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

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

Kurzfassung

Mit dem Wachstum des Cloud-Computings und der zunehmenden Verlagerung sensibler Daten und kritischer Anwendungen in die Cloud gewinnen Trusted Execution Environments (TEEs) an Bedeutung. Das Hauptziel von TEEs ist die Minimierung der Angriffsfläche und die Isolation von Schwachstellen, um die Integrität und Vertraulichkeit der Daten zu gewährleisten. TEEs bieten eine geschützte Umgebung zur sicheren Verarbeitung und Speicherung sensibler Daten, sind jedoch nicht frei von Schwachstellen. Dieses Paper untersucht die Sicherheitslücken feingranularer TEEs und kategorisiert diese in zwei Hauptkategorien: Datenlecks und Datenkorruption. Es wird gezeigt, dass die Schnittstellen zwischen verschiedenen TEEs sowie zwischen sicheren Anwendungen einer TEE und unsicheren Anwendungen zu erheblichen Sicherheitsrisiken führen können. Die Ziele dieser Arbeit sind die Darstellung von potenziellen Angriffen und deren Auswirkungen, um die Sicherheit und Isolation von TEEs zu verbessern. Die Arbeit stützt sich auf die Studien „Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software“ [3] und „A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes“ [6].

1 Einleitung

Die zunehmende Entwicklung im Bereich des Cloud-Computing und Verbreitung von Cloud-Diensten haben die Nachfrage von verteilten Computersystemen stark erhöht. Immer mehr Unternehmen und Organisationen verlagern Teile oder ihre gesamten Anwendungen in die Cloud, mit der Erwartung, von Skalierbarkeit, Flexibilität und Kosteneffizienz zu profitieren [4]. Mit der Nutzung sensibler Daten und kritischer Anwendungen gewinnt die Sicherheit dieser Cloud-Infrastrukturen an Bedeutung, da sie potentiell die sensiblen Informationen von Millionen von Nutzern und Organisationen verwalten.

TEEs sind dadurch zu einer wichtigen Technologie geworden, um die Sicherheit sensibler Daten und kritischer Anwendungen in einer geschützen Umgebung zu gewährleisten. 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 (1) normale, möglicherweise kompromittierte und eine (2) sichere und isolierte Welt aufgeteilt ist [6]. Dabei können kompromittierte oder bösartige Systemsoftwareteile in der normalen, bzw. kompromittierten Welt keine Daten auf dem sicheren Speicherbereich der Enklave, welche einen isolierten Bereich im Adressraum bezeichnet, lesen. In Abschnitt 2.1 wird näher auf Enklaven eingegangen. Vor allem diese, durch die duale Weltansicht entstehende Schnittstelle zwischen verschiedenen TEEs und zwischen TEE und Programmen in der unsicheren Welt, stellen einen potenziellen Angriffsvektor dar. Diese Schwachstelle birgt das Risiko von Datenlecks und Datenkorruption, sowie Seitenkanalangriffen oder Rollback-Angriffen [2, 6]. Ein erfolgreiches Ausnutzen dieser Schwachstellen von Angreifern kann nicht nur sensible Informationen kompromittieren, sondern auch die Integrität der gesamten Enklave gefährden.

Dieses Paper legt den Fokus auf die Sicherheitslücken, sowohl zwischen den unterschiedlichen TEEs als auch zwischen sicheren Anwendungen einer TEE und Anwendungen der unsicheren Welt. Der Fokus liegt dabei auf Angriffen, die zu Datenlecks und Datenkorruption führen können.

2 Grundlagen

Die TEE zeichnet sich durch drei wesentliche Eigenschaften aus, die ihre Funktionalität und Sicherheit beschreiben. Sie gewährleistet (1) die Authentizität des ausgeführten Codes. Ein wichtiger Aspekt dabei ist die Möglichkeit der Remote Attestation. Diese erlaubt es, die Authentizität des Codes aus der Ferne zu verifizieren. Die TEE sichert (2) die Integrität des Codes, indem sie gewährleistet, dass dieser nicht verändert wurde. Außerdem schützt sie (3) die Vertraulichkeit von Code, Daten und Laufzeitvariablen [5].

Aufgrund der Interaktion von Programmen innerhalb der TEE mit Programmen außerhalb kann die TEE auch als Kompartiment betrachtet werden. Diese Struktur ermöglicht eine klare Trennung zwischen den unterschiedlichen Kompartimenten und ermöglicht so eine bessere Kontrolle über den Zugriff.

2.1 Trusted Execution Environment

Der genaue Aufbau und das Verhalten von TEEs variiert, lässt sich aber, wie in Abbildung 1 zu sehen ist, im Allgemeinen in zwei Kategorien einteilen: (1) die Single-World-TEEs, die sich einen Adressraum mit dem unprivilegierten Host teilen, und (2) die Two-Worlds-TEEs, bei denen die CPU konzeptionell in eine normale und eine sichere Welt unterteilt ist [6].

In der ersten Kategorie hat nur die Enklave selbst vollen Zugriff auf ihren Speicherplatz und kann die Daten im Klartext lesen. 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 einen besseren Datenaustausch, birgt jedoch das Risiko unsicherer Speicherzugriffe auf möglicherweise korrupte Daten, was potenziell zu Sicherheitslücken führen kann. Wie in Abbildung 1 dargestellt, können Prozesse der unsicheren Welt oder auch das Untrusted OS, nicht auf die Prozesse des Sicheren OS zugreifen. Auch das Lesen der Daten in der Enklave ist nicht möglich.

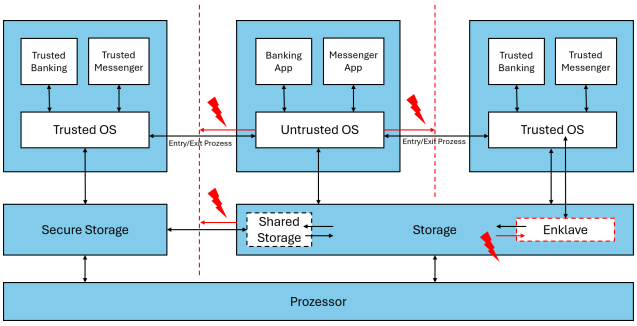

Abbildung 1: Links der Aufbau einer TEE mit separatem Adressraum, rechts eine TEE mit geteilten Adressraum in Form einer Enklave im Adressraum des unsicheren Host.

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. Wie in Abbildung 1 zu sehen ist es nicht möglich die Daten oder Prozesse der sicheren Welt einzusehen.
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. Das TOS übernimmt außerdem den Entry/Exit Prozess der TEE, in dem der Übergang zwischen der normalen und der sicheren Welt ausgeführt wird.
Neben den Angriffen auf das Application Binary Interface (ABI), welche auf Schwachstellen beim Erstellen und Verlassen der TEE abzielen, gibt es auch Angriffe auf das Application Programming Interface (API), welche versuchen, Sicherheitslücken während der Laufzeit zu nutzen [6].
Das Angreifermodell ist ein weiterer Grundgedanke beim Entwurf einer TEE. Die TEE wird unter der Annahme entwickelt, dass sie in einem System ausgeführt wird, das als kompromittiert betrachtet wird und Angreifer die volle Kontrolle über sämtliche Hardware haben.

2.2 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. Ein Kompartiment agiert als Safebox und behandelt alle anderen Programme als Sandbox. So wird kein Zugriff auf die eigenen Daten erlaubt.
Eine Sandbox ist, wie in Abbildung 2 dargestellt, eine isolierte Ausführungsumgebung, in der ein Programm ausgeführt wird, welches keinen Zugriff auf Daten anderer Programme hat. Die Applikation in der Sandbox kann, wenn überhaupt, nur über klar definierte APIs auf andere Programme oder Funktionen zugreifen, während anderen Applikationen vollen Zugriff gewährt wird. Dies gewährleistet nicht nur die Sicherheit sensibler Informationen, sondern verhindert auch die Übernahme von Daten oder die Beeinflussung des Programmes durch potenziell kompromittierte Softwareteile. Ein bekanntes Beispiel dafür ist eine Virtuelle Maschine (VM). Diese wird in einer isolierten Umgebung ausgeführt und hat keinen Zugriff auf jegliche Daten außerhalb der Sandbox.

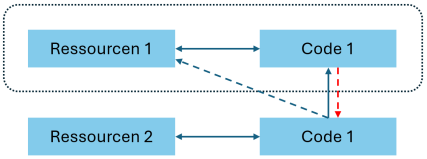

Abbildung 2: Aufbau einer Sandbox: Applikationen in der Sandbox können nur über klar definierte APIs auf andere Programme zugreifen.

Im Gegensatz dazu dient die Safebox, in Abbildung 3 dargestellt, 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. Andere Kompartimente können nur über eine klare Schnittstelle Daten übertragen, während die Programme innerhalb vollen Zugriff haben. Dies spiegelt genau die Funktionsweise einer Enklave wider, in der Programme innerhalb vollen Zugriff haben, deren Daten aber für Programme aus der unsicheren Welt nicht lesbar sind.

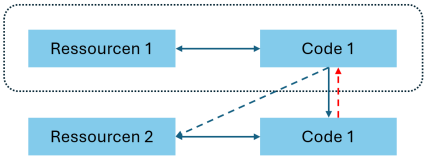

Abbildung 3: Aufbau einer Safebox: Applikationen außerhalb der Safebox können nur über eine klar definierte API auf Code innerhalb zugreifen.

Das Initialisieren eines Kompartiments lässt sich in 3 Stufen unterteilen. Zunächst erfolgt (1) die Identifizierung, bei der festgelegt wird, an welcher Stelle das Programm logisch gut unterteilt werden kann und wie feingranular diese Unterteilung sein soll. Anschließend wird (2) 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 (3) diese Grenzen weiter verhärtet. Hierbei werden die API weiter definiert, wie z. B. die Rückgabewerte und Pointer.
Wie in Abbildung 4 dargestellt, können Applikationen in verschiedenen Kompartimenten nur über eine klar definierte API untereinander Daten austauschen und haben keinen Zugriff auf andere Informationen.

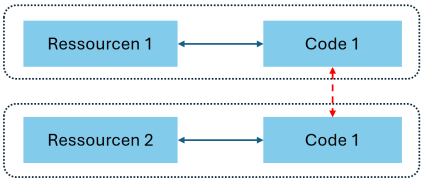

Abbildung 4: Aufbau von Kompartimenten: Applikationen in verschiedenen Kompartimenten können nur über eine klar definierte API Datein austauschen.

3 Ziele von feingranularen TEEs

Ein wichtiger Begriff im Zusammenhang von TEEs ist die Trusted Computing Base (TCB). Die TCB beschreibt den Bereich eines Programms, der besonders vor externen Angriffen geschützt werden muss, da er mit sensiblen Daten arbeitet oder andere kritische Funktionen übernimmt, wie z. B. das Interagieren mit sensiblen Daten. Wird nun ausschließlich dieser Teil der Anwendung in einer TEE ausgeführt, wird von einer feingranularen TEE gesprochen.
Außerdem können mehrere feingranulare TEEs genutzt werden, indem mehrere kritische Funktionen in separaten TEEs ausgeführt werden, was die Sicherheit weiter erhöhen kann.

3.1 Minimierung der Angriffsfläche

Das Hauptziel für die Verwendung von feingranularen TEEs ist die Minimierung der Angriffsfläche und damit die Erhöhung der Sicherheit.
Programme aus der unsicheren Umgebung haben keinen direkten Zugriff auf Daten, die innerhalb der sicheren Enklave verarbeitet werden. Da feingranulare TEEs in diesem Kontext als Kompartimente behandelt werden können, gibt es für die Kommunikation zwischen sicherer und unsicherer Welt eine definierte Schnittstelle, die durch die als Enklave Description Language (EDL) beschrieben wird.
Diese Schnittstelle ermöglicht es, genau zu kontrollieren, welche Daten in welchem Format von außen akzeptiert werden. Obwohl diese Schnittstelle die Sicherheit erhöht, stellt sie dafür einen neuen Angriffsvektor dar und führt zu einer neuen Klasse von Sicherheitsproblemen, den Compartment Interface Vulnerabilities (CIVs) [3].

3.2 Isolierung von Schwachstellen

Ein weiteres wichtiges Ziel feingranularer TEEs ist ihre Fähigkeit zur Isolierung von Schwachstellen. Ziel ist es, die Kompartimente sicherer zu machen und in der Zukunft die Unabhängigkeit der Kompartimente voneinander zu erreichen.
Es wird immer weiter darauf hingearbeitet, dass Sicherheitslücken in einem Kompartiment keine Auswirkungen auf die anderen Kompartimente haben und so eine Isolation der Sicherheitslücken entsteht. 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.

4 Sicherheitsprobleme von feingranularen TEEs

Wird die Sicherheit der Daten betrachtet, offenbart sich ein signifikanter Unterschied zwischen TEEs 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 bringen. Die Konfiguration des TOS kann entweder eine Spiegelung eines Teils des Adressraums oder den Zugriff auf externe Daten für den Datenaustausch in und aus der Enklave ermöglichen. In beiden Fällen ist jedoch kein direkter Zugriff des Programms auf den Speicher der Anwendung möglich.
Hugo Lefeuvre et al. [3] haben zwei große Kategorien von Sicherheitslücken identifiziert: Datenlecks und Datenkorruption. Diese Kategorisierung erweist sich als nützlich, da nahezu alle bekannten Angriffe auf TEEs in eine dieser beiden Kategorien fallen. Auch die Angriffsvektoren, die von Jo Van Bulck et al.[6] gefunden und untersucht wurden, lassen sich in diese Kategorien einteilen, was die Gültigkeit dieser Klassifizierung weiter unterstützt. Die Einteilung in Datenlecks und Datenkorruption ermöglicht eine Identifizierung der Angriffsziele und eine Einschätzung der potenziellen Auswirkungen auf die Sicherheit und Integrität der Daten.

4.1 Datenlecks

Die Kategorie der Datenlecks umfasst Angriffe, die darauf abzielen, Daten aus der Enklave zu extrahieren.

4.1.1 Angriff beim Erstellen und Verlassen

Ein besonders anfälliges Angriffsziel ist der Zustand der TEE beim Verlassen oder Beenden. Unabhängig davon, ob die Beendigung geplant oder durch ein Interrupt erzwungen wird, muss sichergestellt werden, dass keine Daten in einem lesbaren Zustand im Speicher verbleiben. Da das Bereinigen des Speichers eine explizite Softwareaufgabe darstellt, liegt es in der Verantwortung der Entwickler, dies korrekt zu implementieren. Selbst wenn das Bereinigen der Registerzustände in normalen Prozesswechseln üblich ist und automatisch erfolgt, ist diese Schwachstelle nicht zu unterschätzen. Wird dieser Prozess nicht ordnungsgemäß durchgeführt, können Angreifer Registereinträge oder den Stack auslesen und dadurch Rückschlüsse auf die Funktionsweise der Anwendung und die letzten Aktionen innerhalb der TEE ziehen.
Die Bedeutung dieser Sicherheitsmaßnahmen wird durch die potenziellen Konsequenzen unzureichender oder fehlerhafter Speicherbereinigung verdeutlicht. Angreifer könnten sensible Informationen wie kryptografische Schlüssel, Passwörter oder andere vertrauliche Daten extrahieren und somit die Sicherheit der gesamten Anwendung kompromittieren.

4.1.2 Angriff mithilfe von Pointern

Neben dem Zeitpunkt des Verlassens der TEE ist diese auch während der Laufzeit angreifbar. Konkret ist das über die Parameter möglich, die die TEE von unsicheren Prozessen von außen bekommt. Es ist essenziell, diesen Parametern nicht von Anfang an zu vertrauen. Diese Daten müssen zuerst überprüft und gegebenenfalls bereinigt werden. Beispielsweise muss überprüft werden, ob der

Inputpointer und der gesamte Speicherbereich, auf den er zeigt, außerhalb der Enklave liegen, was je nach Datentyp oder Struktur nicht nur aus einer einfachen Überprüfung besteht. Da die Größe des Pointers Teil der Eingabe ist und nach dem Angreifermodell davon ausgegangen wird, dass das komplette System möglicherweise kompromittiert ist, kann diesem Wert nicht vertraut werden.
Eine mögliche Angriffsmethode besteht darin, Situationen zu erzeugen, in denen die Größe wegen Überläufen nicht mehr richtig berechnet werden kann. Dadurch wird nicht erkannt, dass der Datenbereich tatsächlich den der Enklave schneidet. Aus diesem Grund ist es dringend notwendig, beim Berechnen von Adressen und beim grundsätzlichen Verwenden von unsicheren Daten, sichere Arithmetik zu verwenden, um fehlerhafte Berechnungen zu vermeiden.

4.1.3 Angriff mithilfe von Strings

Dieses Angriffsszenario kann aber Mithilfe der Manipulation von Strings noch weitergeführt werden. In Low-Level-Sprachen wie C verhält sich ein String auch als Pointer, der auf ein Array zeigt, wobei die Länge angegeben werden muss. Gelingt es einem Angreifer, den Datenbereich mit dem der Enklave überschneiden zu lassen, kann er nahezu den gesamten Speicher auslesen. Durch die absichtlich falsche Größenangabe und das Weglassen der Nullterminierung, die das Ende eines Strings definiert, kann der Angreifer, über das Programm in der TEE auf die Daten innerhalb der Enklave zugreifen. Damit kann der Angreifer über diesen Seitenkanal die Daten auslesen. Ein Seitenkanalangriff meint dabei, dass Rückschlüsse über die Daten z. B. anhand der Ausführungszeit oder der Cache-Zugriffe gewonnen werden können. Wird nun der String ausgelesen, kann der Angreifer anhand der Dauer feststellen, wann das nächste Nullbyte im Speicher liegt. Wird dies nun an beliebigen oder allen Speicheradressen gemacht, können alle Nullbytes in der Enklave identifiziert werden [6].
Je nach Implementierung der TEE gibt es verschiedene Arten, den Angriff durchzuführen. Beispielsweise ermöglicht das Intel SGX-SDK Angreifern, ein volles Speicherabbild zu generieren. Konkret ermöglicht das EDL-Attribut, welches das Ende eines String definiert, das Auslesen von Speicherinhalten. Standardmäßig wird das Nullbyte verwendet, aber durch das Überschreiben mit anderen Werten ist es möglich, alle entsprechenden Speicherinhalte zu lokalisieren. Wird dieser Vorgang bis 255 wiederholt, kann ein vollständiges Speicherabbild erstellt werden. Die Sicherheit der TEE ist damit vollständig umgangen worden. [1, 6].
Bei diesem Angriff ist anzumerken, dass TEEs nach dem Single-World-Prinzip deutlich anfälliger sind, da die Enklave in den Speicher des Angreifers eingebettet ist. Bei TEEs nach dem Two-Worlds-Prinzip werden die Daten nur über das TOS geteilt. Da aber in keinem System eine absolute Sicherheit möglich ist, gibt es auch hier Schwachstellen, die von Angreifern ausgenutzt werden können.

4.2 Datenkorruption

Angriffe dieser Kategorie versuchen die Daten zu überschreiben, um dadurch das Verhalten der TEE zu beeinflussen oder die Integrität zu untergraben.
Eine Möglichkeit, dies zu erreichen, besteht in der Manipulation von Register Flags. Insbesondere die Alignment Check (AC) Flag und die Direction Flag (DF) beeinflussen, wie Daten abgespeichert und gelesen werden. Durch das gezielte Setzen oder Zurücksetzen dieser Flags kann ein Angreifer die TEE dazu bringen, Daten anders als beabsichtigt abzuspeichern, was zu Datenkorruption führen kann. Auch in diesem Fall bieten Input-Pointer eine große Angriffsfläche. Wenn die Überprüfung der Input-Pointer und die arithmetischen Funktionen nicht korrekt implementiert sind, kann die TEE unabsichtlich ihren eigenen Speicher überschreiben, was die Integrität der Daten zerstört. Während andere Prozesse von außen nicht direkt in die Enklave schreiben können, ist dies für Programme innerhalb der Enklave sehr wohl möglich. Dies stellt ein erhebliches Risiko dar, da ein kompromittiertes Programm innerhalb der TEE bösartigen Code ausführen und somit die Datenintegrität gefährden kann.
Ein Beispiel für einen solchen Angriff ist der Double-Fetch-Angriff. Bei diesem Angriff wird die TEE dazu gebracht, dieselben Daten mehrfach aus dem Speicher zu lesen. Zwischen diesen Lesevorgängen kann der Angreifer die Daten verändern. Durch die gezielte Manipulation der Adressinformationen in den Eingabedaten kann der Angreifer die TEE dazu verleiten, auf spezifische, manipulierte Speicheradressen zuzugreifen. Dies führt dazu, dass die TEE vertrauliche Daten aus unsicheren Speicherbereichen liest und somit die Integrität und Vertraulichkeit der Daten, wie beispielsweise kryptografische Schlüssel oder Benutzerdaten, erheblich gefährdet werden.
Dieser Angriff kann auch als Seitenkanalangriff ausgeführt werden, bei dem die Zugriffszeit auf den Speicher genutzt wird, um festzustellen, ob Daten kürzlich verwendet wurden. Liegen die Daten im Cache, ist die Zugriffszeit kürzer, was dem Angreifer Informationen über die Nutzungshäufigkeit und den Speicherort der Daten gibt. Durch diese Methode kann der Angreifer die TEE dazu bringen, gezielt auf manipulierte Adressen zuzugreifen, wodurch die Integrität der Daten weiter kompromittiert wird. Das kann dazu führen, dass der Angreifer über Seitenkanäle Informationen über das Speicherlayout, Daten oder Funktionsaufrufe erhalten kann.
Diese Angriffe verdeutlichen, wie wichtig es ist, robuste Mechanismen zur Überprüfung und Sicherung der Datenintegrität innerhalb der TEE zu implementieren. Nur durch sorgfältige Überprüfung der Input-Pointer, den Einsatz sicherer arithmetischer Funktionen und das Vermeiden von unsicheren Speicherzugriffen kann die Integrität der TEE und ihrer Daten gewährleistet werden.

5 Evaluation

Dieses Paper basiert hauptsächlich auf den Arbeiten „Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software“ [3] und „A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes“ [6].
Das erstgenannte Paper klassifiziert Sicherheitslücken zwischen Kompartimenten [CIV] und untersucht deren Auftreten. Durch den Einsatz eines Fuzzers, ein automatisiertes Programm zur Indentifikation von Schwachstellen, wird getestet, ob angreifbare Funktionen existieren. Aufgrund der Art und Weise der Untersuchung wird aber lediglich die Schnittstelle zwischen Bibliotheken und Anwendung untersucht. Diese Einschränkung kann jedoch potenzielle Schwachstellen in der Interaktion zwischen anderen Systemkomponenten übersehen.

Im Gegensatz zu dieser praktischen Identifikation von Schwachstellen untersucht das andere Paper die Auswirkung von 10 Angriffen auf die bekanntesten TEEs und beschreibt deren Auftreten. Ein Aspekt dieser Untersuchung ist die theoretische Analyse der Angriffe, die durch konkrete Codebeispiele unterstützt wird.
Interessanterweise zeigen die in der ersten Arbeit identifizierten kompartimentären Kategorien von Sicherheitslücken eine hohe Übertragbarkeit auf die Angriffsvektoren in TEEs. Dies unterstreicht nicht nur die Relevanz der Kategorien, sondern auch die Möglichkeit, TEEs in diesem Kontext als eine Form von Kompartimenten zu betrachten. Diese Erkenntnis unterstützt die Gültigkeit der in der ersten Arbeit entwickelten Klassifikationen und bietet eine solide Grundlage für die weitere Untersuchung der Sicherheitsprobleme in feingranularen TEEs.

6 Fazit

Die Verbesserung der Sicherheit und Robustheit von Trusted Execution Environments (TEEs) ist essenziell, um Angriffe zu verhindern, die Rückschlüsse auf das Speicherlayout ermöglichen oder direkten Zugriff auf spezifische Daten gewähren. Diese Angriffe gefährden die Vertraulichkeit und Integrität der gespeicherten Informationen erheblich. Darüber hinaus untergraben Angriffe, die die Funktionalität der TEEs beeinflussen, das Vertrauen in diese Technologie. Das primäre Ziel von TEEs besteht darin, die Angriffsfläche zu minimieren und somit die Sicherheit zu erhöhen.
Kleinere und leichter zu verwaltende TEEs bieten verbesserte Sicherheit und Isolation, da sie weniger Funktionen übernehmen und somit weniger Daten sicher halten müssen. Diese Reduktion in der Komplexität erleichtert ihre Entwicklung und Verwaltung und verringert potenzielle Fehlerquellen. Zukünftige Entwicklungen sollten darauf abzielen, die Angriffsfläche weiter zu minimieren und eine noch bessere Isolation zu erreichen. Die Implementierung unterschiedlicher Sicherheitsstandards ist ein weiterer wichtiger Aspekt, insbesondere für feingranulare TEEs.
Um diese Ziele zu erreichen, ist noch viel Forschungsarbeit notwendig. Angriffe, die zu Datenlecks oder Datenkorruption führen können, müssen identifiziert und verhindert werden. Seitenkanalangriffe können nicht vollständig unterbunden werden, daher sind Techniken erforderlich, die diese Kanäle so sicher wie möglich implementieren. Trotz aller Anstrengungen wird es immer Angriffsmöglichkeiten geben. Selbst die sicherste TEE kann durch einen kleinen Fehler, ein kleines Detail, angreifbar werden. Diese Unsicherheit verdeutlicht die Notwendigkeit kontinuierlicher Forschung und Entwicklung, um die Sicherheit von TEEs stetig zu verbessern und anzupassen.

Literatur

[1] Victor Costan and Srinivas Devadas. Intel SGX explained. Cryptology ePrint Archive, Paper 2016/086, 2016. https://eprint.iacr.org/2016/086.
[2] Fabian Fleischer, Marcel Busch, and Phillip Kuhrt. Memory corruption attacks within android tees: a case study based on op-tee. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ARES '20, pages 1–9, 2020.
[3] Hugo Lefeuvre, Vlad-Andrei Bădoiu, Yi Chen, Felipe Huici, Nathan Dautenhahn, and Pierre Olivier. Assessing the impact of interface vulnerabilities in compartmentalized software. In *Proceedings 2023 Network and Distributed System Security Symposium*, NDSS '23, 2023.
[4] Jonas Repschläger, Danny Pannicke, and Rüdiger Zarnekow. Cloud computing: Definitionen, geschäftsmodelle und entwicklungspotenziale. *HMD Praxis der Wirtschaftsinformatik*, 47(5):6–15, 2010.
[5] David L Sackett, William M C Rosenberg, J A Muir Gray, R Brian Haynes, and W Scott Richardson. Evidence based medicine: what it is and what it isn't. *BMJ*, 312(7023):71–72, 1996.
[6] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D. Garcia, and Frank Piessens. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1741–1758, 2019.