The Network Expect Framework for Building Network Tools

Eloy Paris
Cisco Systems

Forum of Incident Response Teams (FIRST)
19th Annual FIRST Conference
Sevilla, Spain, June 20th, 2007
Who Am I?

• Member of Cisco's Product Security Incident Response Team (PSIRT)
  – I handle security vulnerabilities in Cisco products
• I worked in Cisco's Technical Assistance Center (TAC)

➔ I have had the need to recreate situations of strange, unusual, or exceptional network traffic
What is Network Expect?

- A packet manipulation framework
- It allows to:
  - Craft and inject network packets
  - Decode received network packets
  - Take decisions based on received traffic

NetExpect is the result of my need to scratch the itch of recreating strange, unusual, and exceptional network traffic
A Brief Example

set target google.com

# Spawn a listener for ICMP messages from our target
spawn_network -i eth0 icmp and src host $target

for {set seq 0} {1} {incr seq} {
    send_network \ 
    ip(dst = $target)/ \ 
    icmp-echo(seq = $seq)/ \ 
    raw(12345678901234567890)

    expect_network -timeout 1 {$icmp(type) == 0 && $icmp(id) == [pid]} {
        puts [format "$pdu(2,tot_len) bytes from $ip(src): icmp_seq=$seq
        ttl=$ip(ttl) time=%.3f ms" [expr [txdelta ip]*1000 ] ]
        sleep [expr 1.0 - [txdelta ip] ]
    }
}
A Brief Example (Cont.)

shell# nexp script.nexp
28 bytes from 64.233.167.99: icmp_seq=0 ttl=245 time=29.010 ms
28 bytes from 72.14.207.99: icmp_seq=1 ttl=243 time=56.790 ms
28 bytes from 64.233.167.99: icmp_seq=2 ttl=245 time=24.312 ms
28 bytes from 64.233.187.99: icmp_seq=3 ttl=246 time=19.919 ms
28 bytes from 64.233.167.99: icmp_seq=4 ttl=245 time=57.324 ms
28 bytes from 72.14.207.99: icmp_seq=5 ttl=243 time=34.879 ms
28 bytes from 72.14.207.99: icmp_seq=6 ttl=243 time=34.339 ms
28 bytes from 64.233.167.99: icmp_seq=7 ttl=245 time=24.263 ms
28 bytes from 64.233.167.99: icmp_seq=8 ttl=245 time=24.427 ms
[...]
What Can We Learn From This Example?

1. Three key commands are the foundation of Network Expect:
   - spawn_network
   - send_network
   - expect_network

2. A common behavior of network-aware applications: action-reaction

3. There is a high-level language that glues everything together
spawn_network [options] [<PCAP filter>]

- Creates network “listeners” and “speakers”
  - A listener is used to read traffic from a source (PCAP file, network interface, UDP or TCP socket)
  - A speaker is used to send traffic to a destination (PCAP file, network interface, standard output, UDP or TCP socket, etc.)
- Think of spawn_network as the equivalent of the Unix socket() call
send_network [-o <speaker>] <packet def>

- Sends traffic to a destination using a “speaker”
  - Speaker is created with spawn_network
  - Speaker can be implicit ($speaker_id variable) or explicit (-o option)
- Great flexibility when defining packets
  - Ethernet, MPLS, 802.1Q, GRE, ARP, RARP, ICMP, ICMPv6, IP, IPv6, IPX, IGMP, BGP, OSPF.
  - Variable fields
- Very efficient
• `<packet def>` defines the packet:
  - `/` separates PDUs
  - PDUs are listed from lower to higher layers
  - PDUs defined with `pdu_name(pdu parms)`
  - Sensible defaults

• Examples:

  ip(dst=1.2.3.4,ttl=64)/tcp(flags=s,dst=80)

  ether(dst=de:ad:be:ef:00:00)/ \
  ip(dst=1.2.3.4)/icmp-echo()/raw('abcedef')
expect_network [-i <listener>] {<expr>} {<code block>}

- Reads packets from a source using a “listener”
  - Listener is created with `spawn_network`
  - Listener can be implicit (`$listener_id` variable) or explicit (`-i` option)
- After reading a packet a condition is evaluated
- If condition is true, a block of code is executed
When a packet is read, several high-level language variables are created – `$ether(src)`, `$ip(dst)`, `$tcp(srcport)`, `$icmp(type)`, etc.

These variables can be used in the expression of the `expect_network` command. For example:

```bash
expect_network { $icmp(type) == 0 } {<code executed when ICMP type is 0>}
```
Action-Reaction: The Expect Connection

- Network Expect was inspired on Don Libes' Expect, the Tcl-based toolkit for automating interactive programs

```
spawn telnet 192.168.1.1
expect "login:"    
send   "eloy\r"
expect "Password:"    
send   "myp4ssw0rd\r"
expect "$prompt"
send   "/usr/local/bin/script.sh\r"
expect "$prompt"
send   "exit\r"
```
The Expect Connection (cont.)

- **spawn ➔ spawn_network**
  - Spawn a network listener or speaker, not a process
- **send ➔ send_network**
  - Send to the network, not to a process
- **expect ➔ expect_network**
  - Expect something from the network, not from a process

➤ If you know Expect then you are well on your way to mastering Network Expect
Do You Mean I Need To Learn Yet Another Language?

- The answer is “it depends” - it depends on what you want to do:
  - For simple packet crafting you only need to know about `send_network`
  - For more complex tasks the answer is, unfortunately, “most likely yes”.

However...

- Tcl is easy (but perhaps not very powerful)
- Tcl is easier to learn than others
- Little Tcl knowledge is needed to accomplish useful things
Something Not in Expect

- **The `send_expect` command**
  - Inspired by Scapy's send-and-receive family of functions
  - Injects stimuli, collects responses, and matches stimuli with responses
  - Very powerful command; allows to build useful tools in a few lines of code
send_expect Example: An ARP Scanner

```
set interface eth0
set network "$iface($interface,ip)/$iface($interface,netmask)"

# Spawn a listener for ARP replies
spawn_network -i $interface {arp[6:2]} == 2

send_expect -o $interface -delay 0.001 -tries 2 \
    ether(dst = BROADCAST)/ \
    arp-request(tha = BROADCAST, \
        tip = "$network", \
        sha = $iface($interface,hw_addr), \n        sip = $iface($interface,ip) )

puts "\nFound [llength $_(received)] hosts alive:\n"

foreach r $_(received) {
    packet decode r
    puts "$arp(sip) is at $arp(sha)"
}
```
Another `send_expect` Example: A TCP Traceroute

```plaintext
set target google.com
set port 80
set interface [outif $target]

spawn_network -i $interface

send_expect -tries 2 -delay 0.010 \ 
  ip(id = random, dst = $target, ttl = 1:30)/ \ 
  tcp(src = random, dst = $port, flags = s)

foreach r $_(received) s $_(sent) {
  packet decode r
  set source $ip(src)
  set pdu_type $pdu(1,type)

  packet decode s

  puts [format "$ip(ttl) $source %.3f ms $pdu_type" [expr [packet tdelta r s]*1000] ]
}
```
The 0trace Proof-of-Concept

• Traceroute that rides an existing TCP session
• Published by security researcher Michal Zalewski in January 2007
• 114 lines of complicated shell scripting that calls `tcpdump`, `cat`, `head`, `tail`, `sed`, `cut`, `grep`, `awk`, the works + 172 lines of C code + out-of-band TCP connection
• 40 lines of NetExpect code
Re-Writing PCAP Files

- Basic skeleton code:

```plaintext
set infile in.pcap
set outfile out.pcap
set filter "tcp and host 192.168.1.1"

spawn_network -r $infile -w $outfile $filter

expect_network {1} {
    send_network raw('$_(packet)')
    nexp_continue
} eof

close_network nexp0
```
Under The Hood

• NetExpect is written in C and some Yacc and Flex
  – “Should” behave well under heavy load
  – Will meltdown your network when injecting packets
• It has been built (and run) on Linux, FreeBSD, OpenBSD, MacOS X, Solaris
• Uses libpcap for reading packets and libdnet for sending packets ➔ great portability
Final Thoughts

- Network Expect will be released with an Open Source license
  - Project's home is www.netexpect.org
- Currently a one-man show
  - Looking for help (code and documentation)
Final Thoughts (Cont.)

- Documentation in a very sorry state
  - Have a mediocre Unix manual page
  - Using examples as documentation
  - Will hopefully improve when I shift my focus from development to documentation
- Contact: eloy@cisco.com
Does The World Need Yet Another Packet Generator?

- Lots and lots of tools out there, but...
  - Didn't know about them when I started
    - Nemesis, Packit, SendIP, Scapy, CASL, hping, etc.
  - Not all of them have the flexibility I need
  - Not all of them do everything I need
  - Some can have steep learning curves
  - I need more than just a packet crafter
  - Competition is good; let the user decide!
Extra Slides
C Versus Tcl Performance

- SYN flood attack in C
  ```
  send_network -count 1000000 ip(src = random, dst=1.2.3.4)/tcp(dst = 80, flags = s)
  ```
- SYN flood attack in Tcl
  ```
  for {set i 0} {$i < 1000000} {
    send_network ip(src = random, dst = 1.2.3.4)/tcp(dst = 80, flags = s)
  }
  ```
- 430 kpps in C versus 10 kpps in Tcl