Bezpečné programovanie 1

RNDr. Richard Ostertág, PhD. (ostertag@dcs.fmph.uniba.sk)

Katedra informatiky
Univerzita Komenského, Bratislava

Zdroje

- SEI CERT Coding Standards (for C, C++, Android, Java, Perl)
  https://www.securecoding.cert.org/
- Secure Coding Guidelines for the Java SE
  http://www.oracle.com/technetwork/java/seccodeguide-139067.html
- Secure Programming for Linux and Unix HOWTO
  https://tldp.org/HOWTO/pdf/Secure-Programs-HOWTO.pdf
- D. B. Johnsson, D. Deogun, D. Sawano: Secure by Design
  https://www.manning.com/books/secure-by-design

Verejné databázy zraniteľností

- MITRE CVE (http://cve.mitre.org/)
  - Common Vulnerabilities and Exposures
  - Common Vulnerabilities Enumeration
  - obsahuje linku na NIST NVD
- MITRE CWE (http://cwe.mitre.org/)
  - Common Weakness Enumeration
  - Komplexný slovník CWE (http://cwe.mitre.org/data/slices/2000.html)
  - 25 najnebezpečnejších softvérových chýb (http://cwe.mitre.org/top25/)
- NIST NVD (https://nvd.nist.gov/)
  - National Vulnerability Database (založená na CVE)
  - kompletná informácia o dodávateľovi a produkte
  - pridáva klasifikáciu zraniteľností

SD3 – Secure by Design, by Default, in Deployment

- systém by mal byť od začiatku navrhnutý s ohľadom na bezpečnosť
- vývojár by mal vedieť, aké možnosti sú nebezpečné
- všetky nebezpečné možnosti by mali mať bezpečné predvolené hodnoty
- ani zákazník nepozná systém lepšie, takže inštalačný a konfiguračný program by mal poskytovať bezpečné predvolené hodnoty
    - výnimky z tohto pravidla by mali zobrazovať varovania

Analýza kódu – statická (bez spustenia kódu)

- vzájomná kontrola návrhu a kódu (design and code review)
- aplikácie pre kontrolu štýlu (code style checker)
- aplikácie pre statickú kontrolu sémantických chýb (linter)
  - nepoužitie deklarovanej premennej (nemusí nájsť všetky)
  - použitie premennej bez jej inicializácie (môže nájsť niečo navyše)
  - tieto problémy sú ťažké $\Rightarrow$ iba konzervatívna aproximácia

```
1  if p:
2    x = 0
3  else:
4    y = 0
5
6  if q:
7    y = 1
8  else:
9    x = 1
```

Analýza kódu – dynamická (počas behu)

- všetka alokovaná pamäť bude uvoľnená (napr. Memcheck od Valgrind)
- kontrola splnenia invariantov
  - pre-conditions a post-conditions
  - assert

```python
from math import sqrt

def fun(x):
        assert x > 0, "x by malo byť kladné"
        return sqrt(x)

print(fun(16))
  # 4.0

print(fun(-1))
  # AssertionError: x by malo byť kladné
```

Používať jednoznačný programátorský štýl

Čo autor skutočne zamýšľal v nasledovnom PHP kóde?

- `if (!$a)` ...
  - `if ($a === false)`
  - `if ($a === 0)`
  - `if ($a === NULL)`

- `if ($a == "{}")` ...
  - `if ($a === "{}")`
  - `if ($a === NULL)`

Prípony vykonateľných súborov (Windows NT)

- Po zadaní príkazu bez prípony, sa postupne skúšajú prípony z premennej prostredia PATHEXT.
- Preddefinovaná hodnota PATHEXT je:

  .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.CPL
- Zmenou PATHEXT môže útočník spôsobiť spustenie inej (než očakávanej) aplikácie.

- Podobné problémy spôsobuje aj premenná prostredia PATH.
  - Určuje poradie adresárov v ktorých systém hľadá zadaný program.
    - Relevantné aj pre Linux.

White list (allow list) vs. black list (deny list)

- Bezpečnosť by sa nemala zakladať na vymenovaní nebezpečných vecí (black list).
  - Ľahko sa na niečo zabudne.
- Naopak, mali by sme vymenovať bezpečné veci a ostatné zamietnuť (white list).
- Príklad nesprávneho prístupu:
  - Pomocou black listu zakázať špecifickú formu objaveného útoku.
  - Útočník pravdepodobne nájde inú cestu, ktorou black list obíde.

Zraniteľnosti vo formátovacom reťazci – C

Vzniká vložením nedôveryhodných dát do formátovacieho reťazca.

- Čo je formátovací reťazec?
  - `printf("Name: %s (age: %11d)", person, age);`
    Name: Einstein (age: 142)
- Obvzlášť nebezpečné je `"%n"`.
  - “Nothing printed. The argument must be a pointer to a signed int, where the number of characters written so far is stored.”

Zraniteľnosti vo formátovacom reťazci – C

- Neuvedomenie si, že funkcia interpretuje text ako formátovací reťazec:

```
snprintf(str, sizeof(str), "Wrong password (user %s)", user); // OK
syslog(LOG_WARNING, str); // Zle
```

  - funkcia `void syslog(int priority, const char *format, ...)` používa svoj druhý argument ako formátovací reťazec
  - `user = "einstein%s%s%s%s"` pravdepodobne spôsobí pád aplikácie
    Segmentation fault (core dumped)
    warning: format not a string literal and no format arguments [-Wformat-security]
- Zlý spôsob výpisu jedného reťazca na viacero miest: `fprintf(log, logmessage);`
  - Správne: `fprintf(log, "%s", logmessage);`

Zraniteľnosti vo formátovacom reťazci – Perl

- Nech skript format2.pl je nasledovný:

```
1 #!/usr/bin/perl
2 $a = "10";
3 printf("Before: $a\n");
4 printf("$ARGV[0]", $a); # <- !!!
5 printf("After: $a\n");
```

- fomat2.pl vypíše:

  Before: 10

  After: 10

- fomat2.pl 123%n vypíše:

  Before: 10

  123After: 3

Zraniteľnosti vo formátovacom reťazci – PHP

- PHP nepodporuje "%n".
- Nech skript format3.php je nasledovný:

```
1 #!/usr/bin/php
2 <?php
3 printf("%s","Hello 1!\n");
4 printf("%s%s","Hello 2!\n"); # <- !!!
5 printf("%s","Hello 3!\n");
6 ?>
```

- fomat3.php vypíše:

  Hello 1!

  PHP Warning: printf(): Too few arguments in format3.php on line 4
  Hello 3!

  - program pokračuje, vypíše sa iba prázdny reťazec
  - možno použiť na potlačenie záznamu do logov

Zraniteľnosti vo formátovacom reťazci – Python

- Python nepodporuje `"%n"`, ani `printf`, ale obsahuje príkaz %.
- Nech skript format4.py je nasledovný:

```
1  #!/usr/bin/python
2  userdata = {"user": "admin", "pass": "usr123"}
3  passwd = raw_input("Password: ")
4  if passwd != userdata["pass"]:
5    print ("Wrong password: " + passwd) % userdata
6  else:
7    print "Welcome %(user)s!" % userdata
```

- Po spustení fomat4.py, môže vyzerať dialóg nasledovne:

  Password: %(pass)s

  Wrong password: usr123
- Nesúlad počtu % a argumentov vedie k výnimke – možno použiť na
  - potlačenie záznamu do logov (ak je nevhodne ošetrená)
  - DoS útok (ak je neošetrená)

Vyčerpanie zdrojov 1/3

- Zdieľané zdroje, ktoré sú vystavené útoku:
  - operačná pamäť
  - diskový priestor
  - prenosová kapacita
  - CPU
  - entropia (pre generovanie náhodných čísiel)
  - tabuľka procesov
  - popisovače súborov
  - databázové a iné servery
  - analytici
  - …
- Môže nastať pokiaľ je k dispozícii iba:
  - konečné množstvo/počet zdrojov
  - konečný výkon (napr. CPU)

Vyčerpanie zdrojov 2/3

- Neobozretné protokoly alebo algoritmy
  - neautentifikované využívanie zdrojov (nerozumné poradie vykonávania)
    - vykonanie série komplexných operácií pred kontrolou oprávnenosti požiadavky
    - napr. pri otvorení spojenia server generuje kľúče ešte pred overením používateľa
  - znásobovanie (cez broadcast, subscriptions, …), asymetrické útoky
    - cena pre útočníka je oveľa nižšia ako pre ochrancu
    - ICMP ping na broadcast adresu s falošnou adresou odosielateľa (smurf attack)
- Chyby v implementácii premenené na zraniteľnosti
  - memory leaks
- Chyby v návrhu
  - absencia kontroly prístupu
  - absencia obmedzení čerpania zdrojov (napr. ulimit)
  - …

Vyčerpanie zdrojov 3/3

- Útoky na algoritmickú zložitosť
  - zneužívajú zložitosť v najhoršom prípade
    - quicksort: $O(n \log n) \to O(n^2)$
    - hash tabuľky: $O(n) \to O(n^2)$
    - regulárne výrazy: $O(n) \to O(2^n)$
  - riešením je používat algoritmy, ktoré nie sú zraniteľné
    - napr. *univerzálne hešovanie* bolo navrhnuté za týmto účeľom
- časovo alebo pamäťovo zložité algoritmy
  - neprerušiteľné úlohy (najmä v jadre OS)

- prorokoly s vnútorným stavom sú nutne náchylnejšie na DoS útoky
- riešením je zmeniť ich na bezstavové
  - idea: zašifrovať stav a poslať ho klientovi
    - nemíňa pamäť na serveri, ale
    - viac zaťažuje CPU a šírku pásma

ReDoS – regular expression denial of service attack

- Po preložení regulárneho výrazu do NKA s m stavmi, môže nasledovať:
  - NKA $\rightarrow$ DKA [konverzia $O(2^m)$ obvykle $O(m)$, vyhľadávanie $O(n)$]
  - backtracking cesty v NKA [vyhľadávanie $O(2^n)$ obvykle $O(n)$]
  - prehľadávanie všetkých ciest v NKA súčasne [$O(m^2 n)$]
  - „lenivá“ konverzia na DKA počas hľadania cesty [$O(m^2 n)$]
- Typický nebezpečný výraz je `"^(a+)+$"`.
- Útočník môže ReDoS aplikovať, keď môže:
  - nebezpečnému RE zadávať nevhodný vstup (napr. `"aaaaaaaaaa!"`)
    - OWASP Validation Regex Repository:
      Java Classname: `"^(([a-z])+.)+[A-Z]([a-z])+$"`
    - ```
      cregex = re.compile(r"^(a+)+$")
      cregex = re.compile(r"^(([a-z])+.)+[A-Z]([a-z])+$")
      match = cregex.match("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa!");
      ```
  - vložiť nebezpečný podvýraz do RE (a potom zadávať nevhodný vstup)
      ```

Chybná správa pamäte

- memory leaks
- viacnásobné uvoľnenie tej istej pamäte
- použitie už uvoľnenej pamäte
- uvoľnenie nesprávnej pamäte
- prístup k pamäti na neplatnej adrese
- únik informácie
  - citlivú informáciu treba prepísať
  - pozor na odloženie pamäte s citlivou informáciou na disk (napr. heslé alebo kľúče)
    - virtual memory, swap space – zamykanie pamäte
    - crash dumps (core files) – zakázať crash dumps