virus 3 / anti-virus 1

last time

cavities to place code in

appending code with 'real' executable formats

replacing returns, calls, etc.

dynamic linking
- extra indirection for library funtions
- information about what to load in headers
- ELF interpreter — dynamic linker in other file

adding linker stubs

program.s

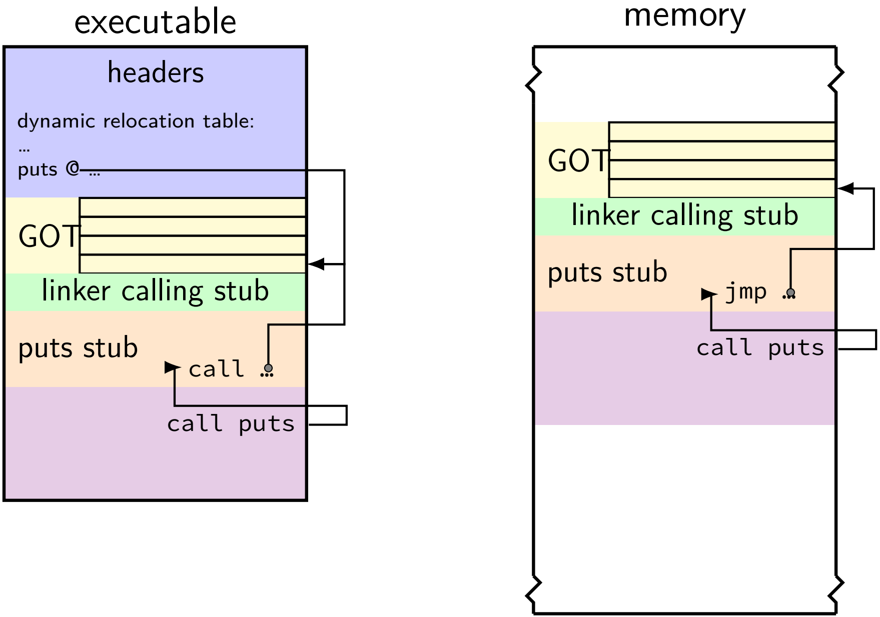
```
main:
    ...
    call puts
    ...
```

program.exe

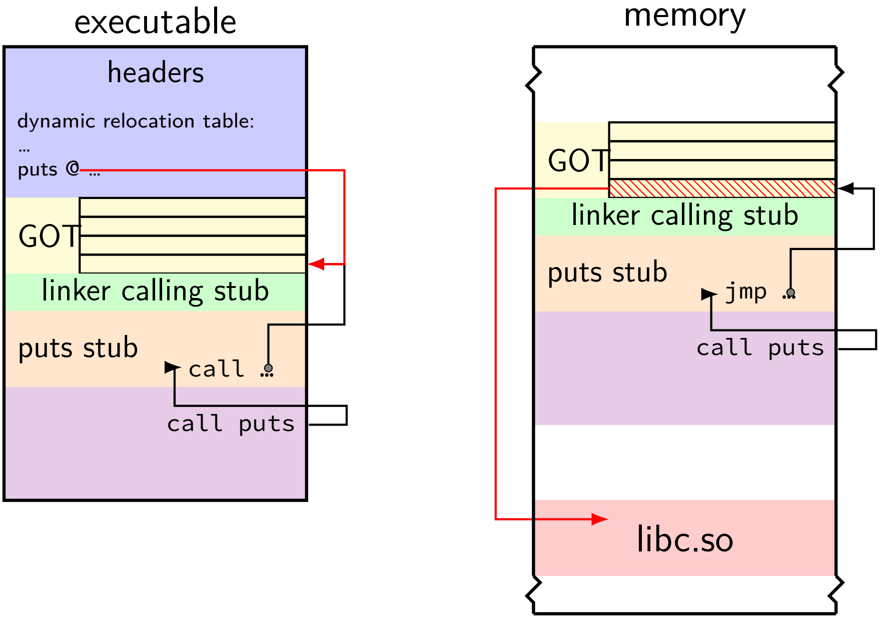
```
GLOBAL_LOOKUP_TABLE:
    ... // placeholders

puts@plt:
    jmp *GLOBAL_LOOKUP_TABLE[index]
    ...
...
main:
    ...
    call puts@plt
    ...
```

+ something that says where `puts` in the lookup table

dynamic puts (picture)



dynamic puts (picture)



dynamic linking information

symbol table in libraries: list of functions/variables to find

- with their locations in the library

relocation records in programs: list of functions/variables

- with locations (probably in lookup table) to fill in

dynamically linked puts (non-lazy)

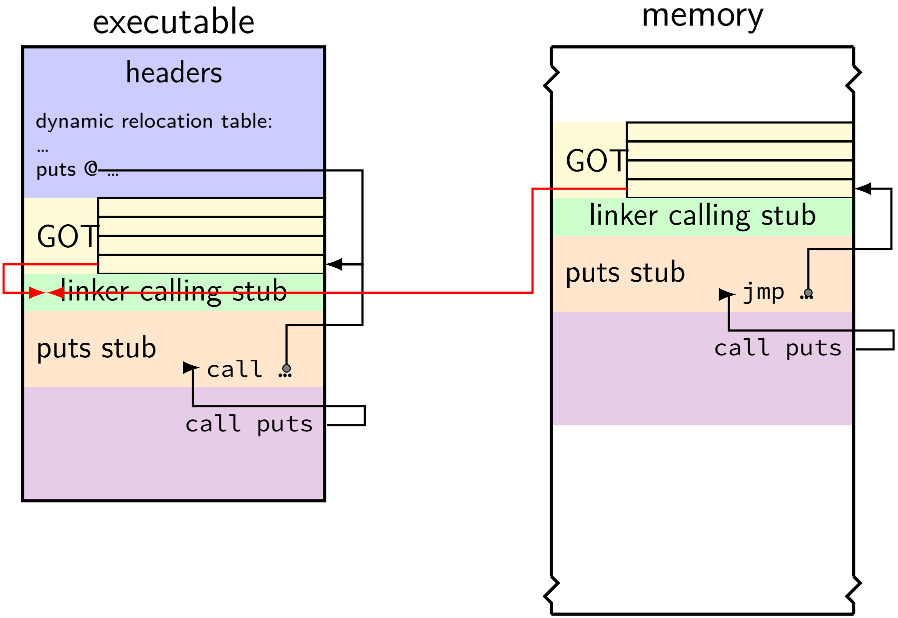
```
DYNAMIC RELOCATION RECORDS
OFFSET                    TYPE                    VALUE
...
0000000000404018 R_X86_64_JUMP_SLOT  puts@GLIBC_2.2.5
...

Text:
0000000000401040 <puts@plt>:
  401040:        f3 0f 1e fa                endbr64
  401044:        f2 ff 25 cd 2f 00 00       bnd jmpq *0x2fcd(%rip) # 404018
```

stub reads pointer from 0x404018, jump to location

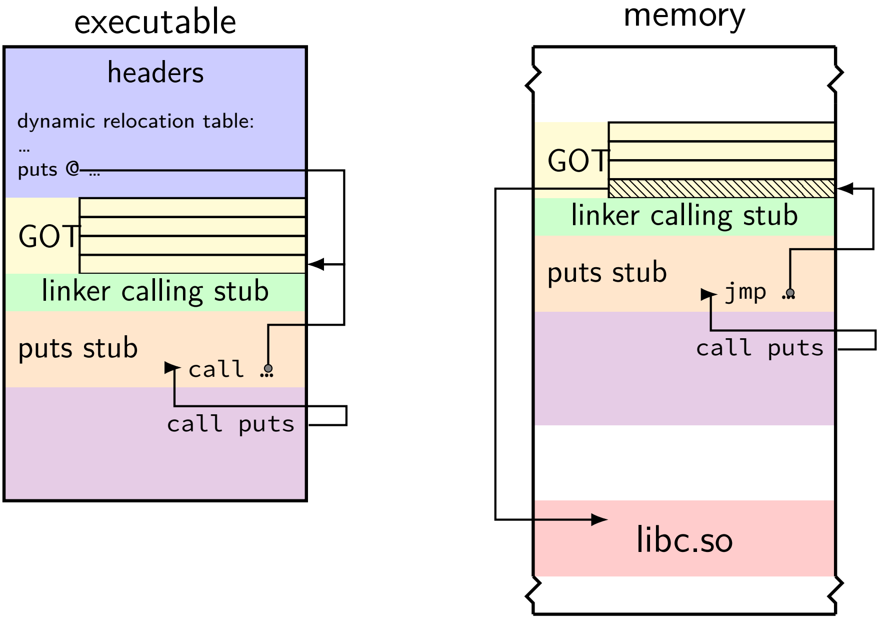
`0x404018` part of 'global offset table' (GOT)

relocation table entry indicates where puts pointer goes

dynamic puts (picture)



dynamic puts (picture)



lazy binding

```
0000000000401040 <puts@plt>:
  401040:       f3 0f 1e fa             endbr64
  401044:       f2 ff 25 cd 2f 00 00    bnd jmpq *0x2fcd(%rip)  # 404018
...
Contents of section .got.plt:
 404000 203e4000 00000000 00000000 00000000   >@.............
 404010 00000000 00000000 30104000 00000000  ........0.@.....
```

initial contents of `0x404018` = `0x401030` (in .got.plt)

not part of standard library????

lazy binding

```
0000000000401040 <puts@plt>:
  401040:       f3 0f 1e fa             endbr64
  401044:       f2 ff 25 cd 2f 00 00    bnd jmpq *0x2fcd(%rip)   # 404018
...
Contents of section .got.plt:
 404000 203e4000 00000000 00000000 00000000   >@.............
 404010 00000000 00000000 30104000 00000000  ........0.@.....
```

initial contents of `0x404018` = `0x401030` (in .got.plt)

not part of standard library????

code found at 0x401030 is routine to invoke dynamic linker code:

```
  401020:       ff 35 e2 2f 00 00       pushq  0x2fe2(%rip)        # 404008 <_GLOBAL_OFFSET_T
  401026:       f2 ff 25 e3 2f 00 00    bnd jmpq *0x2fe3(%rip)        # 404010 <_GLOBAL_OFFSE
  40102d:       0f 1f 00                nopl   (%rax)
  401030:       f3 0f 1e fa             endbr64
  401034:       68 00 00 00 00          pushq  $0x0
  401039:       f2 e9 e1 ff ff ff       bnd jmpq 401020 <.plt>
```

lazy binding

with lazy binding turned on (not always done)

GOT loaded with address of linker routine hard-coded in executable

first call to puts:

- invoke dynamic linker routine pointed to by GOT
- linker routine fills in puts address in `0x404018`
- then jumps to puts

second (and later) call to puts

- 0x404018 contains real address of puts, no indirection

lazy binding pro/con

advantages:

- faster program loading
- no overhead for unused code (often a lot of stuff)

disadvantages:

- can move errors (missing functions, etc.) to runtime
- possibly more total overhead
- means global offset table needs to be writable?

dynamic library headers

```
/lib/x86_64-linux-gnu/libc.so.6:     file format elf64-x86-64
/lib/x86_64-linux-gnu/libc.so.6
architecture: i386:x86-64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x00000000000271f0

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000000040 paddr 0x000000000000004
         filesz 0x0000000000000310 memsz 0x0000000000000310 flags r--
  INTERP off    0x00000000001c16a0 vaddr 0x00000000001c16a0 paddr 0x00000000001c16a
         filesz 0x000000000000001c memsz 0x000000000000001c flags r--
    LOAD off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x000000000000000
         filesz 0x0000000000024940 memsz 0x0000000000024940 flags r--
...
```

DYNAMIC — instead of EXEC_P

dynamic library headers

```
/lib/x86_64-linux-gnu/libc.so.6:     file format elf64-x86-64
/lib/x86_64-linux-gnu/libc.so.6
architecture: i386:x86-64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x00000000000271f0

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000000040 paddr 0x000000000000004
         filesz 0x0000000000000310 memsz 0x0000000000000310 flags r--
  INTERP off    0x00000000001c16a0 vaddr 0x00000000001c16a0 paddr 0x00000000001c16a
         filesz 0x000000000000001c memsz 0x000000000000001c flags r--
    LOAD off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x000000000000000
         filesz 0x0000000000024940 memsz 0x0000000000024940 flags r--
...
```

specifies loading starting at address 0
but dynamic linker will actually choose a different starting address

position-independent executables

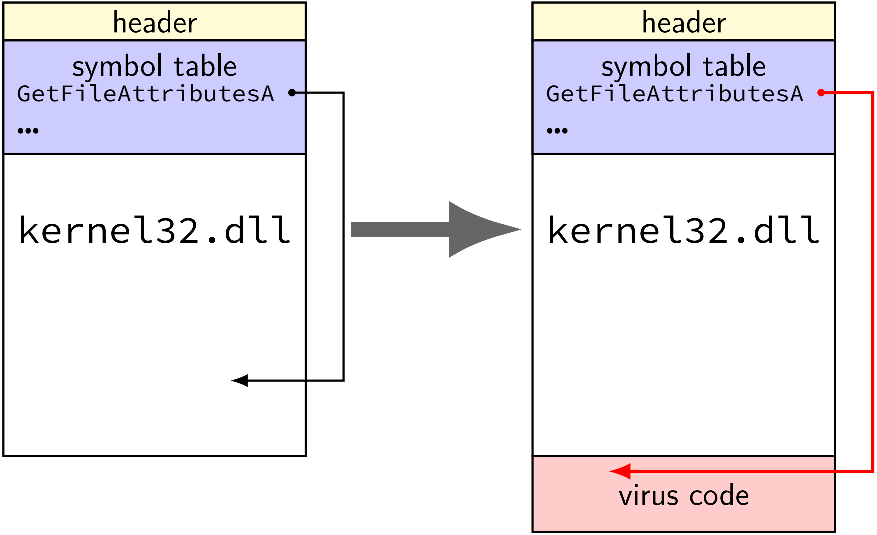
```
hello.exe:     file format elf64-x86-64
hello.exe
architecture: i386:x86-64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x0000000000001080

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000000040 paddr 0x000000000000004
         filesz 0x00000000000002d8 memsz 0x00000000000002d8 flags r--
  INTERP off    0x0000000000000318 vaddr 0x0000000000000318 paddr 0x000000000000031
         filesz 0x000000000000001c memsz 0x000000000000001c flags r--
    LOAD off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x000000000000000
         filesz 0x00000000000005f8 memsz 0x00000000000005f8 flags r--
```

executable with headers like dynamic library
"position-independent executable": can be loaded at any address

position-independent executables

```
hello.exe:     file format elf64-x86-64
hello.exe
architecture: i386:x86-64, flags 0x00000150:
HAS_SYMS, DYNAMIC, D_PAGED
start address 0x0000000000001080

Program Header:
    PHDR off    0x0000000000000040 vaddr 0x0000000000000040 paddr 0x000000000000004
         filesz 0x00000000000002d8 memsz 0x00000000000002d8 flags r--
  INTERP off    0x0000000000000318 vaddr 0x0000000000000318 paddr 0x00000000000031
         filesz 0x000000000000001c memsz 0x000000000000001c flags r--
    LOAD off    0x0000000000000000 vaddr 0x0000000000000000 paddr 0x000000000000000
         filesz 0x00000000000005f8 memsz 0x00000000000005f8 flags r--
```

infecting shared libraries via relocations



other dynamic-linking-based infections

could also modify

relocations on executable

- this isn't the table entry for `puts`, it's the one for `evilvirus`

list of needed libraries?

- the C standard library and virus.so

stubs and calls to stub

- very regular and easy to locate

virus library calls

common for viruses to want to call standard library

in applications:

- just call the stub

problem: stubs located at different parts of different applications

- virus code won't work in every one

some possible solutions:

- hard-code common loaded addresses (hope nothing changes)
- rewrite standard library function yourself
- scan linker data structures to find
- write out new library/executable and run it

preview: exploits and dynamic linking

later we'll talk about memory error exploits

- buffer overflows, etc.

common goal: convert memory overwrite to running code

- bug that "just" allows overwriting memory somewhere
- easy to cause crashes...
- but not so easy to do something in particular

global offset table: function pointers in known location

- useful to overwrite in exploits

summary

how to hide:

- separate executable
- append
- existing “unused” space
- append + compression

how to run:

- change entry point (start address)
- change calls
- change beginning of function
- change dynamic-linking-related pointers
- arrange to run as part of OS

virus: easiest code to find?

what should be easiest/hardest to identify
without many false positives?

A. replaced start location

B. replaced dynamic linker stub

C. replaced dynamic library symbol location

D. replaced function call

E. replaced function return

F. replaced bootloader

G. new automatically started system program

virus choices?

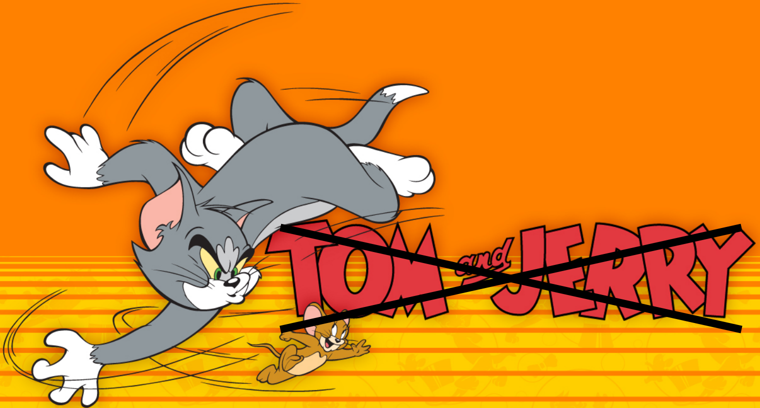
why don't viruses always append/replace?

why don't viruses always change start location?

why did I bother talking about all these strategies?

more on virus strategies

after we talk about anti-virus strategies some



Anti-Virus and Virus

CARTOON NETWORK

anti-malware software goals

prevent malware from running

prevent malware from spreading

undo the effects of malware

anti-malware software goals

prevent malware from running

prevent malware from spreading

undo the effects of malware

key subproblem: detect malware

tripwire

open source tool from c. 2000

also company around that tool, but I don't know about it

"tool for monitoring and alerting on file & directory changes"

targetted at servers with professional administrators

setup: run tool, it records state of system/etc. files (e.g. hashes)

later: run tool, it tells you if anything changed

tripwire as antimalware software?

tripwire idea: detect any changes

notify user (administrator) about them

what is user supposed to do with this info?

what about normal software updates, etc.?

can malware hide in files that are supposed to change?

- "data" files with other programs, scripts?
- …

what if system compromised before setup?

application whitelisting

how about we only let standard applications run, unmodified?

AppStore-based strategy?

not uncommon in corporate environments:

AppLocker

10/16/2017 • 7 minutes to read • +5

Applies to

- Windows 10
- Windows Server

This topic provides a description of AppLocker and can help you decide if your organization can benefit from deploying AppLocker application control policies. AppLocker helps you control which apps and files users can run. These include executable files, scripts, Windows Installer files, dynamic-link libraries (DLLs), packaged apps, and packaged app installers.

case study: Microsoft's AppLocker

AppLocker is Windows 10 feature for limiting what can run

adminstrator sets rules about...

what publisher is allowed

- publisher cryptographically signs applications
- virus-like techniques break signatures
- allows upgrades!

what file hashes are allowed

- requires manual update each time software updates

what locations are allowed

- presumably for administrator-only directories

problems with whitelisting

programs with features/bugs malware could exploit

> "AppLocker does not control the behavior of applications after they are launched. Applications could contain flags passed to functions that signal AppLocker to circumvent the rules and allow another .exe or .dll to be loaded."

users want to install/develop other software

scripting:

> "Not all host processes call into AppLocker and, therefore, AppLocker cannot control every kind of interpreted code"

modern bootloaders — secure boot

"Secure Boot" is a common feature of modern bootloaders

idea: UEFI/BIOS code checks bootloader code, fails if not okay

- requires user intervention to use not-okay code

Secure Boot and keys

Secure Boot relies on cryptographic signatures

- idea: accept only "legitimate" bootloaders
- legitimate: known authority vouched for them

user control of their own systems?

- in theory: can add own keys

what about changing OS instead of bootloader?

- need smart bootloader

malware "signatures"

typically can't rely on whitelisting approach

- software and related files change legitimately
- (note: malware might not be in main executables)

antivirus vendor have signatures for known malware

many options to represent signatures

thought process: signature for Vienna?

goals: compact, fast to check, reliable

what signature for Vienna?

Suppose we wanted to detect Vienna in execs.

What is best to look for in an exectuable...
in terms of performance? false positives? true positives?

A. machine code found in example infected file at the end of the executable
B. machine code found in example infected file at the end of the executable, ignoring parts that change on reinfection
C. portion of virus's machine code that copies itself to a new file anywhere in the executable
D. whether another executable file in same directory changes if we run the executable in a VM
E. for a jump at beginning of the executable to something near the end

exercise: signatures for Vienna

```
jmp 0x0700
mov $0x9e4e, %si
...
/* app code */
...
push %cx
mov $0x8f9, %si
...
mov $0x0100, %di
mov $3, %cx
rep movsb
...
```

```
...
add $0x2f9, %cx
mov %si, %di
sub $0x1f7, %di
mov %cx, (%di)
...
mov $0x288, %cx
mov $0x40 %ah
mov $si, $dx
sub $0x1f9, %dx
int 0x21
...
```

```
pop %cx
xor %ax, %ax
xor %bx, %bx
xor %dx, %dx
mov $0x0100, %di
push %di
xor %di, %di
ret
/* virus data */
```

exercise: signatures for Vienna

```
jmp 0x0700
mov $0x9e4e, %si
...
/* app code */
...
push %cx
mov $0x8f9, %si
...
mov $0x0100, %di
mov $3, %cx
rep movsb
...
```

```
...
add $0x2f9, %cx
mov %si, %di
sub $0x1f7, %di
mov %cx, (%di)
...
mov $0x288, %cx
mov $0x40 %ah
mov $si, $dx
sub $0x1f9, %dx
int 0x21
...
```

```
pop %cx
xor %ax, %ax
xor %bx, %bx
xor %dx, %dx
mov $0x0100, %di
push %di
xor %di, %di
ret
/* virus data */
```

exercise: signatures for Vienna

```
jmp 0x0700
mov $0x9e4e, %si
...
/* app code */
...
push %cx
mov $0x8f9, %si
...
mov $0x0100, %di
mov $3, %cx
rep movsb
...
```

```
...
add $0x2f9, %cx
mov %si, %di
sub $0x1f7, %di
mov %cx, (%di)
...
mov $0x288, %cx
mov $0x40 %ah
mov $si, $dx
sub $0x1f9, %dx
int 0x21
...
```

```
pop %cx
xor %ax, %ax
xor %bx, %bx
xor %dx, %dx
mov $0x0100, %di
push %di
xor %di, %di
ret
/* virus data */
```

exercise: signatures for Vienna

```
jmp 0x0700
mov $0x9e4e, %si
...
/* app code */
...
push %cx
mov $0x8f9, %si
...
mov $0x0100, %di
mov $3, %cx
rep movsb
...
```

```
...
add $0x2f9, %cx
mov %si, %di
sub $0x1f7, %di
mov %cx, (%di)
...
mov $0x288, %cx
mov $0x40 %ah
mov $si, $dx
sub $0x1f9, %dx
int 0x21
...
```

```
pop %cx
xor %ax, %ax
xor %bx, %bx
xor %dx, %dx
mov $0x0100, %di
push %di
xor %di, %di
ret
/* virus data */
```

simple signature (1)

all the code Vienna copies

… except changed `mov` to `%si`

virus doesn't change it to relocate

includes infection code — definitely malicious

signature generality

the Vienna virus was copied a bunch of times

small changes, "payloads" added

- print messages, do different malicious things, ...

this signature will not detect any variants

can we do better?

simple signature (2)

Vienna start code

- weird jump at beginning??

problem: maybe real applications do this?

problem: easy to move jump

simple signature (3)

Vienna infection code

- scans directory, finds files

likely to stay the same in variants?

simple signature (3)

Vienna infection code

- scans directory, finds files

likely to stay the same in variants?

problem: virus writers react to antivirus

simple signature (4)

Vienna finish code

push + ret

very unusual pattern

probably(?) not in “real” programs

real effort to change to something else?

simple signature (4)

Vienna finish code

push + ret

very unusual pattern

probably(?) not in “real” programs

real effort to change to something else?

problem: virus writers react to antivirus

making things hard for the mouse

don't want trivial changes to break detection

want to detect strategies

e.g. require changing relocation logic

...not just reordering instructions, adding nops

need to detect signatures in real time

don't want interrupt user (much)

want to avoid false positive

goals: compact, fast to check, reliable, general?

generic pattern example

another possibility: detect writing near `0x100`

`0x100` was DOS program entry code — no program should do this(?)

problem: how to represent this?

- describe machine code bytes
- multiple possibilities

regular expressions

one method of representing patterns like this:
regular expressions (regexes)

restricted language allows very fast implementations

especially when there's a long list of patterns to look for

homework assignment next week

regular expressions: implementations

multiple implementations of regular expressions

we will target: flex, a parser generator

simple patterns

alphanumeric characters match themselves

`foo`:

- matches exactly `foo` only
- does not match `Foo`
- does not match `foo␣`
- does not match `foobar`

backslash might be needed for others

`C\+\+`

- matches exactly `C++` only

metachars (1)

special ways to match characters

`\n`, `\t`, `\x3C`, …— work like in C

`[b-fi]` — `b` or `c` or `d` or `e` or `f` or `i`

`[^b-fi]` — any character but `b` or `c` or …

`.` — any character except newline

`(.|\n)` — any character

metachars (2)

a* — zero or more as:

(empty string), a, aa, aaa, ...

a{3,5} — three to five as:

aaa, aaaa, aaaaa

(abc){3,5} — three to five abcs: (“grouping”)

abcabcabc, abcabcabcabc, abcabcabcabcabc

ab|cd

ab, cd

(ab|cd){2} — two ab-or-cds:

abab, abcd, cdab, cdcd

metachars (3)

`\xAB` — the byte `0xAB`

`\x00` — the byte `0x00`

flex is designed for text, handles binary fine

`\n` — newline (and other C string escapes)

example regular expressions

match words ending with `ing`:

```
[a-zA-Z]*ing
```

match C `/* ... */` comments:

```
/\*([^*]|\*[^/])*\*/
```

flex

flex is a regular expression matching tool

intended for writing parsers

generates C code

parser function called `yylex`

flex example

```
        int num_bytes = 0, num_lines = 0;
        int num_foos = 0;
%%
foo     {
          num_bytes += 3;
          num_foos += 1;
        }
.       { num_bytes += 1; }
\n      { num_lines += 1; num_bytes += 1; }
%%
int main(void) {
    yylex();
    printf("%d bytes, %d lines, %d foos\n",
           num_bytes, num_lines, num_foos);
}
```

flex example

```
        int num_bytes = 0, num_lines = 0;
        int num_foos = 0;
%%
foo     {
          num_bytes += 3;
          num_foos += 1;
        }
.       { num_bytes += 1; }
\n      { num_lines += 1; num_bytes += 1; }
%%
int main(void) {
    yylex();
    printf("%d bytes, %d lines, %d foos\n",
           num_bytes, num_lines, num_foos);
}
```

three sections

flex example

```
        int num_bytes = 0, num_lines = 0;
        int num_foos = 0;
%%
foo     {
          num_bytes += 3;
          num_foos
        }
.       { num_bytes += 1; }
\n      { num_lines += 1; num_bytes += 1; }
%%
int main(void) {
    yylex();
    printf("%d bytes, %d lines, %d foos\n",
              num_bytes, num_lines, num_foos);
}
```

first — declarations for later C code in output file

flex example

```
        int num_bytes = 0, num_lines = 0;
        int num_fo
%%
foo     {
          num_bytes += 3;
          num_foos += 1;
        }
.       { num_bytes += 1; }
\n      { num_lines += 1; num_bytes += 1; }
%%
int main(void) {
    yylex();
    printf("%d bytes, %d lines, %d foos\n",
           num_bytes, num_lines, num_foos);
}
```

patterns, code to run on match
as parser: return "token" here

flex example

```
        int num_bytes = 0, num_lines = 0;
        int num_foos = 0;
%%
foo     {
          num_bytes += 3;
          num_foos += 1
        }
.       { num_bytes += 1; }
\n      { num_lines += 1; num_bytes += 1; }
%%
int main(void) {
    yylex();
    printf("%d bytes, %d lines, %d foos\n",
           num_bytes, num_lines, num_foos);
}
```

extra code to include

flex: matched text

```
%%
[aA][a-z]* {
                printf("found a-word '%s'\n",
                       yytext);
           }
(.|\n)     {} /* default rule: would output text */
%%
int main(void) {
    yylex();
}
```

flex: matched text

yytext — text of matched thing

```
%%
[aA][a-z]* {
                    printf("found a-word '%s'\n",
                           yytext);
               }
(.|\n)         {} /* default rule: would output text */
%%
int main(void) {
     yylex();
}
```

flex: definitions

```
A           [aA]
LOWERS      [a-z]
ANY         (.|\n)
%%
{A}{LOWERS}* {
                printf("found a-word '%s'\n",
                       yytext);
             }
{ANY}        {} /* default rule would
                   output text */
%%
int main(void) {
    yylex();
}
```

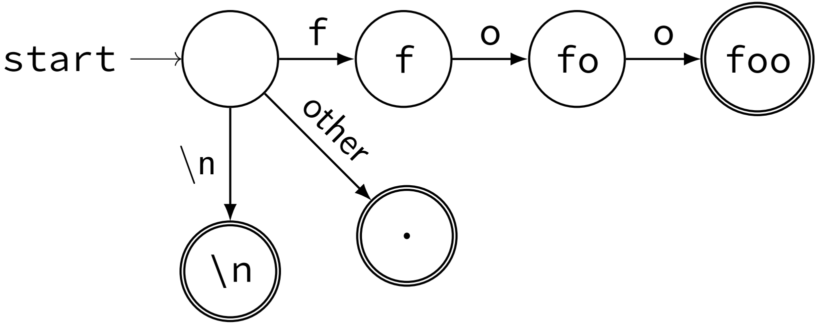
flex: definitions

```
A               [aA]
LOWERS          [a-z]
ANY             (.|\n)
%%
{A}{LOWERS}* {
                    printf("found a-word '%s'\n",
                           yytext);
             }
{ANY}        {} /* default rule would
                   output text */
%%
int main(void) {
    yylex();
}
```

definitions of common patterns included later

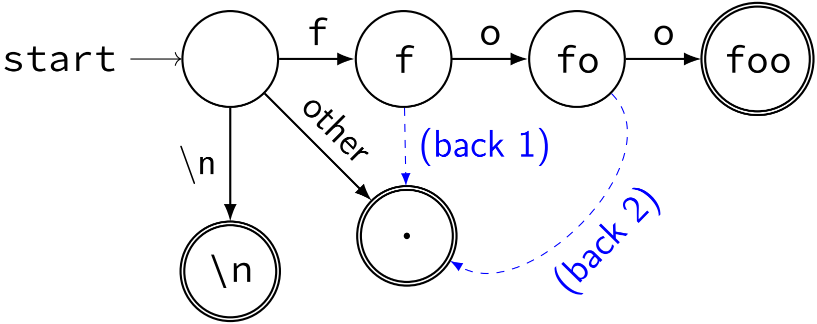
flex: state machines

```
foo       {...}
.         {...}
\n        {...}
```



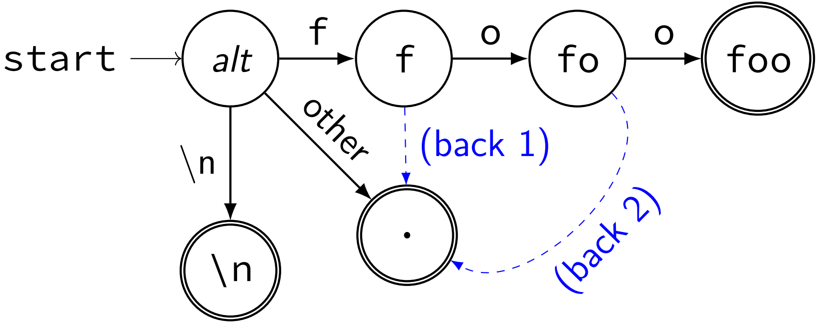
flex: state machines

```
foo     {...}
.       {...}
\n      {...}
```



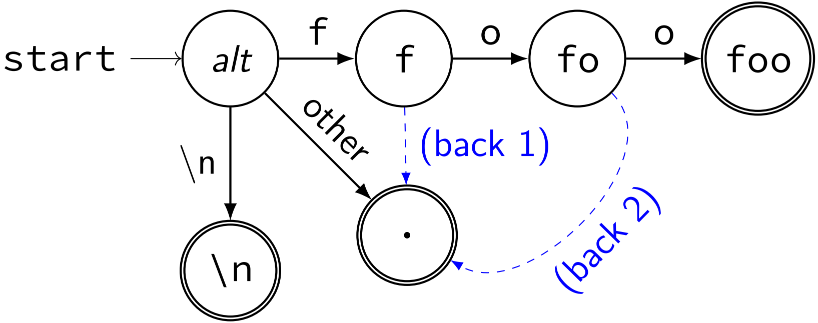
state machine matching

abfoofoabffoo



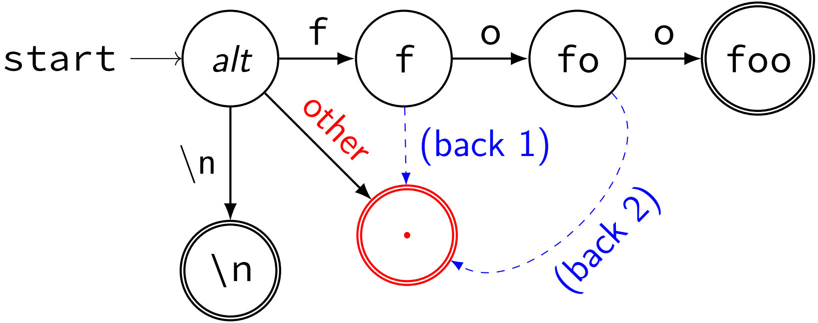
state machine matching

abfoofoabffoo



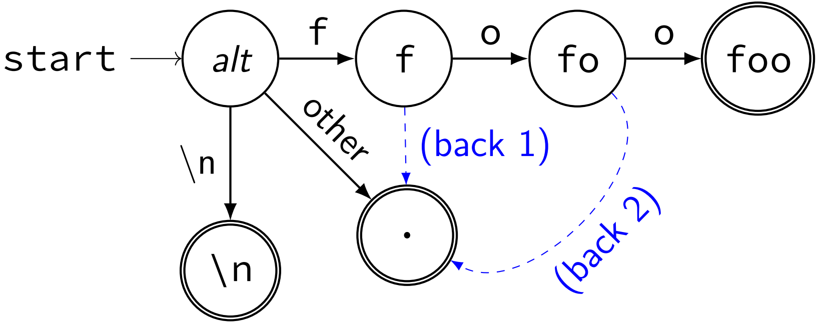
state machine matching

~~a~~bfoofoabffoo



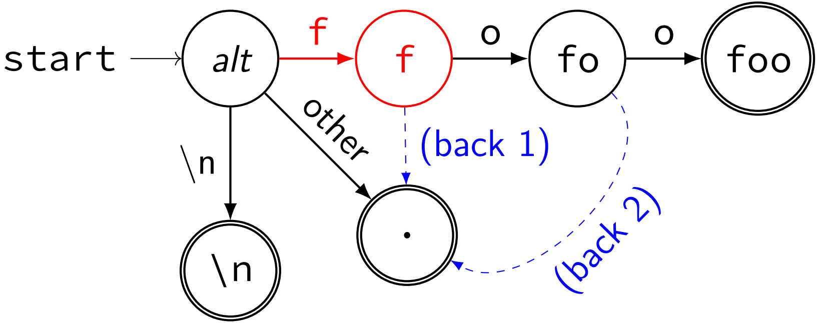
state machine matching

~~ab~~foofoabffoo



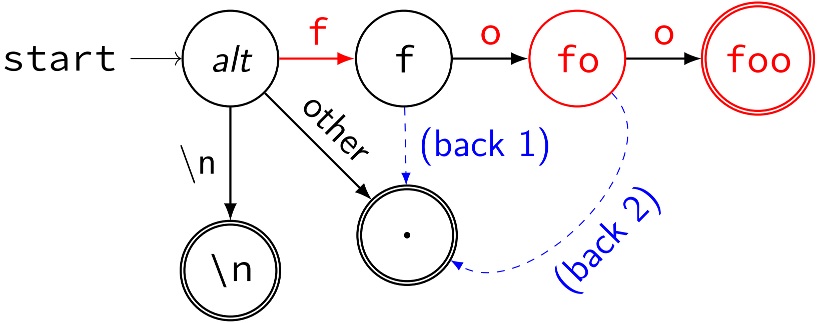
state machine matching

~~ab~~foofoabffoo



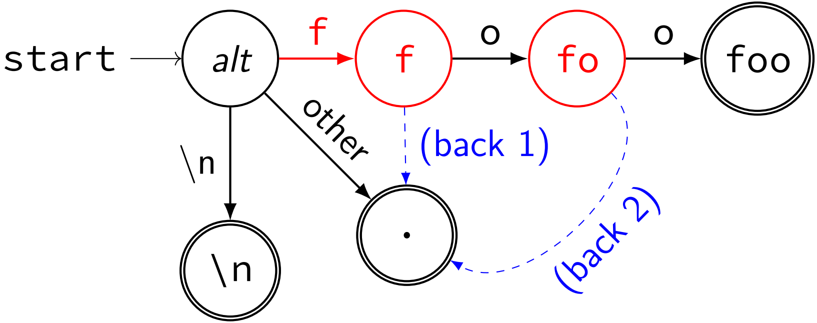
state machine matching

~~ab~~foofoabffoo



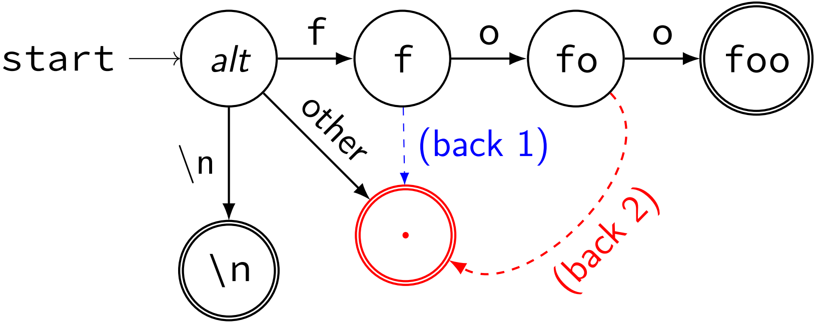
state machine matching

~~abfoo~~foabffoo



state machine matching

~~abfoo~~f~~o~~abffoo



flex states (1)

```
%x str
%%
\"              { BEGIN(str); }
<str>\"         { BEGIN(INITIAL); }
<str>foo        { printf("foo in string\n"); }
foo             { printf("foo out of string\n"); }
<INITIAL,str>(.|\n) {}
%%
int main(void) {
    yylex();
}
```

flex states (1)

```
%x str
%%
\"              { BEGIN(str); }
<str>\"         { BEGIN(INITIAL); }
<str>foo        { printf("foo in string\n"); }
foo             { printf("foo out of string\n"); }
<INITIAL,str>(.|\n) {}
%%
int ma
    yy
}
```

declare "state" to track

which state determines what patterns are active

flex states (1)

```
%x str
%%
\"              { BEGIN(str); }
<str>\"         { BEGIN(INITIAL); }
<str>foo        { printf("foo in string\n"); }
foo             { printf("foo out of string\n"); }
<INITIAL,str>(.|\n) {}
%%
int main(void) {
    yylex();
}
```

flex states (2)

```
%s afterFoo
%%
<afterFoo>foo    { printf("later foo\n"); }
foo              {
                   printf("first foo\n");
                   BEGIN(afterfoo);
                 }
(.|\n)  {}
%%
int main(void) {
    yylex();
}
```

flex states (2)

```
%s afterFoo
%%
<afterFoo>foo     { printf("later foo\n"); }
foo               {
                    printf("first foo\n");
                    BEGIN(afterfoo);
                  }
(.|\n)  {}
%%
int main(void) {
    yylex();
}
```

declare non-exclusive state

backup slides

evading signatures

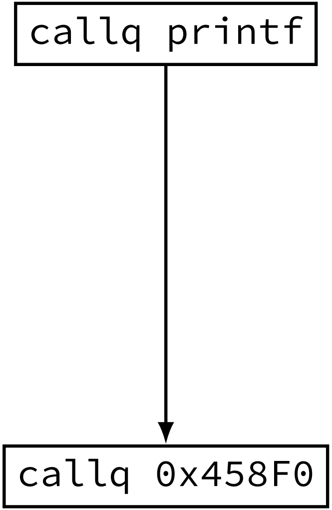
idea for detecting Vienna:

signature = code excluding changing part from relocation

exercise 1: what are some **easy** changes to Vienna that would evade this **strategy for making signatures**?

exercise 2: what are some ways of handling those?

linking



static v. dynamic linking

static linking — linking to create executable

dynamic linking — linking when executable is run

static v. dynamic linking

static linking — linking to create executable

dynamic linking — linking when executable is run

conceptually: no difference in how they work

reality — very different mechanisms

- extra indirection to avoid modifying much code at runtime
- change *lookup table* instead of code for library locations

interlude: strace

`strace` — system call tracer

- on Linux, some other Unices
- OS X approx. equivalent: `dtruss`
- Windows approx. equivalent: Process Monitor

indicates what system calls (operating system services) used by a program

statically linked hello.exe

```
gcc -static -o hello-static.exe hello.s
```

`strace ./hello-static.exe`:

```
execve("./hello-static.exe", ["./hello-static.exe"], [/* 46 vars */]) = 0
uname(sysname="Linux", nodename="reiss-lenovo", ...) = 0
brk(NULL)                               = 0x20a5000
brk(0x20a61c0)                          = 0x20a61c0
arch_prctl(ARCH_SET_FS, 0x20a5880)      = 0
readlink("/proc/self/exe", "/home/cr4bd/spring2017/cs4630/sl"..., 4096) = 62
brk(0x20c71c0)                          = 0x20c71c0
brk(0x20c8000)                          = 0x20c8000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
fstat(1, st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...) = 0
write(1, "Hello, World!\n", 14)         = 14
exit_group(14)                          = ?
+++ exited with 14 +++
```

statically linked hello.exe

`gcc -static -o hello-static.exe hello.s`

`strace ./hello-static.exe`:

```
execve("./hello-static.exe", ["./hello-static.exe"], [/* 46 vars */]) = 0
uname(sysname="Linux", nodename="reiss-lenovo", ...) = 0
brk(NULL)                               = 0x20a5000
brk(0x20a61c0)                          = 0x20a61c0
arch_prctl(ARCH_SET_FS, 0x20a5880)      = 0
readlink("/proc/self/exe", "/home/cr4bd/spring2017/cs4630/sl"..., 4096) = 62
brk(0x20c71c0)                          = 0x20c71c0
brk(0x20c8000)                          = 0x20c8000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
fstat(1, st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...) = 0
write(1, "Hello, World!\n", 14)         = 14
exit_group(14)                          = ?
+++ exited with 14 +++
```

standard library startup

statically linked hello.exe

`gcc -static -o hello-static.exe hello.s`

`strace ./hello-static.exe`:

```
execve("./hello-static.exe", ["./hello-static.exe"], [/* 46 vars */]) = 0
uname(sysname="Linux", nodename="reiss-lenovo", ...) = 0
brk(NULL)                                 = 0x20a5000
brk(0x20a61c0)                            = 0x20a61c0
arch_prctl(ARCH_SET_FS, 0x20a5880)        = 0
readlink("/proc/self/exe", "/home/cr4bd/spring2017/cs4630/sl"..., 4096) = 62
brk(0x20c71c0)                            = 0x20c71c0
brk(0x20c8000)                            = 0x20c8000
access("/etc/ld.so.nohwcap", F_OK)        = -1 ENOENT (No such file or directory)
fstat(1, st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...) = 0
write(1, "Hello, World!\n", 14)           = 14
exit_group(14)                            = ?
+++ exited with 14 +++
```

memory allocation

statically linked hello.exe

```
gcc -static -o hello-static.exe hello.s
```

`strace ./hello-static.exe`:

```
execve("./hello-static.exe", ["./hello-static.exe"], [/* 46 vars */]) = 0
uname(sysname="Linux", nodename="reiss-lenovo", ...) = 0
brk(NULL)                                 = 0x20a5000
brk(0x20a61c0)                            = 0x20a61c0
arch_prctl(ARCH_SET_FS, 0x20a5880)        = 0
readlink("/proc/self/exe", "/home/cr4bd/spring2017/cs4630/sl"..., 4096) = 62
brk(0x20c71c0)                            = 0x20c71c0
brk(0x20c8000)                            = 0x20c8000
access("/etc/ld.so.nohwcap", F_OK)        = -1 ENOENT (No such file or directory)
fstat(1, st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...) = 0
write(1, "Hello, World!\n", 14)           = 14
exit_group(14)                            = ?
+++ exited with 14 +++
```

implementation of puts

statically linked hello.exe

```
gcc -static -o hello-static.exe hello.s
```

`strace ./hello-static.exe:`

```
execve("./hello-static.exe", ["./hello-static.exe"], [/* 46 vars */]) = 0
uname(sysname="Linux", nodename="reiss-lenovo", ...) = 0
brk(NULL)                                = 0x20a5000
brk(0x20a61c0)                           = 0x20a61c0
arch_prctl(ARCH_SET_FS, 0x20a5880)       = 0
readlink("/proc/self/exe", "/home/cr4bd/spring2017/cs4630/sl"..., 4096) = 62
brk(0x20c71c0)                           = 0x20c71c0
brk(0x20c8000)                           = 0x20c8000
access("/etc/ld.so.nohwcap", F_OK)       = -1 ENOENT (No such file or directory)
fstat(1, st_mode=S_IFCHR|0620, st_rdev=makedev(136, 1), ...) = 0
write(1, "Hello, World!\n", 14)          = 14
exit_group(14)                           = ?
+++ exited with 14 +++
```

standard library shutdown

dynamically linked hello.exe

`gcc -o hello.exe hello.s`

`strace ./hello.exe`:

```
execve("./hello.exe", ["./hello.exe"], [/* 46 vars */]) = 0
...
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdfeeb39000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, st_mode=S_IFREG|0644, st_size=137808, ...) = 0
...
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) = 832
fstat(3, st_mode=S_IFREG|0755, st_size=1864888, ...) = 0
mmap(NULL, 3967392, PROT_READ|PROT_EXEC, ..., 3, 0) = 0x7fdfee54d000
mprotect(0x7fdfee70c000, 2097152, PROT_NONE) = 0
mmap(0x7fdfee90c000, 24576, PROT_READ|PROT_WRITE, ..., 3, 0x1bf000) = 0x7fdfee90c000
mmap(0x7fdfee912000, 14752, PROT_READ|PROT_WRITE, ..., -1, 0) = 0x7fdfee912000
close(3)                                = 0
...
write(1, "Hello, World!\n", 14)         = 14
exit_group(14)                          = ?
+++ exited with 14 +++
```

dynamically linked hello.exe

```
gcc -o hello.exe hello.s
```

`strace ./hello.exe`:

```
execve("./hello.exe", ["./hello.exe"], [/* 46 vars */]) = 0
...
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdfeeb39000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, st_mode=S_IFREG|0644, st_size=137808, ...) = 0
...
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) = 832
fstat(3, st_mode=S_IFREG|0755, st_size=1864888, ...) = 0
mmap(NULL, 3967392, PROT_READ|PROT_EXEC, ..., 3, 0) = 0x7fdfee54d000
mprotect(0x7fdfee70c000, 2097152, PROT_NONE) = 0
mmap(0x7fdfee90c000,                                          0x7fdfee90c000
mmap(0x7fdfee912000,                                          e912000
close(3)                                = 0
...
write(1, "Hello, World!\n", 14)         = 14
exit_group(14)                          = ?
+++ exited with 14 +++
```

the standard C library (includes puts)

dynamically linked hello.exe

```
gcc -o hello.exe hello.s
```

strace ./hello.exe:

```
execve("./hello.exe", ["./hello.exe"], [/* 46 vars */]) = 0
...
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdfeeb39000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, st_mode=S_IFREG|0644, st_size=137808, ...) = 0
...
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) = 832
fstat(3, st_mode=S_IFREG|0755, st_size=1864888, ...) = 0
mmap(NULL, 3967392, PROT_READ|PROT_EXEC, ..., 3, 0) = 0x7fdfee54d000
mprotect(0x7fdfee70c000, 2097152, PROT_NONE) = 0
mmap(0x7fdfee90c000,                                             0x7fdfee90c000
mmap(0x7fdfee912000,                                             ee912000
close(3)                                = 0
...
write(1, "Hello, World!\n", 14)         = 14
exit_group(14)                          = ?
+++ exited with 14 +++
```

memory allocation (different method)

dynamically linked hello.exe

`gcc -o hello.exe hello.s`

`strace ./hello.exe`:

```
execve("./hello.exe", ["./hello.exe"], [/* 46 vars */]) = 0
...
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdfeeb39000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, st_mode=S_IFREG|0644, st_size=137808, ...) = 0
...
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) = 832
fstat(3, st_mode=S_IFREG|0755, st_size=1864888, ...) = 0
mmap(NULL, 3967392, PROT_READ|PROT_EXEC, ..., 3, 0) = 0x7fdfee54d000
mprotect(0x7fdfee70c000, 2097152, PROT_NONE) = 0
mmap(0x7fdfee90c000, 2457                    0) = 0x7fdfee90c000
mmap(0x7fdfee912000, 1475                    x7fdfee912000
close(3)                                = 0
...
write(1, "Hello, World!\n", 14)         = 14
exit_group(14)                          = ?
+++ exited with 14 +++
```

read standard C library header

dynamically linked hello.exe

`gcc -o hello.exe hello.s`

`strace ./hello.exe`:

```
execve("./hello.exe", ["./hello.exe"], [/* 46 vars */]) = 0
...
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdfeeb39000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, st_mode=S_IFREG|0644, st_size=137808, ...) = 0
...
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) = 832
fstat(3, st_mode=S_IFREG|0755, st_size=1864888, ...) = 0
mmap(NULL, 3967392, PROT_READ|PROT_EXEC, ..., 3, 0) = 0x7fdfee54d000
mprotect(0x7fdfee70c000, 2097152, PROT_NONE) = 0
mmap(0x7fdfee90c000                                          7fdfee90c000
mmap(0x7fdfee912000                                          912000
close(3)                                = 0
...
write(1, "Hello, World!\n", 14)         = 14
exit_group(14)                          = ?
+++ exited with 14 +++
```

load standard C library (3 = opened file)

dynamically linked hello.exe

`gcc -o hello.exe hello.s`

`strace ./hello.exe`:

```
execve("./hello.exe", ["./hello.exe"], [/* 46 vars */]) = 0
...
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdfeeb39000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, st_mode=S_IFREG|0644, st_size=137808, ...) = 0
...
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\t\2\0\0\0\0\0"..., 832) = 832
fstat(3, st_mode=S_IFREG|0755, st_size=1864888, ...) = 0
mmap(NULL, 3967392, PROT_READ|PROT_EXEC, ..., 3, 0) = 0x7fdfee54d000
mprotect(0x7fdfee70c000, 2097152, PROT_NONE) = 0
mmap(0x7fdfee9                                                e90c000
mmap(0x7fdfee9                                                0
close(3)                                = 0
...
write(1, "Hello, World!\n", 14)         = 14
exit_group(14)                          = ?
+++ exited with 14 +++
```

allocate zero-initialized data segment for C library

where's the linker

Where's the code that calls `open("...libc.so.6")`?

Could check `hello.exe` — it's not there!

where's the linker

Where's the code that calls `open("...libc.so.6")`?

Could check `hello.exe` — it's not there!

instead: "interpreter" `/lib64/ld-linux-x86-64.so.2`

on Linux: contains loading code instead of core OS

- OS loads it instead of program

objdump — the interpreter

excerpt from `objdump -sx hello.exe`:

```
Program Header:
...
  INTERP off    0x0000238 vaddr 0x0400318 paddr 0x0400238 align 2**0
         filesz 0x000001c memsz 0x000001c flags r--
...
Contents of section .interp:
 400318 2f6c6962 36342f6c 642d6c69 6e75782d  /lib64/ld-linux-
 400328 7838362d 36342e73 6f2e3200           x86-64.so.2.
```

dynamic linking: what to load? (1)

excerpt from `objdump -sx hello.exe`:

```
Program Header:
...
 DYNAMIC off    0x0000000000002e20 vaddr 0x0000000000403e20 paddr 0x0000000000403e2
         filesz 0x00000000000001d0 memsz 0x00000000000001d0 flags rw-
...
Dynamic Section:
  NEEDED               libc.so.6
  INIT                 0x0000000000401000
  ...
  STRTAB               0x0000000000400420
  ...
```

program header: identifies where dynamic linking info is

dynamic linking info: array of key-value pairs

- needed libraries
- constructor locations (‘INIT’)
- string table location
- …

stubs

```
0000000000001050 <puts@plt>:
    1050:       f3 0f 1e fa             endbr64
    1054:       f2 ff 25 75 2f 00 00    bnd jmpq *0x2f75(%rip)        # 3fd0 <puts@GLIBC_2.2.
     replace with:
0000000000001050 <puts@plt>:
    1050:       f3 0f 1e fa             endbr64
    1054:       ff 25 XX XX XX XX       jmp VIRUS CODE
    105a:       90                      nop
```

in known location (particular section of executable)

stubs again

```
0000000000001050 <puts@plt>:
    1050:       f3 0f 1e fa             endbr64
    1054:       f2 ff 25 75 2f 00 00    bnd jmpq *0x2f75(%rip)        # 3fd0 <puts@GLIBC_2.2.
```

don't edit stub — edit initial value at `0x3fd0`

- stored in data section of executable

originally: pointer to lazy linking code

malware code can jump to original lazy linking code after

preview of future topics: also can be changed by exploits

- commonly involved in memory-error exploits (buffer overflow, etc.)

relocations?

on executable:

```
hello.exe:     file format elf64-x86-64

DYNAMIC RELOCATION RECORDS
OFFSET           TYPE              VALUE
...
0000000000003fd0 R_X86_64_JUMP_SLOT  puts@GLIBC_2.2.5
    replace with:
0000000000003fd0 R_X86_64_JUMP_SLOT  *ABS*+virus_code_address
...
```

relocation record: where to put library code addresses

symbols?

on library:

```
/lib/x86_64-linux-gnu/libc.so.6:     file format elf64-x86-64

DYNAMIC SYMBOL TABLE:
...
00000000000875a0  w   DF .text  00000000000001dc  GLIBC_2.2.5 puts
    replace with:
00000000000875a0  w   DF .text  (virus code off)  GLIBC_2.2.5 puts
```

symbol table entry: where library code is

virus: easiest code to find?

what should be easiest/hardest to identify
without many false positives?

A. replaced start location

B. replaced dynamic linker stub

C. replaced dynamic library symbol location

D. replaced function call

E. replaced function return

F. replaced bootloader

G. new automatically started system program