Statyczna analiza kodu źródłowego

Zimowisko Linuksowe 2012

Marcin Stępnicki

Statyczna analiza kodu źródłowego

Zimowisko Linuksowe 2012

Marcin Stępnicki

Analiza statyczna – podstawy teoretyczne

- Programy badające inne programy
- Problem matematyczno-informatyczny – udowodnienie zgodności modelu ze specyfikacją
- Problem jest w ogólności nierozwiązywalny - „problem stopu”, nie istnieje możliwość stwierdzenia czy program realizujący dany algorytm zatrzyma się
- tzw. logika Hoare'a służąca do opisu poprawności algorytmów, w dużym skrócie: jeżeli kod C będzie miał na wejściu stan spełniający warunek P oraz zakończy swoje działanie, to na wyjściu będzie miał stan spełniający warunek Q
- Na szczęście nie trzeba tego wiedzieć aby korzystać z przedstawionych narzędzi :-)

Po co to komu?

- Jak zwykle - dla pieniędzy
  - Koszt poprawienia błędu wykrytego wewnątrz firmy jest niższy niż koszt poprawienia błędu wykrytego przez klienta (szczegóły na następnym slajdzie)
  - Koszt człowieka – testera jest wyższy niż koszt oprogramowania mogącego to samo zrobić automatycznie
  - Testerzy z reguły nie mają czasu na przejście wszystkich ścieżek
  - Testerzy również popełniają błędy
- Człowiek jest omylny – nie ma idealnych programistów
- Popularność języków interpretowanych – (w uproszczeniu) informacja o błędzie pojawia się dopiero w momencie wykonywania błędnego kodu (JavaScript/Python/Ruby/PHP/Perl...)

Po co to komu?

- Jak zwykle - dla pieniędzy
  - Koszt poprawienia błędu wykrytego wewnątrz firmy jest niższy niż koszt poprawienia błędu wykrytego przez klienta (szczegóły na następnym slajdzie)
  - Koszt człowieka – testera jest wyższy niż koszt oprogramowania mogącego to samo zrobić automatycznie
  - Testerzy z reguły nie mają czasu na przejście wszystkich ścieżek
  - Testerzy również popełniają błędy
- Człowiek jest omylny – nie ma idealnych programistów
- Popularność języków interpretowanych – (w uproszczeniu) informacja o błędzie pojawia się dopiero w momencie wykonywania błędnego kodu (JavaScript/Python/Ruby/PHP/Perl...)

Koszty błędów

	Time Detected				
Time Introduced	Requirements	Architecture	Construction	System Test	Post-Release
Requirements	1	3	5-10	10	10-100
Architecture	-	1	10	15	25-100
Construction	-	-	1	10	10-25

Static code analysis

- Steve McConnell, "Code Complete, 2nd Edition" Microsoft Press, ISBN: 0-7356-1967-0

Jakie kategorie błędów można wykryć?

- To tylko przykłady nie wyczerpujące tematu:
  - brak zmiennych globalnych (np. użycie procedury z biblioteki bez jej wcześniejszego importu)
  - przekazywanie nieprawidłowej liczby parametrów do funkcji
  - użycie formatowania tekstu niezgodnego z zadeklarowanym typem zmiennych
  - użycie nieistniejących metod lub atrybutów
  - przedefiniowanie funkcji/klasy/metody/zmiennej w danej przestrzeni nazw
  - użycie zmiennej przed jej ustawieniem
  - nieużywane zmienne
  - nieużywane funkcje/klasy/metody
  - brak tzw. docstringów
  - wycieki pamięci
  - sytuacje wyścigu
  - różnice między wersjami 32- i 64-bitowymi
  - niezgodności licencyjne

Rodzaje analizatorów – 1/2

- Proste analizatory składni sprawdzające zgodność z rekomendowanym stylem programowania (nazewnictwo, wcięcia, komentarze) i wyłapujące proste błędy składniowe:
  - Historycznie pierwszy program Lint i jego mutacje dla różnych języków programowania – FLOSS:
    - Splint (http://splint.org/)
    - JSLint (http://www.jslint.com/)
    - PHPLinter (https://github.com/robotis/PHPLinter)
      - php -l
    - Pylint (http://www.logilab.org/857)
    - Laser (http://carboni.ca/projects/laser)

Rodzaje analizatorów – 2/2

- Rozbudowane narzędzia próbujące wniknąć w logikę działania programu:
  - Frama-C (http://www.youtube.com/watch?v=J_xgbO5-32k), FLOSS, C/C++
  - Findbugs (http://findbugs.sourceforge.net/), FLOSS, Java
  - Valgrind – inna kategoria (o tym później) - (http://valgrind.org/), FLOSS, C
  - RATS (https://www.fortify.com/ssa-elements/threat-intelligence/rats.html), FLOSS, C/C++/Perl/PHP/Python
  - Klocwork (własnościowy)
  - Coverity (własnościowy)
  - PVS-Studio (własnościowy)
  - /analyze (własnościowy)
  - Purify – jak Valgrind (własnościowy)

Dodatkowe możliwości i inne narzędzia

- integracja z systemami kontroli wersji / narzędziami do akceptacji (review) kodu (przykład: nie można wprowadzić do repozytorium kodu niespełniającego zasad stylistycznych)
- Stopień pokrycia kodu testami (z reguły są to osobne narzędzia):
  - http://en.wikipedia.org/wiki/Code_coverage
- Inna kategoria, ponieważ wymagają uruchomienia programu (nie jest to analiza statyczna):
  - profilery - identyfikowanie wąskich gardeł, zwłaszcza jeżeli chodzi o prędkość wykonywania
  - Valgrind / Purify – śledzą zachowanie programu w trakcie jego wykonywania, potrafią powiązać go z rzeczywistym kodem źródłowym – trochę analizatory, trochę profilery, w sumie osobna kategoria

Zbytnie zaufanie maszynom 1/4

- Duża liczba false-positives mająca realny wpływ na jakość – nie zawsze należy naprawiać dany błąd tylko dlatego że kazał nam to zrobić jakiś automat
- Czy są na sali developerzy Debiana?

Zbytnie zaufanie maszynom 2/4

- Witaj Bartek :-)

Zbytnie zaufanie maszynom 3/4 - OpenSSL

- OpenSSL - przykład błędu wynikającego nie tylko z nieprawidłowej interpretacji kodu, ale również niedoskonałej komunikacji i wysłania posta na niewłaściwą listę dyskusyjną
  - http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=363516
  - http://marc.info/?t=114651088900003&r=1&w=2
  - http://www.links.org/?p=327#comment-176642
- Szczegóły (w dużym skrócie):
  - Valgrind raportował użycie niezainicjalizowanego fragmentu pamięci – jednak była to tylko **jedna** ze zmiennych używanych do wytworzenia entropii i nie oznaczało to, że cały bufor użyty do jej wytworzenia nie został zainicjalizowany
  - „poprawienie” błędu psuło losowość generowanych kluczy – można było wygenerować wszystkie możliwe klucze (32K) i podszywać się pod oryginalnego wystawcę

Zbytnie zaufanie maszynom 4/4 – VirtualDub

- VirtualDub jest bardzo starym (w pozytywnym, „dojrzałym”, sensie programem służącym do edycji plików video – niektóre procedury były napisane w assemblerze i dopiero później przekonwertowane do C)
- Dlaczego wychodzimy poza pole zdefiniowane dla struktury? http://www.virtualdub.org/blog/pivot/entry.php?id=359

```
struct ConvoluteFilterData {
    long m[9];
    long bias;
    void *dyna_func;
    uint32 dyna_size;
    uint32 dyna_old_protect;
    bool fClip;
};
```

- long rt0=cfd->m[9], gt0=cfd->m[9], bt0=cfd->m[9];

Przykład prawdziwego błędu

- Czym jest wielowątkowość?
  - Ten sam proces uruchamiany jednocześnie wiele razy (nieważne - jeden fizyczny procesor, hyperthreading, wiele rdzeni).
  - Wymaga innego podejścia do programowania – do niedawna wielozadaniowość była tylko złudzeniem wynikającym z powolnej percepcji człowieka

```
void incr()
{
  x++;
}
```

  - Co robi ten kod:
    - (1) pobiera wartość zmiennej x;
    - (2) dodaje do niej jedynkę
    - (3) zapisuje rezultat z powrotem do zmiennej x.
    - Jeżeli dwa wątki zrobią to równolegle, to oba odczytają tę samą wartość, dodadzą do niej jeden i zapiszą tę wartość – czyli zmienna zwiększy się tylko o jeden, zamiast o dwa
  - Jak to naprawić:
    - Mutex, „synchronized”, inne...

Narzędzia FLOSS

- Istnieje wiele możliwości, ale najciekawsze są te które umożliwiają analizę kodu za pomocą wielu narzędzi jednocześnie
- „One Ring to Rule Them All”:
  - Yasca (http://www.yasca.org)
  - Sonar (http://www.sonarsource.org/)
- Oprócz przydatności dla programistów gwarantują „awesomeness factor” dla menadżerów i zarządu

Yasca – wstęp

- Powstał z rzeczywistej potrzeby konsultanta IT – Michaela Scovetty
- Napisany w PHP (na licencji BSD), obsługa dodatkowych narzędzi (mogą one dotyczyć dowolnego języka) jest dostępna jako pluginy
- Działa pod wszystkimi popularnymi systemami operacyjnymi – jednak niektóre pluginy mogą wymagać narzędzi dostępnych tylko dla konkretnego systemu
- Łatwość tworzenia własnych rozszerzeń

Yasca – przykładowy raport

- Potencjalne błędy przepełnienia bufora
- Potencjalne błędy przy formatowaniu łańcuchów tekstowych
- Problemy z entropią
- Nieużywane zmienne i funkcje
- Niezainicjalizowane zmienne
- Wycieki pamięci
- Niebezpieczne konstrukcje
- Widoczność plików testowych/konfiguracyjnych
- Wyciek wrażliwych informacji
- Potencjalne konflikty licencyjne
  - http://embraceunity.com/wp-content/uploads/2010/09/Yasca-Report-20100911054536

Nieprzekonany?

- przykłady błędów w różnych programach open source: http://software.intel.com/en-us/articles/90-errors-in-open-source-projects/
- Trochę statystyki:
  - wykryte przez Coverity błędy: średnio 1 na 1000 linijek kodu (http://tinyurl.com/79eqyzd)
    - Linijki kodu w Windows XP: ok. 45mln. (http://www.knowing.net/index.php/2005/12/06/how-many-lines-of-code-in-windows/)
    - Linijki kodu w Debianie Lenny: ok. 320 mln. (http://libresoft.dat.escet.urjc.es/debian-counting/lenny/)
- John Carmack:
  - "*It is irresponsible to not use it*" - http://altdevblogaday.com/2011/12/24/static-code-analysis/
- Dave Revell:
  - "*The more I push code through static analysis, the more I'm amazed that computers boot at all.*"

Pytania

Q & A

Dziękuję za uwagę