

# ATTACK CLASS: ADDRESS SPOOFING

*L. Todd Heberlein*

Net Squared  
4324 Vista Way  
Davis, CA 95616

*Matt Bishop*

Department of Computer Science  
University of California  
Davis, CA 95616

## Abstract

*We present an analysis of a class of attacks we call address spoofing. Fundamentals of internetwork routing and communication are presented, followed by a discussion of the address spoofing class. The attack class is made concrete with a discussion of a well known incident. We conclude by dispelling several myths of purported security solutions including the security provided by one-time passwords.*

## 1 Introduction

Recently we began analyzing known vulnerabilities and attacks for the purpose of modeling them. We believe a sufficiently complete model will allow us to both predict new instances of general attack classes and build generic schemes for detecting exploitations of general vulnerability classes. This paper discusses one vulnerability/attack class we call address spoofing.

Many of today's network services use host names or addresses for both identification and authentication. A system using such a service composes a message and sends the message to the service on a remote system. The service on the remote system allows or disallows the request solely on the sender's address included in the request. For example, a remote login may be allowed without formal authentication (e.g., no password is required) if that remote login is coming from a "trusted" host. Table 1 describes some of the services using the senders address for authentication. Many higher level network services (e.g., network back-ups) are built on these vulnerable services thereby inheriting or extending their risks.

Unfortunately, addresses were not designed to provide authentication, and an adversary can take advantage of this fact by

forging an artificial request. This paper explores how, why, and under what conditions an adversary can exploit services using address-based authentication. Following a discussion of the problem in the most general sense, we present a specific example of such an attack. Finally, we will conclude by answering some of the questions surrounding this problem.

## 2 Background Fundamentals

In order to more fully understand why and how address spoofing can be performed, we first cover some of the basics of communication and routing. These basic properties will be used to characterize an adversary's capabilities and strategies.

### 2.1 Connectionless vs. Connection-oriented Communication

As mentioned in the previous section, an adversary exploits the services of interest by forging a message; however, before we can define what a "message" is, we must examine some of the fundamentals of network communication.

Communication across a network falls into two broad categories: connectionless and connection-oriented communication. In connectionless communication, typically

<u>SERVICE</u>	<u>EXPLANATION</u>
<b>r* commands</b>	remote login, remote shell, remote copy, etc.; host address can provide authentication by .rhosts and hosts.equiv files.
<b>mountd</b>	file system mounting; host address is used to allow access and access rights. Host access is usually specified in a file called something like /etc/exports.
<b>TCP/UDP wrappers</b>	wrappers around network services; wrappers are often used to deny access except to a few hosts to network services. IP access/restriction can be set in specific configuration files.
<b>firewalls</b>	IP firewalls are used to restrict access into a network to certain services and certain IP addresses. IP access/restrictions can be set in configuration files.

**Table 1**

supplied by a protocol layer such as UDP, no state information about previously exchanged information is kept. If a process wants to send a message to another process which is already waiting, the first process simply constructs the message and gives it to the connectionless protocol layer (e.g., UDP) to deliver. Because no state information is kept, the underlying protocol being used does not guarantee that messages will arrive at their destination or even if the messages will arrive in the order that they were sent. However, this lack of state also makes connectionless protocols such as UDP very efficient and therefore desirable for many network services.

Processes requiring more robust communication, at the cost of some efficiency, use connection-oriented communication; the TCP layer provides such services. Connection-oriented communication "guarantees" that information will both arrive and arrive in order at the destination process, or if delivery could not be made, at least the sending process will be notified. Connection-oriented communication goes through three phases: connection set-up, data exchange, and connection tear-down. Under TCP, the set-up and tear-down process are performed by three way handshakes; the set-up handshake is described below.

The connection set-up is a three way handshake during which each host tells the other its beginning sequence number and acknowledges the beginning sequence number of the other host (see Fig. 1). The connection is NOT considered established until both hosts have acknowledged the other host's sequence number. Once the connection is established, the sequence numbers will be used to guarantee in-order delivery of data. In the first packet exchange in figure 1, Host A (Alice) notifies Host B (Bob) that she wants to establish a connection and provides her starting sequence number X. In the second packet exchange, Bob sends his starting sequence number, Y, and acknowledges that he has received Alice's starting number (it is incremented by one). In the final exchange, Alice acknowledges that she has received Bob's starting sequence number (once again, incrementing Y by one). At this point, the connection is established and data can be exchanged.

An important feature to note is that Bob's sequence number, Y, must be used in the third part of the handshake - Alice's second packet. If Alice is not able to demonstrate to Bob that she knows his

sequence number, Bob will terminate the connection before it is fully established.

## 2.2 Routing

Routing, under the internet protocol suite, is almost magical. A host wanting to send a packet to a remote host somewhere else on the internetwork need only place the packet on the network, and the packet will be automatically routed through the network until it reaches its destination. Neither the sending nor receiving host need to know the underlying architecture of the internetwork (hence, we often refer to an internetwork as a cloud). What is even more interesting for our needs is that, for the most part, during a packet's travels across the internetwork, only the destination address of the packet is examined. Therefore, the source address can be anything, including a non-existent host, and the internetwork will still deliver the message.

In Figure 2, our adversary E (Eve) wants to send a message to B (Bob) pretending to be A (Alice). Fortunately for Eve, she only needs to construct the packet and place it on the internet. The cloud will properly route the packet to Bob, and he will be unable to tell that it was not Alice who sent it. Once again, this feature will be important as we describe the potential attacks.

## Connection Set-up

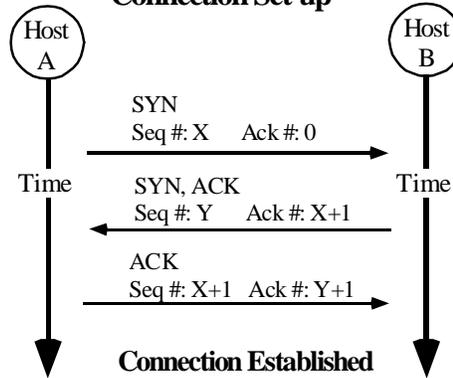


Figure 1

## 3 The Attack

We are now prepared to present the address spoofing attack class. In this section we will explain exactly what we consider is an attack, explain the restriction in the attack, and provide the strategy for an adversary.

### 3.1 Definition

Our model includes three players, Host A (Alice), Host B (Bob), and the adversary, Host E (Eve). Bob explicitly grants Alice special privileges. This granting of privileges is performed by listing Alice's name (or address) in special configuration files (e.g., .rhosts). Thus, Alice is able to get Bob to perform certain actions, actions he will not perform for just anybody, simply because she is who she says she is. Eve's goal is the following: **To get Bob to perform a specific action that he would perform for Alice but not Eve.**

### 3.2 Restrictions

We must concern ourselves with two major issues: (1) the placements of Alice, Bob, and Eve and (2) the nature of the communication used to get Bob to perform the desired actions.

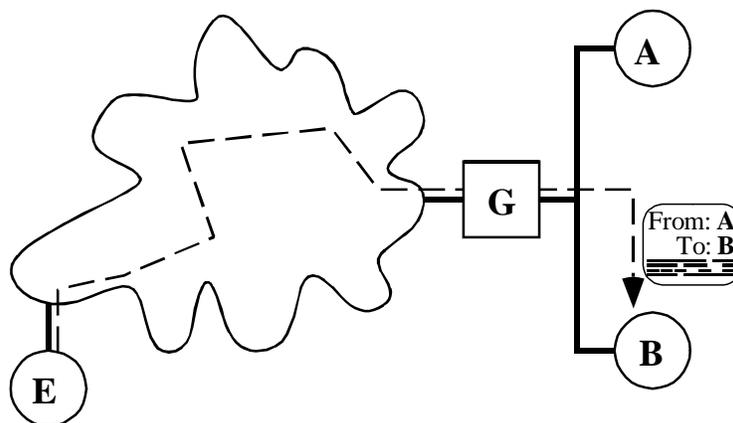


Figure 2

### 3.2.1 Architecture

The placement of the three players can be described as the model's architecture. The most basic architecture has Alice and Bob on the same network

as in figure 2. In this scenario, either Eve is also on the same network or she is outside the network. However, for the purpose of this presentation we will examine the more general architecture where Alice and Bob are on separate networks. In this scenario, Eve's location relative to Alice and Bob can be described by one of the following four categories: (1) on the same network as Bob, (2) somewhere on the path between Alice and Bob, (3) on the same network as Alice, or (4) not on either of Alice or Bob's network and not in the path of the data (see figure 3). Each of Eve's four positions will dictate different strategies used by Eve and different defensive/detection strategies used by Alice or Bob.

Please note that the simpler architecture, where Alice and Bob are on the same network, is really a special case of our more general architecture depicted in figure 3. Namely, E<sub>1</sub> and E<sub>3</sub> collapse into one case, E<sub>4</sub> remains as is, and E<sub>2</sub> is eliminated.

### 3.2.2 Communication Nature

Here we are concerned with how Alice and Bob normally communicate. For if Eve is to get Bob to perform some action by making him believe Alice is requesting it, Eve's communication with Bob must be indistinguishable from Alice's communication with Bob (at least from Bob's perspective). We divide communication into two broad categories we call orders and dialogues. In order communication, Alice sends a single message to Bob. Bob may reply, but we assume he has already carried out the order

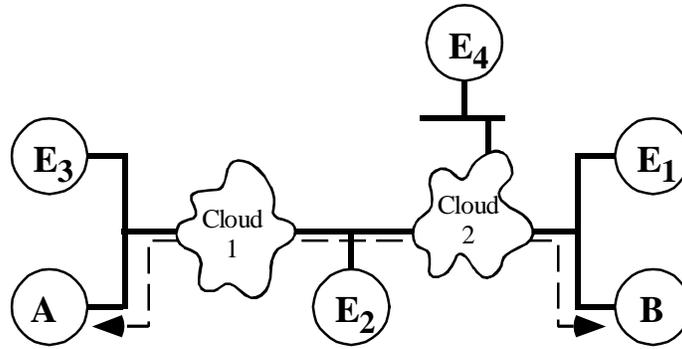


Figure 3

before replying. A popular form of order communication is the remote procedure call (RPC) over UDP.

In dialogue communication, Alice and Bob exchange several messages prior to

Bob carrying out any request. If the dialogue does not make sense from Bob's perspective, Bob will not carry out the requested action (indeed, Bob may stop the dialogue before he even receives the request). Any communication over TCP must be considered a dialogue because as we showed earlier, several messages (packets) must be exchanged to set up a TCP/IP connection. Furthermore, Bob will be replying to Alice (not to Eve, who is pretending to be Alice). If Alice receives Bob's replies, she may tell Bob that she isn't talking to him, at which point Bob will terminate the dialogue. Eve may need to keep the dialogue going for some time, so she will need to prevent Alice from alerting Bob.

The nature of the communication, order or dialogue, used to get Bob to perform the desired action will dictate Eve's strategy.

### 3.3 Strategy

For Eve to complete her goal, she must achieve two main subgoals: establish a forged communication with Bob and prevent Alice from alerting Bob until it is too late. We examine each of these goals and their challenges in the following section.

For Eve to transmit a forged packet to Bob, she must simply construct the packet and place it on the network. The routing software in the network will deliver the packet for Eve. If the communication is order-based in which only a single packet is needed (e.g., a remote procedure call over UDP), then Eve has completed her communication subgoal. However, if communication is dialogue-based, Eve will need to send multiple packets to Bob,

the contents of which will depend on replies that Bob makes (e.g., Bob's sequence number under TCP). If Eve is in positions E<sub>1</sub>, E<sub>2</sub>, or E<sub>3</sub>, she is able to observe Bob's responses thereby allowing her to send meaningful subsequent packets to Bob. If Eve is in position E<sub>4</sub>, she can still observe Bob's responses if she is able to modify the reply path from Bob to Alice. This can easily be done through source routing in IP networks. Modifying router settings are also an option to Eve. Finally, even if Eve is in position E<sub>4</sub> and is unable to direct Bob's traffic to Alice through Eve's own network, if Eve can predict Bob's responses (e.g., what Bob's sequence number will be), she can still carry on the communication with Bob. Predicting sequence numbers is discussed in [Morris 85] and [Bellare 89] and was used in the recently IP spoofing attack.

Eve's second major goal is to prevent Alice from interfering with the attack. Eve can achieve this goal in many ways; we will discuss three: (1) prevent the packets from reaching Alice (or Alice's packets from reaching Bob), (2) take away Alice's ability to respond, and (3) complete the communication before Alice's alerts can reach Bob. The first approach requires Eve to modify the routing behavior of the network. If Eve is a node in the routing path (e.g., she is a router or has used source routing to make the route flow through her), she simply doesn't forward the packets to Alice. Even if Eve is not on the path between Alice and Bob, she could modify the routing information in one of the routers in the path to misroute Alice's packets. Eve could also wait for the internetwork between Alice and Bob to fail and launch her attack then.

The second approach, taking away Alice's ability to respond, can be much simpler for Eve to implement. Eve can (1) cause Alice to crash (not terribly difficult), (2) wait for Alice to go down for other reasons (e.g., maintenance), or (3) block the TCP/IP portion of Alice's operating system so that it cannot process Bob's packets. This latter approach, perhaps the most graceful, was used in the recently publicized IP spoofing attack and originally detailed in [Morris 85].

The third approach, completing the communication (at which time Bob has completed the action) before Alice can alert Bob, is trivial in the order-based communication (e.g., RPC). Bob will have completed any operation prior to sending any messages to Alice; therefore, by the time Alice is aware that something is wrong, she is too late. For dialogue-based communication, the solution is more difficult, because Bob will be sending data to Alice before Bob completes the requested operation. However, if the communication between Eve and Bob is much faster than between Alice and Bob, Eve could complete the attack in time.

### **3.4 Attack Summary**

For Eve to achieve her goal of getting Bob to perform an action for Eve when he thinks he is doing it for Alice, Eve must (1) get the forged message to Bob, (2) if necessary carry on a dialogue with Bob, and (3) prevent Alice from interfering with the communication. Internetwork routing will usually take care of the first subgoal for Eve. The last two goals may be achieved in a number of ways; our suggested approaches were by no means complete.

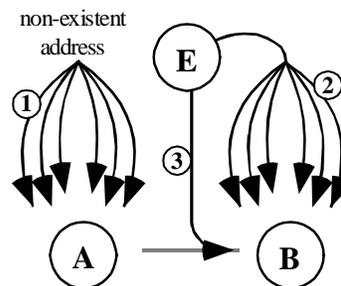
## **4 An Example Attack**

Having mapped out a general plan for Eve to exploit access control which is based on IP addresses or names, we now examine a particular instance of such an attack. The attack, launched at the end of 1994 against Tsutomu Shimamura's machine, has gained the attention of the popular press, the usenet, and CERT. The attack can be mapped directly onto the our general model. Furthermore, the attack was explicitly described in [Morris 85].

This particular attack involved a server (Alice) and an X-client (Bob) (see figure 4). Eve was in position E<sub>4</sub>. That is, she was unable to observe the messages passing between Alice and Bob.

**Step 1:** "Wedge" a portion of Alice's OS such that it cannot process Bob's replies. This is performed by sending multiple connection requests to the rlogin port (port

<b>Players</b>	<b>E</b>	adversary
	<b>A</b>	server
	<b>B</b>	X-client
<b>Steps</b>	<b>1</b>	Prevent Alice From Responding
	<b>2</b>	Probe for sequence number prediction
	<b>3</b>	Forge communication



**Figure 4**

513) from a non-existent host. Alice responds to each request (the second part of the TCP handshake), but since the originating host does not exist, the third part of the handshake never comes. Alice is left with several partially opened connections, each filling up space in her internal data structures. Alice is only able to support up to eight of these partial connections before internal tables fill up and she stops responding to new packets to port 513.

Please note that had Eve listed her own IP address in the forged, artificial requests, her own TCP/IP software would have sent a reset command, RST, following Alice's reply. The RST would have freed Alice's data structures. Therefore, Eve had to use an artificial address as the sender of the requests—one that would never reply to Alice's responses.

**Step 2:** Predict what Bob's sequence number will be. Eve sent 20 connection requests to Bob's remote shell server; the starting sequence number for each connection request incremented by a predictable value of 128,000. Eve most likely used an address of a legitimate host for the connection requests. Thus, when Bob replies with the second part of the handshake, the OS, which did not actually make the initial request, generates a reset message (RST). This prevents Bob's operating system from wedging in the same way Alice's did in Step 1. The address of the forged requests may either be Eve's own or the address of another host. If the address is that of another host, Eve must be able to observe the path between Bob and the other host (in order to observe the sequence numbers).

**Step 3:** Have a dialogue with Bob pretending to be Alice (which is still in a confused state). In this particular case, the dialogue was a TCP/IP connection to the remote shell server. Although Eve could not see Bob's replies, she accurately predicted that Bob's starting sequence number would be 2,024,384,000—exactly 128,000 more than the last requested shell connection in step 2. Eve's requested action: place a "+ +" in the `/.rhosts` file (a shell command such as `echo '+ +' >> /.rhosts` was sent). Bob, believing the connection was from Alice, carried out the request.

At this stage, the goal we set forth for Eve has been completed. She was able to get Bob (the X-client) to perform an action (place a "+ +" in `/.rhosts`) for her; something only Alice could legitimately do. Following this attack, the adversary easily logged into Bob via `rlogin`. In fact, at this point anyone from anywhere could `rlogin` to Bob.

## 5 Popular questions

**Couldn't this attack be simply stopped by configuring routers (or firewalls) to not forward an obviously forged packets?** This is true in limited circumstances. For example, if the gateway G in figure 2 did not forward the packet which already states that it is from A onto A's network, then the forgery in figure 2 could not take place. However, this is only a partial solution. If hosts A and B (Alice and Bob) are not on the same network already, this approach cannot work. Furthermore, even if Alice and Bob are on the same network, this cannot prevent attacks coming from Eve if she

is on the same network already. In short, this solution is limited in scope. A solution should not be dependent on the network architecture.

In general, we define the Point of Convergence as the point where the path from Alice to Bob and Eve to Bob become the same path (see Figure 5). A monitor or gateway placed between the Point of Convergence and Bob would not be able to detect or stop masquerading traffic from Eve.

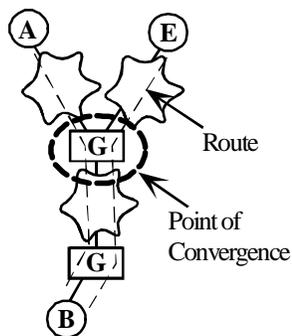


Figure 5

**Couldn't we require all "trusted" hosts to belong to the same physical network (that is, no traffic passes through a gateway) and use lower level addresses such as the ethernet address?** No. While it is widely believed that the ethernet address is a property of a host's ethernet hardware and is therefore unchangeable, we have demonstrated the creation of packets which include a forged ethernet source address. These packets are indistinguishable from legitimate packets. We have reason to believe that this is true with other network media as well.

**Couldn't we simply write a more secure algorithm for choosing initial sequence numbers?** If by "secure" you mean a less predictable starting sequence number, the answer is again true, but only in limited circumstances. This would work if, as in the case in figure 3, the adversary Eve is in position E<sub>4</sub> and unable to alter routing information to get the traffic to flow through her. However, if Eve is in positions E<sub>1</sub>, E<sub>2</sub>, or E<sub>3</sub> or if Eve is in position E<sub>4</sub> and can use source routing or other means to alter routing, a more random initial sequence number would

still not work. Eve is still able to observe Bob's sequence number.

**What other extensions to this attack might exist?** While numerous possibilities exist, perhaps the one which concerns us the most is the placing of a forged request into an already existing TCP/IP connection. This approach is generally referred to as session hijacking, and existing programs, including commercial ones, already have this capability. In such an attack, even password protected services are vulnerable since every packet following a correct sign-on is trusted. Indeed, we have even demonstrated this attack against systems with one-time passwords.

## 6 Summary

We have described a class of attacks we call address spoofing. The reason this class exists rests squarely on the fact that systems and application developers have chosen to use a property which was not designed to provide security, namely the sender's network address, as a means of authentication. We have outlined where and how this vulnerability can be exploited, and we described a real instance of such an attack. Finally, we hope to have convinced you that the solution is not with "fixing" parts of the protocols (addresses and sequence numbers) which are not broken, but with getting systems and application developers to build their security on properties developed with the purpose of providing security in the first place.

## References

- [Bellovin 89] Bellovin, S., "Security Problems in the TCP/IP Protocol Suite," Computer Communication Review, Vol. 19, No. 2, pp. 32-48, April 1989.
- [Morris 85] Morris, R.T., "A Weakness in the 4.2BSD Unix TCP/IP Software," Computing Science Technical Report No. 117, AT&T Bell Laboratories, Murray Hill, New Jersey.