Forensic Science International: Digital Investigation 32 (2020) 300919

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS 2020 EU – Proceedings of the Seventh Annual DFRWS Europe

BMCLeech: Introducing Stealthy Memory Forensics to BMC

Tobias Latzo*, Julian Brost, Felix Freiling

Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany

ARTICLE INFO

Article history:

ABSTRACT

Several system management technologies have been introduced that leverage additional devices on the main board to asynchronously access and control the host's computing resources. One such prominent technology for server systems is the *Baseboard Management Controller* (BMC), a co-processors with some firmware that allows an administrator to monitor and administer a server remotely. This paper introduces BMCLeech, the first software that brings forensic memory acquisition onto the BMC which makes it very useful for incident response teams. BMCLeech is based on the open source BMC implementation OpenBMC and internally leverages the power of PCILeech, a well-known framework for memory acquisition via DMA.

1. Introduction

The forensic analysis of main memory has become an essential part of forensic investigations since it allows to derive insights that could never have been made by analyzing persistent storage alone. For example, memory analysis can be used to retrieve evidence of running processes and open network connections. Furthermore, it can be used to acquire encryption keys for network connections or encrypted containers.

There are several possibilities to perform memory acquisition, e.g., by using kernel support or modules like Lame (Sylve, 2012) or Pmem (Stüttgen and Cohen, 2013) under Linux. If the target system is virtualized, one is able to use standard tools of the hypervisor such as `dumpvmcore` in VirtualBox (OracleVirtualBox). Basically, it is also possible to virtualize the running operating system as done in HyperSleuth (Martignoni et al., 2010). However, most of these techniques require (1) to deploy the forensic software on the target system and (2) to execute that software with root privileges. But even if both conditions are met, the forensic software has to deal with anti-forensic techniques that tamper with or circumvent classical memory acquisition (Sparks and Butler, 2005; Palutke and Freiling). Memory acquisition is best when it is stealthy, i.e., it cannot be detected.

In a recent survey on forensic memory acquisition techniques, Latzo et al. (2019) observed that the level in the *memory access hierarchy* is a critical factor defining the possibility of stealthy memory acquisition. Roughly speaking, the memory access hierarchy corresponds to the privilege level on which the acquisition software is running. For Intel ×86 systems, this level is defined by the *ring* in which software is running. Latzo et al. (2019) distinguish five levels that enable increasingly deeper and more powerful memory access:

- user level (corresponding to ring 3),
- kernel level (ring 0),
- hypervisor level (ring −1),
- synchronous management level (ring −2), and
- asynchronous device level (ring −3).

It was observed that the lower the privilege ring, the more powerful the acquisition software is since it is usually possible for lower layers to access memory of higher layers but not vice versa. The ideal acquisition method is one that resides at the asynchronous device level (ring −3), is available on all systems without having to deploy software on demand, and does not terminate the execution of the software running at higher levels.

Several technologies have been introduced to leverage the privileges of the asynchronous device level. One such prominent technology for server systems is the *Baseboard Management Controller* (BMC), a co-processor with some firmware that allows an administrator to monitor and administer a server remotely, e.g., via protocols like IPMI (IntelHewlett-PackardNECDell). Prominent examples of BMCs are Dell's iDRAC (Dell Inc, 2014) or Hewlett Packard's iLO (Hewlett Packard Enterprise Company). However, we are

* Corresponding author.
E-mail addresses: tobias.latzo@fau.de (T. Latzo), julian.brost@fau.de (J. Brost), felix.freiling@cs.fau.de (F. Freiling).

https://doi.org/10.1016/j.fsidi.2020.300919

not aware that any such system implements generic techniques for forensic memory acquisition.

In this paper, we introduce BMCLeech, the first pre-installable memory acquisition software for the asynchronous device level. BMCLeech is a software that runs on BMCs and exploits the fact that BMCs are usually attached to the PCIe bus that allows *direct memory access* (DMA) to host memory. DMA through the BMC is arguably rather stealthy because BMC is a standard device in many systems and the host therefore cannot distinguish between “good” BMC activities (like server administration) and “bad” ones (taking a memory snapshot). Given that BMC commonly comes with the server motherboard, we consider BMCLeech to be equivalent to a pre-installed acquisition software that can be used by incident response teams similar to the Rekall Agent Server (Cohen, 2019).

1.1. Related work

It is well-known that BMCs come with high capabilities and so are also a valuable target for attackers Farmer (2013) revealed some critical issues and vulnerabilities in the IPMI protocol that is used to control the BMCs and maintain the server systems behind. In the same year, Rapid7 released a guide to pentest BMCs and its IPMI stacks (Rapid 7, 2013). The fact that BMCs are highly privileged and are often directly connected to the Internet — often only protected by a default password — made them ideal attack targets. A good overview of BMC security vulnerabilities and countermeasures was presented at the 2019 Open Source Firmware Conference (Altherr, 2019).

Rather astonishingly, most of today's BMCs do not use secure boot and are therefore vulnerable to bootkit attacks. This has raised some awareness by vendors and clients regarding the security of the BMC and similar co-processors. In 2018, Bloomberg published an article about how China uses a tiny chip to infiltrate U.S. companies (Robertson and Riley). While there is still no academically acknowledged evidence for this “hack”, it received a lot of attention regarding the security implications of co-processors. This was reinforced when security experts demonstrated that this kind of attack is practically possible (Hudson).

Today, leading technology companies like Microsoft, Google, Facebook, Intel, IBM and Dropbox invest a lot of effort into making the BMC open source with OpenBMC, a Linux distribution for BMCs. One target is to have complete control over the BMC because currently the BMC is controlled by the hardware manufacturer and the firmware is often an ugly, intransparent and proprietary binary blob. Dropbox recently even started to standardize the BMC hardware with the RunBMC project (Shobe and Mednick, 2019).

Memory acquisition using DMA is an established technique in memory forensics. One big advantage of DMA-based memory acquisition is that usually no software has to be installed on the host system in advance. Examples of tools that acquire memory via DMA are *inception* (Maartmann-Moe, 2018) and the PCILeech framework (Frisk). There are several devices that support PCILeech — mostly FPGAs that are connected to the host via PCIe, USB3 or IEEE 1394. Furthermore, PCILeech is very feature-rich, e.g., it can not only be used to read and write memory but also to push and pull files, inject code into the kernel or mount RAM as a file. Our new system BMCLeech is implemented as a PCILeech device and therefore inherits all the flexibility of PCILeech regarding acquisition, analysis and offensive capabilities.

Recently, Synacktiv — a French IT security company — made use of an exploit in Hewlett Packard's iLO software to install DMA acquisition software on iLO 4 (Perigaud). They also implemented their acquisition software as a PCILeech device. However, in contrast to their work, BMCLeech does not rely on an exploit but runs on OpenBMC. Furthermore, BMCLeech is faster in reading the host's memory via DMA.

1.2. Contribution

It is well-known that BMCs are very powerful and therefore a good basis for stealthy forensic memory acquisition. With BMCLeech we present a novel system that leverages all advantages of the asynchronous device level and combines them with the benefits of open source software. This shows that trustworthy forensic memory access can be easily deployed on server systems as part of normal system administration at the device level. BMCLeech is inherently OS-independent.

Our implementation of BMCLeech runs on an ASPEED AST2500 BMC (ASPEED Technology Inc, 2500). This kind of BMC is widely distributed and often used by servers that belong to the Open Compute Project (OCP) (Open Compute Project and htt). Furthermore, there are servers (like Facebook Tioga Pass (Zhao and Ning, 2018)) that already support OpenBMC (OpenBMC). Our BMCLeech implementation runs on OpenBMC utilizing a customized kernel driver that performs the DMA access to the host memory. Basically, the host operating system cannot detect whether BMC is retrieving host memory or not. While the host system can observe that there is a BMC connected to the system, this is nothing suspicious because a server usually ships with a BMC. In this sense, BMCLeech is more stealthy than specialized forensic devices (Carrier and Grand, 2004).

To summarize, the contributions of this paper are twofold:

- We introduce BMCLeech, the first software that brings forensic-readiness onto the BMC and whose memory acquisition cannot be detected by the target host system. BMCLeech is implemented as a PCILeech device and thus compatible to well-known memory forensic software, so there is no additional effort needed to analyze a system's memory. even the acquisition software of an analyst does not need to be replaced.
- We provide an evaluation that demonstrates the feasibility and practicality of BMCLeech.

1.3. Outline

In Section 2, background information is given that shall help to understand this paper. Section 3 gives insights into the implementation of BMCLeech. Later in Section 4, BMCLeech is evaluated in terms of correctness and stealthiness. Finally, a discussion and some hints towards future work can be found in Section 5.

2. Background

In this section, some background information on PCIe and standard quality criteria for memory acquisition software are given.

2.1. PCI express and DMA

PCI Express (PCIe) is a high-speed, point-to-point bus system that serves to connect several peripheral devices in modern computer systems (PCI-SIG, 2010). PCIe is known to be used for graphics cards and network cards. In contrast to conventional bus systems like PCI, PCIe is organized in a tree topology. All nodes, i.e., all peripheral PCIe devices and switches that connect further end nodes, are connected to a central unit — the *Root Complex* (RC). Another possibility is that a PCIe device is connected via a switch to the RC. The RC itself is connected to the CPU and to the memory.

Direct Memory Access (DMA) is used to copy data by a peripheral device like a graphics card or a disk from or to the host memory

without using the CPU. After the transaction is completed, the CPU can be informed that a copy operation is finished. When many bytes are copied, DMA is significantly faster than traditional *Programmed I/O*.

PCIe devices can perform DMA to allow faster I/O. For example, if a node in this point-to-point network wants to perform a memory transaction, it sends a *Transaction Layer Packet* that is routed to the RC and memory through the point-to-point network. Of course, this means that PCIe devices that are connected to the computer system must be trusted as they can perform arbitrary operations in memory.

Intel's *Virtualization Technology for Directed I/O* (VT-d) (Intel virtualization tech, 2019) comes with an *Input–Output Memory Management Unit* (IOMMU) that works similar to the common MMU used to implement virtual memory. The IOMMU translates addresses that are accessed by a device to the internal memory addresses. Thus, the host memory is no more unprotected but the memory ranges that can be accessed by a specific device can be limited. Recent research (Markettos et al., 2019) revealed that current IOMMU implementations in Windows, Linux and MacOS do not fully prevent DMA-based attacks, however.

2.2. Criteria for memory acquisition

Vömel and Freiling (2012) defined a memory snapshot to be *correct*, if the memory acquisition tool acquires the actual content of the memory. Correctness is a necessary criterion of memory acquisition software.

Due to the interleaving of memory acquisition with normal system operations, memory images might exhibit effects of system actions for which the actual cause has not been recorded. A memory snapshot is *atomic*, if such inconsistencies do not arise. The atomicity of a snapshot therefore is proportional to the time it takes to take the entire snapshot. For black box evaluation Gruhn and Freiling, (2016) therefore chose to quantify the atomicity of a snapshot by the time span between the acquisition of the first memory region and the last memory region.

Formally, a memory snapshot satisfies *integrity* if the content of memory is not changed after the time an analyst decides to take a snapshot. According to Vömel and Freiling (2012), integrity aims at quantifying the level in which the process of taking the snapshot actually changes the content of memory. Gruhn and Freiling (2016) quantified integrity by measuring the average time over all memory regions from the start of the acquisition until the time when the memory region is acquired. We, however, follow the original intention of the definition of integrity.

3. Implementation

In the following section, we want to provide insights into the implementation and architecture for BMCLeech. First, an overview over the architecture is given. Then, the components that were implemented are shortly introduced and the functionality is explained.

3.1. Architecture

Fig. 1 shows the overall architecture with all relevant components during a forensic analysis. On the right side, one can see the *Forensic Workstation*. When an analyst wants to acquire memory from the target system, he or she can retrieve the corresponding snapshot of the memory by using PCILeech. By giving PCILeech the IP address, the port and the desired memory range, PCILeech connects to the given BMCLeech device. BMCLeech is implemented as a PCILeech device. Our BMCLeech implementation currently works on ASPEED AST2500 BMC and OpenBMC. Basically, it is also possible to use another BMC. Then only the connection to the driver has to be adapted. We implemented `libaspeedxdma` library to abstract the access to kernel driver (`aspeed-xdma`). So, other applications on the BMC can access the host memory more easily, e.g., for test reasons we implemented the `getmem` application that writes the content of the desired physical address from host memory to `stdout`. In the end, the kernel driver then accesses the memory via DMA. Eventually, BMCLeech sends the retrieved memory to PCILeech on the Forensic Workstation where the actual memory dump is stored. The memory can be analyzed using common memory forensic analysis tools like Rekall (Team, 2015) or Volatility (Foundation).

3.2. BMCLeech

BMCLeech implements a PCILeech `rawtcp` device. When starting the BMCLeech daemon on the BMC, it expects the port it shall use for waiting for commands by PCILeech from the Forensic Workstation. After a connection is established, PCILeech sends a request to BMCLeech. The requests are sent as a `rawtcp_cmd` that is structured as shown in Listing 1.

```
enum rawtcp_cmd {
        STATUS,     /* is device ready? */
        MEM_READ,   /* read from memory */
        MEM_WRITE   /* write to memory  */
};

struct rawtcp_msg {
        enum rawtcp_cmd cmd;
        uint64_t addr;   /* the address */
        uint64_t cb;     /* the length  */
};
```

Listing 1: The definitions of the rawtcp_cmd enum and `rawtcp_msg` struct.

If the status is requested, BMCLeech always answers that it is ready, because everything is initialized and ready before it is listening on the port. If PCILeech sends a read request, BMCLeech utilizes the underlying library to read from host memory. The `libaspeedxdma` in turn utilizes the kernel driver to perform the actual DMA operation. Then, the result is sent to the PCILeech client on the Forensic Workstation. If PCILeech wants to write to the host memory, BMCLeech waits to receive the corresponding payload. Again, the payload is written to the host memory using the DMA kernel driver.

It is also possible to use `libaspeedxdma` without PCILeech. This allows to perform analysis or data and code injection directly on the BMC. Since data is not sent back to a Forensic Workstation, this approach will probably be much faster. Basically, running PCILeech directly on the BMC with BMCLeech is also possible but there is limited storage on the BMC.

3.3. Kernel driver

Our implementation of BMCLeech is currently implemented for ASPEED AST2500 BMCs running OpenBMC. Nevertheless,

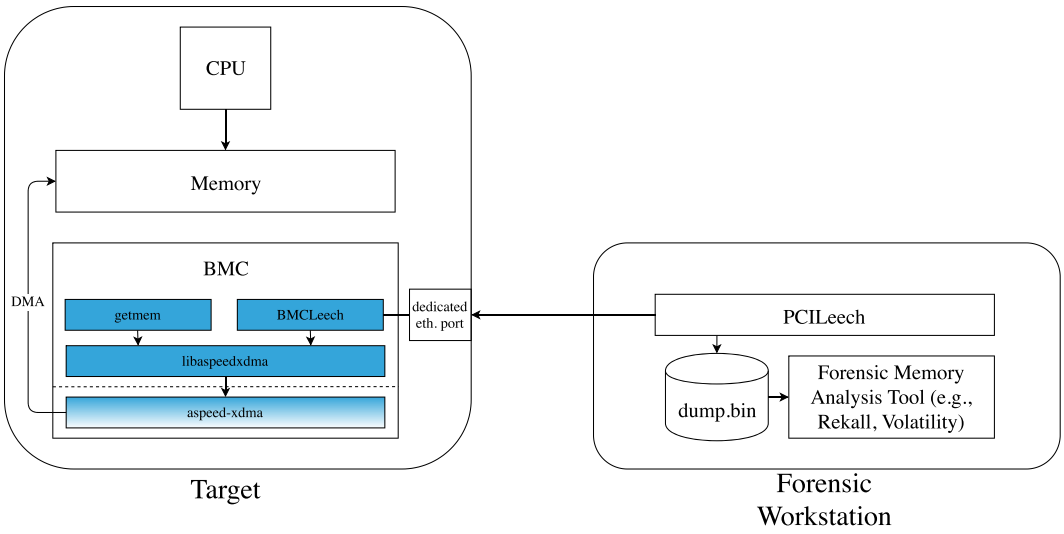

Fig. 1. The architecture of BMCLeech.

BMCLeech can easily be adopted to other DMA engines and Linux distributions. We make use of Eddie James' ASPEED AST2500 XDMA kernel module implementation (James, 2019) with small modifications. Until now James' patches are not merged to the upstream kernel yet. The driver is used to initialize the XDMA hardware as well as performing the actual DMA read and write operations. Basically, the driver provides a device in `/dev/aspeed-xdma` that is used by `libaspeedxdma` to communicate with the kernel module.

4. Evaluation

As described in Section 2.2, Gruhn and Freiling provided a framework to evaluate memory acquisition tools in terms of *correctness*, *atomicity* and *integrity* (Gruhn and Freiling, 2016). Since DMA-based memory acquisition showed a startlingly low level of atomicity (compared for example to software-based approaches) and because BMCLeech is based on DMA, we did not reevaluate atomicity. But since BMCLeech does not change memory during the acquisition at all, it fully preserves integrity according to the textual definition of integrity by Vömel and Freiling (2012). In our evaluation we therefore focus on the *correctness* property (Vömel and Freiling, 2012), therefore demonstrating (1) BMCLeech's feasibility and practicality and (2) the compatibility to PCILeech. The evaluation exhibits how memory changes over time without load and how much may happen in RAM during a memory acquisition.

4.1. Methodology

For the evaluation we compare the memory acquired using BMCLeech and a software-based technique that also acquires the physical memory and does not halt the system. Since our target is a Linux system, we consider *LiME* (Sylve, 2012) to be a reasonable tool that serves as a kind of *ground truth*. Since LiME operates from the kernel level and does not halt the system, the atomicity and integrity are limited and can probably be compared with the Windows kernel level software acquisition tools that behave all similar in Gruhn and Freiling's evaluation Gruhn and Freiling, (2016).

We start the acquisition process with BMCLeech and LiME five times simultaneously. Fig. 2 shows the timeline of memory acquisition. Basically, using BMCLeech takes significantly longer than LiME.

For the analysis, we calculate and visualize the differences (diffs) of several snapshots. We calculate the diffs per 4 KiB page and per byte. For practical reasons, only the page−wise diffs are visualized. The Δt of the times when the snapshots are taken is fixed. This is the time it took to acquire the memory using BMCLeech rounded up to the next full minute.

The following diffs are calculated:

1. The diffs between sequentially following BMCLeech dumps: $\mathrm{diff}(b_0, b_1)$, $\mathrm{diff}(b_1, b_2)$, $\mathrm{diff}(b_2, b_3)$, $\mathrm{diff}(b_3, b_4)$.
2. The diffs between sequentially following LiME dumps: $\mathrm{diff}(l_0, l_1)$, $\mathrm{diff}(l_1, l_2)$, $\mathrm{diff}(l_2, l_3)$, $\mathrm{diff}(l_3, l_4)$.
3. The diffs between the dumps acquired by BMCLeech and LiME simultaneously: $\mathrm{diff}(b_0, l_0)$, $\mathrm{diff}(b_1, l_1)$, $\mathrm{diff}(b_2, l_2)$, $\mathrm{diff}(b_3, l_3)$, $\mathrm{diff}(b_4, l_4)$.
4. The diffs between the dumps acquired by BMCLeech and LiME shifted by 1: $\mathrm{diff}(b_0, l_1)$, $\mathrm{diff}(b_1, l_2)$, $\mathrm{diff}(b_2, l_3)$, $\mathrm{diff}(b_3, l_4)$. Since the acquisition of the BMCLeech dumps takes significant longer, we also compare the dumps of a BMCLeech dump and the following LiME dump because the time when b_x is finished is nearer to the start of l_{x+1} than to l_x.

4.2. Hardware setup

Our implementation of BMCLeech runs on OpenBMC and the ASPEED AST2500 BMC which is common in the OCP market. However, it is quite hard to find such a system that supports OpenBMC and is not costly. Thus, for the evaluation, we kindly were granted remote access to a Facebook *Tioga Pass* (Zhao and Ning, 2018) server that was hosted by the OCP Solution Provider Circle B.[1] The server comes with an Intel Xeon Gold 6130 CPU with 2.10 GHz (16 cores) and 32 GiB of RAM running Ubuntu Linux 18.04.2 LTS with kernel 4.18.0−16.

During the implementation and evaluation on this system, we found out that we cannot access memory above 4 GiB. Since the AST2500 XDMA engine supports 64 bit addressing of the host memory, we assume that this is due an internal PCIe connection. For this reason, we restricted the system to use only the lowest 4

[1] https://circleb.eu/.

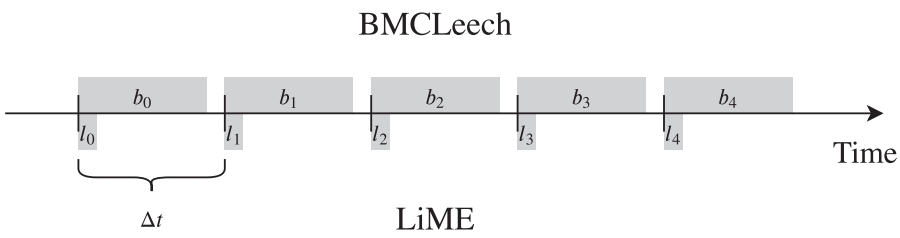

Fig. 2. The timeline of memory acquisition processes with the corresponding starts of the acquisition processes of BMCLeech b_0 to b_4 and of LiME l_0 to l_4.

GiB of its memory via the kernel command line. As a result, the effective memory size is now about 1.6 GiB, i.e., 1731760128 bytes and 422793 pages (4 KiB). These are divided into four memory ranges (see Table 1) that map to physical memory.

Using BMCLeech, the acquisition process over the Internet takes about 6 min. This corresponds to about 5 MiB/s. Local tests on the BMC revealed that the BMC is able read the memory with about 50 MiB/s. However, we were not able to exfiltrate the data with that speed over the Ethernet port. We were connected using the 100 Mbit Network Controller Sideband Interface (NC–SI) port which is probably the bottleneck in this case, i.e., the interconnection to that port. Basically, it should be possible to get considerably better transfer rates using a 1 Gbit port. However, in our case OpenBMC did not work with the dedicated 1 Gbit port. LiME acquires the memory in about 17 s. To get more comparable results, there was no bigger interaction or load on the target system during the acquisition processes.

4.3. Correctness

Vömel and Freiling, (2012) define a snapshot to be *correct* if the actual memory values are acquired when the snapshot was taken. Basically, correctness can be taken as guaranteed (Gruhn and Freiling, 2016). Since we introduce a new acquisition tool, we think we have to show that it works properly, though.

Since DMA memory acquisition works asynchronously and we perform black box evaluation on a native system, it is not possible to determine what the actual content of the memory was when it was read. However, we can compare the BMCLeech snapshots with the LiME snapshots and look if the diffs are reasonable. Additionally, we test several PCILeech payloads to demonstrate BMCLeech's compatibility.

4.3.1. Quantitative analysis

Fig. 3 shows the results of the diffs that are described in Section 4.1. The byte-wise and the page–wise diffs are printed above each page–wise visualized diffs. Note, the visualized diff shows the four memory ranges that are shown in Table 1 as a continuous memory region. Green pixels in the visualized diffs constitute that the pages have the same content while red pixels constitute that the corresponding pages differ. To visualize the memory in a reasonable aspect ratio, we introduced some padding pages at the end of the memory dump in grey. The memory address is growing from the top left to the bottom right.

First, only looking at the relative differences, one can see that the page–wise diffs are overall bigger than the byte-wise diffs. A single byte difference results in the whole corresponding page to be different. Our analysis revealed that on the average about 1200 bytes differ per differing page. However, the average value is not very meaningful in this case. In Fig. 4 one can see the distribution of different bytes per differing page. The horizontal axis shows the number of different bytes while the vertical axis shows the number of pages where x bytes differ. One insight is, that there are many pages that differ only by a small number of bytes.

Basically, all diffs in Fig. 3 appear to be quite similar. This can already be regarded as a first indicator, that BMCLeech works as intended. Furthermore, the most interesting rows, i.e., row two and three — the rows where the BMCLeech snapshots are compared with the LiME snapshots — do not show any major differences between the snapshots. The byte-wise diffs reach a maximum of about 2.5 %. Considering the fact that the acquisition with BMCLeech takes significantly longer than with LiME, these values seem to be sound. Besides that, the diffs between the LiME snapshots (in the last row) appear to be similar to the diffs between BMCLeech and LiME. As expected, the diffs of the snapshots that are started simultaneously (second row) have more differences in the higher addresses and the diffs of the shifted snapshots in row three have more differences in the lower addresses. This is because both tools — LiME and BMCLeech — acquire memory from the lowest to the highest address. The time when a high address of b_x is acquired, is nearer to the time when this address is acquired by l_{x+1} than by l_x (see also Fig. 2). Comparing the diffs of the BMCLeech snapshots, one can see that these come overall with the biggest differences. This is also an expected result since the time between the first byte of b_x and the last byte of b_{x+1} is the longest in our evaluation. However, the extent of differences of theses diffs is comparable.

As a result, what one can see between in the diffs can be regarded as the distinctive volatility of memory of the system within Δt. There is no evidence in the diffs that militates against the correctness of BMCLeech.

What is also striking in Fig. 3 is that it seems there are memory regions that are more volatile than others. Especially, there is a conspicuous red bar in the lower area of every diff. There are also other areas that appear to be more red or green, e.g., in the upper of the diffs is a bigger green area without any red pixels. We want to just mention these observations but do not analyze these further.

Table 1
The memory ranges in our evaluation environment.

Start	End	Size
0x1000	0x9ffff	636 KiB
0x100000	0x66cf9fff	1683432 KiB
0x68dfa000	0x68ff7fff	2040 KiB
0x6f30e000	0x6f7fffff	5064 KiB

4.3.2. PCILeech payloads

The comparison of BMCLeech with LiME showed that the snapshots are very similar. As BMCLeech acts as a device for PCILeech, one can not only acquire memory but can also *write* to the memory. PCILeech comes already with a bunch of payloads. However, we are aware that writing to guest memory can basically not be considered as forensically sound. Nevertheless, we also evaluate the writing capabilities. In total, we used for the evaluation the following PCILeech features:

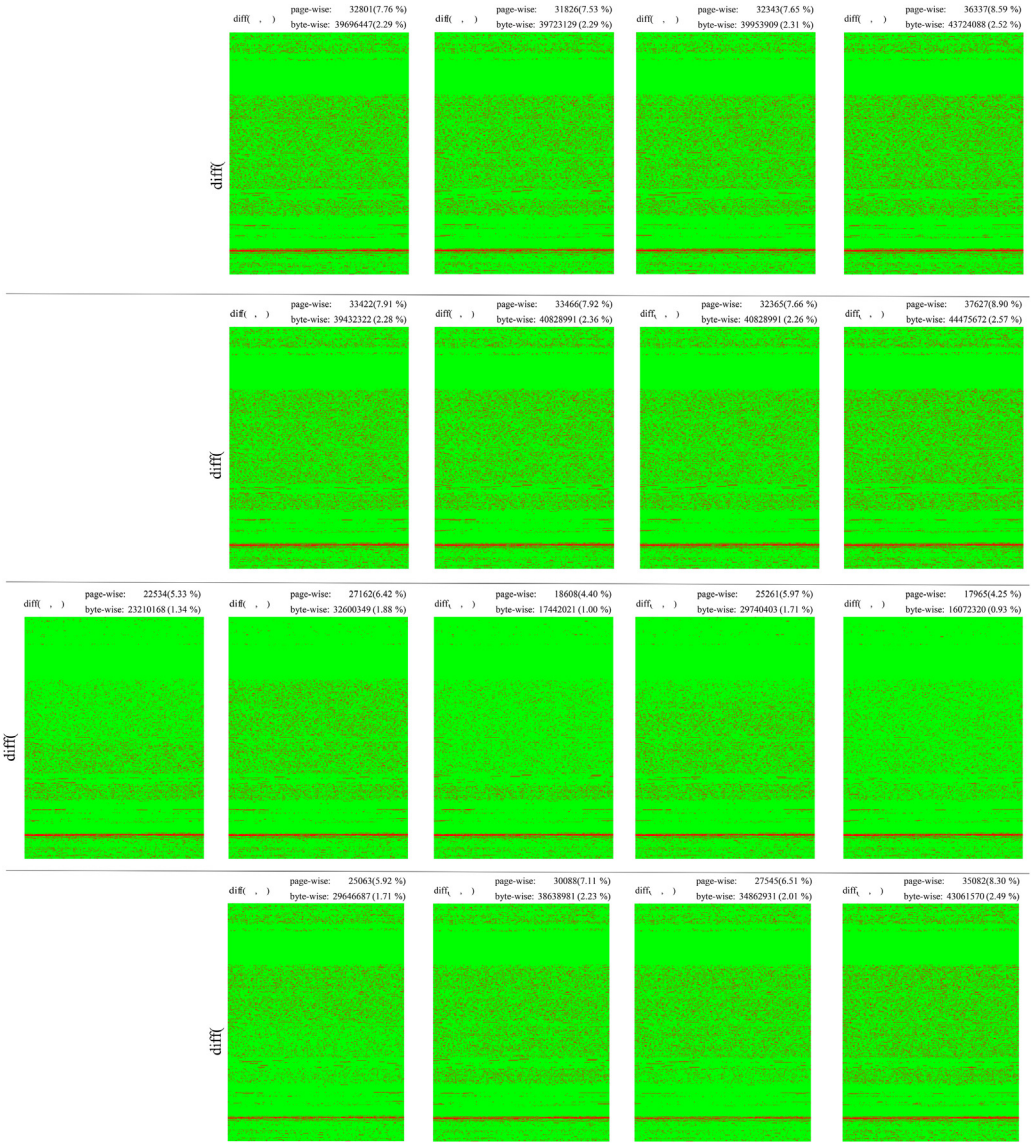

Fig. 3. Visualization of the diffs of the memory snapshots taken by BMCLeech and LiME.

- *Memory snapshot*

The first feature is the basic memory acquisition feature `dump`. As parameter, `dump` expects the corresponding memory range. All our snapshots are made with this feature. Furthermore, we opened a new text file in Vim and wrote some random content into it. The corresponding content could be found in the memory dump using `strings`.

- *Kernel module injection*

PCILeech also supports writing to memory. This allows to inject a kernel module into the host operating system via `kmdload`. PCILeech first searches for the Linux kernel base and is then able to inject the given kernel module. We injected a PCILeech kernel module that allows easier to perform further analysis. Finally, the address of the kernel module is communicated to the analyst. Note that this kernel module can also be used to acquire memory that cannot be accessed by DMA, e.g., in our case when only the lower 4*GiB* can be accessed.

- *File retrieval*

This payload relies on the kernel module injection above. The injected kernel module is now used to pull files from the target system. For this, one needs the address where the PCILeech kernel module is loaded. Then, one can simply download the desired file using `lx64_filepull`. For the evaluation, we downloaded `/etc/passwd` which was downloaded correctly.

5. Conclusion and future work

We presented *BMCLeech*, a tool that brings forensic-readiness features into the BMC. The compatibility with the PCILeech framework with all its features makes it a valuable tool for forensic investigations. For instance, it is possible to acquire the host's memory, inject code into the kernel or pull files from the host's file system. In this paper, we gave insights into the implementation of BMCLeech and showed what steps are necessary to work with PCILeech. The evaluation revealed that this approach is valuable and practical. For this, we calculated diffs between a common

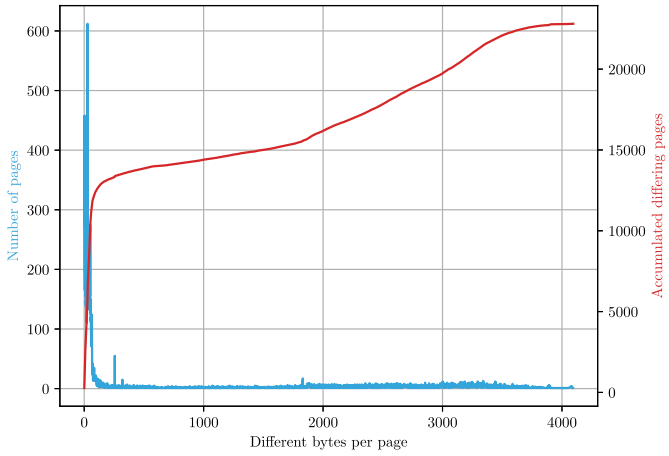

Fig. 4. Distribution of the different bytes per page.

software-based memory acquisition tool (LiME) and compared and interpreted the results quantitatively and visually.

However, the evaluation also revealed some limitations of our implementation of BMCLeech. First, on the tested platform, we were only able to acquire memory in the 32 bit memory range, i.e., below 4 GiB. This is not a limitation of the used BMC itself as the AST2500 supports 64 bit DMA access but probably the interconnection of the BMC in the system. Future work should check which platforms are affected and how it is possible to access the whole memory.

Another insight is, that the performance (and therefore the atomicity and the formal notion of integrity (Vömel and Freiling, 2012)) of BMCLeech should be improved. While we were able to read the memory with higher speeds on the BMC ($\approx$ 50 MiB/s), we were not able to send the dump with that speed over the network.

All DMA-based acquisition tools require that the IOMMU, a feature of Intel's *Virtualization Technology for Directed I/O* (VT-d), does not block the access to the RAM. However, Marketto et al. (Markettos et al., 2019) also showed that setting up the IOMMU is difficult and layers communicating with a (malicious) device are usually not hardened.

In our scenario, where we do not perform attacks but install BMCLeech to make the BMC forensic-ready, we can assume that if an IOMMU is used it is configured in a way that BMCLeech works properly. BMCLeech also demonstrates the high capabilities of internal (and also external) devices that are connected to a computer system. Often these devices are connected via PCIe and so have DMA access. In our use case, the BMC is intended to be used in a benign fashion by incident response teams in a company. However, one should also be aware that such components can also be exploited for malicious purposes.

Acknowledgments

We wish to thank Circle B (https://circleb.eu/) for providing the used hardware and support during this project. This research was supported by the Federal Ministry of Education and Research, Germany, as part of the BMBF DINGfest project (https://dingfest.ur.de) and by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) as part of the Research and Training Group 2475 “Cybercrime and Forensic Computing” (grant number 393541319/GRK2475/1–2019).

References

Altherr, R., 2019. Common BMC Vulnerabilities and How to Avoid Repeating Them. Open Source Firmware Conference.
ASPEED Technology Inc. AST2500. https://www.aspeedtech.com/products.php?fPath=20&rId=440.
Carrier, B.D., Grand, J., 2004. A hardware-based memory acquisition procedure for digital investigations. Digit. Invest. 1 (1), 50–60. https://doi.org/10.1016/j.diin.2003.12.001. URL.
Cohen, M., 2019. Rekall agent user manual. http://www.rekall-forensic.com/documentation-1/rekall-documentation/user-manual.
Dell Inc, 2014. Integrated Dell remote access controller, 8 (iDRAC8).
Farmer, D., 2013. Sold down the river. http://fish2.com/ipmi/river.pdf.
Foundation, V.. Volatility framework - volatile memory extraction utility framework. https://github.com/volatilityfoundation/volatility.
Frisk, U.. Pcileech. https://github.com/ufrisk/pcileech.
Gruhn, M., Freiling, F.C., 2016. Evaluating atomicity, and integrity of correct memory acquisition methods. Digit. Invest. 16, S1–S10.
Hewlett Packard Enterprise Company. HPE integrated lights out (iLO). https://www.hpe.com/de/de/servers/integrated-lights-out-ilo.html.
Hudson, T.. Modchips of the state, chaos communication congress. https://trmm.net/Modchips.
Intel Virtualization Technology for Directed I/o - Architecture Specification, 2019. https://software.intel.com/sites/default/files/managed/c5/15/vt-directed-io-spec.pdf.
Intel, Hewlett-Packard, NEC, Dell, Ipmi Specification v2.0.
James, E., 2019. Add aspeed XDMA engine driver. On Linux Kernel Mailing List. https://lkml.org/lkml/2019/7/1/748.
Latzo, T., Palutke, R., Freiling, F., 2019. A universal taxonomy and survey of forensic memory acquisition techniques. Digit. Invest. 28, 56–69.
Maartmann-Moe, C., 2018. Inception. https://github.com/carmaa/inception.
Markettos, A.T., Rothwell, C., Gutstein, B.F., Pearce, A., Neumann, P.G., Moore, S.W., Watson, R.N.M., 2019. Thunderclap: exploring vulnerabilities in operating system IOMMU protection via DMA from untrustworthy peripherals. February 24-27, 2019. In: 26th Annual Network and Distributed System Security Symposium, NDSS 2019. The Internet Society, San Diego, California, USA. URL. https://www.ndss-symposium.org/ndss2019/.
Martignoni, L., Fattori, A., Paleari, R., Cavallaro, L., 2010. Live and trustworthy forensic analysis of commodity production systems. In: RAID. Springer, pp. 297–316.
Open Compute project. https://www.opencompute.org/.
OpenBMC. Defining a standard baseboard management controller firmware stack. https://www.openbmc.org/.
Oracle, VirtualBox. https://www.virtualbox.org/.
R. Palutke, F. Freiling, Styx: countering robust memory acquisition, Digit. Invest. 24.
PCI-SIG, 2010. PCI Express Base Specification Revision 3.0.
Perigaud, F.. Using your BMC as a DMA device: plugging PCILeech to HPE iLO 4. https://www.synacktiv.com/posts/exploit/using-your-bmc-as-a-dma-device-plugging-pcileech-to-hpe-ilo-4.html.
Rapid 7, 2013. Sold down the river. https://blog.rapid7.com/2013/07/02/a-penetration-testers-guide-to-ipmi/.
J. Robertson, M. Riley, The big hack: how China used a tiny chip to infiltrate us companies, Bloomberg Businessweek 4.
Shobe, E., Mednick, J., 2019. RunBMC: OCP hardware spec solves data center BMC pain points. https://blogs.dropbox.com/tech/2019/08/runbmc-ocp-hardware-spec-solves-data-center-bmc-pain-points/.
Sparks, S., Butler, J., 2005. Shadow walker: raising the bar for rootkit detection. Black Hat Japan 11 (63), 504–533.
Stüttgen, J., Cohen, M., 2013. Anti-forensic resilient memory acquisition. Digit. Invest. 10, S105–S115.
Sylve, J., 2012. LiME. https://github.com/504ensicsLabs/LiME.
Team, R., 2015. Rekall memory forensic framework: about the rekall memory forensic framework. http://www.rekall-forensic.com/about.html.
Vömel, S., Freiling, F.C., 2012. Correctness, atomicity, and integrity: defining criteria for forensically-sound memory acquisition. Digit. Invest. 9 (2), 125–137.
Zhao, W., Ning, J., 2018. Facebook 2S Server Tioga Pass Rev 1.0. https://www.opencompute.org/documents/facebook-2s-server-tioga-pass-specification.