Secure and Reliable Covert Channel

[Extended Abstract]

Baishakhi Ray
Department of Computer Science
University of Colorado, Boulder, CO, USA
Baishakhi.Ray@colorado.edu

Shivakant Mishra
Department of Computer Science
University of Colorado, Boulder, CO, USA
mishras@cs.colorado.edu

We describe the design, implementation and evaluation of a reliable and secure protocol to establish a covert channel that not only maintains a high degree of stealthiness, but also provides reliability despite temporary data loss as well as data confidentiality. An important feature of our proposed protocol is that it can be embedded in any legitimate channel that is based on IP. In this paper, we describe our protocol using ICMP. In particular, the proposed protocol satisfies four important properties: stealthiness, lightweight, confidentiality and integrity, and reliability.

Several methods have been proposed to build covert channels between two compromised hosts, exploiting redundancy in TCP/IP protocol, e.g. [6, 2, 3, 4]. Though covert channel has been implemented over many different channels, they all share one similarity. They do not provide enough bandwidth for any real data transfer. Even if some implementation supports data hiding of upto 4 bytes, they violate statistical property of the systems, and hence are easily detectable.

We propose a protocol to build covert channel over any IP-based channels. In this abstract, we focus on building a covert channel over ICMP Echo-Request packet using this protocol. There are two main reasons to choose ICMP echo request packet over any other protocols. First of all, ICMP Echo Request packets are used to check the status of the network. In fact, a majority of users use ICMP to ping other machines to know whether it is up or not, or to check the status of a link. As such, many firewalls and networks consider ping traffic to be benign and allow it to pass through [2]. Secondly, an intermediary cannot disable ICMP traffic without interrupting normal user services, because ICMP is widely used to communicate abnormal conditions during IP routing, and check network status.

Though [2] has already proposed method to send covert data over ICMP echo request and reply packets, our protocol is unique to the extent that instead of exploiting all the 56 bytes that are available as default for ICMP Echo request data [1], we are using only 5 bytes for data. Ping implementation of Linux uses timestamp in first 8 bytes of data field as OS fingerprint. We embed the covert data in last significant bytes of the OS fingerprint to maintain the same statistical property as the normal ICMP data. In addition, to provide a reliable communication, we have designed an 8-bit protocol header that is embedded inside the ICMP identification field. Hence effectively we send 6 bytes of data per packets.

1. PROTOCOL DESIGN

To establish reliability, we choose the simple stop-and-wait automatic repeat request (ARQ) mechanism. There are of course more sophisticated and popular reliability mechanisms available, e.g. go-back-n or selective repeat request. But the major problem with implementing them is that they tend to use large bandwidth to provide much higher data throughput. Since low bandwidth usage is of paramount importance for our design, these sophisticated mechanisms with large bandwidth requirements do not suit well. Furthermore, high throughput provided by these mechanisms is of limited importance, since our goal is to allow covert exchange of small sized data.

1.1 Security

Our goal is to provide two types of security support in our protocol: data confidentiality and data integrity. While techniques to provide this support are well known, the key challenge is to provide this support with minimal resources. For example, data confidentiality can be provided by using an iterated cryptosystem such as 3-DES or AES. However, iterated cryptosystems are hugely complex and computationally intensive so the chances of being exposed increases significantly. So, we have chosen a secure, lightweight encryption-decryption algorithm based on chaotic dynamics.

This algorithm uses one-time pad (OTP) as the basic encryption mechanism. In this algorithm, plain text M is combined (XOR-ed) with an equal length string of random bits K called key or pad. This key is usually generated by a cryptographically strong pseudo-random number generator (CSPRNG). As this key (pad) is used only once and never used anywhere else by any mechanism, this is called one time pad. Encryption mechanism is defined as

$$C = E(M, K) = M \oplus K \tag{1}$$

Decryption mechanism is similarly defined as

$$M = D(C, K) = C \oplus K \tag{2}$$

One-time pad is the only cryptosystem that can be proven to be fully secure. But the basic requirement is that no portion of the key would be reused for another encryption. This requires random key distribution at both ends of the covert channel, which is not so easy. If the key is transmitted by another covert channel and the intermediary can somehow decipher it, the whole method will be at vein. Hence, using OTP in its basic form is not possible in most of the general-purpose applications, including our protocol.

We use chaotic dynamics based pseudo random number generator (PRNG) as the source of our key material to eliminate this drawback. We need secure distribution of the random string K, and K must get initialized to a new random value for each encryption. The first requirement can be achieved if we can keep the amount of information to be exchanged to a minimum and that is exchanged once per encryption. For solving the second problem, we need a cryptographically strong pseudo random number generator (CSPRING) that produces sequences of values that have the following properties:

1. The values have minimal internal correlation.
2. The values convey minimum possible information about their origin.
3. The values are absolutely dependent upon unique and sensitive initial condition for reproduction.

Hence, we propose a new cryptographically strong pseudo-random number generator based on a well-known chaotic map called the logistic map [5]. The logistic map is defined as follows

$$x_n = r x_{n-1}(1 - x_{n-1}), \text{ where } 0 < x_i < 1 \text{ and } 0 \leq r \leq 4$$

When $3.57 < r < 4$, the iteration values $(x_0, x_1, , x_n, ...)$ are random. Now let the pseudo random number generator function be defined as (here R_k is the k^{th} bit generated by PRNG)

$$R_k = \begin{cases} 0, & \text{if } x_{nk} \geq 0.5 \\ 1, & \text{if } x_{nk} < 0.5 \end{cases}$$

The values of x_0, r and n need to be secretly pre-shared between the sender and the receiver. As this logistic map has even distribution between (0,1) on both sides of 0.5, it serves as a good random number generator.

Based on this, the random number generator function is defined as $R_k(x_0, r, n)$ that takes x_0 ,r and n as input and outputs a bit 0 or 1 for each iteration k. The sender breaks the message in chunks of $P_1, P_2, .., $ of size i each and computes the cipher text blocks $C_1, C_2, ...$ The pads (secret keys) are defined as $B_1, B_2, ..$ from which the each cipher text block is computed.

$$B_1 = R_1..R_i \qquad C_1 = P_1 \oplus B_1$$

$$B_2 = R_{i+1}..R_{2i} \oplus C_1 \qquad C_2 = P_2 \oplus B_2$$

$$B_n = R_{(n-1)i+1}..R_{2ni} \oplus C_{n-1} \qquad C_n = P_n \oplus B_n$$

Since the receiver has knowledge about x_0, r and n, it can easily generate $B_1 = R_1..R_i$. On receiving C_1 the receiver generates the original plain text by simply XOR-ing it with B_1.

$$P_1 = C_1 \oplus B_1$$

The receiver generates the next random key (pad) by

$$B_2 = R_{i+1}..R_{2i} \oplus C_1$$

The received cipher text C_2 is then deciphered as

$$P_2 = C_2 \oplus B_2$$

Similarly,

$$P_n = C_n \oplus B_n$$

This method provides data confidentiality with minimal computational requirements. In addition, a checksum can be provided to provide support for data integrity. This feature is optional, because the legitimate channel with in which covert data is being transmitted may itself provide some support for data integrity. For example, ICMP provides a checksum field that can be used to detect data tampering.

2. IMPLEMENTATION AND EVALUATION

We now present a design to send covert data via ICMP Echo request packets. One Approach is to put the protocol header and covert data together, and send it across after proper encryption. But this is only feasible when the underlying covert channel has enough bandwidth. For Example, if we use ICMP echo request data as our covert channel medium, we can do this. We assume here that the maximum size of the data part is 56 bytes (as normally ICMP data part sent by Ping command has 56 bytes of data). With 56 bytes of payload, it is easy to squeeze the data, the protocol header and cryptographic checksum (optional) together in ICMP payload. But if the covert channel media is something like TCP timestamps where the available bandwidth is 1 bit per packet, this method is impractical.

In addition, generating echo request packets using this method will cause changing the data part with every packet. In general, every operating system puts it finger print in the data part. Hence, ICMP echo request payload carries a fixed bit pattern. If the data part of ICMP echo request keeps on changing with every packet the covert channel will be easily detectable.

In our design we implement two covert channels in the same packet. One is used for sending the protocol header and the other one is used for sending data. For example, in an ICMP echo packet, the last byte of ICMP identifier field can be exploited to carry the protocol header and data can be put in ICMP payload. Necessary care must be taken so that the data sent through covert channels have characteristics of the operating system fingerprint. One of the main advantages of this approach is that it can use two different low-bandwidth covert channels together to send data reliably across, while neither of these channels alone has enough bandwidth to send the data and protocol header.

Now, we will look into how the protocol header can be hidden in the ICMP identifier field to make detection difficult. When we say that it is difficult for an intermediary to detect the covert transfer of data, we mean that by simply inspecting all packets it will not be possible for the intermediary to decide with certainty that a particular packet is carrying some covert data and not normal traffic generated by the end hosts.

As most of the ICMP echo request packets are generated by ping program, we look into how the ping program sets the ICMP identifier field. The current implementation of ping in Linux fills the ICMP identifier field with lower 16 bits of the Process ID of the "ping" process. So each time the ping program is run, it uses different value in the ICMP identifier field. But all packets sent by the same instance of "ping" contain the same value in the ICMP identifier field. In Linux, process IDs are initialized to 300 and incremented by 1 every time a new process is created. When the ID overflows it gets initialized back to 300 again. All the IDs below 300 are reserved for the kernel threads. So, the ICMP identifier field can safely contain random values as the monotonicity of the process IDs won't always reflect in the lower 16 bits, i.e. it can increase and decrease depending on the particular Process ID in use at that time.

Though monotonicity in the identifier is not absolutely necessary, but the last 2 bytes of process ID wrap around after $2^(16)$ processes. Hence to minimize the anomaly caused by insertion of data, we chose to increase the 1st byte of the ICMP identifier field monotonically, while inserting the protocol header in the last byte of the identifier field.

In Linux if time measuring option is enabled (which is enabled by default) the "ping" program fills the first 8 bytes of the payload with the current time. First 4 bytes contain the seconds portion in the network byte order and the next 4 bytes contain the microsecond portion in the network byte order. The rest of the data is filled with values from 8 to 64. So, in the payload, only the first 8 bytes are variable, rest always remains same.

Hence, we suggest using the lower 4 bytes (where the microsecond portion is stored) and the lowest byte of second portion together to hide our data. As this causes minimal changes in the timestamps, it'll be very difficult to differentiate from the clock skew.

2.1 Evaluation

Our protocol is extremely light weight in the sense that its resource consumption is similar to that of the underlying communication channel. The resident memory of the Ping program was 592 KB, while that of covert channel was 584 KB. The shared memory of the Ping program was 500 KB, while that of covert channel was 476 KB. Finally, the running times of the two programs was identical.

We have run a number of experiments using our prototype implementation to see if incorporating our protocol headers and data causes any statistical difference in the bit patterns of the corresponding ICMP packet from that of a normal ICMP packet. Our experimental set up consists of two Pentium PCs running Linux connected to each other via a 10 Mbps Ethernet hub. In each run, we send random data bytes that are embedded in 5 bytes of ICMP payload. Each data set contains 60 modified timestamp values. It is then compared with the corresponding unmodified timstamps.

A preliminary analysis shows that there is no statistical difference between the bit patterns generated by our covert channel laden ICMP packs and normal ICMP packets. Figure 1 shows the value of the time stamp field in ICMP packet under normal conditions, i.e. when no covert data has been embedded, over 70 different runs. Figure 2 shows the value of the time stamp field in ICMP packet when our protocol has been deployed, again over 70 different runs. Notice that both of these graphs show similar statistical behavior.

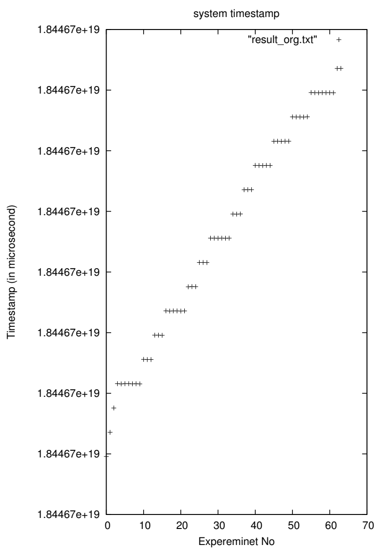

Figure 1: System timestamp values

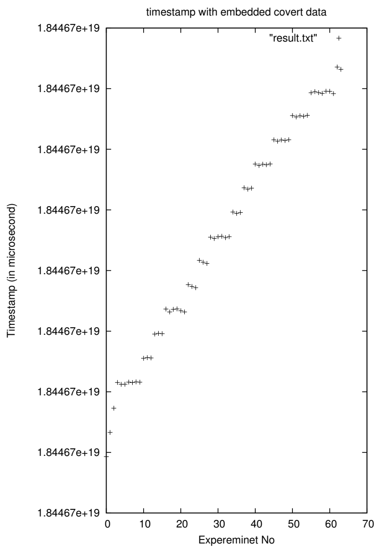

Figure 2: Timestamps in covert channel

3. CONCLUSION AND FUTURE WORK

This paper describes the design and implementation of a protocol that not only supports construction of moderate bandwidth covert channels, i.e., 5 bytes data per packet, but also provides support for reliability and security in covert communication. In fact, a preliminary analysis shows that it is not possible to detect this covert channel based on statistical analysis of sent data. Future work includes a detailed analysis of the protocol over both a short-range and a long-range communication network.

4. REFERENCES

[1] *Man Ping: Linux manual page of PING.*

[2] Project loki. *Phrack Magazine*, 7(49), August 1996.

[3] K. Ahsan and D. Kundur. Practical data hiding in tcp/ip, 2002.

[4] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts. Covert messaging through tcp timestamps.

[5] L. Kocarev and G. Jakimoski. Logistic map as a block encryption algorithm. *Physics Letters A*, 289, 2001.

[6] C. H. Rowland. Covert channels in the tcp/ip protocol suite. *First Monday*, 2(5), May 1997.