# Global Information Assurance Certification Paper

# GIAC-GCIH – Practical Assignment – Version 3

Table of Contents



## 1) STATEMENT OF PURPOSE

In this paper I will show how a company's private network was hacked, with an attack that resulted in a Denial-of-Service situation.
HACME StoX Markets Inc. has installed an electronic trading system and has deployed a 802.11b WLAN to provide brokers with a direct link allowing them to buy and sell stocks remotely from their offices, given brokers are physically distributed within a radius of no more than 3 miles from HACME headquarters.
At the same time, HACME has deployed an electronic data-providing system to provide data vendors and other organizations that buy and sell electronic information from markets to investors with a direct link, allowing them to access the information remotely from their offices. Taking advantage from the fact that all these data vendors are in the same zone as HACME and the brokers, it has been decided to provide them too with wireless access to HACME systems.

For security purposes, HACME implemented encryption on the links, the one provided with the IEEE 802.11b protocol implementation, based on the RC4 algorithm. Knowing about the weakness of the WEP implementation, the people at HACME decided to implement a VPN over the already encrypted links, in order to provide more strength to the solution, specifically for the case of brokers, given live trading information is considered to be of utmost importance and its C. I. & A. must be assured. HACME handled this as a requirement for brokers to be able to trade over the system, and enforced it by providing them with already set "user-unserviceable" desktop computers including everything needed to run as requested.
But with data vendors the situation is different, information provided to them is not exactly real-time information, it has a 5-minute delay so it wasn't considered that it required so much protection, and then no VPN was deployed for the data vendors access.

For this service, HACME implemented a TCP/IP client-server mechanism in which a server process runs at a certain HACME known address and listens at a certain known port, waiting for clients to request for information.
These clients are processes that run at the data vendors sites, on systems with addresses known to HACME and they start their communications from any ephemeral port. Given this service uses no VPN, HACME as an additional security and management control, decided to implement a list of authorized clients to access the data-providing server, based on IP addresses and not allowing others to access it.

In short, the server running at HACME only receives requests from certain known IP addresses belonging to authorized data vendors who have an agreement with HACME. Requests coming from other addresses will not be honored, and these communication attempts will be dropped by the HACME firewall.

The purpose of this paper is to show how an attacker managed to exploit this situation, by dropping data vendor connections whenever s/he wanted, greatly interfering with the service providing information about the trading operations taking place in HACME electronic floor.
It will be also shown that the attacker doesn't need to be an external one, s/he could be another data vendor or someone working for a data vendor interfering with her/his colleagues businesses in her/his own benefit.
There are basically two scenarios that could lead to such a situation:

1. A data vendor who wants to interfere with other colleagues, preventing them from receiving HACME information properly; this could be just to stop others from having the information or at least delaying them in getting it and benefitting from being the first in having it, which could be a good marketing advantage. Another option for the same case is a disgruntled data vendor, who has been banned from receiving HACME information, let's say, for lack of payment, or for some agreement violation, and still has the wireless link, but has been dropped from the access list allowed for the service.
2. Somebody outside of the wireless network, not necessarily a data vendor, who breaks into the wireless security and manages to drop vendors communications. This could be someone hired by some vendor to make this job for the same reasons exposed in point 1, or just someone who wants to attack HACME or one or more data vendors.

With this objective, in this paper I'll do my best effort to demonstrate the capabilities of the TCP Reset attack, to explain what this attack is and why it works, and to describe the steps that must be followed to exploit this particular vulnerability existing in the TCP protocol; also, I'll tell how the attacker in this story proceeded to achieve his/her goals, and finally, I'll describe the incident handling process that took place once the situation was detected and the technical people at HACME started to work in solving it, as well as the lessons that should have been learnt as an aftermath.

The security breach that will be explained in first place, the wireless problem, by itself leads to a loss of Confidentiality, given data intended to be accessible only by the intervening parties suddenly gets exposed to whoever breaks the encrytion schema. Of course, this could lead too to lose Integrity, given the attacker could somehow manage to alter the data being transmitted, mounting different options of other attacks, for example of the kind of "Man-in-the-Middle", where transmissions from point A to point B are intercepted by an intruder sitting somewhere in between the two points, who somehow alters the information flow, making point B receive something not exacly equal to what was sent by point A.
Finally, Availability could be compromised, as we'll see in the rest of this work. An attacker could manage to interfere with communications, making them temporarily or permanently unavailable to the parties involved. This is known as a "Denial-of-Service" attack, or DoS.
Although a DoS attack can be implemented in many different ways, the concept behind the expression is always the same: depriving someone from accessing to something. The extent can be limited to a reduced group of people or it can reach a vast number of affected clients or users of the service, depending on this one and on the characteristics of the attack. For an everyday example consider the telephone service. Cutting the phone line at a house entrance is a DoS for the house dwellers, while cutting a trunk somewhere in the street could leave many blocks with no service, and blowing a central switch could render a whole town with no phoning capability. These are all different kinds of DoS attacks over the same system.

The second exploit that will be studied in this work leads to a DoS situation, with an impact limited only to the involved parties. The TCP Reset attack doesn't affect the server point nor the client, both of them keep on running, it's just THE specific data link that gets dropped, and in most of the cases it can be restablished, except in those situations where the server side accepts just a limited number of reconnections from a certain client.
Observe that this DoS attack has a limited span, given it doesn't lead to a situation where all communications are dropped, but only those specifically attacked; following with the phone service example, it's just as if, given a conversation between numbers X and Y, someone could tell the phone central to act as if X has hanged up, while really X is still talking to Y; this will drop the communication, and X will have to redial Y number to restablish it back again. So, it's not a central switch bombing situation.

## 2) THE EXPLOITS

Two different vulnerabilities must be described in order to completely explain this situation and why this attack was successful.

The one about the wireless technology involved could be skipped given the particular characteristics of this case, but it will be explained anyway just to provide a more general case and not to circumscribe the problem to such an specific situation, which will be explained later.

## 2.1) The wireless problem

The 802.11b is a protocol for wireless communications that bases its security in the WEP (or Wireless Equivalent Privacy) protocol for encrypting the traffic, a basic set of instructions and rules which provides a certain level of privacy.
The WEP protocol uses the RC4 (Rivest Code 4) algorithm to encrypt all the packets that are sent out from a transmitting device and to decrypt them at the receiving side. But it's flawed, and this is what this exploit is about.
As a matter of fact, the RC4 is a secure algorithm in itself and it's just the poor WEP implementation that makes it faulty. By the way, what we all know as the RC4 algorithm is something assumed to be the original algorithm, which is a trade secret of RSA Data Security. In 1994 there was an apparent "leak" in a newsgroup and so the algorithm was made public. Nobody can assure that what we know as the RC4 algorithm is the original one, and RSA never recognized the "leaked" algorithm was it, but they're functionally equivalent. If the original RSA RC4 algorithm is to be used, then a license from RSA is needed, while nothing is needed if the public-domain algorithm known as RC4 is used; this one is often known as ARC4 or "Assumed RC4".
The RC4 algorithm it's a symmetric stream cipher. Although it's beyond the scope of this paper, let's briefly explain what this means.

When thinking about protecting data, both in transit and stored, the two main technologies are symmetric and asymmetric key cryptography. In symmetric key cryptography, also known as secret key crypto, both parties, sender and receiver (in case of messages) must share a common key, preferably only known by them. This common key is used both to encrypt the original plaintext before being sent by the sender as well as to decrypt the final crypto-message once received by the receiver. That's why it's called symmetric, the same key is needed on both sides.
In asymmetric key cryptography, or public key crypto, each one of both parties, has an exclusive pair of keys, a public one and a private one. This is based on complex and strong mathematical concepts and properties, but the idea is that what's encrypted with one of them can only be decrypted with the other one. This way, each party only needs to know the public key of the other side, and public keys are what their name says, public, known to everybody, so there's no need to protect them. On the contrary, the idea is to publish them so that everybody can encrypt messages with them that can only be decrypted by the legitimate receiver, who owns the private key.
There are many encrypting techniques or ciphers, but all of them fall into one of two great groups: block and stream ciphers. The main difference between them is how they treat the original plaintext message; there's a great analogy with storage devices, where we find some work with blocks of bytes while others work on a one-byte-at-a-time basis. Block ciphers take chunks or pieces of the original message of a certain predefined size and apply the encryption algorithm with the chosen key to each block, while stream ciphers operate on a byte-based or bit-based way, this is, a transformation is applied to each character (or even each bit) in the message, one by one.

The RC4 algorithm is a stream cipher based on a 3-byte initialization vector (IV for short) and a 5-byte or 13-byte password, that generates pseudo-random values which are used in the WEP implementation to encrypt a plaintext

message by combining them with the original bytes in the message through an XOR binary operation (the simple idea of the XOR is that it yields 0 if both operands are the same, and 1 if they're different).
The purpose of the IV is implementing the concept of "*state*" in both the encrypting and decrypting processes. This state can be represented by a value held for example in a vector which is cyclically walked-through from beginning to end during each encrypting loop, and it's changed for every group of bytes or block that's transmitted. The purpose of this is to run the algorithm for each transmitted packet or block with a different IV, to provide more security. But this implies that two things need to be known by the communicating parties: the pre-shared password (the secret key of the symmetric cipher) and the IV for each block. And how is the IV shared? It's transmitted in cleartext with each block, which is not secure at all.

Now, not only the IV is transmitted in the clear, but also there's a limited number of possible IVs, given it's implemented in 3 bytes, each of them with 256 possible values, which yields a total of 16.777.216 possible 3-tuples for the IV. This may seem to be a large number but in fact by its very random nature, collisions, or repeated IVs, can be expected to appear after about 5.000 times the IV changes. It must be considered that the IV changes with every transmitted packet, so depending on the traffic on the network, it would take more or less time to obtain a repeated IV. When a repeated IV appears, it means the same password has been used to produce the cyphertext being transmitted. So, in case of knowing the plaintext which produced one of the obtained cyphertext sequences, it could be possible to obtain the other plaintext, and what is worse, the password used to encrypt them, which would lead to a total exposure of the link, given this password is composed by concatenating the 3-byte IV with a secret key. If the password can be obtained for one encryption instance, given the IV is always transmitted in cleartext, the password for every other instance can be trivially deduced.
All this works this way because of the properties of the XOR binary operation, the one used to turn plaintext into cyphertext and because of the small number of bytes chosen for the IV; if instead of 3 bytes a higher number had been considered, chances of repeating an IV would be much smaller than they are, making the guess part of this attack by far more difficult than it is.

Here's the basic binary math for this; given the same IV, the encrypting key, ***K***, is the same. Let's call $P_1$ and $P_2$ to two original plaintext sequences, which after encryption with key ***K*** turned into cyphertext sequences, called $C_1$ and $C_2$.
So,

$$P_1 \text{ xor } K = C_1 \quad \text{and} \quad P_2 \text{ xor } K = C_2$$

Given

$$P_1 \text{ xor } K \text{ xor } P_1 = K = C_1 \text{ xor } P_1 \quad \text{and}$$
$$P_2 \text{ xor } K \text{ xor } P_2 = K = C_2 \text{ xor } P_2$$

Then,

$$C_1 \text{ xor } P_1 = C_2 \text{ xor } P_2$$
$$C_1 \text{ xor } P_1 \text{ xor } C_2 = C_2 \text{ xor } P_2 \text{ xor } C_2$$
$$C_1 \text{ xor } P_1 \text{ xor } C_2 = P_2$$
$$C_1 \text{ xor } P_1 \text{ xor } C_2 \text{ xor } P_1 = P_2 \text{ xor } P_1 = P_1 \text{ xor } P_2$$
$$C_1 \text{ xor } C_2 = P_1 \text{ xor } P_2$$

This way, it can be seen that if one of $P_1$ or $P_2$ is known, the other plaintext can be immediately deduced, given both cyphertexts are also known. The real fact is that hackers have more than an idea of what the original plaintext is at certain communication stages, given they know certain "signatures", or proper byte sequences of standard procedures of well-known operating systems or programs, and even of the encrypting devices. Thus, they can search for instances in the data transmission that can lead them to success.
All the math involved in this guessing task is too tedious and complex to be run by hand, but there are several freely downloadable tools out there that implement mechanisms that mask all this labour and complexity, making the exploit available to simple script-kiddies who don't need to know anything about all these facts, all they need is a computer and a wireless network card running in promiscuous mode.

### 2.2) The TCP problem, the main one

It doesn't have a fancy name, it's just called the "TCP Reset attack", and it's independent of operating systems, applications and services. It just affects any implementation of TCP that complies with the IETF (for Internet Engineering Task Force) RFC-793 original TCP definition of September 1981, as well as those complying with the newer RFC-1323, of 1992.
So, as Alan Paller from SANS Institute (at Bethesda, MD) has said, *"It's a design flaw of TCP, so it's as old as the Internet"*. This is true, but although Denial-of-Service attacks with properly crafted TCP packets were a well-known weakness of the TCP specification, until recently there was the wrong idea that an attack like this was very difficult to implement, or at least with very low propabilities of success (as low as $1/2^{32}$), because, given a 32-bit sequence number is checked for every received packet, it was thought that the attacker had extremely low chances of guessing the right in-sequence number.
But a very important concept had been forgotten or misconsidered, the TCP sliding window, and the loose specification of RFC-793 about how to handle reset packets.
The discoverer of this conceptual mistake is Paul A. Watson, a security expert working for Rockwell Automation (an industry automation company) who clearly described it in his paper "Slipping in the Window: TCP Reset Attacks", shaking both technical and vendors communities with his explanation of the problem.
The MITRE Corporation CVE (Common Vulnerabilities and Exposures) gave this vulnerability the code name CAN-2004-0230, the US-CERT identified it as VU#415294, and the UK-NISCC called it Advisory 236929.

Although the impact this vulnerability has varies from one situation to another, in several circumstances it's critical. It's very simple, and there's not much left for any variants, given the plain basic nature of the attack. It consists in resetting any established TCP communication or conversation, just by injecting a single properly handcrafted packet in the data stream, a RST (reset) packet telling one of the communicating partners to drop the link.
The attack succeeds when the attacker guesses the packet-numbering sequence window that's being used in the specific conversation s/he is trying to attack. This way, when one of the partners receives the spoofed packet with a

valid sequence number, this is, within the sequence window, it accepts it and processes it, in this case dropping the connection with the other valid partner.

Now, why is this somehow exploitable? At first it can be thought that if there's such a vast numbering universe as there is to number the packets (we'll see this later), it would be very difficult for someone to guess at what number the sequence started, and as a second point, it's even more difficult to guess exactly the next number to be used as to make the forged packet to fit unnoticed in sequence. Both things are true, but they're not applicable in this case.
We will see why this attack works, and what it exactly does to take advantage of what is more a characteristic of the TCP protocol than a vulnerability; this is so because the attack doesn't benefit from a programming mistake nor a skipped control; it benefits from what today can be considered a *conceptual* mistake, a poor design. But when TCP was conceived, in the mid-70s, nobody was thinking about attacking communications and nobody worried for having to secure communications from attackers, they were not public domain, so security wasn't a point.
An introductory overview of what TCP is and a simple explanation of how TCP works as well as some related concepts will be given in order to provide the reader with a better explanation of how and why this exploit works.

### 2.2.1) ***A very brief TCP primer***

Usually the term TCP/IP is used to refer to the inner mechanisms of most of nowadays network connections and all of Internet connections. But what is it? TCP/IP is a family of protocols developed in the 70s by the U.S. DoD (Department of Defense) in order to implement what we know today as the Internet. A protocol is a set of rules and conventions previously established between two or more parties, in order to be able to communicate on an orderly and unambiguous manner ("by following the protocol"). These rules describe the format that messages must have and the way they must be exchanged.
In this family, two protocols are the most important and they give name to the group: they're TCP (Transmission Control Protocol) and IP (Internet Protocol).
TCP is a connection-oriented protocol, this is, that it allows establishing a logical connection providing it with flow control and error control between two partners needing to exchange data. This way, it manages to provide reliable links (over an unreliable environment as IP is) and guarantees the delivery of data. This reliability comes from the fact that TCP numbers all and each packet or piece of data it transmits, allowing the receiving end to control if all pieces have been received, and to reassemble them in their original ordering; moreover, the receiver can "tell" the sender if some of the pieces got lost on its way, so that the sender resends it until the receiver gets it and puts it in right place. TCP has many other characteristics, for example, it can control the speed packets are sent and received, this is, if the receiver is not fast enough as to process the packets as it receives them, it can "tell" the sender to slow down, so that it doesn't choke with data, or if the receiver for some reason stops processing the received data, it can tell the sender to wait until operation is resumed.

On the other side, IP is connectionless, and so it's an unreliable protocol that doesn't guarantee the delivery of data. It doesn't control order nor if all packets

are received, so there are no retransmissions and no reassemblies. Of course it's faster than TCP, but it's not enough for communications where order and all packets matter; IP just keeps going, it doesn't wait for success; it's said to be a best-effort protocol, doing as best as possible to do the job, but not caring if something goes wrong.

So, how is that TCP protocol goes over IP protocol? According to the OSI model, TCP runs in layer 4, while IP runs in layer 3. In order to understand this, a brief explanation of models and layers must be given.
Layers, when talking of communication protocols, is a powerful concept that divides the complex communications process into smaller logical groups, easier to understand and manage, as well as to enable interoperability between different technology providers by using industry-standard interfaces, and allowing changes and tuning at a certain level or layer without having to change all the rest of the implementation.
In a layered communications model, data flows from the application at the sender, crossing downward through each layer, from highest to lowest, running along whatever media is used as a physical link (cable, satellite, wireless), and finally crossing upward from the lowest to the highest layer of the model, and finally to the application at the receiver. The idea is that at the sender, each layer receives data from a higher level layer and the protocol operating in it adds its own unique data, based on the information received, in order to contribute to deliver the original piece of information to the destination point. Once at the receiver, each protocol examines and "unwraps" the received packet coming from the immediate lower layer, removing only the data originally attached by its protocol counterpart at the sender, passing all the rest of the received packet to its immediate upper layer for processing.

Two basic layered models are used as a reference to group and describe the complex steps and procedures involved in establishing and maintaining a data link between two partners. One of them was developed in the early 80s by the International Standards Organization (ISO), and was named the Open Systems Interconnection reference model, or OSI. It consists of 7 layers, named, from lowest to highest, *physical* (1), *data link* (2), *network* (3), *transport* (4), *session* (5), *presentation* (6) and *application* (7).
Another layered representation was developed by the U.S. DoD in the 70s, and was named the TCP/IP model; it's similar to the OSI model, except it has 4 layers instead of 7. They're named, from lowest to highest, *network* (1), *internet(work)* (2), *host-to-host* (3) and *application* (4).

This figure tries to give a graphical representation of a data flow going from a sender to a receiver, piercing down through the layers in the sender and up through the same corresponding layers in the receiver:

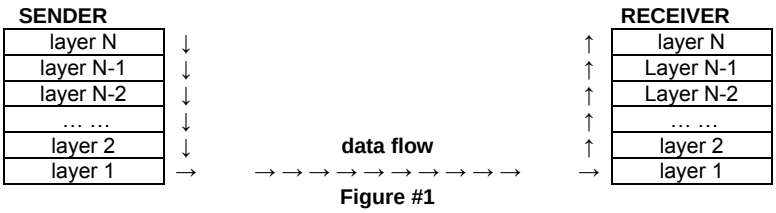



**Figure #1**

Conceptually these two models are similar in that they define layers; one of them divides the communication steps and tasks in more groups than the other, that's the most "visible" conceptual difference. Of course there are many other differences between them, but they're beyond the scope of this work.

So, TCP in the OSI model is considered to be running in layer 4, or *transport* layer, and IP in layer 3, or *network* layer. From the point of view of the TCP/IP model, TCP runs in layer 3 or *host-to-host*, and IP runs in layer 2, or *internet*.

In any case the idea is simple: TCP receives bytes from the application layer (although OSI considers other layers in between, the original data always come from something running at the application layer), groups them into segments and passes them to IP for delivery; when IP time comes, it does its own data chopping, splitting the TCP segments as necessary into IP datagrams, and passes them down to the next lower layer to do its part, until the physical media is reached and data travels between sender and receiver.
The IP protocol has as its main task, to move datagrams or packets from sender to receiver over a networked environment. As many machines can be in the same network, to achieve this goal addresses are used, and they're part of the add-ons of the IP layer; both sender and receiver IP addresses are added to the data coming from TCP, as well as other elements, to conform the new travelling element, the IP datagram.
At the TCP level, a different kind of addressing is needed, because many applications can be running on both sending and receiving machines, so TCP needs to know how to tell one application from another, and then the concept of data ports, or simply, ports, comes to rescue. The idea is simple: each application uses a port number, something like a parking lot or door number, just to differentiate the traffic from one application (such as a web browser) from the traffic of another (such as a telnet emulator). This information is part of the TCP segment, and tells about the port at the sender (source port) and the port at the receiver (destination port).

The next two figures show graphical representations of the structures of TCP segments and IP datagrams (the information for creating these two figures was taken from the "IBM TCP/IP Tutorial and Technical Overview" manual, GC24-3376-03).

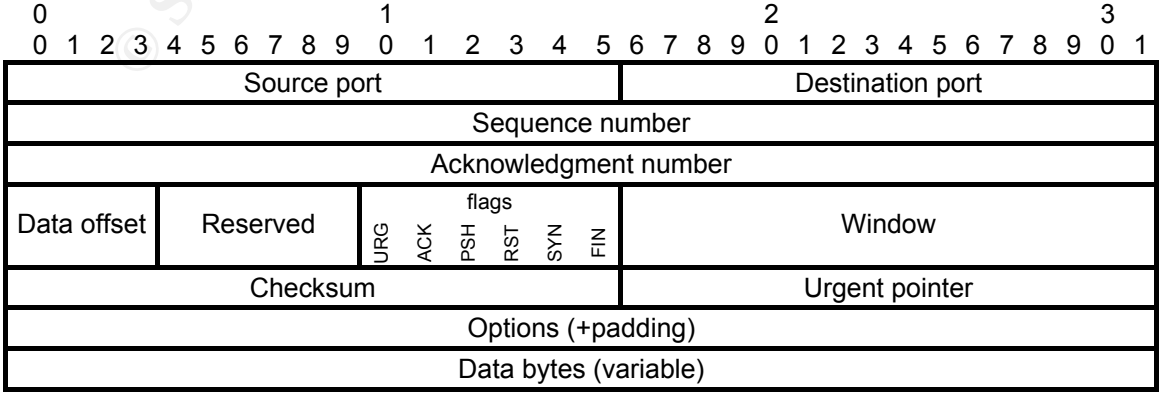



**Figure #2 – the TCP segment**

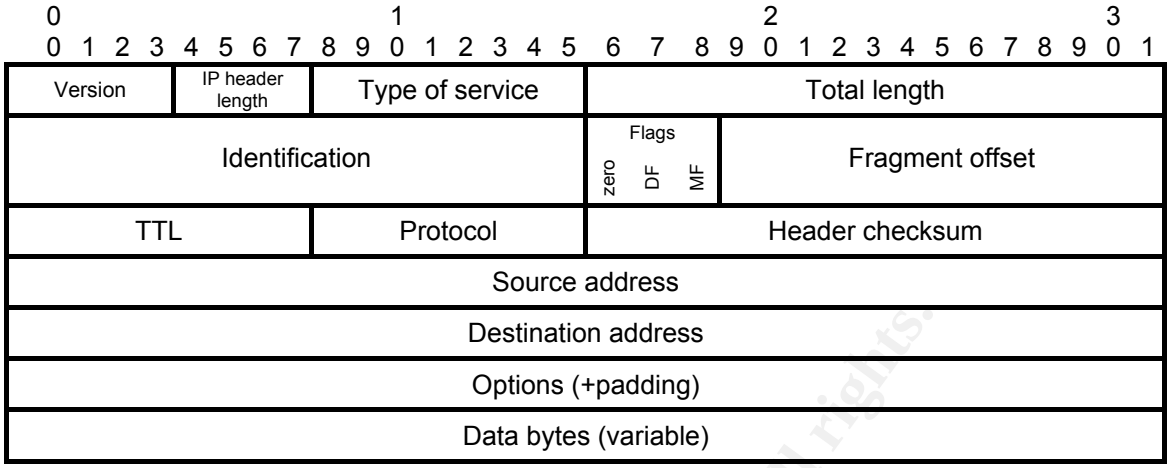



**Figure #3 – the IP datagram**

It can be seen in the TCP segment representation that source and destination ports are represented in 16 bits, so port numbers can go from 0 to 65535 (this is $2^{16}$-1). When the TCP/IP model and protocols were conceived, a group of port numbers were reserved for some applications that are themselves protocols, such as ***telnet*** and ***ftp***, so that they use the same ports in all implementations. These reserved ports are called the "well-known" ports and they range from 0 to 1023. The rest of them (from 1024 to 65535) are not assigned to any particular usage, and can be used by common programs.

From the IP datagram representation it can be seen that the addresses used to identify both sender and receiver, have 32 bits. It's well beyond the scope of this work to explain how these addresses are formed; for our purpose of briefing about IP, let's say these 32 bits are grouped in 4 sub-groups of 8 bits, each of them being able to be represented by a decimal number between 0 and 255, or what's the same, by two hexadecimal digits, from 00 to FF. IP addresses have a "network" part and a "host" part, and depending on how many 8-bit sub-groups are considered in the network part (1,2 or 3) it's said the address belongs to a class (A, B or C) or another.

### 2.2.2) ***TCP and the sliding window concept***

Let's focus a little bit more on TCP, as our exploit bases its action on the basic characteristics of this protocol.

As it was mentioned above, TCP assigns a sequence number to each segment transmitted from the sender. This number is used to acknowledge reception from the receiver in a particular way; here appears the concept of the sliding window.

This is a simple concept: a certain number of segments (the window size) can be transmitted all in a row without having to wait for them to be acknowledged from the receiver, something like a round of segments. The receiver sends back to the sender an acknowledgment (from now on we'll use the TCP naming for it, ACK) referring to the segment following the last it received ***in-sequence***, kind of

a “sofarsogood” signal, and then the sender “slides” its window forward, up to the segment referred in the last ACK; again, the “sofarsogood” concept.

Let’s try to clarify this “window” idea with a simplified example: let’s say the window size is 10 segments; all of the ten first segments are transmitted, but segment #6 doesn’t make its way up to the receiver; then, the last ACK the receiver sends back refers to segment #6, although it has already received segments #7, #8, #9 and #10. Why? Because segment #5 was the last received in the complete sequence, and #6 is the next expected. The sender shifts the window up to the segment #6 reference, and after a certain time if it doesn’t receive the ACK telling segment #6 has arrived, it retransmits it, and then this time the destination receives it, sending back an ACK for #11. Why for #11 if the last properly received segment was #6? Shouldn’t it be referred to #7? No, because the ACK is sent with reference to the segment following the last one received ***in-sequence***, and we said the receiver had already received segments #7, #8, #9 and #10, so #10 is the last in the complete sequence. Once this happens, the sender shifts its window to segment #11 and goes on with this same process.

Actually, TCP implements the window size concept in terms of byte numbers, and not segment numbers as shown in the above example, but the idea is the same.
The fact that the ACK refers to the sequence number of the first byte of the segment expected next by the receiver is called *expectational acknowledgment*.

When a TCP dialog is established, each side defines independently from the other the window size (in bytes) to be used, and it can vary along time; every time one of the partners ACKs a certain segment, it includes information about the window size its working with, this is, how many bytes its ready to receive without choking.

From the point of view of the partner sending information, the window size is how many bytes it can send to the receiver without having to wait for ACKs. Once the sender has sent these many bytes, it has to wait at least for an ACK before it can go on sending more segments. From this it can be deduced that a window size of 1 would force to have each transmitted byte to be ACKnowledged by the receiver, something highly inefficient. So, a larger window size improves performance, because it allows more bytes to be transmitted without having to wait for a confirmation.

The next figure (extracted from the book “Cisco Networking Essentials, Vol. 1, Cisco Press, ISBN 1-587-13004-1) clearly shows what happens when the window size is 1 and when it’s 3, as an example of how performance is affected by this parameter.

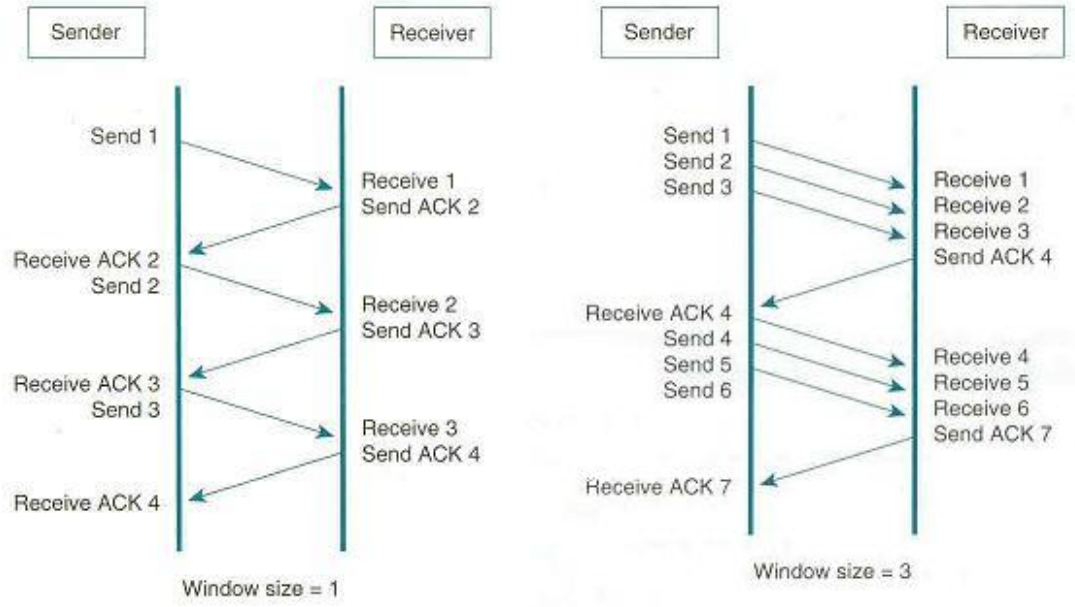



**Figure #4 – differences when window size is 1 and 3**

Of course for practical purposes neither 1 nor 3 are used as real window sizes. In real life we must think of window sizes for example of 16, 32 and up to 64 Kbytes (any other arbitrary values between 1 and 65535 bytes can be used too). The example with 1 and 3 only tries to show the difference in performance that results from using a wider window instead of a narrow one. A wider window provides a better performance, allowing for a faster data transfer.

### 2.2.3) ***The weakness in TCP, or why the RST exploit works***

In the TCP functional specification of Sept. 1981 (RFC-793) is clearly stated that given an established connection, if a packet with the RST or the URG bits set is received, it must be processed immediately even if it's out of sequence, only requiring to verify that the sequence number falls within the window range.

Now, consider these two basic elements:

a) the window size field in the TCP packet has 16 bits, so a maximum size of $2^{16}$-1 (or 65,535) can be specified. This would yield for maximum performance with traditional TCP packets (later we'll see there's an extension described in RFC-1323 which leads to an even worse case for attack purposes).
b) the sequence number field in the TCP packet has 32 bits, so packets can be numbered between 1 and $2^{32}$-1 (or 4,294,967,295).

Then, let's do some simple math: how many adjacent groups of consecutive 65,535 numbers are there between 1 and 4,294,967,295? This is the same as calculating how many times [$2^{16}$-1] fits within [$2^{32}$-1]: [$2^{32}$-1] / [$2^{16}$-1] = [$2^{16}$+1], or 65,537. So, independently of which the starting number is for any packet sequence, it would only take 65,537 guesses in the worst case (for the guesser) to find a number in the same 64k group as the starting number.
In this way, someone trying to drop a TCP link between two partners with a window size of 64k, assuming that s/he knows the IP addresses and TCP ports

involved, would have to try at most with 65,537 spoofed packets, until s/he hits within the established range and drops the connection.
Now some other elements must be considered; to begin with something, the attacker needs to produce the packets and needs to inject them in the carrying media, whatever it is, so it's important what bandwidth the attacker has, in order to determine how long it would take him/her to drop a link.
Another thing is the attacker's packet size: s/he only needs to send a RST message, so the needed TCP segment can be conformed in only 20 bytes, according to Figure #2. These 20 bytes are the payload of the IP datagram to be sent, which in turn can be conformed with another 20 bytes (see Figure #3); so as a result, the attacker needs to transmit 40 bytes (or 320 bits) in each and every try.

Then, depending on the bandwidth the attacker counts with, it would take him/her more or less time to drop a connection. The next table shows some estimations for the worst case for the attacker depending on the available bandwidth, this is, having to send 65,537 40-byte packets (or 2,621,480 bytes) to drop a link.
Numbers in the third column assume and ideal line running at a theoretical transfer speed, something never achieved in real life; but let's be realistic and say we can count on no more than a 60% of the nominal transfer rate; in this case we obtain the numbers shown in the fourth column.

| *bandwidth in bits/sec* | *bandwidth in bytes/sec* | *theoretical time to drop link in secs* | *realistic time to drop link in secs* |
|---|---|---|---|
| **56K** | **7,168** | **366** | **610** |
| **256K** | **32,768** | **80** | **133** |
| **1M** | **131,072** | **20** | **33** |
| **10M** | **1,310,720** | **2** | **3** |

This shows an alarming result; someone from his/her home, with no more than a 56K dialup modem could drop a link (with a 64k TCP window size) between some two partners in the Internet in about ¡10 minutes! Of course this time decreases when bandwidth increases. In case of a 10M link, no more than 3 seconds are necessary to drop such a link. So, exposure is great.
Now, the previous calculations assume something very strong; the attacker knows a lot: both source and destination IP addresses, the destination TCP port number, and the source TCP port number.
It can be considered that the first three elements are easily known, given the attacked link is established between two known-to-the-attacker devices, and the one acting as the server side is listening on a somehow ascertainable port, given it provides a standard service known by the attacker.
Then, the remaining problem for the attacker is the source TCP port number, given it's usually determined at random.
At first, it can be thought that the guessing space is $2^{16}$ possible port numbers, given the port number item has 16 bits in the TCP segment.
But this is not so. To begin, the first 1024 port numbers are the *well-known ports* and are reserved for standard applications, or *well-known services*; they're described in RFC-1700 and they're controlled and assigned by the IANA (Internet Assigned Numbers Authority). Really, numbers below 255 are for public applications and numbers between 255 and 1023 are assigned to

companies for commercial applications (usually on most systems these well-known ports are only used by system processes or by programs run under privileged accounts).
Port numbers 1024 and above, the *high-numbered* ports, are unregulated. But there are some considerations; the IANA classified these ports in two groups:

- from 1024 to 49151: they're called *registered ports*, and the IANA maintains a list of services using ports in this range to minimize conflicting uses. Developers of TCP/UDP services can select a specific number in this range to be registered with IANA.
- from 49152 to 65535: are called *dynamic ports* or *private ports* or *ephemeral ports*. These ports are not managed by any organization and have no usage restrictions. These *ephemeral ports* are temporary ports assigned by TCP implementations every time a client program tries to connect to a server, and they're assigned from the designated range of ports for this purpose. When the connection terminates, the port is available for reuse, although most IP stacks won't reuse that port number until the entire pool of ephemeral ports have been used. This way, if the same client program reconnects or another client program starts a conversation with another server, in any case it will be assigned a different ephemeral port number for the client side of the new connection.

So, according to IANA recommendations, TCP implementations should not assign short-lived ports from the range 1024 to 49151, given these are the registered ports, as well as they should reserve the well-known ports for their standard usage. This reduces enormously the chances a TCP implementation has for assigning these short-lived port numbers, and this is what the ephemeral ports are for, to be assigned for a short-lived conversation, such as those established between a client and a server.
This leaves us with the range from 49152 to 65535 or 16,383 possible port numbers to choose from.
It could be thought this is quite a nice quantity of possible port numbers from where to guess the right one, when trying to spoof a packet into an established conversation. But, through observational analysis, or by searching in providers documentation, it can be known which is the initial port number selection an OS makes, and the step it uses for obtaining the next selections. This way, the guessing task is greatly improved, reducing the time needed to find out the right port number, the one that will make the attacker succeed in spoofing a packet.
To make things even worse for defenders and better for attackers, some TCP implementations differ from the IANA recommedations, and assign the ephemeral ports from other numeric ranges, reducing even more the universe of possibilities, such as in the case of the widely spread Microsoft Windows in its several versions, which assigns these ports from the range 1024 to 5000, a universe of only 3,976 numbers to choose from.
A document written by Mike Gleason, contains good information about the ephemeral ports range assignment in some different TCP implementations.
Moreover, information about this from some OSs development companies can be obtained.

Of particular interest is the fact that none of these two attacks has an easily detectable "signature", something recognizable enough as to identify them by a controlling program or device, such as a firewall or an intrusion detector.
The wireless vulnerability exploit could be prevented implementing additional controls, with software like AiroPeek (www.wildpackets.com) which could raise an alarm in case of detecting a rogue device accessing the network.
The the TCP Reset attack has as its only signature the fact of a sequence number out-of-sequence in the segment. To watch this out is the only possible thing to do in order to avoid this attack.

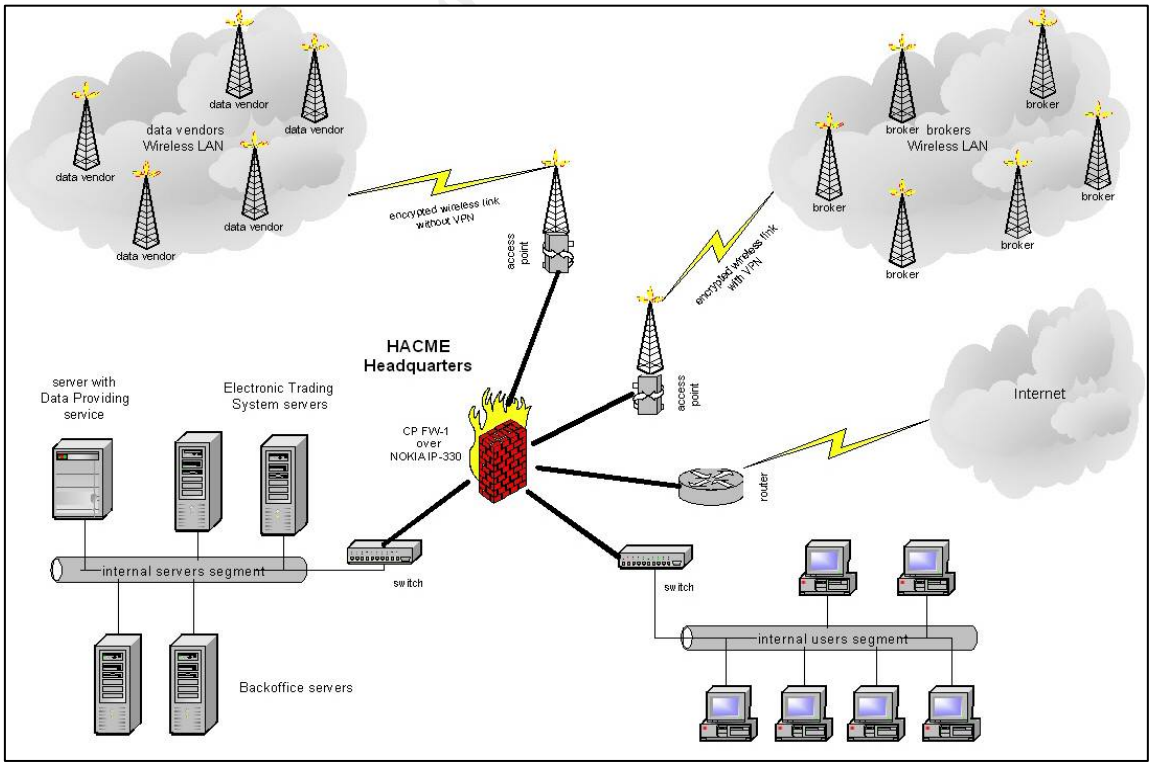
## 3) AFFECTED PLATFORMS and ENVIRONMENTS

According to product vendor information provided by vendors themselves, among the most well-known brands and products affected by the described vulnerability, we can find the following:

- Cisco IOS all versions
- Microsoft Windows all versions
- Linux all versions
- Nokia IPSO all versions
- CheckPoint FireWall-1 all versions prior to R55 HFA-03
- Hewlett-Packard HP-UX all versions
- Cray UNICOS (including mk and mp) systems all versions
- ISS Proventia M Series 1.5

Clearly, the impact it has over the users' community is huge, given the most widely used products are vulnerable. Of course, these are not the only affected brands and products. As it has already been largely explained, any TCP stack that follows the RFC-793 specification and loosely implements the RST bit handling, is exposed. Among these, many Lucent, Juniper, Nortel, Alcatel, Fujitsu and Hitachi products are also affected, just to name a few well-known brands more.
Of course there are some exceptions, such as the Nortel Contivity Family and Nortel BayRS Router Family products, on which the TCP implementation requires an exact sequence number match to reset the TCP connection from its peer, thus frustrating the guessing game on which it's based this exploit.
Some applications that could suffer a higher impact from this kind of attacks are those relying on long lived connections. This is the case of some protocols (H.225 and H.245) used for voice and video signaling, which are established to negotiate parameters for content transfer, and they persist during the whole call, which is terminated if the signaling session is broken.
Another affected application is network storage; in this case, depending on the characteristics of the implementation, the impact could be that users would notice accessing a device slower than usual, or in a worse case, to lose contact with the device.
Some opinions are that any TCP connection during more than one minute is potentially exposed to this kind of attack. On the opposite, many vendors, although being some of their products vulnerable to this exploit, consider that the conditions needed for a successful attack require access levels to the

network not usually available in normal environments. Although this is a valid and respectable point of view, a situation like the one described in this work is not necessarily such a special situation as not to consider it normal, a wired or wireless deployment where members of a group access common resources, and where some of the members might have an interest in denying other members access to the shared resources without getting exposed and without rising suspicions about their acts.
Now, for the effects of the case we're trying to describe here in this paper, the data-providing service is running on a server with SuSE Linux 8.0.
It's listening at port 10000, and it's placed behind a firewall, a CheckPoint FW-1 NG AI FP3 with VPN-1 over a NOKIA IP-330 device, running 5 zones and doing both dynamic and static NAT.
Given the characteristics of the studied vulnerability and exploit, the details of each of the different involved points are quite irrelevant, because any generic environment where the TCP stack implementation allows for a non-strict sequence numbering for RST and URG packets will do. Particularly, the two items involved in the case we're describing, the data-providing server and the firewall, both are running TCP implementations sensitive to this kind of attack.
For purposes of clearly describing the environment, it must be said that the wireless central and remote points are equipped with ORiNOCO RG-1100 Broadband Gateways, from Agere Systems. These pieces of equipment are highly versatile and they provide several functions depending on the chosen firmware configuration. In this case it's being used in the central place as an AP (or Access Point) and in the remote places as an EC (or Ethernet Converter), providing a physical interface between the 802.11b wireless and the wired worlds.
The following diagram describes the network deployment at HACME:



In this deployment there are 5 zones clearly delimited:

- the data vendors wireless LAN, with addresses in the range 10.11.0.x/24
- the brokers wireless LAN, with addresses in the range 10.12.0.x/24
- the internal servers segment, with addresses in the range 192.168.10.x/24
- the internal users segment, with addresses in the range 192.168.1.x/24
- the Internet

In this specific case we'll describe, both source and target networks are the same network, because the attacker is inside the trusted zone, within the security perimeter. In the other possible case, the attacker could be working from a stand-alone rogue system, at a nearby office or even a parked car, and then breaking into the trusted zone.

The server running the data-providing service has the IP address 192.168.10.5, which is NATed by the firewall for the data vendors WLAN as 10.11.0.5, while the firewall port connecting with this WLAN has the IP 10.11.0.1, and it's used by the machines in the WLAN as the default gateway.
At the firewall, there are rules allowing the machines in the data vendors WLAN to access only the corresponding server with traffic for port 10000, and they look something like this:

| source | destination | service | action | audit |
|---|---|---|---|---|
| 10.11.0.x | 10.11.0.5 | 10000 | accept | log |
| 10.11.0.x | 10.11.0.5 | any | drop | log |

Given the firewall processes its rules in a top-down mode, if it receives traffic from the data vendors segment, addressed to the data-providing server at 10.11.0.5 and destination port 10000, it will allow the communication to go ahead, because of the first rule, but if it receives from the same segment an attempt to reach the same server with some other service, such as 23 (telnet) or 80 (http), the firewall will drop that traffic, because of the second rule.

It's been said before that HACME has decided to manage authorizations to access the data-providing server with an access list based on the situation of each data vendor.
For this purpose, a group has been defined with the name ***"allwd_vendors"***, and it's composed of the addresses of those data vendors with clearance to access the information provided by the service. In case a data vendor doesn't pay the monthly fees timely or something, he's removed from this group and he's automatically banned from accessing the server; for this to work, the access rules in the firewall are these:

| source | destination | service | action | audit |
|---|---|---|---|---|
| allwd_vendors | 10.11.0.5 | 10000 | accept | log |
| 10.11.0.x | 10.11.0.5 | any | drop | log |

Now, any traffic coming addressed to the server at 10.11.0.5 would be dropped, no matter the destination port, if the source IP address is not included in the ***"allwd_vendors"*** group.
For NAT to work as intended, the following translation rules were added to the firewall:

| original source | original destination | original service | translated source | translated destination | translated service |
|---|---|---|---|---|---|
| allwd_vendors | 10.11.0.5 | 10000 | original | 192.168.10.5 | original |
| 192.168.10.5 | allwd_vendors | any | 10.11.0.5 | original | original |

This way, HACME people ensured that nobody outside the ***"allwd_vendors"*** group could reach the data-providing server, only allowing traffic to the specific service listening at port 10000, and protecting it from being accessed directly by hiding it behind a static NAT provided by the firewall.

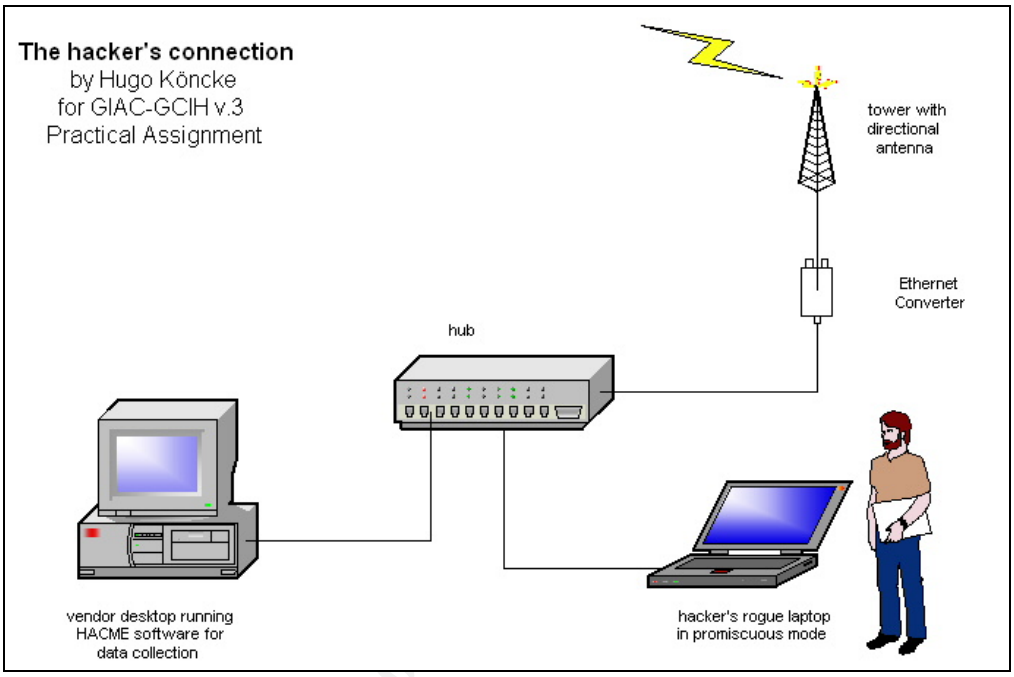
## 4) STAGES OF THE ATTACK

In a very active stocks market with highly volatile instruments, almost-instant information is vital for making a difference, and this may mean big money, earned or lost, depending on when the information is known.
In this case we're describing, the attacker was one data vendor who was trying to take advantage over all the big data vendors, by delaying their data reception and gaining a better position in the data market, by providing his customers with faster information.
Given the objective was clearly predefined in this case, and there was no need for a target selection, this attack had basically two stages, starting with the information gathering phase.
For his purposes, the attacking vendor hired a skillful hacker and explained him his idea, showing him the technical deployment used to access the information, no more than a desktop computer with a NIC connected to the wireless network through an antenna. This computer was provided by the vendor, but the software for collecting data was provided by HACME, as well as its configuration, about IP address (10.11.0.205), mask (255.255.255.0) and default gateway (10.11.0.1).

This information was very valuable for the hacker in this story, it was almost all he needed to know to commit his bad purposes, because what he thought about, was making the HACME server to drop the connections with all other vendors, by sending it faked TCP packets with the RST flag on, as if they were being sent from the other vendors machines. He knew the addressing space for these machines, the only other things he needed were to identify the server address, as well as some specific data vendors addresses.

To find this out, in the reconnaissance stage of the attack, he connected the UTP patchcord coming from the antenna, the one connecting the data-collecting computer and a third one connecting his intruder laptop, all of the three to a simple hub. This way, he was able to sniff all the traffic from his rogue computer, in which he set the NIC into promiscuous mode, this is, with no IP

assigned, not reporting about its presence to the rest of the network, and "reading" all packets arriving to the interface. This worked fine for what he was up to, because the transmissions going from the central point to the remote ones were kind of broadcasted through the omnidirectional antenna at the central point, providing a behavior similar to that of a hubbed network segment, where all NICs connected to the media receive all the traffic. He wasn't able to receive the packets going from the other remote clients to the central point, but anyway he didn't need them, it was enough for him to see just the server half of the conversations to obtain the information he was looking for.




For his rogue sniffing purposes he used Ethereal, a well known useful software crafted to read the contents of packets arriving to a NIC, that can be downloaded for free from the Internet from the web site www.ethereal.com.

This way and after a short analysis, he was able to discover that the HACME server address was 10.11.0.5, and that the port listening to the vendors requests was 10000.

Moreover, and since he was "inside" the WEP-encryped network and traffic had no more encryption than that, everything was in plaintext for him, releasing him from the extra workload of port-guessing for the client side for every connection. These connections were usually kept all day long, so reading traffic for a while every morning provided our hacker with all the information he needed to start the attack. He developed a simple program to read the traffic log from Ethereal in order to extract from it all the addresses, local ports and current sequence numbers of the other vendors.

But he needed to know the specific IPs of certain data vendors, because the idea wasn't to bring them all down but just a few big ones, and a couple or two of smaller ones, just to disguise the move.

Then, one day a young woman with a very charming and delightful voice called on the phone to the communications section of the IT department at HACME, telling she was a communications technician from a company that had been hired by ROUTERS and by many other data vendors, that she was doing a review of the networks at each of the organizations, and that she would be more than thankful if the kind gentleman that was on the line could tell her what the IP addresses of the client machines were for the several data-vendors that had hired her company, so that she could complete her work, which by the way was delayed, and her boss, a very disgusting guy, was about to reprehend her for that. Gary, the HACME guy at the phone, was more than eager to help the cute calling girl and to avoid her having to be observed for not finishing her work in time; immediately he checked in the system documentation and told her all the IP addresses she was needing, and he even asked her not to hesitate in contacting him again in case she needed something else. She thanked him very much, and told him perhaps some day they could meet for lunch or just to have a cup of coffee, an excuse to talk for a while and personally thank him for his kindness.

When they hung up, Gary was walking up in the clouds and the hacker's girlfriend had all the information the hacker needed to finish his attack.
Then he fed all these data to a script that used another program he didn't even need to write it himself, he just downloaded it from the Internet from the site www.iamaphex.net, which allowed him to "fire" rounds of fake RST packets to the HACME server, as if sent from the other vendors' machines (the Delphi code for this piece of logic is shown in the Extras Section of this paper).

This way, the implementation or exploit phase of the attack was quite simple to deploy, its physical needs being just a hub and an additional computer.
Then, every time the attack was to be launched, the program spent a while collecting data about the already established connections, specifically originating ports and sequence numbers, and then produced packets with the IP addresses of the other vendors as origin, the HACME server IP address as destination, the recently found out port number (for each case) as origin, 10000 as destination port, and a sequence number within the sliding window in use.
And a plus for the hacker, he had no access to keep nor tracks to cover; it could be said that this attack is a "hit-and-run" one; once committed, there's no evidence left at all, and it can be repeated as many times as wanted, as long as the attacked target doesn't change something to avoid it.

## 5) THE INCIDENT HANDLING PROCESS

At HACME they had implemented a basic security schema, and they had no incident handling team, IT security was handled just as another more thing supported by the IT department; as ***PREVENTION*** measures they had installed a firewall properly configured watching all the traffic coming to and going from their network. The wireless network was encrypted with WEP and the brokers segment was protected by a IPSec VPN, which was not the case for the data-vendors segment.

All this has already been described in the previous section, as well as the attack process. Of course, this attack cycle could be repeated as many times as requested during the same day, and given the HACME-provided client program running at the vendors sites was programmed to automatically re-establish the connection after waiting for 30 seconds, this ensured that repeating the attack every 2 or 3 minutes during peak activity hours was a very effective way to erode the other vendors' services to their clients.
And all this was happening unnoticed!! Nobody at HACME was watching the packets arriving to the server, as well as nobody was watching the log records of the client program at the other vendors' sites nor at the server itself. Since it all was an automatic process, both the data collection and the reset and reconnections, everything kept running smoothly for a long time.

Then, as time went by, some data vendors started receiving claims from some of their customers, who supposedly had lost business opportunities because they had got delayed information about stock prices and then they late-reacted in consequence. This at first was taken by the vendors just as no more than isolated situations, where perhaps the client's perception wasn't as precise as he alleged.
Moreover, competing vendors don't usually talk to each other about the problems they might be having with their own customers. So, each situation was kept unknown to all the rest, and each vendor tried to cope with it as better as possible, until one day the bomb exploded, on the morning of the last Friday of May, the 28. PARALLEL Funds, a very important client of ROUTERS Inc., one of the biggest data vendors in the game, decided to immediately cancel their contract. The client had already complained several times about losing some good deals for receiving delayed information, until he waited no more. ROUTERS had done nothing when he had told them about this, and this time he had lost big money, very big money. "*To hell with ROUTERS!!*" - the managers at PARALLEL said.
Of course the immediate reaction at ROUTERS was to call HACME and ask them for help, they had lost one of their biggest accounts.
At first HACME people didn't know what to do, because the situation was catching them by surprise. Then, on that same day, May 28, they decided to start an investigation and the security administrator (who wore many other hats as well, depending on the needs) and his staff were involved in it.

Faced with this problem, the following Monday, May 31, they started by collecting a day-long log of the dialog between the server and the ROUTERS client machine, and then when they analysed it they found something very funny: the connection had been restarted almost 50 times during that day !!
The client software running at the vendors' sites was programmed so that in case of an interruption in the communication, it waited 30 seconds before trying again to contact the server, so 50 restarts meant 50 times in which information was received by ROUTERS with a 30-second delay.
Now, how could that be possible?
At first, the initial idea was there was a defective connection somewhere in the network path between ROUTERS offices and HACME headquarters. Just to decide where to begin from, HACME security people decided to collect the activity logs of the dialogs between the server and all the vendors' client

machines, and then two more days went by. Then they saw almost all of the clients were restarting the connection many times a day, and the conclusion was immediate, there was no doubt about it: the problem was in some piece of equipment at HACME side, making the clients lose the connection at random. Or it could be some kind of interference, after all it was wireless.
As it was supposed to be a single point that was failing, on June 3 (Thursday) they switched to the backup cable line between the firewall and switch at the roof (the one that concentrated the APs serving the connections from the different antennas linking to the vendors). They tested the situation again but nothing changed, vendors' machines kept on restarting connections several times a day. Then, the next step was the switch itself; they replaced it on June 4 (Friday) early in the morning; they tested again along that day, and nothing new happened. The last chance was the NIC at the firewall port where the link was connected to, but for this, all the system had to go down. They managed to coordinate the replacement for that weekend (June 5-6), then turned all the network down and changed the suspected offending NIC. The following Monday, June 7, the activity log was closely watched, and connections kept on restarting again and again. Things were beginning to turn weird.
Then, they decided to have measured the radio signals between two known apparently problematic points and to look for some kind of random interference that could be somehow making the links go down; the test was performed between the HACME and the ROUTERS points on June 10 (Thursday), but again nothing was found.
Meanwhile time kept on ticking; almost two weeks had gone by, the links went on getting reset and the situation wasn't easy at all for the affected vendors, having to explain their clients about a problem they couldn't understand and much less explain, and in many cases they even applied discounts up to a 50% to their monthly client fees. Vendors were losing image and money, vendors' clients were losing money, and HACME people were losing calmness.
On the evening of June 11 (Friday), almost in despair browsing the server activity logs, Ron, one of the guys at the HACME security department observed that all restarts were following what seemed to be a certain sequence pattern: every time vendor A restarted the communication, a few seconds later, vendor B restarted it too, and then C and D and some others, almost always in the same order, and they were always the same vendors. He called up his boss immediately, and he rushed back to the office and summoned all his staff for a "war council" next Saturday morning, June 12.
How could this have gone unnoticed until then? The explanation was there was too much activity; if the trace facility was set on at the server for a vendor or group of vendors, it recorded all and every action generated by their client machines. So, finding and correlating specific actions was not easy by hand, and there was no tool for analysing the log records.
After all what had been tried with no results, the problem was obviously hidden somewhere else and this finding could lead to something. It was decided to carefully study the log and then it was verified things were as suspected: definitely the restarts were following a sequence, this meaning the connection drops followed a sequence too. That wasn't at random, someone was conducting the show, and it was programmed.
It was observed that not all vendors were being affected, some of them were not having their connections restarted but they all were small ones, those with

an accumulated market share of no more than 15 to 20%. This meant someone was specifically attacking the bigger data vendors, perhaps trying to make them lose image and clients, who perhaps would turn to other options to get better information, and even more if these other options were cheaper.
Somehow HACME had to stop this, but first they needed to know exactly what was happening, this is, they had to ***IDENTIFY*** what was going on.
They decided to start studying the traffic itself arriving at the server, and to make this somehow easier, they focused their aim in the packets coming from the client machine at ROUTERS.
For this, HACME people decided to use Ethereal, the same software the hacker had used to make his initial traffic study before crafting the attack process. So, next Monday morning (June 14) they connected an additional computer with Windows XP and Ethereal to the same network segment the server was in, although they could have installed the packet sniffer over Linux in the same server, but for some reason someone suggested not to use the affected machine for this purpose.
That same day they started harvesting packets coming from the ROUTERS client to the server, so the origin IP address was 10.11.0.117 and the destination IP address was 10.11.0.5. Curiously, some of these packets looked like the next sample when analysed with Ethereal:

```
Frame 4703 (54 bytes on wire, 54 bytes captured)
Ethernet II, Src: 00:09:6b:d5:f5:53, Dst: 00:a0:8e:1c:a6:20
Internet Protocol, Src Addr: 10.11.0.117 (10.11.0.117), Dst Addr: 10.11.0.5 (10.11.0.5)
   Version: 4
   Header length: 20 bytes
   Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
   Total Length: 40
   Identification: 0x64dc (25820)
   Flags: 0x00
   Fragment offset: 0
   Time to live: 128
   Protocol: TCP (0x06)
   Header checksum: 0xc162 (correct)
   Source: 10.11.0.117 (10.11.0.117)
   Destination: 10.11.0.5 (10.11.0.5)
Transmission Control Protocol, Src Port: 3221 (3221), Dst Port: 10000 (10000), Seq: 74496000, Ack: 74496000, Len: 0
   Source port: 3221 (3221)
   Destination port: 10000 (10000)
   Sequence number: 74496000
   Acknowledgement number: 74496000
   Header length: 20 bytes
   Flags: 0x0014 (RST, ACK)
      0... .... = Congestion Window Reduced (CWR): Not set
      .0.. .... = ECN-Echo: Not set
      ..0. .... = Urgent: Not set
      ...1 .... = Acknowledgment: Set
      .... 0... = Push: Not set
      .... .1.. = Reset: Set
      .... ..0. = Syn: Not set
      .... ...0 = Fin: Not set
   Window size: 32830
   Checksum: 0x0204 (correct)
   SEQ/ACK analysis
      TCP Analysis Flags
         A segment before this frame was lost
```

By watching these packets they found something very interesting: they were arriving at the server from the ROUTERS client machine and they had the RST bit on !! HACME people were able to correlate them with the reconnections, happening always 30 seconds after each RST packet. At first they thought somehow someone had managed to infect the client machines so that a RST packet was sent every certain time, but then they realized the reconections (and obviously the RST packets too) were synchronized and always following the same source address sequence pattern, so the origin had to be the same for all of them: someone was faking these RST packets, making them seem as if coming from each of the client machines so that the server received them as part of the ongoing conversations.

They needed a solution, but they also needed to ***CONTAIN*** the attack somehow while the solution wasn't there.

Based on the premise that someone had somehow found out the IP addresses of the affected data vendors and that the attack was mounted using these IP addresses and the server IP address, HACME people decided to try a containment measure, so they defined another NAT address for the server, and changed the client program configuration at the ROUTERS client machine so that it "talked" to a different server; this new NAT address for the server was 10.11.0.6, and the ROUTERS client IP address (10.11.0.117) was taken from the ***"allwd_vendors"*** group, just in case.

Then, the rules at the firewall were changed and looked like this:

| Source | destination | service | action | audit |
|---|---|---|---|---|
| 10.11.0.117 | 10.11.0.6 | 10000 | accept | log |
| allwd_vendors | 10.11.0.5 | 10000 | accept | log |
| 10.11.0.x | 10.11.0.5<br>10.11.0.6 | any | Drop | log |

This way, traffic to port 10000 coming from the ROUTERS client addressed to the new server at 10.11.0.6 was accepted, traffic coming from the ROUTERS client addressed to the original server was to be dropped, and all the rest wasn't changed.

For NAT to work, the translation rules were changed like this:

| original source | original destination | original service | translated source | translated destination | translated Service |
|---|---|---|---|---|---|
| 10.11.0.117 | 10.11.0.6 | 10000 | original | 192.168.10.5 | Original |
| 192.168.10.5 | 10.11.0.117 | any | 10.11.0.6 | original | Original |
| allwd_vendors | 10.11.0.5 | 10000 | original | 192.168.10.5 | Original |
| 192.168.10.5 | allwd_vendors | any | 10.11.0.5 | original | Original |

These translation rules allowed the ROUTERS client to "talk" to the server at HACME though a different IP address from the one the hacker knew. By controlling and studying during a couple of days the activity log at the server, they knew they were on the right way; ROUTERS communication was started only once a day, while the firewall started to drop packets apparently coming from the ROUTERS client but addressed to the original server NAT address.

This proved HACME suspicions: somewhere in the network there was a program running that was “broadcasting” RST faked packets based on the original IP addressing schema.
They applied the same solution to all the other affected vendors, except to a couple of small ones, suspecting there could be some kind of collusion between smaller vendors against bigger ones, and that these two smaller vendors could be some kind of decoy.

So, by the end of Monday, June 14, they had contained the problem. They created another group at the firewall, ***“clean_vendors”*** they named it, they included in it all the big affected vendors, and they altered the firewall rules to make it work.

| source | destination | service | action | audit |
|---|---|---|---|---|
| clean_vendors | 10.11.0.6 | 10000 | accept | log |
| allwd_vendors | 10.11.0.5 | 10000 | accept | log |
| 10.11.0.x | 10.11.0.5<br>10.11.0.6 | any | drop | log |

For the NAT translation, they altered the rules like this:

| original source | original Destination | original service | translated source | translated destination | translated Service |
|---|---|---|---|---|---|
| clean_vendors | 10.11.0.6 | 10000 | original | 192.168.10.5 | Original |
| 192.168.10.5 | clean_vendors | any | 10.11.0.6 | original | Original |
| allwd_vendors | 10.11.0.5 | 10000 | original | 192.168.10.5 | Original |
| 192.168.10.5 | allwd_vendors | any | 10.11.0.5 | original | Original |

But they only had contained the situation; perhaps it was only a matter of time before the attacker realized what was going on, and somehow managed to find out how to go on making communications go down at will.
They didn’t know where the attack was being launched from, because there were many vendors not being affected, so it could be coming from any of them, or even from many of them in a kind of DDoS, a distributed denial of service.

Among the confusion of the situation, Gary remembered the sweet girl that one day a long time ago had called him up, seizing the information of the IP addresses of several vendors, among them the ROUTERS one. He tried to remember what her name was, but he couldn’t. Had she told him it after all? He didn’t remember. Neither she had ever called him back again. Everyone realized that Gary had been duped, they had “social-engineered” him.
At HACME they realized they had no incriminating evidences at all about anything as to prove someone was specifically involved in the plot, nothing that could lead to a prosecution and a service cancellation. So, they decided there was no point in losing time and effort in trying to find out who was the offending vendor.

The solution was not finding someone to blame (although HACME would have liked this), but implementing better controls that somehow prevented this kind of attack. After finding out a little bit about it, they learnt that many equipment and software providers were releasing versions of their products which avoided these situations. Particularly, they read from CheckPoint the following notice (this is a verbatim copy of the alert published in CheckPoint web site and users lists):

*TCP RFC Alert*
*April 20, 2004*

*Overview:*
*A recently published NISCC advisory (236929/TCP) describes a potential RST attack on any operating system or software that has implemented TCP based on RFC 793 and RFC 1323. While the practical application of this vulnerability is very remote (because an attacker must know both IP addresses of a valid, currently connected pair of computers), if exploited, this vulnerability could allow an attacker to create a Denial of Service condition against existing TCP connections, resulting in premature session termination. For more information about the vulnerability, see the NISCC advisory.*

*Check Point VPN-1/FireWall-1 can protect your entire network against this attack by enforcing that RST packet sequence numbers exactly match the expected sequence within the TCP connection window.*

*Solution:*
*By upgrading to Check Point VPN-1/FireWall-1 R55 HFA-03 or newer, customers are able to protect their entire network from this vulnerability; thus providing additional time and security until other systems and software can be patched.*
*Customers using older versions of NG or NG with Application Intelligence should apply NG FP3 HFA-325 or R54 HFA-410.*

They laughed at the observation of *"this vulnerability is very remote"*, because they happened to be in that marginal percentage affected by the vulnerability, in their case all the conditions were given for the attacker to succeed. Worst of all was that they had received the notice about the upgrade, it was just they didn't pay attention, nobody was in charge of upgrades, so they had to pay the price.

As an ***ERADICATION*** measure, on June 15 (Tuesday) HACME upgraded the firewall software from NG AI FP3 to R55 HFA-03 and more recently to the last R55 HFA-08, which already solved this vulnerability by controlling the sequence number of all the arriving packets, even the RST ones.
These new implementations raise the bar too much for a Reset attack to be successful, because the very exact next sequence number to be sent should be guessed by the attacker, otherwise the firewall will drop the faked packet because of an "out-of-sequence" condition.
Moreover, they decided to implement VPNs between the desktops at the data vendors offices and the HACME site, just like they did from the beginning with brokers. For this, on the last two weeks of June they installed the newest CheckPoint VPN software client available at the moment (CheckPoint SecuRemote NG AI R56) at the remote machines and they set the rules at the firewall to only accept encrypted communications with these remote users. Recently they upgraded it to its last release, Build 311.

An additional control they might have adopted was to activate the MAC addresses filtering at the involved access points. This is a tedious thing to implement, and it doesn't scale well. In the specific case of the products being used by HACME (the ORiNOCO RG-1100), they should specify one by one each and every communication path allowed to be established through each AP, giving the MAC addresses of the two connecting points, this is, for points A and B, both paths, AB and BA should be specified if two-way traffic is to be established, as in any TCP link.
Moreover, wireless MAC addresses can be spoofed so this is not a safe solution.

For real, at HACME they didn't know where the attack had been coming from, the chances were too vast to find. It could have been someone from inside the encrypted WLAN, a kind of "insider", as it really was, or it could have been someone from outside the protected environment, an outsider with the skills and means enough as to break the weak WEP algorithm and find out how to surpass the encryption as to fake traffic as if coming from a site inside the protected network; for a practical example of this, it can be consulted the paper "*War Driving Exploits on Wireless Systems*" by George S. Smith, dated August 2003 (http://www.giac.org/practical/GCIH/George_Smith_GCIH.pdf).
Because of this, HACME people have started thinking about the convenience of going on working with the 802.11b technology or not, although they solved this specific situation; there are some other safer wireless alternatives in the market, but they're more expensive too. They learnt about the LEAP Cisco option, a solution that automatically changes the WEP key every a certain time, but adopting this technology would imply higher costs.
Right now, they're waiting to see what the industry produces for implementing the new 802.11i protocol but because of the money involved and the periodic technological upgrading needs, they're also considering the possibility of moving onto digital leased lines and abandoning the wireless solution, with such a high TCO (Total Cost of Ownership).

There was no place for a ***RECOVERY*** stage, because this was a DoS attack in which nothing was altered in the involved systems or the additional networking components, the attacker performed from outside of them all the time, so there was nothing to be brought back to an original state.
Finally, on the last days of June the HACME security people considered this case as closed. The attacker was not identified, but his or her actions were neutralized with the installation of the new release of the firewall software.

From a ***LESSONS LEARNED*** point of view, they learnt to be more careful, they modified the data-providing application for always generating a log of every request received from the client software and they implemented procedures to control this log everyday. A log is almost useless if nobody watches it. Now if something abnormal starts to happen, it won't take long before it's noticed.

Something else they learnt was to be more aware about the current vulnerabilities, particularly about those directly related to the products they were working with. For this, they subscribed to several lists, among others, the CERT

Bulletin (at http://www.cert.org) the SearchSecurity TechTarget News (at http://searchsecurity.techtarget.com) and the SANS Newsletters and SANS @Risk (at http://www.sans.org). This way, they started to receive daily information about the newest discovered vulnerabilities and problems with systems, programs and devices in almost no time after the news go public.

And another thing they learnt, perhaps one of the most important (specially for Gary), was not to provide information about anything to anybody, it doesn't matter the voice, face or gaze of the requester. On July they started many security awareness activities oriented to the whole organization, with different levels according to who they were addressed to, but always with the global purpose of making people aware of the many risks involved when working with information.
They established that all requests for information and for cooperation must be formally issued and received. Once the situation is checked and cleared then information can be released, but it never can be shared with someone unknown or unchecked, with no management approval. It doesn't matter if the request is done by phone or in person, social engineers work in very effective ways, by phone, mail or personally, so all the staff working at an organization must be aware about this and should not be freely helpful with strangers or even known people asking for not publicly available information without clear (and if possible documented) management involvement.

A final report was written by the IT security departament and raised to HACME's managers, with a description of the whole situation as well as the process followed to solve it and most important, a list of recommendations to be followed by HACME and its personnel. The most relevant items in this list were:

1. Have someone assigned to keep patches updated.
2. Have someone assigned to be aware of new exploit and problems.
3. Have someone assigned to logs watching and better yet, provide him/her with a log analysis tool, to ease and improve the task.
4. Run a security awareness program to educate and train people.
5. Implement a VPN on every external communication, no matter the media.
6. Create an incident handling team, trained and properly equipped.
7. Reconsider keeping a wireless solution; 802.11i promises to be a lot safer than 11b, but high costs might be involved. Leased lines are a good option.
8. If going on wireless, consider deploying an analyzer such as AiroPeek.

## 6) EXTRAS

There are some other countermeasures that could be implemented to avoid this attack to be successful.
For example, it could be adopted the solution proposed in RFC-2385 (by Andy Heffernan, from Cisco Systems) to protect BGP sessions; it consists in signing every TCP segment with an MD5 digest. This solution can be seen as quite good in fighting most kind of faked-packet attacks, and indeed it is, given the MD5 algorithm produces only a 16-byte digest, and although it can be potentially broken it has no known attacks at the moment. Some other stronger

hash algorithms take much more time to compute, making them unsuitable for these purposes.
However, this solution has some other more concrete drawbacks that could render it objectable for pervasively using it. To begin, there's an additional time at the sender for calculating the digest before sending the segment; once at the receiver, some more time has to be added for calculating the digest again, and then comparing it to the one sent together with the segment, as to prove its validity in case they match, and to discard it as faked in case they don't match.
Depending on the nature of the link this additional delays could be inadequate.
Some other problems with this solution are connectionless resets, which are generated for example when trying to connect to a port with no listener, or when sending segments on a stale connection. In these cases, connections will time-out instead of being refused, because reset responses would be "unsigned".

To take a look at what other people is working at about this right now, the document at http://www.ietf.org/ietf/1id-abstracts.txt lists many works in progress about Internet technology problems; among them it can be found the document ***draft-ietf-tcpm-tcpsecure-00.txt*** (published on April 19, 2004) and its review, ***draft-ietf-tcpm-tcpsecure-01.txt*** (effective as of June 2, 2004) written by Mitesh Dalal from Cisco Systems, where he describes some solutions for this TCP Reset problem as well as for some other problems, all consequences of the RFC-793 loose specifications for TCP.
By the way, this draft was commented about by Paul Watson (the same security expert who made public the TCP Reset problem), and he has said that the solution proposed by Cisco could be worse than the original problem, given the device receiving the spoofed RST packet would have to respond to it with an ACK packet, making this way possible to generate a flood of ACK packets going from the receiver device to the supposed origin of the RST packets, nothing but another flavor of a DoS attack.
Cisco Systems has started actions to patent their proposed solution, so that any vendor implementing what they suggest should have to pay some royalties to Cisco for that.
Watson has said that a better solution is included in the standard definition of TCP (RFC-793), given it's enough that vendors implement their TCP solutions so that they verify the sequence of the reset packets before terminating a connection, something that doesn't contravene what's established in the standard. This avoids having to pay any royalties to anyone and it's a better solution after all.

Now, just to provide the reader with a better understanding of the TCP Reset attack described in this paper, here's a piece of the Delphi code that implements it, specifically the main program section is shown and analysed.
The whole exploit (code included) can be downloaded from the site www.iamaphex.net or from http://www.packetstormsecurity.org/0404-exploits and its author is identified as Aphex.
Its command line syntax is: `reset <src ip> <src port> <dest ip> <dest port> <window size> <send delay> [begin seq num]`, but it could be adapted as to be embedded into some more complex program, as the hacker in this story did.

```
...
begin
if Length(ParamStr(1)) < 1 then DoExit;
if Length(ParamStr(2)) < 1 then DoExit;
if Length(ParamStr(3)) < 1 then DoExit;
if Length(ParamStr(4)) < 1 then DoExit;
if Length(ParamStr(5)) < 1 then DoExit;
SourceHost := ParamStr(1);
SourcePort := StrToInt(ParamStr(2));
TargetHost := ParamStr(3);
TargetPort := StrToInt(ParamStr(4));
WindowSize := StrToInt(ParamStr(5));
Delay := StrToInt(ParamStr(6));
Randomize;
WindowPos := Random(4294967295);
if Length(ParamStr(7)) > 0 then WindowPos := StrToInt(ParamStr(7));
Odds := 4294967295 div WindowSize;
WindowCount := 0;
Init;
while WindowCount < Odds do
begin
if WindowPos > 4294967295 then WindowPos := 0;
Send(TargetHost, TargetPort, SourceHost, SourcePort, WindowPos, WindowSize);
Inc(WindowCount);
Inc(WindowPos, WindowSize);
Sleep(Delay);
end;
end.
```

There's no need to be highly proficient in Delphi to understand the idea behind this coding. It first checks the received parameters and then processes them and initializes some internal variables with the provided values for source IP address (`SourceHost`), source port (`SourcePort`), destination IP address (`TargetHost`), destination port (`TargetPort`), window size (`WindowSize`) and delay between sends (`Delay`). If a sequence number to start from was provided, it's considered too (`WindowPos`). If not, a random number is chosen.
Next it divides $2^{32}$ -1 between `WindowSize`, to determine the number of tries it should fire a faked packet to drop the objective connection, as to ensure it hits the window in use (variable `Odds`). If needed, see point 2.2.3 of this paper to understand why this is done.

After this the show begins; a loop is started to cycle `Odds` times, controlling every loop that the sequence number to use (`WindowPos`) is not out of the window space, and in case this is so, it resets it back to zero.
Then a packet is fired from the `Send()` procedure (not shown here), using the specified addresses and ports for source and destination, with the `WindowPos` sequence number. Immediately, the loop counter (`WindowCount`) is increased by one and the window is shifted right by incrementing the sequence number (`WindowPos`) by the size of the window (`WindowSize`). This loop is repeated as many times as the window size in use fits into the sequence number space, this is `Odds` times, as already was said, and then the procedure ends.

## 7) REFERENCES


7.1) IETF RFC-793, TCP Protocol Specification – http://www.ietf.org/rfc/rfc793.txt
7.2) IETF RFC-1323, TCP extensions for high performance - http://www.ietf.org/rfc/rfc1323.txt

7.3) IETF RFC-1700 TCP well-known ports definition - http://www.ietf.org/rfc/rfc1700.txt
7.4) IETF RFC-2385, Protection of BGP Sessions via the TCP MD5 Signature Option - http://www.ietf.org/rfc/rfc2385.txt
7.5) IETF RFC-2827, Network ingress filtering - http://www.ietf.org/rfc/rfc2827.txt
7.6) IETF TCP security considerations – http://www.ietf.org/internet-drafts/draft-ietf-tcpm-tcpsecure-01.txt
7.7) SANS discussion on egress filtering – http://www.sans.org/y2k/egress.htm
7.8) MITRE Corporation CVE CAN-2004-0230 – http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0230
7.9) UK NISCC Advisory 236929 – http://www.uniras.gov.uk/vuls/2004/236929/index.htm
7.10) Open Source Vulnerability Database (OSVDB) – http://www.osvdb.org/displayvuln.php?osvdb_id=4030
7.11) SANS ISC Internet Storm Center - http://isc.sans.org/diary.php?date=2004-04-20
7.12) Paul A. Watson paper "Slipping in the Window: TCP Reset Attacks" – http://www.osvdb.org/reference/SlippingInTheWindow_v1.0.doc
7.13) Mike Gleason paper about ephemeral ports - http://www.ncftpd.com/ncftpd/doc/misc/ephemeral_ports.html
7.14) IEEE 802.11i Overview, by Nancy Cam-Winget from Cisco Systems - http://csrc.nist.gov/wireless/S10_802.11i%20Overview-jw1.pdf
7.15) Article "802.11i: It's Official", by John Cox, posted at Network World Fusion on 06/28/2004 - http://www.nwfusion.com/weblogs/wireless/005548.html

Some generic exploit implementation samples for TCP Reset (as well as for many other vulnerabilities) can be found altogether at the PacketStormSecurity site (http://www.packetstormsecurity.org), a non-profit organization devoted to be a resource of security tools, exploits and advisories:

- http://www.packetstormsecurity.org/0404-exploits/reset.zip
- http://www.packetstormsecurity.org/0404-exploits/reset-tcp.c
- http://www.packetstormsecurity.org/0404-exploits/reset-tcp_rfc31337-compliant.c
- http://www.packetstormsecurity.org/cisco/ttt-1.3r.tar.gz
- http://www.packetstormsecurity.org/0404-exploits/bgp-dosv2.pl
- http://www.packetstormsecurity.org/0404-exploits/Kreset.pl
- http://www.packetstormsecurity.org/0404-exploits/disconn.py
- http://www.packetstormsecurity.org/0404-exploits/tcp_reset.c
- http://www.packetstormsecurity.org/0405-exploits/autoRST.c

## 8. BIBLIOGRAPHY

The following are some of the books among a huge lot of many others, that could contribute to a better understanding of what has been tried to explain in this paper, as well as to broaden the general understanding of IT security.

8.1) *Counter Hack* – by Ed Skoudis, Prentice Hall, ISBN 0-130-33273-9
8.2) *Wireless Maximum Security* – by Cyrus Peikari & Seth Fogie, SAMS Publishing, ISBN 0-672-32488-1
8.3) *The Art of Deception* – by Kevin Mitnick & William Simon, Wiley Publishing, ISBN 0-471-23712-4
8.4) *Hack I.T.* – by T.J. Klevinsky, S. Laliberte & A. Gupta, Addison-Wesley, ISBN 0-201-71956-8
8.5) *Designing Network Security*, by M. Kaeo, Cisco Press, ISBN 1-578-70043-4
8.6) *Network Security*, by C. Kaufman, R. Perlman & M. Speciner, Prentice Hall, ISBN 0-130-46019-2

****** End of the Practical Assignment for GIAC-GCIH ******