Meddle: Framework for piggy-back fuzzing and tool development

Geoff McDonald

glmcdona@gmail.com, unaffiliated presentation

August 8, 2014

DEF CON 22

About Me

- Vancouver, Canada
- Game hacking (Ultima Online MMORPG)
- Reverse-engineering tool developer
- Previously Symantec
- Currently at Microsoft
- Personal website http://www.split-code.com/

Types of Fuzzing

File Format Fuzzing

- PDF, Microsoft Word, or TrueType fonts

Protocol Fuzzing

- RDP, VNC, SSL, or Voip

Application Fuzzing

- COM objects, API calls, or inter-process communication

Web Application Fuzzing

- Joomla, WordPress, or any website

Fuzzing Tools

SPIKE from Immunity [1]

- Network protocols and web applications

Basic Fuzzing Framework (BFF) from CERT [2]

- File format

SAGE from Microsoft [3]

- Input fuzzing

AutoFuzz [4]

- Network protocols by MITM

COMRaider [5]

- COM interface fuzzing

IOCtrlFuzzer from eSage Lab [6]

- NtDeviceIoControlFile driver input fuzzing

Fuzzing Algorithms

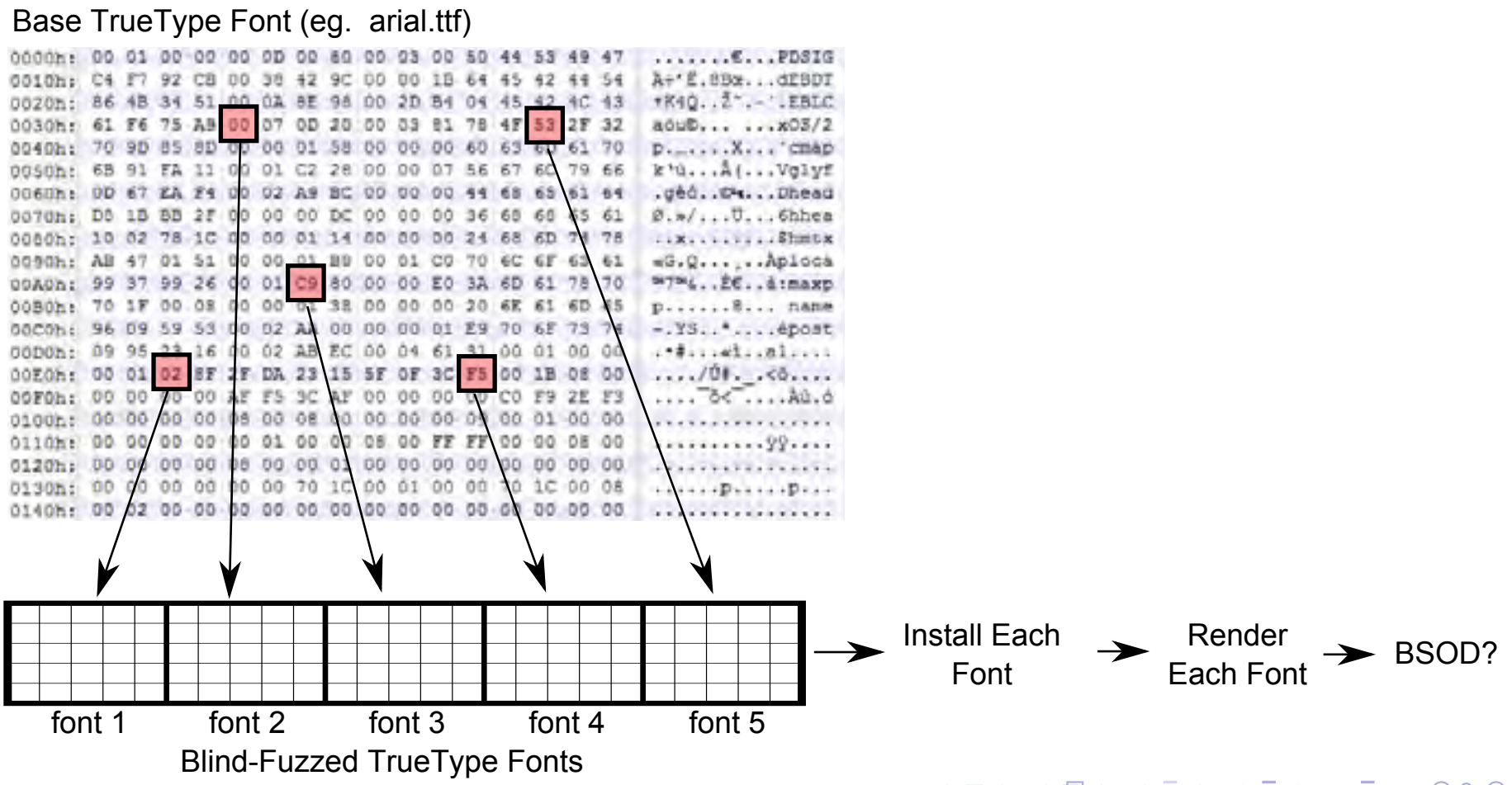
Basic algorithms:

- Naive protocol fuzzing (eg. IOCtrlFuzzer [6])
- Protocol aware fuzzing (eg. SPIKE [1])

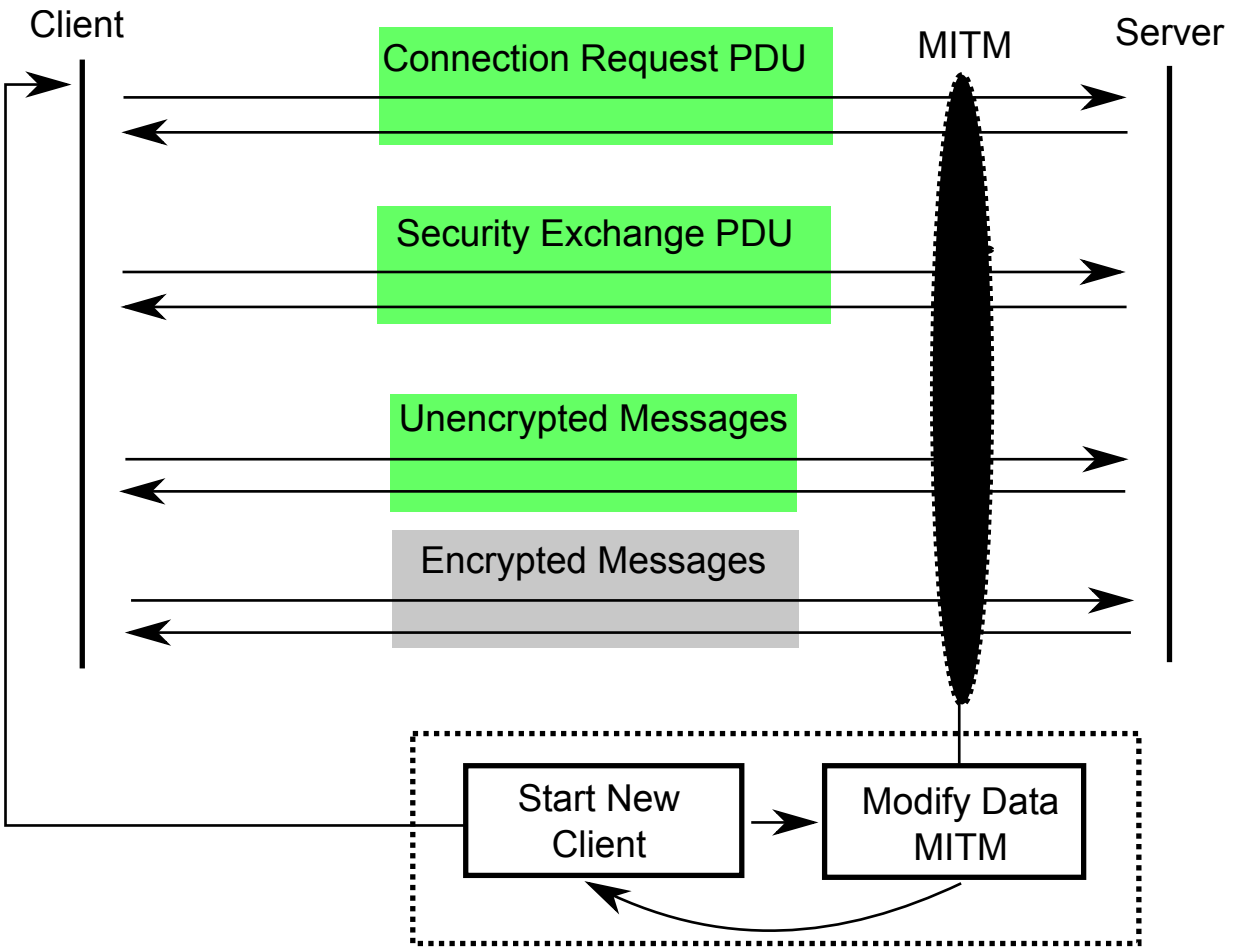
Advanced algorithms:

- Protocol-learning before fuzzing (eg. Autofuzz [4])
- Feedback-driven fuzzing (eg. Sage [3])
- Code coverage fuzzing (eg. Google's Flash fuzzing [7])

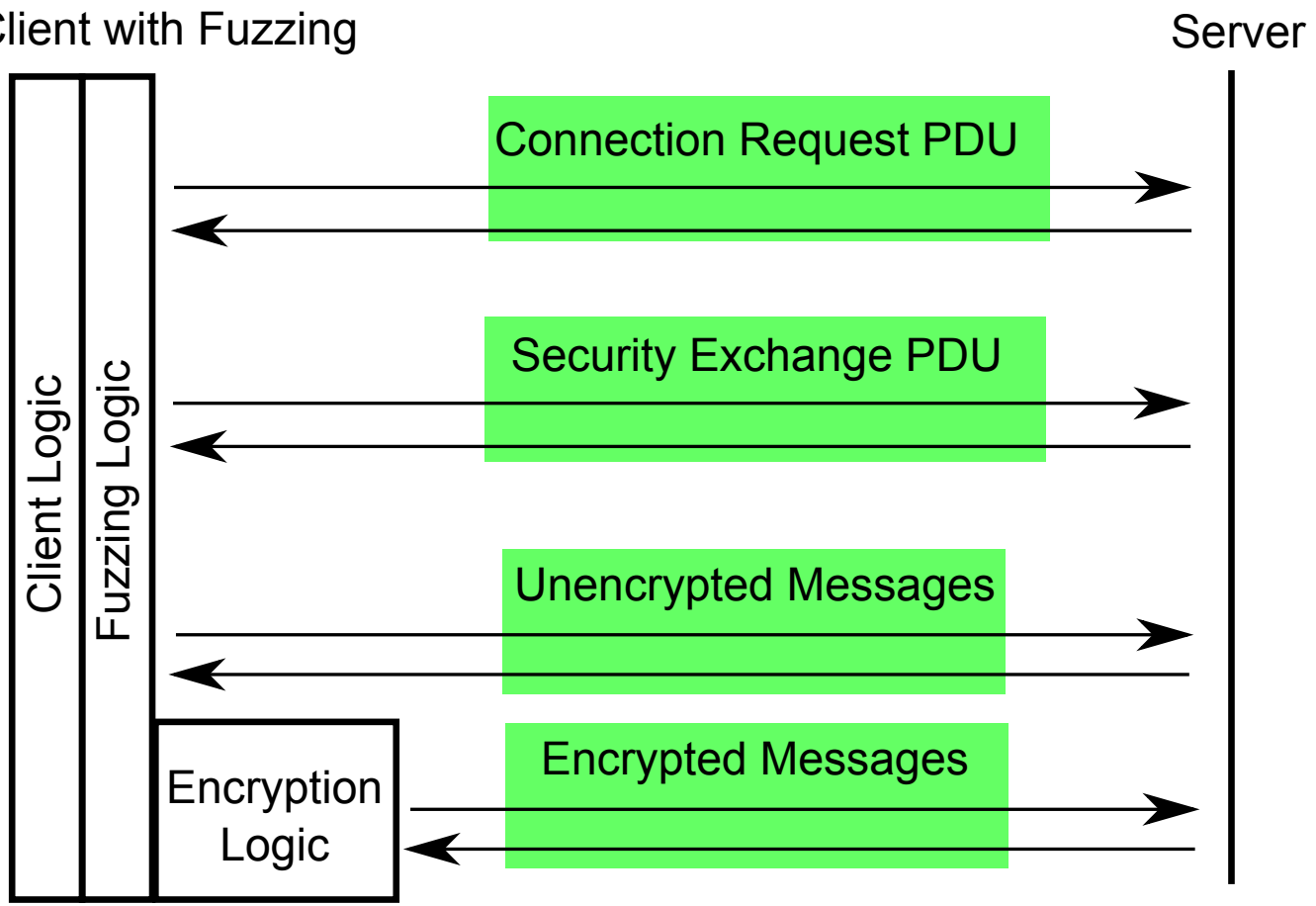
File Format Fuzzing: TrueType



Protocol Fuzzing: RDP by Network MITM



Protocol Fuzzing: RDP by Client Implementation



Protocol Fuzzing: RDP by File Fuzzing

Luigi Auriemma's CVE-2012-0002 POC

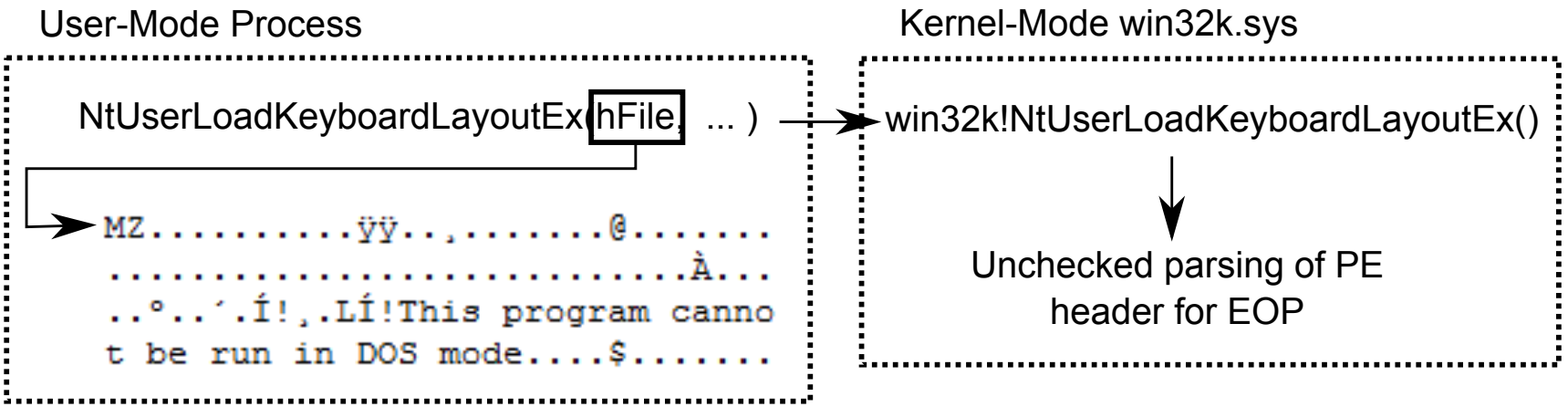
nc SERVER 3389 < termdd_1.dat

- RDP server use after free

Application Fuzzing: API Fuzzing Example

instruder's CVE-2012-0181 related POC win32k.sys

NtUserLoadKeyboardLayoutEx(hFile, 0x0160,0x01AE,&uKerbordname, hKbd, &uStr, 0x666, 0x101)



Meddle: About

Meddle:

- Open source, https://github.com/glmcdona/meddle
- Relatively new project
- Windows only, sorry :(
- Command-line based
- Supports x86, WOW64, and x64 processes
- Framework written in C#
- IronPython for the environment

Meddle: Goals

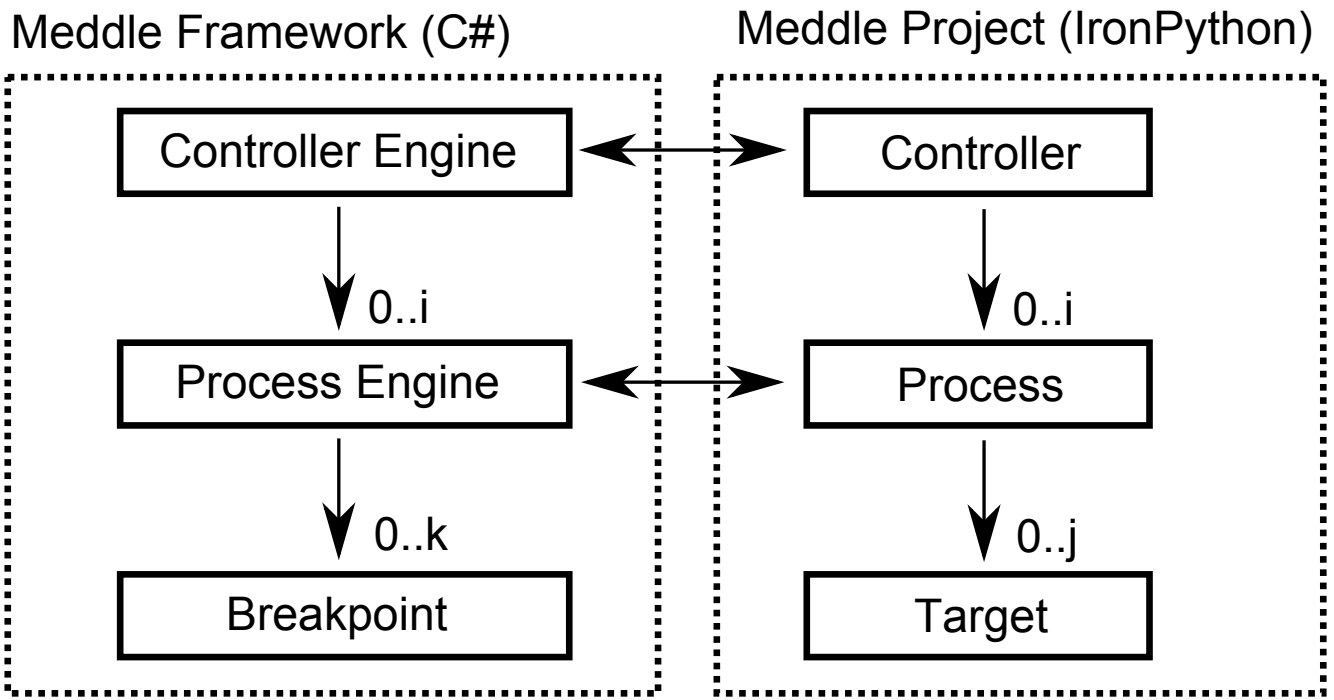
Goals:

- Bring simplicity to fuzzing
- Python for the fuzzing environment
- Extendibility
- Reproducibility

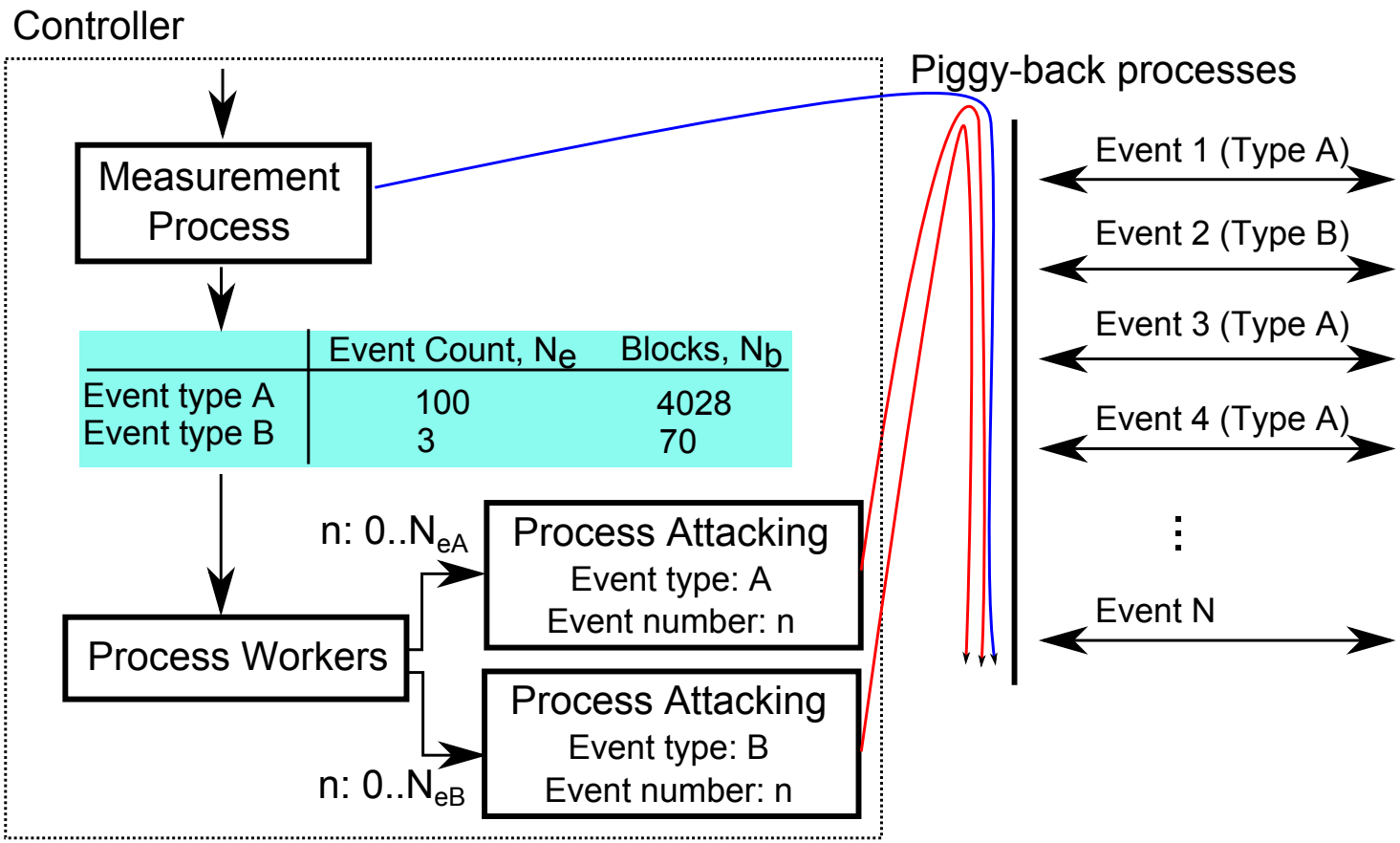
For Simplicity:

- Piggy-back on existing application

Meddle: Structure

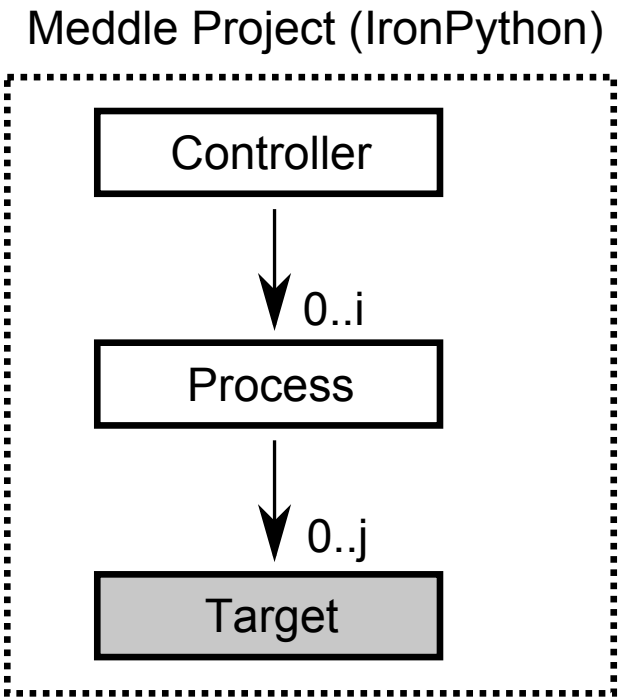


Meddle: Structure



- Equal amount of time on each event type

Target


Meddle Project (IronPython)
Controller
0..i
Process
0..j
Target

Target

```
class Target_Winsock_Send(TargetBase):

  def __init__(self, Engine, ProcessBase):
    # Set options and hook filters

  def breakpoint_hit(self, event_name, address, context, th):
    # Parse arguments and return fuzz blocks for each event
```

Target __init__

```
def __init__(self, Engine, ProcessBase):
  self.Engine = Engine
  self.ProcessBase = ProcessBase
  self.hook_exports = True    # Hook matching exports
  self.hook_symbols = False   # Don't hook matching symbols

  # Libraries to look at
  self.libraries = ["ws2_32.dll"]
  self.libraries_regex = re.compile("a^",re.IGNORECASE)

  # List of function names to add hooks on.
  self.functions = ["send"]
  self.functions_regex = re.compile("a^",re.IGNORECASE)
```

Target breakpoint_hit

```
def breakpoint_hit(self, event_name, address,
                   context, th):
  parameters = [ ... parameter spec ... ]

  [reg_spec, stack_spec] = self.ProcessBase.types.pascal(
                                                parameters )

  arguments = self.Engine.ParseArguments(stack_spec, reg_spec,
                                         context)

  if self.ProcessBase.verbose:
    print arguments.ToString()

  return [arguments.GetFuzzBlockDescriptions(),
          "Winsock Send Event"]
```

Target Parameters

```
parameters = [ {"name": "socket",
                "size": self.ProcessBase.types.size_ptr(),
                "type": None, "fuzz": NOFUZZ },

               {"name": "buffer",
                "size": self.ProcessBase.types.size_ptr(),
                "type": self.ProcessBase.types.parse_BUFFER,
                "type_args": "size", "fuzz": NOFUZZ },

               {"name": "size",
                "size": self.ProcessBase.types.size_ptr(),
                "type": None, "fuzz": NOFUZZ },

               {"name": "flags",
                "size": self.ProcessBase.types.size_ptr(),
                "type": None, "fuzz": NOFUZZ } ]
```

Target Parameter Structures

```
parameters = [ ...
                {"name": "buffer",
                 "size": self.ProcessBase.types.size_ptr(),
                 "type": self.ProcessBase.types.parse_BUFFER,
                 "type_args": "size", "fuzz": NOFUZZ }, ... ]

def parse_BUFFER(self, parent, address, extra_name, type_args):
  if type(type_args) is str: # points to argument name
    size = parent.GetMemberSearchUp(type_args).ToInt()
  else: # contains exact size
    size = type_args
  return [ {"name": extra_name + "BUFFER",
            "size": size,
            "type": None, "fuzz": FUZZ } ]
```

Target Arguments

```
arguments = self.Engine.ParseArguments(...)
print arguments.ToString()
```

```
flags at r9:
00 00 00 00 00 00 00 00                                  ........

size at r8:
13 00 00 00 00 00 00 00                                  ........

buffer at rdx:
E0 98 68 04 00 00 00 00                                  ..h.....

buffer.BUFFER at 0x46898E0:
03 00 00 13 0E E0 00 00   00 00 00 01 00 08 00 03        ........  ........
00 00 00                                                  ...

socket at rcx:
58 07 00 00 00 00 00 00                                  X.......

returnAddress at 0x25AF918:
```

Target Arguments

```
arguments = self.Engine.ParseArguments(...)
print "Sent size = %i" % arguments.size.ToInt()
print arguments.buffer.ToString()
```

```
Sent size = 19
buffer at rdx:
90 ED 29 04 00 00 00 00                                  ..).....

buffer.BUFFER at 0x429ED90:
03 00 00 13 0E E0 00 00   00 00 00 01 00 08 00 03       ........  ........
00 00 00                                                 ...
```

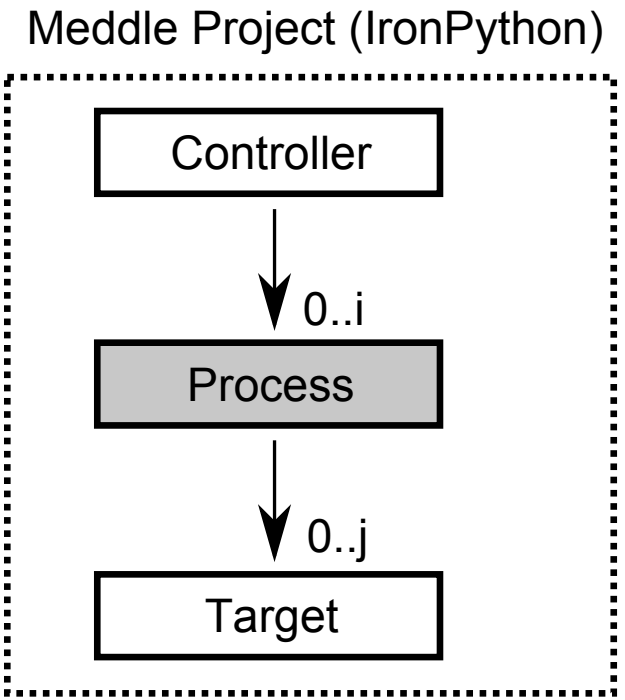
Target breakpoint_hit

```
arguments = self.Engine.ParseArguments(...)
print "Sent size = %i" % arguments.size.ToInt()
print arguments.buffer.BUFFER.ToString()
```

```
Sent size = 19
buffer.BUFFER at 0x4907480:
03 00 00 13 0E E0 00 00   00 00 00 01 00 08 00 03       ........  ........
00 00 00                                                 ...

Sent size = 428
buffer.BUFFER at 0x4907480:
03 00 01 AC 02 F0 80 7F   65 82 01 A0 04 01 01 04       ........  e.......
01 01 01 01 FF 30 19 02   01 22 02 01 02 02 01 00       .....0..  ."......
02 01 01 02 01 00 02 01   01 02 02 FF FF 02 01 02       ........  ........
30 19 02 01 01 02 01 01   02 01 01 02 01 01 02 01       0.......  ........
00 02 01 01 02 02 04 20   02 01 02 30 1C 02 02 FF       .......   ...0....
FF 02 02 FC 17 02 02 FF   FF 02 01 01 02 01 00 02       ........  ........
01 01 02 02 FF FF 02 01   02 04 82 01 3F 00 05 00       ........  ....?...
14 7C 00 01 81 36 00 08   00 10 00 01 C0 00 44 75       .|...6..  ......Du
63 61 81 28 01 C0 D8 00   04 00 08 00 80 07 38 04       ca.(....  ......8.
```

Process


Meddle Project (IronPython)
Controller
0..i
Process
0..j
Target

Process

```
class ProcessRdp(ProcessBase):
  def __init__(self, controller, crashdump_folder,
               breakpoint_handler, pid, unique_identifier,
               verbose):
    # Specific options
    self.path_to_exe = b"C:\\Windows\\System32\\mstsc.exe"
    self.command_line = b"mstsc.exe /v:192.168.110.134"

    # Initialize
    self.initialize(...)

  def on_debugger_attached(self, Engine):
    # Attach the targets to the process
    ...
```

Process

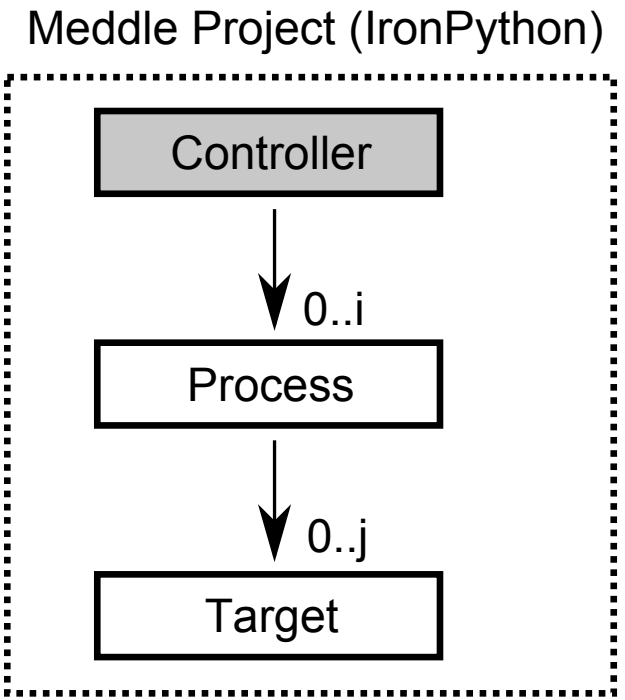
```
def on_debugger_attached(self, Engine):
  # Set the types
  self.Engine = Engine
  self.types = meddle_types(Engine)

  # Add the targets
  #Engine.AddTarget(Target_RDP_RC4)
  Engine.AddTarget(Target_Winsock_Send)

  # Handle process loaded
  Engine.HandleProcessLoaded()

  # Resume the process. Was created suspended.
  if self.start_th >= 0:
    windll.kernel32.ResumeThread(self.start_th);
```

Controller


Meddle Project (IronPython)
Controller
0..i
Process
0..j
Target

Controller Measurement Instance

```
# Perform an initial measurement
mBreakpoint = BreakpointMeasurement()
mProcess = ProcessRdp(self, "C:\\Crash\\", mBreakpoint,
                      -1, 0, True )
self.CEngine.AttachProcess(mProcess)
sleep(5)
measurements = mBreakpoint.measurement
mProcess.stop()
```

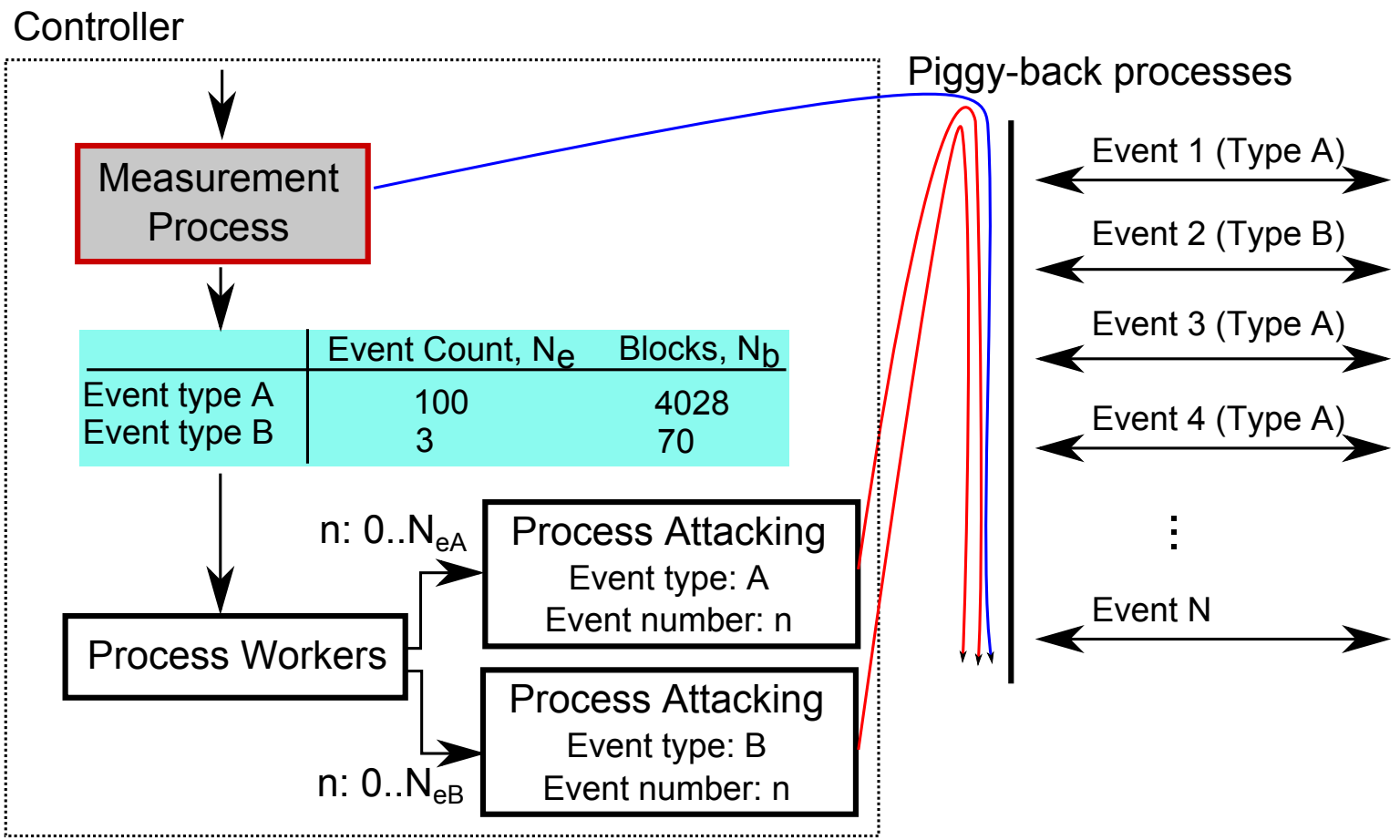
Controller Measurement Instance

```
class BreakpointMeasurement:
  def __init__(self):
    self.measurement = []

  def breakpoint_hit(self, parent, target, event_name,
                     address, context, th):
    [fuzz_blocks, fuzz_name] = target.breakpoint_hit(event_name,
                                                     address, context, th)

    if fuzz_blocks != None:
      # Record the possible attack
      self.measurement += [[target.__class__.__name__,
                            fuzz_name, len(fuzz_blocks)]]
```

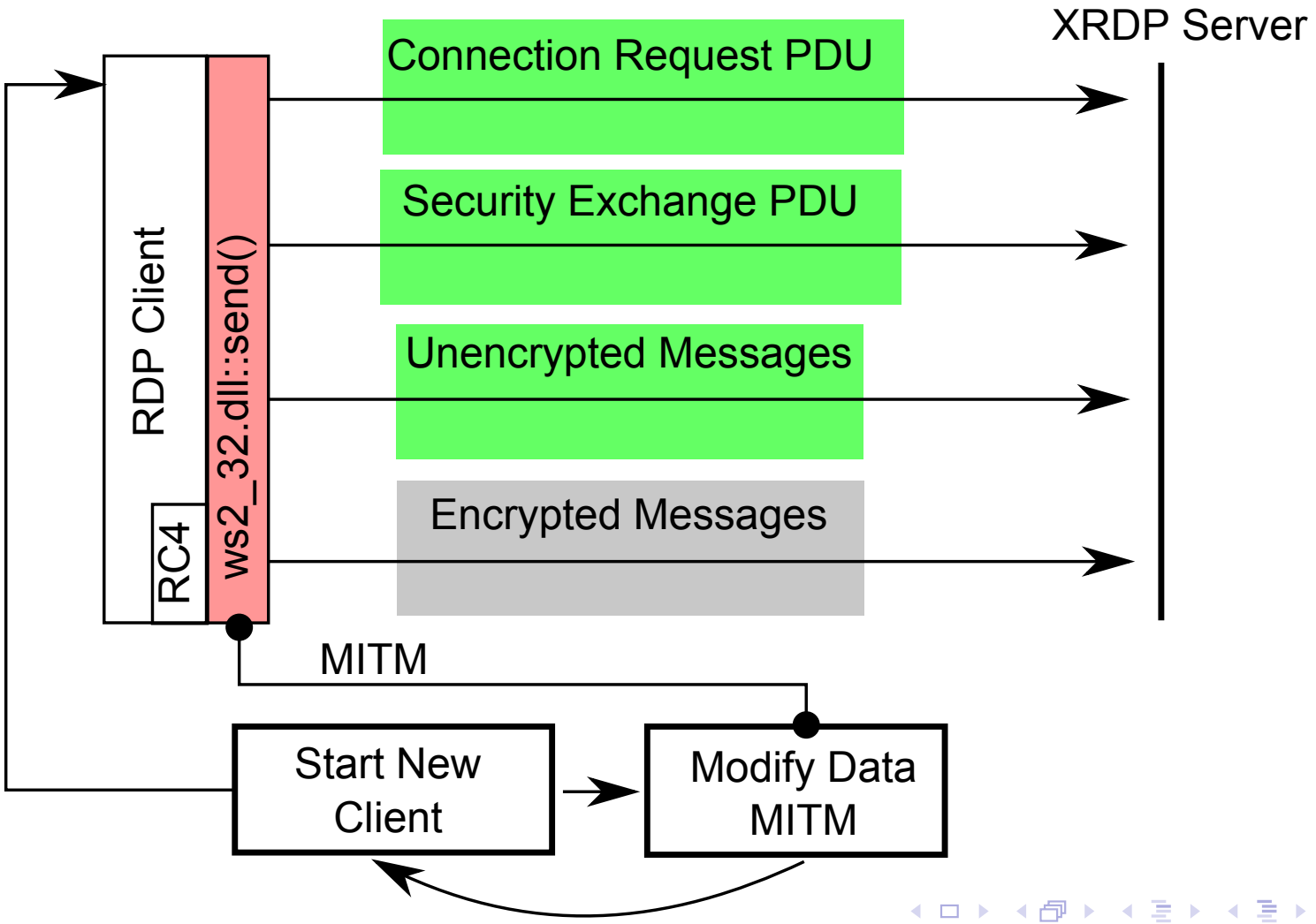
Controller


Controller
Measurement Process
Event Count, N_e
Blocks, N_b
Event type A
Event type B
100
3
4028
70
Process Workers
n: 0..N_{eA}
Process Attacking
Event type: A
Event number: n
n: 0..N_{eB}
Process Attacking
Event type: B
Event number: n
Piggy-back processes
Event 1 (Type A)
Event 2 (Type B)
Event 3 (Type A)
Event 4 (Type A)
Event N

Controller Attack Instance

```
breakpointSeed = self.generator.randint(1,10000000)
newBreakpoint = BreakpointAttack(5, attackEventNumber,
                                 attackEventName, breakpointSeed )
newProcess = ProcessNotepad(self, "C:\\Crash\\", newBreakpoint,
                                -1, uniqueId, False )
self.CEngine.AttachProcess(newProcess)
uniqueId+=1
```

XRDP Demo 1 Diagram

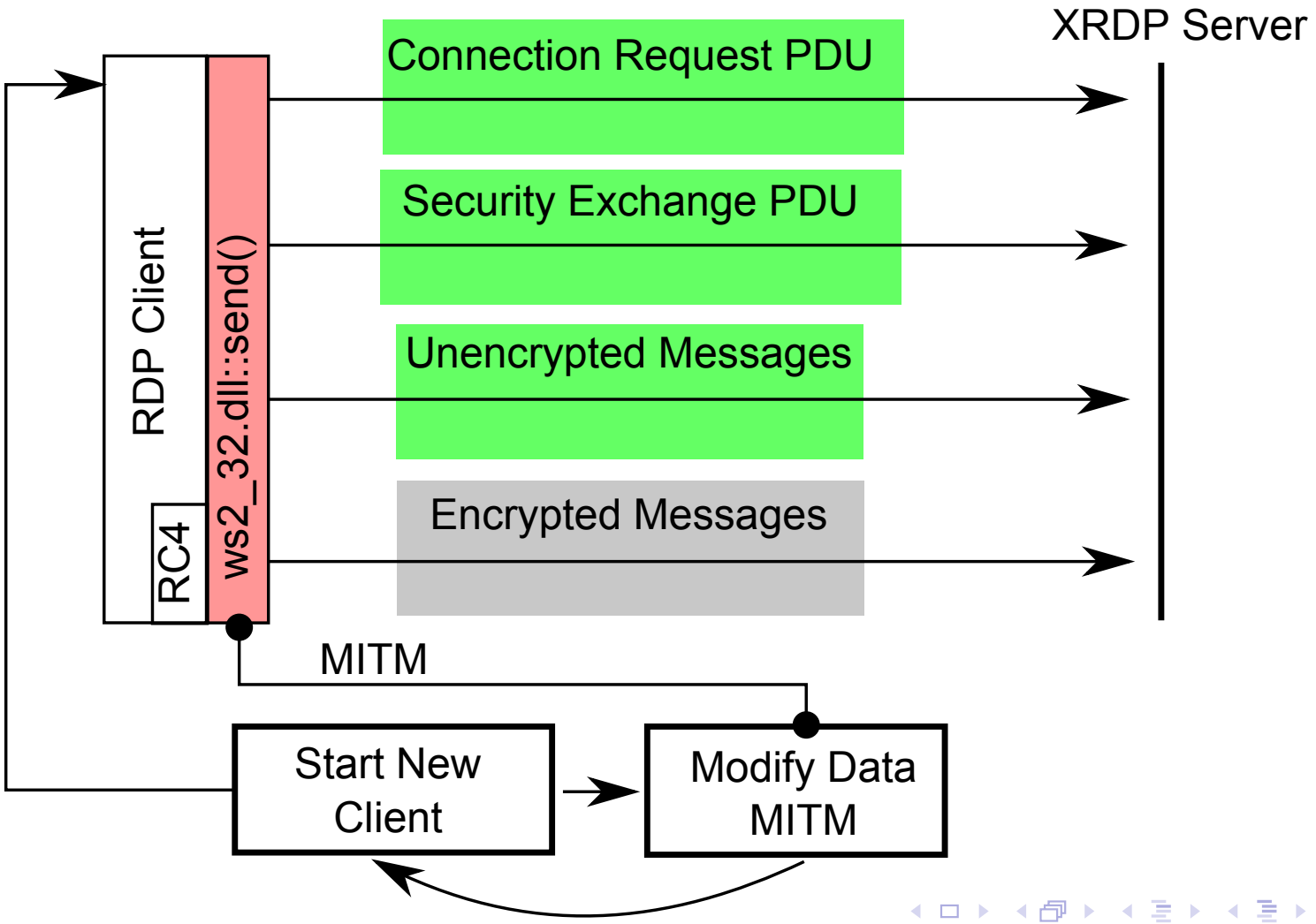


XRDP Demo 1

Demo 1:

- https://github.com/glmcdona/meddle/tree/master/examples/example_mstsc
- Fuzz ws2_32.dll::send() calls from rdp client during connection
- Success: Crash of XRDP server

RC4 Encryption



RC4 Encryption

```
class Target_PrintSymbols(TargetBase):
  def __init__(self, Engine, ProcessBase):
    ...
    self.hook_symbols = True  # Hook pdb symbols
    self.libraries = ["mstscax.dll"]
    ...
    self.functions_regex = re.compile(".*",re.IGNORECASE)
    ...

  def breakpoint_hit(self, event_name, address, context, th):
    print event_name
    return [[],[]]
```

RC4 Encryption

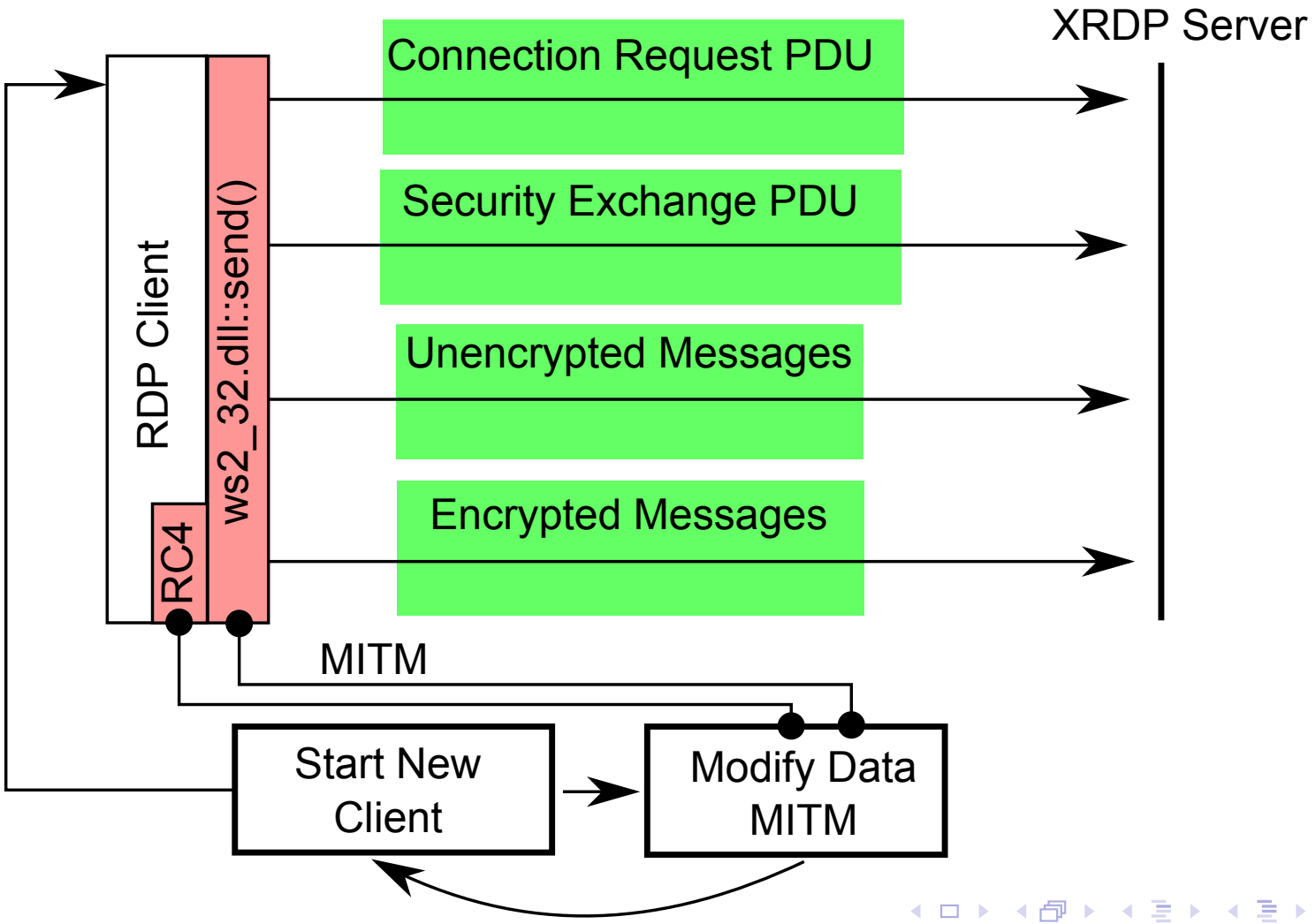
```
...
mstscax.dll::rc4
mstscax.dll::?SendBuffer@CMCS@@UEAAJPEAVITSNetBuffer@@KKKKK@Z
mstscax.dll::?SendBuffer@CTSX224Filter@@UEAAJPEAVITSNetBuffer@@KKKKK@Z
...
mstscax.dll::?RunQueueEvent@CTSThread@@IEAAJPEAVCTSMsg@@@Z
mstscax.dll::?OnTDFlushSendQueue@CTD@@QEAAJPEAVITSAsyncResult@@_K@Z
Sent at 0x4D00EF4:
03 00 00 60 02 F0 80 64   00 01 03 EB 70 52 08 00        ...‘...d  ....pR..
00 00 F4 31 42 EF BD FA   21 3D 36 D1 4C 71 CB 91        ...1B...  !=6.Lq..
CA 03 DB B2 A9 9D B5 86   52 A1 F6 4D 5E 6E C7 8D        ........  R..M^n..
67 B4 D2 53 BE C5 B5 55   98 1C 45 31 13 0A DD CF        g..S...U  ..E1....
06 37 6B 69 C6 60 EF A3   C1 EC F6 AB E5 70 96 73        .7ki.‘..  .....p.s
32 9B 4E ED 7D 40 0E A4   C7 20 F2 A3 69 15 0F 9A        2.N.}@..  . ..i...
```

RC4 Encryption

```
class Target_RDP_RC4(TargetBase):
  def __init__(self, Engine, ProcessBase):
    ...
    self.hook_symbols = True  # Hook pdb symbols
    self.libraries = ["mstscax.dll"]
    ...
    self.functions = ["rc4"]
    ...

  def breakpoint_hit(self, event_name, address, context, th):
    parameters = [ ... ]
    [reg_spec, stack_spec] = self.ProcessBase.types.pascal( para
    arguments = self.Engine.ParseArguments(stack_spec, reg_spec,
    return [arguments.GetFuzzBlockDescriptions(), "RC4 buffer"]
```

XRDP Demo 2 Diagram



XRDP Demo 2

Demo 2:

- https://github.com/glmcdona/meddle/tree/master/examples/example_mstsc
- Fuzz ws2_32.dll::rc4() calls from rdp client during connection
- Success: Crash of XRDP server

Received Data More Complicated

```
def breakpoint_hit(self, event_name, address, context, th):
  if event_name == "ws2_32.dll::recv":
    ...
    self.Engine.AddBreakpoint(self,
          arguments.returnAddress.ToPtr(), "return")
    self.buffers[str(th)] = arguments
    ...
  elif event_name == "return":
    ...
    # Parse the return value and read output buffer
    ...
```

Vulnerabilities

XRDP v0.60 and below vulnerable. Some RCE before authentication:

- Buffer-overflow in xrdp_mcs_recv_connect_initial()
- Out-of-bounds bitmap cache reference xrdp_cache_add_bitmap()
- Large num_events causes information disclosure and DOS conditions
- Number of channels attack xrdp_sec_process_mcs_data_channels()

Vulnerabilities

```
static int APP_CC xrdp_mcs_recv_connect_initial(
                                        struct xrdp_mcs* self)
{
  int len;
  struct stream* s;
  init_stream(s, 8192); // Fixed size buffer
  ...
  // Overflow. 'len' controlled, copied to fixed buffer
  out_uint8a(self->client_mcs_data, s->p, len);
}
```

DeviceIoControl

```
BOOL WINAPI DeviceIoControl(HANDLE hDevice,
                DWORD dwIoControlCode, LPVOID lpInBuffer,
                DWORD nInBufferSize, LPVOID lpOutBuffer,
                DWORD nOutBufferSize, LPDWORD lpBytesReturned,
                lpOverlapped);
```

- Communication to kernel-mode
- Control code to device driver
- Input and output buffer
- eg. low level disk read/write

Devices Communication

Run Notepad → Save As → Network:

```
Number of events being attacked by name:
728     \??\Nsi
64      \??\MountPointManager
20      \Device\LanmanDatagramReceiver
16      \Device\KsecDD
12      \Device\Afd\Endpoint
6       \??\C:
6       \??\NvAdminDevice
4       \??\C:\Users
4       \DEVICE\NETBT_TCPIP_{09AEF42F-B3C7-4854-B4FB-D673B5AD51D5}
4       \??\C:\Users\glmcdona\Documents\Visual Studio 2012\Projects
4       \??\C:\Users\glmcdona\Documents
4       \DEVICE\NETBT_TCPIP_{225A69B0-2055-4DF4-87CB-F3AC50134FE2}
4       \DEVICE\NETBT_TCPIP_{8386C8AD-BABB-4F8E-B85F-3D56FC700D9A}
4       \DEVICE\NETBT_TCPIP_{146BFC43-FB56-4EB3-98D6-E72912BF265E}
```

Demo 3

Using Meddle to dump (or attack) DeviceIoControl:

- https://github.com/glmcdona/meddle/tree/master/examples/example_deviceiocontrol
- ntdll.dll::NtDeviceIoControlFile
- Device handle to name mapping via create hooks
- Dealing with more complex argument types
- Capturing return values/output buffers

Malware Sandbox: Demo 4

Simple sandbox:

- https://github.com/glmcdona/meddle/tree/master/examples/example_sandbox
- Process forking
- Traces
- File read/writes
- Registry changes
- Network activity

Conclusion

Thanks for attending!

- https://github.com/glmcdona/meddle
- Contributors welcome
- Testers needed
- glmcdona@gmail.com

Bibliography I

[1] Immunity, SPIKE, online:http://www.immunitysec.com/resources-freesoftware.shtml

[2] CERT, Basic Fuzzing Framework (BBF), online: http://www.cert.org/vulnerability-analysis/tools/bff.cfm

[3] Godefroid, P., Levin, M. Y., Molnar, D. A. (2008, February). Automated Whitebox Fuzz Testing. In NDSS (Vol. 8, pp. 151-166).

[4] Gorbunov, S., Rosenbloom, A. (2010). Autofuzz: Automated network protocol fuzzing framework. IJCSNS, 10(8), 239. online:http://autofuzz.sourceforge.net/

[5] David Zimmer, COMRaider, online:https://github.com/dzzie/COMRaider

Bibliography II

[6] eSage Lab, IOCTL Fuzzer, online:https://code.google.com/p/ioctlfuzzer/

[7] Google, Fuzzing at Scale, online:http://googleonlinesecurity.blogspot.ca/2011/08/fuzzing-at-scale.html