CS 261 Notes: Hardware Security 3

Scribe: Nick J Riasanovsky

October 1, 2018

1 Introduction

Intel SGX offers security promises against many forms of adversaries including an untrusted operating system attempting to modify computation. However, it crucially fails to protect against various side channel attacks based upon memory accesses. We will explore the risks these vulnerabilities pose by examining the Controlled-Channel Attack [1] and Data Leakage in MapReduce [2]. These notes will review the layout of SGX in section 2, motivate the mechanisms behind the Controlled-Channel Attack in section 3, and discuss the full attack alongside possible mitigations in section 4. Then these notes will transition to the presenter's portion of the lecture. It will briefly review VC3 in section 5 as an example system which could be attacked and then a simplified series of data leakages will be discussed in section 6. Section 7 will detail the actual MapReduce attack and finally section 8 will conclude by briefly discussing other possible attacks on SGX.

2 SGX Components

Before we outline the vulnerabilities presented by side channel attacks on SGX, it is first helpful to summarize SGX's design. We won't do an in depth discussion of the design of Intel SGX, as those were the focus of the previous lecture, and will instead do a review of the components necessary to understand possible exploits. Two details of interest are the trust assumptions made by SGX and how SGX handles paging. For a more complete discussion of SGX and secure enclaves refer to the previous lecture's notes in [3].

2.1 Trusted Computing Base

The Intel SGX consists of three trusted components:

- Intel Attestation Service (IAS)
- Enclave Code
- Intel CPU Chip

IAS is the service responsible for placing user code and any necessarily trusted libraries securely inside an enclave using secret, built-in keys [4]. All existing exploits against IAS mirror traditional attacks against remote servers so we will not discuss them in detail.

Enclave code consists of any code executed inside the enclave. The most obvious source of vulnerability presented by this model results from bugs in the user code itself, which some researchers have attempted to resolve by providing greater memory safety [5]. Alternatively, a more interesting set of attacks are the Iago attacks, in which the untrusted operating system returns invalid system call values to maliciously interfere with the program's execution. For example a program calling malloc could be returned an address on the stack which results in overwriting the saved return address and leads to arbitrary code execution. For this reason, enclave code may also include libraries or monitors designed to avoid relying on the untrusted

operating system [6]. While this attack vector is interesting we will focus on attacks against the actual Intel CPU Chip.

2.2 Paging

While there are many side channel attacks that can utilized, such as the cache timing attack we explored previously in [7], we explore a different attack based on memory accesses. This attack is newly important because we now must consider the operating system in our threat model and the OS traditionally handles memory access through on demand paging.

In Intel SGX there are two components mutually responsible for memory allocation. Inside the enclave memory is stored in the Enclave Page Cache (EPC) and managed by the Enclave Page Cache Map (EPCM). The EPCM contains all virtual to physical mappings and also the permissions associated with each page. This portion is presumed to be trusted. Additionally, since the operating system still needs to manage paging to and from disk, its page table also stores a copy of the permissions associated with each mapping. Critically this operating system controlled page table is not trusted. When determining the permissions for a page accessed inside the enclave, the minimum permissions across both mappings are used because the operating system is still responsible for swapping pages to disk. This means that if a page has read permissions in the EPCM, but not in the page table maintained by the operating system then a page fault will occur. When a page fault occurs, the enclave first captures the fault and clears any information indicative of the programming context, such as the current register values. Then, control passes back to the operating system, informing the operating system of the base address of the page that needs to be fetched (the upper 52 bits of the virtual address). When this page is properly fetched, then control passes back into the enclave through its asynchronous exit pointer, where execution can again resume.

3 Tracking Memory Accesses

Now that we have seen how permissions are handled in SGX, we can use it to construct our attack. The general premise is that our adversary will reduce permissions on all pages except for the one currently in use. Then when a data or code element that lies on the page not currently in use is accessed, it will result in a page fault. The adversary will log what page is being accessed, and then restore the permissions of that page, possibly reverting the permissions of the previous page. Later, this access pattern will be used to uncover hidden data values which become visible from the specific access pattern executed. To better understand this attack, we will look at a couple simple examples.

3.1 Determining a Page Access

For a simple case consider the code segment shown in Figure 1. In this example there is a function f which contains a secret Boolean value, t. Depending on the value of t, either foo or bar will execute. We assume both foo and bar lie on different pages from the currently running page and each other. If so, then when the function executes a page fault will occur, either on page A if t is true or on page B if t is false. Therefore we have uncovered the value of t, intended to be kept secret, by looking at the page access that results from running f. Notice that we did this by looking at the page that was executing not the exact address, as we only have access to the upper 52 bits of the address.

3.2 Determining a Path

While determining which page is accessed can be useful, often large amounts of code can lie on the same page. To combat this issue, rather than looking at simply whether a page was accessed, we can look at a sequence of page accesses for more information. Consider for example the code segment in Figure 2, and imagine we are trying to determine whether f4() or f5() executed. Since both of these lie on the same page,

```
def f(bool t):
    if t:
        foo () //On code page A
    else:
        bar () //On code page B
```

Figure 1: Example code for determining a boolean value from a single page fault. Notice that foo and bar both lie on separate pages and therefore the page requested determines t's value.

```
def f1 (): //Page A
    ...
    f2 () //Page B
    ...
    f3 () //Page C

def f2 ():
    f4 () //Page D
    f5 () //Page D
```

Figure 2: Example code segment involving multiple function calls. We want to determine if f4() or f5() just executed but both lie on the same page and as a result produce the same page fault.

simply looking for f4() or f5() in the page fault will be unsuccessful, as they appear identical. However if we look at Figure 3, we see that by following a trace of the pages involved with the entire call stacks for f4() and f5(), A B D indicates f4(), whereas page A C D indicates f5().

4 Controlled-Channel Attack

Now that we know how to determine secret values from function execution and how to distinguish execution based upon the call stack, we can understand the complete Controlled-Channel Attack. The Controlled-Channel Attack can consist of tracking Data pages, Code pages, or both, depending on the application. Data page tracking is similar to the code tracking example presented previously, except we are exploring

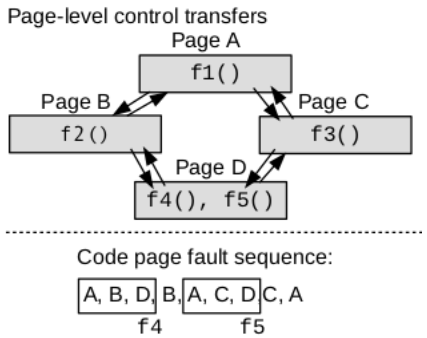

Figure 3: Example page fault sequence. By looking at the path we can distinguish f4 from f5 even though they have the same individual pages. (This Figure is from Xu et al. [1])

	Haven		InkTag	
	words	%	words	%
recovered exactly	35273	88.81	35760	90.03
recovered without affix	2880	7.25	2896	7.29
not recovered or incorrectly resolved ambiguity	1566	3.94	1063	2.68

Figure 4: Accuracy of the Hunspell attack on the *The Wizard of Oz*. (This Figure is from Xu et al. [1])

how data accesses produce paths based upon the content of hidden information. The Controlled-Channel Attack consists of three parts:

1. Path Identification
2. Tracking the Target Program
3. Access Analysis

4.1 Path Identification

In the first step the user code is run outside of the enclave. The data and code pages accessed are tracked and shortest paths are computed for each area of interest. Then any pages which do not need to be tracked to determine an exact path are removed from the attack are a performance concern. Running the code outside the enclave is crucial for using data access, as data access locations may not be known until runtime.

4.2 Tracking the Target Program

In the actual tracking step, the actual program executes inside an enclave. As previously mention all tracked pages are made unavailable except for the one currently in use. This is used to compute a list of page accesses that occur for both code and data pages of interest.

4.3 Access Analysis

In the final step the results from the tracking step are processed. The paths are computed to determine the values of the hidden information in question. This can be run offline so it does not contribute to the delay of the normal application.

4.4 Results

The researchers tested this attack against 3 programs: Freetype [8], Hunspell [9], and libjpeg [10]. However, we will only discuss Hunspell here.

4.4.1 Hunspell

Hunspell is an open source spell checking program. Hunspell first loads a dictionary into a hashtable and then checks for the existence of each word in an input file by doing a lookup against the same hashtable. When checking membership the same data accesses occur as when originally inserted. Therefore running the dictionary insertion offline will allow researchers to track what data accesses are associated with each word. Doing this, researchers were able to recover 88% of the original input. A copy of a subset of the results are included as Figure 4.

	Granularity	Noise Level	Number of Phases	Usage	Permissions Required
Controlled-Channel	$\sim 4KiB$	None	1	Enclaves	Root
Cache Timing	$\sim 64B$	Low-Medium	2	General Use	User

Figure 5: Comparison between a cache timing attack and a Controlled-Channel attack.

4.5 Why Use Controlled-Channel Attack?

The Controlled-Channel attack is another example of a side-channel attack. As there are many to select from its worth evaluating the trade-offs associated with selecting a Controlled-Channel attack as opposed to alternatives. We provide a sample comparison between the Controlled-Channel attack and the cache timing attack in Figure 5.

4.6 Counter Measures

We also present 5 possible attempts to mitigate the Controlled-Channel attacks and access their effectiveness.

Address space layout randomization (ASLR): One mitigation that seems like it would be promising is ASLR, in which the the start of each memory region is randomly permuted. Unfortunately all programs begin with the loader, a section of code responsible for launching the actual enclave application. As a result this known loader location can be used to uncover the ASLR mapping that is currently in use. In fact, the researchers were able to break Windows ASLR in only 2 code page faults [1].

Self-Paging: An alternative approach to prevent operating systems from tracking pages is to have enclave applications handle their own paging entirely. Rather than having the operating system field page faults, all page faults would be handled directly by code running inside the enclave. This is the method of choice on RISC-V, but can also be disruptive, as this reshapes the relationship between the operating system and enclave applications.

Intel T-SGX: T-SGX is Intel's approach to self-paging and mitigating the Controlled-Channel attack. When a page fault occurs on an application running inside the enclave, control switches to the abort handler instead of the operating system. Then the software abort handler that is running handles loading in pages. Since the operating system is not responsible for paging inside the enclave, it cannot produce page faults on each page access [11]. Unfortunately T-SGX is not a complete solution to SGX side channel problems. For example the recent foreshadow attack was launched while T-SGX was activated [12].

Large Pages: Intel CPUs also allow for the use of larger page sizes.This does not provide a fix to the attack but can make it more difficult. Having larger page sizes would cause more data to be located on the same page, making it harder to find distinct page sequences dependent on input values. Since the attack is entirely based on input dependent unique page sequences, removing these sequences would reduce the number of programs susceptible to this attack.

Attack Detection: While it may not always be possible to prevent an attack, an alternative goal may be to simply detect that one is in progress. If a program detects that it is falling victim to a Controlled-Channel Attack, then the program would terminate to prevent complete data leakage. This is possible in some cases because the slowest attack leveraged produces a 354.9 times overhead cost [1]. Additionally, attacks could be detected by tracking the page fault count and aborting upon reaching an abnormal number of page faults. One issue with this approach is that an unfavorable schedule could be falsely interpreted as an attack, causing the program to terminate.

5 VC3

For the second vulnerability in this lecture we must first briefly recap the computation model, VC3, presented by Schuster et al. in [13] and discussed in the previous lecture. VC3 is an implementation of Hadoop

based around performing computation inside of secure enclaves. It follows the MapReduce paradigm across machines in the cloud. Communication between mappers and reducers occurs via encrypted communication. When the actual map or reduce task needs to be completed, a secure enclave is initiated, which acquires the necessary data and performs the desired task. For a more detailed discussion on VC3 refer to the previous lecture's notes in [3].

It is also worth noting that the attacks we will be conducting are not restricted to VC3. We will be focusing on observing metadata of encrypted traffic and cross referencing this against known characteristics. As a result such a technique can be deployed on a variety of other systems and we place all attacks in the context VC3 strictly for convenience.

6 VC3 Simplifcations

The exploit we will launch against VC3 is focused on monitoring network traffic and leveraging known public information to determine the contents of private information being processed. To better understand the construction of this attack, we will consider three simple examples, which collectively demonstrate the underlying principles of the attack.

6.1 Determining the Job

One assumption that we will make to conduct this attack is we will have some knowledge of what job is being executed. This is potentially a strong assumption, but in many cases it is feasible. For example, if we know a health care provider is uploading the data being run we can infer the data is most likely medical records. Similarly we can try compare details about the datasets being uploaded to known datasets. While this data may be encrypted, by looking at for example the size of the data we can attempt to infer if is a known quantity like the United States Census Data. While these are not guarantees, an attacker does have access to such contextual information in the real world and so we will assume that these correlations constitute a basis for knowledge about what job is being run.

6.2 Determining the Dataset

For the first example, we will try and determine which dataset is being processed. Consider three tables: tbl-small has 1 KiB of data, tbl-medium has 512 KiB of data, and tbl-large has 1 MiB of data. Assume a MapReduce operation is run on one of these three tables, and we would like to determine which one. By observing the network traffic we determine that the total amount of data sent was roughly 1 MiB. This gives a strong indication that the dataset being processed was tbl-large. As a result, we see that by examining the amount of input processed we can gain insight into the dataset being used.

6.3 Determining the Operation/Row

For the next example, we will attempt to determine information about what aspect of a table is being processed and additional information about what each entry is. Consider a simple database consisting of three columns as show in Figure 6. All users entries have distinct names, the same age, and Chuck and Fred have the same weight. Assuming that each reducer is associated with a different value, then the result in Figure 7 demonstrates a possible reducing step on this dataset. We see 2 elements arrived at one reducer and one at a second, which gives us a strong indication that we selected based upon the weight column. Furthermore, this allows us to identify element Doug as it holds a unique value. This example demonstrates that if we know the distribution of various possible operations, we can leverage information about the amount of data in use to gain insight into the actual operations being performed, possibly even determining the identity of certain elements.

Name	Age	Weight
Chuck	35	180
Doug	35	165
Fred	35	180

Figure 6: Example small database. There are three categories: name, age, and weight. All entries have different names, share the same age, and two of them have the same weight.

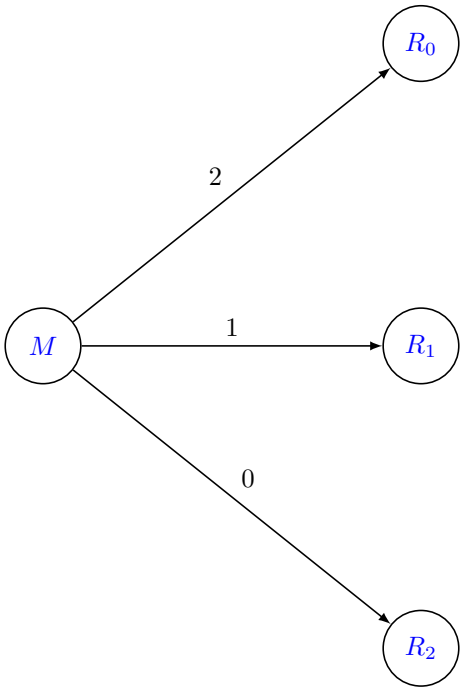

Figure 7: Example graph showing the number of entries sent to each reducer from the mapper. Since there are two reducers receiving data we can determine the column being selected from has two unique values.

6.4 Relating Two Datasets

As a final example, consider relating two datasets, one that is known and public and another that is secret. Assume we have two tables, a patient table that is hidden, and a diseases table is public knowledge. If we are executing the query

```
SELECT patient.name, diseases.treatment WHERE patient.disease = diseases.disease
```

then we can again observe the traffic to check for distributions, assuming once more that each reducer uniquely handles each disease. Since the disease distributions are known we can use this statistical breakdown to determine the identity of each reducer by disease. This means that if we could identify each row we can gain insight into the disease that each row has, which alongside further information may later lead to identifying the patient and their sensitive data.

7 MapReduce Vulnerability

Now that we have demonstrated three examples of the types of information that can be inferred simply by looking at traffic sizes and public information, we can construct the attack. We examine both a passive and an active attack.

7.1 Passive Attacker

We will first consider a passive attacker. For passive attackers we will make the following assumptions.

- The attacker can infer the specific job.
- The attacker can infer some data about the job.
- The attacker has access to public information.
- The attacker can acquire background knowledge about the inputs and outputs to the job (although the actual inputs and outputs will remain encrypted).
- The attacker may be able to determine the identity of the user.

The passive attacker additionally has the ability to observe communication between every element in the network. However, crucially this attacker does not have the ability to influence scheduling at all.

Given this available information the passive attacker's goal is to uncover hidden information about the members of the encrypted dataset, preferably tying information directly to people. The passive attacker will accomplish this by using their one ability to observe the network to determine metadata about the dataset being transferred. In particular the attacker will construct a matrix like the one in Figure 8 mapping the amount of traffic flowing from each mapper to each reducer. This matrix can then be interpreted to determine the size and distribution of the dataset(s) in use. This then enables the passive attacker to leverage our assumptions about access to public information and the specific job being run. For example, in the extreme case a single data value being mapped to a reducer can be used to directly determine the identity of the entry involved. Similarly we can use the simplifications from section 6 to determine additional information for more complicated cases. Crucially the attacker may not be able to uncover all the details of the dataset, but access to publicly available information can be used to learn the private information of people.

7.2 Active Attacker

The active attacker has all of the capabilities of the passive attacker, but has the additional power to schedule and control resources. This means that the active attacker can cause each element to be individual sent to

$$\begin{bmatrix} 15 & 30 & 42 & 0 & ... \\ 11 & 4 & 7 & 3 & ... \\ 1 & 6 & 5 & 27 & ... \\ & & . & & \\ & & . & & \\ & & . & & \end{bmatrix}$$

Figure 8: Example matrix constructed by a passive attacker. Each row represents a different mapper and each column a different reducer. The value in entry i, j is amount of data sent from the i^{th} mapper to the j^{th} reducer.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & ... \\ 0 & 0 & 1 & 0 & ... \\ 1 & 0 & 0 & 0 & ... \\ & & . & & \\ & & . & & \\ & & . & & \end{bmatrix}$$

Figure 9: Example matrix constructed by an active attacker. Each row represents a different entry in the dataset and each column a unique reducer. The value in entry i, j is 1 if the i^{th} entry in the dataset was set to the j^{th} reducer and otherwise 0.

a reducer, allowing the particular entries in the dataset to be tracked (although still encrypted). The active attacker will then create a different matrix. This matrix consists of a different row for each element and one column for each reducer. A 1 is placed in the column of the reducer to which the element is sent and 0s are placed else where. An example of this is shown in Figure 9. Despite the information being encrypted, an attacker can now more directly associate a specific entry with its properties. By tracking each individual elements, the attacker can not only determine the distributions (and therefore possibly the query), but can also now link each entry to this public value. Depending on the available public information, this may uncover more details about individual users.

However, while this attacker is very powerful, it also runs the risk of being detected. To combat this risk, active attackers will send only 1 data element at a time. If they attempt to schedule anything more powerful than this then they run too great a risk of being detected.

7.3 Formalized Definition

The attack conducted is about attempting to leverage available public information to infer details about Map Reduce jobs run under the VC3 paradigm. To formalize it, we describe the attack as a game.

Definition 1 (MapReduce Game) *At the start of the game, the adversary picks two datasets D^0 and D^1 and fixes the number M and R of mappers and reducers. He is then given access to [D], where $D = D^0$ or $D = D^1$, and can observe the run of multiple MapReduce jobs (M, R) of his choice. He observes their full network traffic to guess whether $D = D^0$ or $D = D^1$. The adversarys advantage of winning the game is his probability of guessing correctly minus $\frac{1}{2}$.*

This definition provides the a direct definition of success for an attack. If the attacker has a non-negligible advantage, then they win the game presented by definition 1. However this attack is easily won, if for example, the two datasets have a different size. The paper includes stronger definitions to help characterize games which prove to be more difficult for the adversary, but we will omit those from these notes. Fundamentally the attack is focused on determining details about the dataset being run and is not concerned with determining

absolute or new information. The adversary winning the MapReduce game is problematic as gaining any private knowledge could be used to increase the attack's chance of uncovering sensitive data.

7.4 Counter Measures

The primary method to combat this data leakage is to normalize and randomize the data sent through the network. By sending extra data to each mapper and reducer it is no longer possible to use statistical distributes to leverage known datasets for determining hidden information. The researchers also propose shuffling all data output by mappers and balancing this data evenly across the reducers [2]. In the next lecture we will examine Opaque, a more comprehensive solution to data leakage associated with our MapReduce paradigm [14].

8 Alternative Attacks

In addition to the two attacks discussed in this lecture, we also briefly mentioned a few other interesting attacks mounted against Secure Enclaves. While we will not discuss them in much detail, the curious reader can explore the attached references for more information.

8.1 Page Table Monitoring

Rather than producing page faults on each memory access, it is possible to simply observe the accesses made to page table. All Intel page tables consist of an access bit which indicates a page was recently accessed. By removing the page from the TLB and resetting the access bit, the adversarial operating system can track the pages being accessed without producing a page fault, as shown in [15].

8.2 Branch Prediction Vulnerabilities

Other attacks can be launched focused on exploiting the micro-architectural state of the Intel CPU chip. For example, attacks can be launched which train the branch predictor in order to produce out of order execution of code based upon secret data, such as the SGX variant of Spectre in [16].

8.3 Memory Bus Monitoring

It is also possible to track the memory access patterns associated with a process by monitoring the memory bus directly. All memory accesses inside and outside the enclave must go through the memory bus, unless they are cached. By introducing DIMM-snooping mechanisms to the memory bus, its possible to monitor all of these memory access as shown in [17]. Attempting to leverage this tracing process to produce a similar Controlled-Channel attack is an area of active research.

References

[1] Yuanzhong Xu, Weidong Cui, and Marcus Peinado, *Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems*, in 2015 IEEE Symposium on Security and Privacy, May 2015.

[2] Olga Ohrimenko, Manuel Costa, Cdric Fournet, Christos Gkantsidis, Markulf Kohlweiss, and Divya Sharma, *Observing and Preventing Leakage in MapReduce* in 22nd ACM SIGSAC Conference on Computer and Communications Security, October 2015, pg 1570-1581.

[3] Andrew Low, Hardware Security: Intel SGX and VC3, https://inst.eecs.berkeley.edu/ cs261/fa18/scribe/09_26.pdf.

[4] Victor Costan and Srinivas Devadas *Intel SGX Explained*, https://eprint.iacr.org/2016/086.pdf

[5] Dmitrii Kuvaiskii, Oleksii Oleksenko, Sergei Arnautov, Bohdan Trach, Pramod Bhatotia, Pascal Felber, and Christof Fetzer, *SGXBOUNDS: Memory Safety for Shielded Execution* in EuroSys '17, the Twelfth European Conference on Computer Systems, April 2017, pg 205-221.

[6] Chia-Che Tsai, Donald E. Porter, and Mona Vij, *Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX* in 2017 USENIX Annual Technical Conference, July 2017.

[7] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg, *Meltdown: Reading Kernel Memory from User Space* in 27th USENIX Security Symposium (USENIX Security 18), October 2018.

[8] Freetype. http://www.freetype.org/.

[9] Hunspell. http://hunspell.sourceforge.net/.

[10] libjpeg. http://libjpeg.sourceforge.net/.

[11] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado, *T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs* in Network and Distributed System Security Symposium 2017, January 2017.

[12] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx, *Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution* in 27th USENIX Security Symposium, August 2018.

[13] Felix Schuster, Manuel Costa, Cedric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, Mark Russinovich, *VC3: Trustworthy Data Analytics in the Cloud using SGX* in 2015 IEEE Symposium on Security and Privacy, May 2015.

[14] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica, *Opaque: An Oblivious and Encrypted Distributed Analytics Platform* in 14th USENIX Conference on Networked Systems Design and Implementation, March 2017, pg 283-298.

[15] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter, *Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX* in 2017 ACM SIGSAC Conference on Computer and Communications Security , October 2017, pg 2421-2434.

[16] Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh, *Spectre Returns! Speculation Attacks using the Return Stack Buffer* in 12th USENIX Workshop on Offensive Technologies (WOOT 18), August 2018.

[17] Yungang Bao, Mingyu Chen, Yuan Ruan, Li Liu, Jianping Fan, Qingbo Yuan, Bo Song, and Jianwei Xu, *HMTT: A Platform Independent Full-System Memory Trace Monitoring System* in SIGMETRICS '08 Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, June 2008, pg 229-240.