When the tables turn

A discussion paper on passive strike-back

for Black Hat Asia

SensePost Research

September 2004

Document Information

Description:	**When the tables turn**
Date:	**26/09/04**
Issue:	**1.0**
Last modified:	**2004-09-28**
Author:	**charl van der walt**

SensePost Contact Details

Physical Address	Postal Address	Contact Number	Fax Number
Lakeview 2 Office Building Ground Floor 138 Middle Street Nieuw Muckleneuk Brooklyn South Africa	P.O Box 10692 Centurion 0046 South Africa	+27 12 460-0880	+27 12 460-0880

Contact E-Mail Addresses		
General:	info@sensepost.com	www.sensepost.com
Training:	training@sensepost.com	
Research:	research@sensepost.com	
HackRack:	info@hackrack.com	www.hackrack.com

Revision History

Document Version	Description	Date	Author
0.1	Concept	May 2004	Roelof Temmingh
0.2	1st Implemtation	July 2004	Roelof Temmingh & Haroon Meer
1.0	Release	24/09/04	Charl van der Walt

Table of Contents

Figures

1 Summary

Until now network security defences have largely been about building walls and fences around the perimeter of the network. With this passive approach to security the attacker has the prerogative to strike at will, attacking when and where he chooses. Even if the attack fails the victim carries a high cost in terms of the technology, the bandwidth, the time and other resources required to keep the attacker out. The attacker, on the other hand, carries almost no costs and, using various tools and automation techniques, can continue trying until he finds a kink in the armour and finally achieves success. Therefore this passive-defensive approach to security on the Internet ultimately advantages the attacker.

Contrast this against the idea of spiking the 'walls' and electrifying the 'fences' that traditionally constitute the network security perimeter. By making an attack on our network costly and even dangerous we can force the attacker to proceed cautiously and carefully consider his every move. This approach may not actually improve the level of security, but it does at least even the odds of the conflict

In this paper we discuss obstacles that could be possibly be placed in the path at various phases of an attack in order to slow down or even cripple the attacker's tools. As such obstacles should only ever affect the attacker, and never an innocent bystander, we have labelled the concept "Passive Strike-Back". "Passive Strike-Back" explores techniques and tools that can be used to turn the tables on prospective attackers by using *Camouflage*, *Disinformation*, *Misdirection*, *Obfuscation* and *Proportional Response*.

In the sections that follow we will explore the thinking behind passive strike-back, consider its advantages and disadvantages and then examine some new and existing technologies with which the concept could be implemented.

This paper explores the concept for research purposes only, legal, moral and ethical questions still need to be examined and readers who choose to implement any of these techniques do so at their own risk.

2 Introduction

2.1 Analogies for passive strike-back

There are many illustrations and demonstrations of passive strike-back techniques in fields outside of information security. These analogies serve to stimulate thought on the issue:

2.1.1 Analogies from nature

The kind of passive defensive strategies deployed on computer networks are almost never observed in the animal kingdom. Rather, almost all defensive techniques deployed by animals have an active component. Here are some examples:

- **Vigilance:** An animal that is not vigilant ends up being eaten. Vigilance becomes part of the animal's time budget and must be managed along with other demands on time. Vigilance can also be shared and often drives animals, even from different species, to group together.
- **Crypsis:** An alternative approach is to remain extremely well hidden. By blending with the environment, moving carefully and not panicking an animal can avoid detection by a predator. Other animals disguise themselves as something else completely, like the Scorpion Fish that can look like a rock or the Stick Insect that can look like, well, a stick. Animals sometimes mimic other, dangerous animals in the hope of scaring predators off.
- **Active Defence:** Chemical feeding deterrents carried in body tissues are a form of active defence. This is common in insects, such as the monarch butterfly and in

marine invertebrates. A few vertebrates, such as poison-arrow frogs and birds are poisonous as well.

- **Body Size:** Whilst size is not strictly-speaking a defensive technique, elephants, hippopotami, and, some species of whale are good examples of species in which large size is a clear deterrent to predators.
- **Predator Saturation:** An alternative approach to defence is to produce so many of a species that it doesn't matter if one gets eaten – there are never enough predators to eat them all. In such strategies the individual animal puts up almost no defence at all and the group survives because some individuals always escape predation.

The applicability of these defensive strategies to the Internet world should become clearer as this paper progresses.

2.1.2 Analogies from warfare

The concept of a 'just war' is common in the theory and history of warfare. The just-war tradition is as old as warfare itself. In his *Summa Theologicae* the Saint Thomas Aquinas presents a general outline of what would become the just war theory, discussing the kinds of activities permissible in war as well as the *justification* of war.

The principles of a "just" war are commonly considered to be the following:

- Having just cause
- Being declared by a proper authority
- Possessing right intention
- Having a reasonable chance of success
- The end being proportional to the means used.

Once again we see that a proportional and justifiable *response* has long been considered a legitimate strategy for *defence*.

2.1.3 Analogies from ideology

The principle of "An eye for an eye" is commonly known and used in many parts of the world, and has become almost 'pop culture' here in the west. The phrase "An eye for an eye, a tooth for a tooth", also known as *Lex Talionis*, refers to a form of retributive justice. The phrase is quoted from the book of Exodus in the Jewish Torah (or Christian Bible) and actually sets for the commandment that, in a society bound by the rule of law, the punishment for a crime should be proportional to the crime itself.

So we see again that a proportioned response to some form of injustice is ideologically supported in many spheres of life.

Passive strike-back techniques like *disinformation* (misinformation that is deliberately disseminated in order to influence or confuse rivals) are already commonly used by national and military intelligence services, and even in computer security, as seen in honey pots and similar technologies.

2.2 A cross section of a typical attack

As clichéd as it has started to sound, one really must "know thy enemy". This is especially important for passive strike-back, where our objective is to hit back at clearly identifiable aggressors.

A complete hacking attack over the Internet can usually be broken up into a number of discernable phases. Whilst the exact order of the phases, the emphasis placed on each phase, the tools used etc. may differ from attack to attack, it is likely that one will observe all of the following techniques being applied:

2.2.1 Reconnaissance & Footprinting

Given that the attacker is focusing on a specific 'organization', on some real-world entity like a company, or a government, the attack must begin by extracting possible target IP addresses. As the link between the real world and the Internet world hinges on a company's domain name, this is most often where an attack will begin. The attacker will typically start from the target's DNS domain name and spend time surfing the web and using search engines to understand as much about the target as possible, primarily with a view to deriving other relevant domain names. Automated surfing tools (called "suckers" or "spiders") may be used to automate this process.

The attacker will then use various kinds of DNS queries (e.g. zone transfers) and DNS mining tools to extract as many relevant DNS names as possible from the domains that were found.

Next the DNS names will be translated, again using DNS queries, into target IP addresses that can actually be attacked.

2.2.2 Network Mapping

Having identified a number of individual addresses that could be attacked, a thorough attacker will spend time mapping the network in which those addresses reside. This is done with a view to understanding the victim's network topology and defence systems and with the hope of possibly identifying additional targets.

Various network trouble-shooting techniques will be (ab)used at this point. These include ICMP and TCP *pings*, and the *traceroute* utility. The attacker will analyze the responses to various network-level requests in order to gain an understanding of how the target infrastructure fits together.

2.2.3 Host Mapping

With a number of target addresses in hand, the attacker will attempt to map out the open ports, active services and service versions on each. This is primarily done using various forms of TCP and UDP port scanning. Port scanning tools send numerous network level requests to the host and then interpret the responses to build a picture of the function and configuration of the target. With a good port scanner like *Nmap* and some luck the attacker can pin point the exact operating system and service pack levels of the target.

2.2.4 Vulnerability Discovery

The attacker now has more than enough information with which to select and use a vulnerability-scanning tool. These tools range from the shotgun-like 'Nessus' security scanner, which is capable of identifying thousands of different vulnerabilities, to highly tuned and specialized scanners that attempt to identify one, specific, vulnerability only. All of these scanners share a basic method of working, however: They send out a number of specially-crafted requests then collect the replies and examine them for the telltale signs of a vulnerable system.

Many attackers will catalogue all the vulnerabilities discovered before selecting the preferred avenue of attack.

2.2.5 Vulnerability Exploitation

The attacker selects an attack vector and now begins the process of actually exploiting the first target. The means of attack will of course depend on the vulnerability being exploited, but will as often as not involve executing a program that exploits the problem. A skilled attacker may have to write this code himself, but is just as likely to reuse code that was written by someone else. The Metasploit Framework is a powerful set of open source exploits and exploit writing tools. If a target machine is determined to be vulnerable to some problem, the attacker is most likely to find a working exploit in a framework like Metasploit or at some other private or public exploit repository.

2.2.6 Web Application Hacking

Web-based applications, written in Java, Perl, ASP or the like are flexible and easily developed. However, such convenience comes at a price. Web-based applications represent both an attractive and a convenient target for attack and, because many applications also connect to key business systems, a compromised application can often have extremely serious implications. The Gartner group suggests that 70% of the malicious attacks on the web occur at the application level.

For the attacker web applications represent both an opportunity and a challenge. As such applications are custom written they can't be scanned for 'known' vulnerabilities in the same way that more common applications can.

However, many different kinds of scanners are still used to map, mine and probe web applications. Once again, these scanners use the same principle used by all the other scanners encountered thus far: They send a request over the network, then collect, process and store the responses received. The attacker will then analyze this data for any signs of vulnerability.

Understanding these discernable 'phases' in attack positions us to design and implement strike-back defenses. But there's still a little more we first need to understand.

2.3 Observable trends in "Hacking"

Without needing too much insight, one can easily observe some basics trends or characteristics in the field of computer 'hacking'. By 'hacking' in this context we specifically refer to the act of breaking into computers and networks over the Internet. Some of these observations suggest that the time is ripe for more active defensive techniques like strike-back.

Relevant observations include the following:

2.3.1 People are lazy

People are lazy, and in many cases hackers are *especially* lazy. This does not suggest that these people do not work hard, only that they'll avoid doing work when it's not completely necessary. Hence the massive popularity of tools and techniques that can automatically perform and repeat menial tasks. Brute force tools like "Hydra" are a prime example of this. Surely the smarter hacker would spend time and energy developing, learning or improving a tool like Hydra than running a brute-force attack by hand. There are countless examples of tools that help hackers simplify or automate menial tasks.

This brings us to our next point:

2.3.2 You're only as good as your toolbox?

Whilst many hackers are capable of designing and coding complex and sophisticated software systems, many of the tools hackers use are developed by others and are freely available at little or no cost. The pure dominance of some of these tools of their field (like *Nmap* as a port scanner, or *Ethereal* as a network sniffer) simply cannot be disputed. Thus it is probably fair to say that an attack on your network over the Internet will most likely be conducted using one or more of these leading technologies. Indeed, even if a hacker *were* to develop a private tool for some purpose, it is unlikely that's its basic form of operation will be much different from that used by the dominant technology.

To the degree that this is true we now *know* something about the attacker. Knowing that an attacker uses tools, and what *kinds* of tools an attacker uses, is extremely important when consider the idea of passive strike-back.

2.3.3 A mechanics car is often broken

A hacker, looking to exploit some hole in a security system, only has to get lucky once. One mistake on the part of the security administrator could be enough to allow a successful attack.

One wonders, however, whether the same isn't true for the attackers' own tools and systems. Whether the attackers are not perhaps *themselves* making mistakes that leave them open to attack. The fact that many 'hacking' tools are developed by hobbyists with no formal quality control or review processes, and the hacker's traditional aversion to norms, rules and controls suggests that it wouldn't be surprising to find a hacker using an un-patched workstation system, or hacking software that doesn't do proper bounds checking. Experience suggests that this is, in fact, the case.

2.3.4 Hacking is really just data analysis

If one examines the process used by a hacker to discover and exploit holes in the target system, then it soon becomes apparent that, at almost every stage, hacking is largely just data analysis. Let's consider a few examples:

The hacker performs a DNS 'zone transfer' to derive a list of potential target names. The request is sent and the data (the zone) that is returned is collected, stored, possibly processed and then examined for useful information. The attacker then performs a 'ping sweep' to determine which IP addresses are active within a given range. For every possible address a request is sent out on the network. The replies are collected, stored, possibly processed and the examined for useful information. The same process is applied for a port scan, and again for various kinds of vulnerability scan.

This characteristic of hacking is very important to understand if the full potential of passive strike back is to be grasped.

2.4 Summary

The following points summarize the thinking of this section thus far:

- Current Internet network security techniques are essentially passive in nature
- This passive approach to network security is essentially to the advantage of the attacker, who can continue attacking at little cost until he eventually succeeds.
- There is precedent in various other fields, from nature to religion, for a more active form of defence, based on the principle of justified, proportional response.
- Hacker techniques seldom vary too much. This offers us the advantage of knowing how an attack will look when it occurs.
- Hackers have a larger dependency on technology and tools. Like the technology we're defending, this software can also have bugs and is also vulnerable to attack.
- Hacking involves a large amount of data analysis. The data is generated by sending various kinds of probes to the target over the network then collecting, processing and analyzing the responses received.

3 Why we control the hacker

3.1 There are no rules

It could be said that the art of hacking revolves around understanding the rules that govern technology, and then breaking them. We see this principle all the time. Your email 'Reply To:' field should contain your email address, but what happens if it doesn't? A TCP connection packet should have a high source port, but what happens if it doesn't? A user name should always be less than 50 bytes, but what happens if it isn't?

This blatant disrespect for the standards and conventions of Internet protocols and applications is what gives hackers their edge. However, the same thinking can be also be used in defensive technologies: "A host should only reply to a SYN with a SYN ACK if the port is really open". "A machine should only to an ICMP ECHO REQUEST if it has the corresponding IP address". "A DNS reverse zone should map IP addresses back to their legitimate machine names". "A web server should reply with '404 File Not Found' error

message when asked to serve a file it doesn't have". All of these are conventions that the attacker depends on when probing the network, then blatantly ignores as it suites during the attack.

If network defence systems bend the rules in the same way the information returned by the attacker's probes becomes completely useless, and could even become misleading or dangerous.

3.2 We own the information

Whilst there is a perception that hackers are omniscient the truth is that the attacker is as blind as you are. The Internet is a vast space that separates the attacker from your systems. Thus the attacker never really *knows* how your systems are behaving, he is forced to *deduce* based on the information returned from his probes. This is the fundamental nature of the Internet and there's nothing the attacker can do about it. There is usually no hard link between the probe the attacker sends and the information that is returned. In reality there are two distinct processes: (A) Probe data originates from the attacker and (B) response data originates from your network. Thus all the data generated in response to the attacker's probes originates from your network and is therefore *completely yours to control.*

Every piece of information, every single IP packet that the attacker sees from your network is in essence sent to him by you. This includes:

- IP Packets (and all their features)
- Forward and reverse DNS entries
- Banners
- Error codes, status messages etc.
- Web pages
- Etc.

The data you send is captured by the probe, processed by the probe, stored by the probe and later possibly rendered by the probe. Therefore the network is in at least as good a position to strike at the attacker as visa versa. Moreover, if one recalls what was said earlier about the "mechanic's car", the network may well have a better chance of succeeding than the attacker. Moreover, as the attacker only receives traffic from us in *response* to the probes sent, there is little chance of involving innocent bystanders. As our traffic is always sent as a response passive strike back is essentially self-regulating.

3.3 Summary

The following points summarize the thinking of this section thus far:

- Administrators who realize that almost any rule on the Internet can be broken start to think like hackers themselves. This robs hackers of much of their advantage.
- There is no real concept of a 'circuit' on the Internet. All communications are actually composed of requests and responses.
- All responses originate from the target network, and are therefore completely under the control of the security administrator.
- This means the administrator has at least as much opportunity to attack as the attacker does.
- As the attacker typically doesn't have a defensive mindset, he may well be more vulnerable then the target originally was.

4 Introducing Passive Strike-Back

It should be clear now that strike-back defences are both feasible and possibly justifiable. In this section we explore some of the technical details of passive strike-back defences and look at some examples of such techniques in action.

4.1 Strike-Back at Different Levels

As strike-back will be designed to operate over the Internet, it can possibly be implemented at any of the layers above layer 3 in the OSI stack:

- **Network Layer:** This is possibly the easiest layer at which to implement strike-back. Any characteristic of an IP packet can be manipulated. The most important, and most significant of these is of course the source IP address. As we have full control over the IP packets originating from our network we can easily create massive noise and confusion by generating random ICMP and UDP packets in response to various probes.
- **Connection Layer:** TCP connections can also be toyed with quite easily. A spoofed SYN-ACK is indistinguishable from a real one and can play havoc with port scanner and other probing tools. La Brea tar pits, which play with the TCP window size, can force connections from an attacker to stay open indefinitely without using any resources on the server side.
- **Network Application Level**: Network applications, like mail and web daemons, are most often the targets of malicious activity. Responses sent by these applications over the Internet are fully under our control and huge confusion, perhaps even damage, could be caused by messing with application banners, application error codes and application-level responses.
- **Web Applications**: We pointed out earlier that web applications are currently a special case that is very interesting to attackers. Once again, every element of this application's behaviour is under our control. Banners, error codes and *actual content* can all be crafted in ways that make an attacker's life miserable. As web content is active and is executed within the attackers browser, this layer presents us with numerous opportunities for passive strike-back.
- **Data Level**: The use of *disinformation* has always been common in the intelligence world. An attacker that illegitimately accesses data on your systems, for example, presents himself as a target for strike-back, via misleading information or even malicious content, Trojan horse etc. Recently discovered vulnerabilities in 'passive' data formats like JPEG present us with even more opportunities to use this kind of attack.

Examples of strike-back attacks at all of these levels will be presented at the end of this paper.

4.2 Types of Strike-Back

As we saw with the analogy from the insect world, there are various different kinds of strike-back defence. We have identified the following four groups:

4.2.1 Strike-back that stops individual attacks

This kind of strike-back is already commonly in use. The idea is to identify an attack that is progress and then move to stop it. Detection of the attack would typically be done via signatures, and common responses include reconfiguring firewalls (shunning) and sending TCP RST packets. IDS and IPS commonly implement this technique.

4.2.2 Strike-back that creates noise and confusion

The simplest and most effective forms of strike back are those that simply create noise and confusion. In our analogy from the animal world we described this as 'Body Size' and 'Predator Saturation'. The possibilities here are almost endless, but they range from simply creating multiple, random responses to ping requests, to more complex OS mimicry, to fully mimicking another organization. This kind of strike-back is especially effective in slowing down or stopping automated tools. We'll provide examples of this kind of attack in a short while.

4.2.3 Strike-back that attacks a specific tool

We mentioned earlier in this paper that, at various stages of an attack, there are certain tools that the attacker is almost bound to use. Even in cases where the attacker prefers to write the tools, there is very little that the tool can do differently. As we know, for example, there are only so many ways to run a port scan.

These tools present perfect targets for strike-back. Not only do we know (and control) the data the tool is gathering, the very use of such tools suggests malicious intent and justifies some kind of response.

The objective of this kind of strike-back is actually to cripple the tools used by the attacker at a given stage in the attack.

4.2.4 Strike-back that attacks the attacker's host or network

In extreme cases strike-back can aim to damage or cripple the attacker's host or network. The purpose of this kind of attack is to make the attacker think twice before doing anything malicious. Whereas the attacker's biggest concern to date has been spotting and avoiding IDS and Honey Pots, he's now forced to ask if is workstation and workstation applications are patched, whether his systems are properly firewalled, and whether his attack tools are themselves safe from attack.

4.3 Identifying Malicious Activity

The key to successfully implementing passive strike-back is the ability to always accurately identify malicious activity. Anything less than 100% accuracy could attacks to be launched against innocent parties, possibly with disastrous results. This is where the "passive" component comes into play. The driving principle behind passive strike-back is that the strike-back attack is never 'launched' against anyone. Unlike signature-based defence systems passive strike-back doesn't attempt to spot an attack and then respond, rather passive strike-back allows the attacker to 'fetch' the strike-back attack himself. This is conceptually a little difficult to grasp, but can be likened to the 'active defence' we saw from the animal world. A poisonous frog doesn't bother anyone unless they try to eat him. In the examples that follow at the end of this paper we'll demonstrate how passive strike-back can apply the same principle.

Despite this emphasis on 'passive' strike-back is can be dangerous and should only be implemented with the greatest care. This paper explores the concept for research purposes only, legal, moral and ethical questions still need to be examined and readers who choose to implement any of these techniques do so at their own risk.

4.4 Summary

The following points summarize the thinking of this section thus far:

- Strike-back thinking can be implement at almost any layer of the communications stack.
- There are various different kinds of strike-back. These range from simple 'misinformation' all the way through to aggressive Trojan horse attacks that target the attacker's entire host or network.

- Strike-back has to be “passive” to really work. This means that the attacker must himself be fully responsible for the resultant response. Systems that use signatures to identify attacks and then launch a response can possibly be tricked and are therefore too dangerous.
- Passive strike-back requires the attacker himself to “fetch” the strike-back attack. This concept will be demonstrated in the examples section that follows.

5 Examples

In this section we present examples that demonstrate the principles discussed in the previous sections of this paper. Whilst some of these examples could be implemented in practice, they are not considered to be definitive. Source code for all of the programs discussed here can be downloaded from the research portal at the SensePost website, or by mailing research@sensepost.com.

5.1 Striking back at Footprinting

5.1.1 Overview

In this section we look at how one could strike back at an attacker who is using DNS queries to build a footprint of our network. We’ll essentially use the control we have over DNS zones to perform two different attacks:

1. Create noise and confusion via random DNS entries.
2. Attack the tools used to process and display DNS query information

5.1.2 Attack Tools

The attacker will be using DNS queries. There are essentially three to consider, namely:

1. Tools that perform DNS ‘forward’ lookups (from names to IP addresses)
2. Tools that perform DNS ‘reverse’ lookups (from IP addresses to names)
3. Tools that perform DNS zone transfers

The information returned by these tools will have to be sorted, cleaned, stored (perhaps in a database) and eventually displayed for the attacker to use.

5.1.3 Strike-Back Strategy

We can strike-back in numerous ways:

1. A *name daemon* can be configured to allow zone transfers from unauthorized addresses, but to generate a zone that is random and never ending. The utility performing the zone transfer connects and initiates the transfer, but can never terminate because the data never stops coming. Any data that is received is useless or misleading.
2. A *name daemon* can be configured to return IP addresses for any forward lookup query. Names that don’t actually exist are given addresses that reside far away from our own network; possibly at a location the attacker really wouldn’t want to go, like a country’s defence network. An attacker attempting to brute force our DNS zone will receive replies for every query sent. The actual accurate information is also included in there, but is obfuscated by other inaccurate or misleading data.
3. Reverse entries can be obfuscated in the same way, with every reverse DNS entry returning a result. Accurate results are smothered by inaccurate and misleading data, which could possibly even mislead an unwary attacker into attacking a network he can’t afford to tamper with.

4. By shirking proper DNS name conventions on 'fake' entries we can send the attacker data that will cause havoc on the systems that parse and store query data. DNS names could be made to contain shell commands, HTML tags or SQL injection strings that could cripple or confuse the tools that parse the DNS return data. This can be clearly seen in the screen shots that follow

5.1.4 Strike-Back Tools

SensePost made use of a publicly available half-implemented Java based DNS server (*jnamed*) with a few modifications to permit non-RFC compliant results.

5.1.5 Strike-Back in Action

The following screen shot shows the modified *jnamed* in action:

```
[root@wommwom dnsjava-1.6.2]# java jnamed jnamed.conf
jnamed: listening on 0.0.0.0#53
jnamed: running


$ host -al 67.30.196.in-addr.arpa 196.30.67.73 | grep IN
....
1.67.30.196.in-addr.arpa.    86400 IN PTR pokkeld.sensepost.com.
100.67.30.196.in-addr.arpa.  86400 IN PTR haroon.sensepost.com|ls.
200.67.30.196.in-addr.arpa.  86400 IN PTR s.<B>test</b>.sensepost.com.
10.67.30.196.in-addr.arpa.   86400 IN PTR rexacop.sensepost.com.
102.67.30.196.in-addr.arpa.  86400 IN PTR mh.sensepost.com`ls`.
```

Figure 1: Jnamed inserts dangerous content into DNS zone files

Notice the '<B>' HTML tags and Unix command encapsulation (`). This is can be used to strike at the tools that will be used to view the data by the attacker when it is returned.

A simpler but no less effective technique that can be used in this area is to simply permit DNS zone transfers on our domain after creating a zone file that contains several (hundred?) thousand DNS entries. Automated remote discovery tools (like the QualysMap shown below) have no way of separating wheat from chaff and end up chasing red-herrings till their resource limits / boundaries are met.

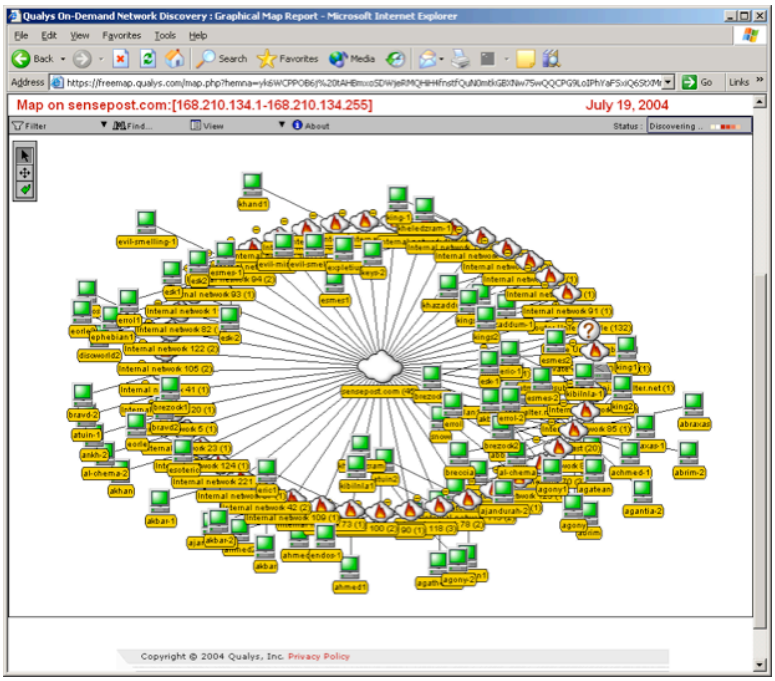


Figure 2: Automated footprinting chokes on endless DNS

Normal DNS forward lookups, reverse lookups on existing IP addresses and zone transfers from legitimate addresses, will all work as normal, thus leaving the legitimate user unaffected.

5.2 Striking back at Network Reconnaissance

5.2.1 Overview

In this section we look at methods for striking back at common network reconnaissance techniques. We use the control we have over all IP traffic originating from our network to perform two different kinds of attack:

1. Misdirect the traceroute tool by sending randomly spoofed ICMP “TTL Expired” messages to any IP addresses performing a traceroute.
2. Mislead ping-scanning tools by sending randomly spoofed ICMP and TCP replies to any address attempting to reach an address in our space that doesn’t exist or should be protected.

5.2.2 Attack Tools

In this example we’re striking back specifically at the traceroute utility and port scanner like Nmap, which can also perform ping scans using ICMP and TCP pings.

5.2.3 Strike-Back Strategy

We can strike-back in two ways:

1. A traceroute is clearly identifiable on the network. When we see traceroute packets entering we begin responding with ICMP “TTL Expired” messages using spoofed source IP addresses (and valid portions of the traceroute probe to prove authenticity to the tracing host). The traceroute utility interprets each of these as a ‘hop’ in the path to the target. We could spoof random addresses or create any ‘path’ that we wish.
2. A network device in promiscuous mode detects incoming requests for machines addresses that don’t exist or should never be reached. Using ‘honeyd’ type technology we respond to any request that’s considered out of band. The response

could either be random or meaningless, or carefully configured to be specifically misleading, as is often done with honey traps.

5.2.4 Strike-Back Tools

SensePost has two simple PERL scripts that implement these two attacks – *whitenoise.pl* and *screwtrace.pl*.

5.2.5 Strike-Back in Action

The screen shot below shows *screwtrace.pl* in action against the *VisualRoute* graphical traceroute utility.

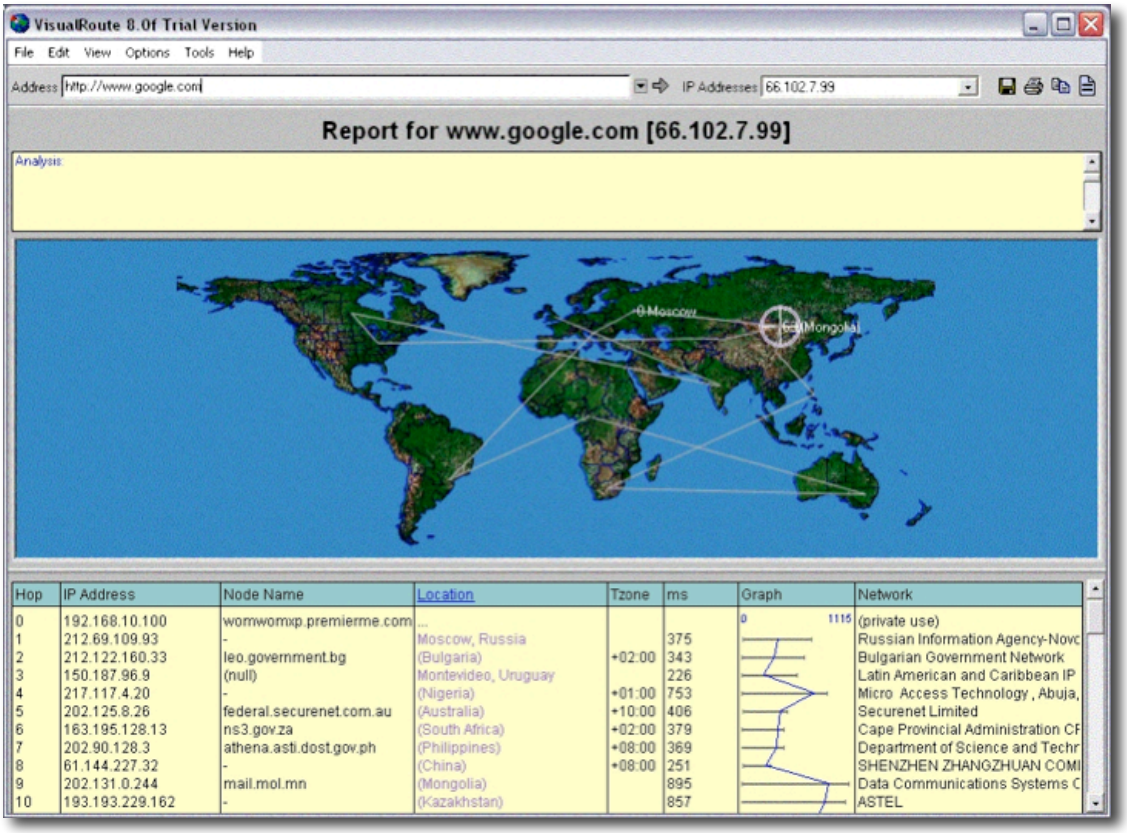


Figure 3: Screwtrace plays with VisualRoute

All the points on the path shown can be selected by us, either to create noise or specifically to be misleading

The next screenshot shows the effect of *whitenoise.pl* on a ping scan by Nmap:

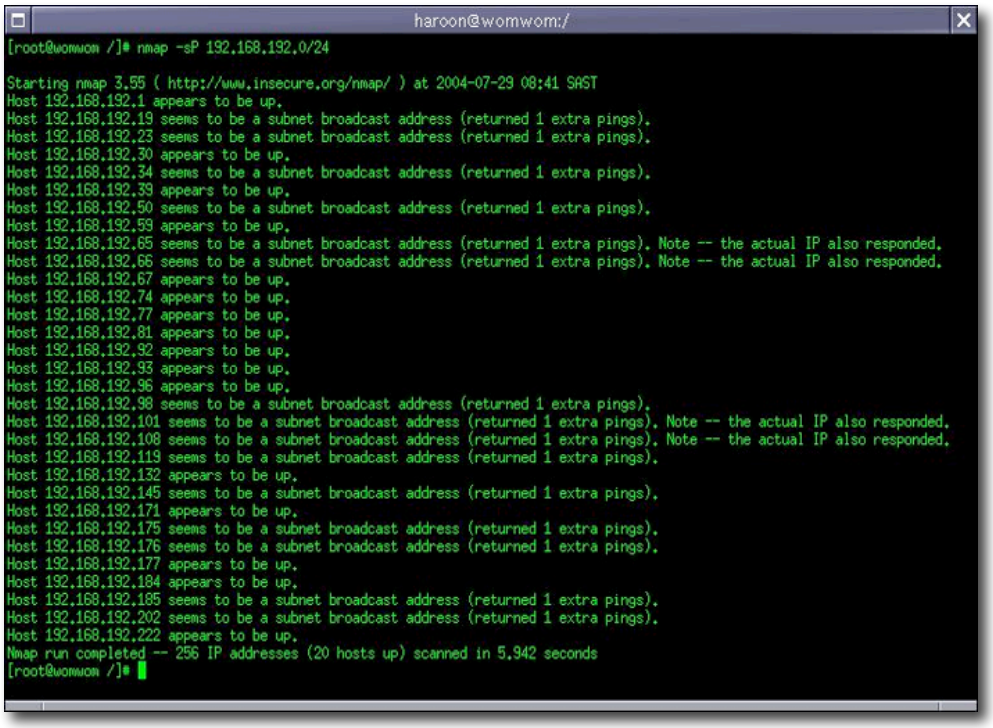


Figure 4: Nmap looses to *whitenoise.pl*

Once again, the data returned is fully under our control and can either just confuse or specifically mislead the attacker.

Legitimate machines remain untouched and legitimate users are unaware of the existence of the strike-back system.

5.3 Striking Back at Vulnerability Scanners

5.3.1 Overview

Most vulnerability scanners work on a simple prompt-and-response basis. They send carefully crafted requests to the target and then analyze the response for signs of the vulnerability being tested for. Many vulnerability scanners will also display the results received, often using HTML. This presents us with various options for strike-back attacks.

5.3.2 Attack Tools

In this example we strike back at web spiders, CGI scanners and vulnerability scanners that test for vulnerabilities. Very few scanners will be impervious to this kind of attack. Scanners that display the results of the scan in HTML format may be especially vulnerable.

5.3.3 Strike-Back Strategy

We can strike-back in four ways:

1. Almost every vulnerability scanner will query the target for service banner information. Sometimes the banner information is all that's required to determine if a service is vulnerable or not. The scanner will often also display the banner it found, sometimes using HTML. Banner information may also be stored in a database for later use. As the banner originates from our servers, *we* control what gets stored and what gets displayed by the attacker's tools. Banners can be modified to contain malicious strings like command encapsulation, command piping, HTML scripting or

SQL Injection. The plan is for these strings to be executed when the attacker views the results of the scan, or writes them to a database.

2. A CGI scanner analyses web servers for the existence of files, scripts or executables with known vulnerabilities. In many cases the scanner has little choice but to rely on the error codes returned by the web server to determine the server's response to a given request. Indeed, many CGI scanners do little more than look for HTTP "200" success codes. As we control the codes returned by our server, these kinds of tools are prime targets for strike-back. By randomly modifying the codes returned for non-existent files or CGI's a scanner can be thrown into total disarray.
3. The first step required to scan a web application is to 'spider' or 'suck' a web site. This works by following each link on a page, copying it to disk and then following each link on those pages recursively until all the pages have been reached. Each stored page can then be analyzed locally for URL parameters or form fields that could possibly be attacked. We can strike back at spiders very simply by sending them a never-ending sequence of recursive links. The spider follows the link, which directs it back to the same link, which directs it back to the same link unendingly. Thus we easily create a tar pit for web spider other similar web application scanning tools.
4. The modern browser is actually a small runtime environment on its own, capable of executing Java, Javascript, Flash and the like. Whilst browsers today are much more secure than they used to be, there's still a lot that can be done to hurt or agitate the user. If we can accurately distinguish an attacker who surfs our site from a legitimate user, we can easily send malicious code to be executed in the browser. There are numerous ways to distinguish attackers from users on web sites:
    a. Check for requests that match known bad signatures.
    b. Send 200-OK messages in response to requests for vulnerable CGIs, like IIS's *showcode.asp*. When the attacker surfs to the CGI to exploit it with his browser we send the malicious code. No legitimate user will be affected.
    c. Send 200-OK messages in response to requests for interesting-sounding directories, like *backup* or *admin*. When the attacker surfs to the location to investigate, we send the malicious content.
    d. Insert invisible HTTP links on the far corners of web pages. Users can't see them, so they'll never click on them. But a spider or a scanner, which reads the HTML source, will.
    e. Hidden fields in HTML forms can be used in a similar way as described above.

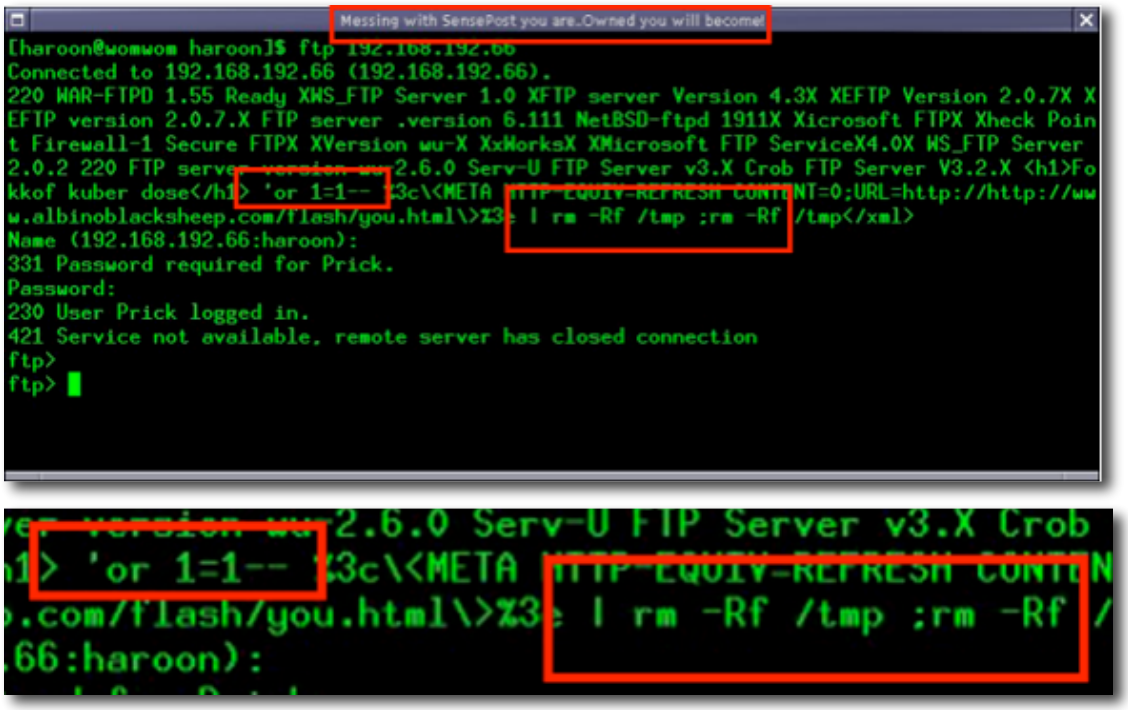
5.3.4 Strike-Back Tools

SensePost has created various tools (e.g. ftp-list.c) that create 'fake' network services with malicious banners. *Spidertrap.pl* is a PERL CGI that simply creates random HTTP links back to itself, thus acting as tar pit for web spiders.

5.3.5 Strike-Back in Action

The screen shot below demonstrates one of the fake banner generators – *ftp-list* – in action. FTP-list pretends to be a legitimate FTP server on any configured IP address. Notice the content of the banner, however, which contains shell commands, HTTP scripting, and SQL Injection strings designed to attack the attackers machine when the data is stored or viewed:

Figure 5: Striking at scanners with evil banners



Banner content could contain control characters also, which allows us to take this kind of attack much further, as we will see in some of the examples that follow. Notice also that we can include banners from any possible FTP server, thus causing the scanner to report huge numbers of false vulnerabilities.

In the next screen shot we see the results of a Nessus scan against a specially configured IIS server. Without any external software the server has been configured to send different 'File Not Found' responses depending on the extension of the file being requested. Even the smarter scanners thus fail to build an accurate '404' signature and report huge numbers of false positives:

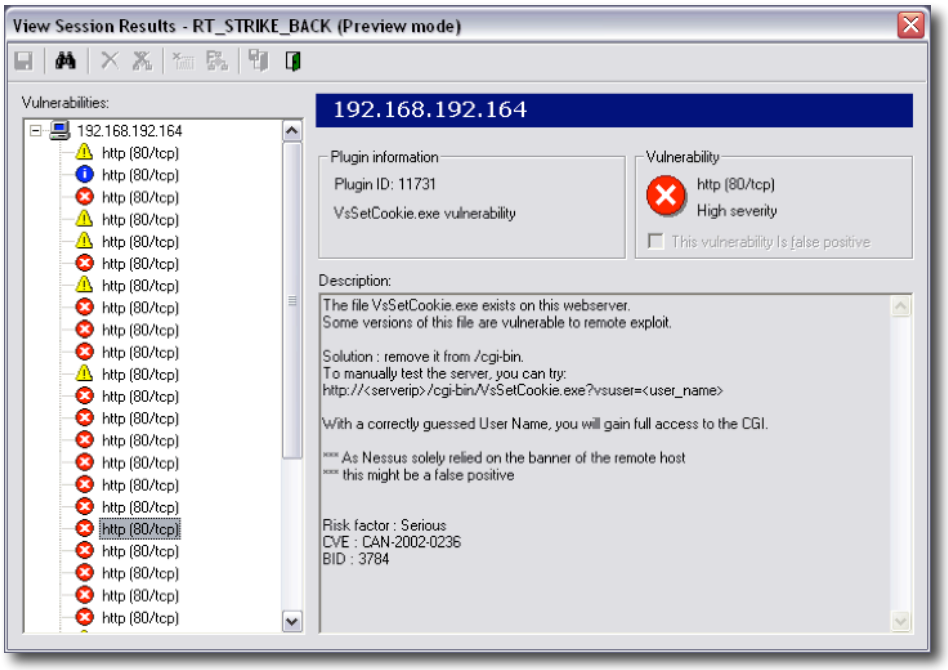


Figure 6: Configuring IIS to tilt Nessus

Of course, this attack would be equally effective against almost any CGI scanner.

The final screen shot shows the output of *spidertrap.pl*. Clicking on any of the links shown would simply create another, similar page with more links back to the same place. Once the spider goes in there, there's no way out again. Humans are protected from the trap because it exists only as an invisible GIF somewhere on the page, visible in the HTML source but impossible for a human to click on:

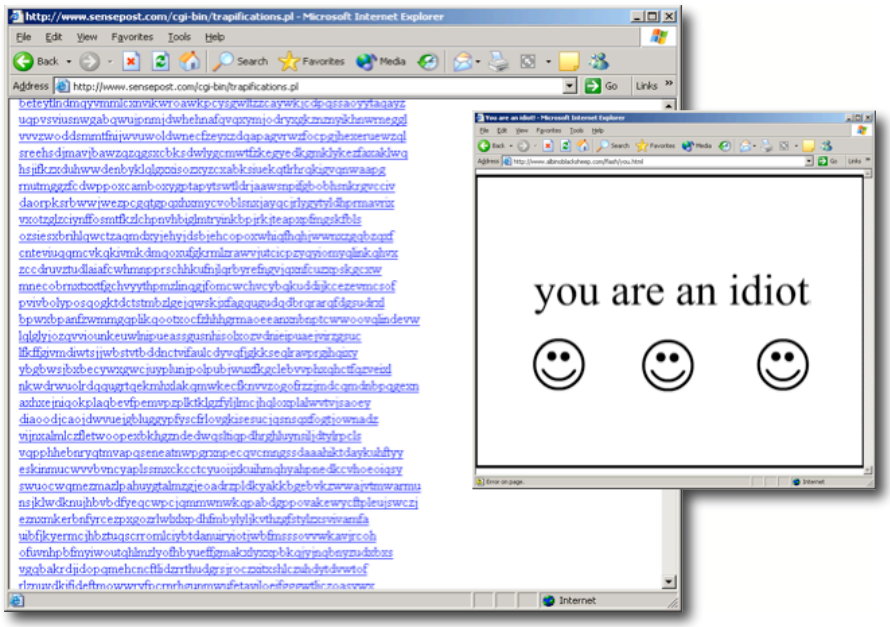


Figure 7: Striking back at CGI Scanners

Also shown in the screen shot above is the Javascript application – "you are an idiot" – that can be used to strike at browser users following the links identified by a CGI scanner. Imagine a scanner reports that there's an 'admin' directory somewhere on a server. There are no links to the directory from elsewhere, so only a scanner performing a brute force search would know of it. Should the attacker choose to surf to the directory with a browser the server sends him the javascript, which opens up thousands of instances of itself, shows a shockwave flash

animation and plays a silly song. Javascript is used to remap the Alt-F4 key, so that attempting to close the window in this manner simply spawns more. In most cases the attacker will be forced to kill his browser or even restart his machine. Examples of this kind of Javascript can be found at http://www.albinoblacksheep.com/you.html. Use with caution.

5.4 Striking back at Exploit Code

5.4.1 Overview

A common approach to exploiting vulnerabilities is to use a malicious payload to open a socket and bind to a shell on the victim server. The attacker can then make a connection to the newly bound shell using a client like telnet or netcat. More sophisticated tools, like the Metasploit Framework, perform both actions at once, executing the exploit and then binding to the shell. Any connection made in this fashion is known be malicious, and is therefore a legitimate target for a strike-back attack. As the attacker is at this stage connected and receiving data from our servers, a strike back attack may well be feasible.

5.4.2 Attack Tools

In this example we look at striking back at attackers who work from a Unix X environment and make connections to one of our servers. Specific attention is given to the Metasploit tools run from an xterm console.

5.4.3 Strike-Back Strategy

This strike back attack occurs in multiple phases. We begin by creating a fake service that pretends to be vulnerable to a known exploit – in the example below we use the IIS 5.0 *.printer* overflow. The attacker finds the 'vulnerable' service and exploits it using Metasploit. He's successful, or so he thinks. Metasploit binds a shell on the web server port and makes a second connection. At this point *we* are in position to send data back to his terminal. This is where things start to get interesting.

For many years Unix hackers have been using xterm escape sequences to set various characteristics of the Unix terminal. Text and background colour can be defined in this way, for example. Thus, if an attacker connects to a network service that *you* control, you could send him such meta sequences to set the colour of his screen, or make it blink, etc. Two such sequences are particularly interesting for strike back, namely, one that can set the terminal title bar, and one that can read from the title bar to the command line.

We therefore have a means of placing text, via the terminal title bar, onto the command line of anyone who connects to our fake server. If we transfer actual Unix commands in this manner, all that's required to have the commands executed. And one <CR> is really not a lot to ask.

5.4.4 Strike-Back Tools

SensePost has written a Java program called 'screwterm' that strikes back at Metasploit by writing to the attacker's terminal.

5.4.5 Strike-Back in Action

We start this example by demonstrating how we can set the title bar of an X terminal when the attacker connects to our service:

Figure 8: Using X meta sequences to play with the terminal

Remember, all that's required at this point is for the attacker to make a TCP connection to our service. At this point we're able to control numerous terminal characteristics.

For example, the string "***ESC] 2 ; ls \a"*** sent down the wire to an xterm should put the letters "ls" in the title bar. The string "***ESC 91 [2 1 t***" will copy the value of the title bar to the actual command prompt, where it will be executed as soon as the user hits enter. Refer to H.D. Moore's paper on terminal security issues for more information (http://www.digitaldefense.net).

By setting up decoy services especially for this purpose, we easily separate malicious from legitimate users.

In the next screen shot you'll see ScrewTerm in action. It starts and binds to port 80.

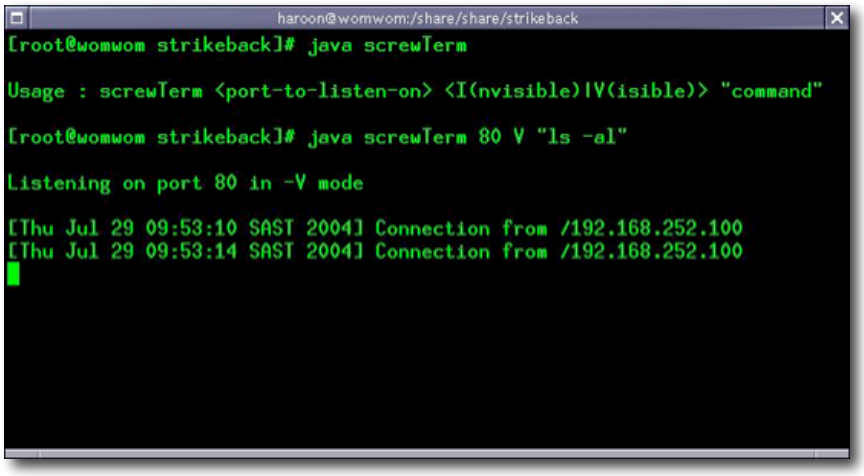


Figure 9: ScrewTerm Ready to Strike Back

Running in 'Visible' mode, screwterm will set the attacker's terminal text colour to something we can see. In a real-life scenario we would run in invisible mode, setting the text colour and the background colour the same, and thus making our attack text invisible to the attacker. Using ScrewTerm we can send any text via the title bar, to the xterm command line, as soon as the connection is established.

In the next screen we'll see the attacker running a Metasploiut exploit from his X terminal against out 'vulnerable' server:

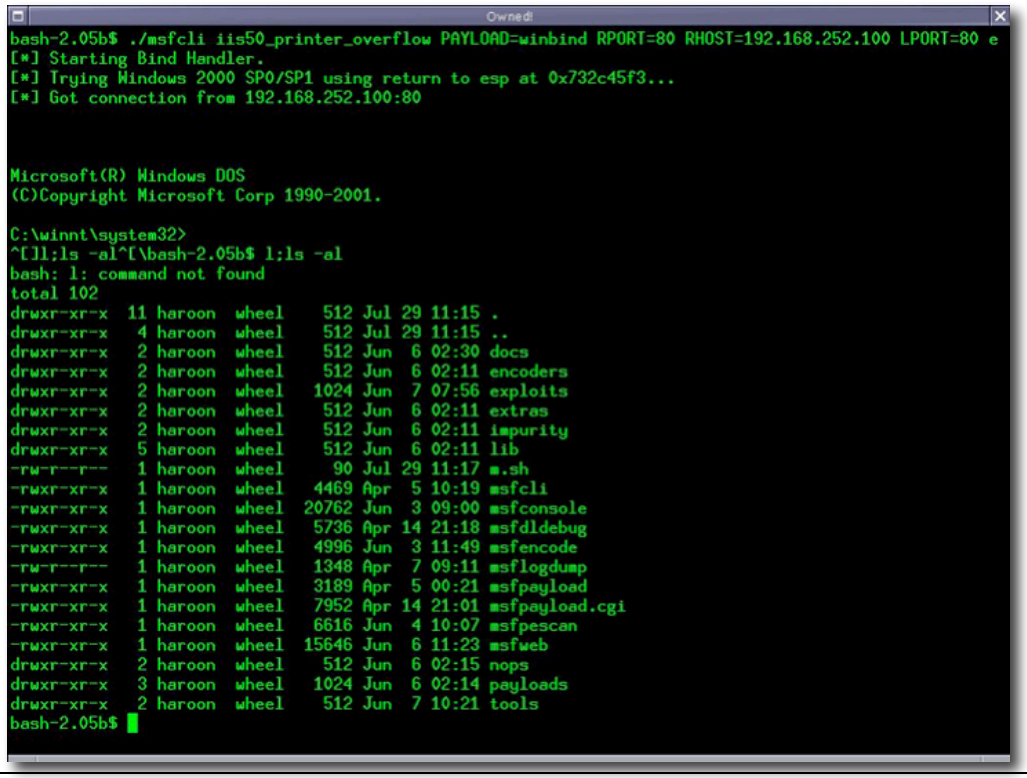


Figure 10: Metasploit Invites Us In

Notice carefully what has happened here:

1. The attacker runs the exploit and establishes a shell on our fake server.
2. The minute the connection is established we send a Unix command to the title bar of the attacker's xterm using known meta sequences.
3. We then copy the command from the title bar to the command line, again using meta sequences. Unlike the example shown above, we would set the text colour to that of the background.
4. We also send a visible copy of a DOS shell prompt, to make the attacker *think* the exploit has succeeded. This text we make visible so that the attacker can see it.
5. Thinking the attack has succeeded, the attacker hits the ENTER key to confirm the connection is established.
6. At this point the invisible Unix commands, invisible but already on the command line, are executed on the attacker's machine and with his privileges.

Once again one has to notice that we can see the command being injected and the output of that command because *ScrewTerm* was run in (V)isible mode and sets the terminal text colour so that it can be seen. In an actual attack *ScrewTerm* would be run in (I)nvisible mode, the terminal text and background colours would be the same and the attacker would see only the fakes Windows command prompt that we sent him. The hidden commands would be executed the minute the attacker saw the Windows shell appear and hit ENTER.

5.5 Striking back Web Application Scanners

5.5.1 Overview

We mentioned earlier that custom web applications have recently become the Internet hacker's target of choice. Analyzing a web application for weaknesses is not trivial, however, and a hacker will deploy various different tools to assist with this task. Web spiders (suckers), interception proxies and scanners are common tools in the attacker's arsenal. Our objective is to strike back by disabling the attacker's tools, rendering them useless and forcing him to do everything by 'hand'. We achieve this through the strategic use of Shockwave Flash applets. Only clients that can interpret Flash are allowed to access the site. This process is completely transparent to a regular user with a browser, but can be a huge obstacle to automated hacking tools like scanners and spiders.

This section will show that by making the attacker to get his hands dirty we not only slow him down, we also position ourselves to strike back at him much more directly.

5.5.2 Attack Tools

Numerous hacking tools emulate the behaviour of a user with a browser, without *actually* being a browser. Web application security scanners, CGI scanners, general vulnerability scanners, web spiders and web application analysis proxies (like @Stake's WebProxy) all operate in this manner and are all potential targets for this kind of strike back.

5.5.3 Strike-Back Strategy

The strategy applied in this type of strike back has been covered in principle already. Essentially the plan is that any new visitor to a protected web site is sent a custom Shockwave Flash applet, which must be executed in order to get a secret session key, which in turn is required to obtain a session cookie, which is required to surf the site. This all sounds a little complex but can be made to happed completely transparently *provided the user's browser can interpret Flash.* As the hacking tools mentioned in the previous section don't do Flash (this is sometimes core to the purpose of the tool) they can never receive the secret key

and therefore never access the site. As Armpit is literally segregates the user form the web site at a network level it becomes a simple case of “no Flash, no visit”.

We therefore refer to Armpit as a “human detector”.

5.5.4 Strike-Back Tools

SensePost has written a proof-of-concept implementation of this concept called ‘Armpit’ (a play on the concept ‘tar pit’). Armpit is functional but is a research project only and is not geared for enterprise-level implementation.

5.5.5 Strike-Back in Action

The ‘Armpit’ Human Detector is a separate network daemon that is installed on the network in front of the web server, typically as part of a firewall. The only way for a client to reach the web server is therefore via the Armpit server. This can be seen from the diagram below:

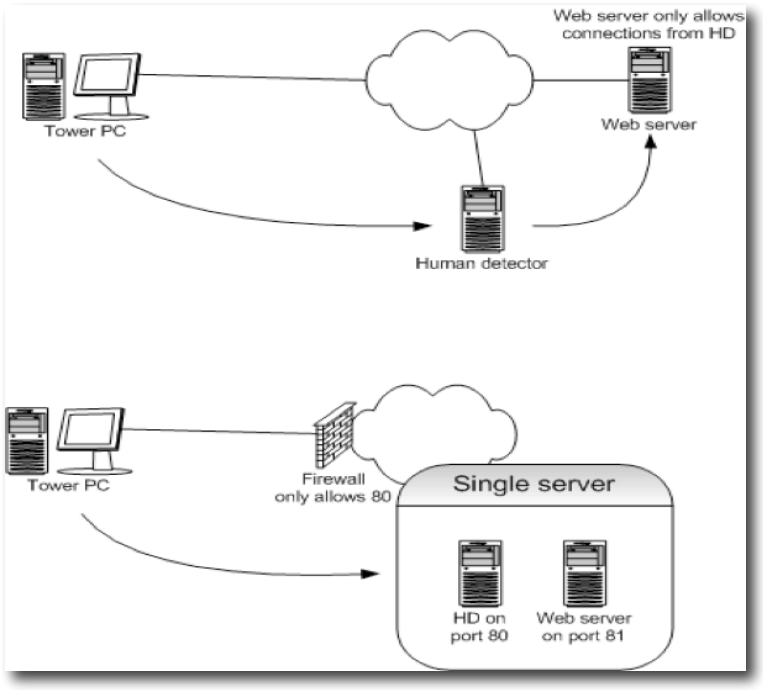


Figure 11: Armpit At A Network Level

As the diagram above depicts, the Armpit daemon could actually also be installed *on* the web server itself, either as separate component or theoretically in the form of a request handler.

Armpit’s primary function is to determine whether a visitor to the site being protected is actually a “human”. As mentioned earlier, it achieves this by sending the visitor a small piece of Shockwave Flash code to be executed. Whilst most current browsers are capable of interpreting Flash, no spiders, scanners, proxies or other assessment tools are. Thus, the Armpit basically requests a secret session code that the client can only get if it successfully executes the Flash. This happens only once, after which standard cookies are used. In this way we’re able to easily differentiate ‘human’ users from automated tools with only minimal additional load at any point.

You can see this process in action in the diagram below:

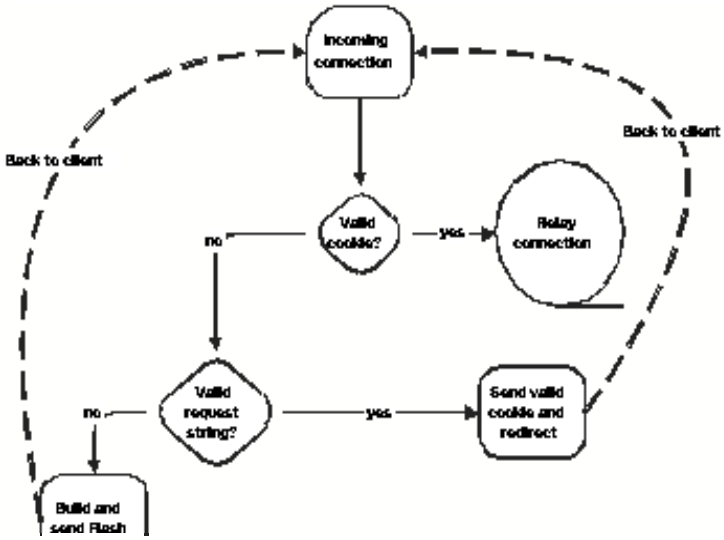


Figure 12: Armpit Basic Logic

Armpit's basic logic can be summarized as follows:

1. The client is directed to the Armpit host IP address by DNS. I.e. The client thinks that the Armpit server *is* the web server. [*http://www.armpit.com*].
2. The Armpit daemon checks the client's cookie. If there is a valid cookie then the HTTP request is forwarded at a network level, in the same manner as with one-to-one NAT.
3. If the client does *not* have a valid cookie, the Armpit daemon dynamically builds a small Shockwave Flash applet, which is sent back to the client to be executed by the browser. [*http://www.armpit.com/reroute.swf*].
4. If the Flash executes properly it simply initiates a new HTTP request to the Armpit server, this time with a unique, cryptographically secure session ID. [*http://www.armpit.com/p=*<unique_secure_id>]. This step is necessary because it convinces us that we're dealing with an *actual* browser that can read and execute Flash, and not a spider, interception proxy or scanner with only basic functionality.
5. If the Armpit server receives a request containing a Flash-generated session ID then the Armpit issues a valid cookie and redirects the client one last time to make a fresh connection.
6. The final request, this time also containing a valid, secure cookie is forwarded at a network level as described in step [2].

At the HTTP level this process can be observed nicely using @Stake's *WebProxy*:

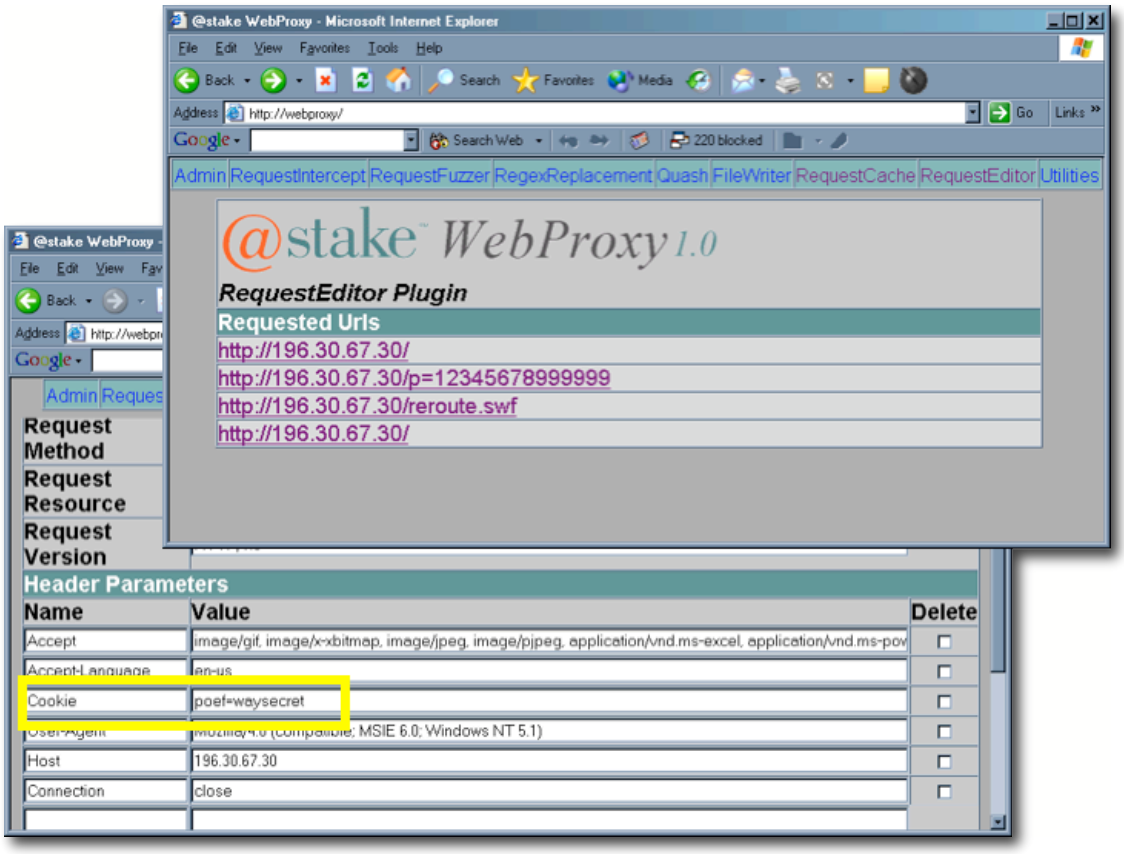


Figure 13: Armpit Human-Detector As Seen By @Stake WebProxy

WebProxy shows the process very nicely: The initial request, the Shockwave Flash (SWF) object, the new request with the unique session ID the final request, this time with a valid cookie. The cookie can be made valid forever, or limited to a fixed number of requests or a finite amount of time.

Combined with a good IDS attack signature database, Armpit can also be made to implement a form of browser "shunning". On detecting malicious activity from any user that user's cookie is black-listed and the user is forced to restart the process – run the flash again re-enter the system to gain a new cookie. This approach is still far from being a firewall, but it should function as a very effective tar pit, as is depicted in the graphic below:

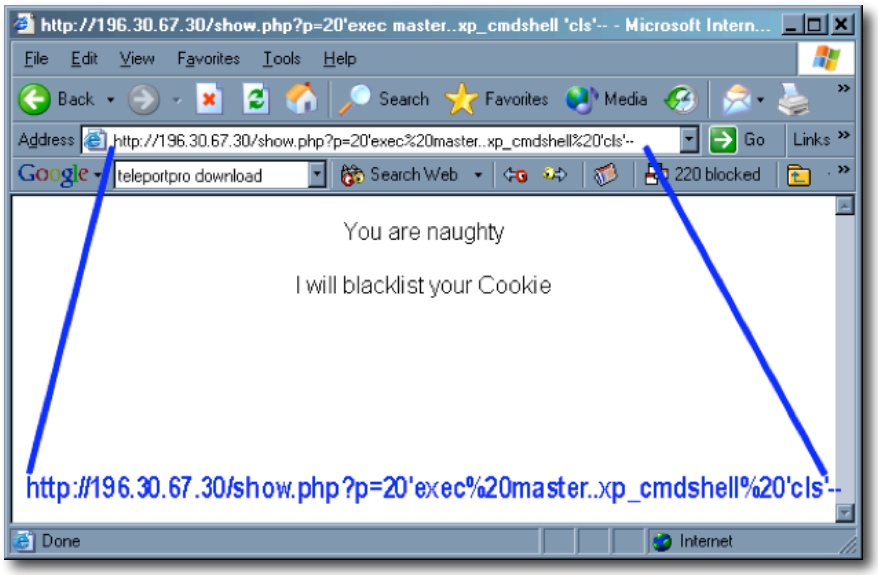
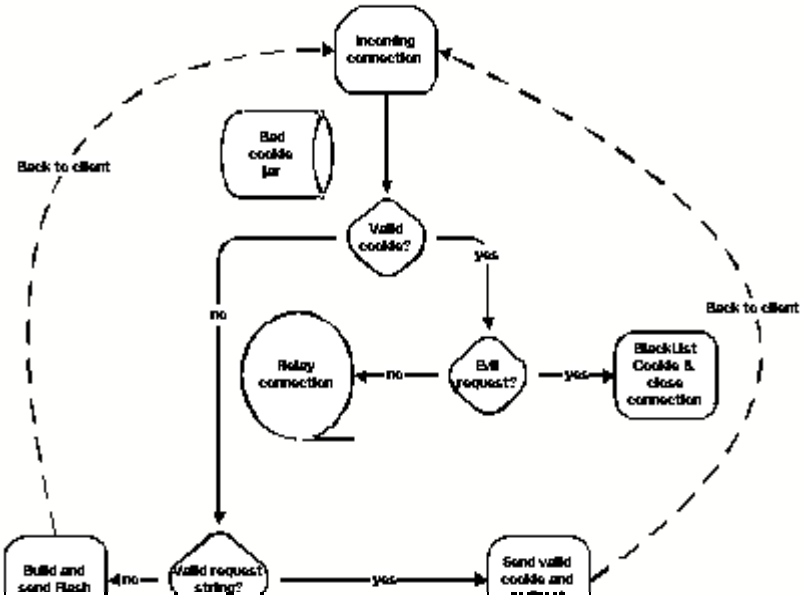


Figure 14: Armpit Boggs Down Malicious Users

More aggressive responses, like the 'You are an idiot' Javascript strike-back we described earlier, could also be used at this point. The logic of the IDS process is depicted below:



Notice the addition of extra logical gate that detects the malicious request and sends the attacker's cookie to the sin-bin.

5.6 Summary

In this section we provided the following examples of strike back at various levels:

Name	Description	Level	Purpose
Jnamed	A fully functional but non-compliant DNS name daemon	Network	1. Create noise & confusion 2. Slow down attacker tools 3. Attack the attacker's host or network
WhiteNoise.pl	Creates random responses to ICMP and TCP ping requests	Network	Create noise and confusion
ScrewTrace.pl	Messes with traceroute utilities by sending ICMP "TTL Expired" messages with spoofed IP source addresses whenever a traceroute is detected	Network	Create noise and confusion
ftp-list.c	Creates fake FTP services on the network with misleading or even malicious banners	Network Application	1. Confuse vulnerability scanners 2. Attack the attackers own host and network

SpiderTrap.pl	Creates random HTTP links back to itself, thus acting as tar pit for web spiders	Network Application	Slows down or kills automated attack tools.
ScrewTerm	Strikes back at attackers running exploits from a Unix xterm.	Network Application	Attack the attackers own host and network
Armpit	Acts as a "human detector" preventing an attacker from using automated tools to analyze your site.	Application Level	1. Slow down or cripple automated attack tools. 2. Attack the attackers own host.

6 Conclusion

In this paper we've discussed the potential of "passive strike-back". Passive strike-back is an Internet defence strategy that focuses on 'raising the bar' for an attacker, making the attack process risky and expensive and thereby discouraging attacks on your network. This paper demonstrates that not only passive strike-back technically feasible; it is also ethically and strategically justifiable. Full, enterprise-level implementation of the concepts described here is left to the experts in that field.

---oo()oo--