Fuzzing Selected Win32 Interprocess Communication Mechanisms

Jesse Burns, Principal Partner, iSEC Partners

Prepared For:
Black Hat 2006, Las Vegas

August 2, 2006

Information Security Partners, LLC
iSECPartners.com

Overview of Presentation

- **Background on Fuzzing and Windows IPC**
- **Identifying Communication Channels to test**
  - Using Systernals tools
  - Using WinDBG
- **Code hooking to find Named Pipes**
- **Code hooking to test Named Pipes**
- **Fuzzing Shared Sections**

There will be a few demos and iSEC has released sample code

Introduction

Fuzzing is the intentional providing of bad data to find bugs

Most software has two kinds of input:

1. **Trusted – If you can write there, you are already trusted**
   - Software binaries themselves are trusted "input"
   - Scripts, and libraries are usually trusted (some exceptions like Java & C#)
   - ACL'd data files may be trusted
2. **Untrusted – A privilege or identity gradient exists between the producer and consumer**
   - Network inputs are generally untrusted
   - Some communications aren't ACL'd against all possible bad guys

Security testing of **all** untrusted input is crucial!
Robustness testing of trusted inputs is nice to have

Fuzzing can help with testing of both kinds of input

Microsoft Windows IPC

Much Windows Interprocess communication (IPC) is Based on "Securable Objects"

- Standardized design
- Feature-rich security including access control and auditing

Local and Remote

- Mostly used locally
- Remote security issues are serious but not today's topic
- Many mechanisms are only local

Highly compatible between versions

- NT 3.1 has similar IPC channels to Windows Vista
- Remote communication may break due to security improvements or heightened security settings
- Surprisingly it often works between NT and W2K3 R2

What is a Win32 IPC channel

All securable objects share attributes

- Discretionary Access Control (DACL)
- Mandatory auditing is available but rarely used (SACL)
- They have Owners
- Under Windows Vista – Integrity Level – not today's topic

Additionally they may have names for sharing

Processes running as different users can name their IPC channels and then communicate through them

Show me a big list of Win32 IPC mechanisms

Glad you asked

- Events
- Event Pair
- Files – you might have heard of these ;)
- Keyed Event – New for XP
- LPC – Fast but not documented :(
- Mailslots
- Mutexes
- Named Pipes
- Registry keys
- Semaphores
- Shared Sections
- Sockets?!? Now securable, not named
- Timers

Selected Win32 IPC mechanisms

Named Pipes

A message or stream-based way to communicate

- Message mode is atomic like UDP
- Streams are more like TCP and may block mid-message

Local or Remote – but the remote story is very dangerous!

- Remote is based on SMB

Shared Sections

A local way of sharing memory pages between processes

- Awesome speed
- Needs external synchronization
- Data validation can be problematic

Local only

Selected Win32 IPC mechanisms

Events

A shared signaling mechanism

- Like yelling "Now" -- no actual message
- Event is context-dependant

Local only

- Often used with Shared Sections to let listeners know a message is waiting

Semaphores

A thread synchronization mechanism

- Works between processes too, allows for counted accesses

Local only

Goals

Primary: To identify local or remote vulnerabilities.

Interested in privilege escalation

- Possible when communicating across a **privilege or identity gradient**
- Why IPC is all about communicating, so lets focus there

Secondary: To work on applications in binary form.

Find bugs in libraries, COM objects, and dependencies

- Sometimes we need to trust things we can't review

Reuse testing tools

- Regression testing should include security tests
- Tools for binaries are easier to reuse across products

Minimize changes for new versions of software

Fuzzing for IPC Bugs – A tool with limits

Ideally find any kind of bug

Actually we are only targeting issues that are easy to see manifest under this technique

Crashers or bugs causing exceptions to catch in the debugger are ideal targets

Bugs that introduce inconsistent states, or screw up the system, database, or record store in ways detectable by automated tests are also good

Fuzzing for IPC Bugs – Ideal Bugs

Targeting code written in C or C++ helps

- Tends to have lots of stupid, technical bugs outside the domain of the application. This improves reusability of the testing tools.

These sort of bugs pop out

- Integer Overflows
- Heap & Stack Overflows
- Pointer problems (i.e. generic writes into buffers with untrusted offsets)
- Parser bugs
- Double frees

Fuzzing for IPC Bugs – Not So Much

Non-language or "domain specific" bugs

- Injection bugs i.e. SQL or Command
- Authorization bypass
- Many, many, others

Intentional "bugs"

- Back doors
- Insecurely designed features like "automatic updates"

IPC naming & creation timing bugs

- Name squatting attacks
- Races prior to DACL establishment

These would be nice to find, but aren't suited to this approach
Surprisingly automated fuzzing isn't a security cureall

Oh well — I guess I can keep my job then

Easy viewing of IPC Objects

Processes Explorer™ and Handle™ from Sysinternals.
http://www.systernals.com/

Unfortunately these tools are not affiliated with iSEC at all.

I am a big fan of their excellent tools however.

Somewhat like a super LSOF for windows.

Screen Shot of Systernals - Process Explorer

Process Explorer - Sysinternals: www.sysinternals.com [FOOZ\jesse]

File Options View Process Find Handle Users Help

Process	PID	CPU	Description	Company Name	Window T
AcMurocHlpr.exe	428				
AcPrfMgrSvc.exe	964				
AcSvc.exe	3164		ThinkVantage Access Connections Main Service Module	Lenovo	
ACWLIcon.exe	2992		ThinkVantage Access Connections Wireless Status Icon	Lenovo	
alg.exe	3832		Application Layer Gateway Service	Microsoft Corporation	
ati2evxx.exe	1884		ATI External Event Utility EXE Module	ATI Technologies Inc.	
ati2evxx.exe	2980		ATI External Event Utility EXE Module	ATI Technologies Inc.	
bash.exe	3700				
BTTray.exe	2864		Bluetooth Tray Application	Broadcom Corporation	
btwdins.exe	1028		Bluetooth Support Server	Broadcom Corporation	
CallLogger.exe	5852				
cmd.exe	5140		Windows Command Processor	Microsoft Corporation	cmd - bash
cmd.exe	3668		Windows Command Processor	Microsoft Corporation	C:\WINDO
cmd.exe					

Type	Name
Directory	\Windows
Directory	\KnownDlls
File	C:\Documents and Settings\jesse\My Documents\Visual Studio 2005\Projects\LogExample\debug
File	\Device\NamedPipe\callLogger
File	C:\WINDOWS\WinSxS\x86_Microsoft.VC80.DebugCRT_1fc8b3b9a1e18e3b_8.0.50727.42_x-ww_f75eb16c
KeyedEvent	\KernelObjects\CritSecOutOfMem
WindowStation	\Windows\WindowStations\WinS

CPU Usage: 19% Commit Charge: 28.50% Process

\Device\NamedPipe\callLogger Properties

Details | Security

Basic Information

Name: \Device\NamedPipe\callLogger

Type: File

Description: A disk file, communications endpoint, or driver interface.

Address: 0x8A3A4D28

References

References: 5

Handles: 3

Quota Charges

Paged: 0

Non-Paged: 0

OK

\Device\NamedPipe\callLogger Properties

Details | Security

Group or user names:

- Administrators (FOOZ\Administrators)
- ANONYMOUS LOGON
- Everyone
- SYSTEM

Add... Remove

Permissions for Administrators	Allow	Deny
Delete	✓	
Execute		
Synchronize	✓	
Query State	✓	
Modify State	✓	
Special Permissions	✓	

For special permissions or for advanced settings, click Advanced. Advanced

OK

Advanced Security Settings for \Device\NamedPipe\callLogger

Permissions | Auditing | Owner

To view more information about Special permissions, select a permission entry, and then click Edit.

Permission entries:

Type	Name	Permission	Inherited From
Allow	SYSTEM	Special	<not inherited>
Allow	Administrators (FOOZ\Administr...	Special	<not inherited>
Allow	Everyone	Special	<not inherited>
Allow	ANONYMOUS LOGON	Special	<not inherited>

Add... Edit... Remove

OK Cancel Apply

Screen Shot of Systernals - Handle

C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\jesse>\tools\systernals\handle.exe -a -u -p CallLogger


Handle v2.2
Copyright (C) 1997-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

------------------------------------------------------------------------------
CallLogger.exe pid: 5852 FOOZ\jesse
   168: File          C:\Documents and Settings\jesse\My Documents\Visual Studio
2005\Projects\LogExample\debug
   7dc: File          \Device\NamedPipe\callLogger
   7e0: WindowStation \Windows\WindowStations\WinSta0
   7e4: Event
   7e8: File          C:\WINDOWS\WinSxS\x86_Microsoft.VC80.DebugCRT_1fc8b3b9a1e18
e3b_8.0.50727.42_x-ww_f75eb16c
   7ec: Port
   7f0: Directory     \Windows
   7f8: Directory     \KnownDlls
   7fc: KeyedEvent    \KernelObjects\CritSecOutOfMemoryEvent

C:\Documents and Settings\jesse>
```

```
C:\Documents and Settings\jesse>\tools\systernals\handle.exe -h

Handle v2.2
Copyright (C) 1997-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

usage: handle [-a] [-u] [-p <processname>] [name]
  -a         Dump all handle information
  -u         Show the owning user name when searching for handles
  -p         Dump handles belonging to process (partial name accepted)
  name       Search for handles to objects with <name> (fragment accepted)

No arguments will dump all file references

C:\Documents and Settings\jesse>
```

Systernals Tools

Yes:

- Great for getting started with understanding windows IPC mechanisms
- Can help identify communication across privilege or identity gradients for testing

No:

- Not the right tool to use for short lived objects
- Not a tool for manipulating the creation of these objects
- Not directly used for fuzzing

WinDbg – The Windows Debugger

- **Great multipurpose tool for working with binaries**
- **Available without charge from Microsoft**
- **Can intercept kernel32 calls that create IPC channels**
  - CreateNamedPipeW/A – Creating named pipes
  - CreateFileMappingW/A – Creating Shared Sections
- **Attaches to existing processes or spawns new ones**

"C:\Documents and Settings\jesse\My Documents\Visual Studio 2005\Projects\CallLogge...

File Edit View Debug Window Help

Command

```
 1 e 7c85fa54     0001 (0001)  0:**** kernel32!CreateNamedPipeA
 2 e 7c80939e     0001 (0001)  0:**** kernel32!CreateFileMappingW
 3 e 7c80946c     0001 (0001)  0:**** kernel32!CreateFileMappingA
0:000> kb
ChildEBP RetAddr  Args to Child
0012ff50 00401137 00402104 00080001 00000004 kernel32!CreateNamedPipeW
0012ff7c 00401334 00000001 00353d88 00354ea0 CallLogger!wmain+0x27 [c:\documer
0012ffc0 7c816d4f 00191f18 00000000 7ffd5000 CallLogger!__tmainCRTStartup+0x1(
0012fff0 00000000 0040147d 00000000 78746341 kernel32!BaseProcessStart+0x23
0:000> dd 12ff50
0012ff50  00000000 00401137 00402104 00080001
0012ff60  00000004 000000ff 00000800 00000800
0012ff70  00000000 00000000 00403b88 00000001
0012ff80  00401334 00000001 00353d88 00354ea0
0012ff90  50eed5ea 00191f18 00000000 7ffd5000
0012ffa0  00000000 00000000 0012ff90 e2d7133a
0012ffb0  0012ffe0 004017e5 50bc0982 00000000
0012ffc0  0012fff0 7c816d4f 00191f18 00000000
0:000> du 402104
00402104  "\\.\pipe\callLogger"
```

0:000>

Ln 67, Col 1 | Sys 0:<Local> | Proc 000:afc | Thrd 000:3ac | ASM | OVR | CAPS | NUM

iSEC PARTNERS

WinDbg – The Windows Debugger

Yes:

- Able to alter program behavior like system calls
- Useful for slowly manipulate the creation of objects
- Breakpoints extend the life of short lived objects so we can examine them
- Postmortem debugging for when our fuzzing finds bugs

No:

- Not graphical, harder to start out with
- Harder to examine ACLs to find potential communication across a privilege gradient
- Difficult to use for directly fuzzing

Demo with Microsoft - WinDbg

Using WinDBG to intercept the creation of a named pipe

What can we do with this?

- **Discover IPC mechanism use**
- **Hook IPC mechanism use for fuzzing**

Demonstration (Fingers crossed)

Discovering IPC mechanism use with WinDbg

Possible but not ideal with WinDbg – informative though

1. **Put a break point on the targeted IPC mechanism**
   1. bp kernel32!CreateNamedPipeW
2. **Wait for the break point to be hit (run and exercise the application)**
3. **Take a look at the first parameter to CreateNamedPipeW with a commands like**
   1. kb, (look up offset of first arg)
   2. du 402104, (402104 being the address of the first arg from above step)
4. **Look – the name of the pipe being created is displayed**

Yes - this is slow and therefore boring to do

But understanding it we can automate it…give me a few slides

Middle person approach to fuzzing

1. **Intercept the creation of an IPC endpoint named "Foo"**
   - For example a named pipe called \\.\pipe\Foo
2. **Alter it so it uses some unexpected name "Bar"**
   - You can use their buffer if you don't change the length \\.\pipe\Bar
   - Change it back after the sys call if you feel paranoid or have problems
   - Todays example just uses a new buffer, and leaks a little memory
3. **Create our own named IPC mechanism named "Foo"**
   - This requires a script, I use python for my examples but Java, Perl, C#, C++ are all fine.
4. **Our script connects to "Bar" when it receives a connection on "Foo" and forwards reads and writes from "Foo" to "Bar"**
   - It can log the content for initial analysis
   - pass everything through for testing
   - Alter random or selected bytes for fuzzing

Code Injection and Hooking

Tools for code injection and hooking are available

- Microsoft's Detours, commercially licensable http://research.microsoft.com/sn/detours/
- Matt Conover's BSD licensed x86hook tinjectdll & friends http://www.cybertech.net/~sh0ksh0k/projects/
- Hook API SDK, a commercial product which I haven't used http://www.hook-api.com/index.html
- MadCodeHook, another commercial product I haven't tried http://www.madshi.net/ (Written with Borland Delphi!)
- Home grown 'ghetto detours' things like Scott Stender, and Andreas Junestam have both independently thrown together at iSEC.

I use Matt's excellent code for this presentation

Matt's code is open, feature-rich and even nice to read

Hooking and Injection

Hooking is a powerful tool, useful far beyond fuzzing

Using injection we place our code into a running process
Usually injecting a DLL for for convenience and packaging

- The injected code replaces parts of the code we hook
- The hooked code calls us, and we handle the call
- Examples of hooking are included with each package

Hookdll_healp.dll is demo'd in the HeapHookDll project

Hooking and Injection – Discovering IPC mechanisms

Hooking allows optimized discovery of IPC mechanisms

Idea is the same as with WinDbg. But:

1. Hook the API rather than break point on it
2. Log the name, ACL, and other parameters of interest rather than examining them manually

Step two is actually easier with hooks!
Your code receives the parameters in a typed, easy to read format.

Hooking and Injection – IPC Discovery Demo

Injecting an logging hook to discover named pipe creation

This one is easy, not boring to use, totally reusable

The program is also very short

Demonstration (Toes crossed)

Hooking and Injection – Fuzzing

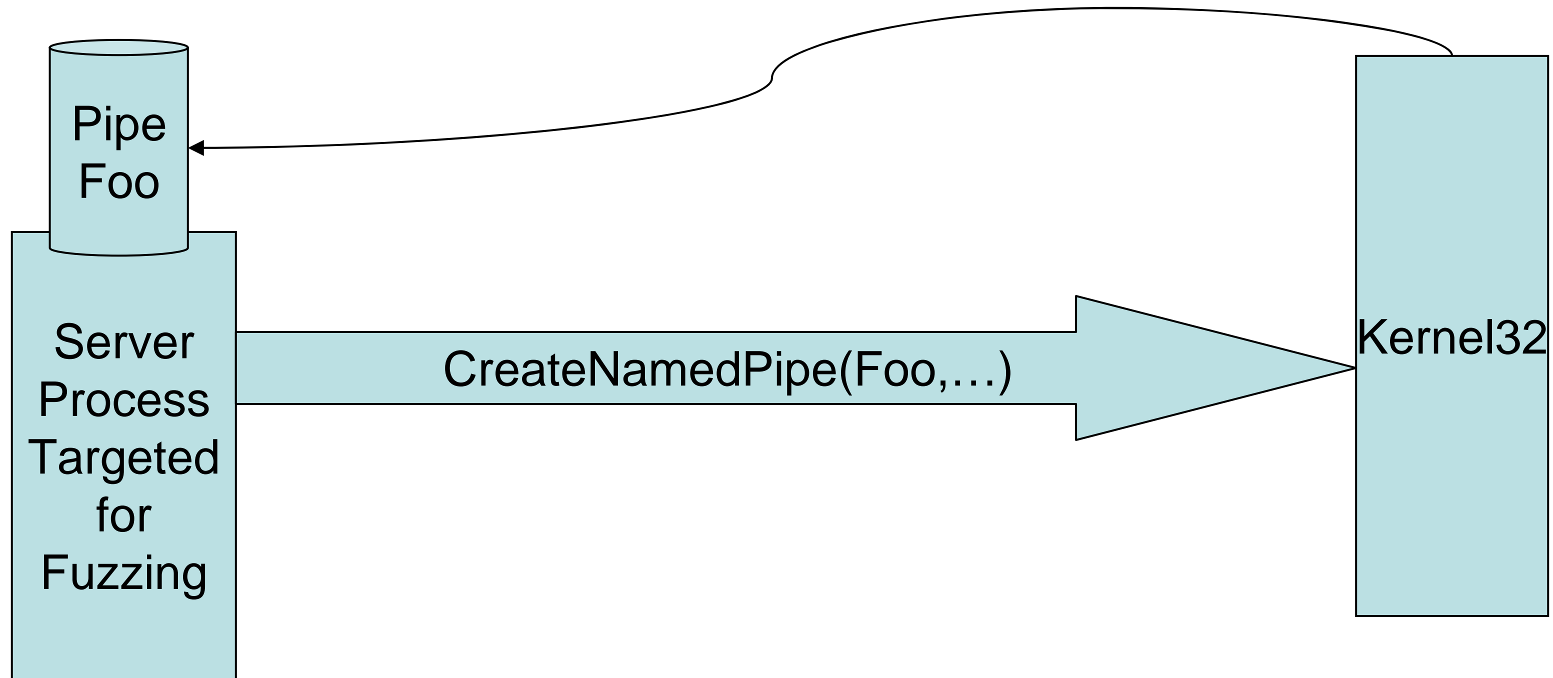
Injecting a middle person!

Just like we did with WinDbg, but now we spawn the ‘script-in-the-middle’ automatically

- **Hook CreateNamedPipeA/W**
- **In the hook call ShellExecute to launch our fuzzing python script**
- **Python script takes the proper name, new name of the script as well as any details like buffer sizes, modes, etc.**
- **Python script fuzzes the data before passing it along**

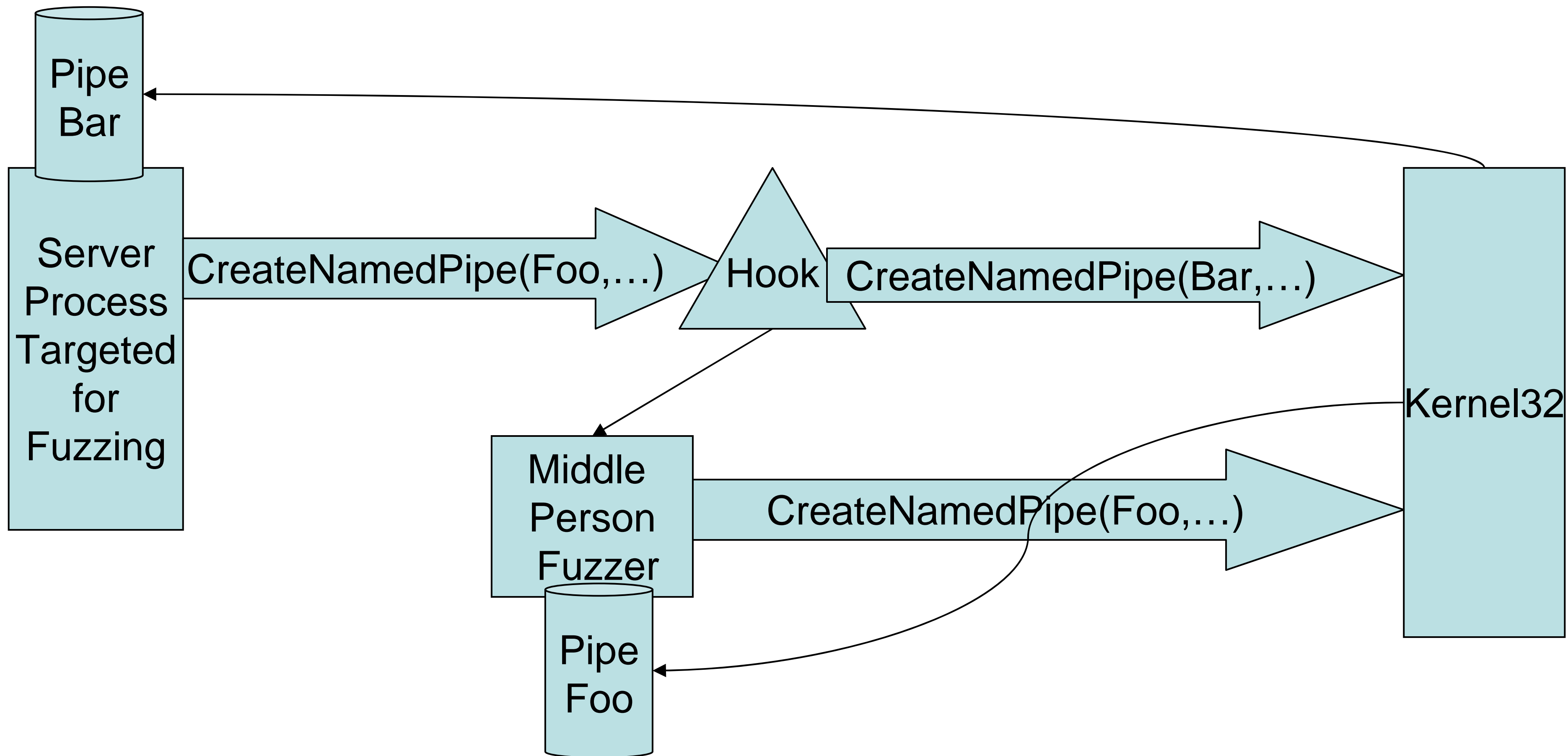
Set WinDbg as post-mortem prior to fuzzing with “windbg -I”

Normal flow of process



Clients connect to pipe Foo

Hooking and Fuzzing



Clients connect to Pipe Foo

Fuzzer forwards reads and writes to Bar after modification

Hooking and Injection – Fuzzing Demo

Hook the creation of a named pipe to fuzz a server!

Example messes with client input, passes server writes unchanged

Demonstration (Cross an appendage)

Hooking and Injection – Fuzzing

Script-in-the-middle can be fairly simple!

```
def main():
    server_name, client_name, mode, type, max, timeout = get_args()
    # make a pipe for the client
    client = CreateNamedPipe(client_name, mode, type, max, bs, bs,
                                          timeout, None)
    while True:
        connect(client)
        # connect to the server
        server = win32file.CreateFile(client_name, …)
        forward_data(client, server, mutator)
        disconnect(client, server)
    …
```

Fuzzing Fuzzies.

So what should we change

- Randomly change values
- Incrementing or decrementing values
- Inserting large numbers like 0xFFFFFFFF
- Overwrite nulls
- Overwrite double nulls for unicode strings
- Extend the size of writes so they are huge
- Change strings specific to the application

Mix it up a bit

The ideal rate of change is often very low

To allow getting into the interaction before causing errors

More fuzzing tool tips

- Log the random number generator’s ‘seed’ used for each run of your fuzzer, to allow re-testing
- Logging pipe traffic allows for easier failure reproduction
- Use injection to perform fuzzing as part of automated tests

I have written a few simple template schemes to specify fuzzing guidelines in, these are simple to write

Shared Sections – A good fuzzing target

Shared sections are just named blocks of shared memory

May be associated with some signaling mechanism like

- Event
- Named Pipe
- Mailslot
- Socket

Discoverable using the previously discussed techniques

Writes to these blocks ‘should be synchronized but that is just an un-enforced convention

In practice updates may come at any time

Shared Sections – Fuzzing Technique

1. Scribbling – create bad data to cause failures
2. Asynchronous Scribbling – Breaking the convention

Usually shared sections already contain the last message
This is a good place to start

Shared Sections – How to Scribble

1. Start a test of the server process
2. Map the shared section into your fuzzers address space
3. Make a copy of the sections content
4. Make changes to the content
5. Signal the event associated with the shared section

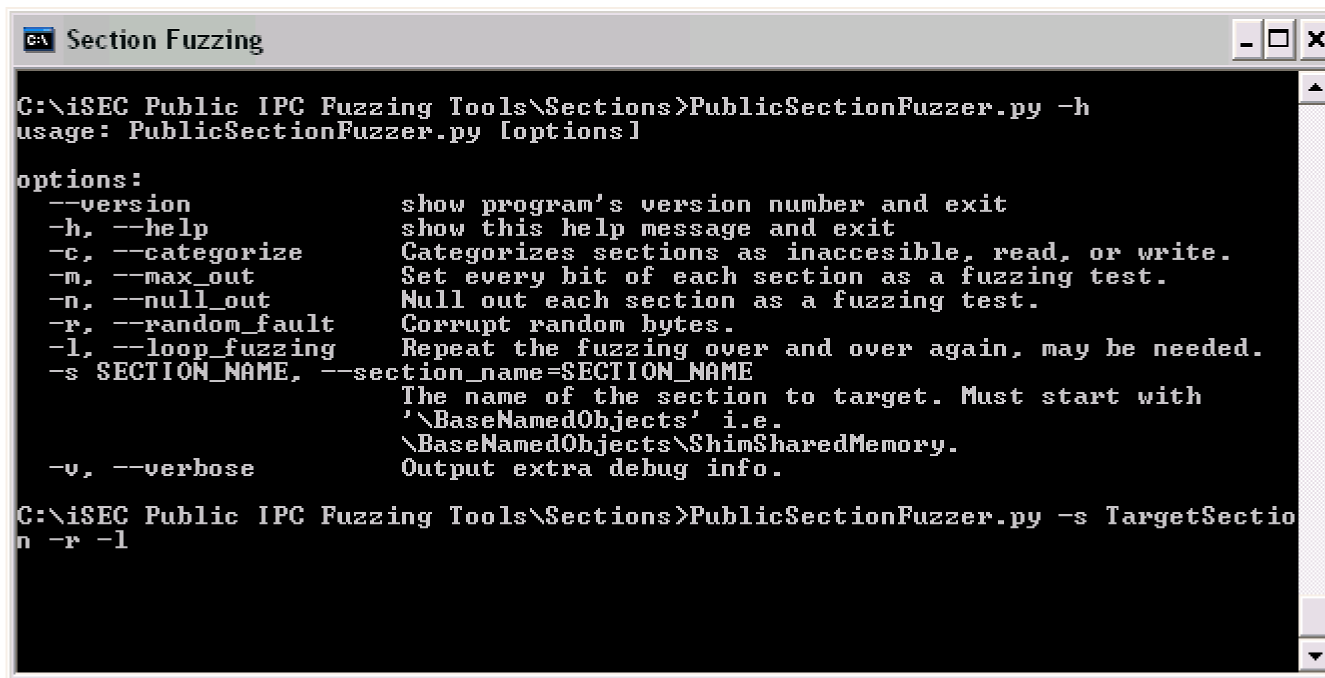
Shared Sections – How to Scribble Asynchronously

1. Start a test of the client and server process
2. Map the shared section into your fuzzers address space
3. Spawn a thread to make continuous changes to the content of the section
4. Optionally signal the event associated with the shared section

Note: Continuously changing the contents can break writes from other clients removing the need to figure out the signaling system.

Shared Section – Scribbling Example

Using a small python program to fuzz a shared section.



Asynchronous section scribble – Defense

Always copy the content of a shared section into non-shared memory before performing data validation.

And consider locking down the ACLs on those objects!

Questions

Thanks to Scott Stender, of iSEC for his important contributions on this topic and for my presentation. He taught me a lot about Win32 IPC, and wrote the first shared section fuzzer I ever saw.

Please get updates including slides and example code from http://www.isecpartners.com

iSEC does commercial work – including helping companies test their products and making reusable testing regimes.

Want to talk more about fuzzing or need some help?

I am reachable as:

jesse_injection_fuzzing (at) isecpartners . com