SGX: the good, the bad and the downright ugly

2014-01-07

Shaun Davenport

Florida Institute of Technology, USA

Richard Ford

Florida Institute of Technology, USA

Editor: Helen Martin

Abstract

A brand new instruction set coming to Intel's processors in the near future has tremendous potential implications both for malware authors and for defenders. Shaun Davenport and Richard Ford describe the SGX technology and how people might use it.

Table of contents

One might be forgiven for having no idea what the acronym SGX stands for, especially with respect to the *Intel* chipset. Even a careful search of *LexisNexis Academic* failed to turn up any useful information. However, these three letters may prove to be the most significant thing to happen in the anti malware space in 2014. SGX stands for 'Software Guard Extensions' and it has the capacity to dramatically change long-held assumptions about how different software packages can coexist and, to some extent, battle each other in memory on untrusted platforms. This has tremendous implications both for malware authors and for defenders, as a whole new set of possibilities now exist.

One of the first articles we came across about the technology was a great post on Joanna Rutkowska's *Invisible Things* blog [**1**]. That post and its follow-up are worth reading for Joanna's take on what could be done with the new instructions. The blog post pre-dated the release of any technical documentation from *Intel* – now that this is available [**2**], we are in a position to take things a little further.

So, what exactly is SGX? Put simply, SGX is a brand new instruction set coming to *Intel*'s processors in the near future. While it may not make it to the desktop (this really is to be determined), it seems likely that it will be a big part of cloud servers in the future. The objective of SGX is to provide secure 'enclaves' in which data and code can execute without fear of inspection or modification. Coupled with remote attestation, it essentially attempts to allow developers to build a root of trust even in an untrusted environment.

As we have never seen a chip with SGX on it in the real world, we will take a rather lengthy quote from *Intel*'s website [**2**] to detail the intent of the new instruction set:

'Much of the motivation for *Intel®* SGX can be summarized in the following eight objectives:

1. Allow application developers to protect sensitive data from unauthorized access or modification by rogue software running at higher privilege levels.
2. Enable applications to preserve the confidentiality and integrity of sensitive code and data without disrupting the ability of legitimate system software to schedule and manage the use of platform resources.
3. Enable consumers of computing devices to retain control of their platforms and the freedom to install and uninstall applications and services as they choose.
4. Enable the platform to measure an application's trusted code and produce a signed attestation, rooted in the processor, that includes this measurement and other certification that the code has been correctly initialized in a trustable environment.
5. Enable the development of trusted applications using familiar tools and processes.
6. Allow the performance of trusted applications to scale with the capabilities of the underlying application processor.
7. Enable software vendors to deliver trusted applications and updates at their cadence, using the distribution channels of their choice.
8. Enable applications to define secure regions of code and data that maintain confidentiality even when an attacker has physical control of the platform and can conduct direct attacks on memory.'

That's a pretty nice set of claims – so much so that it could be a real game changer if SGX delivers on its promises. However, as we shall see in this article, while trust sounds like a good thing, it is most definitely a double edged sword.

Using *Intel*'s roadmap, it is pretty clear to see one of the problem spaces *Intel* was intending to address: trustworthy cloud computing. The use-case for an application designer is pretty straightforward. If software and hardware could be 'sealed' in some way to prevent an attacker from examining data in main memory, even if the attacker had administrator level privileges on the machine, not only could the confidentiality and integrity of data in the cloud be protected, but the algorithms and design of cloud hosted applications could also be hidden from prying eyes.

How does SGX work?

The core idea of SGX is the creation of a software 'enclave'. The enclave is basically a separated and encrypted region for code and data. The enclave is only decrypted inside the processor, so it is even safe from the RAM being read directly.

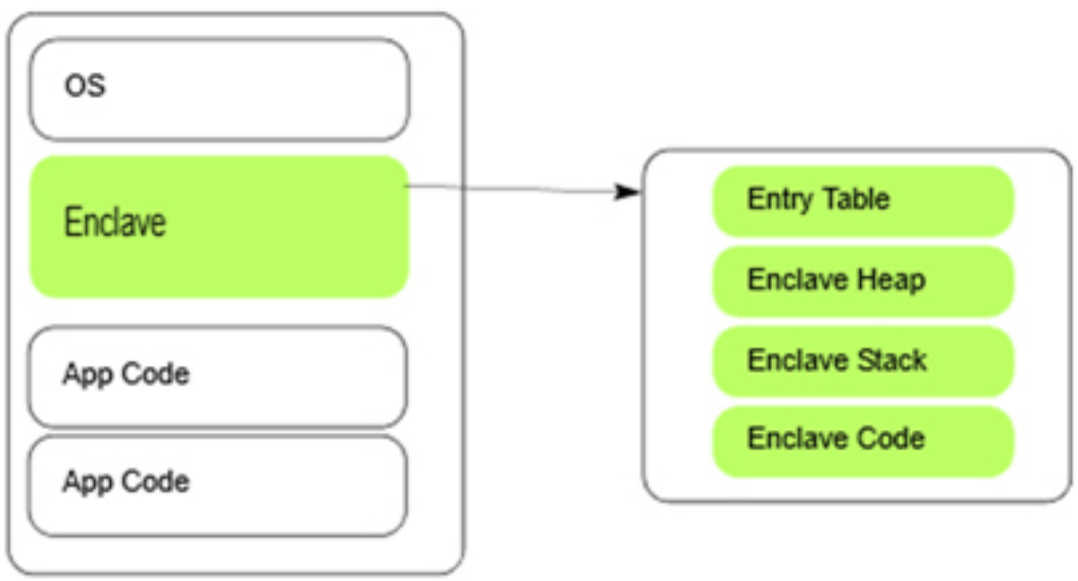

Figure 1. An enclave within the application's virtual address space. (Image source: Intel Software Guard Extensions Programming Reference.)

Creating an enclave is fairly straightforward. As enclave creation is a privileged instruction, the operating system is the intended entity to create it. Thus, we expect an API to be handling requests from user land applications trying to create enclaves. This has the added benefit of giving the operating system the choice to implement some sort of access control on the creation of enclaves. However, direct creation of an enclave should be possible if the software making the request has the appropriate privileges.

As the enclave leverages strong encryption, key generation and management are central to the strength of the security guarantees provided by the technology. The keys used for SGX enclaves are generated by the new instruction 'EGETKEY'. The key is a combination of three factors. First are the SGX Security Version Numbers, in which 'Some of the version numbers indicate the patch level of the relevant phases of the processor boot up and system operations that affect the identity of the SGX instructions' [**3**]. Second is the device ID, which is a 128-bit unique number tied to the processor. The last is the 'Owner Epoch', which gives the owner the ability to add some more entropy to the keys.

Armed with these keys, several new possibilities arise. One of the most powerful features is the ability for an enclave to attest to a remote server reliably. The new instruction 'EREPORT' creates a cryptographic report about an enclave which a remote machine will be able to examine to see if it was generated by SGX. A complete description of the remote attestation features of the SGX instruction set can be found in an *Intel* whitepaper [**4**].

Working with enclaves is particularly interesting when we consider debugger behaviour. An enclave can be debugged, but only if it consents to this activity explicitly. As per Section 7.2.1 of [**3**], if the enclave has not opted into debugging, the entire enclave should appear as a 'giant instruction' to the debugger. This is a boon to those wishing to protect their algorithms, but will play havoc with white hat reverse engineering.

The documentation is fairly clear in stating that while a VM can run an enclave, an enclave cannot be meaningfully emulated. As such, the standard reverse engineering trick of running questionable code inside a VM and gathering information about it is not possible.

Uses of SGX

Now that we know a little more about the SGX technology, it is worth taking a look at how people might use it. As is so often the case, uses range from the good to the bad, and, alas, the downright ugly.

The good

In the right hands, SGX can be a very powerful tool, assuring privacy and protection from malware even when running on an insecure system. For example, running a web browser inside an enclave would prevent even privileged malware from gaining easy access to all your information (though malware can still simply take snapshots of the rendered window). Enclaves would make it harder for malware to take key ring passwords out of memory. VMs could use enclaves to prevent the hypervisor viewing some critical information that only gets decrypted after attesting to a remote server. Video games could put most of their logic code inside an enclave in an attempt to stop some forms of wallhacks/aimbots/etc. Kernels could be made massively more resistant to tampering and hooking. The possibilities are endless.

The bad

Unfortunately, SGX is also a prime weapon for use in malware. For better or worse, it currently looks like *Intel* will not be giving the option for 'trusted anti-malware vendors' to access the contents of enclaves to make sure they are safe. Thus, malware can, in principle, freely create enclaves to prevent the operating system/hypervisor/anti malware from knowing what it is executing. Coupled with ubiquitous connectivity, the spectre of small loaders downloading sophisticated packages of malware remotely via an encrypted link rears its head.

On the bright side, as enclaves are not able to handle exceptions inside themselves, anti malware products might still be able to determine if there is malware running inside them from file IO and other IO. Furthermore, operating systems could choose only to give whitelisted programs permission to create an enclave from the enclave creation API. However, should a piece of malware successfully burrow down to Ring 0, the entire range of SGX functionality would become available to the malware author.

Let's run though some scenarios.

Scenario 1, the botnet creator:

Normal botnet operation is straightforward: after infecting a computer, the bot phones home and downloads and updates malware on the zombie computer. With SGX, the attacker could create an enclave, perform remote attestation with their C&C (command and control) server from inside the enclave, set up some private-public key encryption based on their SGX keys, and receive a payload to execute inside the enclave or any other commands from the C&C server. Furthermore, by leveraging strong encryption, none of this behaviour can be emulated or tracked, with the exception of the C&C traffic itself (which, of course, is encrypted).

This would be a terrible adversary to face in the wild. The defender cannot scan for the malware in memory and cannot create a signature for it. The only way to detect it at this point would be to examine the effects (such as file I/O).

Scenario 2, the video game hacker:

Just as video games can use enclaves, video game hackers can use them too. Currently, most forms of anti cheat technology simply check for signatures of known wallhacks/aimbots/etc. in memory. Attackers could simply put their wallhacks/aimbots/etc. inside an enclave to prevent *VAC* or *Punkbuster* from even knowing that it is running.

Just like the 'good' possibilities, there are infinite possibilities for 'bad'. Potentially, however, it gets even worse.

The ugly

Joanna Rutkowska raised the topic of inter-process communication on her blog, saying: 'For any piece of code to be somehow useful, there must be a secure way to interact with it.' We agree with that, but until some form of secure input/output exists, we cannot consider many of the use cases with SGX to be bullet-proof. From a pure security perspective, it is a step in the right direction. Unfortunately, with the full release of the SGX Reference Manual, it appears that SGX will not be able to provide any form of secure input/output. That's bad for the white-hat use case, but also bad for the black hat.

Furthermore, there is the terrible realization that for defenders to really benefit from SGX, *everything* will have to be run as an enclave, providing strong isolation of parts of code. Inter-process communication will, by definition, require real collaboration between processes. For interoperability purposes, holes will be punched in the defences; such holes will not need to exist on the attack side of the fence. Once the attacker has found *any* way in, it is not clear to us that they can be removed easily.

Conclusions

It is quite easy to find fantastic and exciting new ways for defenders to use the SGX instruction set to make their programs more secure, especially in the cloud. As such, this new extension to the architecture opens up some really interesting defence mechanisms whereby the actual state of a machine – or at least critical parts of it – can be determined remotely. For someone interested in protecting data, that is a powerful thing. However, the challenge comes with the idea of placing this technology into the hands of the attackers, who will doubtless be very early adopters of the instruction set, if only for a proof of concept.

There has been limited discussion about the possibility of a system that allows anti-malware vendors access to enclaves, but this seems impossible to do without having absolute trust in the anti-malware vendors themselves (not to mention the inevitable court cases that will centre on which vendors are deemed ‘trustworthy’ and which are not). A solution here will not be easy, and even if access were granted, attackers would probably turn their attention to the anti malware software itself as a vector of attack.

One last reflection. Amidst the recent revelations about the NSA’s wire tapping programs, industry observers might be forgiven for worrying about backdoors into SGX protected enclaves. This would be a kill-shot for adoption in some scenarios, and sets up an asymmetric battle between attackers and defenders where those that know how to peer through SGX’s encryption have an advantage that is probably not possible to overcome, at least not in the general case. Consider not only the possibilities of snooping, but of truly undetectable malware via such a backdoor.

All this seems a little premature, perhaps. *Intel*, as a company, certainly ‘gets’ security, and so it is hard to believe that some of the issues outlined here have not been anticipated, discussed thoroughly and mitigated. However, at the time of writing, we simply don’t know the state of affairs, despite having access to some pretty detailed documentation.

In all of this uncertainty, there is one thing we do know: the release and adoption of SGX-protected enclaves is likely to require a completely new approach to protecting our machines from the very malware SGX was designed to prevent. We are, then, truly confronted by the good, the bad, and the ugly.

Bibliography

[1] Rutkowska, J. **http://theinvisiblethings.blogspot.com/ 2013/08/thoughts-on-intels-upcoming-software.html (http://theinvisiblethings.blogspot.com/ 2013/08/thoughts-on-intels-upcoming-software.html)**.

[2] Hoekstr, M. Intel SGX for Dummies (Intel SGX Design Objectives). **http://software.intel.com/en us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx (http://software.intel.com/en us/blogs/2013/09/26/protecting-application-secrets-with-intel-sgx)**.

[3] Intel, Software Guard Extensions Programming Reference. **http://software.intel.com/sites/default/files/329298-001.pdf (http://software.intel.com/sites/default/files/329298-001.pdf)**.

[4] Anati, I.; Gueron, S.; Johnson, S.P.; Scarlata, V.R. Innovative Technology for CPU Based Attestation and Sealing. HASP, 2013.

Latest articles:

Behavioural Detection and Prevention of Malware on OS X (/virusbulletin/2016/september/behavioural-detection-and-prevention-malware-os-x/)

Malware on Apple's OS X systems is proving to be an increasing security threat, and one that is currently countered solely with traditional anti-virus (AV) technologies. Traditional AV technologies impose a significant performance overhead on the…

Throwback Thursday: Olympic Games (/virusbulletin/2016/august/throwback-thursday-olympic-games/)

In 1994, along with the Olympic Games came an Olympic virus, from a group of Swedish virus authors calling themselves 'Immortal Riot'. Mikko Hyppönen had the details.

Throwback Thursday: Holding the Bady (/virusbulletin/2016/07/throwback-thursday-holding-bady/)

In 2001, 'Code Red' caused White House administrators to change the IP address of the official White House website, and even penetrated the mighty Microsoft's own IIS servers. In August 2001, Costin Raiu

analysed the Win32/Bady.worm,

The Journey of Evasion Enters Behavioural Phase (/virusbulletin/2016/07/journey-evasion-enters-behavioural-phase/)

No malware author wants their piece of code to be easy to detect. Over time, several different approaches have been put into action to detect malware, and in response, malware authors have put into action different methods of evading them. This paper…

Throwback Thursday: You Are the Weakest Link, Goodbye! - Passwords, Malware and You (/virusbulletin/2016/07/throwback-thursday-you-are-weakest-link-goodbye-passwords-malware-and-you/)

Have you heard the one about the computer user who used their pet's name as their password? Just like jokes, it seems the old ones and the obvious ones are considered the best when it comes to users selecting their passwords. Martin Overton looks at…

VB2013 (Berlin) (/conference/vb2013/)

VB2012 (Dallas) (/conference/vb2012/)

Older conferences (/conference/vb-conference-archive/)

(/rss) **(https://twitter.com/virusbtn)** **(https://www.linkedin.com/company/virus-bulletin)** **(https://plus.google.co**