Introducing Unicornscan

RIDING THE UNICORN

Unicornscan is an open source (GPL) tool designed to assist with information gathering and security auditing. This talk will contrast the real world problems we've experienced using other tools and methods while demonstrating the solutions that Unicornscan can provide.
For the latest copy of this talk go to http://www.dyadlabs.com/defcon05.pdf

Presented By:

DYAD LABS

Outline

1. Introduction to Unicornscan
   - Scatter Connect
   - Unicornscan - History, Background, & Technical Details
   - Demo

Speakers

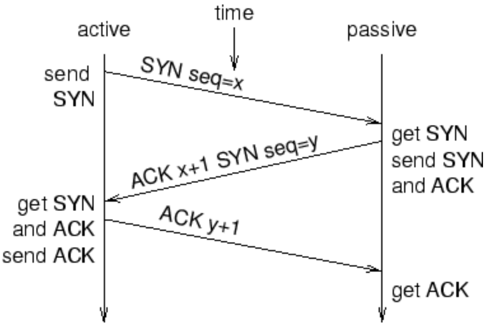
Speakers:

- Robert E. Lee
    - CEO, Dyad Labs, Inc.
    - Director of Projects and Resources, ISECOM
    - OPST & OPSA Certified Trainer

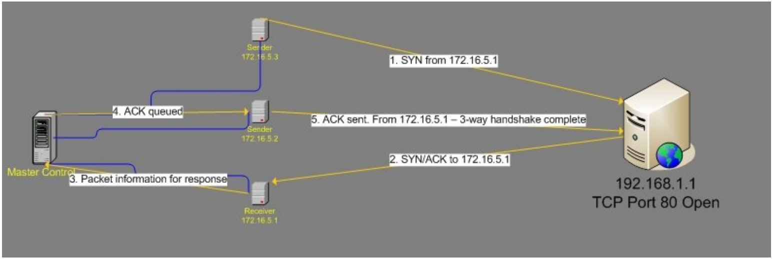
- Jack C. Louis
    - Chief Security Researcher, Dyad Labs, Inc.
    - Systems Programmer
    - OPST & OPSA Certified Trainer

- Anthony de Almeida Lopes
    - Intern, Dyad Labs, Inc.

3-way Handshake Connect Picture



Scatter Connect Picture



Scatter Connect Explained

Move TCP connection state tracking out of kernel space and into user space

- One process is the master control (Unicornscan).
    - Keeps track of what packets need to be sent
    - Who can send them
    - The responses that have come back
    - connection state.
- A second process is the sender (unisend).
    - Assembles the packets and puts them on the wire.
    - Optionally, you can split this function into Batch Sender and Immediate Sender modes.
- A third process is the listener (unilisten).
    - Listens for responses and sends the meta information back to the master control.

Scatter Connect Explained - Cont

- When unilisten sees the SYN/ACK packet, it sends the meta information back to Unicornscan.
- Unicornscan then requests that unisend send a ACK packet back to the host that sent the SYN/ACK to complete the 3-way hand-shake.
- At that point, depending on what other modules or payloads were to be used in the session, Unicornscan would schedule the additional payloads to be sent by unisend.

Common Port Scanning Problems

- No reliable UDP scanning
  - UDP port scanning involves sending UDP probes with no application/protocol specific instructions in the datagram and counting on ICMP responses to indicate "Closed" ports
  - Turns out, people use firewalls
  - Required us to script UDP protocol clients (dig, snmpwalk, etc) to enumerate live services through firewalls

Common Port Scanning Problems - Cont

- TCP enumeration has too many steps
  - Syn Scanning
  - Connect/banner grab scanning the "open" ports
  - Protocol specific stimulus (ala amap, manual testing, etc)
- Networks are getting bigger
  - Tools not intended for testing large (65k-4billion+ IPs) networks
  - Processing the output can be overwhelming

Common Vulnerability Scanning Problems

- These tools have modules that contradict each other
    - Remote host is running Amiga Miami OS
    - Remote host is running IIS 5.0 on TCP Port 80
    - Remote host is vulnerable to Apache Nosejob vulnerability
- To be useful, the modules need to share information better

Timing Problems

- No mechanism for accurate control over the Packets Per Second (PPS) rate
    - Scanrand provided a -b (bandwidth) option, but is bursty and has lots of packet loss at relatively low speeds
    - Nmap's -T option has a few problems
        - Normal, Aggressive, Insane, etc isn't easily translated into PPS ranges
        - Even Insane doesn't scale to high speeds
        - The timing is affected by the responses received

Mindset

Fundamental Design Goals

- Scalability
    - Efficient use of resources (bus analogy)
    - Not the same as just "going faster"
- Accuracy
    - Invalid data collection leads to invalid analysis
    - Tools should introduce stimulus, record response
    - Humans analyze

Mindset

- Flexibility
    - You supply the stimulus
    - We supply the delivery mechanism
    - Dynamic "just in time" decision control
- Security
    - Implementation modeled after IDSSCPP_V1.2
    - We also provide a sample security policy for use with SE Linux

What does it do?

- Provides a distributed user space TCP/IP stack
- Some key features:
  - Asynchronous stateless TCP/UDP port Scanning
  - Asynchronous stateless TCP banner grabbing
  - Active remote OS, application, and component collection

What does it do? - Cont

- Some key features - Cont:
    - Custom static and dynamic payload module support
        - Protocol specific TCP/UDP stimulus & response framework
        - Metamorphic shellcode encoder
    - PCAP file logging and filtering
    - Relational database output

Architecture

- 3 processes (NOT THREADS)
  - Why? - No safe way to do SoD with threads (due to the sharing of memory).
  - There is no good reason to share memory
- Sender (It sends packets)
  - Batch Sender
  - Immediate Sender
- Listener (It Sniffs)
- Master (It does everything else)

Sequence Number Encoding

Stateless TCP scanning that can support TCP streams:
#define TCPHASHTRACK(srcip, srcport, dstport, syncookie)
((syncookie) ˆ ((srcip) ˆ ((srcport) $<<$ 16) + (dstport))))

- With something so simple we can do TCP connections because the algorithm is NOT one way.
- Without this there would be an IPC message for every packet received.
- We don't have an entry in our state table until after we receive a response (SYN+ACK)
- Normal TCP/IP stacks would use a state table entry after sending the initial SYN

Internal Steps for a Syn Scan

How a Scan Works: (example, cluster mode is different, obviously)
Simple 30 step process to running a port scan. (for a basic SYN scan running on one host)

1. User invokes scanner via command line.
2. Scanner reads configuration file.
3. Scanner walks command line options (overriding configuration file).
4. Master process opens shared library modules.
5. Master Process figures out that it needs to fork children.

Internal Steps for a Syn Scan

6. Master Process forks a sender process and records its creation information.
7. Sender Process opens Unix domain socket and awaits master connection.
8. Master thread awaits signals from both children.
9. Master connects to sender and sends an IDENT message.
10. Sender and Listener send back IDENT_SENDER and IDENT_LISTENER message respectively.

Internal Steps for a Syn Scan

11. Master Process forks listener process and records its creation information.
12. Listener Process opens Unix domain socket and awaits master connection.
13. Master ACKS both messages.
14. Sender opens up configuration file and shared libraries looking for UDP payloads.
15. Listener opens pcap interface, looks up interface information, and attempts to drop privs.

Internal Steps for a Syn Scan

16. Sender sends back READY message to Master.
17. Listener sends back READY message with listeners interface information included.
18. Master considers information from Listener and prepares workunit information.
19. Master sends Listener a "sniffing" workunit, including a pcap filter string, etc.
20. Listener gets "sniffing" workunit and responds with a READY message after processing it.

Internal Steps for a Syn Scan

21. Listener goes into loop polling pcap and IPC file descriptor.
22. Master sends Sender a "batch" workunit containing a ‘large’ unit of work to perform.
23. Sender creates a linked list of function pointers to loop on and goes into a batch loop.
24. Master goes into loop reading Sender and Listener file descriptors (mostly from the listener), recording information that it was asked to record.
25. Sender finishes work loop, and sends a WORKDONE message back to the Master.

Internal Steps for a Syn Scan

26. Master checks if it has more work, but does not so goes into a scan timeout state.
27. Master goes into reporting state after timeout finishes, perhaps recording information and displaying it.
28. Master tells Sender to QUIT.
29. Master tells Listener to QUIT.
30. Master awaits the 2 SIGCHLD signals, and itself exits.

TCP Stream Stuff

- Normal stacks are designed for situations where you know all of the Source_IP,Source_port:Dest_IP,Dest_Port combinations that you will want to use.
- Unicornscan's stack was designed for situations where you know the Source_IP,Source_port side, but have a large list of Dest_IP,Dest_Port possibilities.

TCP Stream Stuff

- This is because we don't know who we are going to connect to, or how many things we are going to connect to.
- We don't have a fixed size table because we have _no_ idea how big its going to get
- We should assume that O(log2N)'ish is better than O(1) if we don't have bounds (worst case)

State Table Math

Why?

- We can only have O(1) if we know the size of our data set, we do not. If we have fixed sized tables, we need to guess high, even though we generally wont fill the table (we don't know our endpoints, once again)

Some Numbers for entire Internet on one port:

- Table: 2 ˆ 32 * sizeof(key) = 4294967296 * 8 = 34,359,738,368 bytes
- Or roughly 32GB
- (that's just for the index keys)

State Table Math

If that seems silly to you, understand that devices that do 3way handshakes for every address and port (cellphone networks, DDoS Protection, etc) are becoming more common. While we will do a 3 way handshake, we will not actually transfer data on every connection, we need to find a middle ground.

- We need to be able to withstand networks that have fake 3way handshakes and ones that do not.
- We need an adaptable method that only uses memory when it needs to, without throwing away performance.

TCP State Tracking

We use a binary tree (red-black balancing) as a good compromise, however this is easy to change inside the code, and in the future possibly heuristic detection can switch from a O(1) table to a binary tree on the fly when it detects trouble.

TCP State Tracking

Some numbers from an overloaded state table (average out of 10, input was mostly random)

- objects from (rbtree.o chtbl.o) code were linked into test harness for measurement
- (rdtsc was inlined before and after calls to measured functions)

TCP State Tracking

CPU:	Intel Celeron(R) M at 1.4 Ghz with 1M of L2 cache and 32K of I/D L1 cache.
Compiler:	GNU gcc 3.3.4 (CFLAGS -O2) / Binutils 2.15.92.0.2
OS:	GNU/Linux 2.6.12

```
804a158:  0f 31           rdtsc
...
804a176:  e8 65 fa ff ff  call    8049be0 <chtfind>
...
804a158:  0f 31           rdtsc
```

TCP State Tracking

- (65536 Items, in 73189 Slots CHT load factor of 1.12)
  - RB Tree insert: 2426
  - RB Tree search: 2068
  - CHT insert : 571
  - CHT search : 466
- (65536 Items, in 4673 Slots CHT load factor of 14.02)
  - RB Tree insert: 2528
  - RB Tree search: 1950
  - CHT insert : 4267
  - CHT search : 3924
- (65536 Items, in 313 Slots CHT load factor of 209.38)
  - RB Tree insert: 2577
  - Tree search: 1875
  - CHT insert :55294
  - CHT search :50769

TCP State Tracking - Future

- Future improvements will likely include a hybrid MFU (not MRU) table cache in front of the rbtree.
  - Why?
  - we need to avoid caching RST+ACK's (most frequent situation is a port to be closed, and we still need to report this condition)
- IPC is currently via Unix domain sockets or TCP sockets (more transports can be added)

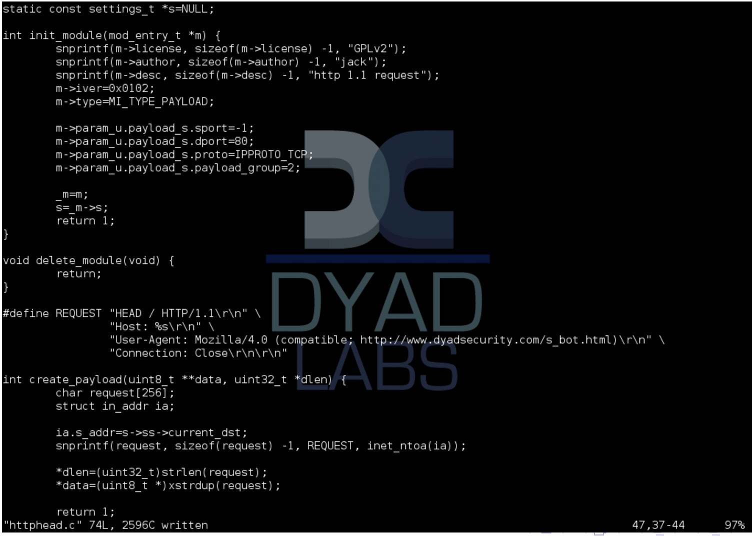
payloads.conf

```
/* pc anywhere */
udp 5632 -1 1 {
                    "NQ"
};

udp 5632 -1 1 {
                    "ST"
};
```

payloads.conf - Cont

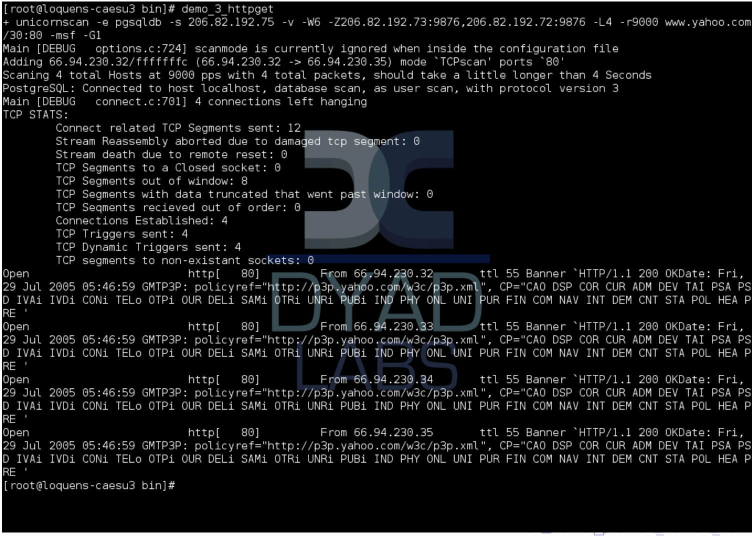
```
/* Open Relay */
tcp 25 -1 1 {
        "HELO FOO\r\n"
        "Mail From: mailfrom@notyourdomain.com\r\n"
        "Rcpt To: rcptto@notyourdomain.com\r\n"
        "Data\r\n"
        "Subject: Testing for Open Relay\r\n"
        "\r\n"
        "This is only a test\r\n"
        ".\r\n"
        "\r\n"
        "\r\n"
};
```

HTTP HEAD Request Source



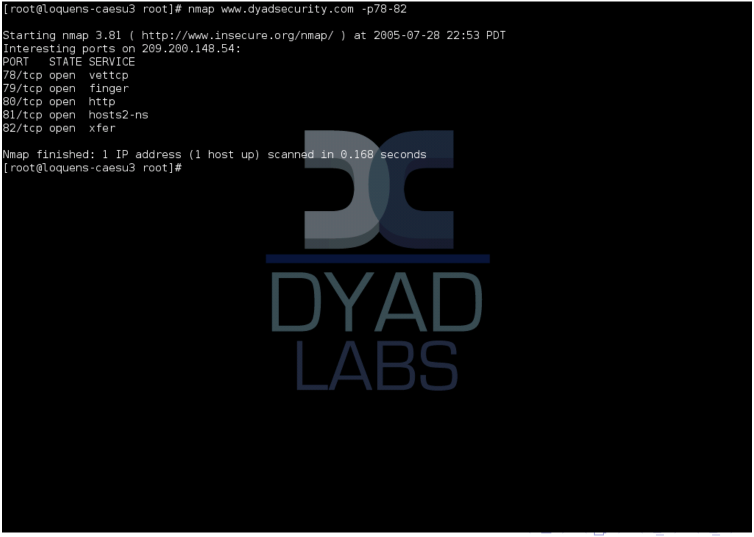
```
static const settings_t *s=NULL;

int init_module(mod_entry_t *m) {
        snprintf(m->license, sizeof(m->license) -1, "GPLv2");
        snprintf(m->author, sizeof(m->author) -1, "jack");
        snprintf(m->desc, sizeof(m->desc) -1, "http 1.1 request");
        m->iver=0x0102;
        m->type=MI_TYPE_PAYLOAD;

        m->param_u.payload_s.sport=-1;
        m->param_u.payload_s.dport=80;
        m->param_u.payload_s.proto=IPPROTO_TCP;
        m->param_u.payload_s.payload_group=2;

        _m=m;
        s=_m->s;
        return 1;
}

void delete_module(void) {
        return;
}

#define REQUEST "HEAD / HTTP/1.1\r\n" \
                "Host: %s\r\n" \
                "User-Agent: Mozilla/4.0 (compatible; http://www.dyadsecurity.com/s_bot.html)\r\n" \
                "Connection: Close\r\n\r\n"

int create_payload(uint8_t **data, uint32_t *dlen) {
        char request[256];
        struct in_addr ia;

        ia.s_addr=s->ss->current_dst;
        snprintf(request, sizeof(request) -1, REQUEST, inet_ntoa(ia));

        *dlen=(uint32_t)strlen(request);
        *data=(uint8_t *)xstrdup(request);

        return 1;
"httphead.c" 74L, 2596C written                                          47,37-44      97%
```

HTTP GET Request In Action



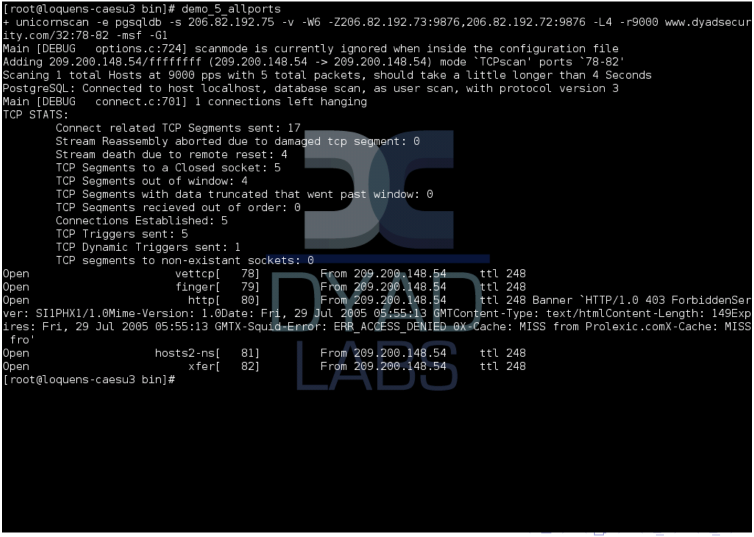
```
[root@loquens-caesu3 bin]# demo_3_httpget
+ unicornscan -e pgsqldb -s 206.82.192.75 -v -W6 -Z206.82.192.73:9876,206.82.192.72:9876 -L4 -r9000 www.yahoo.com
/30:80 -msf -G1
Main [DEBUG   options.c:724] scanmode is currently ignored when inside the configuration file
Adding 66.94.230.32/fffffffc (66.94.230.32 -> 66.94.230.35) mode `TCPscan' ports `80'
Scaning 4 total Hosts at 9000 pps with 4 total packets, should take a little longer than 4 Seconds
PostgreSQL: Connected to host localhost, database scan, as user scan, with protocol version 3
Main [DEBUG   connect.c:701] 4 connections left hanging
TCP STATS:
        Connect related TCP Segments sent: 12
        Stream Reassembly aborted due to damaged tcp segment: 0
        Stream death due to remote reset: 0
        TCP Segments to a Closed socket: 0
        TCP Segments out of window: 8
        TCP Segments with data truncated that went past window: 0
        TCP Seqments recieved out of order: 0
        Connections Established: 4
        TCP Triggers sent: 4
        TCP Dynamic Triggers sent: 4
        TCP segments to non-existant sockets: 0
Open                        http[   80]         From 66.94.230.32       ttl 55 Banner `HTTP/1.1 200 OKDate: Fri,
29 Jul 2005 05:46:59 GMTP3P: policyref="http://p3p.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PS
D IVAi IVDi CONi TELo OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA P
RE '
Open                        http[   80]         From 66.94.230.33       ttl 55 Banner `HTTP/1.1 200 OKDate: Fri,
29 Jul 2005 05:46:59 GMTP3P: policyref="http://p3p.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PS
D IVAi IVDi CONi TELo OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA P
RE '
Open                        http[   80]         From 66.94.230.34       ttl 55 Banner `HTTP/1.1 200 OKDate: Fri,
29 Jul 2005 05:46:59 GMTP3P: policyref="http://p3p.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PS
D IVAi IVDi CONi TELo OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA P
RE '
Open                        http[   80]         From 66.94.230.35       ttl 55 Banner `HTTP/1.1 200 OKDate: Fri,
29 Jul 2005 05:46:59 GMTP3P: policyref="http://p3p.yahoo.com/w3c/p3p.xml", CP="CAO DSP COR CUR ADM DEV TAI PSA PS
D IVAi IVDi CONi TELo OTPi OUR DELi SAMi OTRi UNRi PUBi IND PHY ONL UNI PUR FIN COM NAV INT DEM CNT STA POL HEA P
RE '
[root@loquens-caesu3 bin]#
```

All Ports Open - Nmap



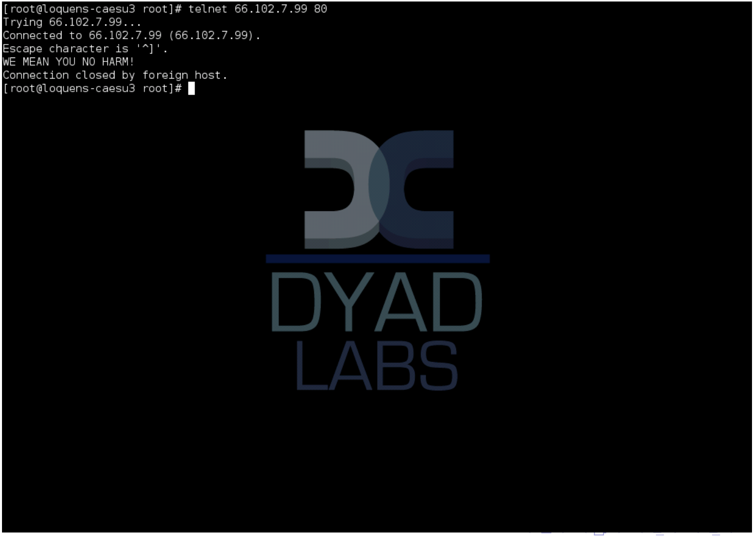
```
[root@loquens-caesu3 root]# nmap www.dyadsecurity.com -p78-82

Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-07-28 22:53 PDT
Interesting ports on 209.200.148.54:
PORT   STATE SERVICE
78/tcp open  vettcp
79/tcp open  finger
80/tcp open  http
81/tcp open  hosts2-ns
82/tcp open  xfer

Nmap finished: 1 IP address (1 host up) scanned in 0.168 seconds
[root@loquens-caesu3 root]#
```

All Ports Open - Unicornscan



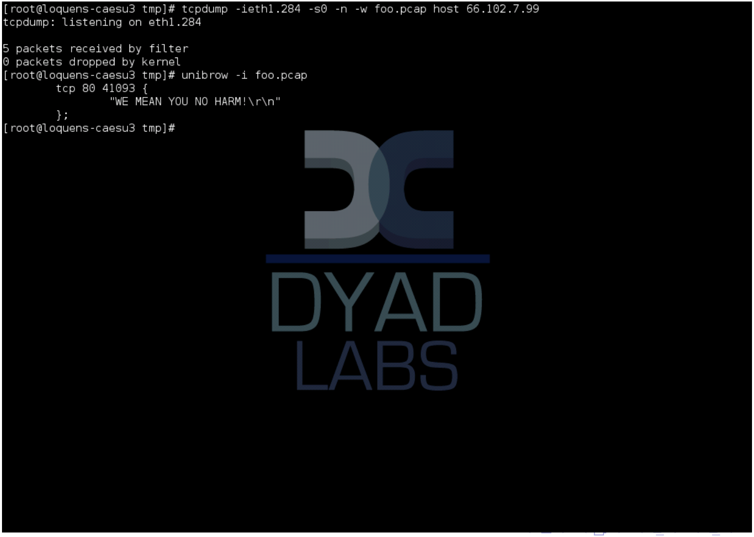
```
[root@loquens-caesu3 bin]# demo_5_allports
+ unicornscan -e pgsqldb -s 206.82.192.75 -v -W6 -Z206.82.192.73:9876,206.82.192.72:9876 -L4 -r9000 www.dyadsecur
ity.com/32:78-82 -msf -G1
Main [DEBUG   options.c:724] scanmode is currently ignored when inside the configuration file
Adding 209.200.148.54/ffffffff (209.200.148.54 -> 209.200.148.54) mode `TCPscan' ports `78-82'
Scaning 1 total Hosts at 9000 pps with 5 total packets, should take a little longer than 4 Seconds
PostgreSQL: Connected to host localhost, database scan, as user scan, with protocol version 3
Main [DEBUG   connect.c:701] 1 connections left hanging
TCP STATS:
        Connect related TCP Segments sent: 17
        Stream Reassembly aborted due to damaged tcp segment: 0
        Stream death due to remote reset: 4
        TCP Segments to a Closed socket: 5
        TCP Segments out of window: 4
        TCP Segments with data truncated that went past window: 0
        TCP Seqments recieved out of order: 0
        Connections Established: 5
        TCP Triggers sent: 5
        TCP Dynamic Triggers sent: 1
        TCP segments to non-existant sockets: 0
Open                      vettcp[   78]         From 209.200.148.54     ttl 248
Open                      finger[   79]         From 209.200.148.54     ttl 248
Open                        http[   80]         From 209.200.148.54     ttl 248 Banner `HTTP/1.0 403 ForbiddenSer
ver: SI1PHX1/1.0Mime-Version: 1.0Date: Fri, 29 Jul 2005 05:55:13 GMTContent-Type: text/htmlContent-Length: 149Exp
ires: Fri, 29 Jul 2005 05:55:13 GMTX-Squid-Error: ERR_ACCESS_DENIED 0X-Cache: MISS from Prolexic.comX-Cache: MISS
 fro'
Open                   hosts2-ns[   81]         From 209.200.148.54     ttl 248
Open                        xfer[   82]         From 209.200.148.54     ttl 248
[root@loquens-caesu3 bin]#
```

Static Payload Example



```
[root@loquens-caesu3 root]# telnet 66.102.7.99 80
Trying 66.102.7.99...
Connected to 66.102.7.99 (66.102.7.99).
Escape character is '^]'.
WE MEAN YOU NO HARM!
Connection closed by foreign host.
[root@loquens-caesu3 root]# 
```

Static Payload Example - Cont



```
[root@loquens-caesu3 tmp]# tcpdump -ieth1.284 -s0 -n -w foo.pcap host 66.102.7.99
tcpdump: listening on eth1.284

5 packets received by filter
0 packets dropped by kernel
[root@loquens-caesu3 tmp]# unibrow -i foo.pcap
        tcp 80 41093 {
                "WE MEAN YOU NO HARM!\r\n"
        };
[root@loquens-caesu3 tmp]#
```

Fireworks

1. TCP Connect Scan
2. If port 80 syn/ack's, perform OS detection
3. 3-way handshake completes
4. Unicornscan generates a metamorphic 1st stage encoder before encoding the OS specific stage 1 exploit payload
5. Payload sent

Fireworks - Cont

6. If successful
    - Truncate Apache error log
    - Create a socket
    - Connect back to a pre-arranged address
    - Tell the pre-arranged address what platform it is
    - Ask the pre-arranged address for OS specific stage 2
    - mmap a memory area that is writable and executable
    - read from the 2nd stage server while writing
    - then transfer control to the 2nd stage code
    - Will not write to the disk

Recapitulation

- What did I just see?
  - A new method for TCP state tracking (Scatter Connect)
  - Unicornscan is a distributed Stimulus/Response framework (not a port scanner)

  - Or to quote Winhat:
  - If x=4 and y=8, then I just wasted your time because I rule!

The End

Thank you for your time.

- For more information, see:
    - http://www.unicornscan.org
- Or just write us:
    - Jack at dyadsecurity dot com
    - Robert at dyadsecurity dot com