Another Unbreakable System
Why You Should Care

- **Wide Deployment**
  - Switches
  - Routers
  - Access Points

- **Keys To The Kingdom (MITM)**
  - Control the network traffic
  - Packet sniff in far off lands
  - Modify traffic
  - Break weakly authenticated encryption (passwords, etc)
Some Review: Basic Techniques

- **Stack Overflows**
  - Overwrite return address on the stack

- **Heap Overflows (Pointer Exchange)**
  - Traditionally we use heap chunk linkage
  - Any linked list will do

**Typical linked list delink looks like:**

```c
foo->prev->next = foo->next;
foo->next->prev = foo->prev;
```
Misconceptions

- Routers And Switches Are Just Hardware
- It Is Not Possible To Overflow Buffers On IOS
- There Is No Way To Exploit Buffer Overflows On IOS
- Every Router Is So Different That An Exploit Might Work On One Router But Never Another
Wrong!

- Routers And Switches Run Software On General Purpose CPUs
- Buffers Do Exist And It Is Not So Rare That They Overrun
- Exploitation Is Possible
- Exploitation Can Be Made Reliable And Cross Platform (more on this later)
**IOS Basics**

- **Monolithic**
  - No loadable modules (yet)
  - All addresses are static
  - All addresses are different per build

- **Real Time OS**
  - If you are running you own the CPU (mostly)
  - We have to exit or yield properly or we will crash
  - Once our code is running we have won any race

- **Stability**
  - IOS tends to favor rebooting over correcting errors
A Word On Code Quality

- Much Better Than Most Platforms
  - They check heap linkage
  - They are very aware of integer issues
  - They almost never use the stack
  - They have a process to check all heaps
  - Very old, very well tested code

- Bugs Exist Anyways
  - Green pastures
  - We can get around some checks
  - We will use some of these checks against them
The Dreaded Check Heaps Process

- Walks All Heaps Looking For Bad Linkage
  - Even if our chunk is not freed check heaps will detect bad linkage
  - Is run every 30 to 60 seconds depending on load

- This Is The Main Reason Heap Overflows Can Be Hard
**Rules of Engagement**

- **Stack Overflows**
  - Rare, but if we find one, it's fair game

- **Heap Overflows**
  - They check next and previous pointers
  - We either have to beat check heaps or not offend it
  - We must either know the values for the previous pointer or we must get around this somehow

- **Monolithic Architecture**
  - For heap overflows we must have exact offsets per version (more on this later)
A Look At IOS Heap Structures

- We Can’t Overflow Past Next Pointer
- We Can’t Overwrite Magic Number
  - Magic Number is 0xAB1234BC
- We Can’t Overwrite Red Zone
  - Red Zone value is 0xFD1001DF

<table>
<thead>
<tr>
<th>Magic Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
</tr>
<tr>
<td>Address</td>
</tr>
<tr>
<td>Address</td>
</tr>
<tr>
<td>Address</td>
</tr>
<tr>
<td>Next</td>
</tr>
<tr>
<td>Previous</td>
</tr>
<tr>
<td>Size</td>
</tr>
<tr>
<td>Something</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Red Zone</td>
</tr>
</tbody>
</table>
Big Ups To FX

- His Previous Presentations
  - Blackhat 2002
  - Defcon X

- His Technique
  - Uncontrolled pointer exchange (more on this later)
  - Flash invalidating
  - Guessing previous pointer

- His Limitations
  - Flash invalidation trick only works against very old routers
  - Guessing previous pointer values is usually infeasible
Overcoming The Obstacles

- **Disassembly Ninjitsu**
- **Lots Of Hard Work**
- **Cisco Helps Us Out Some**
  - Built in debugger (sort of)
  - show mem commands
  - show context
  - Forced core dumps
  - debug all
- **Finding The IOS Version**
  - CDP
  - SNMP
  - Read Only Buffer Overflows
Getting IOS In A Disassembler

- Use A Core Dump Image
  - This will show you memory contents of the system during runtime

- Decompress The Firmware Image
  - Stuffit expander
  - WinRar
  - Fixup the ELF header

- Be Prepared for IDA To Run Dog Slow
Getting Execution

- **Stack Overflows**
  - These just work if you can find them

- **Heap Overflows**
  - We need a pointer exchange
  - Its best if we can overwrite something other than heap linkage
  - Hijack any number of callbacks

- **Using Heap Linkage**
  - We can’t overflow past the next pointer
  - Maybe we could use FX’s uncontrolled pointer exchange method for something useful
Easy Heap Overflows

- **Overwrite Linked List In Same Chunk**
  - Doesn’t clobber heap chunks
  - Take control with pointer exchange
  - Easy and reliable, but somewhat rare

- **Overwrite Linked List In Another Chunk**
  - We are racing against check heaps
  - Our chunk must not be freed
  - We are racing check heaps
  - Very hard in practice unless we can deal with check heaps
Racing Check Heaps
- We have between a few seconds and a minute to get execution or we’ll be busted by check heaps
- Sometimes we can trigger the unlink and force us to win the race
- Sometimes we can’t

Let's Kick Check Heaps In The Nuts
- What if we could make check heaps go away
- What if we could not let the router crash
- This would greatly increase our chances of success
- Let's take a look at how the system crashes
Inside The abort() Routine

```
  stwu  sp, var_18(sp)
  mflr  r0
  stmw  r29, 0x18+var_C(sp)
  stw   r0, 0x18+arg_4(sp)
  lis   r9, (crashing_already_ >> 16)
  lwz   r0, (crashing_already_ & 0xFFFF)(r9)
  cmpwi r0, 0
  bne   loc_80493D18     # return
```
I Never Liked Check Heaps Anyways

- Use Uncontrolled Pointer Exchange To Trick System Into Thinking It Is Already Crashing
  - Router can no longer crash synchronously
  - Check heaps will eventually be killed due to CPU hog watchdog

- This Buys You A Few Minutes
  - The system will still eventually crash on an unhandled exception anyways

- This Gives The Potential To Exploit Arbitrary Heap Overflows
  - After check heaps is dead it may be possible to use uncontrolled pointer exchange to get execution
  - You can now guess previous pointer values and the system can’t crash
Building The Shellcode

- Memory Allocation
  - malloc

- Process Management
  - CreateThread
  - exit

- TTY Management
  - allocateTTY
  - Seting up a tty

- Sockets (well, sort of)
  - TCBCreate
  - Connect
Finding malloc()

```assembly
li r3, 0xC
bl malloc
mr. r4, r3
beq loc_80E44D0C
stw r28, 4(r4)
stw r27, 8(r4)
addi r3, r30, 0xCC
bl sub_8049CA30
lwz r9, 0x0C0(r30)
stbx r28, r9, r27
lwz r9, 0x0C0(r30)
stbx r28, r9, r28
lwz r9, 0x8C(r30)
stbx r27, r9, r28
b loc_80E44E08

loc_80E44D0C:
lis r3, ((aAddressAddFail+0x10000) >> 16) # 0x821289CC # "Address add failed
subi r3, r3, 0x7634 # aAddressAddFail # "Address add failed due to no memory"
bl printf_
b loc_80E44Dc4
```
Finding CreateThread()
Using `CreateThread()`

```c
void *CreateThread(void *entryPoint,
                    char *name,
                    int something,
                    int dunno);
```
Finding exit()

```
loc_80484098:
    lbz     r0, 0x78(r31)
    cmpwi   r0, 0
    beq     loc_804840c4
    lis     r9, ((kern_err_msg+0x10000) >> 16)
    lwz     r0, (kern_err_msg & 0xFFFF)(r9)
    lis     r3, ((off_81DFFE44+0x10000) >> 16)  # 0x81DFFE44
    subi    r3, r3, 0x1BC  # off_81DFFE44
    lwz     r4, 0xD8(r31)
    lwz     r5, 0x88(r31)
    mtlr    r0
    blrl

loc_804840c4:
    lis     r9, ((CURRENT+0x10000) >> 16)  # this current process?
    lwz     r0, (CURRENT & 0xFFFF)(r9)  # this current process?
    cmpw    r31, r0
    bne     loc_804840dc
    bl      exit
    b       loc_80484148  # return
```
An Example Of TTY Creation

```
loc_80b10a80:
    li      r3, 0x17
    lis     r4, ((pad_io+0x10000) >> 16) # 0x82f0e38
    subi    r4, r4, 0x1c48 # pad_io
    bl      get_ttygroup # not entirely sure what this is, returns something
                # to pass in to allocatetty
    mr.     r30, r3
    ble     loc_80b10a9c
    mr.     r3, r30
    li      r4, 1
    bl      allocateTTY
    mr.     r31, r3
    bne     loc_80b10ac8
```
Using TTY Routines

```c
ttygroup
*getTTYGroup(int twentyOne, io_t *ioStruct);

tty_t
*allocateTTY(ttygroup *group, int one);
```
We Need A Socket

- Too Bad, This Is Not Unix, Its Not Even Close
  - Actually they do have BSD style sockets, they are just never used and are not helpful to us

- TCB’s
  - I don’t know what this stands for, and neither did the people at Cisco I spoke with
  - This is the socket like thing we have to use
  - They seem comparable to sockets, but work in an asynchronous way
Let's See How TCB's Are Used

```
li    r3, 0
addi  r4, sp, 0x140+var_30
li    r5, 0
bl tcp_create  # creates some structure used in their socket like
               # thing...
```

# CODE XREF: sub_805B7434+344↑j
Another Example

```
loc_80558c58:
    mr   r3, r31
    mr   r4, r27
    mr   r5, r25
    mr   r6, r26
    bl   _tcp_connect
    stw  r3, @r29
    cmpwi r3, 1
    mr   r3, r31
    beq  loc_80558c84
    # CODE XREF: tcp_create_connect+7c↑j
    # tcp_create_connect+98↑j
    # tcb
    # address
    # port
    # flag/mode
    # backend of connect functions

loc_80558c7c:
    bl   tcp_close
    # CODE XREF: tcp_create_connect+ao↑j
    # something to do with closing tcb's
```
Using TCB Routines

TCB

*tcp_create_connect1(int zero,
short remotePort,
sockaddr *remoteAddr,
short localPort,
sockaddr *localAddr,
int *error,
int zero);
A Dead Process Tells No Tales

- **Let's Cover Our Tracks**
  - We could flush the logs
  - We could modify the log strings on the heap
  - We could sabotage the logging functions

- **Or We Could Just Kill The Logger Daemon**
  - Some messages still appear on reboot, but only to console as best I can tell
Finding Kill

```
.byte 0
.aNotdead: .string "NOTDEAD"
off_81DFFE88: .long aSys

.long aNoprocess
.long aNoSuchProcessD_0 # "No such process %d"
```

# DATA XREF: RAM:81DFFE48↑o
# DATA XREF: sub_80482744+8C↑o
# sub_80482744+90↑o ...
# "SYS"
# "NOPROCESS"
# "No such process %d"
Shellcode Check List

1. Get Execution
2. Clean Up What We Broke
3. Spawn Process
4. Allocate And Setup TTY
5. Make Connect-Back TCB
6. Start Shell
7. Kill Logger Process
8. Exit Initial Process
9. World Domination
Is This The End Of The World

- Yes And No (Mostly No)
  - Cisco is working on this
  - Keep your firmware images up to date and you will probably be fine
  - Because you have to have different offset for different firmware versions worms would be very difficult to make

- But Then Again
  - Stack overflows do not need to know router versions to gain execution
  - Up coming versions of IOS use “virtual processes” this means that offsets will be static between firmware versions
Questions?