Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.829 Fall 2002 **Security Problems with the Internet Architecture** September 26, 2002

1 Introduction

To this point, we've learned that the Internet was designed with many of the following features in mind:

- Scaling
- Heterogeneity
- End-to-end Principle (Complexity at edges)
- Autonomy/Flexibility

So what about security? In fact, we'll see that some of the design decisions that were meant to accommodate the above actually result in *worse* security. Why might this be the case? In some cases, reasonable security mechanisms might seem to break fundamental design principles and therefore security was not really taken into consideration until later (or after it was too late). In other cases, we'll notice some design oversights that have been fixed.

To get a rough idea of where priority for security falls, just take a grep through some of the RFCs. In particular, roughly 600 of them explicitly acknowledge that they "punt" on security issues. It's not just standards bodies that have a tendency to ignore secuurity concerns, too. Last time, we talked about how router vendors are constantly trying to pack more features into their routers. Given the choice of incorporating latest new feature X or mitigating security problems, vendors are more likely to be inclined to add features.

But how big of a problem is security, anyhow? Are we really just talking about theoretical attacks? Not really. The Honeynet Project (`http://project.honeynet.org/`) has set up a "honeypot" network, given it an enticing name and juicy content (of course, they won't tell you where it is, and it regularly changes), and watched what happens. (Their study was from July 2001.) They found that a random computer on the Internet is scanned dozens of times per day (you might see these scans yourself if you run your own machine); the average "life expectancy" of a RedHat 6.2 server before being hacked was a mere 72 hours (the minimum time was 15 minutes!). A Windows98 machine with standard file sharing was hacked 5 times in 4 days!

But why should we, as people in the networking community care about people's machines getting hacked? It's a personal problem, right? Not exactly. These machines are commonly hacked by a single entity, and turned into a platform for mounting a DDoS attack. A study from CAIDA from August 2001 saw 12,000 attacks against 5,000 distinct targets in 2,000 distinct organizations over a 3-week period [9]. The recent Apache/SSL worm, for example, has been found to occupy a fair amount of bandwidth with its control traffic alone.

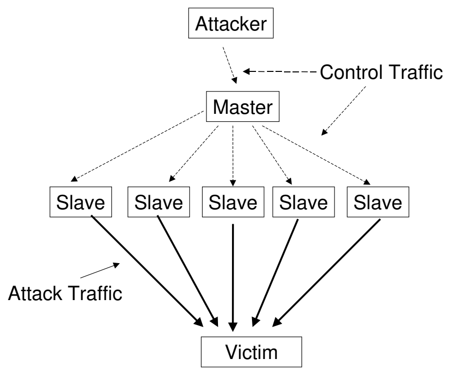

Figure 1: Architecture of bandwidth attacks.

The remainder of the lecture is divided into three parts. We first discuss security weaknesses of the Internet architecture; next, we explore features of the Internet architecture that have made these weaknesses possible. Finally, we discuss possible defenses and describe to a few studies that have been conducted to better understand the nature of attacks in today's Internet.

2 A Taxonomy of Attacks

We'll categorize attacks into three main categories: Denial of Service (DoS) and Distributed DoS (DDoS) attacks, intrusion attacks, and control path attacks. Note that there is a fourth category, infrastructure attacks (i.e., taking a bomb to a central office, etc.), but we'll not concern ourselves with infrastructure attacks in this lecture.

2.1 DoS/DDoS Attacks

Denial of Service (DoS) and Distributed DoS (DDoS) attacks typically have one of two intents: to exhaust the resources of the targetted host, or to exhaust the bandwidth of a particular link (of course, it's possible to kill two birds with one stone).

These types of attacks have gained a fair amount of media attention in recent years. With good reason: if you remember, in February 2000, a DDoS attack took down eBay, Yahoo, CNN, and several other sites for several hours. How was this possible? We'll look at this briefly.

2.1.1 Flooding Attacks

A DoS attack will typically exhaust the bandwidth of a particular link using a technique known as "flooding" — sending a lot of packets down a particular link. The problem with mounting a flooding attack as a traditional-style DoS attack is that the resources required to mount such an attack require more than is typically available to a vanilla home user (particularly bandwidth, since a lot of these kiddies are dialup users).

The solution, then, is to gain control over a large number of machines, to act as "slaves" as shown in Figure 1. Each slave need only send a small amount of traffic towards the target. However, in the aggregate, the victim's bandwidth will be largely occupied by traffic originating from the slaves. What are common ways to implement this? One possibility is with ICMP and IP address spoofing; note that this attack doesn't require much control over the hosts at all. Consider the following sequence of steps:

1. The master issues commands to the slaves, instructing them to mount an attack.
2. Slaves issue an ICMP echo request, with the source address spoofed to be the address of the victim's machine.
3. The slaves all send ICMP echo reply packets to the victim's machine.

Ugly. But even easier things are possible. Some networks (called smurf aplifier networks) will respond to an echo request sent to the network broadcast address. If the source address of that ICMP packet is set to the victim's IP address, the victim is going to get a lot of ICMP replies! The best flooding attacks work by sending many small packets at a very high packet per second rate, since *routers are not typically limited by bandwidth, but by packet processing rate.*

Other common DDoS tools are known as trinoo, TFN, Stacheldraht, etc. All of which have multiple attack options, including UDP, ICMP, TCP SYN, and broadcast ping floods. CodeRed is a more recent example, which exploited a known bug in Microsoft IIS Web servers.

Two other types of flooding attacks, "SYN Floods" and SSL request floods, are indended to exhaust the resources of the target host. Recall the TCP 3-way handshake. The initiator of the connection sends a SYN packet to the server, at which point the server goes into a `SYN_RECD` state. So what's the problem? The server has devoted resources to a client, but the client didn't have to do anything but send one packet! (The server is left with a bunch of half-open connections; these connections will eventually time out and close, but, as we know, that may take awhile.) Therefore, it's easy for an attacker (or attackers) to bombard the target with SYN packets, since it takes very little resources on the part of the attacker.

As another example of a flooding attack that is intended to exhaust a server's resources, consider the Secure Sockets Layer (SSL) protocol, which is most commonly used for things like secure Web transactions. The way the SSLv3 is specified, after the `ServerHello` message is received by the server, the server sends a certificate to the client, and the client send a *pre-master secret* for the session, encrypted with the server's public key (from which the session keys are later derived). The point here is that the client can force the server to perform an RSA decryption (an expensive operation) without doing any "real work" itself. One study showed that 800 kbps of traffic is enough to bring an SSL server to its knees [4].

2.1.2 Spam

Email spam, in some ways, can be considered a denial of service attack — it slows down mail relays, fills up mail queues, etc. Spammers typically use one of several methods to send mail. A formerly popular method that seems to becoming less popular is to use an *open mail relay* (i.e., a mail server that will send mail from anyone, to anyone) to send spam. The advantage to this method is that the spammer does not need to buy a high-bandwidth pipe to send large amounts of spam; rather, it can just use an open mail relay to send one message with hundreds or thousands of people in the `To:` field, thereby making the relay do all of the work. However, people are becoming more aggressive about shutting down open mail relays (see Secrion 4), so it's becoming more common for spammers to buy a T1 line, etc. and use their own bandwidth to spam.

Spammers also have been known to make use of common intrusion and control path attacks to send spam. Something related to wireless insecurity and intrusion (Section 2.2) is that a technique called "drive-by" spamming is gaining popularity: a spammer will get bandwidth by hopping onto an existing unprotected wireless network, sned a bunch of mail, and go away. While being fairly difficult to track, this method is more blatantly illegal than most. To make the source of spam difficult to track, spammers may also use the routing infrastructure to their advantage — by advertising a prefix block, sending spam from some address in that block, and then withdrawing the prefix. This method is also used in other types of attacks, described in Section 2.3.

2.1.3 Congestion Control

A TCP client may lie about the losses that it is experiencing to attempt to gain more bandwidth.

2.2 Intrusion

While there are many types of intrusion attacks that are the result of weak host security (e.g., remote buffer overflow exploits, etc.), in this section we focus on intrusion attacks that result from weaknesses in network protocols. We'll mostly focus on weaknesses in TCP, but also briefly discuss client authentication problems on the Web and link-layer intrusion attacks.

2.2.1 TCP Connection Hijacking

Intrusion attacks due to attacks on TCP are typically the result on "connection hijacking" attacks — IP address spoofing allows an end host to pretend as though it's another host. This was particularly deadly for older services, such as *rsh*, that authenticated a host solely on the origin of the packets. In short, a connection hijacking attack is a way to either overtake someone else's TCP connection, or make it look as though the TCP connection endpoint was coming from someplace else. TCP connection hijacking really requires several steps:

1. Spoofing the IP address of the packets, to make them appear as though they have originated from the hijacked connection.
2. Guessing the initial sequence number that the server will send to the client to set up the connection.
3. Making sure the spoofed client doesn't respond (e.g., with a FIN packet) to the server.

The first and third steps are relatively easy (although there are some defenses against the first, which we will discuss later). The hard (or, in some cases not-so-hard) part is guessing the initial sequence number (ISN) that the server returns to the spoofed IP address. How does one do this? The attacker could make a few legitimate TCP connections to the server himself, notice the pattern by which the ISN increments, and make an educated guess about the ISN that the server returned for the connection he's trying to hijack. In the early days of the Internet, this number was chosen quite predictably. A reasonable solution to this seems to be to choose a random increment to the ISN.

Historically, this has not seemed to solve the problem. People scrambled to use pseudo-random number generators to select ISNs, but weak PRGs have the propery that it's possible to guess the next "random" number by knowing some of the previously output random numbers. Even as recently as May 2001, weaknesses in methods that add a series of numbers together can reduce the variance on the ISNs, thus making ISNs more predictible (this is a result of the Central Limit Theorem) [3].

In 1996, Steve Bellovin noted that using random ISNs alone is not really a good fix at all [1]. We'll discuss this, as well as other defenses to connection hijacking in Section 4.

2.2.2 Web Client Authentication

A recent study showed that, in many cases in the real world, people are using very weak schemes in an attempt to provide client authentication [5]. Since SSL/TLS is a bit too heavyweight for making sure that someone is authorized to read *The Wall Street Journal* online, etc., many sites have come up with "home-brewed" client authentication schemes that allow an interrogative adversary to gain unauthorized access with relative ease. Many of the problems that are related to Web client authentication can be traced to the fact that HTTP was designed to be a stateless protocol.

2.2.3 Link-Layer Attacks

802.11 wireless networks transmit data via radio; the data sent over wireless networks is supposedly protected by a built-in security feature, called Wired Equivalent Privacy (WEP), which encrypts data as it is being transmitted. However, in August 2001, a weakness in the key scheduling algorithm of RC4 stream cipher was used to crack the key that WEP used to encrypt its data. The attack is ciphertext-only and grows linearly with the size of the key. The basic idea is that a large number of keys are "weak", where the initial outputs of the stream cipher are affected by a very small number of the key bits. More seriously, however, is a key vulnerability that results when the same secret part of the key is used with many different ciphertexts, it's possible for the attacker to recover the the initial word of the keystreams with relatively little work.

2.3 Control

Most of the attacks presented above, while as yet unsolved, have reared their ugly heads in the wild. We now turn to a class of attacks that, while less observed to this day, could have catastrophic consequences. As we know, the Internet requires routing (e.g., BGP) and naming (e.g., DNS) protocols for interconnectivity. An attack on one of these control paths could, in certain cases, result in severe problems. We discuss in turn naming attacks and routing attacks.

2.3.1 Naming Attacks

The Domain Name System (DNS) is used to map names (e.g., `www.etrade.com`) to IP addresses (e.g., `12.153.224.22`). An attack that is geared to wreck the mapping of names to IP addresses (or otherwise degrade the DNS) could obviously have some bad consequences. How might one mount such an attack?

One way would be to mount an attack on the DNS is to attack the top-level domain servers (i.e., those which maintain the authoritative DNS servers for .com, etc.). Some solutions have been posed for protecting the top-level DNS [2], but there still is much work to be done in reducing the vulnerability of the top-level DNS servers. In October 2001, there was a security breach on the .AU TLD server (details on the incident are not readily available). While there are something like 13 top level DNS servers, one could imagine that a determined attacker could mount a DDoS attack against these servers to wreak havoc.

One attack that requires some social engineering, but was mounted as recently as August 2001, is to convince a large ISP to update their DNS records to point to the attacker's DNS server as an authoritative DNS server for a particular domain, and set the TTLs for the bogus DNS replies to be extremely high.

2.3.2 Routing Attacks

Note that, in some sense, the security of a BGP session rests on the security of the underlying TCP connection over which BGP messages are exchanged. If one was able to mount a TCP hijacking attack on a BGP session (such ISN vulnerabilities were present in some versions of Cisco IOS as recently as March 2001), then an attacker would have the ability to arbitrarily insert routing messages. To protect against hijacking, BGP sessions will commonly authenticated messages using the TCP MD5 option [6].

More serious problems exist in verifying the validity of routing advertisements heard on a BGP session with a neighbor AS. It is an administrative nightmare to keep track of who is allowed to annouce which prefixes, and verifying the validity of all attributes in the BGP announcement (for example, it's tough to verify that the advertised AS path actually is the AS path that should be advertised).

We've already discussed how spammers can use routing attacks to their advantage. Those attackers interested in doing DNS cache poisoning can use similar tricks to limit their traceability.

Note that misconfiguration of routing protocols like BGP can sometimes result in connectivity issues [8]. Along these lines, a large ISP was known to have connectivity problems when another ISP accidentally starting announcing reachability to a prefix that contained the DNS servers for the large ISPs dialup service. This is commonly referred to as "blackholing". The fix? Since there's no good way to prevent someone from making such advertisements (aside from making phone calls and yelling at the appropriate people), the quickest fix is to temporarily advertise a slightly longer prefix (or prefixes) for that address region until the problem is resolved.

3 Weaknesses in the Internet Architecture

Now that we have a basic idea about the classes of attacks that exist, let's discuss the nature of these types of attacks? Why do they exist? In many cases, these weaknesses resulted in poor design decisions. In others, however, the weaknesses are a result of an explicity choice that was made to favor a different design goal. We'll focus our discussion on the design tradeoffs that were made.

3.1 Scalability vs. Accounting

Accounting in the phone network is pretty nice. For example, the phone network folks really have a good idea about the statistics of all calls that were placed: who made them (i.e., from what number they were placed), the destination of the call, how long they lasted, etc. We don't have such luxuries in the Internet. Part of this results from the fact that the Internet is not a circuit-switched network; packets can be sent without any state being present in the network. But, without the presence of state, it's hard to verify that a packet (or group of packets) ever traversed an exact path, or even that a particular path existed at a given point in time. We'll look at a particular solution in Section 4 called *traceback* that trades off maintaining some state in routers to achieve some level of accounting.

3.2 Flexibility vs. Accountability

Prefixes can come and go, and there is no hierarchy implied by the IP addressing itself. In theory, a particular prefix and mask $a.b.c.d/m$ could be attached via a wide variety of autonomous systems. This provides nice flexibility — if an ISP decides to change its upstream provider, then it doesn't have to change its IP addresses. However, the face that this flexibility exists makes it more difficult for an upstream provider if a particular address should be advertised from a particular network (this administrative nightmare becomes more severe for networks that are closer to the core). Thus, it becomes reasonably easy for someone connected to a large ISP to "spoof" IP addresses, since that ISP is not likely to check the validity of the source IP address. Note that checking the source IP is an operation that requires resources that can potentially more difficult to perform as line rates increase.

3.3 Flexibility/Soft-state vs. Auditing

Continuing along the lines of a flexible design that allows a certain prefix block to be advertised from anywhere, we note that prefix advertisements may come and go. This might happen for various reasons: certain sub-aggregates may come and go for traffic engineering purposes, etc. The point is that the routing infrastructure allows for this flexibility, and, thus, it's possible for routing advertisements to appear for a short period of time.

Aside from the problems presented above, you should also notice that this added flexibility, combined with the fact that the network should be soft-state creates an auditing problem — information goes away, and many ISPs don't have the capability (or desire) to log every announcement that they have ever heard on every BGP session (a mammoth task). Thus, it is possible for certain attacks to be mounted from transient prefixes that are reasonably untraceable. If we knew that a certain prefix were always originated via a particular ISP, we could trace the path back with relative ease, but the need to flexible addressing eliminates that possibility.

3.4 Autonomy vs. Coordination

The Internet has developed as a conglomeration of many independently operated networks. This has many nice properties — in particular, how one person operates his or her network (i.e., the interior routing protocols that are chosen for that network, internal configuration, topology, etc.)

can be decided completely independently from how others operate their networks. This provides a nice degree of autonomy, while still allowing for global reachability.

However, this federation-style arrangement makes tracking a packet back to its source a coordinated effor that requires cooperation across multiple domains, which can be tricky, both technically and from an administrative standpoint.

4 Defenses

In this section, we discuss defenses against the types of attacks we have discussed in Section 2. We consider each type of attack in turn.

4.1 DoS/DDoS Attacks

There are two basic classes of defenses to DoS/DDoS attacks: *proactive* defenses, which attempt to thwart the attack in the first place, and *reactive* defenses, which take some approach to mitigate the attack after it has begun.

4.1.1 Proactive Defenses

We'll first consider proactive defenses against the resource attacks, i.e., SYN flooding, and SSL connection flooding. In both of these cases, the fundamental problem with the protocols is that the attacker can force the target to do work, without having to do any work itself. We can consider fixing this problem in two ways:

1. the target (server) postpones devoting resources/performing work until as late as possible, or
2. force the attacker (client) to do some work *before* the target performs any work

SYN Cookies An approach that takes the first tack is something called "SYN Cookies". When the server receives a SYN packet from a particular host, it sends back a SYN-ACK to that host, *but it does not yet enter the SYN_RECD state.* Rather, it computes the initial sequence number that should be used based on a hash of the properties of that connection:

$$ISN \quad = \quad H(\text{src\_addr}, \text{src\_port}, \text{dst\_addr}, \text{dst\_port}, \text{key})$$

where *key* is some server-specific secret key. Thus, when the third packet in the 3-way handshake comes back, the server need only check to see if the ACK for the ISN (plus one) matches the hash on the easily computable values (at which point the server can then devote resources to the connection). Note that the secret should be rotated with some frequency to prevent reuse of the same ISN from the same source at a later time (but not so fast that the hash can't be checked when the SYN-ACK comes back...close to the max possible RTT seems like a good choice).

Client Puzzles A similar way to prevent against attacks on SSL servers (and also SYN floods [7]) is to require the client (or attacker) to do some work before the server agrees to set up the connection. This is typically called a *client puzzle*. The puzzle is most commonly presented as a partial inversion inversion of a cryptographically secure hash function, i.e.:

$$\begin{aligned} S \rightarrow C &: m', H(x \circ m') = h \\ C \rightarrow S &: x \end{aligned}$$

where the number of bits for x can be chosen based on how difficult the problem should be (i.e., more bits if the client should take a longer time to solve the puzzle).

DDoS Prevention In general DDoS prevention is a very hard problem. There are a few things, however, that network operators can do (or hope for) in order to minimize the possibility of being the victim of, or helping to perpetrate, a DDoS attack.

Smaller networks can add *ingress filters* and *egress filters* to their border routers to control the source IP addresses of packets that traverse network boundaries. Specifically, that claim to be from an IP address that are contained within that destination network, but are coming from outside, should be rejected (ingress filtering). Similarly, packets that attempt to exit a network with a source address that claims to be from outside that network should be dropped (egress filtering). While not universally possible (we've discussed why this is the case), these practices make source IP address spoofing more difficult. In addition, packets that claim to be from a source address that it part of the private address space [10] upon exit from a network should be dropped. A reasonable defense against amplification attacks is to not receive or respond to broadcast traffic, except from specified sources (e.g., DHCP clients).

If an ISP has knowledge about the patterns of traffic that it expects to see on it's network, it can apply traffic shaping to certain classes of traffic. (We haven't discussed traffic shaping yet, but just assume that it can be done.) Note that this requires some knowledge of traffic characteristics. For example, ICMP and SYN packets are commonly rate-limited.

Intrustion detection systems can often detect anomalies in traffic patterns, or recognize well-known attacks that bear certain "signatures", but this is more difficult to do for newer attacks where signatures have not been developed.

Defenses against spam are difficult. The most aggressive stance thus far has been to try to avoid having open relay mail servers. Certain organizations (see `http://www.mail-abuse.org/`) maintain lists of servers that are running open relays, and an ISP may choose to blackhole the mail traffic that is coming from these open relays (this can be done via routing and/or naming). Note that this does not help in the case of preventing direct spamming techniques.

4.1.2 Reactive Defenses

The first step with any reactive DoS defense mechanism is to classify the traffic that is mounting the DoS attack (e.g., where is it coming from? what kind of traffic is it? is it running on a specific port?). This alone can be difficult to do. A common trick is to use access control lists (that permit all packets, but do classification/logging on those packets) on routers as a poor-man's packet sniffer. Once classification is done, appropriate steps can be taken to rate-limit or filter that traffic. The problem is that an attacker is likely to change the pattern of attack (source address, ports, etc.), so the act of classifying and blocking an attack can be a bit of a cat-and-mouse game.

Pushback One way of rate-limiting an attacker is via pushback [8]. It's more generally a mechanism for controlling high bandwidth aggregates. The essence of the idea is that one router can ask its upstream routers to control an aggregate. When serious congestion is detected, the router finds the responsible aggregate. The scheme then imposes rate limiting on high-bandwidth aggregates and, in some cases, will ask an upstream router to rate limit a particular aggregate. Pushback is typically invoked if there is information that a DoS attack is in progress, or if the drop rate for a rate-limited aggregate remains high for several seconds. Some issues with pushback are deployment issues, authentication of pushback messages, and the effect pushback may have on legitimate traffic.

Traceback In addition to mitigating the effects of an attack, the network that is the victim of the attack would find it very useful to be able to trace the path of a packet back through the source of the network. The basic idea is that one network can tell with relative ease that the attack is coming from a specific neighbor; the operator of that network asks the operator to trace the path back an additional hop, and so forth. It would be nice to automate this process in some fashion. Hence, the topic of today's reading on hash-based IP traceback [11].

Why is traceback hard in general? The first problem arises from the fact that IP addresses of attackers are commonly spoofed, concealing the true source of the packet. Performing filtering, etc., can consume a fair amount of resources, and NAT, etc. can complicate things as well. How about the traceback scheme presented in the paper? Packet transformations make it more difficult to capture the "packet snapshot". Auditing can consume a fair amount of resources; storing logs at a very high speed router takes up a lot of space. Other methods use probabilistic packet marking to track large flows.

A couple key ideas from the traceback paper:

- Bloom filters: hash the packet, take the n bit result, map it as a "1" into a 2^n-sized array. Do this for k independent hash functions, that map highly correlated input values as uniformly over the space as possible. $n = 32$ in their implementation.
- Dealing with packet transformations (e.g., packet encapsulation, such as IPSec). Some of these transforms are not invertible, so we need to keep some extra information around. Use a transform lookup table (64 bits: 29 for the digest, 3 for the transform type, 32 for the packet data). If the indirect flag is set in the TLT, the data field is treated as an indirect pointer to an external data structure.

Still, there are many problems with the traceback mechanisms. Coming up with some of the potential problems is the focus of a problem set question.

4.2 Intrusion Attacks

We discuss protection against TCP hijacking attacks. We forego discussion of defenses against the other types of intrusion attacks.

4.2.1 TCP Connection Hijacking

As mentioned, coming up with a nice random ISN increment is not really good enough. The problem here lies in the fact that ISNs are incremented by some random variable accoruding to a mean, a variance, and some distribution. As the number of samples gets large, though, this distribution starts to look like a uniform distrubution with that mean. Thus, a sum of random ISN increments will become more predictable as time goes on, and simply selecting a random increment is not good enough. Bellovin suggests solving the problem by giving each connection (i.e., source and destination port and address) its own sequence number space, and incrementing the ISN *within* that space. Therefore, knowledge of the ISN for one pair of hosts will not all ow an attacker to guess what the ISN might be for a completely different (i.e., spoofed) source.

RFC 2385 describes a TCP MD5 signature option that is intended to prevent against spoofed segments [6]. Note that this option was introduced to protect TCP segments on BGP sessions, with TCP resets being a primary concern.

Another defense against TCP connection hijacking is to use IPSec, which proposes introducing security at the network layer, in a manner transparent to applications. IPSec also has the goal of making eavesdropping more difficult (confidentiality), protecting the identity of endpoints (authenticity), and ensuring that transactions are not modified in flight (integrity). The protocol format itself boils down to two distinct pieces — the authentication header (AH), which is used to ensure the integrity of an IP datagram using a keyed-hash function (e.g., MD5 or SHA-1), and the encapsulating security payload (ESP), which is used to provide confidentiality, as well as integrity or authenticity. In either case, host authentication is done using either pre-shared keys, or some public-key technique (which, of course, requires the existence of a PKI).

IPSec operates in one of two modes: transport mode, or tunnel mode. In *transport mode*, only the payloads of the IP packet are encrypted, and the header is left intact. This allows for processing based on fields in the IP header, and also allows nodes to see the source (which may be spoofed, still) and destination of each packet. An eavesdropper could then notice that IP traffic was sent from A to B, but would not know what application those IP packets were associated with.

In *tunnel mode*, on the other hand, the entire original IP datagram is encrypted and becomes the payload in a new IP packet. This allows routers to act as IPSec proxies, or entry and exit points to the tunnel. This gives the IPSec proxy the ability to forward that packet via some tunnel, and make traffic analysis more difficult.

IPv6 also claims to offer some security enhancements. This will be discussed in tutorial.

4.3 Control-Path Attacks

We discuss defenses against some attacks on the routing and naming infrastructures. We first discuss a proposal to protect against certain types of routing attacks, called Secure BGP (S-BGP), and then briefly discuss what DNSSec is all about and what problems it does (and doesn't) solve.

4.3.1 Routing Attacks

Secure BGP uses a public key infrastructure to support authentication for IP address block and autonomous system number ownership. It also attempts to authenticate a particular router's identity and it's ability to advertise routes on behalf of a certain autonomous system. In addition, it has the ability to ensure that every update was received from a peer was in fact received by that peer, and that the owner of the IP address space advertised as the origin AS in that AS path is actually the owner of that address space.

In S-BGP, routers authenticate each other using IPSec, which handles message integrity problems between routers. Certificates are used to verify that an originating AS owns a specified portion of the IP address space, or that the owner of that space has authorized the AS to advertise that space on its behalf (the top level certificates for ARIN, RIPE, and AP NIC are signed by the root, which in turn sign sub-delegation certificates, and so forth). A separate set of certificates binds a public key to an organization and a set of AS numbers (issued by APNIC, ARIN, or RIPE), and a third binds a public key to an AS number and router ID (issued by the ISP in question). S-BGP uses the concept of *attestations* to allow each BGP speaker along the path to verify that each AS along the route has been authorized by the preceding AS along the route to advertise that path.

There seem to be some very high barriers to deployment, as well as scalability issues with this solution.

4.4 Naming Attacks

In DNSSec, keys are associated with DNS names. DNS resource records are associated with digital signatures, signed by that particular zone. Typically, a zone will have a singly private key. A DNS resolver learns the public key of the zone either by reading it from the DNS or having it staticly configured. Learning it from the DNS requires that that key be signed by some key that the resolver trusts. DNSSec is implemented via additional resource records. Specifically, the SIG resource record cryptographically binds the set of resource records being signed, as well as to a validity interval. What about authenticated responses for non-existent names? For that, there is a new NXT RR, which asserts a range of non-existent names in a zone, in a manner that can be reliably authenticated. What about TTLs? Still have to have them, since they're all about database consistency, but a changing TTL field would be tough to verify, since the signature is changing. The solution to this problem is to have the server sign the original TTL, and send that data along with the current TTL.

5 Studies on Attacks

In order to better understand attacks, it's useful to try to gain some insight into how prevalent they are, how DDoS platforms grow via worm propagation, etc. We'll first survey the the "backscatter" technique, which is a way of inferring DDoS activity on the Internet. For completeness, we should mention Vern Paxson's study on worm propagation, although this will be covered in a future lecture.

5.1 Backscatter

The *backscatter* technique was a scheme introduced with the intent of answering: "How prevalent are denial of service attacks on the Internet today?" The key observation of the backscatter technique is that, for a direct denial over service attack, spoofed source addresses are selected at random. Depending on the type of packet sent by the attacker, a particular type of packet will be returned to the spoofed address from the victim (e.g., a TCP SYN to a closed port will elicit a RST, etc.). The method then assumes that, for an attack of m packets, a site that is monitoring n hosts has probability $nm/2^{32}$ probability of seeing some packet related to that attack. They also use a similar technique to infer the rate of the attack in packets per second (i.e., the actual rate is at least $2^{32}/n$ times the interarrival rate of backscatter from the monitoring host). The method of course depends on address uniformity, reliable delivery of attack traffic, and the assumption that unsolicited packets at the monitoring site actually represent backscatter.

References

[1] S. Bellovin. *Defending Against Sequence Number Attacks.* Internet Engineering Task Force, May 1996. RFC 1948.

[2] Randy Bush et al. Protecting the bgp routes to top level dns servers daniel massey. In *NANOG25*, Toronto, Canada, June 2002.

[3] Statistical weaknesses in tcp/ip initial sequence numbers, May 2001. `http://www.cert.org/advisories/CA-2001-09.html`.

[4] Drew Dean and Adam Stubblefield. Using client puzzles to protect tls. In *10th Usenix Security Symposium*, Washington, D.C., August 2001.

[5] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. Dos and don'ts of client authentication on the web. In *10th Usenix Security Symposium*, Washington, D.C., August 2001.

[6] A. Heffernan. *Protection of BGP Sessions via the TCP MD5 Signature Option.* Internet Engineering Task Force, August 1998. RFC 2385.

[7] A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *NDSS*, 1999.

[8] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding BGP misconfiguration. In *SIGCOMM2002*, August 2002.

[9] David Moore, Geoffrey M. Voelker, and Stefan Savage. Inferring internet denial-of-service activity. In *10th Usenix Security Symposium*, Washington, D.C., August 2001.

[10] Y. Rekhter et al. *Address Allocation for Private Internets.* Internet Engineering Task Force, February 1996. RFC 1918.

[11] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer. Single-packet ip traceback. *IEEE/ACM Transactions on Networking (ToN) (to appear)*, 10(6), December 2002.

