18 Allgemeine Schwachstellen im Web

Bisher haben wir der Klassifikation gängiger Schwachstellen im Web wenig Aufmerksamkeit geschenkt. Es ist schließlich weitaus wichtiger, Einsicht in die Technik zu gewinnen, die Webanwendungen zugrunde liegt, als sich einige Tausend zufällige und häufig nicht notwendige Begriffe und Abkürzungen zu merken. Bezeichnungen wie *improper restriction of operations within the bounds of a memory buffer* (Common Weakness Enumeration) oder *insecure direct object references* (Open Web Application Security Project) haben in einem vernünftigen Gespräch keinen Platz – und das ist auch ganz gut so.

Trotzdem hat die Branche eine Handvoll einigermaßen präziser Ausdrücke hervorgebracht, die Sicherheitsforscher tagtäglich benutzen. Nachdem wir die innere Funktionsweise des Browsers gründlich erörtert haben, ist es an der Zeit, die Terminologie, auf die der Durchschnittsleser mit einiger Wahrscheinlichkeit stößt, kurz zusammenzufassen.

18.1 Spezifische Schwachstellen von Webanwendungen

Die in diesem Abschnitt skizzierten Begriffe sind spezifisch für die im Web verwendeten Technologien und haben oft kein direktes Gegenstück in der Welt »herkömmlicher« Anwendungssicherheit.

18.1.1 Cross-Site Request Forgery (CSRF, XSRF oder »Sea-Surf«)

Diese Schwachstelle wird dadurch verursacht, dass ein Webserver in der Regel nicht überprüfen kann, ob ein bestimmter, auf dem Server zu Statusänderungen führender HTTP-Request überhaupt vom erwarteten Ursprung auf Clientseite stammt. Dieser Fehler ermöglicht es beliebigen, im Browser geladenen Webseiten, Aktionen im Namen des Opfers durchzuführen. Ein im Browser geladener Tab kann beispielsweise versuchen, die Datei *logout.php* eines anderen Ursprungs als Bild zu laden. Ist kein CSRF-Schutz installiert, dürfte der Anwender tatsächlich ausgeloggt werden.

➠ *In Kapitel 4 finden Sie eine ausführlichere Erörterung von CSRF.*

18.1.2 Cross-Site Script Inclusion (XSSI)

Die Ursache dieses Mangels liegt im fehlenden Schutz sensibler JSON-ähnlicher Antworten gegen das Laden auf Sites Dritter mittels `<script src=...>`. Dadurch können benutzerspezifische Daten in der Antwort für Angreifer offengelegt werden. Vor einigen Jahren konnte man auf diesem Wege das Google-Adressbuch eingeloggter Anwender von beliebigen Seiten aus klauen. Verwandt mit dieser Problematik ist das sogenannte JSON-Hijacking – eine etwas raffiniertere *XSSI-Variante*.

➠ *In Kapitel 6 finden Sie einen Überblick über das Problem (und mögliche Lösungen).*

18.1.3 Cross-Site-Scripting (XSS)

Die ungenügende Prüfung der Eingabe oder die fehlende Maskierung der Ausgabe können einen Angreifer in die Lage versetzen, eigenes HTML-Markup oder eigene Skripte in eine anfällige Site einzuschleusen. Diese Skripte bekommen Zugriff auf die gesamte Zielanwendung und vielfach auf HTTP-Cookies, die auf dem Client gespeichert sind. Das Zusatzattribut *reflected* bezieht sich auf Fälle, in denen die eingeschleuste Zeichenfolge einfach das Ergebnis der fehlerhaften Rückgabe eines Teils des Requests ist. Die Attribute *stored* oder *persistent* stehen für Situationen, in denen die Daten einen komplizierteren Weg einschlagen. *DOM-based* bedeutet, dass die Schwachstelle durch das Verhalten des clientseitigen Teils der Webanwendung verursacht wurde (d.h. durch JavaScript).

➠ *In Kapitel 4 finden Sie häufige XSS-Vektoren in HTML-Dokumenten.*

➠ *In Kapitel 6 finden Sie einen Überblick über XSS-Risiken auf DOM-Basis.*

➠ *In Kapitel 13 finden Sie XSS-Vektoren im Zusammenhang mit Content-Sniffing.*

➠ *In Kapitel 9 finden Sie eine Erörterung des normalen Sicherheitsmodells für JavaScript-Code.*

18.1.4 Header Injection/Response Splitting

Mit Header Injection bzw. Response Splitting bezeichnet man die ungenügende Maskierung von Zeilenumbrüchen (oder gleichwertigen Zeichen) in HTTP-Antworten im serverseitigen Teil einer Webanwendung. Dieses Verhalten führt üblicherweise zu XSS sowie zu Browser- und Proxy-Cache-Poisoning und mehr.

➠ *In Kapitel 3 finden Sie eine ausführliche Erörterung dieses Fehlers.*

18.1.5 Mixed Content

Mixed Content (gemischter Inhalt) entsteht durch das Laden von HTTP- oder anderen unverschlüsselt übertragenen Ressourcen in HTTPS-Seiten. Was Skripte und Applets betrifft, macht dieses Verhalten die Anwendung auf einfache Weise anfällig für aktive Angriffe, die zu XSS und Schlimmerem führen können. Dies gilt insbesondere in offenen WLANs (in Cafés, auf Flughäfen usw.) und macht fast alle Vorteile von HTTPS zunichte. Die Folgen solcher Fehler im Zusammenhang mit Stylesheets, Schriftarten, Bildern und anderen passiven Ressourcen sind normalerweise auch ziemlich ernst, aber begrenzter.

- *In Kapitel 4 und 8 finden Sie inhaltsbezogene Vorsichtsmaßnahmen für HTTPS-Sites.*
- *In Kapitel 11 finden Sie einen Überblick über Verarbeitungsregeln* für Mixed Content.

18.1.6 Open Redirection

Ein Begriff für Anwendungen, die Requests auf HTTP- oder Skriptbasis an URLs durchführen, die der Benutzer bereitstellt, ohne die möglichen Ziele wirksam einzuschränken. Open Redirection (offene Umleitung) ist nicht ratsam und lässt sich in manchen Szenarios ausnutzen. Normalerweise ist es aber nicht besonders gefährlich.

In Kapitel 10 finden Sie Fälle, in denen unbeschränkte Umleitung abhängig vom verwendeten Browser zu XSS führen kann.

18.1.7 Referrer-Leaks (Referrer Leaking)

Mit diesem Begriff bezeichnet man die versehentliche Offenlegung einer sensiblen URL durch Einbetten einer Site-externen Ressource oder durch Bereitstellen eines Links, der die Site verlässt. Sicherheitsrelevante oder private Daten in der URL des übergeordneten Dokuments sickern dabei in den Header `Referer` durch; Ausnahme ist hier der Fragment-Identifier.

Lassen Sie sich nicht von den zwei Schreibweisen irritieren: *Referrer* (als allgemeine Bezeichnung) und `Referer` (der Name des Headers). Der RFC enthielt damals einen Rechtschreibfehler, der zur Benennung des Headers mit einem statt zwei »r« führte.

- *In Kapitel 3 finden Sie einen Überblick über die Referer-Logik.*

18.2 Probleme beim Design von Webanwendungen

Die im folgenden Abschnitt skizzierten Probleme sind eine zwangsläufige Begleiterscheinung beim Aufenthalt im Internet. Sie müssen beim Design und bei der Implementierung neuer Webanwendungen sorgfältig berücksichtigt werden.

18.2.1 Cache Poisoning

Die Möglichkeit der langfristigen »Vergiftung« des Browsercache (oder der dazwischenliegenden Proxys) durch eine verfälschte, bösartige Version der Zielanwendung wird als Cache Poisoning bezeichnet. Verschlüsselte Webanwendungen können aufgrund von Response-Splitting-Schwächen zum Ziel eines Angriffs werden. Bei nicht verschlüsselten Webanwendungen können aktive Angreifer des Netzwerks möglicherweise auch die beim Anfordernden eingehenden Antworten modifizieren.

➠ *In Kapitel 3 finden Sie einen Überblick über die Verhaltensweisen beim Caching von HTTP-Code.*

18.2.2 Clickjacking

Clickjacking bezeichnet die Möglichkeit, einen Teil einer anderen Webanwendung zu »framen« oder anderweitig zu dekorieren bzw. zu verdecken, sodass das Opfer bei der Interaktion mit der Site des Angreifers nicht merkt, dass einzelne Klicks oder Tastendrücke an die andere Site übermittelt werden. Dies führt dazu, dass im Namen des Benutzers unerwünschte Aktionen ausgeführt werden.

➠ *In Kapitel 11 finden Sie eine Erörterung des Clickjacking und ähnlicher Probleme mit der Benutzerschnittstelle.*

18.2.3 Content-Sniffing/Charset-Sniffing

Mit Content-Sniffing wird ein Szenario beschrieben, in dem der Browser die maßgeblichen Daten vom Server über Inhaltstyp und Zeichensatz ignoriert und das gelieferte Dokument falsch wiedergibt.

➠ *In Kapitel 13 finden Sie eine Erörterung der Logik zum Content-Sniffing.*

➠ *In Kapitel 4 und 8 finden Sie Szenarios, in denen Content-Type-Daten ignoriert werden.*

18.2.4 Cookie Forcing/Cookie Injection

Dies bezeichnet die Möglichkeit, aufgrund von Implementierungsproblemen in modernen Browsern und oft leicht zu übersehender Bugs in Webapplikationen blind HTTP-Cookies in den Kontext einer sonst vor Angriffen sicheren Weban-

wendung einzuschleusen. (*Cookie Stuffing* ist ein weniger gängiger Begriff; er bezeichnet speziell das böswillige Löschen von Cookies, die einer anderen Anwendung gehören, durch einen bewusst herbeigeführten Überlauf des sogenannten »Cookie Jar«.)

- *In Kapitel 9 finden Sie weitere Informationen zum Gültigkeitsbereich von Cookies.*
- *In Kapitel 3 finden Sie eine allgemeine Erörterung der Funktionsweise von HTTP-Cookies.*

18.2.5 DoS-Angriffe (Denial of Service)

Dieser breite Begriff umfasst alle Möglichkeiten und Techniken, mit denen ein Angreifer einen Browser oder Server zum Absturz bringen oder die Benutzung einer konkreten Zielanwendung auf andere Art erheblich erschweren kann.

- *In Kapitel 14 finden Sie einen Überblick über DoS-Themen im Zusammenhang mit JavaScript.*

18.2.6 Frame Busting

Wenn eine Seite in einem Frame das Top-Level-Dokument an eine neue URL leiten kann, ohne dass eine Same-Origin-Prüfung durchgeführt wird, spricht man von Frame Busting. Dieses Verhalten kann für Phishing-Angriffe oder einfach für trivialen Unfug genutzt werden. Oft werden Frame Buster sogar als Schutzmethode eingesetzt – je nach Kontext handelt es sich also gar nicht um einen Angriff.

- *In Kapitel 11 finden Sie diese und andere Mängel bei der Framenavigation.*

18.2.7 HTTP-Downgrades

Wenn aktive Angreifer den Benutzer daran hindern, eine HTTPS-Version einer bestimmten Site zu erreichen, oder wenn sie eine bestehende HTTPS-Sitzung auf HTTP herabstufen, spricht man von HTTP-Downgrades.

- *In Kapitel 3 finden Sie einen Überblick über HTTPS.*
- *In Kapitel 16 finden Sie etwas über Strict Transport Security, einen Lösungsvorschlag für dieses Problem.*

18.2.8 Network Fenceposts

Damit ist gemeint, dass Websites im Internet den Browser zur Interaktion mit Zielen einsetzen können, die für den Angreifer nicht direkt zugänglich sind, zum Beispiel mit den Systemen im Intranet des Opfers. Man stelle sich etwa vor, eine Seite im Internet versucht, die Ressource *http://intranet/ausloggen.aspx* zu laden.

Derartige Angriffe sind eng verwandt mit CSRF und können oft nur blind durchgeführt werden. Der Angreifer kann (mithilfe von Angriffen wie DNS-Rebinding) aber in der Lage sein, Antworten auf alle Requests zu sehen.

- *In Kapitel 12 finden Sie einen Überblick über Grenzen und Richtlinien jenseits der SOP in modernen Browsern.*
- *In Kapitel 15 wird das Zonenmodell des Internet Explorer beschrieben, ein potenzieller Ansatz für dieses Risiko.*

Hinweis

Vorsicht bei Bugs ohne Buzzword! Nicht alle Schwachstellen haben einprägsame Namen. Webentwickler sollten auf zahlreiche andere Implementierungs- und Designprobleme achten, die außerhalb des Fokus dieses Buches liegen, trotzdem aber wehtun können. Beispiele sind schwache Pseudozufallsgeneratoren für Zahlen (insbesondere für die Sitzungsverwaltung), nicht ausreichende Authentifizierungs- und Autorisierungsprüfungen (insbesondere übermäßiges Vertrauen auf Daten, die vom Browser stammen), nicht korrekte Verwendung von Kryptografie (die Erfindung eigener Algorithmen ist üblicherweise streng verboten) usw. Eine sehr ausführliche Erörterung dieser und vieler anderer Fehlermuster finden Sie in *The Art of Software Security Assessment* von Dowd, McDonald und Schuh (Addison-Wesley 2006).

18.3 Häufige Probleme bei serverseitigem Code

Die folgenden Probleme sind im Allgemeinen im serverseitigen Teil einer Webanwendung anzutreffen und kommen dank der Verbindung mit bestimmten Programmiersprachen oder Softwarekomponenten auf der Clientseite wahrscheinlich nicht vor.

18.3.1 Buffer Overflow

Ein Pufferüberlauf (Buffer Overflow) tritt auf, wenn ein Programm das Ablegen von mehr Informationen in einem bestimmten Speicherbereich zulässt, als Platz für die eingehenden Daten vorhanden ist. Dies kann dann zum unerwarteten Überschreiben anderer wesentlicher Datenstrukturen führen. So etwas kommt hauptsächlich in niederen Programmiersprachen wie C oder C++ vor und kann dort häufig ausgenutzt werden, um vom Angreifer gelieferten Code auszuführen.

18.3.2 Command-Injection (SQL, Shell, PHP usw.)

Von Command-Injection spricht man, wenn die Filterung von Eingaben oder die Maskierung von Ausgaben ungenügend durchgeführt wird und vom Angreifer gesteuerte Zeichenfolgen dadurch als Anweisungen einer von der Anwendung

benutzten Interpretersprache verarbeitet werden (ähnelt entfernt XSS). Die Folgen sind von den Fähigkeiten der Sprache abhängig. Meistens ist das Endergebnis jedoch, dass beliebiger Code ausgeführt wird.

18.3.3 Directory/Path Traversal

Bei diesem Problem kann eine Anwendung aufgrund unzureichender Eingabefilterung (meistens die fehlerhafte Erkennung und Handhabung von `../`-Segmenten in Dateinamen) dazu verleitet werden, Dateien an beliebigen Orten auf der Festplatte zu lesen oder zu schreiben. Die Folgen eines solchen Problems hängen oft davon ab, wie die Applikation intern mit den eingelesenen Daten umgeht. Sobald ein Angreifer jedoch Dateien von »Remote« einbinden kann, ist man üblicherweise nicht weit von einer »Remote Code Execution« entfernt.

18.3.4 File Inclusion

Wird der Begriff File Inclusion mit dem Präfix *Local* (LFI) oder ohne nähere Bezeichnung verwendet, ist er weitgehend synonym zum klassischen Directory Traversal mit Lesezugriff auf lokale Dateien des angegriffenen Servers. *Remote File Inclusion (RFI)* ist dagegen eine alternative Methode, eine File-Inclusion-Schwachstelle durch Angabe einer URL anstelle eines gültigen Dateipfads auszunutzen. In manchen Skriptsprachen – darunter natürlich an erster Stelle PHP – öffnen gängige APIs lokale Dateien und rufen je nach Konfiguration URLs von anderen Servern ab. In diesen Fällen kann die Fähigkeit, eine Datei von einem vom Angreifer kontrollierten Server abzurufen, die Auswirkung der Lücke wesentlich verschlimmern, je nachdem, wie die Daten später verarbeitet werden.

18.3.5 Format-String-Schwachstellen

Eine Handvoll häufig verwendeter Bibliotheksfunktionen akzeptiert Templates (»Format Strings«), auf die eine Reihe von Parametern folgt, die die Funktion an vordefinierten Stellen in die Vorlage einsetzen soll. Solche Ansätze kommen besonders in C häufig vor (`printf(...)`, `syslog(...)` usw.), sind aber nicht auf diese Sprache beschränkt. Diese Schwächen werden dadurch verursacht, dass Angreifer versehentlich die Erlaubnis bekommen, einer dieser Funktionen das Template bereitzustellen. Abhängig von den Fähigkeiten des Template-Systems und den Besonderheiten der Sprache kann dieser Fehler Folgen haben, die von kleineren Datenlecks bis hin zur Ausführung von Code reichen. In Webapplikationen finden sich derlei Probleme eher selten – es sei denn, die Applikation ist in einer Sprache wie C++ oder gar C verfasst. Auch dies soll hin und wieder vorkommen.

18.3.6 Integer Overflow

Integer Overflow (Integer-Überlauf) ist eine spezifische Schwäche von Sprachen mit begrenzter oder fehlender Bereichsprüfung. Sie tritt auf, wenn der Entwickler nicht merkt, dass eine Integerzahl den möglichen Höchstwert überschritten hat und dann auf null, auf eine sehr große negative Integerzahl oder in einen anderen hardwarespezifischen und unerwarteten Zustand zurückgefallen ist. Je nach Verwendung des Wertes kann dies das Programm in einen inkonsistenten Zustand versetzen oder, noch schlimmer, dazu führen, dass Daten an einer falschen Speicherstelle gelesen oder geschrieben werden (was sich wiederum anbietet, um Code auszuführen). Der gegenteilige Effekt ist *Integer Underflow*: das Unterschreiten des zulässigen Minimalwertes und der Übergang zu einer sehr großen positiven Zahl.

18.3.7 Schwachstellen in der Pointer-Arithmetik

In Sprachen, die die Verwendung roher Pointer auf Speicheradressen begünstigen oder gar verlangen (hauptsächlich C und C++), ist es möglich, Pointer zu verwenden, die entweder nicht initialisiert oder nicht mehr gültig sind (»stale« oder »abgestanden«). Dies führt zu Schwachstellen wie *use after free*, *double free* und vielen anderen. Sie beschädigen den internen Zustand des Programms und erlauben einem Angreifer normalerweise, von ihm gelieferten Code auszuführen.