# Automated string de-gobfuscation

kryptoslogic.com/blog/2020/12/automated-string-de-gobfuscation/

🗧 💿 🕲	c9b30d5(	2e211	031089607ce699520188154e3d7988a9db80112a2e4378bf41.bndb — Binary Ninja	a second a s
Symbola	88 1		5c9b30d, 41 bridb (ELF Linear)	
type_eq_line_ParsError time_fileStaterror_Error type_eq_l_time_zone type_eq_l_time_zone type_eq_l_time_zone type_eq_l_time_zonetrans fpkkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframejfg][foj_accdmbdpffpenfofhded_earfeor fpkfenframe]foff][foj_accdmbdffpenfofhded_earfeor fpkfenframe]foff][foj_accdmbdffpenfofhded_earfeor fpkfenframe]foff][foj_accdmbdffpenfofhded_earfeor fpkfenframe]foff][foj_accdmbdff	eer leer eer eer eer eer eer	0045 0055 0045 0045 0045 0045 0045 0045	<pre>cff (f (+data_05cb30 i= 0)     rei_2 = deta_04b2a0     rei_2 = rei_4  d004     rei_4 = rei_4  d005     rei_4 = rei_4  d00     rei_4 = rei_4  d004     rei_4 = rei_4  d005     rei_4 = rei_4  d00     rei_4 = rei_4  d004     rei_4 = rei_4  d00      rei_4  d00      rei_4  d00      rei</pre>	_glob_func3(rd1_2, rs1_3, _glob_func4(rd1_2, rs1_4,_
	Pyl	thom Cor	auto anti anti anti anti anti anti anti anti	
	(1) (1) (1) (1) (1) (1)	, Loadeo Loadeo	python3 plugʻin "get_proc_eddress_dlayn" nativa plugʻin pythonplugʻin	¢2) 51
tee49d256 jmp 0x49d88d		Loaded Detect Found Added ELF pa Analys	UI plugin triage ed Little Endian endian ELF string table of size Exa5 at offset Ex3017a8 Ufruz-xRé_54 entry point at 0x46sba9 rsing took 8.733 seconds is update took 8.397 seconds Selection: 0x49d00d 10 0x49d012 (0x5 bytes) E	LF + Linear + Options +

Authored by: Jamie Hankins on Wednesday, December 2, 2020

Last week the Network Security Research Lab at 360 released a blog post on an <u>obfuscated</u> <u>backdoor written in Go</u> named Blackrota. They claim that the Blackrota backdoor is available for both x86/x86-64 architectures which is no surprise given how capable <u>Golang's cross</u> <u>compilation</u> is.

For the last 4 years we have been using Golang for our internal services, and I can definitely see the allure that Golang has for malware authors:

- Statically compiled binaries by default
- Cross compilation is often as simple as setting two environment variables
- Strong package ecosystem allowing you to pull in code that you need from other sources
- No runtime dependencies
- Esoteric runtime with a non-standard calling convention breaks most decompilation tools forcing reverse engineers to read assembly

Blackrota uses <u>gobfuscate</u> to obfuscate their source code before it gets compiled by the Go toolchain. Gobfuscate presents a number of challenges to reverse engineers but the one I'll be focusing on today is string obfuscation.

## **String Obfuscation**

Malware has been using XOR encoded strings for years now, but Blackrota takes this a step further<sup>1</sup>. It generates a random XOR key per string and wraps the string in a function that XORs the string at runtime to return the correct one.

gobfuscate runs before the compilation process to produce an obfuscated version of your code which is then compiled by the Go compiler:

```
Before:
package main
import "fmt"
func main() {
        fmt.Println("Hello world!")
}
After:
package main
import "fmt"
func main() {
        fmt.Println((func() string {
                mask := []byte("\x21\x0f\xc7\xbb\x81\x86\x39\xac\x48\xa4\xc6\xaf")
                maskedStr :=
[]byte("\x69\x6a\xab\xd7\xee\xa6\x4e\xc3\x3a\xc8\xa2\x8e")
                res := make([]byte, 12)
                for i, m := range mask {
                        res[i] = m ^ maskedStr[i]
                }
                return string(res)
        }()))
}
```

You can use a tool like <u>GCHQ's CyberChef</u> to verify that the result of those 2 byte arrays XORed together is: <u>Hello world!</u>.

While you *could* go through the effort of manually XORing each string in a binary you're reversing, it'll get tedious very quickly.

### How do I solve it at scale?

So I know that Golang has a very capable cross compiler and that I want to deal with these XORed strings across different architectures, but how?

Binary Ninja (aka Binja) has a <u>very powerful intermediate language (IL)</u>, which allows us to operate on a representation of a function regardless of the system architecture (assuming there is an architecture loader for it)<sup>2</sup>.

When deciding on an approach here I specifically wanted to use Binary Ninja's IL as a learning exercise. The solution could certainly be implemented using any number of alternative tools (e.g. <u>Unicorn</u>), however Binja has the benefit of being very easy to use on any of its supported platforms. Another benefit, of course, is that the IL eliminates the need to understand the target architecture at all, which (depending on implementation) may not be the case using something like Unicorn.

So the solution will need a way of identifying what functions we want to extract the strings from (Candidate Identification) and a way to extract the correct string from Binary Ninja's low level IL.

### **Candidate Identification**

Luckily for us, the Go compiler doesn't inline this function call so it's seen as an entirely new function:

Assuming the approach I take can avoid the Go runtime functions:

\_runtime.morestack\_noctxt and \_runtime.slicebytetostring it should be pretty easy to emulate.

The solution will need to find functions in the binary which call \_\_runtime.slicebytetostring and make use of an xor instruction.

In my first attempt you'll see that I'm not using the IL for this, but that's trivially solved in the final solution.

## **String Extraction - Attempt One**

My first attempt at writing this didn't go very well… Binary Ninja implements multiple different ILs at varying levels of readability based on how much lifting, transformation and control flow recovery takes place. You can see examples of <u>LLIL (low level intermediate language)</u>, <u>MLIL (medium level intermediate language)</u> and <u>HLIL (high level intermediate language)</u> ILs in the corresponding links.

So I made the assumption that HLIL would be the best choice because there was less text. I also didn't really understand that an IL operation could consist of multiple different IL operations:



#### Example of a XOR/assign operation in HLIL

You can see my <u>first attempt here</u>. I wish I had read <u>Josh Watson's blog post</u> on Binja's IL before I started working on it as it details some of the non obvious details

I managed to get some results from this approach but it was not consistent and broke very easily.

## **String Extraction - Attempt Two**

I started chatting to <u>Jordan Wiens</u> one of the founders of <u>Vector35</u>, the company who makes Binary Ninja – and he suggested that I should rewrite it using LLIL and a full blown emulator rather than a state machine.

At first this was daunting, but I started to play around with it and thanks to some incredibly helpful plugins which I'll detail later, as well as a <u>project from Josh Watson</u> which had the foundations of an LLIL emulator, I was able to start making headway.

Using a <u>helpful snippet from Jordan</u> I was able to work out what LLIL operations I needed to implement and in total there were around 28 of them:

IL operation	Description
LLIL_TAILCALL	Call another function without writing a return address to the stack
LLIL_CALL	Call another function and push a return address to the stack

IL operation	Description
LLIL_RET	Pop a return address from the stack and jump to it
LLIL_PUSH	Push a value onto the stack
LLIL_POP	Pop a value from the stack
LLIL_XOR	XOR 2 values
LLIL_ZX	Zero Extends
LLIL_GOTO	Set the instruction pointer
LLIL_STORE	Write some data to memory
LLIL_READ	Read some data from memory
LLIL_SET_FLAG	Sets a flag
LLIL_FLAG	Reads a flag
LLIL_CMP_NE	Is not equal
LLIL_CMP_E	Is equal
LLIL_CMP_SLE	Signed less than or equal
LLIL_CMP_SGT	Signed greater than
LLIL_CMP_SGE	Signed greater than or equal
LLIL_CMP_UGE	Unsigned greater than or equal
LLIL_CMP_UGT	Unsigned greater than
LLIL_CMP_ULT	Unsigned less than
LLIL_CMP_SLT	Signed less than comparison
LLIL_CMP_ULE	Unsigned less than or equal
LLIL_IF	Check a conditional and set the instruction pointer based on the outcome
LLIL_SET_REG	Set a register value
LLIL_CONST	Get a constant value
LLIL_CONST_PTR	Get a constant value that happens to be a pointer
LLIL_REG	Read a register value

IL operation	Description	
LLIL_SUB	Subtract two values	
LLIL_ADD	Add two values	

Alone none of these seem too complex right? But together they allow us to fully implement what is required to decode the XOR obfuscation in less than 500 lines of code. There's a huge amount of heavy lifting that Binary Ninja does under the hood that helps us keep the code as simple as possible, and I'm sure I'm missing some tricks.

Once I had something that *kinda* worked Jordan jumped in and converted it into a functional Binary Ninja plugin and cleaned up some of the code smell that occurs when you're hacking away on a problem.

While developing the plugin I came across some incredibly useful Binary Ninja plugins which I feel need a mention:

#### SENinja

<u>SENinja</u> is a symbolic execution engine for Binja, built using the <u>Z3 SMT solver</u>. It implements a LLIL emulator that builds and manipulates Z3 formulae. Although the intentions of SENinja are much more complex then what I was using it for, it was really useful to have something I could compare with (I didn't want to turn on a Linux VM and use a debugger and deal with possible anti debugging tricks).

	5c9b30d502e2f103f089607ce699520f88154e3d7988a9db801f2a2a	4378bf41.bndb — Binary Ninja
Symbols 🛛 🕅	Sc9b30d41.bndb (ELF Linear) 💠 Instruction Graph (0x6aebf2) 💥 Instruction Graph (0x6aebf2) 🖄	Feature Map 830
sub_6ab84a 🗠	11.0.005aphdf [rep + $0x^2$ (var a)] d = 0	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	12 $\theta$ 606 acbeb 7 [rsp + 0.3a [var_er], 0 = 0	
tpkktentnamejtgjltoj_accdnbdpttpontoth	13 # 066aebee ex = 0	
fpkkfenfnamejfgjlfoj_accdnbdpffponfofb	14 @ 006aebf0 goto 15 @ 0x6aec09	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofb		
fokkfenfnameifgilfoi accdmbdoffponfofh:	15 @ 006aec09 if (rax s< 6) then 16 @ 0x6aebf2 else 22 @ 0x6aec0b	
fpkkfenfnameifgilfoj accdmbdpffponfofh		
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	16 @ 006aebf2 ecx = zx.d([rsp + rax + 0x34].b)	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	17 @ 006aebf7 edx = zx.d([rsp + rax + 0x2e].b)	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh		
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	19 @ GOGAUDIE [FSP + FAX + OXSA].D - CC	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh		
fpkkfenfnamejfgjlfoj_accanbaptfponfofn		
fokkfenfnamejfgjlfoj_accdmbdpffponfofb	22 @ 006aec0b [rsp {var_48}].q = 0	
fpkkfenfnameifgilfoi accdmbdpffponfofh	23 @ 006aec13 rax = rsp + 0x3a {var_e}	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	24 @ 006aec18 [rsp + 8 {var_40}].q = rax {var_e}	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	25 @ 006aecid [rsp + 0x10 {var_38}].q = 6	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	26 @ @@Gaec26 call(runtime_slicebytetostring)	
fpkkfenfnamejfgjlfoj_accdnbdpffpomfofh	27 @ 000aec20 rax = [rsp + 0x18 {var_30}].q	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	20 $\oplus$ 000 dec 30 f LA = [15] $\neq$ 0 Az0 [Val_26]].4	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	$30 \oplus 000 = 0.35$ [rsp + 0.55 [arg 10], $q = 10$	
fokkfenfnamejfgjlfoj_accdmbdoffnonfofb	31 @ 006aec3f rbp = [rsp + 0x40 (var 8)].q	
fokkfenfnameifgilfoi accdmbdoffoomfofb	32 @ 006aec44 rsp = rsp + 0x48	
fpkkfenfnamejfgjlfoj_accdmbdpffpomfofh	33 @ 006aec48 <return> jump(pop)</return>	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh		
fpkkfenfnamejfgjlfoj_accdmbdpffpomfofh	006aec53 cc	SENinja Memory
fpkkfenfnamejfgjlfoj_accdnbdpffponfofh	int64 + fnkkfanfnamaifailfai accdebdaffaanfafbded nabiabfablblklannali Ddfcadiibliaabechala func64	Monitor Memory Back
tpkktentnamejtgjltoj_accdmbdpttpomtoth	int64 t* are1, int16 t* are2, uint64 t are3, int64 t are4, char* are5, int64 t are6,	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofb	void* arg7 @ rbp)	0 1 2 3 4 5 6 7 8 9 A 8 C D E F
fokkfenfnameifgilfoi accdmbdoffponfofh		7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	int64_t var_50 {Frame offset -50}	
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	int64_t var_48 {Frame offset -48}	77677777777777777777777777777777777777
fpkkfenfnamejfgjlfoj_accdmbdpffpomfofh	int64_t var_46 (Frame offset -46)	sel: [#r33, #r39] ]est #r7
fpkkfenfnamejfgjlfoj_accdmbdpffponfofh	into4_t var_30 (Frame offset -30)	
tpkktenfnamejfgjltoj_accdmbdpffponfofh		SENinja Buffers
fokkfoofoamoifgilfoi accdmbdoffoonfofb		
fpkkfenfnameifgilfoi accdmbdoffponfofk	Log	New Buffer
4	<segment: 0x400000-0x5blaaa,="" r-x=""> 0x400000 -&gt; 0x2blaaa</segment:>	Address Name Size Constraints
Tags Symbole	<pre><segment: 0x052000-0x877370,="" r=""> 0x052000 -&gt; 0x45370</segment:></pre>	
	loading finished!	
Cross References 🖾 🖄	Analysis update took 0.003 seconds	
Filter (1)	Analysis update took 0.003 seconds	
✓ Code References     {1}	Analysis update took 0.003 seconds	SENinja Registers
<ul> <li>fpkkfenfnamejfgjlfoj_accdmbdpffpor {1}</li> </ul>	Analysis update took 0.003 seconds	Register Value
B06aec89 jl 0x6aebf2	Analysis update took 0.003 seconds	chy chy init
	Analysis update took 0.003 seconds	
	Analysis update took 0.003 seconds	rcx 9x868666000000001
	Analysis update took 0.003 seconds	
	Analysis update took 0.004 seconds	Pax execcededebebbbbas
	Analysis update took 0.003 seconds	rsi rsi_init
	Analysis update took 0.005 seconds	
	Analysis update took 0.003 seconds	rdi rdi_init
×		
		Selection: 0x6aebf2 to 0x6aebf7 (0x5 bytes) ELF • Linear • Options

#### **BNIL Instruction Graph**

<u>BNIL Instruction Graph</u> allows you to click on a line of IL and generate a graph of the IL operations that make up that line.



LLIL



#### HLIL

The difference in complexity between a line of LLIL and HLIL is surprising. However it makes sense once you understand that HLIL exists to allow higher level control flows to be recovered with the intention of producing a source code representation, typically in an emulator though you don't need this kind of high level control flow data.

Another powerful feature of BNIL Instruction Graph is the ability to generate specific IL matching templates that are convenient starting points for building IL code:

```
def match_LowLevelIL_10a6df6_0(insn):
    \# ecx = ecx \wedge edx
    if insn.operation != LowLevelILOperation.LLIL_SET_REG:
        return False
    if insn.dest.name != 'ecx':
        return False
    # ecx ^ edx
    if insn.src.operation != LowLevelILOperation.LLIL_XOR:
        return False
    # ecx
    if insn.src.left.operation != LowLevelILOperation.LLIL_REG:
        return False
    if insn.src.left.src.name != 'ecx':
        return False
    # edx
    if insn.src.right.operation != LowLevelILOperation.LLIL_REG:
        return False
    if insn.src.right.src.name != 'edx':
        return False
    return True
```

## Can haz please?

As the usage of Golang by malware authors increases, so will their understanding and capabilities within the Go ecosystem. Right now gobfuscate is the main obfuscator used by actors using Golang, but this won't always be true. As a <u>Hacker News comment puts it</u>: this is fairly weak. Gobfuscator doesn't implement control flow obfuscation, the runtime functions aren't obfuscated and at most gobfuscate serves mostly as an annoyance and a thinly veiled layer of obscurity.

( <b>• • •</b> •	5c9b30	d502e2f1	03f089607ce699520f88154e3d7988a9db801f2a2a4378bf41.bndb — Binary Ninja	
Symbols			5c9b30d41.bndb (ELF Linear) 💠	
symous type_eq_time_ParseError type_eq_lime_ParseError type_eq_lime_zone type_eq_l_time_zone type_eq_l_time_zoneTrans fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam fpkkfenfnamejfg]Ifoj_accdmbdpffpomfofhdmd_eam trypto_subtle_ConstantTimeCopy math_log2 math_log2 math_init math_log2 strconv_baseError strconv_baseError strconv_baseError strconv_baseInot	feor feor feor feor feor feor feor feor	•         •           •         •	Scybol_41.bndb (EF Linear) ::         92cff9 if (*data_96cb10 != 0)         9424a (rd_2 = data_93b250)         runtime_gcWriteBarrier(data_93b250, rsi_3, var_28, var_30, r8_3, var_38)         94056 else         94066 viste         94056 viste         94056 viste         94057 viste         94058 viste         94059 viste         94050 viste	•
Cross References + Filter (2) - Data References	88 {1}			
<ul> <li>B049ca40 fpkkfenfnamejfgjlfoj_accdmbdpff</li> <li>Code References</li> <li>fpkkfenfnamejfgjlfoj_accdmbdpffpomfofhdmd_ea</li> <li>B049d256 jmp</li> <li>0x49d00d</li> </ul>	pomfof {1} u {1}	Log Loader Loader Detec Found Added ELF p Analy	d python3 plugin 'get_proc_address_dlsym' d native plugin pythonplugin d UI plugin triage ted Little Endian endian ELF string table of size 0x55 at 0ffset 0x3a17a0 linux=x86_64 entry point at 0x46ab80 arsing took 0.733 seconds	
<b>†</b>			Selection: 0x49d00d to 0x49d012 (0x5 bytes) ELF + Linear + Options +	

#### Solver at work!

You can download the Binary Ninja plugin on the <u>plugin manager</u> or download the code from the <u>GitHub repository linked here</u>. Additionally you can find the Blackrota <u>32-bit</u> and <u>64-bit</u> Binary Ninja databases to look at on Binary Ninja cloud.

#### Thanks:

- A massive thanks to the people in the Binary Ninja Slack who answered some of my questions while I was getting started.
- Josh Watson, for his <u>permissibly licensed emulator</u> that I was able to use to get an idea of what was required to achieve my goal.
- Jordan Wiens, for providing assistance during development, cleaning up messy code, filing a couple of Binja bug reports for me and generally answering questions about Binary Ninja.