The Port-in-Use Covert Channel Attack

Dmitry Efanov[1] and Pavel Roschin[1]

National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, Russian Federation

Abstract

We propose a port-is-in-use attack, which is intended for leaking sensitive information in multilevel secure operating systems. Our approach is based on TCP socket mechanism widely used in Linux for interprocess communication. Despite the strong limitations inherent in operating systems with mandatory access control, sockets may not be restricted by the security policy, which makes it possible theoretically to transfer information from one process to another from a high security level to a low one. The proposed attack belongs to the operating system storage transition-based class attack. The main idea is to use the availability of TCP port, which is shared among processes at more than one security level, as the communication medium. The possibility or impossibility of binding a socket to a predefined port is used to transmit a bit of 0 or 1 respectively. We implement proof-of-concept exploit, which was used to check the idea and to evaluate covert channel capacity. Experimental results show that the proposed technique provides high rate covert channel, that means a significant threat of confidentiality in multilevel secure operating systems.

Keywords: covert channel, information flow, TCP socket, proof-of-concept exploit, multilevel security, mandatory access control, interprocess communication

1 Introduction

The development of the concept of covert channels in operating systems includes several important steps. The term of covert channels was introduced as one kind of communication channel that a service program should be confined to use [9]. Later, the shared resource matrix methodology was proposed [8], which focused primarily on the discovery of covert channels in a formal top-level design specification of operating system kernels and trusted processes rather than in source code. The minimum conditions for the existence of a storage channel are as follows: (1) there must exist a global variable in the system to which both the sending and the receiving processes have access; (2) there must be a way (usually by invoking system calls) in which the sending process can alter the value of the global variable, and the receiving process can detect or observe (view) any change in the global variable; and (3) there must be a way in which the sending and the receiving processes can synchronize their operations so that the events of information flow can happen.

Covert channels are of great interest in the context of multilevel secure (MLS) operating systems researching from the security point of view. It is thus important that there be no covert channels that allow a malicious user (a rogue employee, who abuses his trusted privileges) to transfer information from a high security level to a low one.

We propose a get-port-in-use attack, which is based on TCP socket mechanism widely used in Linux for interprocess communication. Despite the strong limitations inherent in MLS operating systems, sockets may not be restricted by the security policy and may be used to transfer information from one process to another regardless of their security levels.

Our attack corresponds to the classical three communication steps in covert channel attack [13]. First, a send mechanism emits bits of data by manipulating a shared resource. Then, a receive mechanism infers the bits by monitoring a shared resource. Finally, an optional feedback mechanism back to the sender provides direct synchronization and reduces noise.

2 Covert Channel Classification

For the last twenty years different approaches have been taken to classify the covert channels [4, 2, 5, 15, 10, 21, 12].

Traditionally the covert channels are divided into two main types: storage and timing channels [11]. Although fundamentally the same, they differ in the way that information is encoded. In a storage channel, there is a shared global resource in the system that acts as the medium for information transfer, where a user can potentially change its value, and another user can potentially view the change directly or indirectly. Timing channels send information in a way that involves manipulating the timing properties of a component of the system.

The problem is that there are a variety of mechanisms in operating systems to transmit timing and storage information between processes, including many data structures that can potentially be manipulated (directly or indirectly). New techniques of attack are constantly emerging [16, 6, 14, 1, 7, 3, 23, 20, 17].

There are basically two classes of covert storage channels: resource exhaustion and event count [19]. Resource-exhaustion channels exist wherever system resources are shared among users at more than one security level. In event-count channels, the sending process encodes multiple bits by requesting or not requesting a shared system resource. Furthermore, information can be transmitted by value-based and transition-based channels [22]. The key distinction between these methods is that value-based channels transmit information based on the actual value present somewhere in the system, whereas transition-based channels use the change of a value to transmit information.

In this paper we use the classification scheme with three orthogonal criteria: storage/timing, network/OS/hardware, and value/transition-based [13].

3 Threat Model

Consider the operating system with MLS security policy. Malicious user has access to some sensitive files with high security level. He wants to violate MLS security policy and compromise sensitive information. He has advanced technical knowledge. He has a range of security clearances and has the rights to run processes at different security levels.

4 Attack Scenario

The idea of possible covert channel was noted in [24]. We assume two processes are running at different security levels. The sending process has a high security clearance and the receiving processes has a low security clearance. The security policy prohibits processes from communicating directly in read/write manner.

The low process can send a signal (by kill() function) to the high process. Both processes have permissions to create a socket and to bind a socket to the predefined port. Obviously they cannot simultaneously bind the socket to the same port. Suppose that there is some mechanism that allows the processes to agree in advance on the port number.

The proposed attack is based on using of the TCP socket mechanism. The main idea is to use a state of network port, which is shared among processes at more than one security level, as the communication medium. The possibility (port is free) or impossibility (port is in use) of binding a socket to a predefined port is used to transmit a bit of 0 or 1 respectively from high security level to low security level.

An TCP socket address is defined as a combination of an IP interface address and a 16-bit port number. A socket is created by calling the socket(2) function. A newly created socket has no remote or local address and is not fully specified. When a process wants to receive new incoming connections, it should bind a socket to a local address and port using bind(2) and then call listen(2) to put the socket into the listening state. Only one TCP socket may be bound to any given local (address, port) pair. If a process try to bind to an address already in use, it will get an EADDRINUSE error.

In Linux kernel inet_csk_get_port() is used to obtain a reference to a local port for the given socket. Arguments passed to this function is sock structure associated with the socket and the port number to which a socket needs to be bound [18].

The sending process encodes a bit of 1 or 0 by binding or not binding a socket to a local address respectively. The receiving process detects the bit by trying to bind a socket to the same port number. By observing the return result of bind() system call that allocates the port number, the receiving process can determine the value of the bit from the sending process.

The proposed attack belongs to operating system storage transition-based class attack according to [13]. The transition-based channels appear to be the more difficult to prevent, since all that is required is the ability to change a value to anything, not necessarily a specific value [22]. It also belongs to event-count class channel according to [19].

5 Implementation Details

We present a proof-of-concept exploit, which implements suggested the get-port-in-use attack. The sequence diagram shows processes interactions arranged in time sequence (Figure 1). The source code of exploit is also provided for study and evaluation here github.com/scriptum/port-in-use-covert-channel.

As shown earlier, the attack is based on the AF_INET protocol family with SOCK_STREAM socket type, which provides sequenced, reliable, two-way, connection-based byte streams.

Since the specific address does not matter, INADDR_ANY (0.0.0.0) is specified in the bind call, so the socket will be bound to all local interfaces. Note that the address and the port are always stored in network byte order. So it's necessary to call htons(3) on the number that is assigned to a port.

A high process generates a message and encodes it in the form of 0 and 1. In our implementation, the message is an array of bits.

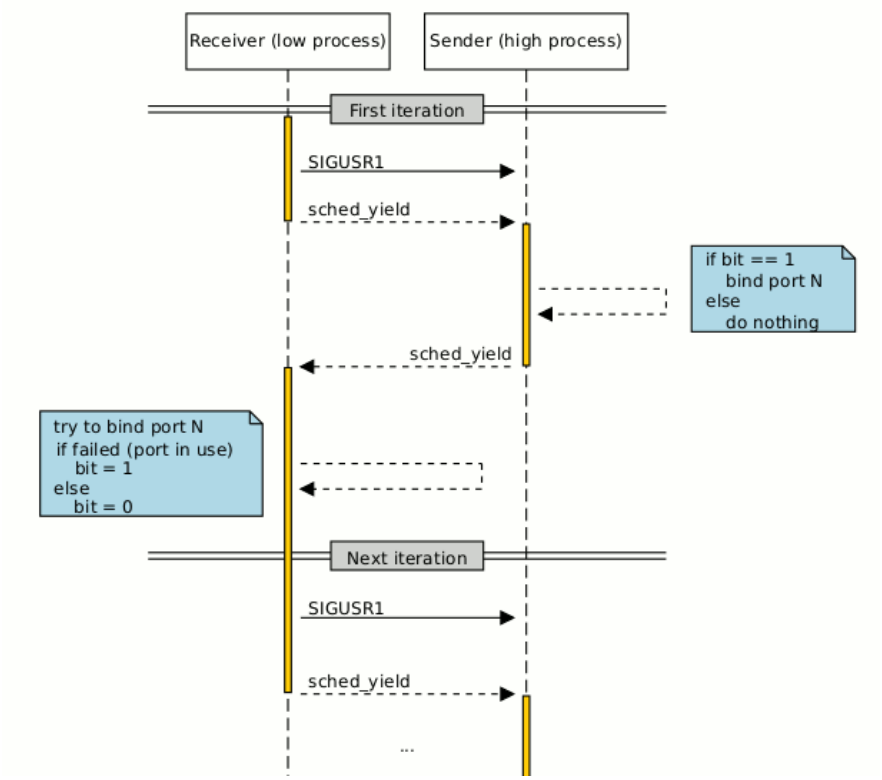

Figure 1: The diagram of the port-in-use attack

A high process sets the disposition of the signal SIGUSR1 to a signal handler. When the signal SIGUSR1 is delivered to the high process, handler is called and performs the following actions:

1. Closes the server socket, if it was opened previously.
2. Depending on the value of the current bit of the message it performs:
   - If the bit is 1, it creates a server socket, binds it to the predefined port, and starts listening to it.
   - If the bit is 0, then it does not create a server socket.
3. Increments the counter by one.
4. Frees the processor (call sched_yield()).

After setting the signal handler, the high process enters the infinite loop in which it sleeps.

The low process starts and enters a loop in which the message is sequentially received bit by bit. To do this, it forms a buffer for receiving the message and sets the bit counter to 0.

The low process sends a signal (in our implementation – SIGUSR1) to the high process.

Then the low process creates a client socket and try to bind to the predefined port.

If the server listens to this port, that means that this port is in use. So, the client will not be able to bind to and this will mean that the client received a bit with a value of 1.

If the server does not create a socket and the predefined port is free, then the client will be able to bind to the port and this will mean that the client received a bit with a value of 0.

After that, the low process increments the counter by one and closes the socket.

To increase the capacity and reduce the noise of covert channel significantly, we propose using an operating system scheduler to synchronize sender and receiver processes. Modern hardware come with CPU with more than one core. The scheduler attempts to use all available cores, so synchronization using the scheduler is difficult, since each core dedicated to a particular process. However, Linux allows to assign a process to a specific CPU core by calling sched_setaffinity() function. Thus, by assigning the sender and receiver to the same CPU core, scheduler will have to split the time between those two processes.

We use the call sched_yield() as synchronization tool, which interrupts the current process and forces it to the waiting state and causes the kernel to switch to the next process that is waiting in the queue. Since there are only two CPU consumers (sender and receiver), they are switching one by one and synchronize without system timer. This technique works well on modern Linux kernels and allows to transfer the data between processes on relatively high speed without the noise.

Our attack is implemented on a computer with an Intel(R) Core(TM) i7-3770 CPU at 3.40 GHz. We were running CentOS 7.2 with kernel version 3.10.0-327.22.2. We obtained high rate more than 73000 bps. The results are obtained without any attempt to optimize the implementation. The channel capacity is measured, in each case, by sending a large bit stream through the channel, timing the transfer, and counting the error bits.

6 Acknowledgements

This work was supported by Competitiveness Growth Program of the Federal Autonomous Educational Institution of Higher Professional Education National Research Nuclear University MEPhI (Moscow Engineering Physics Institute).

7 Conclusion

In this paper we proposed a new get-port-in-use attack, which is based on TCP socket mechanism and belongs to the class of operating system storage transition-based attack. The proof-of-concept exploit was presented as well. Experimental results showed high rate covert channel, that means a significant threat of confidentiality in multilevel secure operating systems.

References

[1] Swarup Chandra, Zhiqiang Lin, Ashish Kundu, and Latifur Khan. *Towards a Systematic Study of the Covert Channel Attacks in Smartphones*, pages 427–435. Springer International Publishing, Cham, 2015.

[2] Patrick R Gallagher Jr. A guide to understanding covert channel analysis of trusted systems provides a set of good. 1993.

[3] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering*, pages 1–27, 2016.

[4] C. G. Girling. Covert channels in lan's. *IEEE Transactions on Software Engineering*, SE-13(2):292–296, Feb 1987.

[5] Theodore G. Handel and Maxwell T. Sandford. *Hiding data in the OSI network model*, pages 23–38. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[6] D. Harnik, B. Pinkas, and A. Shulman-Peleg. Side channels in cloud services: Deduplication in cloud storage. *IEEE Security Privacy*, 8(6):40–47, Nov 2010.

[7] Hermine Hovhannisyan, Wen Qi, Kejie Lu, Rongwei Yang, and Jianping Wang. Whispers in the cloud storage: A novel cross-user deduplication-based covert channel design. *Peer-to-Peer Networking and Applications*, pages 1–10, 2016.

[8] Richard A. Kemmerer. Shared resource matrix methodology: An approach to identifying storage and timing channels. *ACM Trans. Comput. Syst.*, 1(3):256–277, August 1983.

[9] Butler W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613–615, October 1973.

[10] Song Li and A. Ephremides. A covert channel in mac protocols based on splitting algorithms. In *IEEE Wireless Communications and Networking Conference, 2005*, volume 2, pages 1168–1173 Vol. 2, March 2005.

[11] Steven B. Lipner. A comment on the confinement problem. *SIGOPS Oper. Syst. Rev.*, 9(5):192–196, November 1975.

[12] Aleksandra Mileva and Boris Panajotov. Covert channels in tcp/ip protocol stack - extended version-. *Central European Journal of Computer Science*, 4(2):45–66, 2014.

[13] H. Okhravi, S. Bak, and S. T. King. Design, implementation and evaluation of covert channel attacks. In *2010 IEEE International Conference on Technologies for Homeland Security (HST)*, pages 481–487, Nov 2010.

[14] Tobias Pulls. *(More) Side Channels in Cloud Storage*, pages 102–115. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

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

[16] Mickaël Salaün. Practical overview of a xen covert channel. *J. Comput. Virol.*, 6(4):317–328, November 2010.

[17] Abdulrahman Salih, Xiaoqi Ma, and Evtim Peytchev. *Implementation of Hybrid Artificial Intelligence Technique to Detect Covert Channels Attack in New Generation Internet Protocol IPv6*, pages 173–190. Springer International Publishing, Cham, 2017.

[18] Sameer Seth and M. Ajaykumar Venkatesulu. *TCP/IP Architecture, Design and Implementation in Linux*. Wiley-IEEE Computer Society Press, 2008.

[19] Shiuh-Pyng Shieh. Estimating and measuring covert channel bandwidth in multilevel secure operating systems. *Journal of information science and engineering*, 15(1):91–106, 1999.

[20] Sheng Wang, Weizhong Qiang, Hai Jin, and Jinfeng Yuan. Covertinspector: Identification of shared memory covert timing channel in multi-tenanted cloud. *International Journal of Parallel Programming*, 45(1):142–156, 2017.

[21] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 473–482, Dec 2006.

[22] Zhenghong Wang and Ruby B. Lee. New constructive approach to covert channel modeling and channel capacity estimation. In *Proceedings of the 8th International Conference on Information Security*, ISC'05, pages 498–505, Berlin, Heidelberg, 2005. Springer-Verlag.

[23] Ziqi Wang, Rui Yang, Xiao Fu, Xiaojiang Du, and Bin Luo. A shared memory based cross-vm side channel attacks in iaas cloud. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 181–186, April 2016.

[24] George Wilson, Klaus Weidner, and Loulwa Salem. *Extending Linux for Multi-Level Security*. DEStech Publications, Inc., USA, 2007.