ROBLOX

რობლოქსის ექსპლოიტინგის სცენა წარსდგება რამოდენიმე ჯგუფისგან, ყველაზე პოპულარული იწყება External-იდან რომელიც უმეტეს შემთხვევაში შედგება ორი კომპონენტისგან IOCTL ანუ კერნელ Ring 0 დონეზე შემუშავებული დრაივერი რომელსაც შეუძლია დაამყაროს UserMode აპლიკაციასთან კავშირი რაც გვაძლევს უფლებას რო ჩვეულებრივი აპლიკაციიდან ვმართოთ მეხსიერება მაღალ დონეზე, რადგანაც Hyperion/Byfron არის Anti-Tamper და არა Anti-Cheat, რობლოქსში უკვე არის რამოდენიმე წითელი დროშები რომელსაც შეიძლება კერნელის დონის გამოკენებისასაც მიარტყათ და დრაივერი ვერ დაგიცავთ.

შემდეგ მოდის Internal, ამჯერად უკვე უფრო ძნელადაა საქმე რადგანაც ჩვენ არ შეგვიძლია მარტივად კერნელის გამოყენება რომ მეხსიერება წავიკითხოთ ჩვენ გვჭირდება უფრო ჭკვიანური გამოსავალი. გავითვალისწინოთ რომ Hyperion ამოწმებს ყველა DLL, თრედსა თუ მოდულსა რომელიც კავშირს ამყარებს რობლოქს-თან და ეს ხელს უწყობს მათ დაბლოკვაში რადგანაც Hyperion მხოლოდ რობლოქსზე მუშაობს ის ზუსტად გათვლილი Anti-Tamperია. ჩვენდა საბედნიეროდ Hyperion არ აპირებს Ring 0 ზე გადასვლას რაც ტოვებს მას Ring 3-ზე რომელზეც ჩვენ ვიმყოფებით, როგორც წელან ვახსენე Hyperion-ს შეუძლია კავშირების შემოწმება მაგრამ რადგანაც Hyperion-მა მოახერხა ეგ Ring 3-დან, ჩვენც შეგვიძლია მაგის ჩვენ გემოზე შეცვლა :)

ინტერნალი იმისდა მიუხედავად რომ ძნელად გამოიყურება თავიდან, როგორცაა ინჟექტორის გაკეთება არც ისე ძნელია მაგრამ არცერთი მეთოდი რომელიც სახალხოდაა ცნობილი არ არის უსაფრთხო მაშასადამე ვერ გამოვრიცხავთ იმას რომ შეიძლება ნებისმიერ წამს ბანი დაგვედოს. ერთერთ ინჟექტორის ყველაზე პოპულარული და ჯერჯერობით ასე თუ ისე უსაფრთხო მეთოდი არის WinVerifyTrust-ის დაწყებითი Byteბის მოდიფიცირება და ახლით ჩანაცვლდება რის შედეგადაც Hyperion ველარ შეძლებს კავშირების შემოწმებას.

რადგანაც საბაზისო ცოდნით გაწათლდით ალბათ გიჩნდებათ მარტივი კითხვა თუ რომელი ჯობია ინტერნალი თუ ექსტერნალი. მოდი იმით დავიწყოთ რა განსხვავებაა, რობლოქსში აქვს თითეულ სკრიპტს, პლუგინს, კონსოლს და ასე შემდეგ თავიანთი სამზრძანებლო დონე, დაახლოებით როგორც ძველი იერარქია ისე იყოს, გლეხები რომელთაც უფლებები არ გააჩნიათ ეს შეგიძლიათ შეადაროთ identity 0-ს, საშუალო მაცხოვრებელთა წარმომადგენლები რომელთაც გააჩნიათ სახელიდან გამომდინარე საშუალო უფლებები, არც იქეთ და არც აქეთ ანუ ეს შეგიძლიათ მოათავსოთ identity 2-ზე რომელზეც ეშვება სკრიპტები ლოქალსკრიპტები და ასე შემდეგ. შემდეგ მოდის აზნაური რომლებიც უფრო მალლა დგანან ვიდრე საშუალო დონის წარმომადგენელი ხალხი და გააჩნიათ მათზე მეტად ბევრი უფლება, ეს ჯდება identity 3 ზე რომელიც იყენებენ External ექსპლოიტები, მაგრამ როგორ? მარტივია, ძველი ტექნიკაა რომელიც პირველად იქნა გამოყენებული RC7 ექსპლოიტში, Bytecode Conversion მაგრამ ამის გასაგებად ჩვენ გვესაჭიროება გაგვმარტოთ Bytecode და შემდგომ Conversion, როცა C++ ში თქვენ ქმნით აპლიკაციას კომპაილერი ანუ MSVC ილებს თქვენს დაწერილ კოდს და თარგმნის შემდგომ ძალიან მარტივ ინსტრუქციებად სახელად assembly, რომელიც შემდგომ ოპტიმიზირებულია სიჩქარით ან ზომით და გარდაიქმნება შემდგომ გასახსნელ ფაილად(Executable file). ეს გასახსნელი ფაილი შეიცავს Bytecode-ს (და ნუ რამოდენიმე ინფორმაციასაც). მოდი ავლა Lua-ს Bytecode-ზე ვისაუბროთ რადგანაც ჩვენ ეს გვესაჭიროება და არა MSVC Bytecode.. როგორც გავიგეთ Bytecode შედგება ინსტრუქც-იებისგან მაგრამ Lua-ში რა ტიპის ინსტრუქციებით შედგება? მოდი გავიგოთ!

თითოეულ ინსტრუქციას აქვს ოთხი პარამეტრი: opcode, A რეგისტრი, B რეგისტრი და C რეგისტრი. თიტოეული მათგანი ერთი ბაითის სიგრძისაა, რომელიც ქმნის ერთ signed integer-ს. ინსტრუქცია, რომელიც იყენებს სამივე რეგისტრს, არის iABC ტიპის. ტარკვეული ინსტრუქციები აერთიანებს B რეგისტრს C რეგისტრთან, რათა შეიქმნას მოკლე (მთლიანი რიცხვი ორი ბაიტით), რომელსაც შეუძლია შეინახოს უფრო დიდი მნიშვნელობები. ეს ქმნის Bx (B გაფართოებულ) რეგისტრს. ინსტრუქცია, რომელიც იყენებს A და Bx რეგისტრებს, არის iABx ტიპის. (ინსტრუქცია, A, Bx) ზოგიერთ ინსტრუქციას აქვს ხელმოწერილი Bx რეგისტრი (ის შეიძლება შეიცავდეს უარყოფით მნიშვნელობებს). ისინი iAsBx ტიპისაა. (ინსტრუნქცია, A, signed Bx) და არის უნსტრიქცუები, რომლებიც უბრალოდ იყენებს A რეესტრს. ისინი iA ტიპის არიან. (ინსტრუქცია, A) თუმცა ეს იშვიათია.

მე ავღნიშნე opcodes, მაგრამ რა არის ის სინამდვილეში? opcodes არის ის, რაც ეუბნება ინტერპრეტერს რასაც ინსტრუქცია აკეთებს. თუ ინსტრუქციის opcode არის OP_ADD (iABC), მაშინ ის გვეუბნება ინტერპრეტერმა A რეესტრში ჩადოს B და C-ზე მდებარე მნიშვნელობების ჯამი. ლუას აქვს ზუსტად 37 ინსტრუქცია, რაც მცირეა, თუ შევადარებთ ლუას ინსტრუქციების კომპლექტს მაგ. x86-ის ინსტრუქციების ნაკრებისთვის (მინიშნება: საშუალოდ ~ 2000 ინსტრუქციაა. CPUები განსხვავდება, ამიტომ ყველას აქვს სხვადასხვა ინსტრუქციის ნაკრები, ამითომ ინსტრუქციების რაოდენობა განსხვავდება CPU-ს ბრენდისა და არქიტექტურის მიხედვით.)იმის გამო, რომ ოპკოდი ინახება როგორც ერთი ბაიტი, შეიძლება იყოს მაქსიმუმ 255 ინსტრუქციები, თუ ლუას განვითარების გუნდს სურდა მეტი ინსტრუქციის დამატება. ეს რიცხვი, მაშინ მათ უნდა გადაერთონ 64 ბიტიან მთელ რიცხვზე (8 ბაიტი). ესეც ნიშნავს, რომ მათ შეეძლებათ მეტი რეგისტრის დამატება, რაც შეიძლება ნიშნავდეს უფრო ეფექტურ VM-ს, მაგრამ ეს მხოლოდ ოცნებაა რომელიც არასოდეს ახდება.

მოდი განვიხილოთ ფაქტები, Lua Bytecode არის კომპილირებული Lua სკრიპტები (ეს გამოგვადგება მომავალში ანუ ჩეთის დაწერაში), როდესაც აწარმოებთ ლუა სკრიპტს, ლუა ავტომატურად აგროვებს მას შუამავალ ფორმატში ემსგავსება ბაიტკოდს ვირტუალურ მანქანაზე მიწოდებამდე. ეს "შუა მავალი ფორმატი" (რომელიც უბრალო პროტო სტრუქტურაა) შეიძლება სერიული იყოს მაშინ შემდგომ გადმოცემული როგორც ბაიტკოდათ, რომელიც შეიძლება იყოს შენახული როგორც ფაილი, შემდეგ თუ გსურთ ბაიტკოდის გაშვება უბრალოდ უნდა მიაწოდოთ ის luaU_undump, რომელიც გადააქცევს ბაიტკოდს ლუას შუამავალ ფორმატში (პროტოდ). პროტოს მიღების შემდეგ, შეგიძლიათ ჩასვათ ის ლუა ფუნქციაში (LClosure) შემდეგ დაგეგმეთ(Task Scheduler-ის გამოყენებით) რომ გაეშვას ის. ზემოთ ხსენებული შეგიძლიათ ააგოთ თქვენით და გაუშვათ ისე კოდი რობლოქსში მაგრამ, ჩვენ შეგვიძლია უბრალოდ ავაგოთ ბაიტკოდი და შემდგომ რობლოქსის ფუნქციის გამოყენებით გავაგზავნოთ ბაიტკოდი და გააქტიურეს კომპაილერში ჩვენი გაგზავნილი ინფორმაცია. სამწუხაროდ რადგანაც საშუალო დონის დოკუმენტია, არ განვიხილავ თუ როგორ შევძლოთ კოდის გაშვება ისე როგორც მე ვახსენე ამ გვერდის დასაწყისში.

გავითვალისწინოთ რომ რობლოქსის რევერსი განსხვავდება ბევრი სხვა თამაშისგან, რადგანაც ამ სიტუაციაში თქვენ გესაჭიროებათ LuaU-ს სტრუქტურის შესწავლა და კარგად გაანალიზება, როგორ ასერიალიზებს კოდს რობლოქსი. იმისდა მიუხედავად რომ ვეცდები ყველა ჩემი ცოდნა ჩავტიო ამ დოკუმენტში, ეს იმას არ ნიშნავს რომ ყველაფერი იქნება ნახსენები და თქვენ შეძლებთ სრულყოფილად რობლოქსის ჩეთის აგებას. ასევე გასათვალისწინებელია ჰაიპერიონის დაპირისპირებაც რომელიც როგორც ვახსენე წინაზე რთულია მაგრამ არა შეუძლებელი.

მოდი ზოგადი განათლება დავიწყოთ რომელიც გამოგვადგება ინტერნალზეც და ექსტერნალზეც, რადგანაც უკვე ავხსენი identity და რას წარმოადგენს ის, არ დავხარჯავ მაგაში დროს. მოდი განვიხილოთ მთავარი რამ რაშიც ინახება ყველაფერი რაც VM-ში ეშვება, სკრიპტები, ობიექტები და ნუ ასე შემდეგ, უფრო მარტივად რომ აგიხსნათ დიდი ალბათობით ერთხელ მაინც გინახიათ ან გამოგიყენებიათ DataModel, ან როგორც ცნობილია "game" რომელიც გამოიყენება სკრიპტებში როგორც game.workspace და ასე შემდეგ. DataModel-ის საპოვნელად ჩეთში ცოტა უცნაურადა საქმე, ექსტერნალზე ჩვენ ვიყენებთ ლოგების მეთოდს რომელიც სამწუხაროდ არ ვიცი ვისი შემუშავებულია მაგრამ დიდი მადლობა მას რადგანაც რობლოქსის ექსპლოიტინგის სცენაში ბევრი პიროვნება იყენებს, როცა ჩვენი ჩეთი გაეშვება ვამოწმებთ რობლოქსის ლოგებს რომელიც მოთავსებულია AppData-ში, შემდგომ ვეძებთ ყველაზე ახალი რომელიც იქნა შექმნილი და ვეძებთ სტრინგს "initialize view" და მის გვერძე იქნება მოცემული ადრესი, მაგრამ ამით არ გვიპოვია ჯერ DataModel-ი, ამით ჩვენ აღმოვაჩინეთ RenderView, ის არის ეგრეთ წოდებული სცენა, რომელიც წარმოგვიდგენს ეკრანზე ყველაფერს, კამერის პონტში. ადრესის აღმოჩენის შემდეგ ჩვენ უნდა ვიპოვოთ ყალბი DataModel რომელიც მიგვიყვანს ნამდვილ DataModel-თან, ოფსეტების აღმოჩენა თავისთავად შეგიძლიათ ReClass-ის გამოყენებით და შედეგი დაახლოებით შემგომ უნდა გამოიყურებოდეს:

(renderview + 0x118) + 0x198

სადაც 0x118 არის ყალბი დათამოდელის ოფსეტი და 0x198 არის ნამდვილი დათამოდელის ოფსეტი.

წინა გვერდზე განვიხილეთ თუ როგორ მოვიპოვოთ DataModel-ი იმისდა მიუხედავად რომ იგივეს გაკეთება შეგვიძლია ინტერნალ -ზე, უბრალოდ არ იქნება სწორი რადგანაც ყოველთვის ვერ გაამართლებს ეგ მეთოდი. სანამ ინტერნალზე მოპოვების განხილვას დავიწყებდეთ მაქამდე ვისაუბროთ ერთერთ სერვისზე სახელად TaskScheduler