GAI NetConsult GmbH - 13507 Berlin – Am Borsigturm 58 – www.gai-netconsult.de – Tel: 030-417898-0

Sicherheit bei Webanwendungen

Detlef Weidenhammer

Whitepaper *Januar 2007*

Immer mehr Unternehmen setzen im Zeitalter des aufkommenden eBusiness verstärkt auf Webanwendungen, um über das Internet Partnern und Kunden einen einfachen Zugriff auf bereit gestellte Geschäftsvorgänge und -daten zu ermöglichen. Dieser Zugriffsweg ist aber auch mit hohen Sicherheitsrisiken verbunden, wie aktuelle Vorkommnisse und die Ergebnisse von Sicherheitsüberprüfungen deutlich zeigen. In diesem Beitrag werden typische Schwachstellen von Webanwendungen aufgezeigt und Schutzmaßnahmen von der Programmierung bis zum Betrieb vorgestellt.

Webanwendungen sind technisch gesehen Client/Server-Anwendungen, die über das Transportprotokoll HTTP eine Interaktion zwischen Unternehmen oder von Unternehmen zu Kunden realisieren. Den Möglichkeiten einer Webanwendung sind dabei kaum Grenzen gesetzt. Diese reichen von der einfachen Recherche in lokalen Dateiverzeichnissen bis hin zu den komplexen Funktionen eines Online-Sales oder Online-Bankings. Wurden zunächst vorwiegend einfache CGI-Anwendungen implementiert, die auf dem Webserver selber abliefen, so basieren heutige Webanwendungen zumeist auf Java und laufen verteilt auf mehreren Servern. Eine strenge Definition des Begriffs Webanwendung gibt es bisher nicht, üblicherweise versteht man darunter Anwendungen, die zumindest 3-stufig mit Presentation-, Application- und Data-Layer aufgebaut sind und über HTTP mit einem Anwender oder einer anderen Webanwendung kommunizieren. In diesem Sinne zählen auch Webportale und Web-Services dazu. Die in diesem Beitrag angesprochenen Sicherheitsprobleme sind weitgehend identisch, auf deutliche Unterschiede wird hingewiesen. Eine ausführliche Abhandlung zu „Web-Services und Security“ findet sich auch in der Ausgabe 10 unseres *Security Journals* (www.gai-netconsult.de/journal).

Dem Thema Sicherheit ist bei der Erstellung und Nutzung von Webanwendungen eine besondere Bedeutung beizumessen. Dies ist für die Nutzerakzeptanz immens wichtig, ebenso aber auch für den Anbieter selber, der für die bereit gestellten Ressourcen Verantwortung trägt. Für den Nutzer ist der Schutz seiner persönlichen Daten und der Transaktionsdaten während der Übertragung von großer Bedeutung. Für den Anbieter ergeben sich ganz neue Risiken, wenn die Webanwendung komplexe Funktionen mit Zugriff auf interne Datenbestände bereitstellt. Damit hat prinzipiell jeder Teilnehmer im Internet die Möglichkeit mittels einfacher HTTP-Requests die Sicherheit dieser Anwendung zu testen. Angriffsversuche verbergen sich im legitimen HTTP-Datenstrom und werden durch keine Firewalls oder Intrusion-Detection Systeme erkannt und auch nicht durch den Einsatz einer SSL-Verschlüsselung verhindert. Es ergeben sich damit eine Vielzahl von Sicherheitsproblemen bei Webanwendungen (siehe Abb. 1).

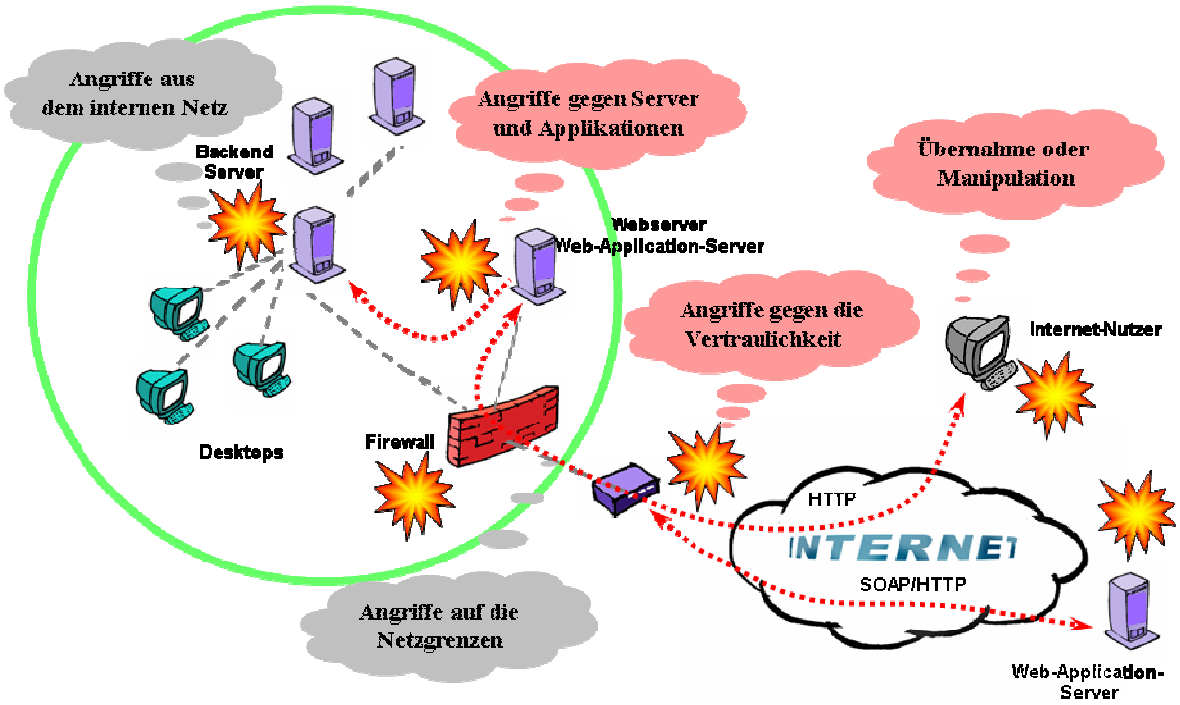

Abbildung-1: Sicherheitsprobleme bei Webanwendungen

Als wesentlicher Punkt ist festzuhalten: die Webanwendung ist Teil der Netzgrenze des Unternehmens (des Perimeters) und damit ebenso gut wie diese zu schützen. Doch wie sieht die Realität aus? Herkömmliche Schutzmaßnahmen mit Firewalls haben ihre Stärken in der Kontrolle auf Netzebene, können aber auf Anwendungsebene bestenfalls offensichtlichen Missbrauch erkennen. Aktuelle Angriffe zielen aber eher auf Schwachstellen innerhalb einer Anwendung. Ursache hierfür ist zumeist die mangelhafte Beachtung von Sicherheitsaspekten bei der Programmierung. Besonders attraktiv für Angreifer ist die Tatsache, dass für einen Großteil der möglichen Attacken kein spezielles Tool benötigt wird, ein Standard-Webbrowser ist vollkommen ausreichend. Manipulierte HTTP-Requests sind somit in der Lage, Schwachstellen auf allen Ebenen einer Webanwendung auszunutzen (siehe Abb. 2).

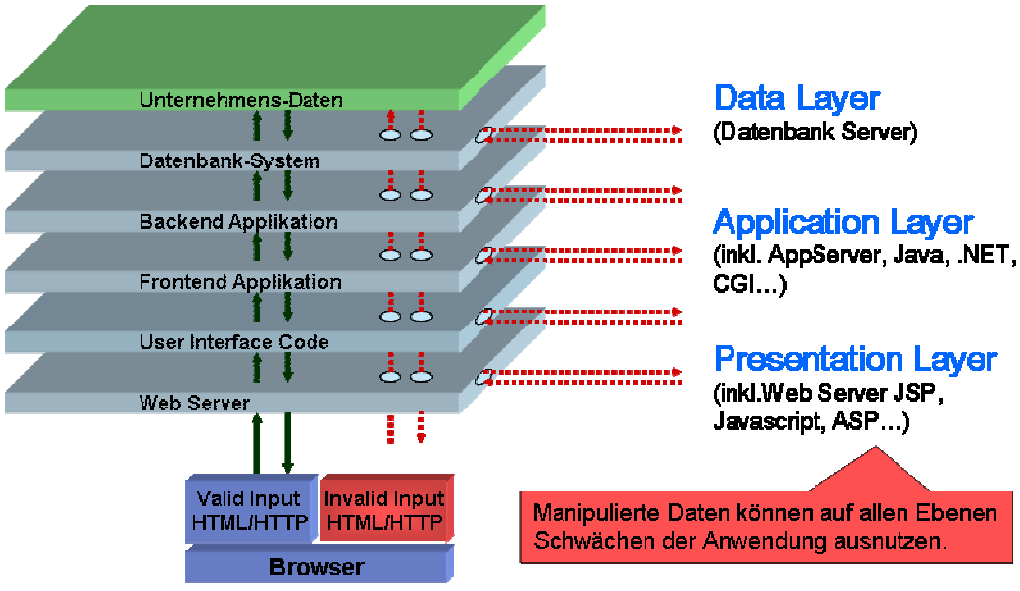

Abbildung-2: Zielpunkte manipulierter HTTP-Requests

Wie deshalb auch zu erwarten war, konnte man in der letzten Zeit eine deutliche Zunahme der Angriffe auf Webanwendungen registrieren. Die alljährliche Studie von CSI/FBI (www.gocsi.com) ermittelte in den letzten drei Jahren einen starken Anstieg und im Report-2006 bei insgesamt 92% der befragten Unternehmen Angriffe auf deren Websites. Diese erschreckenden Ergebnisse decken sich mit unseren eigenen Erfahrungen aus einer Vielzahl von Sicherheitsüberprüfungen. Dabei konnten wir bei über 90% der untersuchten Webanwendungen die gefundenen Schwachstellen zu gravierenden Manipulationen ausnutzen. Hierzu gehörten:

- Ausführen von Kommandos auf dem Webserver oder auf fremden Clients
- Vollständige Übernahme des Webservers
- Eindringen in das Unternehmensnetz
- Verändern der Webdarstellung („defacement“)
- Übernahme von fremden Sessions
- Auslesen von (intern gehaltenen) Datenbank-Inhalten

Welche Haltung die Hersteller zum Thema Sicherheit bei Webanwendungen einnehmen, lässt sich am ehesten mit einer Aussage von Microsoft zu Web-Services belegen:

"Since SOAP relies on HTTP as the transport mechanism, and most firewalls allow HTTP to pass through, you'll have no problem invoking SOAP endpoints from either side of a firewall".

Diese schon länger als „Port-80 Problematik“ bekannte Unsitte, den bei den meisten Unternehmen erlaubten HTTP-Verkehr zum Durchtunneln von Firewalls auch für andere Dienste zu nutzen, wird uns also wohl noch längere Zeit begleiten. Grund genug, sich mit den wichtigsten Bedrohungen bei der Nutzung von Webanwendungen intensiv zu beschäftigen.

Schwachstellen bei Webanwendungen

Es ergeben sich eine Vielzahl von Sicherheitsproblemen bei Webanwendungen, die in einer Studie der Firma Imperva (www.imperva.com) sehr deutlich aufgezeigt wurden. Über vier Jahre wurden die Ergebnisse von Penetrationstests bei Webanwendungen aufgezeichnet und ausgewertet. Insgesamt waren 92% der getesteten Webanwendungen verwundbar. Es zeigte sich in den letzten Jahren auch ein deutlicher Anstieg des Risikos, Opfer von Attacken in diesem Bereich zu werden. Die ermittelten Schäden verteilten sich im Wesentlichen auf Informationsdiebstahl (57%), direkte finanzielle Verluste (22%), Denial-of-Service Angriffe (11%) und Ausführung von eingebrachten Code (8%). Hinter jeder Schadenskategorie verbergen sich unterschiedlich viele konkrete Angriffstechniken, auf die später noch eingegangen wird. Ebenfalls aus der Untersuchung von Imperva ergaben sich als häufigste Angriffspunkte bei Webanwendungen:

- Cross-Site Scripting (80 %)
- SQL-Injection (62 %)
- Parameter Manipulation (61 %)
- Cookie Poisoning (36 %)
- Known Vulnerabilities Database Server (33 %)
- Known Vulnerabilities Web Server (27 %)
- Buffer Overflows (20 %)

Nachfolgend werden die am häufigsten anzutreffenden Fehler bei der Implementierung von Webanwendungen und die daraus resultierenden Sicherheitslücken und Bedrohungen aufgezeigt und Möglichkeiten angegeben, wie diese vermieden werden können.

Sinnvoll ist zunächst die Kategorisierung einzelner Angriffstechniken und Schwachstellen. Hierzu gibt es mehrere Ansätze, von denen wohl die Veröffentlichung des OWASP-Projekts

(**O**pen **W**eb **A**pplication **S**ecurity **P**roject, www.owasp.org) „The Ten Most Critical Web Application Security Vulnerabilities – 2004 Update“ am weitesten verbreitet sein dürfte.

Schwachstellen	Beschreibung
Nicht validierte Eingabeparameter	Web-Requests werden nicht auf Manipulationen geprüft. Diese sind möglich z.B. in URL, sämtlichen HTTP-Headern, Cookies und Formularfeldern.
Ungenügende Zugriffskontrolle	Beschränkungen für authentisierte Nutzer werden nur unzureichend umgesetzt. Angreifer können dies ausnutzen, um auf andere Nutzerkonten zuzugreifen, sensible Daten ausspähen oder geschützte Funktionen ausführen.
Ungenügende Authentisierung und Session Management	Zugangs- und Sessioninformationen werden nicht ausreichend geschützt. Angreifer können Passwörter ermitteln, Session Token oder andere Credentials fälschen und damit unberechtigte Aktivitäten ausführen.
Cross-Site Scripting (XSS)	Die Webanwendung wird benutzt, um Angriffe gegen andere Nutzer durchzuführen. Damit können z.B. Session Cookies ausgelesen oder gefälschte Webinhalte untergeschoben werden
Buffer Overflows	Durch ungeprüftes Überschreiben von Puffergrenzen können Anwendungsserver gestört oder sogar ganz übernommen werden.
Einschleusen von Befehlen	Webanwendungen kommunizieren mit anderen Systemen durch Parameterübergabe. Kann ein Angreifer eigene Befehle in die Parameter einbringen, werden diese bei fehlender Kontrolle ausgeführt. Bekanntestes Beispiel ist die SQL-Injection, deren Folgen die Modifikation von Webseiten, die Kompromittierung sensibler Daten oder – bei bestimmten Datenbankservern – auch die Ausführung beliebiger Systembefehle umfassen kann.
Unzureichende Handhabung von Anwendungsfehlern	Werden Fehlermeldungen der Webanwendung ungefiltert weitergegeben, besteht die Gefahr, dass ein Angreifer daraus wertvolle Informationen gewinnen kann.
Unzureichende Nutzung von kryptographischen Methoden	Werden ungeeignete kryptographische Methoden eingesetzt oder geeignete Methoden fehlerhaft implementiert, besteht kein ausreichender Schutz mehr für sensible Informationen.
Denial-of-Service Angriffe (DoS)	Angriffe gegen die Verfügbarkeit von Systemen oder Anwendungen.
Unsichere Konfiguration der Server	Ist eine sichere Serverkonfiguration nicht gegeben, bestehen auch hohe Risiken für eine Webanwendung. Da eine Standardauslieferung zumeist nicht ausreichend ist, muss ein Hardening erfolgen.

Beispiel zum Hacken einer Webanwendung

Um die Arbeitsweise eines Hackers transparenter zu machen, wird diese nachfolgend beispielhaft vorgestellt. Die gezeigten Techniken sind bekannt und leicht im Internet nachzulesen, trotzdem weisen wir ausdrücklich darauf hin, dass eine missbräuchliche Verwendung zu unterlassen ist.

Zielsuche mit Google

Die Suche nach verwundbaren Webanwendungen gestaltet sich meist mehr als einfach. Legt der Angreifer es nicht auf bestimmte Webseiten bzw. Anwendungen – deren Unterscheidung ja meist fließend ist – an, bringt meist schon eine gut formulierte Google-Suche ausreichend „Material“.

Als Beispiel haben wir für diesen Artikel die webbasierte Portalanwendung „PortalApp“ ausgewählt. Diese Anwendung basiert auf ASP-Seiten mit Datenbankanbindung, hier kann entweder MS Access oder der MSSQL-Server genutzt werden. In der Default-Konfiguration verwendet diese Anwendung den HTML <title> Tag „//ASPapp/PortalApp/Freeware Portal Software/Links, Content and Community Management“. Google ermöglicht über den Parameter „intitle“ die direkte Suche nach diesem HTML-Tag. Eine Suche in Google nach dem String „intitle://ASPapp/PortalApp“ hat dann auch immerhin über 7.000 Hits ergeben. Wohlgemerkt, hier wurden nur Webseiten gefunden, die den originalen HTML <title> Tag nicht geändert haben. Es wird vermutlich noch wesentlich mehr Installationen geben. Der Angreifer darf sich also fühlen, wie der Hecht im Karpfenteich.

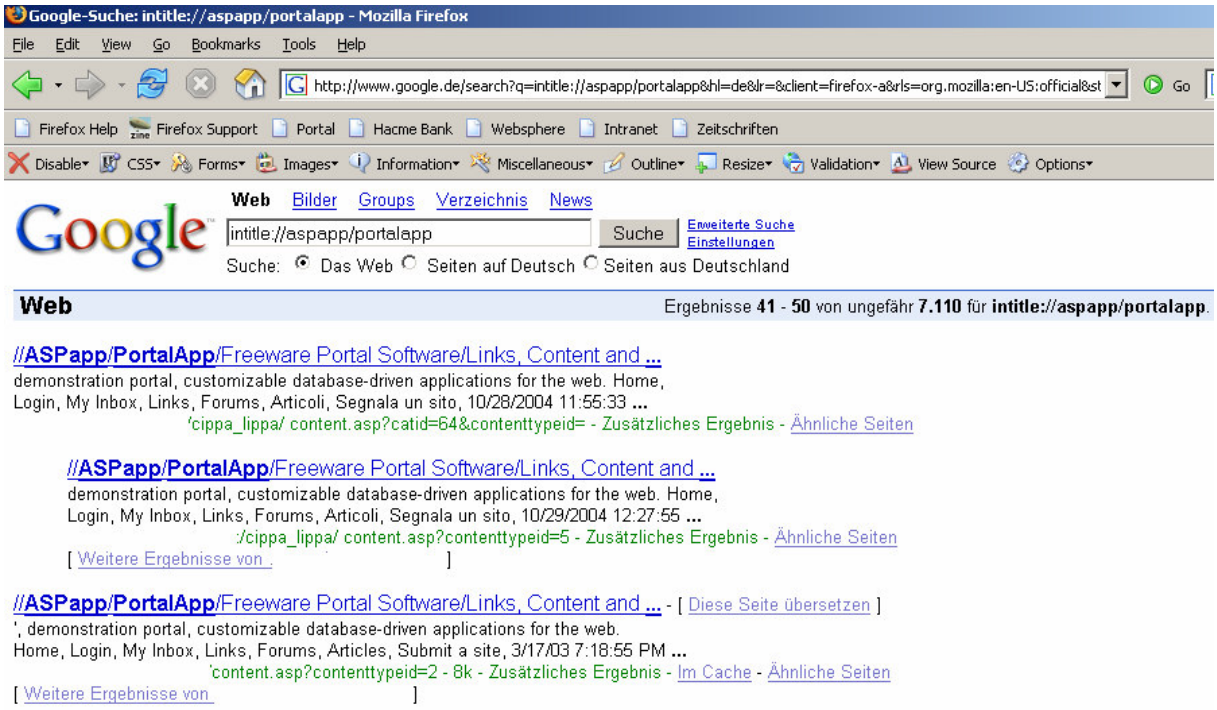

Abbildung-3: Screenshot des Resultats einer Google-Suche nach intitle://aspapp/portalapp

Natürlich werden wir unseren beispielhaften Angriff nicht auf eine produktiv eingesetzte Anwendung durchführen. Die im Folgenden geschilderten Angriffe beziehen sich daher auf eine Testinstallation im Labor der GAI NetConsult, wären aber sicherlich auch auf einige der im Internet genutzten Anwendungen anwendbar.

Fehlerhafte Beispielskripte

Bei vielen Web- und Webanwendungsservern werden Skripte mitgeliefert, um beispielhaft bestimmte Funktionen des Systems zu zeigen. So auch bei dem in unserem Beispiel genutzten Microsoft Internet Information Server. Eine der hier mitgelieferten ASP-Beispielskripte - die Codebrws.asp – verfügt über eine so genannte Directory Traversal Schwachstelle. Eigentlich soll die Codebrws.asp nur den Quellcode von ASP-Seiten innerhalb des Webserver-Verzeichnisses /iissamples anzeigen. Die Schwachstelle ermöglicht es einem Angreifer jedoch, den Quellcode jedes ASP-Skripts auszulesen. Details zu dieser Sicherheitslücke finden sich unter: http://www.securityfocus.com/bid/4525.

Diese Schwachstelle ist nicht in jedem Fall kritisch, da ASP-Quellcode ja nicht unbedingt sensible Informationen enthalten muss. Analysiert man aber den Quellcode unser Anwendung kann festgestellt werden, dass die zum Zugriff auf die Datenbank genutzten Authentisierungsdaten im ASP-Quellcode gehalten werden:

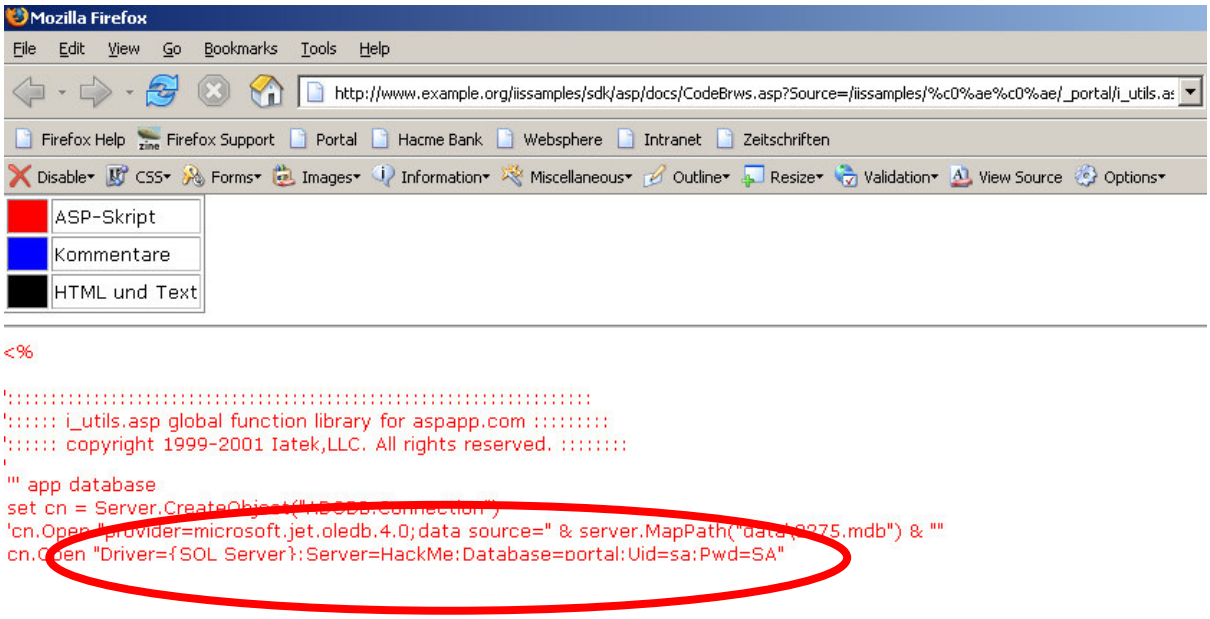

Abbildung-4: Ausgelesener Quellcode der Konfiguration des Datenbankzugriffs

Ersichtlich ist hier, dass zum Datenbankzugriff das administrative Konto des SQL-Servers „sa“ genutzt wird. Welche besonderen Implikationen sich aus dieser Konfiguration ergeben wird später noch erläutert.

Angriff !

In derselben ASP-Datei ist auch die zum Datenbankzugriff genutzte Funktion to_sql enthalten, die innerhalb der Anwendung global für die Formulierung dynamischer SQL-Statements genutzt wird:

```
function to_sql(Value,DataType)
    if Value = "" or isNull(Value) then
        to_sql = "NULL"
    elseif DataType = "date" AND instr(lcase(cn.Provider),"jet") > 0 then
        to_sql = "#" & Replace(Value, "'", "''") & "#"
    elseif DataType <> "number" then
        to_sql = "'" & Replace(Value, "'", "''") & "'"
    else
        to_sql = Value
    end if
end function
```

Abbildung-5: Screenshot der Funktion to_sql

Der entscheidende Punkt ist hier, dass bei allen Variablen vom Typ „number“ (genauer gesagt, bei allen Variablen, die kein Datum oder keine Nummer sind) die durch den Nutzer beeinflussten Parameterwerte (z.B. contentid=6) direkt an den Datenbanktreiber übergeben werden! Dies führt letztlich zu dem als **SQL-Injection** bezeichneten Sicherheitsproblem, da der Angreifer beliebige zusätzliche SQL-Statements an die Datenbank übergeben – injecten – kann.

Möglich sind durch Ausnutzung von SQL-Injection Schwachstellen quasi beliebige Datenbankoperationen. Beispielhaft wird in dem folgenden Angriff der Nutzernamen und das Passwort des administrativen Kontos der Anwendung ausgelesen.

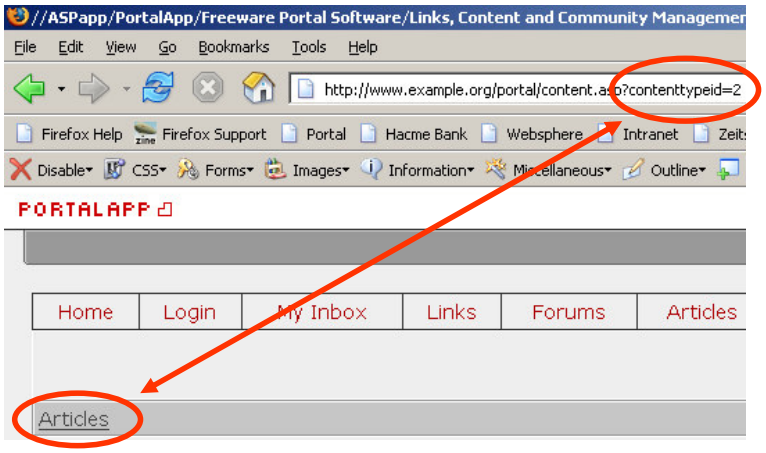

Abbildung-6: Screenshot der Anwendung vor der Manipulation

Der Parameter Contenttypeid=2 besagt, dass es sich bei dem Typ des Contents um einen „Article“ handelt. Über Manipulation des übergebenen Parameterwertes können nun beliebige zusätzliche SQL-Statements an die eigentliche Abfrage angehängt werden:

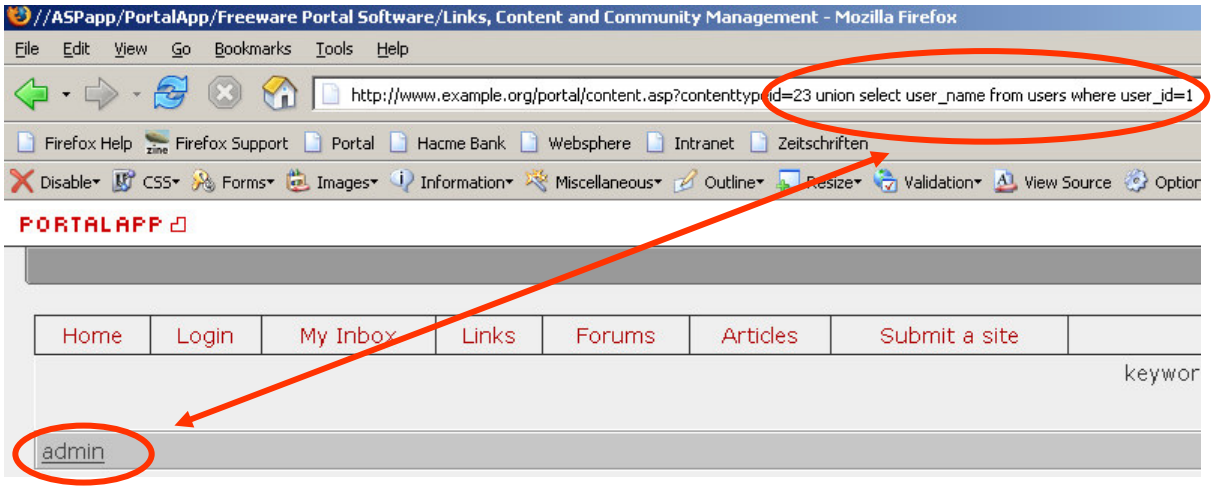

Abbildung-7: Anhängen von SQL-Statements

Das angehängt UNION SELECT gaukelt der Anwendung vor, dass das Resultat der eigentlichen SELECT Anfrage dem Resultat des UNION SELECT entspräche. Voraussetzung ist, dass die ursprüngliche SQL-Abfrage nach der Contenttypeid kein Ergebnis bringt. Der Wert für die Contenttypeid muss daher auf einen nicht existenten Wert geändert werden. Ebenso kann nun auch das Passwort des administrativen Nutzerkontos der Anwendung ausgelesen werden:

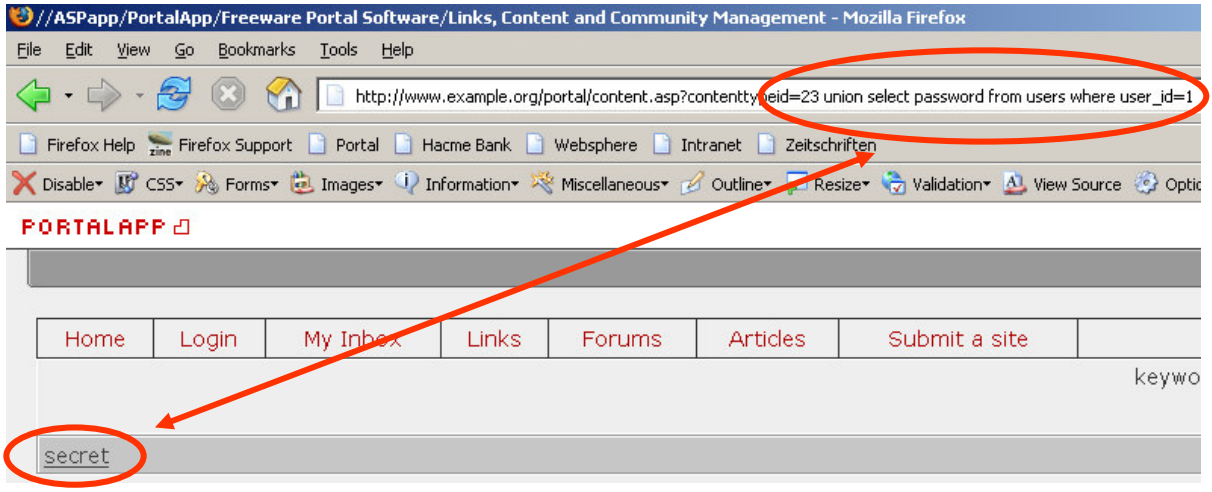

Abbildung-8: Auslesen des Passworts per SQL-Statement

Bei SQL-Injection Angriffen sind die im Beispiel verwendeten UNION SELECT Statements besonders beliebt, da so beliebige Daten aus jeder Tabelle der Datenbank ausgelesen werden können, ohne die eigentlichen Datenbankinhalte zu modifizieren.

Ausführung von Systembefehlen

Durch Analyse des Quellcodes der Anwendung konnte festgestellt werden, dass zum Zugriff auf die Datenbank das administrative Nutzerkonto „sa“ des MSSQL-Servers genutzt wird. Dies ist in den meisten Fällen völlig unnötig und ermöglicht dem Angreifer bei SQL-Injection Schwachstellen – neben dem Auslesen und der ebenfalls möglichen Manipulation von Datenbankinhalten – zusätzlich die unmittelbare Ausführung von Systembefehlen. Der MSSQL-Server liefert per Default eine so genannte erweiterte Stored Procedure (Extended Stored Procedure – abgekürzt mit xp) mit, die die Ausführung von Systembefehlen mit den Rechten des Nutzerkontos des SQL-Servers erlaubt. Dies bedeutet also letztlich, dass ein Angreifer durch einfaches Anhängen von zusätzlichen SQL-Statements an die URL im Browser jeden beliebigen Systembefehl auf dem SQL-Server ausführen kann! Da der MSSQL-Server in der Default Konfiguration mit SYSTEM-Privilegien arbeitet, kann sich ein Angreifer zusätzlich an den maximalen auf einem Windows-System möglichen Nutzerrechten erfreuen.

Ein Problem hat der Angreifer allerdings, die Ausgaben seines Befehls sind meist nicht direkt sichtbar, da die Anwendung die Ausgabe des Systembefehls nicht als Eingabe für den Aufbau der Webseite erwartet. In unserem Beispielangriff ist dies allerdings kein großes Problem, da Anwendung und Datenbankserver auf einem System laufen. Die Ausgabe eines Systembefehls kann daher einfach in eine Datei im Webroot-Verzeichnis des Internet Information Servers umgeleitet werden:

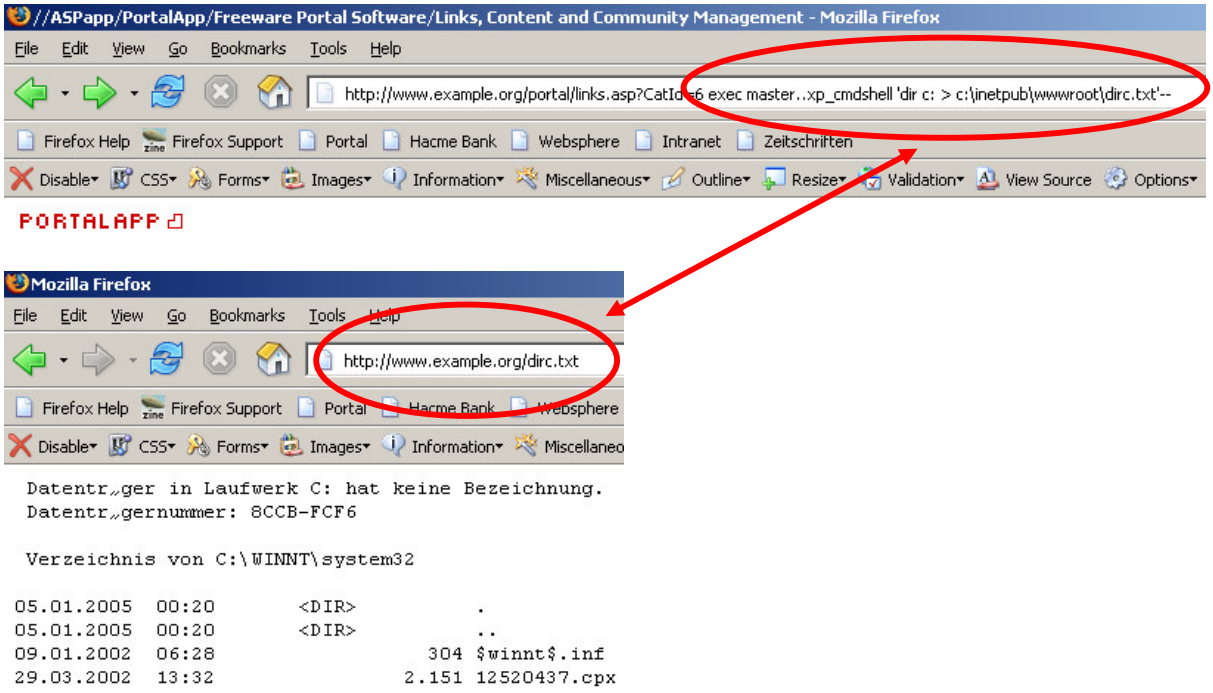

Abbildung-9: Umleiten der Ausgabe

Aber selbst wenn Web- und Datenbankserver auf unterschiedlichen Systemen betrieben werden, ist es für den Angreifer nicht sonderlich kompliziert, die Ausgaben seiner Befehle zu erhalten. Beispielsweise kann für diese Zwecke eine temporäre Datenbank angelegt werden – über SQL-Injection versteht sich - die Ausgaben der Befehle werden dann in diese Datenbank umgeleitet und mittels der demonstrierten UNION SELECT Technik wieder ausgelesen werden. Letztlich sind die Auswirkungen des Angriffs hier nur von der Kreativität des Angreifers abhängig und weniger von technischen Hindernissen.

Zusammenfassung der ausgenutzten Fehler in der Anwendung:

Beispielanwendungen wurden nicht entfernt

Über die Beispielanwendung Codebrws.asp konnte der Quellcode jeder ASP-Seite ausgelesen werden. Der Angreifer konnte sich so alle Details der Datenbankanbindung beschaffen und dieses Wissen bei späteren Angriffen einsetzen.

Fehlende Validierung von Nutzereingaben

Die Werte der übergebenen Parameter werden innerhalb der Anwendung nicht weiter überprüft, sondern dynamisch an den Datenbanktreiber übergeben. Dies ermöglicht die Durchführung von SQL-Injection Angriffen.

Zu weit gehende Rechte beim Zugriff auf die Datenbank

Zum Zugriff auf die Datenbank wurde das administrative Nutzerkonto „sa“ des MSSQL-Servers verwendet. Im Zusammenspiel mit SQL-Injection können so mit erweiterten Privilegien beliebige Systembefehle auf dem Datenbankserver ausgeführt werden.

Die 10 Tips zur Sicherung von Webanwendungen

Die folgende Aufstellung bietet Ihnen eine komprimierte Hilfe zum Aufbau von sicheren Webanwendungen. Es werden die häufigsten "Baustellen" und sicherheitskritischen Bereiche bei der Entwicklung und beim Betrieb von Webanwendungen betrachtet.

#1. Bereitstellen einer sicheren Serverumgebung

Die sicherste Anwendung hilft nicht gegen Sicherheitslücken in den darunter liegenden Komponenten. Das gesamte System bestehend aus Hardware, Betriebssystem, evtl. eingesetzter Middleware und der eigentlichen Anwendung muss grundsätzlich zusammen betrachtet und gesichert werden. Sicherheitspatches sind einzuspielen und die Konfiguration ist zusätzlich zu härten. Besonders gilt das für den eingesetzten Web-Server. Vergessene Demoanwendungen und ausführbare Directorylistings sind nur zwei Beispiele von leicht zu beseitigenden, aber trotzdem häufig vorzufindenden Schwachstellen.

#2. Netzwerkabsicherung der Webanwendung

Webanwendungen sollten immer in einem eigenen gesicherten Netzwerksegment platziert werden. So kann der Zugriff auf die Webanwendung durch eine Firewall auf die zwingend notwendigen Ports beschränkt werden (meist http/80 und ssl/443). Um einen Schutz auch auf Protokollebene zu ermöglichen, sollten dabei Application Level Proxies eingesetzt werden. Mittlerweile sind auch SSL-Proxies verfügbar, die in der Lage sind auch verschlüsselte Datenströme zu kontrollieren. Für einen weiter gehenden Schutz bieten sich so genannte Web Application Firewalls an, die als filternde Proxies vor die eigentliche Webanwendung geschaltet werden.

#3. Multi Tier Ansatz zur Auftrennung der Anwendungsmodule

Eine Auftrennung der Anwendung in verschiedene Module (häufig in Präsentations-, Anwendungs- und Daten-Schicht) und deren Verteilung auf verschiedene Server ermöglicht eine bessere Absicherung der einzelnen Teile. Die Kommunikation der einzelnen Module kann durch zusätzliche trennende Firewalls überwacht und kontrolliert werden. Auf keinen Fall sollten auf dem extern erreichbaren Web-Server sensible Daten lokal gespeichert werden. Eine einfache Trennung kann schon durch Vorschalten eines Reverse-Proxies erfolgen.

#4. Verwendung minimaler Rechte

Prozesse und die dazugehörigen User sollten immer nur mit den minimalen, gerade für die beabsichtigte Aufgabe ausreichenden Rechten arbeiten. Das betrifft z.B. den Web-Server, der normalerweise nur lesenden Zugriff auf die hinterlegten Inhalte besitzen muss, aber keine Schreibrechte benötigt. Ein weiteres Beispiel sind SQL-Anfragen an die Datenbank. Die zum Zugriff benutzte Datenbank-Kennung sollte nur die tatsächlich benötigten Rechte, auf keinen Fall aber weit reichende administrative Rechte (wie z.B. Änderung des Datenbankdesigns, Import/Export von Inhalten etc.) besitzen. Durch die konsequente Anwendung minimaler Rechte erschwert man Angreifern die einfache Ausnutzung noch vorhandener Sicherheitslücken.

#5. Nutzung anerkannter Sicherheits-Algorithmen

Im Bereich der Authentisierung und Verschlüsselung gibt es eine Reihe gut eingeführter und allgemein als sicher anerkannter Algorithmen. Die Entwickler sollten auf diese Algorithmen und Libraries zurückgreifen. Eigenentwicklungen sind ein kaum abzuschätzendes Sicherheitsrisiko. Gerade für die Standardaufgaben, wie Kennwortverschlüsselung, sichere Datenübertragung oder die Signierung von Daten gibt es für jede Entwicklungsumgebung eine reiche Auswahl an fertigen und sicheren Lösungen.

#6. "Security by obscurity" funktioniert nicht

Grundsätzlich sind sämtliche relevanten Ressourcen über Zugriffsmechanismen zu schützen. Nichtverlinkung oder „Verstecken“ alleine ist kein ausreichender Schutz. Ebenso sind Authentisierungsmerkmale, die leicht herausgefunden werden können (z.B.: Postleitzahl, Kundennummer etc.) oder vielleicht sogar in öffentlichen Teilen der Anwendung ersichtlich sind, als sehr schwach einzustufen. Bei der Prüfung einer Anwendung auf ihre Sicherheit sollte man immer von der Annahme ausgehen, dass dem Angreifer alle derartigen Informationen zur Verfügung stehen.

#7. Rigide Überprüfung von Ein-/Ausgabeparametern

Sämtliche Parameter, die vom Anwender (bzw. seinem Web-Browser) an die Webanwendung gesendet werden sind genauestens auf Gültigkeit hin zu überprüfen. Dies gilt auch für Parameter, die von der Webanwendung selbst in einem vorhergehenden Schritt zum Anwender geschickt wurden – auch diese können clientseitig mit entsprechenden Tools leicht manipuliert werden. Durch eine derartige Prüfung können eine ganze Reihe von typischen Schwachstellen (Code Injection, Buffer Overflows, etc.) vermieden werden. Dabei ist insbesondere zu beachten, dass der http-Standard die Kodierung von Zeichen in verschiedenen Repräsentationen vorsieht (z B. Unicode). Vor Prüfung der Parameter sind die Inhalte in dem verwendeten Zeichensatz zu normalisieren.

#8. Sichere Behandlung von Fehlerfällen

Alle Programme enthalten Fehler – entscheidend für die Sicherheit ist, wie Anwendungen in Fehlersituationen reagieren. Es sollte grundsätzlich ein "Fail save" Verhalten implementiert werden. Ein Sicherheitsmodul zur Authentisierung darf nicht bei einem Ausfall Anwender auch ohne Kennwort akzeptieren. Ein weiteres Beispiel ist Verschlüsselung (z.B. SSL): Der Fallback auf niedrige Verschlüsselungsstärke im Rahmen der Initialisierungsprozesse zwischen Client und Server muss explizit verhindert werden. Auch sollte der Anwender zwar mit Fehlermeldungen bei Fehlverhalten informiert werden, dabei dürfen aber keine für einen Angreifer verwertbaren Informationen mitgeliefert werden. Solche Informationen gehören ausschließlich in ein nur intern zugängliches Logfile.

#9. Beseitigen der häufigsten Schwachstellen in Webanwendungen

Es hat sich gezeigt, dass einige wenige typische Fehler bei der Implementierung zu einem Großteil der vorhandenen Schwachstellen beitragen. Zu beachten sind insbesondere:

a. *Ungeprüfte Ein- Ausgabeparameter*
   Hierzu gehören die hinlänglich bekannten Buffer Overflows, die aus fehlender Längenprüfung der Parameter resultieren, aber auch Fehler auf Anwendungsebene bei Übergabe von Daten.
b. *Zugriffsrechte fehlerhaft*
   Die Zugriffsrechte auf Dateien, Datenbanken oder Prozesse sind fehlerhaft oder unvollständig umgesetzt. Angreifer erhalten Zugriffe, zu denen sie nicht berechtigt sind.
c. *Fehlerhaftes Sessionmanagement*
   Die "Credentials" oder Tokens, die zur Herstellung einer Sessionzuordnung verwendet werden, sind nicht ausreichend gegen Manipulation geschützt. Angreifer können auf diese zugreifen und damit die geschützten Daten anderer Anwender lesen.
d. *Cross Site Scripting*
   Die Webanwendung kann dazu missbraucht werden, gefährlichen Scriptcode an die Browser anderer Anwender zu schicken. Der Angreifer kann dadurch z.B. die Session des betroffenen Anwenders übernehmen.
e. *Command Injection*
   Die Anwendung reicht ungeprüft Parameter an andere Anwendungen (z.B. SQL) oder das Betriebssystem weiter. Der Angreifer kann dadurch gefährliche Befehle an die Anwendung senden.

#10. Security Audit der Webanwendung

Zumindest Webanwendungen, die einen erhöhten Schutzbedarf besitzen, sollten vor Ihrer Produktivsetzung einem Security Audit unterzogen werden. Dabei ist zu beachten, dass nicht nur auf bekannte Fehler in Betriebssystem, Web-Server, Middleware etc geprüft wird, sondern auch die Webanwendung selbst auf die o.g. Probleme hin untersucht wird. Erfahrungsgemäß sollten dabei sowohl Tests ohne gültige Benutzerkennung als auch solche mit Zugang als registrierter Benutzer durchgeführt werden.

Fazit

Der Betrieb von Webanwendungen führt bereits heute zu den größten Sicherheitsrisiken für Unternehmen, die sich dieser Gefahren aber nur langsam bewusst werden. Durch neu aufkommende Techniken wie Web-Services wird diese Problematik in den nächsten Jahren eher noch gesteigert werden, deshalb sollte man sich beizeiten damit beschäftigen. Bei Eigenentwicklungen können einfache Programmierrichtlinien bereits die größten Schwachstellen von Beginn an vermeiden. Zusätzlichen Schutz liefern dann Produkte aus dem aufkommenden Markt der Web Application Firewalls (WAF).