Programowanie Defensywne

Przeglądy bezpieczeństwa kodu

Arkadiusz Chrobot

Katedra Systemów Informatycznych

10 maja 2024

Plan

Wprowadzenie

Przeglądy kodu (ang. *code review*) dzielą się na kilka kategorii. Każda z nich może pomóc zidentyfikować problemy związane z bezpieczeństwem:

- przegląd API/projektu — pozwala zbadać wpływ architektury oprogramowania na jego bezpieczeństwo i sprawdzić, czy zastosowano zabezpieczenia dla używanego API,
- przegląd utrzymania kodu (ang. *maintainability code review*) — uwzględnia standardy tworzenia kodu, które obowiązują w danej firmie/organizacji i pozwala zidentyfikować złożone komponenty, które mogą zawierać podatności bezpieczeństwa,
- przegląd integracji kodu — również związany jest ze standardami tworzenia kodu danej organizacji/firmy i sprawdza, czy kod stworzony przez organizacje/firmy zewnętrzne został uznany i zaakceptowany przez kierownictwo projektu,
- przegląd testów — dotyczą przypadków testowych pokrywanych przez testy jednostkowe, np. czy prawidłowo jest wykonywana obsługa wyjątków.

Cele i korzyści

Celem przeglądu kodu jest wyeliminowanie defektów, w tym podatności, na jak najwcześniejszym etapie rozwoju oprogramowania (ang. *Software Development Life Cycle*). Łącznie z automatycznymi i manualnymi *testami penetracyjnymi* pozwala znacznie zwiększyć efektywność i opłacalność weryfikacji bezpieczeństwa aplikacji.
Przegląd nakierowany na bezpieczeństwo pozwala realnie ocenić ryzyko związane z odkrytymi podatnościami, gdyż recenzent ma bezpośredni wgląd w kod aplikacji i zna jego kontekst. Pozwala to ocenić prawdopodobieństwo powodzenia ataku z wykorzystaniem podatności oraz rozmiar potencjalnych strat nim spowodowanych.
Przeglądy bezpieczeństwa kodu pozwalają nie tylko znaleźć podatność, ale zidentyfikować jej *pierwotną przyczynę*, bo recenzent najczęściej zna nie tylko kod aplikacji, ale również zewnętrzne komponenty od których on jest uzależniony oraz konfigurację aplikacji.

Cele i korzyści

Przegląd kodu pozwala również stwierdzić, czy zostały w aplikacji wprowadzone odpowiednie zabezpieczenia i czy zostały poprawnie zastosowane.
Kod źródłowy współczesnego oprogramowania jest bardzo duży, zatem nie jest możliwe wykonanie całościowego przeglądu kodu. W identyfikacji komponentów, na które warto zwrócić uwagę mogą pomóc narzędzia analizy statycznej. Należy jednak zachować ostrożność podczas korzystania z nich. Ich wynik zawsze musi być poddany ocenie człowieka posiadającego odpowiednia wiedzę i doświadczenie, nie tylko w zakresie bezpieczeństwa, ale także budowy badanej aplikacji.

Cele i korzyści

Badania przeprowadzone w 2015 roku wykazały, że przeglądy bezpieczeństwa kodu dają lepsze wyniki w wykrywaniu problemów związanych z podatnościami, zapewnianiem prywatności i logiką biznesową niż testy penetracyjne i zautomatyzowane skanowanie kodu w poszukiwaniu wzorców wskazujących na problemy. Jednakże skuteczność tej techniki jest porównywalna z pozostałymi w przypadku weryfikacji zgodności ze standardami i mała w przypadku badania dostępności. Zatem metoda ta musi być łączona z pozostałymi wymienionymi, aby zapewnić odpowiedni poziom weryfikacji bezpieczeństwa oprogramowania.

Wprowadzenie w firmie/organizacji przeglądów bezpieczeństwa kodu, które byłby skuteczne, wymaga powołania *głównego zespołu ds. bezpieczeństwa* (ang. *core security team*), składającego się z programistów znających zagadnienia bezpieczeństwa i ekspertów od bezpieczeństwa (ang. *Subject Matter Experts* (SME)). W dużych firmach/organizacjach obok takiego zespołu mogą powstawać mniejsze, które zazwyczaj nazywa się *satelitarnymi*.

Cele i korzyści

Przeglądy kodu pozwalają zapoznać się młodszym członkom zespołu z kodem i standardami tworzenia bezpiecznego oprogramowania (ogólnymi i stosowanymi w danej firmie/organizacji). Raporty z przeglądów (np. w postaci zapisów w oprogramowaniu, takim jak GitLab lub JetBrains Surface), pozwalają udokumentować decyzje projektowe i prześledzić rozwój aplikacji. Przegląd bezpieczeństwa kodu pozwala również wcześnie wykryć symptomy problemów z integracją komponentów i je usunąć, zanim staną się realne.

Cele i korzyści

Przeglądy kodu można zintegrować z manualnymi i automatycznymi testami penetracyjnymi. Znając strukturę kodu i wykryte w nim defekty związane z bezpieczeństwem można dobrać tak przypadki testowe, aby zweryfikować, czy dana podatność została prawidłowo usunięta, bądź ocenić, na ile realne jest wykorzystanie jej przez intruza. Takie przypadki testowe mogą być także wykorzystane w *testach regresyjnych*, które sprawdzają, czy nowo dodany kod nie wprowadza z powrotem usuniętej podatności, albo nie czyni jej łatwiejszą do wykorzystania. W końcu, te same przypadki testowe można wykorzystać do testowania innych miejsc w aplikacji, celem sprawdzenia, czy ta sama podatność w nich nie występuje. Zatem przeglądy kodu umożliwiają zastosowanie *metody testowania strukturalnego* (*przezroczystej skrzynki*) w testach penetracyjnych.

Cele i korzyści

Przeglądy bezpieczeństwa kodu są swoiste dla weryfikowanej aplikacji. Pozwalają wykryć niebezpieczne zakończenia przepływu sterowania (np. nieprawidłowe procedury obsługi wyjątków) oraz problemy ze współbieżnością. Jeśli w aplikacji już zastosowano mechanizmy zabezpieczające, to przegląd kodu pozwala ocenić na ile są one odpowiednie i czy są stosowane w sposób spójny w poszczególnych częściach tego oprogramowania. Przegląd kodu umożliwia także przeprowadzenie *analizy typu źródło i ujście*. *Źródłem* jest wejście danych do aplikacji, a *ujściem* miejsce gdzie są one przetwarzane. Taka analiza umożliwia prześledzenie danych od źródła do ujścia i ocenę zabezpieczeń tego ostatniego.

Cele i korzyści

Przeglądy bezpieczeństwa kodu są wymagane przez takie standardy, jak PCI-DSS i PA-DSS. Pozwalają one określić także zgodność weryfikowanego oprogramowania z innymi standardami np. HIPAA.

Organizacja przeglądu kodu

Firma/organizacja chcąca wprowadzić przeglądy kodu musi przewidzieć i zapewnić na nie odpowiednie zasoby (czas, ludzie, narzędzia). Powinna wprowadzić także priorytety przeglądów dla poszczególnych komponentów weryfikowanego oprogramowania. Mniej ważne komponenty mogą nie przechodzić pełnego cyklu przeglądu i mogą być sprawdzane tylko przez jedną osobę. Ważne komponenty muszą przejść cały cykl przeglądu.

Umiejscowienie przeglądu kodu w SDLC

Przeglądy kodu wykazują dużą skuteczność w wykrywaniu i neutralizowaniu podatności, jeśli są przeprowadzane na kodzie, który jeszcze nie trafił do repozytorium. Opóźnia to pracę nad oprogramowaniem, ale zapewnia, że niebezpieczny kod nie trafi do niego, a wszelkie podatności łatwiej będzie usunąć. Przegląd kodu może być też prowadzony po dodaniu kodu do repozytorium. Zaletą tego podejścia jest to, że prace nad aplikacją nie są opóźniane, ale trudniej jest usnąć wykryte podatności, bo mogą powstać zależności weryfikowanego kodu z kodem napisanym przez innych programistów. W podejściu zwinnym takie przeglądy przeprowadza się w ramach tak zwanych *iteracji bezpieczeństwa* (ang. *security sprints*). Przeglądy bezpieczeństwa kodu mogą być także wykonywane w ramach okresowych *audytów*, które mogą być także przeprowadzane po wykryciu jakiejś podatności. W takich wypadkach analizowana baza kodu jest duża i konieczne jest wspomaganie tego procesu narzędziami analizy statycznej.

Raport

Raport z przeglądu kodu nie powinien być rozwlekłym dokumentem. Nie musi także być wykonany w wersji papierowej. Współcześnie dostępne są narzędzia (choćby te wspomniane wcześniej), które pozwalają te raporty tworzyć, przechowywać i zarządzać nimi. Raport z przeglądu bezpieczeństwa kodu powinien zawierać:

1. czas i datę wykonania przeglądu,
2. nazwę aplikacji i zweryfikowanego modułu,
3. nazwiska programistów i recenzentów,
4. identyfikator sprawdzanego zadania, cechy (ang. *feature*),
5. krótką klasyfikację i priorytetyzację podatności, jakie czynności są konieczne do ich usunięcia i czy należy powtórzyć przegląd,
6. odwołania do dokumentacji (testów, projektu, wymagań, modelu zagrożeń),
7. listę kontrolną, jeśli jej użyto,
8. testy jakie programista wykonał przed przeglądem,
9. czy i jakich narzędzi do weryfikacji kodu użyto przed przeglądem.

Ryzyko w przeglądach kodu

Ryzyko związane z wykrytymi podatnościami może zostać *zredukowane*, *zaakceptowane* lub *uniknięte*. *Przeniesienie* ryzyka nie jest na tym etapie prac nad oprogramowaniem możliwe. Akceptacja następuje tylko wtedy, gdy podatności nie da się usunąć bez spowodowania niespełnienia reguł logiki biznesowej. Redukcja jest normalną procedurą usuwania podatności, a unikanie sprowadza się do decyzji, aby nie dodawać kodu z podatnością do repozytorium, jeśli stanowi to zbyt duże ryzyko.

Lista kontrolna

Lista kontrolna jest dokumentem, który może funkcjonować jako forma standardu dla przeglądów bezpieczeństwa w firmie/organizacji oraz jako wytyczne dla prac nad rozwojem oprogramowania. Stanowi ona zbiór pytań weryfikacyjnych, na które odpowiedzi są zawsze jednoznaczne (tak/nie) i które mogą należeć do następujących kategorii: weryfikacja danych, uwierzytelnianie, zarządzenie sesją, autoryzacja, kryptografia, obsługa wyjątków, logowanie, bezpieczna konfiguracja i architektura sieciowa.

Przygotowanie

Przygotowanie do przeglądu obejmuje zapoznanie się z działaniem, budową i dokumentacją aplikacji oraz wykonanie modelowania zagrożeń.

Metryki

Metryki pozwalają ocenić jakość bezpieczeństwa kodu, ale także sam proces przeglądu. Zaliczamy do nich:

liczba wierszy kodu (LOC) zgrubne oszacowanie rozmiaru aplikacji,

liczba punktów funkcyjnych liczba instrukcji wykonujących określoną operację (np. pojedyncza klasa),

gęstość defektów liczba defektów przypadających na wiersze kodu — wysokopoziomowa ocena jakości,

gęstość ryzyka słowna ocena ryzyka (niskie, średnie, wysokie) przypadająca na liczbę wierszy kodu,

złożoność cyklometryczna liczba prostych warunków+1 — pozwala ocenić poziom skomplikowania kodu,

częstość inspekcji (ang. *inspection rate*) pozwala oszacować wymagany czas trwania przeglądu wyrażony liczbą analizowanych wierszy kodu (podstawa: 250),

Metryki

częstość wykrywania defektów liczba wykrytych defektów w jednostce czasu — pozwala oszacować efektywność zespołu dokonującego przeglądu,

częstość ponownych inspekcji defektów pozwala oszacować regresję w kodzie.

Zalecenia techniczne — przykład dla JavaScript

Ponieważ JavaScript jest stosowany po stronie klienckiej, to najczęściej jest celem ataków. Najpoważniejsze podatności, to np. XSS bazujące na DOM. Wykrycie ich może być utrudnione ze względu na gmatwanie (ang. *obfuscation*) i kompresję kodu JS. Zaleca się, aby kod został poddany przeglądowi przed takimi zabiegami, choć przeczy to zasadzie, że badany ma być kod przygotowany do instalacji w środowisku produkcyjnym. Dodatkowym utrudnieniem może być to, że kod JS może być częścią frameworka, np. Angular, JQuery, Java Server Faces, Microsoft .NET. Zaleca się zatem połączenie manualnego przeglądu, z użyciem narzędzi do analizy statycznej. Celem przeglądu jest analiza typu źródło i ujście z użyciem *testów skażenia* (ang. *taint testing*). Polegają one na wprowadzeniu do źródła „skażonych” danych i sprawdzeniu, czy dotrą do źródła i jaki będzie tego skutek.

Zalecenia techniczne — przykład dla JavaScript

Przykłady podatnego kodu:

```
<html>
<script type="text/javascript">
var pos=document.URL.indexOf("name=")+5;
document.write(
 ↪  document.URL.substring(pos,document.URL.length));
</script>
<html>
```

Zalecenia techniczne — przykład dla JavaScript

Przykłady podatnego kodu:

```
var url = document.location.url;
var loginIdx = url.indexOf(‘login’);
var loginSuffix = url.substring(loginIdx);
url = ‘http://mySite/html/sso/’ + loginSuffix;
document.location.url = url;
```

Zalecenia techniczne — przykład dla JavaScript

Intruz może przesłać adres URL w np. takiej postaci:
`http://hostname/welcome.html#name=<script>alert(1)</script>`
Spowoduje to wykonanie kodu w nim zawartego przez przeglądarkę klienta. Drugi fragment niekoniecznie musi być podatny, ale stwierdzenie tego wymaga dokładniejszej analizy. Inne przykłady:
Źródło: `document.url`
Ujście: `document.write()`
Wynik: `document.write("<script>malicious code</script>");`
Intruz mógł kontrolować następujące elementy DOM: `document.url`, `document.location`, `document.referrer`, `window.location`.

Zalecenia techniczne — przykład dla JavaScript

Rozważyć następujące punkty:

1. Kod przekazywany do funkcji `eval()` jest wykonywany z uprawnieniami wykonawcy, czyli złośliwy kod jest wykonywany z uprawnieniami strony na przeglądarce klienta.
2. Złośliwy kod może być skonstruowany przez intruza, który zna kontekst przekazania tego kodu do `eval()`.
3. Nie należy używać `eval()` lub `new Function()` do przetwarzania JSON.

Wymagania dla poprawności JavaScript:

1. Walidacja danych musi być wykonana po stronie serwera — klient może mieć nawet wyłączony interpreter JavaScript.
2. Sprawdzaj wszystkie ataki XSS DOM, nie ufaj danym dostarczonym przez użytkownika, znaj źródła i ujścia (tzn. zmienne, które zawierają dane dostarczone przez użytkownika).
3. Sprawdzaj, czy wykryto podatności w bibliotekach JavaScript i aktualizuj je często.

Pytania

?

Koniec

Dziękuję Państwu za uwagę!