POSTER: Towards Detecting DMA Malware

[Extended Abstract]

Patrick Stewin, Jean-Pierre Seifert, and Collin Mulliner
Security in Telecommunications, Technische Universität Berlin & Deutsche Telekom Laboratories
Ernst-Reuter-Platz 7
10587 Berlin, Germany
patrickx,jpseifert,collin@sec.t-labs.tu-berlin.de

ABSTRACT

Malware residing in dedicated isolated hardware containing an auxiliary processor such as present in network, video, and CPU chipsets is an emerging security threat. To attack the host system, this kind of malware uses the direct memory access (DMA) functionality. By utilizing DMA, the host system can be fully compromised bypassing any kind of kernel level protection. Traditional anti-virus software is not capable to detect this kind of malware since the auxiliary systems are completely isolated from the host CPU. In this work we present our novel method that is capable of detecting this kind of malware. To understand the properties of such malware we evaluated a prototype that attacks the host via DMA. Our prototype is executed in the chipset of an x86 architecture. We identified key properties of such malware that are crucial for our detection method. Our detection mechanism is based on monitoring the side effects of rogue DMA usage performed by the malware. We believe that our detection mechanism is general and the first step in the detection of malware in dedicated isolated hardware.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*invasive software*

General Terms

Security

Keywords

Dedicated Hardware; Intel Active Management Technology (iAMT); Manageability Engine (ME); Northbridge; Rootkit

1. DMA MALWARE

Rootkits are the most prominent type of malicious software today. The ongoing battle between malware authors and the anti-malware community forced malware creators to move to dedicated isolated hardware, or more precisely, to auxiliary processors present in today's x86 computer systems. Auxiliary CPUs are components present in hardware such as the network interface card (NIC), the video card, and the actual x86 chipset, for example, in the memory controller hub (MCH).

CCS'11, October 17–21, 2011, Chicago, Illinois, USA.
ACM 978-1-4503-0948-6/11/10.

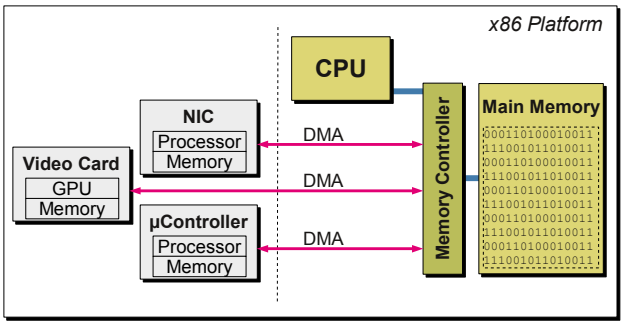

Figure 1: Overview of Dedicated Isolated Hardware potentially exploitable by Rootkits

Hardware based malware running in a NIC was shown in [1]. Efforts were started to prove the feasibility of such malware in video cards (using Graphics Processing Units, GPU) [2]. The first attacks were already presented in [8].

All these devices support Direct Memory Access (DMA), which enables this kind of malware to attack the host platform. The use of DMA circumvents all known protection mechanisms build into the operating system (OS) kernels and thus presents a very serious threat. Furthermore, malware resident on dedicated isolated systems can survive reboots and even the power off state.

Since DMA usage is a key property of malware executed in isolated hardware (see Figure 1), we call this *DMA malware*. The execution environment of DMA malware is unaccessible from the host platform's CPU. Anti-virus software therefore is unable to detect and disable it. To the best of our knowledge no previous work has presented mechanisms to detect malware resident in dedicated isolated hardware.

In this work we present a novel approach for detecting DMA malware. Our approach is based on DMA side effects that we observed during the evaluation of our own DMA malware prototype.

We developed a prototype DMA malware to resemble representative functionality used by other previously developed DMA malware. Our specific implementation targets the auxiliary processor that is part of the platform's memory MCH included in current x86 architecture chipsets.

So far we developed, implemented, and evaluated our mechanism that is able to detect rogue DMA usage that is not initiated by the host system. Our method is able to detect a general side effect pattern and thus we believe it is suited to

detect other kinds of DMA malware besides the prototype we implemented.

The main contributions of this work are:

- **Novel Detection Mechanism:** We present a novel detection mechanisms to reveal DMA malware executed in isolated hardware environments. Our work shows that DMA malware produces surprising side effects that we measure reliably utilizing widely used and cross platform available CPU features.
- **DMA Malware Evaluation:** To fully understand DMA malware we evaluated a realistic and representative prototype in form of a fully functioning keystroke logger that operates in the platform's memory controller hub. Our malware searches the host memory for keystroke codes and exfiltrates them via the network.

2. RELATED WORK

Our focus is on dedicated isolated hardware with a separate processor and separate memory as known from NICs and video cards. An example of a stealth secure shell (SSH) was shown by [7]. The shell is hidden using the NIC as well as the GPU environment, that communicate via the Peripheral Component Interconnect (PCI) bus. The SSH daemon is installed by reflashing the firmware. A defense mechanism proposed by [7] is counting PCI-to-PCI transfers. Our prototype malware does not communicate with other PCI devices (i.e., counting PCI-to-PCI transfers is useless) and does not require to reflash firmware.

In [1], Duflot et al. demonstrated how to exploit a host during runtime by utilizing a NIC and DMA. The authors concede that the internal memory of the NIC can be accessed by the host. Their attack is not completely isolated from the host, whereas our prototype malware is. The authors propose countermeasures that will most probably result in an arms-race. A detection mechanism was not proposed.

Another example for malware executed in a GPU was shown in [8]. The execution of that malware is assisted by the GPU, i.e., some parts are executed on the main CPU, thus, they are not completely isolated from the host.

Executing arbitrary code in the Intel Active Management Technology environment (see [4]), that means, in the platform's MCH, was demonstrated by [6]. The authors discovered an exploit that they use to inject code into the iAMT environment. Their proof-of-concept (PoC) code does nothing except to reveal itself by writing to a hard coded address in the main memory via DMA. The PoC has no malware functionality and is therefore not suited for our evaluation.

The attack properties of our prototype are quite similar to the presented related work. The attacks are based on DMA. Hence, the detection mechanism based on DMA side effects we present in this work is also able to detect DMA attacks originating from other devices, such as NICs or GPUs.

3. OUR DMA MALWARE PROTOTYPE

To evaluate DMA malware we extended the implementation of our keystroke logger prototype presented in [5], that is based on an exploit described in [6]. Our prototype is executed in the isolated iAMT environment that is part of every modern Intel x86 chipset. The logger uses DMA to find the physical address of the keyboard buffer of a USB keyboard that is connected to a Linux based platform. After the keystroke logger has found the address the logger monitors the buffer permanently and exfiltrates captured keystroke codes to an external platform via an isolated network channel.

We determined that DMA malware has the following common attack properties. An attack against the host consists at least of two phases: (i) search system memory for valuable data such as the address of a kernel structure (search phase), and (ii) read from (monitor phase) and/or write to the found memory addresses. Even in the case where the attacker wants to inject something into the host environment, the attacker has to find the right address such as of a certain kernel data structure in the host memory (see [1]). The search phase cannot be avoided.

For our prototype we chose a read only DMA attack to implement more stealthy DMA malware. We implemented a fully functioning keystroke logger for USB keyboards. Our prototype DMA malware implements exactly the above mentioned attack properties. We think that these are common for all DMA malware. Hence, our prototype is representative DMA malware.

We could have implemented another prototype, that extracts cryptographic keys from the systems memory. Such a prototype would also implement the search and the monitor phase, even if the latter is quite short. The important point is to create a realistic attack that we can analyze.

4. DETECTING DMA SIDE EFFECTS

Our investigation into detecting malicious DMA usage is based on the knowledge that both, the main CPU and the iAMT environment, can access the main system memory at the same time. The interesting question for us was if this parallel memory access is with or without side effects. If side effects are present and measurable then we can use these to detect malicious behavior.

We booted a Linux kernel and started only a root shell to ensure that the system workload was as little as possible. We performed a memory stress three times: without keystroke logger (baseline), keystroke logger in search mode, and keystroke logger in monitor mode. For the tests we used a 100 MB file that we copied from one location to another within a RAM based file system. We repeated the tests 1,000 times and calculated the means. The results are depicted in Figure 2. The diagram shows how we refined our strategy with different more specialized measurement tools.

GNU time: First we tried the common system tool GNU time to determine a delay. GNU time measures system resource usages of a process, in our case the memory stress test tool. As shown in Figure 2 on the left hand side the means of the test runs are nearly the same. We concluded that the measurement resolution of GNU time is not sufficient to reveal delays in our experiment.

Time Stamp Counter (TSC): We repeated our measurements with a more accurate hardware based measurement tool, the TSC (cf. [3], Vol. 3A 16-49). The TSC counts clock ticks. The results are shown in the middle in Figure 2. We were able to (re)produce an overhead of 2% when our prototype malware is in search mode. DMA was originally introduced to disburden the CPU. That means, to perform memory transfers without the involvement of the host CPU. *Hence, that overhead is surprising and a first piece of evidence that detectable DMA side effects exists.* When our prototype malware is in monitor mode we cannot see note-

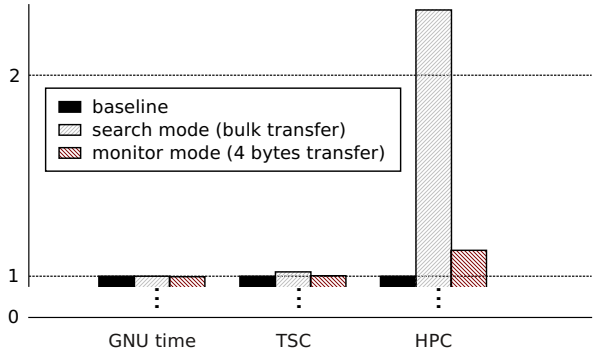

Figure 2: Memory Stress Measurements: Search/ Monitor Phase are shown relative to the Baseline

worthy overhead when using TSC. The critical difference between the two modes is that in search mode our malware copies at least a memory page where it searches for valuable data, but in monitor mode the malware copies just 4 bytes from the keyboard buffer.

Hardware Performance Counter (HPC): We repeated the measurements with a third approach using HPCs, a monitoring tool for code optimization. The counters are special processor registers on Intel processors (see [3], Vol. 3B 30) that count certain events such as cache misses, branch prediction misses, and resource stalls. Similar HPC are also available on other platforms such as ARM and SPARC. The Intel platform we used for our experiments supports 340 events.[1] We evaluated all of them and determined that especially resource stalls are a DMA side effect. HPC events are more precise than TSC measurements. We assume the counted resource stalls are a direct result of the delays we can measure with TSC. Exemplarily, we show the result of a HPC called `RAT_STALLS:ROB_READ_PORT` in Figure 2. Compared to the baseline the overhead is more than twice as much. Without our prototype malware the mean of our measurements was 1,359,898 counted events. With our prototype in search mode the mean was 3,161,868 counted events, and in monitor mode it was 1,535,054 counted events. The latter is slightly higher compared to the baseline. The refined measurements show the more accurate we measure the better is the visibility of the DMA side effect.

Detection: Based on our findings, DMA side effects can be reliably measured. This means we can design a DMA malware detection mechanism. The mechanism works by establishing a measurement baseline, reference values for the TSC/HPC. During runtime, our system monitors the TSC/HPC values and compares them to the reference values. If the values deviate from the reference values malware is detected. As this research is work in progress we acknowledge that an implementation needs some more work.

5. CONCLUSION AND FUTURE WORK

Malware executed in isolated hardware is an emerging threat. This type of malware is mostly based on accessing the host memory via DMA. In this work we aim to detect this type of malware. We first developed a malware prototype for the iAMT environment. The iAMT environment is part of every modern Intel x86 chipset. Our malware is a fully working keystroke logger that exfiltrates the captured keystrokes via the network. Through the development of our malware as well as through the analysis of existing malware we determined key properties of DMA based malware. The most interesting property is the required bulk DMA transfers to search the memory for valuable data to carry out attacks. Our detection method is based on the surprising observation that parallel memory accesses from the isolated hardware (via DMA) and the main CPU produce measurable side effects. Our work is still in progress, but we are convinced that our approach is applicable to other DMA based malware executed in similar isolated hardware environments such as NICs and video cards.

[1]We used the Performance API, that is available at `http://icl.cs.utk.edu/papi/software/index.html`, to work with HPC.

6. ACKNOWLEDGMENTS

The authors would like to thank Benjamin Michéle, Dmitry Nedospasov, Juliane Krämer, Matthias Lange, and Steffen Liebergeld for their help and support in this work.

7. REFERENCES

[1] L. Duflot, Y.-A. Perez, G. Valadon, and O. Levillain. Can you still trust your network card?, Mar. 2010. [Online:] http://www.ssi.gouv.fr/IMG/pdf/csw-trustnetworkcard.pdf.

[2] G. Hoglund. video card rootkit feasibility study, Mar. 2010. [Online:] HBGary Email Viewer: http://leaks.anonamegame.com/greg_hbgary_com/14960.html.

[3] Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3 (3A & 3B): System Programming Guide, Apr. 2011. [Online]: http://www.intel.com/Assets/PDF/manual/325384.pdf.

[4] A. Kumar, P. Goel, and Y. Saint-Hilaire. *Active Platform Management Demystified*. Richard Bowles, 2009. Intel Press.

[5] P. Stewin and J.-P. Seifert. "In God We Trust All Others We Monitor" - [Extended Abstract]. In *CCS '10: Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages p.639–641, New York, NY, USA, October 2010. ACM.

[6] A. Tereshkin and R. Wojtczuk. Introducing Ring -3 Rootkits. Black Hat USA, July 2009. [Online]: http://www.blackhat.com/presentations/bh-usa-09/TERESHKIN/BHUSA09-Tereshkin-Ring3Rootkit-SLIDES.pdf.

[7] A. Triulzi. Project Maux Mk.II, Nov. 2008. [Online]: http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-PACSEC08-Project-Maux-II.pdf.

[8] G. Vasiliadis, M. Polychronakis, and S. Ioannidis. GPU-Assisted Malware. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pages 1 –6, oct. 2010.