GoLang Reversing Primer
How to not crap the bed when you encounter Go Malware

Jimmy Wylie, Dragos Inc.
@mayahustle

Who am I?

- Jimmy Wylie
  - Principal Reverse Engineer @ Dragos, Inc.
  - Hunting and analyzing threats to critical infrastructure
  - Implementing malware analysis automation

- Before:
  - Instructor and Course Developer @ Focal Point Academy
  - Malware Analyst for various DoD Contractors (Fortego)
Why Go?

01 EKANS Ransomware

02 Exaramel Linux

03 General Curiosity

04 Work made me

What’s Go?

- Statically typed, Compiled language designed by Robert Griesmer, Rob Pike, and Ken Thompson
  - Memory safe, structured typing, garbage collection
  - Designed with networking, concurrency and multi-processing in-mind (goroutines, channels)
  - Arrays (by value), Slices (“by ref”), immutable Strings
- Basically, modern PL designers looked at C++ and said, “Well this isn’t right”.

My desire to nap

But I already did something today

Wired

Mysterious New Ransomware Targets Industrial Control Systems

EKANS appears to be the work of cybercriminals, rather than nation-state hackers—a worrying development, if so.

New TeleBots backdoor: First evidence linking Industroyer to NotPetya

Analysis of the Linux/Exaramel backdoor

The backdoor is written in the Go programming language and compiled as a 64-bit ELF binary. Attackers can deploy the backdoor in a chosen directory under any name.

ESET
Sample Function (string.go)

```go
// The constant is known to the compiler.
// There is no fundamental theory behind this number.
const tmpStringBufSize = 32

type tmpBuf [tmpStringBufSize]byte

func stringtoslicebyte(buf *tmpBuf, s string) []byte {
    var b []byte
    if buf != nil && len(s) <= len(buf) {
        *buf = tmpBuf{
            b = buf[:len(s)]
        }
    } else {
        b = rawbyteslice(len(s))
        copy(b, s)
    }
    return b
}
```

Type definitions:
type typeName <shape>

Return type

Arguments with type designation

First Look

Go Build ID in Strings

Large: Runtime included

Src references hint at obfuscation
First Look – IDA Initial Analysis

No function Recognition for 5000- functions

Strings are in a singular blob. References are not always present

First Look - Disassembly

Top of the stack allocated once and top slots re-used for functions.

HexRays decompilation fails often.
First Look - Obfuscation

XORs two nonsense bytestrings

100s of these but with different bytestring references.

Priorities

1. Get Strings and References labeled
2. De-obfuscate Strings
3. Get Runtime functions labeled
4. Understand complex types passed to runtime functions. Focus shifts from Windows API
5. Then... Finally... Actual work (disassembly analysis, yara signatures, etc.)
redress

- Command-line tool for analyzing stripped Go binaries
- Prints compiler, type, package information, and more
- Part of the Go Reverse Engineering Toolkit
- Includes Go, Python, and C libs for writing tools for analyzing Go
- https://go-re.tk/redress/
Redress - Compiler information

- Returns compiler version + time when release tag was created
- Important for intel reasons, but compiler version lets you know which version of Go source to download
Redress

- Great for a quick overview of capabilities and identifying developer code vs std lib
- But, there’s no easy way to transfer that information to IDA (which is what I care about).
- Developer did release a plugin for integrating this information for radare (which I don’t hate myself enough to use.. j/k don’t @ me).
- So how do we get this info in IDA?

IDAGolangHelper

- Plugin for transferring function and type information to IDA Pro
IDAGolangHelper

while ((unsigned int)&retaddr <= *((DWORD *)&NtCurrentTeb().NtTib.ArbitraryUserPointer + 8))
    runtime_morestack_noctxt();
runtime_stringtoslicebyte(v13, byte_61741D, 15);
v14 = v6;
runtime_stringtoslicebyte(v12, &byte_6170F3, 15);
    *((DWORD *)&v9) = 0;
    *((DWORD *)&v9[3]) = 0;
v10 = 0;
v11 = 0;
v9 = v14;
for (i = 0; (int)i < (int)v7; ++i)
{
    if (i >= v7 || i >= 0xF)
        runtime_panicindex(v3, v4, v5);
    v9[i] = (*((BYTE *)&v6 + i) ^ (*((BYTE *)&v0 + i) + 42));
}
runtime_slicebyteToString(0, v9, 15, 15);
return v8;

IDAGolangHelper - add structs

<table>
<thead>
<tr>
<th>Name</th>
<th>Ins/Del</th>
<th>Create/Dele Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct</td>
<td>N</td>
<td>rename structure or structure member</td>
</tr>
<tr>
<td></td>
<td>D/A</td>
<td>create structure member (data/ascii)</td>
</tr>
<tr>
<td></td>
<td>U</td>
<td>delete structure member</td>
</tr>
</tbody>
</table>

String struct:

```c
struct (sizeof=8x8, mapped=F000)
```

Slice struct:

```c
struct (sizeof=8xC, mapped=F000)
```
IDAGolangHelper - Notes

- Appears to only run on Python2. Need to update print statements for Python3 versions of IDA
  
  Or switch IDA to Python2

- Also relies on old IDC style functions in IDA python

  ```python
  import idc
  import idc_bc695 as idc
  
  Could also modify idapython.cfg
  (AUTOIMPORT_COMPAT_IDA695)
  ```

IDAGolangHelper - Results

- `main`: 1640
- `baaipecheakgkaiodnme`: 404
- `ljajcmidoepkeidljcnm`: 296
- `std`: 3294
- Total: 5634

  Names almost all the functions, but there are still a lot of “unknowns”.

  Still 2000+ leftover
  Many of these appear to be decoders.
Gobfuscate

- Open source obfuscator for
  - Package Names (with restrictions).
  - Global Names (vars, consts, funcs, types)
  - Struct Methods (with restrictions)
  - Strings...

- Mostly it hashes the above and replaces name with hash. Packages (and containing names or methods) are not obfuscated if they use CGO or assembly.

Gobfuscate - Strings

Strings are replaced with a function.

From Gobfuscate docs

Malware decoder adds a shift

Name and String obfuscation seem to follow Gobfuscate patterns
Solution – IDAPython

- Modify script by @w3ndige that recognizes and decodes strings

```python
all = []
func_count = 0
for m in all:
    m.start = m.start() /
    m.end = m.end()
    m.data = binascii.unhexlify(group())
    mask_rva = struct.unpack_from('>{i}', m.data[2])
    length = struct.unpack_from('>{i}', m.data[4]())
    mask_str_rva = struct.unpack_from('>{i}', m.data[17]()) - base
    d1 = bytesarray(m.data(mask_rva, mask_rva() length))
    d2 = bytesarray(m.data(mask_str_rva, mask_str_rva() length))
    out = []
    for i in range(len(d1)):
        out.append([d1(i), d2(i)]) % 2
    out_string = ''.join(chr(ord) for x in out)
```

Get addresses

Decode String

Solution – IDAPython

- Create comments by instruction and data locations

```python
# create the array of bytes correctly with length for mask and masked_str
idaapi.del_items(mask_ea, 0, length, None)
idaapi.create_data(mask_ea, idaapi.FF_BYTE, length, 0)
idaapi.del_items(masked_str_ea, 0, length, None)
idaapi.create_data(masked_str_ea, idaapi.FF_BYTE, length, 0)
# set comment with the string it's associated with
idaapi.set_cmt(mask_ea, "Mask for " + out_string, False)
idaapi.set_cmt(masked_str_ea, "Masked String for " + out_string, False)
# set comment on instruction with decoded name
idaapi.set_cmt(load_mask_ea, "Mask for " + out_string, False)
idaapi.set_cmt(load_masked_str_ea, "Masked String for " + out_string, False)
```
Solution – IDAPython

- Rename function to “decode_<str>”

```python
i = idaapi.get_func(load_mask_ea)
if i:
    func_count = func_count + 1
    # set name of function and force IDA to do replacement of bad chars
    old_name = idaapi.get_func_name(i.start_ea)
    idaapi.set_name(i.start_ea, out_string, idaapi.SN_NOCHECK|idaapi.SN_FORCE)
    # get the legal name back
    f_name = idaapi.get_func_name(i.start_ea)
    # set function comment and name function with prefix
    f_com = "Get Decoded: " + out_string + "\n Old Name: " + old_name
    idaapi.set_name(i.start_ea, "decode_" + f_name, idaapi.SN_NOCHECK|idaapi.SN_FORCE)
    # set comment on function with decoded name
    idaapi.set_func_cmt(f, f_com, True)
    # rename data items with appropriate name
    idaapi.set_name(mask_ea, "mask_" + f_name, idaapi.SN_NOCHECK|idaapi.SN_FORCE)
    idaapi.set_name(masked_str_ea, "masked_" + f_name, idaapi.SN_NOCHECK|idaapi.SN_FORCE)
    # now store the name and the function, so that we can count xrefs
    all.append((out_string, f, load_mask_ea, mask_ea))
```

Results

- Ransom message and other strings

```plaintext
<table>
<thead>
<tr>
<th>what happened to your files?</th>
</tr>
</thead>
<tbody>
<tr>
<td>we breached your corporate network and encrypted the data on your computers. The encrypted data includes documents, databases, photos and more - all were encrypted using a military grade encryption algorithms (AES-256 and RSA-2048). You cannot access those files right now. But don’t worry! You can still get those files back and be up and running again in no time.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How to contact us to get your files back?</th>
</tr>
</thead>
<tbody>
<tr>
<td>The only way to restore your files is by purchasing a decryption tool loaded with a private key we created specifically for your network. Once run on an affected computer, the tool will decrypt all encrypted files – and you can resume day-to-day operations, preferably with better cyber security in mind. If you are interested in purchasing the decryption tool, contact us at xx</td>
</tr>
</tbody>
</table>
```
Results – Distribution of Funcs

decode: 1611
baaipecheakgkaiodnme: 399
ljajcmidoepkeidljcnm: 296
main: 34
std: 3285

Main module is now much easier to stomach.
Instead of 5000 functions, now only 729 unknown. (Not great, but better).

Analyzing a Function Call

Go uses a stack-based calling convention. Both arguments and return values are passed on the stack.

![](image)

- Initial stack growth for calling function
- Arguments
- Call net.LookupIP from Go libs
- Probably return values
LookupIP Definition

// An IP is a single IP address, a slice of bytes.
// Functions in this package accept either 4-byte (IPv4)
// or 16-byte (IPv6) slices as input.
//
// Need to understand:
// string type
// []IP – Slice
// Error type

// LookupIP looks up host using the local resolver.
// It returns a slice of that host’s IPv4 and IPv6 addresses.
func LookupIP(host string) ([]IP, error) {
    if err != nil {
        return nil, err
    }
    ips := make([]IP, len(address))
    for i, ia := range address {
        ips[i] = ia.IP
    }
    return ips, nil
}

Prototype Conversion Go to C

From Go:
func LookupIP(host string) ([]IP, error)

To C (IDA prototype):
int net_LookupIP(int, int, int, int, int, int, int)
Prototype Conversion Go to C

From Go:
func LookupIP(host string) ([]IP, error)

To C (IDA prototype):
int net_LookupIP(char* pStr, int strlen, int, int, int, int, int)

From GoLangHelper structs

Prototype Conversion Go to C

From Go:
func LookupIP(host string) ([]IP, error)

To C (IDA prototype):
int net_LookupIP(char* pStr, int strlen, char* sliceData, int sliceLen, int sliceCap, int, int)

From GoLangHelper structs
Prototype Conversion Go to C

From Go:
func LookupIP(host string) ([]IP, error)

To C (IDA prototype):

```c
int net_LookupIP(char* pStr, int strlen,
    char* sliceData, int sliceLen, int sliceCap,
    char* itab, char* ptr)
```

From GoDoc.org

```c
error_union struc ; (sizeof=0x8)
err
.err 
.error_union ends |
```

ESP
<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>+4C host.pStr</td>
</tr>
<tr>
<td>+48 host.strlen</td>
</tr>
<tr>
<td>+44 IP.sliceData</td>
</tr>
<tr>
<td>+40 IP.sliceLen</td>
</tr>
<tr>
<td>+3C IP.sliceCap</td>
</tr>
<tr>
<td>+38 Error.itab</td>
</tr>
<tr>
<td>+34 Error.ptr</td>
</tr>
</tbody>
</table>
Prototype Conversion Go to C

```c
int net_LookupIP(char* pStr, int strlen, char* sliceData, int sliceLen, int sliceCap, char* itab, char* ptr)
```

Labeling Fail

Seems like a good idea..

```
sub esp, 4Ch
lea eax, hostname
mov [esp+4Ch+var_4C], eax
mov [esp+4Ch+var_48], 11h
call net_LookupIP
mov eax, [esp+4Ch+var_40]
mov ecx, [esp+4Ch+var_38]
mov edx, [esp+4Ch+var_44]
test ecx, ecx
jnz loc_54589B
```

A few lines later 😃

```
mov [esp+4Ch+var_28], ecx
mov [esp+4Ch+var_20], bl
mov [esp+4Ch+var_4], edx ; IP_slicedata
eax, [edx+8]
mov ecx, [edx+4]
moveb [esp+4Ch+host_pStr], ebx
mov [esp+4Ch+host_strlen], ecx
mov [esp+4Ch+IP_sliceData], eax
call net_IP_string
mov eax, [esp+4Ch+var_3C]
mov ecx, [esp+4Ch+IP_sliceLen]
cmp eax, 9
jz short loc_5457F5
```
In Practice – generic argspace

Label top of stack as arg/ret space

Use comments to label types

So What’s Left?

- Identifying those randomly named modules
- Essentially, it’s the “statically compiled lib” problem.
  1. Examine evidence in modules
  2. Look for open source modules that seem to match capabilities
  3. Compile using same compiler version
  4. Diff programs using Diaphora and port symbol information
- Worst case: have to analyze the whole module by hand

One of the modules might be a Go WMI module
https://github.com/StackExchange/wmi
So What’s Needed?

- Current tools are good, but could be improved
  GoLangHelper could use some updating (port to latest IDApython API and python3)
  Possibly, an IDA plugin for GoRE may be easier to support.
- Fix for de-compilation error (“call analysis failed”)
  Probably due to Go’s weird calling convention causing stack analysis errors.
  Noticed that sometimes functions that previously decompiled will fail to decompile after GoLangHelper runs.
- Need tool for propagating type information for function calls.
  Unclear, how to do that as IDA doesn’t support stack variable re-use well.
  Might be able to with a decompiler plugin if we can fix aforementioned error

Takeaways

- Current Go Tools are great for giving the “lay of the land”
  Labeling functions, parsing strings, and extracting type information
- However, more work needs to be done to support the line by line analysis work (argument labeling and type propagation)
- Stack based calling convention is annoying, but like most of RE, patience and meticulousness makes it achievable
- Go compiler source and GoDoc are invaluable resources
- Likely that Go becomes more common in malware, so it’s best to start figuring out these problems now.
Links and Resources

- https://www.wired.com/story/ekans-ransomware-industrial-control-system/
- https://golang.org/doc/effective_go.html#introduction
- https://golang.org/ref/spec#Introduction
- https://golang.org/dl/
- https://go-re.tk/redress/
- https://lekstu.ga/posts/pclntab-function-recovery/
- https://github.com/sibears/IDAGolangHelper
- https://github.com/unixpickle/gobfuscate
- https://gist.github.com/W3ndige/c80e7cce80ff12e01c37eb98f7dc70db
- https://twitter.com/w3ndige/status/1258321900788428800?s=20
- https://drknz.net/go-calling-convention-x86-64.html
- https://godoc.org/builtin#error
- https://blog.osiris.cyber.nyu.edu/2019/12/19/go-deepdive/

Questions?

Jimmy Wylie
Twitter: @mayahustle
jimmy@dragos.com