HARDENING THE BROWSER ADDRESS SPACE

Jared Candelaria - @odiumeh
Alex Rad - @defendtheworld

OUTLINE

- Browser Hardening Motivation
- Types of Bugs
- Existing Mitigations
- New Mitigations
- Conclusion

ALEX RAD

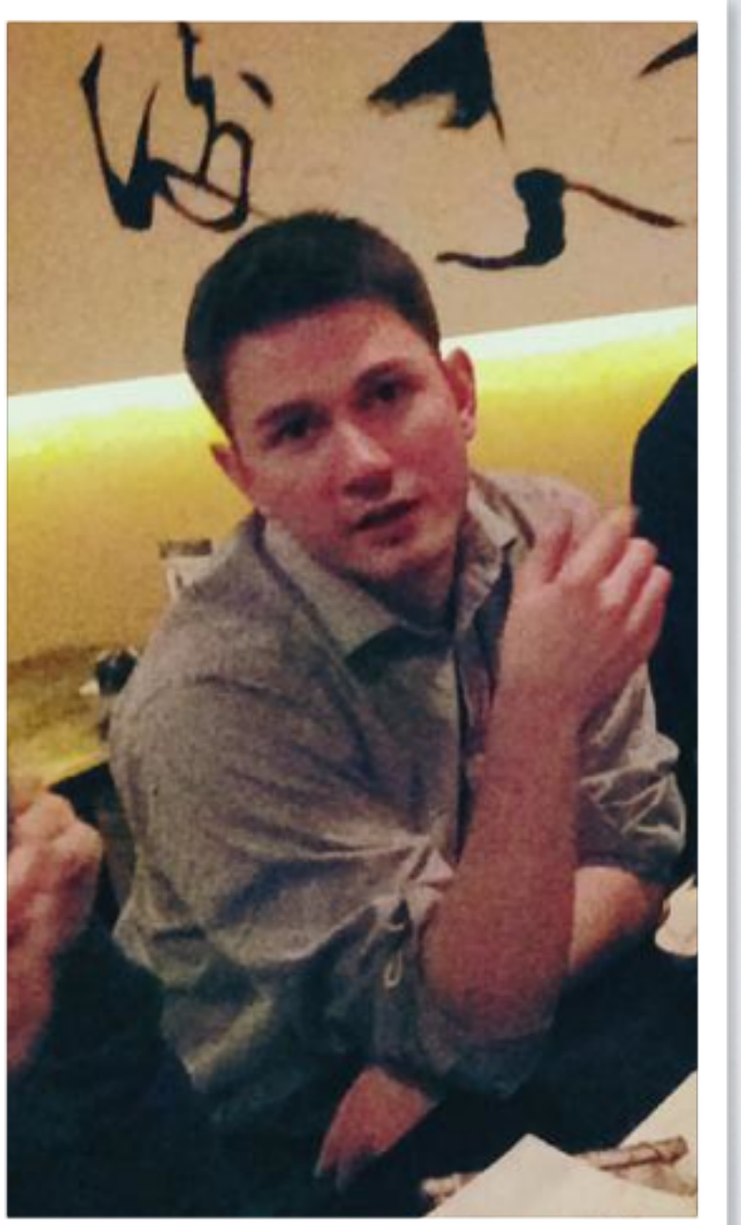
Started playing war-games

Likes

- a discipline of programming
- raising the bar

Dislikes

- fragile software
- noisy signals



JARED CANDELARIA

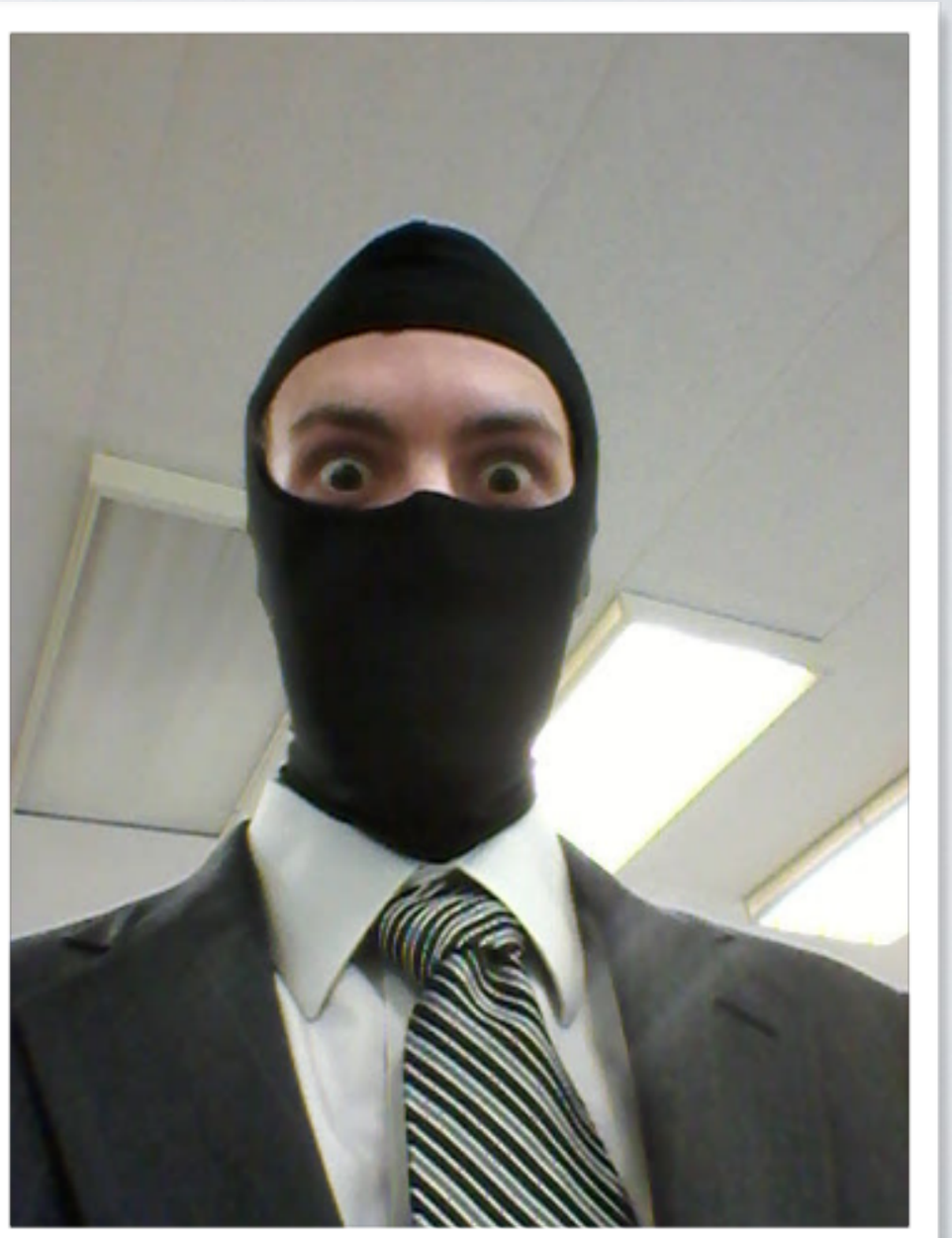
Wanted to become a code wizard

Likes

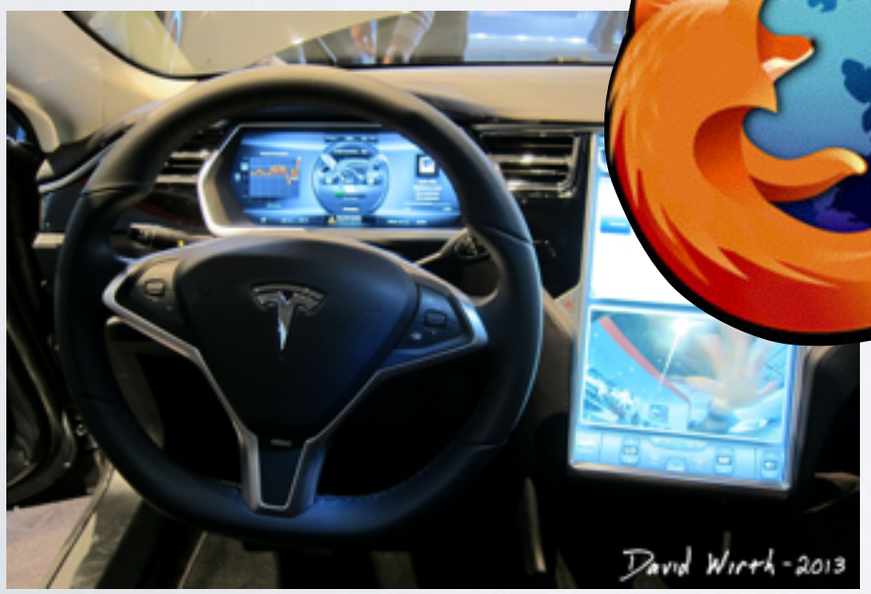
- functional programming
- impractical things

Dislikes

- cmd.exe
- stale pointers



BROWSERS ARE EVERYWHERE



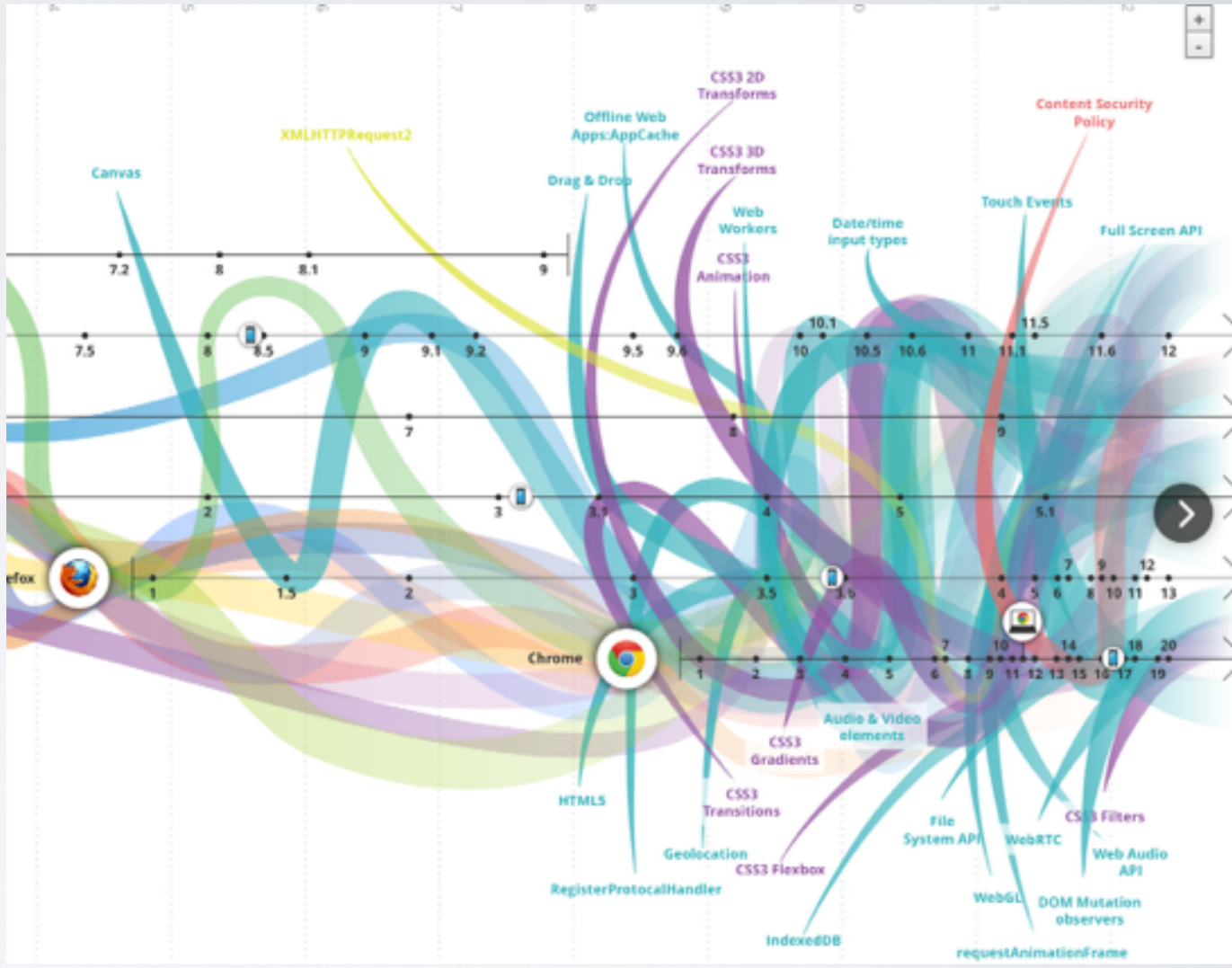
BROWSERS ARE COMPLICATED

- Insanely so

- Untrusted data, async, reflection, rendering everything, parsing everything

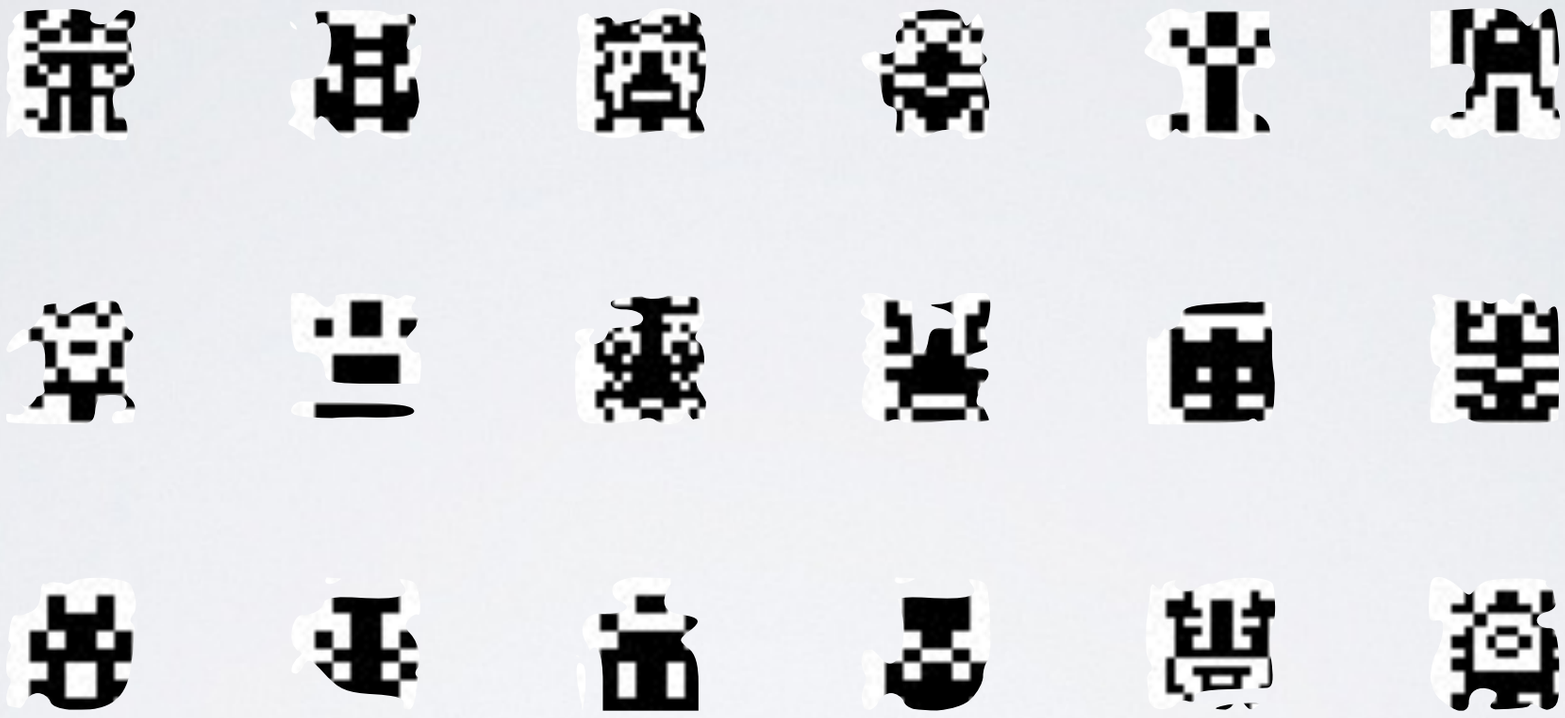
BROWSERS ARE CONSTANTLY DEVELOPED

- New code, new features, frequent releases



http://evolutionoftheweb.com

WHY SHOULD THE SECURITY COMMUNITY CARE



- There are bugs. Infinite bugs

WHAT ARE THEY EXPLOITED FOR?

- Targeted attacks
- Jailbreaks
- Stealing mom's cookies
  - and her credit cards

WHAT ARE ALL THESE BUGS?

- Memory trespass errors
- Content & policy errors
- Metadata injection - UXSS, etc

MEMORY TRESPASS ERRORS

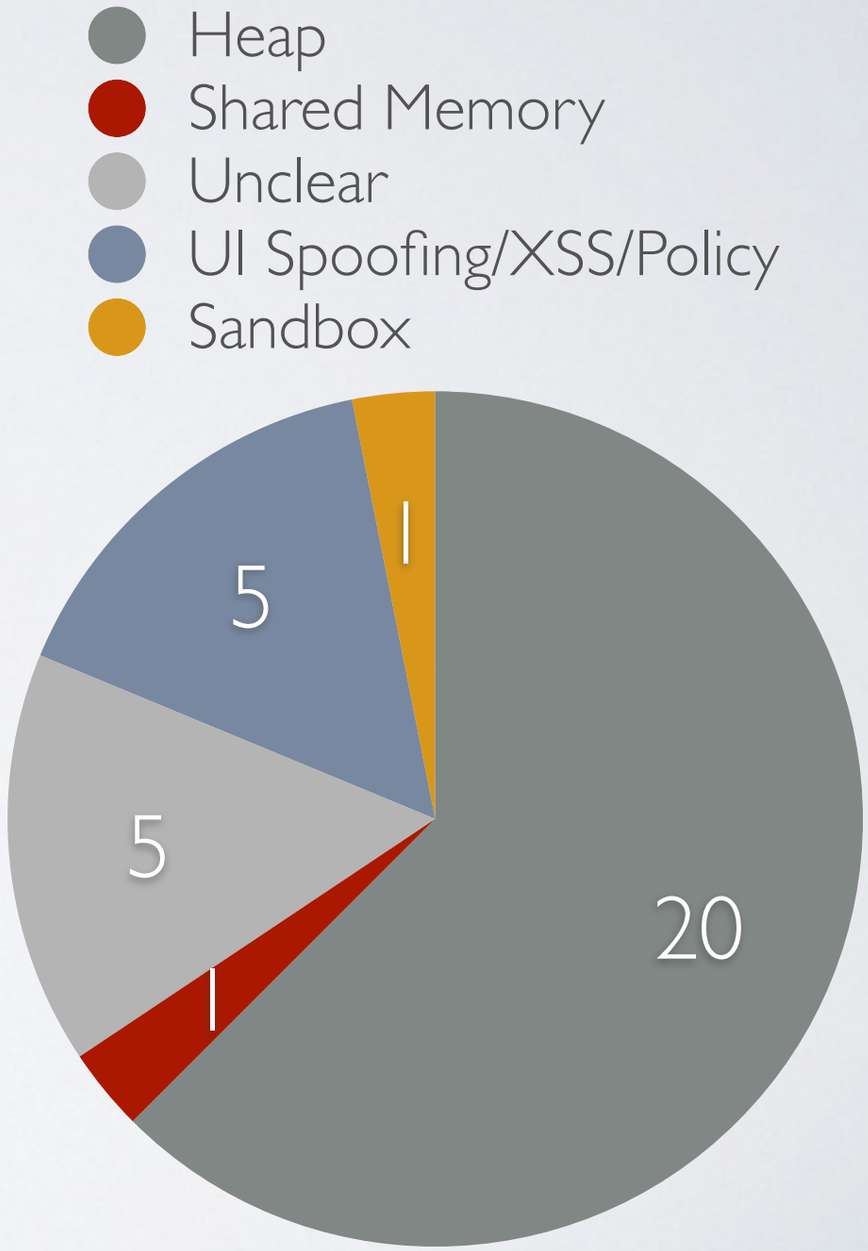
- Root cause: C/C++
- Overflows & Underflows
- Type Confusion
- Dangling References/ Use After Free

CHROME BUG STATISTICS. APRIL 2014-JULY 2014

- 33 CVEs for Chrome

CVE-2014-3154, CVE-2014-3155, CVE-2014-3156, CVE-2014-3157, CVE-2014-1743, CVE-2014-1745:, CVE-2014-1746, CVE-2014-1730, CVE-2014-1731, CVE-2014-1732, CVE-2014-1735, CVE-2014-1717, CVE-2014-1719:, CVE-2014-1720, CVE-2014-1722, CVE-2014-1724, CVE-2014-1725, CVE-2014-1727, CVE-2014-1721, CVE-2014-1749, CVE-2014-1744, CVE-2014-3152, CVE-2014-1734, CVE-2014-1729, CVE-2014-1728, CVE-2014-1747, CVE-2014-1748, CVE-2014-1733, CVE-2014-1716, CVE-2014-1718, CVE-2014-1723, CVE-2014-1726

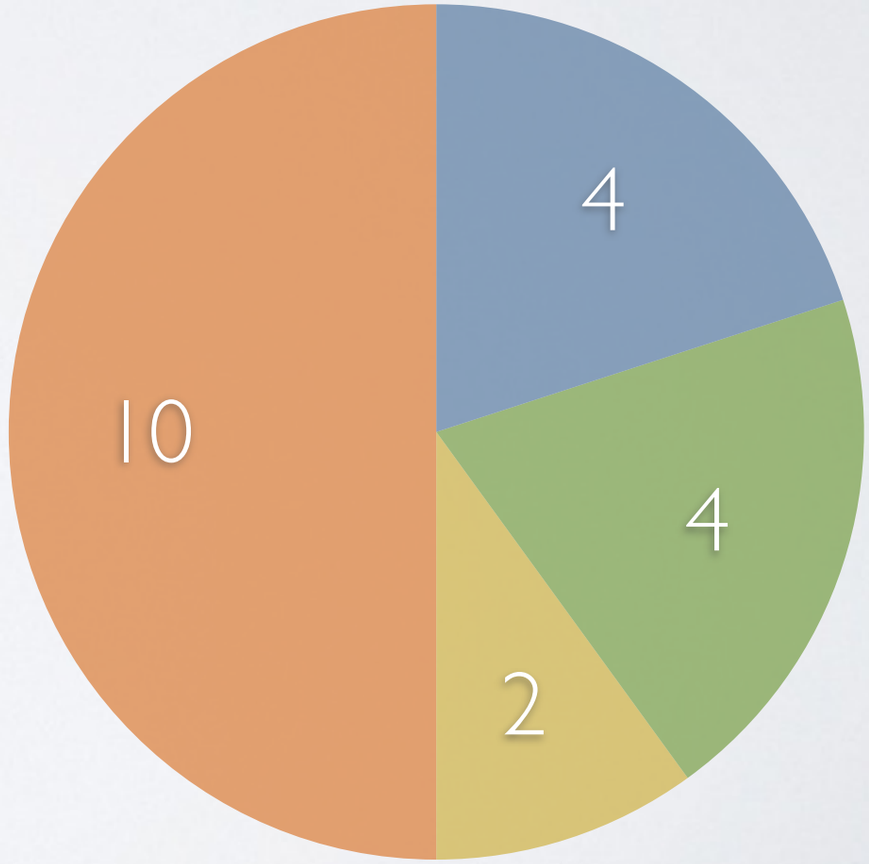
- Of 29 with details, 20 heap trespass errors



HEAP CVE BY VULNERABILITY TYPE

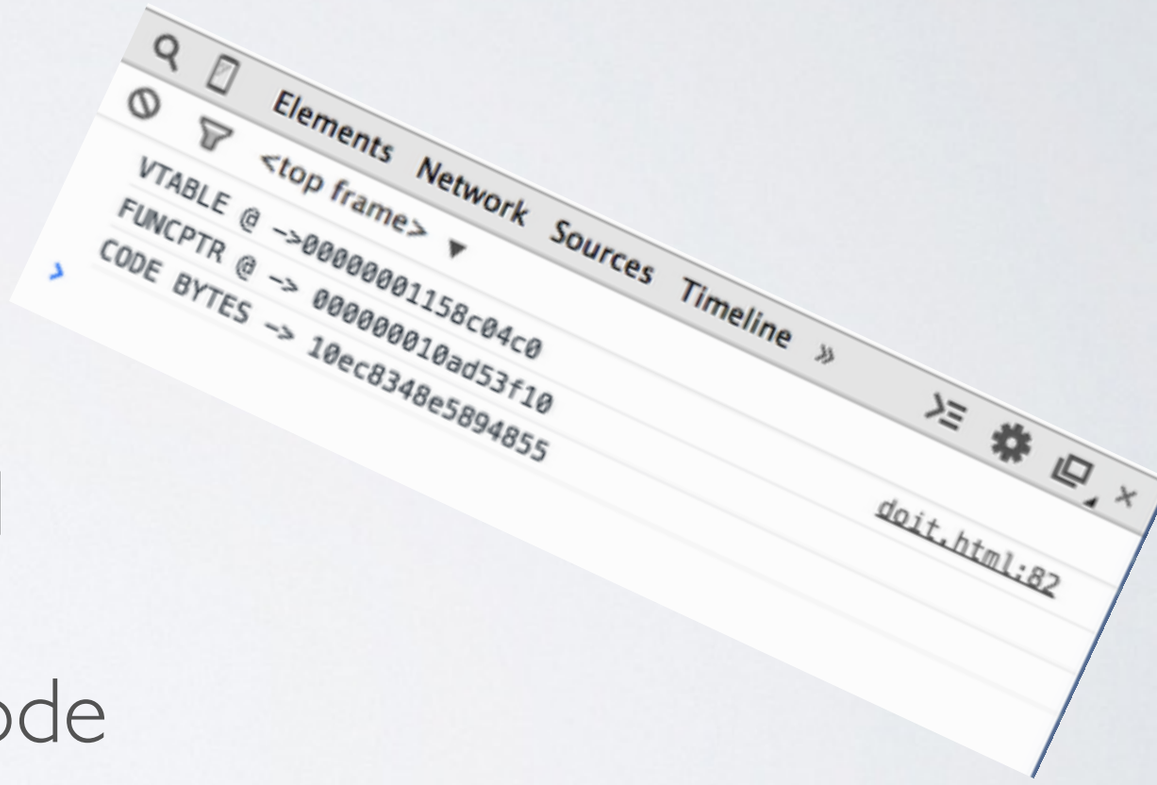
- Overflow Write
- OOB Read
- Type Confusion
- Use After Free

- Half were use after frees



HOW ARE THESE BUGS EXPLOITED?

- UaF - have a dangling reference.
- Point vtable to fake vtable
  - Entries point to attacker payload
- Trigger object, execute arbitrary code



WHICH MEM TRESPASS MITIGATIONS EXIST?

- NX data
- ASLR
- Isolated heaps
- Stack protection, canaries

ISOLATED HEAPS

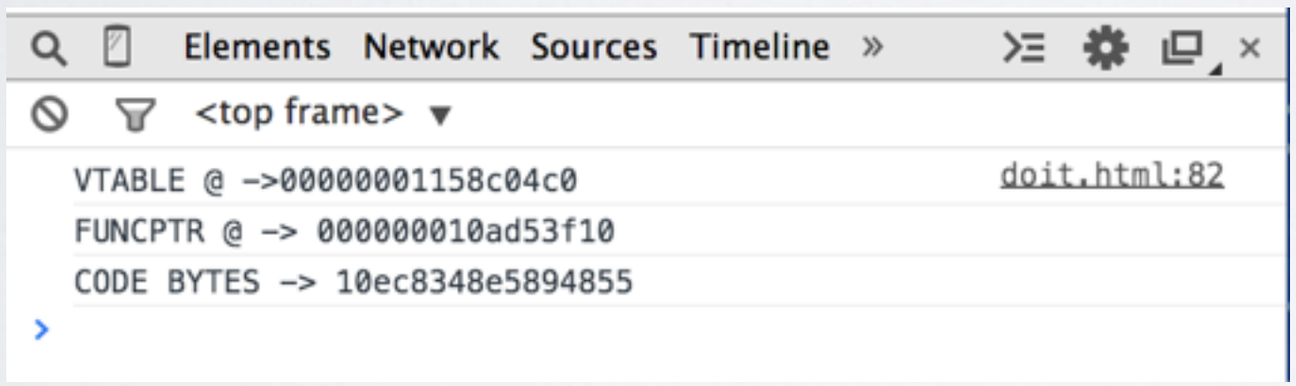
- great hardening against UaF
- some guard pages
- complicates exploit heap-grooming

VTABLE POINTERS

- All over the heap
- Each polymorphic class has one
- Contents of the vtable reveal code addresses

QUICK DEMO

- *[Chromium demo of relative->arb read/write]*



Click Me

```
[0]+ 0x000000011c5364c0
[1]+ 0x699081c100000001
[2]+ 0x00007fda699081c1
[3]+ 0x0000000100007fda
[4]+ 0xab00000000000001
[5]+ 0x001c101cab000000
[6]+ 0xabababab001c101c
[7]+ 0x00000000abababab
[8]+ 0x0000000000000000
[9]+ 0x5b20b36800000000
[10]+ 0x000015895b20b368
[11]+ 0x0000000000001589
[12]+ 0x0000000000000000
[13]+ 0x0000000000000000
[14]+ 0x0000000000000000
[15]+ 0x0000000000000000
[16]+ 0x0000000000000000
[17]+ 0x0000000000000000
[18]+ 0x0000000000000000
[19]+ 0x0000000000000000
[20]+ 0x0000000000000000
[21]+ 0x47439a3000000000
[22]+ 0x000009ed47439a30
[23]+ 0x00000000000009ed
[24]+ 0x0000000000000000
[25]+ 0x1c536948000000000
VTABLE @ -> 0x000000011c5364c0
FUNCPTR @ -> 0x00000001119c9e90
CODE BYTES : -> 0x10ec8348e5894855
```

DO THEY WORK?

WHY NOT?

- "JS is a general purpose programming language for laying out the heap address space"

- Information leaks reveal content and memory layout
- Most heap vulnerabilities can be used to read memory and craft arbitrary r/w
- ROP bypasses NX data

WHAT CAN WE DO ABOUT IT?

- Focus for this presentation:
  - Heap memory trespass errors
  - Majority of the bug fixes in chrome for the past 3 months
- Not in focus: JIT, stack-memory trespass, others

ASSUME ATTACKERS HAVE AN ARBITRARY READ/WRITE PRIMITIVE

- Relative to the heap
- Absolute
- Current mitigations do not try to defend against this assumption, which is largely true.

EXPLOIT ECONOMICS 101

- ~~Value = Lifetime x Reliability~~

$$value(exploit) = \begin{cases} 6 figures & exploit\ never\ fails \\ \text{☹} & otherwise \end{cases}$$

- Exploit mitigations may only need to work some noticeable amount of the time to deter exploitation

WHAT IS THERE TO BE DONE?

- Make locating code for ROP hard
  - Make failing to locate code game-over
- Make reading memory via undefined behavior perilous & unreliable

EXPLOIT MITIGATION ECONOMICS

- The more invisible the better
- Acceptable performance impact
- Can't break tools (debuggers, diagnostics,…)
- ~~Can't break ABI~~ (too much)

PERFORMANCE REQUIREMENTS

- Micro-benchmarks
- Real-world benchmarks
- Memory limitations
- 5% hard upper performance limit

The Ads Must Flow
(Dune 1984)

HOW CAN WE MAKE FINDING CODE FOR ROP HARD

Use 64-bit

Turn the heap into a minefield

Distribute multiple builds

Develop pointer subterfuge

64-BIT BUILDS

- High entropy ASLR
- Can't Spray & Pray anymore
- Javascript number limitations make info leaks harder
  - Integer attributes truncate full addresses
- On Windows, 64-bit results in opt-in security features
- Browser vendors have already done the hard work

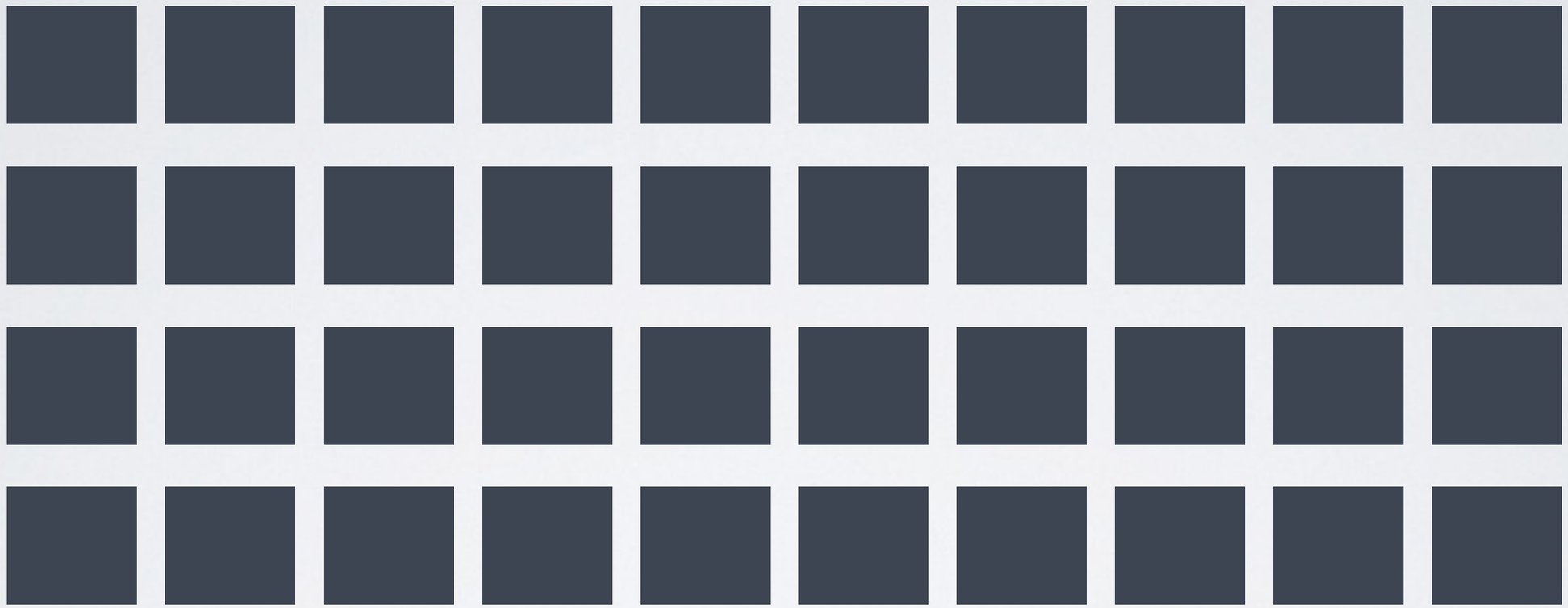
HOW CAN WE MAKE FINDING CODE HARD

- Use 64-bit
- **Turn the heap into a minefield**
- Distribute multiple builds
- Develop pointer subterfuge

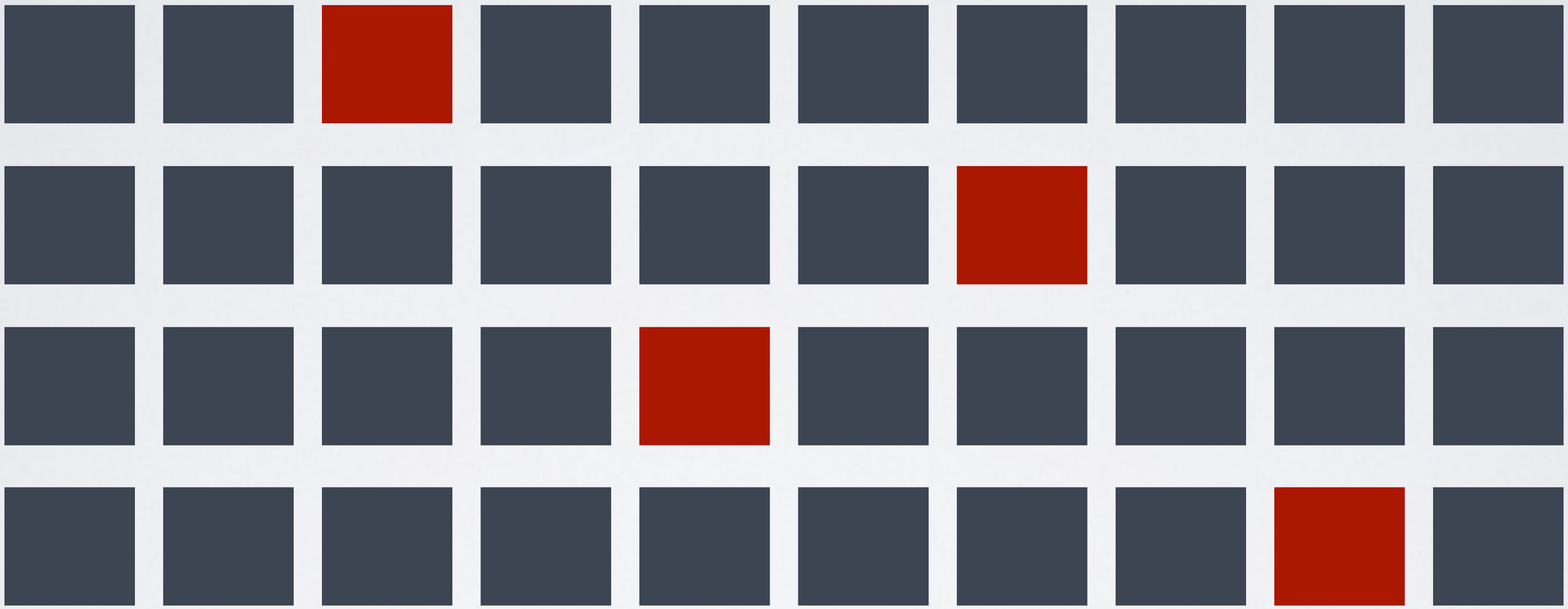
TURN THE HEAP INTO A MINEFIELD

- Use more guard pages
- Randomized spacial locality
- Windows 8 heap already does this
- Increases probability of a memory trespass being adjacent to a guard page

PHASE 1: RANDOM GUARD PAGES



PHASE 1: RANDOM GUARD PAGES



PHASE 2: RANDOMIZE ALLOCATION LOCALITY

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

3	14	5	7
9	6	16	13
1	11	4	10
8	15	2	12

HOW CAN WE MAKE FINDING CODE HARD

Use 64-bit

Turn the heap into a minefield

Distribute multiple builds

Develop pointer subterfuge

STOPPING ROP WITH CODE RANDOMIZATION

- Already have OS-Provided ASLR
- Load-time code shifting / IDR
- Install-time code isomorphism
- Build-time code isomorphism

	Breaks Tools	*Performance Impact*	*Implementation Complexity*
Load Time Code Shifting	**Yes**	High impact upon loading	Requires OS support, compiler support, breaks signatures
Install Time Code Isomorphism	**Yes++**	High impact upon installation	Requires compiler support, breaks signatures
Build Time Code Isomorphism	No	None	Requires compiler support

BUILD-TIME CODE ISOMORPHISM

- Apply polymorphic techniques to obtain isomorphic code
- Produce & distribute multiple builds
- ROP requires identifying which version is loaded

DETAILS OF ISOMORPHIC TRANSFORMS

- Randomize register assignment, instruction ordering, equivalent operations, etc
- Transform requirements:
  - Ensure no function offset differences, since they would fingerprint sub-build

BUILD RANDOMIZATION FEASIBILITY

- Compare sub-builds to prevent shared ROP gadgets
- UUIDs sufficient for debug tools
- Build engineering overhead: build times, signing
- Test engineering overhead
- Crash binning & crash deduplication overhead

HOW CAN WE MAKE FINDING CODE HARD

Use 64-bit

Turn the heap into a minefield

Distribute multiple builds

Develop pointer subterfuge

POINTER SUBTERFUGE

- Add uncertainty when an attacker follows a pointer
- Makes identifying the sub build—i.e., gadgets— difficult

THE ZEN OF POINTER SUBTERFUGE

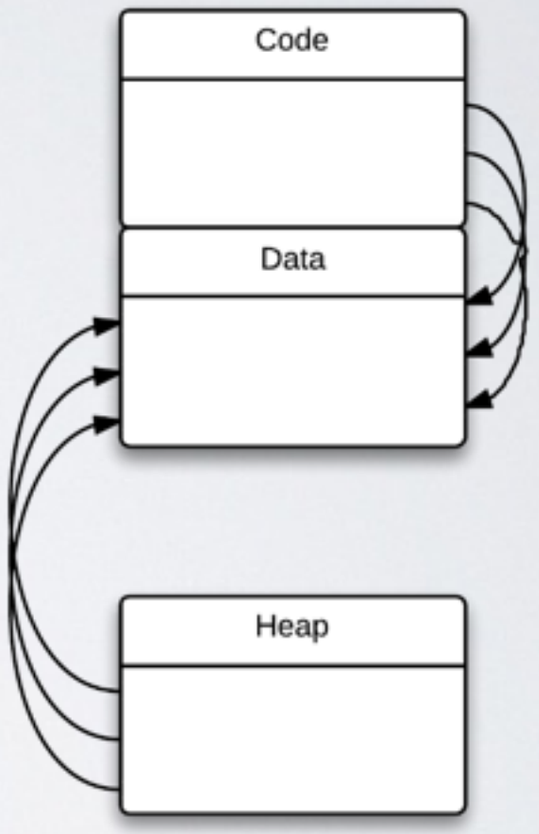
- Decouple data pointers and code pointers
- Class struct mangling
- Fake pointers in vtables, fake function pointers

THE CODE AND DATA CONNECTION

- Code and data sections are adjacent

- Data pointers

  - String constants, globals, addresses of vtable



POTENTIAL SOLUTIONS FOR NEUTERING DATA POINTER LEAKS

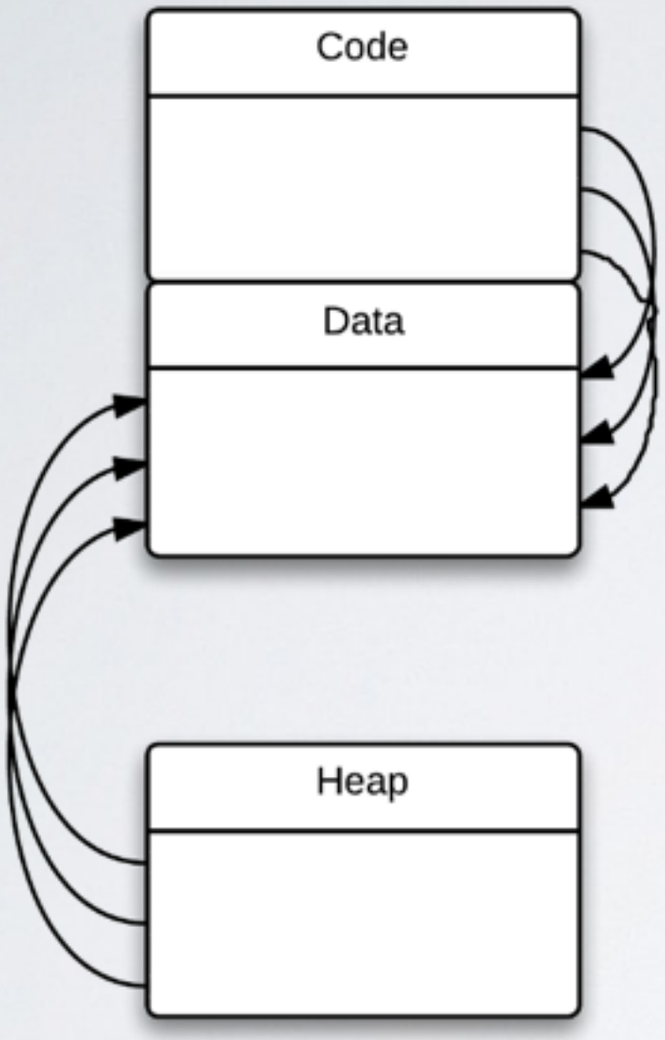
- Indirection
- Gap randomization
- Alternative: execute only code

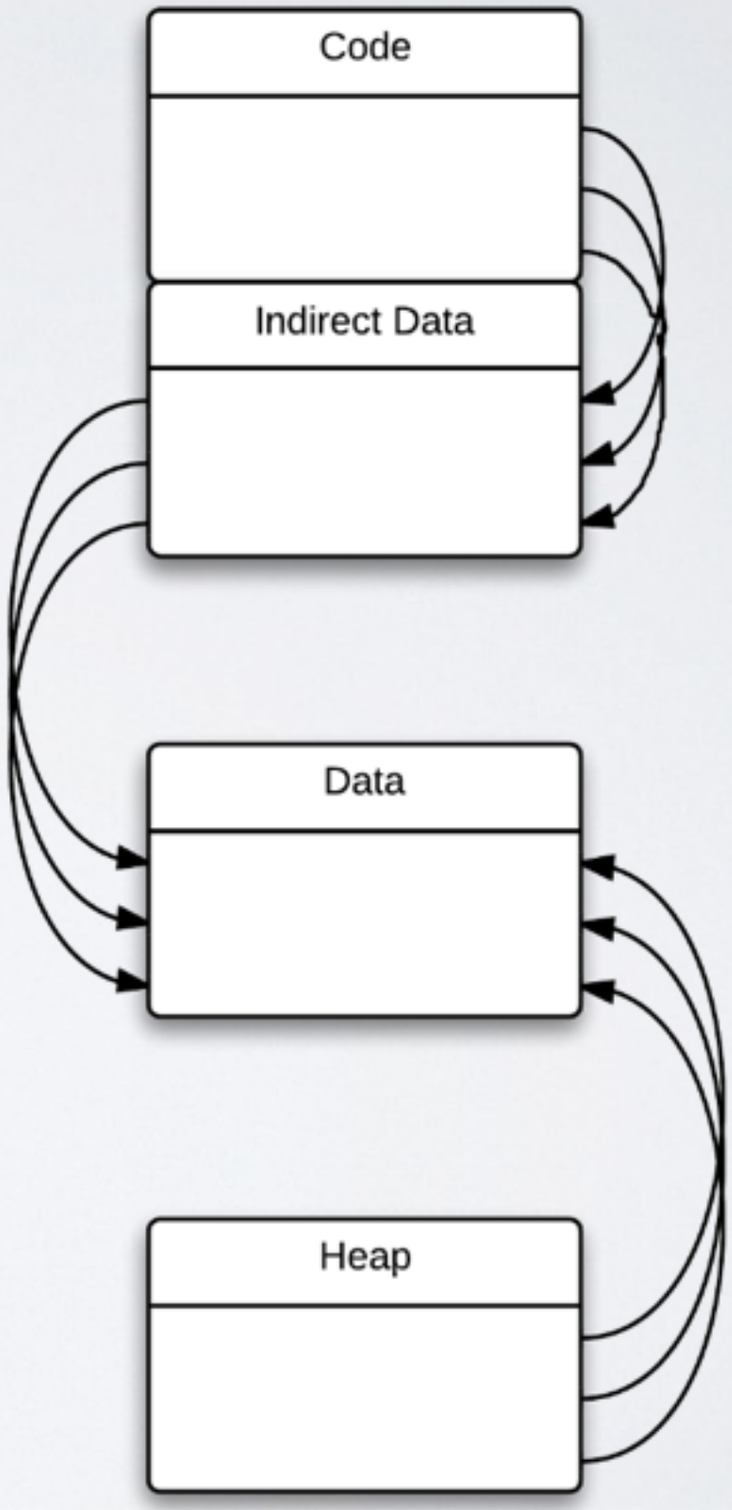
  http://scarybeastsecurity.blogspot.com/2014/06/execute-without-read.html

INDIRECTION

- With loader support, data segment loaded at a random location
- An indirection table is referenced relative to code
- Requires loader support, and runtime support

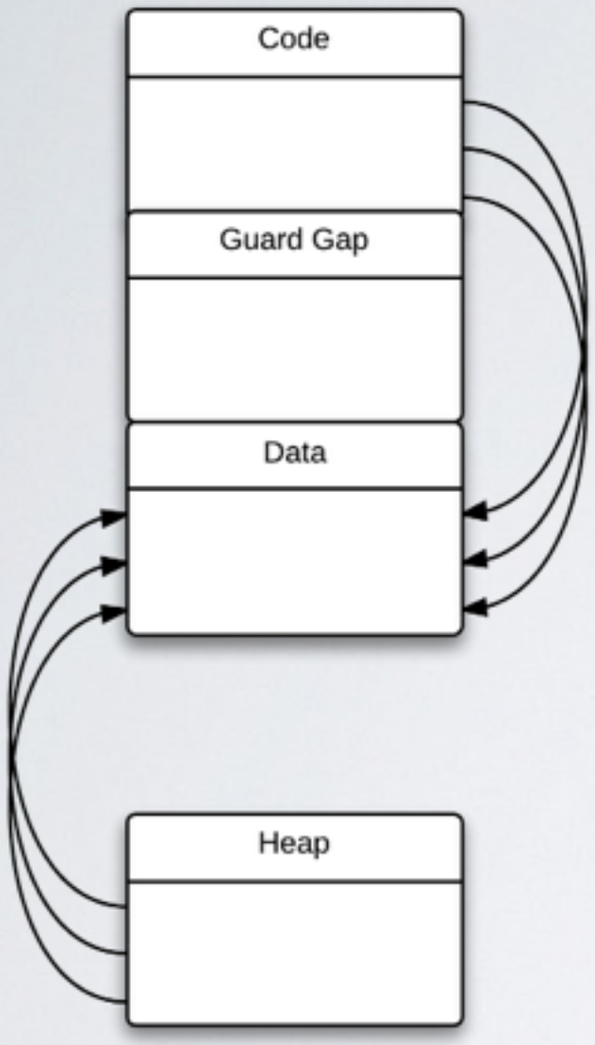
INDIRECTION

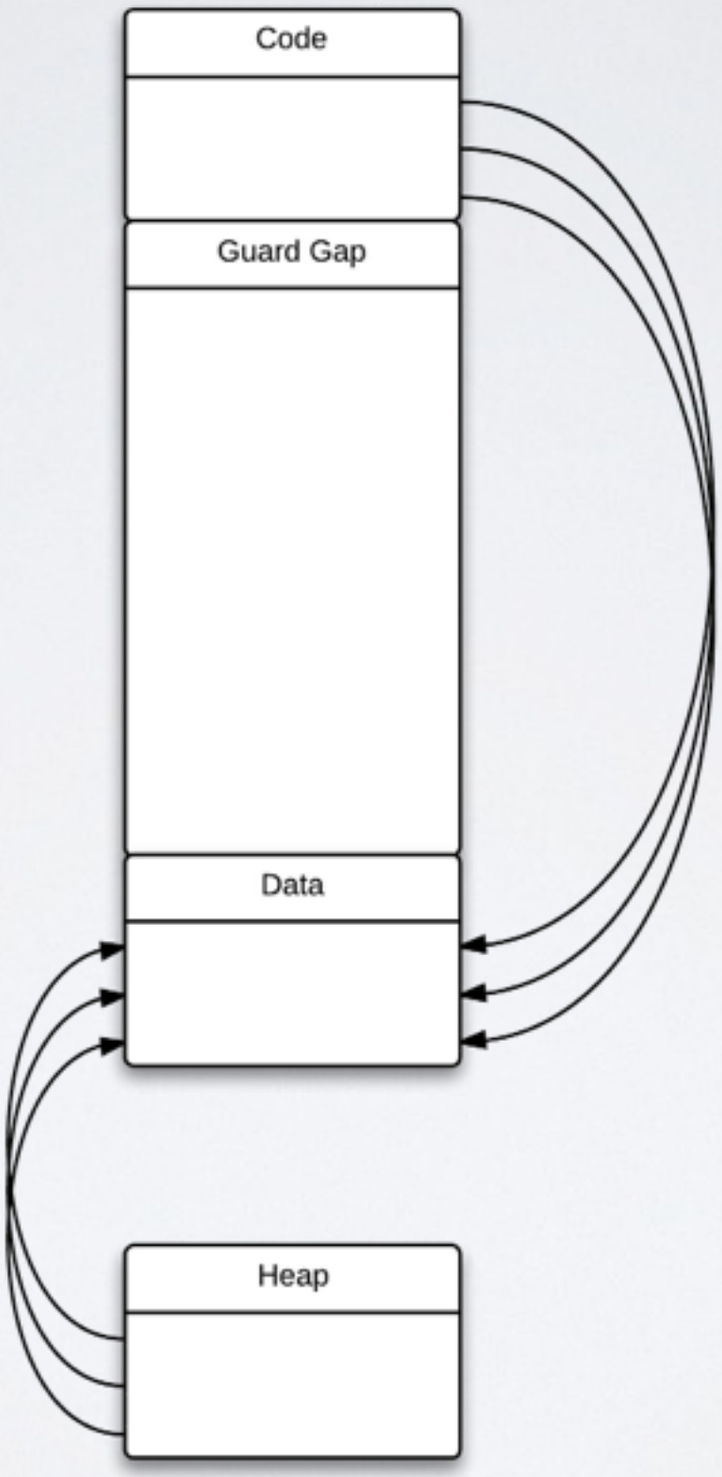


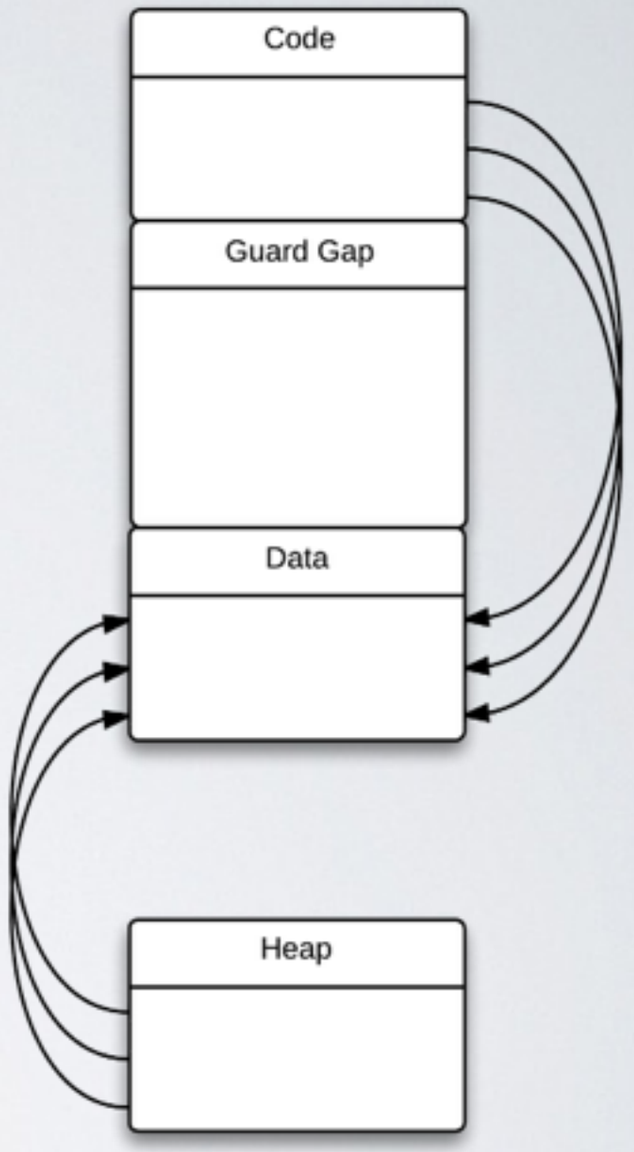


GAP RANDOMIZATION

- Per-build randomization of space between code and data sections
- Gap granularity must be equivalent to ASLR slots for module
- No linker, runtime support needed

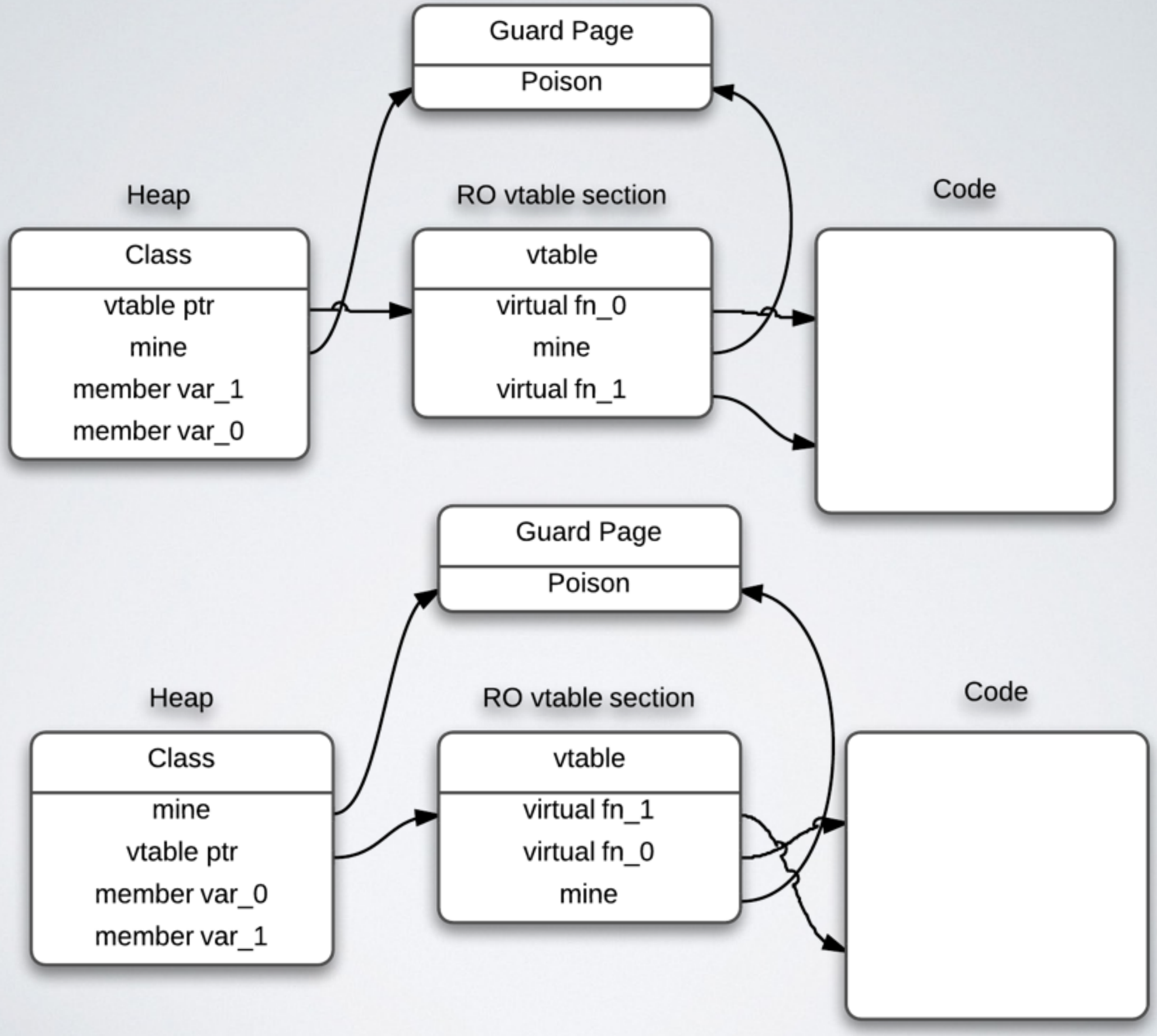

Code
Guard Gap
Data
Heap


Code
Guard Gap
Data
Heap


Code
Guard Gap
Data
Heap

CLASS STRUCT MANGLING

- Add fake vtable pointers
- Add fake function pointers
- Shuffle member variables


Guard Page
Poison
Heap
RO vtable section
Code
Class
vtable ptr
mine
member var_1
member var_0
vtable
virtual fn_0
mine
virtual fn_1
Guard Page
Poison
Heap
RO vtable section
Code
Class
mine
vtable ptr
member var_0
member var_1
vtable
virtual fn_1
virtual fn_0
mine

CLASS STRUCT MANGLING FEASIBILITY

- Can not be used for code for or from dynamic libraries
- Offset differences need to be propagated to debug symbols
- Noticeable memory overhead from paired pointers
- Negligible performance impact

FAKING POINTERS

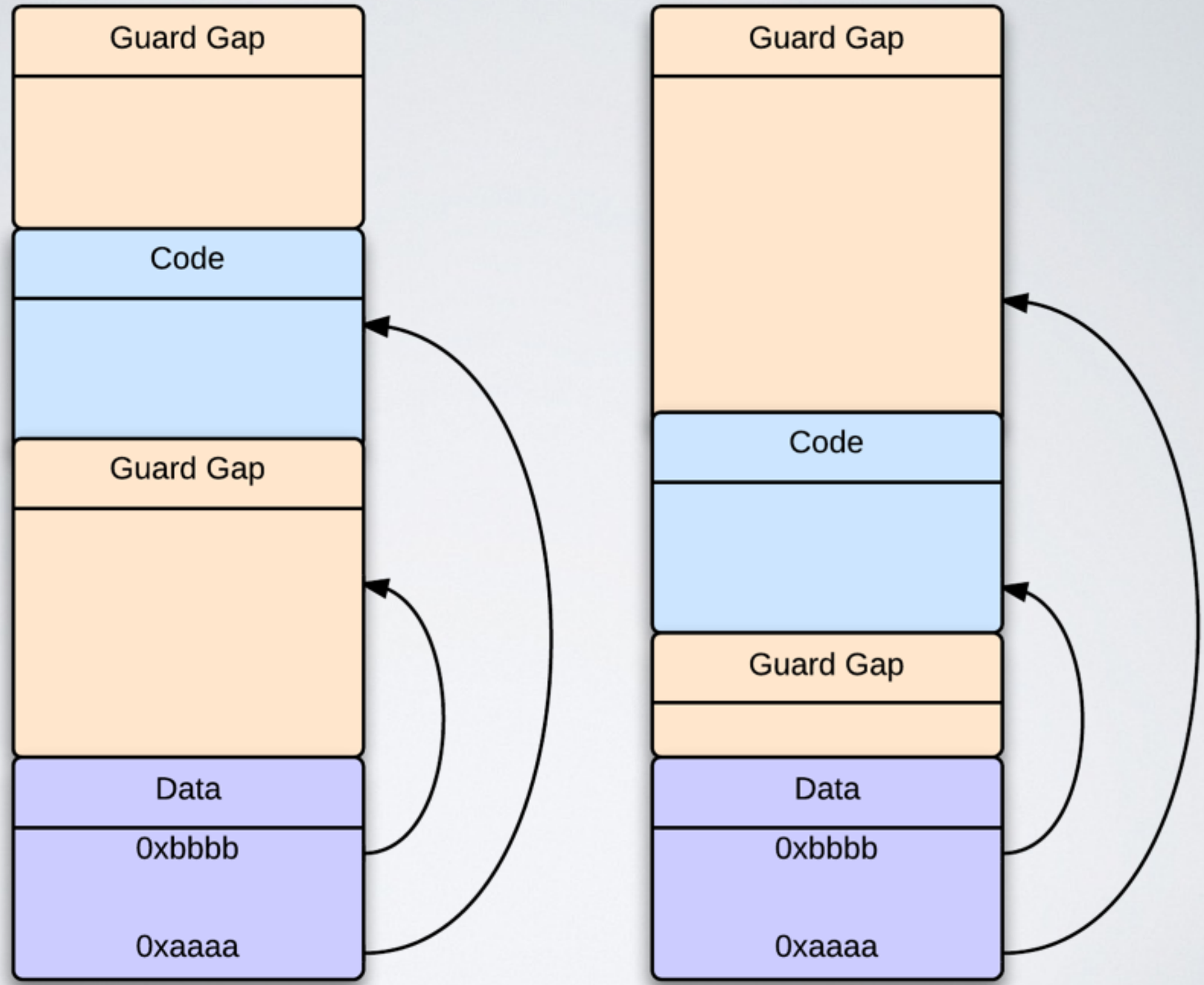
- Randomly generated fake pointers won't fool an attacker
- Heap sampling can distinguish real addresses from fake ones

IMPLEMENTING FAKE POINTERS

- Leverage build randomization & gap randomization

- Fake pointers are identical to real pointers from other builds

- Fake pointers land in guard pages in one subset of builds, but the real code in the other subset


Guard Gap
Code
Guard Gap
Data
0xbbbb
0xaaaa
Guard Gap
Code
Guard Gap
Data
0xbbbb
0xaaaa

FAKE POINTER FEASIBILITY

- Performance & debug impact: no additional changes from class struct randomization

MITIGATIONS SUMMARY

Use 64-bit

Turn the heap into a minefield

Distribute multiple builds

Develop pointer subterfuge

IMPACT OF MITIGATIONS ON A TYPICAL EXPLOIT

- Uncertainty in reading heap
  - Could hit a guard page more often for certain vulnerabilities
- Code base insufficient, need sub-build identification
- Uncertainty in dereferencing function pointers to build gadgets

NEXT STEPS

- LLVM Work

- Preparing a set of Chromium builds

FURTHER WORK

- Guard dynamic branches
- Additionally protect external code, heaps mapped in the address space
- OS support for execute-only code
- OS support for zero-overhead guard pages

REFERENCES

Bugs

http://support.apple.com/kb/HT6293

http://support.apple.com/kb/HT6254

http://googlechromereleases.blogspot.com/search/label/Stable%20updates

https://github.com/miaubiz/let-me-see-you-scrape

Exploits

https://labs.mwrinfosecurity.com/system/assets/538/original/mwri_polishing-chrome-slides-nsc_2013-09-06.pdf

https://labs.mwrinfosecurity.com/blog/2014/06/20/isolated-heap-friends---object-allocation-hardening-in-web-browsers/

https://code.google.com/p/chromium/issues/detail?id=352369

http://scarybeastsecurity.blogspot.kr/2013/02/exploiting-64-bit-linux-like-boss.html

Research & Ideas

https://twitter.com/grsecurity/status/427180045585481728 (RANDSTRUCT)

http://scarybeastsecurity.blogspot.com/2014/06/execute-without-read.html

https://twitter.com/comex/status/474656633281196032

http://files.accuvant.com/web/files/AccuvantBrowserSecCompar_FINAL.pdf

https://www.utdallas.edu/~zxl111930/file/DIMVA09.pdf

감사합니다

EXTRA SLIDES

- following, more things

NOTE ON FUNC METHOD REUSE

- Stale values in registers pointing to stack could leak via reusing incorrect function methods

FIXED POINT CALL SITES

- Stack return addresses mitigated

- [diagram]

DYNAMIC BRANCH PROTECTION

- Can't hijack most function pointers, must be in protected regions.

- http://scarybeastsecurity.blogspot.com/2014/06/execute-without-read.html