Stealth debugging of programs in Qemu emulator with WinDbg debugger

M.A. Abakumov <mikhail.abakumov@ispras.ru>
P.M. Dovgalyuk <dovgaluk@ispras.ru>
Yaroslav-the-Wise Novgorod State University,
41, Great St. Petersburg st., Velikiy Novgorod, 173003, Russia

Abstract. When programs are analyzed for the presence of vulnerabilities and malicious code, there is a need for a quality isolation of the analysis tools. There are two reasons for this. At first, the program can influence the tool environment. This problem is solved by using the emulator. At second, the tool environment can influence behavior of the analyzed program. So, the programmer will think that the program is harmless, but in fact it is not. This problem is solved by the mechanism of stealth debugging. The WinDbg debugger has the possibility of connecting to a remote debug service (Kdsrv.exe) in the Windows kernel. Therefore, it is possible to connect to the guest system running in the QEMU emulator. Interaction between WinDbg client and server occurs through packets by protocol KDCOM. However, kernel debugging is possible only with the enabled debugging mode in boot settings. And it reveals the debugging process. We developed special module of WinDbg debugger for Qemu emulator. It is an alternative of the remote debugging service in the kernel. Thus, the debugger client tries to connect to the WinDbg server, but module intercepts all packets, generates all the necessary information from the Qemu emulator and sends response to the client. Module completely simulates the behavior of the server, so the client does not notice the spoofing and perfectly interacts with it. At the same time for debugging there is no need to enable debugging mode in the kernel. This leads to stealth debugging. Our module supports all features of WinDbg regarding remote debugging, besides interception of events and exceptions.

Keywords: WinDbg; Qemu; Windows; remote debugging; stealth debugging

DOI: 10.15514/ISPRAS-2018-30(3)-6

For citation: Abakumov M.A., Dovgalyuk P.M. Stealth debugging of programs in Qemu emulator with WinDbg debugger. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018. pp. 87-92. DOI: 10.15514/ISPRAS-2018-30(3)-6

1. Introduction

When performing a dynamic analysis of binary (executable) code, the problem arises of qualitatively isolating the code and the instrumentation on which this analysis is performed. The need for isolation is dual. On the one hand, it is necessary to limit the impact of the code being studied, since it is able to affect the state of the instrument machine, which is especially important in the study of malicious software. On the other hand, analysis tools can indirectly change the behavior of the program being studied. The most indicative are the situations when errors in working with dynamic memory and race conditions cease to appear in the debugging mode.

The search for undocumented features in a binary code encounters a similar problem. Various techniques and techniques are known [1], with the help of which malware reveals that its execution takes place in a controlled environment, and does not fulfill its objective functions. To find the debugger to be connected, check the int 3 handler and hardware debug registers, evaluate the behavior of certain API functions, and track the progress of the system time.

It is possible to divide potential sources of information, which makes it possible to identify the fact of working in a controlled environment into three disjoint groups. The first is the interaction with external, uncontrolled components, the program being studied, such as remote servers. To the same category, it is necessary to include speed checks. Successful struggle with such sources allows the mechanism of deterministic reproduction [2]. If you write the progress of the system in advance, when debugging and analyzing it during playback, there will be no effect on the guest's state because all time characteristics are fixed during recording. The second group of sources is the discrepancy in the behavior of the equipment [3]. The implementation of virtual equipment in software emulators is not always ideal. Known inaccuracies can be used to determine the emulator in which the program runs. The third group is the analysis tools present in the runtime. This kind of facility occurs even when the debugger is running in conjunction with a virtual machine.

2. Related work

In the Qemu emulator at the moment there is only a module of the GDB debugger, which allows debugging the kernel of the system, but in itself it has relatively small functionality and does not have a GUI. You can use IDA Pro Disassembler [4] ore to connect to the emulator via the GDB interface, but this will not extend the range of the GDB's features, but will only increase the ease of use. In addition, there is a utility called Winbagility [5], which allows the debugger WinDbg to connect to the kernel without debugging mode of the operating system. It is utility for the VirtualBox emulator and is the intermediary between the debugger and the emulator. There is the FDP protocol between Winbagility and the emulator - the introspection API for VirtualBox. It is a minus in this implementation, since the number of provided functions limits the interface.

3. WinDbg

The WinDbg debugger is one of the most advanced debuggers for Windows operating systems. WinDbg is claimed by developers, because it extracts symbolic information from applications and libraries, displays the contents of internal Windows data structures, performs remote debugging of a physical or virtual machine. WinDbg can be used for debugging user applications, device drivers, the operating system itself in kernel mode, to analyze memory dumps in kernel mode created after the so-called Blue Screen of Death, which occurs when an error is issued. It can also be used for debugging custom mode crash dumps. WinDbg supports several debugging modes: debugging the local process, debugging the kernel, and remote debugging.

Target applications can easily detects local debugging process. Remote debugging requires enabled debugging mode in kernel. In this mode kernel uses the debugging server (KdSrv.exe) for interacting with remotely client. But It is also reveals system control (Fig. 1).

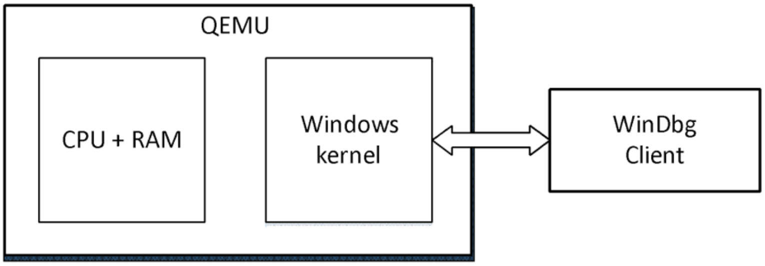

Fig. 1. Direct kernel debugging

4. Stealth debugging

We developed a mechanism for stealth debugging for the QEMU emulator, which allows WinDbg to be remotely connected. The mechanism is an analysis module built into the emulator, and turns out to be an external tool in relation to the guest system. The needs of the KdSrv service in the kernel of the debugging system is not required - the analysis module itself extracts the necessary data from the system and transfers it to the remote client debugger (Fig. 2). The programs running in the guest system cannot track the presence of the connected debugger through functions such as IsDebuggerPresent or through the state of the hardware registers.

One way to remotely kernel debugging using the WinDbg debugger is to debug through the COM port. Interaction between the computers takes place via a private KDCOM protocol, the specification of which has been restored. One of the computers in this case is represented by a virtual machine. The second is an instrumental computer with Windows OS where this machine is started. Running

WinDbg client connects to the emulator via a named pipe, through which the COM-port of the virtual machine is forwarded.

The developed module for the emulator fully implements the KDCOM protocol, within the framework of the restored specification, so the debugger WinDbg interacts with it, as with the debugging module of the Windows kernel, without noticing the substitution. It should be noted that the use of the QEMU emulator as a runtime opens the possibility of debugging during deterministic playback of the virtual machine. The recorded scenarios can be debugged many times in the emulator, which would not be possible if the Windows debug module running inside the guest system were used.

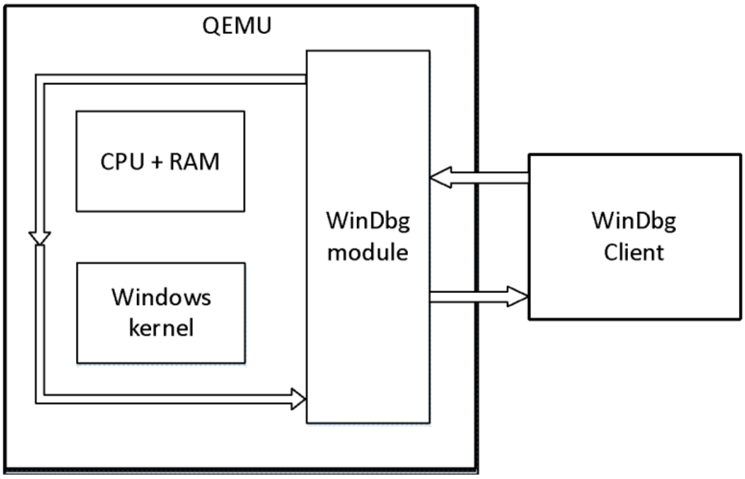

Fig. 2. Kernel debugging through the special module

5. Results and contributons

The developed module supports almost all features of WinDbg regarding remote debugging, besides interception of events and exceptions. It is open source project placed in: github.com/ispras/qemu/tree/windbg. The official community recognized the module as useful. In addition, patches have already been prepared for inclusion in the official repository.

6. Acknowledment

The work was supported by the Ministry of Education and Science of Russia, research project No. 2.6146.2017/8.9.

References

[1]. Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, 2014, pp. 447- 458.

[2]. P. Dovgalyuk. Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging. In Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering, 2012, pp. 553-556.

[3]. Roberto Paleari, Lorenzo Martignoni, Giampaolo Fresi Roglia, and Danilo Bruschi. A fistful of red-pills: how to automatically generate procedures to detect CPU emulators. In Proceedings of the 3rd USENIX conference on Offensive technologies (WOOT'09). 2009.

[4]. IDA ProDisassembler. Available at: https://www.hex-rays.com/products/ida/index.shtml, accessed 19.06.2018.

[5]. Winbagility. Available at: https://winbagility.github.io/, accessed 19.06.2018.

Скрытая отладка программ отладчиком WinDbg в эмуляторе Qemu

М.А. Абакумов <mikhail.abakumov@ispras.ru>
П.М. Довгалюк <dovgaluk@ispras.ru>
Новгородский государственный университет имени Ярослава Мудрого,
173003, Россия, Великий Новгород, Большая Санкт-Петербургская, д. 41

Аннотация. При анализе программ на наличие уязвимостей и вредоносного кода бывают ситуации, в которых возникает необходимость качественной изоляции инструментов анализа. Этому есть две причины. Во-первых, анализируемая программа может влиять на инструментальную среду. Эта проблема решается использованием эмулятора. Во-вторых, инструменты анализа могут влиять на программу. Так, программист может подумать, что программа безопасна, хотя на самом деле это может быть не так. Эта проблема может быть решена механизмом скрытой отладки. Отладчик WinDbg имеет функцию подключения к удаленному отладочному серверу (Kdsrv.exe), запущенному в ядре Windows. Поэтому есть возможность подключиться к гостевой системе, запущенной в эмуляторе QEMU. Клиент взаимодействует с сервером через пакеты по протоколу KDCOM. Однако отлаживать ядро можно лишь с включенным режимом отладки в настройках запуска, что раскрывает процесс отладки. Мы разработали специальный модуль отладчика WinDbg для QEMU, который является альтернативой удаленному отладочному сервису в ядре. Модуль перехватывает пакеты при взаимодействии клиента отладчика WinDbg с сервером, самостоятельно генерирует всю необходимую отладочную информацию, используя возможности эмулятора Qemu, и отправляет ответ клиенту. Модуль полностью эмулирует поведение отладочного сервера, поэтому клиент на замечает подмены и успешно взаимодействует с ним. При этом отпадает необходимость в отладочном режиме ядра. Тем самым происходит скрытая отладка. При использовании модуля работоспособны все возможности WinDbg, которые он представляет для удаленной отладки, кроме перехвата событий и исключений.

Ключевые слова: WinDbg; Qemu; Windows; удаленная отладка; скрытая отладка

DOI: 10.15514/ISPRAS-2018-30(3)-6

Для цитирования: Абакумов М.А., Довгалюк П.М. Скрытая отладка программ отладчиком WinDbg в эмуляторе Qemu. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 87-92 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-6

Список литературы

[1]. Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, 2014, pp. 447- 458.

[2]. P. Dovgalyuk. Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging. In Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering, 2012, pp. 553-556.

[3]. Roberto Paleari, Lorenzo Martignoni, Giampaolo Fresi Roglia, and Danilo Bruschi. A fistful of red-pills: how to automatically generate procedures to detect CPU emulators. In Proceedings of the 3rd USENIX conference on Offensive technologies (WOOT'09). 2009.

[4]. IDA Pro Disassembler. Режим доступа: https://www.hex-rays.com/products/ida/index.shtml, дата обращения 19.06.2018.

[5]. Winbagility. Режим доступа: https://winbagility.github.io/, дата обращения 19.06.2018.