This is an old revision of the document!

Tema 3

- Deadline Soft: 12.01.2018 23:55
- Deadline Hard: 12.01.2018 23:55
- Data publicării: 12.12.2017 04:00
- Data ultimei modificări: 12.12.2017 04:00
- Responsabili:
  - Silvia Pripoae [mailto:pripoaesilvia@gmail.com]
  - Vladimir Diaconescu [mailto:vladimir.diaconescu@cs.pub.ro]

Enunț

Să se analizeze un executabil, atât prin metode statice cât și dinamice, pentru a:

- înțelege funcționalitățile expuse
- înțelege logica programului
- determina un ”serial key” valid
- identifica și exploata potențiale vulnerabilități

Tema are trei părți:

1. Analiză statică și dinamică - 30p

Programul trebuie analizat în mod amănunțit (dar cu mai puțin accent pe detalii) prin metodele învățate în laborator. Ce se urmărește: - înțelegerea codului propriu-zis. Executabilul conține mai multe funcții și variabile globale, identificate doar prin adrese. În cadrul README-ului va trebui să faceți o descriere a acestor funcții și variabile: adresa la care se găsesc în cod, câte argumente au, tipul acestora, ce rezultat(e) întorc, ce nume le-ați atribui, unde sunt folosite, la ce sunt folosite, precum și o descriere de nivel înalt a codului (respectiv, să vă gândiți cum ați scrie într-un pseudocod simplificat, sau cum ați descrie foarte sumar esențialul).

Spre exemplu:

```
.text:00402510 sub_402510      proc near
.text:00402510                 push    ebp
.text:00402511                 mov     ebp, esp
.text:00402513                 push    ecx
.text:00402514                 push    edi
.text:00402515                 mov     edi, [ebp+8]
.text:00402518                 or      ecx, 0FFFFFFFFh
.text:0040251B                 xor     eax, eax
.text:0040251D                 repne scasb
.text:0040251F                 not     ecx
.text:00402521                 add     ecx, 0FFFFFFFFh
.text:00402524                 cmp     ecx, 4
.text:00402527                 jz      short loc_40252D
.text:00402529                 xor     eax, eax
.text:0040252B                 jmp     short loc_4025A0
...
```

Descriere de nivel scăzut: funcția salvează vechiului ebp pe stivă, apoi se actualizează valoarea la vârful curent al stivei. Apoi se salvează pe stivă regiștrii ecx și edi etc.

Descriere de nivel înalt: funcția are un singur argument, o adresă a unui șir. Se verifică dacă lungimea șirului este 4.

Tema urmărește înțelegerea codului într-un sens mai abstract; drept urmare, nu prezintă interes dacă un registru se inițializează la valoarea 0 prin mov eax, 0 sau xor eax, eax.

Toate aceste lucruri trebuie documentate în README.

În cadrul punctajului acestei părți intră opțiunea ”0” din binar, în care trebuie să găsiți un serial key valid. Pentru aceasta, trebuie să înțelegeți în ce constă codul de verificare și ce condiții trebuie să valideze serial key-ul.

2. Identificare vulnerabilități - 20p

Pentru această parte, trebuie să urmăriți cu atenție datele locale din funcții și să testați dacă nu cumva aveți de-a face cu situații de tipul buffer overflow. Documentați în README bug-urile și vulnerabilitățile găsite, cu explicații aferente pentru fiecare.

3. Exploatare - 50p

În ultima parte, va trebui să vă alegeți cu grijă vulnerabilitățile pe care să le exploatați, prin construcția unor date de intrare, în vederea alterării fluxului de execuție într-un sens bine-stabilit.

Această parte este construită sub forma unui ”puzzle” în care trebuie să apelați anumite funcții pentru a putea avea acces la o altă funcție. Datele de intrare vor trebui construite în așa manieră încât să treacă o serie de condiții și în același timp să poată modifica execuția programului.

Scopul exploatării este de a apela funcția care afișează mesajele “Win!”, respectiv “Try harder!” in baza unei verificări, astfel încât condiția să fie îndeplinită (se afișează “Win!”). O soluție în care săriți peste verificare direct la afișarea propriu-zisă a mesajului obține același rezultat, dar nu constituie obiectul temei.

Pentru a avea adrese consistente la fiecare rulare, trebuie să dezactivați ASLR (*Address Space Layout Randomization*). Folosiți comanda de mai jos pentru a crea un shell cu ASLR dezactivat, orice programe rulate din shell având și ele ASLR dezactivat:

```
setarch $(uname -m) -R /bin/bash
```

Setup

Pentru dezvoltarea temei va trebui să folosiți mașina virtuală de Linux de 32 de biți descrisă în secțiunea Mașini virtuale din pagina de resurse. Această mașină virtuală este folosită și pentru verificarea temei pe vmchecker [https://vmchecker.cs.pub.ro/ui/#IOCLA].

Puteți testa anumite funcționalități pe alt sistem, dar implementarea voastră trebuie să meargă pe mașina virtuală. Este posibil ca, datorită mediului diferit, anumite payload-uri să meargă pe un alt sistem dar nu pe mașina virtuală. Folosiți, ca referință, mașina virtuală.

Pentru a obține programul aferent temei, rulați comanda:

```
echo "Prenume Nume Grupa" | nc 141.85.224.119 1337 > tema3
```

Cele 3 câmpuri trebuie să conțină doar litere mici, litere mari, cifre și cratimă. Fiecare temă va fi construită în funcție de datele voastre.

De exemplu, pentru `Ionescu Anca-Monica` de la grupa `323CA`, modul de obținere al temei asociate va fi:

```
echo "Anca Ionescu 323CA" | nc 141.85.224.119 1337 > tema3
```

Comanda de mai sus va crea fișierul `tema3`. Acesta este fișierul propriu-zis al temei care trebuie analizat și rezolvat.

Implementare

La nivelul implementării, aveți de determinat serial key-ul valid, de documentat în README și de generat un fișier cu date de intrare pentru program care să-i altereze execuția în așa fel încât să rezolvați puzzle-ul în starea de ”Win!”

Recomandăm să generați acest fișier de intrare folosind un script Python (sau orice alt limbaj de programare).

Trimitere și notare

Temele vor trebui încărcate pe platforma vmchecker [https://vmchecker.cs.pub.ro/ui/#IOCLA] (în secțiunea IOCLA) și vor fi testate automat. Arhiva încărcată va fi o arhivă `.zip` care trebuie să conțină:

- fișierul `serial_key`, care va conține serial key-ul pentru rezolvarea opțiunii “0” din binar
- fișierul README, care va conține:
    - analiza codului
    - enunțarea și explicarea vulnerabilităților găsite
    - descrierea întregului proces de exploatare, pe pași
- fișierul `payload` pentru program
    - Dacă ați folosit un script atunci adăugați în arhiva submisă și scriptul cu care ați obținut acest payload
        - Dacă nu ați folosit un script atunci adăugați în README modul de obținere a payload-ului.
- fișierul `credentials` care conține numele, prenumele și grupa, exact așa cum au fost folosite pentru a genera tema, fiecare pe câte o line, astfel:

```
<Nume>
<Prenume>
<Grupa>
```

- scriptul (sau scripturile) de generare a fișierului `payload`

Precizări suplimentare

Datele din fișierul `credentials` trebuie să fie corecte și să coincidă cu cele pe care le-ați folosit pentru a obține tema. În cazul în care aveți două prenume, alegeți unul pe care să-l folosiți.

Recomandări

Întrucât binarul nu are simboluri (funcțiile nu au nume, sunt doar adrese), este foarte dificil de lucrat doar cu objdump și gdb, recomandăm IDA Free [https://www.hex-rays.com/products/ida/support/download_freeware.shtml] pentru analiza statică. Vă simplifică viața prin faptul că vă permite să redenumiți variabile și funcții și să adăugați comentarii pe marginea codului, iar codul poate fi afișat sub forma unui graf din care se văd mai clar buclele.

La partea de analiză, încercați să vedeți ”big picture” și să evitați să vă pierdeți în detalii.

Resurse ajutătoare

- Laborator 9: Interfața în linia de comandă, analiza statică și dinamică
- Laborator 10: Gestiunea bufferelor. Buffer overflow
- Laborator 11: Exploatarea memoriei. Shellcodes

iocla/teme/tema-3.1513982908.txt.gz · Last modified: 2017/12/23 00:48 by silvia.pripoae