Lest We Remember: Cold Boot Attacks on Encryption Keys

J. Alex Halderman∗, Seth D. Schoen†, Nadia Heninger∗, William Clarkson∗, William Paul‡, Joseph A. Calandrino∗, Ariel J. Feldman∗, Jacob Appelbaum, and Edward W. Felten∗

∗Princeton University, †Electronic Frontier Foundation, ‡Wind River Systems

DRAM is generally assumed to lose data once power is lost almost instantaneously. This paper seeks to show that this is not the case, and that it is in fact possible to recover the data stored in DRAM using cold-boot techniques designed to reduce data loss and algorithms designed to recovery decayed data even given error rates as high as 50%. In doing so, the authors test several different DRAMs with varying storage density and manufacture, and establish that the techniques outlined are generally applicable across multiple types of memory and platforms. In addition, they apply these techniques to several high-profile commercial encryption systems and algorithms, and discuss potential countermeasures.

Previous research done on DRAM has indicated that the data stored can survive reboots. This potentially meant that data thought to be secured by the rapid bit decay expected of DRAM might be vulnerable to recovery, either malicious or forensic. Furthermore, it is well known that decreased temperatures significantly slow this decay, meaning that the data may survive potentially minutes or hours after power loss. However, this paper is the first to specifically address those security concerns and show verifiable real-world attack techniques, even against systems thought to be (theoretically) secure against such an attack. Furthermore, previous research into the subject has primarily focused on a “burn in” effect, whereby data stored in the same memory location for protracted lengths of time cause a physical change in the memory that can be used to reconstruct the data stored. This paper does not depend or indeed even address this effect, instead focusing on capacitor remanence, the time it takes for a capacitor to decay to its ground state.

To begin, this paper tests the decay rate of various DRAM technologies. The authors first filled a section of memory with pseudorandom data, then prevented the DRAM from refreshing the data for various lengths of time before reading it back and checking the error rate of the data. From this, they consistently found that the memory very quickly decays with a similar logistic line of best fit across all technologies, though the exact steepness of the curve varied based on the memory density of the DRAM. This is consistent with what was expect, with the error rate generally evening out around 50% for all memory density and manufacture after a few seconds. They then repeated the test at a low temperature of -50°C. This was achieved by inverting cans of compressed air so that the contents was released as a liquid, rapidly cooling the memory. This, as expected, massively decreased the rate of decay such that 60 seconds after being data was retrievable with 99.9% accuracy. Finally, they repeated the test once more using liquid nitrogen, first cooling the chips with the inverted cans of compressed air and then disconnecting them and

placing them into a bath of liquid nitrogen. After 60 minutes the modules were removed and reconnected, and the authors found a decay of just 0.17%.

With this baseline established, the authors go on to look at patterns in the decay of the data. They observe that the decay of the individual cells is highly predictable based on manufacture with rate, order, and overall pattern being consistent at a fixed temperature. They visualize this decay by loading a bitmap into memory then repeatedly checking the data at various intervals of time between 5 seconds and 5 minutes. This consistency will be used further on to predict uncorrupted values for the data in a way significantly more efficient than the naive approach.

The last thing they look at with regard to the DRAM modules themselves is how the operating systems and BIOS interact with the memory on start up. Generally, on start up the BIOS overwrites a small portion of the memory with its own data. In addition, dependent on BIOS and in some cases the settings for the BIOS, it may overwrite the DRAM in its entirety during its POST. This, also, will be discussed further in the paper as a barrier to successful recovery of the data.

After establishing the decay rates and the various aspects thereof that might effect data recovery, the authors go on to outline the process for imaging the memory. They suggest multiple methods which might enable an attacker to negate the BIOS overwriting as much as is possible, which generally involve either booting the target computer using as small a boot environment as possible (a PEX network boot with a footprint of 9KB or EFI boot), or using an external medium (USB drives or iPods as an example of a covert method of doing this). They suggest methods to implement these including both a cool and warm boot, which as an aside also prevent the OS and applications from deleting the data before exiting. They also suggest simply transferring the DRAM to a separate device owned by the attacker, which has the additional benefit of enabling the imaging of the area that would typically be overwritten on boot. In the last case, the attacker would use the techniques discussed earlier to lower the temperature of the module before transfer, reducing decay.

Now that the authors have established a method to secure an image of the memory, they move on to discussing methods to recover cryptographic keys. The first point here that they address is how to reconstruct a key of various types after they've undergone some degree of decay. A brute force approach to this quickly becomes intractable, with just a 10% decay of a 256 bit key leading to 2^{56} possible keys to try. The solution to this is to use other data that is derived from the key and that is significantly more structured to determine which keys are likely or possible. Specifically, the authors used the key schedule used to decrypt blocks of memory as an error correcting code for the original key, with speeds between a fraction of a second and 2.5 minutes, depending on the kind and size of the key.

Once a method to recover a decayed key has been determined, they authors go on to finding the decayed keys themselves in memory. For a 128 bit symmetric key in 1GB of memory

with no decay, a brute force approach yields 2^{28} possible keys. However, even a small amount of decay quickly makes this approach intractable. To solve this, they take a similar approach to recovering the keys — look for the more structured key schedule. The one exception to this method, RSA keys, is made because a better method exists. For these keys, the authors instead look for searching directly for the private keys. To do this, they look for a known value (the public modulus) and use that as a basis to determine the rest of the key. Using this method returns no false positives. For all methods and all keys, the Hamming distance is used to deal with potential decay of the keys.

Having determined methods for imaging the memory, and finding and recovering cryptographic keys, the next step is to apply these methods. The authors successfully developed attacks against BitLocker, FileVault, TrueCrypt, dm-crypt, and Loop-AES. For the majority of these, the keys were recovered with no decay errors. The BitLocker attack developed is an entirely self-contained, automated attack for which the only manual operation necessary is to insert and boot from a USB drive. This attack takes approximately 25 minutes. In addition, while performing the attack on FileVault it was discovered that Mac OS X 10.4 and 10.5 store the user's login password in memory, which enables an attacker to access the user's keychain, giving them access to any number of passwords potentially including the one used to decrypt FileVault.

Overall, I very much like the approach the authors took towards developing their attacks. Their insistence on using no specialized equipment, and furthermore demonstrating the viability of their methods against commonly used commercial encryption software goes a long way towards convincing readers that the vulnerabilities outlined in this paper are a real threat to be taken seriously. They outline their methods in a way such that even someone with no specialized experience in cryptography, hardware, or security can understand the broad strokes of their approach, while still delving into the detail necessary for a more experienced reader to understand exactly what and how they are doing at each step of the process. They establish the viability of their attack at every step in the process, from the moment the DRAM module loses power to the final decryption of the target data hosted on a modern laptop. I can find very little to criticize in either the approach taken towards the attack itself or the presentation.

The final thing discussed in this paper is possible countermeasure to be taken against their attack.

One kind of counter measure they suggest pertains to precautions the owner of the computer can take to make it more difficult or impossible to execute these attacks. The first of these discussed is to prevent the computer from booting from external media (USB drives, etc.). While this is generally a good idea, the first section of this paper does a good deal to show why this is not necessarily sufficient protection. It was clearly shown that it is possible to remove the DRAM with little to no decay and place it in an attacker owned device, bypassing this protection. Another of these kinds of protections is very simply to turn off the device when it's left unattended. This is, in my opinion, the best of all suggested countermeasures the authors propose simply because it's easily implementable and foolproof — doing this and keeping an eye on it for

a minute afterwards guarantees that the DRAM decays fully, making it unrecoverable. The final countermeasure of this type is to prevent the DRAM from being removed at all. As the attack on BitLocker shows, this is not enough on its own to prevent an attack. However, when combined with one or both of the above precautions this would be sufficient to make a successful attack very difficult or impossible to carry out.

The second set of precautions relate to measures the OS or programs running on it could take to prevent the data being taken. The simplest measure is simply wiping the memory, either at boot time or after the program is done using the encrypted data. The issue with this is that, again, if an attacker simply pulls power and moves the DRAM module to a new computer, this protection is circumvented. Another is to avoid storing the results of computation made using these keys (key schedules, for example). The goal with this is to deny the attacker the means used to recover keys after decay, making the attack computationally intractable. Unfortunately this attack comes at the expense of speed, as these computations are generally expensive. A similar approach simply performs some transformation on the keys so that they are still usable, but more difficult to find. So long as the transformation remains unknown to the attacker, this should provide a solid defense against attack.

The final measures relate to things that can be done at a hardware level to prevent attacks. The first of these is to either increase the DRAM decay rate, or to add specialized key-store modules that are erased when power is lost. Neither of these would protect current computers, and there is a good deal of desire to keep DRAM decay rates high so there is no bit decay in day-to-day usage. The other suggestion is to implement the encryption on the disk controller itself. I like this idea a lot, because it deals very well with every avenue of attack outlined in this paper (if the keys are never stored in memory, it is impossible to recover them from memory). However, the authors did not analyze any systems like this.

The authors did briefly mention a fourth countermeasure, Trusted Computing, but they only did so enough to establish that this countermeasure does not work against their attacks. Trusted computing prevents a malicious OS from loading keys into memory, but does nothing to protect the keys once they are already there, which is what this paper focuses on.

Overall, the authors conclude that there is no easy, guaranteed method to protect current-day computers from this kind of attack. Specifically, they mention that laptops are highly vulnerable to these and that any hardware changes that may lower these vulnerabilities will take both time and money to create and implant on a wide reaching scale. They suggest that DRAM may have to be treated as untrusted, but that this is not possible without wide-reaching changes to how and where cryptographic keys are stored.

Personally, the conclusions I have drawn from this paper are primarily that my computer is not nearly as safe as I thought even using hard drive encryption systems, and that I ought power down my computer when it’s left unattended.