Programowanie Defensywne

Tworzenie bezpiecznego oprogramowania
Procesy i narzędzia

Arkadiusz Chrobot

Katedra Systemów Informatycznych

28 czerwca 2024

Plan

Wstęp

Wielkoskalowe wytwarzanie oprogramowania przeszło rozwój od modelu kaskadowego (ang. *waterfall model*) do podejścia iteracyjnego, które obecnie jest praktykowane głównie w postaci metodyk zwinnych (ang. *agile methodologies*). W tym podejściu stosowane jest tworzenie ewolucyjne. Rozpoczyna się ono od prototypu lub minimalnej wersji produktu (ang. *Minimum Viable Product*, MVP), zapewniającej najważniejsze z punktu widzenia klienta funkcje. Kolejne są dodawane do tego oprogramowania przyrostowo. To wymaga znaczących zmian w procesie jego tworzenia. Każda zmiana, każdy przyrost musi być jak najszybciej zintegrowany z całością i przetestowany, głównie pod kątem pojawienia się regresji. To spowodowało opracowanie rozwiązań typu *ciągła integracja* (ang. *Continuous Integration*, CI). Częste dostarczanie odbiorcom wersji z niewielkimi zmianami znacznie ułatwia uzyskanie od nich informacji zwrotnych i prace nad rozwojem oprogramowania. Dlatego powstały narzędzia do *ciągłego wdrażania* (ang. *Continuous Delivery*, CD).

DevOps

Kolejną nowością stało się dostarczanie oprogramowania odbiorcom przez producentów za pomocą cienkich klientów (ang. *thin client*), najczęściej w postaci przeglądarek WWW. To rozwiązanie zostało nazwane *oprogramowaniem jako usługą* (ang. *Software as a Service, SaaS*). Nakłada ono nowe obowiązki na producenta, takie jak utrzymywanie kontenerów i maszyn wirtualnych, ale również daje nowe możliwości i ułatwienia, takie jak monitorowanie w czasie rzeczywistym działania aplikacji, zapewnienie stabilnego środowiska wykonawczego i selektywne udostępnianie nowych wersji. Wymaga ono również połączenia prac rozwojowych (ang. *development*) z operacyjnymi (ang. *operations*) oraz ich automatyzacji. Odpowiedzią na te wymagania jest metodologia *DevOps*.

DevSecOps

Tradycyjnie kwestie bezpieczeństwa były poruszane *na końcu* prac rozwojowych dotyczących oprogramowania. Zazwyczaj sprowadzały się one do usuwania podatności po przeprowadzeniu testów penetracyjnych lub audytu, albo po wykryciu incydentu bezpieczeństwa. Były podejmowane próby zmiany tego stanu rzeczy. Jedną z nich przeprowadziła firma Microsoft, wprowadzając w 2008 roku model rozwoju oprogramowania *Secure Development Lifecycle*, SDL. W tym podejściu zagadnienie bezpieczeństwa jest uwzględniane na każdym etapie prac rozwojowych, od określania wymagań, aż do wdrożenia. Wymaga ono zatrudnienia w firmie eksperta od kwestii bezpieczeństwa (ang. *security-matter expert*) i ustanowienia „mistrza" bezpieczeństwa w każdym zespole (ang. *security champion*). Ten model jest jednakże pochodną paradygmatu kaskadowego.

DevSecOps

W podejściu iteracyjnym stosowane jest rozwiązanie bazujące, w pewnym zakresie, na pomysłach z SDL. Nazywa się ono *DevSecOps* [1, 2]. W tym modelu kwestie bezpieczeństwa obecne są od początku, od oceny ryzyka, poprzez modelowanie zagrożeń, projektowanie, implementację, testowanie, wdrożenie, monitorowanie i reakcje na incydenty bezpieczeństwa. Ponieważ ta metodologia jest stosowana do opracowywania i utrzymania (ang. *maintenance*) systemów, z których mogą korzystać nawet miliardy użytkowników, to istotnym zagadnieniem staje się automatyzowanie czynności administratorskich, ale również umożliwienie dokonywania ręcznych zmian w dowolnym momencie. Ciekawym przykładem zastosowania automatyzacji jest mechanizm nazwany *chaos monkeys*, opracowany przez firmę Netflix. Jego działanie polega na losowym wyłączaniu serwerów, routerów, klastrów, a nawet całych centrów danych, aby przetestować odporność całego systemu na niepodziewane awarie.

Analizatory statyczne

Podejście DevSecOps wymaga zatem odpowiednich narzędzi. Do nich zaliczają się między innymi *analizatory statyczne*, wyszukujące potencjalne podatności w kodzie źródłowym. Z tego sposobu działania bierze się też ich nazwa — badanie oprogramowania przy ich pomocy nie wymaga jego uruchomienia. Stosują one różne techniki wykrywania defektów, od prostego wyszukiwania wzorców bezpośrednio w kodzie lub w reprezentującym go drzewie składni abstrakcyjnej (ang. *Abstract Syntax Tree*, AST), po wnioskowanie o semantycznej spójności stanów programu, bazujące na przepływie danych i sterowania w tym programie [1, 3]. Ten ostatni typy analizy wymaga skonstruowania *grafu przepływu sterownia* (ang. *Control Flow Graph*, CFG). Każdy wierzchołek takiego grafu reprezentuje blok podstawowy (ang. *basic block*), czyli sekwencję instrukcji, w której instrukcja rozgałęzienia lub zatrzymania może wystąpić jedynie na końcu. Zatem sterowanie wchodzi do bloku podstawowego na jego początku i opuszcza go na końcu.

Analizatory statyczne

Krawędzie w CFG reprezentują przepływ sterowania między blokami, inaczej *ścieżki* (ang. *paths*). Graf przepływu sterowania jest grafem skierowanym. Jeśli dany węzeł w tym grafie ma tylko krawędzie wchodzące to jest nazywany blokiem *wyjściowym* (ang. *exit*), a jeśli tylko ścieżki wychodzące, to blokiem *wejściowym* (ang. *entry*).
Inną techniką wykorzystywaną w analizatorach statycznych jest *analiza skażenia* (ang. *taint*). Polega ona na zidentyfikowaniu zmiennych, których wartości mogą być bezpośrednio kontrolowane przez użytkownika — *źródeł* (ang. *source*) — i prześledzeniu przepływu tych wartości do potencjalnie podatnych podprogramów — *ujść* (ang. *sink*). Jeśli taka dana zostanie przekazana do ujścia bez jej wcześniejszej walidacji i neutralizacji (ang. *sanitize*), to zmienna, z której ona pochodzi jest oznaczana jako problematyczna.
Analizatory statyczne są przydatne, ale należy pamiętać, że mogą one generować zarówno wyniki fałszywie pozytywne, jak i fałszywie negatywne.

Analizatory statyczne

Przykłady

Zestawienie analizatorów statycznych stworzonych z myślą o wykrywaniu podatności (ang. *Static Application Security Testing (*SAST*) Tools*) można znaleźć na stronie OWASP [4]. Jednym z najstarszych i najprostszych narzędzi tego typu jest ▸ splint, bazujący na historycznie pierwszym analizatorze statycznym: LINT. Splint jest programem z tekstowym interfejsem użytkownika, przeznaczonym do analizy kodu napisanego w języku C. Jego dokładność można zwiększyć stosują adnotacje, czyli specjalne komentarze w kodzie, które są przez niego interpretowane. Do ciekawych analizatorów należy OWASP ▸ Dependency-Check. Nie jest to typowy analizator statyczny, bo ma tylko jeden cel działania, którym jest wykrywanie, czy w kodzie programu nie są użyte biblioteki dostarczane przez innych producentów, które mają znane podatności. W tym celu korzysta on z baz danych znanych podatności.

Analizatory statyczne

Przykłady

Jednym z najczęściej używanych narzędzi do analizy statycznej jest SonarQube. Jest to program, które umożliwia analizę kodu napisanego w różnych językach programowania, dzięki wtyczkom (ang. *plugins*). Daje on możliwość definiowania *bramek jakości*, które pozwalają sprawdzić, czy kod ma określony poziom jakości i podjąć decyzję, czy może być zintegrowany z resztą projektu, czy powinien być wcześniej poprawiony. SonarQube wylicza również *dług techniczny* (ang. *technical debt*), czyli czas, liczony w *osobodniach*, który trzeba będzie poświęcić, aby usunąć z kodu wszystkie odkryte przez niego problemy. Najczęściej analizator ten integrowany jest w środowisku CI/CD.

Analizatory dynamiczne

Analizatory dynamiczne używane są w procesie testowania dynamicznego, gdzie pozwalają obliczyć pokrycie kodu testami, tj. określić ile instrukcji weryfikowanego kodu jest wykonywanych podczas testów. Mogą one również służyć do innych celów, jak np. wykrywanie wycieków pamięci lub problemów ze współbieżnością [1]. Użycie takiego analizatora można połączyć z *testowaniem rozmytym* (ang. *fuzz testing*). Polega ono na automatycznym generowaniu dużej liczby egzemplarzy danych wejściowych wywodzących się z określonego wzorca lub wzorców, przekazywaniu ich do testowanego programu i sprawdzaniu jak on je przetwarza. To pozwala wykryć problemy z zarządzeniem pamięcią, odpornością na błędy i raportowaniem o wyjątkach. Tego typu metoda, użyta w ramach testów typu *ramię w ramię*, umożliwia także zweryfikowanie, czy nowa wersja używanego komponentu zachowuje się tak samo jak jej poprzedniczka, dla takich samych danych wejściowych.

Analizatory dynamiczne

Przykłady

Jednym z najpopularniejszych analizatorów dynamicznych jest Valgrind [5]. Jest on w zasadzie platformą (ang. *framework*) do tworzenia narzędzi analizy dynamicznej, a nie zamkniętym rozwiązaniem. Dzięki temu można dołączać do niego moduły, które wykrywają problemy z zarządzaniem pamięcią, synchronizacją wątków, a także pozwalają sporządzić profil działania oprogramowania, który może być przydatny w optymalizacji. Współczesne kompilatory, takie jak GCC i clang umożliwiają, w systemach kompatybilnych z Uniksem, wstrzykiwanie do funkcji z rodziny `malloc()` kodu wykrywającego problemy z przydzielaniem i zwalnianiem pamięci. Funkcje, związane z tym mechanizmem są zadeklarowane w pliku nagłówkowym `mcheck.h`. Należą do nich między innymi: `mtrace()` i `mcheck()`. Jest również dostępne narzędzie `mtrace`, które ułatwia analizę raportów generowanych przez funkcję o tej samej nazwie.

Literatura I

- OWASP *DevSecOps Guideline — v*-0.2. URL: https://owasp.org/www-project-devsecops-guideline/latest/ (term. wiz. 27. 06. 2023).
- Ryan Dewhurst i in. OWASP *Source Code Analysis*. URL: https://owasp.org/www-community/controls/Static_Code_Analysis (term. wiz. 27. 06. 2023).
- Dave Wichers i in. OWASP *Source Code Analysis Tools*. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools (term. wiz. 27. 06. 2023).
- *Valgrind*. URL: https://valgrind.org/ (term. wiz. 27. 06. 2023).

Literatura II

Heather Adkins i in. *Building Secure & Reliable Systems: Best Practices for Design, Implementing Maintaining Systems*. Sebastopol, CA, USA: O'Reilly Media, 2020. URL: https://sre.google/books/building-secure-reliable-systems/.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!