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

Die exponentielle Entwicklung im Bereich des Cloud-Computing und die zunehmende Verbreitung von Cloud-Diensten haben die Abhängigkeit von verteilten Computersystemen stark erhöht. Immer mehr Unternehmen und Organisationen verlagern Teile oder ihre gesamten Anwendungen in die Cloud, um von Skalierbarkeit, Flexibilität und Kosteneffizienz zu profitieren. Durch die Nutzung von sensiblen Daten und kritischen Anwendungen gewinnt die Sicherheit dieser Cloud Infrastrukturen eine wichtige Bedeutung, weil sie potentiell die sensiblen Informationen von Millionen von Nutzern und Organisationen verwalten.
Trusted Execution Environments (TEEs) sind dadurch zu einer wichtiger Technologie geworden, um die Sicherheit sensibler Daten und kritischer Anwendungen zu sichern. TEEs bieten eine sichere und geschützte Umgebung, in der sensible Daten vor potenziellen Bedrohungen geschützt sind. Die Nutzung von Cloud Diensten und die Verlagerung sensibler Daten in große Rechenzentren zeigt, dass TEEs als Schlüsselkomponente in der Datensicherheit zu sehen sind.
Doch trotz ihrer wichtigen Rolle sind TEEs nicht frei von Schwachstellen. Das Grundprinzip der TEEs erzwingt eine duale Weltansicht in den Hardwarekomponenten, welche in eine normale, möglicherweise kompromittierte und eine sichere und isolierte Welt aufgeteilt ist. Dabei können kompromittierte oder bösartige Systemsoftwareteile in der normalen, bzw. kompromittierten Welt keine Daten auf den sicheren Speicherbereich der Enklave lesen, welche einen isolierten Bereich im Adressraum bezeichnet, der aber näher in 2.1 beschrieben wird. Vor allem diese Schnittstelle zwischen den TEEs und zwischen TEE und unsicheren Welt stellt einen potenziellen Angriffsvektor dar. Diese Schwachstelle birgt das Risiko von Datenlecks und Datenkorruption, sowie Seitenkanalangriffen oder Rollbackangriffen. Ein erfolgreiches Ausnutzen von Angreifern könnte nicht nur sensible Informationen kompromittieren, sondern auch die Integrität der gesamten TEE gefährden.
Dieses Paper legt den Fokus auf die Sicherheitslücken zwischen den Kompartimenten, insbesondere zwischen sicheren Anwendungen innerhalb einer TEE und Anwendungen in der unsicheren Welt. Der Fokus liegt dabei auf Angriffen, die zu Datenlecks und Datenkorruption führen können. (? Zusätzlich wird diskutiert, ob Risiken gemindert und die Sicherheit von TEEs gestärkt werden können)

2 GRUNDLAGEN

Die Trusted Execution Environment (TEE) zeichnet sich durch drei wesentliche Eigenschaften aus, die ihre Funktionalität und Sicherheit beschreiben. Sie gewährleistet (1) die Authentizität des ausgeführten Codes, indem sie sicherstellt, dass dieser tatsächlich von der beabsichtigten Quelle stammt und nicht manipuliert wurde. Sie sichert (2) die Integrität des Codes, indem sie während der Ausführung sicherstellt, dass dieser nicht verändert wurde und sie schützt (3) die Vertraulichkeit von Code, Daten und Laufzeitvariablen, indem sie sicherstellt, dass diese nur von privilegierten Parteien eingesehen werden kann.
Aufgrund der Interaktion von Programmen innerhalb der TEE und Programmen außerhalb, kann die TEE auch als Kompartiment betrachtet werden. Diese Struktur ermöglicht eine klare Trennung zwischen den unterschiedlichen Kompartiments und ermöglicht so eine bessere Kontrolle über den Zugriff.

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, jedoch auf Kosten der Datenübertragungsraten.
In beiden Kategorien gewährleistet sowohl der Prozessor, dass extern kein Zugriff auf die Daten möglich ist als auch das TOS, dass der Code in der Enklave sicher ist. Ein weiterer Grundgedanke beim designen einer TEE ist die Annahmen, dass die TEE in einem System ausgeführt wird, das als kompromittiert betrachtet wird und Angreifer die volle Kontrolle über sämtliche Hardware haben.

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 Implementierungen, wie Intel SGX (!Link), nutzen aber auch die niedrigstufigen Prozessorbefehle eenter und eexit.

Beim Betreten der Enklave erfolgt der Wechsel in die TEE 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 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 Kompartiments.

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 ein Zugang zu vertraulichen Daten erschwert wird.

Das Initialisieren eines Kompartments lässt sich in 3 Stufen unterteilen. Zunächst erfolgt die Identifizierung, wo das Programm logisch gut unterteilt werden kann und wie feingranular diese Unterteilung sein soll. Anschließend wird die Durchsetzung der Grenzen vorgenommen, indem die Programme logisch voneinander getrennt und ausschließlich über eine API zur Kommunikation zugelassen werden.

In der letzten Stufe werden diese Grenzen weiter verstärkt. Hierbei werden die API-Schnittstellen klarer definiert, der Datenfluss möglichst reduziert und Strategien implementiert, wie mit potenziell unsicheren Daten von der unsicheren Welt umgegangen wird.

3 WARUM FEINGRANULARE TEES?

Ein wichtiger Begriff im Zusammenhang von TEEs ist die Trusted Computational Base (TCB). Die TBC beschreibt den Bereich eines Programms, der besonders vor externen Angriffen geschützt werden muss, da er mit sensiblen Daten arbeitet oder andere kritischen Funktionen übernimmt.

Wird nun ausschließlich dieser Teil der Anwendung in einer TEE ausgeführt, wird von einer feingranularen TEE gesprochen. In einer feingranularen TEE läuft also nicht das gesamte Programm, sondern nur ein spezifischer Teil davon.

Außerdem können mehrere feingranulare TEEs genutzt werden, indem mehrere kritische Funktionen in separaten TEEs ausgeführt werden, was die Sicherheit weiter erhöhen kann. Unabhängig von der Anzahl der verwendeten TEES, bieten sie mehrere Vorteile.

3.1 Minimierung der Angriffsfläche

Der Hauptgrund für die Verwendung von TEEs ist die Minimierung der Angriffsfläche. Diese Minimierung wird erzielt, da der Prozess, der innerhalb einer sicheren Enklave läuft, keinen uneingeschränkten Zugriff auf externe Ressourcen hat.

Ebenso haben Programme in der unsicheren Umgebung keinen direkten Zugriff auf Daten, die innerhalb der sicheren Enklave verarbeitet werden. Da TEEs in diesem Kontext als Kompartiments behandelt werden können, gibt es für die Kommunikation zwischen sicherer und unsicherer Welt eine definierte Schnittstelle, auch als API bekannt.

Diese Schnittstelle ermöglicht es, genau zu kontrollieren, welche Daten und in welchem Format sie von außen akzeptiert werden. Obwohl diese Schnittstelle die Sicherheit erhöht, stellt die API-Schnittstelle einen potenziellen Angriffsvektor dar und führt zu einer neuen Klasse von Sicherheitsproblemen, den Compartment Interface Vulnerabilities (CIVs).

3.2 Isolierung von Schwachstellen

Eine weitere wichtige Eigenschaft feingranularer TEEs ist ihre Fähigkeit zur Isolierung von Schwachstellen. Jedes Kompartiment operiert unabhängig und sicher, ohne auf die Integrität anderer Kompartimente angewiesen zu sein.

Dadurch haben Sicherheitslücken in einem Kompartiment keine Auswirkungen auf die anderen. Je mehr Kompartimente eingesetzt werden, desto widerstandsfähiger wird das gesamte System gegen Angriffe. Allerdings steigt mit der Anzahl der feingranularen TEEs

auch der Aufwand für die Erstellung, Überprüfung und Verwaltung der Daten sowie der gesamte Overhead. Dennoch kann ein erfolgreicher Angriff auf ein einzelnes Kompartiment möglicherweise nur einen Teil der Daten kompromittieren, während der Rest des Systems sicher bleibt.

3.3 Unterschiedliche Sicherheitsstandards

Ein Ziel neben der weiteren Minimierung der Angriffsfläche könnte die Einführung von verschiedenen Sicherheitsprofilen sein. Diese Strategie beinhalten nicht nur die Berücksichtigung der Interaktion zwischen den Kompartimenten bei ihrer Identifizierung, sondern auch die Festlegung welche Funktionen geschützt werden müssen und welche nicht.
Werden mit diesem Gedanken mehrere Kompartimente erstellt, können unterschiedliche Sicherheitsstandards eingeführt werden. So wird bei den Kompartimenten, in denen kritische Funktionen ausgeführt werden, besonders auf die Sicherheit geachtet. Durch diese Implementierung ist es möglich den Overhead von TEEs zu reduzieren, während ein nahezu gleiches Sicherheitsniveau beibehalten wird.
Erwähnenswert ist auch, dass bei der Identifizierung der Funktionen überprüft werden kann, ob bestimmte Funktionen einen ausschließlich lesenden oder schreibenden Zugriff erfordern. Funktionen mit einer dieser Eigenschaft können nicht zu Datenlecks oder Datenkorruption führen.

4 SICHERHEITSPROBLEME VON FEINGRANULAREN TEES

Wird die Sicherheit der Daten betrachtet, offenbart sich ein signifikanter Unterschied zwischen Trusted Execution Environments (TEE) mit einem gemeinsamen Adressraum und solchen mit einem separaten. Bei streng separierten Enklaven gestaltet es sich äußerst schwierig für unsichere Prozesse, die TEE zur Freigabe von Daten aus ihrem Speicher zu bewegen. Die Konfiguration des Trusted Operating Systems (TOS) kann entweder eine Spiegelung eines Teils des Adressraums oder den Zugriff auf externe Daten für den Datenaustausch in/von der Enklave ermöglichen. In beiden Fällen ist jedoch ein direkter Zugriff des Programms auf den Speicher der Anwendung nicht mehr gegeben.
Die Klassifizierung der Angriffsvektoren von Jo Van Bulck et al. erfolgt in zwei Hauptkategorien: Datenlecks und Datenkorruption. Diese Einteilung dient dazu, die Ziele der Angriffe zu charakterisieren und die potenziellen Auswirkungen auf die Sicherheit und Integrität der Daten zu bewerten.

4.1 Datenlecks

Unter Datenlecks versteht man Schwachstellen in denen ein Angreifer Informationen über das Layout, den Inhalt oder die Nutzung des Speichers der TEE gewinnen kann. Auch wenn es nicht möglich ist, die Daten der Enklave direkt zu lesen, kann die TEE durch externe Einflüsse dazu beinflusst werden, sich anders zu verhalten und so ungewollt Informationen preiszugeben. Das beginnt bei der Freigabe der letzten aufgerufenen Adresse, über die Anzahl der Aufrufe oder einzelnen Daten bis hin zum kompletten Speicherlayout der Enklave.
Durch den 2. Angriffsvektor ist es möglich den Callstack auszulesen. Bei dem Angriffsvektor handelt es sich um die sichere Speicherung und anschließende Bereinigung des Callstacks beim Erstellen und Verlassen einer TEE. Wird dies nicht ordnungsgemäß durchgeführt, kann der Angreifer z.B. lesen, welche Funktionen aufgerufen wurden und so Rückschlüsse über die Funktionsreihenfolge, Art der Funktion und Zusammenhang zwischen den Funktionen erlangen.
Eine weitere Schwachstelle bei dem Erstellen und Verlassen einer TEE betrifft andere Speicherzustände, wie die Registern. Nach dem 3. Angriffsvektor ist es für Angreifer möglich, nach dem Verlassen der TEE die Registerzustände auszulesen, wenn diese nicht ordnungsgemäß bereinigt werden. Selbst wenn das Bereinigen der Registerzustände in normalen Prozesswechseln üblich ist und automatisch erfolgt, kann es in einigen TEE-Implementierungen manuell erfolgen, was es zur Softwareverantwortung macht. Darüber hinaus muss bedacht werden, dass alle Angriffe, die auf die hinterlassenen Daten abzielen, auch zutreffen, wenn die TEE durch ein Interrupt gestoppt wird.
Die TEE muss sicherstellen, dass der Inputpointer komplett außerhalb der Enklave liegt. Dieser 7. Angriffsvektor ist nur bei TEEs möglich, welche sich einen Adressraum mit der unsicheren Welt teilen, da es sich sonst um unterschiedliche Adressräume handelt und die Daten nicht nebeneinander in dem Adressraum liegen. Bei einem solchen Angriff kann ein Angreifer versuchen, den Anfang des Zeigers auf eine Adresse kurz vor der Enklave zeigen zu lassen, jedoch befindet sich der Buffer dann zum Teil in der Enklave, da sich beide den Adressraum teilen. Deshalb muss die TEE immer überprüfen, ob die gesamten übertragenen Daten außerhalb der Enklave liegen, um Datenklau zu vermeiden.
Ein noch spezialisierter Angriff betrifft die Manipulation von Strings, wie im 5. Angriffsvektor beschrieben ist. Bei Verwendung einer Low-Level-Sprache wie C verhält sich ein String als Array aus Chars, wobei zusätzlich die Länge angegeben werden muss, was von Angreifern ausgenutzt werden kann. Durch absichtlich falsche Angaben zur Länge und das Weglassen der Nullterminierung kann ein Angreifer die Position jedes /0 im Speicher der Enklave identifizieren. Da die TEE den String weiterhin ausliest und vollen Zugriff auf ihren Speicher hat, geht sie fälschlicherweise davon aus, dass der String bis zum nächsten /0 reicht. Über die Zeit und Reaktion der TEE kann der Angreifer Daten aus der Enklave bekommen. Dieser Seitenkanalangriff kann aber noch weiter verfeinert werden. Überschreibt man die Variable, die definiert, wie ein String behandelt wird, kann dadurch jedes andre einzelne Byte aus dem Speicher gewonnen werden und Angreifer könnten ein volles Abbild der Daten der Enklave haben.

4.2 Datenkorruption

5 FAZIT

ToDo: Fazit zu meinem Paper

5.1 A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes

Meinung zum Paper

5.2 Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software

Meinung zum Paper