




# Prying CoW: Inferring Secrets across Virtual Machine Boundaries

Gerald Palfinger<sup>1</sup><sup>a</sup>, Bernd Prünster<sup>2</sup><sup>b</sup> and Dominik Ziegler<sup>3</sup><sup>c</sup>

<sup>1</sup>A-SIT Secure Information Technology Center Austria, Seidlgasse 22 / Top 9, 1030 Vienna, Austria

<sup>2</sup>Institute of Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria

<sup>3</sup>Know Center GmbH, Inffeldgasse 13, 8010 Graz, Austria

**Keywords:** Deduplication, Side Channel, Cloud, File System, Copy-on-Write, CoW, ZFS, Storage, Virtual Private Server, Virtual Machine.

**Abstract:** By exploiting a side channel created by *Copy-on-Write* (CoW) operations of modern file systems, we present a novel attack which allows for detecting files in a shared cloud environment across virtual machine boundaries. In particular, we measure deduplication operation timings in order to probe for existing files of neighbouring virtual machines in a shared file system pool. As a result, no assumptions about the underlying hardware and no network access are necessary. To evaluate the real-world implications, we successfully demonstrate the feasibility of our attack on the ZFS file system. Our results clearly show that the presented attack enables the detection of vulnerable software or operating systems in a victim's virtual machine on the same file system pool with high accuracy. Furthermore, we discuss several potential countermeasures and their implications.

## 1 INTRODUCTION


Data deduplication is an efficient way to share a limited set of resources, like storage or memory, among several parties. It allows for eliminating redundant or repeating data by carefully analysing chunks of information. Once redundant parts have been identified, they are replaced with reference to the unique chunk of data. In the event that referenced parts are modified, techniques such as CoW ensure that affected chunks are replicated again. In other words, data deduplication ensures that only one instance of information needs to be retained, effectively eliminating the need to store the same data multiple times.


In recent years, there has been an increasing interest in data deduplication in environments where resources are limited or shared, like in multi-user or *Virtual Private Server* (VPS) systems. For example, a significant challenge in virtualised environments is how to cope with storage space, typically associated with a large number of virtual machines. In fact, virtual machine images can quickly grow to several


gigabytes in size. Notwithstanding, virtual machines usually have large overlapping areas due to similar software or operating systems. Thus, deduplicating data can help to improve the efficiency of the system, thus reducing the operating costs.

This issue has been acknowledged in academia as well. For instance, Jin and Miller (2009) have demonstrated that "deduplication of virtual machine images can save 80% or more of the space required to store the operating system". Furthermore, file systems specifically designed for deduplication of virtual machine images, like LiveDFS (Ng et al., 2011) or Liquid (Zhao et al., 2014), clearly demonstrate the demand for storage deduplication in cloud environments.

Although deduplication comes with clear storage space advantages, it can be susceptible to memory disclosure attacks. Consequently, information about systems can be leaked across security boundaries. For instance, Suzaki et al. (2011) have shown that deduplication of the main memory can be used to perform side-channel attacks, disclosing memory contents to an attacker. The authors have successfully shown how an adversary can measure the write access times in one virtual machine to detect applications or downloaded files in another virtual machine. This

<sup>a</sup>  <https://orcid.org/0000-0001-6633-858X>

<sup>b</sup>  <https://orcid.org/0000-0001-7902-0087>

<sup>c</sup>  <https://orcid.org/0000-0002-5930-8216>

attack effectively breaks one of the core security assumptions of cloud computing, i.e., that virtual machines are confided environments which separate different users' confidential data from each other. While countermeasures like restricting access or obfuscation exist, deploying them generally decreases the performance or efficiency of a system. Thus, they are typically not acceptable in VPS or shared cloud environments where resources need to be used efficiently.

In this paper, we present a novel attack that builds upon this property, which allows for breaking out of the strict sandbox environment of virtual machines. We discuss our contribution in detail in the following section.

## 1.1 Contribution

We present a novel attack to determine the existence of confidential files in a shared file system pool, by relying on a side channel created by data deduplication techniques of modern file systems. In a nutshell, our contribution is threefold:

1. We successfully demonstrate for the first time how write operation times of modern file systems for virtual machine storage can leak critical information about chunks of data beyond the context of a virtual machine. We showcase how this exploit can be used to detect the existence of files in a shared file system pool. In particular, our attack is accurate enough that it enables an attacker to probe for the installation of vulnerable software or operating systems on neighbouring virtual machines. Still, our approach does not make any assumptions about the underlying host CPU or the victim and does not rely on network access.
2. We evaluate the real-world implications of our attack on the ZFS file system. ZFS is one of the most mature file systems which support data deduplication. We evaluate ZFS regarding three metrics. First, by detecting any alignment differences between the probed file on the virtual machine's internal file system and the underlying ZFS storage pool. Second, by determining the operating systems of adjacent virtual machines. Third, by detecting specific applications and their major and minor version. Our evaluation shows that we were able to successfully detect block offsets between the files inside the victim's virtual machine and the ZFS pool, as well as the used operating system. Furthermore, we were able to detect specific versions of an application in a victim's virtual machine with high accuracy.

3. We propose several countermeasures to cope with the presented attack. The majority of solutions, however, either limits the efficacy of data deduplication or requires changes inside the victim's virtual machine. This fact again underlines the severe impact of our findings.

## 1.2 Outline

The paper proceeds by providing background information about deduplication in Section 2. Section 3 discusses related work and puts our contribution into context. Afterwards, Section 4 covers our results and shows the feasibility of our approach. We evaluate the practicability of the attack on ZFS, which is one of the most advanced file systems to support data deduplication. Section 5 continues by discussing the limitations of this attack and possible countermeasures. Finally, Section 6 concludes this work and provides an outlook on future work.

## 2 BACKGROUND

This section provides background information about the deduplication support in modern file systems and describes different types of deduplication. Furthermore, we take a closer look at the ZFS file system, which is one of the file systems supporting deduplication.

### 2.1 Deduplication Support

Modern file systems such as ZFS<sup>1</sup> and btrfs<sup>2</sup> offer an extended range of functionality compared to conventional file systems. Among a plethora of new features, one of the main selling points of these file systems is the support for *Copy-on-Write* (CoW). By supporting CoW, these file systems can reference individual blocks of data multiple times. This technology enables various features, such as the support for deduplication of files with (partially) the same content. Further advantages are, for instance, the support for snapshots, which enable the user to create images of whole partitions without doubling the storage space. Developers of more traditional file systems have also recognised the advantage of CoW — for example, the XFS file system received support for CoW with the Linux kernel version 4.9 and thus

<sup>1</sup><https://docs.oracle.com/cd/E19253-01/819-5461/zfs-over-2/index.html>

<sup>2</sup><https://btrfs.wiki.kernel.org/>

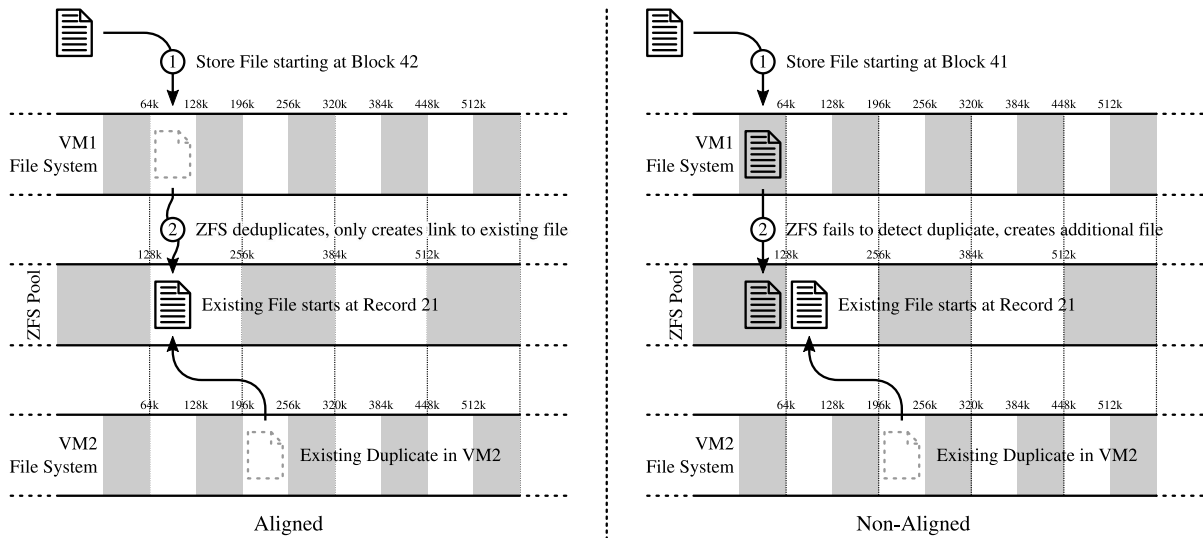


Figure 1: Effect of (mis-)alignment when operating with different block sizes.

also supports deduplication<sup>3</sup>.

Deduplication support can be implemented on three different levels — either on a file, block, or byte level. When implemented on the file level, files are only deduplicated if the exact same file already exists on the volume. While this approach may be applicable to individual files, it does not perform well with large files such as virtual machine images, as these generally only have partial overlap. When deduplication is performed on the block or byte level, it is possible to deduplicate files with only partially matching contents. While deduplication on the byte level achieves the highest deduplication ratio due to its fine granularity, it also has the highest overhead, as metadata, such as the reference to each byte, have to be stored for each use of this byte individually. Hence, most file systems operate on blocks of data. These provide a sweet spot between the metadata overhead and the granularity of the blocks, which indirectly defines the deduplication efficacy.

Some file systems allow tweaking the size of the blocks for different use cases. As virtual machine images are generally stored in a single monolithic file, tuning the block size is essential to achieve a high level of deduplication. For instance, the ext4 file system, which is the standard file system for many Linux distributions, works with a default block size of 4 KiB<sup>4</sup>. Furthermore, the file formats for virtual machine images may also operate on blocks of data,

<sup>3</sup><https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=35a891be96f1f8e1227e6ad3ca827b8a08ce47ea>

<sup>4</sup>[https://ext4.wiki.kernel.org/index.php/Ext4\\_Disk\\_Lay\\_out](https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Lay_out)

influencing the deduplication efficiency to a lesser degree. For example, one of the default file formats in QEMU operates on blocks of with a size of 64 KiB<sup>5</sup>. To achieve a high level of deduplication, aligning the different block sizes is recommended. Otherwise, even if exactly the same file is stored on two different virtual machines, it may not be deduplicated by the file system. An overview of this issue is illustrated in Figure 1.

Let us assume that the file system inside the virtual machine uses blocks with a size of 64 KiB, while the file system storing the virtual machine images is operating on blocks of 128 KiB. In this case, the replicated file is only deduplicated if it is aligned to the same part of a block. Particularly, if the beginning of both files is to be stored at the beginning of a file system block, or the 64 KiB mark, the files will be deduplicated as expected. However, if the existing file is stored at the beginning of the block but the new file is to be stored at the 64 KiB mark, none of the blocks will be exactly the same. As the blocks do not align in this case, none of the two files will be deduplicated.

In general, there are two different approaches for supporting deduplication that differ in the point of time in which deduplication is executed. The following subsection will detail the differences between these two types of deduplication.

<sup>5</sup><https://blogs.igalia.com/berto/2017/02/08/qemu-and-the-qcow2-metadata-checks/>

## 2.2 In-line- and Post-process Deduplication

With in-line or in-band deduplication, the deduplication process is performed directly when the data are written. Therefore, data that already exist in this form on the medium are immediately referenced to the existing data blocks and do not have to be written to the medium first. In order to avoid unnecessary slowdowns of the writing process, fingerprints of the blocks already present on the data storage medium are generally cached in the main memory during online deduplication. These fingerprints are compared with the fingerprint of the data to be written. However, this cache increases main memory consumption as the amount of data on the volume increases.

With post-process deduplication, on the other hand, the data are first written to the medium, similar to conventional file systems. Only at a later point in time are the data compared and duplicates are removed. This process can usually be started manually by the user or automatically at regular intervals. This approach avoids the main memory overhead of in-line-deduplication but increases the amount of storage required until the deduplication is executed. Additionally, it also increases the amount of data that have to be written to disk, producing more wear on the storage devices. The ZFS file system supports in-line deduplication, while btrfs and XFS rely on post-process deduplication.

In summary, in-line deduplication provides immediate storage space reductions, while post-process deduplication only consumes main memory while it is executed. In the next subsection, we will take a look at ZFS, which is the most mature file system to support deduplication.

## 2.3 The ZFS File System

ZFS operates on storage pools, generally referred to as *zpool*s. Inside these storage pools, datasets can be created. For these datasets, options such as deduplication or compression can be configured individually. Even if two identical files are stored on different datasets with activated deduplication, the data will still be deduplicated across these different datasets as long as these datasets reside on the same *zpool*<sup>6</sup>. In general, ZFS utilises block-level deduplication. These blocks are referred to as *records*. The maximum size of these records can be configured through the *recordsize* parameter. By default,

<sup>6</sup><https://www.fujitsu.com/global/Images/ZFS%20Implementation%20and%20Operations%20Guide.pdf>

these records have a size of 128 KiB. A peculiarity of ZFS is that records can also be smaller than the defined *recordsize*. This property is used to store small files more efficiently. However, as we deal with monolithic virtual machine images in this paper, this property does not influence any of our experiments.

When committing data to the file system, ZFS computes a hash value of the written records and saves it alongside other metadata to the disk. This value is generated regardless of the usage of deduplication as it is also used, for example, to verify the integrity of stored records. When deduplication is activated, the hashes and other metadata required to detect replicated records are kept in the *DeDuplication Table* (DDT), which is stored inside ZFS's *Adaptive Replacement Cache* (ARC). The ARC generally resides in the main memory of the system, which may be supplemented by separate cache devices. If deduplication is activated, ZFS compares the computed hash value with the hashes inside the DDT. If the hash exists in the DDT, the reference count for this record is increased, and the write is finished. Therefore, the record does not have to be written to disk. If, on the other hand, the hash cannot be found in the DDT, the record has to be committed to the underlying storage media. Hence, it is reasonable to assume that the writing process will be finished earlier if the file already exists on disk. Alternatively, when writing to an existing file that has been deduplicated, ZFS has to remove the reference to the existing record, reserve new blocks on the file system, possibly copy the existing data to these new blocks, and subsequently commit the new data. In this case, we assume that overwriting deduplicated records will take longer than overwriting non-deduplicated records. Before we put these assumptions to the test in Section 4, we will discuss related work in the following section.

## 3 RELATED WORK

The side channels resulting from memory deduplication have been the source for a variety of vulnerabilities. In general, these fall into two categories:

**Memory-based** attacks, where RAM deduplication or cache timings are exploited.

**Storage-based** attacks, utilising persistent data deduplication, as we do in our work.

While the former has received wide-spread attention due to the wide applicability of *flush+reload* (Yarom and Falkner, 2014) or *prime+probe* (Tromer et al., 2010) attacks, the latter has not been applied in the



context of multi-tenant cloud computing. Rather, attacks based on the deduplication features of persistent data has mostly focused on cloud storage providers like *Dropbox*<sup>7</sup>, for example.

This section first delves into memory-based attacks and summarises their potential compared to our work. Afterwards, we discuss previous storage deduplication attacks and argue why our contribution presents a significant advancement over existing solutions.

### 3.1 Memory-based Deduplication Attacks

As our solution is based on storage deduplication features, we briefly discuss attacks based on memory deduplication to better differentiate the possibilities and attack scenarios of both approaches. In 2014, Yarom and Falkner presented a last-level cache attack (based on the *prime+probe* principle as outlined by Tromer et al. (2010)) aimed specifically at cloud computing scenarios. This attack exploits timing differences when accessing data structures based on whether it has been loaded into the last-level cache or not. Since hypervisors implement memory deduplication and the last-level cache is shared between CPU cores, two otherwise isolated virtual machines actually share some memory. This makes it possible for attackers to extract confidential information across virtual machine boundaries, as long as the target virtual machine is located on the same physical machine as the attacker's (Yarom and Falkner, 2014). Based on this principle, Maurice et al. (2017) have shown that it is even possible to establish an SSH connection between two virtual machines over a covert channel, fast enough even to stream high definition video. More closely related to our contribution is the work by Irazoqui et al. (2015) aimed at detecting installed cryptographic libraries across virtual machine boundaries. Due to the high precision of this side channel, it is possible to pinpoint the exact version of an installed library and thus, probe for versions containing known vulnerabilities to be exploited.

Although all of these attacks make it possible to extract information in a very fine-grained manner and can therefore be devastating, they require attacker and victim VMs to be located on the same physical machine. Moreover, some of these exploits are specific to certain CPU architectures. Our storage-based side channel attack, on the other hand, is OS-independent and makes no assumptions about the host machine's architecture. In its current form it does, however, not enable high-resolution attacks or high-speed covert

channels and is thus better suited to probe for installations of known vulnerable software to be exploited separately, similarly to the work of Irazoqui et al. The following section outlines attacks on storage deduplication to put our work into context.

### 3.2 Storage-based Deduplication Attacks

Already in 2010, Harnik et al. have shown that client-side, cross-user deduplication in cloud storage settings can be exploited to violate some privacy guarantees typically assumed in such a context. In essence Harnik et al. (2010) show how it is possible to detect whether or not some file is present on a cloud storage platform by observing traffic when uploading a file. This works, because prior to uploading a file, its hash is transmitted to the cloud storage provider to check whether the file is already present and if it is, no upload takes place. After projects like *dropship*<sup>8</sup> started to actively exploit this feature for covert, cross-user file sharing, countermeasures were put in place by Dropbox.

While our work is based on the same principle, it targets a multi-tenant cloud computing context. It can therefore be considered an extension of memory-deduplication attacks across the physical machines hosting virtual machines and makes it even possible to infer information about the hosts themselves. This opens up new attack vectors compared to memory-based side channel attacks, since our only requirement is a shared ZFS pool and thus no assumptions about sharing a CPU with a victim need to be made. This is especially devastating in a managed cloud scenario, where tenants do not have full access to the instances they are using and are thus incapable of deploying countermeasures. Pooranian et al. (2018), for example, propose to group storage requests as to make guesses about which chunk of data may be already present less reliable. Since this defence targets only the cloud storage sector it is difficult to transfer this approach to performance-critical settings such as the multi-tenant cloud computing sector. Other approaches based on convergent encryption, such as *CloudDedup* proposed by Puzio et al. (2013), on the other hand, would require invasive changes to the software stack deployed by cloud providers. In effect, it is unlikely that countermeasures proposed for cloud storage are applicable to the cloud computing sector due to highly different constraints.

Our work essentially closes the gap between cross-user cloud storage deduplication attacks and

<sup>7</sup><https://dropbox.com/>

<sup>8</sup><https://github.com/driverdan/dropship>

memory-based side channel exploits directly aimed at the cloud computing sector. This means that we are able to apply storage-based techniques to establish multi-bit covert channels between virtual machines, even if they do not share a host, as long as their virtualised drives reside on the same ZFS pool. Consequently, schemes like those proposed by Hovhannisyan et al. (2015) to establish more than just single-bit covert channels based on the existence of a particular file can directly be applied in our setting. While the original publication on this matter presented a significant advancement over single-bit channels, it is still limited to bandwidth constraints under real-world settings. Our attack scenario, on the other hand, is not subject to such limitations as it puts no strain on network links. Consequently, it works even when no network access is available, similar to the attack presented by Maurice et al. (2017). Far more effective than applying file-based deduplication side channels to establish covert communication, however, is using our approach to probe for the installation of vulnerable software, as explained by Irazoqui et al. (2015).

To actually recover the IP address of a virtual machine that has a vulnerable software installed, we can apply the same mechanism as proposed by Irazoqui et al.: In effect, once a vulnerable version of a service is found, we can simply probe the IP neighbourhood of the virtual machine we control and try to mount known attacks on each reachable IP address to see which one responds to the attack.

## 4 PRYING COW

The following subsections describe the test setup, the experiments performed, and the results obtained, showing that it is indeed possible to infer information about neighbouring virtual machines.

### 4.1 Test Setup

In our evaluation, we concentrate on the file system ZFS, as it is the most mature representative of file systems supporting deduplication. The experiments were performed on a computer with an Intel® Core™ i5-6200U CPU and 12 GB RAM. The host operating system was Ubuntu 18.04 LTS with Linux Kernel 4.15 and ZFS on Linux 0.7.5. While ZFS was originally developed for the Solaris operating system, ZFS has also been ported to Linux and is, for example, officially supported by Ubuntu since version 16.04

LTS<sup>9</sup>. The virtual machines in the experiment are running on QEMU/KVM. The images of the virtual machines are stored on a common ZFS pool, which uses a recordsize of 64 KiB. The adjacent virtual machines generally run in the background during the experiments.

The measurement program was executed on a virtual machine running Ubuntu 18.10. For this purpose, the files to be detected were loaded into the virtual machine of the attacker. In order to exclude false positives, these files were saved onto a tmpfs, i.e., they were written directly into the allocated main memory of the virtual machine instead of the storage volume. In general, the files stored on the virtual machine executing the measurement program should have as little overlap as possible with the data used to detect the victims. Due to the immediate deduplication of the contents using in-line deduplication, measurements can be performed directly when writing data. Compared to post-process deduplication as used in btrfs, there is no need to wait until the deduplication process is complete. During the evaluation phase, the measuring program directly memory maps the file. The application then proceeds by writing the mapped file to the storage volume of the virtual machine, which is backed by the ZFS pool. To measure the execution time more precisely, the file descriptor used to write the file is opened with the option `O_SYNC`. As a result, the operating system waits until all data have been written, and, compared to the default behaviour, does not continue the program flow before the data are committed to the system. In addition, the execution times of the write operations were measured on the basis of the guidelines provided by Intel® for benchmarking code as outlined by Paoloni (2010). This prevents the measurement results from being falsified by the out-of-order execution of modern processors.

Simultaneous processes on the storage volume or the system could, in general, lead to differences in the execution time. While this has not been an issue for large files in our experiments, as the difference between the execution times is more significant, it may lead to wrong conclusions when writing smaller files. In order to compensate for such differences, the measurements can be performed several times and then statistically analysed. To restore the virtual machine to the starting point after a measurement, the file is overwritten with random data and subsequently deleted. We measure both the time it takes to write the file initially, and the duration of overwriting the file afterwards. As mentioned previously, our

<sup>9</sup><https://arstechnica.com/gadgets/2016/02/zfs-file-system-will-be-built-into-ubuntu-16-04-lts-by-default/>

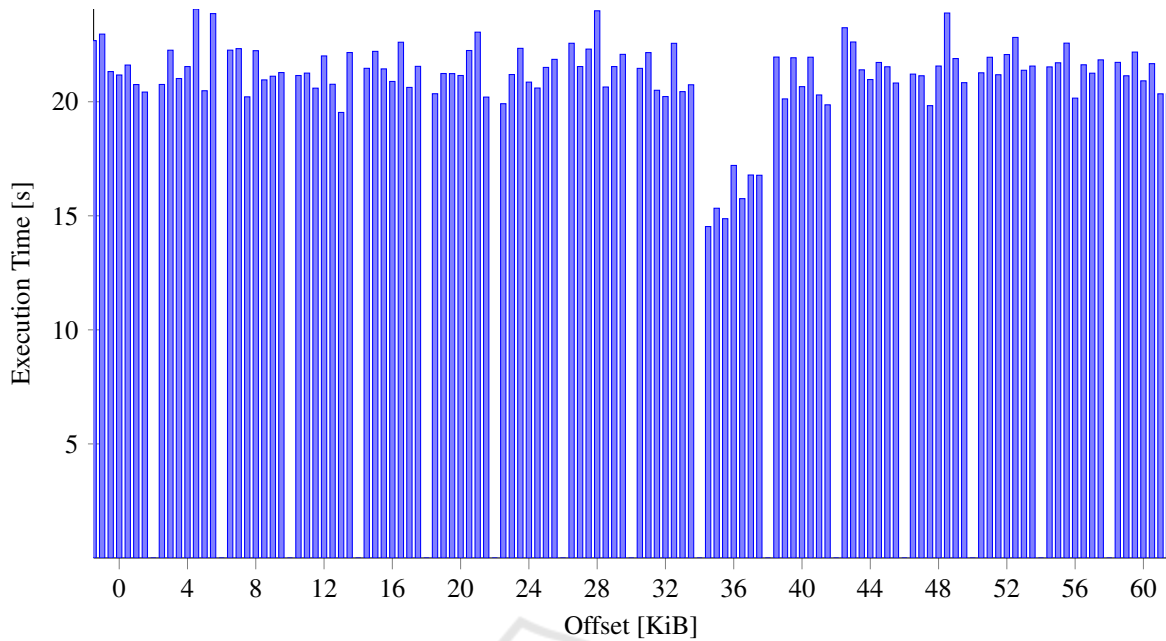


Figure 2: Detection of the the block offset inside a ZFS record.

assumption is that writing the file initially is faster if the file already (partially) exists on the volume, while overwriting a deduplicated file is slower than overwriting a non-deduplicated file. In our experiments, we found that the difference is more pronounced in the first case. Therefore, the stated measurements in the following case studies are from the initial writes, if not mentioned otherwise. The following subsection proceeds by showing how to detect the block offset, in case the blocks stored in the virtual machine image are not aligned with the recordsize.

### 4.2 Detection of the Block Offset

As stated in the previous section, our ZFS pool works on records with a size of 64 KiB. However, we observed that most file systems created during the installation of an operating system use a block size of 4 KiB. Thus, replicated files may not be aligned to the recordsize and may not get deduplicated, as outlined in Section 2.1. While we observed that some operating systems, such as Ubuntu 18.10, align (large) files to 64 KiB, other operating systems such as Windows 10 do not. To account for a possible alignment mismatch on real-world systems, the measurements can be invoked repeatedly with padded files. Without loss of generality, we assume that the victim virtual machine uses a block size of 4 KiB. Therefore, we will pad the file at the beginning in steps of 4 KiB. To avoid skewing the measurements by writing paddings of different size, we always pad the written file by

64 KiB. The part of the pad that is not used at the beginning of the file will be attached to the end of the file. Figure 2 shows the results of detecting a non-aligned file inside a virtual machine running Windows 10. As shown in this figure, writing the file with an offset of 36 KiB takes less time than writing the file at other offsets. Therefore, we detected that the file exists in another virtual machine at this offset. In the following subsections, we assume that the files are aligned to the recordsize of the ZFS pool.

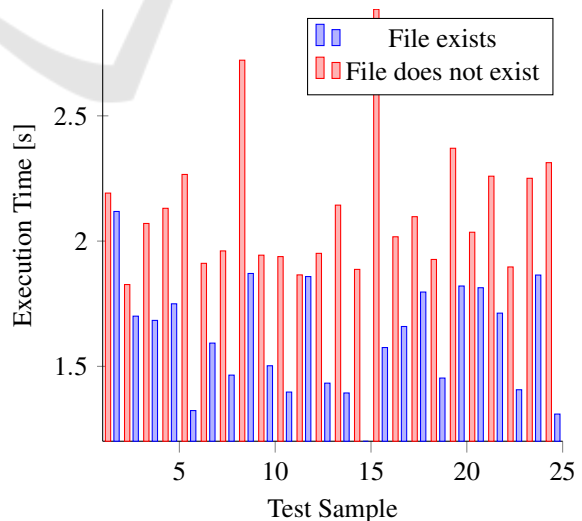


Figure 3: Execution times of writing the file *gnome-3-26-1604.70.snap* (part of Ubuntu 18.10) to disk.

### 4.3 Detection of Adjacent Operating Systems

In the first case study, we identify the operating system an adjacent virtual machine is running. To identify the operating system, a file unique to the operating system was chosen. For example, in this experiment, we first detect whether or not a neighbouring virtual machine has Ubuntu installed. The *gnome-3-26-1604\_70.snap* file was chosen to identify the operating system, as it comes by default with the operating system, even when choosing a minimal set of applications during the installation. As the virtual machine used to conduct the measurements also runs Ubuntu 18.10, the corresponding file was removed from this system prior to the measurement runs to avoid any false negatives. The measurements can be found in Figure 3. The red bars show the experiments where Ubuntu was not installed on the same volume, while the blue bars show the experiments where Ubuntu was installed. The Figure depicts 25 different measurements for each of the cases, out of the 200 measurements conducted. As shown in the figure, the writing time is generally much higher if the library file does not exist, confirming our earlier assumption. Due to some outliers, it is advisable to perform multiple measurements before coming to a conclusion about the utilised operating system. However, even considering these outliers, using k-means we were able to cluster 81% of the measurements correctly. As each measurement only takes a few seconds, it is possible to detect the adjacent operating system in a short amount of time, even when conducting multiple measurements.

For the sake of completeness, we also identify a neighbouring Windows 10 virtual machine. The measurements are depicted in Figure 4. Similar to the previous illustration, the red bars show the experiments where Windows 10 was not installed, while the blue bars show the experiments where Windows 10 was installed. In this case, the *Microsoft.Photos.dll* library was chosen to identify the operating system, as it only exists in Windows 10. As in the previous experiment, it is again possible to distinguish whether or not the operating system is deployed in the vicinity. Summing up, it is possible to detect both Linux- and Windows-powered virtual machines.

### 4.4 Detection of Applications

The following case study shows that it is even possible to detect individual applications in a virtual machine running in parallel. The adjacent virtual machine is running Ubuntu 18.10. The application

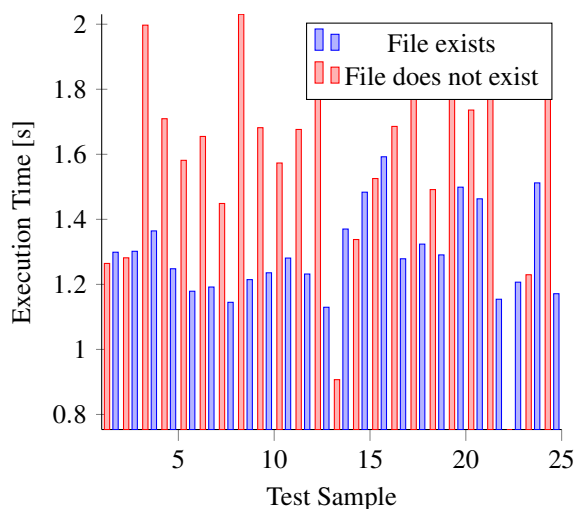


Figure 4: Execution times of writing the *Microsoft.Photos.dll* library (part of Windows 10) to disk.

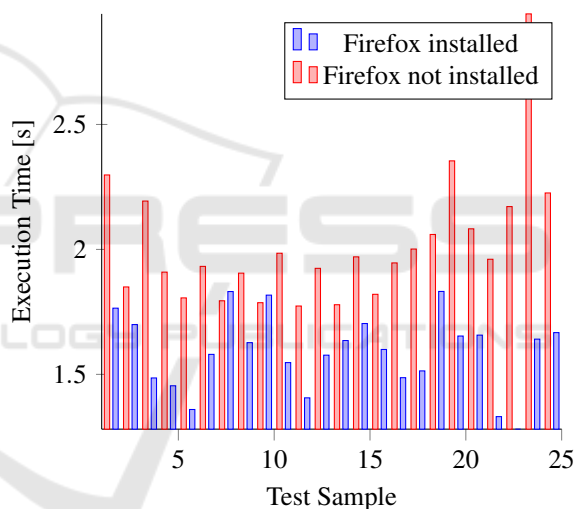


Figure 5: Execution times of writing Firefox's *libxul.so* to disk.

to be detected is the Firefox web browser. The measurements conducted for this experiment are shown in Figure 5. Similar to the previous case study, 25 measurements are depicted, where the red bars illustrate the case where Firefox is not installed, while the blue bars show the experiments where Firefox was installed in a neighbouring virtual machine. In the test cases where Firefox is installed, it was installed from the official repositories of Ubuntu. We identify Firefox by detecting the *libxul.so* shared object file, which is an integral part of the web browser. In total, we conducted 100 measurement runs (50 for each case) for this experiment. In addition to giving an intuition about the detectability of an application, we again used k-means to cluster the results into



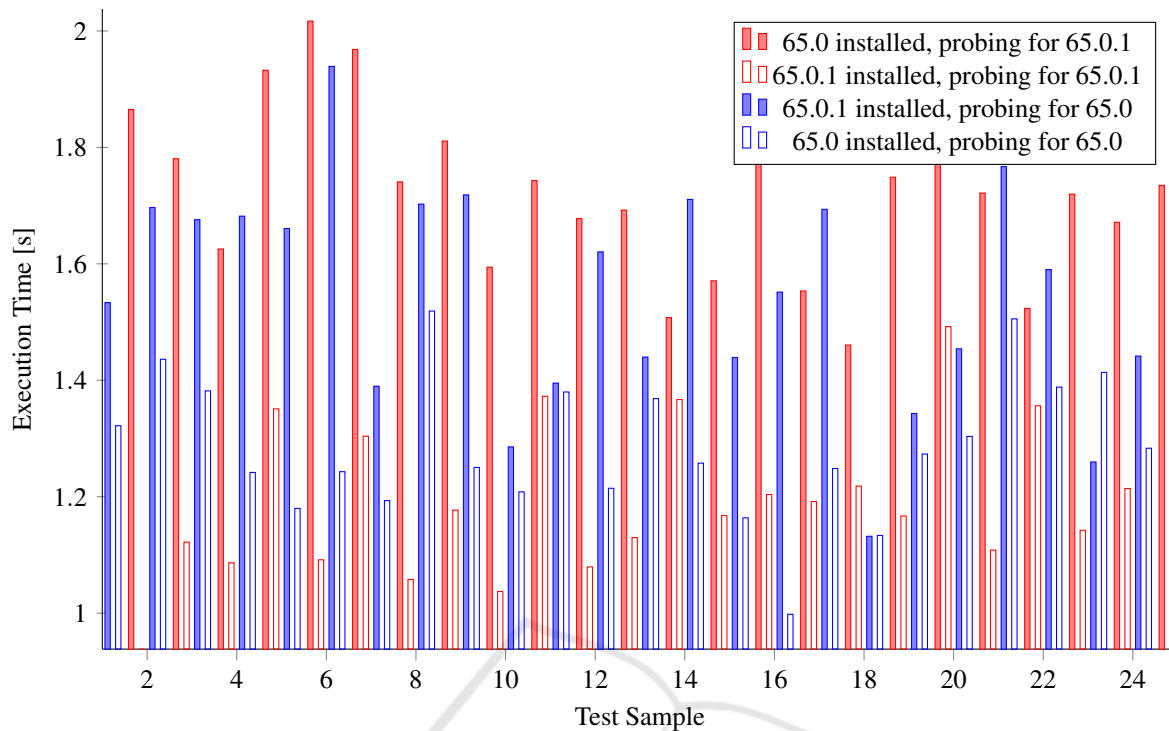


Figure 6: Execution times of writing different versions of Firefox’s *libxul.so* to disk.

two classes. It was possible to partition 83% of the measurements correctly, showing that it is indeed feasible to detect the existence of an application in a neighbouring virtual machine.

Even more interesting than the mere presence of an application is the patch level of a program, as accompanying changelogs provide information about potential vulnerabilities. Thus, we show in the next experiment that it is possible to detect the exact version number of Firefox, down to the minor version. Figure 6 shows these results. In the experiment, we distinguish four different cases:

1. Firefox 65.0 installed, but probing for Firefox 65.0.1
2. Firefox 65.0.1 installed, probing for Firefox 65.0.1
3. Firefox 65.0.1 installed, but probing for Firefox 65.0
4. Firefox 65.0 installed, probing for Firefox 65.0

Like in the previous experiments, the diagram shows 25 measurements for each case. This experiment shows that an adversary can detect the exact version of writing the corresponding file. Consequently, it is possible to detect applications with known vulnerabilities in a neighbouring virtual machine. For example, Firefox 65.0.1 closed four different security vulnera-

bilities<sup>10</sup>, some of which are openly documented<sup>11</sup>. The update did not include any other changes. An attacker could potentially use the known vulnerabilities to compromise an adjacent virtual machine.

Similar to the previous experiments, we used k-means to cluster all the 200 conducted measurement runs (100 for each case) into two clusters, i.e., into application detected and not detected. With k-means we were able to correctly partition 91.5% of the measurements between case 1 and 2. By combining both writing measurements and overwriting measurements, we were able to increase the accuracy to 94%. On its own, the overwriting measurements achieve an accuracy of 79.5%. When applying k-means to case 3 and 4, 93.5% of the writing measurements can be partitioned correctly. Combining the writing and overwriting measurements did not increase the achieved accuracy. Still, this experiment shows that it is possible to detect the installed version of an application with high accuracy, even by evaluating only a single measurement.

<sup>10</sup><https://www.mozilla.org/en-US/security/advisories/mfsa2019-05/>

<sup>11</sup><https://googleprojectzero.blogspot.com/2019/02/the-curious-case-of-convexity-confusion.html>

## 5 DISCUSSION

This section deals with limitations of the attack vector and possible countermeasures on the part of the operators of cloud services and the users of the virtual machines.

### 5.1 Limitations

The attack cannot determine whether a program or virtual machine is currently running or only resident on the disk. If multiple adjacent virtual machines are present, it is hard to tell which of the virtual machines is running a specific operating system or application. Thus, this attack is comparable to attacks on the page deduplication of the main memory, such as outlined by Suzaki et al. (2011). In their attack, no definitive statement can be made about currently running programs, as non-executed program files can also be stored in RAM. Likewise, it is not possible to recognise which neighbouring virtual machine a program is located in. Even when considering these limitations, the ability to recognise files in other virtual machines undermines the security and privacy assumptions associated with virtualisation.

### 5.2 Countermeasures

To avoid this vulnerability on the side of the operators of VPSs, images of different virtual machines can be stored on separate ZFS pools or, in general, on partitions that are deduplicated independently. This means that files in other virtual machines can no longer be identified by an attacker. However, by applying this countermeasure, only the blocks within a single image can be deduplicated. Since the storage advantages result primarily from the fact that the operating system and application files of different images are similar, the benefits of deduplication are greatly reduced.

As an alternative to deduplication, compression can be utilised. Although, while compression can save storage space, it does not prevent similar files from being replicated on the medium. On the other hand, compression requires less memory compared to in-line deduplication, since the information required for deduplication does not have to be stored in the main memory. However, especially with many duplicated blocks, the reduction in storage space achieved by compression alone is likely to be lower.

On the side of the end users of the virtual machines, encryption can be applied to the stored data. This effectively prevents applications or data stored in the virtual machine from being recognised by an

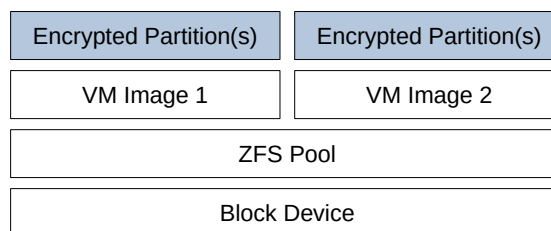


Figure 7: Applying encryption on the user side to avoid deduplication.

attacker. Encryption, in general, completely prevents deduplication of the data, as it renders each replicated block of data different. Therefore, encryption has to be applied above the deduplication layer, as shown in Figure 7. At the same time, this approach nullifies any advantage of deduplication. Moreover, even when enabling full hard disk encryption in the installation process of some operating systems, data sufficient for identification of the operating system type may be stored unencrypted (Landau, 2018). For this reason, it should be ensured that all files are properly encrypted as outlined by Kogan (2014) — even the files required for the boot process, such as the Linux kernel image.

## 6 CONCLUSIONS

Data deduplication is typically used in cloud environments to reduce the required storage space and thus improve efficiency and in turn provide cost reductions. However, as we have shown in this paper, relying on deduplication can also reveal information about neighbouring systems that compromises the privacy and security of virtual machine users. More specifically, this work has demonstrated a novel attack which exploits the deduplication feature of modern file systems. We have successfully demonstrated for the first time how timing attacks on write operations in a shared file system pool enable an adversary to probe for installed software or operating systems inside adjacent virtual machines, including version numbers and patch levels of applications.

Our approach differs from existing attacks in the sense that it is purely based on measuring write operation timings of modern file systems and does not require elevated user rights. Furthermore, the approach does not make assumptions about the underlying hardware or network access. As a result, the attack effectively breaks one of the core security assumptions of cloud computing.

In general, the attack can be prevented by relying on separate pools for each virtual machine or by

relying solely on compression instead of deduplication to reduce storage usage. Another approach to counter this attack is to encrypt the data inside the virtual machines. However, all of the discussed countermeasures impact the achievable storage space reductions negatively. Therefore, we plan to explore how to better mitigate such attacks in future work. While this paper focused on detecting individual files, we believe that it is also possible to discover multiple consecutive files. In this case, each of the written files has to be aligned to the block size of the file system. We leave the extension of the measurement application as future work. Nevertheless, due to the severity of the presented attack, we propose to reconsider relying on data deduplication in multi-user and shared cloud environments.

## REFERENCES

- Harnik, D., Pinkas, B., and Shulman-Peleg, A. (2010). Side Channels in Cloud Services: Deduplication in Cloud Storage. *IEEE Security & Privacy*, 8:40–47.
- Hovhannisyanyan, H., Lu, K., Yang, R., Qi, W., Wang, J., and Wen, M. (2015). A Novel Deduplication-Based Covert Channel in Cloud Storage Service. In *2015 IEEE Global Communications Conference, GLOBECOM 2015, San Diego, CA, USA, December 6-10, 2015*, pages 1–6. IEEE.
- Irazoqui, G., Inci, M. S., Eisenbarth, T., and Sunar, B. (2015). Know Thy Neighbor: Crypto Library Detection in Cloud. *PopETs*, 2015:25–40.
- Jin, K. and Miller, E. L. (2009). The effectiveness of deduplication on virtual machine disk images. In *The Israeli Experimental Systems Conference – SYSTOR 2009*, ACM International Conference Proceeding Series, page 7. ACM.
- Kogan, P. (2014). Full disk encryption with luks (including /boot). <https://www.pavelkogan.com/2014/05/23/luks-full-disk-encryption/>. Last accessed on Mar 13, 2019.
- Landau, P. (2018). Full-system encryption needs to be supported out-of-the-box including /boot and should not delete other installed systems. <https://bugs.launchpad.net/ubuntu/+source/ubiquity/+bug/1773457/>. Last accessed on Mar 13, 2019.
- Maurice, C., Weber, M., Schwarz, M., Giner, L., Gruss, D., Boano, C. A., Mangard, S., and Römer, K. (2017). Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In *Network and Distributed System Security Symposium – NDSS 2017*. The Internet Society.
- Ng, C., Ma, M., Wong, T., Lee, P. P. C., and Lui, J. C. S. (2011). Live Deduplication Storage of Virtual Machine Images in an Open-Source Cloud. In *International Middleware Conference 2011*, volume 7049 of *LNCIS*, pages 81–100. Springer.
- Paoloni, G. (2010). How to benchmark code execution times on intel ia-32 and ia-64 instruction set architectures. *Intel Corporation*, page 123.
- Pooranian, Z., Chen, K., Yu, C., and Conti, M. (2018). RARE: Defeating side channels based on data-deduplication in cloud storage. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops, INFOCOM Workshops 2018, Honolulu, HI, USA, April 15-19, 2018*, pages 444–449. IEEE.
- Puzio, P., Molva, R., Önen, M., and Loureiro, S. (2013). ClouDedup: Secure Deduplication with Encrypted Data for Cloud Storage. In *IEEE 5th International Conference on Cloud Computing Technology and Science, CloudCom 2013, Bristol, United Kingdom, December 2-5, 2013, Volume 1*, pages 363–370. IEEE Computer Society.
- Suzaki, K., Iijima, K., Yagi, T., and Artho, C. (2011). Memory deduplication as a threat to the guest OS. In *European Workshop on System Security – EUROSEC 2011*, page 1. ACM.
- Tromer, E., Osvik, D. A., and Shamir, A. (2010). Efficient Cache Attacks on AES, and Countermeasures. *J. Cryptology*, 23:37–71.
- Yarom, Y. and Falkner, K. (2014). FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *USENIX Security Symposium 2014*, pages 719–732. USENIX Association.
- Zhao, X., Zhang, Y., Wu, Y., Chen, K., Jiang, J., and Li, K. (2014). Liquid: A Scalable Deduplication File System for Virtual Machine Images. *IEEE Trans. Parallel Distrib. Syst.*, 25:1257–1266.