



```
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 2...
```

Port Scanning improved

New ideas for old practices

<http://www.recurity-labs.com>

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

Invent & Verify

Copyright © 2007 Recurity Labs GmbH

Why reinventing wheels?

- The world changes constantly
- The world is full of crappy software
- The requirements of software change
 - When the requirements changed sufficiently, the software no longer fits the purpose
 - Some software didn't fit the purpose to begin with
- Reality is your measure

```
move    $a0, $t7
lw      $a0, dword_35A6C
jal     sub_20A00
addiu   $a0, 0x18
beqz    $v0, loc_20A44
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, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lw      $t1, 3
jal     sub_20A00
lw      $a0, dword_35A6C
lw      $t1, 3
lw      $t7, dword_35A6C
lw      $t6, dword_35A70
subu    $t8, $t6, $t7
addiu   $t2, $t6, 8
sllw    $t1, $v0, $t2
beqz    $t1, loc_20A24
nop
```

Invent & Verify



Redo-Software: When to start?

```
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 $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
```

- Only for people who have a **realistic** chance to actually finish the project
 - The crappy original is still better than the unfinished sequel
- Extrapolate if the problem you are planning to solve is going to get better or worse in the future without your solution.
- Don't make a schedule!
- Make it as good as you possibly can!

Invent & Verify



```
move $a1, $v0
lw $a0, 0x10+var_4($sp)
jal sub_2DAB8
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA64
move $v1, $v0
la $t1, 0x35A6C
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t1, $t0, $t1
sw $t1, 0($t1)
sw $t1, 0($t1)
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Redo-Software: How do you start?

- Set your requirements
 - Remember, they are your requirements
 - Don't try to please everyone you talk to, tell them to fsck off
- Don't import requirements from the existing software
 - Do you really need to be portable?
 - Do you really have to have this feature?
- Don't read too much of the "other" code
 - Think for yourself first
 - Compare your solution with the "other" code later

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($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, 8
sltu $t1, $v0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1
```

```
move $a2, $v0
lw $a0, dword_35A6C
jal sub_2DA24
addiu $a1, $v0
beqz $v0, loc_2DA44
move $v0, $0
la $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



Warning: Redo-Software is uncool!

*“But I want to research
quantumcybercryptofeminism
and its impact on onion-routed
RFID Sex 2.0 !”*

Go ahead !

Invent & Verify



```
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
```

```
addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
$ra0, 0x18+arg_0($sp)
lw      $t1, 3
jal     sub_2DAD4
lw      $a0, dword_35A6C
lw      $t1, 3
lw      $t7, dword_35A6C
lw      $t6, dword_35A70
subu    $t8, $t6, $t7
addiu   $t2, $t6, 8
sllw    $t1, $v0, $t2
lqz     $t1, loc_2DA24
sub     $t1, $t1, $t2
```

A Port Scanner? *Yawn*

- Port scanning is fun for most people
 - Needs random scanning
 - Needs 1337 output
 - Needs 23 different scanning types
- Port scanning is work for some people
 - Needs Accuracy
 - Needs Speed
 - Speed → Time → Money
 - Will use dedicated machines

Invent & Verify



```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_4A5
addiu $a1, $v0
beqz $v0, loc_2DA44
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
```

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal sub_4DAB8
lw $t0, dword_35A6C
li $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sll $t1, $v0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, $t2
```

My hat is off to Fyodor!

- nmap was the first general purpose port scanning tool available
 - Some of you might remember the times when you had to use synscan or similar
 - Nobody really misses them
- nmap introduced many important inventions
 - Granted, most do not belong into a port scanner
 - They are nice and useful anyway
- Redo-Software just doesn't mean the original is bad, worthless or outdated
 - It just means you need something else

Invent & Verify



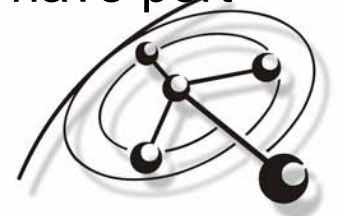
Why not nmap?

- 3 * 255 Hosts in 30 days with nmap
 - I'm actually coming of age
 - Your scanner is not 1337 if it takes 13:37 per host!
 - No, **--disable-waiting-for-things-that-dont-happen** doesn't cut it
- Professionals don't scan hosts that are ...
 - ... powered off
 - ... disassembled
 - ... currently being carried around in the office
- Large scale network scanning is application stocktaking, not vulnerability identification
 - Little interest in the one fully filtered host with only port 23420 open
 - Much interest in how many systems in five Class B networks have port 12345 open

```
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 $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
nop
sub 2...
```

```
move $a0, $v0
lw $a0, dword_35A70
jal sub_2DAB8
addiu $a1, $v0, 0x10
beqz $1, 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



And on a more abstract level...

- All discovery methods depend on a single set of information: the list of open, closed and filtered TCP ports

- OS Fingerprinting
- Service probing
- Banner grabbing

- Accordingly, we need this list first, and quickly at that

```
move $a0, $t7
lw $a0, dword_35A6C
jal $a0, 0
addiu $a1, $v0, 0x10
beqz $v0, loc_2BA44
move $v0, $t1
la $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, -0x18
sw $ra, 0x18+var_4($sp)
sw $r7, 0x18+arg_0($sp)
lw $t3, 3
jal $t3, sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sllr $t1, $v0, $t2
beqz $t1, loc_2BA24
```

Invent & Verify



Our Requirements

- TCP SYN Scanning only, no XMAS trees
- No UDP Scanning
 - UDP scanning is a negative scan method
 - Information value of a UDP scan of a properly firewalled host with UDP services is exactly zero
- Constant access to result data
 - Offloading fingerprinting tasks right when results become available
- Design for embedded use
- Engine design with variable front ends
- Bottom line: Do just one thing, but do it right.

Invent & Verify



```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($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, 8
sltu $t1, $v0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1
```

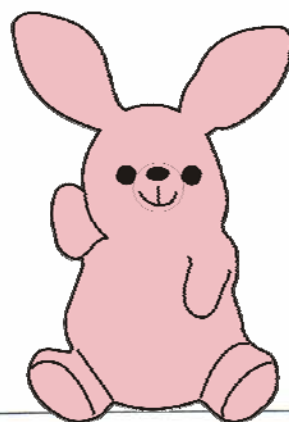
```
move $a0, $t1
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a0, 0
beqz $v0, loc_2DA24
move $v0, $0
la $t1, 0x0
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t1, $t0, $t1
sw $t1, 0($t1)
sw $v0, dword_35A6C
```



```
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 2...
```

PortBunny

A kernel-based port-scanner



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

Invent & Verify

Copyright © 2007 Recurity Labs GmbH

PortBunny

```
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 $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
nop
sub $t2, $t2, 8
```

- Portbunny scans faster by sending more
- Portbunny builds a bridge between **TCP congestion control** and port-scanning.
- Portbunny shows that vanilla TCP-SYN port-scans already leave you with lots of room for research.

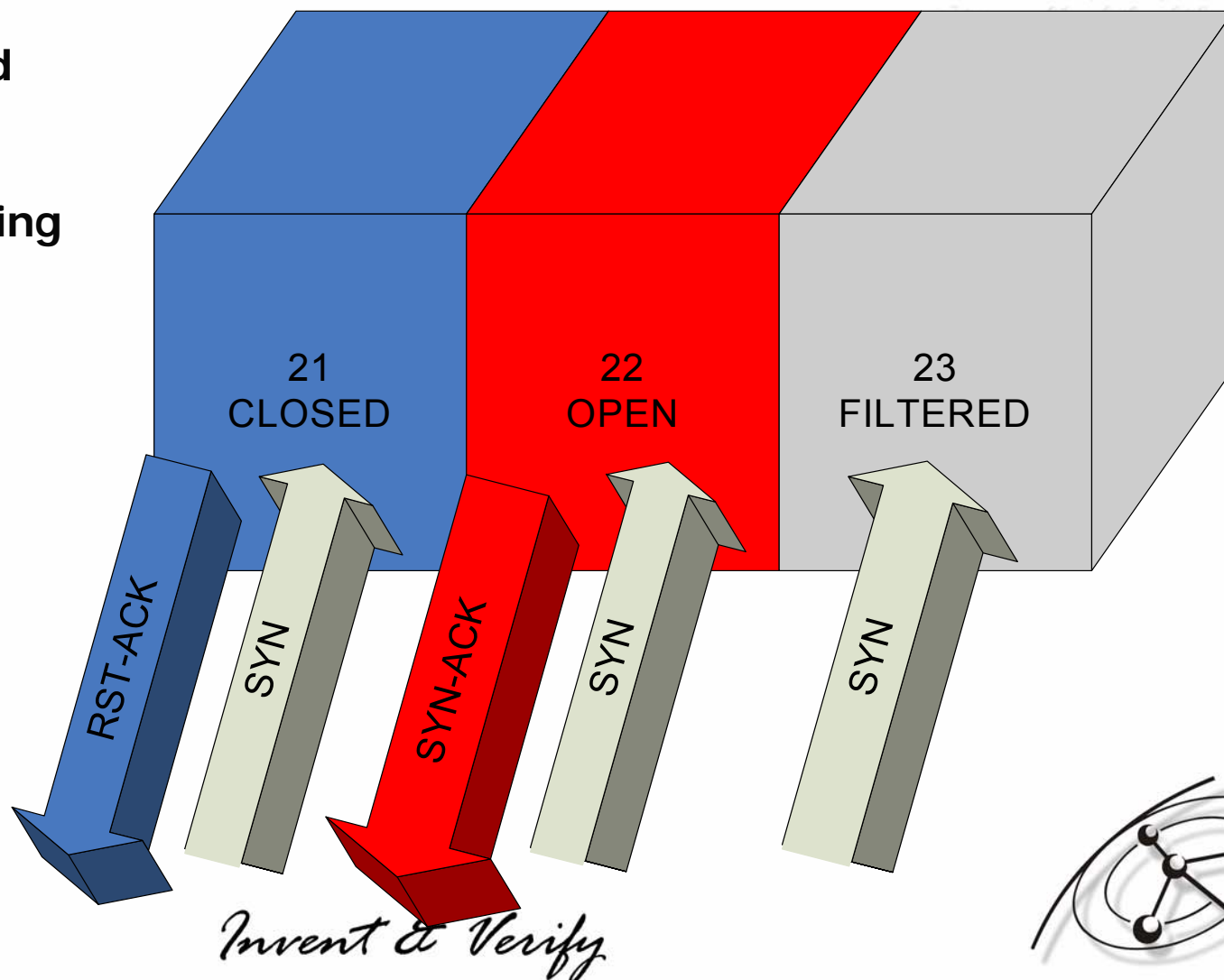
```
move $a0, $v0
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a0, $a0, 1
beqz $v0, loc_2DA24
move $v0, $0
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



1. Port-Scanning - Basics

Identify open, closed and filtered ports by sending connection requests and observing responses.



(TCP-SYN or "half-open"-scanning)

```

move $a0, $t2
lw $t1, dword_35A70
jal $t1, sub_20A04
add $a0, $t1
beqz $v0, loc_20A44
move $v0, $0
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 $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lw $t1, 3
sll $t1, 3
sll $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t9, $t6, 0
    
```

Naive port-scanner

```
foreach p in ports_to_scan:
    send_request_to(p)
    get_response()
```

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
sll $t1, sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sllw $1, $v0, $t2
beqz $1, loc_2DA24
nop
sub $t2, $t2, 1
```

- Won't quite do it.
- Sending as fast as possible may result in dropped packets or even congestion collapse.
- Open/Closed ports will be falsely reported as being filtered.
- Optimal speed may change over time!

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

Invent & Verify



Tell us to slow down, please.

- *Q: Will the network explicitly tell us that we should slow down?*

A: In general, no.

- Exception: ICMP source-quenches,
- Exception: ECN for IPv6

```

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
    
```

```

addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lw      $t1, 3
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, 8
sllw    $t1, $v0, $t2
beqz    $t1, loc_2DA24
sub     $t1, $t1, $t2
    
```

Invent & Verify



What info do we have?

- If a response is received, we have a round-trip-time.
- Packet-drops can be detected given that we know a certain packet should have provoked an answer.

- That's all.

```

move    $a0, $t7
lw      $a0, dword_35A6C
jal     sub_2DAD4
addiu   $a1, $v0, 0x10
beqz    $v0, loc_2DA44
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

```

```

addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lw      $t1, 3
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, 8
sll     $t1, $v0, $t2
beqz    $t1, loc_2DA24

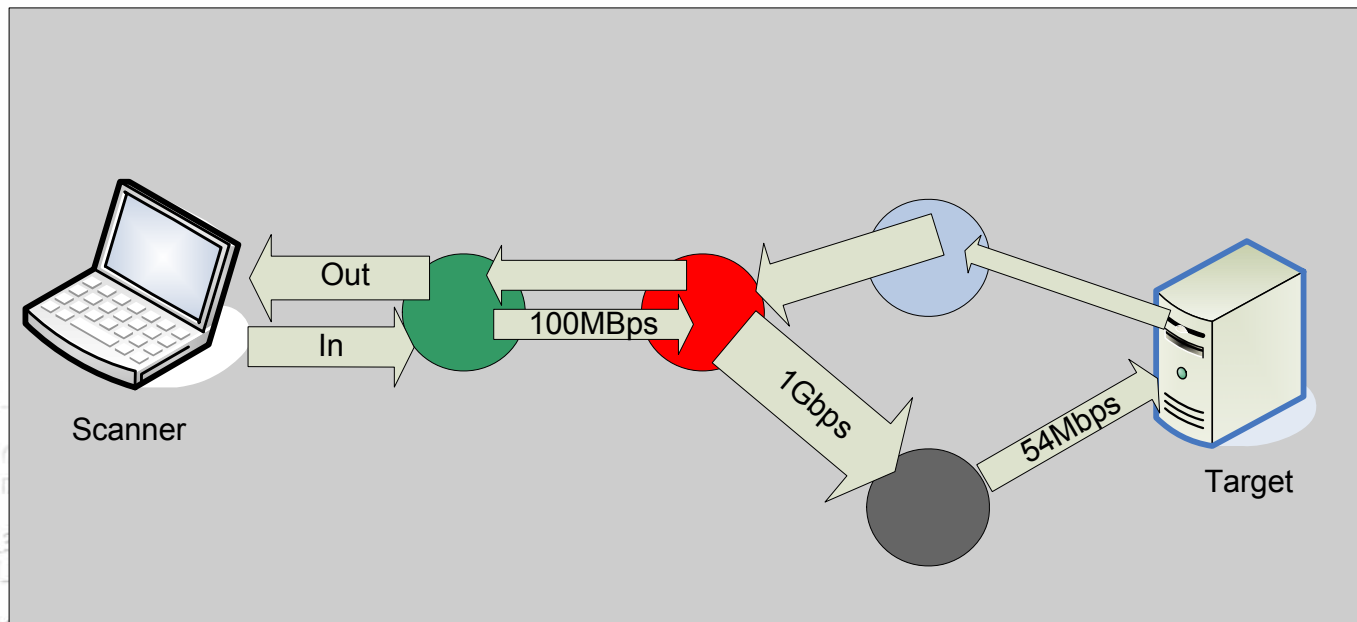
```

Invent & Verify



2. A network model

- Edges: Throughput (Delay), Reliability
- Nodes: Queuing-capacity



Invent & Verify



```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 1
beqz $v0, loc_2DAD4
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

```

```

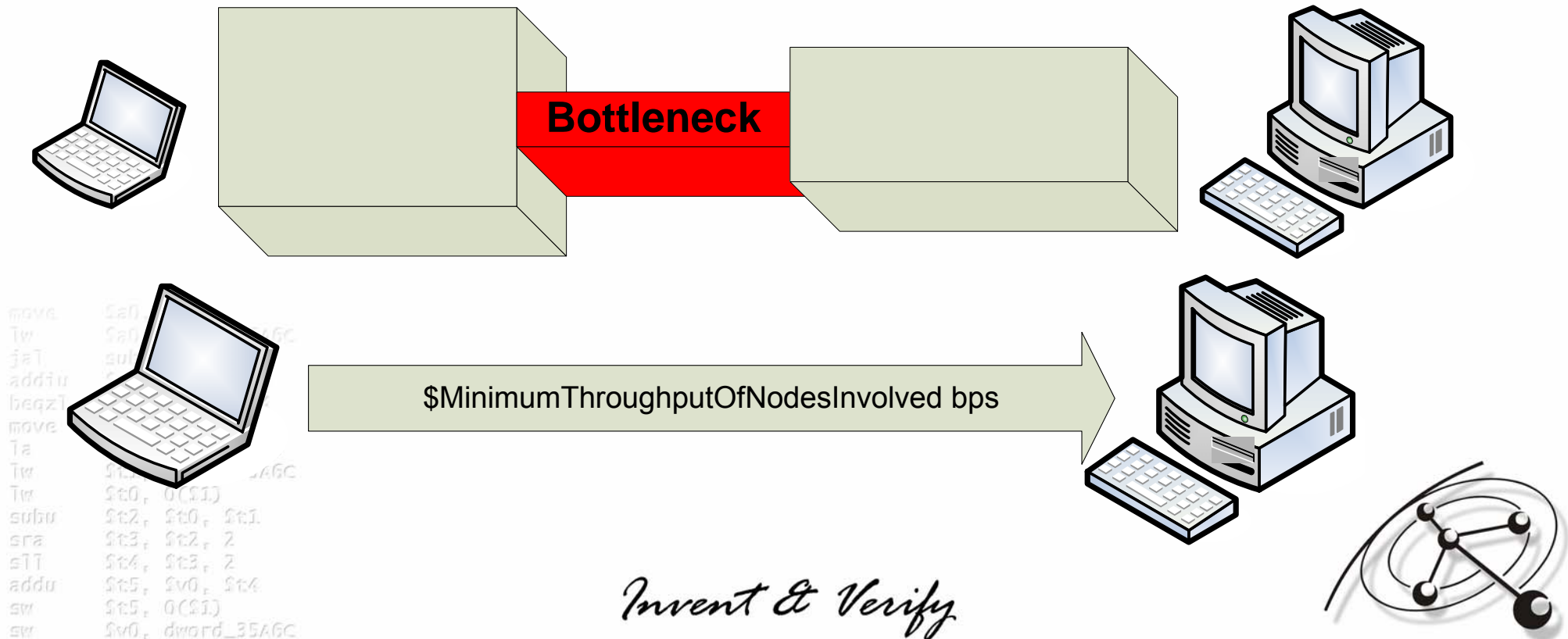
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jlt sub_2DAB8
$ra, dword_35A6C
$1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sll $t1, $v0, $t2
beqz $t1, loc_2DAD4
nop
sub $t1, $t1, 1

```

Simplification

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAE8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
nop
```

- Model implicitly suggested by the term “bottleneck” and by experience from socket-programming.



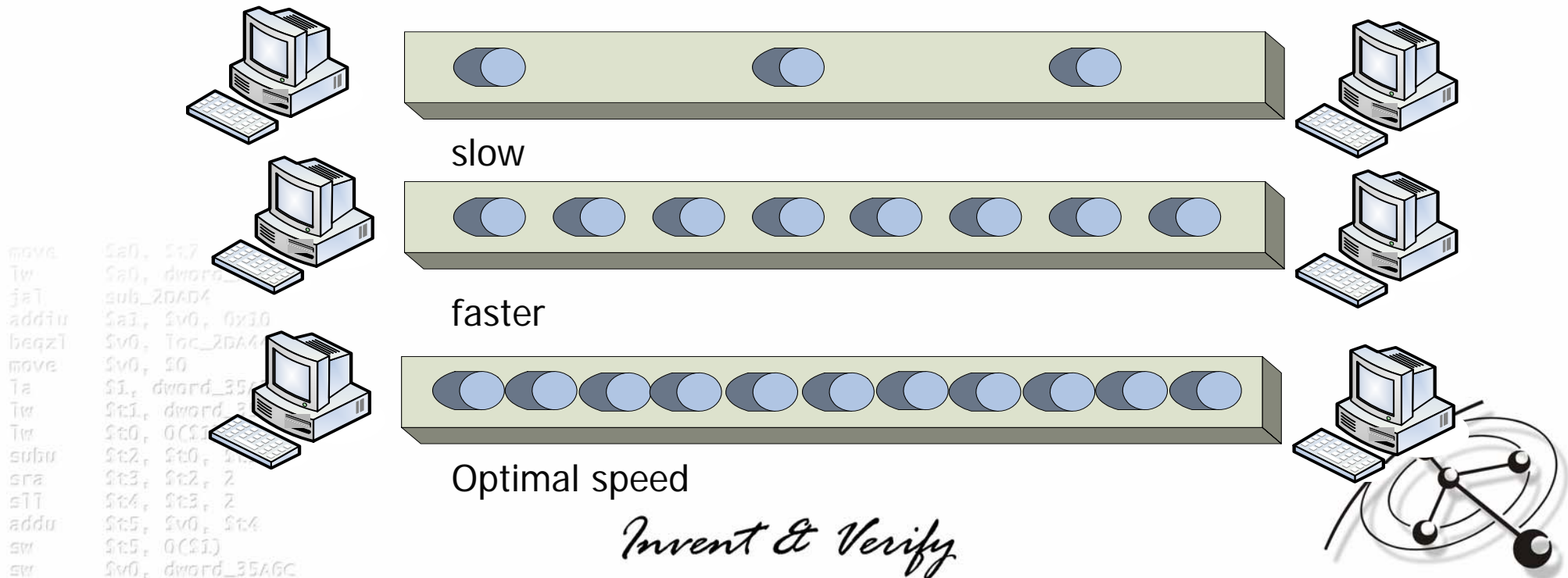
```
move $a0, $v0
lw $a0, 0($v0)
jal sub_2DAE8
addiu $t2, $v0, 8
beqz $t2, loc_2DA24
move $t3, $v0
la $t4, 0($v0)
lw $t0, 0($t1)
lw $t0, 0($t1)
subu $t2, $t0, $t1
era $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Optimal speed

```
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 $t2, $t6, 8
sltu $t1, $v0, $t2
beqz $t1, loc_2DA24
```

- Speed is the number of packets sent per time-frame.

Find the optimal delay.



So much for theory...

- ... but finding the optimal delay will fail in practice!

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($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, 8
sltu $t1, $v0, $t2
beqz $t1, loc_2DA24
nop
sub
```

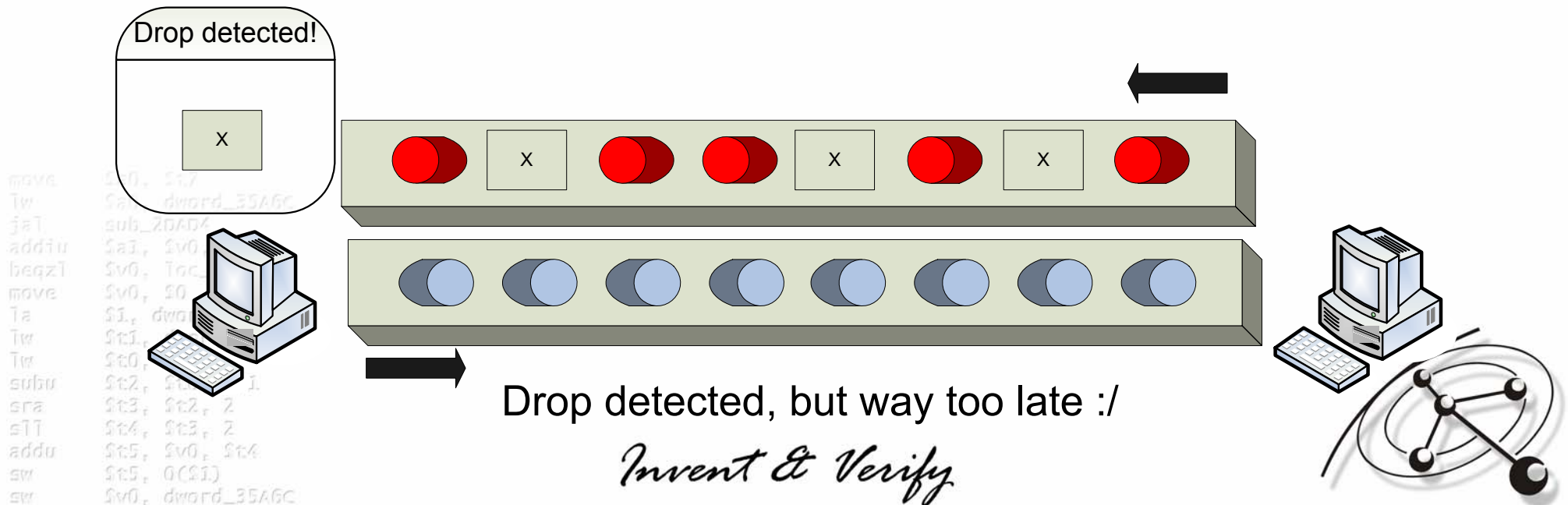
```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

Invent & Verify



The round-trip-time problem

- Dropped packets can't be detected before a complete round-trip-time has passed.
- At that time about rtt/delay other packets have already been sent to maintain the “optimal delay”.



Queuing capacity

- “You can fire 10 packets at a delay of 0 but that doesn’t mean you can do the same with 100 packets.” Why?
- The network has limited ability to queue data.
 - This very Important property of the network suggests a new model.

```

move    $a0, $t7
lw      $a0, dword_35A6C
jal     sub_2DA44
addiu   $a1, $v0, 0x1
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, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lui     $t1, 3
jal     sub_2DAE8
lw      $a0, dword_35A6C
lui     $t1, 3
lw      $t7, dword_35A6C
lw      $t6, dword_35A70
subu    $t8, $t6, $t7
addiu   $t2, $t6, 8
sllw    $t1, $v0, $t2
lw      $t5, 0($t1)
    
```

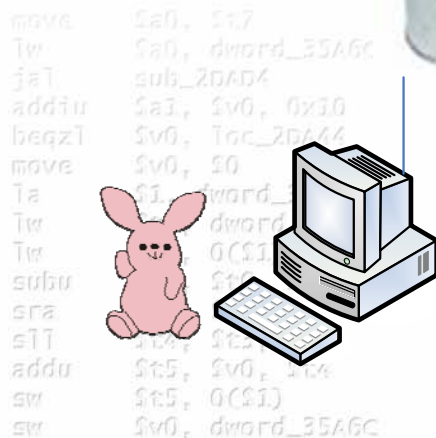
Invent & Verify



The "bucket-model"

Think of each host as a bucket with a hole at the bottom. The optimal speed has been reached when buckets are at all times filled completely.

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
nop
sub $t2, $t2, 8
```



MacOsX



Invent & Verify



New model, new question

- Old question:
“How long should I wait before sending the next packet”
- New question:
“How much data can be out in the network at once?”

```
move    $a0, $t2
lw      $a0, 0($a0)
jal     sub_35A68
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, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lw      $t1, 3
subu    $t2, $t1, 3
lw      $a0, dword_35A6C
lw      $t7, dword_35A6C
lw      $t6, dword_35A70
subu    $t8, $t6, $t7
addiu   $t2, $t6, 8
sllw    $t1, $v0, $t2
beqz    $t1, loc_2DA24
nop
```

Invent & Verify



TCP Congestion Control

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
sll $a1, 20
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sll $t1, $t0, 2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1
```

- TCP congestion control schemes **ask that exact same question!**
- Note: NMAP's timing-code is based on the classic TCP-congestion-control algorithm **"TCP-Reno"**.

```
move $a0, dword_35A6C
lui $a1, 0
addiu $a1, 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
```

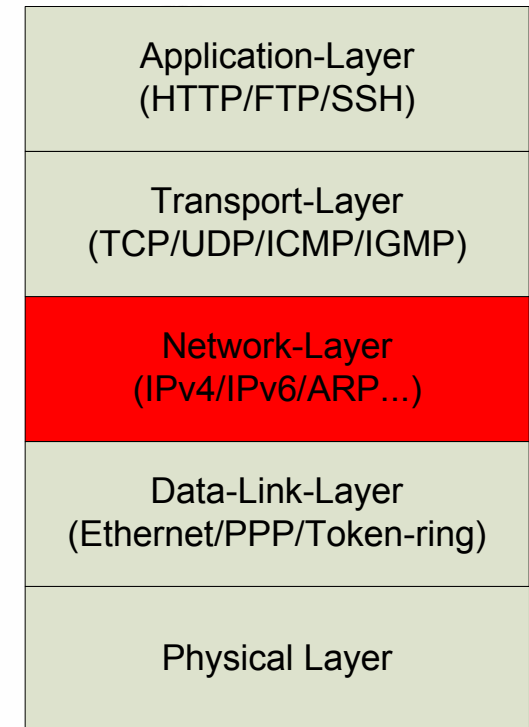
Invent & Verify



Doesn't that work automatically?

```
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 $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
```

- Why do we have to implement congestion control at all?
- Doesn't TCP provide congestion control to upper layers?
- **No established TCP-connection**
- Control the emission of IP-packets which happen to be TCP-SYNs.



Invent & Verify



TCP vs. Port-Scanning

TCP

Receiver acks packets.

Timeouts are error-conditions

Sequence-numbers are used

Port-Scanning

Packets may not produce answers.

Timeouts are not error-conditions

No sequence numbers

Invent & Verify



... in other words:

```
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 $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
nop
sub 2 ...
```

- The TCP-receiver is cooperative
- A port-scanned host is not cooperative.
- Of course, that doesn't mean we can't force it to be.

```
move $a1, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 0x10
beqz $v0, loc_2DA44
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



Triggers - forcing cooperation

- Before starting the scan, find one or more packets which trigger a response.
- PortBunny tries the following:
 - ICMP-Echo Requests
 - ICMP Timestamp Requests
 - ICMP Address-Mask Requests
 - TCP-SYN Port 22/80/139/135 ...
 - UDP Port ...

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $v0, 0x10
beqz $v0, loc_2DAB8
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

```

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($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, 8
sll $t1, $v0, $t2
beqz $t1, loc_2DAB8
sub $t1, $t1, 1

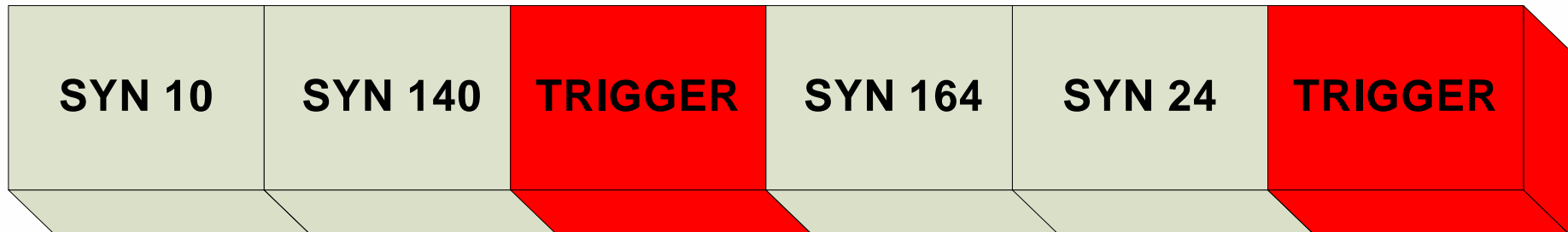
```

Invent & Verify



Inserting triggers into the probe-stream

- Insert these packets into the packet-stream and base your timing-code on the triggers



```

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

```

```

addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($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, 8
sllw    $t1, $v0, $t2
sw      $t1, 0x18+var_4($sp)

```

Invent & Verify



What's that good for?

- Trigger-responses now play the same role Acknowledgments play in TCP's congestion control!
- We receive constant information about the network's performance no matter if it is largely filtered or not!
- A timeout is actually a signal of error!

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
sll $t0, $t1, 2
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sllr $t1, $t0, $t2
beqz $t1, loc_2DA24
```

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_35A4F
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



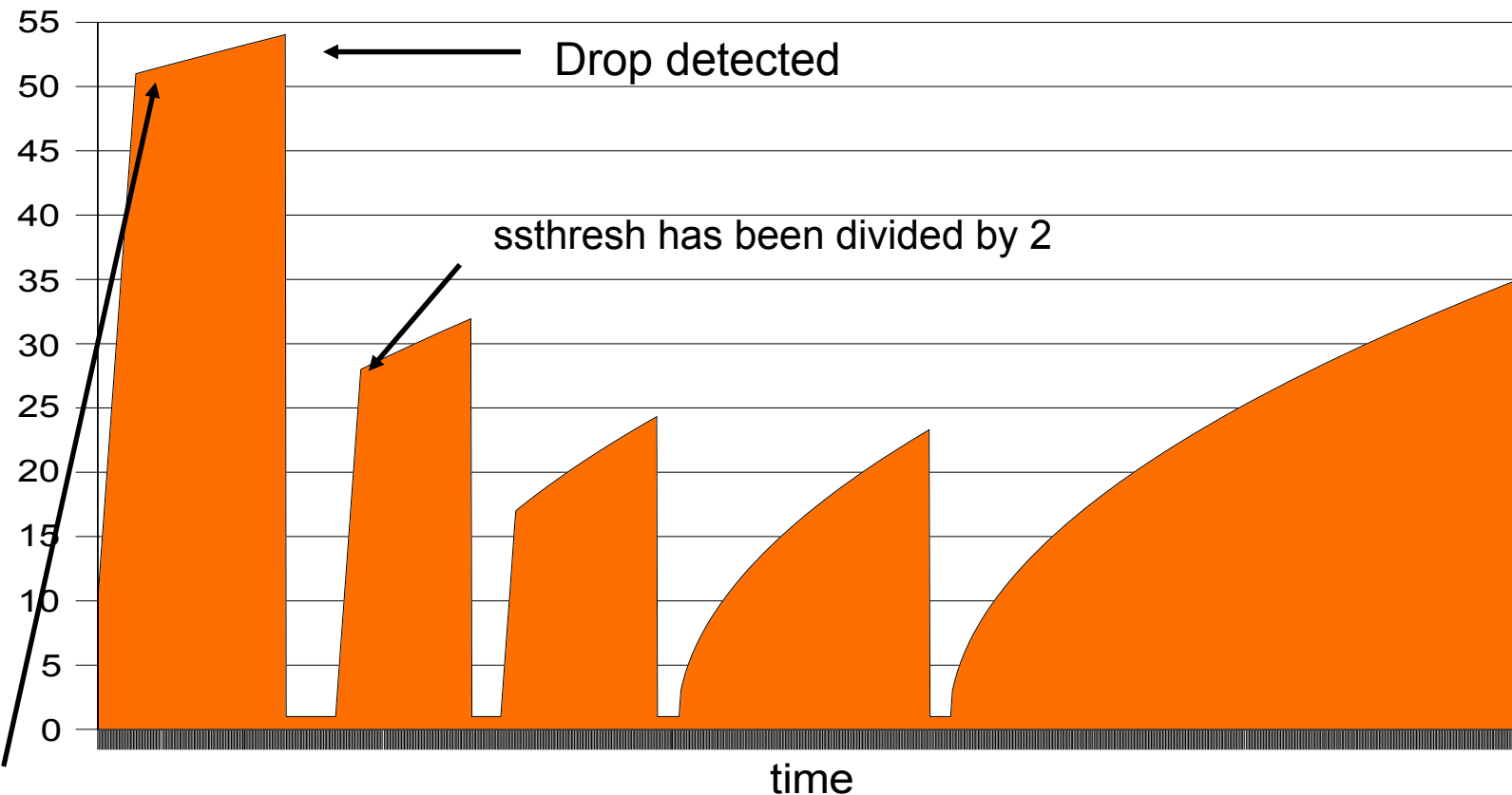
What NMAP Had in Mind

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
sub $t1, $t1, 2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sllv $t1, $t0, $t2
lw $t1, 0x18+var_4($sp)

```

NMAP on a responsive host



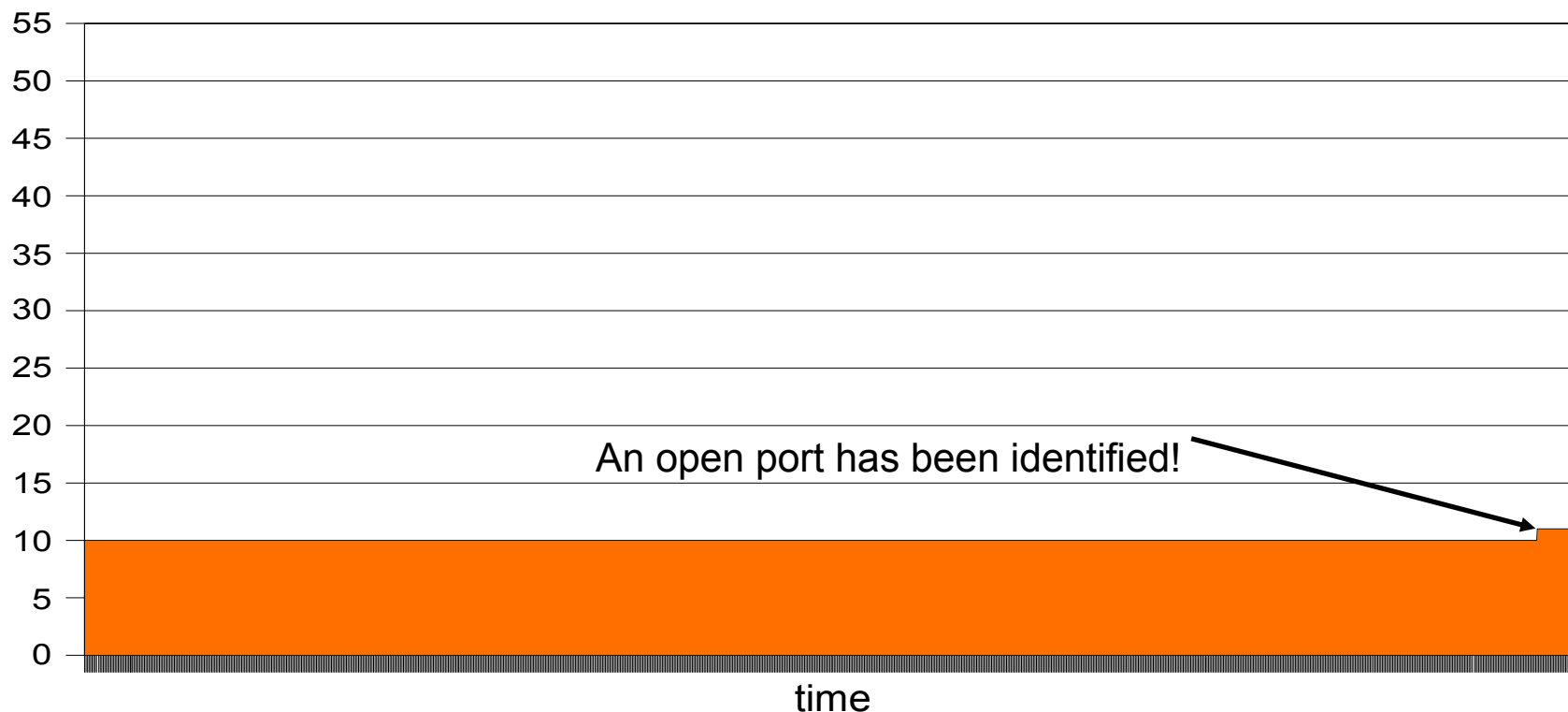
Invent Et Verify

What nmap forgot.

```

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 $t2, $t6, 8
sltu $1, $v0, $t2
bne $1, $zero, loop_35A72
    
```

NMAP scanning a mostly filtered host



```

move $t2, $t0, $t1
lw $t3, $t2, 2
addiu $t4, $t3, 2
move $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
    
```

Invent & Verify



But let's be fair:

```
/* When a successful ping response comes back, it
counts as this many "normal" responses, because the
fact that pings are necessary means we aren't
getting much input. */
```

- If a host has not responded in **5 seconds**, a ping is sent.
- A response is then counted as **3 regular responses**.

■ ***g***

Invent & Verify



```
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 $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DAB24
sub 2DAB24
```

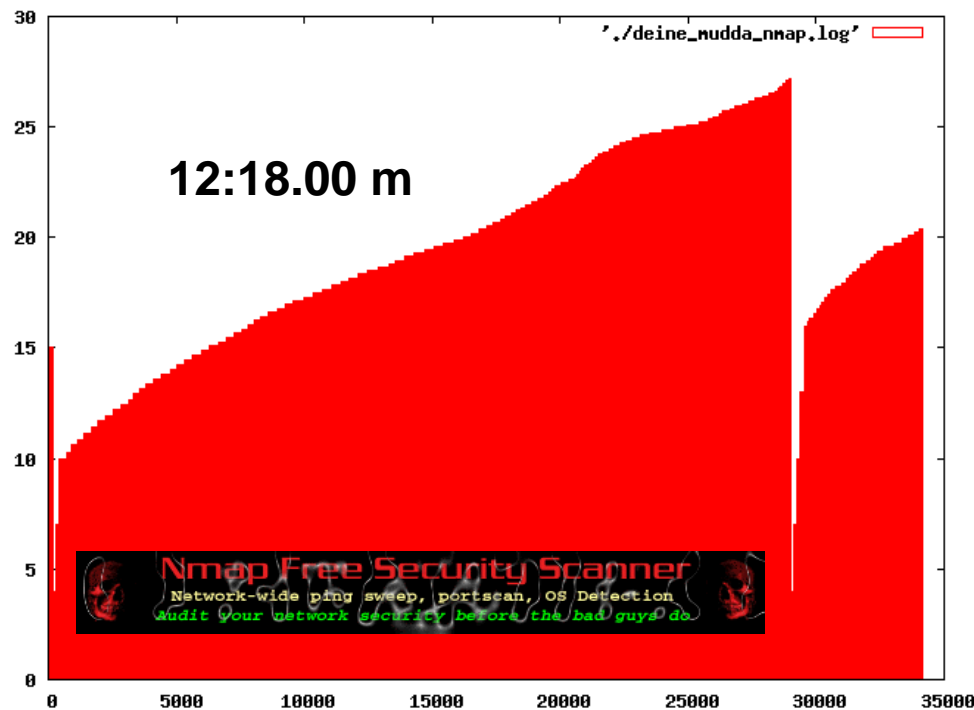
```
move $a0, $t1
lw $a0, dword_35A6C
jal sub_2DAB4
addiu $a0, $a0, 1
beqz $v0, $t0
move $v0, $t0
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

... and then there are
filtered hosts ☺

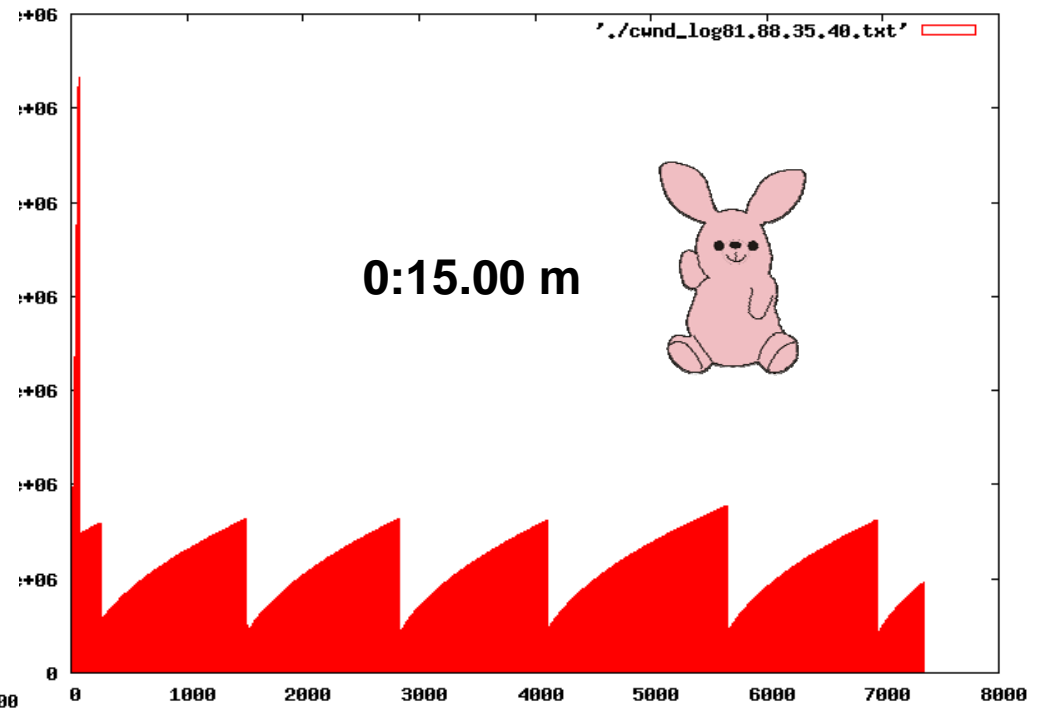
- 66535 ports, mostly filtered, Internet.

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
li $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
li $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sltu $t1, $a0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1
```

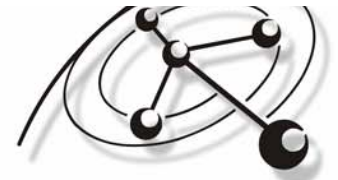
NMAP CMD development



PortBunny CMD development

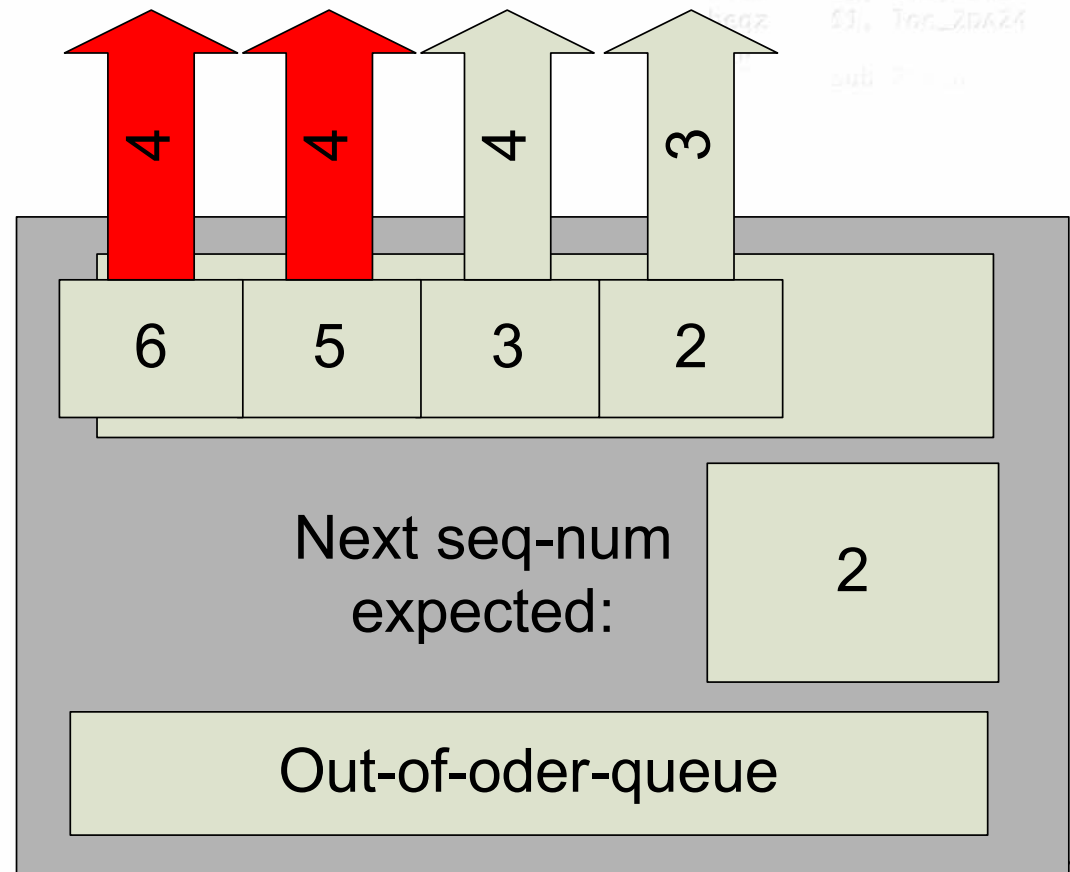


Invent & Verify



Why mention Sequence-Numbers?

- An Ack is sent by the receiver for each packet
- Duplicate Acks indicate packet-loss!
- Fast-retransmit



```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($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, 8
sltu $t1, $a0, $t2
beqz $t1, loc_2DA24
sub $t1, $t1, 1

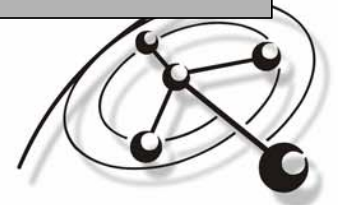
```

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, $a0, 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 $t1, $t2, 1
sll $t1, $t1, 1
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```

Invent & Verify



Trigger Sequence-Numbers

- When integrating sequence-numbers into triggers, an algorithm similar to **fast-retransmit** can be implemented:

Trigger-Response 5
Trigger-Response 6 MISSING
Trigger-Response 7
Trigger-Response 8
Trigger-Response 9

Example:

- Responses for 7, 8 and 9 have been received but there's no response for 6.
- One can assume that 6 has been dropped even if its timeout-value has not been reached!

Invent & Verify



NMAP – Timeout-detection

```
/* A previous probe must have been lost ... */.
```

- NMAP can only detect drops after resending
- If a resent probe produces an answer, obviously, the initial probe was dropped.
- Each probe has its own timeout-clock. That doesn't scale well, so there are interesting hacks to solve this.

Invent & Verify



Consequence

```
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 $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
nop
sub $t2, $t2, 8
```

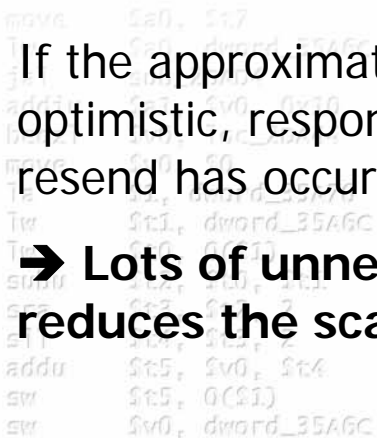
- To stay responsive to drops, NMAP must resend the probe that may have just dropped straight away!
- This makes NMAP extremely vulnerable to the “late-responses”-problem

```
move $a0, $1
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a1, 0
beqz $v0, loc_2DA44
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



Problem



If the approximation is optimistic, response is resent has occurred

➔ Lots of unnecessary retransmissions reduces the scalability

If the approximation is optimistic, response is resent has occurred

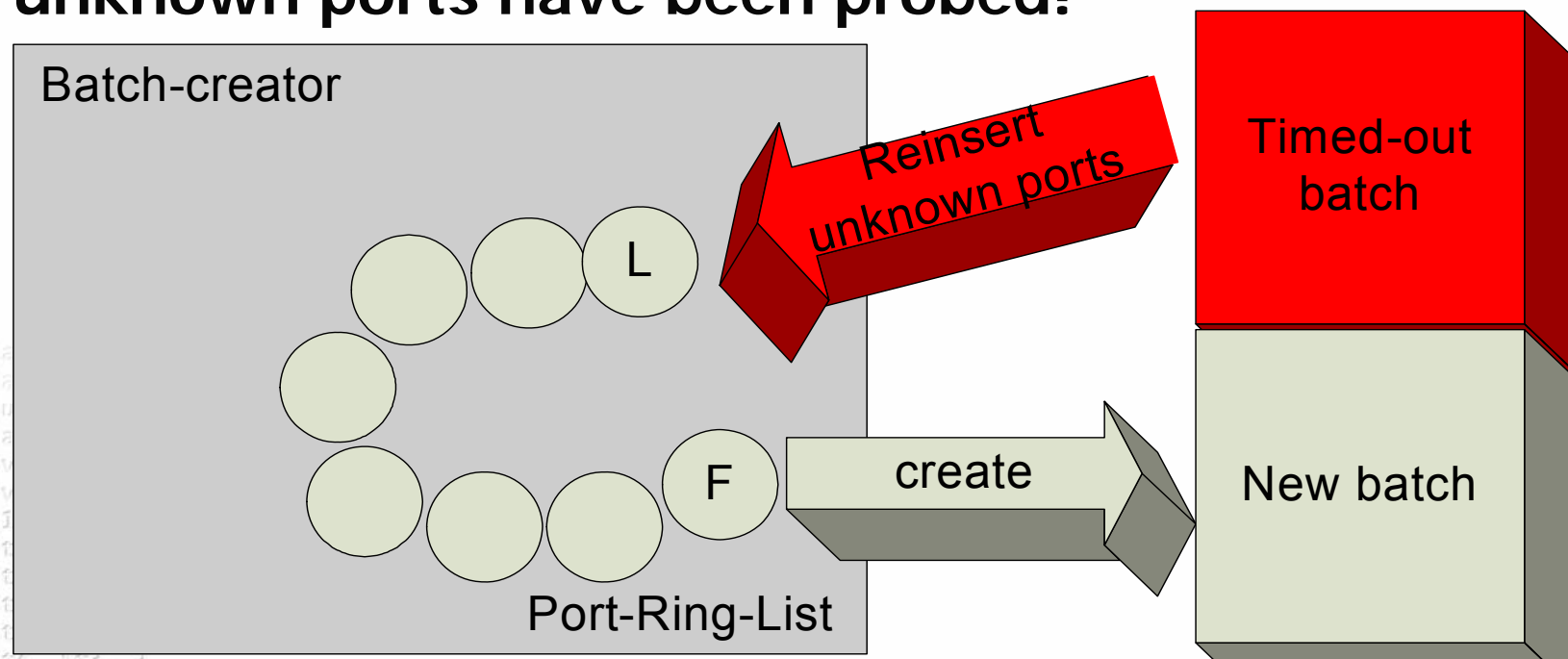
➔ Lots of unnecessary retransmissions reduces the scalability

Invent & Verify

Defeating late-responses (with triggers)

PortBunny does not rely on immediate resends to detect packet-loss!

→ The probe can be resent after ALL other unknown ports have been probed!



Invent & Verify

```

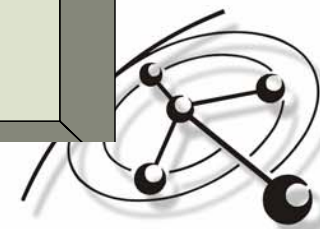
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $t1, 3
jal sub_2DAB8
lw $a0, dword_35A70
subu $t6, $t7
addiu $t2, $t6, 8
sltu $t1, $a0, $t2
beqz $t1, loc_2DA24
sub $t1, $t1

```

```

move $a0, $t2, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C

```



Triggers vs. TCP

```
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, 1
```

TCP

Receiver acks packets.

Timeouts are error-conditions

Sequence-numbers are used

Trigger-based scanning

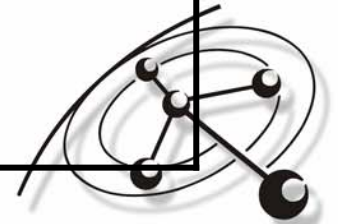
Triggers are acknowledged.

Trigger-Timeouts are error-conditions.

Sequence-numbers are used for all triggers.

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAD4
addiu $a1, $v0, 1
beqz $v0, loc_2DAD4
move $v0, $0
la $1, dword_35A70
lw $t1, dword_35A70
lw $t0, 0($t1)
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 8
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

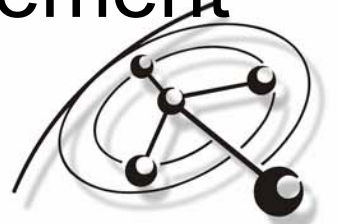
Invent & Verify



Benefits of trigger-use

- Filtered hosts can be scanned properly
- Packet-drops can be detected much earlier leading to better responsiveness to drops.
- Immediate probe resends are not necessary anymore which helps reduce useless extra traffic.
- Port-Scanning has been ported to the tcp-congestion control domain! We can implement any TCP-congestion-control scheme!

Invent & Verify



Problems with triggers

- Not all triggers have the same quality:
 - ICMP-triggers and UDP-triggers could be rate-limited while probes aren't.
 - TCP-triggers are the best available triggers.
 - QoS might be a problem, some times
- A host may not respond to any supported trigger.

Invent & Verify



```

addiu    $sp, -0x18
sw       $ra, 0x18+var_4($sp)
sw       $a0, 0x18+arg_0($sp)
lwf      $t1, 3
jal      sub_2DAB8
sw       $a0, dword_35A6C
lwf      $t1, 3
lw       $t7, dword_35A6C
lw       $t6, dword_35A70
subu     $t8, $t6, $t7
addiu    $t2, $t6, 8
sllw     $t1, $v0, $t2
beqz     $t1, loc_2DA24
nop
sub      $t1, $t1, $t8

```

```

move     $a0, $t7
lw       $a0, dword_35A6C
jal      sub_2DAD4
addiu    $a1, $v0, 0x10
beqz     $v0, loc_2DA44
move     $v0, $a0
la       $t1, 0x00000000
lw       $t1, dword_35A6C
lw       $t1, 0($t1)
subu     $t2, $t1, $t7
sra      $t3, $t2, 2
sll      $t4, $t3, 2
addu     $t5, $v0, $t4
sw       $t5, 0($t1)
sw       $v0, dword_35A6C

```

Fixes

- Try to find TCP-SYN-triggers first and use ICMP and UDP-triggers as a fallback-solution.
- If a TCP-SYN-trigger can be found at scan-time, add it to the list of triggers in use and discard fallback-triggers.

```

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

```

```

addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($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, 8
sllw    $t1, $v0, $t2
beqz    $t1, loc_2DA24

```

Invent & Verify

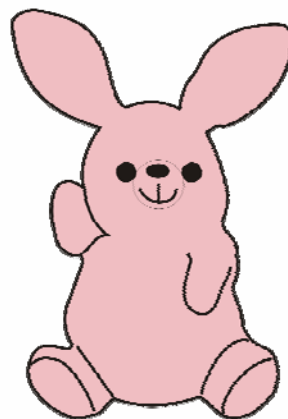


Racing on responsive hosts

- PortBunny sends 10% more data because of the triggers? Can it still compete with NMAP on responsive hosts?



VS



Invent & Verify



```
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
```

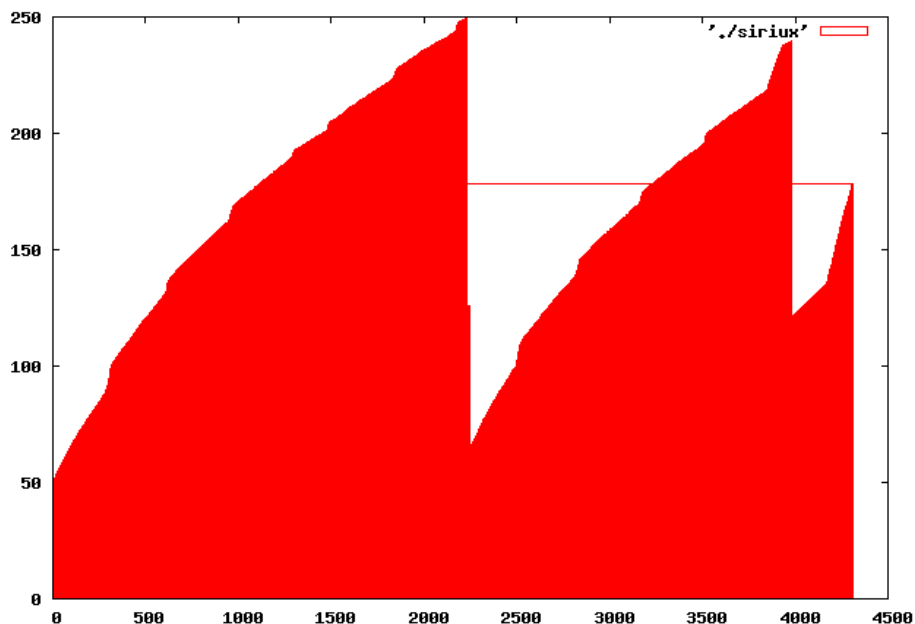
```
addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lw      $t1, 3
sub     $t2, $t1, 3
jal     sub_2DAD8
lw      $t7, dword_35A6C
lw      $t6, dword_35A70
subu    $t8, $t6, $t7
addiu   $t3, $t6, 8
sll     $t1, $v0, $t3
beqz    $t1, loc_2DA44
```

Nothing's for free

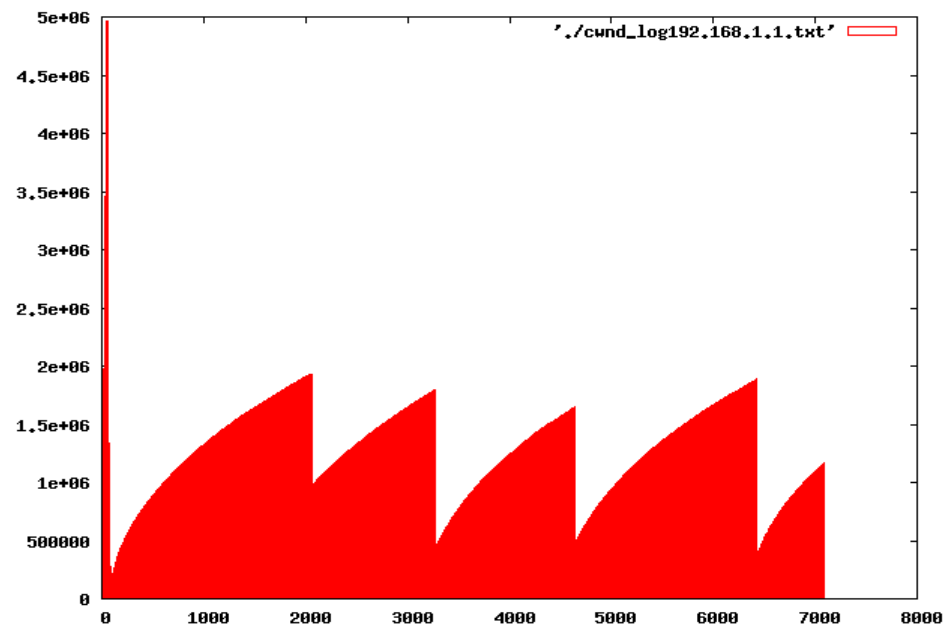
```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $1, 3
jal sub_2DAB8
lw $a0, dword_35A6C
lui $1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sltu $1, $a0, $t2
beqz $1, loc_2DA24
nop
sub $t2, $t2, 8
```


- 65535 ports, mostly closed, WRT.

NMAP CHND development



PortBunny CHND development

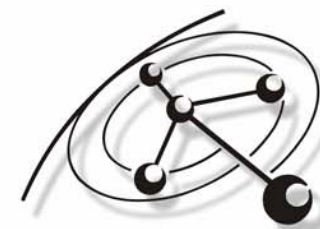



Nmap Free Security Scanner
 Network-wide ping sweep, portscan, OS Detection
 Audit your network security before the bad guys do

0:30.17 m

00:32.05 m

Invent & Verify

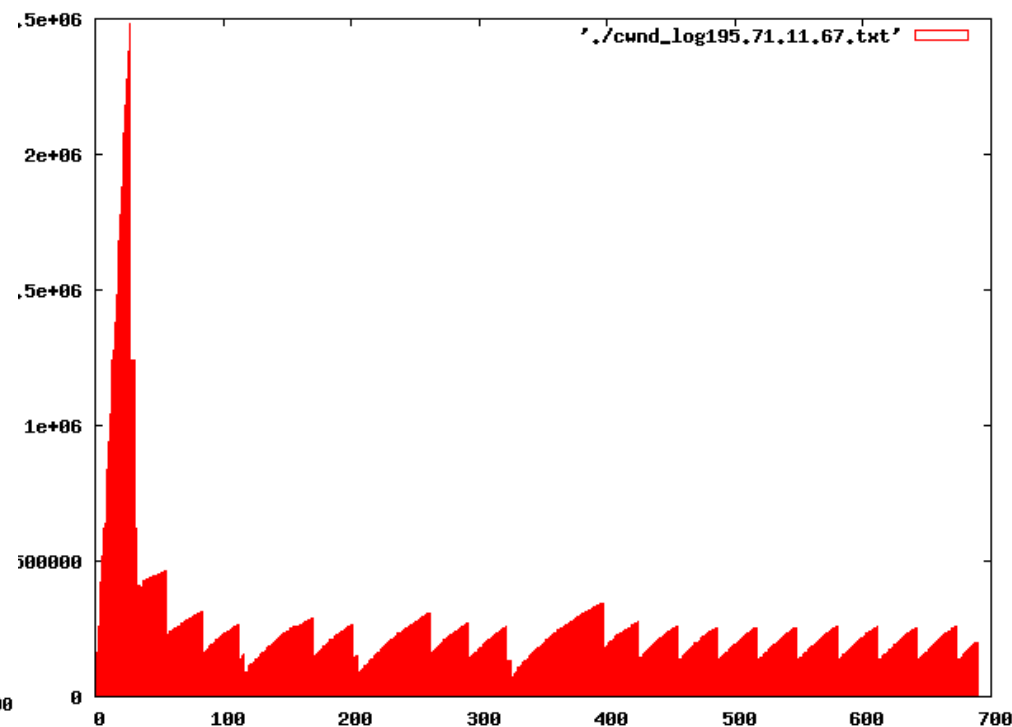
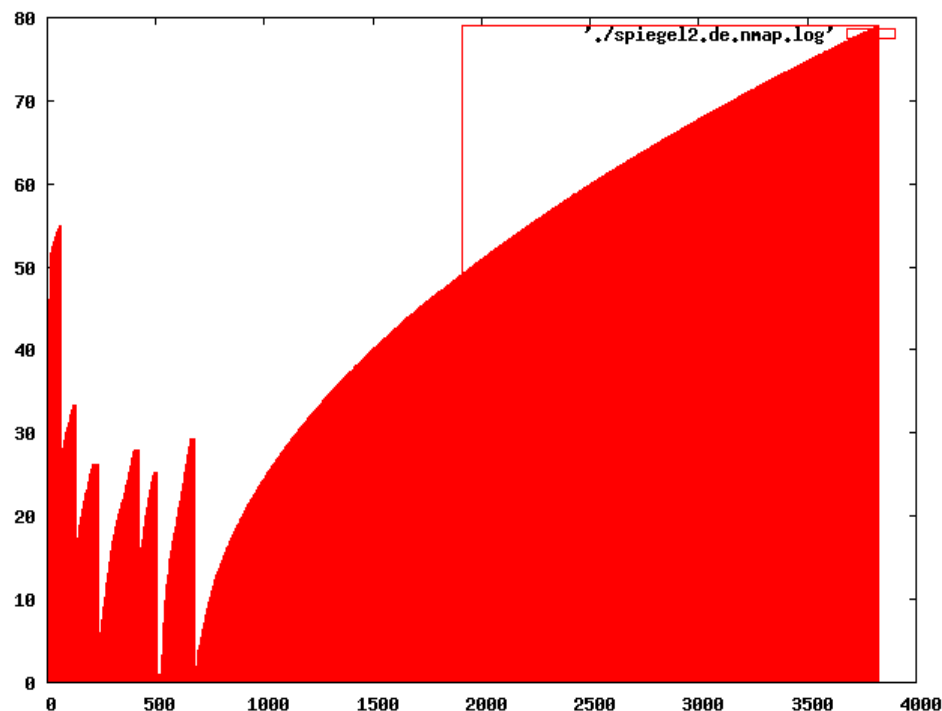


... but that doesn't count much.

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $0, 0x18+arg_0($sp)
lui $t1, 0
jal sub_20A68
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
```

NMAP CWND development

PortBunny CWND development



```
beqz $v0, loc_2DA44
move $v0, $0
la $t1, dword_35A70
```

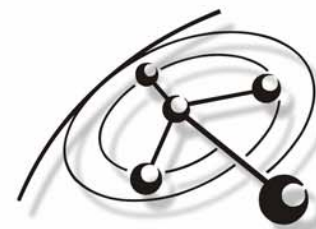


```
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```

00:41.14 m

0:28.04 m

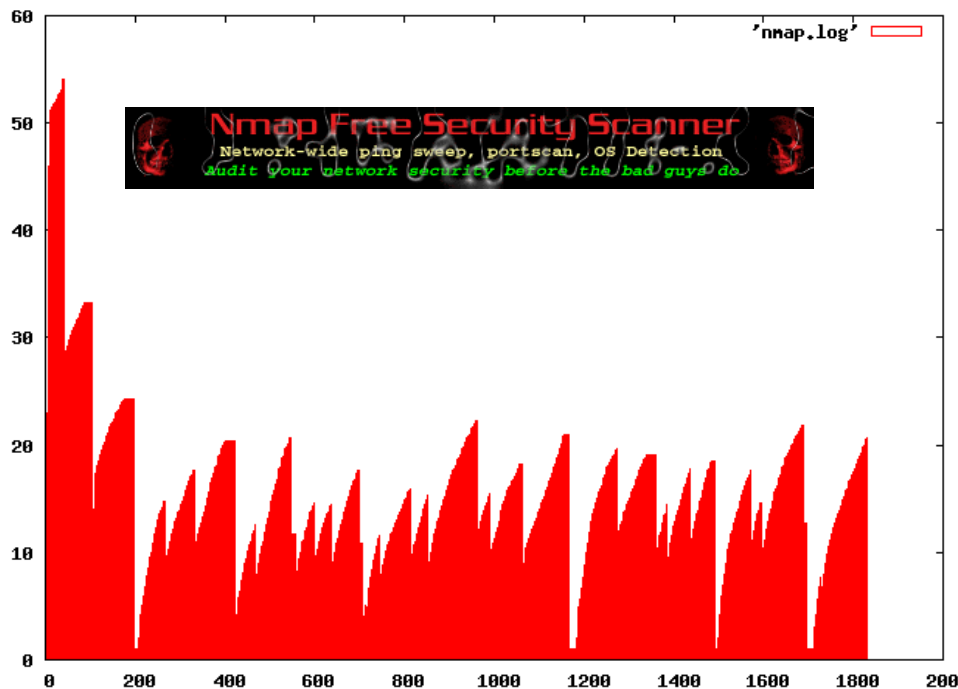
Invent & Verify



... Still PortBunny often wins this race

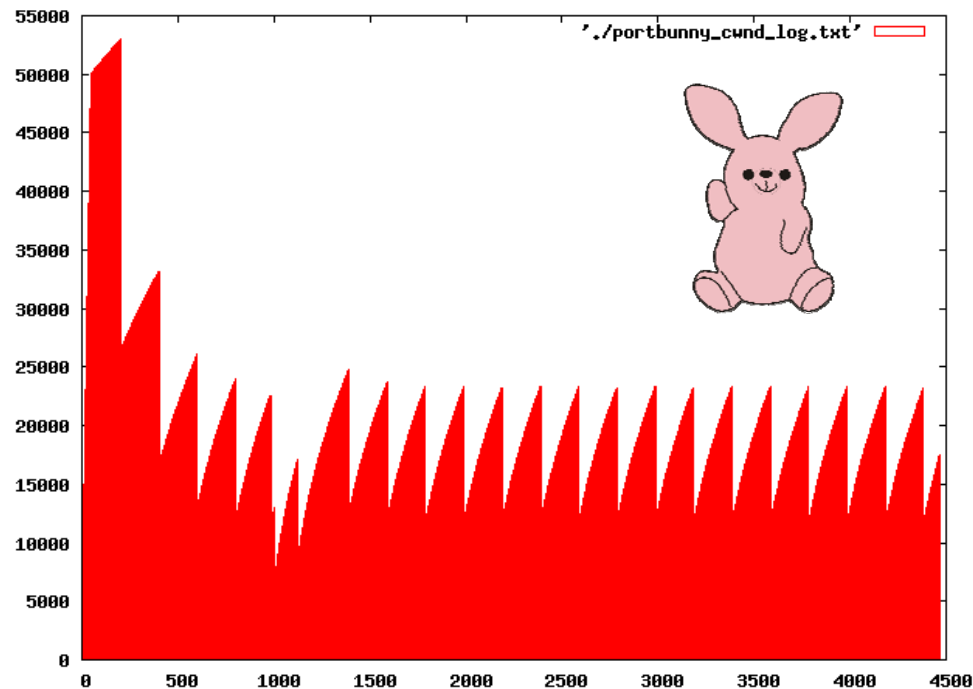
```
addiu $sp, -0x18
sw $r0, 0x18+var_4($sp)
sw $r1, 0x18+var_8($sp)
lui $t1, 0x00000000
jal $t1, sub_20A2B8
lw $a0, dword_35A6C
lui $t1, 3
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 0
sltu $t1, $a0, $t2
beqz $t1, loc_20A2A4
nop
sub $t1, $t1, 1
```

NMAP CHND development



0:30.20 m

NMAP CHND development



0:25.23 m

Invent & Verify



And then there are serious bugs

```
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 $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
mov $t2, $v0
sub $t2, $t2, 1
```

/ If packet drops are particularly bad, enforce a delay between packet sends (useful for cases such as UDP scan where responses are frequently rate limited by dest machines or firewalls) */*

Translates to: If packet-drops are particularly bad, break the entire timing-concept.

⇒ The CWND will not reflect the number of probes out at once anymore!

⇒ The self-clocking-property is being ignored!

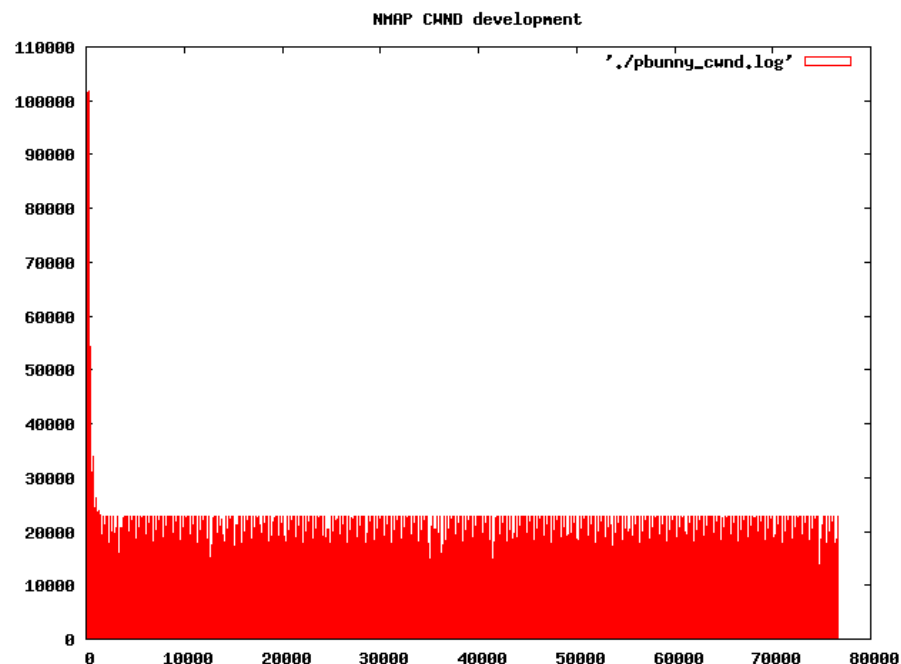
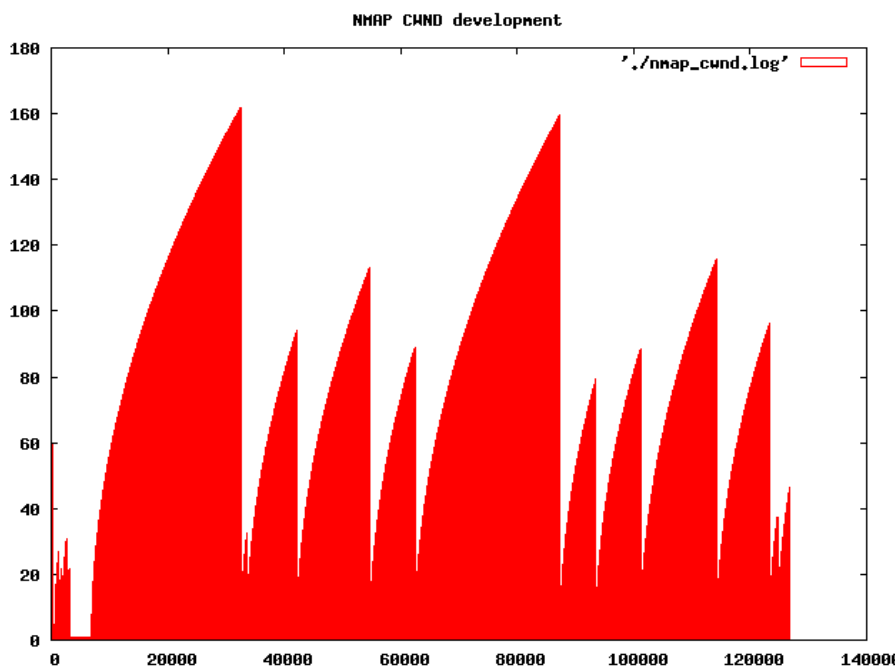
Invent & Verify



```
move $a0, $t7
lw $a0, sub_2DAB8
jal sub_2DAB8
addiu $a1, $v0, 0x10
beqz $a1, loc_2DA44
move $v0, 10
la $1, dword_35A70
lw $t1, dword_35A6C
lw $t0, dword_35A70
subu $t2, $t0, $t1
sra $t3, $t2, 2
sll $t4, $t3, 2
addu $t5, $v0, $t4
sw $t5, 0($t1)
sw $v0, dword_35A6C
```


Scanning the IPHONE

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lwi $t1, 3
jal sub_2DAB8
sw $a0, dword_35A6C
lwi $t1, 3
lwr $t7, dword_35A6C
lwr $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sltu $t1, $v0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1
```



24:41.51 m

Invent & Verify

7:58.03 m



Scanning in parallel

- PortBunny can scan a large number of hosts in parallel but by default, it will scan one host at a time. Why?
- Is a parallel scan always faster than a sequential scan?

```

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
    
```

```

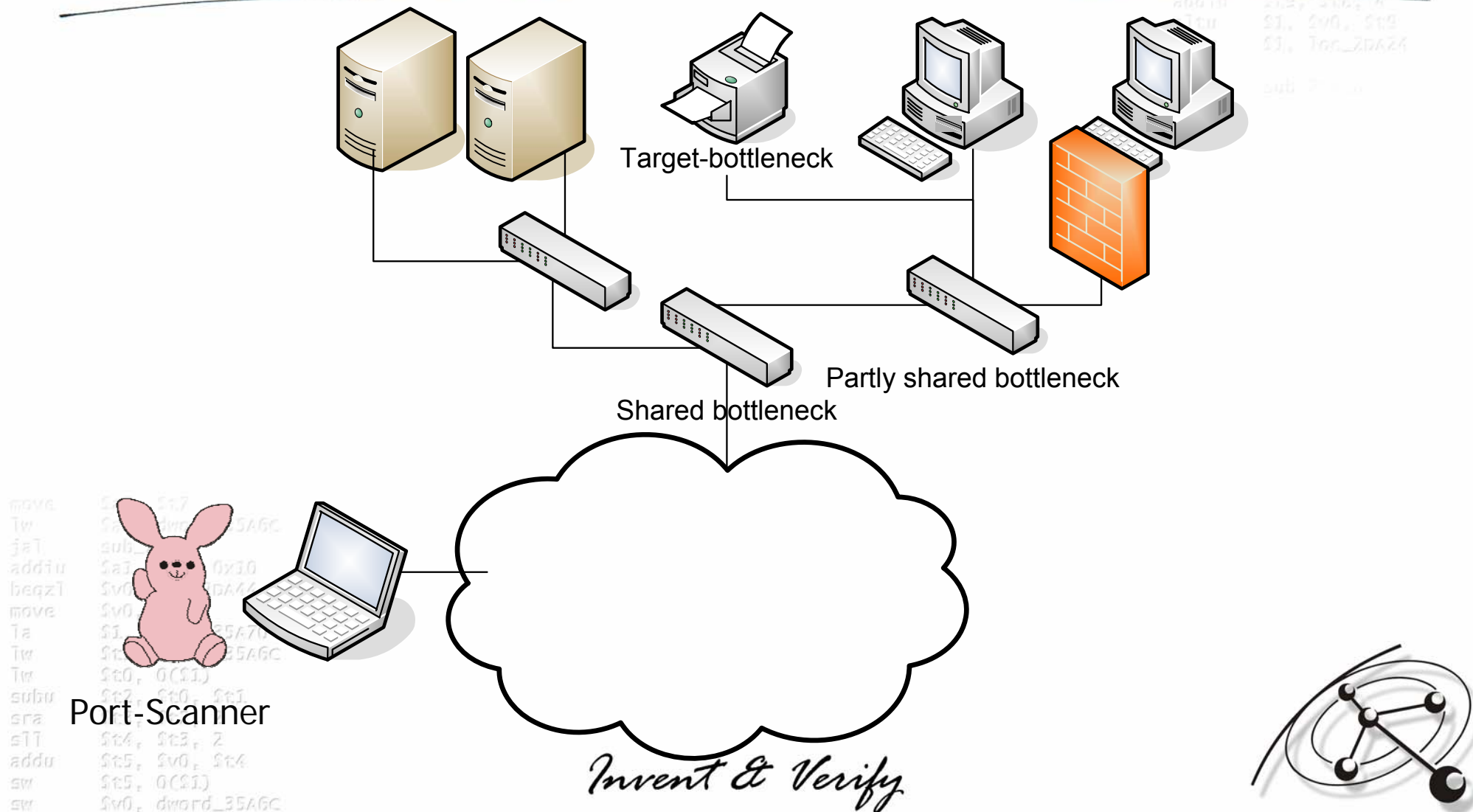
addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lw      $t1, 3
jal     sub_2DAB8
lw      $a0, dword_35A6C
lw      $t1, 3
lw      $t7, dword_35A6C
lw      $t6, dword_35A70
subu    $t8, $t6, $t7
addiu   $t2, $t6, 8
sllw    $t1, $v0, $t2
sllw    $t1, $t1, 2DA24
    
```

Invent & Verify

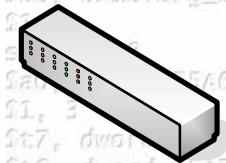


Bottlenecks

Will parallel scans win?



Shared bottleneck



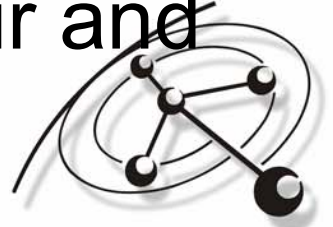
```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t0, 0
jal $t0, 0
lw $t0, 0x18+var_4($sp)
lui $t1, 2
lw $t7, dword_35A70C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sllw $t1, $t0, $t2
```

- If there's a bottleneck shared among all scan-jobs (common case), then there is no gain in scanning in parallel!

- ... assuming that the congestion-control-scheme actually works correctly (even for filtered hosts!)

- In fact, more unpredicted drops will occur and they will slow us down!

Invent & Verify



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

Target-bottlenecks



- If the target is the bottleneck, there is a gain in parallel scanning.
- It's possible to **do timing on a per-host basis** entirely: TCP-congestion-control-schemes were created with this scenario in mind!

- “Fairness” has been considered.

Invent & Verify



```

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, 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
    
```

What does NMAP do?

- Implement the same timing-algorithm for a global system which is informed of all answers and packet-drops to address **shared bottlenecks**.

- A scan-job may only send a new packet if the per-host-timing **AND** the global timing allows that.

Invent & Verify



Problem with this solution

- A badly performing host (target-bottleneck) will keep good performing host from firing.
- This timing is biased towards the performance of the worst scan-job.
- CWND is not “the number of packets out” anymore => again, the concept was broken.

```

move $a0, $t0
lw $a0, dword_35A6C
jal $ra
addiu $a0, $a0, 0x18
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, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($sp)
lui $t1, 3
jal $ra
lw $a0, dword_35A6C
lw $t7, dword_35A6C
lw $t6, dword_35A70
subu $t8, $t6, $t7
addiu $t2, $t6, 8
sllr $t1, $v0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1

```

Invent & Verify



Portbunny's solution

- Each scan-job performs its own timing based on a tcp-cc-scheme.
- This is similar to starting several independent http-downloads.
- You can only do that if the congestion-control-scheme actually works!
- By default: scan sequentially because single shared bottlenecks are the most common scenario.

Invent & Verify



Research in parallel scanning

- Old congestion control schemes must generate losses to find boundaries. Think wireless ;)
- Modern congestion control techniques are based on detecting changes in round-trip-time.
- Correlations between changes in round-trip-time can be used to detect shared bottlenecks!

```

move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB8
addiu $a0, $a0, 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, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($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, 0
sllw $t1, $v0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 0

```

Invent & Verify



... which is why Bunny
is in the kernel.

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
la $a0, 0x18+arg_0($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, 8
sltu $t1, $v0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1
```

- Timing is as precise as it can get.
- The “scanner-bottleneck”-issue for a large number of hosts is addressed not just algorithmically.

```
move $a0, $t7
lw $a0, dword_35A6C
jal sub_2DAB4
addiu $a0, 0x18
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
```

- We get a reliable sniffer for free.

Invent & Verify



Kernel-based sniffer

Port-bunny adds packet-handler by calling

dev_add_pack(struct packet_type *pt)

from net/core/dev.c

ARP_RCV

PPPOE_RCV

IP_RCV

PORT_BUNNY_RCV

Ethernet-
Frames

Invent & Verify



```

movl    $0, %eax
lwr     $0, dword_35A6C
jnl     sub_2DAD4
addiu   $a1, $v0, 0x10
beqz    $v0, loc_2DA44
move    $v0, $0
lwr     $t1, dword_35A70
lwr     $t1, dword_35A6C
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
    
```

```

addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lwr     $t1, 3
jnl     sub_2DAB8
lwr     $a0, dword_35A6C
lwr     $t1, 3
lwr     $t7, dword_35A6C
lwr     $t6, dword_35A70
subu    $t8, $t6, $t7
addiu   $t2, $t6, 8
sllw    $t1, $v0, $t2
lwr     $t1, 3
    
```

The user's perspective

- Chat with **/dev/portbunny** 😊
- The protocol is text-based and very simple.
- You can use portbunny with cat, echo and friends... but don't worry, we have a UI.

```
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
```

```
addiu   $sp, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($sp)
lw      $t1, 3
jal     sub_2DAB8
lw      $t0, dword_35A6C
lw      $t1, 3
lw      $t7, dword_35A6C
lw      $t6, dword_35A70
subu    $t8, $t6, $t7
addiu   $t2, $t6, 8
sllw    $t1, $v0, $t2
beqz    $t1, loc_2DA24
nop
```

Invent & Verify



Example input

- **# echo \$command > /dev/portbunny**
- **\$command:**
- *create_scanjob 192.168.1.1 FLOOD*
- *set_ports_to_scan 192.168.1.1 FLOOD 1-500*
- *execute_scanjob 192.168.1.1 FLOOD*

```

move $a0, $a0
lw $a1, $a0
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

```

```

addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($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, 8
sll $t1, $v0, $t2
beqz $t1, loc_2DA24
sub $t1, $t1, $t2

```

Invent & Verify



Example output

- **# cat /dev/portbunny**
- *SCAN_JOB_CREATED 192.168.1.1 FLOOD*
- *SCAN_JOB_EXECUTED 192.168.1.1 FLOOD*
- ...
- *RESULT 192.168.1.1 FLOOD PORT_STATE 79
CLOSED*
- *RESULT 192.168.1.1 FLOOD PORT_STATE 80
OPEN*
- ...
- *SCAN_JOB_REMOVED 192.168.1.1 FLOOD*

```
addiu $sp, -0x18
sw $ra, 0x18+var_4($sp)
sw $a0, 0x18+arg_0($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, 8
sltu $t1, $v0, $t2
beqz $t1, loc_2DA24
nop
sub $t1, $t1, 1
```

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

Invent & Verify



The PortBunny UI

- \$ portbunny host
- -p <port|port-range> ... ports to scan
- -d discover-mode
- -t <trigger> ... triggers to try
- -g generate data for gnuplot

▪ And that's all.

```

move    $a0, $t7
lw      $a0, dword_35A6C
jal     sub_2DAD4
addiu   $a1, $v0, 0x10
beqz    $v0, 1pc_35A44
move    $v0, $a1
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, -0x18
sw      $ra, 0x18+var_4($sp)
sw      $a0, 0x18+arg_0($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, 8
sllw    $t1, $v0, $t2
beqz    $t1, 1pc_2DA24
nop
sub     $t1, $t1, 1
    
```

Invent & Verify



Thank you!

```
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 $t2, $t6, 8
sltu $1, $v0, $t2
beqz $1, loc_2DA24
nop
sub $t2, $t2, 8
```



Recurity Labs

Fabian 'fabs' Yamaguchi
fabs@recurity-labs.com

Felix 'FX' Lindner
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 $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

