Atakowanie uwierzytelniania

Na pierwszy rzut oka uwierzytelnianie jest koncepcyjnie jednym z najprostszych ze wszystkich mechanizmów bezpieczeństwa stosowanych w aplikacjach internetowych. W typowym przypadku użytkownik podaje swoją nazwę użytkownika i hasło, a aplikacja musi zweryfikować poprawność tych elementów. Jeśli tak, wpuszcza użytkownika. Jeśli nie, nie. Uwierzytelnianie leży również u podstaw ochrony aplikacji przed złośliwym atakiem. Jest pierwszą linią obrony przed nieautoryzowanym dostępem. Jeśli atakującemu uda się pokonać te zabezpieczenia, często uzyska pełną kontrolę nad funkcjonalnością aplikacji i nieograniczony dostęp do przechowywanych w niej danych. Bez niezawodnego uwierzytelniania, na którym można polegać, żaden z innych podstawowych mechanizmów bezpieczeństwa (takich jak zarządzanie sesją i kontrola dostępu) nie może być skuteczny. W rzeczywistości, pomimo pozornej prostoty, opracowanie bezpiecznej funkcji uwierzytelniania jest subtelną sprawą. W rzeczywistych aplikacjach internetowych uwierzytelnianie jest często najsłabszym ogniwem, które umożliwia atakującemu uzyskanie nieautoryzowanego dostępu. W tej części szczegółowo omówiono różnorodne wady projektowe i implementacyjne, które często dotykają aplikacje internetowe. Zwykle wynikają one z tego, że projektanci i programiści aplikacji nie zadają prostego pytania: co może osiągnąć osoba atakująca, jeśli zaatakuje nasz mechanizm uwierzytelniania? W większości przypadków, gdy tylko zostanie zadane to pytanie w odniesieniu do konkretnej aplikacji, pojawia się szereg potencjalnych luk, z których każda może wystarczyć do złamania aplikacji. Wiele z najczęstszych luk w zabezpieczeniach związanych z uwierzytelnianiem nie wymaga myślenia. Każdy może wpisać słowa ze słownika w formularzu logowania, próbując odgadnąć prawidłowe hasło. W innych przypadkach subtelne defekty mogą czaić się głęboko w przetwarzaniu aplikacji, które można wykryć i wykorzystać dopiero po żmudnej analizie złożonego wieloetapowego mechanizmu logowania. Opiszemy pełne spektrum tych ataków, w tym techniki, którym udało się złamać uwierzytelnianie niektórych z najbardziej krytycznych pod względem bezpieczeństwa i solidnie chronionych aplikacji internetowych na świecie.

Technologie uwierzytelniania

Szeroka gama technologii jest dostępna dla twórców aplikacji internetowych podczas wdrażania mechanizmów uwierzytelniania:

* Uwierzytelnianie oparte na formularzach HTML

* Mechanizmy wieloczynnikowe, takie jak łączenie haseł i tokenów fizycznych

* Certyfikaty klienta SSL i/lub karty inteligentne

* Uwierzytelnianie podstawowe i szyfrowane HTTP

* Uwierzytelnianie zintegrowane z systemem Windows przy użyciu protokołu NTLM lub Kerberos

* Usługi uwierzytelniania

Zdecydowanie najbardziej powszechny mechanizm uwierzytelniania stosowany w aplikacjach internetowych wykorzystuje formularze HTML do przechwytywania nazwy użytkownika i hasła oraz przesyłania ich do aplikacji. Ten mechanizm odpowiada za ponad 90% aplikacji, które prawdopodobnie napotkasz w Internecie. W aplikacjach internetowych o większym znaczeniu dla bezpieczeństwa, takich jak bankowość internetowa, ten podstawowy mechanizm jest często rozszerzany na wiele etapów, wymagając od użytkownika podania dodatkowych danych uwierzytelniających, takich jak kod PIN lub wybrane znaki tajnego słowa. Formularze HTML są nadal zwykle używane do przechwytywania odpowiednich danych. W aplikacjach o największym znaczeniu dla bezpieczeństwa, takich jak

bankowość prywatna dla zamożnych osób, często spotyka się mechanizmy wieloczynnikowe wykorzystujące tokeny fizyczne. Te tokeny zwykle generują strumień jednorazowych kodów dostępu lub wykonują funkcję odpowiedzi na wyzwanie na podstawie danych wejściowych określonych przez aplikację. Ponieważ koszt tej technologii spada z czasem, prawdopodobne jest, że więcej aplikacji będzie wykorzystywać ten rodzaj mechanizmu. Jednak wiele z tych rozwiązań w rzeczywistości nie eliminuje zagrożeń, dla których zostały opracowane - przede wszystkim ataków typu phishing i wykorzystujących trojany po stronie klienta. Niektóre aplikacje internetowe wykorzystują certyfikaty SSL po stronie klienta lub mechanizmy kryptograficzne zaimplementowane w kartach inteligentnych. Ze względu na narzut związany z administrowaniem i dystrybucją tych elementów są one zwykle używane tylko w kontekstach o krytycznym znaczeniu dla bezpieczeństwa, w których baza użytkowników aplikacji to centra handlowe, takich jak oparte na sieci VPN sieci VPN dla pracowników zdalnych biur. Mechanizmy uwierzytelniania oparte na protokole HTTP (podstawowe, szyfrowane i zintegrowane z systemem Windows) są rzadko używane w Internecie. Znacznie częściej spotyka się je w środowiskach intranetowych, w których użytkownicy wewnętrzni organizacji uzyskują dostęp do aplikacji korporacyjnych, podając swoje zwykłe poświadczenia sieciowe lub domeny. Następnie aplikacja przetwarza te poświadczenia przy użyciu jednej z tych technologii. Sporadycznie spotykane są usługi uwierzytelniania innych firm, takie jak Microsoft Passport, ale obecnie nie zostały one przyjęte na żadną znaczącą skalę. Większość luk i ataków pojawiających się w związku z uwierzytelnianiem można zastosować do dowolnej z wymienionych technologii. Ze względu na przytłaczającą dominację uwierzytelniania opartego na formularzach HTML, każdą lukę w zabezpieczeniach i atak opiszemy w tym kontekście. W stosownych przypadkach wskażemy wszelkie szczególne różnice i metodologie ataków, które są istotne dla innych dostępnych technologii.

Wady projektowe mechanizmów uwierzytelniania

Funkcjonalność uwierzytelniania podlega większej liczbie słabości projektowych niż jakikolwiek inny mechanizm bezpieczeństwa powszechnie stosowany w aplikacjach internetowych. Nawet w pozornie prostym, standardowym modelu, w którym aplikacja uwierzytelnia użytkowników na podstawie nazwy użytkownika i hasła, niedociągnięcia w projekcie tego modelu mogą narazić aplikację na nieautoryzowany dostęp.

Złe hasła

Wiele aplikacji internetowych nie stosuje żadnej kontroli lub stosuje minimalną kontrolę nad jakością haseł użytkowników. Często spotyka się aplikacje, które zezwalają na hasła, które są:

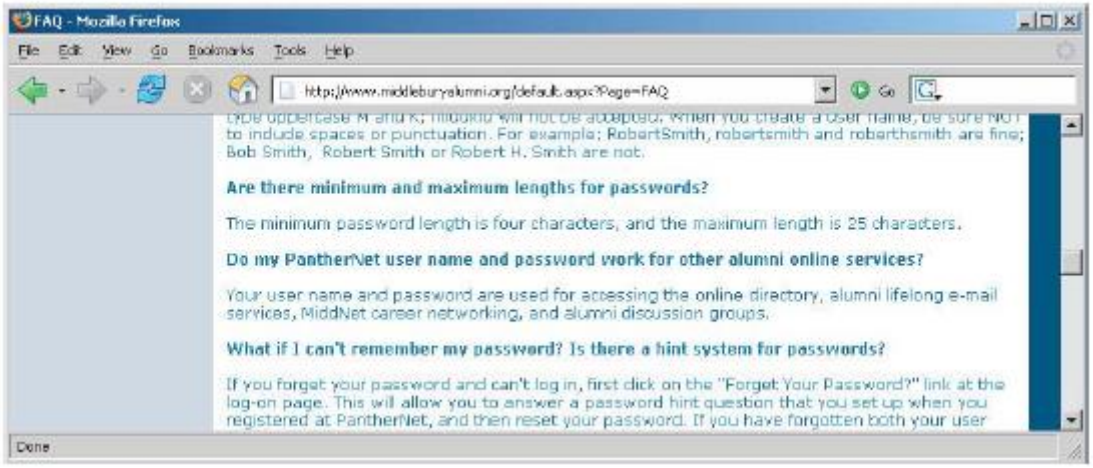
* Bardzo krótkie lub puste

* Wspólne słowa lub nazwy ze słownika

* To samo co nazwa użytkownika

* Nadal ustawione na wartość domyślną

Rysunek przedstawia przykład słabych reguł jakości hasła.



Użytkownicy końcowi zazwyczaj wykazują niewielką świadomość kwestii bezpieczeństwa. Dlatego jest wysoce prawdopodobne, że aplikacja, która nie wymusza silnych standardów haseł, będzie zawierała dużą liczbę kont użytkowników z ustawionymi słabymi hasłami. Atakujący może łatwo odgadnąć te hasła do kont, przyznając mu nieautoryzowany dostęp do aplikacji.

KROKI HACKOWANIA

Spróbuj odkryć wszelkie zasady dotyczące jakości hasła:

1. Przejrzyj witrynę pod kątem opisu zasad.

2. Jeśli możliwa jest samodzielna rejestracja, spróbuj zarejestrować kilka kont z różnymi rodzajami słabych haseł, aby dowiedzieć się, jakie zasady obowiązują.

3. Jeśli kontrolujesz jedno konto i możliwa jest zmiana hasła, spróbuj zmienić hasło na różne słabe wartości.

UWAGA: Jeśli zasady jakości haseł są egzekwowane tylko za pomocą kontroli po stronie klienta, nie stanowi to samo w sobie problemu z bezpieczeństwem, ponieważ zwykli użytkownicy nadal będą chronieni. Zwykle nie jest zagrożeniem dla bezpieczeństwa aplikacji, że przebiegły atakujący może przypisać sobie słabe hasło.

Brute-Forcowe logowanie

Funkcja logowania stanowi otwarte zaproszenie dla osoby atakującej do próby odgadnięcia nazw użytkowników i haseł, a tym samym uzyskania nieautoryzowanego dostępu do aplikacji. Jeśli aplikacja pozwala atakującemu na wielokrotne próby logowania z różnymi hasłami, dopóki nie odgadnie właściwego, jest bardzo podatna na ataki nawet dla atakującego-amatora, który ręcznie wprowadzi niektóre popularne nazwy użytkowników i hasła do swojej przeglądarki. Niedawne włamania do znanych witryn zapewniły dostęp do setek tysięcy rzeczywistych haseł, które były przechowywane w postaci zwykłego tekstu lub przy użyciu skrótów wymuszonych metodą brute-force. Oto najpopularniejsze hasła w świecie rzeczywistym:

* hasło

* Nazwa strony

* 12345678

* qwerty

* abc123

* 111111

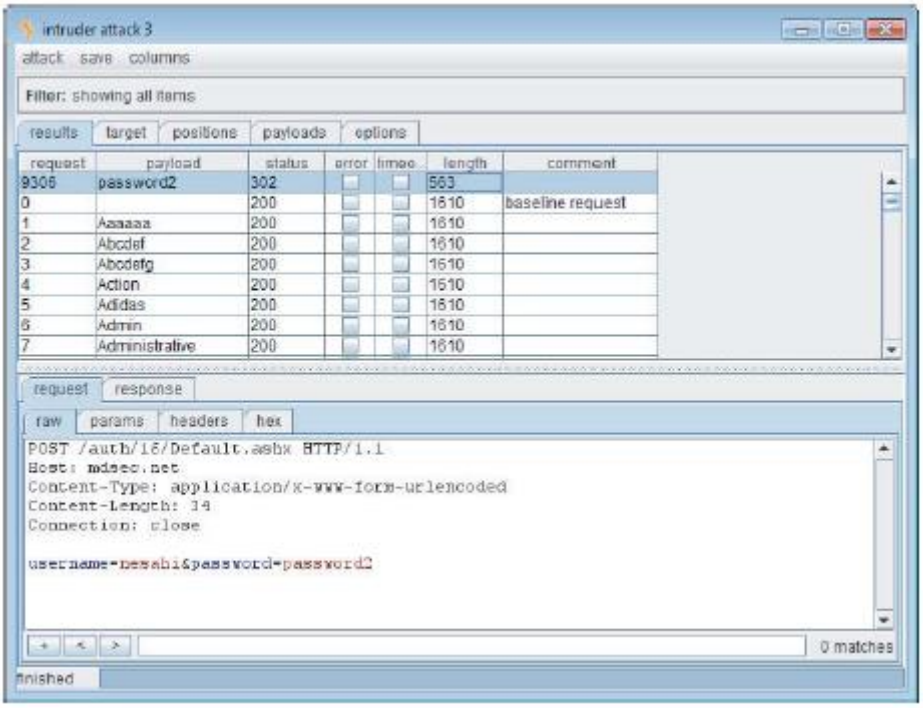
* małpa

* 12345

* wpuść mnie

UWAGA: Hasła administracyjne mogą być w rzeczywistości słabsze, niż pozwala na to polityka haseł. Mogły zostać ustawione przed wejściem w życie polityki lub mogły zostać skonfigurowane za pomocą innej aplikacji lub interfejsu.

W takiej sytuacji każdy poważny atakujący użyje zautomatyzowanych technik próby odgadnięcia hasła na podstawie długich list typowych wartości. Biorąc pod uwagę dzisiejszą przepustowość i możliwości przetwarzania, możliwe jest wykonanie tysięcy prób logowania na minutę ze standardowego komputera i połączenia DSL. W tym scenariuszu nawet najbardziej niezawodne hasła zostaną ostatecznie złamane. Różne techniki i narzędzia umożliwiające wykorzystanie automatyzacji w ten sposób opisano szczegółowo w Części 14. Rysunek przedstawia udany atak polegający na odgadywaniu hasła na pojedyncze konto przy użyciu Burp Intruder.



Pomyślną próbę logowania można wyraźnie rozpoznać po różnicy w kodzie odpowiedzi HTTP, długości odpowiedzi oraz braku komunikatu „nieprawidłowy login”. W niektórych aplikacjach kontrole po stronie klienta są stosowane w celu zapobiegania atakom polegającym na odgadywaniu hasła. Na przykład aplikacja może ustawić plik cookie, taki jak nieudane logowanie=1 i zwiększać go po każdej nieudanej próbie. Po osiągnięciu określonego progu serwer wykrywa to w przesłanym pliku cookie i odmawia przetworzenia próby logowania. Ten rodzaj obrony po stronie klienta może uniemożliwić przeprowadzenie ręcznego ataku tylko przy użyciu przeglądarki, ale można go oczywiście łatwo obejść, jak opisano w Części 5. Odmiana powyższej luki występuje, gdy licznik nieudanych logowań jest utrzymywany w bieżącej sesji. Chociaż po stronie klienta może to nie wskazywać, wystarczy, że osoba

atakująca uzyska nową sesję (na przykład wstrzymując plik cookie sesji) i będzie mogła kontynuować atak polegający na odgadywaniu hasła. Wreszcie, w niektórych przypadkach aplikacja blokuje docelowe konto po odpowiedniej liczbie nieudanych logowań. Jednak odpowiada na dodatkowe próby logowania komunikatami, które wskazują (lub pozwalają atakującemu wywnioskować), czy podane hasło było poprawne. Oznacza to, że osoba atakująca może przeprowadzić atak polegający na odgadywaniu hasła, nawet jeśli konto docelowe jest zablokowane. Jeśli aplikacja automatycznie odblokowuje konta po pewnym czasie, atakujący musi po prostu poczekać, aż to nastąpi, a następnie zalogować się jak zwykle za pomocą odkrytego hasła.

KROKI HACKOWANIA

1. Ręcznie wykonaj kilka nieudanych prób logowania do konta, które kontrolujesz, monitorując otrzymywane komunikaty o błędach.

2. Jeśli po około 10 nieudanych logowaniach aplikacja nie zwróciła komunikatu o zablokowaniu konta, spróbuj zalogować się poprawnie. Jeśli to się powiedzie, prawdopodobnie nie ma zasady blokowania konta.

3. Jeśli konto jest zablokowane, spróbuj powtórzyć ćwiczenie, używając innego konta. Tym razem, jeśli aplikacja wygeneruje jakiekolwiek pliki cookie, użyj każdego pliku cookie tylko do jednej próby logowania i uzyskaj nowy plik cookie przy każdej kolejnej próbie logowania.

4. Ponadto, jeśli konto jest zablokowane, sprawdź, czy podanie prawidłowego hasła powoduje jakąkolwiek różnicę w zachowaniu aplikacji w porównaniu z błędnym hasłem. Jeśli tak, możesz kontynuować atak polegający na odgadywaniu hasła, nawet jeśli konto jest zablokowane.

5. Jeśli nie kontrolujesz żadnych kont, spróbuj wyliczyć prawidłową nazwę użytkownika (patrz następna sekcja) i wykonaj przy jej użyciu kilka błędnych logowań. Monitoruj komunikaty o błędach dotyczące blokady konta.

6. Aby przeprowadzić atak brute-force, najpierw zidentyfikuj różnicę w zachowaniu aplikacji w odpowiedzi na udane i nieudane logowanie. Możesz wykorzystać ten fakt do rozróżnienia między sukcesem a porażką w trakcie automatycznego ataku.

7. Uzyskaj listę wyliczonych lub wspólnych nazw użytkowników oraz listę wspólnych haseł. Użyj wszelkich uzyskanych informacji o regułach jakości haseł, aby dostosować listę haseł, aby uniknąć zbędnych przypadków testowych.

8. Użyj odpowiedniego narzędzia lub niestandardowego skryptu, aby szybko wygenerować żądania logowania przy użyciu wszystkich permutacji tych nazw użytkowników i haseł. Monitoruj odpowiedzi serwera, aby zidentyfikować udane próby logowania. W części 14 opisano szczegółowo różne techniki i narzędzia do przeprowadzania niestandardowych ataków z wykorzystaniem automatyzacji.

9. Jeśli atakujesz jednocześnie kilka nazw użytkowników, zwykle lepiej jest przeprowadzić tego rodzaju atak brute-force w sposób wszerz, a nie w głąb. Wiąże się to z iteracją listy haseł (zaczynając od najczęstszego) i próbowaniem każdego hasła po kolei dla każdej nazwy użytkownika. Takie podejście ma dwie zalety. Po pierwsze, szybciej odkrywasz konta ze wspólnymi hasłami. Po drugie, jest mniej prawdopodobne, że uruchomisz jakąkolwiek obronę przed blokadą konta, ponieważ istnieje opóźnienie między kolejnymi próbami użycia każdego indywidualnego konta.

Pełne komunikaty o błędach

Typowy formularz logowania wymaga podania przez użytkownika dwóch informacji - nazwy użytkownika i hasła. Niektóre aplikacje wymagają kilku dodatkowych informacji, takich jak data urodzenia, niezapomniane miejsce lub kod PIN. Gdy próba logowania nie powiedzie się, można oczywiście wywnioskować, że przynajmniej jedna informacja była błędna. Jeśli jednak aplikacja poinformuje Cię, która informacja była nieprawidłowa, możesz wykorzystać to zachowanie do znacznego zmniejszenia skuteczności mechanizmu logowania. W najprostszym przypadku, gdy logowanie wymaga nazwy użytkownika i hasła, aplikacja może zareagować na nieudaną próbę logowania, wskazując, czy przyczyną niepowodzenia była nierozpoznana nazwa użytkownika, czy też nieprawidłowe hasło, jak pokazano na rysunku.

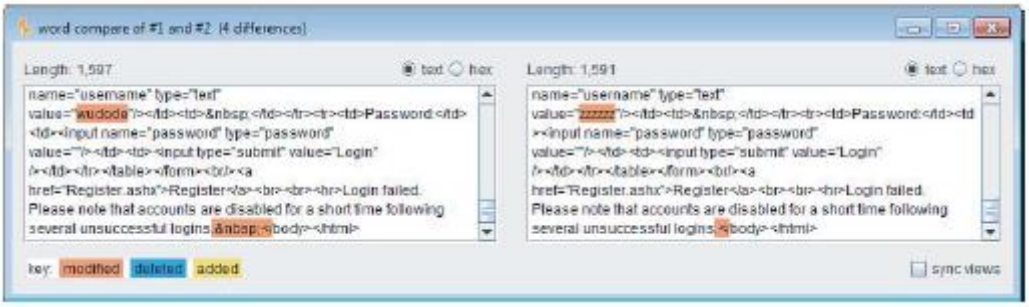
W tym przypadku możesz użyć zautomatyzowanego ataku, aby przejrzeć dużą listę popularnych nazw użytkowników, aby wyliczyć, które z nich są prawidłowe. Oczywiście nazwy użytkowników zwykle nie są uważane za tajemnicę (na przykład nie są maskowane podczas logowania). Jednak zapewnienie atakującemu łatwego sposobu identyfikacji prawidłowych nazw użytkowników zwiększa prawdopodobieństwo, że złamie zabezpieczenia aplikacji, mając wystarczająco dużo czasu, umiejętności i wysiłku. Lista wyliczonych nazw użytkowników może być wykorzystana jako podstawa do różnych kolejnych ataków, w tym zgadywania haseł, ataków na dane lub sesje użytkowników lub socjotechniki.

Oprócz podstawowej funkcji logowania, wyliczanie nazwy użytkownika może pojawić się w innych elementach mechanizmu uwierzytelniania. W zasadzie każda funkcja, w której przesyłana jest rzeczywista lub potencjalna nazwa użytkownika, może być wykorzystana do tego celu. Jednym z miejsc, w których często występuje wyliczanie nazw użytkowników, jest funkcja rejestracji użytkowników. Jeśli aplikacja umożliwia nowym użytkownikom rejestrację i określanie własnych nazw użytkowników, wyliczenie nazwy użytkownika jest praktycznie niemożliwe aby zapobiec, jeśli aplikacja ma zapobiegać rejestracji zduplikowanych nazw użytkowników. Inne miejsca, w których czasami można znaleźć wyliczenie nazwy użytkownika, to funkcje zmiany hasła i zapomnianego hasła, jak opisano w dalszej części tej części.

UWAGA: Wiele mechanizmów uwierzytelniania ujawnia nazwy użytkowników w sposób dorozumiany lub jawny. Na koncie poczty internetowej nazwą użytkownika jest często adres e-mail, który z założenia jest powszechnie znany. Wiele innych witryn ujawnia nazwy użytkowników w aplikacji, nie biorąc pod uwagę korzyści, jakie daje to atakującemu, lub generuje nazwy użytkowników w sposób, który można przewidzieć (na przykład użytkownik1842, użytkownik1843 itd.).

W bardziej złożonych mechanizmach logowania, gdzie aplikacja wymaga od użytkownika podania kilku informacji lub przejścia przez kilka etapów, obszerne komunikaty o błędach lub inne elementy dyskryminujące mogą umożliwić atakującemu ukierunkowanie kolejno każdego etapu procesu logowania, zwiększając prawdopodobieństwo, że uzyska nieautoryzowany dostęp.

UWAGA: Ta luka w zabezpieczeniach może powstać w bardziej subtelny sposób niż pokazano tutaj. Nawet jeśli komunikaty o błędach zwracane w odpowiedzi na prawidłową i nieprawidłową nazwę użytkownika są powierzchownie podobne, mogą istnieć między nimi niewielkie różnice, które można wykorzystać do wyliczenia prawidłowych nazw użytkowników. Na przykład, jeśli wiele ścieżek kodu w aplikacji zwraca „ten sam" komunikat o błędzie, mogą występować drobne różnice typograficzne między każdym wystąpieniem komunikatu. W niektórych przypadkach odpowiedzi aplikacji mogą być identyczne na ekranie, ale mogą zawierać subtelne różnice ukryte w źródle HTML, takie jak komentarze lub różnice w układzie. Jeśli nie ma oczywistego sposobu wyliczania nazw użytkowników, należy przeprowadzić dokładne porównanie odpowiedzi aplikacji na prawidłowe i nieprawidłowe nazwy użytkownika. Możesz użyć narzędzia Comparer w pakiecie Burp Suite, aby automatycznie przeanalizować i wyróżnić różnice między odpowiedziami dwóch aplikacji, jak pokazano na rysunku. Pomaga to szybko określić, czy ważność nazwy użytkownika powoduje jakąkolwiek systematyczną różnicę w odpowiedziach aplikacji.



KROKI HACKOWANIA

1. Jeśli znasz już jedną prawidłową nazwę użytkownika (na przykład konta, które kontrolujesz), prześlij jedną nazwę logowania, używając tej nazwy użytkownika i nieprawidłowego hasła, oraz inną nazwę logowania, używając losowej nazwy użytkownika.

2. Rejestruj każdy szczegół odpowiedzi serwera na każdą próbę logowania, w tym kod statusu, wszelkie przekierowania, informacje wyświetlane na ekranie i wszelkie różnice ukryte w źródle strony HTML. Użyj przechwytującego serwera proxy, aby zachować pełną historię całego ruchu do iz serwera.

3. Spróbuj odkryć oczywiste lub subtelne różnice w odpowiedziach serwera na dwie próby logowania.

4. Jeśli to się nie powiedzie, powtórz ćwiczenie we wszystkich miejscach aplikacji, w których można wprowadzić nazwę użytkownika (na przykład samodzielna rejestracja, zmiana hasła i zapomniane hasło).

5. Jeśli zostanie wykryta różnica w odpowiedziach serwera na prawidłowe i nieprawidłowe nazwy użytkowników, uzyskaj listę popularnych nazw użytkowników. Użyj niestandardowego skryptu lub zautomatyzowanego narzędzia, aby szybko przesłać każdą nazwę użytkownika i odfiltrować odpowiedzi, które oznaczają, że nazwa użytkownika jest prawidłowa.

6. Przed przystąpieniem do wyliczania sprawdź, czy po określonej liczbie nieudanych prób logowania aplikacja blokuje konto (patrz poprzedni rozdział). Jeśli tak, pożądane jest zaprojektowanie ataku wyliczającego z uwzględnieniem tego faktu. Na przykład, jeśli aplikacja przyzna ci tylko trzy nieudane próby logowania na dane konto, ryzykujesz „zmarnowaniem" jednej z nich na każdą nazwę użytkownika odkrytą przez automatyczne wyliczanie. Dlatego podczas przeprowadzania ataku polegającego na wyliczaniu nie należy podawać naciąganego hasła przy każdej próbie logowania. Zamiast tego podaj pojedyncze wspólne hasło, takie jak hasło1 lub samą nazwę użytkownika jako hasło.

Jeśli reguły dotyczące jakości haseł są słabe, jest wysoce prawdopodobne, że niektóre próby logowania, które wykonasz w ramach ćwiczenia wyliczania, zakończą się sukcesem i ujawnią zarówno nazwę użytkownika, jak i hasło w jednym trafieniu. Aby ustawić pole hasła tak, aby było takie samo jak nazwa użytkownika, możesz użyć trybu ataku „taran" w Burp Intruder, aby wstawić ten sam ładunek w wielu pozycjach w żądaniu logowania.

Nawet jeśli reakcje aplikacji na próby logowania zawierające prawidłowe i nieprawidłowe nazwy użytkownika są identyczne pod każdym względem, nadal może istnieć możliwość wyliczenia nazw użytkowników na podstawie czasu potrzebnego aplikacji na odpowiedź na żądanie logowania. Aplikacje często wykonują bardzo różne przetwarzanie zaplecza na żądanie logowania, w zależności od tego, czy zawiera ono prawidłową nazwę użytkownika. Na przykład po przesłaniu prawidłowej nazwy użytkownika aplikacja może pobrać dane użytkownika z wewnętrznej bazy danych, wykonać różne operacje na tych danych (na przykład sprawdzić, czy konto wygasło), a następnie zweryfikować hasło (co może obejmować algorytm mieszania intensywnie korzystający z zasobów) przed zwróceniem ogólnej wiadomości, jeśli hasło jest nieprawidłowe. Różnica w czasie między dwiema reakcjami może być zbyt subtelna, aby można ją było wykryć podczas pracy tylko z przeglądarką, ale zautomatyzowane narzędzie może być w stanie je rozróżnić. Nawet jeśli wyniki takiego ćwiczenia zawierają duży odsetek fałszywych alarmów, nadal lepiej jest mieć listę 100 nazw użytkowników, z których około 50% jest prawidłowych, niż listę 10 000 nazw użytkowników, z których około 0,5% jest prawidłowych . Zobacz rozdział 15, aby uzyskać szczegółowe wyjaśnienie, w jaki sposób wykryć i wykorzystać ten typ różnicy czasu w celu wydobycia informacji z aplikacji.

WSKAZÓWKA: Oprócz samej funkcji logowania mogą istnieć inne źródła informacji, w których można uzyskać prawidłowe nazwy użytkownika. Przejrzyj wszystkie komentarze do kodu źródłowego wykryte podczas mapowania aplikacji, aby zidentyfikować widoczne nazwy użytkowników. Wszelkie adresy e-mail programistów lub innego personelu w organizacji mogą być prawidłowymi nazwami użytkowników, w całości lub tylko z prefiksem charakterystycznym dla użytkownika. Każda dostępna funkcja rejestrowania może ujawnić nazwy użytkowników.

Wrażliwa transmisja poświadczeń

Jeśli aplikacja używa nieszyfrowanego połączenia HTTP do przesyłania danych logowania, podsłuchujący, który jest odpowiednio umieszczony w sieci, może oczywiście je przechwycić. W zależności od lokalizacji użytkownika potencjalni podsłuchujący mogą przebywać:

* W sieci lokalnej użytkownika

* W dziale IT użytkownika

* W ramach usługodawcy internetowego użytkownika

* W sieci szkieletowej Internetu

* W ramach usługodawcy internetowego obsługującego aplikację

* W dziale IT zarządzającym aplikacją

UWAGA: Dowolna z tych lokalizacji może być zajęta przez upoważniony personel, ale potencjalnie także przez osobę atakującą z zewnątrz, która w inny sposób naruszyła odpowiednią infrastrukturę. Nawet jeśli uważa się, że pośrednicy w danej sieci są zaufani, bezpieczniej jest używać bezpiecznych mechanizmów transportowych podczas przesyłania przez nią wrażliwych danych.

Nawet jeśli logowanie odbywa się przez HTTPS, dane uwierzytelniające mogą zostać ujawnione nieupoważnionym stronom, jeśli aplikacja obsługuje je w niebezpieczny sposób:

* Jeśli poświadczenia są przesyłane jako parametry ciągu zapytania, a nie w treści żądania POST, mogą one być rejestrowane w różnych miejscach, na przykład w historii przeglądarki użytkownika, w dziennikach serwera WWW oraz w dziennikach wszelkie odwrotne serwery proxy stosowane w infrastrukturze hostingowej. Jeśli atakującemu uda się naruszyć którykolwiek z tych zasobów, może zwiększyć uprawnienia, przechwytując przechowywane tam poświadczenia użytkownika.

* Chociaż większość aplikacji internetowych korzysta z treści żądania POST w celu przesłania samego formularza logowania HTML, zaskakująco często zdarza się, że żądanie logowania jest obsługiwane przez przekierowanie do innego adresu URL z tymi samymi danymi uwierzytelniającymi przekazanymi jako parametry ciągu zapytania. Nie jest jasne, dlaczego twórcy aplikacji uważają za konieczne wykonywanie tych odrzuceń, ale jeśli zdecydowali się to zrobić, łatwiej jest je zaimplementować jako przekierowania 302 do adresu URL niż jako żądania POST przy użyciu drugiego formularza HTML przesłanego za pośrednictwem JavaScript.

* Aplikacje internetowe czasami przechowują dane uwierzytelniające użytkownika w plikach cookie, zwykle w celu wdrożenia źle zaprojektowanych mechanizmów logowania, zmiany hasła, „zapamiętaj mnie” i tak dalej. Te dane uwierzytelniające są podatne na przechwycenie w wyniku ataków, które narażają pliki cookie użytkowników, a w przypadku trwałych plików cookie przez każdego, kto uzyska dostęp do lokalnego systemu plików klienta. Nawet jeśli dane uwierzytelniające są zaszyfrowane, osoba atakująca nadal może po prostu odtworzyć plik cookie, a tym samym zalogować się jako użytkownik, nie znając jego danych uwierzytelniających. Części 12 i 13 opisują różne sposoby, w jakie osoba atakująca może atakować innych użytkowników w celu przechwycenia ich plików cookie.

Wiele aplikacji używa HTTP dla nieuwierzytelnionych obszarów aplikacji i przełącza się na HTTPS w momencie logowania. W takim przypadku właściwym miejscem przełączenia na HTTPS jest załadowanie strony logowania do przeglądarki, co umożliwia użytkownikowi sprawdzenie autentyczności strony przed wprowadzeniem poświadczeń. Jednak często spotyka się aplikacje, które ładują samą stronę logowania za pomocą protokołu HTTP, a następnie przełączają się na HTTPS w momencie przesyłania poświadczeń. Jest to niebezpieczne, ponieważ użytkownik nie może zweryfikować autentyczności samej strony logowania, a zatem nie ma pewności, że poświadczenia zostaną przesłane w bezpieczny sposób. Odpowiednio ustawiony atakujący może przechwycić i zmodyfikować stronę logowania, zmieniając docelowy adres URL formularza logowania, aby korzystał z protokołu HTTP. Zanim bystry użytkownik zorientuje się, że dane uwierzytelniające zostały przesłane za pomocą protokołu HTTP, będą one już przejęte.

KROKI HACKOWANIA

1. Przeprowadź pomyślne logowanie, monitorując cały ruch w obu kierunkach między klientem a serwerem.

2. Zidentyfikuj każdy przypadek, w którym dane uwierzytelniające są przesyłane w dowolnym kierunku. Możesz ustawić reguły przechwytywania w przechwytującym serwerze proxy, aby oflagować wiadomości zawierające określone ciągi.

3. Jeśli zostaną znalezione przypadki, w których poświadczenia są przesyłane w ciągu zapytania adresu URL lub jako plik cookie albo są przesyłane z powrotem z serwera do klienta, należy zrozumieć, co się dzieje i spróbować ustalić, w jakim celu twórcy aplikacji próbowali osiągnąć. Spróbuj znaleźć wszelkie

sposoby, za pomocą których osoba atakująca może ingerować w logikę aplikacji, aby naruszyć dane uwierzytelniające innych użytkowników.

4. Jeśli jakakolwiek poufna informacja jest przesyłana niezaszyfrowanym kanałem, jest ona oczywiście narażona na przechwycenie.

5. Jeśli nie zostaną zidentyfikowane żadne przypadki faktycznego przesyłania danych uwierzytelniających w sposób niezabezpieczony, zwróć szczególną uwagę na wszelkie dane, które wydają się być zaszyfrowane lub zaciemnione. Jeśli obejmuje to wrażliwe dane, możliwe może być wykonanie inżynierii wstecznej algorytmu zaciemniania.

6. Jeśli poświadczenia są przesyłane przy użyciu protokołu HTTPS, ale formularz logowania jest ładowany przy użyciu protokołu HTTP, aplikacja jest narażona na atak typu man-in-the-middle, który może zostać wykorzystany do przechwycenia poświadczeń.

Funkcjonalność zmiany hasła

Co zaskakujące, wiele aplikacji internetowych nie umożliwia użytkownikom zmiany hasła. Funkcjonalność ta jest jednak niezbędna dla dobrze zaprojektowanego mechanizmu uwierzytelniania z dwóch powodów:

* Okresowa wymuszona zmiana hasła zmniejsza ryzyko naruszenia hasła. Zmniejsza okno, w którym dane hasło może być celem ataku polegającego na zgadywaniu. Zmniejsza również okno, w którym złamane hasło może zostać użyte bez wykrycia przez atakującego.

* Użytkownicy, którzy podejrzewają, że ich hasła mogły zostać naruszone, muszą mieć możliwość szybkiej zmiany hasła, aby zmniejszyć zagrożenie nieautoryzowanym użyciem.

Funkcjonalność zmiany hasła

Co zaskakujące, wiele aplikacji internetowych nie umożliwia użytkownikom zmiany hasła. Funkcjonalność ta jest jednak niezbędna dla dobrze zaprojektowanego mechanizmu uwierzytelniania z dwóch powodów:

* Okresowa wymuszona zmiana hasła zmniejsza ryzyko naruszenia hasła. Zmniejsza okno, w którym dane hasło może być celem ataku polegającego na zgadywaniu. Zmniejsza również okno, w którym złamane hasło może zostać użyte bez wykrycia przez atakującego.

* Użytkownicy, którzy podejrzewają, że ich hasła mogły zostać naruszone, muszą mieć możliwość szybkiej zmiany hasła, aby zmniejszyć zagrożenie nieautoryzowanym użyciem.

Chociaż jest to niezbędna część skutecznego mechanizmu uwierzytelniania, funkcja zmiany hasła jest często podatna na ataki z założenia. Luki w zabezpieczeniach, których celowo unika się w głównej funkcji logowania, często pojawiają się ponownie w funkcji zmiany hasła. Wiele funkcji zmiany hasła aplikacji internetowych jest dostępnych bez uwierzytelniania i wykonuje następujące czynności:

* Podaj szczegółowy komunikat o błędzie wskazujący, czy żądana nazwa użytkownika jest prawidłowa.

* Zezwalaj na nieograniczone odgadywanie pola „istniejące hasło”.

* Sprawdź, czy pola „nowe hasło” i „potwierdź nowe hasło” mają taką samą wartość dopiero po sprawdzeniu poprawności istniejącego hasła, co pozwala atakowi na bezinwazyjne odkrycie istniejącego hasła.

Typowa funkcja zmiany hasła obejmuje stosunkowo duże logiczne drzewo decyzyjne. Aplikacja musi identyfikować użytkownika, sprawdzać poprawność dostarczonego istniejącego hasła, integrować się z wszelkimi zabezpieczeniami przed blokadą konta, porównywać dostarczone nowe hasła ze sobą iz regułami jakości haseł oraz w odpowiedni sposób przekazywać użytkownikowi informację zwrotną o błędach. Z tego powodu funkcje zmiany hasła często zawierają subtelne błędy logiczne, które można wykorzystać do obalenia całego mechanizmu.

KROKI HACKOWANIA

1. Zidentyfikuj wszelkie funkcje zmiany hasła w aplikacji. Jeśli nie jest to wyraźnie powiązane z opublikowanymi treściami, nadal może zostać zaimplementowane. Część 4 opisuje różne techniki wykrywania ukrytych treści w aplikacji.

2. Wysyłaj różne żądania do funkcji zmiany hasła, używając nieprawidłowych nazw użytkowników, nieprawidłowych istniejących haseł i niedopasowanych wartości „nowe hasło” i „potwierdź nowe hasło”.

3. Spróbuj zidentyfikować wszelkie zachowania, które mogą być wykorzystane do wyliczania nazw użytkowników lub ataków siłowych (zgodnie z opisem w sekcjach „Brutalne logowanie” i „Rozszerzone komunikaty o błędach”).

WSKAZÓWKA: Jeśli formularz zmiany hasła jest dostępny tylko dla uwierzytelnionych użytkowników i nie zawiera pola nazwy użytkownika, podanie dowolnej nazwy użytkownika może być nadal możliwe. Formularz może przechowywać nazwę użytkownika w ukrytym polu, które można łatwo modyfikować. Jeśli nie, spróbuj podać dodatkowy parametr zawierający nazwę użytkownika, używając tej samej nazwy parametru, która jest używana w głównym formularzu logowania. Ta sztuczka czasami udaje się zastąpić nazwę użytkownika bieżącego użytkownika, umożliwiając brutalne wymuszenie poświadczeń innych użytkowników, nawet jeśli nie jest to możliwe przy głównym logowaniu.

Funkcja zapomnianego hasła

Podobnie jak funkcja zmiany hasła, mechanizmy przywracania po zapomnianym haśle często powodują problemy, których można było uniknąć w głównej funkcji logowania, takie jak wyliczanie nazwy użytkownika. Oprócz tego zakresu defektów, słabości projektowe funkcji związanych z zapomnianym hasłem często sprawiają, że jest to najsłabsze ogniwo do ataku na ogólną logikę uwierzytelniania aplikacji. Często można znaleźć kilka rodzajów słabości projektowych:

* Funkcja zapomnianego hasła często polega na przedstawieniu użytkownikowi dodatkowego wyzwania zamiast głównego loginu, jak pokazano na rysunku .

Odpowiedź na to wyzwanie jest często znacznie łatwiejsza dla osoby atakującej niż próba odgadnięcia hasła użytkownika. Pytania o nazwiska panieńskie matek, niezapomniane daty, ulubione kolory i tym podobne generalnie będą miały znacznie mniejszy zestaw potencjalnych odpowiedzi niż zestaw możliwych haseł. Ponadto często dotyczą one informacji, które są publicznie znane lub które zdeterminowany atakujący może odkryć przy niewielkim wysiłku.

W wielu przypadkach aplikacja umożliwia użytkownikom ustawienie własnego wyzwania odzyskiwania hasła i odpowiedzi podczas rejestracji. Użytkownicy mają skłonność do stawiania wyjątkowo niepewnych wyzwań, prawdopodobnie w fałszywym założeniu, że tylko oni zostaną im postawieni. Przykładem jest „Czy mam łódź?" W tej sytuacji atakujący, który chce uzyskać dostęp, może użyć zautomatyzowanego ataku, aby przejrzeć listę wyliczonych lub typowych nazw użytkowników, zarejestrować wszystkie wyzwania związane z odzyskiwaniem hasła i wybrać te, które wydają się najłatwiejsze do odgadnięcia.

* Podobnie jak w przypadku funkcji zmiany hasła, twórcy aplikacji często pomijają możliwość brutalnego wymuszenia odpowiedzi na wezwanie do odzyskania hasła, nawet jeśli blokują ten atak na głównej stronie logowania. Jeśli aplikacja umożliwia nieograniczone próby odpowiedzi na wyzwania związane z odzyskiwaniem hasła, istnieje duże prawdopodobieństwo, że zostanie naruszona przez zdeterminowanego atakującego.

* W niektórych aplikacjach wezwanie do odzyskania jest zastępowane prostą „wskazówką" dotyczącą hasła konfigurowaną przez użytkowników podczas rejestracji. Użytkownicy często ustawiają niezwykle oczywiste wskazówki, być może nawet identyczne z samym hasłem, w fałszywym założeniu, że tylko oni je kiedykolwiek zobaczą. Ponownie atakujący z listą popularnych lub wyliczonych nazw użytkowników może łatwo przechwycić dużą liczbę wskazówek do hasła, a następnie zacząć zgadywać.

* Mechanizm, za pomocą którego aplikacja umożliwia użytkownikom odzyskanie kontroli nad kontem po prawidłowej odpowiedzi na wyzwanie, jest często podatny na ataki. Jednym z dość bezpiecznych sposobów wdrożenia tego jest wysłanie unikalnego, niemożliwego do odgadnięcia, ograniczonego czasowo adresu URL odzyskiwania na adres e-mail podany przez użytkownika podczas rejestracji. Odwiedzenie tego adresu URL w ciągu kilku minut umożliwia użytkownikowi ustawienie nowego hasła. Jednak często spotykane są inne mechanizmy odzyskiwania konta, które z założenia są niepewne:

* Niektóre aplikacje ujawniają serwerowi istniejące, zapomniane hasło po pomyślnym zakończeniu wyzwania, umożliwiając atakującemu korzystanie z konta w nieskończoność bez ryzyka wykrycia przez właściciela. Nawet jeśli właściciel konta następnie zmieni zdmuchnięte hasło, osoba atakująca może po prostu powtórzyć to samo wyzwanie, aby uzyskać nowe hasło.

* Niektóre aplikacje natychmiast przełączają użytkownika w uwierzytelnioną sesję po pomyślnym zakończeniu wyzwania, ponownie umożliwiając atakującemu korzystanie z konta przez czas nieokreślony bez wykrycia i bez konieczności znajomości hasła użytkownika.

* Niektóre aplikacje wykorzystują mechanizm wysyłania unikalnego adresu URL odzyskiwania, ale wysyłają go na adres e-mail określony przez użytkownika w momencie zakończenia wyzwania. Nie zapewnia to absolutnie żadnego zwiększonego bezpieczeństwa procesu odzyskiwania poza ewentualnym rejestrowaniem adresu e-mail użytego przez atakującego.

WSKAZÓWKA: Nawet jeśli aplikacja nie udostępnia pola ekranowego, w którym można podać adres e-mail, aby otrzymać adres URL odzyskiwania, aplikacja może przesłać adres za pośrednictwem ukrytego pola formularza lub pliku cookie. Daje to podwójną szansę: możesz odkryć adres e-mail użytkownika,

którego naruszyłeś, i możesz zmodyfikować jego wartość, aby otrzymać adres URL odzyskiwania na wybrany przez siebie adres.

* Niektóre aplikacje pozwalają użytkownikom zresetować wartość hasła bezpośrednio po pomyślnym ukończeniu wyzwania i nie wysyłają użytkownikowi żadnego powiadomienia e-mail. Oznacza to, że włamanie się do konta przez atakującego nie zostanie zauważone, dopóki właściciel nie spróbuje ponownie się zalogować. Może nawet pozostać niezauważone, jeśli właścicielka uzna, że zapomniała hasła i w ten sam sposób zresetuje je. Osoba atakująca, która po prostu chce uzyskać dostęp do aplikacji, może następnie przejąć kontrolę nad kontem innego użytkownika na pewien czas i w ten sposób może nadal korzystać z aplikacji w nieskończoność.

KROKI HACKOWANIA

1. Zidentyfikuj wszelkie funkcje zapomnianego hasła w aplikacji. Jeśli nie jest to wyraźnie powiązane z opublikowanymi treściami, nadal może zostać zaimplementowane.

2. Dowiedz się, jak działa funkcja zapomnianego hasła, wykonując pełną instrukcję przy użyciu konta, które kontrolujesz.

3. Jeśli mechanizm wykorzystuje wyzwanie, określ, czy użytkownicy mogą ustawiać lub wybierać własne wyzwanie i odpowiedź. Jeśli tak, użyj listy wyliczonych lub popularnych nazw użytkowników, aby zebrać listę wyzwań i przejrzyj ją pod kątem tych, które wydają się łatwe do odgadnięcia.

4. Jeśli mechanizm używa „podpowiedzi” do hasła, wykonaj to samo ćwiczenie, aby zebrać listę wskazówek do hasła i wybrać te, które są łatwe do odgadnięcia.

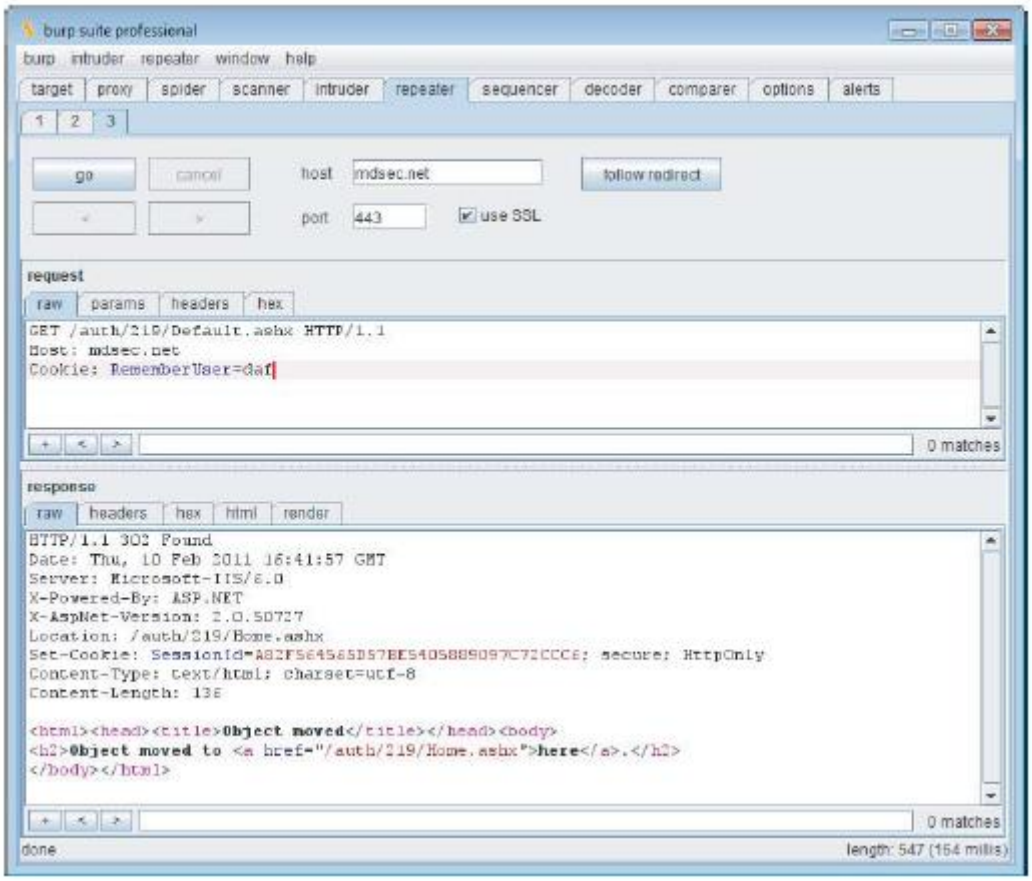
5. Spróbuj zidentyfikować jakiekolwiek zachowanie w mechanizmie zapomnianego hasła, które może zostać wykorzystane jako podstawa do wyliczania nazw użytkowników lub ataków typu brute-force (patrz poprzednie szczegóły).

6. Jeśli aplikacja wygeneruje wiadomość e-mail zawierającą adres URL odzyskiwania w odpowiedzi na żądanie zapomnienia hasła, uzyskaj liczbę tych adresów URL i spróbuj zidentyfikować wzorce, które mogą umożliwić przewidywanie adresów URL przydzielanych innym użytkownikom. Zastosuj te same techniki, które są odpowiednie do analizy tokenów sesji pod kątem przewidywalności

Funkcjonalność „Zapamiętaj mnie”.

Aplikacje często implementują funkcje „zapamiętaj mnie” dla wygody użytkowników. Dzięki temu użytkownicy nie muszą ponownie wpisywać nazwy użytkownika i hasła za każdym razem, gdy korzystają z aplikacji z określonego komputera. Funkcje te są często niezabezpieczone z założenia i narażają użytkownika na ataki zarówno lokalne, jak i ze strony użytkowników na innych komputerach:

* Niektóre funkcje „zapamiętaj mnie” są realizowane przy użyciu prostego trwałego pliku cookie, takiego jak RememberUser=daf .



Kiedy ten plik cookie jest przesyłany na początkową stronę aplikacji, aplikacja ufa plikowi cookie w celu uwierzytelnienia użytkownika i tworzy sesję aplikacji dla tej osoby, pomijając login. Osoba atakująca może użyć listy popularnych lub wyliczonych nazw użytkowników, aby uzyskać pełny dostęp do aplikacji bez uwierzytelniania.

* Niektóre funkcje „zapamiętaj mnie” ustawiają plik cookie, który zawiera nie nazwę użytkownika, ale rodzaj trwałego identyfikatora sesji, np. RememberUser=1328. Gdy identyfikator zostanie przesłany na stronę logowania, aplikacja wyszukuje powiązanego z nim użytkownika i tworzy dla niego sesję aplikacji. Podobnie jak w przypadku zwykłych tokenów sesji, jeśli identyfikatory sesji innych użytkowników można przewidzieć lub ekstrapolować, osoba atakująca może iterować przez dużą liczbę potencjalnych identyfikatorów, aby znaleźć identyfikatory powiązane z użytkownikami aplikacji, a tym samym uzyskać dostęp do ich kont bez uwierzytelniania. Zobacz rozdział 7, aby zapoznać się z technikami wykonywania tego ataku.

* Nawet jeśli informacje przechowywane w celu ponownej identyfikacji użytkowników są odpowiednio chronione (zaszyfrowane), aby uniemożliwić innym użytkownikom ich określenie lub odgadnięcie, nadal mogą być narażone na przechwycenie w wyniku błędu, takiego jak skrypty międzywitrynowe, lub atakującego, który ma lokalny dostęp do komputera użytkownika.

KROKI HACKOWANIA

1. Aktywuj dowolną funkcję „zapamiętaj mnie” i ustal, czy funkcja rzeczywiście „zapamiętuje” użytkownika w pełni, czy też pamięta tylko jego nazwę użytkownika i nadal wymaga podania hasła przy kolejnych wizytach. W tym drugim przypadku jest znacznie mniej prawdopodobne, że funkcjonalność ujawni jakąkolwiek lukę w zabezpieczeniach.

2. Dokładnie sprawdź wszystkie ustawione trwałe pliki cookie, a także wszelkie dane, które są utrwalone w innych lokalnych mechanizmach przechowywania, takich jak userData programu Internet Explorer, izolowany magazyn Silverlight lub lokalne udostępnione obiekty Flash. Poszukaj zapisanych danych, które wyraźnie identyfikują użytkownika lub wydają się zawierać przewidywalny identyfikator użytkownika.

3. Nawet jeśli przechowywane dane wydają się być mocno zakodowane lub zaciemnione, przejrzyj to dokładnie. Porównaj wyniki „zapamiętywania" kilku bardzo podobnych nazw użytkowników i/lub haseł, aby zidentyfikować wszelkie możliwości inżynierii wstecznej oryginalnych danych. W tym miejscu użyj tych samych technik, które opisano w części 7, aby wykryć znaczenie i wzorce w tokenach sesji.

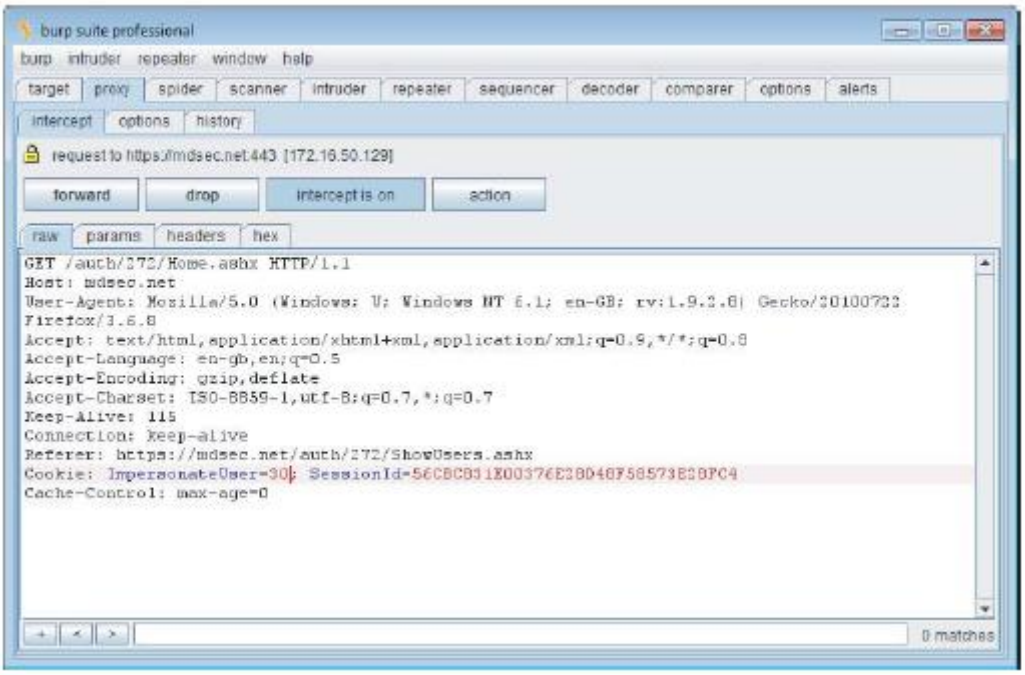
4. Spróbuj zmodyfikować zawartość trwałego pliku cookie, aby spróbować przekonać aplikację, że inny użytkownik zapisał swoje dane na Twoim komputerze.

Funkcjonalność personifikacji użytkownika

Niektóre aplikacje umożliwiają uprzywilejowanemu użytkownikowi aplikacji podszywanie się pod innych użytkowników w celu uzyskania dostępu do danych i wykonywania działań w kontekście użytkownika. Na przykład niektóre aplikacje bankowe umożliwiają operatorom centrum pomocy ustne uwierzytelnienie użytkownika telefonu, a następnie przełączenie sesji aplikacji na kontekst tego użytkownika, aby mu pomóc. W ramach funkcji podszywania się często występują różne wady projektowe:

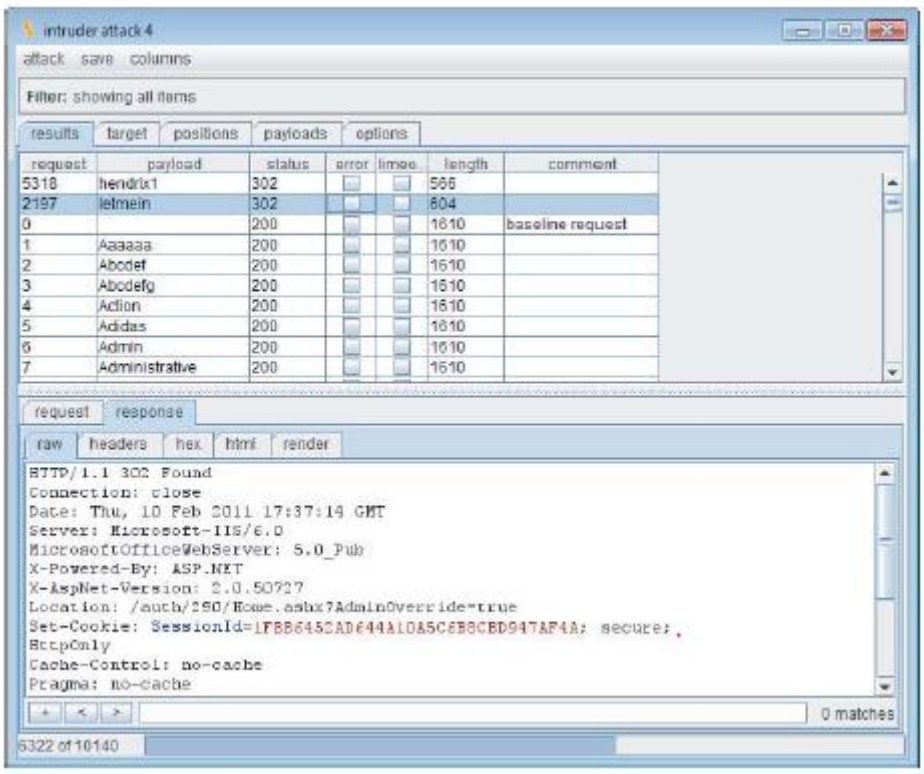
* Może być zaimplementowana jako funkcja „ukryta", która nie podlega odpowiedniej kontroli dostępu. Na przykład każdy, kto zna lub odgadnie adres URL /admin/ImpersonateUser.jsp, może skorzystać z tej funkcji i podszywać się pod dowolnego innego użytkownika.

* Aplikacja może ufać danym kontrolowanym przez użytkownika podczas ustalania, czy użytkownik podszywa się pod inne osoby. Na przykład, oprócz ważnego tokena sesji, użytkownik może przesłać plik cookie określający, z którego konta aktualnie korzysta jego sesja. Osoba atakująca może zmodyfikować tę wartość i uzyskać dostęp do innych kont użytkowników bez uwierzytelniania, jak pokazano na rysunku .



* Jeśli aplikacja umożliwia podszywanie się pod użytkowników administracyjnych, jakakolwiek luka w logice personifikacji może skutkować luką w zabezpieczeniach umożliwiającą pionowe zwiększenie uprawnień. Zamiast po prostu uzyskać dostęp do danych innych zwykłych użytkowników, osoba atakująca może uzyskać pełną kontrolę nad aplikacją.

* Niektóre funkcje personifikacji są zaimplementowane jako proste hasło „backdoor”, które można przesłać na standardową stronę logowania wraz z dowolną nazwą użytkownika w celu uwierzytelnienia jako ten użytkownik. Ten projekt jest bardzo niepewny z wielu powodów, ale największą szansą dla atakujących jest to, że prawdopodobnie odkryją to hasło podczas wykonywania standardowych ataków, takich jak brutalne wymuszanie logowania. Jeśli hasło backdoora zostanie dopasowane przed rzeczywistym hasłem użytkownika, osoba atakująca prawdopodobnie odkryje funkcję hasła backdoora, a tym samym uzyska dostęp do konta każdego użytkownika. Podobnie atak siłowy może skutkować dwoma różnymi „trafieniami”, ujawniając w ten sposób hasło backdoora, jak pokazano na rysunku



Niekompletna walidacja poświadczeń

Dobrze zaprojektowane mechanizmy uwierzytelniania wymuszają różne wymagania dotyczące haseł, takie jak minimalna długość lub obecność zarówno wielkich, jak i małych liter. Odpowiednio, niektóre źle zaprojektowane mechanizmy uwierzytelniania nie tylko nie egzekwują tych dobrych praktyk, ale także ich nie przyjmują pod uwagę własnych prób ich przestrzegania przez użytkowników. Na przykład niektóre aplikacje obcinają hasła i dlatego sprawdzają poprawność tylko pierwszych n znaków. Niektóre aplikacje sprawdzają hasła bez uwzględniania wielkości liter. Niektóre aplikacje usuwają nietypowe znaki (czasami pod pretekstem sprawdzania poprawności wprowadzonych danych) przed sprawdzeniem haseł. W ostatnim czasie tego rodzaju zachowanie zostało zidentyfikowane w niektórych zaskakująco głośnych aplikacjach internetowych, zwykle w wyniku prób i błędów ciekawskich użytkowników. Każde z tych ograniczeń sprawdzania poprawności hasła zmniejsza o rząd wielkości liczbę wariantów dostępnych w zbiorze możliwych haseł. Eksperymentując, możesz określić, czy hasło jest w pełni sprawdzane, czy też obowiązują jakieś ograniczenia. Następnie możesz dostroić swoje automatyczne ataki przeciwko loginowi, aby usunąć niepotrzebne przypadki testowe, znacznie zmniejszając w ten sposób liczbę żądań niezbędnych do naruszenia bezpieczeństwa kont użytkowników.

KROKI HACKOWANIA

1. Korzystając z konta, które kontrolujesz, spróbuj zalogować się z różnymi odmianami własnego hasła: usunięciem ostatniego znaku, zmianą wielkości liter i usunięciem specjalnych znaków typograficznych. Jeśli którakolwiek z tych prób się powiedzie, kontynuuj eksperymentowanie, aby spróbować zrozumieć, jaka walidacja faktycznie ma miejsce.

2. Przekaż wyniki z powrotem do zautomatyzowanych ataków opartych na odgadywaniu haseł, aby usunąć zbędne przypadki testowe i zwiększyć szanse powodzenia.

Nieunikalne nazwy użytkowników

Niektóre aplikacje obsługujące samodzielną rejestrację umożliwiają użytkownikom określenie własnej nazwy użytkownika i nie wymuszają wymogu unikalności nazw użytkowników. Chociaż jest to rzadkie, autorzy napotkali więcej niż jedną aplikację z takim zachowaniem. Stanowi to wadę projektową z dwóch powodów:

* Jeden użytkownik, który dzieli nazwę użytkownika z innym użytkownikiem, może również wybrać to samo hasło co ten użytkownik podczas rejestracji lub późniejszej zmiany hasła. W takiej sytuacji aplikacja albo odrzuca hasło wybrane przez drugiego użytkownika, albo zezwala dwóm kontom na posiadanie identycznych danych uwierzytelniających. W pierwszej kolejności zachowanie aplikacji skutecznie ujawnia jednemu użytkownikowi dane uwierzytelniające drugiego użytkownika. W drugim przypadku kolejne logowania jednego z użytkowników skutkują uzyskaniem dostępu do konta drugiego użytkownika.

* Osoba atakująca może wykorzystać to zachowanie do przeprowadzenia skutecznego ataku siłowego, nawet jeśli nie jest to możliwe gdzie indziej ze względu na ograniczenia dotyczące nieudanych prób logowania. Osoba atakująca może wielokrotnie rejestrować określoną nazwę użytkownika z różnymi hasłami, jednocześnie monitorując odpowiedź różnicową wskazującą, że konto z tą nazwą użytkownika i hasłem już istnieje. Osoba atakująca ustali hasło użytkownika docelowego bez podejmowania pojedynczej próby zalogowania się jako ten użytkownik.

Źle zaprojektowana funkcja samorejestracji może również zapewnić środki do wyliczania nazw użytkowników. Jeśli aplikacja nie zezwala na zduplikowane nazwy użytkowników, osoba atakująca może próbować zarejestrować dużą liczbę wspólnych nazw użytkowników, aby zidentyfikować istniejące nazwy użytkowników, które zostaną odrzucone.

KROKI HACKOWANIA

1. Jeśli możliwa jest samodzielna rejestracja, spróbuj dwukrotnie zarejestrować tę samą nazwę użytkownika z różnymi hasłami.

2. Jeśli aplikacja zablokuje drugą próbę rejestracji, możesz wykorzystać to zachowanie do wyliczenia istniejących nazw użytkowników, nawet jeśli nie jest to możliwe na głównej stronie logowania lub w innym miejscu. Wykonaj wiele prób rejestracji z listą typowych nazw użytkowników, aby zidentyfikować już zarejestrowane nazwy blokowane przez aplikację.

3. Jeśli rejestracja zduplikowanych nazw użytkowników powiedzie się, spróbuj dwukrotnie zarejestrować tę samą nazwę użytkownika z tym samym hasłem i określ zachowanie aplikacji:

A. Jeśli pojawi się komunikat o błędzie, możesz wykorzystać to zachowanie do przeprowadzenia ataku brute-force, nawet jeśli nie jest to możliwe na głównej stronie logowania. Celuj w wyliczoną lub odgadniętą nazwę użytkownika i próbuj wielokrotnie zarejestrować tę nazwę użytkownika za pomocą listy typowych haseł. Gdy aplikacja odrzuci określone hasło, prawdopodobnie znalazłeś istniejące hasło do docelowego konta.

B. Jeśli nie pojawi się żaden komunikat o błędzie, zaloguj się przy użyciu podanych poświadczeń i zobacz, co się stanie. Być może będziesz musiał zarejestrować kilku użytkowników i zmodyfikować różne dane przechowywane na każdym koncie, aby zrozumieć, czy takie zachowanie może zostać wykorzystane do uzyskania nieautoryzowanego dostępu do kont innych użytkowników.

Przewidywalne nazwy użytkowników

Niektóre aplikacje automatycznie generują nazwy użytkowników kont zgodnie z przewidywalną sekwencją (cust5331, cust5332 itd.). Kiedy aplikacja zachowuje się w ten sposób, osoba atakująca,

która jest w stanie rozpoznać sekwencję, może szybko uzyskać potencjalnie wyczerpującą listę wszystkich prawidłowych nazw użytkowników, które mogą posłużyć jako podstawa do dalszych ataków. W przeciwieństwie do metod wyliczania, które opierają się na wysyłaniu powtarzających się żądań opartych na listach słów, ten sposób określania nazw użytkowników można przeprowadzić w sposób nieinwazyjny przy minimalnej interakcji z aplikacją.

KROKI HACKOWANIA

1. Jeśli aplikacja generuje nazwy użytkowników, spróbuj uzyskać kilka w krótkich odstępach czasu i sprawdź, czy można rozpoznać jakąś sekwencję lub wzór.

2. Jeśli to możliwe, ekstrapoluj wstecz, aby uzyskać listę możliwych prawidłowych nazw użytkowników. Może to służyć jako podstawa do ataku siłowego na login i innych ataków, w których wymagane są prawidłowe nazwy użytkownika, takich jak wykorzystanie luk w kontroli dostępu

Przewidywalne hasła początkowe

W niektórych aplikacjach wszyscy użytkownicy są tworzeni jednocześnie lub w dużych partiach i automatycznie przypisywane im są początkowe hasła, które następnie są im przekazywane w jakiś sposób. Sposoby generowania haseł mogą umożliwić atakującemu przewidzenie haseł innych użytkowników aplikacji. Ten rodzaj luki występuje częściej w aplikacjach korporacyjnych opartych na intranecie — na przykład każdy pracownik ma utworzone konto w swoim imieniu i otrzymuje wydrukowane powiadomienie o swoim haśle. W najbardziej narażonych przypadkach wszyscy użytkownicy otrzymują to samo hasło lub hasło ściśle powiązane z nazwą użytkownika lub funkcją służbową. W innych przypadkach wygenerowane hasła mogą zawierać sekwencje, które można zidentyfikować lub odgadnąć, mając dostęp do bardzo małej próbki początkowych haseł.

KROKI HACKOWANIA

1. Jeśli aplikacja generuje hasła, spróbuj uzyskać kilka w krótkim odstępie czasu i sprawdź, czy można rozpoznać jakąś sekwencję lub wzór.

2. Jeśli to możliwe, dokonaj ekstrapolacji wzorca, aby uzyskać listę haseł dla innych użytkowników aplikacji.

3. Jeśli hasła wykazują wzór, który można skorelować z nazwami użytkowników, możesz spróbować zalogować się przy użyciu znanych lub odgadniętych nazw użytkowników i odpowiednich wywnioskowanych haseł.

4. W przeciwnym razie możesz użyć listy wywnioskowanych haseł jako podstawy do ataku siłowego z listą wyliczonych lub powszechnych nazw użytkowników.

Niebezpieczna dystrybucja poświadczeń

Wiele aplikacji wykorzystuje proces, w którym poświadczenia dla nowo utworzonych kont są dystrybuowane do użytkowników poza pasmem ich normalnej interakcji z aplikacją (na przykład pocztą, e-mailem lub SMS-em). Czasami dzieje się tak z powodów związanych z bezpieczeństwem, takich jak zapewnienie, że podany przez użytkownika adres pocztowy lub e-mail faktycznie należy do tej osoby. W niektórych przypadkach ten proces może stanowić zagrożenie dla bezpieczeństwa. Załóżmy na przykład, że dystrybuowana wiadomość zawiera zarówno nazwę użytkownika, jak i hasło, nie ma ograniczeń czasowych na ich użycie i nie ma wymogu zmiany hasła przez użytkownika przy pierwszym logowaniu. Jest wysoce prawdopodobne, że duża liczba, a nawet większość użytkowników aplikacji nie zmodyfikuje swoich początkowych danych uwierzytelniających i tyle wiadomości

dystrybucyjnych będzie istniało przez długi czas, podczas którego mogą uzyskać do nich dostęp osoby nieupoważnione. Czasami dystrybuowane są nie same dane uwierzytelniające, ale adres URL „aktywacji konta", który umożliwia użytkownikom ustawienie własnego hasła początkowego. Jeśli seria tych adresów URL wysyłanych do kolejnych użytkowników wykazuje jakąkolwiek sekwencję, osoba atakująca może to zidentyfikować, rejestrując wielu użytkowników w krótkim odstępie czasu, a następnie wywnioskować aktywacyjne adresy URL wysłane do ostatnich i przyszłych użytkowników. Powiązanym zachowaniem niektórych aplikacji internetowych jest umożliwienie nowym użytkownikom rejestracji kont w pozornie bezpieczny sposób, a następnie wysłanie powitalnej wiadomości e-mail do każdego nowego użytkownika zawierającej jego pełne dane logowania. W najgorszym przypadku świadomy bezpieczeństwa użytkownik, który zdecyduje się natychmiast zmienić swoje hasło, które prawdopodobnie zostało naruszone, otrzymuje kolejną wiadomość e-mail zawierającą nowe hasło „do wykorzystania w przyszłości". To zachowanie jest tak dziwaczne i niepotrzebne, że użytkownikom zaleca się zaprzestanie korzystania z aplikacji internetowych, które sobie na to pozwalają.

KROKI HACKOWANIA

1. Uzyskaj nowe konto. Jeśli nie musisz ustawiać wszystkich poświadczeń podczas rejestracji, określ sposób, w jaki aplikacja rozsyła poświadczenia do nowych użytkowników.

2. Jeśli używany jest adres URL aktywacji konta, spróbuj zarejestrować kilka nowych kont w krótkim odstępie czasu i zidentyfikuj kolejność otrzymanych adresów URL. Jeśli można określić wzorzec, spróbuj przewidzieć aktywacyjne adresy URL wysyłane do ostatnich i przyszłych użytkowników i spróbuj użyć tych adresów URL, aby przejąć kontrolę nad ich kontami.

3. Spróbuj wielokrotnie użyć jednego aktywacyjnego adresu URL i sprawdź, czy aplikacja na to pozwala. Jeśli nie, spróbuj zablokować konto docelowe przed ponownym użyciem adresu URL i sprawdź, czy teraz działa.

Błędy implementacyjne w uwierzytelnianiu

Nawet dobrze zaprojektowany mechanizm uwierzytelniania może być wysoce niepewny z powodu błędów popełnionych w jego implementacji. Błędy te mogą prowadzić do wycieku informacji, całkowitego pominięcia logowania lub osłabienia ogólnego bezpieczeństwa mechanizmu zgodnie z założeniami. Błędy implementacyjne są zwykle bardziej subtelne i trudniejsze do wykrycia niż wady projektowe, takie jak słabej jakości hasła i brutalna siła. Z tego powodu są one często owocnym celem ataków na najbardziej krytyczne z punktu widzenia bezpieczeństwa aplikacje, w przypadku których liczne modele zagrożeń i testy penetracyjne prawdopodobnie przyniosły jakiekolwiek nisko wiszące owoce. Autorzy zidentyfikowali każdą z opisanych tu wad implementacyjnych w aplikacjach internetowych wdrożonych przez duże banki.

Mechanizmy logowania w przypadku awarii

Logika fail-open to rodzaj błędu logicznego , który ma szczególnie poważne konsekwencje w kontekście mechanizmów uwierzytelniania. Poniżej znajduje się dość wymyślony przykład mechanizmu logowania, którego otwarcie nie powiedzie się. Jeśli wywołanie metody db.getUser() z jakiegoś powodu zgłosi wyjątek (na przykład wyjątek pustego wskaźnika powstały, ponieważ żądanie użytkownika nie zawiera parametru nazwy użytkownika lub hasła), logowanie powiedzie się. Chociaż wynikowa sesja może nie być powiązana z konkretną tożsamością użytkownika, a zatem może nie być w pełni funkcjonalna, nadal może umożliwić atakującemu dostęp do niektórych poufnych danych lub funkcji.

```
public Response checkLogin(Session session) {
try {
String uname = session.getParameter(“username”);
String passwd = session.getParameter(“password”);
User user = db.getUser(uname, passwd);
if (user == null) {
// invalid credentials
session.setMessage(“Login failed. “);
return doLogin(session);
}
}
catch (Exception e) {}
// valid user
session.setMessage(“Login successful. “);
return doMainMenu(session);
}
```

W terenie nie spodziewałbyś się, że taki kod przejdzie nawet najbardziej pobieżną kontrolę bezpieczeństwa. Jednak ten sam błąd koncepcyjny jest znacznie bardziej prawdopodobny w bardziej złożonych mechanizmach, w których wykonuje się liczne wywołania metod warstwowych, w których może pojawić się wiele potencjalnych błędów i być obsługiwanych w różnych miejscach, a bardziej skomplikowana logika walidacji może wymagać utrzymywania istotny stan o postępie logowania.

KROKI HACKOWANIA

1. Wykonaj pełne, prawidłowe logowanie przy użyciu konta, które kontrolujesz. Rejestruj wszystkie dane przesłane do aplikacji i każdą otrzymaną odpowiedź, korzystając z przechwytującego serwera proxy.

2. Wielokrotnie powtarzaj proces logowania, modyfikując fragmenty przesłanych danych w nieoczekiwany sposób. Na przykład dla każdego parametru żądania lub pliku cookie wysłanego przez klienta wykonaj następujące czynności:

A. Prześlij pusty ciąg znaków jako wartość.

B. Całkowicie usuń parę nazwa/wartość.

C. Podaj bardzo długie i bardzo krótkie wartości.

D. Prześlij ciągi znaków zamiast liczb i odwrotnie.

E. Prześlij ten sam przedmiot wiele razy, z tą samą i inną wartością.

3. W przypadku każdego złożonego nieprawidłowo sformułowanego wniosku dokładnie przejrzyj odpowiedź aplikacji, aby zidentyfikować wszelkie rozbieżności w stosunku do przypadku podstawowego.

4. Wykorzystaj te obserwacje z powrotem podczas opracowywania przypadków testowych. Gdy jedna modyfikacja powoduje zmianę w zachowaniu, spróbuj połączyć to z innymi zmianami, aby przesunąć logikę aplikacji do granic możliwości.

Wady w wieloetapowych mechanizmach logowania

Niektóre aplikacje używają skomplikowanych mechanizmów logowania obejmujących wiele etapów, takich jak:

* Wprowadzenie nazwy użytkownika i hasła

* Wyzwanie dla określonych cyfr z PIN-u lub niezapomnianego słowa

* Przesłanie wartości wyświetlanej na zmieniającym się tokenie fizycznym

Wieloetapowe mechanizmy logowania zostały zaprojektowane tak, aby zapewnić większe bezpieczeństwo w porównaniu z prostym modelem opartym na nazwie użytkownika i haśle. Zazwyczaj pierwszy etap wymaga od użytkowników identyfikacji za pomocą nazwy użytkownika lub podobnego elementu, a kolejne etapy przeprowadzają różne kontrole uwierzytelnienia. Takie mechanizmy często zawierają luki w zabezpieczeniach - w szczególności różne błędy logiczne

POWSZECHNY MIT

Często zakłada się, że wieloetapowe mechanizmy logowania są mniej podatne na obejścia zabezpieczeń niż standardowe uwierzytelnianie za pomocą nazwy użytkownika/hasła. To przekonanie jest błędne. Wykonanie kilku kontroli uwierzytelnienia może znacznie zwiększyć bezpieczeństwo mechanizmu. Ale równoważąc to, proces jest bardziej podatny na błędy we wdrażaniu. W kilku przypadkach, gdy występuje kombinacja wad, może to nawet skutkować rozwiązaniem mniej bezpiecznym niż zwykłe logowanie oparte na nazwie użytkownika i haśle. Niektóre implementacje wieloetapowych mechanizmów logowania przyjmują potencjalnie niebezpieczne założenia na każdym etapie dotyczące interakcji użytkownika z wcześniejszymi etapami:

* Aplikacja może zakładać, że użytkownik, który uzyskuje dostęp do etapu trzeciego, musi mieć wyczyszczone etapy pierwszy i drugi. Dlatego może uwierzytelnić atakującego, który przechodzi bezpośrednio z etapu pierwszego do etapu trzeciego i poprawnie go kończy, umożliwiając atakującemu zalogowanie się tylko przy użyciu jednej części różnych normalnie wymaganych poświadczeń.

* Aplikacja może ufać niektórym danym przetwarzanym na etapie drugim, ponieważ zostały one zweryfikowane na etapie pierwszym. Jednak osoba atakująca może być w stanie manipulować tymi danymi na etapie drugim, nadając im inną wartość niż zweryfikowana na etapie pierwszym. Na przykład na pierwszym etapie aplikacja może ustalić, czy konto użytkownika wygasło, jest zablokowane lub znajduje się w grupie administracyjnej lub czy musi przejść dalsze etapy logowania poza etapem drugim. Jeśli osoba atakująca może ingerować w te flagi, gdy logowanie przechodzi między różnymi etapami, może zmodyfikować zachowanie aplikacji i spowodować, że będzie ona uwierzytelniać go tylko za pomocą częściowych danych uwierzytelniających lub w inny sposób podnosić uprawnienia.

* Aplikacja może zakładać, że do ukończenia każdego etapu używana jest ta sama tożsamość użytkownika; może jednak nie sprawdzić tego jawnie. Na przykład etap pierwszy może obejmować przesłanie prawidłowej nazwy użytkownika i hasła, a etap drugi może obejmować ponowne przesłanie

nazwy użytkownika (teraz w ukrytym polu formularza) i wartości ze zmieniającego się fizycznego tokena. Jeśli osoba atakująca prześle prawidłowe pary danych na każdym etapie, ale dla różnych użytkowników, aplikacja może uwierzytelnić użytkownika jako jedną z tożsamości używanych na dwóch etapach. Umożliwiłoby to atakującemu, który posiada własny token fizyczny i odkryje hasło innego użytkownika, zalogowanie się jako ten użytkownik (lub odwrotnie). Chociaż mechanizm logowania nie może zostać całkowicie naruszony bez uprzedniej informacji, jego ogólny stan bezpieczeństwa jest znacznie osłabiony, a znaczne koszty i wysiłek związany z wdrożeniem mechanizmu dwuskładnikowego nie przynoszą oczekiwanych korzyści.

KROKI HACKOWANIA

1. Wykonaj pełne, prawidłowe logowanie przy użyciu konta, które kontrolujesz. Rejestruj wszystkie dane przesyłane do aplikacji za pomocą przechwytującego serwera proxy.

2. Zidentyfikuj każdy odrębny etap logowania i dane gromadzone na każdym etapie. Określ, czy jakakolwiek pojedyncza informacja jest zbierana więcej niż raz, czy też kiedykolwiek jest przesyłana z powrotem do klienta i przesyłana ponownie za pośrednictwem ukrytego pola formularza, pliku cookie lub wstępnie ustawionego parametru adresu URL.

3. Powtórz proces logowania wiele razy z różnymi zniekształceniami:

A. Spróbuj wykonać kroki logowania w innej kolejności.

B. Spróbuj przejść bezpośrednio do dowolnego etapu i kontynuować stamtąd.

C. Spróbuj pominąć każdy etap i przejść do następnego.

D. Użyj swojej wyobraźni, aby wymyślić inne sposoby uzyskania dostępu do różnych etapów, których programiści mogli nie przewidzieć.

4. Jeśli jakieś dane są przesyłane więcej niż jeden raz, spróbuj podać inną wartość na różnych etapach i sprawdź, czy logowanie nadal się udaje. Może się zdarzyć, że niektóre zgłoszenia są zbędne i nie są faktycznie przetwarzane przez aplikację. Może się zdarzyć, że dane zostaną zweryfikowane na jednym etapie, a następnie zaufane. W takim przypadku spróbuj podać poświadczenia jednego użytkownika na jednym etapie, a następnie przełącz się na następnym, aby faktycznie uwierzytelnić się jako inny użytkownik. Może się zdarzyć, że ten sam fragment danych zostanie zweryfikowany na więcej niż jednym etapie, ale w ramach różnych kontroli. W takim przypadku spróbuj podać (na przykład) nazwę użytkownika i hasło jednego użytkownika na pierwszym etapie, a nazwę użytkownika i PIN innego użytkownika na drugim etapie.

5. Zwracaj szczególną uwagę na wszelkie dane przesyłane przez klienta, które nie zostały wprowadzone bezpośrednio przez użytkownika. Aplikacja może wykorzystywać te dane do przechowywania informacji o stanie postępów logowania, a aplikacja może im zaufać, gdy zostaną przesłane z powrotem na serwer. Na przykład, jeśli żądanie dotyczące etapu trzeciego zawiera parametr stage2complete=true, możliwe jest przejście bezpośrednio do etapu trzeciego przez ustawienie tej wartości. Spróbuj zmodyfikować przesyłane wartości i określ, czy umożliwia to przejście do kolejnych etapów, czy ich pominięcie.

Niektóre mechanizmy logowania wykorzystują losowo zmieniające się pytanie na jednym z etapów procesu logowania. Na przykład po podaniu nazwy użytkownika i hasła użytkownik może zostać poproszony o jedno z różnych „tajnych” pytań (dotyczących nazwiska panieńskiego matki, miejsca urodzenia, nazwy pierwszej szkoły) lub o podanie dwóch losowych liter z tajnego wyrażenia. Uzasadnieniem takiego zachowania jest to, że nawet jeśli atakujący przechwyci wszystko, co

użytkownik wpisze przy jednej okazji, nie umożliwi mu to zalogowania się jako ten użytkownik przy innej okazji, ponieważ zostaną zadane inne pytania. W niektórych implementacjach ta funkcjonalność jest zepsuta i nie osiąga swojego celu:

* Aplikacja może prezentować losowo wybrane pytanie i przechowywać szczegóły w ukrytym polu formularza HTML lub pliku cookie, a nie na serwerze. Następnie użytkownik przesyła zarówno odpowiedź, jak i samo pytanie. To skutecznie pozwala atakującemu wybrać pytanie, na które ma odpowiedzieć, umożliwiając atakującemu powtórzenie logowania po jednorazowym przechwyceniu danych wprowadzonych przez użytkownika.

* Aplikacja może prezentować losowo wybrane pytanie przy każdej próbie logowania, ale nie zapamiętuje, jakie pytanie zadał dany użytkownik, jeśli nie prześle odpowiedzi. Jeśli ten sam użytkownik zainicjuje ponowną próbę logowania chwilę później, generowane jest inne losowe pytanie. To skutecznie pozwala atakującemu przechodzić przez kolejne pytania, aż otrzyma takie, na które zna odpowiedź, co umożliwia mu powtórzenie logowania po jednorazowym przechwyceniu danych wprowadzonych przez użytkownika.

UWAGA: Drugi z tych warunków jest naprawdę dość subtelny, w wyniku czego wiele aplikacji w świecie rzeczywistym jest podatnych na ataki. Na pierwszy rzut oka może się wydawać, że aplikacja, która wyzywa użytkownika na dwie losowe litery zapamiętanego słowa, działa poprawnie i zapewnia zwiększone bezpieczeństwo. Jeśli jednak litery są wybierane losowo za każdym razem, gdy przechodzi poprzedni etap uwierzytelniania, osoba atakująca, która jednorazowo przechwyciła login użytkownika, może po prostu ponownie uwierzytelnić się do tego momentu, dopóki nie zostaną zażądane dwie znane mu litery, bez ryzyka blokada konta.

KROKI HACKOWANIA

1. Jeśli na jednym z etapów logowania używane jest losowo zmieniające się pytanie, sprawdź, czy wraz z odpowiedzią przesyłane są szczegóły pytania. Jeśli tak, zmień pytanie, prześlij poprawną odpowiedź powiązaną z tym pytaniem i sprawdź, czy logowanie nadal się powiodło.

2. Jeżeli aplikacja nie umożliwia atakującemu zadania dowolnego pytania i odpowiedzi, wykonaj kilkukrotne częściowe logowanie do jednego konta, przechodząc za każdym razem do pytania zmiennego. Jeśli pytanie zmienia się za każdym razem, osoba atakująca może nadal skutecznie wybrać, na które pytanie odpowiedzieć.

UWAGA: W niektórych aplikacjach, w których jeden składnik logowania zmienia się losowo, aplikacja gromadzi wszystkie dane uwierzytelniające użytkownika na jednym etapie. Na przykład główna strona logowania może prezentować formularz zawierający pola na nazwę użytkownika, hasło i jedno z różnych tajnych pytań. Za każdym razem, gdy ładowana jest strona logowania, zmienia się tajne pytanie. W tej sytuacji losowość tajnego pytania nie przeszkadza atakującemu w ponownym odtworzeniu prawidłowego żądania logowania po przechwyceniu danych wprowadzonych przez użytkownika przy jednej okazji. Procesu logowania nie można zmodyfikować w obecnej formie, ponieważ osoba atakująca może po prostu ponownie załadować stronę, dopóki nie otrzyma pytania, na które zna odpowiedź. W odmianie tego scenariusza aplikacja może ustawić trwały plik cookie, aby „zapewnić", że to samo zmieniające się pytanie zostanie przedstawione dowolnemu użytkownikowi, dopóki osoba ta nie odpowie na nie poprawnie. Oczywiście środek ten można łatwo obejść, modyfikując lub usuwając plik cookie.

Niebezpieczne przechowywanie poświadczeń

Jeśli aplikacja przechowuje dane logowania w sposób niezabezpieczony, bezpieczeństwo mechanizmu logowania jest zagrożone, nawet jeśli sam proces uwierzytelniania może nie mieć żadnej wady. Często spotyka się aplikacje internetowe, w których poświadczenia użytkownika są przechowywane w bazie danych w sposób niezabezpieczony. Może to obejmować hasła przechowywane w postaci czystego tekstu. Ale jeśli hasła są haszowane przy użyciu standardowego algorytmu, takiego jak MD5 lub SHA-1, nadal pozwala to atakującemu po prostu wyszukać zaobserwowane skróty w wstępnie obliczonej bazie danych wartości skrótów. Ponieważ konto bazy danych używane przez aplikację musi mieć pełny dostęp do odczytu/zapisu tych danych uwierzytelniających, wiele innych luk w zabezpieczeniach aplikacji może zostać wykorzystanych w celu umożliwienia dostępu do tych danych uwierzytelniających, takich jak błędy poleceń lub iniekcji SQL (patrz część 9) oraz niedociągnięcia w kontroli dostępu (zob. część 8).

KROKI HACKOWANIA

1. Przejrzyj wszystkie funkcje aplikacji związane z uwierzytelnianiem, a także wszelkie funkcje związane z obsługą użytkowników. Jeśli znajdziesz przypadki, w których hasło użytkownika jest przesyłane z powrotem do klienta, oznacza to, że hasła są przechowywane w sposób niezabezpieczony, albo w postaci zwykłego tekstu, albo przy użyciu odwracalnego szyfrowania.

2. Jeśli w aplikacji zostanie zidentyfikowana jakakolwiek luka umożliwiająca wykonanie dowolnego polecenia lub zapytania, spróbuj znaleźć lokalizację w bazie danych lub systemie plików aplikacji, w której przechowywane są dane uwierzytelniające użytkownika:

A. Zapytaj je, aby ustalić, czy hasła są przechowywane w postaci niezaszyfrowanej.

B. Jeśli hasła są przechowywane w formie zaszyfrowanej, sprawdź, czy nie są unikatowe wartości, wskazujące, że konto ma przypisane wspólne lub domyślne hasło i że zaszyfrowane hasła nie są solone.

C. Jeśli hasło jest haszowane za pomocą standardowego algorytmu w postaci niesolonej, przeszukaj internetowe bazy danych hash, aby określić odpowiednią wartość hasła w postaci zwykłego tekstu.

Zabezpieczanie uwierzytelniania

Wdrożenie bezpiecznego rozwiązania do uwierzytelniania obejmuje próbę jednoczesnego spełnienia kilku kluczowych celów związanych z bezpieczeństwem, a w wielu przypadkach osiągnięcie kompromisu w stosunku do innych celów, takich jak funkcjonalność, użyteczność i całkowity koszt. W niektórych przypadkach „więcej” bezpieczeństwa może faktycznie przynieść efekt przeciwny do zamierzonego. Na przykład zmuszanie użytkowników do ustawiania bardzo długich haseł i częstej ich zmiany często powoduje, że użytkownicy zapisują swoje hasła. Ze względu na ogromną różnorodność możliwych luk w zabezpieczeniach uwierzytelniania oraz potencjalnie złożone mechanizmy obronne, których aplikacja może potrzebować w celu złagodzenia wszystkich z nich, wielu projektantów i programistów aplikacji akceptuje pewne zagrożenia jako pewnik i koncentruje się na zapobieganiu najpoważniejszym atakom . Oto kilka czynników, które należy wziąć pod uwagę, aby uzyskać odpowiednią równowagę:

* Krytyczne znaczenie bezpieczeństwa, biorąc pod uwagę funkcjonalność, jaką oferuje aplikacja

* Stopień, w jakim użytkownicy będą tolerować i pracować z różnymi typami kontroli uwierzytelniania

* Koszt obsługi mniej przyjaznego dla użytkownika systemu

* Koszt finansowy konkurencyjnych alternatyw w stosunku do przychodów, które aplikacja może wygenerować lub wartości chronionych przez nią aktywów

W tej sekcji opisano najskuteczniejsze sposoby pokonania różnych ataków na mechanizmy uwierzytelniania. Pozostawiamy Ci decyzję, które rodzaje obrony są najbardziej odpowiednie w każdym przypadku.

Użyj silnych poświadczeń

* Należy egzekwować odpowiednie minimalne wymagania dotyczące jakości hasła. Mogą one obejmować zasady dotyczące minimalnej długości; wygląd znaków alfabetycznych, numerycznych i typograficznych; wygląd zarówno wielkich, jak i małych liter; unikanie słownikowych słów, nazw i innych powszechnych haseł; uniemożliwienie ustawienia hasła do nazwy użytkownika; i zapobieganie podobieństwu lub dopasowaniu do wcześniej ustawionych haseł. Podobnie jak w przypadku większości środków bezpieczeństwa, dla różnych kategorii użytkowników mogą obowiązywać różne wymagania dotyczące jakości hasła.

* Nazwy użytkowników powinny być unikalne.

* Wszelkie generowane przez system nazwy użytkowników i hasła powinny być tworzone z wystarczającą entropią, aby nie można było ich zsekwencjonować ani przewidzieć - nawet przez atakującego, który uzyska dostęp do dużej próbki kolejno generowanych instancji.

* Użytkownicy powinni mieć możliwość ustawiania wystarczająco silnych haseł. Na przykład powinny być dozwolone długie hasła i szeroki zakres znaków

Potajemnie obsługuj dane uwierzytelniające

* Wszystkie dane uwierzytelniające powinny być tworzone, przechowywane i przesyłane w sposób, który nie prowadzi do nieupoważnionego ujawnienia.

* Cała komunikacja klient-serwer powinna być chroniona przy użyciu dobrze znanej technologii kryptograficznej, takiej jak SSL. Niestandardowe rozwiązania do ochrony przesyłanych danych nie są ani konieczne, ani pożądane.

* Jeśli preferowane jest użycie HTTP dla nieuwierzytelnionych obszarów aplikacji, upewnij się, że sam formularz logowania jest ładowany przy użyciu HTTPS, zamiast przełączać się na HTTPS w momencie przesyłania logowania.

* Do przesyłania poświadczeń na serwer należy używać tylko żądań POST. Nigdy nie należy umieszczać danych uwierzytelniających w parametrach adresu URL lub plikach cookie (nawet efemerycznych). Poświadczenia nigdy nie powinny być przesyłane z powrotem do klienta, nawet w parametrach przekierowania.

* Wszystkie komponenty aplikacji po stronie serwera powinny przechowywać dane uwierzytelniające w sposób uniemożliwiający łatwe odzyskanie ich pierwotnych wartości, nawet przez atakującego, który uzyska pełny dostęp do wszystkich istotnych danych w bazie danych aplikacji. Zwykłym sposobem osiągnięcia tego celu jest użycie silnej funkcji skrótu (takiej jak SHA-256 w momencie pisania tego tekstu), odpowiednio zasolonej, aby zmniejszyć skuteczność wstępnie obliczonych ataków offline. Sól powinna być specyficzna dla konta, do którego należy hasło, tak aby osoba atakująca nie mogła odtworzyć ani zastąpić wartości skrótu.

* Funkcja „zapamiętaj mnie” po stronie klienta powinna zasadniczo zapamiętywać tylko nietajne elementy, takie jak nazwy użytkowników. W aplikacjach o mniejszym znaczeniu dla bezpieczeństwa można uznać za właściwe zezwolenie użytkownikom na włączenie funkcji zapamiętywania haseł. W

takiej sytuacji na kliencie nie należy przechowywać poświadczeń w postaci zwykłego tekstu (hasło powinno być przechowywane w postaci zaszyfrowanej odwracalnie przy użyciu klucza znanego tylko serwerowi). Ponadto użytkownicy powinni zostać ostrzeżeni o zagrożeniach ze strony osoby atakującej, która ma fizyczny dostęp do ich komputera lub zdalnie włamuje się do komputera. Szczególną uwagę należy zwrócić na wyeliminowanie w aplikacji podatności typu cross-site scripting, które mogą posłużyć do kradzieży przechowywanych danych uwierzytelniających.

* Należy wdrożyć funkcję zmiany hasła, a użytkownicy powinni być zobowiązani do okresowej zmiany hasła.

* W przypadku, gdy dane uwierzytelniające do nowych kont są przekazywane użytkownikom poza pasmem, należy je przesyłać tak bezpiecznie, jak to możliwe, i powinno to być ograniczone w czasie. Użytkownik powinien być zobowiązany do ich zmiany przy pierwszym logowaniu i powinien zostać poinformowany o zniszczeniu komunikacji po pierwszym użyciu.

* W stosownych przypadkach rozważ przechwycenie niektórych danych logowania użytkownika (na przykład pojedynczych liter zapamiętanego słowa) za pomocą menu rozwijanych zamiast pól tekstowych. Zapobiegnie to przechwytywaniu wszystkich danych przesłanych przez użytkownika przez keyloggery zainstalowane na komputerze użytkownika. (Należy jednak pamiętać, że prosty keylogger to tylko jeden ze sposobów, za pomocą których osoba atakująca może przechwycić dane wprowadzane przez użytkownika. Jeśli osoba atakująca już włamała się do komputera użytkownika, w zasadzie może zarejestrować każdy typ zdarzenia, w tym ruchy myszy, wysłanie formularza przez HTTPS i zrzuty ekranu).

Prawidłowo zweryfikuj dane uwierzytelniające

* Hasła powinny być sprawdzane w całości - to znaczy z uwzględnieniem wielkości liter, bez filtrowania lub modyfikowania jakichkolwiek znaków oraz bez obcinania hasła.

* Aplikacja powinna agresywnie bronić się przed nieoczekiwanymi zdarzeniami występującymi podczas przetwarzania logowania. Na przykład, w zależności od używanego języka programistycznego, aplikacja powinna używać obsługi wyjątków typu catch-all wokół wszystkich wywołań API. Powinny one jawnie usuwać wszystkie lokalne dane sesji i metod, które są wysyłane w celu kontrolowania stanu przetwarzania logowania i powinny jawnie unieważniać bieżącą sesję, powodując w ten sposób wymuszone wylogowanie przez serwer, nawet jeśli uwierzytelnianie zostanie w jakiś sposób ominięte.

* Cała logika uwierzytelniania powinna zostać dokładnie sprawdzona pod kątem kodu, zarówno pseudokodu, jak i rzeczywistego kodu źródłowego aplikacji, w celu zidentyfikowania błędów logicznych, takich jak warunki otwarcia awaryjnego.

* Jeśli zaimplementowana jest funkcja wspierająca podszywanie się pod użytkownika, należy to ściśle kontrolować, aby upewnić się, że nie można jej niewłaściwie wykorzystać w celu uzyskania nieautoryzowanego dostępu. Ze względu na krytyczność funkcjonalności często warto usunąć tę funkcjonalność z aplikacji publicznej i wdrożyć ją tylko dla wewnętrznych użytkowników administracyjnych, których użycie personifikacji powinno być ściśle kontrolowane i audytowane.

* Logowania wieloetapowe powinny być ściśle kontrolowane, aby uniemożliwić atakującemu ingerowanie w przejścia i relacje między etapami:

* Wszystkie dane o postępie przez kolejne etapy oraz wyniki poprzednich zadań walidacyjnych powinny być przechowywane w obiekcie sesji po stronie serwera i nigdy nie powinny być przesyłane ani odczytywane z klienta.

* Użytkownik nie powinien przekazywać żadnych informacji więcej niż raz, a użytkownik nie powinien mieć możliwości modyfikowania danych, które zostały już zebrane i/lub zweryfikowane. Jeżeli element danych, taki jak nazwa użytkownika, jest używany na wielu etapach, powinien być przechowywany w zmiennej sesyjnej podczas pierwszego zbierania, a następnie odwoływać się do niego.

* Pierwszym zadaniem realizowanym na każdym etapie powinno być zweryfikowanie tego wszystkiego jak poprzednie etapy zostały poprawnie zakończone. Jeśli tak nie jest, próba uwierzytelnienia powinna zostać natychmiast oznaczona jako błędna.

* Aby zapobiec wyciekowi informacji o tym, który etap logowania się nie powiódł (co umożliwiłoby atakującemu wycelowanie w każdy etap po kolei), aplikacja powinna zawsze przechodzić przez wszystkie etapy logowania, nawet jeśli użytkownik nie wykonał poprawnie wcześniejszych etapów, oraz nawet jeśli pierwotna nazwa użytkownika była nieprawidłowa. Po przejściu przez wszystkie etapy, na zakończenie ostatniego etapu aplikacja powinna wyświetlić ogólny komunikat „logowanie nie powiodło się", bez podawania informacji o tym, gdzie wystąpił błąd.

* Jeśli proces logowania obejmuje losowo zmieniające się pytanie, upewnij się, że atakujący nie może skutecznie wybrać własnego pytania:

* Zawsze stosuj wieloetapowy proces, w którym użytkownicy identyfikują się na początkowym etapie, a losowo zmieniające się pytanie jest im przedstawiane na późniejszym etapie.

* Gdy danemu użytkownikowi zostanie przedstawione zmienne pytanie, zapisz to pytanie w jego trwałym profilu użytkownika i upewnij się, że ten sam użytkownik otrzymuje to samo pytanie przy każdej próbie logowania, dopóki nie udzieli na nie pomyślnej odpowiedzi.

* Kiedy losowo zmieniające się wyzwanie jest przedstawiane użytkownikowi, zapisz pytanie, które zostało zadane w zmiennej sesyjnej po stronie serwera, a nie w ukrytym polu w formularzu HTML, i sprawdź poprawność kolejnej odpowiedzi względem tego zapisanego pytania.

UWAGA: Subtelności związane z opracowaniem bezpiecznego mechanizmu uwierzytelniania są tutaj głębokie. Jeśli nie zachowa się ostrożności podczas zadawania losowo zmieniających się pytań, może to prowadzić do nowych możliwości wyliczania nazw użytkowników. Na przykład, aby uniemożliwić osobie atakującej wybranie własnego pytania, aplikacja może przechowywać w profilu każdego użytkownika ostatnie zadane mu pytanie i wyświetlać to pytanie, dopóki użytkownik nie odpowie na nie poprawnie. Atakujący, który zainicjuje kilka logowań przy użyciu dowolnej nazwy użytkownika, spotka się z tym samym pytaniem. Jeśli jednak atakujący przeprowadzi ten sam proces przy użyciu nieprawidłowej nazwy użytkownika, aplikacja może zachowywać się inaczej: ponieważ żaden profil użytkownika nie jest powiązany z nieprawidłową nazwą użytkownika, nie zostanie zapisane żadne pytanie, więc zostanie wyświetlone inne pytanie. Atakujący może wykorzystać tę różnicę w zachowaniu, przejawiającą się w kilku próbach logowania, aby wywnioskować ważność danej nazwy użytkownika. W ataku skryptowym będzie w stanie szybko zebrać wiele nazw użytkowników. Jeśli aplikacja chce się bronić przed taką możliwością, musi dołożyć wszelkich starań. Gdy próba logowania zostanie zainicjowana przy użyciu nieprawidłowej nazwy użytkownika, aplikacja musi gdzieś zapisać losowe pytanie, które zadała dla tej nieprawidłowej nazwy użytkownika, i upewnić się, że kolejne próby logowania przy użyciu tej samej nazwy użytkownika spotkają się z tym samym pytaniem. Idąc jeszcze dalej, aplikacja może okresowo przełączać się na inne pytanie, aby zasymulować normalne zalogowanie się nieistniejącego użytkownika, co skutkuje zmianą w następnym pytaniu! W pewnym momencie jednak projektant aplikacji musi postawić granicę i przyznać, że całkowite zwycięstwo nad tak zdeterminowanym napastnikiem prawdopodobnie nie jest możliwe.

Zapobiegaj wyciekom informacji

* Różne mechanizmy uwierzytelniania używane przez aplikację nie powinny ujawniać żadnych informacji o parametrach uwierzytelniania, ani poprzez jawne komunikaty, ani wnioskowanie z innych aspektów zachowania aplikacji. Atakujący nie powinien mieć możliwości określenia, który element z różnych przesłanych elementów spowodował problem.

* Pojedynczy komponent kodu powinien być odpowiedzialny za reagowanie na wszystkie nieudane próby logowania za pomocą ogólnego komunikatu. Pozwala to uniknąć subtelnej luki w zabezpieczeniach, która może wystąpić, gdy rzekomo nieinformacyjna wiadomość zwrócona z różnych ścieżek kodu może zostać wykryta przez atakującego z powodu różnic typograficznych w wiadomości, różnych kodów stanu HTTP, innych informacji ukrytych w HTML i tym podobnych.

* Jeśli aplikacja wymusza jakąś blokadę konta, aby zapobiec atakom typu bruteforce (jak omówiono w następnej sekcji), należy uważać, aby nie doprowadzić do wycieku informacji. Na przykład, jeśli aplikacja ujawnia, że określone konto zostało zawieszone na X minut z powodu Y nieudanych logowań, to zachowanie można łatwo wykorzystać do wyliczenia prawidłowych nazw użytkowników. Ponadto ujawnienie dokładnych metryk zasad blokowania umożliwia atakującemu optymalizację wszelkich prób kontynuowania odgadywania haseł pomimo zasad. Aby uniknąć wyliczania nazw użytkowników, aplikacja powinna odpowiadać na każdą serię nieudanych prób logowania z tej samej przeglądarki ogólnym komunikatem informującym, że konta są zawieszane w przypadku wielu niepowodzeń i że użytkownik powinien spróbować ponownie później. Można to osiągnąć za pomocą pliku cookie lub ukrytego pola do śledzenia powtarzających się błędów pochodzących z tej samej przeglądarki. (Oczywiście ten mechanizm nie powinien być używany do wymuszania jakiejkolwiek rzeczywistej kontroli bezpieczeństwa - tylko do dostarczania przydatnej wiadomości zwykłym użytkownikom, którzy mają trudności z zapamiętaniem swoich danych uwierzytelniających).

* Jeśli aplikacja obsługuje samodzielną rejestrację, może uniemożliwić użycie tej funkcji do wyliczania istniejących nazw użytkowników na dwa sposoby:

* Zamiast zezwalać na samodzielny wybór nazwy użytkownika, aplikacja może utworzyć unikalną (i nieprzewidywalną) nazwę użytkownika dla każdego nowego użytkownika, eliminując w ten sposób potrzebę ujawniania, że wybrana nazwa użytkownika już istnieje.

* Aplikacja może wykorzystywać adresy e-mail jako nazwy użytkowników. W tym przypadku pierwszy etap procesu rejestracji polega na podaniu przez użytkownika adresu e-mail, po czym wystarczy poczekać na wiadomość e-mail i postępować zgodnie z zawartymi w niej instrukcjami. Jeśli adres e-mail jest już zarejestrowany, użytkownik może zostać o tym poinformowany w wiadomości e-mail. Jeśli adres nie jest jeszcze zarejestrowany, użytkownik może otrzymać unikalny, niemożliwy do odgadnięcia adres URL, który należy odwiedzić, aby kontynuować proces rejestracji. Uniemożliwia to atakującemu wyliczenie prawidłowych nazw użytkowników (chyba że zdarzyło się, że naruszył już dużą liczbę kont e-mail).

Zapobiegaj atakom brutalnej siły

* Środki muszą być egzekwowane w ramach wszystkich różnych wyzwań realizowanych przez funkcję uwierzytelniania, aby zapobiegać atakom, które próbują sprostać tym wyzwaniom za pomocą automatyzacji. Obejmuje to samo logowanie, a także funkcje zmiany hasła, odzyskiwania po zapomnianym haśle i tym podobne.

* Używanie nieprzewidywalnych nazw użytkowników i zapobieganie ich wyliczaniu stanowi istotną przeszkodę dla całkowicie ślepych ataków brute-force i wymaga od atakującego, aby w jakiś sposób odkrył jedną lub więcej określonych nazw użytkowników przed przystąpieniem do ataku.

* Niektóre aplikacje o krytycznym znaczeniu dla bezpieczeństwa (takie jak banki internetowe) po prostu wyłączają konto po niewielkiej liczbie nieudanych logowań (np. trzech). Wymagają również, aby właściciel konta podjął różne działania poza pasmem w celu ponownej aktywacji konta, takie jak telefon do obsługi klienta i udzielenie odpowiedzi na szereg pytań bezpieczeństwa. Wadą tej zasady jest to, że umożliwia ona osobie atakującej odmowę świadczenia usług uprawnionym użytkownikom poprzez wielokrotne wyłączanie ich kont oraz koszt świadczenia usługi odzyskiwania konta. Bardziej zrównoważoną zasadą, odpowiednią dla większości aplikacji świadomych bezpieczeństwa, jest zawieszenie kont na krótki okres (np. 30 minut) po niewielkiej liczbie nieudanych prób logowania (np. trzech). Służy to znacznemu spowolnieniu wszelkich ataków polegających na odgadywaniu haseł, jednocześnie zmniejszając ryzyko ataków typu „odmowa usługi", a także zmniejszając pracę call center.

* W przypadku wprowadzenia polityki czasowego zawieszenia konta należy zadbać o jej skuteczność:

* Aby zapobiec wyciekowi informacji prowadzącemu do wyliczenia nazwy użytkownika, aplikacja nigdy nie powinna wskazywać, że jakiekolwiek konto zostało zawieszone. Powinien raczej reagować na każdą serię nieudanych logowań, nawet tych z użyciem nieprawidłowej nazwy użytkownika, komunikatem informującym, że konta zostaną zawieszone, jeśli wystąpi wiele niepowodzeń, i że użytkownik powinien spróbować ponownie później (jak już omówiono).

* Metryki polityki nie powinny być ujawniane użytkownikom. Samo powiedzenie legalnym użytkownikom, aby „spróbowali ponownie później", nie obniża poważnie jakości ich usług. Ale dokładne poinformowanie atakującego, ile nieudanych prób jest tolerowanych i jak długi jest okres zawieszenia, umożliwia mu optymalizację wszelkich prób kontynuowania odgadywania haseł pomimo polityki.

* Jeśli konto jest zawieszone, próby logowania powinny być odrzucane nawet bez sprawdzania danych uwierzytelniających. Niektóre aplikacje, które wdrożyły zasady zawieszania, pozostają podatne na ataki siłowe, ponieważ nadal w pełni przetwarzają próby logowania w okresie zawieszenia i zwracają subtelnie (lub nie tak subtelnie) inny komunikat, gdy przesyłane są prawidłowe dane uwierzytelniające. Takie zachowanie umożliwia przeprowadzenie skutecznego ataku brute-force z pełną prędkością, niezależnie od zasad zawieszania.

* Środki zaradcze dla poszczególnych kont, takie jak blokada konta, nie chronią przed jednym rodzajem ataku siłowego, który często jest bardzo skuteczny — iteracją przez długą listę wyliczonych nazw użytkowników, sprawdzaniem pojedynczego słabego hasła, takiego jak hasło. Na przykład, jeśli pięć nieudanych prób skutkuje zawieszeniem konta, oznacza to, że osoba atakująca może spróbować wprowadzić cztery różne hasła do każdego konta, nie powodując żadnych zakłóceń dla użytkowników. W typowej aplikacji zawierającej wiele słabych haseł taki atakujący może naruszyć bezpieczeństwo wielu kont.

Skuteczność tego rodzaju ataku zostanie oczywiście znacznie zmniejszona, jeśli inne obszary mechanizmu uwierzytelniania zostaną zaprojektowane w sposób bezpieczny. Jeśli nazw użytkowników nie można wyliczyć ani wiarygodnie przewidzieć, osoba atakująca zostanie spowolniona przez konieczność wykonania brutalnego ćwiczenia polegającego na odgadywaniu nazw użytkowników. A jeśli obowiązują surowe wymagania dotyczące jakości hasła, jest znacznie mniej prawdopodobne, że atakujący wybierze hasło do testowania, które wybrał nawet jeden użytkownik aplikacji. Oprócz tych kontroli aplikacja może w szczególności chronić się przed tego rodzaju atakiem, wykorzystując

wyzwania CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) na każdej stronie, która może być celem ataków typu brute-force. Jeśli jest to skuteczne, środek ten może uniemożliwić automatyczne przesyłanie danych do dowolnej strony aplikacji, zapobiegając w ten sposób ręcznym wykonywaniu wszelkiego rodzaju ataków polegających na odgadywaniu hasła. Należy zauważyć, że przeprowadzono wiele badań nad technologiami CAPTCHA, a zautomatyzowane ataki na nie były w niektórych przypadkach niezawodne. Ponadto wiadomo, że niektórzy napastnicy wymyślają konkursy rozwiązywania CAPTCHA, w których nieświadomi członkowie społeczeństwa są wykorzystywani jako drony do pomocy atakującemu. Jednak nawet jeśli określony rodzaj wyzwania nie jest w pełni skuteczny, większość przypadkowych atakujących nadal spowoduje, że zrezygnuje i znajdzie aplikację, która nie wykorzystuje tej techniki.

WSKAZÓWKA: Jeśli atakujesz aplikację, która używa kontrolek CAPTCHA w celu utrudnienia automatyzacji, zawsze dokładnie przejrzyj źródło HTML strony, na której pojawia się obraz. Autorzy napotkali przypadki, w których rozwiązanie zagadki pojawia się w formie dosłownej w atrybucie ALT znacznika obrazu lub w ukrytym polu formularza, umożliwiając atakowi skryptowemu pokonanie ochrony bez faktycznego rozwiązania samej zagadki.

Zapobiegaj niewłaściwemu użyciu funkcji zmiany hasła

* Zawsze należy wdrożyć funkcję zmiany hasła, aby umożliwić okresowe wygaśnięcie hasła (jeśli jest to wymagane) oraz aby umożliwić użytkownikom zmianę hasła, jeśli zechcą to zrobić z dowolnego powodu. Jako kluczowy mechanizm bezpieczeństwa należy go dobrze chronić przed nadużyciami.

* Funkcja powinna być dostępna tylko z poziomu uwierzytelnionej sesji.

* Nie powinno być możliwości podania nazwy użytkownika, ani bezpośrednio, ani za pośrednictwem ukrytego pola formularza lub pliku cookie. Użytkownicy nie mają uzasadnionej potrzeby podejmowania prób zmiany haseł innych osób.

* Jako środek dogłębnej obrony, funkcja powinna być chroniona przed nieautoryzowanym dostępem uzyskanym przez inną lukę w zabezpieczeniach aplikacji - taką jak luka w zabezpieczeniach polegająca na przejęciu sesji, skrypty między witrynami, a nawet nienadzorowany terminal. W tym celu użytkownicy powinni być zobowiązani do ponownego wprowadzenia dotychczasowego hasła.

* Nowe hasło należy wprowadzić dwukrotnie, aby uniknąć pomyłek. Aplikacja powinna w pierwszym kroku porównać pola „nowe hasło” i „potwierdź nowe hasło” i zwrócić błąd informacyjny, jeśli się nie zgadzają.

* Funkcja powinna zapobiegać różnym atakom, które można wykonać na główny mechanizm logowania. Pojedynczy ogólny komunikat o błędzie powinien być używany do powiadamiania użytkowników o wszelkich błędach w istniejących poświadczeniach, a funkcja powinna być czasowo zawieszana po niewielkiej liczbie nieudanych prób zmiany hasła.

* Użytkownicy powinni zostać powiadomieni poza pasmem (na przykład za pośrednictwem poczty e-mail), że ich hasło zostało zmienione, ale wiadomość nie powinna zawierać ani starych, ani nowych danych uwierzytelniających.

Zapobiegaj niewłaściwemu użyciu funkcji odzyskiwania konta

* W aplikacjach o największym znaczeniu dla bezpieczeństwa, takich jak bankowość internetowa, odzyskiwanie konta w przypadku zapomnienia hasła odbywa się poza pasmem. Użytkownik musi wykonać połączenie telefoniczne i odpowiedzieć na serię pytań zabezpieczających, a nowe dane uwierzytelniające lub kod reaktywacyjny są również wysyłane poza pasmem (pocztą tradycyjną) na

zarejestrowany adres domowy użytkownika. Większość aplikacji nie chce lub nie potrzebuje tego poziomu bezpieczeństwa, więc funkcja automatycznego odzyskiwania może być odpowiednia.

* Dobrze zaprojektowany mechanizm odzyskiwania hasła musi zapobiegać naruszeniu konta przez nieupoważnioną osobę i minimalizować wszelkie zakłócenia dla legalnych użytkowników.

* Nigdy nie należy używać funkcji takich jak „podpowiedzi" do hasła, ponieważ pomagają one głównie atakującemu przeszukać konta, które mają ustawione oczywiste wskazówki.

* Najlepszym zautomatyzowanym rozwiązaniem umożliwiającym użytkownikom odzyskanie kontroli nad kontami jest wysłanie do użytkownika e-maila z unikalnym, ograniczonym czasowo, niemożliwym do odgadnięcia, jednorazowym adresem URL odzyskiwania. Wiadomość ta powinna zostać wysłana na adres podany przez użytkownika podczas rejestracji. Odwiedzenie adresu URL umożliwia użytkownikowi ustawienie nowego hasła. Po wykonaniu tej czynności należy wysłać drugi e-mail z informacją o dokonaniu zmiany hasła. Aby uniemożliwić atakującemu odmowę świadczenia usług użytkownikom poprzez ciągłe żądanie wiadomości e-mail z reaktywacją hasła, istniejące poświadczenia użytkownika powinny pozostać ważne do czasu ich zmiany.

* Aby dodatkowo zabezpieczyć się przed nieautoryzowanym dostępem, aplikacje mogą stawiać użytkownikom dodatkowe wyzwanie, które muszą wykonać przed uzyskaniem dostępu do funkcji resetowania hasła. Upewnij się, że projekt tego wyzwania nie wprowadza nowych luk w zabezpieczeniach:

* Wyzwanie powinno obejmować to samo pytanie lub zestaw pytań dla wszystkich, wymagane przez aplikację podczas rejestracji. Jeśli użytkownicy podają własne wyzwanie, prawdopodobnie niektóre z nich będą słabe, a to również umożliwia atakującemu wyliczenie prawidłowych kont poprzez zidentyfikowanie tych, które mają ustawione wyzwanie.

* Odpowiedzi na wyzwanie powinny zawierać wystarczającą entropię, aby nie można było ich łatwo odgadnąć. Na przykład pytanie użytkownika o nazwę jego pierwszej szkoły jest lepsze niż pytanie o jego ulubiony kolor.

* Konta powinny być tymczasowo zawieszane po kilku nieudanych próbach ukończenia wyzwania, aby zapobiec atakom siłowym.

* Z aplikacji nie powinno wyciekać żadnych informacji w przypadku nieudanych odpowiedzi na wyzwanie — dotyczących ważności nazwy użytkownika, ewentualnego zawieszenia konta itp.

* Pomyślne ukończenie wyzwania powinno nastąpić przez opisany wcześniej proces, w którym na zarejestrowany adres e-mail użytkownika wysyłana jest wiadomość zawierająca adres URL reaktywacji. W żadnym wypadku aplikacja nie powinna ujawniać zapomnianego hasła użytkownika ani po prostu przenosić użytkownika do uwierzytelnionej sesji. Nawet przejście bezpośrednio do funkcji resetowania hasła jest niepożądane. Odpowiedź na wyzwanie odzyskania konta będzie na ogół łatwiejsza do odgadnięcia przez osobę atakującą niż oryginalne hasło, więc nie należy polegać na niej samej w celu uwierzytelnienia użytkownika.

Rejestruj, monitoruj i powiadamiaj

* Aplikacja powinna rejestrować wszystkie zdarzenia związane z uwierzytelnianiem, w tym logowanie, wylogowanie, zmianę hasła, resetowanie hasła, zawieszenie konta i odzyskanie konta. W stosownych przypadkach należy rejestrować zarówno nieudane, jak i udane próby. Dzienniki powinny zawierać wszystkie istotne szczegóły (takie jak nazwa użytkownika i adres IP), ale nie powinny zawierać tajemnic

bezpieczeństwa (takich jak hasła). Logi powinny być silnie chronione przed nieautoryzowanym dostępem, ponieważ są krytycznym źródłem wycieku informacji.

* Anomalie w zdarzeniach uwierzytelniania powinny być przetwarzane przez funkcję ostrzegania w czasie rzeczywistym i zapobiegania włamaniom. Na przykład administratorzy aplikacji powinni być świadomi wzorców wskazujących na ataki brute-force, aby można było rozważyć odpowiednie środki obronne i ofensywne.

* Użytkownicy powinni być powiadamiani poza pasmem o wszelkich krytycznych zdarzeniach związanych z bezpieczeństwem. Na przykład aplikacja powinna wysyłać wiadomość na zarejestrowany adres e-mail użytkownika za każdym razem, gdy zmieni on swoje hasło.

* Użytkownicy powinni być powiadamiani w paśmie o często występujących zdarzeniach związanych z bezpieczeństwem. Na przykład po udanym logowaniu aplikacja powinna informować użytkowników o czasie i źródłowym IP/domenie ostatniego logowania oraz liczbie nieudanych prób logowania wykonanych od tego czasu. Jeśli użytkownik zostanie poinformowany, że jego konto jest celem ataku mającego na celu odgadnięcie hasła, jest bardziej prawdopodobne, że będzie często zmieniał swoje hasło i ustawiał je na silną wartość.

Streszczenie

Funkcje uwierzytelniania są prawdopodobnie najbardziej widocznym celem ataków typowej aplikacji. Z definicji mogą do nich dotrzeć nieuprzywilejowani, anonimowi użytkownicy. W przypadku uszkodzenia zapewniają dostęp do chronionych funkcji i poufnych danych. Leżą one u podstaw mechanizmów bezpieczeństwa używanych przez aplikację do samoobrony i stanowią pierwszą linię obrony przed nieautoryzowanym dostępem. Mechanizmy uwierzytelniania w świecie rzeczywistym zawierają niezliczone wady projektowe i implementacyjne. Skuteczny atak na nich musi przebiegać systematycznie, przy użyciu ustrukturyzowanej metodologii, aby przepracować każdą możliwą drogę ataku. W wielu przypadkach pojawiają się otwarte cele - złe hasła, sposoby na znalezienie nazw użytkowników, podatność na ataki typu brute-force. Na drugim końcu spektrum defekty mogą być bardzo trudne do wykrycia. Mogą wymagać skrupulatnego zbadania zawiłego procesu logowania, aby ustalić przyjęte założenia i pomóc ci dostrzec subtelny błąd logiczny, który można wykorzystać, aby przejść przez drzwi. Najważniejszą lekcją podczas atakowania funkcji uwierzytelniania jest szukanie wszędzie. Oprócz głównego formularza logowania mogą istnieć funkcje umożliwiające rejestrację nowych kont, zmianę haseł, zapamiętywanie haseł, odzyskiwanie zapomnianych haseł oraz podszywanie się pod innych użytkowników. Każdy z nich stanowi bogaty cel potencjalnej wady, a problemy, które zostały świadomie wyeliminowane w ramach jednej funkcji, często pojawiają się ponownie w innych. Poświęć czas na zbadanie i zbadanie każdego centymetra powierzchni ataku, jaki możesz znaleźć, a Twoje nagrody mogą być wspaniałe.

Pytania

1. Podczas testowania aplikacji internetowej logujesz się przy użyciu poświadczeń joe i pass. Podczas procesu logowania w przechwytującym serwerze proxy pojawia się prośba o następujący adres URL:

http://www.wahh-app.com/app?action=login&uname=joe&password=pass

Jakie trzy luki w zabezpieczeniach możesz zdiagnozować bez dalszych badań?

2. W jaki sposób funkcje samorejestracji mogą wprowadzać luki w wyliczaniu nazw użytkowników? Jak można zapobiegać tym lukom w zabezpieczeniach?

3. Mechanizm logowania obejmuje następujące kroki:

(a) Aplikacja żąda podania nazwy użytkownika i kodu dostępu.

(b) Aplikacja żąda dwóch losowo wybranych liter z niezapomnianego słowa użytkownika.

Dlaczego wymagane informacje są wymagane w dwóch oddzielnych krokach? Jaką wadę zawierałby mechanizm, gdyby tak nie było?

4. Wieloetapowy mechanizm logowania najpierw żąda nazwy użytkownika, a następnie różnych innych elementów na kolejnych etapach. Jeśli któryś z dostarczonych elementów jest nieważny, użytkownik natychmiast wraca do pierwszego etapu. Co jest nie tak z tym mechanizmem i jak można naprawić tę lukę?

5. Aplikacja zawiera mechanizm antyphishingowy w swojej funkcjonalności logowania. Podczas rejestracji każdy użytkownik wybiera określony obraz z dużego banku zapadających w pamięć obrazów, które przedstawia mu aplikacja. Funkcja logowania obejmuje następujące kroki:

(a) Użytkownik wprowadza swoją nazwę użytkownika i datę urodzenia.

(b) Jeśli te dane są prawidłowe, aplikacja pokazuje użytkownikowi wybrany przez nią obraz; w przeciwnym razie wyświetlany jest losowy obraz.

(c) Użytkownik weryfikuje, czy wyświetlany jest właściwy obraz. Jeśli tak, wpisuje swoje hasło.

Ideą tego mechanizmu antyphishingowego jest umożliwienie użytkownikowi potwierdzenia, że ma do czynienia z autentyczną aplikacją, a nie jej klonem, ponieważ tylko prawdziwa aplikacja zna prawidłowy obraz do wyświetlenia użytkownikowi. Jaką lukę wprowadza ten mechanizm antyphishingowy do funkcji logowania? Czy mechanizm skutecznie zapobiega phishingowi?