The Devil is in the Detail: Issues with Fine-Grained Trusted Execution Environments

Jonathan Adshead
jonathan.adshead@fau.de
Friedricht-Alexander-Universität Erlangen-Nürnberg

ABSTRACT

ToDo

1 IDEE

Trusted Execution Environments (TEEs) haben sich als wichtige und notwendige Idee im Bereich der Computersicherheit etabliert, um durch robuste und handwaregestützte Sicherheitsmechanismen insbesondere Schutz vor Bedrohungen wie Seitenkanalangriffen und Speicher Sichersicherheitslücken, welche zu Datenklau und Datenmanipulation führen können.

TEEs, wie etwa Intel SGX, ARM TrustZone oder Sancus, ermöglichen die Isolation und Signierung sicherer Anwendungsbereiche, die als Enklaven bezeichnet werden. /ToDo links

3.Hinweis

Das Prinzip der TEEs erzwingt eine duale Weltansicht der Hardware, welche in eine normale, möglicherweise kompromittierte und einer sicheren und isolierten Welt aufteilt. Dabei haben kompromittierte oder bösartige Systemsoftware in der normalen Welt keinen Zugriff auf den sicheren Speicherbereich der Enklave , welche einen isolierteren Bereich auf dem selben Prozessor und dem selben Adressraum hat.

4. Hinweis

Dadurch kann die Trusted Computing Base (TCB) reduziert werden. TCB umfasst alle Hardware und Software, die für die sichere Ausführung von vertrauenswürdigen Code notwendig sind.

Das Ziel ist es, den TCB zu minimieren, da eine größere TCB mehr potenzielle Angriffspunkte bietet und schwerer zu überprüfen und abzusichern ist. Eine kleinere TCB erhöht die Sicherheit, da sie weniger potenzielle Schwachstellen bietet und einfacher zu warten und abzusichern ist. So muss nur der Code, der in der Enklave läuft überprüft und kontrolliert werden.

Dennoch bieten TEEs nur eine grobgranulare Speicherisolation auf Hardware-Ebene und überlassen es den Enklaven- und Applikationsentwicklern, die Sicherheit auf Software-Ebene sicherzustellen.

Dies führt zu verschiedenen Herausforderungen, da nicht alle dieser Schwachstellen, wie Seitenkanalangriffe, vollständig behebbar sind. Trotz fortwährender Bemühungen von den Entwicklern, werden kontinuierlich neue Schwachstellen identifiziert. Selbst bei erfolgreicher Behebung bestehender Lücken manifestieren sich unaufhörlich neue mögliche Angriffspunkte, wodurch die Sicherheit der Enklaven gefährdet ist.

In einem solchen Fall könnte die gesamte Enklave kompromittiert werden, was die Sicherheit der gesamten Anwendung gefährdet. Wegen dieser kritischen Schwachstelle kann ein zweiter Ansatz zur Stärkung der Sicherheit notwendig sein, die Kompartimentierung der Software.

8. Hinweis

Gemeint ist dadurch, dass der gesamte Code in TEEs ausgeführt wird und mithilfe einer Softwarelösung wird das Programm aufgeteilt. Anstatt eine einzige TEE zu nutzen, werden mehrere TEEs implementiert.

Dabei ist es auch möglich verschiedene Sicherheitsstufen der einzelnen Enklaven zu unterstützen. Diese Aufteilung bedeutet, dass ein potenzieller Angreifer mehrere, wenn nicht alle TEEs kompromittieren müsste, um das gesamte System zu übernehmen, was die Angriffskomplexität und das Risiko drastisch senkt.

Da damit aber auch der größte Nachteil der Kompartimentierung auftritt, ist die Notwendigkeit, dass die einzelnen Programmteile nach dem Zero-Trust-Prinzip arbeiten müssen. Das heißt, dass jeder Programmteil als potentiell kompromittiert betrachtet werden muss.

Diese neuen Angriffsmöglichkeiten, die davor nicht in Betracht gezogen wurden, da das Programm als sicher galt, werden als Compartment Interface Vulnerabilities (CIV) bezeichnet.

Das Paper beschäftigt sich mit den Vorteilen, Problemen und Lösungsmöglichkeiten im Zusammenhang mit fein granularen TEEs. Insbesondere wird analysiert, wie und ob die Sicherheit durch die Implementierung solcher fein granularer Tees gesteigert werden kann, ohne dabei die Komplexität und Angriffsfläche der Anwendung unnötig zu erhöhen, sowie Möglichkeiten zur Minimierung von Sicherheitslücken und des Angriffsbereichs.

2 GRUNDLAGEN

Die Trusted Execution Environment (TEE) zeichnet sich durch drei wesentliche Eigenschaften aus. Sie gewährleistet die (1) Authentizität des ausgeführten Codes, indem sie sicherstellt, dass dieser tatsächlich von der beabsichtigten Quelle stammt und nicht manipuliert wurde. Sie sichert die (2) Integrität des Codes, indem sichergestellt wird, dass dieser während der Ausführung nicht verändert wurde und sie schützt die (3) Vertraulichkeit von Code, Daten und Laufzeitvariablen, indem sie sicherstellt, dass diese nur von privilegierten Parteien eingesehen werden kann.

2.1 TEE Design

Der genaue Aufbau und das Verhalten von Trusted Execution Environments (TEEs) variieren, lassen sich aber im Allgemeinen in zwei Kategorien einteilen: (1) TEEs, die sich einen Adressraum mit dem unprivilegierten Host teilen, und (2) TEEs, bei denen die CPU konzeptionell in eine normale und eine sichere Welt unterteilt ist.

In der ersten Kategorie erzwingt der Prozessor, dass nur die Enklave selbst auf ihren Speicherplatz zugreifen kann. Eine Enklave ist ein isolierter Speicherbereich innerhalb einer TEE, der dazu dient, Code und Daten vor unautorisiertem Zugriff und Manipulation zu schützen. Selbst privilegierte Benutzer und das Betriebssystem haben keinen Zugriff auf die innerhalb der Enklave gespeicherten

Daten. Programme innerhalb der Enklave dürfen jedoch auch auf den Adressraum außerhalb der Enklave zugreifen.

Diese Struktur ermöglicht eine flexible Datenverarbeitung, birgt jedoch das Risiko unsicherer Speicherzugriffe, was potenziell zu Sicherheitslücken führen kann.

In der zweiten Kategorie ist der Adressraum strikt in zwei separate Bereiche unterteilt, wobei keiner der beiden Bereiche auf den jeweils anderen zugreifen kann. Der Datenaustausch zwischen diesen beiden Welten erfolgt über das Trusted Operating System (TOS). Da das TOS privilegierten Zugriff hat, kann es einen geteilten Adressraum einrichten, in dem Daten sicher ausgetauscht werden können. Diese Methode bietet eine stärkere Isolation und somit ein höheres Maß an Sicherheit, da direkte Speicherzugriffe zwischen der normalen und der sicheren Welt verhindert werden, aber auf Kosten der Datenübertragungsraten.

In beiden Kategorien gewährleistet sowohl der Prozessor, dass extern kein Zugriff auf die Daten möglich ist und das TOS, dass die Enklave nicht kompromittiert ist. Die Daten werden ausschließlich im Prozessor entschlüsselt, und es existieren Mechanismen zur Verifizierung der Integrität der TEE.

Hinweis 15

2.2 TEE erstellen und verlassen

Der Enklave Entry/Exit-Prozess spielt eine zentrale Rolle in Trusted Execution Environments (TEEs) und umfasst die Mechanismen ecall (Entry) und ocall (Exit), die die Kontrolle über den Übergang zwischen der normalen und der sicheren Welt, der Enklave, ermöglichen. Einige Implementationen, wie Intel SGX (!Link), nutzen aber auch die niedrigstufigen Prozessorbefehle eenter und eexit.

Beim Betreten der Enklave erfolgt dies typischerweise über einen ecall. Hierbei wird von der normalen Welt aus eine spezielle Anweisung an das TOS oder, wie im Falle von Intel SGX, an den Prozessor gesendet, um den Eintritt in die Enklave zu initiieren.

Bevor die Ausführung der Enklaven Anwendung gestartet wird, bereinigt das TOS den Prozessorzustand, um sicherzustellen, dass keine Parameter, Flags oder andere Daten die Funktionsweise der TEE beeinflussen können. Die Parameter werden dabei als untrusted parameters an die CPU weitergegeben, damit diese erst überprüft werden, bevor sie als sicher gelten.

Der Enklave-Exit-Prozess wird durch ocall realisiert. Dies geschieht, wenn die Enklave ihre Ausführung beendet, eine externe Anfrage erhält, die den Übergang in die normale Welt erfordert oder ein Interrupt ausgelöst wird. Durch den Aufruf von ocall wird der Zustand der Enklave gespeichert, und die Kontrolle wird an das TOS zurückgegeben.

Das TOS bereinigt den Prozessorzustand, um sicherzustellen, dass keine sensiblen Informationen zurückbleiben, bevor der Prozessor aus dem Enklave-Modus in den normalen Betriebsmodus wechselt. Je nach Design werden die Daten für das unsichere OS direkt im Speicher abgelegt oder dafür an das TOS weitergegeben.

Die ecall- und eexit-Mechanismen bieten eine standardisierte Möglichkeit, den Eintritt und Austritt aus Enklaven zu verwalten und gewährleisten eine sichere Ausführung von Anwendungen innerhalb einer Enklave. Diese Mechanismen ermöglichen einen sicheren Übergang zwischen der normalen und der sicheren Welt unter strikter Kontrolle des Trusted Operating Systems.

2.3 Kompartimentierung

Die Kompartimentierung ist ein Konzept, das darauf abzielt, ein Computersystem oder einzelne Anwendungen in separate Bereiche aufzuteilen und diese möglichst isoliert voneinander operieren zu lassen. Diese Praxis beruht auf den Prinzipien der Sandbox und der Safebox, die eine sicherere und stabilere Systemumgebung ermöglichen.

Eine Sandbox ist eine isolierte Ausführungsumgebung, in dem ein Programm ausgeführt wird, welches keinen Zugriff auf Daten anderer Programme hat. Dies gewährleistet nicht nur die Sicherheit sensibler Informationen, sondern verhindert auch die Übernahme von Daten oder die Beeinflussung des Programmes durch potenziell kompromittierte Softwareteile.

Im Gegensatz dazu dient die Safebox dem Schutz der eignen Daten. Dies wird ermöglicht, indem nur privilegierten Prozessen Zugriff auf die Daten gestattet wird. Diese Zugangskontrolle verhindert den Zugriff auf Daten eines anderen Kompartimenten.

Die Kombination aus Sandboxing und Safeboxing ermöglicht eine Trennung zwischen den einzelnen Teilen eines Programmes und sorgt so dafür, dass eine Übernahme des Systems oder einen Zugang zu vertraulichen Daten erschwert wird.

3 SICHERHEITSPROBLEME VON TEES

Kein Programm ist vollkommen sicher. Obwohl das Konzept von Trusted Execution Environments (TEEs) darauf abzielt, Programme besser zu schützen, bestehen dennoch Angriffsmöglichkeiten. Jo Van Bulck et al. (Link) haben die von ihnen untersuchten Sicherheitslücken in zwei Kategorien eingeteilt: die Application Binary Interface (ABI)-Ebene und die Application Programming Interface (API)-Ebene. Die ABI beschreibt die Binäre Schnittstelle zwischen Funktionen/Programmen auf Maschinenebene, während die API die Schnittstelle zwischen verschiedenen Softwarekomponenten auf Quelltextebene definiert.

Mit ABI-Ebene sind damit die (1) potentiellen Schwachstellen beim Erstellen und Verlassen der Enklave gemeint. Dazu gehören Zustände und Flags der Register, die während der Enklaven-Erstellung gesetzt sein können und das Veralten des Programmes beeinflussen, sowie die Bereinigung des Stacks. Außerdem müssen beim Verlassen sämtliche Daten der Register, der CPU und sonstige Medien gelöscht werden, um Rückschlüsse auf die innerhalb der Enklave verarbeiteten Daten zu verhindern.

Die API-Ebene beschreibt das funktionale Interface der Enklave, insbesondere, wie Daten zwischen der sicheren und unsicheren Welt übertragen werden. Hierzu gehört die Überprüfung, dass (2) Zeiger innerhalb des erwarteten, freigegebenen Speicherbereichs liegen. Die (3) Länge der Strings muss überprüft werden und es muss sichergestellt werden, dass die Zeichenkette nullterminert. Wird das nicht gemacht, sind z.B. Seitenkanalangriffe möglich, die Informationen über den internen Speicher der Enklave preisgeben können.

Ein weiterer wichtiger Punkt ist die Gewährleistung, dass (4) der gesamte Speicherbereich, auf den ein Zeiger verweist, außerhalb der Enklave liegt, wenn ein Buffer oder ein größerer Datentyp übertragen wird. Wird dies nicht berücksichtigt, ist es möglich, dass der privilegierte Enklave Prozess den eigenen oder andere privilegierte Speicherbereiche überschreibt. (5) Integer-Überläufe

bei der Berechnung von Adressen müssen berücksichtigt werden. Auch dieser Angriff zielt darauf ab, die Grenzen des privilegierten Speicherbereiches der TEE zu durchbrechen.

Eine weitere potenzielle Angriffsmöglichkeit ist der (6) double fetch. Dieser Angriff nutzt einen Zeiger auf eine Adresse, worin sich ein weiterer Zeiger auf die eigentlichen Daten befindet. Der sichere Prozess muss die Adresse zwischenspeichern, während der Pointer überprüft wird. Wird dies nicht durchgeführt, kann ein kompromittiertes Programm die Adresse des inneren Zeigers ändern, da sich der Speicherbereich außerhalb der Enklave befindet.

4 FEINGRANULARE TEES

Unter feingranularen TEEs versteht man das Konzept, nur bestimmte sicherheitskritische Teile eines Programms in isolierten Enklaven auszuführen, anstatt das gesamte Programm in einen gesicherten Kontext zu packen. Dies ermöglicht eine gezielte Absicherung einzelner Komponenten, ohne dass die gesamte Anwendung in TEEs laufen muss.

4.1 Vorteile feingranularer TEEs

Hinweis 23 Daraus verspricht man sich mehrere Vorteile. Der offensichtlichste Vorteil ist die Minimierung der Angriffsfläche. Durch die Aufteilung des Programms in mehrere kleinere Kompartimente ist das Programm nicht mehr als Ganzes angreifbar, sondern nur noch die einzelnen, isolierten Kompartimente. Diese Trennung bedeutet, dass selbst wenn ein Angreifer erfolgreich in eine Enklave eindringen kann, der Zugang zu den anderen Programmen erheblich eingeschränkt ist.

Zusätzlich vertrauen sich die einzelnen Kompartimente nicht mehr Gegenseitig.

Jede Enklave operiert unabhängig und sicher, ohne auf die Integrität der anderen angewiesen zu sein. Durch diese Eigenschaften ermöglicht die feingranulare Struktur außerdem die Isolierung von Schwachstellen. Sicherheitslücken in einem Kompartiment haben keine Auswirkung auf die übrigen Kompartimente. Ein erfolgreicher Angriff auf eine ein einzelnes Kompartiment kann daher keine oder nur wenige Daten freigeben.

Darüber hinaus erlaubt die feingranulare Struktur die Möglichkeit der Implementierung unterschiedlicher Sicherheitsstandards für einzelne Enklaven. Bei der einzelnen Aufteilung des Programms kann statisch ermittelt werden, welche Teile aufgeteilt werden können und welche besonders kritische Funktionen enthalten. Hinweis 26

Zusammengefasst bieten fein granulare TEEs durch die Reduzierung der Angriffsfläche, die Isolierung von Schwachstellen, die gegenseitige Unabhängigkeit der Kompartimente und die Möglichkeit, spezialisierte Sicherheitsstandards zu implementieren, eine flexible, transparente und robuste Verbesserung der Sicherheit moderner Systeme.

4.2 Sicherheitsprobleme feingranularer TEEs

Problem + Lösung, wenn vorhanden

4.3 Ziel von feingranularer TEEs

Idealfall/Wunsch

5 FAZIT

große Änderung durch feingranularen TEEs? Fazit von meinem Text

Fazit vom den 2 vorgegeben Paper + mögliche Fehler in der Arbeit/denkweise (z.B. –> nur Poof of concept, Sicherheitslücken wurden gepatcht)

BEMERKUNGEN

das Referenzieren mit BibTeX muss ich mir noch machen.
bei 3. kann/soll ich so zittieren "Jo Van Bulck et al haben ..."? In den Arbeiten letztes Semster wurde es unter anderem so gemacht.