```
SS                      SS      SS               SS   SS SSSSSS SSSSSS SSSS
SS                      SS                       SS   SS SS     SS       SS
SSSSSs,  SSSSs, SSSSSs, SSSSSs, SS ,sSSSs, db.db SS   SS SS     SS       SS
SS  `SS     `SS SS  `SS SS  `SS SS SS` `SS USSSP SS   SS SSSSS  SSSSS    SS
SS   SS ,sSSSSS SS      SS   SS SS SSSSSSS  USP  SS   SS SS     SS       SS
SS  ,SS SS`  SS SS      SS  ,SS SS SS,       Y   SS,  SS SS     SS       SS
SSSSS*' `*SSSSS SS      SSSSS*' SS `*SSSS        `*SSSSS SSSSSS SS     SSSS
```

Learning about the BIOS

2019-04-28 1.6k words 9 minute(s) bios low level platform

or why do we discuss how to authenticate the user to the machine, but never the machine to the user

My path into low-level security

I have been away for a while as you may or may not have noticed and the reason for that is a great one! I am learning new things and as usual I will try to share my notes here. They are going to be chaotic and I can not give you any guarantee that I got it right, so please let me know if something looks weird ;)

The BIOS and system management mode internals

BIOS is part of what we call firmware, living in the flash chip soldered to your motherboard. Its main job is to configure your hardware and it is responsible for the basic I/O system. The BIOS also loads the OS bootloader (funny hum?) which is responsible for loading the fully featured environment you are using to read this blog post (yes, the OS!) and the OS is also the first on the chain to be located in the hard drive, being our very first loaded software.

So the bootloader loads the OS or our Hypervisor, which is basically handling resources and is responsible for managing the applications, which … do whatever the application is supposed to do (or not!). Seems pretty simple and straight forward, but it is a bit more complicated. As I found out, the BIOS is IMHO the most complex, sensitive and powerful “OS” we have in our machines, for reasons that I could have thought about but you know… Never really got the time to think about it :P

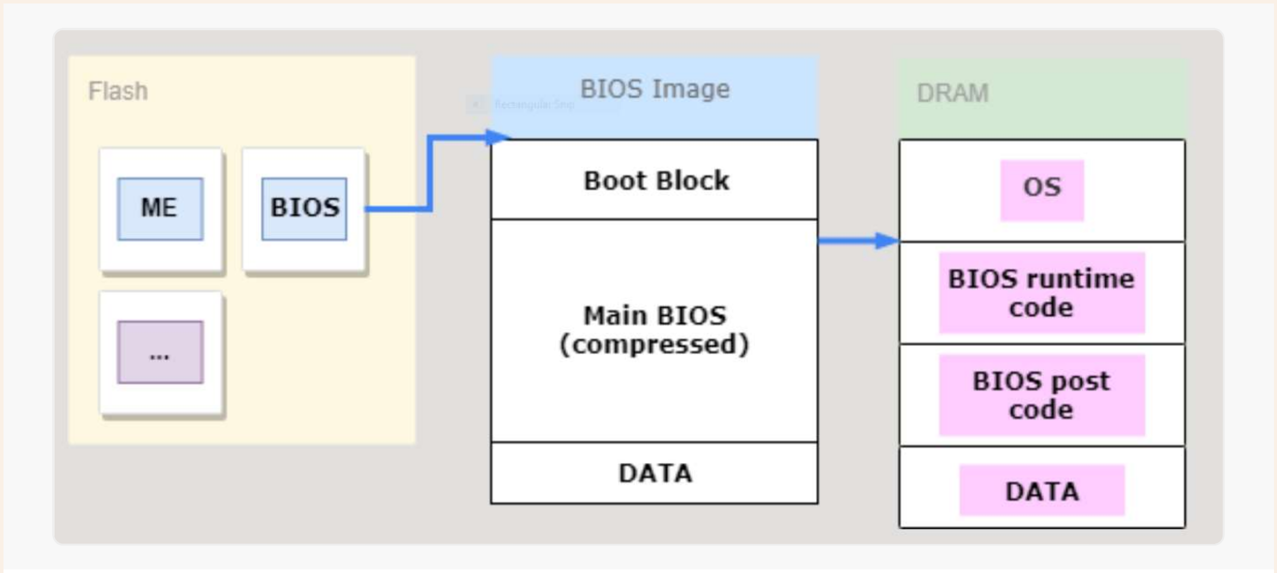
The BIOS

BIOS is the only code in the platform that knows the details of the motherboard. It acts as a abstraction layer between the motherboard and silicon and between the OS and the hardware (at least as long as we don't have an OS running). As I am coming from the malware community, my first thoughts always go this way when learning about a new technology and thinking about security. So why someone would like to attack the BIOS, if it is so much easier to attack a web application right? So I made a short list of why someone would prefer to get the harder way: first, if your malicious code is in the BIOS, you achieved serious persistence. Let's be honest, how often do you update your firmware / BIOS code? And now think about people who don't really focus on security and have normal lives… See what I mean? Second, it is a good place to hide, since few are actually checking it. How many AVs are running in the firmware level again? Not many right? And finally, the coolest one is, from the BIOS you are able to get to the SMM or any other portion of the system and when talking about x86, the SMM is the real God mode!

So, once the BIOS is backdoored, you can infect the bootloader, the hypervisor, the OS, any application and of course do any useful thing you wish. Sweet… the issue on actually backdooring the BIOS is, if one single bit is corrupted, the system will not boot… and everything turns a bit bitter.

What does the BIOS do anyway?

Right after you press the power on button, the firmware will select something called "flat protected mode" and check for CPU microcode updates. Then it will set the CPU cache as RAM (something called CRAM) up and do all the chipset initialization like the memory and some configuration and tests. If all the tests pass, the firmware is copied from the flash to the memory, the memory layout is set up (the so called PAM registers) and finally we have a STACK (which is not in the CPU cache anymore) !



BIOS code unpacking in memory

Lots of magic starts then like configuring the DRAM, the I/O and finally interrupts are enabled (PIC, LAPIC, APIC, and so many acronyms that I am still getting used to…) and the Interrupt tables are created (IVT & IDT). The code goes on setting up the timers and the memory caching control before finally starts the processor discovery and initialization. Just after having the processor in place, the I/O devices are going to be started up and the whole process of discovery, enumeration, detection and execution of the PCI devices are up, followed by the creation of the memory map and the non-volatile storage, where the hand off to the bootloader takes place.

As you see, the BIOS does a bunch and not only small things but basically the most important things. It literally set ups the computer for you :)

With that in mind and after going to this list, maybe it makes sense for you now why I think we are not giving the BIOS the attention it deserves?

Why is the BIOS so sensitive?

The BIOS is (was?) considered the root of trust, which means, once the system boots, you can start a so called trustworthy computation and assume everything is alright. As it is the very first code that is going to run into the processor, it is also code which is able to modify the OS and also has fully privileged access to all hardware components, without mentioning again that it provides the SMM code…

What has been done to protect the BIOS?

Besides the basics – like write protection and sandboxing – lots of other technologies have been placed around the BIOS to secure the "root of trust", like the TPM, Intel Boot Guard and SGX. Also it is important to mention that the EFI standard offers pretty neat features for developers, giving them way more flexibility than before.

The Trusted Platform Module (TPM)

The TPM was the first attempt to create measurement algorithm which gives a post factum notification, that the executed firmware code is not exactly what $whatever is expecting. The good side on this is, if the measurement is not what you expect to be, your cryptographic keys are safe. One neat implementation which is making use of this technology is Bitlocker from MS.

Intel Boot Guard

The Intel Boot Guard offers two different modes: the measured boot and the verified boot mode. In both cases we have a trusted piece of code, provided by the processor, which reads a authenticated code module (ACM) from the flash, measures the next block of code and then checks against the former block. This trusted code is special since it is provided by the CRTM (Core root of trust for measurement) and it is ROM – based. This CRTM is an immutable portion of the host platform's initialization code that executes upon a host platform reset and acts as an anchor right in the beginning of the cycle.

This way, the CRTM extends the TPM's checks with the hash of the measured block.

The Trusted Execution Technology (TXT)

TXT focus was a bit different than the technologies before, since it isn't there to check and validate the root of trust but it tries to shorten the super long chain of trust existing at this stage: RESET > BIOS / UEFI > Bootloader > OS loader > OS kernel and replace it with $stuff that are not trusted. So we have $stuff > TXT_Lauch > Hypervisor/ OS loader.

Through TXT is possible to separate the root of trust for platform measurement and exclude the BIOS and Boot from the chain of trust. TXT achieves it by resetting the chain without actually restarting the platform like "magic".

UEFI Secure Boot

The UEFI Secure Boot performs this way a check before handing it off to the next ring in the chain!

I will probably have UEFI at some point here too – here is just the preliminary idea!

Intel Software Guard Extensions (SGX)

Intel SGX promises to shorten the Chain of Trust by eliminating not only the BIOS and the whole firmware (which includes other devices, like network devices, Intel ME, GPU…) but also most of the OS (including the OS's kernel). SGX achieves it by keeping the data and code of processes running inside SGX enclaves inaccessible to the kernel and encrypting any DRAM pages used by the process automatically.

Good to point out is, that SGX processes are software implementations and the SGX binaries are not encrypted by default. It is possible to create software impossible to reverse engineer but this also depends on the enclave development itself.

Intel SGX does not replace the secure boot technology!

Source:

`https://software.intel.com/en-us/articles/intel-sdm#combined`

Author: barbie aka Thaís