

Information Security - 5 - Secure Systems Engineering
Professor Chester Rebeiro
Indian Institute of Technology, Madras
The Rowhammer Attack

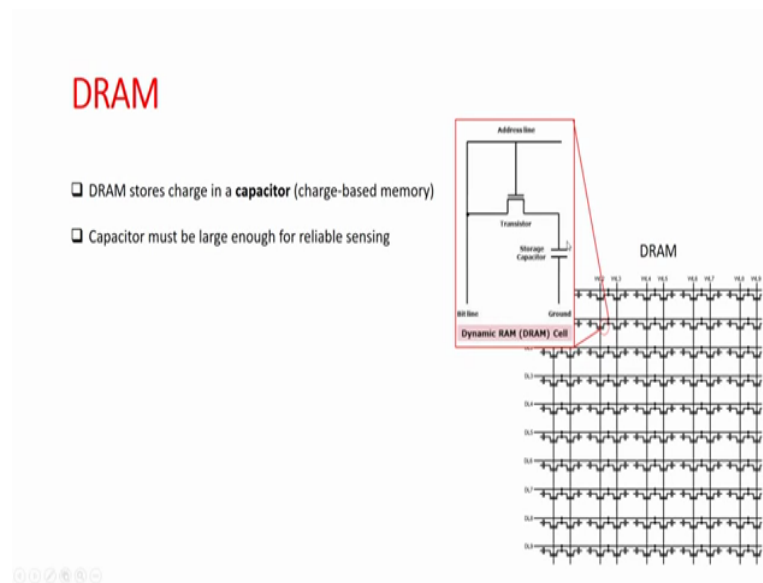
Hello and welcome to this video lecture in the course for secure systems engineering. In this video lecture we will look at a recent hardware attack which is known as a rowhammer, so this attack would let you flip bits stored in the DRAM without actually accessing them, it is a quite a recent attack it was discovered in 2014 and since then there have been various exploits which have used this technique to mount different types of attacks on applications and systems. Also there have been many countermeasures that have been developed in the recent past to prevent this attack.

(Refer Slide Time: 1:00)



A lot of these slides are actually borrowed from Professor Onur Mutlu's talk indeed in 2017.

(Refer Slide Time: 1:08)



So before we go into what row hammer means we would like a small background about DRAM's, DRAM as you know is one of the typical structures which are used for random access memories in systems, so a typical DRAM would look something like this way, these DRAMs are capacitive memories and they are arranged in this matrix like thing with rows and columns with a capacitor at each node. One particular DRAM cell would look like this, there is a transistor and an associated capacitor.

So when a 1 needs to be stored on this capacitor the capacitor is charged and in order to read the capacitance this transistor is turned ON and the data either 1 or 0 whether the capacitor is charged or discharged is actually read through this particular bit line. So essentially the charge of the capacitor defines whether this memory cell is storing a 1 or a 0 and capacitor must be large enough. The problem that may occur in capacitors is that the charge on the capacitor may gradually leak over a period of time, thus in order that a DRAM cell holds its charge it is required that the DRAM cells be recharged periodically.

(Refer Slide Time: 2:21)

DRAM Refresh Cycles

- ❑ As time passes, the charges in the memory cells (capacitors) leak away, so without being refreshed the stored data would eventually be lost.
- To prevent this, external circuitry periodically reads each cell and rewrites it, restoring the charge on the capacitor to its original level.
- ❑ Refresh cycles are in orders of milliseconds though, consume some memory bandwidth

So in order to achieve this what would happen is that there would be a special external circuitry, which periodically leads every cell in the DRAM and rewrites it therefore restoring the charge of the capacitor. So this refreshing phase ensures that the charge on the capacitance is restored and maintained for a longer period. So this particular figure over here shows how the refreshing occurs, so refreshing typically occurs periodically in a DRAM structure, so these particular areas is where the DRAM is inaccessible due to the refreshing period where the capacitors are recharged, while the accessible regions in the DRAM is over here. So any load or a store operation to a DRAM has to be within this particular time period.

(Refer Slide Time: 3:34)

DRAM Cells arrangement

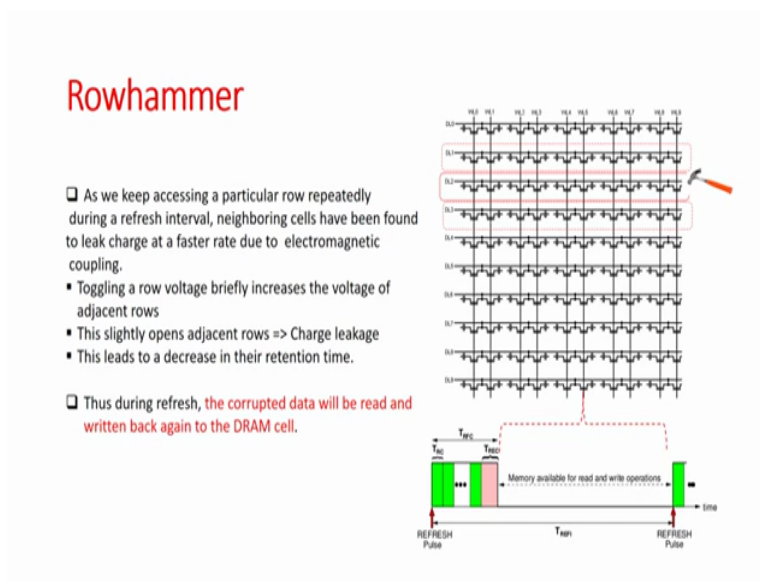
- ❑ Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]
- ❑ With reduction in transistor sizes, DRAM cells became smaller
- ❑ As DRAM cells became smaller, the space between two such cells reduces
- ❑ Closer the two charged bodies, higher the electromagnetic interference

With everything stored as a charge, can one row affect the other?

Now as we mentioned DRAMs are arranged in rows, so every time we want to actually read a particular memory location the entire row is activated and the charge is stored (from stored) in the various capacitors are then copied into this row buffer, from the row buffer it would then move to the various other components of the system including the cache memories and the processors. Now as time progressed and memories became more and more complex it was required to actually scale down and have more dense memories. As a result the number of such rows present in a DRAM was actually reducing over a period of time.

Now a consequence of this scaling was that the gap between these rows also reduced and since the cells in each row worked due to the charge present as the space between the rows reduced it became more and more likely that there would be interference between these various rows, essentially the closer the charged bodies are, the higher the electromagnetic interference between the various rows, this fact was actually utilized in the row hammer.

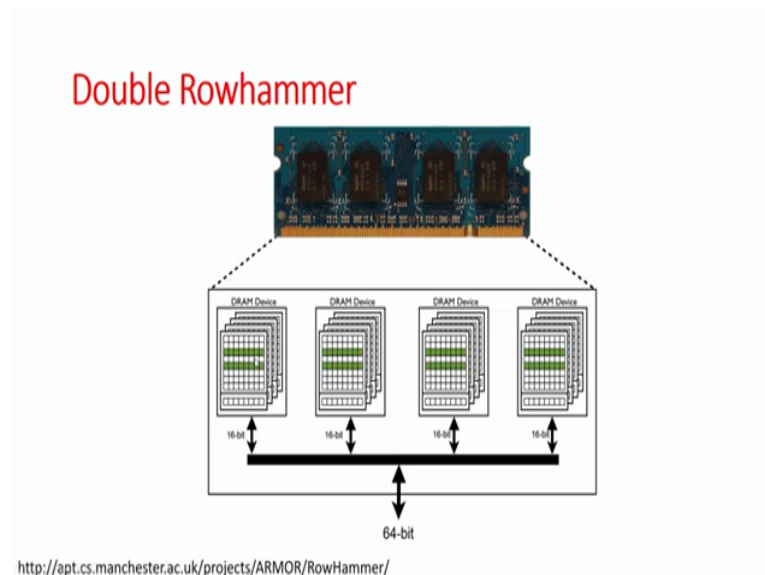
(Refer Slide Time: 4:46)



What the row hammer vulnerability actually showed was that if a particular row in the DRAM was continuously accessed this continuous memory access could actually influence the neighbouring rows. So for example over here we had one specific row which has been continuously accessed and it could influence the charge stored in the adjacent rows, the reason for this is that continuously toggling the row voltage slightly opens the adjacent rows, thus forcing the adjacent rows to leak much more quickly.

In other words the adjacent rows would actually discharge much more faster than the refresh period. The result is that certain cells on the adjacent rows may get corrupted and the data present in them may actually be toggled.

(Refer Slide Time: 5:40)



Now this is an example of how a row hammer attack would work and this animation is (taking) taken from this particular website from Manchester University, what the attacker would do is that he would toggle specific rows in the DRAM and access them continuously within a refresh period. Now this would influence the neighbouring rows and force the data present in the neighbouring rows to be toggled.

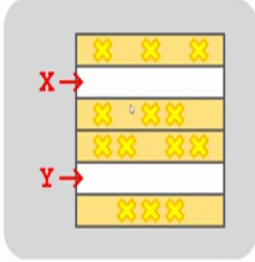
So this is represented here by these green and red blocks, the green block show what the attacker is legally accessing, while the red block show what show the effect of falls induced due to the row hammering of the DRAM. So what we are able to achieve is that we have been able to toggle adjacent rows just by accessing memory locations at a very high rate. Now why this is actually critical is that you are modifying the integrity of the storage of the data stored in the memory.

For example let us say that we have operating system specific data present in this specific row. For example this may be say of a particular page table and more particularly this (()) (7:02) may specify that a page is accessible only by the kernel. Now by inducing an error through the row hammering of the adjacent rows we would be able to modify the contents of the page table. So we would for example be able to convert a page which is meant only for the kernel to be accessible by user programs also.

(Refer Slide Time: 7:26)

A Simple program

```
loop:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp loop
```



The diagram shows a vertical stack of six DRAM rows. Each row contains three yellow 'X' symbols. Two red arrows point to the second and fifth rows from the top, labeled 'X' and 'Y' respectively. A red line connects the 'clflush (X)' instruction in the code to the 'X' label in the diagram.

To Avoid cache hits => Flush x from cache
To void row hits to x in the row buffer => Read y in another row

Download from: <https://github.com/CMU-SAFARI/rowhammer>

A typical row hammered program would look something like this, you could actually look at the source code over here from where this code has been borrowed, it is quite simple it has a small loop in which we target very specific rows within the DRAM. So in this particular case we are targeting this row x and y and forcing memory accesses to be done on these rows, note that the contents of this particular memory location is loaded into the eax register and ebx register as shown over here.

The CLF flush instructions is supported by x86 systems to flush the data corresponding to x and y from the cache memory. So this would ensure that the DRAMs are always accessed during these load instructions and the load does not actually obtain the data from the cache memories. The mfence instruction is used to serialize the transfers to the DRAM. So this small set of instructions is repeated over and over again at a very high rate thus posing continuous access to rows in the DRAM.

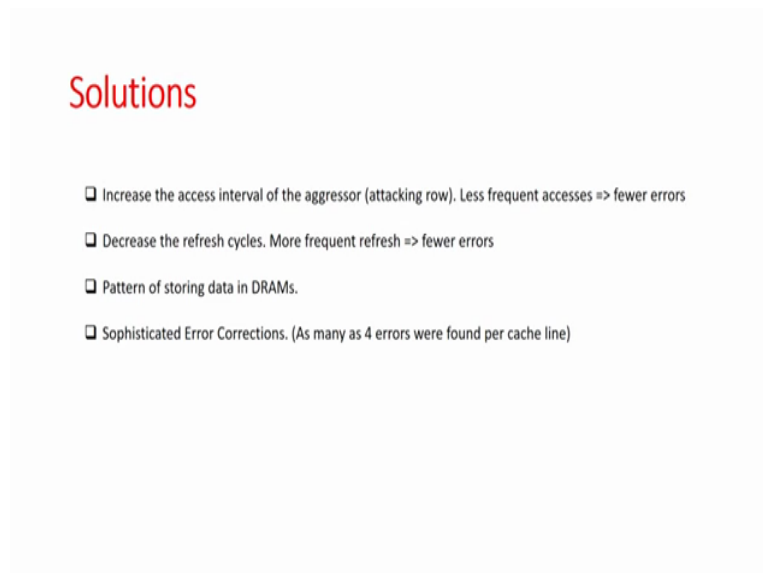
Now as we have seen in the previous animations doing so within at a rate much faster than the refresh period of the DRAM would cause the adjacent rows to lose charge and induce errors and falls in the adjacent rows.

(Refer Slide Time: 8:55)



This vulnerability has been used to actually mount several different attacks these attacks for example can be written in JavaScript and force web applications to obtain root privileges. Other attacks are the rampage attacks and the glitch attacks you could actually look up these attacks for more details.

(Refer Slide Time: 9:12)



Since the discovery of row hammer there have been several solutions that have been proposed, so these solutions have been done at various levels at the hardware level now DRAMs are designed in such a way so that the effect of such that row hammer is less likely to happen. Also things like increasing the refresh rate, adding more sophisticated error

correction techniques have been used in the hardware to actually make this to take care of these solutions.

Similarly techniques in the operating system like ensuring that the DRAM is partitioned and sensitive and non-sensitive data are not placed adjacent to each other on the DRAM. Other techniques like identifying patterns of usage in the DRAM is done using things like performance, counters present in modern processors and this is then used to prevent any row hammer like attacks. So the code for the attack can be downloaded, thank you.