Network Security II: Network perimeters, DNS, DDoS
Network Perimeter Defense

- Idea: network defenses on “outside” of organization (e.g., between org and Internet)
  - Pros/Cons?

- Typical elements
  - Firewalls
  - Network Address Translation
  - Network Intrusion Detection/Prevention Systems (NIDS/NIPS)
    (and other kinds of network content analysis)
Firewalls

- Problem: Protecting or isolating one part of the network from other parts
  - In particular: protect your network from global Internet

- Need to filter or otherwise limit network traffic
  - How to configure this information?

- Questions:
  - What information do you use to filter?
  - Where do you do the filtering?
Kinds of Firewalls

▪ Personal firewalls
  – Run at the end hosts
  – e.g. Norton, Windows, etc.
  – Benefit: has more application/user specific information

▪ Network firewalls
  – Intercept and evaluate communications from many hosts

▪ Filter Based
  – Operates by filtering based on packet headers

▪ Proxy based
  – Operates at the level of the application
  – e.g. HTTP web proxy
Network Firewalls

- Filters protect against “bad” communications.
- Protect services offered internally from outside access.
- Provide outside services to hosts located inside.
Packet Filtering Firewalls

- Packet filtering firewalls can take advantage of the following information from network and transport layer headers:
  - Source IP
  - Destination IP
  - Source Port
  - Destination Port
  - Flags (e.g. ACK)

- Some firewalls keep state about open TCP connections
  - Allows conditional filtering rules of the form “if internal machine has established the TCP connection, permit inbound reply packets”
Ports

- Ports are used to distinguish applications and services on a machine.
- Low numbered ports are often reserved for server listening.
- High numbered ports are often assigned for client requests.

- Port 7 (UDP,TCP): echo server
- Port 13 (UDP,TCP): daytime
- Port 20 (TCP): FTP data
- Port 21 (TCP): FTP control
- Port 22 (TCP): ssh
- Port 25 (TCP): SMTP (Mail)
- Port 80 (TCP): HTTP
- Port 123 (UDP): NTP
- Port 143 (TCP): IMAP
- Port 2049 (UDP): NFS
- Ports 6000 to 6xxx (TCP): X11
Filter Example

<table>
<thead>
<tr>
<th>Action</th>
<th>ourhost</th>
<th>port</th>
<th>theirhost</th>
<th>port</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>*</td>
<td>*</td>
<td>BAD</td>
<td>*</td>
<td>untrusted host</td>
</tr>
<tr>
<td>allow</td>
<td>mailhst</td>
<td>25</td>
<td>*</td>
<td>*</td>
<td>allow our SMTP port</td>
</tr>
</tbody>
</table>

Apply rules from top to bottom with assumed *default* entry:

<table>
<thead>
<tr>
<th>Action</th>
<th>ourhost</th>
<th>port</th>
<th>theirhost</th>
<th>port</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>default</td>
</tr>
</tbody>
</table>

Problematic entry intended to allow outbound connections to SMTP from inside:

<table>
<thead>
<tr>
<th>Action</th>
<th>ourhost</th>
<th>port</th>
<th>theirhost</th>
<th>port</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>25</td>
<td>connect to their SMTP</td>
</tr>
</tbody>
</table>

This allows all connections from port 25, but an outside machine can run *anything* on its port 25!
Filter Example Continued

<table>
<thead>
<tr>
<th>Action</th>
<th>src</th>
<th>port</th>
<th>dest</th>
<th>port</th>
<th>flags</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>132.239.6.*</td>
<td>*</td>
<td>*</td>
<td>25</td>
<td>*</td>
<td>their SMTP</td>
</tr>
<tr>
<td>allow</td>
<td>*</td>
<td>25</td>
<td>*</td>
<td>*</td>
<td>ACK</td>
<td>their replies</td>
</tr>
</tbody>
</table>

Permit *outgoing* calls to port 25.

This filter doesn’t protect against IP address spoofing. The bad hosts can “pretend” to be one of the hosts with addresses 132.239.6.*.
Principles for Firewall Configuration

- **Least Privilege:**
  - Turn off everything that is unnecessary
    (e.g. Web Servers should disable port 25 [SMTP])

- **Failsafe Defaults:**
  - By default should reject
  - (Note that this can cause usability problems...)

- **Egress Filtering:**
  - Filter outgoing packets too!
  - You know the valid IP addresses for machines internal to the network, so drop those that aren’t valid.
  - This can help prevent DoS attacks in the Internet.
Proxy-based Firewalls

- Proxy acts like *both* a client and a server.
- Able to filter using application-level info
  - For example, permit some URLs to be visible outside and prevent others from being visible.
- Proxies can provide other services too
  - Caching, load balancing, etc.
  - Key escrow (e.g., reverse proxies for ssh, SSL)
Firewalls Pro/Con

- **Benefits**
  - Reduced “attack surface” against external attackers
  - Filter out lots of “noise” in network traffic
    (helps focus attention)
  - Reduced liability (common practice)

- **Costs**
  - Actual cost: both hardware and administration
  - Bottleneck and single point of failure on network
  - False sense of security
    - Limited language (addresses, ports); doesn’t help with worms/viruses, ssh exploits, cross-site scripting, etc
    - Inside vs outside model is fragile (once an internal host is compromised firewall does no good); What about wireless laptops?
Network Address Translation

- Idea: Break the invariant that IP addresses are globally unique
Typical NAT Behavior

- NAT maintains a table of the form:
  \(<\text{client IP}> \ <\text{client port}> \ <\text{NAT ID}>\)

- Outgoing packets (on non-NAT port):
  - Look for client IP address, client port in the mapping table
  - If found, replace client port with previously allocated NAT ID (same size as PORT #)
  - If not found, allocate a new unique NAT ID and replace source port with NAT ID
  - Replace source address with NAT address
NAT Behavior

- Incoming Packets (on NAT port)
  - Look up destination port number as NAT ID in port mapping table
  - If found, replace destination address and port with client entries from the mapping table
  - If not found, the packet is not for us and should be rejected

- Table entries expire after 2-3 minutes to allow them to be garbage collected
NAT Pro/Con

▪ Benefits
  – Only allows connections to the outside that are established from *inside*.
    ▪ Hosts from outside can only contact internal hosts that appear in the mapping table, and they’re only added when they establish the connection
  – Don’t need as large an external address space
    ▪ (i.e. 10 machines can share 1 IP address)

▪ Costs
  – Rewriting IP addresses isn’t so easy (what if they appear in protocol data? Then what happens to sequence numbers?)
  – Breaks some protocols (e.g., streaming protocols commonly have client invoke server and then server opens new connection to client)
Network content analysis

- Lots of devices want to look at network traffic for security
  - Network Intrusion detection/prevention Systems (NIDS/NIPS)
    - Try to find signatures of attacks or malware
  - Spam filters
    - Try to detect unwanted e-mail
  - Data leakage
    - Try to prevent sensitive information from leaving company
  - Traffic differentiation
    - Filter or slow down BitTorrent/Magneto traffic (e.g. Comcast)
Challenges

▪ Expensive to look into each packet
  – 10Gbps -> ~1M packets per second... ns’ per byte

▪ Network vantage point is imperfect
  – What does a packet mean?
  – What if a session is encrypted?
  – Network evasion?
Network evasion

- Typically network intrusion detection systems are deployed like firewalls (between internal network and Internet)
- Key assumption is that NIDS sees the same traffic as destination host
- Not quite true...
  - Lots of ways to evade a NIDS by exploiting ambiguity
TTL evasion

- Suppose destination host is 2 hops inside network after NIDS box
- This is what NIDS sees (each letter is one pkt)

```
<table>
<thead>
<tr>
<th>ttl=2</th>
<th>ttl=2</th>
<th>ttl=2</th>
<th>ttl=2</th>
<th>ttl=1</th>
<th>ttl=2</th>
<th>ttl=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>A</td>
<td>M</td>
<td>B</td>
<td>E</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>
```

- This is what the destination host sees

```
<table>
<thead>
<tr>
<th>ttl=1</th>
<th>ttl=1</th>
<th>ttl=1</th>
<th>ttl=1</th>
<th>ttl=1</th>
<th>ttl=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>A</td>
<td>M</td>
<td>B</td>
<td>A</td>
<td>D</td>
</tr>
</tbody>
</table>
Sequence # evasion

- Suppose attacker sends two packets

```
TCP Seq #
1 2 3 4 5
I A M B E
```

- What does destination see?
  - IAMBED? IAMBAD? IAMBD? Depends on host

- Lots of other evasion techniques...
Solution

- Protocol normalization
  - NIDS rewrites packets to remove all ambiguity
  - E.g.
    - all packets have ttl rewritten to reach any internal destination
    - IDS tracks each flow and does not allow overlapping packets

- Can be very tricky to get right and expensive
  - Potentially must buffer large amounts of data
  - What if you get seq #2 through #100, but not seq #1?
  - Tradeoff: when to drop data vs when to buffer
Switching gears: DNS

- We humans do not tend to remember 32bit IP numbers...

- Solution: domain names
  - Human readable identifiers (e.g., www.cs.ucsd.edu)

- Problem: how to map DNS names to IP addresses?
  - In the old days we had a big file – literally (download from sri-nic.arpa)
  - Today we use a distributed name servers called the Domain Name System (DNS)
Domain Name System (DNS)

- Hierarchical Name Space
DNS Root Name Servers

- Hierarchical service
  - Root name servers for top-level domains
  - Authoritative name servers for subdomains
  - Local name resolvers contact authoritative servers when they do not know a name
DNS Lookup Example

DNS record types (partial list):
- **NS**: name server (points to other server)
- **A**: address record (contains IP address)
- **MX**: address in charge of handling email
- **TXT**: generic text (e.g. used to distribute site public keys (DKIM))
Caching

- **DNS responses are cached**
  - Quick response for repeated translations
  - Useful for finding servers as well as addresses
    - NS records for domains

- **DNS negative queries** are cached
  - Save time for nonexistent sites, e.g. misspelling

- Cached data periodically times out
  - Lifetime (TTL) of data controlled by owner of data
  - TTL passed with every record
Basic DNS Vulnerabilities

- Users/hosts trust the host-address mapping provided by DNS:
  - Used as basis for many security policies:
    - Browser “same origin” policy, URL address bar, user trust

- Obvious problems
  - Interception of requests or compromise of DNS servers can result in incorrect or malicious responses
    - e.g., if you can observe request then can spoof response
    - e.g., hijack network route to spoof DNS

- Less obvious
  - Name server can delegate name to another ns and then may also supply its IP address (trouble)
What do you do with DNS poisoning?

- One example: pharming
  - Change IP addresses to redirect URLs to fraudulent sites
  - Potentially more dangerous than phishing attacks
  - Potentially allows man-in-the-middle attack
  - No email solicitation is required

- Lots of DNS poisoning attacks have occurred in the wild
  - January 2005, the domain name for a large New York ISP, Panix, was hijacked to a site in Australia.
  - In November 2004, Google and Amazon users were sent to Med Network Inc., an online pharmacy
  - In March 2003, a group dubbed the "Freedom Cyber Force Militia" hijacked visitors to the Al-Jazeera Web site and presented them with the message "God Bless Our Troops"
**DNS Packet**

- **Query ID:**
  - 16 bit random value
  - Links response to query

---

<table>
<thead>
<tr>
<th>ver</th>
<th>hlen</th>
<th>TOS</th>
<th>pkt len</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>flg</td>
<td>fragment offset</td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>protocol</td>
<td>header checksum</td>
<td></td>
</tr>
</tbody>
</table>

- **Source IP address**
- **Destination IP address**

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP length</td>
<td>UDP checksum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query ID</th>
<th>Opcode</th>
<th>AN</th>
<th>Z</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question count</td>
<td>Answer count</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authority count</td>
<td>Addl. Record count</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DNS question**
**or answer data**

(from Steve Friedl)
Resolver to NS request

<table>
<thead>
<tr>
<th>IP</th>
<th>32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>ver</td>
<td>hlen</td>
</tr>
<tr>
<td>identification</td>
<td>flg</td>
</tr>
<tr>
<td>TTL</td>
<td>protocol</td>
</tr>
<tr>
<td>src IP = 68.94.156.1</td>
<td></td>
</tr>
<tr>
<td>dst IP = 192.26.92.30</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UDP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>src port = 5798</td>
<td>dst port = 53</td>
</tr>
<tr>
<td>UDP length</td>
<td>UDP cksum</td>
</tr>
<tr>
<td>QID = 43561</td>
<td></td>
</tr>
<tr>
<td>Q: What is a record for <a href="http://www.unixwiz.net">www.unixwiz.net</a>?</td>
<td></td>
</tr>
<tr>
<td>RD=1 - recursion desired</td>
<td></td>
</tr>
<tr>
<td>OP=0 - standard query</td>
<td></td>
</tr>
<tr>
<td>QR=0 - this is a query</td>
<td></td>
</tr>
</tbody>
</table>

dns1.sbcglobal.net
c.gtld-servers.net
Response to resolver

Response contains IP addr of next NS server (called “glue”)

Response ignored if unrecognized QueryID
DNS additional section

- Answers to questions you didn’t ask...
  - There is a good reason for it... if I tell you that the name server for foo.com is ns1.foo.com... how do you find its IP address?

- But this is a problem... what if I run a DNS server for foo.com and when I get asked for the IP address for bar.foo.com, I put some additional stuff in the “additional section” like:
  - You can find the IP address for paypal.com at 41.2.6.2 (address I control)
  - And you can also find the IP address for amazon.com and chase.com there too...
Early Attack Strategy


2. The user’s computer asks the targeted name server to translate www.BadGuysAreUs.com into an IP address.

3. The targeted name server has not cached the address, so the query is routed through a root name server, a .com name server, and finally the BadGuysAreUs.com name server.

4. The BadGuysAreUs name server responds with an IP address but adds a false IP address for a completely different Web site, www.paypal.com.

5. The targeted name server stores the false IP address for paypal.com.

6. When people using this name server attempt to go to www.paypal.com, they are directed to a Web site that looks like PayPal’s but works only to harvest their user names and passwords.
Authoritative response to resolver

**bailiwick checking:**
response is cached **only**
if it is within the same
domain of query
(i.e. a.com cannot
set NS for b.com)

---

**final answer**
Bailiwick Checking Rule from BIND

But we forgot something

- A decade goes by and Dan Kaminsky realizes that the bailiwick checking rule doesn’t really protect us

- Unnoticed hole that allows arbitrary DNS poisoning at a distance

- Biggest operation to do secret mass migration of DNS infrastructure ever
DNS cache poisoning  (a la Kaminsky’08)

- Victim machine visits attacker’s web site, downloads Javascript

![Diagram of DNS cache poisoning]

**Diagram Explanation:**
- Local DNS resolver sends the query with QID=`x_1`.
- Attacker's resolver returns a response with `ns.bank.com` and `attackerIP`.
- The resolver caches 256 such responses, each with a random QID.
- Attacker wins if there exists a QID `j` such that `x_1 = y_j` and the response is cached, and the attacker owns `bank.com`.

**Additional Details:**
- **Response:**
  - `NS  bank.com=ns.bank.com`
  - `A  ns.bank.com=attackerIP`

**User Interaction:**
- User visits attacker's website and downloads Javascript.
- Resolver sends query to attacker's resolver.
- Attacker's resolver responds with cached responses.
- Attacker wins if cached response matches attacker's domain.

**Attack Scenario:**
- Victim visits attacker's website.
- Javascript downloads and modifies DNS cache.
- Attacker's resolver serves cached response.
- Victim's resolver caches response, now pointing to attacker.

**Technical Details:**
- QID (Query ID) is used to correlate responses.
- Attacker manipulates cache to redirect victim traffic.
- Security implications include phishing and DoS attacks.

**Further Reading:**
If at first you don’t succeed ...

- Victim machine visits attacker’s web site, downloads Javascript

```
user browser
    ▸ Query: b.bank.com
    ▸ Local DNS resolver
    ▸ b.bank.com QID=x_2
    ▸ ns.bank.com
```

256 responses:
Random QID: y_1, y_2, ...

- NS bank.com=ns.bank.com
- A ns.bank.com=attackerIP

attacker wins if \( \exists j: x_2 = y_j \)

response is cached and attacker owns bank.com

success after \( \approx 256 \) tries (few minutes)
Kaminsky DNS Attack

1. An attacker issues a DNS query for the nonexistent aaa.paypal.com.

2. The attacker immediately sends fake responses to his own query, each containing a different query ID number and a false IP address for www.paypal.com.

3. The attacker repeats steps 1 and 2 using different prefixes: aab.paypal.com, aac.paypal.com, and so on, until the targeted server finally accepts a spoofed response. The spoofed response "poisons" the cache of the name server with a false address for www.paypal.com.

4. Users accessing www.paypal.com through the poisoned name server are directed instead to a Web site that looks like PayPal’s but works only to harvest their user names and passwords.
Defenses

- Increase Query ID size. How? Some proposals
  - Randomize src port, additional 11 bits
    - Now attack takes several hours
  - Ox20 encoding – randomly vary capitalization (DNS is case insensitive)
    - check that you get same capitalization back
  - Ask every DNS query twice:
    - Attacker has to guess QueryID correctly twice (32 bits)
    - Doubles load on DNS system
- Try to detect poisoning
  - Ignore responses not directly necessary to query
- Authenticated requests/responses
  - Provided by DNSsec (digital signatures on DNS records) ... but few domains use DNSsec
DNS Summary

- Current DNS system does not provide strong evidence binding request to response
- Response can provide more data than was asked for
- Together allows attacker to “poison” DNS and divert traffic to their sites
Denial-of-service

- Attack against *availability*, not confidentiality, integrity, authenticity, etc.

- Two kinds of attacks:
  - Logic vulnerabilities: exploit bugs to cause crash
    - e.g. Ping-of-Death, Land
    - Fix via filtering and patching
  - Resource consumption: overwhelm with spurious requests
    - e.g. SYN flood, Smurf, bandwidth overflow
    - Much tougher to fix...

- Distributed denial-of-service attacks (DDOS)
  - Lots of hosts attack a victim at once
Resource consumption of Service

- **Server CPU/Memory resources**
  - Consumes connection state (e.g. SYN flood)
  - Time to evaluate messages (interrupt livelock)
    - Some messages take “slow path” (e.g. invalid ACK)
  - Can cause new connections to be dropped and existing connections to time-out
  - Make DB process lots of queries
    - Attack cache – lots of random queries

- **Network resources**
  - Many routers packet-per-second limited, FIFO queuing
  - If attack is greater than forwarding capacity, good data will be dropped
What to do?

- Defenses against address spoofing
  - For attacks from randomly spoofed addresses
- Filtering based on attack features or IP address
- Make attacker do work
- Buy more resources
Address spoofing

- Filter packets with incorrect source addresses
  - Network egress: filter packets on a link whose source addresses are not reached using the link as the next hop
  - Network ingress: filter packets whose source address are not in the routing table at all

- SYN Cookies
  - Issue: allocating per TCP session state is expensive (that’s why the SYN flood attack works)
  - Delay allocation of state until remote host commits to three-way handshake
  - Send back SYN/ACK packet without allocating state on server; server’s initial sequence number (ISN) encodes a secret “cookie” that is function of src, dst, srcport, dstport and time.
  - Allocate state when client sends ACK to server’s SYN/ACK (using cookie to validate)
Address spoofing(2)

- Puzzles
  - Don’t commit state until client has done a bunch of “work” for you (i.e. solved computationally tough problem)
  - Server provides puzzle to client
    - Hardness can be determined by load
  - Client must solve puzzle (easy to verify by server) to allocate state

  - Tricky: if validation isn’t super cheap, bad guy can sent lots of invalid puzzle solutions to server
Address spoofing(3)

- **TTL filtering**
  - From a given host the TTL is decremented by a certain number of hops (based on network topology)
  - Std IP implementations set the packet TTL value to a small set of values (32, 64, 128, 255) [can normalize because Internet diameter is mostly < 32]
  - Thus, keep track of TTLs for each source network and if attack starts, filter packets whose TTLs are inconsistent
Address spoofing(4)

- **Traceback**
  - Router support for tracking packets back to their source
  - Probabilistic packet marking [Savage, etc]
    - With some probability p, a router encodes the identity of the link the packet will traverse. Victim uses these packet “marks” to reconstruct path back to victim
  - Packet logging [Snoeren, etc]
    - Routers hash packet header and store in database
    - Victim queries router about whether they’ve seen a given packet and if so, from where... repeat
  - A range of others...

- **Main issue:** then what? What do you do with a list of 200k hosts attacking you?
Packet filtering

- Idea, if there is a common feature to the packet (i.e. “Die, you loser” in the payload) then look for those packets and drop them.
- If no feature exists then find way to add a “good” feature
  - Hash for packets from well-behaved connections
  - Filter IP addresses that aren’t solving puzzles
  - Integrate with traceback so packet marks for spoofed packets can be used to filter
- Instead of dropping packets, can simply rate-limit packets that are suspicious
- Third-party services will offer these capabilities on your behalf in the network
Buy more resources

- Large content distribution networks (e.g. Akamai, CloudFlare) can handle very large attacks

- Each attacker gets diverted to local Akamai server
  - Total bandwidth Akamai can handle is the product of the bandwidth to all Akamai servers
  - Akamai has weathered attacks in excess of 1TB/s

- Issue: who pays for that? $$$
Reflection attacks

- Spoof source address to be that of victim

- Common example
  - Send name server request to 1000s of DNS servers *on behalf* of victim
  - All name servers send responses to victim

- Advantages (for attacker)
  - Amplification: frequently responses >> requests
  - Anonymity: attack doesn’t come from attacker’s machines

- Solution: try not to have “open” Internet services that allow this
DoS Summary

- In general, some of the toughest problems to solve
  - Network service model allows unsolicited requests
  - Bad guys can leverage large # of resources
  - Hard to attribute network actions
  - Few systems can account for effort spent per request or isolate impact of some requests from others

- DDoS-based extortion and retribution (e.g., against security companies) is not uncommon
Next time

- Malware and criminal use (e.g., botnets, etc)