ՀՀ ԿՐԹՈՒԹՅԱՆ ԵՎ ԳԻՏՈՒԹՅԱՆ ՆԱԽԱՐԱՐՈՒԹՅՈՒՆ
ՀՀ ԳԱԱ ԳԻՏԱԿՐԹԱԿԱՆ ՄԻՋԱԶԳԱՅԻՆ ԿԵՆՏՐՈՆ

ՄԱԳԻՍՏՐՈՍԱԿԱՆ ԹԵԶ

Բաբկեն Վարդանյան Սամվելի

Մասնագիտություն՝	Ինֆորմատիկա և հաշվողական տեխնիկա
Թեմա՝	Սերվերային համակարգերի անվտանգությունը գնահատող ծրագրի մշակում
Գիտական ղեկավար՝	ֆիզ.-մաթ. գիտ. դոկտոր, պրոֆեսոր Մարիամ Հարությունյան

Թույլատրել պաշտպանության '___' '_____' 2016 թ.

Ամբիոնի վարիչ՝	ֆիզ.-մաթ. գիտ. թեկնածու, դոցենտ Վլադիմիր Սահակյան

ԵՐԵՎԱՆ 2016

Կցանկանայի խորին երախտագիտությունս հայտնել իմ ղեկավար, ֆիզ.-մաթ. գիտ. դոկտոր, պրոֆեսոր Մարիամ Հարությունյանին (ԻԱՊԻ), ով ինձ աջակցել և խրախուսել է այս աշխատանքի ժամանակ:

Բաբկեն Վարդանյան

Բովանդակություն

1 Առաջաբան

1.1 Ինչու՞ է տեղեկատվական անվտանգությունը կարևոր

1.1.1 Ի՞նչ է տեղեկատվական անվտանգությունը

Տեղեկատվական անվտանգությունը տեղեկատվական ռեսուրսների չլիազորված օգտագործման կանխման և հայտնաբերման պրոցեսն է: [23]

Տեղեկատվական ռեսուրսների չլիազորված օգտագործման կանխումը չարամիտ չլիազորված անձանց (նաև ասում են «հակառակորդներ», «հարձակվողներ», «ներխուժողներ», «հաքերներ») կողմից ծրագրային ապահովման կամ տվյալների որոշ մասի օգտագործման դեմ ուղղված միջոցառումների համակարգն է:

Տեղեկատվական ռեսուրսների չլիազորված օգտագործման հայտնաբերումը չլիազորված մուտքի փորձի առկայության ստուգման պրոցեսն է: Եթե նման փորձ առկա է, ապա նաև՝ արդոք այն հաջողվել է, և թե կոնկրետ ինչ է տեղի ունեցել: [20]

1.1.2 Ինչու՞ հոգալ տեղեկատվական անվտանգության մասին

Այսօր համակարգիչները և էլեկտրոնային տեխնիկան օգտագործվում են կյանքի գրեթե բոլոր ոլորտներում: Բանկային համակարգի ու ներդրումների ոլորտից մինչև գնումների և հեռահաղորդակցության ոլորտ համակարգիչները դարձել են յուրաքանչյուր բիզնեսի անբաժանելի մասը: Դժվար է նշել մի ոլորտ, որը օգուտ չի քաղել տեղեկատվական տեխնոլոգիաների բուռն զարգացումից:

Չնայած ընկերությունների կողմից պահվող ոչ բոլոր տվյալները կարելի է դասակարգել որպես «հույժ գաղտնի», ցանցային ադմինիստրատորները հավանաբար չեն ուզենա որ անծանոթ անձինք հնարավորություն ունենան հետևել իրենց ընկերության ներքին հաղորդակցությանը, իրենց անձնական ինֆորմացիային, կամ փոփոխություններ կատարեն իրենց վստահված համակարգերում:

Այդ պատճառով տեղեկատվական անվտանգությունը մնում է բիզնեսի և հասարակության առտև ծառացած ամենակարևոր չհաղթահարված խնդիրներից մեկը: [21]

Սերվերի ղեկավարման հնարավորությունը հակառակորդի կողմից ռիսկի տակ է դնում ոչ միայն ընկերությանը, այլ նաև ընկերության հաճախորդներին, ինչպիսիք են օրինակ վեբ կայքի այցելուները:

1.1.3 Ինչու՞ ինչ-որ մեկը կցանկանա կոտրել որոշակի համակարգ

Հակառակորդներին հաճախ չի հուզում, թե ով է օգտագործողը կամ ընկերությունը, որի վրա իրականացվում է հարձակումը:

Հակառակորդի հիմնական նպատակներն են՝

- **Դրամական եկամուտ**
  Կոտրված համակարգչից կամ սերվերից օգտագործողի կամ ընկերության բանկային հաշվի և վարկային քարտի տեղեկությունները գողանալու միջոցով:

- **Բիզնեսի աշխատանքի խոչնդոտում**
  Մի ընկերություն կարող է վարձել հակառակորդին իրենց մրցակցի համակարգչային ցանցում քաոս ստեղծելու նպատակով:

- **Ինֆորմացիայի գողություն**
  Մի ընկերություն կարող է վարձել հակառակորդին իրենց մրցակից ընկերության գաղտնիքները գողանալու և այդպիսով մրցակցային առավելություն ձեռք բերելու նպատակով:

- **DDoS (Distributed Denial of Service - Բաշխված Ծառայության Ընդհատման գրոհ) գրոհներ իրականացնելու նպատակով այլ սերվերների վրա**
  DDoS գրոհի նպատակն է սերվիսը անհասանելի դարձնել՝ տարբեր աղբյուրներից չափազանց շատ հարցումներ իրականացնելու միջոցով: [32]

  Նման հարձակման դեպքում կոտրված սերվերների քանակը ուղիղ համեմատական է գրոհի հաջողությանը:

- **SEO (Որոնման համակարգերի օպտիմալացում)**
  Կոտրված կայքը կարող է օգտագործվել այլ կայքերի SEO-ն բարձրացնելու նպատակով՝ կոտրված կայքում տեղադրելով հղումեր դեպի այդ կայքը:

- **Հենակետ հետագա գրոհների համար**
  Կոտրված սերվերը կարող է օգտագործվել որպես հենակետ՝
  - նույն ընկերության ցանցում հետագա ավելի լայնածավալ հարձակումների համար,
  - ավելի շատ սերվերներին տիրանալը օգնում է հակառակորդին թաքցնել իր ինքնությունը (IP հասցեն)՝ այլ ընկերությունների դեմ հետագա հարձակումների ժամանակ

- **Զվարճանք**
  Հակառակորդը կարող է կոտրել սերվերը զուտ հետաքրքրության կամ զվարճանքի համար:

1.2 Սերվերային անվտանգության ժամանակակից պրակտիկան

Սերվերների անվտանգությունը ապահովելու այսօր ընդունված ամենատարածված պրակտիկաներից են՝

1. **Անջատել կամ ջնջել ոչ անհրաժեշտ սերվիսները**
   Օպերացիոն համակարգերի լռելյայն կոնֆիջուրացիան երբեմն ապահով չէ: Սովորաբար տեղադրված են բազմաթիվ չօտագործվող սերվիսներ, ինչպես օրինակ՝ պրինտ սերվերը, «Սամբա» ֆայլերի բաշխման համակարգը և այլն: Այս սերվիսները մեծացնում են հարձակման հարթությունը, բացելով ավելի շատ հնարավոր եղանակներ չարամիտ օգտագործողի համար՝ համակարգը չարաշահելու նպատակով:

Ադմինիստրատորները պետք է անջատեն կամ մեկուսացնեն բոլոր չօգտագործվող սերվիսները, օրինակ՝ firewall-ի օգնությամբ:

2. **Հեռակառավարում**
   Չպաշտպանված, հանրային ցանցերով մուտքը սերվեր հնարավոր է դարձնում հակառակորդների կողմից տարաբնույթ հարձակումներ, ինչպիսին է մարդը մեջտեղում (man-in-the-middle) և տվյալների գողություն:

   Ադմինիստրատորը պետք է համոզվի, որ բոլոր հեռակառավարման կապերը դեպի սերվեր պաշտպանված են գաղտնագրմամբ և գաղտնաբառով:

3. **Թույլտվություններ և արտոնություններ**
   Թույլտվությունների հստակ կառավարման համակարգը կարևոր դեր է խաղում սերվերային անվտանգության մեջ: Եթե չարամիտ օգտագործողը կամ պրոցեսը ունենա ավելի շատ արտոնություններ, քան իրեն անհրաժեշտ է, այդ հանգամանքը կարող է նպաստել սերվերի կոտրմանը:

   Ադմինիստրատորը պետք է համոզվի, որ բոլոր օգտագործողները մուտք ունեն միայն այն ֆայլերին և ռեսուրսներին, որոնք իրենց անհրաժեշտ են աշխատանքը իրականացնելու համար, և ոչ ավելին:

4. **Ժամանակին տեղադրել անվտանգության թարմացումները**
   Կարևոր է տեղադրել օպերացիոն համակարգը և ծրագրային ապահովումը վերջին թարմացումներով և անվտանգության կարկատաններով (patches):

   Օպերացիոն համակարգի և ծրագրային ապահովման ստեղծողները ժամանակ առ ժամանակ թողարկում են թամացումներ (updates): Դրանք հաճախ պարունակում են անվտանգության թարմացումներ, որոնք փակում են հայտնաբերված խոցելիություններն օպերացիոն համակարգում:

   Ադմինիստրատորները պետք է համոզվեն որ թարմացումները տեղադրվում են ժամանակին:

5. **Դիտարկում և գրանցամատյանների գրառումների (logs) հաշվեքննություն**
   Լոգեր ստեղծվում են բոլոր տեսակի ծրագրային ապահովման կողմից - օպերացիոն համակարգի, վեբ հավելվածների, բոլոր տեսակի սերվիսների, տվյալների բազաների, ցանցային սարքերի, երթուղավորիչների, կոմուտատորների (network switch) և այլն կողմից:

   Այս լոգերը պետք է դիտարկվեն և հաճախ ստուգվեն, քանի որ դրանք երբեմն կարող են զգուշացնել գալիք վտանգի մասին: Նույնիսկ հաջող հարձակման դեպքում սերվերների լոգերը հաճախ դատական փորձաքննություն իրականացնելու միակ միջոցն են:

6. **Օգտագործողի հաշիվներ**
   Չօգտագործվող օգտագործողի հաշիվները, ինչպիսիք են աշխատանքից

ազատված աշխատակիցները, պետք է անջատվեն: Պետք է անջատվեն նաև զանազան սերվիսների կողմից ստեղծված օգտագործողների հաշիվները:

Յուրաքանչյուր օգտագործողի հաշիվ մեծացնում է հարձակման հարթությունը: Նախկին աշխատակիցը կարող է ընկերությանը վնաս հասցնելու դրդապատճառներ ունենալ, և եթե նրա նախկին օգտագործողի հաշիվը անջատված չլինի՝ նա հնարավորություն կունենա ցանկացած գործողություն կատարել իր օգտագործողի իրավասություններով:

Յուրաքանչյուր ադմինիստրատոր և օգտագործող, ով մուտք է գործում սերվեր, պետք է ունենա իր սեփական հաշիվը և գաղտնաբառը, և ճիշտ իրավասություններ: Գաղտնաբառը չպետք է բաշխվի օգտագործողներիր միջև:

7. **Ջնջել չօգտագործվող մոդուլներ և ընդլայնումներ**
   Հավելվածները, ինչպես օրինակ վեբ սերվերները, հաճախ կարող են պարունակել որոշակի լռելյայն ընդլայնումներ և ծրագրային միավորներ: Այս ծրագրային միավորները կարող են պարունակել խոցելիություններ, և այդպիսով մեծացնել հնարավոր հարձակման հարթությունը հակառակորդի համար:

   Ադմինիստրատորը պետք է համոզվի, որ հնարավորության դեպքում միայն վեբ հավելվածների համար անհրաժեշտ միավորներն են առկա:

8. **Լինել տեղեկացված**
   Այսօր օպերացիոն համակարգերի և ծրագրային ապահովման, այդ թվում՝ դրանց անվտանգության մասին ինֆորմացիան ազատորեն հասանելի է համացանցում:

   Ադմինիստրատորները պետք է համոզվեն, որ իրենք և իրենց օգտագործողները մշտապես տեղեկացված են հարձակումների և խոցելիությունների մասին վերջին նորություններին:

9. **Օգտագործել սկզբնական կոդի (source code) անվտանգության սկաներներ**
   Սկաներները ծրագրեր են, որոնք ավտոմատացնում և հեշտացնում են սերվերի և հավելվածների պաշտպանության գործընթացը:

   Ծրագրային կոդի ստատիկ և դինամիկ անալիզի գործիքները, ինչպիսիք են Sonar -ը Java լեզվի համար, Valgrind-ը C լեզվի համար և այլն, օգնում են գտնել ծրագրային սխալներ (bugs) և խոցելիություններ ծրագրի կենսափուլի (lifecycle) վաղ շրջանում:

10. **Ընտրել գաղտնագրման և հեշավորման ապահով ալգորիթմներ**
    Պետք է խուսափել կոտրված գաղտնագրման, հաղորդակցության և հեշավորման հաղորդակարգերի օգտագործումից, ինչպիսիք են՝ DES, SSL, MD5:

    Այս հաղորդակարգերի թուլությունը հարձակման հնարավոր վեկտոր է բացում հակառակորդի համար:

    Այսպիսի հաղորդակարգերը պետք է փոխարինվեն ժամանակակից, չկոտրված և գաղտնագրման լայն հանրության վստահությանը արժանացած հաղորդակարգերով:

11. **Օգտագործել հակավիրուս**

Windows օպերացիոն համակարգի վրա հիմնված սերվերներում անհրաժեշտ է տեղադրել հակավիրուսային ծրագրային ապահովում: [16]

Հակավիրուսը սկանավորում է ծրագիրը հետևյալ պայմաններում՝

(a) ամբողջական սկաներ - թողարկվում են պարբերաբար կամ օգտագործողի կողմից,

(b) աշխատանքի ժամանակ, այսինքն երբ համակարգով փոխանցվում են տվյալներ

Հակավիրուսները օգտագործում են վիրուսների հայտնաբերման հետևյալ տեխնոլոգիաները՝

(a) ստորագրության վրա հիմնված հայտնաբերում, երբ ֆայլը համեմատվում է հայտնի չարամիտ կոդի հետ,

(b) փորձարարության վրա հիմնված հայտնաբերում, երբ ֆայլի վարվելաձևը համեմատվում է հայտնի չարամիտ նմուշների հետ,

(c) վարվելակերպի վրա հիմնված հայտնաբերում, որը հաճախ կատարվում է ՆՀՀ-երում

Linux-ի վրա հիմնված համակարգերում հակավիրուս հաճախ չի օգտագործվում: [17]

Linux-ի վրա հիմնված համակարգերում հակավիրուսի անհրաժեշտություն կարող է առաջանալ միայն այն պարագայում, երբ այն օգտագործվում է Windows համակարգերի միջև ֆայլերի փոխանակաման համար: [19]

12. **Օգտագործել ցանցային սկաներներ**

Ցանցային սկաներները օգնում են ադմինիստրատորներին համոզվել իրենց սերվերների անվտանգության մեջ: Այսպիսի գործիքները կարողանում են հայտնաբերել բաց պորտեր, խոցելի սերվիսներ, և նույնիսկ վիրուսներ: Հայտնի ցանցային սկաներներից են՝

(a) Nmap,

(b) Nessus,

(c) Accunetix

Համակարգային ադմինիստրատորների տարածված պարտականություններից է իրենց վստահված համակարգերում պորտերի սկանավորման իրականացումը: Այսպիսի սկանավորումները օգնում են ադմինիստրատորներին գտնել խոցելիություններ իրենց համակարգերում ավելի վաղ, քան հնարավոր հակառակորդը: Այսպիսի սկանավորումներ իրականացնելու համար օգտագործվում են այնպիսի

գործիքներ, ինչպիսիք են՝ nmap, nessus, accunetix և այլն: Ցանցային պորտերի սկանավորման պրոցեսը հաճախ ունի ներքոնշյալ հաջորդականությունը:

(a) Ադմինիստրատորը որոշում է հասցեների և պորտերի շրջանակը, որոնք պետք է ենթարկվեն սկանավորման:

(b) Նա տալիս է ծրագրին այդ պարամետրերը և սկսում է սկանավորումը:

(c) Ծրագիրը փորձարկում է IP հասցեների և պորտերի բոլոր տրված համադրությունները:

(d) Եթե պարզվում է, որ պորտը բաց է, ապա աշխատեցվում է հատուկ ծրագրային սցենար (script), որը փորձում է գուշակել աշխատող սերվիսի մասին տվյալները՝ անունը, տարբերակը, կոնֆիգուրացիան, մատչելի օգտագործողների անունները, և այլն:

(e) Տվյալները տրվում են ադմինիստրատորին նրա նախընտրած ֆորմատով՝ XML, ելք հրամանային տողում կամ ծրագրին հատուկ ֆորմատով:

2 Խնդրի դրվածքը

2.1 Խնդրի արդիականությունը

Նախորդ բաժնի վերջին կետում մշված ցանցային սկաներների ներկայիս իրականացումը ունի որոշակի թերություններ:

1. Ցանցում բազմաթիվ համակարգերի գոյության դեպքում յուրաքանչյուր TCP (Transmission Control Protocol) և UDP (User Datagram Protocol) պորտի սկանավորումը պահանջում է բավականին երկար ժամանակ: Սկանավորումը արագացնելու նպատակով հնարավոր է սկանավորել միայն պորտերի սահմանափակ բազմություն, սակայն այդ դեպքում պատկերը ամբողջական չի լինի, քանի որ ոչ հայտնի պորտերի տակ նույնպես հնարավոր է աշխատի ինչ-որ սերվիս, և այն չի հայտնաբերվի նման սկանավորման ժամանակ:

2. Ցանցային սկաները ծախսում է ցանցային ռեսուրսներ և կարող է որոշ համակարգեր անհասանելի դարձնել սկանավորման ընթացքում:

3. Որոշակի սցենարների դեպքում պորտերի սկանավորումը կարող է հանգեցնել IDS-ում (Intrusion Detection System - Ներխուժում Հայտնաբերող Համակարգ) կեղծ ահազանգի:

4. Հնարավոր են կեղծ դրական արդյունքներ և սերվիսների սխալ նույնականացումներ:

5. Չեն հայտնաբերվում բացթողումներ հետևյալ ասպարեզներում՝

    - թույլտվություններ և արտոնություններ,
    - թարմացումների առկայություն,
    - օգտագործողի հաշիվներ,
    - գաղտնագրման և հեշավորման ապահով ալգորիթմների օգտագործում,
    - հակավիրուսի օգտագործում:

Այսպիսով անհրաժեշտ է որոնել սկանավորում իրականցնելու մեկ այլ եղանակ, որը զերծ կլինի վերը նշված թերություններից:

2.2 Այլընտրանք

Այս աշխատանքում մենք ներկայացնում ենք սերվերների խոցելիությունների հայտնաբերման այլընտրանքային եղանակ, որը սկանավորում է համակարգերը ներսից, և այդպիսով զերծ է վերը նշված թերություններից:

Յուրաքանչյուր բաց պորտի համար ծրագիրը սկանավորում է այդ սերվիսի կոնֆիգուրացիոն ֆայլը խոցելիությունների և դատարկ գաղտնաբառերի առկայության համար և հայտնում է արդյունքները օգտագործողին:

Բացի որոշելուց, թե արդյոք պորտը բաց է թե ոչ, այն նաև ստուգում է, թե արդյոք այն ֆիլտրված է firewall-ով:

2.3 Նախկին փորձի ուսումնասիրություն

2.3.1 Microsoft Baseline Security Analyzer

Այս ծրագրային ապահովման ճարտարապետությունը մասամբ ոգեշնչվել է MSBA ծրագրի կողմից: [24]

MSBA-ը Windows համակարգերի համար նախատեսված անվտանգության սկաներ է, ստեղծված Microsoft ընկերության կողմից: Այն գնահատում է Windows համակարգի և Microsoft-ի կողմից ստեղծված մի շարք այլ ծրագրային ապահովման անվտանգությունը առավել հաճախ հանդիպող սխալների առկայության համար և արդյունքները ներկայացնում է օգտագործողին:

MSBA-ը ունի որոշակի սահմանափակումներ՝

- աշխատում է միայն Windows ճարտարապետության համակարգերում,
- ստուգումներ իրականցնում է միայն Microsoft ընկերության կողմից ստեղծված ծրագրերում:

2.3.2 Buck-Security

Buck-Security-ն անվտանգության սկանավորիչ է Debian և Ubuntu Linux օպերացիոն համակարգերի համար: [25]

Այս աշխատանքում ներկայացվող ծրագիրը որոշ չափով նման է Buck-Security-ին:

Buck-Security-ն նույնպես ունի որոշակի սահմանափակումներ՝

- նախատեսված է միայն Debian և Ubuntu համակարգերի համար,
- գտնվում է Beta փուլում, և խորհուրդ չի տրվում այն օգտագործել արտադրության համակարգերում:

2.3.3 Lynis

Lynis-ը անվտանգության աուդիտի և համակարգի ամրապնդման (hardening) գործիք է UNIX համակարգերի համար: Այն օգնում է ադմինիստրատորներին արագ հայտնաբերել և լուծել անվտանգության սխալները [29]:

Աշխատում է Unix ընտանիքի օպերացիոն համակարգերի միջավայրում: Հանրային տարբերակը տարածվում է GPL3 լիցենզիայի տակ, կա նաև վճարովի, առևտրային լիցենզիայով տարբերակ [30]:

2.3.4 Tiger

Tiger-ը անվտանգությունը գնահատող ծրագիր է UNIX համակարգերի համար:

Ցավոք, այն ներկայումս ակտիվորեն չի մշակվում: Վերջին կայուն տարբերակը թողարկվել է 2010 թվականին:

Աշխատում է Unix ընտանիքի օպերացիոն համակարգերի միջավայրում: Տարածվում է GPL3 լիցենզիայի տակ [31]:

3 Ծրագրին ներկայացվող պահանջներ

Այս աշխատանքի նպատակն է ստեղծել ծրագրային հավելված, որը Linux-ի վրա հիմնված սերվերային համակարգի վրա տեղադրման պարագայում աշխատեցնելիս կգնահատի համակարգերի անվտանգությունը և կհայտնի արդյունքները օգտագործողին:

Ծրագրային հավելվածի առաջնային նպատակն է օգտագործողին ներկայացնել համակարգի անվտանգության ընդհանուր պատկերը:

Ծրագիրը պետք է աշխատի բոլոր ժամանակակից Linux համակարգերի տակ: Հնարավորության դեպքում՝ նաև UNIX ընտանիքի այլ համակարգերում:

Ծրագրի տեղադրումը պետք է լինի հնարավորինս պարզ:

Ծրագրի ստուգումներից յուրաքանչյուրը պետք է հնարավոր լինի անջատել՝ մյուսներից անկախ:

Եթե ծրագրի մի մոդուլը իրականացնում է բազմատեսակ ստուգումներ, ապա դրանցից յուրաքանչյուրը պետք է հնարավոր լինի անջատել՝ մյուսներից անկախ:

Ստորև ներկայացվում է ծրագրի կողմից կատարվող ստուգումների ցանկը:

3.1 Ծրագրային թարմացումներ

Ծրագիրը պետք է ստուգի թե վերջին անգամ երբ է թարմացվել օպերացիոն համակարգը: Եթե դա կատարվել է բավականաչափ ուշ, ապա օգտագործողը պետք է զգուշացվի, հայտնելով վերջին թարմացման ժամանակը:

Այդ ժամանակային միջակայքը պետք է հնարավոր լինի կարգաբերել ծրագրի կոնֆիգուրացիայով:

Ստորև ներկայացված են Linux-ի յուրաքանչյուր տարբերակին առանձնահատուկ վերջին թարմացման ժամանակի ստուգումները՝

3.1.1 Debian/APT-ի վրա հիմնված համակարգեր

Ծրագիրը պետք է որոշի վերջին թարմացման ժամանակը /var/lib/apt/periodic/update-success-stamp ֆայլի ստեղծման ժամանակով: [8]

Դա Ubuntu ընտանիքին հատուկ ֆայլ է, որի ստեղծման ժամանակը համընկնում է apt-get update հրահանգի վերջին կատարման ժամանակի հետ: Դա պայմանավորված է նրանով, որ նշված հրահանգը կատարելիս աշխատեցվում է /etc/apt/apt.conf.d/15update-stamp ծրագիրը, որը և թարմացնում է նշված դրոշմ-ֆայլը՝ վերագրելով նրա ստեղծման ժամանակը ներկայիս պահին: [34]

3.1.2 Red Hat/YUM-ի վրա հիմնված համակարգեր

Ծրգաիրը պետք է որոշի վերջին թարմացման ժամանակը `yum history՝ հրահանգի ելքը վերլուծելով: [9]

yum history հրահանգը արտածում է նմանօրինակ ելք՝

Սկզբնական կոդ 1: .

```
# yum  history
Loaded plugins: fastestmirror, refresh-packagekit
ID     | Login user               | Date and time    | Action(s)      | Altered
-------------------------------------------------------------------------------
    41 | root <root>              | 2012-04-27 20:17 | Install        |   19
    40 | root <root>              | 2011-11-20 10:09 | Install        |   10
    39 | root <root>              | 2011-11-20 08:14 | Install        |    1 E<
    38 | root <root>              | 2011-11-19 15:46 | Update         |    1
```

[35]

Օպերացիոն համակարգի վերջին թարմացման ժամանակը հնարավոր է որոշել այս ելքը վերջից փնտրելով 'Update' բառը, այնուհետև առաջին համընկնող տողում վերլուծելով 'Date and time' սյան տեքստը:

3.1.3 Arch Linux/Pacman-ի վրա հիմնված համակարգեր

Pacman փաթեթների մենեջերի գործողությունների գրանցամատյանը գտնվում է `/var/log/pacman.log` ֆայլում: [10] Ծրագիրը պետք է ստուգի վերջին թարմացման ժամանակը վելուծելով վերոնշյալ ֆայլի պարունակությունը: Այն ունի նմանօրինակ պարունակություն՝

Սկզբնական կոդ 2: .

```
[2016-04-05 10:22] [ALPM] transaction started
[2016-04-05 10:23] [ALPM] installed pycharm-community (2016.1-1)
[2016-04-05 10:23] [ALPM] transaction completed
[2016-04-05 10:24] [PACMAN] Running 'pacman -S -y -y -u'
[2016-04-05 10:24] [PACMAN] synchronizing package lists
[2016-04-05 10:24] [PACMAN] starting full system upgrade
[2016-04-05 10:24] [PACMAN] Running 'pacman -S -y -y -u'
[2016-04-05 10:24] [PACMAN] synchronizing package lists
[2016-04-05 10:24] [PACMAN] starting full system upgrade
[2016-04-05 10:28] [ALPM] transaction started
[2016-04-05 10:28] [ALPM] upgraded tzdata (2016b-1 -> 2016c-1)
[2016-04-05 10:28] [ALPM] upgraded alsa-utils (1.1.0-1 -> 1.1.0-2)
[2016-04-05 10:28] [ALPM] upgraded graphite (1:1.3.6-1 -> 1:1.3.8-1)
[2016-04-05 10:28] [ALPM] upgraded harfbuzz (1.2.3-1 -> 1.2.4-1)
[2016-04-05 10:28] [ALPM] upgraded fontconfig (2.11.1-2 -> 2.11.94-1)
[2016-04-05 10:28] [ALPM-SCRIPTLET] updating font cache... done.
[2016-04-05 10:28] [ALPM] installed tslib (1.1-1)
[2016-04-05 10:28] [ALPM] upgraded libxkbcommon (0.5.0-1 -> 0.6.0-1)
```

Օպերացիոն համակարգի վերջին թարմացման ժամանակը հնարավոր է որոշել այս ելքը վերջից փնտրելով 'starting full system update' նախադասությունը, այնուհետև առաջին համընկնող տողում վերլուծելով ժամանակը, որը գտնվում է առաջին քառակուսի փակագծերի մեջ:

3.2 Ֆայլերի և դիրեկտորիաների թույլտվություններ

Բոլոր ֆայլերը և դիրեկտորիաները պետք է ունենան ճիշտ թույլտվություններ: Հակառակ դեպքում համակարգը խոցելի է: Բոլոր օգտագործողների կողմից գրման հնարավորություն ունեցող (worldwritable) ֆայլերը և դիրեկտորիաները կարող են օգտագործվել հակառակորդի կողմից՝ կամեցած ֆայլի կամ դիրեկտորիայի մեջ ցանկացած բան փոփոխելու կամ ջնջելու համար: [26][25]

Բացառություն են կազմում այն worldwritable դիրեկտորիաները, որոնք ունեն sticky բիթ, ինչպես նաև այն բոլոր ֆայլերը որոնք չեն սկսվում կետով և չեն պատկանում համակարգային օգտագործողին:

Linux-ի ֆայլային համակարգերում Stiky բիթը դիրեկտորիայի հատուկ ատրիբուտ է: Եթե այն առկա է, ապա նշանակում է որ նրանում պարունակվող ֆայլերի տերը միայն (owner) և համակարգային օգտագործողը (root user) իրավունք ունեն ջնջել կամ վերանվանել վերոնշյալ ֆայլը:

Այսպիսով, այս ստուգման ժամանակ նման ֆայլերի և դիրեկտորիաներ որոնման ժամանակ պետք է կիրառել հետևյալ ֆիլտրը՝

- worldwritable դիրեկտորիաները պետք է ունենան sticky բիթ
- worldwritable ֆայլերը չպետք է սկսվեն կետով ('.')
- worldwritable ֆայլերը չպետք է պատկանեն համակարգային օգտագործողին (root user)

Նման ֆայլերի հայտնաբերման դեպքում ծրագիրը պետք է զգուշացնի, և թվարկի այդ ֆայլերը:

3.3 Բաց TCP և UDP պորտեր

Յուրաքանչյուր բաց պորտ մեծացնում է հարձակման հարթությունը: [37]

Այդ պատճառով անհրաժեշտ է սահմանափակել բաց պորտերի քանակը, կամ դրանք ծածկել firewall-ով:

Ծրագիրը պետք է օգտագործողին ներկայացնի բոլոր բաց TCP և UDP պորտերի ցանկը, և դրանց տակ աշխատող սերվիսների անունները, հնարավորության դեպքում՝ նաև այն կատարվող ֆայլի անունը, որը գործարկվել է սերվիսը աշխատեցնելիս:

3.4 Մուտք որպես համակարգային օգտագործող

Համակարգային օգտագործողով մուտքը համակարգ համարվում է վատ գործելակերպ անվտանգության տեսանկյունից ներքոնշյալ պատճառներով:

1. Շատ հաճախ առօրյա աշխատանքում հասարակ օգտագործողի իրավասությունները միանգամայն բավական են ամենօրյա գործունեությունը իրականացնելու համար: Բացառություն է կազմում համակարգում ծրագրեր տեղադրելը, կոնֆիգուրա-

ցիայի փոփոխությունը և այլ ադմինիստրատիվ գործեր, որոնք հազվադեպ են անհրաժեշտ լինում:

2. Օգտագործողը կարող է պատահաբար սխալ հրաման գործարկել, որը որոշ դեպքերում կարող է կործանարար հետևանքներ ունենալ:

3. Խոցելիությունները և ծրագրային սխալները (bugs) շատ ավելի մեծ վնաս կարող են հասցնել համակարգին: Եթե մի ծրագիր իրականացվում է համակարգային օգտագործողի իրավասություններով, և նրանում առկա է խոցելիություն կամ ծրագրային սխալ (bugs), ապա այդ խոցելության կամ սխալի ի հայտ գալու պարագայում խոցելի է դառնում ամբողջ համակարգը: Իսկ որպես հասարակ օգտագործող իրականացնելիս խոցելի է միայն նշված օգտագործողին պատկանող ֆայլերը և նրա իրավասության տակ գտնվող այլ արտոնություններ:

[38]

Եթե օգտագործողը մուտք է գործել համակարգ որպես համակարգային օգտագործող (root user), ապա պետք է զգուշացնել: [12]

Բացառություն է կազմում այն դեպքը, եթե ծրագիրը իրականցվում է `sudo՝ հրահանգով: Այս դեպքում զգուշացում չի կատարվում:

3.5 Օգտագործողների UMASK ստուգում

UMASK-ը օգտագործողի ատրիբուտներ է, որը որոշում է, թե նոր ստեղծված ֆայլերը ինչ թույլտվություններ պետք է ունենան:

Եթե օգտագործողի UMASK-ը այնպիսին է, որ ստեղծում է worldwritable ֆայլեր, ապա սա ունի նույն թերությունները ինչ նախորդ բաժնում նկարագրված թերությունը: [28]

3.6 Ֆայլերի և դիրեկտորիաների SETUID և SETGID ստուգում

Եթե SETUID բիթը դրվում է կատարվող ֆայլի (executable file) վրա, ապա այն պրոցեսը, որը ստեղծվում է այդ ֆայլը աշխատեցնելիս, աշխատում է այդ ֆայլի օգտագործողի թույլտվություններով:

Նույնը կատարվում է SETGID բիթի տեղադրման ժամանակ՝ ֆայլի խմբի համար [27]:

3.7 Դատարկ կամ թույլ գաղտնաբառեր

Եթե սերվերը արտաքինից հասանելի է սերվիսների միջոցով, որոնք իսկության ստուգման համար օգտագործում են լոկալ linux-ի օգտագործողների հաշիվները, ապա ծրագիրը ստուգում է, թե արդյոք այդ օգտագործողները ունեն դատարկ գաղտնաբառեր [13]:

Դա կատարվում է՝

3.8 Համոզվել որ ոչ համակարգային օգտագործողների UID-ն 0 է

[15]

3.9 Ամենատարածված սերվիսներում ոչ անվտանգ կոնֆիգուրացիաների առկայության ստուգումներ

3.9.1 SSHd

SSH սերվերի կոնֆիգուրացիոն ֆայլն է՝
/etc/ssh/ssh_config

Այսօր խոցելի համարվող SSH v1 հաղորդակարգըը չպետք է միացված լինի: Այս համակարգի խոցելիությունը խոցվել է վայրի միջավայրում WOOT նախագծի կողմից: [7]

Ծրագիրը նաև ստուգում է՝ արդյոք SSH-ի գաղնտաբառով մուտքի հնարավորությունը թույլատրվում է: Եթե այո, ապա օգտագործողը զգուշացվում է: [11]

cat PasswordAuthentication no

3.9.2 MySQL

MySQL-ը այսօր ամենատարածված ռելացիոն տվյալների բազաներից է աշխարհում: [4] Ծրագիրը սկանավորում է հետևյալ կոնֆիգուրացիոն ֆայլերը սխալների համարէ

/etc/my.cnf /etc/mysql/my.cnf /.my.cnf

3.9.3 Telnet

Եթե telnet-ի աշխատող սերվիս է հայտնաբերվում, ապա օգտագործողը զգուշացվում է: [13]

3.9.4 FTP

Բացառությամբ այն դեպքի, որ աշխատող FTP սերվիսը միայն կարդացվող և հանրորեն հասանելի է, օգտագործողը զգուշացվում է FTP-i օգտագործման դեմ: FTP հաղորդակարգը ապահով չէ, քանի որ օգտագործողի անունը և գաղտնաբառը փոխանցվում են բացիեբաց: [14]

4 Իրականացում

4.1 Ծրագրավորման լեզվի ընտրություն

Ծրագրի իրականացման համար ընտրվել է Python 3 լեզուն: Այն ունի մի շարք առավելություններ նշված խնդրի իրականացման համար՝ [36]

1. Python-ի շարահյուսությունը չափազանց հեշտ է և՛ սովորել և՛ հասկանալ,
2. Python-ը անվճար է և ունի ազատական լիցենզիա,
3. Python-ը աշխատում է բոլոր հիմնական օպերացիոն համակարգերում՝ Windows, Linux, OS X,
4. Python-ը ունի ներդրված և հասանելի գրադարանների առատ բազմություն:

4.2 Ծրագրային պահանջներ

Ծրագրի աշխատանքի համար անհրաժեշտ է՝

1. Unix-ի վրա հիմնված օպերացիոն համակարգ,
2. Python 3.5.1+
3. `psutil` (python process and system utilites) Python գրադարանը:

4.3 Մոդուլների իրականացումը

4.3.1 Հիմնական մոդուլը՝ lmap.py

Հիմնական մոդուլը կարդում է ծրագրի կոնֆիգուրացիան config.yml ֆայլից, որը գտնվում է նույն դիրեկտորիայում, ինչ և lmap.py ֆայլը: Այնուհետև հերթով աշխատեցվում են բոլոր ստուգող ենթամոդուլները:

Ամեն ենթամոդուլի աշխատանքից առաջ տպվում է մոդուլի անունը: Աշխատանքի ավարտից հետո տպվում է թե արդյոք ստուգումը հաջող է անցել: Եթե ոչ՝ այնուհետև տպվում է հաղորդագրությունը:

4.3.2 Ծրագրային թարմացումներ՝ update.py

Նախ ծրագիրը որոշում է թե ինչ օպերացիոն համակարգի միջավայրում է այն աշխատում: Այս խնդրի լուծման համար օգտագործվել է platform.linux_distribution() կանչը: Կախված այդ կանչի արդյունքից աշխատեցվում է օպերացիոն համակարգին հատուկ ծրագիրը, որը որոշում է համակարգի վերջին թարմացման ժամանակը:

- Եթե ծրագիրը աշխատում է Arch Linux միջավայրում, ապա ծրագիրը տող-առ-տող կարդում է /var/log/pacman.log ֆայլի պարունակությունը: Եթե հերթական տողում

առկա է 'starting full system upgrade' բառակապակցությունը, ապա ծրագիրը դադարեցնում է կարդալ տողերը և վերջին թարմացման ժամանակը համարում է նշված տողի առաջին քառակուսի փակագծերի միջև գտնվող ժամանկային գրառումը:

Եթե 'starting full system upgrade' բառակապակցությունը չի հայտնաբերվել, ապա ծրագիրը հայտնում է ստուգման անհնարինության մասին:

- Եթե ծրագիրը աշխատում է Debian կամ Ubuntu Linux միջավայրում, ապա ծրագիրը որոշում է /var/lib/apt/periodic/update-success-stamp ֆայլի վերջին փոփոխման ժամանակը:

  Եթե ֆայլը չի հայտնաբերվել, ապա ծրագիրը հայտնում է ստուգման անհնարինության մասին:

- Եթե ծրագիրը աշխատում է Red Hat/CentOS միջավայրում, ապա ծրագիրը տող-առ-տող կարդում է yum history հրահանգի ելքը: Եթե հերթական տողում առկա է 'Update' բառը, ապա ծրագիրը դադարեցնում է կարդալ տողերը և վերջին թարմացման ժամանակը համարում է նշված տողի Date and time սյունակում գտնվող ժամանկային գրառումը:

  Եթե 'Update' բառը չի հայտնաբերվել, ապա ծրագիրը հայտնում է ստուգման անհնարինության մասին:

Ստացված ամսաթիվը համեմատվում է ծրագրի կոնֆիգուրացիայի update.warn_last_update_interval_days գրառման հետ: Եթե վերջին թարմացման ամսաթիվը ավելի նոր է, ապա ստուգումը համարվում է հաջող, հակառակ դեպքում՝ անհաջող:

4.3.3 Ֆայլերի և դիրեկտորիաների թույլտվություններ՝ worldwritable.py

Նախ ծրագիրը փնտրում է հետևյալ ֆիլտրին համապատասխանող բոլոր ֆայլերը և դիրեկտորիաները՝

1. ֆայլեր, որոնք worldwritable են և սկսվում են կետով,
2. դիրեկտորիաներ, որոնք worldwritable են և չունեն sticky բիթ,
3. ֆայլեր, որոնք worldwritable են և պատկանում են համակարգային օգտագործողին (root):

Ֆայլերի և դիրեկտորիաների փնտրումը կատարվում է os.walk() կանչի օգնությամբ, իսկ ատրիբուտների ստուգումը՝ os.stat() կանչով:

Եթե վերոնշյալ պայմաններին համապատասխանող ֆայլեր կամ դիրեկտորիաներ հայտնաբերվել են, ապա ծրագիրը արտածում է թե որ պայմանն է խախտվել, և այդ ֆայլերի կամ դիրեկտորիաների ցուցակը:

Եթե մի քանի պայմաններ են խախտվել, ապա յուրաքանչյուր պայմանի համար կատարվում է առանձին ելք:

4.3.4 Բաց TCP և UDP պորտեր՝ openports.py

Նախ ծրագիրը ստանում է համակարգում առկա բոլոր inet տեսակի միացումների ցանկը psutil.net_connections() կանչի օգնությամբ: Այնուհետև ֆիլտրվում են բաց և դրսից հասանելի TCP և UDP պորտերի միացումները: Ելքում ստացվում է մի աղյուսակ որի սյուներն են՝

1. պորտի միացման տիպը՝ TCP կամ UDP,
2. IP հասցեն,
3. պորտը՝ TCP միացման դեպքում,
4. PID՝ Պրոցեսի նույնականացման համարը,
5. Username՝ Օգտագործողի անունը,
6. Command line՝ Հրահանգը, որը աշխատեցվել է այդ պորտի տակ լսող ծրագիրը գործարկելիս:

4.3.5 Մուտք որպես համակարգային օգտագործող՝ root.py

Ծրագիրը ստուգում է ներկայիս օգտագործողի UID-ն:

Նկատենք, որ Unix համակարգերում համակարգային օգտագործողի UID-ն միշտ 0 է: Այլ օգտագործողների UID-ները 0-ից տարբեր են:

Բացի դրանից հնարավոր է որ այս ծրագիրը իրականացվի sudo հրամանի միջավայրում: Սա համարվում է նորմալ գործելակերպ: Որպեսզի ստուգենք, թե գտնվում ենք sudo միջավայրում թե ոչ՝ պետք է ստուգել SUDO_UID միջավայրի փոփոխականի առկայությունը:

Այսպիսով, ծրագիրը գործարկվում է համակարգային օգտագործողի կողմից, եթե՝

1. օգտագործողի UID-ն 0 է,
2. SUDO_UID միջավայրի փոփոխականը առկա չէ:

Հակառակ դեպքում ստուգումը հաջողված է:

5 Եզրակացություն

Ներկայումս օգտագործվող ցանցային սկանավորիչները սահմանափակ են իրենց կարողություններում՝ անվտանգությանը նպաստելու տեսանկյունից:

Այս աշխատանքի շրջանակներում ստեղծվել է Linux սերվերային համակարգերի անվտանգությունը գնահատող ծրագիր՝ գրված Python լեզվով:

Այն օգնում է համակարգային ադմինիստրատորներին տեղեկանալ իրենց վստահված սերվերների անվտանգության մակարդակի մասին, այսպիսով նպաստելով անվտանգության մակարդակի բարձրացմանը:

Նմանօրինակ ծրագրային հավելվածը օգտակար կարող է լինել միայն բազմակողմանի աջակցության և երկարատև զարգացման դեպքում:

Գրականության ցանկ

1. 1
   http://sectools.org/
2. 2
   https://docs.python.org/2/library/socket.html
3. 3
   https://pythonhosted.org/psutil/
4. 4
   http://db-engines.com/en/ranking
5. 5
   http://www.yolinux.com/TUTORIALS/LinuxTutorialInternetSecurity.html
6. 6
   http://www.yolinux.com/TUTORIALS/LinuxTutorial-woot-project.html
7. 7
   http://www.iss.net/threats/advise100.html
8. 8
   http://serverfault.com/questions/20747/find-last-time-update-was-performed-with-apt-get
9. 9
   http://serverfault.com/questions/389650/how-to-check-when-yum-update-was-last-run
10. 10
    https://bbs.archlinux.org/viewtopic.php?id=150428
11. 11
    https://www.digitalocean.com/community/tutorials/7-security-measures-to-protect-your-servers
12. 12
    http://askubuntu.com/questions/16178/why-is-it-bad-to-login-as-root
13. 13
    http://www.tecmint.com/linux-server-hardening-security-tips/

14. 14
https://www.digitalocean.com/community/tutorials/an-introduction-to-securing-your-linux-vps
15. 15
http://www.cyberciti.biz/tips/linux-security.html
16. 16
http://serverfault.com/questions/632/do-you-run-antivirus-on-your-windows-servers
17. 17
http://www.howtogeek.com/135392/htg-explains-why-you-dont-need-an-antivirus-on-linux-and-when-you-do/?PageSpeed=noscript
18. 18
https://antivirus.comodo.com/how-antivirus-software-works.php
19. 19
http://security.stackexchange.com/a/53462/37546
20. 20
http://cybercellmumbai.gov.in/html/general-tips/what_is_computer_security.html
21. 21
http://www.acunetix.com/websitesecurity/webserver-security/
22. 22
https://www.onehoursitefix.com/why-would-hackers-hack-my-website/
23. 23
http://searchsecurity.techtarget.com/definition/information-security-infosec
24. 24
https://msdn.microsoft.com/en-us/library/ff647642.aspx
25. 25
http://www.buck-security.net/buck-security.html
26. 26
http://www.softpanorama.org/Access_control/Permissions/world_writable_files_problem.shtml

27. 27
    http://shop.oreilly.com/product/9780596527631.do
28. 28
    http://www.cyberciti.biz/tips/understanding-linux-unix-umask-value-usage.html
29. 29
    https://github.com/CISOfy/lynis/
30. 30
    https://cisofy.com/pricing/
31. 31
    http://git.savannah.gnu.org/cgit/tiger.git/
32. 32
    http://www.digitalattackmap.com/understanding-ddos/
33. 33
    https://security.illinois.edu/content/updates-and-patches
34. 34
    http://serverfault.com/questions/20747/find-last-time-update-was-performed-with-apt-get
35. 35
    http://serverfault.com/questions/389650/how-to-check-when-yum-update-was-last-run
36. 36
    https://en.wikiversity.org/wiki/Python/Why_learn_Python
37. 37
    http://superuser.com/questions/82488/why-is-it-bad-to-have-open-ports
38. 38
    http://unix.stackexchange.com/questions/52268/why-is-it-a-bad-idea-to-run-as-root

TODO: iso 27001 TODO: http://lazy2hack.blogspot.am/2010/03/collection-of-security-checks-for-linux.html

ՀԱՎԵԼՎԱԾ

Սկզբնական կոդ

Սկզբնական կոդ 3: `lmap.py`

```
import os

import yaml

from scanner.openports import OpenPorts
from scanner.root import Root
from scanner.update import Update
from scanner.worldwritable import WorldWritable


def main():
    """
    Runs the program
    """
    config = get_config()
    scanners = [
        Update(config),
        OpenPorts(config),
        WorldWritable(config),
        Root(config)
    ]
    for scanner in scanners:
        print('-' * 79)
        print('Running:', scanner.__class__.__name__)
        result = scanner.scan()
        print('Status:', result[0])
        print('Message:\n' + result[1])


def get_config():
    """
    :return: A dictionary containing the program's configuration.
    """
    with open(os.path.dirname(os.path.realpath(__file__)) + '/' + 'config.yml',
        'r') as f:
        return yaml.load(f)


if __name__ == '__main__':
    main()
```

Սկզբնական կոդ 4: `lmap_test.py`

```
import unittest
from unittest.mock import MagicMock
```

```
import lmap


def raise_file_not_found_error(x):
    raise FileNotFoundError


class Test(unittest.TestCase):
    def test_get_config(self):
        # Prepare data and mocks

        # Run test scenario
        result = lmap.get_config()

        # Assertions
        self.assertIsNotNone(result)
        self.assertIsInstance(result, dict)

    @unittest.mock.patch('lmap.OpenPorts')
    @unittest.mock.patch('lmap.Update')
    @unittest.mock.patch('lmap.WorldWritable')
    def test_main(self, mock_open_ports, mock_update, mock_world_writable):
        # Prepare data and mocks
        config = MagicMock()
        lmap.get_config = MagicMock(return_value=config)

        # Run test scenario
        lmap.main()

        # Assertions
        lmap.get_config.assert_called_once_with()


if __name__ == '__main__':
    unittest.main()
```

Սկզբնական կոդ 5: `config.yml`

```
update:
  warn_last_update_interval_days: 5
```

Սկզբնական կոդ 6: `scan_status.py`

```
from enum import Enum


class ScanStatus(Enum):
    success = 1
    fail = 2
```

```
    unknown = 3
```

Սկզբնական կոդ 7: base_scanner.py

```
class BaseScanner:
    def scan(self):
        # Returns a tuple:
        # (ScanStatus, message)
        raise NotImplementedError()
```

Սկզբնական կոդ 8: openports.py

```
from socket import SOCK_DGRAM
from socket import SOCK_STREAM

import psutil

from scanner.base_scanner import BaseScanner
from scanner.scan_status import ScanStatus


class OpenPorts(BaseScanner):
    def __init__(self, config):
        # config is a dictionary of this program's configuration
        self.config = config

    def scan(self):
        """
        :returns: a tuple of (ScanStatus, message).
        """
        output = 'Type, IP, Port, PID, Username, Command line\n'
        for connection in psutil.net_connections(kind='inet'):
            if self.is_port_is_open_and_externally_accessible(connection):
                output += self.get_line_for_connection(connection)
        return ScanStatus.success, output

    def get_line_for_connection(self, connection):
        """
        :returns: the single-line information about given connection. The line e
            nds with a newline.
        """
        line = [
            self.get_connection_type_string(connection.type),
            str(connection.laddr[0]) + ':' + str(connection.laddr[1]),
            connection.pid,
        ]
        if connection.pid is not None:
            process = psutil.Process(connection.pid)
            line.append(process.username())
            line.append(' '.join(process.cmdline()))
```

```
        return ' '.join(str(i) for i in line)

    def is_port_is_open_and_externally_accessible(self, connection):
        # Checks whether given connection's port is open and externally accessib
            le
        # Returns True if is, otherwise False
        if connection.type == SOCK_STREAM:
            return connection.status == 'LISTEN'
        if connection.type == SOCK_DGRAM:
            return True

    def get_connection_type_string(self, connection_type):
        # connection_type: socket.SOCK_STREAM or friends
        # Returns 'tcp' for socket.SOCK_STREAM, 'udp' for socket.SOCK_DGRAM
        # Raises ValueError if connection_type is anything else
        if connection_type == SOCK_DGRAM:
            return 'udp'
        elif connection_type == SOCK_STREAM:
            return 'tcp'
        else:
            raise ValueError('Unknown connection type: ', connection_type)
```

Սկզբնական կոդ 9: openports_test.py

```
import unittest
from socket import SOCK_DGRAM, SOCK_STREAM
from unittest.mock import MagicMock, call

from scanner.openports import OpenPorts
from scanner.scan_status import ScanStatus


def raise_file_not_found_error(x):
    raise FileNotFoundError


class Test(unittest.TestCase):
    @unittest.mock.patch('scanner.openports.psutil')
    def test_scan(self, mock_psutil):
        # Prepare data and mocks
        test_subject = OpenPorts(None)
        mock_psutil.net_connections = MagicMock(return_value=['closed connection
            ', 'open connection'])
        test_subject.is_port_is_open_and_externally_accessible = MagicMock(side_
            effect=[False, True])
        test_subject.get_line_for_connection = MagicMock(return_value='open conn
            ection line')

        # Run test scenario
```

```
        result = test_subject.scan()

        # Assertions
        self.assertEquals(result[0], ScanStatus.success)
        self.assertIsNotNone(result[1])
        mock_psutil.net_connections.assert_called_once_with(kind='inet')
        test_subject.is_port_is_open_and_externally_accessible.assert_has_calls(
            [
                call('closed connection'),
                call('open connection')
            ]
        )
        test_subject.get_line_for_connection.assert_called_once_with('open conne
            ction')

    def test_port_is_open_and_externally_accessible_when_udp(self):
        # Prepare data and mocks
        test_subject = OpenPorts(None)
        connection = MagicMock()
        connection.type = SOCK_DGRAM

        # Run test scenario
        result = test_subject.is_port_is_open_and_externally_accessible(connecti
            on)

        # Assertions
        self.assertTrue(result)

    def test_port_is_open_and_externally_accessible_when_tcp_and_connection_stat
        us_is_listen(self):
        # Prepare data and mocks
        test_subject = OpenPorts(None)
        connection = MagicMock()
        connection.type = SOCK_STREAM
        connection.status = 'LISTEN'

        # Run test scenario
        result = test_subject.is_port_is_open_and_externally_accessible(connecti
            on)

        # Assertions
        self.assertTrue(result)

    def test_port_is_open_and_externally_accessible_when_tcp_and_connection_stat
        us_is_not_listen(self):
        # Prepare data and mocks
        test_subject = OpenPorts(None)
        connection = MagicMock()
```

```
        connection.type = SOCK_STREAM
        connection.status = 'Not LISTEN'

        # Run test scenario
        result = test_subject.is_port_is_open_and_externally_accessible(connecti
            on)

        # Assertions
        self.assertFalse(result)

    def test_get_connection_type_string_when_tcp(self):
        # Prepare data and mocks
        test_subject = OpenPorts(None)

        # Run test scenario
        result = test_subject.get_connection_type_string(SOCK_STREAM)

        # Assertions
        self.assertEqual(result, 'tcp')

    def test_get_connection_type_string_when_udp(self):
        # Prepare data and mocks
        test_subject = OpenPorts(None)

        # Run test scenario
        result = test_subject.get_connection_type_string(SOCK_DGRAM)

        # Assertions
        self.assertEqual(result, 'udp')

    def test_get_connection_type_string_when_other(self):
        # Prepare data and mocks
        test_subject = OpenPorts(None)

        # Run test scenario
        with self.assertRaises(ValueError):
            test_subject.get_connection_type_string(None)

            # Assertions

    def test_get_line_for_connection_when_unknown_pid(self):
        # Prepare data and mocks
        test_subject = OpenPorts(None)
        connection = MagicMock()
        connection.type = SOCK_DGRAM
        connection.laddr = ('127.0.0.1', 80)
        connection.pid = None
        test_subject.get_connection_type_string = MagicMock(return_value='udp')
```

```
        # Run test scenario
        result = test_subject.get_line_for_connection(connection)

        # Assertions
        self.assertEquals('udp 127.0.0.1:80 None', result)

    @unittest.mock.patch('scanner.openports.psutil')
    def test_get_line_for_connection_when_known_pid(self, mock_psutil):
        # Prepare data and mocks
        test_subject = OpenPorts(None)
        connection = MagicMock()
        connection.type = SOCK_DGRAM
        connection.laddr = ('127.0.0.1', 80)
        connection.pid = 300
        process = MagicMock()
        process.username = MagicMock(return_value='Babken')
        process.cmdline = MagicMock(return_value=['ls', '-l', '-a'])
        test_subject.get_connection_type_string = MagicMock(return_value='udp')
        mock_psutil.Process = MagicMock(return_value=process)

        # Run test scenario
        result = test_subject.get_line_for_connection(connection)

        # Assertions
        self.assertEquals('udp 127.0.0.1:80 300 Babken ls -l -a', result)
        mock_psutil.Process.assert_called_once_with(connection.pid)


if __name__ == '__main__':
    unittest.main()
```

Սկզբնական կոդ 10: update.py

```
import os
import platform
import re
from datetime import datetime, timedelta

from scanner.base_scanner import BaseScanner
from scanner.scan_status import ScanStatus


class Update(BaseScanner):
    def __init__(self, config):
        # config is a dictionary of this program's configuration
        self.config = config

    def scan(self):
```

```
        distribution_name = platform.linux_distribution()[0]
        if distribution_name == 'arch':
            return self.scan_arch()
        elif distribution_name == 'debian':
            return self.scan_debian()
        elif distribution_name == 'redhat':
            return self.scan_redhat()
        else:
            return ScanStatus.unknown, ''

    def scan_arch(self):
        # Scans the pacman log file, and checks whether the last system update d
            ate is older than required
        # Returns a tuple: (ScanStatus, message)
        config_update_interval_days = int(self.config['update']['warn_last_updat
            e_interval_days'])
        with open('/var/log/pacman.log') as f:
            last_update_date = self.get_pacman_last_update_date(f.readlines())
            if last_update_date is None:
                return ScanStatus.unknown, ''
            elif datetime.today() - last_update_date < timedelta(days=config_upd
                ate_interval_days):
                return ScanStatus.success, ''
            else:
                return ScanStatus.fail, ''

    def get_pacman_last_update_date(self, contents):
        # contents: list of lines of pacman.log
        # Returns the datetime instance of last update date or None if last upda
            te date is not found
        for line in reversed(contents):
            if 'starting full system upgrade' in line:
                match_date = re.search('\[(.*?)\]', line)
                if match_date:
                    last_update_date_string = match_date.group(1)
                    return datetime.strptime(last_update_date_string, '%Y-%m-%d
                        %H:%M')

    def scan_debian(self):
        # Scans update-success-stamp file's creation date and checks whether it'
            s older than the last system update date
        # Returns a tuple: (ScanStatus, message)
        config_update_interval_days = int(self.config['update']['warn_last_updat
            e_interval_days'])
        last_update_date = self.get_apt_last_update_date('/var/lib/apt/periodic/
            update-success-stamp')
        if last_update_date is None:
            return ScanStatus.unknown, ''
```

```
        elif datetime.today() - last_update_date < timedelta(days=config_update_
            interval_days):
            return ScanStatus.success, ''
        else:
            return ScanStatus.fail, ''

    def get_apt_last_update_date(self, update_success_stamp_file_location):
        # update_success_stamp_file_location: usually `/var/lib/apt/periodic/upd
            ate-success-stamp`
        # Returns the datetime instance of last update date or None if last upda
            te date is not found
        try:
            file_modification_epoch = os.path.getmtime(update_success_stamp_file
                _location)
        except FileNotFoundError:
            return None
        return datetime.fromtimestamp(file_modification_epoch)

    def scan_redhat(self):
        # Not implemented, returns (ScanStatus.unknown, '')
        # Returns a tuple: (ScanStatus, message)
        return ScanStatus.unknown, ''
```

Սկզբնական կոդ 11: update_test.py

```
import os
import unittest
from datetime import datetime, timedelta

from scanner.update import Update
from unittest.mock import patch, mock_open, MagicMock
from scanner.scan_status import ScanStatus
from tempfile import NamedTemporaryFile


def raise_file_not_found_error():
    raise FileNotFoundError


class Test(unittest.TestCase):
    def test_get_pacman_last_update_date_when_not_found(self):
        # Prepare data and mocks
        file_contents_list = ['no such lines here']

        # Run test scenario
        last_update_date = Update(None).get_pacman_last_update_date(file_content
            s_list)

        # Assertions
```

```
        self.assertIsNone(last_update_date)

    def test_get_pacman_last_update_date_when_found(self):
        # Prepare data and mocks
        file_contents_list = ['line1', '[2016-04-05 10:24] [PACMAN] starting ful
            l system upgrade', 'line3']

        # Run test scenario
        last_update_date = Update(None).get_pacman_last_update_date(file_content
            s_list)

        # Assertions
        self.assertEqual(last_update_date, datetime.strptime('2016-04-05 10:24',
             '%Y-%m-%d %H:%M'))

    def test_scan_arch_when_is_older(self):
        # Prepare data and mocks
        with patch('builtins.open', mock_open()) as mock_file:
            config = {
                'update': {
                    'warn_last_update_interval_days': '2'
                }
            }
            update = Update(config)
            update.get_pacman_last_update_date = MagicMock(return_value=datetim
                e.today() - timedelta(days=10))

            # Run test scenario
            result = update.scan_arch()

            # Assertions
            self.assertEqual(result[0], ScanStatus.fail)
            mock_file.assert_called_once_with('/var/log/pacman.log')
            update.get_pacman_last_update_date.assert_called_once_with([])

    def test_scan_arch_when_is_newer(self):
        # Prepare data and mocks
        with patch('builtins.open', mock_open()) as mock_file:
            config = {
                'update': {
                    'warn_last_update_interval_days': '2'
                }
            }
            update = Update(config)
            update.get_pacman_last_update_date = MagicMock(return_value=datetim
                e.today() - timedelta(days=1))

            # Run test scenario
```

```
            result = update.scan_arch()

            # Assertions
            self.assertEqual(result[0], ScanStatus.success)
            mock_file.assert_called_once_with('/var/log/pacman.log')
            update.get_pacman_last_update_date.assert_called_once_with([])

    def test_scan_arch_when_last_update_date_not_found(self):
        # Prepare data and mocks
        with patch('builtins.open', mock_open()) as mock_file:
            config = {
                'update': {
                    'warn_last_update_interval_days': '2'
                }
            }
            update = Update(config)
            update.get_pacman_last_update_date = MagicMock(return_value=None)

            # Run test scenario
            result = update.scan_arch()

            # Assertions
            self.assertEqual(result[0], ScanStatus.unknown)
            mock_file.assert_called_once_with('/var/log/pacman.log')
            update.get_pacman_last_update_date.assert_called_once_with([])

    def test_scan_when_unknown(self):
        # Prepare data and mocks
        with patch('platform.linux_distribution', lambda: ('Unknown linux distri
            bution', None, None)):
            update = Update(None)

            # Run test scenario
            result = update.scan()

            # Assertions
            self.assertEqual(result[0], ScanStatus.unknown)

    def test_scan_when_arch(self):
        # Prepare data and mocks
        with patch('platform.linux_distribution', lambda: ('arch', None, None)):
            update = Update(None)
            update.scan_arch = MagicMock(return_value=(ScanStatus.success, 'mess
                age'))

            # Run test scenario
            result = update.scan()
```

```
        # Assertions
        self.assertEqual(result, (ScanStatus.success, 'message'))
        update.scan_arch.assert_called_once_with()

    def test_scan_when_debian(self):
        # Prepare data and mocks
        with patch('platform.linux_distribution', lambda: ('debian', None, None)
            ):
            update = Update(None)
            update.scan_debian = MagicMock(return_value=(ScanStatus.success, 'me
                ssage'))

            # Run test scenario
            result = update.scan()

            # Assertions
            self.assertEqual(result, (ScanStatus.success, 'message'))
            update.scan_debian.assert_called_once_with()

    def test_scan_when_redhat(self):
        # Prepare data and mocks
        with patch('platform.linux_distribution', lambda: ('redhat', None, None)
            ):
            update = Update(None)
            update.scan_redhat = MagicMock(return_value=(ScanStatus.unknown, 'me
                ssage'))

            # Run test scenario
            result = update.scan()

            # Assertions
            self.assertEqual(result, (ScanStatus.unknown, 'message'))
            update.scan_redhat.assert_called_once_with()

    def test_get_apt_last_update_date_when_file_does_not_exist(self):
        # Prepare data and mocks
        with patch('builtins.open', raise_file_not_found_error):
            update = Update(None)

            # Run test scenario
            result = update.get_apt_last_update_date('/tmp/this/file/does/not/ex
                ist')

            # Assertions
            self.assertIsNone(result)

    def test_get_apt_last_update_date_when_file_exists(self):
        # Prepare data and mocks
```

```
        with patch('builtins.open', raise_file_not_found_error), NamedTemporaryF
            ile() as temp_file:
            update = Update(None)
            seconds_epoch_in_1999 = 923398970
            os.utime(temp_file.name, (seconds_epoch_in_1999, seconds_epoch_in
                _1999))

            # Run test scenario
            result = update.get_apt_last_update_date(temp_file.name)

            # Assertions
            self.assertEqual(result.timestamp(), seconds_epoch_in_1999)

    def test_scan_debian_when_is_older(self):
        # Prepare data and mocks
        config = {
            'update': {
                'warn_last_update_interval_days': '2'
            }
        }
        update = Update(config)
        update.get_apt_last_update_date = MagicMock(return_value=datetime.toda
            y() - timedelta(days=10))

        # Run test scenario
        result = update.scan_debian()

        # Assertions
        self.assertEqual(result[0], ScanStatus.fail)
        update.get_apt_last_update_date.assert_called_once_with('/var/lib/apt/pe
            riodic/update-success-stamp')

    def test_scan_debian_when_is_newer(self):
        # Prepare data and mocks
        config = {
            'update': {
                'warn_last_update_interval_days': '2'
            }
        }
        update = Update(config)
        update.get_apt_last_update_date = MagicMock(return_value=datetime.toda
            y() - timedelta(days=1))

        # Run test scenario
        result = update.scan_debian()

        # Assertions
        self.assertEqual(result[0], ScanStatus.success)
```

```
        update.get_apt_last_update_date.assert_called_once_with('/var/lib/apt/pe
            riodic/update-success-stamp')

    def test_scan_debian_when_last_update_date_not_found(self):
        # Prepare data and mocks
        config = {
            'update': {
                'warn_last_update_interval_days': '2'
            }
        }
        update = Update(config)
        update.get_apt_last_update_date = MagicMock(return_value=None)

        # Run test scenario
        result = update.scan_debian()

        # Assertions
        self.assertEqual(result[0], ScanStatus.unknown)
        update.get_apt_last_update_date.assert_called_once_with('/var/lib/apt/pe
            riodic/update-success-stamp')

    def test_scan_redhat(self):
        # Prepare data and mocks
        update = Update(None)

        # Run test scenario
        result = update.scan_redhat()

        # Assertions
        self.assertEqual(result[0], ScanStatus.unknown)


if __name__ == '__main__':
    unittest.main()
```

Սկզբնական կոդ 12: `worldwritable.py`

```
import os
import stat

from scanner.base_scanner import BaseScanner
from scanner.scan_status import ScanStatus


class WorldWritable(BaseScanner):
    def __init__(self, config):
        # config is a dictionary of this program's configuration
        self.config = config
```

```
def scan(self):
    # Scans the file system for world writable files and directories with pe
        rmission anomalies
    worldwritable_files_starting_with_dot = self.scan_worldwritable_files_st
        arting_with_dot()
    worldwritable_directories_with_no_sticky_bit_set = self.scan_worldwritab
        le_directories_with_no_sticky_bit_set()
    worldwritable_files_owned_by_root = self.scan_worldwritable_files_owned_
        by_root()
    if not worldwritable_files_starting_with_dot and \
            not worldwritable_directories_with_no_sticky_bit_set and \
            not worldwritable_files_owned_by_root:
        scan_status = ScanStatus.success
        message = ''
    else:
        scan_status = ScanStatus.fail
        message_parts = []
        if worldwritable_files_starting_with_dot:
            message_parts.append(self.get_scan_text('World writable files st
                arting with dot',
                                                    worldwritable_files_star
                                                        ting_with_dot))
        if worldwritable_directories_with_no_sticky_bit_set:
            message_parts.append(self.get_scan_text('World writable director
                ies with no sticky bit set',
                                                    worldwritable_directorie
                                                        s_with_no_sticky_bit
                                                        _set))
        if worldwritable_files_owned_by_root:
            message_parts.append(self.get_scan_text('World writable files ow
                ned by root',
                                                    worldwritable_files_owne
                                                        d_by_root))
        message = '\n'.join(message_parts)
    return scan_status, message

def get_scan_text(self, check_name, locations):
    # Generates a formatted string for given check's name and file locations
    if locations:
        return 'Failure: ' + check_name + ':\n\t' + '\n\t'.join(locations)
    else:
        return 'Success: ' + check_name

def scan_worldwritable_directories_with_no_sticky_bit_set(self):
    # Searches for worldwritable directories in the system which do not have
         the sticky bit set
    # Returns a list of such directories
    result = []
```

```
        for directory, subdirectories, files in os.walk('/'):
            if self.is_world_writable(directory) and not self.is_sticky_bit_set(
                directory):
                result.append(directory)
        return result

    def scan_worldwritable_files_starting_with_dot(self):
        # Searches for worldwritable files in the system whose name starts with
            dot
        # Returns a list of such files
        result = []
        for directory, subdirectories, files in os.walk('/'):
            for file in files:
                if self.is_world_writable(file) and self.is_starts_with_dot(fil
                    e):
                    result.append(file)
        return result

    def scan_worldwritable_files_owned_by_root(self):
        # Searches for worldwritable files in the system which are owned by root
        # Returns a list of such files
        result = []
        for directory, subdirectories, files in os.walk('/'):
            for file in files:
                if self.is_world_writable(file) and self.is_owned_by_root(file):
                    result.append(file)
        return result

    def is_world_writable(self, path):
        # Checks whether the file or directory at given path is world writable
        # Only the "Others" number in permission triple is checked for the prese
            nce of "Writeable" bit
        # Returns true if is world writable, false otherwise
        # Returns false if the file is not found
        try:
            file_statistics = os.stat(path)
        except FileNotFoundError:
            return False
        return bool(file_statistics.st_mode & stat.S_IWOTH)

    def is_sticky_bit_set(self, path):
        # Checks whether the file or directory at given path has sticky bit set
        # Returns true if is set, false otherwise
        # Returns false if the file is not found
        try:
            file_statistics = os.stat(path)
        except FileNotFoundError:
            return False
```

```
        return bool(file_statistics.st_mode & stat.S_ISVTX)

    def is_owned_by_root(self, path):
        # Checks whether the file or directory at given path is owned by the roo
            t user
        # A root user has UID of 0
        # Returns true if is owned by root, false otherwise
        # Returns false if the file is not found
        try:
            file_statistics = os.stat(path)
        except FileNotFoundError:
            return False
        return file_statistics.st_uid == 0

    def is_starts_with_dot(self, path):
        # Checks whether name of the file or directory at given path starts with
             dot
        # Returns true if does, false otherwise
        # Returns false if the file is not found
        basename = os.path.basename(path)
        if not basename:
            return False
        return basename[0] == '.'
```

Սկզբնական կոդ 13: `worldwritable_test.py`

```
import os
import unittest
from unittest.mock import patch, MagicMock, call

from scanner.scan_status import ScanStatus
from scanner.worldwritable import WorldWritable


# noinspection PyUnusedLocal
def raise_file_not_found_error(x):
    raise FileNotFoundError


class Test(unittest.TestCase):
    def test_scan_when_success(self):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        test_subject.scan_worldwritable_files_starting_with_dot = MagicMock(retu
            rn_value=[])
        test_subject.scan_worldwritable_directories_with_no_sticky_bit_set = Mag
            icMock(return_value=[])
        test_subject.scan_worldwritable_files_owned_by_root = MagicMock(return_v
            alue=[])
```

```
        # Run test scenario
        result = test_subject.scan()

        # Assertions
        test_subject.scan_worldwritable_files_starting_with_dot.assert_called_on
            ce_with()
        test_subject.scan_worldwritable_directories_with_no_sticky_bit_set.asser
            t_called_once_with()
        test_subject.scan_worldwritable_files_owned_by_root.assert_called_once_w
            ith()
        self.assertEqual(result[0], ScanStatus.success)
        self.assertEqual(result[1], '')

    def test_scan_when_two_failures(self):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        test_subject.scan_worldwritable_files_starting_with_dot = MagicMock(retu
            rn_value=[])
        test_subject.scan_worldwritable_directories_with_no_sticky_bit_set = Mag
            icMock(return_value=['/some/failure'])
        test_subject.scan_worldwritable_files_owned_by_root = MagicMock(return_v
            alue=['/other/failure'])
        test_subject.get_scan_text = MagicMock(side_effect=['Test2 Failed', 'Tes
            t3 Failed'])

        # Run test scenario
        result = test_subject.scan()

        # Assertions
        test_subject.get_scan_text.assert_has_calls(
            [
                call('World writable directories with no sticky bit set', ['/som
                    e/failure']),
                call('World writable files owned by root', ['/other/failure']),
            ]
        )
        test_subject.scan_worldwritable_files_starting_with_dot.assert_called_on
            ce_with()
        test_subject.scan_worldwritable_directories_with_no_sticky_bit_set.asser
            t_called_once_with()
        test_subject.scan_worldwritable_files_owned_by_root.assert_called_once_w
            ith()
        self.assertEqual(result[0], ScanStatus.fail)
        self.assertEqual(result[1], 'Test2 Failed\nTest3 Failed')

    def test_get_scan_text_when_list_does_not_have_items(self):
        # Prepare data and mocks
```

```
        test_subject = WorldWritable(None)

        # Run test scenario
        result = test_subject.get_scan_text('Check name', [])

        # Assertions
        self.assertEqual(result, 'Success: Check name')

    def test_get_scan_text_when_list_has_items(self):
        # Prepare data and mocks
        test_subject = WorldWritable(None)

        # Run test scenario
        result = test_subject.get_scan_text('Check name', ['file1', 'file2'])

        # Assertions
        self.assertEqual(result, 'Failure: Check name:\n\tfile1\n\tfile2')

    @unittest.mock.patch('scanner.worldwritable.os')
    def test_scan_worldwritable_directories_with_no_sticky_bit_set(self, mock_o
        s):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        mock_os.walk.return_value = [
            ('/world/writable/not/sticky', (), ()),
            ('/not/world/writable/not/sticky', (), ()),
            ('/world/writable/sticky', (), ()),
        ]
        test_subject.is_world_writable = MagicMock(side_effect=[True, False, Tru
            e])
        test_subject.is_sticky_bit_set = MagicMock(side_effect=[False, True])

        # Run test scenario
        result = test_subject.scan_worldwritable_directories_with_no_sticky_bit_
            set()

        # Assertions
        self.assertEqual(result, ['/world/writable/not/sticky'])
        mock_os.walk.assert_called_once_with('/')
        test_subject.is_world_writable.assert_has_calls(
            [
                call('/world/writable/not/sticky'),
                call('/not/world/writable/not/sticky'),
                call('/world/writable/sticky'),
            ]
        )
        test_subject.is_sticky_bit_set.assert_has_calls(
            [
```

```
            call('/world/writable/not/sticky'),
            call('/world/writable/sticky'),
        ]
    )

@unittest.mock.patch('scanner.worldwritable.os')
def test_scan_worldwritable_files_starting_with_dot(self, mock_os):
    # Prepare data and mocks
    test_subject = WorldWritable(None)
    mock_os.walk.return_value = [
        ('/dir1', ('/dir1/subdir1',), ('world_writable_starting_with_dot', '
            not_world_writable_starting_with_dot')),
        ('/dir2', (), ()),
        ('/dir3', ('/dir3/subdir3',), ('world_writable_not_starting_with_dot
            ',)),
    ]
    test_subject.is_world_writable = MagicMock(side_effect=[True, False, Tru
        e])
    test_subject.is_starts_with_dot = MagicMock(side_effect=[True, False])

    # Run test scenario
    result = test_subject.scan_worldwritable_files_starting_with_dot()

    # Assertions
    self.assertEqual(result, ['world_writable_starting_with_dot'])
    mock_os.walk.assert_called_once_with('/')
    test_subject.is_world_writable.assert_has_calls(
        [
            call('world_writable_starting_with_dot'),
            call('not_world_writable_starting_with_dot'),
            call('world_writable_not_starting_with_dot'),
        ]
    )
    test_subject.is_starts_with_dot.assert_has_calls(
        [
            call('world_writable_starting_with_dot'),
            call('world_writable_not_starting_with_dot'),
        ]
    )

@unittest.mock.patch('scanner.worldwritable.os')
def test_scan_worldwritable_files_owned_by_root(self, mock_os):
    # Prepare data and mocks
    test_subject = WorldWritable(None)
    mock_os.walk.return_value = [
        ('/dir1', ('/dir1/subdir1',), ('world_writable_owned_by_root', 'not_
            world_writable_owned_by_root')),
        ('/dir2', (), ()),
```

```
            ('/dir3', ('/dir3/subdir3',), ('world_writable_not_owned_by_root',))
                ,
        ]
        test_subject.is_world_writable = MagicMock(side_effect=[True, False, Tru
            e])
        test_subject.is_owned_by_root = MagicMock(side_effect=[True, False])

        # Run test scenario
        result = test_subject.scan_worldwritable_files_owned_by_root()

        # Assertions
        self.assertEqual(result, ['world_writable_owned_by_root'])
        mock_os.walk.assert_called_once_with('/')
        test_subject.is_world_writable.assert_has_calls(
            [
                call('world_writable_owned_by_root'),
                call('not_world_writable_owned_by_root'),
                call('world_writable_not_owned_by_root'),
            ]
        )
        test_subject.is_owned_by_root.assert_has_calls(
            [
                call('world_writable_owned_by_root'),
                call('world_writable_not_owned_by_root'),
            ]
        )

    @unittest.mock.patch('scanner.worldwritable.os')
    def test_is_world_writable_when_file_does_not_exist(self, mock_os):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        mock_os.stat.side_effect = raise_file_not_found_error
        path = '/does/not/actually/exist'

        # Run test scenario
        result = test_subject.is_world_writable(path)

        # Assertions
        self.assertFalse(result)
        mock_os.stat.assert_called_once_with(path)

    @unittest.mock.patch('scanner.worldwritable.os')
    def test_is_world_writable_when_is(self, mock_os):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        mock_os.stat.return_value = os.stat_result((0o40002, 0, 0, 0, 0, 0, 0,
            0, 0, 0))
        path = '/does/not/actually/exist'
```

```
        # Run test scenario
        result = test_subject.is_world_writable(path)

        # Assertions
        self.assertTrue(result)
        mock_os.stat.assert_called_once_with(path)

    @unittest.mock.patch('scanner.worldwritable.os')
    def test_is_world_writable_when_is_not(self, mock_os):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        mock_os.stat.return_value = os.stat_result((0o40005, 0, 0, 0, 0, 0, 0,
            0, 0, 0))
        path = '/does/not/actually/exist'

        # Run test scenario
        result = test_subject.is_world_writable(path)

        # Assertions
        self.assertFalse(result)
        mock_os.stat.assert_called_once_with(path)

    @unittest.mock.patch('scanner.worldwritable.os')
    def test_is_sticky_bit_set_when_is_not_set(self, mock_os):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        mock_os.stat.return_value = os.stat_result((0o00000, 0, 0, 0, 0, 0, 0,
            0, 0, 0))
        path = '/does/not/actually/exist'

        # Run test scenario
        result = test_subject.is_sticky_bit_set(path)

        # Assertions
        self.assertFalse(result)
        mock_os.stat.assert_called_once_with(path)

    @unittest.mock.patch('scanner.worldwritable.os')
    def test_is_sticky_bit_set_when_is_set(self, mock_os):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        mock_os.stat.return_value = os.stat_result((0o01000, 0, 0, 0, 0, 0, 0,
            0, 0, 0))
        path = '/does/not/actually/exist'

        # Run test scenario
        result = test_subject.is_sticky_bit_set(path)
```

```
        # Assertions
        self.assertTrue(result)
        mock_os.stat.assert_called_once_with(path)

    @unittest.mock.patch('scanner.worldwritable.os')
    def test_is_sticky_bit_set_when_file_does_not_exist(self, mock_os):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        mock_os.stat.side_effect = raise_file_not_found_error
        path = '/does/not/actually/exist'

        # Run test scenario
        result = test_subject.is_sticky_bit_set(path)

        # Assertions
        self.assertFalse(result)
        mock_os.stat.assert_called_once_with(path)

    @unittest.mock.patch('scanner.worldwritable.os')
    def test_is_owned_by_root_when_is_not(self, mock_os):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        mock_os.stat.return_value = os.stat_result((0, 0, 0, 0, 1, 0, 0, 0, 0,
            0))
        path = '/does/not/actually/exist'

        # Run test scenario
        result = test_subject.is_owned_by_root(path)

        # Assertions
        self.assertFalse(result)
        mock_os.stat.assert_called_once_with(path)

    @unittest.mock.patch('scanner.worldwritable.os')
    def test_is_owned_by_root_when_is(self, mock_os):
        # Prepare data and mocks
        test_subject = WorldWritable(None)
        mock_os.stat.return_value = os.stat_result((0, 0, 0, 0, 0, 0, 0, 0, 0,
            0))
        path = '/does/not/actually/exist'

        # Run test scenario
        result = test_subject.is_owned_by_root(path)

        # Assertions
        self.assertTrue(result)
        mock_os.stat.assert_called_once_with(path)
```

```
@unittest.mock.patch('scanner.worldwritable.os')
def test_is_owned_by_root_when_file_does_not_exist(self, mock_os):
    # Prepare data and mocks
    test_subject = WorldWritable(None)
    mock_os.stat.side_effect = raise_file_not_found_error
    path = '/does/not/actually/exist'

    # Run test scenario
    result = test_subject.is_owned_by_root(path)

    # Assertions
    self.assertFalse(result)
    mock_os.stat.assert_called_once_with(path)

def test_is_starts_with_dot_when_does(self):
    # Prepare data and mocks
    test_subject = WorldWritable(None)
    path = '/starts/with/.dot'

    # Run test scenario
    result = test_subject.is_starts_with_dot(path)

    # Assertions
    self.assertTrue(result)

def test_is_starts_with_dot_when_does_not(self):
    # Prepare data and mocks
    test_subject = WorldWritable(None)
    path = '/does/not/start/with/dot'

    # Run test scenario
    result = test_subject.is_starts_with_dot(path)

    # Assertions
    self.assertFalse(result)

def test_is_starts_with_dot_when_empty_string(self):
    # Prepare data and mocks
    test_subject = WorldWritable(None)
    path = ''

    # Run test scenario
    result = test_subject.is_starts_with_dot(path)

    # Assertions
    self.assertFalse(result)
```

```
if __name__ == '__main__':
    unittest.main()
```

Սկզբնական կոդ 14: `root.py`

```
import os
from scanner.base_scanner import BaseScanner
from scanner.scan_status import ScanStatus


class Root(BaseScanner):
    def __init__(self, config):
        # config is a dictionary of this program's configuration
        self.config = config

    def scan(self):
        """
        Checks whether the user running this program is the root user with UID o
            f 0.
        Takes into account the usage of the program sudo.
        :returns: a tuple of (ScanStatus, message).
        """
        if os.getuid() == 0:
            if os.environ.get('SUDO_UID'):
                return ScanStatus.success, ''
            else:
                return ScanStatus.fail, 'Do not use the root user account'
        else:
            return ScanStatus.success, ''
```

Սկզբնական կոդ 15: `root_test.py`

```
import unittest
from unittest.mock import patch, MagicMock

from scanner.root import Root
from scanner.scan_status import ScanStatus


class Test(unittest.TestCase):
    @unittest.mock.patch('scanner.root.os')
    def test_scan_when_getuid_is_other(self, mock_os):
        # Prepare data and mocks
        mock_os.getuid = MagicMock(return_value=1000)

        # Run test scenario
        result = Root(None).scan()

        # Assertions
```

```
            mock_os.getuid.assert_called_once_with()
            self.assertEquals(result[0], ScanStatus.success)

        @unittest.mock.patch('scanner.root.os')
        def test_scan_when_getuid_is_0_and_environ_sudo_uid_is_set(self, mock_os):
            # Prepare data and mocks
            mock_os.getuid = MagicMock(return_value=0)
            mock_os.environ.get = MagicMock(return_value=1000)

            # Run test scenario
            result = Root(None).scan()

            # Assertions
            mock_os.getuid.assert_called_once_with()
            mock_os.environ.get.assert_called_once_with('SUDO_UID')
            self.assertEquals(result[0], ScanStatus.success)

        @unittest.mock.patch('scanner.root.os')
        def test_scan_when_getuid_is_0_and_environ_sudo_uid_is_not_set(self, mock_o
            s):
            # Prepare data and mocks
            mock_os.getuid = MagicMock(return_value=0)
            mock_os.environ.get = MagicMock(return_value=None)

            # Run test scenario
            result = Root(None).scan()

            # Assertions
            mock_os.getuid.assert_called_once_with()
            mock_os.environ.get.assert_called_once_with('SUDO_UID')
            self.assertEquals(result[0], ScanStatus.fail)


    if __name__ == '__main__':
        unittest.main()
```

Իրականացման աշխատանքը

Նկար 1: Իրականացման աշխատանքի օրինակ

```
Running: Update
Finished: Update
Status: ScanStatus.fail
Message:

--------------------------------------------------------------------------------
Running: OpenPorts
Finished: OpenPorts
Status: ScanStatus.success
Message:
Type, IP, Port, PID, Username, Command line
tcp 127.0.0.1:63342 20837 babken /opt/pycharm-community/bin/../jre/jre/bin/java -Xbootclasspath/a:/op
--------------------------------------------------------------------------------
Running: WorldWritable
Finished: WorldWritable
Status: ScanStatus.fail
Message:
Failure: World writable directories with no sticky bit set:
    /home/babken/Quake 2
    /home/babken/Quake 2/docs
    /home/babken/Quake 2/docs/quake2_manual
    /home/babken/Quake 2/docs/quake2_manual/images
    /home/babken/Quake 2/Q2
    /home/babken/Quake 2/Q2/docs
    /home/babken/Quake 2/Q2/docs/quake2_manual
    /home/babken/Quake 2/Q2/docs/quake2_manual/images
    /home/babken/Quake 2/Q2/ROGUE
    /home/babken/Quake 2/Q2/baseq2
    /home/babken/Quake 2/Q2/baseq2/video
    /home/babken/Quake 2/Q2/baseq2/maps
    /home/babken/Quake 2/Q2/baseq2/save
```