the path to EL1 in iOS 11

@i41nbeer

Please expand the speaker notes if you’re reading this deck!

I find bugs, and sometimes exploit them

15 years ago...

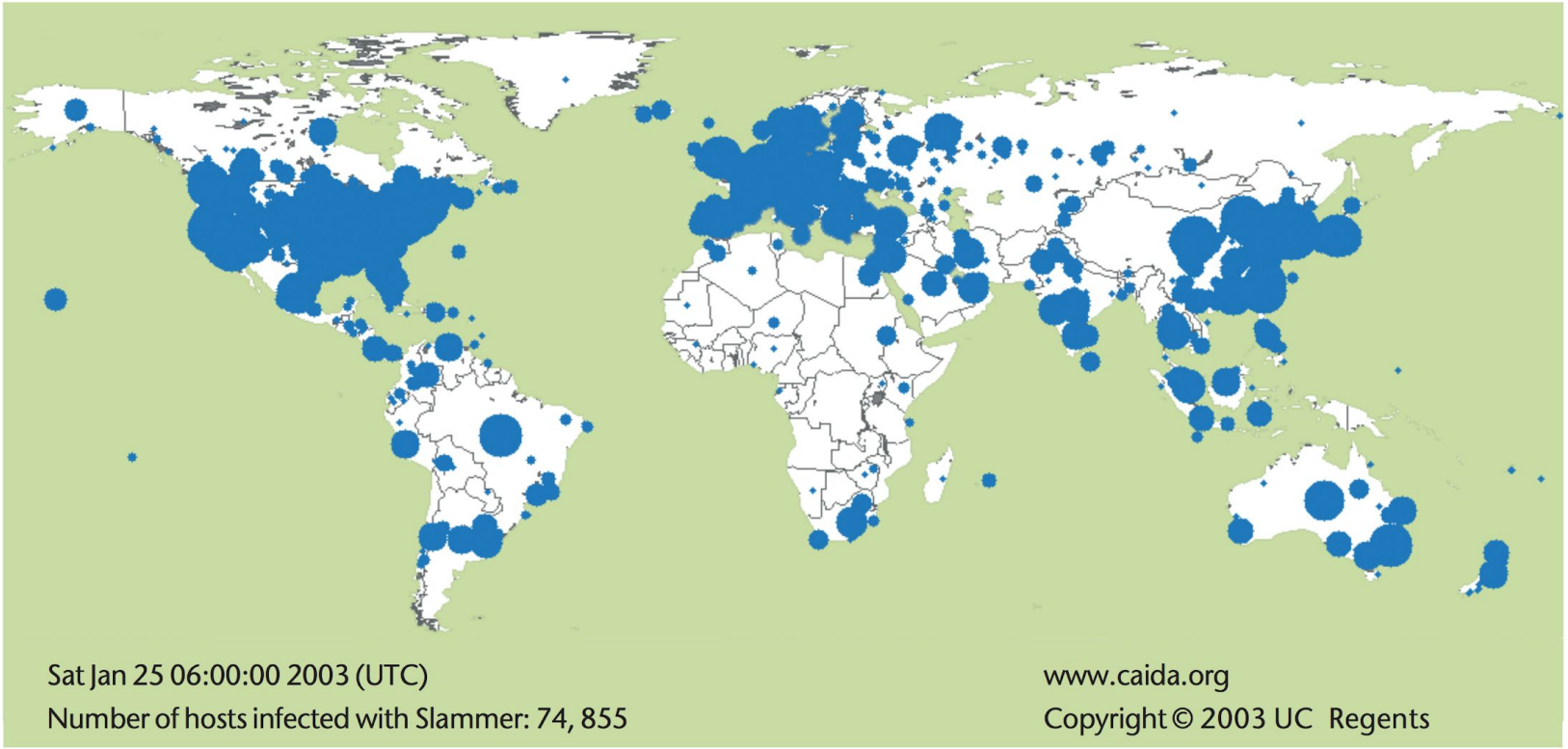


Figure 1. The geographical spread of Slammer in the 30 minutes after its release. The diameter of each circle is a function of the logarithm of the number of infected machines, so large circles visually underrepresent the number of infected cases in order to minimize overlap with adjacent locations. For some machines, we can determine only the country of origin rather than a specific city.

http://cseweb.ucsd.edu/~savage/papers/IEEESP03.pdf

```
char db_name[16];
...
strcpy(db_name, &udp_packet[1])
```

poor software development practices used to lead to CLEARLY VISIBLE harm

"Our new design approaches need to dramatically reduce the number of such issues that come up in the **software** that Microsoft, its partners and its customers create"

https://www.wired.com/2002/01/bill-gates-trustworthy-computing/

"At the same time, we're in the process of training all our developers in the latest secure coding techniques."

https://www.wired.com/2002/01/bill-gates-trustworthy-computing/

All my friends are using `strcpy`. But I'm not, because I understand how dangerous it is. They say I could protect myself, but I know that only avoiding `strcpy` is 100% effective.

And why would I take a chance like that?

natashenka.ca/strcpy

Canadian Joke Council

True Bugs Wait ♡

@natashenka

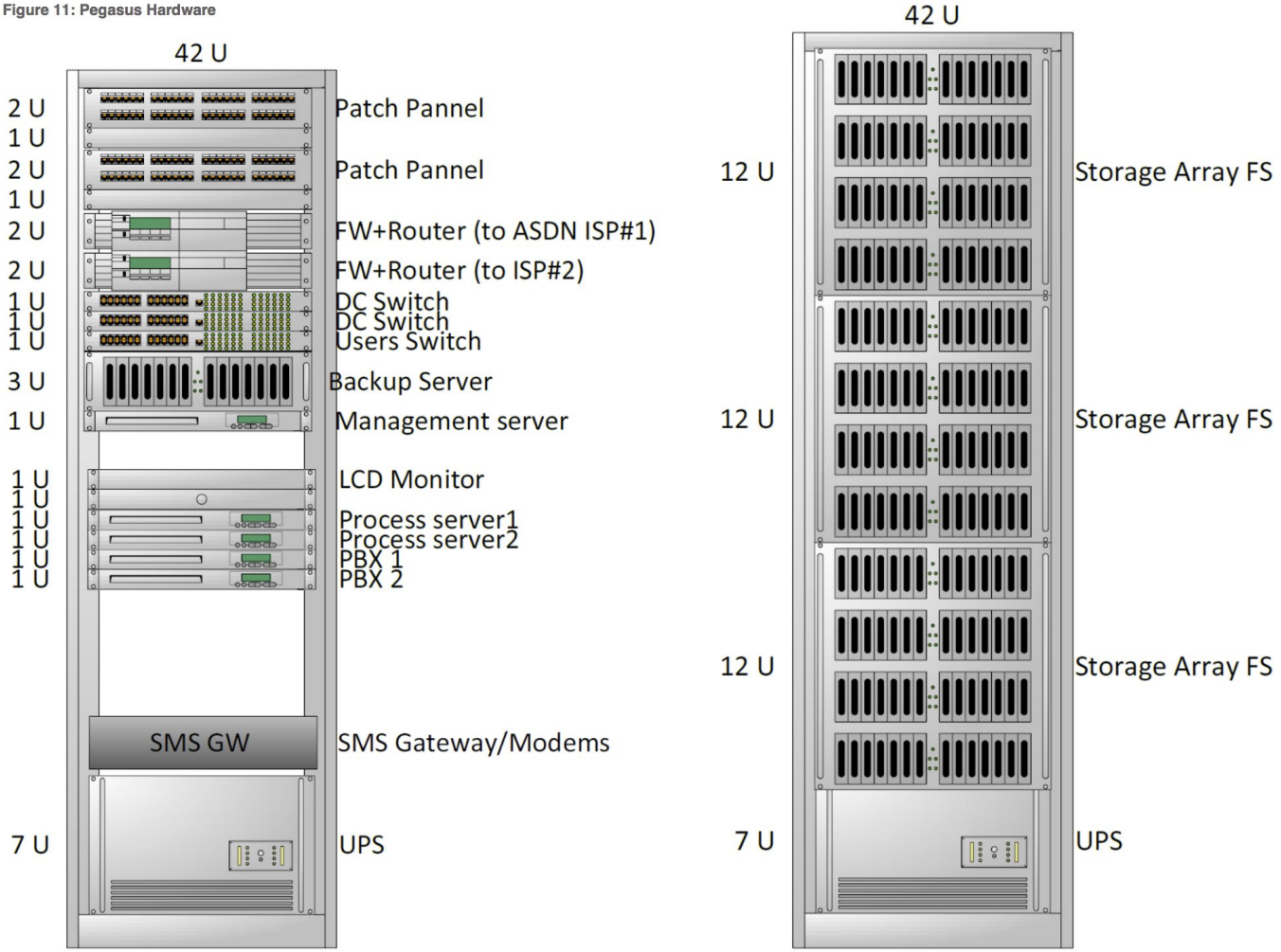
#truebugswait

"So now, when we face a choice between adding features and resolving security issues, we need to choose security."

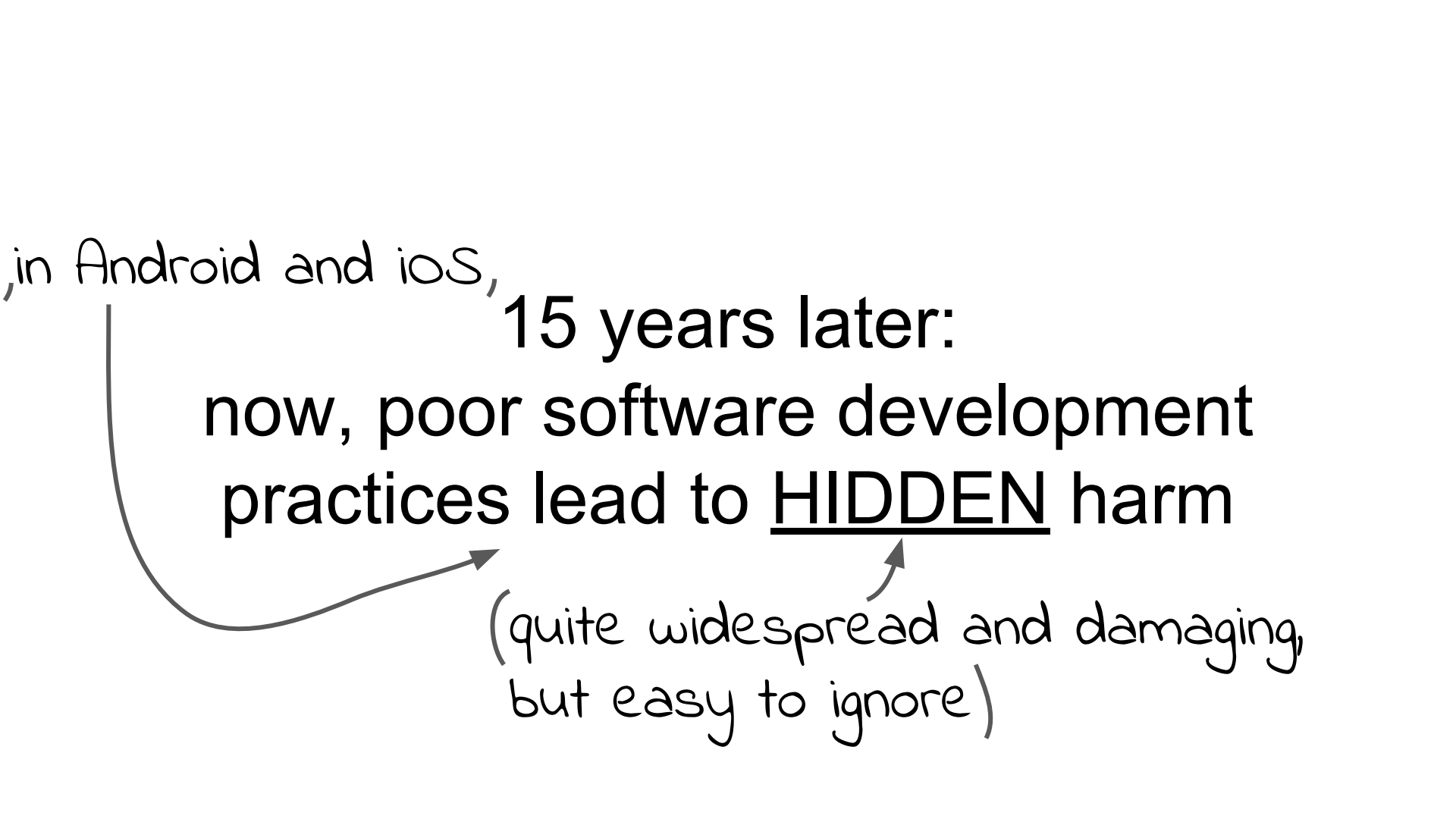
https://www.wired.com/2002/01/bill-gates-trustworthy-computing/

15 years later

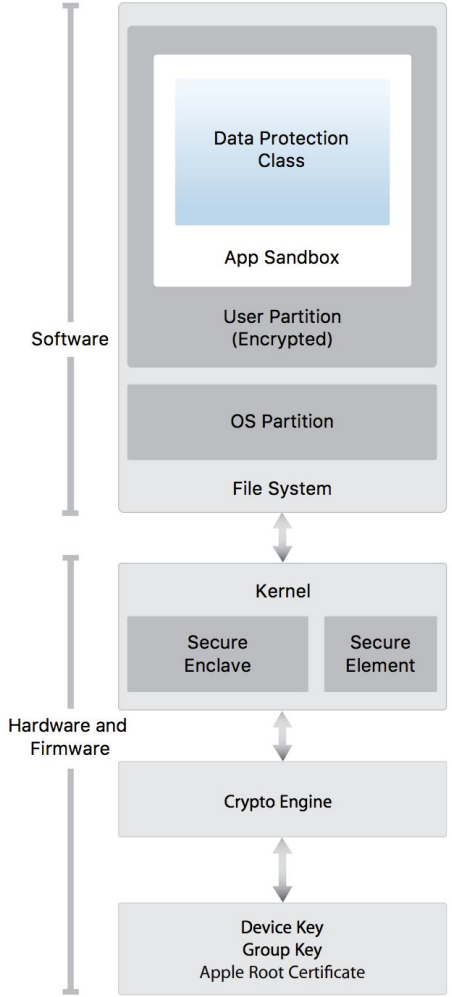
Figure 11: Pegasus Hardware



https://assets.documentcloud.org/documents/4599753/NSO-Pegasus.pdf


,in Android and iOS,
15 years later:
now, poor software development
practices lead to HIDDEN harm
(quite widespread and damaging,
but easy to ignore)

security systems design is making promises which poor software development practises cannot keep



https://www.apple.com/business/docs/iOS_Security_Guide.pdf

given sufficient bug density, security design is irrelevant

"feedback that we've heard pretty consistently, both from my red team at Apple and also from researchers directly, is that it's getting increasingly more difficult to find those most critical types of security vulnerabilities."

```
mach_voucher_extract_attr_recipe_trap(
  struct mach_voucher_extract_attr_recipe_args *args) {
  ipc_voucher_t voucher = IV_NULL;
  kern_return_t kr = KERN_SUCCESS;
  mach_msg_type_number_t sz = 0;

  if (copyin(args->recipe_size, (void *)&sz, sizeof(sz)))
    return KERN_MEMORY_ERROR;
.....
  uint8_t *krecipe = kalloc((vm_size_t)sz);
    if (!krecipe) {
      kr = KERN_RESOURCE_SHORTAGE;
      goto done;
    }

  if (copyin(args->recipe, (void *)krecipe, args->recipe_size)) {
```

```
if (dst->sa_family == AF_INET &&
    dst->sa_len != sizeof(mpte->__mpte_dst_v4)) {
  error = EINVAL;
  goto out;
}

if (dst->sa_family == AF_INET6 &&
    dst->sa_len != sizeof(mpte->__mpte_dst_v6)) {
  error = EINVAL;
  goto out;
}

if ((mp_so->so_state & (SS_ISCONNECTED|SS_ISCONNECTING)) == 0) {
  memcpy(&mpte->mpte_dst, dst, dst->sa_len);
}
```

legacy & esoteric

Mac OS X and iOS Internals

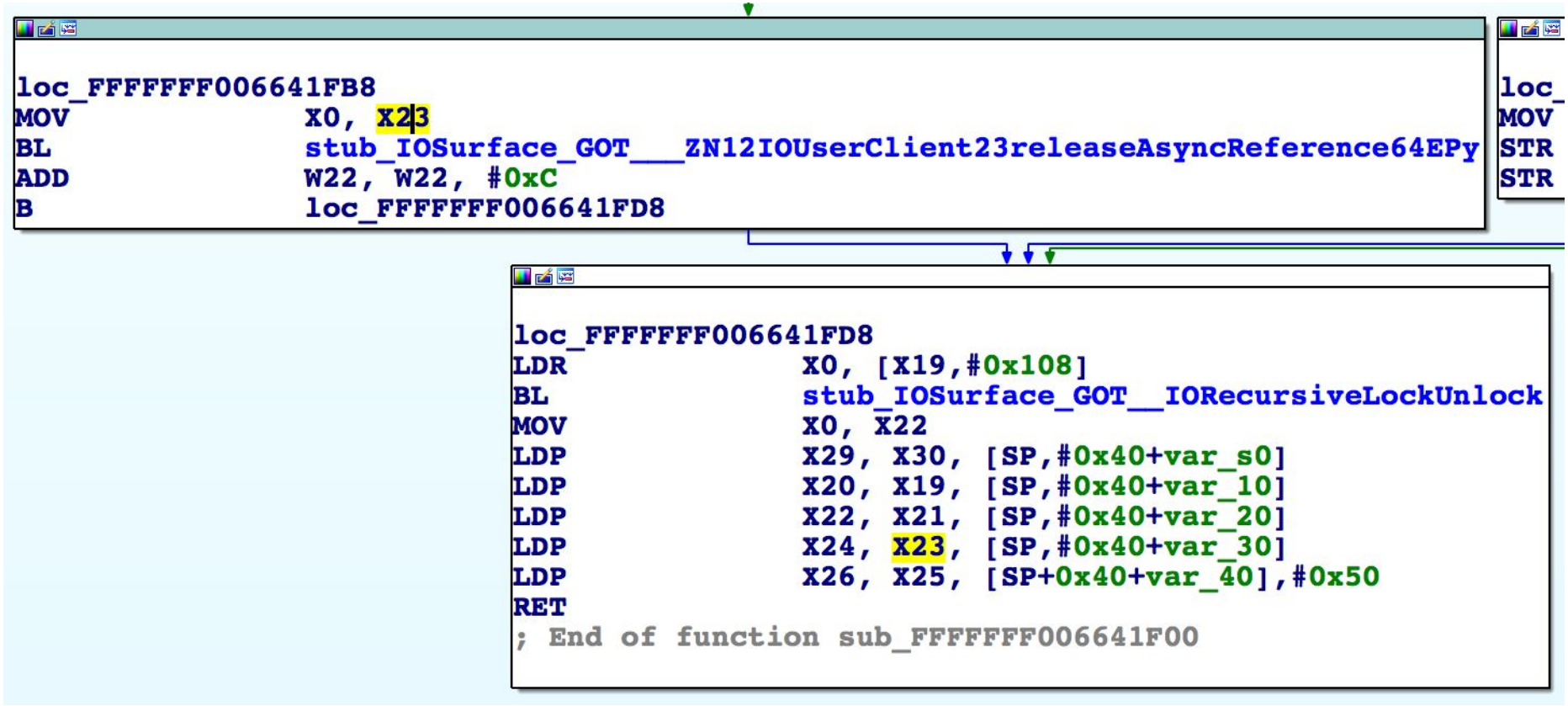
To the Apple's Core

Jonathan Levin

LISTING 10-19: Displaying the tasks on the default processor set

```
void main(int argc, char **argv)
{

  host_t                  myhost = mach_host_self();
  mach_port_t             psDefault;
  mach_port_t             psDefault_control;
  task_array_t            tasks;
  mach_msg_type_number_t numTasks;
  int                     t;                  // a task index

  kern_return_t kr;

  // Get default processor set
  kr = processor_set_default(myhost, &psDefault);

  // Request control port
  kr = host_processor_set_priv(myhost, psDefault, &psDefault_control);
  if (kr != KERN_SUCCESS) { fprintf(stderr, "host_processor_set_priv - %d", kr);
  exit(1); }

  // Get tasks. Note this behaves a bit differently on iOS.
  // On OS X, you can also get the threads directly (processor_set_threads)

  kr = processor_set_tasks(psDefault_control, &tasks, &numTasks);
  if (kr != KERN_SUCCESS) { fprintf(stderr,"processor_set_tasks - %d\n",kr); exit(2); }


  // Iterate through tasks. For now, just display the task ports and their PIDs
  // We use "pid_for_task" to map a task port to its BSD process identifier

  for (t = 0; t < numTasks; i++)
        {
                int pid;
                pid_for_task(tasks[t], &pid);
                printf("Task: %d pid: %d\n", tasks[i],pid);

            // Stay tuned:
            // In the next chapter, this experiment will be expanded to list task
            // information, as well as the threads of each task
        }
}
```

The output of the program in this example differs slightly in iOS: `processor_set_tasks` *will not return PID 0 (the* `kernel_task`*), as getting a handle to the kernel_task can open up potentially dangerous access to the kernel memory maps. Likewise,* `processor_set_threads` *is (apparently intentionally) not supported. There is therefore no legitimate way (jailbreaks not withstanding) to obtain kernel thread or memory handles from user mode — which is just the way Apple would like to keep it.*

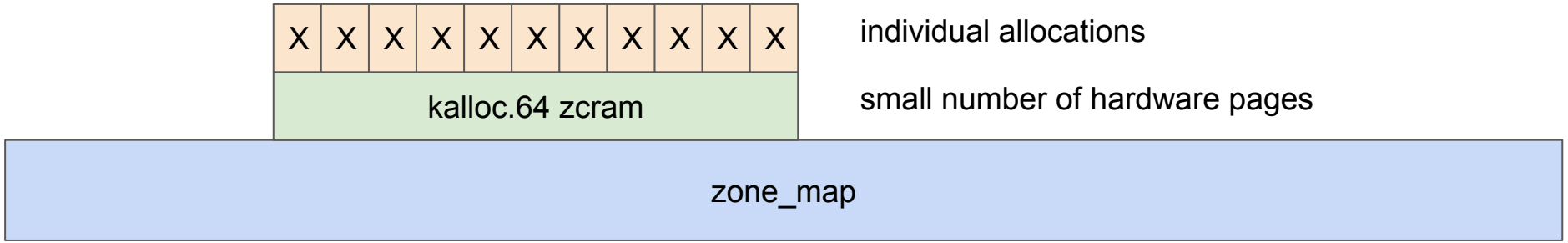

loc_FFFFFFF006641FB8
MOV X0, X23
BL stub_IOSurface_GOT___ZN12IOUserClient23releaseAsyncReference64EPy
ADD W22, W22, #0xC
B loc_FFFFFFF006641FD8
loc_
MOV
STR
STR
loc_FFFFFFF006641FD8
LDR X0, [X19,#0x108]
BL stub_IOSurface_GOT__IORecursiveLockUnlock
MOV X0, X22
LDP X29, X30, [SP,#0x40+var_s0]
LDP X20, X19, [SP,#0x40+var_10]
LDP X22, X21, [SP,#0x40+var_20]
LDP X24, X23, [SP,#0x40+var_30]
LDP X26, X25, [SP+0x40+var_40],#0x50
RET
; End of function sub_FFFFFFF006641F00

```
static void
WriteCheckArgSize(FILE *file, routine_t *rt, argument_t *arg, const char *comparator)
{
  register ipc_type_t *ptype = arg->argType;


  fprintf(file, "\tif (((msgh_size - ");
  rtMinRequestSize(file, rt, "__Request");
  fprintf(file, ") ");
  if (PackMsg == FALSE) {
    fprintf(file, "%s %d)", comparator, ptype->itTypeSize + ptype->itPadSize);
  } else if (IS_OPTIONAL_NATIVE(ptype)) {
    fprintf(file, "%s (In%dP->__Present__%s ? _WALIGNSZ_(%s) : 0))" , comparator,
      arg->argRequestPos, arg->argMsgField, ptype->itServerType);
  } else {
    register ipc_type_t *btype = ptype->itElement;
    argument_t *count = arg->argCount;
    int multiplier = btype->itTypeSize;

    if (multiplier > 1)
      fprintf(file, "/ %d ", multiplier);
    fprintf(file, "< In%dP->%s) ||\n", count->argRequestPos, count->argMsgField);
    fprintf(file, "\t    (msgh_size %s ", comparator);
    rtMinRequestSize(file, rt, "__Request");
    fprintf(file, " + ");
    WriteCalcArgSize(file, arg);
    fprintf(file, ")");
  }
  fprintf(file, ")\n\t\treturn MIG_BAD_ARGUMENTS;\n");
}
```

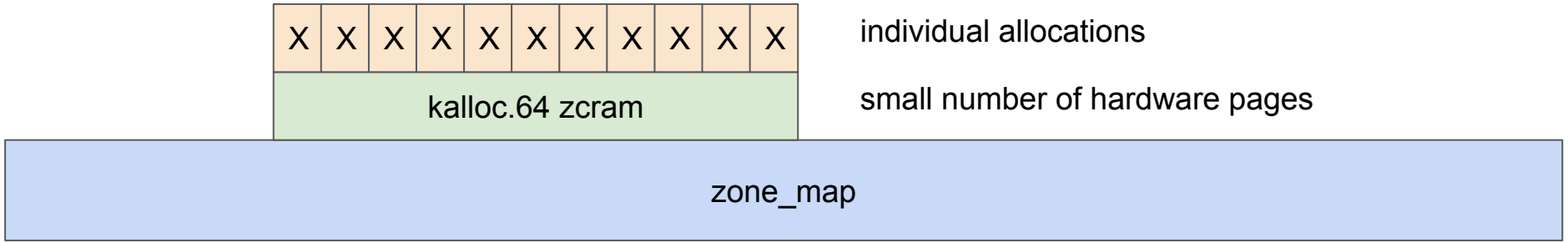
mitigations

weak mitigations can be trivially defeated with more bugs; if you don't care about bugs there's no point shipping weak mitigations

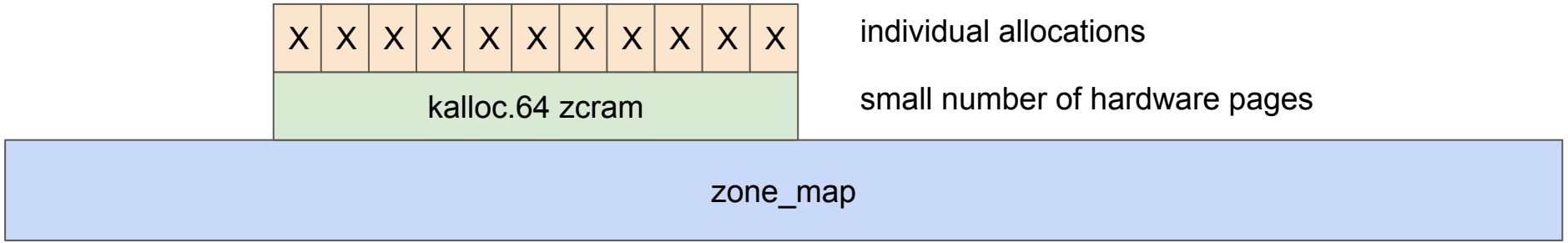
randomization



384 MB reserved kernel vm region (across all iOS 11 devices)

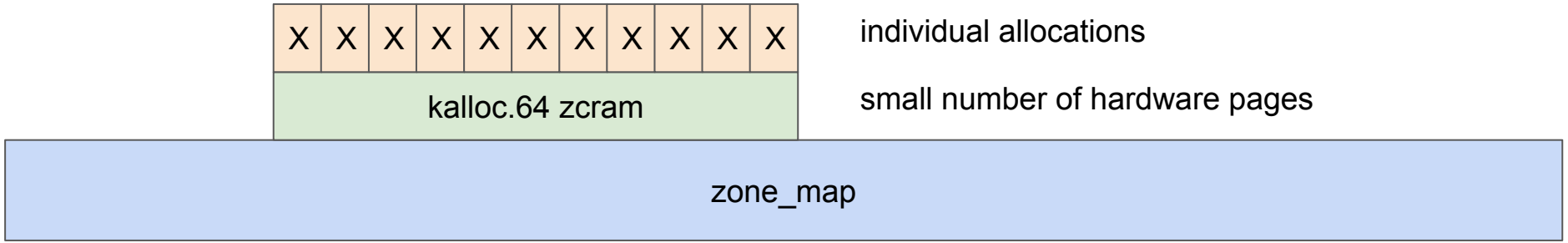


384 MB reserved kernel vm region (across all iOS 11 devices)



384 MB reserved kernel vm region (across all iOS 11 devices)

```
unsigned int random_bool_gen_bits(
  struct bool_gen *bg, unsigned int *buffer, unsigned int count, unsigned int numbits)
{
  unsigned int index = 0;
  unsigned int rbits = 0;

  for (unsigned int bitct = 0; bitct < numbits; bitct++) {
    // Find a portion of the buffer that hasn't been emptied.
    // We might have emptied our last index in the previous iteration.
    while (index < count && buffer[index] == 0)
      index++;

    // If we've exhausted the pool, refill it.
    if (index == count) {
      random_bool_gen_entropy(bg, buffer, count);
      index = 0;
    }
    // Collect-a-bit
    unsigned int bit = buffer[index] & 1;
    buffer[index] = buffer[index] >> 1;
    rbits = bit | (rbits << 1);
  }
  return rbits;
}
```



384 MB reserved kernel vm region (across all iOS 11 devices)

1) there's still a pattern

2) larger objects are less randomized

3) can we just sort the freelist?

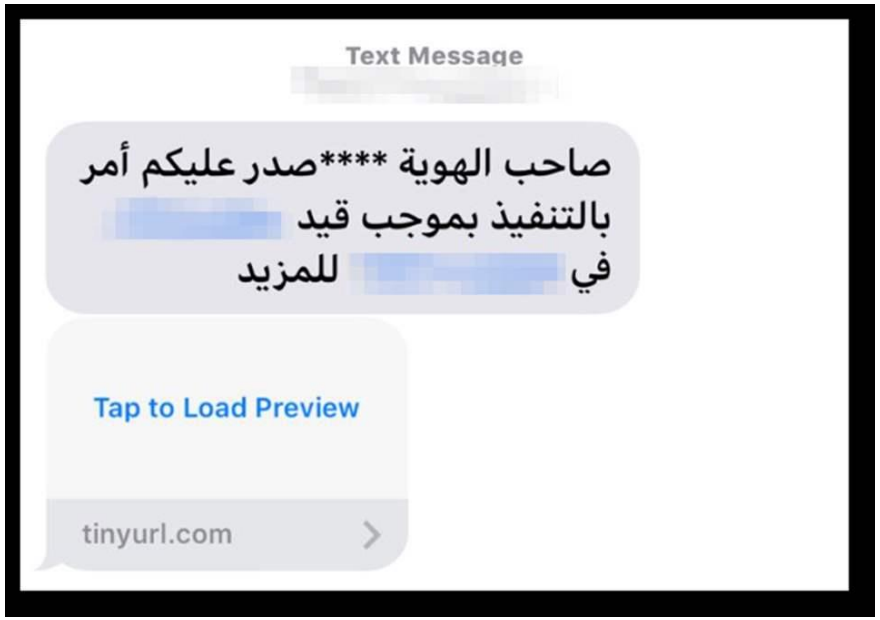
please stop just spot-fixing bugs;
learn from them, and act on that

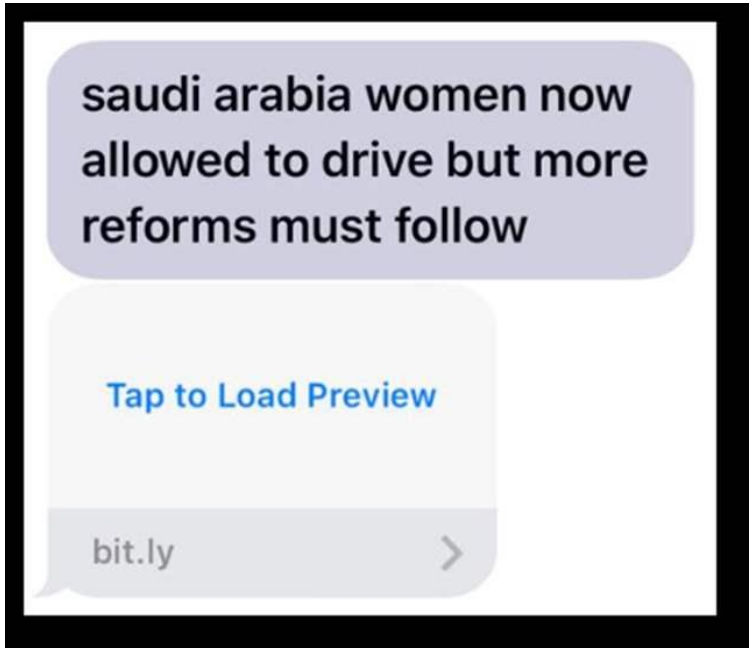
...one more thing

Amnesty International Among Targets of NSO-powered Campaign

1 August 2018, 00:01 UTC

https://www.amnesty.org/en/latest/research/2018/08/amnesty-international-among-targets-of-nso-powered-campaign/


Text Message
صاحب الهوية ****صدر عليكم أمر
بالتنفيذ بموجب قيد
في للمزيد
Tap to Load Preview
tinyurl.com


saudi arabia women now allowed to drive but more reforms must follow
Tap to Load Preview
bit.ly

apple bug bounty

"it's not meant to be any kind of exclusive club. So, if someone who is not in the program, brings to us a vulnerability that would be covered under the program, we welcome that, we're going to take a look, and if the work merits it we'll invite them in to the program and we'll reward the vulnerability that they found."

"In addition to these exact amounts we will let researchers donate the reward to a charity of their choosing, we will at our option match that 1 to 1, doubling the donation."

bug	with charity match
CVE-2016-7612	$100'000
CVE-2016-7621	$100'000
CVE-2016-7644	$100'000
CVE-2017-2353	$100'000
CVE-2017-2370	$100'000
CVE-2017-2360	$100'000
CVE-2017-2473	$100'000
CVE-2017-2474	$100'000
CVE-2017-2478	$100'000
CVE-2017-2501	$100'000
CVE-2017-2482	$100'000
CVE-2017-2490	$100'000
CVE-2017-13867	$100'000
CVE-2017-13847	$100'000
CVE-2017-13861	$100'000
CVE-2018-4241	$100'000
CVE-2018-4243	$100'000
694799600	$100'000
696255847	$100'000
CVE-2016-7637	$50'000
CVE-2016-7661	$50'000
CVE-2016-7660	$50'000
CVE-2017-2522	$50'000
CVE-2017-2523	$50'000
CVE-2017-2524	$50'000
CVE-2017-7047	$50'000
CVE-2018-4206	$50'000
695930632	$50'000
695992129	$50'000
696619631	$50'000
	$2'450'000