**Penetration Testing with dsniff**
Christopher R. Russel
February 18, 2001

**What is dsniff?**

The ability to access the raw packets on a network interface (known as network sniffing), has long been an important tool for system and network administrators. For debugging purposes it is often helpful to look at the network traffic down to the wire level to see exactly what is being transmitted. Dsniff, as the name implies, is a network sniffer - but designed for testing of a different sort. Written by hacker Dug Song, dsniff is a package of utilities that includes code to parse many different application protocols and extract interesting information, such as usernames and passwords, web pages being visited, contents of email, and more. Additionally, it can be used to defeat the normal behaviour of switched networks and cause network traffic from other hosts on the same network segment to be visible, not just traffic involving the host dsniff is running on.

These capabilities alone are enough to attract interest, but in its latest version (2.3 as of this writing, released late December 2000), it includes new programs to launch man-in-the-middle attacks on the SSH and HTTPS protocols, which would allow viewing of the traffic unencrypted, and even the possibility of taking over interactive SSH sessions. These new abilities have caused some uproar in the security community, however this document is not an ethical or technical debate; I only hope to show dsniff in action to help people understand its capabilities and therefore be better able to see weaknesses in their own network.

**The dsniff Package**

To give you an idea of exactly what dsniff can be used for, here is a list of tools included in the dsniff package, and a brief description of their function, partially taken from the dsniff README:

| | |
|---|---|
| **arpspoof** | redirects packets on a LAN to defeat the host-isolating behaviour of the switch. |
| **dnsspoof** | forges replies to DNS queries. |
| **dsniff** | password sniffer with ability to handle FTP, Telnet, SMTP, HTTP, POP, poppas, NNTP, IMAP, SNMP, LDAP, Rlogin, RIP, OSPF, PPTP MS-CHAP, NFS, VRRP, YP/NIS, SOCKS, X11, CVS, IRC, AIM, ICQ, Napster, PostgreSQL, Meeting Maker, Citrix ICA, Symantec pcAnywhere, NAI Sniffer, Microsoft SMB, Oracle SQL*Net, Sybase and Microsoft SQL authentication info. |
| **filesnarf** | saves files sniffed from NFS traffic. |
| **macof** | causes LAN switch to fail-open (ie. Act as a hub and broadcast traffic to all hosts). |
| **mailsnarf** | saves email messages sniffed from SMTP and POP traffic. |
| **msgsnarf** | saves messages and chat sessions sniffed from most Instant Messenger protocols and IRC. |
| **tcpkill** | kills specified in-progress TCP connections. |
| **tcpnice** | slows down specified TCP connections. |

| | |
|---|---|
| **urlsnarf** | reports URLs sniffed from HTTP traffic. |
| **webspy** | sends sniffed URLs to your local Netscape browser, allowing you to browse in real-time along with the target. |
| **sshmitm** | proxies and sniffs SSH traffic redirected by dnsspoof, captures password logins and optionally allows hijacking interactive sessions. |
| **webmitm** | proxies and sniffs HTTP/HTTPS traffic redirected by dnsspoof, capturing SSL-encrypted logins and form submissions. |

**Penetration Testing**

Dsniff's author refers to it as "a collection of tools for network auditing and penetration testing", and it is from this perspective that I'll demonstrate putting it to use.

Penetration testing is a form of security auditing, where you attempt to break in to a network - with the consent of those in charge of the network – trying to find weaknesses and vulnerabilities you were not aware of before, which hopefully leads to fixing those security holes. In my tests with dsniff, I used my own private home LAN. Unfortunately I do not have a switch at my disposal, so some of the switch-defeating mechanisms were not tested. However, I have seen some of these techniques used successfully for other purposes (such as clustering) and I have no doubt they work unless extraordinary precautions and limitations are enforced in the design of the network architecture.

WARNING: If you plan on testing dsniff on a network you do not own, be sure you have written permission from appropriate authorities, as password and message capture are extremely sensitive topics and probably illegal without precise guidelines.

In the following sections, we will see how dsniff can be used in several different attack scenarios. It is worth noting that although most of the techniques used by dsniff are not new, the fact that they are contained in a single easy-to-use package increases concern since it brings previously complex and difficult attack scenarios within the capability of the average script kid.

**Attack Preliminaries: Accessing the Target Network Traffic**

Before dsniff can be used on a LAN, naturally you first need root or administrator access to a host connected to the LAN. The dsniff toolkit is known to run on Linux, OpenBSD, FreeBSD, and Solaris. It is likely to be possible to port it to other UNIX platforms as well. There is even a version of dsniff for Windows NT, although it is older and lacks some of the features in version 2.3.

Once dsniff is compiled and ready on the attacking host within the LAN, there are three different possibilities for accessing the network traffic:

1. The LAN uses a hub: In this case, nothing extra needs to be done, all network traffic going to any host on the LAN is visible to all other hosts on the LAN.

2. The LAN uses a switch: In the switched architecture, all hosts are connected to the switch on their own isolated port, and the switch keeps track of which host is on which port, and then only sends traffic intended for that host to its port. However, this is mainly intended to increase performance (since each host gets a dedicated connection instead of being shared like with a hub), and the security benefit is mostly a side effect. Therefore, it is easy to defeat by simply confusing the switch so it cannot be sure which host is on which port. Most switches respond to this condition by "failing-open", which means it begins to act like a hub and send all traffic to all ports. This will also cause network performance to suffer greatly, which might be noticed by others on the network. To accomplish this:

- Use macof to flood the network with random MAC addresses, which will overflow the internal translation table of the switch. Without an accurate translation table, the switch can no longer know which host is on which port, which leaves the switch with no choice but to forward all packets to all ports (like a hub would do). This is known as "fail-open". If the switch does not fail-open, the only other option is to "fail-closed" in which case no network traffic will get through to any host on the LAN at all. Although this latter case is not an option for most equipment, it would be an especially easy denial-of-service attack.

Note: Most switches can be configured to allow only one, fixed, MAC address per port – In that case, macof would have no effect, but it is a difficult arrangement to manage.

3. The LAN uses a switch and you wish to target a specific host: If you do not want to sniff all the traffic on the LAN but only wish to target a specific host on it, you can leave the switch alone and just confuse that specific host into thinking you are the gateway/router. Therefore any traffic that host wishes to send to outside the LAN will go to your host first. This is the most clandestine form of sniffing, since it only affects the target host, and only in the way you specify, therefore it is not likely to be noticed by others. To implement this:

- Since we are targeting a particular host and not the entire LAN, we only need to cause the target host to misdirect its network packets to our MITM attack host (on the LAN). To accomplish this, we use arpspoof to send fake ARP packets to the target host, telling it that the MITM attack host is the gateway. That way, any traffic it tries to send outside the LAN will actually go to the MITM attack host. Before doing this, we have to tell the attack host to forward packets on to the real gateway otherwise it will be noticed quickly that the target host can no longer communicate outside the LAN.

**Attack One: Password Sniffing**

Once the network traffic from the target host, or the entire LAN, has been made available to the sniffing host using the techniques above, it is a simple matter to start the dsniff program, which will then automatically detect and capture interesting information from the traffic. In fact, even a normal sniffer will capture the traffic just as well, but normal sniffers do not have the built-in knowledge of various application protocols that dsniff has.

**Attack Two: Message and File Capture**

Using msgsnarf and filesnarf, capturing email messages and files transferred via NFS are just as easy as capturing passwords. Both these programs do the work of interpreting the relevant network traffic to get results. msgsnarf saves the captured email in an mbox-format file for standard UNIX mail-readers to use. filesnarf saves the captured files in the directory it is run from.

**Attack Three: URL capture**

urlsnarf is similar to the other tools mentioned so far, but specifically written to capture HTTP requests and log the URL being referenced. webspy is along the same lines, but instead of logging the URL, it actually commands your local Netscape web browser to follow the URLs as they are sniffed, allowing you to surf along with the target in real-time. As the README says, this is "a fun party trick."

**Attack Four: Man-in-the-Middle**

All the previous attacks are considered to be passive since they do not involve altering the behaviour of the systems being targeted (causing the switch to fail-open can be seen as an active attack on the switch, but the network traffic is merely observed, not intercepted or modified en route). On the other hand, a Man-in-the-Middle attack is an active attack since the attacking host plays an important role managing the network traffic between the source and destination targets.

A MITM attack is when the target host is fooled into thinking it is connecting to a desired destination host when in fact it is connecting to the attacker host, which then handles the connection to the desired destination host and proxies traffic between the two from that point on. In this way, the attacker host now completely controls the connection and can view and/or modify information passing between the connection it has forged with the source and destination hosts.

This type of attack is particularly effective when dealing with connections encrypted with public-key cryptography. If you are not familiar with this subject, one good overview is at the Netscape developer site. Public-key cryptography

is an extremely effective encryption concept, but it does require that connecting host have a copy of the public key from the host being connected to. If the connecting host does not already have the public key from a previous connection with that host then it will have to get it from somewhere – with protocols such as SSH, the destination host will supply its public key itself. MITM attacks take advantage of this by intercepting the initial connection attempt and substituting their own "forged" public key (which they have the private key for and can therefore decrypt the data). If the user at the connecting end has never seen the correct public key before, it is likely the forgery will not be noticed and the attack will be a success. Even in the case where the user does have the correct public key to compare with it, often it just results in a small warning being printed saying the key has changed and if they want to continue connecting. Most users will simply click OK without another thought. This is not the fault of the protocol. This is a problem with user education and also with implementations that print simple warnings when an event as serious as a key change occurs.

**Setting up the Attack**

Executing the MITM attack is more complex than the others, but dsniff's tools make it almost as easy. In this scenario there are three systems (plus the gateway) involved in our sample network:

**Client Host:** Windows, with PuTTY installed (a free Windows SSH program)

```
IP=192.168.1.77 (MAC address irrelevant)
```

**MITM Attacker: (running dsniff)**

```
MAC=08-00-BA-DD-FE-ED, IP=192.168.1.210
```

**Destination:** MAC address irrelevant, IP is outside 192.168.1.x

```
Hostname = "external.host.com"
```

**Gateway (and DNS server):** Serves 192.168.1.x LAN

```
MAC=08-00-DE-AD-BE-EF, IP=192.168.1.1
```

1. We have to fool the client (target) host into thinking the MITM host is the gateway. This is done using arpspoof. At the start, you can see the client has the correct ARP entry for the gateway:

```
[Client] C:\WINDOWS>arp -a
Interface: 192.168.1.77 on Interface 0x2000003
Internet Address Physical Address Type
192.168.1.1 08-00-de-ad-be-ef dynamic
```

Running arpspoof on the MITM host will change that. We must remember to enable ip-forwarding so the MITM host will hand off the packets to the real gateway. To minimize the visibility of this attack, we will specifically target the client system:

```
[MITM Host] % Arpspoof –t 192.168.1.77 192.168.1.1
```

Now we can check the client's ARP table again to see if it worked:

```
[Client] C:\WINDOWS>arp -a
Interface: 192.168.1.77 on Interface 0x2000003
Internet Address Physical Address Type
192.168.1.1 08-00-ba-dd-fe-ed dynamic
```

You can see that this is the MAC address of the MITM host and all traffic destined for the gateway from the client will go to the MITM host instead. This phase of the attack is complete.

Next we need to look for DNS queries in the traffic, and when we find one for a destination host we want to impersonate, we respond to it ourselves instead of passing it on to the real DNS server. We reply with the IP address of the MITM attack host instead of the real IP. We use dnsspoof to respond to any request for hostname "external.host.com", which has an IP-address outside the 192.168.1.x network. We can be very specific by creating an /etc/hosts-format file specifying which hostnames to fake, but by default dnsspoof will just fake ALL hostname lookups which is fine for our example:

```
[MITM Host] % dnsspoof
```

Now to check that it is working on the client:

```
[Client] C:\WINDOWS>ping external.host.com
Pinging external.host.com [192.168.1.210] with 32 bytes of data:

…
```

It is working, external.host.com is reported to have the IP of the MITM host.

The Target host will then begin the connection to the hostname the user specified, but to the IP address WE specified (192.168.1.210), which means the MITM host. If the application being used is SSH or a web browser using HTTPS, we can use sshmitm or webmitm to handle both those protocols (incidentally, sshmitm only supports the SSH1 protocol, not the newer SSH2 protocol, although it is still subject to the same trust requirement for the key exchange just as any public-key encryption is). sshmitm or webmitm sends a forged public-key to the target host, and then also begins a connection of its own to the real destination server. Once this is done, we actually have two connetions; one going from the target/victim host à MITM host, and another going from the MITM host à real destination host. At this point the MITM host can proxy the traffic between the two hosts, decrypting and re-encrypting all traffic in the process, all the while monitoring the decrypted traffic. Additionally, since they are two independent connections, we can always terminate the target hosts' connection and simply take over the interactive session from the MITM à destination host ourselves.

To verify this is working, we try connecting to example.host.com using PuTTY on the client host. This is a host we've connected to before, so we already have a copy of the real public key. What we get is a warning message:

WARNING – POTENTIAL SECURITY BREACH!

The server's host key does not match the one PuTTY has stored in the registry

…

Then the user has three options: Yes, No, and Cancel. Yes and No both continue the connection (Yes updates the stored key, No does not), and only Cancel stops the connection. Yes is the default choice if the user just hits enter without reading or thinking.

At least PuTTY is good about warning the user in clear terms. Some other clients such as SecureCRT (a commercial client) do not explain what the implications of the warning are at all and simply ask the user if they wish to continue connecting (defaulting to yes). UNIX SSH clients can be configured for "strict" host key checking which will automatically disallow any connecting to a host who's host key has changed. This forces the user to manually remove the existing known-host key if they are sure there is no security problem.

HTTPS traffic can be attacked in a similar way, using webmitm instead of sshmitm.

**Summary**

It is hard to study dsniff without being at least slightly troubled by the ease at which you can gather passwords, emails, files, and eavesdrop on encrypted connections, even on switched networks. This paper has demonstrated some of these cases and hopefully provides some of the perspective on these problems that real penetration testing

provides. Of course, these vulnerabilities are not the fault of dsniff or its author and if dsniff causes a greater concern for the basic security of network architectures, application protocol design, and public-key infrastructure then it will have done a service for us all.

**References:**


Drury, Jason. "Sniffers: What they are and How to Protect From Them." 11 November 2000.
http://www.sans.org/infosecFAQ/switchednet/sniffers.htm

Song, Dug. "dsniff."
http://www.monkey.org/~dugsong/dsniff/

Seifreid, Kurt. "The End of SSH and SSL? Follow-up." 22 December 2000.
http://www.securityportal.com/seifried/sslssh-followup20001222.html

Song, Dug. "dsniff Frequently Asked Questions."
http://www.monkey.org/~dugsong/dsniff/faq.html

Saso. "Re: Cisco Catalyst switches." 13 June 2000.
http://archives.neohapsis.com/archives/vuln-dev/2000-q2/0916.html

Netscape Communications, Inc. "Introduction to Public-Key Cryptography."
http://developer.netscape.com/docs/manuals/security/pkin/contents.htm

Tatham, Simon. "PuTTY: A Free Win32 Telnet/SSH Client."
http://www.chiark.greenend.org.uk/~sgtatham/putty/