PISANIE BEZPIECZNEGO KODU

WPROWADZENIE.

Tematu bezpiecznego kodowania nie da się odpowiednio omówić w jednym rozdziale. Niestety programy są z natury trudne do zabezpieczenia ze względu na dużą liczbę sposobów, w jakie wykonanie może przejść przez kod w wyniku różnych sekwencji wejściowych i wartości danych. Dodatkowa presja na wprowadzenie aplikacji na rynek może również odbić się na bezpiecznym procesie kodowania. Ten rozdział zawiera punkt wyjścia i dodatkowe zasoby dla specjalistów ds. bezpieczeństwa, architektów systemów i programistów, aby stworzyć skuteczną i bezpieczną metodologię rozwoju. Pisanie bezpiecznego kodu wymaga koordynacji i współpracy różnych obszarów funkcjonalnych w organizacji i może wymagać fundamentalnych zmian w obecnym sposobie projektowania, pisania, testowania i wdrażania oprogramowania, a także zmiany kultury korporacyjnej w zakresie tworzenia oprogramowania .

KWESTIE POLITYKI I ZARZĄDZANIA

Osoby piszące kod i rozwijające oprogramowanie napotykają niezliczone przeszkody w zakresie bezpieczeństwa. Obecnie zależność od niezawodności i bezpieczeństwa zautomatyzowanego systemu jest prawie całkowita. Dla coraz większej liczby organizacji rozproszone procesy informacyjne, realizowane za pośrednictwem środowisk sieciowych, stały się krytycznym elementem operacyjnym ich działalności. Nie tylko system przetwarzania musi działać w razie potrzeby, ale także przetwarzane informacje musi zachowywać swoją integralność, aby można było jej zaufać w użyciu. Ze względu na ogólny brak podstawowej dyscypliny organizacyjnej IT oraz przestrzegania podstawowych zasad rozwoju i konserwacji oprogramowania i systemów, większość wysiłków na rzecz rozwoju oprogramowania była w najlepszym razie znacznie słabsza. Jeśli te same nieodpowiednie praktyki pozostaną niezmienione w obliczu narastających zagrożeń cybernetycznych, będą one nadal przyczyniać się do powstawania niezabezpieczonych systemów jutra, tak jak przyczyniły się do powstania niezabezpieczonych systemów dzisiaj. Podstawowym problemem leżącym u podstaw tego rodzaju twierdzenia jest trudność w uzasadnieniu tego, co często postrzegane są jako działania niegenerujące przychodów oraz konieczność wprowadzenia produktu na rynek w terminie lub przed terminem dostawy. Trudność wynika również z niemożności udowodnienia negatywu. Na przykład, nie można udowodnić niezaangażowanemu lub niewzruszonemu obserwatorowi, że czas, pieniądze i zasoby ludzkie poświęcone na zapobieganie są dobrze wydawane tylko dlatego, że uniknięto katastrofy lub incydentu. Taki obserwator przyjmie fakt, że nie wystąpiło to jako dowód, że taka możliwość nigdy tak naprawdę nie istniała lub że problem był wyolbrzymiony. W ten sam sposób sceptyczny obserwator może uznać brak naruszeń bezpieczeństwa za dowód, że takie zagrożenie nie istniało lub że nigdy nie było tak złe, jak twierdzi. Problem nasila się po poświęceniu czasu i wysiłku na ustanowienie zasad bezpieczeństwa, kontroli bezpieczeństwa i mechanizmów egzekwowania. W obu przypadkach pieniądze, czas i wysiłek włożony w zapobieganie negatywnym skutkom przetwarzania danych, które mogłyby zaszkodzić organizacji, są postrzegane z podejrzliwością i nie kwalifikują się na poziomie uzasadniającym zmianę harmonogramu dostarczania lub finansowania. Jednak takie podejrzenia nie są niczym nowym w świecie przetwarzania informacji. W historii IT takie podejście istniało w odniesieniu do działań zapobiegających problemom, takich jak jakość zapewnienie, testowanie kontroli jakości, dokumentacja, zarządzanie konfiguracją, zarządzanie zmianami i inne kontrole zarządzania systemem zaprojektowane w celu poprawy jakości, niezawodności i łatwości konserwacji systemu. Jeśli uznasz, że analogia tworzenia aplikacji jest podobna do tego, jak buduje się dom, znacznie trudniej jest coś zaimplementować i zintegrować po fakcie. Bezpieczeństwo nie jest i nie powinno być traktowane jako powierzchowne ćwiczenie podobne do wieszania zasłon lub wymiany żarówki. Przypomina to instalację wodno-kanalizacyjną w domu i należy ją wdrożyć, mając to na

uwadze. Problemy z bezpieczeństwem to problemy z zarządzaniem. Zarząd określa, ile czasu i pieniędzy można przeznaczyć na działania związane z bezpieczeństwem, takie jak zarządzanie ryzykiem i testowanie oprogramowania. Ile czasu i energii zainwestowano w projektowanie, budowanie, testowanie i naprawianie oprogramowania, to w końcu rozmowa z kierownictwem. Dlatego głównym problemem tych, którzy dążą do poprawy bezpieczeństwa oprogramowania, musi być naprawa tych samych podstawowych i fundamentalnych wad zarządzania IT; w przeciwnym razie nigdy nie można osiągnąć ani utrzymać skutecznego bezpieczeństwa.

Oprogramowanie Total Quality Management

Istotą wyzwań związanych z bezpieczeństwem komputerów są dynamicznie zmieniające się podatności i zagrożenia, oparte na szybko zmieniających się technologiach, wykorzystywanych w coraz bardziej złożonych aplikacjach biznesowych. Organizacje muszą budować oprogramowanie wokół idei modelu ciągłego doskonalenia, spopularyzowanego przez kompleksowe zarządzanie jakością (TQM) wyrażone w rodzinie norm ISO 9000. W celu poprawy jakości model ten był postrzegany jako ciągły cykl planu-wykonaj-sprawdź-i-działaj. Ze względu na bezpieczeństwo oprogramowania można go traktować jako cykl planowania, naprawy, monitorowania i oceny. Proces zarządzania bezpieczeństwem w celu realizacji tego modelu musi zostać ustanowiony i zintegrowany z codzienną działalnością firmy. Integracja oznacza, że inicjatywa bezpieczeństwa nie jest samodzielna jako strażnik lub działa jedynie jako funkcja audytu i musi odejść od funkcji policyjnej. Bezpieczeństwo należy traktować jako istotną część wszystkich innych działań związanych z rozwojem i zarządzaniem systemami, w tym definiowaniem wymagań, projektowaniem systemów, programowaniem, integracją pakietów, testowaniem, dokumentacją systemów, szkoleniem, zarządzaniem konfiguracją, obsługą i konserwacją. Bezpieczeństwo należy postrzegać jako dobrą architekturę. Innymi słowy, bezpieczeństwo musi być częścią całego cyklu rozwoju i bram zarządzania projektem. Proces zarządzania bezpieczeństwem musi być zaprojektowany w taki sposób, aby czynności planowania, naprawiania, monitorowania i oceny były wykonywane w sposób iteracyjny dla systemów, które są definiowane, projektowane i wdrażane oraz dla systemów, które są obecnie wykonywane na co dzień. dzienne operacje biznesowe. W wielu organizacjach nierzadko zdarza się, że oprogramowanie jest budowane bez zwracania uwagi na kwestie bezpieczeństwa. W rezultacie wiele aplikacji o znaczeniu krytycznym dla firmy weszło do pełnej produkcji z nieodpowiednimi lub nieistniejącymi zabezpieczeniami. Na przykład szybko zbudowana aplikacja, która ma na celu zaspokojenie początkowej potrzeby śledzenia informacji w bazie danych za pośrednictwem interfejsu internetowego, może nie zawierać wszystkich konfiguracji kontroli dostępu zaprojektowanych w aplikacji, które są potrzebne do zapewnienia odpowiedniego bezpieczeństwa. Służy szybkiemu celowi, ale dzięki swojemu sukcesowi staje się narzędziem długoterminowym. Ze względu na charakter działalności (przechodzenie od jednego zadania do drugiego) tak naprawdę nie robi się nic, aby przekształcić go w funkcję, którą teraz wypełnia. Ponadto im dłużej aplikacja pozostaje w użyciu, tym większe prawdopodobieństwo, że pierwotni programiści przejdą na migrację, a wszelka wiedza historyczna lub dokumentacja zostaną utracone. Sprawia to, że możliwość aktualizacji lub poprawiania zabezpieczeń aplikacji jest mniej pożądana. W rezultacie użytkownicy lub hakerzy mogą w sposób niezamierzony lub celowo uzyskać dostęp i modyfikować krytyczne informacje w bazie danych lub w samym systemie. Na szczęście ten trend zaczął się zmieniać, ale nadal wymaga szczególnej uwagi. Ponieważ takie systemy już działają, jedynym sposobem realnej identyfikacji i oceny ich zagrożeń bezpieczeństwa poza całkowitym zastąpieniem aplikacji aplikacją, która spełnia wymagania bezpieczeństwa, jest rozpoczęcie agresywnego monitorowania pod kątem włamań, a następnie zaprojektowanie i wdrożenie korygujących polityk i kontroli bezpieczeństwa na podstawie tego, co zostało odkryte. Efektywna sekwencja działań bezpieczeństwa dla systemów już w stanie operacyjnym

będzie polegała na monitorowaniu bieżących operacji, ocenie podejrzanej aktywności, planowaniu polityki naprawczej i kontroli technologii, wdrażaniu kontroli w systemie, a następnie monitorowaniu

Należyta staranność

Aby skutecznie zająć się oprogramowaniem i bezpieczeństwem kodu, musi ono stać się problemem dla kierownictwa, podobnie jak inne problemy związane z bezpieczeństwem zaczęły wymagać uwagi kierownictwa. Dyrektorzy biznesowi i menedżerowie IT są teraz zbyt zaznajomieni z pojęciem należytej staranności w odniesieniu do zastosowań technologii informacyjnej. Ze względu na potencjalne skutki prawne konieczne stało się obejrzenie działania związanego z IT poprzez definicje należytej staranności i rozsądnej staranności. Znaczenie pojęć należytej staranności i rozsądnej staranności polega na tym, że pozwalają one na ewolucję miernika, z którym można porównywać rozważania, decyzje i działania organizacji dotyczące bezpieczeństwa aplikacji. Porównanie dotyczy zwykle podobnej organizacji, w podobnych okolicznościach wrażliwości i zagrożenia oraz z podobnymi przewidywalnymi negatywnymi skutkami dla klientów, partnerów, udziałowców, pracowników i opinii publicznej. Na przykład, jeśli jedna firma stosuje kontrolę bezpieczeństwa i nie doświadcza żadnych naruszeń bezpieczeństwa, którym ta technika miała zapobiegać, fakt ten można wykorzystać do ustalenia punktu odniesienia, z którym można by porównać inne podobne organizacje. Jeśli wystarczająco dużo podobnych organizacji zastosuje tę samą technikę, kontrola bezpieczeństwa może zostać sklasyfikowane jako najlepsze praktyki dla tej branży. Jeżeli jednak inna firma z tej samej branży nie zatrudniła zabezpieczenia kontroli i czy doświadczyło naruszeń bezpieczeństwa typu, któremu technika miała zapobiegać, może to wyraźnie wskazywać na brak należytej staranności lub należytej staranności. Decyzje i wdrożenia dotyczące bezpieczeństwa oprogramowania nie są jednorazowe, ale muszą podlegać ciągłemu procesowi oceny ryzyka, zarządzania i doskonalenia. Decyzje zaczynają się od wyboru języka programowania i od tego momentu są kontynuowane. Wybór języka interpretowanego lub skompilowanego może mieć wpływ nie tylko na wymagane umiejętności programistyczne i ramy czasowe, ale także na sposób korzystania z bibliotek językowych i wpływ, jaki przyszłe aktualizacje mogą osiągnąć klient. Dobrym przykładem są niedawne problemy z bezpieczeństwem, które nękają produkty Java i Microsoft. Dlatego konieczne jest, aby wykazać kierownictwu zdolność do nieprzerwanej należytej staranności, ustanowić udokumentowany program zarządzania ryzykiem bezpieczeństwa komputerowego, niezależnie od wewnętrznego lub zewnętrznego rozwoju oprogramowania, oraz zintegrować go z ogólnymi procesami zarządzania rozwojem oprogramowania. proces. Nic innego nie pokaże, że firma i jej zarząd są w stanie ocenić zagrożenia bezpieczeństwa komputerowego i działać w rozsądny sposób. W przypadku twórców oprogramowania pracujących nad oprogramowaniem komercyjnym przeznaczonym do sprzedaży konkretnemu klientowi w ramach umowy, a nawet szerokiemu gronu klientów objętych umowami licencyjnymi użytkownika końcowego, zachowanie należytej staranności i zwiększone bezpieczeństwo może zmniejszyć ryzyko pozwów sądowych o odszkodowanie za zaniedbanie . Niezależnie od tego, czy są udane, czy nie, takie procesy nigdy nie są pozytywną reklamą dla twórców oprogramowania.

Rozważania dotyczące przepisów i zgodności

Kwestie dotyczące przepisów i zgodności stały się istotnym aspektem dla większości firm. Dodatkowy aspekt tworzenia oprogramowania zwiększa potrzebę bycia świadomym wpływu, jaki stanowe, federalne lub międzynarodowe przepisy mogą mieć na projekt programistyczny. Konieczne może być również rozważenie zgodności z branżami pozarządowymi. Jeśli organizacja podlega określonym regulacjom (np. ustawodawstwu Sarbanes-Oxley), wówczas dokumentacja napotkanych błędów i wynikających z nich wewnętrznych działań sprawozdawczych i naprawczych ma kluczowe znaczenie. Błąd powinien jasno wskazywać, w jaki sposób został zidentyfikowany, kto go zidentyfikował, w jaki sposób został zgłoszony kierownictwu, jaka będzie remediacja i kiedy przewiduje się jej zakończenie.

Bez tych szczegółów kierownictwo może napotkać znaczne trudności w potwierdzeniu, że istnieją odpowiednie mechanizmy kontroli wewnętrznej, wraz z odpowiednim i odpowiednim zaangażowaniem kierownictwa. Ponadto błąd może skutkować zidentyfikowaniem przez audytorów zewnętrznych słabości kontroli lub, co gorsza, jako słabości istotnej (istotną słabością byłoby użycie języka programowania, który ma znaną nieodłączną wadę, a nie innego, bardziej odpowiedniego i bez skazy, podobnie jak w przypadku mostu z aluminium zamiast stali), w zależności od jego charakteru i surowości. Dodatkowo, gdy zostaną zauważone błędy, które mają wpływ na systemy lub aplikacje multilokacyjne, znaczenie i istotność - to znaczy zakres, w jakim błędy wpływają lub mogą wpływać na system lub jakikolwiek inny system, z którym interfejsy aplikacji - muszą być rozpatrywane z punktu widzenia zarówno filii, jak i lokalizacji centrali. Ponieważ przepisy różnią się w poszczególnych stanach i krajach, to, co można uznać za prawnie dopuszczalne standardy prywatności lub rachunkowości w jednym miejscu, może nie być akceptowalne w innym miejscu. W rezultacie, staranne rozważenie wymagań i zamierzonych zastosowań lub klientów musi być ustalone w ramach procesów inicjowania projektu i należytej staranności.

KWESTIE TECHNICZNE I PROCEDURALNE

Bezpieczeństwo powinno być zintegrowane na każdym etapie cyklu życia aplikacji. Obejmuje to analizę wymagań, etap projektowania oprogramowania i jądro bezpieczeństwa systemu operacyjnego, opracowywanie polityki korporacyjnej oraz programy uświadamiające, szkoleniowe i edukacyjne. Pisanie bezpiecznego kodu może być trudnym i technicznie trudnym przedsięwzięciem. Nie oznacza to, że nie można tego zrobić, ale wymaga to od dewelopera współpracy z resztą zespołu projektowego i innymi kluczowymi interesariuszami w celu sprostania wyzwaniom, które należy zdefiniować, uzgodnić i przezwyciężyć przed wydaniem oprogramowanie lub system. Dodatkowy nacisk na rynek, który może napędzać proces szybkiego rozwoju, może również mieć negatywny wpływ na bezpieczeństwo aplikacji. Problem należy rozwiązać na dwa sposoby, techniczny i proceduralny. Z technicznego punktu widzenia programiści muszą być świadomi pułapek związanych z tworzeniem aplikacji i ich unikać. Pod względem proceduralnym zespół programistów i organizacja muszą przestrzegać spójnej metodologii rozwoju. Ta spójność musi również obejmować identyfikację zagrożeń bezpieczeństwa na każdym etapie procesu, w tym analizę wymagań.

Analiza wymagań

Bezpieczeństwo musi być częścią analizy wymagań w każdym projekcie deweloperskim. Większość twórców aplikacji wie, że dodanie zabezpieczeń po fakcie zwiększa koszty i czas oraz staje się bardziej skomplikowane. Wymagania mogą pochodzić z różnych źródeł:

* Potrzeby funkcjonalne systemu (w tym system operacyjny, na którym aplikacja będzie działać)

* Normy lub wytyczne krajowe, międzynarodowe lub organizacyjne

* Ograniczenia prawne

* Poziom czułości danych

* Istniejące polityki bezpieczeństwa

* Analiza kosztów i korzyści

* Ocena ryzyka

Celem analizy jest określenie, jakie informacje i procesy są potrzebne do obsługi pożądanych celów i funkcji oprogramowania oraz modelu biznesowego.

Projekt

Dane informacyjne zebrane podczas zbierania analizy trafiają do projektu oprogramowania jako wymagania. Wynikiem analizy są dane, logika i projekt proceduralny. Deweloperzy biorą dane i dane modelu informacyjnego i przekształcają je w struktury danych, które będą wymagane do wdrożenia oprogramowania. Projekt logiki definiuje relacje między głównymi strukturami i komponentami aplikacji. Projekt proceduralny przekształca elementy konstrukcyjne w procedury opisowe. Wybierane są mechanizmy kontroli dostępu, określane są prawa i uprawnienia, oceniane są wszelkie inne specyfikacje bezpieczeństwa i ustalane są rozwiązania. Struktura podziału pracy obejmuje etapy rozwoju i wdrożenia. Struktura obejmuje harmonogram i szczegółowe czynności związane z testowaniem, rozwojem, etapami, testami integracyjnymi i dostarczaniem produktów. Decyzje podjęte w fazie projektowania mają kluczowe znaczenie dla rozwoju aplikacji. Projekt jest jedynym sposobem, w jaki wymagania są przekładane na komponenty oprogramowania. W ten sposób projektowanie oprogramowania jest podstawą procesu rozwoju i ma ogromny wpływ na jakość i konserwację oprogramowania. Jeśli dobry projekt produktu nie zostanie wdrożony na początku, reszta procesu rozwoju będzie o wiele trudniejsza.

System operacyjny

Jądro bezpieczeństwa systemu operacyjnego jest odpowiedzialne za egzekwowanie zasad bezpieczeństwa w systemie operacyjnym i aplikacji. W związku z tym architektura systemu operacyjnego jądra jest zwykle warstwowa, z jądrem w warstwie najbardziej uprzywilejowanej. Jest to niewielka część systemu operacyjnego, ale wszystkie odniesienia do informacji i zmiany w autoryzacji przechodzą przez jądro. Aby być bezpiecznym jądro musi spełniać trzy podstawowe kryteria:

1. Kompletność. Cały dostęp do informacji musi przejść przez jądro.

2. Izolacja. Jądro musi być chronione przed nieautoryzowanym dostępem.

3. Weryfikowalność. Należy udowodnić, że jądro spełnia specyfikacje projektowe.

Jak wspomniano, jądro działa w najbardziej uprzywilejowanej warstwie. Większość systemów operacyjnych ma dwa tryby dostępu do procesora, użytkownika i jądra. Ogólny kod aplikacji działa w trybie użytkownika, podczas gdy system operacyjny działa w trybie jądra. Tryb jądra umożliwia procesorowi pełny dostęp do całej pamięci systemowej i instrukcji procesora. Gdy aplikacje nie są napisane bezpiecznie, to rozdzielenie trybów może zostać naruszone, umożliwiając wykorzystanie luk w zabezpieczeniach za pomocą dowolnego kodu, przepełnienia bufora i innych technik.

Najlepsze praktyki i wytyczne

„Najlepsze praktyki" to termin, który może wywołać ożywioną debatę, zwłaszcza dotyczącą bezpieczeństwa. Ogólnie rzecz biorąc, najlepsze praktyki, a szczególnie w odniesieniu do bezpieczeństwa, często padają ofiarą dogmatycznego, a czasem ślepego oddania bezsensownym praktykom, które mają niewiele wspólnego z bezpieczeństwem, a więcej z wiarą lub tradycją. Niezależnie od tego, najlepsze praktyki pomagają zapewnić zestaw wskazówek, które pomagają zapewnić strukturę. Ponadto najlepsze praktyki można dostosować, aby pomóc w zaspokojeniu potrzeb konkretnej sytuacji. Przykładem może być specjalna seria 800 publikacji National Institute of Standards and Technology (NIST). Seria ta zawiera najlepsze praktyki i zalecenia, które można dostosować lub zintegrować z innymi praktykami, aby pomóc w poprawie i wyeliminowaniu tradycji określonej praktyki, która może nie mają już uzasadniony użytek. Biorąc to pod uwagę, większość ogólnych podręczników bezpieczeństwa zawiera zalecenia dotyczące aspektów programowania związanych z bezpieczeństwem — patrz na przykład Stallings — które w rzeczywistości mają bardzo

realną korzyść w tworzeniu bezpieczniejszego oprogramowania. Oprócz projektowania zabezpieczeń w systemie od samego początku, istnieją również oczywiste wskazówki, które pomogą w tworzeniu bezpiecznego oprogramowania:

* Narzucaj silną identyfikację i uwierzytelnianie (I&A) dla krytycznych i wrażliwych systemów oprócz I&A dostępnego z systemu operacyjnego; najlepiej użyć uwierzytelniania opartego na tokenie lub uwierzytelniania biometrycznego jako części fazy inicjowania aplikacji.

* Dokładnie udokumentuj swój kod, w tym za pomocą słowników danych dla pełnej definicji dopuszczalnych danych wejściowych i wyjściowych funkcji oraz dopuszczalnego zakresu i typu wartości dla wszystkich zmiennych.

* Podczas przechowywania wrażliwych używaj zmiennych lokalnych, a nie zmiennych globalnych

dane, które powinny być używane tylko w ramach określonej procedury (tj. użyj architektury stosu procesów, aby ograniczyć nieumyślny lub nieautoryzowany dostęp do danych w stosie).

* Ponownie zainicjuj tymczasowe przechowywanie natychmiast po ostatnim prawidłowym użyciu zmiennej, co utrudnia oczyszczanie złoczyńców.

* Ogranicz funkcjonalność w określonym module do tego, co jest wymagane dla określonej pracy (np. nie używaj tego samego modułu do funkcji nadzorczych i rutynowych funkcji wykonywanych przez personel biurowy).

* Zdefiniuj widoki danych w bazach danych, które są zgodne z wymaganiami funkcjonalnymi i ograniczają dostęp do danych wrażliwych (np. widok danych z bazy danych dokumentacji medycznej powinien wykluczać identyfikatory pacjentów, gdy pracownik działu finansowego korzysta z bazy danych do agregacji statystycznej).

* Używaj silnego szyfrowania (nie szyfrowania własnego), które ma standardowe w branży procedury do ochrony poufnych i krytycznych danych na dysku. Lokalnie opracowane, domowe szyfrowanie na ogół nie jest tak bezpieczne.

* Nie zezwalaj programistom na dostęp do produkcyjnych baz danych.

* Randomizuj lub w inny sposób maskuj dane wrażliwe podczas generowania podzbiorów testowych z danych produkcyjnych.

* Użyj monitorów pokrycia testów, aby zweryfikować, czy wszystkie sekcje kodu źródłowego są rzeczywiście wykonywane podczas testów zapewnienia jakości; zbadać funkcje kodu, który nigdy nie jest wykonywany.

* Zintegruj możliwość logowania ze wszystkimi aplikacjami w celu debugowania, odzyskiwania danych po awariach w trakcie transakcji oraz w celach bezpieczeństwa, takich jak analiza kryminalistyczna.

* Utwórz rekordy pliku dziennika, które zawierają kryptograficznie dźwiękowy kod uwierzytelniania wiadomości (MAC), który sam zawiera adres MAC poprzedniego rekordu jako dane wejściowe algorytmu; ta technika zapewnia, że sfałszowanie lub zmodyfikowanie pliku dziennika będzie trudniejsze dla złoczyńcy.

* Rejestruj wszystkie inicjacje procesów dla programu i rejestruj zakończenie procesu; zawierać pełne informacje o tym, kto załadował program lub moduł.

* Rejestruj wszystkie modyfikacje rekordów i opcjonalnie udostępniaj również rejestrowanie w celu odczytu.

* Użyj blokowania na poziomie rekordu, aby zapobiec przypadkowemu nadpisaniu danych w rekordach, do których uzyskuje się dostęp jednocześnie. Pamiętaj, aby odblokować sekwencję blokad w odwrotnej kolejności sekwencji blokad, aby zapobiec zakleszczeniu. (Tak więc, jeśli zablokujesz zasób A, B i C w tej kolejności, odblokuj C, następnie B, a następnie A.)

* Podpisz swój kod źródłowy za pomocą podpisów cyfrowych.

* Używaj sum kontrolnych w produkcyjnych plikach wykonywalnych, aby utrudnić ukrycie nieautoryzowanych modyfikacji.

Mike Gerdes, były menedżer w AtomicTangerine, przedstawił te sugestie w trakcie dyskusji:

* Poleć czytelnikom przyjęcie praktyki projektowania kodu w bardziej holistyczny sposób. Powszechną praktyką jest pisanie i testowanie procedur w sposób, który weryfikuje, jak kod przetwarza dane w zamierzony sposób. Aby uniknąć skutków złośliwego kodu i ataków z wprowadzaniem danych, programista musi również napisać kod, który zajmuje się tym, co nie powinno być przetwarzane. Bardziej kompletna metodologia projektowania obejmowałaby również testowanie wszystkich informacji przychodzących, aby zapewnić wykluczenie wszelkich danych, które nie spełniają wymagań dotyczących akceptowalnych danych. Ta metoda powinna być stosowana w aplikacjach o wysokim ryzyku i tych, które mają wyjątkowo żmudny cykl testowy, i wyeliminuje wiele powszechnie stosowanych obecnie metod ataku.

* Ustal kryteria określania poziomu wrażliwości informacji zawartych lub przetwarzanych przez aplikację i podprogramy.

* Jeśli nie są one jeszcze obecne, rozważ wdrożenie formalnych procedur kontrolnych w metodologii programowania oprogramowania, aby upewnić się, że wszystkie dane są przeglądane podczas procesów zapewniania jakości oraz są sklasyfikowane i obsługiwane odpowiednio do przypisanego poziomu.

* Zidentyfikuj i uwzględnij wszelkie obowiązkowe cechy systemu operacyjnego i bezpieczeństwa sieci dla systemu produkcyjnego w specyfikacjach oprogramowania. Oprócz dostarczenia zespołom programistycznym i QA pewnej definicji środowiska, w którym ma działać oprogramowanie, dając administratorowi i użytkownikom końcowym wyobrażenie o tym, jakie były Twoje oczekiwania podczas tworzenia kodu, może być niezwykle przydatne w określaniu, gdzie oprogramowanie może, lub nie może być używany.

* W stosownych przypadkach zweryfikuj podpisy cyfrowe procedur przetwarzających poufne dane, gdy kod jest ładowany do wykonania.

* Jeśli uwzględniasz sumy kontrolne plików wykonywalnych dla kodu produkcyjnego, dołącz procedury, które weryfikują sumy kontrolne przy każdym ponownym uruchomieniu systemu.

Zespół Carnegie-Mellon Software Engineering Institute Computer Emergency Response Team (CERT) zapewnia dobrą listę „Top 10 bezpiecznych praktyk kodowania". Poniższa lista zawiera kilka dodatkowych elementów, które należy wziąć pod uwagę oprócz elementów wymienionych wcześniej:

* Sprawdź poprawność danych wejściowych ze źródeł danych. Właściwa walidacja danych wejściowych może wyeliminować wiele luk w zabezpieczeniach oprogramowania.

* Skompiluj kod przy użyciu najwyższego poziomu ostrzeżeń dostępnego dla kompilatora i wyeliminuj ostrzeżenia, modyfikując kod.

* Utwórz architekturę oprogramowania i zaprojektuj oprogramowanie w celu wdrożenia i egzekwowania zabezpieczeń. Carnegie-Mellon ma wstępnie ustalony standard programowania według języka, który można znaleźć na stronie www.cert.org/secure-coding/scstandards.html

* Pamiętaj o zasadzie K-I-S-S (Keep It Simple, Stupid). Złożone projekty zwiększają prawdopodobieństwo wystąpienia błędów lub luk w zabezpieczeniach oraz potrzebę zwiększenia złożoności mechanizmów bezpieczeństwa lub wynikających z nich środków łagodzących.

* Opieraj decyzje o dostępie na pozwoleniu, a nie na wykluczeniu, aby przestrzegać zasady najmniejszych uprawnień.

* Oczyszczanie danych wysyłanych do innych systemów, takich jak powłoki poleceń, relacyjne bazy danych i gotowe komponenty komercyjne (COTS).

* Przećwicz dogłębną obronę w aplikacji, tak aby jeśli jedna warstwa ochrony okazała się niewystarczająca, inna może zapobiec lub ograniczyć lukę w zabezpieczeniach przed przekształceniem się w lukę, którą można wykorzystać.

* Ustanowienie dobrych technik zapewniania jakości, aby skuteczniej identyfikować i eliminować luki w zabezpieczeniach.

* Opracuj i/lub zastosuj bezpieczny standard kodowania dla docelowego języka i platformy programistycznej.

* Zdefiniuj wymagania bezpieczeństwa na początku projektu.

* Myśl jak haker i korzystaj z modelowania zagrożeń, aby przewidywać zagrożenia, na jakie może być narażone oprogramowanie. Jeśli zastanowisz się, jak można go zaatakować, będziesz w stanie zapobiec atakowi, zanim się pojawi.

Języki

Do chwili obecnej żadne popularne języki komputerowe nie mają wbudowanych funkcji zabezpieczeń. Java zawiera postanowienia dotyczące ograniczania dostępu do zasobów poza „piaskownicą" zarezerwowanych dla procesu, jak opisano w książkach Felten i McGraw. Niemniej jednak Java jest nadal jednym z języków będących celem ataku wielu implementacji szkodliwego oprogramowania. PASCAL wykorzystuje silne typowanie i wymaga pełnego zdefiniowania struktur danych, co utrudnia dostęp do danych i kodu poza maszyną wirtualną zdefiniowaną dla danego procesu. W przeciwieństwie do tego, C i C++ pozwalają programistom na dostęp do dowolnego regionu pamięci w dowolnym momencie, gdy system operacyjny na to pozwala. Języki komputerowe pozwalają programistom pisać kod tak dobrze, jak potrafią. Języki z silną typizacją mogą stwarzać większe ograniczenia programistom, ale zasadniczym wymogiem jest to, aby programiści nadal myśleli o bezpieczeństwie podczas projektowania i kompilacji kodu. Istnieje kilka zestawów narzędzi bezpieczeństwa i zasobów dostępnych dla programistów; na przykład RSA ma wiele zestawów narzędzi kryptograficznych. Niektóre podręczniki (np. Schneier's Applied Cryptography) zawierają płyty CD-ROM z przykładowym kodem. Ponadto Centrum Koordynacji Zespołu Reagowania na Awarie Komputerowe (CERT/CC) zostało uruchomione w grudniu 1988 r. przez Agencję Zaawansowanych Projektów Badawczych Obrony (DARPA), która była częścią Departamentu Obrony USA. W CERT/CC znajduje się Software Engineering Institute, ośrodek badawczy finansowany ze środków federalnych, prowadzony przez Carnegie Mellon University. CERT/CC bada luki w zabezpieczeniach Internetu, świadczy usługi dla zaatakowanych stron internetowych i publikuje alerty bezpieczeństwa. Działalność badawcza CERT/CC obejmuje obszar obliczeń w sieci WAN i rozwijanie ulepszonego bezpieczeństwa w Internecie. Firma

Carnegie-Mellon ustanowiła bezpieczne standardy kodowania oparte na języku, które mogą pomóc w ustaleniu jednolitego zestawu zasad i wytycznych w całej grupie programistów. Niezależnie od języka wybranego do tworzenia oprogramowania, ważne jest, aby podjąć decyzję w oparciu o mocne i słabe strony języka. Oznacza to, że analiza ryzyka musi być częścią projektowania każdego projektu programistycznego, w tym wyboru języka programistycznego. Dzięki zrozumieniu tych mocnych i słabych stron, można podejmować lepsze decyzje, aby zapobiec nieumyślnemu ujawnieniu luk w zabezpieczeniach języka podczas opracowywania.

RODZAJE BŁĘDÓW OPROGRAMOWANIA.

Nowi programiści powinni przejrzeć zakres głównych przyczyn błędów oprogramowania. Taki przegląd jest szczególnie przydatny dla studentów, którzy ukończyli szkolenie, które nie obejmowało omówienia metodologii systematycznego zapewniania jakości. Kolejne sekcje mogą służyć jako podstawa do tworzenia efektywnych zestawów danych testowych i procedur testowych dla testów jednostkowych nowych lub zmodyfikowanych procedur.

Wewnętrzne błędy projektowe lub implementacyjne

Ogólna definicja błędu oprogramowania to niezgodność między programem a jego specyfikacją; bardziej konkretną definicją jest niepowodzenie programu w wykonywaniu tego, czego rozsądnie oczekuje użytkownik końcowy. Istnieje wiele rodzajów błędów oprogramowania. Niektóre z najważniejszych to:

* Inicjalizacja

* Przepływ logiczny

* Obliczenia

* Naruszenia warunków brzegowych

* Przekazywanie parametrów

* Warunki wyścigu

* Stan obciążenia

* Wyczerpanie zasobów

* Konflikt zasobów, adresu lub programu z systemem operacyjnym lub aplikacjami

* Względy dotyczące zgodności z przepisami

* Inne błędy

Inicjalizacja

Błędy inicjalizacji są podstępne i trudne do znalezienia. Najbardziej podstępne programy zapisują informacje inicjujące na dysku i zawodzą tylko przy pierwszym użyciu - to znaczy przed utworzeniem pliku inicjującego. Przy drugim uruchomieniu programu przez danego użytkownika nie ma dalszych błędów inicjalizacji. W związku z tym błędy pojawiają się tylko dla pracowników i klientów, gdy aktywują nową kopię wadliwego programu. Inne programy z błędami inicjalizacji mogą pokazywać dziwne obliczenia lub inne błędy przy pierwszym użyciu lub inicjalizacji; ponieważ nie przechowują swoich wartości inicjalizacji, te błędy inicjalizacji będą pojawiać się ponownie za każdym razem, gdy program jest używany.

Przepływ logiczny

Moduły przekazują kontrolę sobie nawzajem lub innym programom. Jeśli wykonanie przejdzie do niewłaściwego modułu, wystąpił błąd przepływu logicznego. Przykłady obejmują wywołanie niewłaściwej funkcji lub rozgałęzienie do podprogramu, w którym brakuje instrukcji RETURN, tak że wykonanie przechodzi przez logiczny koniec modułu i rozpoczyna wykonywanie innego modułu kodu.

Obliczanie

Gdy program błędnie interpretuje skomplikowane formuły i traci precyzję podczas obliczeń, prawdopodobnie wystąpił błąd obliczeniowy; na przykład wartość pośrednia może być przechowywana w tablicy z 16 bitami precyzji, gdy potrzebuje 32 bitów. Ta kategoria błędów obejmuje również błędy obliczeniowe spowodowane nieprawidłowymi algorytmami.

Naruszenia warunków granicznych

Granice odnoszą się do największych i najmniejszych wartości, z którymi program może sobie poradzić; na przykład tablica może być zwymiarowana z 365 wartościami, aby uwzględnić dni w roku, a następnie zawieść w roku przestępnym, gdy program zwiększy licznik dni do 366 i tym samym spróbuje zapisać wartość pod nieprawidłowym adresem. Programy, które ustawiają zmienne zakresy i alokację pamięci, mogą działać poprawnie w granicach, ale jeśli są niepoprawnie zaprojektowane, mogą ulec awarii przy lub poza granicami. Pierwsze użycie programu również można uznać za warunek brzegowy. Jednym z najważniejszych rodzajów naruszeń granic jest przepełnienie bufora. W przypadku tego błędu dane umieszczone w pamięci przekraczają zdefiniowany maksymalny rozmiar i przechodzą do sekcji pamięci zidentyfikowanej jako należąca do jednej lub więcej różnych zmiennych. Konsekwencje mogą obejmować uszkodzenie danych (np. jeśli przepełnienie nadpisuje dane, które są interpretowane jako wartości liczbowe lub dosłowne) lub zmiany w przepływie wykonywania (np. jeśli zmienione dane zawierają flagi logiczne, które są testowane w instrukcjach rozgałęzienia). Przepełnienia buforów zostały wykorzystane przez twórców szkodliwego kodu, którzy umieszczają dane, które przepełniają obszary pamięci interpretowanych programów, a tym samym są wykonywane jako kod. Wszystkie programy powinny sprawdzać, czy dane przechowywane w tablicy lub w dowolnej lokalizacji pamięci nie przekraczają oczekiwanego rozmiaru danych wejściowych. Dane przekraczające oczekiwany rozmiar powinny zostać odrzucone lub przynajmniej obcięte, aby zapobiec przepełnieniu bufora.

Przekazywanie parametrów

Czasami występują błędy w przekazywaniu danych tam i z powrotem między modułami. Na przykład wywołanie funkcji może zostać przypadkowo przekazana zła nazwa zmiennej, aby funkcja działała na niewłaściwych wartościach. W przypadku wystąpienia tych błędów przekazywania parametrów dane mogą zostać uszkodzone, a ścieżka wykonania może zostać zmieniona z powodu nieprawidłowych wyników obliczeń lub porównań. W rezultacie ostatnie zmiany danych mogą zostać utracone lub wykonanie może wpaść w procedury obsługi błędów, nawet jeśli zamierzone dane były poprawne.

Warunki wyścigu

Gdy między zdarzeniem A a zdarzeniem B dochodzi do wyścigu, do prawidłowego działania wymagana jest określona sekwencja zdarzeń, ale program nie zapewnia tej sekwencji. Na przykład, jeśli proces A blokuje zasób 1 i czeka na odblokowanie zasobu 2, podczas gdy proces B blokuje zasób 2 i czeka na odblokowanie zasobu 1, może wystąpić śmiertelne uścisk, który zawiesza operacje, jeśli procesy nakładają się na siebie podczas wykonywania. Jeśli się nie nakładają, wtedy nie ma problemu. W systemach wieloprocesorowych i systemach interaktywnych można się spodziewać warunków wyścigowych, ale mogą one być trudne do odtworzenia; na przykład opisany właśnie śmiertelny uścisk

może się zdarzyć tylko raz na 1000 transakcji, jeśli średni czas transakcji jest bardzo krótki. W związku z tym warunki wyścigu należą do najtrudniejszych do wykrycia podczas testów zapewniania jakości i są najlepiej identyfikowane podczas przeglądów kodu. Programiści powinni ustanowić i przestrzegać standardów dotyczących operacji sekwencyjnych, które wymagają wyłącznego dostępu do więcej niż jednego zasobu. Na przykład, jeśli wszystkie procesy blokują zasoby na wyłączność w danej kolejności i odblokowują je w odwrotnej kolejności, nie może być śmiertelnego objęcia.

Stan obciążenia

Wszystkie programy i systemy mają ograniczenia dotyczące pojemności pamięci, liczby użytkowników, transakcji i przepustowości. Błędy obciążenia są spowodowane przekroczeniem ograniczeń pojemności pamięci, transakcji, użytkowników i sieci mogą wystąpić z powodu dużego wolumenu, który obejmuje dużą ilość pracy przez długi czas, lub dużego obciążenia, które obejmuje maksymalne obciążenie jednocześnie . Na przykład, jeśli całkowita teoretyczna liczba transakcji spowoduje żądanie w systemie dyskowych operacji we/wy, które przekroczy przepustowość kontrolera dysku, procesy z konieczności zaczną gromadzić się w kolejce czekając na zakończenie dyskowych operacji we/wy. Chociaż obliczenia teoretyczne mogą pomóc w zidentyfikowaniu możliwych wąskich gardeł w procesorach, pamięci, dyskach i zasobach sieciowych, przydatnym dodatkiem jest automatyczne testowanie, które umożliwia symulację maksymalnych obciążeń określonych w umowach dotyczących poziomu usług.

Wyczerpanie zasobów

Programowi wyczerpuje się pamięć o dużej szybkości (RAM), pamięć masowa (dysk), cykle jednostki centralnej (CPU), wpisy w tabeli systemu operacyjnego, semafory, przepustowość sieci lub inne zasoby, które mogą spowodować awarię programu. Na przykład nieodpowiednia pamięć główna może powodować nadmierną wymianę danych na dysk (thrashing), zwykle powodując drastyczne zmniejszenie przepustowości, ponieważ operacje we/wy dysku są zazwyczaj 1000 razy wolniejsze niż dostęp do pamięci.

Konflikty międzyaplikacyjne

W przypadku tak złożonych systemów operacyjnych (OS) producenci systemów operacyjnych rutynowo dystrybuują wymagania dotyczące kodu i określone parametry do producentów oprogramowania aplikacji, aby zminimalizować prawdopodobieństwo konfliktów programów lub nieoczekiwanych przestojów. Chociaż z pewnością pomaga to zmniejszyć liczbę problemów i poprawia zgodność w przód i wstecz z poprzednimi wersjami systemu operacyjnego, nawet dostawcy systemów operacyjnych czasami doświadczają lub powodują trudności, gdy nie są zgodne z parametrami ustalonymi dla ich własnych programów.

Inne źródła błędów

Często zdarza się, że programy wysyłają złe dane do urządzeń, ignorują powracające kody błędów, a nawet próbują używać urządzeń, które są zajęte lub których brakuje. Sprzęt może być zepsuty, ale oprogramowanie jest również uważane za błędne, gdy nie odzyskuje sprawności po takich warunkach sprzętowych. Dodatkowe błędy mogą wystąpić przez niewłaściwe kompilacje pliku wykonywalnego; na przykład, jeśli stara wersja modułu jest połączona z najnowszą wersją reszty programu, mogą pojawić się nieprawidłowe ekrany logowania, mogą zostać wyświetlone nieprawidłowe komunikaty o prawach autorskich, mogą pojawić się nieprawidłowe numery wersji i różne mogą wystąpić inne niedokładności.

Interfejs użytkownika

Mówiąc ogólnie, termin „interfejs użytkownika” oznacza wszystkie aspekty systemu, które są istotne dla użytkownika. Można go ogólnie określić jako wirtualną maszynę użytkownika (UVM). Obejmuje to wszystkie ekrany, mysz i klawiaturę, wydruki i wszystkie inne elementy, z którymi użytkownik wchodzi w interakcję. Poważny problem pojawia się, gdy projektanci systemów nie mogą postawić się na miejscu użytkownika i nie mogą przewidzieć problemów, jakie będą mieli użytkownicy technologicznie zaawansowani z interfejsem zaprojektowanym przez osobę znającą technologię. Dokumentacja jest kluczową częścią każdego systemu. Każda faza rozwoju – wymagania, analiza, rozwój, kodowanie, testowanie, błędy, rozwiązania błędów i modyfikacje, implementacja i utrzymanie – musi być udokumentowana. Wszystkie dokumenty i ich różne wersje muszą być zachowane zarówno do przyszłego wykorzystania, jak i do celów audytu. Dodatkowo ważne jest udokumentowanie prawidłowego użytkowania systemu oraz zapewnienie użytkownikowi odpowiednich materiałów instruktażowych i referencyjnych. Polityki bezpieczeństwa oraz związane z nimi egzekwowanie i kary również muszą być udokumentowane. Najlepiej byłoby, gdyby dokumentacja umożliwiała każdej osobie wykwalifikowanej technicznie naprawę lub modyfikację dowolnego elementu, o ile system działa.

Funkcjonalność

W programie występuje błąd funkcjonalności, jeśli wydajność, której można racjonalnie oczekiwać, jest zagmatwana, niezręczna, trudna lub niemożliwa. Błędy funkcjonalne często dotyczą kluczowych cech lub funkcji, które nigdy nie zostały zaimplementowane. Dodatkowe błędy funkcjonalności występują, gdy:

* Funkcje nie są udokumentowane.

* Brak wymaganych informacji.

* Program nie potwierdza prawidłowych danych wejściowych.

* Występują błędy rzeczowe lub sprzeczne nazwy funkcji.

* Występuje przeciążenie informacjami.

* Materiał jest napisany na nieodpowiednim poziomie czytania.

* Kursor znika lub znajduje się w niewłaściwym miejscu.

* Wyświetlacze na ekranie są nieprawidłowe.

* Instrukcje są niejasne.

* Identyczne funkcje wymagają różnych operacji na różnych ekranach.

* Istnieją niepoprawnie sformatowane ekrany wprowadzania.

* Hasła lub inne poufne informacje nie są odpowiednio zasłaniane ani chronione.

* Śledzenie wpisu lub zmian danych użytkownika jest niedostępne lub niekompletne.

* Segregacja obowiązków nie jest egzekwowana. (Może to być szczególnie krytyczne dla organizacji podlegających wymogom prawnym i regulacyjnym).

Struktura sterowania (polecenia)

Błędy struktury kontrolnej mogą powodować poważne problemy, ponieważ mogą skutkować:

* Użytkownicy gubią się w programie.

* Użytkownicy marnują czas, ponieważ muszą radzić sobie z mylącymi poleceniami.

* Utrata danych lub niepożądane ujawnienie danych.

* Opóźnienie w pracy.

* Koszty finansowe.

* Nieoczekiwane narażenie na wyciek danych lub narażenie na niebezpieczeństwo; może to skutkować znaczną odpowiedzialnością w przypadku ujawnienia danych osobowych konsumentów (PII).

* Dane nie są szyfrowane zgodnie z przeznaczeniem lub nie są widoczne dla nieautoryzowanych użytkowników.

Niektóre typowe błędy to:

* Brak możliwości poruszania się między menu.

* Mylące i powtarzające się menu.

* Niedopuszczenie odpowiednich wpisów w wierszu poleceń.

* Wymaganie wpisów wiersza poleceń, które nie są ani intuicyjne, ani wyraźnie zdefiniowane na ekranie.

* Niezgodność aplikacji z konwencjami systemu operacyjnego.

* Brak rozróżnienia między plikami źródłowymi a parametrami, skutkujący udostępnieniem użytkownikowi błędnych wartości za pośrednictwem interfejsu lub nieidentyfikacją źródła błędu

* Niewłaściwe użycie klawiatury, gdy nowe programy nie spełniają standardów klawiatury, która ma przypisane klawisze funkcyjne do standardowych znaczeń.

* Brakujące polecenia w kodzie i ekranach, co powoduje, że użytkownik nie może uzyskać dostępu do informacji, korzystać z programów lub zapewnić kopii zapasowej i możliwości odzyskania systemu. Istnieje wiele innych poleceń, które mogą pozostawić system w stanie nieoptymalnym.

* Nieodpowiednia prywatność lub bezpieczeństwo, które mogą skutkować ujawnieniem poufnych informacji, całkowitą zmianą lub utratą danych bez możliwości odzyskania, złym raportowaniem, a nawet niepożądanym dostępem osób z zewnątrz.

Wydajność

Szybkość jest ważna w interaktywnym oprogramowaniu. Jeśli użytkownik uważa, że program działa wolno, może to być natychmiastowy problem. Problemy z wydajnością obejmują powolną reakcję, niezapowiedzianą czułość na wielkość liter, niekontrolowane i nadmiernie częste automatyczne zapisywanie, niemożność zapisania i ograniczoną prędkość przewijania. Istnieje pięć podstawowych potencjalnych wąskich gardeł, które należy wziąć pod uwagę, analizując każdy problem z wydajnością komputera i sieci, związany z dostępem i szybkością:

*Centralna(e) jednostka(e) przetwarzająca(e)

* Pamięć główna

* Pamięć dodatkowa (np. dyski twarde)

* Projekt aplikacji

* Przepustowość sieci

Powolne działanie może zależeć (ale nie tylko) od systemu operacyjnego, innych uruchomionych aplikacji, nasycenia pamięci i thrashingu (nadmierna zamiana zawartości pamięci na pamięć wirtualną na dysku lub na innych, stosunkowo wolnych zasobach pamięci, takich jak dyski flash), wyciek pamięci (niepowodzenie cofania alokacji pamięci, która nie jest już potrzebna), nieefektywność operacji we/wy dysku (np. odczytywanie pojedynczych rekordów z bardzo dużych bloków) oraz konflikty programów (np. błędy blokowania). Więcej informacji na temat wydajności programu. Na innym poziomie wydajność spada, gdy projekty programów utrudniają zmianę ich funkcjonalności w odpowiedzi na zmieniające się wymagania. W projekcie bazy danych zdefiniowanie podstawowego pola indeksu, które określa kolejność, w jakiej rekordy są przechowywane na dysku, może znacznie przyspieszyć dostęp do rekordów podczas sekwencyjnych odczytów wartości kluczy dla tego indeksu — ale może przynieść efekt przeciwny do zamierzonego, jeśli dominująca metoda dostępu do rekordów to sekwencyjne odczyty na zupełnie innym indeksie.

Format wyjściowy

Błędy formatu wyjściowego mogą być frustrujące i czasochłonne. Uznaje się, że wystąpił błąd, gdy użytkownik nie może zmienić czcionki, podkreślenia, pogrubienie i odstępy, które wpływają na ostateczny wygląd wydruku; alternatywnie mogą wystąpić opóźnienia lub błędy podczas drukowania lub zapisywania dokumentu. Błędy pojawiają się, gdy użytkownik nie może kontrolować zawartości, skalowania i wyglądu tabel, rysunków i wykresów. Dodatkowo występują błędy wyjściowe, które wiążą się z wyrażeniem danych z nieodpowiednim poziomem precyzji.

NARZĘDZIA I TECHNIKI ZAPEWNIAJĄCE

Społeczność ds. bezpieczeństwa wiedziała o lukach w zabezpieczeniach związanych z przepełnieniem bufora, jako przykład, przez ostatnie 25 lub 30 lat, ale luki te wciąż rosną. Na przykład National Vulnerability Database zgłosił 418 nowych błędów przepełnienia bufora w 2013 r. w porównaniu z 392 w 2011 r. Oczywiście konieczne są większe wysiłki, aby uczyć i egzekwować świadomość, wdrażanie, i testowanie kodu źródłowego.

Zasoby edukacyjne

Build Security In (BSI) to projekt programu Software Assurance Oddziału Inicjatyw Strategicznych Krajowego CyberWydziału Bezpieczeństwa (NCSD) Departamentu Bezpieczeństwa Wewnętrznego Stanów Zjednoczonych. BSI (http://buildsecurityin.us-cert.gov) zawiera najlepsze praktyki, narzędzia, wytyczne, reguły, zasady i inne zasoby, które twórcy oprogramowania, architekci i specjaliści ds. bezpieczeństwa mogą wykorzystać do wbudowania zabezpieczeń w oprogramowanie w każdym faza jego rozwoju. Treść BSI opiera się na zasadzie, że bezpieczeństwo oprogramowania jest zasadniczo problemem inżynierii oprogramowania i musi być rozwiązywane w sposób systematyczny przez cały cykl życia oprogramowania. Oprócz BSI i spośród mnóstwa witryn w Internecie, które zajmują się pewnym aspektem bezpiecznego kodowania, niektóre witryny są wymienione obok, aby pomóc w zabezpieczeniu aplikacji :

* Instytut Inżynierii Oprogramowania Carnegie-Mellon: www.cert.org/secure-coding

* Informacje dotyczące bezpieczeństwa FreeBSD: www.freebsd.org/security/security.html

* Międzynarodowe Stowarzyszenie Inżynierii Bezpieczeństwa Systemów (ISSEA): www.issea.org

* Otwórz projekt bezpieczeństwa aplikacji internetowych: www.owasp.org

*Bezpieczne, wydajne i łatwe programowanie w języku C: http://irccrew.org/~cras/security/cguide.html

* Bezpieczne programowanie dla systemów Linux i UNIX HOWTO: http://www.dwheeler.com/secure-programs/

* Zabezpieczenia systemu Windows: www.windowsecurity.com

* Często zadawane pytania dotyczące bezpieczeństwa sieci WWW: http://www.w3.org/Security/Faq/

Techniki badania kodu i testowania aplikacji

Jedno skanowanie nie wystarczy; ciągła ocena aplikacji jest niezbędna do wdrożenia skutecznych praktyk bezpiecznego programowania, a co za tym idzie, bezpiecznej aplikacji. Podobnie jak wiele innych aspektów technologii informacyjnej i bezpieczeństwa komputerowego, potrzebne jest również warstwowe podejście do testowania. W rezultacie istnieje wiele technik i podejść, zarówno ręcznych, jak i automatycznych, do testowania bezpieczeństwa oprogramowania. Ta sekcja zapewni pobieżny przegląd głównych technik i podejść.

Analiza statyczna

Analiza statyczna lub przegląd kodu źródłowego to proces ręcznego sprawdzania kodu źródłowego pod kątem słabych punktów bezpieczeństwa. Czasami spojrzenie na kod jest jedynym sposobem rozwiązania problemów. Zrozumienie intencji wewnętrznego działania aplikacji może zidentyfikować poważne problemy z bezpieczeństwem, których nie można zidentyfikować w inny sposób. Niektóre problemy, które szczególnie sprzyjają wykrywaniu poprzez analizę statyczną, obejmują problemy ze współbieżnością, wadliwą logikę biznesową, problemy z kontrolą dostępu, tylne drzwi, trojany, pisanki, bomby czasowe/logiczne i inne formy złośliwego kodu.

Analiza kodu binarnego

Analiza kodu binarnego wykorzystuje narzędzia obsługujące analizę plików wykonywalnych, takich jak dekompilatory, deasemblery i skanery kodu binarnego. To znaczy, że analiza binarna to inżynieria wsteczna pliku wykonywalnego. Najmniej inwazyjną techniką jest skanowanie binarne. Skanery binarne analizują kod maszynowy w celu modelowania neutralnej językowo reprezentacji zachowań programu, przepływów sterowania/danych, drzew wywołań i wywołań funkcji zewnętrznych. Zautomatyzowany skaner podatności przeszukuje następnie model w celu zlokalizowania podatności spowodowanych częstymi błędami kodowania i prostymi tylnymi drzwiami. Kolejną najmniej inwazyjną techniką jest deasemblacja pliku binarnego. Kod binarny jest odtwarzany wstecznie do pośredniego języka asemblera, ale wadą deasemblacji jest to, że wymaga on eksperta, który zarówno dokładnie rozumie ten konkretny język asemblera, jak i który jest biegły w wykrywaniu konstrukcji istotnych z punktu widzenia bezpieczeństwa w kodzie asemblera. Najbardziej inwazyjną techniką inżynierii wstecznej jest dekompilacja kodu. Kod binarny jest całkowicie poddawany inżynierii wstecznej w kodzie źródłowym, który może być następnie poddany tym samym technikom przeglądu kodu bezpieczeństwa i innym testom białej skrzynki, co oryginalny kod źródłowy. Należy jednak zauważyć, że dekompilacja jest technicznie problematyczna, ponieważ jakość kodu źródłowego generowanego przez dekompilację jest często bardzo niska. W rezultacie kod rzadko jest tak łatwy w nawigacji lub zrozumiały jak oryginalny kod źródłowy. Analiza zdekompilowanego kodu źródłowego zawsze będzie trudniejsza i bardziej czasochłonna niż analiza oryginalnego kodu źródłowego. Z tego powodu użycie dekompilacji dla analizy bezpieczeństwa powinna być używana tylko dla najbardziej

krytycznych systemów. Inną kwestią do rozważenia w odniesieniu do inżynierii wstecznej jest to, że może ona być również prawnie zabroniona. Większość umów licencyjnych z dostawcami oprogramowania zabrania inżynierii odwrotnej. Ponadto wielu producentów oprogramowania wielokrotnie przytacza jako uzasadnienie ustawę Digital Millennium Copyright Act (DMCA) z 1999 r., mimo że ustawa DMCA wyraźnie zwalnia badania nad inżynierią wsteczną i szyfrowaniem z zakazów dotyczących obchodzenia ochrony przed kopiowaniem.

Podejście Białej Skrzynki

Strategia testowania białej skrzynki zajmuje się wewnętrzną logiką i strukturą kodu. Testowanie w białych skrzynkach jest również znane jako testowanie szkła, strukturalne, otwarte pudełko lub przezroczyste pudełko. Aby zaimplementować testowanie białoskrzynkowe, tester musi poradzić sobie z kodem, a zatem musi posiadać wiedzę z zakresu kodowania i logiki (tj. Wewnętrzne działanie kodu). Testy białoskrzynkowe wymagają również, aby tester zajrzał do kodu i stwierdził, która jednostka/instrukcja/fragment kodu działa nieprawidłowo. Zalety testowania białej skrzynki to:

* Bardzo łatwo jest dowiedzieć się, jaki rodzaj danych wejściowych/danych może pomóc w skutecznym testowaniu aplikacji.

* Pomaga w optymalizacji kodu.

* Pomaga w usuwaniu dodatkowych linijek kodu, które mogą wprowadzać ukryte defekty.

Wady testowania białej skrzynki to:

* Do przeprowadzenia tego typu testów potrzebny jest wykwalifikowany tester, co zwiększa koszty.

* Jest prawie niemożliwe, aby zajrzeć do każdego kawałka kodu, aby znaleźć ukryte błędy, które mogą powodować problemy, powodując awarię aplikacji.

Podejście czarnoskrzynkowe

Testowanie czarnoskrzynkowe to testowanie bez wiedzy o wewnętrznym działaniu testowanego elementu. Testowanie czarnoskrzynkowe jest również znane jako behawioralne, funkcjonalne, nieprzezroczyste i zamknięte pudełko. Z tego powodu tester i programista mogą być od siebie niezależni, unikając uprzedzeń programisty do własnej pracy. Często używane są grupy testowe. Ze względu na charakter testowania czarnoskrzynkowego planowanie testów można rozpocząć natychmiast po napisaniu specyfikacji. Zalety testowania czarnoskrzynkowego to:

* Jest bardziej skuteczny w przypadku większych jednostek kodu niż testowanie w szklanym pudełku.

* Tester nie potrzebuje znajomości implementacji, w tym konkretnych języków programowania.

* Tester i programista są od siebie niezależni.

* Testy przeprowadzane są z punktu widzenia użytkownika.

* Pomoże to wyeksponować wszelkie niejasności lub niespójności w specyfikacjach.

* Przypadki testowe można zaprojektować zaraz po skompletowaniu specyfikacji.

Wady testowania czarnoskrzynkowego to:

* Tylko niewielka liczba możliwych wejść może być faktycznie przetestowana; przetestowanie każdego możliwego strumienia wejściowego w złożonym systemie zajęłoby prawie wieczność.

* Bez jasnych i zwięzłych specyfikacji trudno jest zaprojektować przypadki testowe.

* Może wystąpić niepotrzebne powtarzanie wejść testowych, jeśli tester nie zostanie poinformowany o przypadkach testowych, które programista już wypróbował. Zawsze ważne jest, aby zweryfikować wszelkie twierdzenia testowe za pomocą raportów lub wyników, aby uniknąć powielania. Podobnie, zawsze ważne jest ponowne uruchamianie testów, które mogą dawać niejednoznaczne wyniki.

* Może pozostawić wiele ścieżek programu nieprzetestowanych.

* Nie może dotyczyć konkretnych segmentów kodu, które mogą być bardzo złożone.

Podejście mieszane z szarym pudełkiem

Testowanie szarej skrzynki to technika testowania oprogramowania, która wykorzystuje kombinację testowania czarnej skrzynki i testowania białej skrzynki. Testowanie szarej skrzynki nie jest testowaniem czarnej skrzynki, ponieważ tester zna niektóre wewnętrzne działanie testowanego oprogramowania. W testach szarej skrzynki tester stosuje ograniczoną liczbę przypadków testowych do wewnętrznego działania testowanego oprogramowania. W pozostałej części testowania szarej skrzynki tester stosuje podejście czarnej skrzynki, wprowadzając dane wejściowe do testowanego oprogramowania i obserwując dane wyjściowe.

Błąd wtrysku

Istnieją dwie podstawowe kategorie wstrzykiwania błędów: kod źródłowy lub binarne wstrzykiwanie błędów. Wstrzyknięcie usterki służy do wywołania stresu w oprogramowaniu, tworzyć problemy ze współdziałaniem między komponentami, symulować błędy w środowisku wykonawczym, a tym samym ujawniać błędy zagrażające bezpieczeństwu, które nie są widoczne przy użyciu tradycyjnych technik testowania. Bezpieczeństwo rozszerza standardowe wstrzykiwanie błędów, dodając wstrzykiwanie błędów, umożliwiając testerom analizę wpływu zachowań na bezpieczeństwo i zmiany stanu wynikające z wystawienia oprogramowania na zmiany w danych środowiskowych. Te zmiany danych mają na celu symulację rodzajów błędów, które mogą wystąpić podczas niezamierzonych błędów użytkownika lub celowych ataków na oprogramowanie za pośrednictwem jego środowiska, w tym ataków na samo środowisko. Zmiana danych to po prostu zmiana danych, które środowisko wykonawcze przekazuje do oprogramowania lub między składnikami oprogramowania. Wstrzykiwanie błędów ujawnia zarówno wpływ błędów bezpieczeństwa na zachowanie poszczególnych komponentów, jak i zachowanie systemu jako całości. Binarne wstrzykiwanie błędów jest przydatne, gdy jest wykonywane w ramach testów penetracyjnych, aby umożliwić testerowi uzyskanie bardziej całościowego obrazu reakcji oprogramowania na ataki. Jednak błędy nie powinny ograniczać się do symulacji ataków w świecie rzeczywistym. Scenariusze wstrzykiwania błędów powinny być zaprojektowane tak, aby zapewnić pełne zrozumienie wpływu na bezpieczeństwo, jaki mają zachowania, stany i właściwości aplikacji we wszystkich możliwych warunkach pracy. Główne wyzwania w testowaniu wstrzykiwania usterek polegają na określeniu najbardziej znaczącej kombinacji i ilości usterek do wstrzyknięcia oraz, jak w przypadku wszystkich testów, interpretacji wyników. Co więcej, tylko dlatego, że wstrzykiwanie błędów nie powoduje, że oprogramowanie zachowuje się w sposób niezabezpieczony ani nie działa w stanie niezabezpieczonym, nie można tego interpretować w ten sposób, że oprogramowanie będzie działać tak samo, gdy zostanie wystawione na bardziej złożone dane wejściowe, które zwykle otrzymuje podczas wykonywania.

Fuzzing

Testowanie rozmyte lub fuzzing to technika często zautomatyzowana lub półautomatyczna, która polega na dostarczaniu nieprawidłowych, nieoczekiwanych lub losowych danych do danych

wejściowych programu komputerowego. Program jest następnie monitorowany pod kątem wyjątków, nieudanych wbudowanych potwierdzeń kodu lub znalezienia potencjalnych wycieków pamięci. Istnieją dwie formy fuzzingu, oparte na mutacjach i oparte na pokoleniu, które można zastosować w ramach testów białej, czarnej lub szarej skrzynki. Ideą testowania fuzz jest poszukiwanie interesującego zachowania programu, które wynika z wstrzykiwania szumu i może wskazywać na obecność luk lub błędów. Aplikacje fuzzujące, fuzzery, są zazwyczaj specyficzne dla określonego typu danych wejściowych i napisane w celu przetestowania określonego programu; nie można ich łatwo ponownie wykorzystać, ale formaty plików i protokoły sieciowe są najczęstszymi celami testowania. Dowolny typ danych wejściowych programu może zostać zamazany, w tym elementy, które normalnie nie są uważane za „dane wejściowe" (np. bazy danych, pamięć współdzielona itp.). Wartość tkwi w specyfice, co oznacza, że fuzzing często może ujawnić luki w zabezpieczeniach, których ogólne narzędzia testujące, takie jak skanery podatności i wstrzykiwacze błędów, nie są w stanie. Z drugiej strony, całkowicie losowe fuzzing jest zazwyczaj nieefektywnym sposobem wykrywania problemów w aplikacji, w wyniku czego technologia fuzzingu ewoluowała, obejmując bardziej inteligentne techniki.

Skanowanie podatności

Automatyczne skanowanie podatności to zautomatyzowana technika skanowania oprogramowania na poziomie aplikacji, a także serwerów internetowych, systemy zarządzania bazami danych i niektóre systemy operacyjne. Narzędzia skanują oprogramowanie aplikacji w poszukiwaniu danych wejściowych i wyjściowych znanych wzorców powiązanych ze znanymi lukami w zabezpieczeniach. Wzorce lub sygnatury luk w zabezpieczeniach są podobne do sygnatur używanych przez skanery antywirusowe lub konstrukcji kodowania wyszukiwanych przez automatyczne skanery kodu źródłowego, dzięki czemu skaner luk w zabezpieczeniach jest zasadniczo narzędziem do automatycznego dopasowywania wzorców. Niektóre skanery luk w zabezpieczeniach aplikacji sieci Web dodatkowo próbują przeprowadzić zautomatyzowaną ocenę stanu przy użyciu symulowanych wzorców ataków rozpoznawczych i technik rozmycia w celu wykrycia znanych lub powszechnych luk w zabezpieczeniach. Jak każdy skaner oparty na sygnaturach, skanery luk w zabezpieczeniach mogą zgłaszać fałszywe alarmy. W rezultacie tester musi mieć wystarczającą wiedzę w zakresie oprogramowania i bezpieczeństwa, aby sensownie zinterpretować wyniki skanera, aby ograniczyć liczbę fałszywie pozytywnych lub negatywnych wyników, aby zapobiec stworzeniu incydentu, w którym nie istnieje. Dlatego ważne jest, aby zastosować podejście warstwowe i połączyć różne techniki testowe w celu zbadania oprogramowania pod kątem luk w zabezpieczeniach. Luki w oprogramowaniu są często maskowane przez zabezpieczenia środowiskowe, takie jak zapory sieciowe i na poziomie aplikacji. Ponadto warunki środowiskowe mogą tworzyć unikalne luki w zabezpieczeniach, których nie można znaleźć za pomocą narzędzia opartego na sygnaturach, ale wykorzystującego kombinację innych testów czarnoskrzynkowych, w szczególności testów penetracyjnych wdrożonego oprogramowania w rzeczywistym środowisku docelowym. Skanery luk w zabezpieczeniach są najskuteczniejsze w ramach:

* Oceny bezpieczeństwa komponentów binarnych;

* Lub przed testami penetracyjnymi (w celu wyeliminowania konieczności uruchamiania scenariuszy testów penetracyjnych dla typowych luk w zabezpieczeniach).

Testy penetracyjne oprogramowania

Testy penetracyjne od wielu lat są powszechną techniką testowania bezpieczeństwa sieci i są również znane jako testy czarnej skrzynki i etyczne hakowanie. Jak opisano w podejściu czarnej skrzynki, testy penetracyjne to zasadniczo testowanie działającej aplikacji (bez znajomości wewnętrznego działania aplikacji) w celu zidentyfikowania luk w zabezpieczeniach. W testach penetracyjnych cały system

oprogramowania jest środowiskiem „na żywo”. Zazwyczaj zespół penetracyjny miałby dostęp do aplikacji jak typowi użytkownicy. Testerzy zachowują się wtedy jak atakujący, próbując znaleźć i wykorzystać potencjalne luki w zabezpieczeniach. Testy penetracyjne powinny koncentrować się na aspektach zachowania systemu, interakcji i podatności, których nie można zaobserwować za pomocą innych technik testowania wykonywanych poza żywym środowiskiem. Oznacza to, że penetracja powinna narazić system na wyrafinowane wielowzorcowe ataki mające na celu wywołanie złożonych zachowań w poszczególnych składnikach systemu. Ponadto testy powinny próbować znaleźć problemy z bezpieczeństwem, które mogą pochodzić z architektury i projektu, a nie wady kodowania, ponieważ tego typu problem jest często pomijany przez inne techniki testowania. Oznacza to również, że plan testów powinien zawierać najgorsze scenariusze, które odtwarzają wektory zagrożeń uważane za wysoce szkodliwe, takie jak scenariusz zagrożenia wewnętrznego. Wiele osób używa testów penetracyjnych aplikacji jako podstawowej techniki testowania i chociaż ma to miejsce w programie testowym; nie należy jej uważać za podstawową lub jedyną technikę testowania. Testy penetracyjne mogą potencjalnie prowadzić do fałszywego poczucia bezpieczeństwa. Tylko dlatego, że aplikacja przeszła test penetracyjny, nie oznacza to, że jest wolna od luk w zabezpieczeniach. Podobnie, jeśli aplikacja nie powiedzie się, jest to silna wskazówka, że są z nią poważne problemy. Jednak skoncentrowane testy penetracyjne, czyli testy mające na celu wykorzystanie luk wykrytych w poprzednich testach, mogą być przydatne w wykrywaniu, czy te luki zostały rzeczywiście naprawione w kodzie.

Standardy i najlepsze praktyki

Testowanie jest samodzielną dyscypliną inżynierii oprogramowania. Napisano tomy o różnych technikach stosowanych przez inżynierów oprogramowania do testowania i walidacji swoich aplikacji. Podstawowe najlepsze praktyki można zastosować niezależnie od wybranej metodyki testowania.

* Wykonaj automatyczne testy.

* Zrób plan testu.

* Postępuj zgodnie z określoną metodologią.

* Testuj na każdym etapie.

* Przetestuj wszystkie elementy systemu.

Oprócz tych najlepszych praktyk, stosowanie tych standardów może zapewnić dodatkowe zasoby:

*ISO 17799, Technologia informacyjna: Kodeks postępowania w zarządzaniu bezpieczeństwem informacji

* ISO/IEC 15408, Kryteria oceny bezpieczeństwa IT (Wspólne kryteria)

* SSE-CMM, model dojrzałości możliwości inżynierii bezpieczeństwa systemu

* ISO/IEC WD 15443, Technologia informacyjna: Techniki bezpieczeństwa

UWAGI KOŃCOWE

Praktycy bezpieczeństwa komputerowego i menedżerowie najwyższego szczebla muszą zrozumieć, że braki w zarządzaniu IT, pozostawione bez zmian, będą sabotować wysiłki na rzecz ustanowienia i utrzymania skutecznych programów bezpieczeństwa komputerowego. Ponadto na tych, którzy kodują, spoczywa pomoc w kierowaniu projektem i kształcenie kierownictwa w kierunku modelu, który obejmuje bezpieczeństwo w całym cyklu rozwoju. Koszty zaniedbania bezpieczeństwa na początku

będą nadal stanowić poważny problem, dopóki bezpieczeństwo oprogramowania nie zostanie w pełni zintegrowane z każdym aspektem cyklu życia oprogramowania. Bezpieczeństwo musi być wbudowane, a budowanie bezpieczeństwa w systemach zaczyna się od zespołów programistycznych, praktyk i technik używanych do tworzenia tych systemów.