



```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t8, 4
sltu $1, $v0, $t9
beqz $1, loc_2DA24
nop
sub 7, 0
```

Developments in Cisco IOS Forensics

Felix 'FX' Lindner

DEFCON

Las Vegas, August 2008

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqzl $v0, loc_2DA44
move $v0, $0
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($1)
subu $t2, $t0, $t5
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($1)
sw $v0, dword_35A6C
```

Invent & Verify

Agenda

- IP Routing Infrastructure and Cisco IOS
- Cisco IOS Internals
- Debugging and Post Mortem Analysis Today
- A New Analysis Approach
 - Proposal
 - Features
 - Challenges
- Public Offer
- Future Work

```

addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $1, 2
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sltu $t1, $v0, $t8
$3, 1or_2DA24
sub $t1, $t1, $t2

```

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD0
addiu $a1, $v0, 0x10
beqz $v0, $t1, $t2, 2
move $v0, $t1
lw $t1, dword_35A70
lw $t1, dword_35A6C
subu $t2, $t1, $t3
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



IP Routing Infrastructure

- The Internet and corporate networks almost exclusively run on the Internet Protocol
 - IP Version 4 is still prevalent protocol
 - IP Version 6 coming up very slowly
- The design of IP requires intelligent nodes in the network to make routing decisions
 - This is a design principle of the protocol and cannot be changed
 - “Flat” networks have their own issues

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_35AD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $ap, $ra, 0x10
sw $ra, $a0
sw $a0, $a0
lwf $t1, 2($t1)
fsl $f1, 35AB8
lwf $t1, dword_35A6C
lwf $t7, dword_35A6C
lwf $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 0
sllw $t1, $v0, $t8
lwf $t1, 0($t1)
```

Invent & Verify



IP Infrastructure & Security

- All security protocols on top of IP share common design goals:
 - Guarantee end-to-end integrity (some also confidentiality) of the traffic
 - Detect modification, replay, injection and holding back of traffic
 - Inform the upper protocol layers
- None of them can recover from attacks rooted in the routing infrastructure
 - Security protocols cannot influence routing

Invent & Verify



```
move $a0, $t7
lw $a1, dword_35A6C
jal $a0, 0
addiu $a0, $v0, 0
beqz $v0, loc_35A44
move $v0, 0
la $t1, dword_35A70
lw $t0, 0($t1)
subu $t2, $t0, 1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lwf $t1, 2($sp)
lwf $t2, 2($sp)
lw $a0, dword_35A6C
lwf $t1, 2($sp)
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllw $t1, $v0, $t8
sllw $t1, $v0, $t8
sub $t1, $t1, $t2
```

Infrastructure Monoculture

- Cisco Systems' routing platforms form the single largest population of networking equipment today
 - Equivalently distributed in the Internet core, government and corporate networks
 - Many different hardware platforms with different CPUs
 - Large investment sums bound to the equipment
 - Hard to replace
 - All run basically the same operating system
- Protecting this infrastructure is critical
- Therefore, in-depth analysis and diagnostics are of paramount importance

Invent & Verify



```
move $a0, $v7
lw $a1, dword_35A6C
jal $a1, $v0, $v0
addiu $a1, $v0, $v0
beqz $v0, loc_2DA44
move $v0, $v0
la $t1, dword_35A6C
lw $t1, dword_35A6C
lw $t1, dword_35A70
subu $t1, $t1, $t7
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $ep, $ra, $v0
sw $ra, $ra, $v0
sw $a0, $a0, $v0
lwf $t1, 2($t1)
lwf $t1, 2($t1)
subu $t1, $t1, $v0
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, $v0
sll $t1, $v0, $t8
lwf $t1, loc_2DA24
```



Cisco IOS

```
addiu $sp, $ra, -40
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 2
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $v0, $t8
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1
```

- Cisco® Internetwork Operating System®
- Monolithic operating system
- Compile-time linked functionality – the 3 dimensional complexity of IOS
 - Platform dependent code
 - Feature-set dependent code
 - Major, Minor and Release version dependent code
- Several ***tens of thousands different*** IOS images used in today's networks
 - Over 10.000 still officially supported

```
move $a0, $v0
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a0, $a0, 4
beqz $v0, loc_2DA24
move $v0, $0
la $t1, 0($t1)
lw $t1, 0($t1)
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify





Inside Cisco IOS

```

addiu $sp, $ra, 4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t0, $t8
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1

```

- One large ELF binary
- Essentially a large, statically linked UNIX program
 - Loaded by ROMMON, a kind-of BIOS
- Runs directly on the router's main CPU
 - If the CPU provides virtual memory and privilege separation (for example Supervisor and User mode on MIPS), it will not be used

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 4
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



```
addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t1, $t8
beqz $t1, loc_2DA24
nop
sub $t1, $t1, $t8
```

Inside Cisco IOS

- Processes are rather like threads
 - No virtual memory mapping per process
- Run-to-completion, cooperative multitasking
 - Interrupt driven handling of critical events
- System-wide global data structures
 - Common heap
 - Very little abstraction around the data structures
 - No way to force abstraction

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 4
beqz $v0, loc_2DA24
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



The IOS Code Security Issue

- 12.4(16a) with enterprise base feature set consists of 25.316.780 bytes binary code!
 - This is a 2600 with PowerPC CPU
 - Not including 505.900 bytes firmware for E1T1 and initialization
- All written in plain C
- Sharing the same address space
- Sharing the same heap
- Sharing the same data structures
- Sharing millions of pointers

Invent & Verify



```
move $a0, $v0
lw $a0, dword_35A6C
jal $a0
addiu $a0, $v0, 2
beqz $v0, loc_2DA44
move $v0, $0
la $t0, dword_35A6C
lw $t0, 0($t0)
subu $t0, $t0, $v0
sra $t0, $t0, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $sp, -4
sw $ra, 4($sp)
sw $a0, 8($sp)
lwi $t1, 2($sp)
subu $t1, $t1, 2
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t1, $t8
```

The IOS Code Security Issue

- A single mistake in the most unimportant piece of code can influence anything on the system, including kernel, security subsystems and cryptographic code.
- Therefore, **everything** on IOS is a good target for remote code execution exploits in kernel context.

```
move $a0, $t7
lw $a1, 0($a0)
jal sub_2DAB8
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lw $t1, 8($sp)
sub $t1, $t1, 2
lw $t1, 2DAB8($t1)
lw $t1, 2DAB8($t1)
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sll $t1, $v0, $t2
beqz $t1, loc_2DA24
sub $t1, $t1, 2
```

Invent & Verify



Isn't Cisco aware of that?

- Cisco recently started the distribution of the next generation IOS-XR
 - Commercial QNX microkernel
 - Real processes (memory protection?)
 - Concurrent scheduling
 - Significantly higher hardware requirements (as in Cisco 12000 !)
- People never use the latest IOS
 - Production corporate networks usually run on 12.1 or 12.2, which 12.5 is already available
 - Not even Cisco's own engineers would recommend the latest IOS release to a customer
 - That only covers people actively maintaining their network, not everyone running one

Invent & Verify



```
move $a0, $t0
lw $a0, 0($a0)
jal sub_2DA44
addiu $a1, $v0, 2DA44
beqz $v0, $t0, 2DA44
move $v0, $t0
la $t1, dword_35A70
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

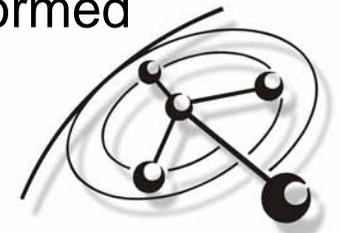
```
addiu $ep, $ra, 4
sw $ra, 0($ep)
sw $a0, 4($ep)
lwf $t1, 0($ep)
lwf $t2, 4($ep)
lwf $t3, 8($ep)
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t8, 4
sllw $t9, $t9, 8
lwf $t9, 0($ep)
subu $t9, $t9, $t8
```

Just, how often are routers hacked?

- Keynote speaker Jerry Dixon at BlackHat Washington DC mentioned not updated routers as a cause for concern
 - Do you know how expensive that is?
- Old vulnerabilities like the HTTP level 16 bug are still actively scanned for
 - The router is used as a jump pad for further attacks
- TCL backdoors are commonly used
- Patched images are not rare
 - IOS images cost money
 - People will use images from anywhere
 - Patching images is not hard
- Lawful Interception is its own can of worms
 - The router's operator is not supposed to know that LI is performed
 - Who watches the watchers?

move \$a0, \$t2
lw \$a0, dword_35A6C
jal sub_2DAB8
addiu \$a1, \$v0, 0x10
beqz \$v0, \$t2
move \$t2, \$t2
la \$t1, dword_35A70
lw \$t1, dword_35A70
lw \$t0, 0(\$t1)
subu \$t2, \$t0, \$t4
sra \$t3, \$t2, 2
sll \$t4, \$t3, 2
addu \$t5, \$v0, \$t4
sw \$t5, 0(\$t1)
sw \$v0, dword_35A6C

Invent & Verify



```
addiu $sp, $ra, -4  
sw $ra, 0($sp)  
sw $a0, 4($sp)  
lui $t1, 3  
jal sub_2DAB8  
lw $a0, dword_35A6C  
lui $t1, 3  
lw $t7, dword_35A6C  
lw $t6, dword_35A70  
subu $t8, $t6, $t7  
addiu $t2, $t6, 4  
li $t9, 0  
li $t9, 0
```

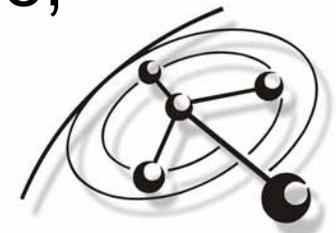


And the future?

- Ever noticed attackers take on the target with the lowest efforts required and the highest return of invest?
 - Windows became just a lot harder
 - UNIXes are hardened, even OS X
 - Infected PCs leave obvious traces
- The question is not:
“Will routers become a target?”
- The question should be:
“Do we want to know when they did?”
- Check the speaking schedule: 3 IOS talks here, 2 of them on attack methods

```
move $a1, $t1  
lw $a0, dword_35A6C  
jal sub_2DAB8  
addiu $a1, $a0, 4  
beqz $v0, loc_2DA44  
move $v0, $a1  
la $t1, dword_35A6C  
lw $t1, 0($t1)  
subu $t1, $t1, $t2  
sra $t3, $t2, 2  
sll $t4, $t3, 2  
addu $t5, $v0, $t4  
sw $t5, 0($t1)  
sw $v0, dword_35A6C
```

Invent & Verify



Summary – Part I

```

addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllw $t1, $v0, $t8
beqz $t1, loc_2DA24
nop
sub_2DAB8

```

- A significant share of the Internet, governmental and corporate networks runs on:
 - one out of several tens of thousands of builds
 - of more or less the same code base
 - in a single process environment

... and we cannot bypass it, even if we could tell that it's compromised

```

move $a1, $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t2, 0($t1)
lw $t3, 4($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Next question: **How can we even tell?**

Invent & Verify



Error Handling and Recovery

- The software architecture of IOS dictates how exception handling has to be done
 - Remember, IOS is like a large UNIX process
 - What happens when a UNIX process segfaults?
- Upon an exception, IOS can only restart the entire system
 - Even on-board, scheduled diagnostic processes can only forcefully crash the system

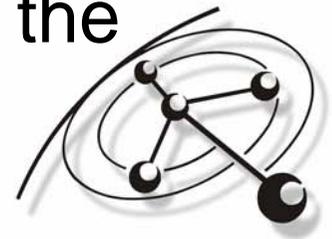
```
move $a0, $t7  
lw $a0, dword_35A6C  
jal sub_3DAB8  
addiu $a1, $v0, 0  
beqz $v0, loc_2DA44  
move $v0, $0  
la $t1, dword_35A70  
lw $t1, dword_35A6C  
lw $t0, 0($t1)  
subu $t2, $t0, $t1  
sra $t3, $t2, 2  
sll $t4, $t3, 2  
addu $t5, $v0, $t4  
sw $t5, 0($t1)  
sw $v0, dword_35A6C
```



Crash Cause Evidence

- Reboot is a clean recovery method
- Reboot destroys all volatile evidence of the crash cause
 - Everything on the router is volatile!
 - Exception: startup configuration and IOS image
- Later IOS releases write an information file called “crashinfo”
 - Crashinfo contains very little information
 - Contents depend on what IOS thought was the cause of the crash

Invent & Verify



```
move $a0, $v0
lw $a1, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0
beqz $v0, $0
move $v0, $0
la $t1, dword_35A70
lw $t1, $v0
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $ep, $ra, 0
sw $ra, 0($ep)
sw $a0, 4($ep)
lwf $t1, 8($ep)
sub_2DAB8 $t1, 3
lw $a0, dword_35A6C
lw $t1, 3($ep)
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 0
sllr $t1, $v0, $t8
beqz $t1, 1cc_2DA24
nop
sub_2DAB8
```

Runtime Evidence

- Crashinfo is only written upon device crashes
- Successful attacks don't cause device crashes
- The available methods are:
 - Show commands
 - Debug commands
 - SNMP monitoring
 - Syslog monitoring

```
move $a0, $v0
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0
beqzl $v0, loc_2D614
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $ap, $ra, 4
sw $ra, 0($ap)
sw $a0, $a0
lw $t1, 2
jal sub_2DAB8
lw $a0, dword_35A6C
lw $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllw $t1, $v0, $t8
beqzl $t1, loc_2D624
```



Show Commands

```

addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sltu $t1, $v0, $t8
beqz $t1, loop_35A74

```

- IOS offers a plethora of inspection commands known as the “show” commands
 - Requires access to the command line interface
- Geared towards network engineers
- Thousands of different options and versions
- Almost no access to code
- 12.4 even limits memory show commands

```

move $a1, $v0
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 4
beqz $v0, loop_35A74
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify





Debug Commands

```
addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
stwu $1, 0($t2)
beqz $1, loc_2DA24
nop
sub $t2, $t2, 4
```

- “debug” enables in-code debugging output
- Debug output has scheduler precedence
 - Too much debug output halts the router
 - Not an option in production environments
- Enabling the right debug output is an art
 - Turn on the wrong ones and you see very little
 - Turn on too many and the router stops working
- Commands depend on the IOS version
- For debug commands to be useful, you have to know what you are looking for **before it happens**
 - Not very useful for security analysis

```
move $a0, $t0
lw $a0, 0($a0)
jal sub_2DAD4
addiu $a0, $a0, 0
beqz $v0, loc_2DA24
move $v0, $0
la $t0, dword_35A70
lw $t0, 0($t0)
subu $t2, $t0, $t0
sw $t3, $t2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



SNMP and Syslog Monitoring

- Commonly accepted method for monitoring networking equipment
- SNMP depending on the implemented MIB
 - Geared towards networking functionality
 - Very little process related information
- Syslog is about as useful for security monitoring on IOS as it is on UNIX systems
- Both generate continuous network traffic
- Both consume system resources on the router
- Then again, someone has to read the logs.

Invent & Verify



```
move $a0, $v0
lw $a1, 0($a0)
jal sub_2DAD4
addiu $a0, $v0, 0x10
beqz $v0, 0($a0)
move $v0, 10
la $t0, dword_35A70
lw $t1, 0($t0)
lw $t2, 4($t0)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $ra, 0x4
sw $ra, 0($sp)
sw $a0, 4($sp)
lwi $t1, 8($sp)
jal sub_2DAD8
lw $a0, dword_35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 0
sllr $t1, $v0, $t2
beqz $t1, 0($t1)
sub $t1, $t1, 1
```

Summary – Part II

```

addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t0, $t8
beq $t1, $t0, 0x0

```

- Identifying compromised routers using today's tools and methods is hard, if not impossible.
- There is not enough data to perform any post mortem analysis of router crashes, security related or not.
- We cannot distinguish between a functional problem, an attempted attack and a successful attack on infrastructure running IOS.

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a0, $t0, 1
beqz $v0, loc_2DA44
move $v0, $t0
la $t1, dword_35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



A (not so) New Approach

- We need the maximum amount of evidence
 - A full snapshot of the device is just enough
- We don't need it continuously
 - We need it on-demand
 - We need it when the device crashes
- We need an independent and solid analysis framework to process the evidence
 - We need to be able to extend and adjust it

```
move $a0, $v0
lw $a0, dword_35A6C
jal $a0
addiu $a0, $v0
beqz $v0, loc_2DA44
move $v0, $0
la $t1, loc_2DA44
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $sp, -4
sw $ra, 4($sp)
sw $a0, 8($sp)
lui $t1, 2
sll $t1, $t1, 2
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t1, $t8
sllr $t1, $t1, $t2
lui $t1, 2
sllr $t1, $t1, $t2
```

Invent & Verify



Getting the Evidence

- Cisco IOS can write complete core dumps
 - Memory dump of the main memory
 - Memory dump of the IO memory
 - Memory dump of the PCI memory (if applicable)
- Core dumps are written in two cases
 - The device crashes
 - The user issues the “write core” command

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 2
beqzl $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $ra, -4
sw $ra, 4($sp)
sw $a0, 8($sp)
lw $t1, 4($sp)
jal sub_2DAB8
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $v0, $t2
beqz $t1, loc_2DA24
```



Core Dump Destinations

- IOS supports various destinations
 - TFTP server (bug!)
 - FTP server
 - RCP server
 - Flash file system (later IOS releases)
- Core dumps are enabled by configuration
 - Configuration commands do not differ between IOS versions
 - Configuration change has no effect on the router's operation or performance

```
move $a0, $t0
lw $a0, dword_35A6C
jal sub_2DAD0
addiu $a1, $v0, 0
beqz $v0, loc_2DA44
move $v0, $0
la $t1, word_35A68
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, 2
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $sp, -4
sw $ra, 4($sp)
sw $a0, 8($sp)
lw $t1, 4($sp)
sub $t2, $t1, 2
sub $t3, $t2, 2DAB8
lw $t4, dword_35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllw $t1, $t1, $t8
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 4
```

Invent & Verify



Analyzing Core Dumps

Disclaimer:

- Any of the following methods can be implemented in whatever your preferred programming language is.
- This presentation will be centric to our implementation: Recurity Labs CIR.

```

addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 2
fe7 sub_2DAB8
lw $a0, dword_35A6C
sw $t1, 8($sp)
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $v0, $t8
beqz $t1, loc_2DA24
nop
sub_2DAB8

```

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Core Dump Analyzer Requirements



```
addiu $sp, $ra, -4  
sw $ra, 4($sp)  
sw $a0, 8($sp)  
lui $t1, 3  
jal sub_2DAB8  
lw $a0, dword_35A6C  
lui $t1, 3  
lw $t7, dword_35A6C  
lw $t6, dword_35A70  
subu $t8, $t6, $t7  
addiu $t2, $t6, 4  
stwu $t1, 0($t2)  
beqz $t1, loc_2DA24  
nop  
sub $t1, $t1, 1
```

- Must be 100% independent
 - No Cisco code
 - No disassembly based analysis
- Must gradually recover abstraction
 - No assumptions about anything
 - Ability to cope with massively corrupted data
- Should not be exploitable itself
 - Preferably not written in C

```
move $a0, $t7  
lw $a0, dword_35A6C  
jal sub_2DAB8  
addiu $a0, $a0, 1  
beqz $v0, loc_2DA44  
move $v0, $0  
la $t1, dword_35A70  
lw $t1, 0($t1)  
lw $t0, 0($t1)  
subu $t2, $t0, $t1  
sra $t3, $t2, 2  
sll $t4, $t3, 2  
addu $t5, $v0, $t4  
sw $t5, 0($t1)  
sw $v0, dword_35A6C
```

Invent & Verify



The Image Blueprint

```

addiu $sp, $ra, -4
sw $a0, 0($sp)
lwi $t1, 2($sp)
fa $a0, sub_2DAB8
lwi $t1, 3($sp)
lwr $t7, dword_35A6C
lwr $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t0, $t8
lwi $t1, 1($sp)
sub $t1, $t1, $t2

```

- The IOS image (ELF file) contains all required information about the memory mapping on the router.
 - The image serves as the memory layout blueprint, to be applied to the core files
 - We wish it were as easy as it sounds
- Using a known-to-be-good image also allows verification of the code and read-only data segments
 - Now we can easily and reliably detect runtime patched images

```

move $a0, $t7
lwr $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 0x10
beqz $v0, 1($sp)
move $v0, 10($sp)
la $t1, dword_35A70
lwr $t1, dword_35A70
lwr $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Heap Reconstruction

```

addiu $sp, $ra, -4
sw $a0, 0($sp)
sw $a0, 4($sp)
lui $t1, 2
jal sub_2DAB8
$ra, dword_35A6C
$1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t0, $t8
beqz $t1, loc_2DA24
nop
sub_2DAB8

```

- IOS uses one large heap
- The IOS heap contains plenty of meta-data for debugging purposes
 - 40 bytes overhead per heap block in IOS up to 12.3
 - 48 bytes overhead per heap block in IOS 12.4
- Reconstructing the entire heap allows extensive integrity and validity checks
 - Exceeding by far the on-board checks IOS performs during runtime
 - Showing a number of things that would have liked to stay hidden in the shadows ☹️

```

move $a0, $t0
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 2
beqz $v0, loc_2DA44
move $v0, $t0
la $t1, dword_35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Heap Verification

```

addiu $sp, $ra, -4
sw $a0, 0($sp)
sw $a0, 4($sp)
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t6, $t8
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1

```

- Full functionality of “CheckHeaps”
 - Verify the integrity of the allocated and free heap block doubly linked lists
- Find holes in addressable heap
 - Invisible to CheckHeaps
- Identify heap overflow footprints
 - Values not verified by CheckHeaps
 - Heuristics on rarely used fields
- Map heap blocks to referencing processes
- Identify formerly allocated heap blocks
 - Catches memory usage peaks from the recent past

```

move $a0, $t1
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a0, $a0, 1
beqz $v0, $v0, 50
move $t1, $a0
lw $t1, $a0
lw $t0, 0($t1)
subu $t2, $t0, $a0
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Process List

```
addiu $sp, $ra, -4  
sw $ra, 0($sp)  
sw $a0, 4($sp)  
lui $t1, 2  
jal sub_2DAB8  
lw $a0, dword_35A6C  
lui $t1, 3  
lw $t7, dword_35A6C  
lw $t6, dword_35A70  
subu $t8, $t6, $t7  
addiu $t2, $t6, 4  
sllr $t1, $t0, $t8  
beqz $t1, loc_2DA24  
nop  
sub $t1, $t1, 1
```

- Extraction of the IOS Process List
 - Identify the processes' stack block
 - Create individual, per process back-traces
 - Identify return address overwrites
 - Obtain the processes' scheduling state
 - Obtain the processes' CPU usage history
 - Obtain the processes' CPU context
- Almost any post mortem analysis method known can be applied, given the two reconstructed data structures.

```
move $a0, $t7  
lw $a0, dword_35A6C  
jal sub_2DAB8  
addiu $a0, $a0, 4  
beqz $v0, loc_2DA44  
move $v0, $a0  
la $t1, 0($v0)  
lw $t1, dword_35A6C  
lw $t0, 0($t1)  
subu $t3, $t1, $t0  
sra $t3, $t3, 4  
sll $t4, $t3, 2  
addu $t5, $v0, $t4  
sw $t5, 0($t1)  
sw $v0, dword_35A6C
```

Invent & Verify



TCL Backdoor Detection

- TCL scripting is available on later Cisco IOS versions
- TCL scripts listening on TCP sockets
 - Well known method
 - Used to simplify automated administration
 - Used to silently keep privileged access to routers
 - Known bug:
 - not terminated when the VTY session ends (fixed)
 - Simple TCL backdoor scripts published
- CIR can extract all TCP script chunks from IOS heap and dump them for further analysis
 - There is still some reversing work to do

Invent & Verify



```
move $a0, $v0
lw $a0, dword_35A6C
jal $a0, 0
addiu $a0, $v0, 0
beqz $v0, 0, 0
move $v0, $v0
la $t1, 0
lw $t1, dword_35A6C
lw $t0, 0
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $ra, -4
sw $ra, $ra, 4($sp)
sw $a0, $a0, 4($sp)
lw $t1, $t1, 4($sp)
fnc $t1, 2
fnc $t1, 2DAB8
lw $t0, $t0, dword_35A6C
lw $t7, $t7, dword_35A6C
lw $t6, $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sll $t9, $t1, 2
sub $t9, $t9, $t8
```

Random Applications

```

addiu $sp, $ra, -4
sw $a0, 0($sp)
sw $a0, 4($sp)
lui $1, 3
jal sub_2DAB8
$ra, dword_35A6C
$1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $1, $v0, $t8
beqz $1, loc_2DA24
nop
sub_2DAB8

```

- Find occasional CPU hogs
- Detect Heap fragmentation causes
- Determine what processes where doing
- Finding attacked processes
 - See examples (Semi-DEMO)
- Research tool
 - Pointer correlation becomes really easy
 - Essential in a shared memory environment

```

move $a1, $v0
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA24
move $v0, $0
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify





IOS Packet Forwarding Memory

```

addiu $sp, $ra, -4
sw $a0, 0($sp)
lwi $t1, 0($sp)
jal sub_2DAB8
lwi $a0, dword_35A6C
lwi $t1, 0($sp)
lwi $t7, dword_35A6C
lwi $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 0
sllr $t1, $t0, $t8
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1

```

- IOS performs routing either as:
 - Process switching
 - Fast switching
 - Particle systems
 - Hardware accelerated switching
- Except hardware switching, all use IO memory
 - IO memory is written as separate code dump
 - By default, about 6% of the router's memory is dedicated as IO memory
 - In real world installations, it is common to increase the percentage to speed up forwarding
- Hardware switched packets use PCI memory
 - PCI memory is written as separate core dump

```

move $a0, $t1
lwi $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x31
beqz $v0, loc_2DA44
move $v0, $0
lwi $t1, dword_35A70
lwi $t0, 0($t1)
lwi $t2, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



IO Memory Buffers

- Routing (switching) **ring buffers** are grouped by packet size
 - Small
 - Medium
 - Big
 - Huge
- Interfaces have their own buffers for locally handled traffic
- IOS tries really hard to not copy packets around in memory
- New traffic does not automatically erase older traffic in a linear way

Invent & Verify



```
addiu $sp, $ra, -4
sw $a0, 0($sp)
lwi $t1, 0($sp)
jal sub_2DAB8
lwi $a0, dword_35A6C
lwi $t1, 3
lwi $t7, dword_35A6C
lwi $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllv $t1, $t0, $t8
lwi $t1, 0($t1)
sub $t1, $t1, $t2
move $a0, $t1
jal sub_2DAB8
addiu $a0, $a0, 4
beqz $v0, loc_2DAA4
move $v0, $t1
lwi $t1, dword_35A6C
lwi $t0, 0($t1)
subu $t1, $t1, $t0
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```



Traffic Extraction

```

addiu $sp, $ra, -4
sw $a0, 0($sp)
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sltu $t1, $t0, $t8

```

- CIR dumps packets that were process switched by the router from IO memory into a PCAP file
 - Traffic addressed to and from the router itself
 - Traffic that was process switching inspected
 - Access List matching
 - QoS routed traffic
- CIR could dump packets that were forwarded through the router too
 - Reconstruction of packet fragments possible
 - Is it desirable?

```

move $a1, $t2
lw $a0, 0($a1)
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, $v0, 10
move $a1, $t0
la $t1, dword_35A70
lw $t1, 0($t1)
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Advanced Traffic Extraction

- Writing core to a remote server uses IO memory
 - Overwrites part of the traffic evidence
- CIR can use a GDB link instead of a core dump
 - Serial GDB protocol allows direct access to router memory via the console
 - Uses Zynamics GDB debug link
- Disconnecting all network interfaces preserves IO and PCI memory contents
 - Using GDB halts the router
- All data is preserved – useful for emergency inspections

```
move $a0, dword_35A6C
lw $a0, dword_35A6C
jal sub_2DA44
addiu $a1, $v0
beqz $v0, loc_2DA44
move $v0, $v0
la $t1, dword_35A6C
lw $t1, dword_35A6C
lw $t1, dword_35A6C
subu $t3, $t2, 2
sra $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $ep, $ra, 4
sw $ra, 4($ep)
sw $a0, 8($ep)
lwf $t1, 4($ep)
lwf $t2, 8($ep)
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t8, 4
sllw $t9, $t9, 2
lwf $t9, 4($t9)
sub $t9, $t9, $t8
```



Traffic Extraction Applications

- Identification of attack jump pad routers
- 0day identification against systems on segmented network interfaces
 - If you got the packet, you got the 0day
- Spoofing attack backtracking
 - One hop at the time, obviously
- LE detection

```
move $a0, $v7
lw $a0, dword_35A6C
jal $a0, 2
addiu $a0, $v0, 2
beqz $v0, $v0, 2
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $ra, -4
sw $ra, 4($sp)
sw $a0, 8($sp)
lw $t1, 4($sp)
jal $t1, 2
subu $t2, $t1, 2
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sll $t9, $t0, $t8
beqz $t9, $t9, 2
sub $t9, $t9, 4
```





Reality Check: March's Vulnerabilities

- “Cisco IOS Virtual Private Dial-up Network Denial of Service Vulnerability”
 - Memory exhaustion / leak
 - Visible by heap usage analysis
- “Cisco IOS User Datagram Protocol Delivery Issue For IPv4/IPv6 Dual-stack Routers”
 - “The show interfaces command can be used to view the input queue size to identify a blocked input interface.”
 - CIR could output all the packets that are still in the queue, even allowing source identification
- “Vulnerability in Cisco IOS with OSPF, MPLS VPN, and Supervisor 32, Supervisor 720, or Route Switch Processor 720”
 - see above

```

move $a2, $a0
lw $a0, 0($a0)
jal $ra, 0($ra)
addiu $a0, $a0, 0x2DAB8
lw $a0, dword_35A6C
lui $t1, 3
st7, dword_35A6C
st6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
st7, $t2, $t8
st7, $t2, $t8
sub $t2, $t2, 1

```

Invent & Verify





Challenges

- The analysis framework has to handle the complexity of the Cisco IOS landscape
 - Hardware platforms
 - Image versions
 - Any-to-Any relation!
- CIR is currently IOS feature set independent
- CIR successfully tested against IOS 12.1 – 12.4
- Official support starts with:
 - Cisco 2600
- Internal testing already covers:
 - Cisco 1700
 - Cisco 2691
 - Cisco 6200
- The platform is the major source of work, testing and verification

```

addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 2
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
lui $t0, 2
sw $t0, 0($t2)
sub $t0, $t0, $t1

```

```

move $a0, $t7
lw $a0, dword_35A70
jal sub_2DAD4
addiu $a1, $v0, 4
beqz $v0, Toc_210744
move $t1, $v0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Summary – Part III

```

addiu $sp, $ra, 4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 2
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllw $t1, $t0, $t8

```

- Writing core dumps is a viable method for obtaining IOS evidence when it is needed.
 - The evidence includes forwarded and received packets.
- An independent analysis framework can distinguish between bugs and attacks, enabling real forensics on IOS routers.
- Recurity Labs' CIR already reliably identifies many types of attacks and IOS backdoors.
 - CIR is work-in-progress
 - CIR's future depends on the feedback we receive from the community.

```

move $a1, $v0
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0
beqz $v0, loc_212A44
move $v0, $0
la $t1, dword_35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify





Availability

```

addiu $sp, $sp, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 2
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t0, $t8
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1

```

1. CIR Online Service (free)
2. CIR Rootkit Detector (free)
3. CIR Professional (non-free)

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify





CIR Online

```

addiu $sp, $ra, -4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 2
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllw $t1, $t0, $t8
subu $t0, $t1, $t2

```

- An analysis framework's quality is directly related to the amount of cases it has seen
 - CIR needs a lot more food to grow up
 - We want to provide it to everyone while constantly developing and improving it
- Free Service: **<http://cir.recurity-labs.com>**
 - Processing on our servers
 - Always using the latest version
 - Right now, CIR Online runs in BETA state

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 0x10
beqz $v0, $t0, $t1
move $v0, $t0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify





CIR Rootkit Detector

- Detection of image modification
- Detection of runtime code modification
- Support for all access layer platforms
- Freely available at <http://cir.recurity-labs.com>
- Currently in BETA state

```

addiu $sp, $ra, 4
sw $ra, 0($sp)
sw $a0, 4($sp)
lui $t1, 2
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $v0, $t8
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1

```

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



At the end, it's all up to you!

- We think CIR could be useful
 - For the networking engineer
 - For the forensics professional
 - To finally know the state of our infrastructure
- We know what we can do
- We need advise on where you want this tool to be in the future

```
move $a0, $v0
lw $a0, dword_35A6C
jal $a0, 0
addiu $a0, 0
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

```
addiu $sp, $sp, -4
sw $ra, 4($sp)
sw $a0, 8($sp)
lw $t1, 4($sp)
jal $t1, 2
sub $t0, $t0, 2DA68
lw $t0, dword_35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t1, $t8
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 4
```



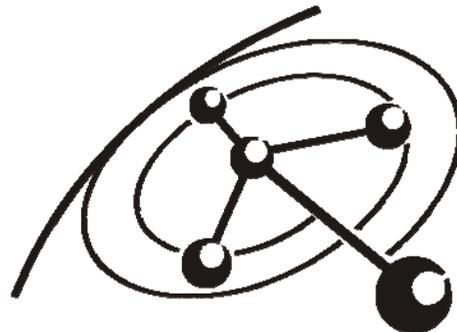


cir.recurity-labs.com

```

addiu $sp, $ra, -4
sw $a0, 0($sp)
lui $t1, 3
sub $t7, $t1, 2
lw $t6, dword_35A6C
subu $t8, $t6, $t7
addiu $t2, $t6, 4
sllr $t1, $t0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1

```



Recurity Labs

Felix 'FX' Lindner
Head

fx@recurity-labs.com

Recurity Labs GmbH, Berlin, Germany
<http://www.recurity-labs.com>

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqzl $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify

