ბინარული კოდის უკუ-ინჟინერია

ოთარ შონია, დავით ნაჭყებია
საქართველოს ტექნიკური უნივერსიტეტი

რეზიუმე

განხილულია კომპილირებული პროგრამული კოდის უკუ-ინჟინერიის საკითხები. განიხილება რივერს-ენჯანირინგის გამოყენების ძირითადი სფეროები, უკუ-ინჟინერიის მნიშვნელობა ვირუსული კოდის წინააღმდეგ ბრძოლაში, ასევე მისი საკვანძო როლი პროგრამული უზრუნველყოფის მეკობრეობაში. მოცემულია პროცესორულ კოდში და ბაიტკოდში წარმოდგენილი კოდის მოდულების შესწავლის სირთულეები და თავისებურებები. აღნიშნულია მიზეზები, რის გამოც ეს სფერო ნაკლებად შესწავლილია აკადემიურ დონეზე.

საკვანძო სიტყვები: უკუ-ინჟინერია, რივერს-ენჯანირინგი, პროგრამული უზრუნველყოფის მეკობრეობა, კომპიუტერული ვირუსი, დისასემბლირება, დეკომპილაცია.

1. შესავალი

პროგრამული უზრუნველყოფის მდგომარეობა, მისი ფიზიკური წარმოდგენის მიხედვით, შეგვიძლია უხეშად ორ კატეგორიად დავყოთ - კომპილაციამდე, და კომპილაციის შემდეგ. კომპილაციამდე პროგრამული კოდი ღია მდგომარეობაშია, ანუ მისი შესწავლა და მოდიფიცირება სტანდარტული გზით ხორციელდება და პრობლემას არ წარმოადგენს ადამიანისთვის რომელსაც ეს კოდი გააჩნია. კოდი ადამიანისათვის გასაგები ინსტრუქციებისაგან შედგება და თავისთავად ამ ფორმაში უტილიზებისთვის არ არის მზად. თუმცა პროგრამული პროდუქტის დანიშნულებისამებრ გამოყენებისათვის აუცილებელია მისი კომპილაცია - განსაზღვრულ აპარატურულ მოწყობილობებზე შესასრულებლად.

კომპილაცია საზოგადოდ ფარდობითი ცნებაა და მრავალნაირად შეიძლება იქნეს განმარტებული. თუმცა მოცემული საუბრის ფარგლებში კომპილაციის ქვეშ ვიგულისხმოთ პროგრამული კოდის გადაყვანა იმ მდგომარეობაში, რომელშიც მისი შესრულება სამიზნე პროგრამულ/აპარატურულ სისტემაში იქნება შესაძლებელი. საბოლოოდ კოდი ადრე თუ გვიან უშუალოდ პროცესორის მიერ უნდა იქნეს შესრულებული, და ამდენად პროცესორისთვის გასაგებ დაბალი დონის მიკრო-ინსტრუქციებში თარგმნილი. გარკვეული პროგრამული პლატფორმებისთვის (მაგალითად C\C++\Delphi) კომპილაცია როგორც წესი პირდაპირ ნიშნავს ტექსტური კოდის გადაყვანას სწორედაც პროცესორულ კოდში. სხვა პლატფორმებისთვის კომპილაცია კოდის საშუალედო ფორმატში (რომელსაც ზოგადად ბაიტკოდს უწოდებენ) გადაყვანას გულისხმობს. ასეთი საშუალედო ფორმატის მაგალითს წარმოადგენს CIL (Common Intermediate Language) მაკროსოფტის .NET Framework-ის შემთხვევაში და Java-ს ბაიტკოდი. სტანდარტულად ბაიტკოდი რანთაიმში დინამიურად ან პირდაპირ პროცესორულ კოდში ითარგმნება (Just-in-time compilation – JIT), ანაც ინტერპრეტირდება შესაბამისი ინტერპრეტატორის მიერ (რომელიც თავის მხრივ პროცესორულ კოდში იშვება). პროგრამულ გარსს რომელიც ამ პროცესს უზრუნველყოფს ვირტუალურ მანქანას (Virtual machine) უწოდებენ.

ნებისმიერ შემთხვევაში, პროგრამული პროდუქტი წარმოდგენილია პირდაპირ პროცესორულ კოდში თუ საშუალედო ბაიტკოდში, კომპილაციის შემდეგ იგი როგორც წესი არ გულისხმობს ადამიანის მიერ მის შესწავლას და მითუმეტეს მოდიფიცირებას. ეს ფაქტი ზოგადად

პროგრამულ/აპარატურული არქიტექტურით არის განპირობებული, რადგან ადამიანისთვის გასაგები ინსტრუქციები ანუ სორს-კოდი უკვე აღარ არის წარმოდგენილი პროდუქტის შესაბამის ფაილებში. ამ ფაქტს ერთი მნიშვნელოვანი იმპლიკაცია ახლავს - პროგრამული უზრუნველყოფის ავტორებს აქვთ შესაძლებლობა პროგრამული პროდუქტი საჭიროების შემთხვევაში გაავრცელონ ისე რომ მისი მოქმედების პრინციპების შესწავლა და მოდიფიცირება შეუძლებელი აღმოჩნდება სტანდარტული მექანიზმებით ნებისმიერი გარეშე პირისათვის, რომელსაც სორს-კოდი არ გააჩნია. თუმცა არსებობს ,,არასტანდარტული მექანიზმები", რომლებიც დეკომპილაციისა თუ დისასემბლირების გზით საშუალებას იძლევა შესწავლილი იქნეს კომპილირებული პროგრამული კოდი, და საჭიროების შემთხვევაში - მოდიფიცირებული. აღნიშნულ პროცესს, მოცემული ტექსტის ფარგლებში, მოვიაზრებთ როგორც შეტევას, რამდენადაც როგორც წესი იგი პროგრამული უზრუნველყოფის - როგორც ინტელექტუალური საკუთრების - ფლობელის ნების საწინააღმდეგო ქმედებას წარმოადგენს. ასეთი შეტევა შეიძლება არღვევდეს გარკვეული სუბიექტების უფლებებს, და გარკვეული კანონმდებლობის ფარგლებში აკრძალულიც იყოს რაიმე ფორმით, თუმცა შესაძლოა სრულიად კანონიერ, და მეტიც - სასარგებლო საქმიანობას წარმოადგენდეს. თუმცა ამაზე ოდნავ მოგვიანებით.

ძირითადი ნაწილი

საინფორმაციო ტექნოლოგიების იმ ქვემიმართულებას, რომელიც კომპილირებული პროგრამული უზრუნველყოფის შესწავლის (ხშირად შემდგომი მოდიფიკაციისთვის, რომელიც როგორც წესი პროდუქტის ავტორის ნების საწინააღმდეგოდ ხორციელდება) მეთოდებს იკვლევს, მოიხსენიებენ როგორც Reverse Code Engineering, ხშირად აბრევიატურით RCE, რაც პირდაპირი ქართული თარგმნით კოდის უკუ-ინჟინერიას ნიშნავს. ამ მიმართულების კვლევები ყველაზე მეტად რელევანტურია IT ინდუსტრიის შემდეგი ორი დარგის ფარგლებში: ვირუსული კოდის ანალიზი, და პროგრამული უზრუნველყოფის მეკობრეობასთან (Software piracy) ბრძოლა. ქვემოთ მოკლედ მიმოვიხილავ თითოეულს.

მალვეარის (Malware, malicious software) წინააღმდეგ ბრძოლაში გადამწყვეტ როლს სწორედ რივერს-ენჯანირინგი ასრულებს. ვირუსები როგორც წესი პროცესორულ კოდში კომპილირებული სახით ვრცელდება, და ბუნებრივია მათი შესწავლა აუცილებელია პრევენციული ზომების მისაღებად. ვირუს-მეიკერები ცდილობენ ხელოვნურად გაართულონ ეს პროცესი, და ამისათვის სხვადასხვა მექანიზმებს იგონებენ. მეორე მხარეს კი ინფორმაციული უსაფრთხოების ექსპერტები ჯერ ამ მექანიზმების განეიტრალებას, ხოლო შემდეგ კოდის შესწავლას და საწინააღმდეგო, თავდაცვითი მეთოდების შემუშავებას ცდილობენ.

პროგრამული უზრუნველყოფის მეკობრეობა იწყება და მთავრდება რივერს-ენჯანირინგით. მისი საშუალებით ჰაკერები პროგრამული პროდუქტის ლიცენზირების სისტემის და თავდაცვითი მექანიზმების შესწავლას ახორციელებენ, და ახდენენ მათ ნეიტრალიზებას რათა შესაძლებელი გახადონ პროდუქტის არასანქცირებული მოხმარება და დისტრიბუცია. მეორე მხარეს, პროგრამული უზრუნველყოფის ავტორები, უფრო ზუსტად კი როგორც წესი მათ მიერ დაქირავებული ინფორმაციული უსაფრთხოების ექსპერტები (რომლებიც გარკვეული გაგებით თავად წარმოადგენენ ჰაკერებს) ახდენენ თავდაცვითი მექანიზმების შემუშავებას, რათა მაქსიმალურად გაართულონ ეს პროცესი.

ზემოაღნიშნული ორი მიმართულების გარდა, არსებობს რამდენიმე დარგი, სადაც RCE აქტიურად გამოიყენება. მაგალითად დახურული კოდის პროგრამული უზრუნველყოფის შესწავლა შეიძლება მოხდეს ბაგების ან ექსპლოიტების აღმოსაჩენად, ასევე სხვა პროგრამულ სისტემებთან ინტეგრაციისათვის, როდესაც არ არსებობს სასურველი ხარისხის პროგრამული ინტერფეისი ამისათვის. თუმცა ზემოთ განხილული ორი მიმართულება განსაკუთრებულ აღნიშვნას იმსახურებს შემდეგი გარემოების გამო: ორივე შემთხვევაში საქმე გვაქვს ურთიერთსაპირისპირო ინტერესების მქონე ადამიანების ორ ჯგუფთან - შემტევები და დამცველები. ეს განაპირობებს RCE-ს სივრცეში კონკურენციას და ტექნოლოგიების მუდმივ განვითარებას. ერთ შემთხვევაში პირობითად „ბოროტ“ მხარეს დაცვა წარმოადგენს - ანუ ვირუსმეიკერები, ხოლო „კეთილ“ მხარეს - მალვეართან მებრძოლნი, მაგალითად ანტივირუსული კომპანიები, რომლებიც შემტევის როლში გვევლინებიან. მეორე შემთხვევაში პირიქითაა - დაცვის მხარეს პროგრამული უზრუნველყოფის მფლობელები წარმოადგენენ, რომელთაც საკუთარი ინტელექტუალური საკუთრების კანონიერი დაცვა სურთ, ხოლო მეორე მხარეს შემტევის როლში ჰაკერების სპეციალური კასტა - ე.წ. „კრეკერები“ გვევლინებიან რომლებიც ცდილობენ პროგრამული უზრუნველყოფის თავდაცვითი მექანიზმების „გატეხვას“ რაც არაპირდაპირ ასეთი პროგრამული პროდუქტების შავ ბაზარზე მოხვედრას განაპირობებს მოგვიანებით.

მიუხედავად იმისა, თუ „ფრონტის“ რომელ მხარეს დგანან ჰაკერები რომლებიც ამ პროცესში არიან ჩართულნი, მათ თითქმის ერთი თემატიკის გარშემო უწევთ მუშაობა და მათი უნარ-ჩვევები როგორც წესი მიახლოებით ერთნაირია. არის შემთხვევებიც როცა ერთიდაიგივე ჰაკერები ერთის-მხრივ ანტივირუსული კომპანიის კედლებში მუშაობენ, მეორეს მხრივ კი ამავდროულად ე.წ. „კრეკინგით“ არიან დაკავებულნი. RCE მათთვის განიხილება როგორც ერთგვარი სპორტი, გამოწვევა შესაძლებლობების დასამტკიცებლად, და არა მაინცდამაინც კონკრეტული მატერიალური მიზნის მისაღწევი საშუალება. ასეთ წრეებში ხშირად RCE-ს მოიხსენიებენ როგორც ხელოვნების დარგს. ამდენად ხშირ შემთხვაში, მე ვიტყოდი - უმრავლეს შემთხვევებში, ჰაკერები რომლებიც „ფრონტის“ „ბოროტ“ მხარეს წარმოადგენენ, თავიანთ ქმედებებს - რომლებიც შესაძლოა თავისუფლების ფასიც დაუჯდეთ - სრულიად უსასყიდლოდ, მხოლოდდამხოლოდ სპორტული ინტერესის, ან იდეური შეხედულებების გამო ახორციელებენ, და არა რაიმე მატერიალური სარგებლის მისაღებად.

მრავალი მიზეზის გამო, RCE ნაკლებად შესწავლილია აკადემიურ წრეებში. ერთერთ მიზეზს წარმოადგენს ის, რომ უახლესი მიღწევები ამ სფეროში რეალიზდება კომერციულ პროდუქტებში, და ინფორმაციის საჯარო გამომზეურება არამომგებიანია გარკვეული სუბიექტებისთვის. ამასთან, ასეთ კომერციულ პროდუქტებში აღმოჩენილი ნოუ-ჰაუების საჯაროდ გაცხადება მრავალი ქვეყნაში ინტელექტუალური საკუთრების დაცვის მოტივით სამართლებრივად აკრძალულია. კიდევ ერთი მიზეზი გახლავთ მსგავსი ინფორმაციის თავისუფალი გავრცელებისას შესაძლო საფრთხეები - ინფორმაციის „ბოროტად“ გამოყენების პრობლემა. ეს განაპირობებს იმას რომ ინფორმაცია რომელიც ამ სფეროში უახლეს, ბოლო მიღწევებს ასახავს, ძირითადად ჰაკერების ვიწრო წრეებში, ანდერგრაუნდ ონლაინ რესურსების საშუალებით იცვლება, რომლებიც ხშირ შემთხვევაში სხვადასხვა ქვეყნის სამართლებრივი სტრუქტურების მიერ იდევნებიან.

იმის გამო, რომ ნებისმიერი კომპილირებული პროდუქტი შესაბამის პროგრამულ თუ აპარატურულ სისტემასთან მჭიდროდ არის დაკავშირებული, სხვადასხვა პლატფორმაზე RCE-ს სხვადასხვა მექანიზმები გამოიყენება. პროცესორულ კოდში კომპილირებული პროდუქტის რივერს-ენჯანირინგისათვის აუცილებელია ერთის მხრივ შესაბამისი პროცესორის, და მეორეს მხრივ

შესაბამისი ოპერაციული სისტემის არქიტექტურის სიღრმისეული ცოდნა. ბაიტკოდში კომპილირებული მოდულის შესასწავლად კი აუცილებელია შესაბამისი რანთაიმ-გარსის, ვირტუალური მანქანის არქიტექტურის და ამდენად ბაიტკოდის ფორმატის ცოდნა. პროცესორის და ოპერაციული სისტემის არქიტექტურა ამ შემთხვევაში როგორც წესი მეორეხარისხოვანია. თუმცა იმ მარტივი ფაქტიდან გამომდინარე, რომ ნებისმიერი ბაიტკოდი საბოლოოდ მაინც პროცესორულ კოდში უნდა ითარგმნოს, შესაძლებელია ასეთი პროგრამული პროდუქტების შესწავლა სწორედ პროცესორულ დონეზე, ანუ იმ დროს როცა იგი რანთაიმში წარმოდგინდება უკვე პროცესორული კოდის სახით.

პროცესორულ კოდში კომპილირებული კოდის წარმოდგენა როგორც წესი ასეთივე დაბალი დონის ინსტრუქციების საშუალებითაა შესაძლებელი - ანუ ასემბლერის ენაზე. არსებობს სხვადასხვა დისასემბლერები და დებაგერები ამისათვის. დისასებლერი კოდის ოფლაინ ანალიზისთვის, ანუ სტატიკური შესწავლისთვის გამოიყენება, ხოლო დებაგერი - კოდის გაშვების დროს ანუ რანთაიმში მისი მონიტორინგისთვის. როგორც წესი ხდება ამ ორი მეთოდის კომბინირება კომპლექსური პროგრამული სისტემის შესწავლის დროს.

ბაიტკოდში კომპილირებული კოდის შესწავლა საზოგადოდ გაცილებით მარტივია. ამას განაპირობებს ის ფაქტი, რომ ბაიტკოდი როგორც წესი ბევრად უფრო მაღალი დონის პროგრამული სტრუქტურაა, ვიდრე პროცესორული კოდი. ის შეიცავს გაცილებით უფრო მეტ მაღალი დონის ინფორმაციას (მაგალითად ეს შეიძლება იყოს ფუნქციების თუ ცვლადების სახელები) ვიდრე პროცესორული კოდი. ამის გამო შესაძლებელია ბაიტკოდისაგან სტრუქტურულად კიდევ უფრო მაღალი დონის კოდის მიღება - ანუ დეკომპილაცია. დეკომპილაცია გულისხმობს ბაიტკოდის პირვანდელთან მიახლოებულ - ანუ სორს-კოდის დონის ერთგვარ ფსევდოკოდში წარმოდგენას. ეს ფსევდოკოდი შეიძლება იყოს ძალიან მიახლოებული, ან სულაც იდენტური პროგრამის რეალური სორს-კოდის. პროცესორული კოდის შემთხვევაში აღნიშნული „კომფორტი“ შეუძლებელია ერთის მხრივ ასეთი კოდის ძალიან დაბალი დონის პრიმიტიული ინსტრუქციების გამო, მეორეს მხრივ კომპილატორის მიერ ამ ინსტრუქციების აპარატურისადმი ძლიერ-ოპტიმიზირებულობის გამო (რის გამოც მაღალი დონის პროგრამული კონსტრუქციები ხშირად უბრალოდ ქრება), და მესამეს მხრივ მეტაინფორმაციის უდიდესი ნაწილის არარსებობის გამო, რომელიც კომპილაციის პროცესში იკარგება. ამდენად ასეთი კოდის შესწავლა როგორც წესი ასემბლერის ინსტრუქციების დონეზე ხორციელდება.

დასკვნა

ბინარული კოდის უკუ-ინჟინერია სპეციფიკური დანიშნულების კრიტიკულად მნიშვნელოვან დისციპლინას წარმოადგენს. ინტერესთა კონფლიქტი პროგრამული უზრუნველყოფის ავტორებსა და მათი შესწავლით დაინტერესებულ პროფესიონალებს შორის განაპირობებს ერთის მხრივ ურთულესი დაცვითი სისტებემის, და მეორეს მხრივ ამ სისტემების გაუვნებელყოფის კომპლექსური მექანიზმების შემუშავებას. რივერს-ენჯანირინგის დიდი როლი სხვადასხვა კანონსაწინააღმდეგო მოქმედებებში, ასევე კომერციული დაცვის სისტემებში რეალიზებული მიღწევების კონფიდენციალურობა, განაპირობებს ამ სფეროში ფართო პუბლიკისთვის ინფორმაციის დიდი ნაწილის მიუწვდომლობას.

ლიტერატურა:

1. ჩოგოვაძე გ., გოგიჩაიშვილი გ., სურგულაძე გ., შეროზია თ., შონია ო.. მართვის ავტომატიზებული სისტემების დაპროექტება და აგება. სტუ. თბილისი. 2001.
2. Raja, Vinesh; Fernandes, Kiran J.. Reverse Engineering – An Industrial Perspective. Springer. 2008.

BINARY CODE REVERSE ENGINEERING

Shonia Otar, Nachkebia David

Georgian Technical University

Summary

The article presents the issues of compiled software code reverse engineering. It discusses the main usage vectors of reverse engineering, its significance in a fight against malicious software and its key role in a software piracy. The problems and peculiarities of analyzing the binary modules compiled in processor code and byte-code are presented. The reasons why this field is relatively less researched at the academic level are explained.

ОБРАТНАЯ ИНЖЕНЕРИЯ БИНАРНОГО КОДА

Шония О., Начкебия Д.

Грузинский Технический Университет

Резюме

Рассмотрены вопросы обратной инженерии скомпилированного программного кода. Рассматриваются основные области использования риверс-енджаниринга, значение обратной инженерии в борьбе против вирусного кода, а также его узловая роль в пиратстве программного обеспечения. Показаны трудности и своеобразие изучения модулей кодов, представленных в процессорном коде и баиткоде. Указаны причины слабого изучения этой области на академическом уровне.