exploiting vulnerabilities

who am i ?

H D Moore <hdm [at] metasploit.com>

Metasploit project

Core developer and project lead

BreakingPoint Systems

Director of Security Research

what is this about ?

- Vulnerability classification
- Exploiting memory corruption
- Developing new exploits
- Attacking with Metasploit

why listen ?

- Understand impact of flaws
- Understand exploit design
- Learn to create new exploits
- Learn to use Metasploit

vulnerability classification

classification

- Classify by the cause
  - Useful for the developer
- Classify by the impact
  - Useful for the attacker

cause classification

- Buffer overflow
- Integer overflow
- Format string
- Input validation

impact classification

- Machine code execution
- Other code execution
- Authorization bypass
- Denial of service

example 1

- Microsoft DCOM buffer overflow
  - Insufficient length checking
  - Machine code execution
- Many ways to execute code
  - Standard stack overflow
  - Exception handler pointer overwrite

example 2

- WordPress SQL injection
  - Insufficient validation
  - Data manipulation
  - Authentication bypass
  - Code execution via templates

memory corruption

memory corruption

- Corruption is caused by
  - Incorrect string termination
  - Insufficient length checking
  - Insufficient data verification
  - Uninitialized variable use

exploiting corruption

- Goal is arbitrary code execution
- Stack overflows are simple
  - Place shellcode in memory
  - Modify return address
  - Application returns to shellcode

exploiting heap overflows

- Depends on library and application
    - Overwrite variables on heap
    - VTables for C++ on Windows
    - Standard “write-what-where”
    - Overwrite a function pointer
    - Force function to be called

exploiting format strings

- Depends on library and application
  - Use “%n” to overwrite a pointer
  - Force the pointer to be called
- Non-standard implementations
  - Specific applications
  - Mac OS X (%@)

exploiting corruption

- The same common formula
  - Place shellcode into memory
  - Modify memory in some way
  - Force shellcode to be called

exploiting seh

- Structured exception handling
  - Patented by Microsoft
- Implements try - except - catch
- SEH structure pushed to the stack
- SEH structure contains next pointer

exploiting seh

- Exploit crazy bugs on Windows
  - memcpy(dst, src, -1)
- Application-specific handlers
  - Brute-force return addresses

shellcode

shellcode

- Tiny programs written in ASM
- Work within the exploited process
- Make use of existing resources
- Original “shellcode” runs /bin/sh

unix shellcode

- Use system call interface
- Find or create socket handle
- Map stdio to the socket handle
- Execute /bin/sh

windows shellcode

- System calls are rarely used
- PEB to resolve Kernel32.dll
- Kernel32 has LoadLibrary()
- LoadLibrary loads anything

common shellcode

- Shell via existing socket
- Shell via reverse connect
- Shell via listening socket
- Download and execute binary

advanced shellcode

- Load new code from network
- Inject DLL into memory
- “Syscall Proxying”
- Write binary to disk and execute

exploit design

exploit design

- Common exploit behavior
  - Connect to the victim
  - Negotiate protocol
  - Trigger code execution
  - Interact with payload

exploit connection

- Connection methods
  - Connect via TCP service
  - Send a UDP request
  - Use DCERPC to call functions
  - Listen for new connection

exploit protocols

- Network protocols
  - Telnet, HTTP, FTP, POP3
  - SMB, DCERPC, SUNRPC
  - 802.11, ICMP, IGMP

exploit buffers

- Buffer contents
  - Normal padding data
  - Target shellcode
  - Target return address

exploit interaction

- Interact with the payload
  - Listen for incoming connection
  - Connect to the target system
  - Check existing socket for shell

metasploit framework

metasploit framework

- An exploit development platform
  - Security researchers
  - Penetration testers
  - Security vendors
  - Script kiddies

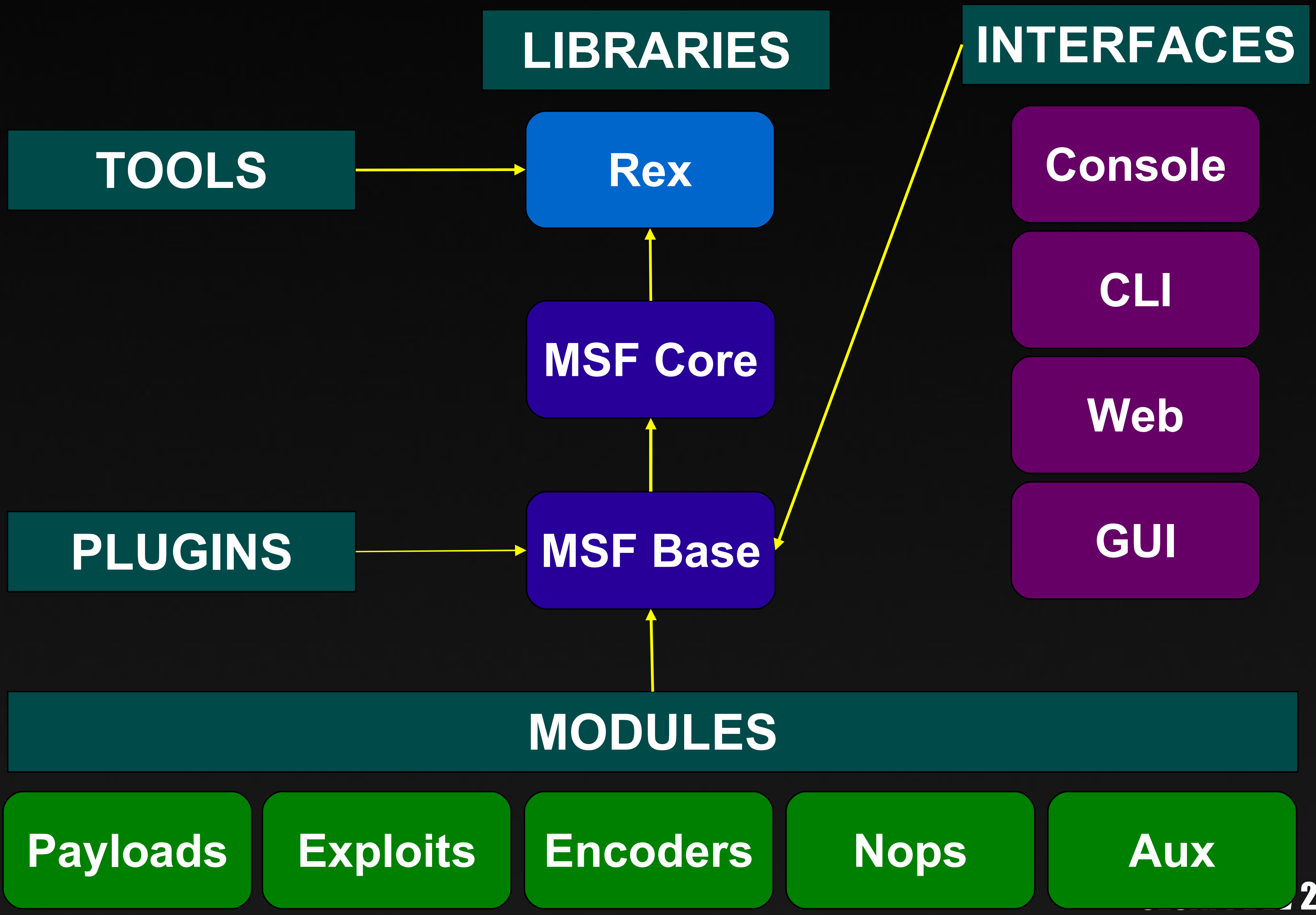
metasploit history

- Version 1.0 (2003-2004)
  - Perl, 15 exploits, curses UI
- Version 2.0 (2004-2006)
  - Perl, 150+ exploits, 3 UIs
- Version 3.0 (2007+)

metasploit 3.0

- 100,000 lines of Ruby
- 53,000 lines of C/C++
- 8000 lines of ASM
- 350 unique modules
- 2 years to develop

architecture



the Rex library

- Text manipulation
- CPU instructions
- Fancy sockets
- File formats
- Protocols
  - SMB, DCERPC, SUNRPC, HTTP

metasploit modules

- Simple Ruby classes
- Dynamically loaded
- Rich meta-information
- Expose type-specific methods

metasploit exploits

- Modules inherit Msf::Exploit
- Heavy use of Ruby mixins
  - TCP, UDP, SMB, HTTP
  - Active, Passive, Brute force
  - WiFi, Pcap, Bluetooth

exploit header

```
class Exploits::FTP_OVERFLOW < Msf::Exploit::Remote
include Exploit::Remote::FTP
'Name'              => 'Microsoft FTP Overflow',
'Description'       => 'This module exploits...'
'Author'            => [ 'hdm' ],
'License'           => MSF_LICENSE,
'Version'           => '$Revision: 4419 $',
'Payload'           =>
{  'Space'     => 1024,
   'BadChars'  => "\x00\x0a\x0d\x5c\x5f\x2f\x2e" }
```

exploit code

```
connect
print_status("Trying target #{target.name}...")
buf = Rex::Text.rand_text_english(8192)
buf[1004, 4] = [target.ret].pack('V')
buf[1008, payload.length] = payload.encoded
send_cmd( ['USER', buf] , false)
handler
disconnect
```

user interfaces

- msfconsole
- msfcli
- msfweb
- msfgui

demonstration

creating exploits

creating exploits

- Create a new Metasploit module
- Fill in the meta-information
- Add appropriate mixins
- Define the exploit() method

demonstration

running exploits

running exploits

- Select a supported target
- Select a supported payload
- Complete all options
- Launch the exploit

demonstration

summary

summary

- Little bugs have a huge impact
- Exploit from bug in ~10 minutes
- Metasploit is publicly available

questions?