needs to be done, and the assessment is used measure how effective the other phases in the cycle have been. A number of companies are even starting to use the outcome of these repeated assessments to measure the performance of their technical personnel. Some companies even use security assessments as a key performance area for regular personnel. Now there's an interesting idea.

Reasons for performing an Internet Security Assessment

Hopefully I've convinced you now of the value of a technical security assessment. But I've also said that this paper is limited to a discussion of Internet security assessments only. Does it make sense to focus on one area of your system like that? Actually, no. But Rome wasn't built in a day, and a complete assessment of a large environment will typically be broken up into a number of distinct and manageable phases. The Internet is only one of a number of different areas we should examine.

However, Internet-connected systems are the single area we assess more than any other. And, given limited time and resources, it's sometimes the only area we consider for clients. Here's a summary of the reasons companies still to perform *Internet Security Assessments*:

- 1. Internet systems are an obvious part of the problem:** Given the almost overwhelming size of the complete information security problem, it's often hard to know

where to start. Internet systems are very often a clearly defined subset of the complete infrastructure and can be easily isolated, analyzed and secured. Although we realize that this only a small part piece in a much larger puzzle, it very certainly is a piece. If we can confirm that the Internet systems are secure many managers feel “Whew, at least *that’s* out of my hair”.

2. **The Internet is different:** The tools and methodologies that we apply in analyzing Internet security are different from those we use when looking at ‘internal’ spaces like WANs, LANs and Extranets. For this reason we tend to see an Internet assessment as a separate body of work from the rest of the assessment and tackle it separately.
3. **Internet systems are an obvious target:** Attack via the Internet is by no means the only threat your company faces, but it is a clear and obvious threat and one would be foolish to ignore it. And, just like you want to be sure you’ve locked your front door, you want to be sure you’ve secured your connections to the Internet. The threat of attack via the Internet is easily identified, tested and eliminated. We test our Internet security because then we can know that it’s been done and move on.
4. **Internet systems are a high-profile target:** It smarts to be hacked from the Internet. Even though the financial impact of such an attack is often smaller than other forms of attack, a defaced web site and other forms of Internet attack can often do huge damage to your company’s reputation. For this reason we want to know that our Internet security has been taken care of.
5. **Internet systems are often beyond our control:** The Internet began its life a utopian exercise in you-scratch-my-back-I’ll-scratch-yours-ism. Although this early utopianism has long since evaporated and the Internet has now developed in a battlefield for new-world commerce, there are still a rather scary number of uncontrolled inter-dependencies that make it possible for your company to operate on the Internet. The magical routing of IP packets from one network to the next is one example of this. The mapping of machine names to IP addresses via the Domain Name System is another. Yet we have no real control over these systems. They are critical to the safe operation of our Internet infrastructure and yet their security is beyond our control. Similarly, we have no control over when new vulnerabilities will be discovered in our Internet technologies. Quite simply, the only defense we have is to regularly assess this infrastructure for safe and secure operation. This is probably more true for the Internet than for other areas of your infrastructure.

Conclusion

In this section I’ve tried to convince you of the value of doing a technical risk assessment and to explain why we often consider the Internet systems separately from the rest of the infrastructure. In the section that follows, I’ll give you an overview of the steps that we follow in performing this kind of assessment. The methodology is designed to ensure that our work is complete and consistent.

Hacking by Numbers – an Internet Assessment Methodology

Why all the fuss about a Methodology?

If you ever read anything SensePost publishes on assessments, or if you attend our training, you’ll notice we tend to go on a bit about *methodology*. The methodology is the set of steps we follow when performing a certain task. We try and work according to methodologies with just about everything we do. We’re not fanatical about it, and the methodologies change and adapt to new and different environments, but they always play a big role in the way we approach our work. Why such a big fuss then? There are a few good reasons for performing assessments according to a strict methodology:

Firstly, it gives us a game plan. Rather than stare blankly at a computer screen or a network diagram, an analyst now has a fixed place to start and a clear task to perform. This takes the

whole 'guru' element out what we do. The exact steps that we will follow are clear, both to us, and to the customer, from the outset.

Secondly, a methodology ensures that our work is consistent and complete. I've worked on projects where the target organization has in excess of 150 registered DNS domains. Can you imagine how many IP addresses that eventually translates to. I don't have to imagine – I know it was almost 2000. Consider how hard it must be to keep track of every DNS domain, every network and every IP to ensure that you don't miss something. Consider also what happens when the actual 'hacking' starts (we'll get to this later) and the analyst's heart is racing. A strict methodology ensures that that we always cover all the bases and that our work is always of the same quality. This really holds true no matter how big all small the environment is that you're assessing.

Finally, our methodology gives our customers something to measure us against. Remember, to date there are really no norms or standards for technical assessment work. How does the customer know that she's getting what she paid for? This is an especially pertinent question when the assessment findings are (how can I put this?) dull. By working strictly according to a sensible methodology with clear deliverables at each stage we can verify the quality of the assessment even when there's very little to report.

A Methodology that Works

I'm completely sure that, when it comes to security assessment, there's more than one way to skin the cat. What follows is a description of a methodology that we like to use when performing security assessments over the Internet. It's certainly not the only way to approach this task, but it's one way that that I believe works.

1. Intelligence Gathering

The first thing we do when we begin an assessment is to try and figure out who the target actually is. We use the web for this mainly. Starting with the customer's own web site(s), we mine for information about the customer that might be helpful to an attacker. Miscellaneous tidbits of useful data aside, our primary objective is to derive the DNS domain names that the target uses. If you're assessing your own infrastructure, you may already have this information but if the organization is big, it can be a fascinating exercise. Later, these domain names will be mapped to the IP addresses we will actually analyze. Some companies have a small Internet presence and discovering the DNS names they use may be simple. Other companies we've worked with have hundreds of domains, and discovering all of them is no mean feat.

How do we get the DNS domain names? Well, usually we have an email address, the company's name or some other logical place to begin. From there we have a number of techniques:

1. We use search engines to search of all instances of the company's name. This not only provides links to the company's own site (from which DNS domain information can be easily derived), we also obtain information about mergers and acquisitions, partnerships and company structure that may be useful.
2. We use a tool like *httrack* to dump all the relevant web sites to disk. We then scan those files to extract all mail and HTTP links, which are then parsed again to extract more DNS domains.
3. Then, we use the various domain registries. Tools like *geektools.com*, *samspace.org*, *register.com* and the like can often be used in one of two ways:
 - a) To help verify whether the domains we have identified actually belong to the organization we are assessing.

- b) To extract any additional information that may be recorded in a specific domain's record. For example, you'll often find that the technical contact for a given domain has provided an email address at a *different* domain. The second domain then automatically falls under the spotlight as a potential part of the assessment.
 - c) Many of the registries provide for wildcard searches. This allows us to search for all domains containing a certain string, like "*abc*". I would use such a search to identify all the domains that may be associated with the company *ABC Apples Inc*, for example.
4. Then, we need to apply some human intelligence – using the information we read on web sites, company reports and news items we attempt to make logical jumps to other domains that may be relevant to our analysis.

The output of this phase is a comprehensive list of DNS domains that are relevant to the target company. You'll notice that the next phase of the assessment may provide additional domain names that weren't found during this phase. In that case, those domains are used as inputs during this phase and the entire process is repeated. Phases 1 and 2 may recur a number of times before we've located all the relevant domains. Typically, we'll check this list with the customer once we're done to ensure we haven't missed anything or included something inappropriate.

2. Foot printing

At the start of phase two we have a list DNS domains – things like *apples.com*, *apples-inc.com*, *applesonline.com*, *apples.co.uk*, etc. The reasons these domains exist is to provide Internet users with a simple way of reaching and using the resources they require. For example, instead of typing *http://196.30.67.5*, a user simply needs to remember *www.sensepost.com*. Within a domain, therefore, there are a number of *records* – specific mappings between machine *names* and their actual Internet Protocol (*IP*) numbers. The objective of this phase is to identify as many of those IP/name mapping as we possibly can in order to understand which address spaces on the Internet are actually being used by the target organization. There are a few different techniques for identifying these mappings. Without going into too much detail, these techniques are all derived from the same assumptions, namely:

1. Some IP/name mapping **must** exist for a domain to be functional. These include the *name server records* (NS) and the *mail exchanger records* (MX). If a company is actually using a domain then you will be able to request these two special entries. Immediately you have one or more actual IP addresses to work with.
2. Some IP/name mappings are very **likely** to exist on an active domain. For example, 'www' is a machine that exists in just about every domain. Names like 'mail', 'firewall' and 'gateway' are also likely candidates. We have a long list of common names that we test. This is by no means a watertight approach but one is more often lucky than not.
3. An organization's machines usually live close together. This means that if we've found one IP address, we have a good idea of where to look for the rest of the addresses.
4. The Name -> IP mapping (the *forward* lookup), and the IP -> Name mapping (the *reverse* lookup) need not necessarily be the same.
5. The technology is fundamentally verbose. DNS, as a technology, was designed for dissemination of what is essentially considered 'public' information. With one or two simple tricks we can usually extract all the information there is to be had. The DNS *zone*

transfer – a feature of DNS literally designed for the bulk transfer of DNS records - is a fine example of this. Other, craftier, techniques fall beyond the scope of this paper.

Once we have all the relevant DNS names we can find, we attempt to identify the distinct network 'blocks' in which the target organization operates. As stated previously, IPs tend to be grouped together. The nature of IP networking is to group addresses together in what are known as *subnets*. The expected output of this phase is a list of all the IP subnets in which the target organization has machines active. At this stage our reasoning is broadly that if we find even a single IP in a given subnet we include that entire subnet in the list. The technically astute among you will already be crying 'False assumption! False assumption!', and you'd be right. But bear with me. At this stage we tend rather to over-estimate than to under-estimate. Later we will do our best to prune the list to a more accurate depiction of what's actually there.

2. Vitality

We ended the last phase with a list of IP subnets in which we believe the target organization to have a presence and a horde of technocrats objecting loudly to our assumptions about the subnet size. Let's quickly make a list of the some of the facts we need to know before we can move on with the process:

1. An organization does not need to *own* the entire subnet in which it operates. IP addresses can be lent, leased or shared. Neither do all an organization's IPs *have* to be grouped together, they can be as widely spread across the Internet as they wish.
2. Just because a Name / IP mapping exists for a machine, doesn't mean that machine actually exists. Conversely, just because a Name / IP mapping *doesn't* exist for a machine, doesn't mean the machine *doesn't* exist. There are thousands of nameless addresses on the Internet. Yes, its sad, but true nevertheless.

So we see that, although DNS gives us a logical starting point for our search, it by no means provides a comprehensive list of potential targets. This is why we work with the rather loose subnet definitions we derived in the previous phase. The objective of the 'Vitality' phase of the assessment is to determine, within the subnet blocks that we have, what IP addresses are actually active and being used on the Internet. We now leave the wonderful world of DNS behind us, and begin to concentrate solely on the IP address space.

So how does one determine if an address is active on the Internet or not? Well, the first, and most obvious technique is the famous IP 'ping'. 'Ping' works just like sonar. You send a *ping* to a specific address and the machine responds with a 'pong' indicating that it is alive and received your request. Ping is a standard component of the Internet Protocol (IP) and machines that talk IP are compelled to respond when they receive a ping request. With simple and freely available tools we are able to 'ping' an entire subnet. This is know as a 'ping scan'. Without going into too much detail, the response of such a ping scan can be interpreted as follows:

1. A reply from an IP address indicates that the address is probably in use and accessible from the Internet.
2. Multiple replies from a single IP address indicate that the address is probably actually a subnet address or a broadcast address and suggest a subnet border.
3. No reply can only be interpreted to mean that the machine is not replying to IP ping requests.

I realize that the latter point is a bit vague but that really is the only conclusion that can be drawn from the information available. I said that all machines that speak IP are obliged to respond to ping requests. Why not simply conclude that if the IP doesn't respond, it isn't being used? The confusion is introduced by modern network security products like firewalls and screening routers. In the real world, one often sees networks configured in such a way that the

IP ping packet is blocked by the firewall before the packet reaches the machine. Thus the machine would respond if it could, but it's prevented from doing so.

So we haul out the heavy artillery. Just about every machine on the Internet works with a series of little Internet 'post boxes' called *ports*. Ports are used to receive incoming traffic for a specific service or application. Each port on a machine has a number and there are 65536 possible port numbers. A modern machine that is connected to the Internet and actually functioning is almost certain to be using at least one port. Thus, if an IP addresses did not respond to our ping request, we can further probe it using a tool called a 'port scanner'. Port scanners are freely available software utilities that attempt to establish a connection to every possible port within a specified range. If we can find just one port that responds when probed, we know that the IP is alive. Unfortunately, the amount of work required to probe all 65,000 plus ports is often prohibitive. Such an exercise can takes hours per single IP address and days or even weeks for an entire range. So we're forced to make yet another assumption: If an IP address is active on the Internet, then it's probably there for a reason. And there are only so many reasons to connect a machine to the 'net:

1. The machine is a web server (and thus uses port 80 or 443)
2. The machine is a mail server (and thus uses port 25)
3. The machine is a DNS server (and thus uses port 53)
4. The machine is another common server – FTP, database, time, news, etc)
5. The machine is a client. In this case it is probably a Microsoft machine and uses port 139.

Thus, we can now modify our scan to search for only a small number of commonly used ports. This approach is called a *host* scan. It is by no means perfect, but it generally delivers accurate results and is efficient enough to be used with large organizations. The common ports we scan for can be adjusted to better suite the nature of the organization being analyzed, if required. The *nmap* network utility (available from www.insecure.org) is a powerful tool that serves equally well as ping scanner and a port scanner.

Thus, by the end of this phase we have fine-tuned our list of IP subnets and generated a list of actual IP addresses that we know to be 'alive' and that therefore qualify as targets for the remainder of the process. At this point, our findings are usually presented to the customer to ensure that we're still on the right track.

3. Visibility

At the start of this phase we're armed with a list of IP addresses that serve as targets to be attacked or in our case, analyzed. Our approach to analyzing (or attacking) a machine will depend on what network services that machine is offering. For example, a machine could be a mail server, a web server, a DNS server or all three. For each case our approach for the next phases will vary somewhat. Moreover, our strategy will vary according to the operating system of the machine in question and the precise version of the services it offers. During this phase, our precise objectives can be described as follows:

1. Determine what services are active on the target machine (i.e. what ports are open).
2. Determine the type and version of the active services.
3. Determine the Operating System on the target machine.

This information will allow us to plan and execute the remainder of the assessment.

It should be easy to see how we determine the active services on a machine. Once again we haul out our trusty port scanner. This time, we configure the scanner to test for all the ports and aim it at the machines, one by one. The port scanner will return a list of all the ports that are open on

the machine. As services and applications generally use the same port numbers, we are able to derive a list of the active services on the host and thus an indication of the host's function.

Let's examine briefly how a port scanner actually works. To do this, we first have to revisit the TCP connection. A TCP connection is established by means of a three step *handshake* protocol, which can be depicted as follows:

Client	Sends a connection request packet (called a SYN) to indicate that it wishes to make a connection to a given port.	SYN
Server	If the server is able to handle the connection, it replies with a SYN packet of its own, accompanied by an acknowledgement packet (called an ACK).	SYN/ACK
Client	On receiving the response from the server the client responds with an acknowledgement and the connection is established.	ACK

This is exactly the process the port scanner needs to go through to establish whether a given port is open or not. The handshake is executed for each of the ports specified in the range. Whenever the handshake is successfully completed, the port is recorded as open. Notice that no data needs to be transferred and the client doesn't have to transact with the server at the application level. Once again, *nmap*, is probably the tool of choice.

You're probably tiring of bullet lists by now, but once again there are a few facts that need to be noted:

1. The port scanner will only give you an indication of what ports are visible to you. There may be other ports that are actually open on the host, but are filtered by a router, a firewall or some other security device.
2. Even on a limited number of hosts, a full port scan can take a very long time. Most services listen on standard ports that are generally known. To save time, the port scanner can be configured to only scan ports that are generally known. This has the advantage of making the scan more efficient, but at the risk that unusual ports might be missed.
3. As the port scanner only requires a server's *SYN/ACK* response to mark a port as open, you can never be 100% certain that there is a service actually functional at that port. There are situations where a server may respond to a TCP *ACK* request even though you aren't able to actually communicate with the service in question. To fully determine whether there really is a service active that can be communicated with, one probably needs to exercise some human intuition. But there is one more automated technique that can help us improve the odds – banner grabbing. Banner grabbing involves querying the service on a given port to request its type and version, and it's the next test we perform in this phase.

Apart from the TCP handshake, most networked applications have their own handshaking protocol that commences only after the TCP connection has already been established. Somewhere within the handshake there's usually a way to request the service's name and version. Indeed, some services display their version details right at the start, before the handshake commences.

Thus, we use a simple utility called a banner grabber. The grabber is programmed to know how to extract the version information from a number of common network services. We aim it at a list of addresses and it will pull the banner information for any services it recognizes. In this way we can also better understand whether the application we expect really is listening behind a given port. There are a number of programs that will do banner grabbing for you. A number of port

scanners do this (e.g. Super Scanner), as do several vulnerability scanners (like Nessus). You can also do it using a simple telnet session, if you know the protocols well enough.

So, now we know what ports are open on a given host. We also know the type of application and the version that is listening behind each of these ports. In many cases this is all the information we need to identify the operating system. For example, the telnet port (23) is open on a given IP. We use *telnet* to connect to it:

```
# telnet 196.3x.2x.7x

Trying 196.3x.2x.7x...
Connected to xxx.xx.co.za.
Escape character is '^]'.
HP-UX u46b00 B.10.20 A 9000/831 (ttypl)
login:
```

Pretty obvious, huh? There are other, only slightly less obvious, clues also. For example, the *IIS* web server only runs on Microsoft machines, *sendmail* is a Unix mail server and ports 264 and 265 are a dead give away for a CheckPoint Firewall-1.

In addition to the simple tricks described above, we can also use a sophisticated technique called *OS Fingerprinting* to identify the operating system. For an excellent paper on the issue, consult www.insecure.org. This is the home of *Fyodor* – author of the acclaimed *Nmap* port scanner. Basically, OS fingerprinting is a bit like identifying someone's nationality from their accent. Many operating systems communicate on the Internet with some unique characteristics which make them distinguishable from other operating systems. Programs like *nmap* and *Queso* and many of the commercial security scanners have databases of OS *signatures* against which the 'accent' of a particular operating system can be compared. Some of the characteristics that are examined include unique combinations of open ports (ergo our example earlier), TCP initial sequence numbers and responses to unexpected requests. Fyodor, the author of the *nmap* utility, described it best himself. Read his paper at the link above if you want to know more. But do note the following: This technology is guess work at best and many OSs share the same signature and are therefore indistinguishable over the network. Moreover, servers are often housed behind firewalls and other security devices that may mask the real OS. A good fair deal of common sense needs to be applied when interpreting fingerprinting results.

All the technical inadequacies notwithstanding, at the end of this phase we're armed with as much information about each target machine as we can possibly extract. With this information in hand, we can now plan and execute the rest of our analysis.

4. Vulnerability Scanning

Before we progress with the rest of the methodology, let's take a moment to revisit where we are in terms of the bigger picture. You'll remember the question that started the whole process: What are the threats that my Internet-connected systems face and what are the chances of those threats being realized?

We started our quest with a *reconnaissance* exercise, through which we attempted to identify all the Internet-facing infrastructure that is relevant to the target organization. We did this with a number of steps. Firstly, we used various different resources to build a list of all the DNS domains that might be relevant to the target. We then used a set of DNS tools to map those domains to single IP addresses, which we in turn expanded to complete subnets on the (slightly brash) assumption that the IP addresses of a single organization are most often grouped together in the same space of the Internet.

We then explored the fact that the existence of a DNS Name-IP mapping does not necessarily prove that that IP is actually alive and active on the Internet. In order to narrow our vague list of subnets to a more accurate list of single IP addresses that are actually 'active' on the Internet we ran a set of *vitality* tests – designed to determine as accurately as possible whether a given IP address can somehow be reached on the Internet. The vitality tests included core router queries *ping* scans and TCP *hosts* scans.

Having built a list of IP addresses that are alive and well on the Internet, we set out to *fingerprint* those addresses – essentially to identify the operating system, the active services and the versions of those services for each address in our list. This information is the key input for our next phase – *vulnerability discovery*.

Before we can discover vulnerabilities on our targets, I guess we need to understand what a vulnerability actually is. My Oxford dictionary defines a vulnerability as an *exposure to danger or attack*. Why don't we simply think of a vulnerability as a weak point on a host or a system that may allow some form of security compromise. Let's explore a few different ways of compromising a system:

1. **Available Channels:** Often, everything an attacker needs to access your systems is right there in front of your face. The Internet, by its very nature, is verbose. Often a combination of verbose services provide enough information to compromise a system.
2. **Configuration Errors:** Very often, technology is smarter than the people that use it. In the headlong rush to add differentiating features software vendors often add functionality that users and administrators don't fully understand. An attacker who understands a system better than the administrator may use configuration errors to gain some form of unauthorized access.
3. **Programmatic Errors:** Computer programmers are just people, and like other people they sometimes make mistakes. Modern programs and operating systems are incredibly complex and are developed at frightening speeds. Sometimes programmers fail to cater for a specific scenario. An attacker who can identify that scenario can use it to force the program into a state that the programmer never catered for. In this uncontrolled state the program may cease to function correctly, offer up information that it shouldn't or even execute commands on the attackers behalf.
4. **Procedural Errors:** Occasionally errors exist, not in the technology itself, but in the way the technology is used. These weak points are the targets of *social engineering* attacks where an attacker manipulates people and the way they think to obtain information or some other form of unauthorized access to a system.
5. **Proximity Errors:** The Internet is a highly-connected system. This extreme level of interdependency sometimes results in strong systems being compromised through their trust relationships with systems that are weak.

Amazingly, a huge number of these kinds of vulnerabilities have already been identified, publicly raised and discussed in some forum. There are a number of sources for new vulnerability information. The first is the vendors themselves. When a software vendor becomes aware of a security vulnerability in one of their products they create a patch, or design a work-around and then publish a notification informing their clients of the problem and how it can be fixed. Many security product vendors also have researchers dedicated to discovering and documenting possible security vulnerabilities. The way this works is similar to the way virus scanner vendors discover and publish new virus signatures. Typically, the manufacturers of *Intrusion Detection Systems* and *Vulnerability Scanners* (which we'll get to a bit later) will do this kind of vulnerability research in order to differentiate their products. Finally, a multitude of security researchers occasionally discover vulnerabilities and notify the vendors or publish them in public forums. There is growing controversy regarding exactly how new vulnerabilities are discovered,

how these discoveries are motivated, how they are published and who benefits the most in the long run. But that's a topic for a paper all on its own. Suffice is to realize that, one way or another, new vulnerabilities are discovered and one way or another find their way into the public eye.

Now, the good and the bad guys have equal access to information about how systems could be hacked. What we want in the *vulnerability discovery* phase is to identify any of these known vulnerabilities so that those problems can be rectified before they are discovered by anyone else and perhaps used to compromise our systems.

How can we determine if the systems we are analyzing are subject to any of these known vulnerabilities? Well, recall the information that we've derived so far: for every visible machine we have the operating system, a list of the active services, and the type and version of those services. That's all we need to query a database of known vulnerabilities. Where does one find such a database? Why, the Internet of course! There are a number of sites that openly publish databases of known vulnerabilities. Two that I like very much are www.securityfocus.com and www.securiteam.com. My company has a similar database at www.hackrack.com. In fact, the Internet itself is a massive repository of information on known vulnerabilities. Simply use your favorite search engine (mine is www.google.com) and search for <the operating system name>, <the service name>, <the relevant version number> and the word "sploit" and you're bound to stumble on what you're looking for. Try it – you'll be amazed.

A well performed *discovery* exercise and a good search may be all you need to perform a vulnerability discovery analysis, but it's a slow and difficult process that requires a high level of technical competence. On a large and complex network this approach is probably completely infeasible. Fortunately, help is at hand in the form of automated vulnerability scanners. Such scanning software can be found in three general forms:

- a. **On host:** Host-based scanners are physically installed on the machine being assessed. They work very much like automated best-practice checklists, running a number of tests to determine whether the machine has been installed and configured according to what is considered to be best practice for that operating system in that environment. Such scanners typically run as a privileged user on the target system and are thus able to perform a comprehensive set of tests.
- b. **Network-based scanners:** These scanners have a knowledge base of tests that they then perform against the target systems over the network. Although these tests are run without any special privilege on the target system and are therefore less comprehensive, they do have a number of advantages: They do not have to be installed on each system being tested, and they more accurately reflect the possibilities open to an external attacker. In a sense, they identify the issues germane to system penetration from the outside, rather than internal configuration issues, and may better reflect the priorities of the security administrator.
- c. **Application Scanners:** Application scanners can be seen as specialized scanners that assess the security configuration of specific applications and services. Such scans may be performed on a host or over the network. Services like Web, database and NT domains that are difficult to configure and prone to security flaws can often be assessed in this way.

There are a number of commercial products available in each of these categories. I'm loath to mention names, but amongst the leading vendors in this space one must include the *ISS*, *Axent* (now Symantec), *Eeye*, *Next Generation Software*, *BindView* and *Foundstone*. There are many other commercial products, but the list would never be complete without at least one freeware product – *Nessus* from Renaud Derraison. The open-source *Nessus* scanner can hold its ground against the best of the commercial scanners on just about every front, bar support and reporting.

Most freeware scanners concentrate on detecting only a specific type or category of vulnerabilities (like web server vulnerabilities) and the list of names is almost endless.

All of these scanners will have access to a database of known vulnerabilities and will use a scanning engine to connect the target machine, execute a set of tests and report a list of vulnerabilities for each server tested. Like any golden goose, however, scanners don't quite live up to the promise. There are two main issues that plague all vulnerability scanners, namely:

1. Such tools can usually only test for *known* security vulnerabilities. Their effectiveness depends to a great extent on the accuracy and timeliness of the source of vulnerability information. Sometimes this information cannot be captured in a format the scanners can understand or simply does not yet exist publicly.
2. The test for a known vulnerability may cause a system to fail. Sometimes the only way to *really* determine remotely whether or not a system is vulnerable to some known weakness is to try and exploit that weakness and observe how the system behaves. This is an accurate form of testing but may have a detrimental effect on system operation. The alternative is to collect relevant information (like the service type and version) and make the decision on that basis. This *Non-Intrusive* approach, while safer, is much less accurate and often leads to a large number of 'false positives' – vulnerability reports that are based on superficial information and prove to be wrong upon further investigation.

A new generation of scanning software uses somewhat more intelligent scanning techniques, and may help to reduce the dependence on knowledge of the attack being used. *Retina* (www.eeye.com) is an example of a scanner claiming to belong to this group.

Whilst intelligent scanning is an exciting development, it's my believe that the failings of automated scanners are symptomatic of fundamental flaws in the *concept* of vulnerability scanning. From this we should learn an important lesson: You can't plug out your brain when you plug in your scanner. Always remember that a scanner will never discover all possible vulnerabilities on a system and that a scanner will always report problems that are not real. Thus every report must be carefully evaluated.

5. Web Application Analysis

In July 2002 the monthly *Netcraft* web site survey reported **37,235,470** active web servers (www.netcraft.com). Of all the possible services on the Internet, DNS, mail and web are by far the most pervasive. Of these, web services the most complex and the most frequently abused. Originally a simple text-based information service, the web has developed into a highly functional interactive application development platform that is being used for almost every possible application on both the Internet and Intranet.

Major vendors have recognized the power of the web and have made significant investments in web development platforms. These include Sun (Java), Microsoft (.asp, Site Server and Commerce Server), the open source community (PHP) and others (ColdFusion). Such platforms make it very easy for standard administrators to develop complex applications. A simple wave of web development wand and suddenly everybody's a programmer. Even me! Web applications are often developed by under qualified and inexperienced developers. Unknowingly many programmers make errors that provide us (and others) with precisely the vector we need to penetrate the private network. As these applications are built on standard platforms they tend to look similar, and tend to have similar security weaknesses.

As an increasingly prominent technology on the Internet, custom web applications must be considered part of every security assessment. Here's the problem though: We know that almost everyone who runs DNS is running BIND (the Berkley Internet Name Domain). Thus, if there is a

vulnerability associated with BIND, then it will affect everyone. The vulnerability will become publicly known and the signature will eventually find its way into vulnerability scanners and vulnerability databases. This is true for most common Internet services. We can test systems using such scanners and identify vulnerabilities that may impact us. But how does a vulnerability associated with *Joe Soap Inc's* home made customer support web site become known so that it can be included in our handy vulnerability scanner? Answer: It doesn't. Because web applications are so often "home made", there are no 'known' vulnerabilities and each application has to be looked at individually in an effort to identify security errors that the programmer may have made.

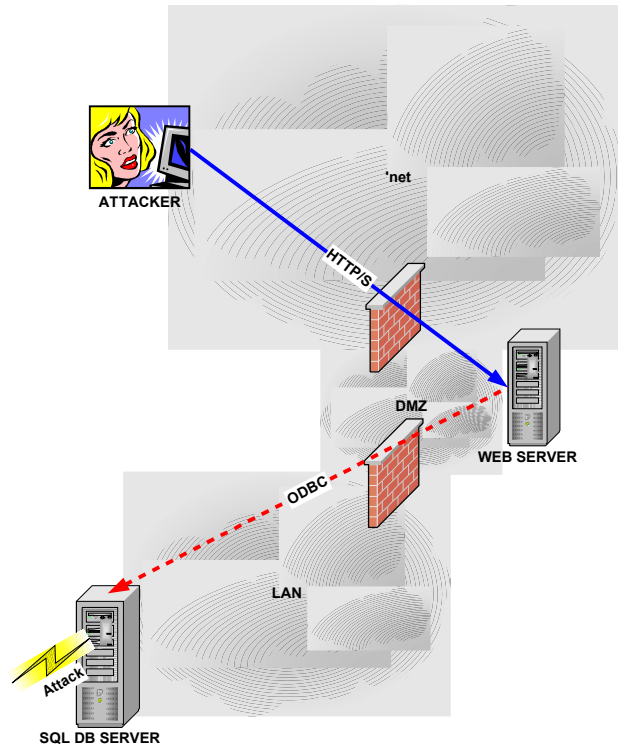
Web application assessments are a whole new ball game and in many respects have raised the bar for security analysts. Previously valid assumptions, methodologies and tools are no longer applicable and even leading practitioners are still only beginning to understand all the possible issues with these kinds of systems. One of the leading thinkers in this area is the *OWASP* group – the Open Web Application Security Project (www.owasp.org). The first draft release of their assessment framework is expected in mid October of 2002. Another valuable resource is 'Web Hacking – Attacks and Defense' by McClure, Shah and Shah. Also coming soon is a Syngress publication - "Special Ops" (see www.syngress.com/ops/) - that will have a nice introduction to the topic. As always there are countless others.

Our Web Application testing is mainly done 'black-box', which means that we examine the applications over the network without access to the source code. Of course, without access to the source code there's a fairly good chance that you're going to miss something important. Call this the 'fuzz' factor. We've addressed the 'fuzziness' of web application testing by introducing two frameworks of our own. The first is a simple list of questions that need to be asked when considering the application and the second is a simple list of common programming errors that should always be checked for. Even though every programmer (and thus every application) is unique, one does see the same errors being made consistently. We've compiled a list of common programming errors that can be checked for, even over the network. I'll describe the frameworks in some detail a little later, but first I'll need to quickly cover some basic concepts you'll need to understand.

- **HTML** is the *Hyper Text Markup Language*. It's the simple language used to describe how web pages should be formatted and displayed. HTML uses mark-ups, called 'tags', to tell the browser how a web page should look and behave. Examples of tags include <A HREF>, which indicates that the text to follow is actually a *hyperlink* to another page, or , which indicates that the text to follow should be displayed in bold characters.
- **HTTP** is the *Hyper Text Transfer Protocol*. It uses a TCP connection (typically on port 80) to transfer data between the web server and the browser. HTTP is used to transport all the files used in HTML, including text, graphics, sound, video and active elements like *Shockwave Flash* and *Java Script*.
- A **form** is a special kind of HTML element that allows the user to submit data back to the web site. A form consists of any number of **fields** of various types. A **hidden field** contains a value that is not displayed by the browser (more about this later) and is occasionally used by programmers to submit control information that the user doesn't need to know about. When dealing with normal form input fields, the user can enter data into the field, which is then passed back to the web site and extracted and processed by a server-side application, typically a CGI.
- A web site that wishes to keep track of its visitors will often make use of **cookies**, small data objects (usually stored as files) that are sent to the browser to be stored for later use. The next time the user visits that site the cookie is automatically resubmitted, complete with all the information the web site previously wrote there. You've seen this mechanism in use by sites that 'remember' your preferences from one visit to the next. Actually, your browser is using cookies to remind them.

- **CGI** is the *Common Gateway Interface*. It describes a standard way for web servers to pass information to programs and send the output from those programs back to the browser to be displayed. With CGI a web developer is no longer restricted to using only static pages. Instead, one can accept input from the user, pass that input to a program to be processed, and then send the output to the browser to be displayed. CGI is an old technology but has recently really come to its own. Other technologies like Microsoft's ASP (Active Server Pages) perform a similar function.
- A **script** is a small program that is not compiled into executable format before it is run. Instead, each line is read, compiled and executed by an *interpreter* whenever the program is run. Scripts are powerful because they are lightweight and easy to write and can often be run on multiple platforms without being modified. Increasingly web applications are being written in scripting languages like ASP, PHP, PERL and Java.
- **SSL** is the *Secure Sockets Layer*. It's a protocol that runs on top of TCP and uses a combination of public key and symmetric encryption technology to ensure confidentiality, integrity and authenticity for HTTP. When you select a web site with "HTTPS://" you're instructing your browser to tunnel the HTTP communication over SSL. SSL is a very powerful mechanism, but is most often used only to authenticate the web server and to encrypt the link between the browser and the server.
- **ODBC** is the *Open Data Base Connectivity* protocol. It's a standard way for applications to communicate with databases without needing to understand the peculiarities of that particular database implementation. ODBC is typically used between the web server and the database in architectures like the one we depict below.
- **Application development environments** are used to make the development of complex web applications quicker and easier. This will typically involve a number of elements, including: a web-optimized scripting language, reusable components, samples and example scripts, session key generation, database connectivity and the like.
-

Irrespective of the development platform being used, the vast majority of web applications have the same basic **architecture**, as depicted below:



This picture depicts a popular network architecture for web applications. The web server is located in a specially protected network segment called a Demilitarized Zone (DMZ). Two firewalls are used: One to protect the DMZ from the Internet and one to protect the internal LAN from the DMZ. This is sound security thinking and is done to minimize the impact of a web server compromise. However, two channels must be opened for this architecture to work. The user must be able to connect to the web server from the Internet and the web server must be able to connect to the database server. Remember this configuration, as these same two channels are what an attacker will exploit to gain access to the internal network.

Key Questions

Now we have the background we need to understand web application assessments. We've also looked briefly at a fairly simple, although very common network architecture. So, let's take a deeper look at the first of the frameworks we mentioned earlier. What questions should you ask yourself when faced with a custom-built web application? These are generic questions that border on the philosophical. However, they are important if we hope to see beyond the surface into the heart of the application we are assessing. The following are questions we ask:

1. Who are these people?

The design and architecture will more than often reflect the style and culture of the organization that wrote it. Conversely, much can be learned about the design and architecture of an application by understanding the style and culture of the organization. We seek this understanding at two levels:

- **Generically:** What is the *group* we are dealing with? What is their core business? What is their style and culture?
- **Specifically:** Who designed and developed the application we are considering? What is their background, their training and their style?

The answers to these questions will give us insight into how the application is likely to be constructed and where possible errors are likely to be found.

2. **How does this thing work:**

Next, we need to understand how the application in question was put together. The key to success here is an enquiring mind. No stone should ever be left un-turned in your effort to understand what makes the application tick. Here are some specific elements you should be considering:

a. **Where is the interactivity?**

Not every site is interactive. And some sites are more interactive than others. You need to identify the elements of the site that are interactive and determine how interactive they really are.

b. **How did they build this system?**

Remember the application development platforms that we discussed earlier? Each development framework has its own inherent strengths and weaknesses. By identifying and understanding the base component on which the application was built we know where to start looking for possible security vulnerabilities.

c. **Where do they get their data from?**

An interactive application needs to get its information from somewhere. As web applications are stateless, a place must be found to store the user's data. Determining where the back-end data source is located will teach us a lot about how the application works and how it might be compromised.

d. **How do they get their data?**

So the site's getting its data from somewhere else. Perhaps even a remote server. How? There are many options, including HTTP requests, XML, basic file access and the ever-popular ODBC. Again, each of these approaches has strengths and possibly weaknesses that we need to identify.

e. **How do they know who I am?**

On many sites you'll work anonymously. That is, without first authenticating you. However, many sites will want to know who you are before they grant you access. If this is the case you need to understand how authentication is performed. What is the method (e.g. Basic Authentication) and where is the user data stored (e.g in the NT SAM).

f. **How do they keep track of state?**

Once a user has been authenticated, a stateless application needs to keep track of a person as she/he navigates their way around. State tracking is also needed for complex processes, where the output from one step needs to be remembered as the input to the next. In each case, state tracking is an area much prone to error and one must have an in-depth understanding of this when an application is assessed.

3. **Why did they do it this way?**

Whatever the answers to the previous questions are, and whatever the architecture that you've mapped; there is always a reason the system was built in the way it was. If we can understand the motivating factors behind the design, we can gain valuable insight into where the system might be vulnerable. Here are some common motivating factors that I that we'll be looking for:

- **Laziness:** Often a specific architecture is chosen simply because it's the cheapest or the easiest. A lazy design is an insecure design.
- **Policy:** Often technical people are forced to stick with certain technologies because of high-level strategic- or political decisions within the organization. For example, one often sees an organization strategically aligned with Microsoft or with open-source technologies. Such architectures tend to be standard and are well known.
- **Inexperience:** We've already alluded to the fact that many web application developers are inexperienced and under-skilled. Such people tend to 'borrow' and copy code from example sites and other public sources and are prone to making silly, *amateur* mistakes.
- **Over-experience:** In our industry people who are highly skilled in one area often achieve this at the expense of other areas. Identifying the developer's particular "hobby-horse" will often tell you where *not* to look for vulnerabilities.

4. What are typical mistakes in this architecture?

Now that you've 'scoped' out the system you're assessing, you can begin to look for security errors that might be expected. You're working your way under the skin of the developer and attempting to see the mistakes that he/she may have overlooked. In the sections that follow, we will be discussing some common categories of programming errors and looking at some specific examples.

Risk Categories

A detailed discussion of all the possible vulnerabilities associated with custom web applications would be somewhat beyond the scope of this paper. What I've done instead is to list and describe all the categories. I'll then single out some specific instances and describe them in more detail as examples.

Here then a list of mistakes that are commonly made, even by experienced programmers. Having answered all the other questions from the previous list, we will carefully search for the following common vulnerability categories:

1. Server Application Vulnerabilities:

As with any network service, web servers are applications that are often exploitable via the *known* vulnerabilities we discussed in the previous section – 'Vulnerability Scanning'. Specialized 'CGI' scanners like *Whisker*, *Nikto* and the appropriately name *CGI-Scanner* are especially designed to find vulnerabilities in web servers. We've discussed this concept at length already, so I won't spend much time on it again.

2. Sample Scripts:

Many web servers and web development platforms ship with sample applications built to serve as examples of what can be achieved. These applications are designed to serve as examples only and have not been built with security in mind. Indeed, sample scripts are often blatantly insecure. Once again, most of these scripts are commonly known and can quickly be identified with a CGI scanner.

3. Hidden directories:

Directories that are accessible to the web server but are considered 'hidden' because there are no references or HTTP 'links' to them. Once more, scanners like Nikto and Whisker make short work of discovering such directories through brute force.

4. Administrative Back-ends:

Complex web sites are often developed by specialists at a high price. To facilitate cost-effective management and updates of such sites the developers often provide an administrative back-end through which the site owner can easily add or modify content without having to understand the HTML and other underlying technologies. Discovering such a back-end (again via brute-force) presents an easy vector for web site defacement and other attacks. More on this later...

5. Directory Traversal:

A directory traversal attack involves stepping through multiple levels of the file system structure using "..\". This feature is often used by attackers to trick an application into accessing a part of the file system it was never meant to. More on this later...

6. Input sanitation:

These wise words from my colleague Haroon Meer: "People have been screaming about poor (non) validation of user input for as long as I can remember so I don't even think that any of the normal excuses apply anymore. By now developers should simply have learned that 'all user input should be inherently distrusted' and therefore sanitized. Unfortunately most of the sites you come across seem to ignore sanitization of user input completely or do it selectively (often forgetting hidden fields)." A common mistake on web applications is the assumption about the inputs that user will provide, albeit it manually or via some automated process. Manual inputs include the data entered by users into fields and forms. Automated inputs are submitted without the user being directly involved and include things like cookies, URLs and hidden HTTP fields. Nothing that is provided by the user should ever be trusted and an application that does so is opening itself up to abuse. Possibly the most significant category of attack that stems from inappropriate input sanitation is SQL Injection.

7. State Tracking:

An enormous problem facing any web application developer is the question of how to keep track of a single user as he or she makes their way through the various areas of the web site. This is difficult for many reasons. The primary obstacle is the fact that each page of an HTML application and each object on an HTML page is fetched using a new HTTP session. Unlike 'stateful' applications like telnet or FTP, HTTP is 'stateless' and there is no standard way for the application to keep track of a user who has logged in. Mechanisms used for this include Cookies, Session Keys and Hidden Fields. Mistakes made in the way state is tracked can allow an attacker to 'highjack' a session and gain access to applications and data without being properly authenticated. An example of such an attack follows later.

8. Cross-Site Scripting:

Cross-site scripting (also called XSS) is somewhat different to the other attacks discussed thus far. Rather than attack the server or the application XSS attacks are aimed at the end-user's browser. XSS is not a problem of *input* sanitation, but rather a problem of *output* sanitation. It is typically used against sites that redisplay values that are input by the user. If the information presented by the user is not properly sanitized before it is displayed then an attacker may be able to use HTML tags to influence the way it is displayed by the site. This becomes particularly dangerous when the attacker inserts the <SCRIPT> tag, thereby forcing other users' browsers to execute the code the attacker specifies. An easy way to test for this is to insert HTML like this:

```
<SCRIPT>alert('Vulnerable!')</SCRIPT>
```

If the web site does not clean this up before displaying it will cause a pop-up message to be displayed by the visitor's browser. Obviously an attacker could do much worse than cause pop-ops, particularly when the site is trusted by other users.

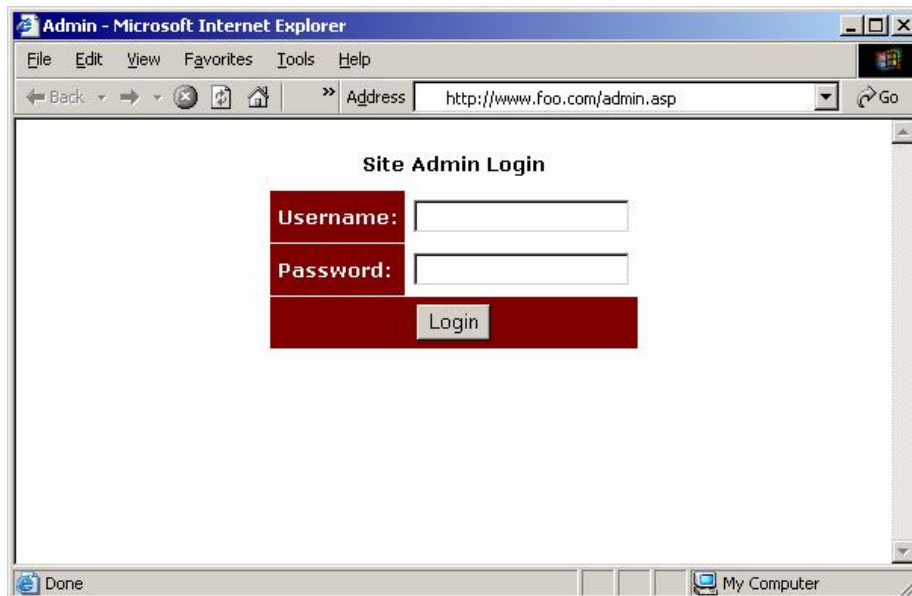
Having looked at a number of different threat categories at a very high level, I'd like to progress to some more detailed examples. These are:

1. A common problem with Administrative Back-Ends
2. A common problem Directory Traversal
3. A common problem with Input Sanitation
4. A common problem with State Tracking

Bear in mind that what follow are examples only. This is by no means a definitive study of this area.

A common problem with Administrative Back-Ends:

Complex web sites are often developed by specialists at a high price. To facilitate cost-effective management and updates of such sites the developers often provide an administrative back-end through which the site owner can easily add or modify content without having to understand the HTML and other underlying technologies. Discovering such a back-end (again via brute-force) presents an easy vector for web site defacement and other attacks. Amazingly enough, sites often depend on the fact that the backend is 'hidden' to protect it. Hiding the admin pages simply means putting them in a directory that's not published or linked to from other pages. Thus, you have to know the location of the page in order to use it. There's no magic trick to this, it's just a matter of brute-forcing common options. Directories like */admin/* and */site_admin/admin.html* are common examples. Here's an example:



Once you've found the relevant pages, there's often little or nothing to stop you from using them.

A Common Problem with Directory Traversal:

Try this from your Windows command prompt:

- `cd \\winnt\system32\`
- `cd`
- `cd ..\..\`

You've just performed a directory traversal – stepping through multiple levels of the file system structure using “..”. This feature is often used by attackers to trick an application into accessing a part of the file system it was never meant to. For example, imagine a simple ASP that's designed to fetch files from a special 'download' directory. Watch how we abuse it to steal the windows SAM file:

```
http://www.foo.com/scripts/getfile.asp?location=/download/../../../../../winnt/repair/sam._
```

This is a classical (if highly simplified) example of a directory traversal attack. You won't see this in real world any more, but you will occasionally see similar mistakes being made with web applications that access the file system directly.

A common Problem with Input Sanitation – SQL Injection:

Remember the site we visited earlier? A typical web page that requested a user name and password before it would allow access:



So, you enter your details. The correct user name and password are usually stored in a database, probably located somewhere on the private network, as depicted earlier. When you click the 'Login' button the values you entered are submitted to the web application. There, they are inserted into a *Structured Query Language* (SQL) query. In many applications the query will end up looking something like this:

```
@result = "select * from Users where  
user = '$username' and  
pass = '$password';"
```

Can you see what's happening? We're requesting from the *Users* table all the records where the *user* and the *pass* fields match the values that were entered in the form. The logic of the program will go on to say that if any records are returned then the user must have entered valid credentials. Its simple, its logical and it works. Here's where it goes wrong: The developer is trusting the user to input valid data. Instead, we enter a value like this:

```
x' OR 1=1--
```

into one of the fields. Observe the impact this has on the SQL query:

```
select * from Users where  
user = '$username' and  
pass = 'x' OR 1=1--'
```

The two dashes at the end (--) are comments in SQL and serve to nullify everything that appears to their right. We use them to remove the trailing single quote (') and thus clean up the SQL query. The query will now return all the records from the table *Users* where the username is equal to some value (*\$username*) and the password is equal to 'x' (and here comes the important part) **OR 1=1**. As 1=1 is always true it is clearly also true for every record in the *Users* table, which results in every record being returned. By the logic of the program, which was waiting for

at least one record to be returned, we must be a valid user and are allowed access to the site. TADA!

But SQL Injection can get even uglier. By carefully constructing SQL queries, an attacker might query the entire database, insert entries into the database or even get the database to execute commands. The latter is probably the most insidious of the SQL Injection attacks. Here's an example of how it would look against a Microsoft SQL server. In a field that's 'injectable' we enter a query like this:

```
x' exec master..xp_cmdshell 'dir'--
```

This of course gets integrated into the complete SQL statement to produce something like this:

```
select * from Users where  
user = '$username' and  
pass = 'x' exec master..xp_cmdshell 'dir'--'
```

Let's step through that slowly. The SQL query is now broken into two distinct parts:

1. Select all the records from the *Users* table where the *user* is some value (\$username) and the password is 'x'. The result: Nothing. Let's move on...
2. The second part of the query is distinct and runs after the 1st part is finished. Let's break it down into its various sub-components:
 - a. **exec** instructs the SQL server to run a stored procedure
 - b. **master** is the name of a database default on all MS SQL servers. It contains a number of pre-configured stored procedures.
 - c. **xp_cmdshell** is the name of the *extended stored procedure* to be run. A stored procedure is simply a small program that the SQL server can execute. *xp_cmdshell* allows one to specify any DOS command to be run, as if from the command prompt. Thus, any command that you can execute from your beloved C:\ prompt can also be called in this fashion from within a SQL query.
 - d. **dir** (the command within the quotes) is the command we want executed. I've used a simple 'dir' as an example, but it can really be almost anything you can do from the DOS prompt.

Simply put: By carefully crafting the right kind of SQL statement, an attacker may be able to gain access to the host itself, not just to the database.

The root cause for all the variations of SQL Injection attack are the same – user input is not properly sanitized. An easy test for vulnerability to this type of attack is to observe how the application behaves when one inserts a single quote in one of the fields. What we're looking for is an error message like this:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
```

```
[Microsoft][ODBC ... Driver] Syntax error in string in query expression 'username=" AND password ="'.
```

```
/login.asp, line 32
```

This error shows us two very important things:

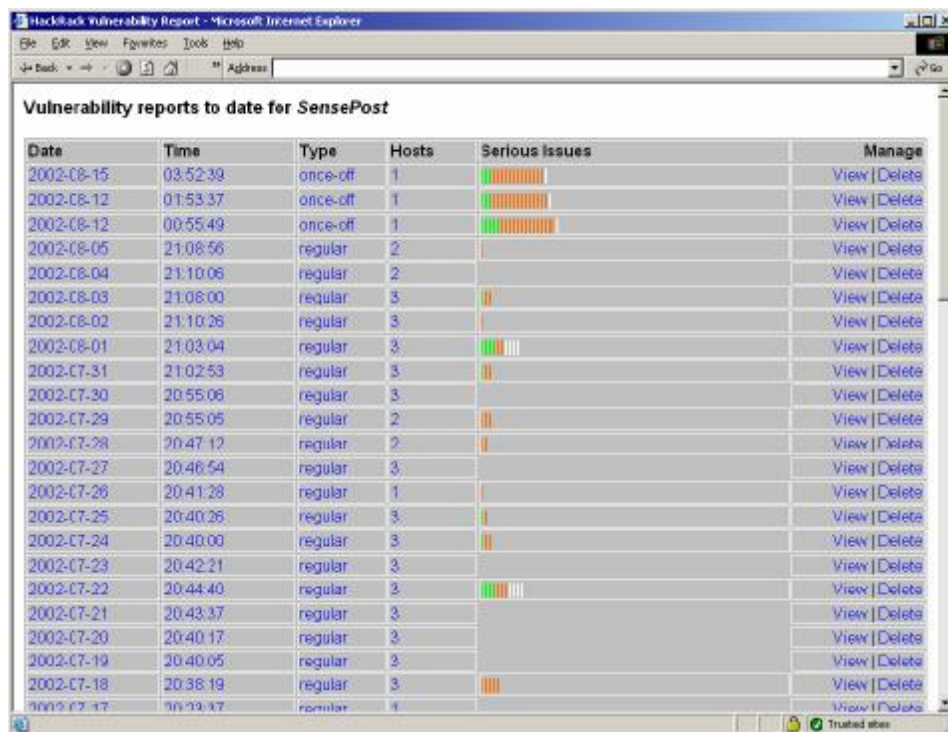
- 1) The application has a typical architecture using ODBC to make queries to some SQL database.

- 2) The programmer is neither sanitizing the user's input nor trapping errors from the database.

These are exactly the kind of errors that put the application at risk. SensePost has an automated tool called 'mieliekoek' that helps us search for these errors on a large scale. What the tool does is to run through the entire site, identify every field in every form, insert a single quote and check to see whether it returns an ODBC error.

A common problem with State Tracking

Below is an example from a simple web application developed using PERL. The page you see shows a summary of reports that have been generated by the system. By clicking on one of the reports the user can open a second browser window that displays the report itself. Note that each user has their own account. The reports are account-specific and contain highly sensitive information that should never be accessed by other users.



Date	Time	Type	Hosts	Serious issues	Manage
2002-08-15	03:52:39	once-off	1		View Delete
2002-08-12	01:53:37	once-off	1		View Delete
2002-08-12	00:55:49	once-off	1		View Delete
2002-08-05	21:08:56	regular	2		View Delete
2002-08-04	21:10:06	regular	2		View Delete
2002-08-03	21:08:00	regular	3		View Delete
2002-08-02	21:10:26	regular	3		View Delete
2002-08-01	21:03:04	regular	3		View Delete
2002-07-31	21:02:53	regular	3		View Delete
2002-07-30	20:55:06	regular	3		View Delete
2002-07-29	20:55:05	regular	2		View Delete
2002-07-28	20:47:12	regular	2		View Delete
2002-07-27	20:46:54	regular	3		View Delete
2002-07-26	20:41:28	regular	1		View Delete
2002-07-25	20:40:26	regular	3		View Delete
2002-07-24	20:40:00	regular	3		View Delete
2002-07-23	20:42:21	regular	3		View Delete
2002-07-22	20:44:40	regular	3		View Delete
2002-07-21	20:43:37	regular	3		View Delete
2002-07-20	20:40:17	regular	3		View Delete
2002-07-19	20:40:05	regular	3		View Delete
2002-07-18	20:38:19	regular	3		View Delete
2002-07-17	20:39:37	regular	3		View Delete

If you select one of the links, the URL will look something like this:

https://.../HR_VRPTFRM.pl?TNum=7669...ef2&Report_ID=123&Verbose=No

where the number after the part 'Tnum=' is actually a 32 byte hexadecimal string. Notice how we maintain state between these two connections. A session ID (Tnum=...) and a report ID (Report_ID=...). The session ID is used to remember who is logged in (it's mapped to an entry in a database on the server) and the Report_ID is used to communicate which report was requested. Just about all applications will take care of *authentication* by verifying the validity of the session ID (Tnum). A common mistake, however, is to fail in *authorization*. For example one could easily omit to consistently check whether the owner of the session key was in fact authorized to access the report ID contained in the URL. Remember that the web application has no control over what is put in the URL. For an attacker to keep a valid session ID but change the value of the report ID would be trivial. It's hard to believe that this kind of error is extremely common, even in high-end applications like Internet Banking.

Web Security Assessment Tools

Al right, so if you've decided to assess the security of your web applications and you just can't bear the thought of paying someone like me to do it, what do you need in your tool box? Here's a good start:

- **Whitehat Arsenal** (www.whitehatsec.com)
To the best of my knowledge the *WhiteHat Arsenal* is the first (and, to date, only) comprehensive web application security scanning tool. The tool was written by Jeremiah Grossman and is commercially available from the address given above. All of the other tools I'll be discussing are available under some form of freeware arrangement.
- **@Stake Web Proxy** (www.atstake.com)
The @Stake proxy is an amazingly tight and powerful tool that allows you to intercept and manipulate the data that your browser would normally send to a server. Thus one can play around with fields, URLs, cookies and the like through a handy graphical interface. This tool is by far the slickest offering of its kind and serves to suggest that @stake may well have been well ahead of the rest us in this particular game. The @stake proxy is a must.
- **Spike v2.6** (www.immunitysec.com)
The Spike proxy is written by Dave Aitel and is really two programs combined into one. The Spike proxy does very much the same thing as the @stake proxy, though with perhaps a little less finesse. The second component is what's known as a protocol "Fuzzer", which is essentially used to 'brute-force' applications over the network in search of possible overflow conditions. This element of the tool is relatively new and exciting, but is a little beyond the scope of this paper.
- **HTTrack** (www.httrack.com)
HTTrack is what's known as a web site 'whacker'. It basically performs a mirror of the selected site to your local drive, mimicking the sites structure and recreating each file. A local mirror allows one to examine and analyze the site at your leisure. We use a mirror, amongst other things, to search for active content, to search for links to other sites and as an input to the *Mieliekoek* SQL Injection scanner. HTTrack is both smart and robust and has to be one of the best of this generation of tools.
- **Mieliekoek v2** (www.sensepost.com)
Mieliekoek can probably best be described as an SQL Injection brute-forcer. You have to have a locally mirrored copy of the site, as provided by HTTrack. Point Mieliekoek at the mirror and it will identify every field in every form. For each field identified Mieliekoek will then attempt to inject the character you specify (typically `) and measure the server's response. The correct kind of SQL error indicates that the site may be vulnerable to attack. Mieliekoek will allow you to browse through the findings and also to experiment with different inputs to measure their impact on the site.
- **WebSleuth** (www.geocities.com/dzzie/)
Call WebSleuth the hacker's browser. It's essentially a standard web browser that affords you a higher level of interaction and control over the site you're browsing. Available functions include the ability to disable scripts, to view hidden fields and to edit data before it is submitted to the site. Websleuth supports the use of independently written plugins and has gained the support of various respected security practitioners in this way.
- **Nessus** (www.nessus.org):

Nessus is the excellent open-source network security scanner written by Renaud Derraison. Although this not a web-specific tool it does include a number of scripts that search for issues like *cross-site scripting*. Nessus was discussed in more detail in the previous section.

- **Nikto** (www.cirt.net)
Nikto, like *Whisker*, *Arirang* and others, is probably best described as a CGI-scanner. As discussed in earlier sections, the CGI scanner is a specialized vulnerability scanner that has been optimized to find known problems with web server vulnerabilities. Apart from the usual stuff, CGI scanners can also be used to track down false positives and the like.

--- the end ---