Hacking Without Re-Inventing the Wheel

Nitesh Dhanjani
[nitesh at dhanjani dot com]
&
Justin Clarke
[blackhat at justinclarke dot com]

Blackhat, Las Vegas 2004

Objectives

- Custom Vulnerability Checks
- Why Open Source tools rock
- Why Re-invent the Wheel?
- Nessus Architecture.
- Using Nessus.
- Nessus Attack Scripting Language (NASL).
- Writing Nessus Plug-ins.
- Learn to modify Hydra.
- Writing service signatures for Nmap.

The Need for Custom Vulnerability Checks

- Plethora of home-grown applications and services implemented everyday.
- Custom applications and services are also susceptible to local and remote vulnerabilities.
- Patches for home-grown applications and services need to be implemented locally. But, first, they must be detected!

Closed Source Tools

- For ‘vanilla’ A&P needs, out of the box tools do their job well.
- Well, almost. Difficult to study _how_ tools work if you can’t look inside the hood (code).
- Cannot tweak closed source tools to suit your needs (in most cases).
- ‘Custom’ vulnerabilities or protocols? Good luck!

Closed Source Tools

- Closed source tools cannot be easily extended by end users to scan for custom vulnerabilities or protocols.
- Vendors sometimes do not release updates to their scanners that allow you to scan for vulnerabilities that may effect you.
- Costly.

Closed Source Tools

- This talk is NOT an attack on closed sourced software.
- Closed source tools do have many advantages (example: support, legal, quality (?)). But these are out of scope for this talk ;-)

Open Source A&P Tools

- Don’t understand it? Look at the code.
- Don’t like it? Modify it.
- Use existing code for your unique scanning needs.
- ‘Extend’ exiting tools.

Why Re-invent the Wheel?

Make use of open source enumeration and scanning tools:

- Nessus: Write NASL plug-ins to scan for vulnerabilities in your application.
- Nmap : Alter Nmap's nmap-service-probes database to detect unique services running on alternate ports.
- Hydra : Develop Hydra modules to brute-force your applications and services for weak passwords.
- [and many others we don’t have time to cover]

Introduction to Nessus

- Open Source vulnerability scanner and framework.
- Client / Server architecture.
- Write plug-ins using Nessus Attack Scripting Lanaguge (NASL).

Installing Nessus

[notroot]$**lynx -source http://install.nessus.org | sh**

OR

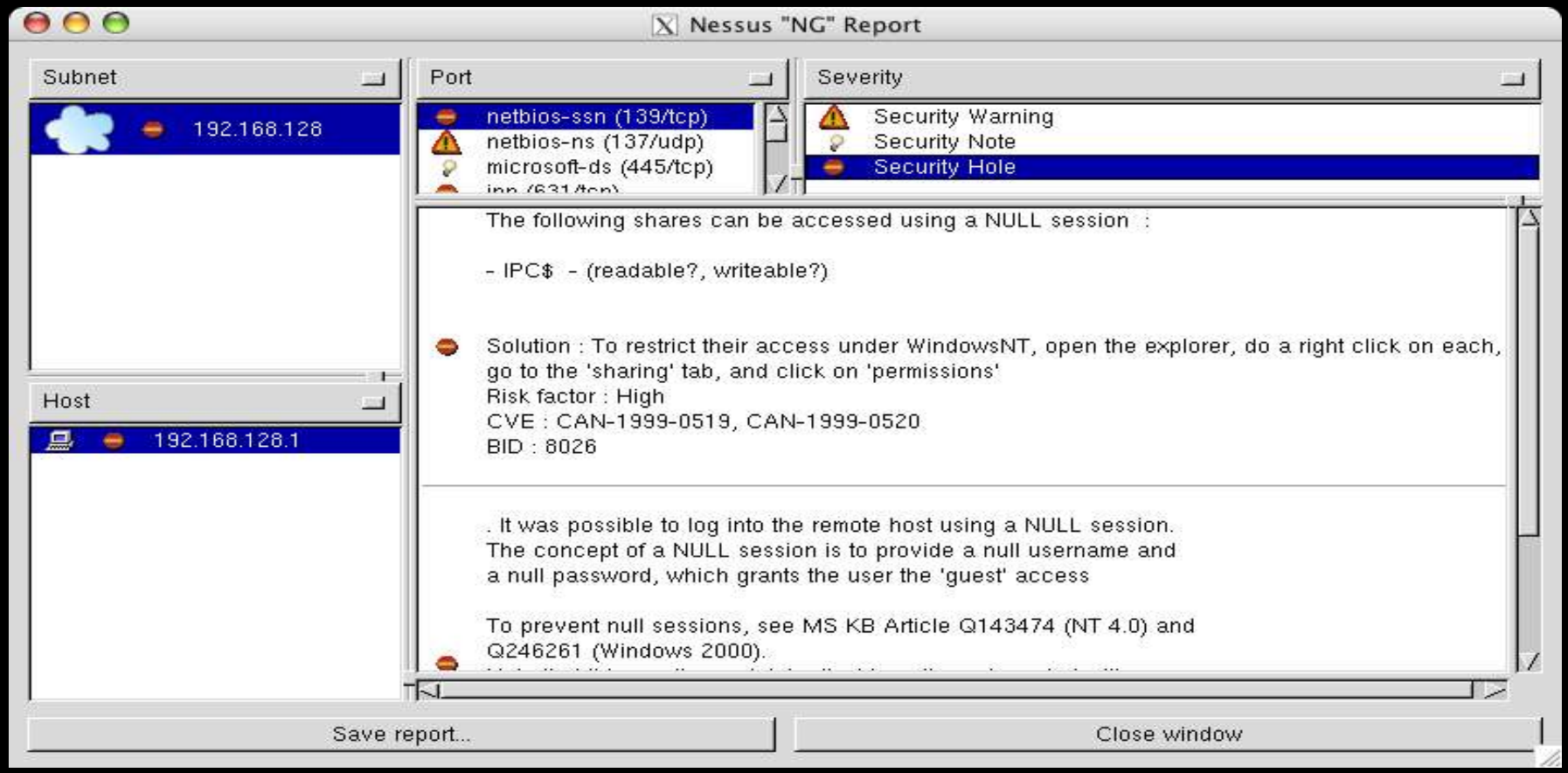
Install the following by hand (tar xvf, ./configure, make, make install):

- nessus-libraries
- libnasl
- nessus-core

Using Nessus

- Remotely connect to the Nessus server using the Nessus client.
- Configure Scans
  - Filter & Select Plug-ins by “Family”: Backdoors, CGI abuses, Denial of Service, Gain root remotely, e.t.c.
  - Other options: TCP scanning technique, port ranges, “Safe Checks”, target hosts, e.t.c
- “Start the Scan”

Nessus Report



Nessus Report

- Plug-in authors are responsible for categorizing the findings:
  - Security Note: Misc. issues.
    - Example: popserver_detect.nasl
  - Security Warning: Mild flaw
    - Example: ftp_anonymous.nasl
  - Security Hole: Severe flaw
    - Example: test-cgi.nasl

The nasl interpreter

[notroot]$ **nasl –v**

nasl 2.0.10

Copyright (C) 1999 - 2003 Renaud Deraison
<deraison@cvs.nessus.org>
Copyright (C) 2002 - 2003 Michel Arboi <
arboi@noos.fr>

See the license for details

/usr/local/lib/nessus/plugins/

[notroot]$ **ls /usr/local/lib/nessus/plugins/cgi*.nasl | more**

/usr/local/lib/nessus/plugins/cgibin_browsable.nasl

/usr/local/lib/nessus/plugins/cgibin_in_kb.nasl

/usr/local/lib/nessus/plugins/cgicso_command_execution.nasl

/usr/local/lib/nessus/plugins/cgicso_cross_site_scripting.nasl

/usr/local/lib/nessus/plugins/cgiforum.nasl

…

…

nasl Usage

```
Usage : nasl [-vh] [-p] [ -t target ] [-T trace_file] script_file
        -h : shows this help screen
        -p : parse only - do not execute the script
        -t target : Execute the scripts against the target(s)
                    host
        -T file : Trace actions into the file (or '-' for stderr)
        -s : specifies that the script should be run with 'safe
                    checks' enabled
        -v : shows the version number
```

Executing .nasl scripts

[notroot]$ **nasl -t 192.168.1.1 finger.nasl**

The 'finger' service provides useful information to attackers,

since it allows them to gain usernames, check if a machine

is being used, and so on...

Here is the output we obtained for 'root' :

Login: root Name: System Administrator

Directory: /var/root Shell: /bin/sh

On since Wed 5 May 08:51 on ttyp2 from localhost:0.0

[SNIP]

Solution : comment out the 'finger' line in /etc/inetd.conf

Hello World

```
--helloworld.nasl BEGIN
display(“Hello World\n”);
--END
```

[notroot]$ **nasl ./helloworld.nasl**

Hello World

Data Types

- Integers

  Examples: 11, 0x1B (27)

- Strings

  Examples: “I love NASL”

Arrays

```
myarray=make_list(1,"two");
display("The value of the first item is ",
   myarray[0]," \n");
display("The value of the second item is",myarray
   [1]," \n");
```

Hashes

- Elements in a Hash have a ‘key’ associated with them.

```
myports=make_array('telnet',23,'http',80);
```

- myports[‘telnet’] will evaluate to 23
- myports[‘http’] will evaluate to 80

Loops

- for
- foreach
- repeat…until
- while

Functions

```
function is_even (port)
{
    return (!(port%2));
}
my_port=22;
display (myport,” is “);
if(is_even(port:i))
            display ("even!");
   else
            display ("odd!");
display ("\n")
```

Knowledge Base

- Shared memory space.
- Allows plug-ins to communicate with each other.
- Set a KB item:

  set_kb_item(name:”SSL-Enabled”,value:TRUE);
- Get a KB item:

  value = get_kb_item(name:”SSL-Enabled”);
- Get multiple KB items:

  tcp_ports = get_kb_list(“Ports/tcp/*”);

Example Vulnerability

- Web application serves /src/passwd.inc
- This file contains usernames and passwords (hashes)
- Our plug-in will scan for this vulnerability, and report it as a Security Hole (severe)

Writing Plug-ins

1. Provide description: Category, Version, Author information (you), Required ports, Description of vulnerability.
2. Test for vulnerability
3. Report vulnerability

if (description)

- The value of description is set to TRUE when the .nasl plug-in is executed by the Nessus server.
- The first statement of a .nasl plug-in should test for description, and provide plug-in details when set to TRUE.

Important description Functions

- Unique script ID: script_id(99999);
- Version: script_version(“$Revision: 1.00$”);
- Name: script_name(english:"Checks for /src/passwd.inc");
- Description: script_description(english:desc [“english”]);
- Required ports: script_require_ports (“Services/www”,80);

Important description Functions

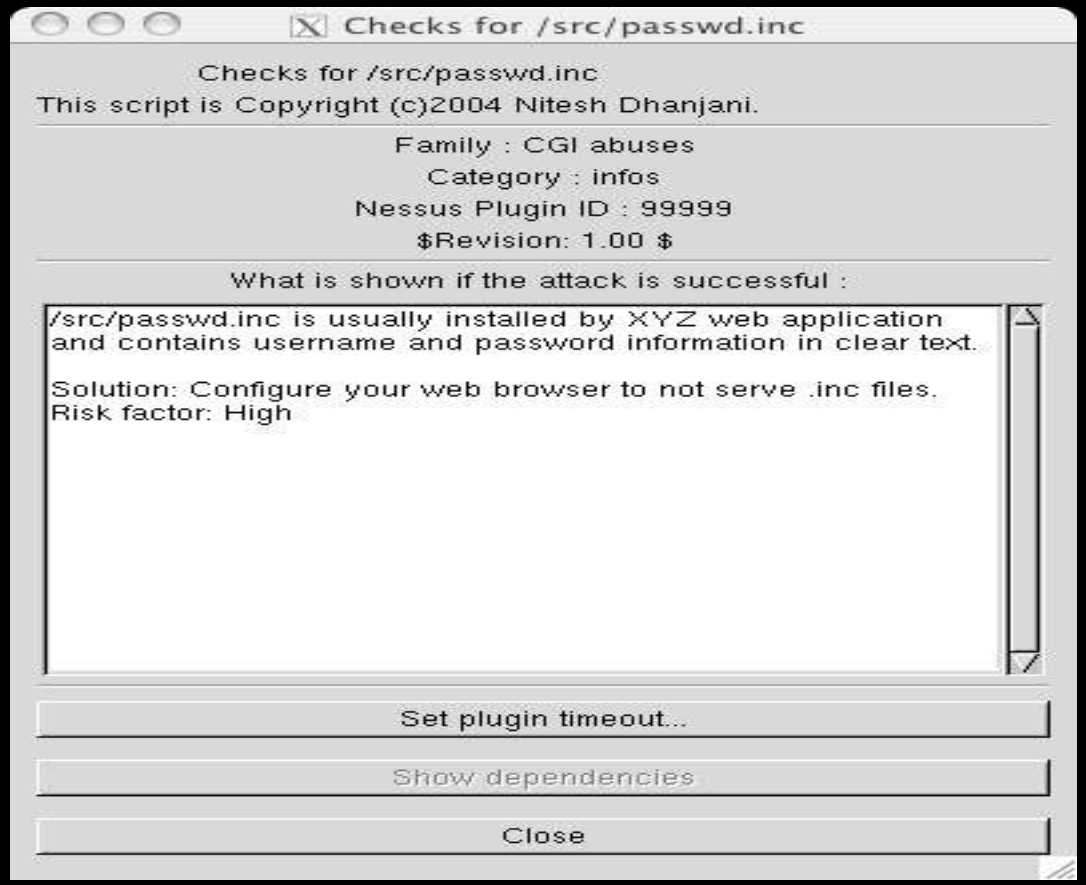
- Category: script_category (ACT_GATHER_INFO)
- Other categories:

  ACT_ATTACK, ACT_DENIAL, ACT_DESTRUCTIVE_ATTACK, ACT_KILL_HOST, ACT_MIXED_ATTACK, e.t.c

Important description Functions

- Family: script_family(english:”CGI abuses”);
- Copyright information: script_copyright (“english: This script is copyright© 2004 Nitesh Dhanjani”);

GUI Client Plug-in Description



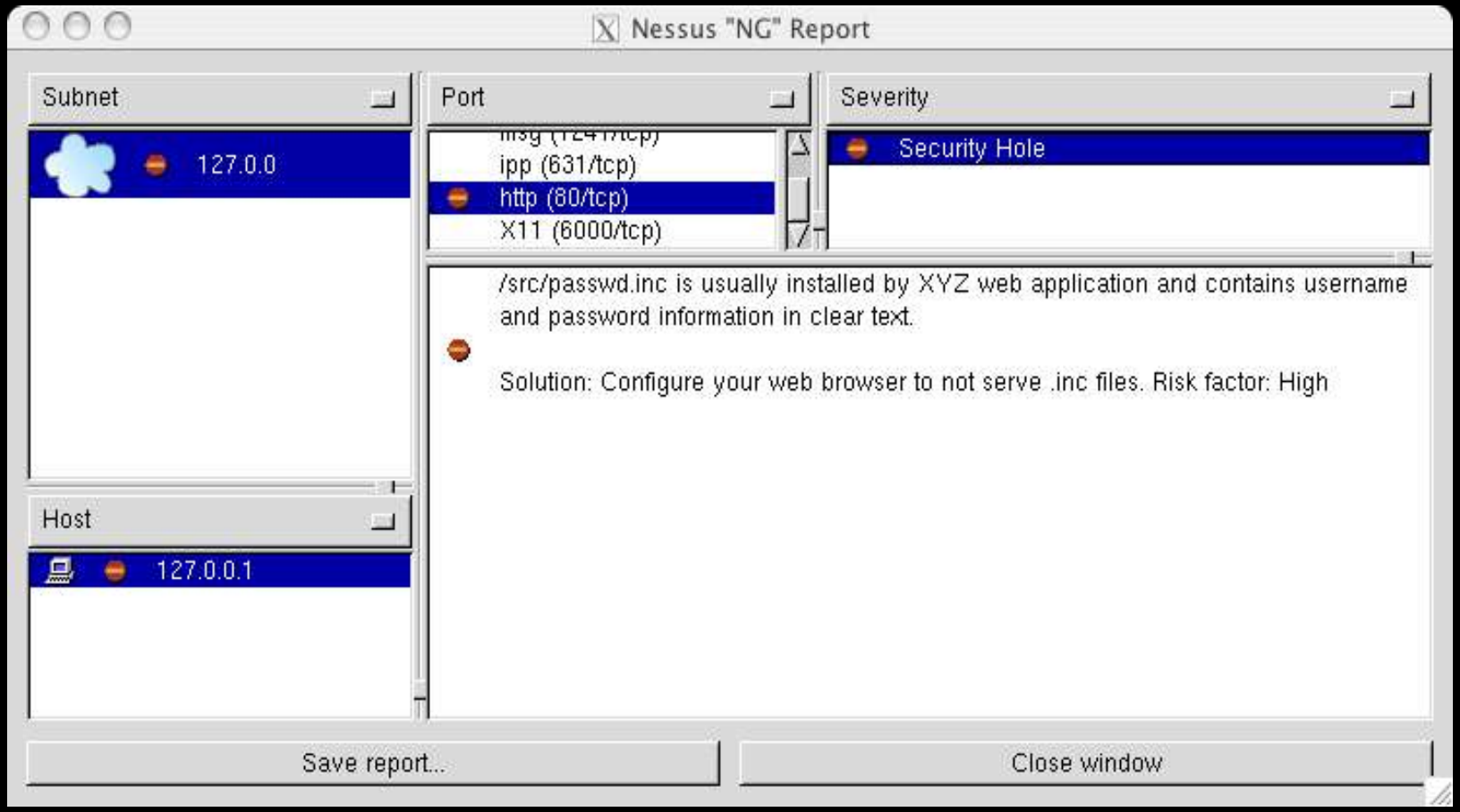
Testing for /src/passwd.inc

```
include ("http_func.inc");

port=get_http_port(default:80);

if(is_cgi_installed(item:"/src/passwd.inc",port:port))
        security_hole(port);
```

Output from Our Plug-in



Dissecting finger.nasl

- Finger client connects to port 79 of target

  host.
- Sends the string “username\r\n”.
- Similarly, finger.nasl:

```
buf = string("root\r\n");
send(socket:soc, data:buf);
data = recv(socket:soc, length:65535);
```

Dissecting finger.nasl

- The plug-in looks for “User”, “login”, “Login”, or “logged”:

```
if(egrep(pattern:".*User|[lL]ogin|logged.*", string:data))
{
    …
    …
    security_warning(port:port, data:report);
    set_kb_item(name:"finger/active", value:TRUE);
}
```

pop3_overflow.nasl

- Older versions of POP3 servers have been known to crash when the following commands are sent in addition to a very long argument:

  auth

  user

  pass

pop3_overflow.nasl

```
c = string("AUTH ", crap(2048), "\r\n");
  send(socket:soc, data:c);
  d = recv_line(socket:soc, length:1024);
  if(!d)security_hole(port);
```

pop3_overflow.nasl

```
else {
        c = string("USER ", crap(1024), "\r\n");
        send(socket:soc, data:c);
        d = recv_line(socket:soc, length:1024);
        if(!d)security_hole(port);
        else
        { ...
                        /* Similarly, try PASS*/
          …
        }
```

vnc.nasl

- Detects VNC (Virtual Network Computing servers).
- Probes ports 5900, 5901, and 5902
- Checks for the following pattern upon connect [VNC Banner]:

  ^RFB 00[0-9]\.00[0-9]$

vnc.nasl

```
...
r = recv(socket:soc, length:1024);
   version = egrep(pattern:"^RFB 00[0-9]\.00[0-9]
    $",string:r);
   if(version)
   {
      security_warning(port);
      security_warning(port:port, data:string("Version of
    VNC Protocol is: ",version));
   }
...
```

NASL Reference Manual

- Written by Michael Arboi
- Available from http://nessus.org/documentation.html
- Exhaustive list of NASL functions and language reference.

Introducing Hydra

- Parallelized multi-protocol brute forcer
- Written by van Hauser
- Available from http://www.thc.org/thc-hydra/
- Open GPL derived license
- Plugin to Nessus

Installing

```
tar zxvf hydra-4.1-src.tar.gz
./configure
make
make install (as root)
```

Using Hydra

- Supports a wide variety of protocols:
  - TELNET, FTP, HTTP, HTTPS, HTTP-PROXY, LDAP, SMB, SMBNT, MS-SQL, MYSQL, REXEC, SOCKS5, VNC, POP3, IMAP, NNTP, PCNFS, ICQ, SAP/R3, Cisco auth, Cisco enable, Cisco AAA
- In general:
  - Hydra –L userlist –P passlist server protocol

Structure of Hydra

- Each protocol is supported by a module named hydra-<service>
- Each protocol provides an identical interface to Hydra
- Each protocol leverages the supplied functions for accessing data supplied by the user, network functionality, and callbacks

Adding a new protocol

- We’re going to add SMTP AUTH LOGIN
  - For reference:
    - Connect to port 25
    - EHLO someserver.com
    - AUTH LOGIN
    - Base64 Username
    - Base64 Password

Adding a new protocol (cont)

```
#include "hydra-mod.h"
extern char *HYDRA_EXIT;
…
void service_smtpauth(unsigned long int ip, int sp,
    unsigned char options, char *miscptr, FILE * fp, int
    port)
…

  hydra_register_socket(sp);
…
```

Adding a new protocol (cont)

```
…
sock = hydra_connect_tcp(ip, myport);
… OR …
sock = hydra_connect_ssl(ip, mysslport);
…
if (sock < 0) {
        hydra_report(stderr, "Error: Child with pid %d terminating, can not
    connect\n", (int) getpid());
        hydra_child_exit(1);
}
```

Adding a new protocol (cont)

```
…
while (hydra_data_ready(sock)) {
        if((buf = hydra_receive_line(sock)) == NULL)
          exit(-1);
        free(buf);
      }
…
      if (hydra_send(sock, buffer, strlen(buffer), 0) < 0)
        exit(-1);
…
```

Adding a new protocol (cont)

```
start_smtpauth(sock, ip, port, options, miscptr, fp);
…
if (sock >= 0)
   sock = hydra_disconnect(sock);
…
hydra_child_exit(0);
```

Adding a new protocol (cont)

```
int start_smtpauth(int s, unsigned long int ip, int port, unsigned
    char options, char *miscptr, FILE * fp)
{
  char *empty = "";
  char *login, *pass, buffer[300], buffer2[300];
…
  if (strlen(login = hydra_get_next_login()) == 0)
    login = empty;
  if (strlen(pass = hydra_get_next_password()) == 0)
    pass = empty;
```

Adding a new protocol (cont)

```
while (hydra_data_ready(s) > 0) {
   if ((buf = hydra_receive_line(s)) == NULL) return (1);
   free(buf);
 …
sprintf(buffer, "AUTH LOGIN\r\n");
 if (hydra_send(s, buffer, strlen(buffer), 0) < 0) return 1;
 if ((buf = hydra_receive_line(s)) == NULL)
 …
 if (strstr(buf, "334") == NULL) {
   hydra_report(stderr, "Error: SMTP AUTH LOGIN error: %
    s\n", buf);
```

Adding a new protocol (cont)

```
  hydra_tobase64((unsigned char *) buffer2);
  sprintf(buffer, "%.250s\r\n", buffer2);
  if (hydra_send(s, buffer, strlen(buffer), 0) < 0)
…
  if ((buf = hydra_receive_line(s)) == NULL)
…
  if (strstr(buf, "334") == NULL) {
    hydra_report(stderr, "Error: %s\n", buf);
…
```

Adding a new protocol (cont)

```
  if (strstr(buf, "235") != NULL) {
    hydra_report_found_host(port, ip, "smtpauth", fp);
    hydra_completed_pair_found();
…
if (memcmp(hydra_get_next_pair(),
    &HYDRA_EXIT, sizeof(HYDRA_EXIT))== 0)
…
hydra_completed_pair();
  if (memcmp(hydra_get_next_pair(),
    &HYDRA_EXIT, sizeof(HYDRA_EXIT))== 0)
```

Adding a new protocol (cont)

- Add references in:
  - Makefile.am
    - Add in modules
  - Hydra.c
    - See the /* ADD NEW SERVICES HERE */ entries
    - Add in appropriate entries – refer to the entries for the other modules
  - Hydra.h
    - Add ports

Adding a new protocol (cont)

- Sample run: ./hydra –l justin –p badpwd mail.foo.com smtpauth

Hydra v4.1 (c) 2004 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2004-06-05 22:52:58
[DATA] 1 tasks, 1 servers, 1 login tries (l:1/p:1), ~1 tries per task
[DATA] attacking service smtpauth on port 25
[STATUS] attack finished for mail.foo.com (waiting for childs to finish)
[25][smtpauth] host: 64.219.211.30 login: justin password: badpwd
Hydra (http://www.thc.org) finished at 2004-06-05 22:52:59

Other useful Hydra functions

- hydra_get_next_pair()
- hydra_connect_udp()
- hydra_recv()

Nmap Service Detection

- nmap –sV or nmap –A
- Uses probes and responses defined in nmap-service-probes
- We're going to demo a canned protocol (after all, so many common things are supported already!) – cred.py from Twisted:
  - http://twistedmatrix.com/documents/current/examples/cred.py

Nmap Service Detection (cont)

nmap –sV or nmap –A

Uses prosudo nmap -A -p 1-65535 127.0.0.1

Starting nmap 3.50 (http://www.insecure.org/nmap/) at 2004-06-05 23:45 EDT

Interesting ports on localhost (127.0.0.1):

(The 65534 ports scanned but not shown below are in state: closed)

PORT STATE SERVICE VERSION

4738/tcp open unknown

1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at http://www.insecure.org/cgi-bin/servicefp-submit.cgi :

SF-Port4738-TCP:V=3.50%D=6/5%Time=40C29393%P=i686-pc-linux-gnu%r(NULL,59,"

SF:Login\x20with\x20USER\x20<name>\x20followed\x20by\x20PASS\x20<password

Nmap Service Detection (cont)

- We have a couple of options
  - Submit the service signature to Fyodor
    - http://www.insecure.org/cgi-bin/servicefp-submit.cgi
    - If nmap detects enough from it's probe to identify the service
  - Write your own probe statements and submit them
  - Write customs probes and matches ourselves

Nmap-service-probes

- File goes:
  - Probes
  - Ports
  - SSLPorts
  - Matches
- Probes
  - Probe <Protocol> <probe name> <probe string>
  - e.g. Probe TCP GetReq q|GET / HTTP1/0\r\n\r\n|

Nmap-service-probes

- Ports
  - ports <list>
  - e.g. ports 80,8080
- SSL Ports
  - sslports <list>
  - e.g. sslports 443
- Matches
  - match <service> <pattern> [version info]
  - e.g. match http m|^HTTP/1\.1 400 .*\r\nServer: Microsoft-IIS/(\d[-.\w]+)\r\n| v/Microsoft IIS webserver/$1//

Questions?

Thank-you!

;-)