The Rop Mate

Visually Assisting the Creation of ROPChain based Exploits

Pietro Borrello - Serena Ferracci

Abstract

ROPChain based exploits are increasingly present in the scenery of advanced attack techniques. Being able to test software against such types of attacks is fundamental to any RedTeam hacker. Automatic tool that try to create ROPChains exists and are well known, but they often fail with non usual binaries.

In this paper we propose `RopMate`, a visually aiding tool to help the human ROPChain builder to create its ROPChain. We strongly believe that there is the need of such a tool, since the existing state of the art, only propose text based approach with, less or more, informative lists of the gadgets found by the tool.

RopMate presents the builder with a clear interface of available gadgets, he is able to choose which of them to add to the building ROPChain, and, once selected, to filter the list accodring to wanted guarantees on preserved registers and accessed memory. The user is also able to find similar gadget in respect with the one choosen.

1 Introduction

Return-oriented programming (ROP) is an exploiting technique that allows an attacker to introduce arbitrary behaviour in a vulnerable program without actually injecting any code, but through a chain of redirections in the program memory itself. [1]

The attack scenario is based on a controlled stack frame, where the return address can be overwritten. This is an advanced version of the "return to libc attack" [2] where multiple pieces of code are called in sequence to provide the needed code semantics execution: the attack combines a large number of short instruction sequences (called *gadgets* from now on) that allow arbitrary computation. This is resilient to mitigation as non executable memory areas.

Each gadget is in the form of a couple of instruction followed by a return. This allows the attacker to place a sequence of gadget addresses on the stack (called *ROPChain*) from the return address on, that will be executed thanks to the semantics of the ret instruction.

2 The Problem

There exists automatic tools to find the suitable gadgets to create the final chain, but all of these tools present the attacker with a textual list of all the found gadgets. Then the attacker has to filter that list on the terminal with `grep` or sim-

1

ilar commands. This becomes frustrating and infeasible with the growing of the complexity of the needed chain.

This is an example of the most simple ROPChain to spawn a shell in the libc standard library:

```
rop = ''
rop += p64 (...) # pop rdi; ret;
rop += '/bin/sh\x00'
rop += p64 (...) # pop rsi; ret;
rop += p64 (WRITABLE_ADDRESS)
rop += p64 (...) # mov [rsi], rdi;
    ret;
rop += p64 (...) # pop rdi; ret;
rop += p64 (WRITABLE_ADDRESS)
rop += p64 (...) # pop rsi; ret;
rop += p64 (0x0)
rop += p64 (...) # pop rax; ret;
rop += p64 (...)
rop += p64 (...) # syscall; ret;
```

Where `p64()` transforms an address to the string representation of it in 64 bits, to be able to place it on the stack:

```
>>> p64 (0x1122334455667788)
        '\x88\x77\x66\x55\x44\
            x33\x22\x11'
```

The comment near each instruction represents the instructions in that gadget to be executed.

The problem is that it never happens that the gadget are so simple and clear. Mixed with the instruction the attacker needs, there can be register modifications and memory accesses that can ruin the generated semantics, so choosing the next gadget to use is a delicate task. This becomes a pain, as the size of the chain grows and the number of constraints to maintain in mind increases. Moreover the number of gadget available in a binary is huge (from 1.500 for a middle sized binary to more than 15.000 for a standard library) so naively searching for useful and correct gadgets between them is impossible.

Automatic tools that generate the chain exist but usually they fail and leave the attacker alone. So there is a lack of an effective tool to help developing the exploit.

3 Proposed Solution

We propose to apply Visual Analytics methods to the problem. The aim is to develop an interface that will help the construction of the chain for the exploit. The binary that will be the source of the gadgets is analyzed by the backend server that produces a list of semantically meaningful gadgets. This means that only gadgets that have a clear effect are maintained.

The interface will contain the list of all meaningful gadget, divided by class, and by effects on parameters, in which the user can make queries on the desired features and immediately visualize the gadgets that satisfy them. Queries may involve searching for gadgets that have a particular semantical meaning, but for example, don't modify some registers that have yet been set or access the memory only by some controlled registers, not to crash the program.

Clicking on a gadget will show its features (that will be encoded with additional visual hints). Each class of gadget has a set of attributes listed in the table 1.

Each gadget has a specific type, between:

```
LoadConst, SetZero, IncReg, CopyReg,
BinOp, ReadMem, WriteMem,
```

2

Table 1: Gadget attributes

Gadget	
type	the class of the gadget
params	the parameters of the gadget
hex	the x86 bytes instructions in the gadgets
disasm	the x86 disassembled instructions
address	the address where the gadget starts in memory
address_end	the address where the gadget ends in memory
modified_regs	the registers modified by the execution of the gadget
mem	the registers that the gadget uses to access memory
stack_fix	the delta in the stack after the gadget execution
retn	how many words the ret instruction at the end pops out

`OpEsp, Lahf, ReadMemOp, WriteMemOp`

The gadgets in the main view are hierarchically ordered by class, and then by parameters. The size of each node in the main view encodes the number of gadgets of that class, to quickly give an hint on the overall view.

Once clicked on a gadget the user can add it to the chain view in which is displayed the ROPChain that is building. This will trigger the recomputation for the registers the user has successfully set with the chain, and by default the view of the whole gadgets will display only the gadgets safe with respect to modified registers, that the user can choose.

Inside a gadget class the user can additionally trigger a cluster view, in which similar gadgets are displayed near each other to be able to quickly discriminate the possibilities from which it has to chose.

References

[1] Hovav Shacham, The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86), CCS, 2007.

[2] Nergal, The advanced return-into-lib(c) exploits (PaX case study), Phrack Magazine 58(4), Dec. 2001.