Covert Channels

Renaud Bidou - Radware <renaudb@radware.com>
Frédéric Raynal - MISC Magazine <pappy@miscmag.com>

keywords : covert, channels, storage, timing, resource, subliminal, network, bounce, multiplexing, portknocker

Abstract

Information and communication dissimulation is not a new topic. However applications remains numerous and most recent techniques make such channels more difficult to detect. This can be a good thing if covert channels are used to protect privacy or increase security of critical communication. However when applied to security policy bypassing, information leak or compromised system control the knowledge of such techniques becomes mandatory to enhance detection engine.

This article will focus on the concept of covert channels, from the genesis of the computer age to actual protocol and applications, providing examples of application and detailing advantages and drawbacks.

Introduction

Covert Channels are not everywhere. However they CAN be everywhere, thus providing answers to several issues raised by the use of encryption : legal restrictions and lack of discretion. In the first case the main concern is the protection of personal privacy. One may want his communications not to become public. In the second one the issue is to have communications remain undetected. If the content of an encrypted mail should not be readable, the communication itself is not stealth. And this piece of information may be valuable, mainly if one knows that two entities tries to protect the privacy of their communication.

As most security concepts, covert channels do have a dark side. As they provide a stealth and secure communication channel, they can undoubtedly be used to establish connections that are theoretically prohibited by the security policy. Then information leaks become possible as well as asynchronous command channels between the compromised system and its master.

History and concepts

Lampson's Covert Channels

Covert Channels have be defined for the first time by Lampson in 1973 as *a communication channel not designed for any kind of information transfer* [Lampson73]. Unsurprisingly covert channels defined in 1973 are system based. However they are still relevant in any shared environment.

- *Storage Channels* : the most basic indeed, based on the used of a shared data storage area. Most evolved techniques rely on locks or semaphores which positioning may be a piece of information such as a 1 when no lock means 0.
- *Timing Channels* : the time needed by a process to perform an operation can be voluntarily manipulated to provide information to another process.
- *Termination Channels* : a process launches a task. It this task is finished at a specified time it means 1, 0 otherwise.
- *Resource Exhaustion Channels* : the value (0 or 1) is provided by the availability of a specific resource which may be filled up (hard disk), overloaded (100% cpu) etc.
- *Power Channels* : in this case the information is based on power consumption.

Subliminal Channels

In 1984 Simmons introduces the concept of subliminal channel through steganography. This concept, applied to the case of two prisoners [prisoners] exchanging sensitive information about how to escape through an open channel, is then extended to channels built over digital signatures schemes [subliminal]. In this case the channel is not stealth. But, as information is hidden into information, the data exchanged doesn't look sensitive and shall not raise suspicion.

Covert Channels then usually refer to one or both concepts : stealth channel and/or hidden information. In any case the main concern is information and more precisely the transmission of such information through what is called *information stream*.

Network Channels

The first publication on network covert channels is an article published in 1987 [Girling87]. It points out 3 covert channels (2 *storage channel* and 1 *timing*) to show the possibility to create a channel through a LAN. This study didn't measure the impact on the network bandwidth, however it opened the door to a new field of investigations.

This work has been the basis of [Wolf89] where the author shows the possibility to create such channels in IEEE 802.2, 3, 4, and 5 networks with padding and unused bits used to transmit information. Then in [Handel96], the OSI model is analyzed and will provide "theoritical" channels when [Rowland96] provides limited but realistic channels through TCP/IP.

Covert Channels and Security Policy

It clearly appears that Covert Channels are deeply linked to the security policy.

In the case of a DAC based security policy, covert channels don't make sense as the user is defining its own security policy. In such case malware, spywares and trojan just have to use the identity of the user and the operating system will not be able to make a difference between the legitimate user and the spoofed one. On the other hand with MAC, RBAC and the like models the user has no way (theoretically) to control the security policy. Then information transmission will need a communication channel that was not originally designed for any kind of information transfer : a covert channel.

Covert Channels characteristics

Covert Channels are hidden. However they have the same characteristics than other communication channels :

- *Capacity* : this is the quantity of information that can be transmitted through the channel;
- *Noise* : is the amount of perturbations that can interfere with the information while it is transmitted trough the channel;
- *Transmission Mode* : can be synchronous when the information is received and managed on the fly by the destination entity, or asynchronous otherwise

Capacity is a very important part of the global quality of a channel. From a security point of view, a larger capacity channel will make possible for more information to leak. On the other hand capacity may impact the overall furtivity of the channel when this one relies on usual but quiet channels.

Noise will highly rely on the nature of the channel. As an example Simmons' *storage channels* are usually not very sensitive to noise as probabilities for a process not to be able to write on a file it owns are quite low. On the opposite *timing channels* are very noisy because of the number of external factors that can have an impact on execution time of a process.

Network Channels

It seems obvious that the very usage of covert channels is the stealth and secure transmission of information from one system to another through a shared and potentially managed network. Some applications such as information leaking or security policy bypassing immediately come to mind. Moreover, these channels can also be used to transmit discretely an opening sequence to a portknocker or orders from the master to his DDoS zombies. In such cases the difficulty to reproduce or to trace back the traffic is a key advantage.

Basic Techniques

The very first point for *network channels* setup is to make sure that the packets will be normal. They must be perfectly RFC compliant and belong to a legitimate traffic on the

network. If those two conditions are not satisfied, packets will raise alerts, if they are not blocked.

This is probably the reason why ICMP was one of the first protocols that was used for covert channels, often considered (and is still sometimes) as innocuous and offering a great "legitimate" possible load thanks to the data field. This field provides from 84 bytes (ICMP Redirect/Destination unreachable) to (MTU-28) bytes (ICMP echo request/replies), which is a very good capacity.

A trivial implementation is provided by the `icmpchat` tool [icmpchat]. This is an ICMP stealth equivalent of `talk` using any ICMP type and code and providing encryption.

Into network protocols

The use of the data field to transmit information is not the most subtle way. In the preceding example the increase of ICMP traffic with important (thus encrypted) payload may raise suspicion or at least appear like an anomaly that may be investigated by network operators.

Then other fields should be used as long as they meet the following requirements :

- There value can be set at will as long as it remains legitimate;
- The spectrum of possible values must be the largest as possible in order to increase the capacity of the channel.
- They must not be systematically and randomly modified during transmission (as dynamic NAT would do with TCP source port).

IP Header

IP options don't look very appropriate to *network channels* as they are not always supported by IP stacks. Also they may be changed or removed by some internetworking equipments on the path, such as routers. Some firewalls also systematically block packets with IP otpions. This makes channeling this way impossible.

Source IP address looks more stable as it will only be changed in case of static or dynamic network address translation. On the other hand a bi-directional communication will be possible only if both extremities of the channel are hard coded which lacks of flexibility and discretion.

The last one is then the IPID which has been unsurprisingly used a lot [ipid][stegtunnel] to create *network caennels*. However its overall capacity is only 2 bytes, which makes it efficient for command launching and portknocking but useless in terms of data transfer.

UDP Header

UDP offers more capacities as 3 fields out of for can be used to transmit information : source port, data length and checksum. The source port is quite interesting as it will only be modified in case of dynamic NAT. The other fields can also be used as long as no inline equipment on the path makes layer 4 checks which is generally the case, mainly because of performances issues.

The use of all the fields makes possible to transfer up to 12 bytes per packet, with no guarantee on the reliability and integrity of the communication. However this drawback can be turned into advantage as such traffic is far more difficult to detect for *stateful* engines than any TCP *handshake* violation.

TCP Header

As it is longer the TCP header offer more possibilities for communication channels : sequence numbers, TCP windows size, source port, checksum, flags and options, more particularly the time stamp. The theoretical load of information for each packet is then of 38 bytes.

However, anomaly detection engines, would they be based on communication state or RFC compliance - or both, do have a huge impact on this load. As an example a TCP session must start with a SYN (with the Push flag optional) and the acknowledgement sequence number must be 0. The initial load is now 32 bytes and 2 bits. Also sequence number values must follow some rules, limiting the range of possible values for the corresponding field. What is more, SYNFlood protection mechanisms such as SYNCookies make impossible the use of acknowledgement sequence number to transmit information.

Application channels

At the application layer, covert channels will make possible to use legitimate traffic both from the security policy and communication standards point of view. Also such traffic should not look too suspicious as important HTTP traffic always look more natural than ICMP.

Due to outgoing restrictions, most application traffics used to create channels are the web, mail and DNS ones. When they are linked to really mechanisms such as proxy chains or anonymous remailers, these channels ensure integrity (in the TCP meaning) of the communication, very good capacity and an indubitable resistance to source trace-back. What is more, and because application protocols are full of functionalities, possibilities to create channels are almost unlimited.

Also detection may be quite impossible If not based on other criteria than protocol compliance to security policy and standards. However, behavior analysis will make possible to identify that a HTTP connection lasting half an hour is not normal, and make possible to detect the channel.

Bouncing

Mechanisms described before have low capacity. Also they are easy to trace back to the source. It becomes then possible to definitively block the source and make the channel unusable. Bouncing techniques take advantage of a third party equipment that will relay the information between the source and the destination. Moreover the bouncing host is not necessarily aware of this and can be changed for each packet, making filtering operations almost impossible.

SYN/ACK bounce

According to the RFC the sequence number transmitted by a client in a SYN packet to an open TCP port is incremented by 1 by the server and sent back to the client in the acknowledgement sequence number field. Thanks to this standard behavior, it is possible to transmit a piece of information by using any server and spoofing the source IP address so that it points to the other extremity of the channel.

Example: Computer A (10.0.0.1) wants to communicate with computer B (192.168.0.1) by bouncing on the server C (www.bounce.com) in order to transmit the following message `service ssh start\n`, which is 18 bytes long and encoded in hexadecimal (`\73\65\72\76\69\63\65\20\73\73\68\20\73\74\61\72\74\0A`). Using sequence numbers (4 bytes), this message will be divided in 5 blocks :

- sequence 1 : `0x73657276`
- sequence 2 : `0x69636520`
- sequence 3 : `0x73736820`
- sequence 4 : `0x73746172`
- sequence 5 : `0x740A0000` - two null bytes have been used for padding

In order to establish the channel, 5 SYNs have to be sent to server C on port 80, with B (192.168.0.1) as a source address and with the sequence numbers that have been calculated before. C will then send the SYN/ACKs to the spoofed source (B). In these SYN/ACK packets the acknowledgement sequence numbers will be respectively `0x73657277`, `0x69636521`, `0x73736821`, `0x73746173` and `0x740A0001`, which are the previous sequence numbers sent by A and increased by 1. Reassembly on the other side of the channel is then trivial.

However depending on the implementation of the channel terminator, its stack, receiving out of session SYN/ACK may generate RST. In such case these couples of (SYN/ACK,RST) packets may look suspicious and lead to the detection of the channel.

ICMP Error bounce

The same principle can be applied to ICMP error messages. The RFC specifies that for `ICMP destination/port unreachable`, `ICMP time exceeded` and `ICMP redirect` the

data field of the packet must contain the UDP header as well as the first 48 bytes of data of the packet which has generated the error.

The first use of this mechanism is obvious for computer A and B willing to establish a covert channel thanks to C :

1. A sends a packet that will generate an error. The source of this packet is spoofed to be B. Data to be transmitted can be either in the UDP header or in the first 48 bytes of the UDP data.
2. C identifies the error and sends the appropriate ICMP message to the source : B
3. B receives the packet and gets the data.

Generating such errors is quite simple. As an example an `ICMP port unreachable` packet is generated when a UDP packet is sent to a closed UDP port. In this case the bouncing server can be almost any server on the Internet. However, in such case our UDP packet must be RFC compliant, which means that fields of the header must be correct. Then we loose 16 bytes which means the total capacity of the channel is 48 bytes per packet.

Another, and more interesting way, to generate ICMP error messages is to send a packet with a particularly low TTL. In this case the error is generated after analysis of the IP header. It means that the layer 4 information don't have to be RFC compliant, leaving the capacity unchanged to 64 bytes per packet. What is more, there is no need to identify a specific bouncing server, as the ICMP message will be generated by the router which has detected the TTL=0 field. As usual the source of the packet will be spoofed and the error message will be sent to the second extremity of the tunnel. Then the bouncing server may be any router on the Internet. Then, and as the destination of the packet possibilities to identify the channel are lowered once more.

Application layer bounces

Bouncing can also be applied to application level. The main problem however is that the TCP session establishment will be very difficult with a spoofed source IP so we have to rely on UDP which remains a privileged base. SIP is a very good example of what can be done. Basically the main SIP headers are the following.

```
1 INVITE sip:bob@sipserver.com SIP/2.0
2 Via: SIP/2.0/UDP www.prendsmoipouruncon.com;branch=z9hG4bK776asdhds
3 Max-Forwards: 70
4 To: Bob <sip:bob@sipserver.com>
5 From: Renaud <sip:renaud.bidou@prendsmoipouruncon.com>;tag=1928301774
6 Call-ID: a84b4c76e66710@www.prendsmoipouruncon.com
7 CSeq: 314159 INVITE
8 Contact: <sip:admin@prendsmoipouruncon.com>
9 Content-Type: application/sdp
10 Content-Length: 142
```

This header is a connection request. It can be simply compared to a SYN request in TCP sessions. In the line #1 the header contains the `INVITE` command which identifies the destination with its ID. Line #2 provides the address `www.prensmoipouruncon.com` to which the answer must be sent. Then on line #7 is a command sequence number, coded on 4 bytes, that will be copied into the answer to the request.

As SIP doesn't pay attention to IP information, mainly because of proxy mechanism, the channel appears clearly. The sequence number will contain the data, the bounce server is a SIP server and the extremity of the tunnel is the spoofed, at the application level, destination of the answer provided through the field `VIA` on line #2.

Multiplexing bounces

Bouncing techniques can be enhanced thanks to a simple multiplexing technique. The objective is not to use 1 but *n* bounce servers. in our example, using 5 TCP sequence numbers it is possible to use 5 different servers to generate the SYN/ACK packets. It is then necessary to let some delay between each packet but tracing and detection operation will be far more difficult to perform.

Hands-On

One application of portknockers is to launch services on demand. Then as an example, it is possible to have SSH service run only when needed. This will reduce the risk of exploitation by new or unpublished vulnerabilities. In the mean time hiding as much as possible the opening sequence is also important, to prevent from any replay attempt. Then a covert channel with bounces multiplexing might be the solution.

Using apk [apk] the server configuration file (`apks.conf`) would look like the following.

```
#TYPE ID TTL SOURCE SPORT DPORT SEQ SEQ_ACK URG ACK PSH RST SYN FIN WIN DATA  ACTION                    KEY
E     1  *   *      80    *     *   12345   0   1   0   0   1   0   *   *     NEXT(2)                   NULL
F     2  *   *      80    *     *   67890   0   1   0   0   1   0   *   *     NEXT(3)                   NULL
F     3  *   *      80    *     *   13579   0   1   0   0   1   0   *   *     CMD("service ssh start")  NULL
```

The ssh service will be launched if the potknocker receives 3 SYN/ACK packets coming from web servers (`SPORT=80`) with sequence numbers respectively 12345, 67890 and 13579. In order to have those operations automated the client-side configuration file (`apkc.conf`) will be the following.

```
#TYPE ID TTL SOURCE SPORT DPORT SEQ   SEQ_ACK URG ACK PSH RST SYN FIN WIN DATA  ACTION                    KEY
E     1  *   *      *     80    12344 0       0   0   0   0   1   0   *   *     NEXT(2)                   NULL
F     2  *   *      *     80    67889 0       0   0   0   0   1   0   *   *     NEXT(3)                   NULL
F     3  *   *      *     80    13578 0       0   0   0   0   1   0   *   *     CMD("service ssh start")  NULL
```

This way each step will be programmed to send 3 packets with a SYN flag on port 80. Servers will be specified each time by the user. Sequence numbers are the one requested by the portknocker minus 1.

Conclusion

If covert channels can be almost everywhere the quality of the vector may vary a lot. Their efficiency highly depends on their purpose and the environment in which they will be setup. However, if they are used in malicious way they remain a threat that has to be considered in order to ensure the security and the integrity of any IT infrastructure. Then prevention mechanisms based on stateful engines, protocol anomaly detection, behavior analysis and protocol repartition and trends has to be improved and carefully tuned in order to provide reliable and efficient detection and reaction.

On the other hand such channels, used on unmanaged and shared environment such as the Internet, provides increased security for communications, as long as their existence and the way they are implement for each particular operation remain secret. Security by obscurity one would say...

References

[apk] Advanced Port Knocker - http://www.iv2-technologies.com/~rbidou/apk-1.0.tar.gz

[Girling87] Covert channels in LAN's - C. G. Girling
IEEE Transactions on Software Engineering, 1987

[Handel96] Hiding Data in the {OSI} Network Model - T. G. Handel et M. T. Sandford
Workshop on Information Hiding, 1996

[icmpchat] icmpchat - http://icmpchat.sourceforge.net/

[ipid] passivecc_ipid - http://invisiblethings.org/tools/passivecc_ipid.c

[Lampson73] A Note on the Confinement Problem - B. W. Lampson -
Communication of the ACM, 1973

[Pinchuk] Covert Channels in Networking - Evgeny Pinchuk -
http://www.cs.tau.ac.il/tausec/lectures/Network_Covert_Channels.pps

[prisoners] The Prisoners' Problem and the Subliminal Channel
G.J. Simmons, Proceedings of CRYPTO '88, Plenum Press (1984) pp 51--67

[Rowland96] Covert channels in the TCP/IP protocol suite - C. H. Rowlan
http://www.firstmonday.dk/issues/issue2_5/rowland/

[subliminal] The Subliminal Channel and Digital Signatures
G.J. Simmons, Advances in Cryptology -- EUROCRYPT '84, Springer LNCS v 209

[stegtunnel] stegtunnel - http://www.synacklabs.net/projects/stegtunnel/

[wolf89] Covert channels in LAN protocols - M. Wolf
Workshop on LAN security (LANSEC'89), 1989