TOO LEJIT TO QUIT

EXTENDING JIT SPRAYING TO ARM

WILSON LIAN, HOVAV SHACHAM, STEFAN SAVAGE
UC SAN DIEGO

SOFTWARE HAS BUGS

SOFTWARE HAS CONTROL FLOW VULNS

DEP

ENFORCES SEPARATION OF CODE AND DATA

ASLR

RANDOMIZES LOCATIONS OF SEGMENTS IN MEMORY

	-DEP	+DEP
-ASLR	Stack Smashing	Return-Oriented Programming
+ASLR	Heap Spraying	JIT Spraying

JIT Spraying

- Dion Blazakis, BlackHat DC 2010
- ActionScript (Flash Player) JIT on x86
- Specially-crafted ActionScript input
- Encode instructions in constants
- Execute JIT code from unintended offset

JIT Spraying (x86)

```
var x = (0x3c909090 ^ 0x3c909090 ^ 0x3c909090 ^ …);
```

mov eax, 3c909090h	xor eax, 3c909090h	xor eax, 3c909090h	…

NOP	NOP	NOP	cmp al, 35h	NOP	NOP	NOP	cmp al, 35h	NOP	NOP	NOP	cmp al, 35h

NOP	NOP	NOP	cmp al, 35h	NOP	NOP	NOP	cmp al, 35h	NOP	NOP	NOP	cmp al, 35h	…

…	pop esi	xor edx, edx	cmp al, 35h	push edx	push esi	NOP	cmp al, 35h	mov ecx, esp	NOP	cmp al, 35h	…

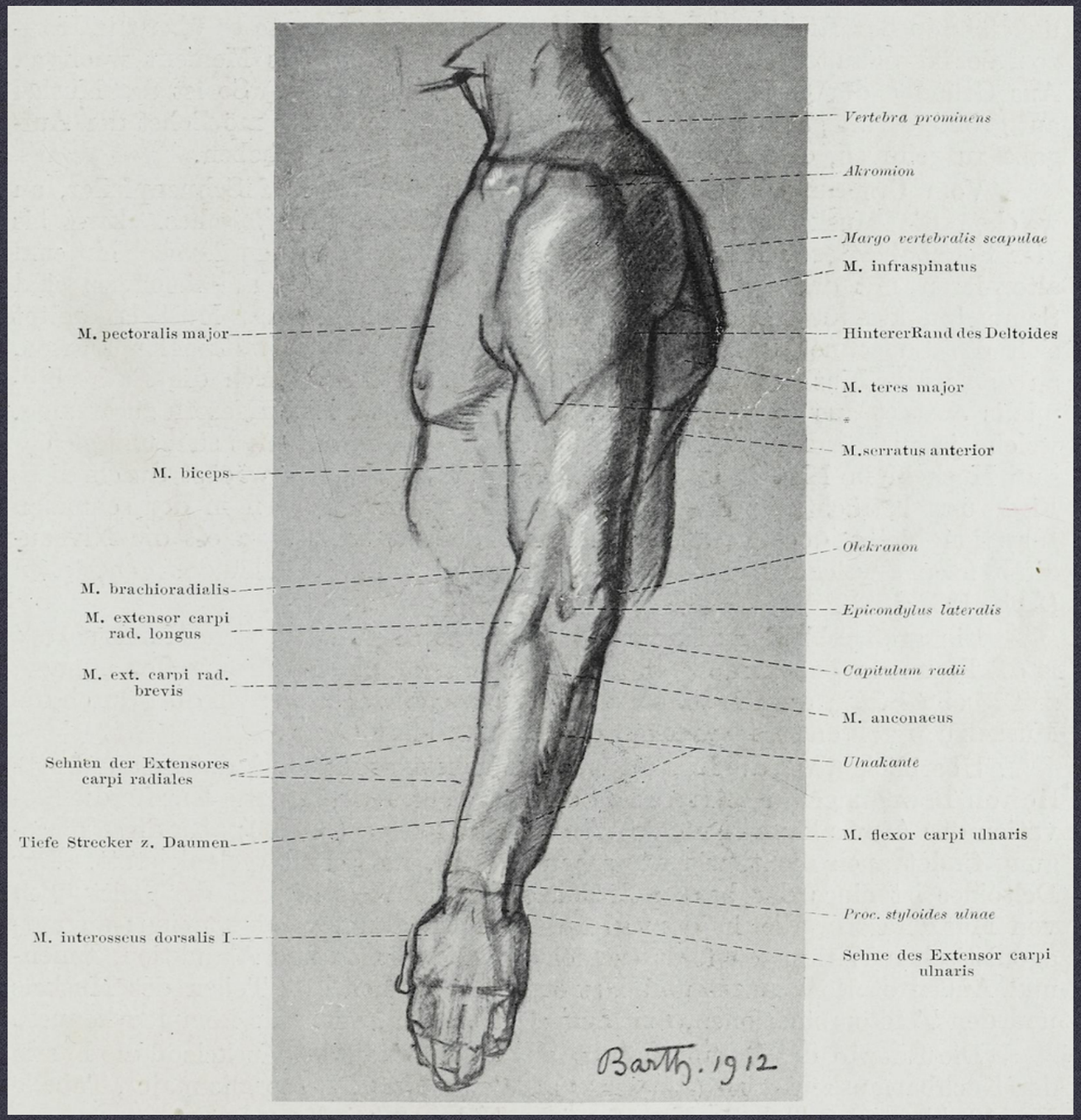
Is JIT spraying limited to x86?

- Variable-length, unaligned instructions
- 32-bit immediates encoded as 4 consecutive bytes

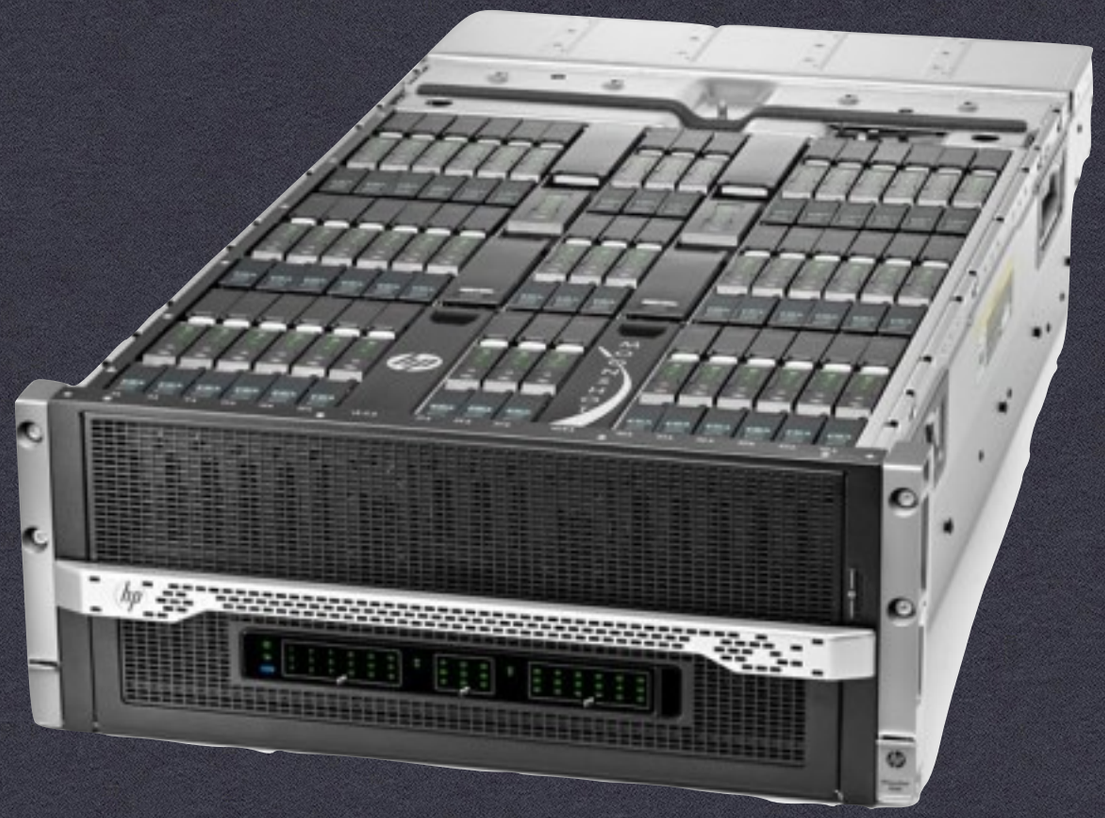
Contributions

- Show that RISC architectures are vulnerable to JIT spraying
- Gadget chaining: augmenting high level code with unsafe computation as a callable primitive
- PoC JIT spray against JavaScriptCore on ARMv7-A

ARM Architecture



ARM Architecture



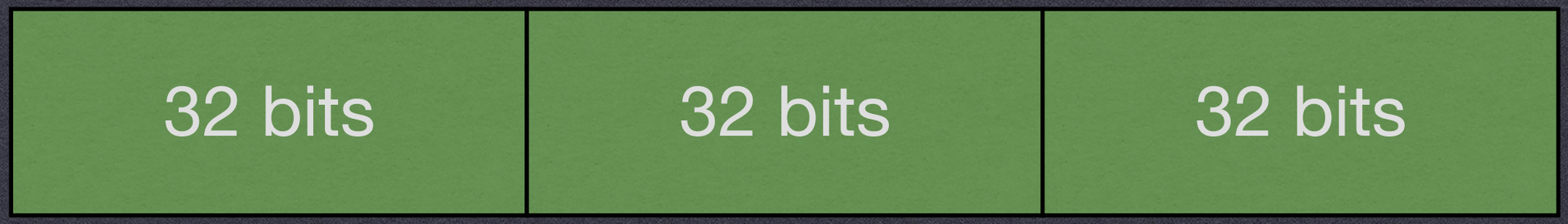
ARM Architecture

- Fixed-width(ish), aligned(ish) instructions
  - ARM: 32-bits wide, 4-byte aligned
  - Thumb: 16-bits wide, 2-byte aligned
  - Thumb-2: Mixed 16/32-bits wide, 2-byte aligned

CAN WE JIT SPRAY ON ARM JUST LIKE x86?

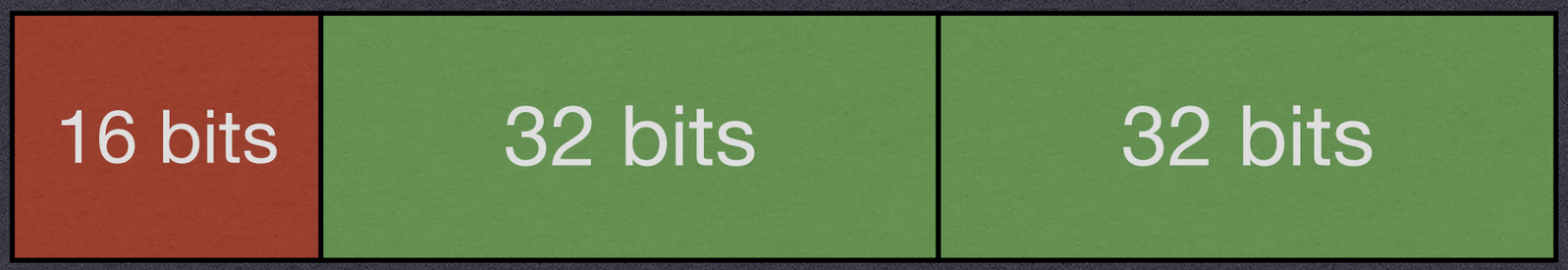
The Resynchronization Problem

What you start with:



What you want:

What you get:



JAVASCRIPT IS TURING-COMPLETE

BUT JAVASCRIPT IS ALSO MEMORY SAFE

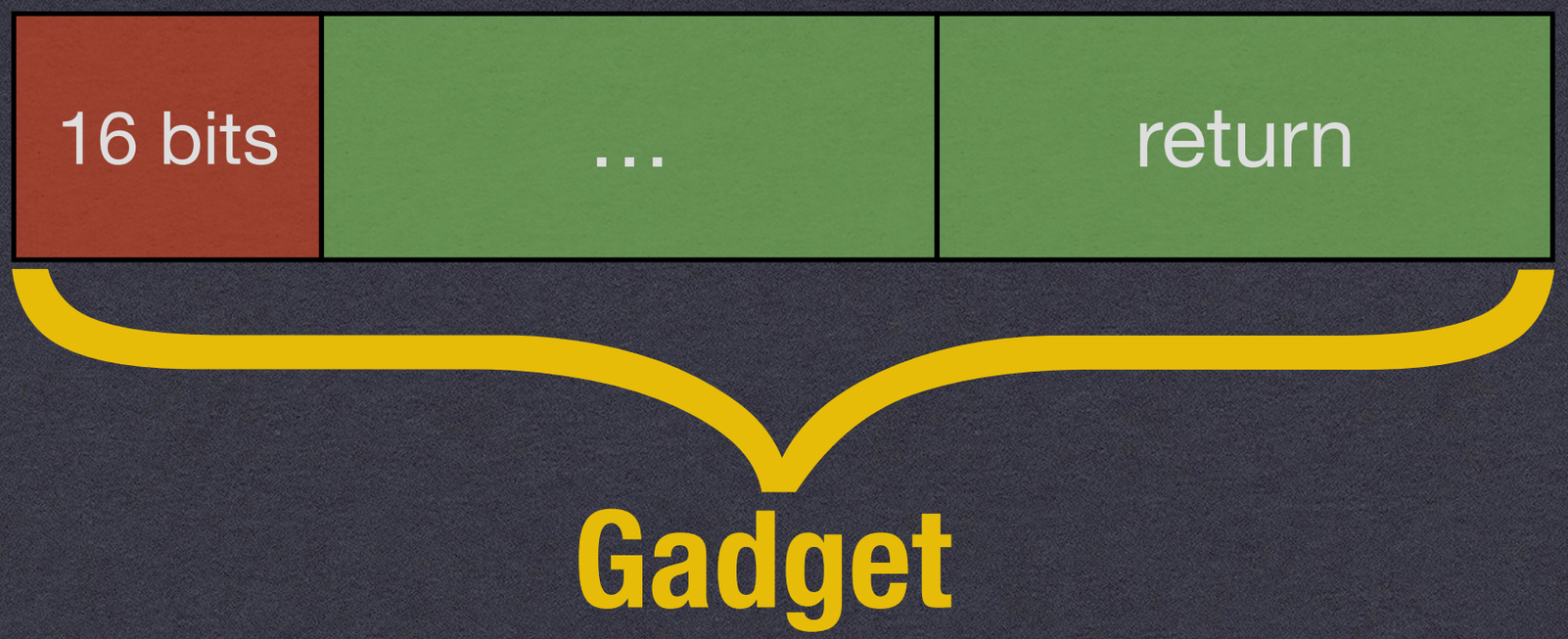
LET'S COMBINE JAVASCRIPT AND UNINTENDED INSTRUCTIONS

```
...
var baseAddr = getObjectAddress(obj);
for (var i = 0; i < objSize; i++) {
    var b = readMemByte(baseAddr + i);
    if (b & 0x3f)
        writeMemByte(baseAddr + i, 0xff);
}
...
```

Gadgets

Intended instructions:

Executed instructions:



```
...
var baseAddr = ctrlFlowVuln1(obj);
for (var i = 0; i < objSize; i++) {
    var b = ctrlFlowVuln2(addr + i);
    if (b & 0x3f)
        ctrlFlowVuln3(baseAddr + i, 0xff);
}
...
```

Address Disclosure Gadget

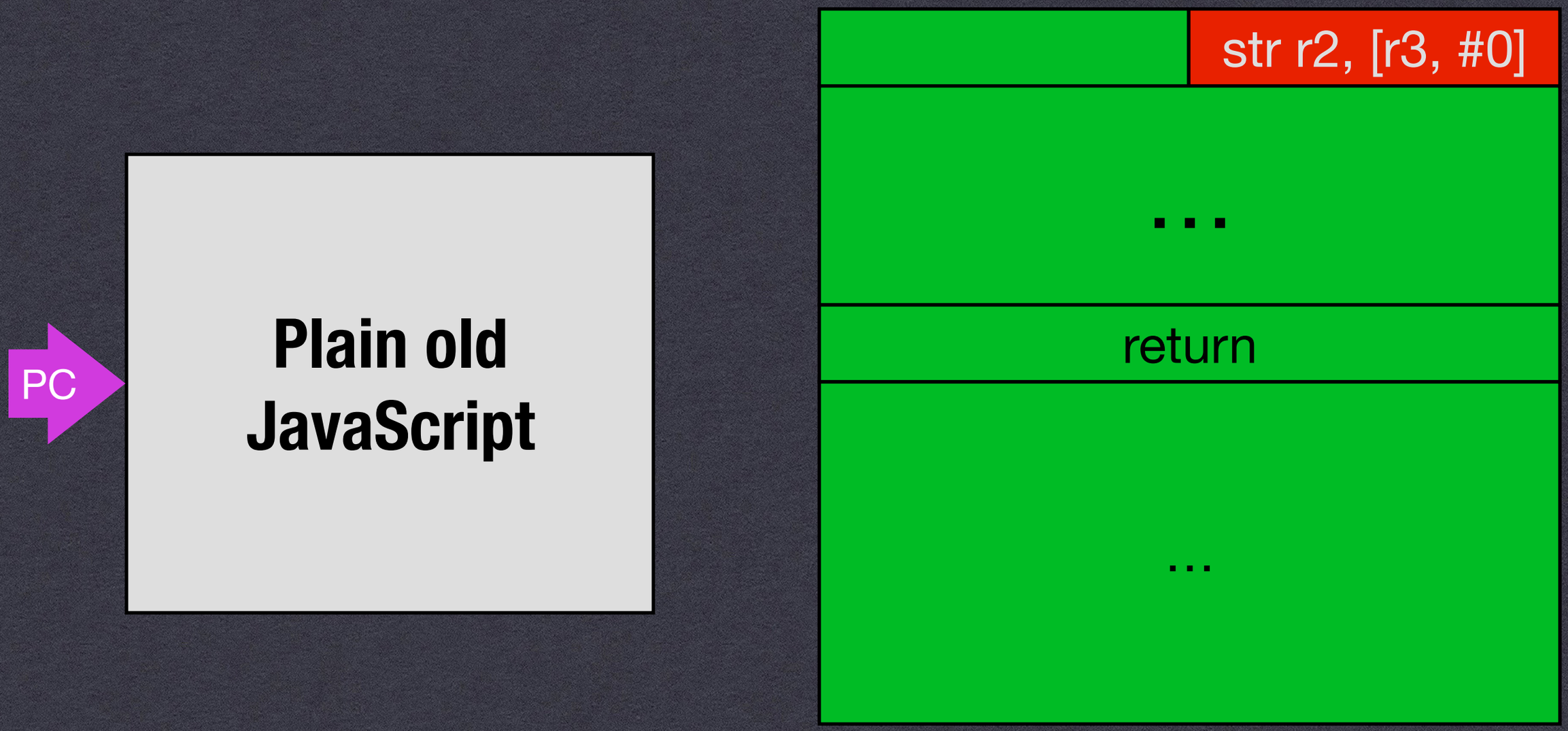
Memory Load Gadget

Memory Store Gadget

WE CALL THIS
GADGET CHAINING

USING A MEMORY-STORE GADGET, WE CREATED A PROOF OF CONCEPT JIT SPRAY AGAINST JAVASCRIPTCORE ON ARMV7-A

Store gadget chaining



Store gadget chaining

PC

Plain old JavaScript

str r2, [r3, #0]

. . .

return

Shellcode pt. 1

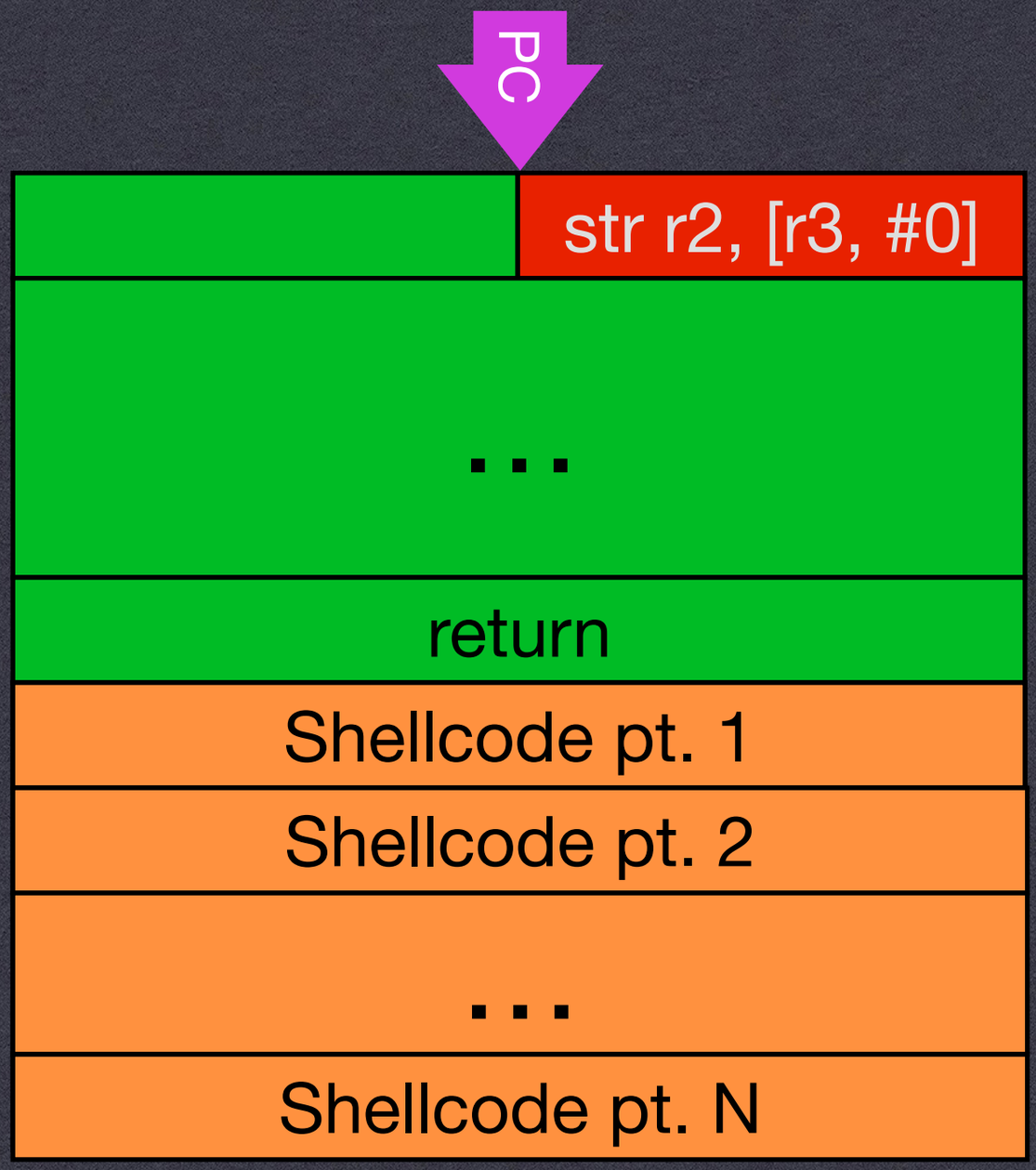
Shellcode pt. 2

. . .

Shellcode pt. N

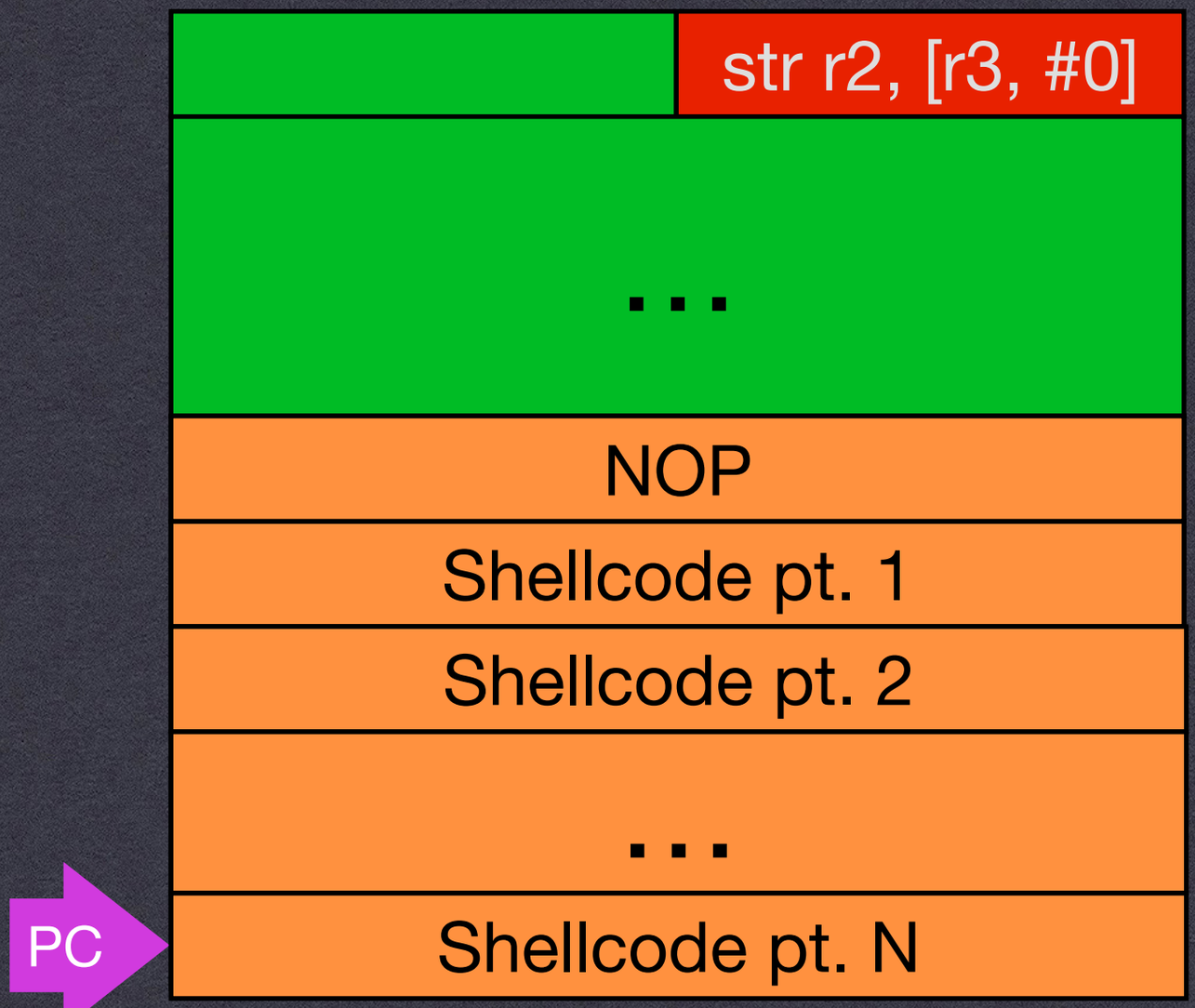
Store gadget chaining

Plain old JavaScript



Store gadget chaining

Plain old JavaScript



See paper for full details

- How do you…
  - reliably guess gadget addresses?
  - populate gadget argument registers when calling gadgets?
  - make sure you return from gadgets without crashing?

Conclusion

- JIT spraying is possible on RISC
- Gadget chaining
- Decouple safe computation from unsafe computation
- Unsafe computation on demand

Q & A

WILSON LIAN
WLIAN@CS.UCSD.EDU