Schutz von Datenbanken vor fehlerhaften Webanwendungen

Benjamin Große · Saša Prijović · Peter Trommler

Georg-Simon-Ohm Hochschule Nürnberg
benjamin.grosse@i-woas.net
sasa@prijovic.de
peter.trommler@ohm-hochschule.de

Zusammenfassung

Ein weit verbreiteter Typ von Sicherheitslücken in Webanwendungen sind *Insecure Direct Object References* (OWASP Top10: A-4). Um sich dagegen abzusichern, wird in diesem Beitrag eine Security-Appliance vorgestellt, die als externe Komponente eine bestehende Webanwendung absichert, ohne dass Änderungen an deren Quelltext vorgenommen werden müssen. Eine solche Security-Appliance ist eine Web Application Firewall, die neben einem Reverse-HTTP-Proxy zwischen Browser und Webanwendung einen SQL-Proxy zwischen Webanwendung und Datenbank betreibt. Mit diesen zusätzlichen Proxies kann Einfluss auf die Daten die der Webanwendung zur Verfügung stehen genommen werden. Es wird gezeigt, wie damit eine feingranulare Zugriffskontrolle im Rahmen einer *Defence in Depth*-Strategie umgesetzt werden kann. Hierzu wird gezeigt, wie sich Benutzer anhand ihres HTTP-Verkehrs erkennen lassen, ihnen Datenbankabfragen zugeordnet werden können und wie sich mit diesen Informationen eine Webanwendung nachträglich absichern lässt.

1 Einleitung

Bankkonten, Kreditkarten, Steuerdaten und kritische Unternehmensdaten sind begehrte Ziele für Kriminelle. Viele Webanwendungen benötigen diese Informationen, um Services aus den Bereichen eGovernment, eBusiness und eCommerce bereit zu stellen. Solche Systeme sind ständigen Angriffen aus dem Internet ausgesetzt und müssen durch geeignete Sicherheitsmaßnahmen geschützt werden.

Das Open Web Application Security Project (OWASP) veröffentlicht seit 2003 in unregelmäßigen Abständen die Top Ten der Sicherheitslücken in Webanwendungen. In diese Liste wurde die *Insecure Direct Objekt Reference* im Jahr 2007 aufgenommen. Sie steht über die Jahre unverändert auf Platz 4. Nach Eckert [Ecke11] ist eine direkte Objektreferenz ein Verweis in Form einer URL oder Parametern auf ein intern implementiertes Objekt wie zum Beispiel eine Datei, ein Verzeichnis oder auch ein Datenbankeintrag. Angreifer manipulieren solche Referenzen, um so Zugriff auf andere interne Objekte zu erhalten, ohne dafür autorisiert zu sein.

In einer Arbeit aus dem Jahr 2007 schlugen Roichman und Gudes [RoGu07] daher vor die Webanwendung als nicht voll vertrauenswürdig zu behandeln und stattdessen zusätzlich im Sinne einer *Defence in Depth*-Strategie in der Datenbank eine weitere Zugriffskontrolle durchzuführen. Diese Strategie basiert auf der Idee, die angeforderte Datenmenge für jeden Benutzer individuell einzugrenzen, indem eine Benutzerkennung oder ein Sicherheitstoken aus dem sich die Benutzerkennung ableiten lässt bei Datenbankabfragen mit an die Datenbank übergeben

P. Schartner · P. Trommler (Hrsg.) · D•A•CH Security 2013 · syssec (2013) 1-11.

werden. Alle Abfragen wirken folglich nur auf die Datenmenge für die der jeweilige Benutzer berechtigt ist. Dadurch können Fehler in der Webanwendung nicht für einem unberechtigten Zugriff ausgenutzt werden. Im Folgenden bezeichnen wir eine Datenbankabfrage, die den Zugriff auf die Datensätze der Datenbank entsprechend der Benutzerkennung beschränkt, als *autorisierte Datenbankabfrage*.

Roichman implementiert diese Idee mit Hilfe von *Parameterisierten Views*, die nicht Bestandteil von Standard-SQL sind und auch nur von wenigen Datenbanken unterstützt werden. Roichman gibt eine Umsetzung mit *user defined functions* für Datenbanken ohne Parametrisierte Views an. In einem Beitrag zur D-A-CH Security 2012 [RGPT12] haben wir gezeigt, wie autorisierte Datenbankabfragen mittels *query rewriting*, dem Umschreiben der Datenbankabfragen in Standard-SQL, realisiert werden können. Wird dieses Umschreiben in einem Proxy vor der Datenbank vorgenommen, ist es möglich, autorisierte Datenbankabfragen mit jeder SQL-Datenbank umzusetzen.

Bei Neuentwicklungen ist es möglich, die Webanwendung von Beginn an so zu entwickeln, dass autorisierte Datenbankabfragen erzeugt und damit die Identität des Benutzers bei Anfragen an die Datenbank mitgeliefert werden. Ein Anpassen bestehender Anwendungen ist jedoch mit hohem Aufwand verbunden oder aufgrund von Lizenzrahmenbedingungen unmöglich. Daher untersuchen wir in diesem Beitrag, wie Datenbanken gegen bestehende Webanwendungen geschützt werden können, ohne dabei in den Quelltext der Webanwendung eingreifen zu müssen.

Diesen Lösungsansatz setzen wir in Form einer Netzwerk-Appliance (Security-Appliance) um, die mit Hilfe zweier Proxies eine Webanwendung umschließt, den Datenverkehr analysiert und daraus autorisierte Datenbankabfragen einschließlich der Benutzerkennungen erzeugt. Diese Security-Appliance, als eigenständiges Netzwerkgerät, lässt sich in vorhandene Infrastrukturen integrieren und ist aus Sicht der Webanwendung transparent.

Zur Realisierung einer solchen Security-Appliance zur Absicherung von Datenbanken gegen Insecure Direct Object References ist folgende Fragestellung zu untersuchen: Wie lässt sich die Benutzerkennung des Initiators den SQL Statements zuordnen, die von der Webanwendung an die Datenbank gesendet werden?

Zwischen Browser und Webanwendung können Benutzer durch ihre Benutzerkennung unterschieden werden, d.h. jeder *HTTP-Request* (Webanfrage) kann einer Benutzerkennung zugeordnet werden. Zwar enthalten Datenbankabfragen diese Benutzerkennung nicht mehr, jedoch sind andere Parameter des HTTP-Requests in den Datenbankabfragen wieder zu finden. Daher wird der Lösungsansatz verfolgt, HTTP-Requests samt ihrer Benutzerzuordnung mit den Datenbankabfragen zu verknüpfen.

Diese Arbeit gliedert sich in sechs Abschnitte. Die Einleitung wird in diesem Absatz abgeschlossen. Der zweite Abschnitt beschäftigt sich mit der Security-Appliance (2). Der generelle Aufbau wird in 2.1 thematisiert. In 2.2 werden wichtige Grundbegriffe erklärt. Die Verarbeitungsschritte, die in der Security-Appliance vorgenommen werden, finden sich in den Abschnitten 2.3, 2.4 sowie 2.5. Abschnitt 3 behandelt das Problem der Unterscheidbarkeit von Datenbankzugriffen. Es werden die Szenarien sowie Lösungsmöglichkeiten behandelt. Wie die Mapping-Engine konfiguriert werden muss, wird in Abschnitt 4 erläutert. Die letzten beiden Abschnitte 5 und 6 widmen sich den verwandten Arbeiten sowie einer Zusammenfassung des erreichten und weiteren Forschungspotentialen für den hier vorgestellten Ansatz.

2 Security-Appliance

Zur Umsetzung autorisierter Datenbankabfragen sowie zur nachträglichen Absicherung von Webanwendungen, muss die entsprechende Infrastruktur geschaffen werden.Mit dieser soll es ermöglicht werden in die Datenbankabfragen der Webanwendung einzugreifen und diese zu beeinflussen. Folgende Architektur soll diese Anforderungen erfüllen. Sie entspricht einer Web Appliction Firewall mit zusätzlichem Proxy.

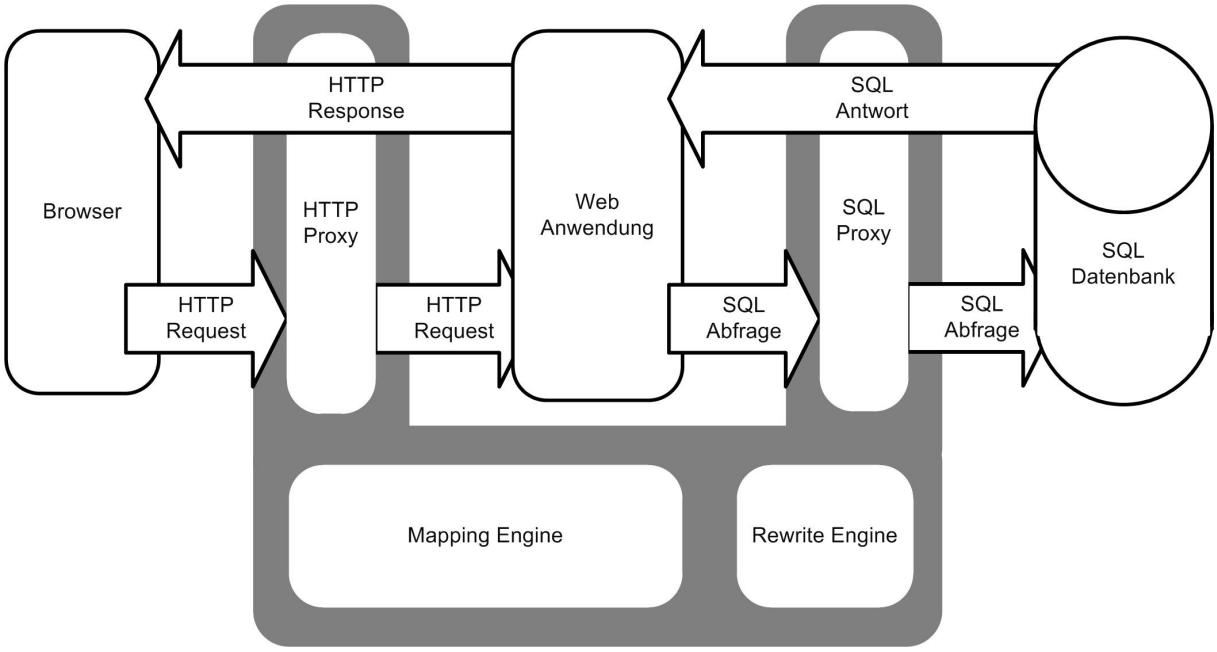

Abb. 1: Security-Appliance

Dieses Kapitel zeigt die Architektur der Security-Appliance(siehe Abschnitt 2.1) sowie den Prozess bei Erstellung einer autorisierten Datenbankabfrage (siehe Abschnitte 2.3, 2.4 und 2.5).

2.1 Aufbau Security-Appliance

Die Information über die Identität des Benutzers in der Webanwendung liegt bei Entscheidungen der Geschäftslogik vor und damit zwangsläufig auch in der Kommunikation zwischen dem Benutzer und der Webanwendung. Hier kann ähnlich einer Web Application Firewall durch einen Reverse-HTTP-Proxy eingegriffen und die Identität des Benutzers extrahiert werden (siehe 2.3).

Ein SQL-Proxy zwischen Webanwendung und Datenbank analysiert den Datenbankverkehr und liefert die einzelnen Datenbankabfragen an die Mapping Engine. Dieser SQL-Proxy gibt außerdem die autorisierten Datenbankabfragen an die Datenbank weiter sowie deren Ergebnissen an die Webanwendung zurück (siehe Abschnitt 2.4).

Die Mapping-Engine wertet die durch die Proxies gesammelten Informationen aus und verknüpft die Datenbankabfragen mit Benutzerkennungen und gibt beides an die Rewrite-Engine weiter (siehe Abschnitt 2.5).

Aus SQL-Abfragen und Benutzerkennung erstellt die Rewrite-Engine autorisierte Datenbankabfragen und sichert den Datenzugriff dadurch ab [RGPT12].

2.2 Terminologie

Zur Abstraktion von Teilfunktionen der Webanwendung werden folgende Begriffe eingeführt und wie folgt definiert: *Aktionen* sind einzelne Zugriffe auf die Webanwendung, wie beispielsweise das Absenden eines Formulars. Jede Aktion entspricht einem HTTP-Request, der wiederum eine oder mehrere Datenbankabfragen auslösen kann. Aktionen werden in unserem Ansatz mit Hilfe von *Aktionssignaturen* klassifiziert, die eine Funktionalität in der Anwendung, wie z. B. das Annehmen von Anmeldedaten, zur Verfügung stellt. Jede Aktion lässt sich eindeutig einer vordefinierten Aktionssignatur durch die Mapping-Engine zuordnen. Diese Aktionssignaturen werden im Rahmen der Konfiguration erstellt (siehe Abschnitt 4).

Analog dazu wird der Datenverkehr von der Webanwendung zur Datenbank in *Datenbankabfragen* zerlegt. Diese Datenbankabfragen sind einzelne SQL-Statements und damit die kleinst mögliche Einheit von ausführbarem SQL-Code. Zur Wiedererkennung werden diese Datenbankabfragen mit Hilfe von *Abfragesignaturen* klassifiziert. Abfragesignaturen enthalten neben einem regulären Ausdruck der zur allgemeinen Beschreibung dieser Klasse genutzt wird noch Platzhalter für Parameter. Durch diese Platzhalter lassen sich *parametrierte Aktionssignaturen* erstellen und ermöglichen eine genauere Unterscheidung von Datenbankabfragen, da bei einer Prüfung auch die Parameter übereinstimmen müssen.

2.3 HTTP-Verarbeitung

Der Reverse-HTTP-Proxy extrahiert die für die Verarbeitung benötigten Daten aus einem HTTP-Request und übergibt sie der Mapping-Engine (siehe Abbildung 2). Mit Hilfe dieser Daten werden zwei Aufgaben erfüllt:

1. Zuordnung einer Benutzerkennung zu jeder HTTP-Request
2. Zuordnung der Aktion mit ihrer Aktionssignatur und dem Parametersatz

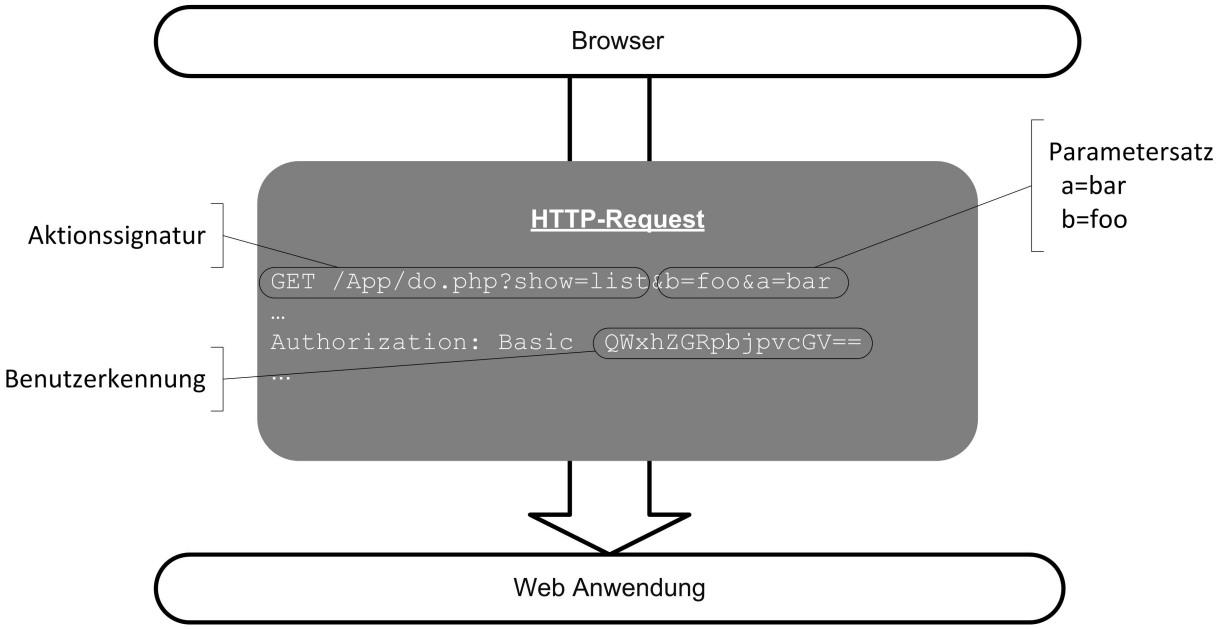

Abb. 2: HTTP-Request

Jeder Aktion wird ein Benutzer zugeordnet, indem aus der HTTP-Request die Benutzerkennung

extrahiert wird. Wie dies geschieht, hängt vom verwendeten Authentisierungsmechanismus ab. Authentisierungsmechanismen lassen sich wie folgt klassifizieren:

- Authentisierungsinformation bei jeder HTTP-Request
- Einmalige Authentisierung und Erstellung eines Authentisierungstoken
- Authentisierung auf der Transportschicht (SSL/TLS)

Bei der Authentisierung mit den Standardmechanismen von HTTP [FHBHL+99] wie *Basic Authenticaton* und *Digest Authentication* wird die Authentisierungsinformation bei jedem HTTP-Request mitgesendet. Bei diesen Verfahren kann die Benutzerkennung direkt aus dem HTTP-Request extrahiert werden.

Bei der Authentisierung über ein Authentisierungstoken, z.B einer Session-Id, geschieht dies in zwei Stufen. In der ersten Stufe weist sich der Benutzer, beispielsweise durch Eingabe von Benutzerkennung und Passwort, gegenüber der Webanwendung aus. Nach erfolgreicher Authentisierung wird ein Authentisierungstoken erzeugt, das an den Browser zurückgesendet wird. In der zweiten Stufe wird mit jedem HTTP-Request auch dieses Authentisierungstoken mitgesendet. Die Anmeldung wird protokolliert und daraus die Benutzerkennung sowie das Authentisierungstoken extrahiert. Diese Zuordnung von Authentisierungstoken zur Benutzerkennung wird in Form einer Tabelle durch die Mapping-Engine verwaltet. Ihre Zuordnungspaare werden gelöscht sobald sich der Benutzer abmeldet oder ihre Gültigkeitsdauer überschritten ist.

Authentisierungsmethoden mittels externer Partner wie z. B. OpenID, stellen ähnlich einer formularbasierten Anmeldung ein Authentisierungstoken bereit und benötigen zwar in der ersten Stufe eine spezielle Behandlung, lassen sich allerdings in der zweiten Stufe, genau so wie vorher auf die Zuordnungstabelle abbilden.

Bei den oben genannten Arten der Authentisierung muss ein Identifikationsnachweis bei jeder Aktion mitgesendet werden, entweder direkt die Benutzerkennung oder das Sessiontoken. Wenn *TLS Client Authentication (SSL)* eingesetzt wird ist das nicht der Fall. Daher muss die TLS-Verbindung am Proxy terminiert werden. Um Authentisierungsinformation zu erstellen und an die Webanwendung und die Mapping-Engine weiter zu geben.

Gegebenenfalls müssen mehrere Authentisierungsmethoden parallel betrachtet werden, beispielsweise für Webanwendungen die *Basic Authentication* als Fallback für eine *Cookie-Session Authentication* bereitstellen. In diesem Fall gilt ein Benutzer als identifiziert wenn er sich über eines der Verfahren angemeldet hat.

HTTP-Requests und damit Aktionen werden mit Hilfe von vorkonfigurierten Aktionssignaturen klassifiziert (siehe Abschnitt 4). Aktionen werden dazu mit Aktionssignaturen verglichen und bei Übereinstimmung aller Merkmale dieser Klasse zugeordnet (siehe Abschnitt 2.5). Sämtliche Aktionen die keinen Datenbankzugriff initiieren, müssen nicht weiter betrachtet werden. Die Aktionssignaturen für die restlichen Aktionen müssen so gestaltet sein, dass keine Mehrdeutigkeiten entstehen, um eine eindeutige Zuordnung zu gewährleisten. Hierzu bestehen die Aktionssignaturen sowohl aus HTTP-Verb und URL als auch einem frei konfigurierbaren Satz an Parametern des HTTP-Headers, die zur Prüfung verwendet werden.

Damit sich Aktionen eindeutig identifizierten lassen wird eine eindeutige Aktions-Id generiert. Dadurch kann ein HTTP-Response seinem Aufrufendem HTTP-Request zugeordnet werden, um das Ende einer Aktion zu erkennen. Hierzu werden folgende Eigenschaften von HTTP genutzt. HTTP baut auf TCP auf und besteht aus Request-Response Paaren. Jedem HTTP-Request

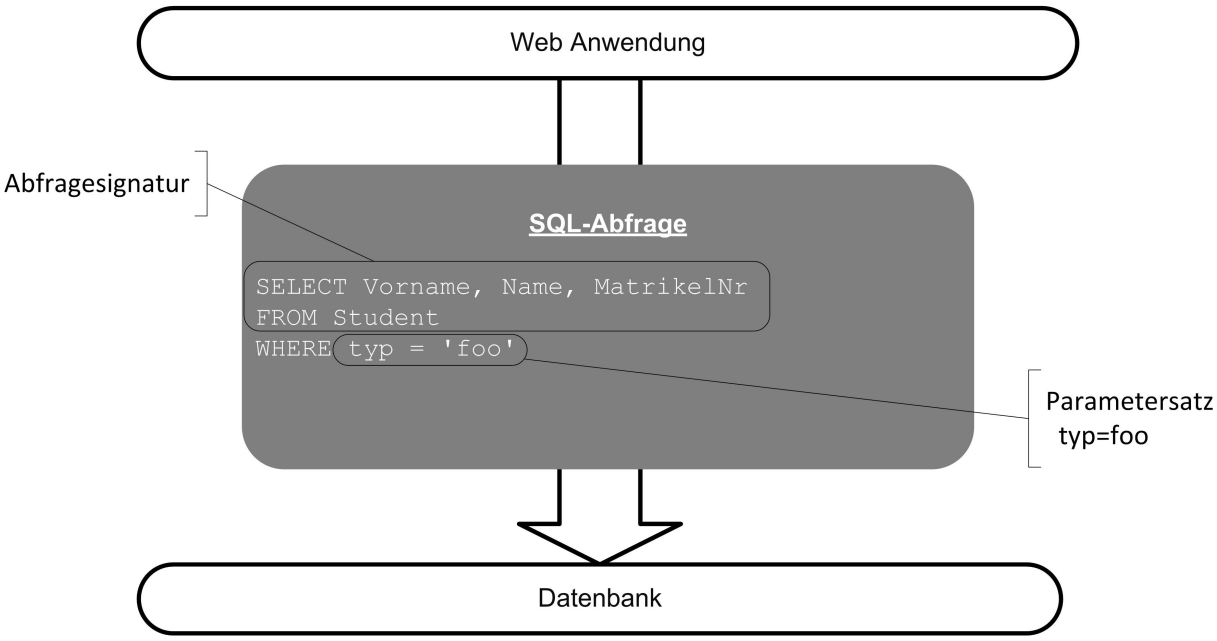

Abb. 3: Datenbankabfrage

wird innerhalb der gleichen TCP Verbindung beantwortet. Bei der asynchronen Verwendung entspricht die Reihenfolge der Responses die der Requests [FGMF+99]. Deshalb lassen sich Aktionen eindeutig durch Quell-Port, Quell-IP-Adresse und einer fortlaufenden Nummer identifizieren.

2.4 SQL-Verarbeitung

Der SQL-Proxy zwischen Webanwendung und Datenbank extrahiert aus einem spezifischen Protokoll die einzelne Datenbankanfragen und liefert diese an die Mapping Engine. Die Mapping-Engine ordnet dem SQL-Befehl einen Benutzer zu (siehe Abschnitt 2.5) und verwendet die Rewrite-Engine um die Datenbankabfragen zu autorisieren. Diese autorisierte Datenbankabfrage wird anschließend durch den SQL-Proxy an die Datenbank weitergeleitet.

Analog zum HTTP-Request werden aus der Struktur der Datenbankabfrage Informationen extrahiert, die sich, wie in Abbildung 3 zu sehen ist, wie folgt zusammensetzen:

- Abfragesignatur
- Parametersatz

Mit diesen Informationen wird in der Mapping-Engine die zur Datenbankabfrage gehörende Benutzerkennung bestimmt. Dazu wird sie mit den aktuell freigebenden parametrierten Datenbankabfragesignaturen verglichen und dem entsprechenden Benutzer zugeordnet. Damit wird die Datenbankabfrage um die Benutzerkennung ergänzt. Die Aufgabe der Mapping-Engine besteht darin, die Webanfragen den noch anonymen Datenbankabfragen zuzuordnen.

2.5 Verarbeitung durch die Mapping-Engine

Die Webanfragen werden vom HTTP-Proxy und die Datenbankabfragen vom SQL-Proxy abgefangen und der Mapping-Engine zur Verarbeitung übergeben.

Eine Aktion gliedert sich, wie auf Abbildung 2 zu sehen ist, in die Bestandteile Aktionssignatur, Parametersatz sowie Benutzerkennung. Diese Bestandteile werden direkt nach der Übergabe durch den HTTP-Proxy extrahiert. Anschließend wird diese Aktion mit Hilfe der Aktionssignaturen klassifiziert. Zur Klassifizierung einer Aktion wird diese mit allen konfigurierten Aktionssignaturen auf Übereinstimmung ihrer URL verglichen. Die daraus entstehende Untermenge von Aktionssignaturen wird durch einen Vergleich der HTTP-Verben weiter eingeschränkt. Falls die Zuordnung dadurch noch nicht eindeutig ist, werden weiter Bestandteile des HTTP-Headers verglichen. Dazu werden reguläre Ausdrücke verwendet, die während der Konfiguration der Security-Appliance erstellt wurden.

Anschließend wird der in der Aktion enthaltene Parametersatz ausgelesen. Entsprechend der Aktionssignatur wird die Gruppe der zugehörigen Abfragesignaturen (siehe dazu Abbildung 3) um den ausgelesenen Parametersatz ergänzt, um zukünftige Datenbankabfragen eindeutig dieser Aktion zuordnen zu können.

Aus den gesammelten Informationen entsteht somit für jede Aktion ein Satz aus ein oder mehreren freigegebenen parametrierten Abfragesignaturen. Zu jeder dieser Abfragesignaturen wird außerdem der Aktionsidentifikator und die Benutzerkennung gespeichert. Die Abfragesignaturen gelten somit ab diesem Zeitpunkt als freigegeben. Es können damit die dazu passenden Datenbankabfragen ausgeführt werden bis die Aktion beendet ist, d. h. der HTTP-Response versendet wurde.

Von der anderen Seite, dem SQL-Proxy, wird die ursprüngliche Datenbankabfrage übergeben und wird mit den freigegeben, parametrierten Abfragesignaturen verglichen. Passt eine Datenbankabfrage dabei auf eine der Abfragesignaturen, wird sie der zugehörigen Aktion und damit dem aufrufenden Benutzer zugeordnet. Falls die Datenbankabfrage zu keiner freigeschalteten Abfragesignatur passt, wird dieser Zugriff als anonym betrachtet. Dazu wird die Abfrage einem anonymen Benutzer zugeordnet und der Zugriff so auf öffentliche Daten beschränkt.

Anschließend wird die Datenbankabfrage samt Benutzerkennung an die Rewrite-Engine übergeben. Diese schreibt die Datenbankabfrage entsprechend der feingranularen Zugriffskontrolle in eine autorisierte Datenbankabfrage um. Dadurch werden nur Daten zurückgeliefert, die für den jeweiligen Benutzer freigegeben sind. Es werden also nur Daten an die Webanwendung zurückgegeben, die im Kontext des aufrufenden Benutzers für die Bearbeitung der aktuellen Aktion benötigt werden.

3 Unterscheidbarkeit von Datenbankzugriffen

Die vorgestellte Security-Appliance ordnet Datenbankzugriffen ihren Initiatoren zu. Eine Aktion kann dabei mehre Datenbankabfragesignaturen freigeben, die somit von der Datenbank verwendet werden können. Bei dieser Zuordnung sind zwei Aspekte zu betrachten. Zum einen wie zielsicher lassen sich Datenbankanfragen bei gleichzeitigem Zugriff zuordnen und zum anderen wie wirksam ist diese Lösung gegen Insecure Direct Object References?

Bei gleichzeitigem Zugriff auf eine Aktionssignatur ohne Parameter entsteht eine Race Condition, so dass eine eindeutige Zuordnung der Benutzer zu ihren Datenbankabfragen nicht mehr möglich ist. Greift beispielsweise Benutzer A auf eine Aktion zu, bevor der HTTP-Response eines Aufrufs der gleichen Aktion durch Benutzer B erfolgt, kann keine Unterscheidung vorgenommen werden, ob die dazugehörigen Datenbankabfrage von A oder B initiiert wurde. In diesem Fall werden die Datenbankabfragen so autorisiert, dass sie maximal Daten für A und

B enthalten ($Daten_{AB} = Daten_A \cup Daten_B$), jedoch weiterhin der Zugriff auf Daten anderer Benutzer nicht möglich ist.

Bei gleichzeitigen Aktionen, die einen Parametersatz enthalten, können Datenbankabfragen den jeweiligen Benutzern eindeutig zugeordnet werden. Dies geschieht unter der Bedingung, dass die Werte innerhalb des Parametersatzes unterschiedlich sind. Die Parameterwerte der Datenbankabfrage können mittels den übergebenen Parameterwerten der Aktion verglichen und, bei Übereinstimmung, dem Benutzer zugeordnet werden.

Eine Aktion mit gleichen Parametern von unterschiedlichen Benutzern kann jedoch weiterhin nicht unterschieden werden. Bei Aktionen, die idempotente Schreibzugriffe initiieren stellt dieses Verhalten aus Sicht der Datenkonsistenz kein Problem dar. Anders jedoch bei Aktionen, die Lesezugriffe und Schreibzugriffe initiieren und nicht idempotent sind. Hier kann der Angreifer in der Zeit, in der ein legitimer Benutzer auf Daten zugreift dies ebenso tun. Das entstehende Zeitfenster für diesen Zugriff ist allerdings auf die Zeit, die zur Bearbeitung der legitimen Aktion benötigt wird, beschränkt. Durch das Serialisieren solcher ununterscheidbarer Aktionen kann ein derartiger Angriff ausgeschlossen werden. Dadurch wird zu jedem Zeitpunkt nur genau eine der ununterscheidbare Aktion von der Webanwendung bearbeitet und die daraus resultierenden Datenbankabfragen können somit eindeutig dem aufrufenden Benutzer zuzuordnen. Die Serialisierung erreicht man durch Verzögern der Weitergabe des HTTP-Requests an die Webanwendung, bis die laufende gleichartige Aktion beendet ist. Dieses Vorgehen garantiert in jedem Fall eine genaue Zuordnung, erhöht allerdings die Latenz einzelner Abfragen und sollte deshalb nur bei ausgewählten, kritischen Aktionen eingesetzt werden.

Um die Wirksamkeit der Security-Appliance zu betrachten verwenden wir in dieser Arbeit ein Modell, in dem die Webanwendung als unsicher betrachtet wird. Es wird davon ausgegangen, dass ein potentieller Angreifer sie nach Belieben beeinflussen kann. Bezogen auf die Problemstellung der Insecure Direct Object References bedeutet dies, dass Angreifer durch unsichere Objektreferenzen auf sämtliche Datenbankeinträge zugreifen kann. Ein nicht angemeldeter Benutzer kann jedoch keine Abfragesignaturen in der Mapping-Engine erzeugen. Daher werden alle seine Datenbankabfragen als anonym betrachtet und entsprechend verarbeitet, folglich werden nur öffentlich zugängliche Daten zurückgeliefert.

Gelingt es einem Angreifer als angemeldeter Benutzer gültige Aktionen mit dazu passenden SQL-Abfragen zu erzeugen, entsprechen diese dem regulären Geschäftsprozess. Der Schaden beschränkt sich auf die Datenmenge des angemeldeten Benutzers, da der Zugriff auch nur auf Daten beschränkt ist, die mit dem Benutzer verknüpft sind. Auf Daten fremder Benutzer kann dadurch nicht zugegriffen werden. Auch falls durch fehlerhafte Konfiguration eine ungewollte Datenmanipulation möglich ist, bleibt der Schaden jedoch begrenzt.

4 Konfiguration der Mapping-Engine

Mit der Konfiguration wird die Mapping-Engine an die jeweilige Webanwendung angepasst. Dies geschieht für jede Webanwendung individuell und beschreibt deren Verhalten. Die Konfiguration wird erstmalig bei der Installation der Security-Appliance erstellt und muss lediglich bei Änderungen der Webanwendung angepasst werden.

Die Konfiguration besteht aus Aktionssignaturen, die zur Klassifizierung der Aktionen genutzt werden, sowie einem den Aktionssignaturen zugeordneten Satz von Abfragesignaturen (siehe Abbildung 4). Die Aktionssignaturen sind als Blaupausen für HTTP-Request-Header zu verste-

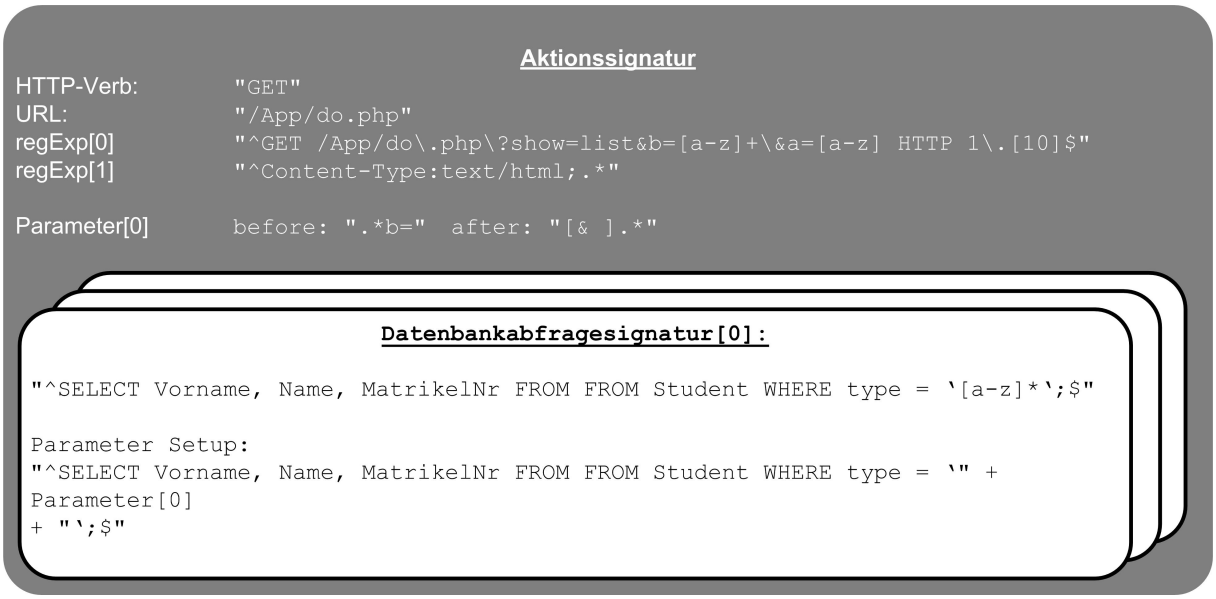

Abb. 4: Aktionssignatur

hen. Sie beinhalten neben einem HTTP-Verb und einer URL noch einen Satz frei gestaltbarer regulärer Ausdrücke. Diese regulären Ausdrücke werden auf die HTTP-Header-Felder angewendet und müssen für eine erfolgreiche Zuordnung der Aktion übereinstimmen. Weiterhin enthält die Aktionssignatur eine Beschreibung, wie die Aktionsparameter aus dem HTTP-Request extrahiert werden können, ebenfalls in Form von regulären Ausdrücken. Jede Aktionssignatur enthält eine ihr zugeordnete Gruppe von Abfragesignaturen. Diese Gruppe entspricht den Datenbankabfragen, die im Rahmen einer auf die Aktionssignatur passenden Aktion ausgeführt werden dürfen. Außerdem sind diese Abfragesignaturen mit Platzhaltern für das Parametrieren versehen.

Zum Konfigurieren der Mapping-Engine wird eine Methode verwendet, die wir als *unterstütztes Lernen durch Beobachten* bezeichnen. Hierzu wird die Webanwendung in einem Testaufbau wie folgt analysiert. Die Security-Appliance protokolliert sowohl den HTTP-Verkehr als auch den SQL-Verkehr. Um Mehrdeutigkeiten zu vermeiden, die bei gleichzeitigem Eintreffen von Requests auftreten (siehe Abschnitt 3), läuft die gesamte Security-Appliance in einem Single Request Modus. In diesem Single Request Modus wird durch den Proxy sichergestellt, dass jeweils immer nur ein HTTP-Request verarbeitet wird. Dadurch wird die Komplexität reduziert, indem die Funktionalität der Webanwendung in einzelne Aktionen zerlegt wird. Es wird zudem sichergestellt, dass alle entstehende Datenbankabfragen zur entsprechenden Aktion gehören. In diesem Modus werden Use Cases mithilfe von teil- oder vollautomatisierten Tests durchgespielt, wobei sichergestellt werden muss, dass die Funktionalität der gesamten Anwendung getestet wird. Durch diese Tests wird eine Sammlung von Aktionen mit zugeordneten Datenbankabfragen generiert, in der sämtliche Aktionen enthalten sind, die eine Datenbankabfrage ausgelöst haben. Anschließend wird diese Sammlung weiterverarbeitet, indem die Aktionen zu Aktionssignaturen und die Datenbankabfragen zu Abfragesignaturen verallgemeinert werden. Die gesamte Konfiguration erfolgt durch eine Person, die mit der Security-Appliance, Webanwendung, dem Datenmodell der Webanwendung, HTTP und SQL vertraut ist. Diese Person kann z. B. ein Administrator der Webanwendung oder ein Datenbankentwickler sein, der sich

in die Webanwendung eingearbeitet hat.

5 Verwandte Arbeiten

Möglichkeiten zur Absicherung von Webanwendungen wurden in diversen Publikationen diskutiert. Darunter die Forschung von Roichman und Gudes in [RoGu07]. Diese stufen Webanwendungen als generell nicht vertrauenswürdig ein und schlagen ein Verfahren vor wie mit parametrisierten Views Daten vor Fremdeinwirkungen geschützt werden können. Einen ähnlichen Ansatz verfolgten Rizvi et al. [RMSR04]. Diese nehmen alle SQL-Abfragen entgegen und schreiben diesen entsprechend der geltenden Zugriffsregeln für jeden Benutzer um.

Die Ideen von Roichman und Gudes flossen in [PrTr11] und [RGPT12] ein. Dabei wurde eine domänenspezifische Sprache entworfen, die es ermöglicht Zugriffsregeln in fast natürlicher Sprache auszudrücken, um Query Rewriting Regeln bzw. parametrisierte Views zu erstellen.

6 Zusammenfassung und Ausblick

Diese Arbeit beschreibt eine Möglichkeit Datenbanken gegen Insecure Direct Object References in Webanwendungen durch den Einsatz einer Security-Appliance nachträglich zu schützen, ohne dabei am Quelltext der Webanwendung Änderungen vornehmen zu müssen. Die Absicherung baut auf der Methode auf zwei Proxies einzusetzen, die jeweils zwischen Browser und Webanwendung sowie Webanwendung und Datenbank positioniert sind. Kernstück der Security-Appliance ist die Mapping-Engine, die für die Zuordnung der HTTP-Requests zu Datenbankabfragen zuständig ist.

Dabei stellte die Arbeit eine Vorgehensweise vor, an Hand derer eine solche Zuordnung möglich ist. Durch die vorgestellte Security-Appliance konnte eine aus einem HTTP-Request resultierende Datenbankabfrage mit einem Benutzer verknüpft werden ohne dabei auf die Webanwendung selbst angewiesen zu sein. Es wurde zudem eine zentrale Herausforderung thematisiert wie mehrere gleichzeitige Datenbankabfragen unterschieden werden können.

Diese Datenbankabfragen werden in Folge der regulären Verarbeitung in autorisierte Datenbankabfragen umgewandelt, indem sie abhängig von der Benutzerkennung so eingeschränkt werden, dass nur noch für den Benutzer zugelassene Operationen in der Datenbank ausgeführt werden.

In zukünftigen Arbeiten wird untersucht werden, wie die Erzeugung der Regeln für die Mapping-Engine automatisiert werden kann. Einen Ansatz zur Analyse der Datenströme sowie das selbstständige Lernen von Mustern und der daraus resultierenden Zuordnung bieten *Künstliche Neuronale Netze* [KoSH01]. Außerdem ist zu prüfen, wie sich andere Typen von Sicherheitslücken mit Hilfe dieser Security-Appliance schließen lassen.

Literatur

[Ecke11] C. Eckert: IT-Sicherheit: Konzepte - Verfahren - Protokolle. Oldenbourg Wissenschaftsverlag (2011), .

[FGMF+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard) (1999), , updated by RFCs 2817, 5785, 6266, 6585.

[FHBHL+99] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart: HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard) (1999), .

[KoSH01] T. Kohonen, M. R. Schroeder, T. S. Huang (Hrsg.): Self-Organizing Maps. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd Aufl. (2001).

[PrTr11] S. Prijovic, P. Trommler: Zugriffskontrolle in der Datenbank: Vamos eine Fallstudie. *In: P. Schartner, J. Taeger (Hrsg.), D-A-CH Security 2011* (2011), 492 – 503.

[RGPT12] M. Rossel, B. Große, S. Prijovic, P. Trommler: Zugriffskontrolle in Webdatenbanken mit Query Rewriting. *In: P. Schartner, J. Taeger (Hrsg.), D-A-CH Security 2012* (2012), 219 – 230.

[RMSR04] S. Rizvi, A. Mendelzon, S. Sudarshan, P. Roy: Extending query rewriting techniques for fine-grained access control. *In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, ACM, New York, NY, USA (2004), 551–562.

[RoGu07] A. Roichman, E. Gudes: Fine-grained access control to web databases. *In: Proceedings of the 12th ACM symposium on Access control models and technologies*, SACMAT '07, ACM, New York, NY, USA (2007), 31–40, .