The Original Elevat0r

History of a private iOS Jailbreak

August, 2017

Who ?

Stefan Esser

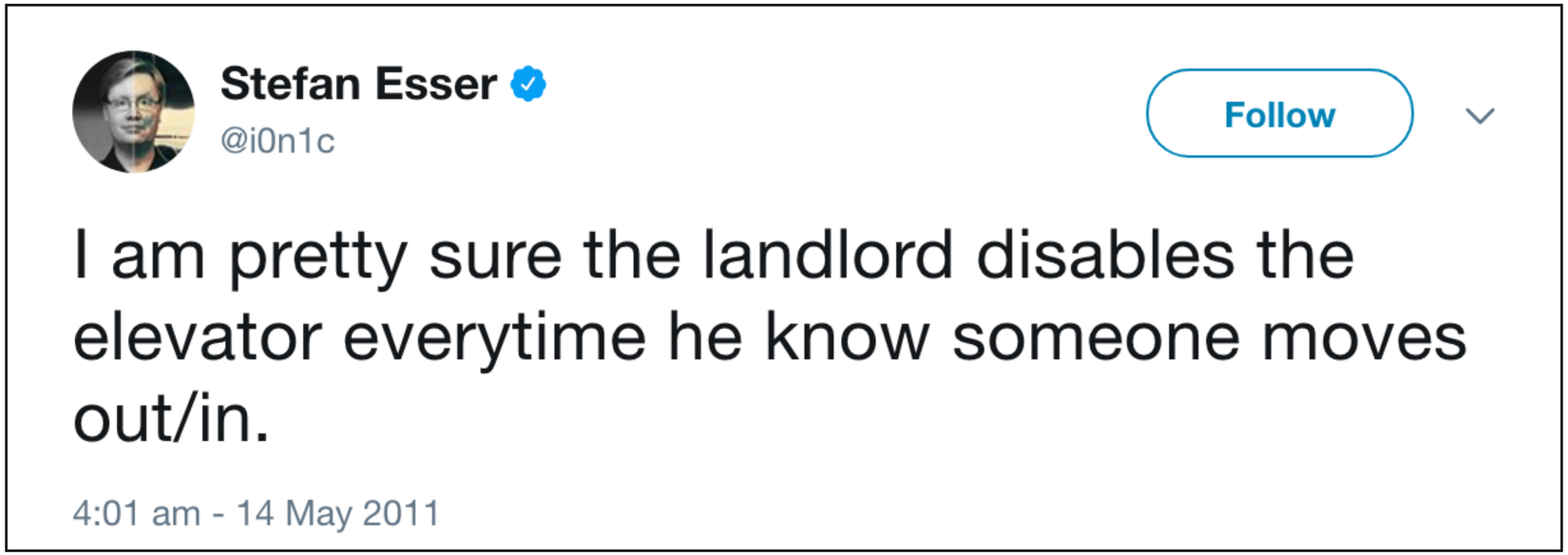
- in Information Security since 1998
- “the PHP security guy who migrated to iOS security”

- SektionEins GmbH 2007-2016
- **AntidOte UG** 2013-now
- **AntidOte SG Pte. Ltd.** 2017-now

Introduction

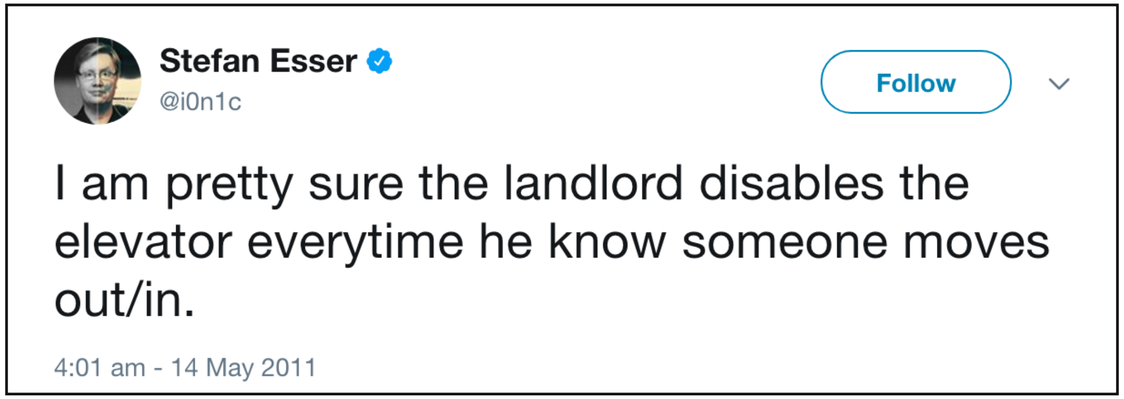
What is elevat0r? (I)

- in 2011 it all started with a harmless tweet



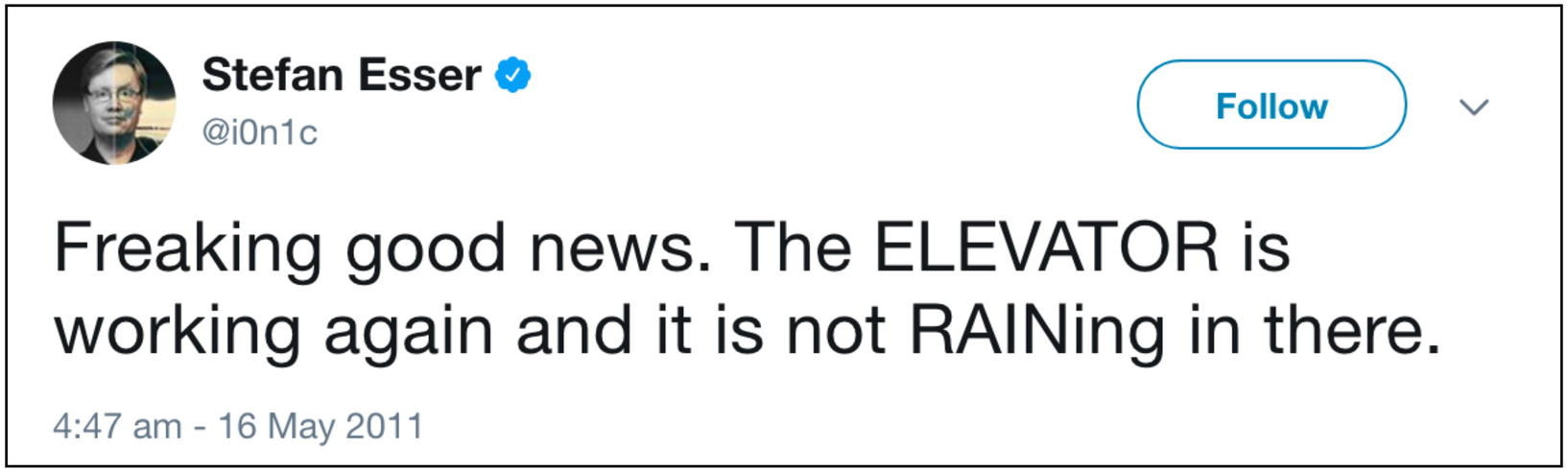
What is elevat0r? (II)

- I was literally complaining about a broken elevator
- but some jokers on Twitter commented on this tweet
- they made people believe that **elevat0r** was a secret codename
- within minutes jailbreak news sites reported about it



What is elevatOr? (III)

- no amount of clarification was able to stop the hype
- so we went along with it and manipulated the JB media



What is elevat0r? (IV)

- since that day **elevat0r** has really become the internal codename for all my private iOS jailbreaks
- this talk is about the first **elevat0r**

The Vulnerability

The Vulnerability

- the original **elevat0r** exploits a kernel memory corruption in the **setattrlist()** system call

```
int setattrlist(const char *path, struct attrlist *alist, void *attributeBuffer,
                size_t bufferSize, u_long options)
```

- this system call allows the modification of file attributes
- is / was reachable from most of the sandboxes

setattrlist()

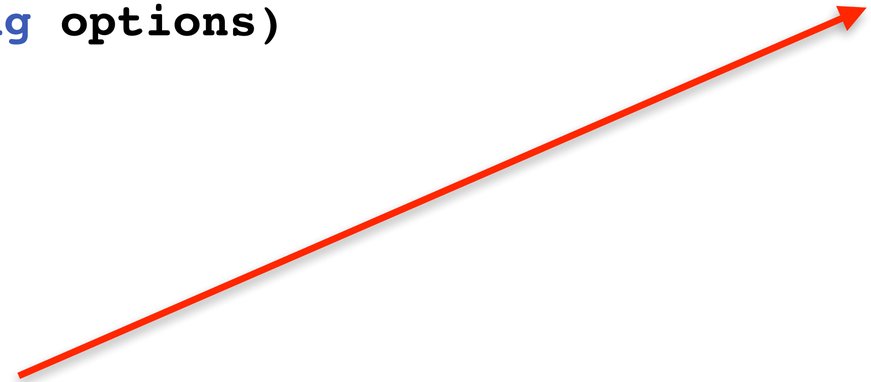
```
int setattrlist(const char *path, struct attrlist *alist, void *attributeBuffer,
                size_t bufferSize, u_long options)
```

alist bit masks control what attributes to set

```
struct attrlist {
    u_short bitmapcount;            /* number of attr. bit sets in list (should be 5) */
    u_int16_t reserved;           /* (to maintain 4-byte alignment) */
    attrgroup_t commonattr;         /* common attribute group */
    attrgroup_t volattr;            /* Volume attribute group */
    attrgroup_t dirattr;            /* directory attribute group */
    attrgroup_t fileattr;           /* file attribute group */
    attrgroup_t forkattr;           /* fork attribute group */
};
```

setattrlist()

```
int setattrlist(const char *path, struct attrlist *alist, void *attributeBuffer,
                size_t bufferSize, u_long options)
```



attributeBuffer contains data for attributes

setattrlist() - attributeBuffer parsing

- attributeBuffer is copied into a buffer on kernel heap

```
if (uap->bufferSize > ATTR_MAX_BUFFER) {
        VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: buffer size %d too large", uap->bufferSize);
        error = ENOMEM;
        goto out;
}
MALLOC(user_buf, char *, uap->bufferSize, M_TEMP, M_WAITOK);  // <----- allocation of buffer
if (user_buf == NULL) {
        VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: could not allocate %d bytes for buffer", uap->bufferSize);
        error = ENOMEM;
        goto out;
}
if ((error = copyin(uap->attributeBuffer, user_buf, uap->bufferSize)) != 0) {  // <---- copying of data
        VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: buffer copyin failed");
        goto out;
}
```

setattrlist() - attributeBuffer parsing

- attributeBuffer is parsed step by step

```
/*
 * Unpack the argument buffer.
 */
cursor = user_buf;
bufend = cursor + uap->bufferSize;

/* common */
if (al.commonattr & ATTR_CMN_SCRIPT) {
    ATTR_UNPACK(va.va_encoding);
    VATTR_SET_ACTIVE(&va, va_encoding);
}
if (al.commonattr & ATTR_CMN_CRTIME) {
    ATTR_UNPACK_TIME(va.va_create_time, proc_is64);
    VATTR_SET_ACTIVE(&va, va_create_time);
}
if (al.commonattr & ATTR_CMN_MODTIME) {
    ATTR_UNPACK_TIME(va.va_modify_time, proc_is64);
    VATTR_SET_ACTIVE(&va, va_modify_time);
}
```

cursor
always points
to current buffer
position

setattrlist() - attributeBuffer parsing

- attributeBuffer is parsed step by step

```
/*
 * Unpack the argument buffer.
 */
cursor = user_buf;
bufend = cursor + uap->bufferSize;

/* common */
if (al.commonattr & ATTR_CMN_SCRIPT) {
    ATTR_UNPACK(va.va_encoding);
    VATTR_SET_ACTIVE(&va, va_encoding);
}
if (al.commonattr & ATTR_CMN_CRTIME) {
    ATTR_UNPACK_TIME(va.va_create_time, proc_is64);
    VATTR_SET_ACTIVE(&va, va_create_time);
}
if (al.commonattr & ATTR_CMN_MODTIME) {
    ATTR_UNPACK_TIME(va.va_modify_time, proc_is64);
    VATTR_SET_ACTIVE(&va, va_modify_time);
}
```

attributeBuffer contains only selected attributes

setattrlist() - attributeBuffer parsing

- attributeBuffer is parsed step by step

```
/*
 * Unpack the argument buffer.
 */
cursor = user_buf;
bufend = cursor + uap->bufferSize;

/* common */
if (al.commonattr & ATTR_CMN_SCRIPT) {
    ATTR_UNPACK(va.va_encoding);
    VATTR_SET_ACTIVE(&va, va_encoding);
}
if (al.commonattr & ATTR_CMN_CRTIME) {
    ATTR_UNPACK_TIME(va.va_create_time, proc_is64);
    VATTR_SET_ACTIVE(&va, va_create_time);
}
if (al.commonattr & ATTR_CMN_MODTIME) {
    ATTR_UNPACK_TIME(va.va_modify_time, proc_is64);
    VATTR_SET_ACTIVE(&va, va_modify_time);
}
```

ATTR_UNPACK*()
read data from cursor
and ensure no out of bounds
access happens

setattrlist() - attributeBuffer parsing

- some attribute data is a bit bigger
- stored somewhere in buffer
- code parses an **attreference_t** instead

```
typedef struct attrreference {
    int32_t      attr_dataoffset;
    u_int32_t    attr_length;
} attrreference_t;
```

relative position from here to attribute data

length of attribute data

setattrlist() - The Vulnerable Code

- vulnerability in parsing of **ATTR_VOL_NAME** attribute data

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffe
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

setattrlist() - The Vulnerable Code

- vulnerability in parsing of **ATTR_VOL_NAME** attribute data

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffe
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

volname
set to current
buffer position

setattrlist() - The Vulnerable Code

- vulnerability in parsing of **ATTR_VOL_NAME** attribute data

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffe
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

attrreference is unpacked

setattrlist() - The Vulnerable Code

- vulnerability in parsing of **ATTR_VOL_NAME** attribute data

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffe
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

volname is adjusted to relative data position

setattrlist() - The Vulnerable Code

- vulnerability in parsing of **ATTR_VOL_NAME** attribute data

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffe
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

check against end of buffer

setattrlist() - The Vulnerable Code

- vulnerability in parsing of **ATTR_VOL_NAME** attribute data

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffe
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

zero terminate volname inside the attributeBuffer

setattrlist() - The Vulnerability

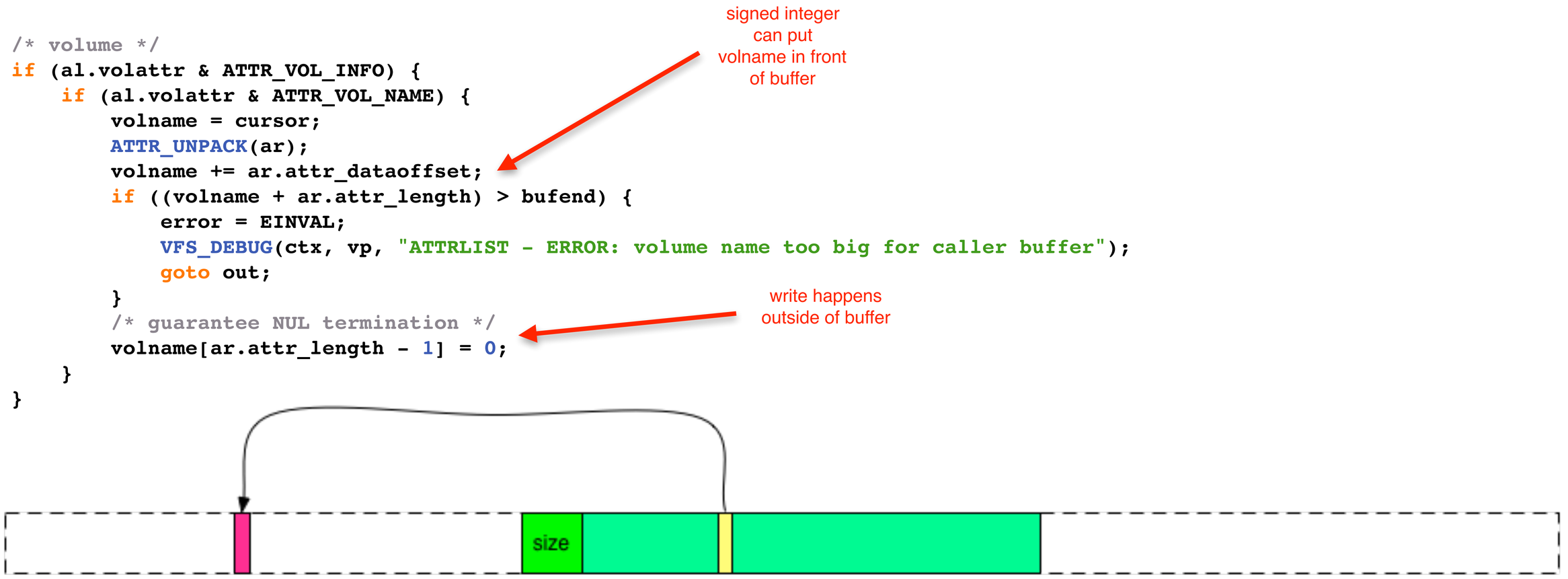
- vulnerability in parsing of **ATTR_VOL_NAME** attribute data

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffe
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

signed integer can put volname in front of buffer

write happens outside of buffer

setattrlist() - The Vulnerability




```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffer");
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

Apple Fixes

setattrlist() fixes

- Apple developers were kinda aware of the problem
- they started fixing the vulnerable code in iOS 6
- but they did not get it right for a while
- one reason might have been that the developer fixing the security problem never escalated the security bug to the security team

setattrlist() - Fix 1 in iOS 6.0

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        /* attr_dataoffset cannot be negative! */
        if (ar.attr_dataoffset < 0) {
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: bad offset supplied (2) ", ar.attr_dataoffset);
            error = EINVAL;
            goto out;
        }

        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffer");
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

setattrlist() - Fix 1 in iOS 6.0

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        /* attr_dataoffset cannot be negative! */
        if (ar.attr_dataoffset < 0) {
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: bad offset supplied (2) ", ar.attr_dataoffset);
            error = EINVAL;
            goto out;
        }

        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffer");
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

Apple now detects negative attr_dataoffset

setattrlist() - Fix 2 in iOS 7.0

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        /* attr_length cannot be 0! */
        if ((ar.attr_dataoffset < 0) || (ar.attr_length == 0)) {
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: bad offset supplied (2) ", ar.attr_dataoffset);
            error = EINVAL;
            goto out;
        }

        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffer");
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

setattrlist() - Fix 2 in iOS 7.0

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        /* attr_length cannot be 0! */
        if ((ar.attr_dataoffset < 0) || (ar.attr_length == 0)) {
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: bad offset supplied (2) ", ar.attr_dataoffset);
            error = EINVAL;
            goto out;
        }

        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffer");
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

Apple now detects a attr_length of 0

0-1 would be in MALLOC() size field

setattrlist() - Remaining Problem up to iOS 9.0

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        /* attr_length cannot be 0! */
        if ((ar.attr_dataoffset < 0) || (ar.attr_length == 0)) {
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: bad offset supplied (2) ", ar.attr_dataoffset);
            error = EINVAL;
            goto out;
        }

        volname += ar.attr_dataoffset;
        if ((volname + ar.attr_length) > bufend) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffer");
            goto out;
        }
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

integer wrap possible on 32 bit volname will point before buffer

on 32 bit systems we can address anything before buffer

setattrlist() - Final Fix in iOS 9.0

```
/* volume */
if (al.volattr & ATTR_VOL_INFO) {
    if (al.volattr & ATTR_VOL_NAME) {
        volname = cursor;
        ATTR_UNPACK(ar);
        /* attr_length cannot be 0! */
        if ((ar.attr_dataoffset < 0) || (ar.attr_length == 0) ||
            (ar.attr_length > uap->bufferSize) ||
            (uap->bufferSize - ar.attr_length < (unsigned)ar.attr_dataoffset)) {
                VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: bad offset supplied (2) ", ar.attr_dataoffset);
                error = EINVAL;
                goto out;
        }

        if (volname >= bufend - ar.attr_dataoffset - ar.attr_length) {
            error = EINVAL;
            VFS_DEBUG(ctx, vp, "ATTRLIST - ERROR: volume name too big for caller buffer");
            goto out;
        }
        volname += ar.attr_dataoffset;
        /* guarantee NUL termination */
        volname[ar.attr_length - 1] = 0;
    }
}
```

parameter verification overkill

Exploitation

Exploitation

- lifetime of bug was from early iOS to iOS 8.4.1
- we knew of it around the time of iOS 5
- lots of changes during that time to iOS
- different iOS versions required different exploits

Exploitation on iOS 5

- at time of iOS 5 there were no mitigations in kernel land
- there was no user-land dereference protection at all in iOS
- also no protection against kernel code execution from user pages
- back then kernel exploits usually
  - mapped malicious kernel data structures in user land
  - used memory corruption to change kernel pointers into user land pointers
  - from there code execution was never far away

Exploitation of setattrlist() in iOS 5

- Heap-Feng-Shui
  - fill kernel heap with C++ objects by opening driver connections to **AppleJPEGDriver** via **io_service_open_extended**() and XML properties

  - poke holes into allocation by reusing dictionary keys in XML property lists

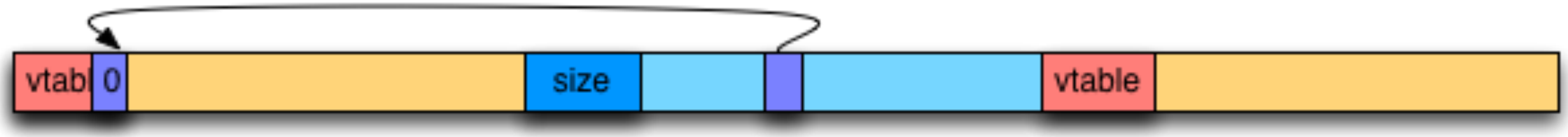
  - for details about this technique see
    “BlackHat 2012 - Stefan Esser - iOS Kernel Heap Armageddon Revisited”

Exploitation of setattrlist() in iOS 5

- Memory Corruption
  - call **setattrlist**() with a buffer size that puts the **MALLOC**() buffer into one of the poked holes

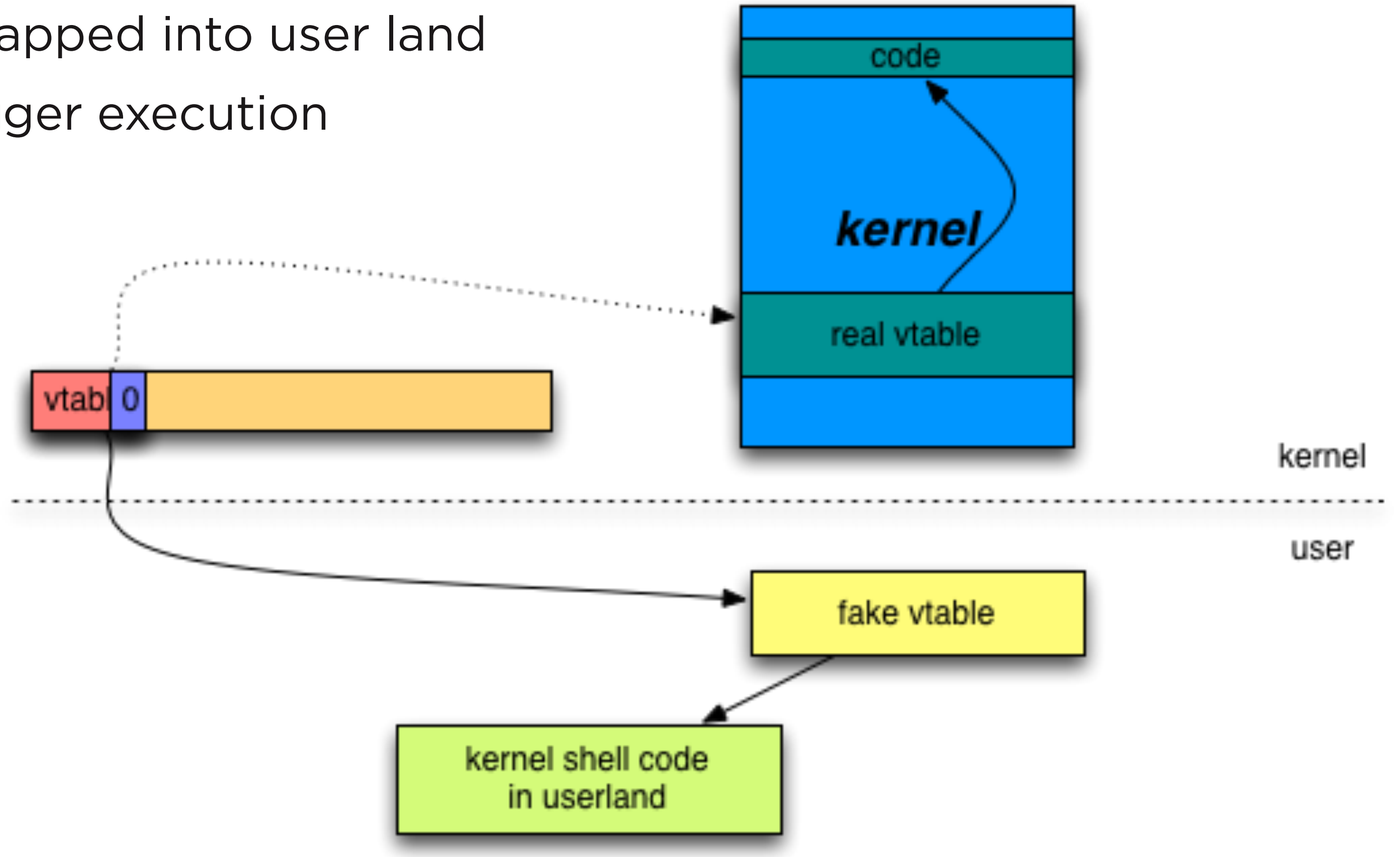
Exploitation of setattrlist() in iOS 5

- Memory Corruption
  - trigger the out of bounds 0 byte write
  - target the highest byte of an adjacent C++ object's vtable pointer



Exploitation of setattrlist() in iOS 5

- Memory Corruption
  - vtable pointer now points to user-land where we map a fake vtable
  - fake vtable points to “shell code” mapped into user land
  - close IOKit driver connection to trigger execution



Exploitation on iOS 6

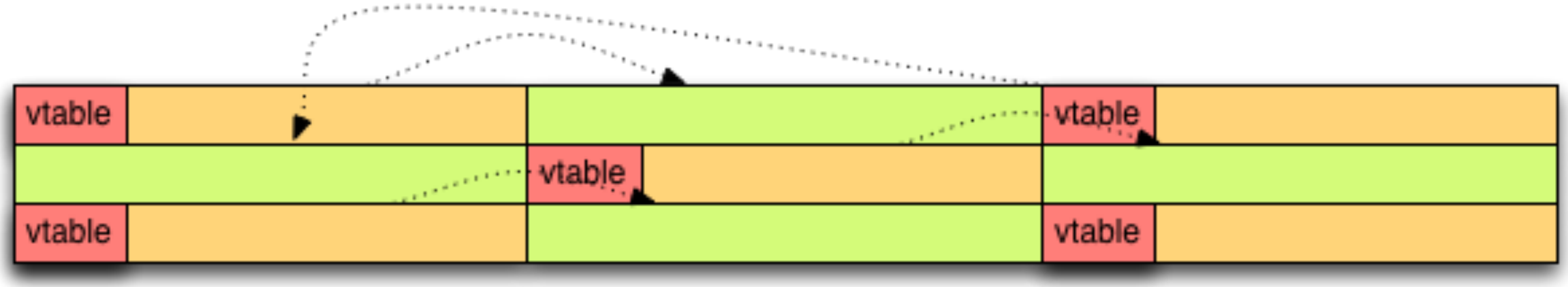
- at time of iOS 6 the kernel got a number of mitigations
  - KASLR required info leaking to determine kernel base address
  - no more user space dereference
- we need to first break KASLR
- then trigger execution of code in kernel land

Exploitation on iOS 6 - Breaking KASLR

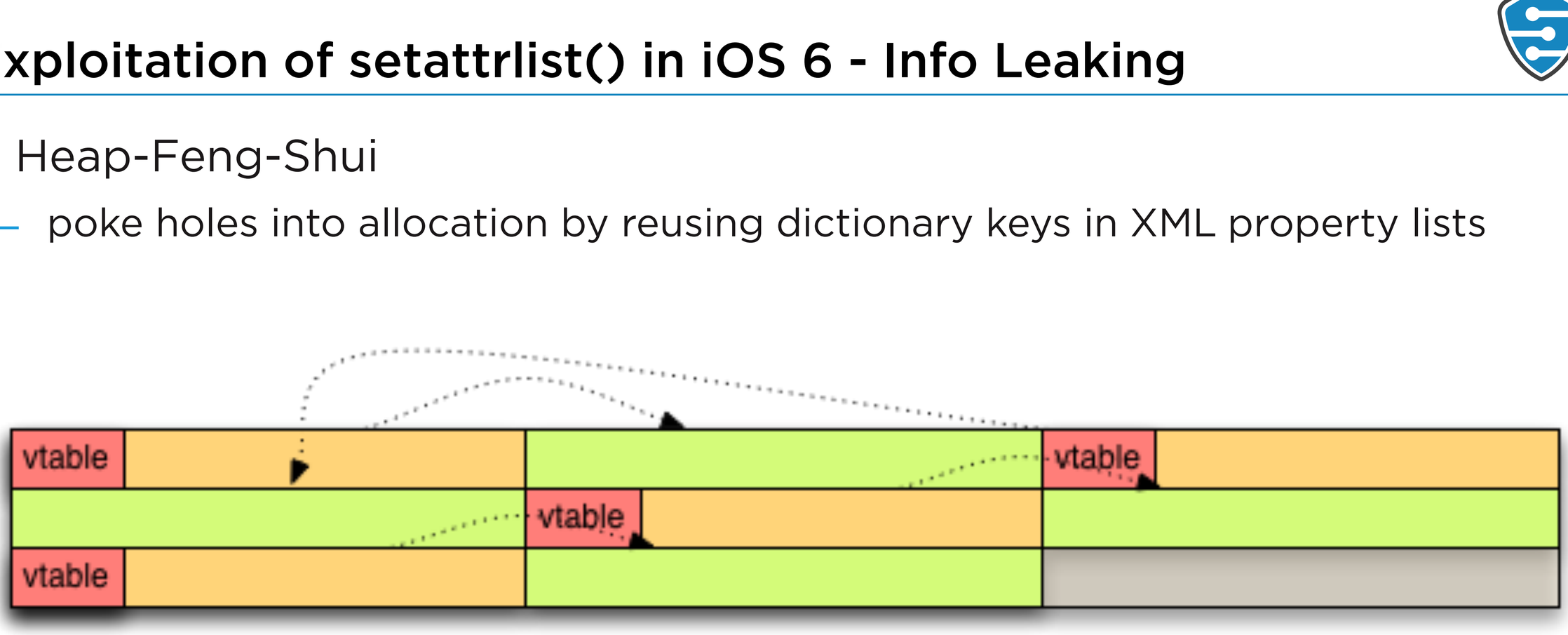
- a bunch of info leaks vulnerabilities in iOS
- many have been known to various parties since iOS 6.0 required them
  - mach_port_kobject()
  - kext_request()
  - io_registry_entry_get_property_bytes()
- however because this would be too easy we make our own

Exploitation of setattrlist() in iOS 6 - Info Leaking

- Heap-Feng-Shui
  - fill kernel heap with IOKit data objects by opening driver connections to **AppleJPEGDriver** via **io_service_open_extended**() and XML properties
  - ensure that data object storage buffers are interleaved with C++ objects



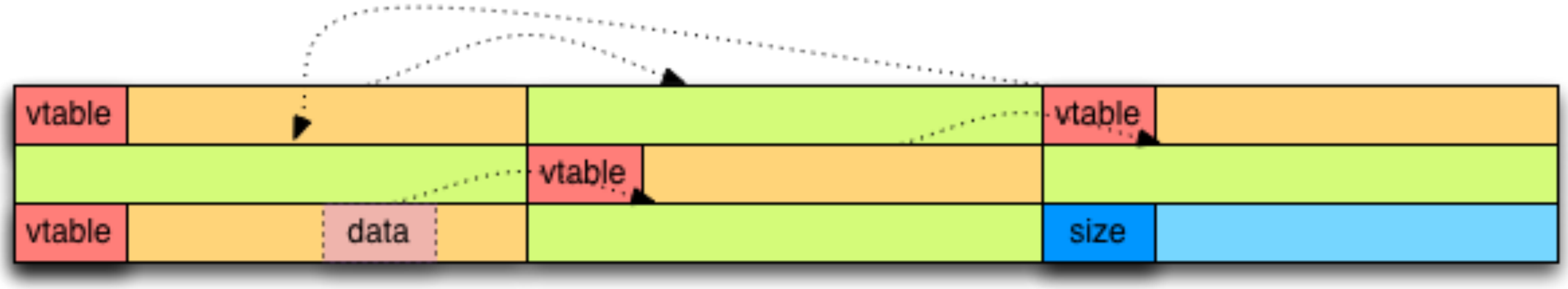
Exploitation of setattrlist() in iOS 6 - Info Leaking



- Heap-Feng-Shui
  - poke holes into allocation by reusing dictionary keys in XML property lists

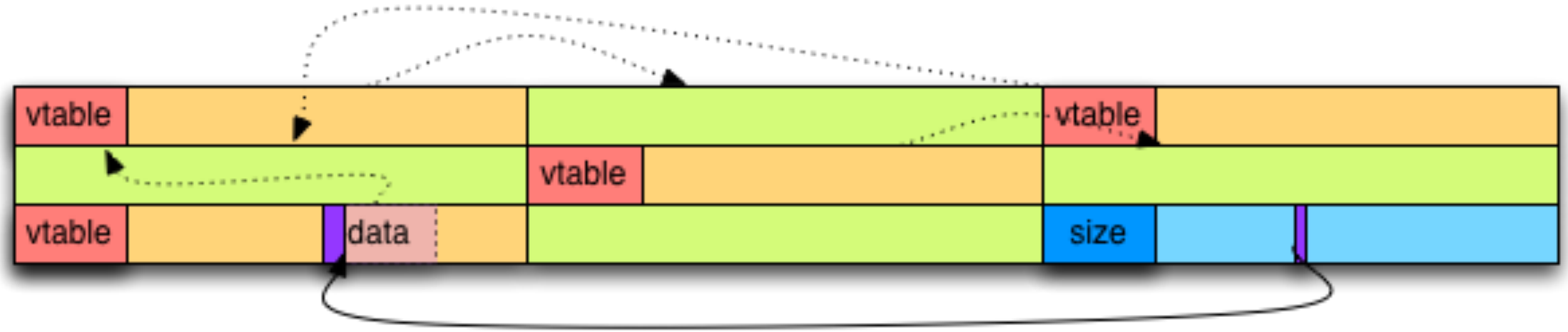
Exploitation of setattrlist() in iOS 6 - Info Leaking

- Memory Corruption
  - call **setattrlist**() with a buffer size that puts the **MALLOC**() buffer into one of the poked holes



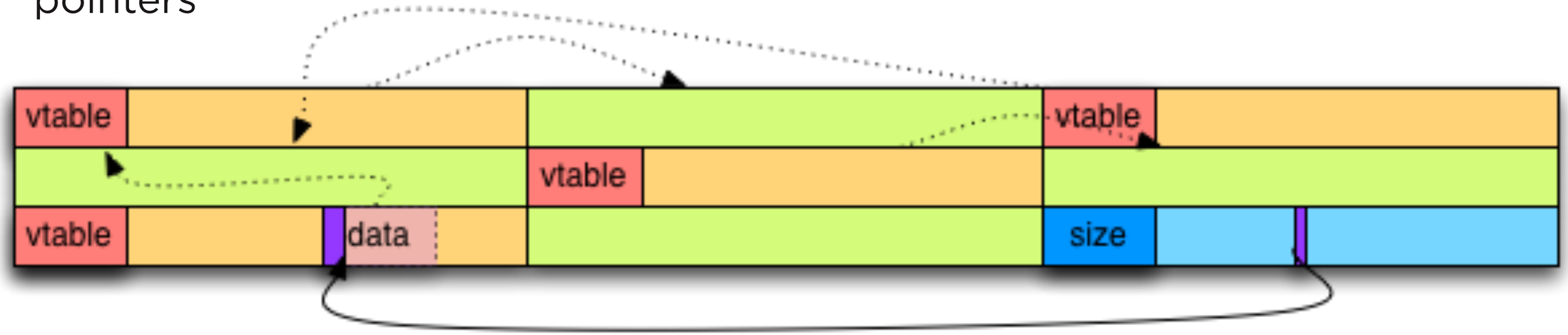
Exploitation of setattrlist() in iOS 6 - Info Leaking

- Memory Corruption
  - trigger the out of bounds 0 byte write and target the lowest byte of an adjacent IOKit data object's storage pointer

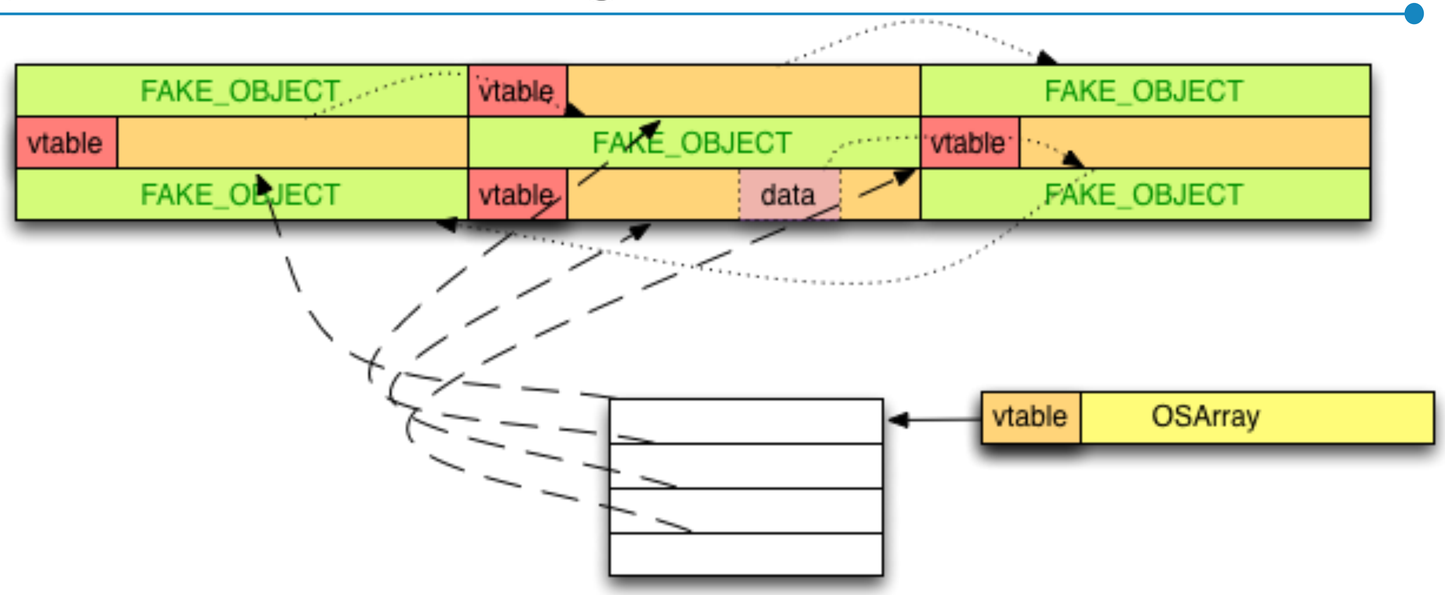


Exploitation of setattrlist() in iOS 6 - Info Leaking

- Memory Corruption
  - use **io_registry_entry_get_property_bytes**() to read back the data
  - because data storage is interleaved with C++ object this will get us a vtable pointer which is inside the kernel image and therefore breaks KASLR
  - depending on heap layout this also leaks a heap pointer at the same time otherwise we need to redo the info leak and this time target heap location pointers

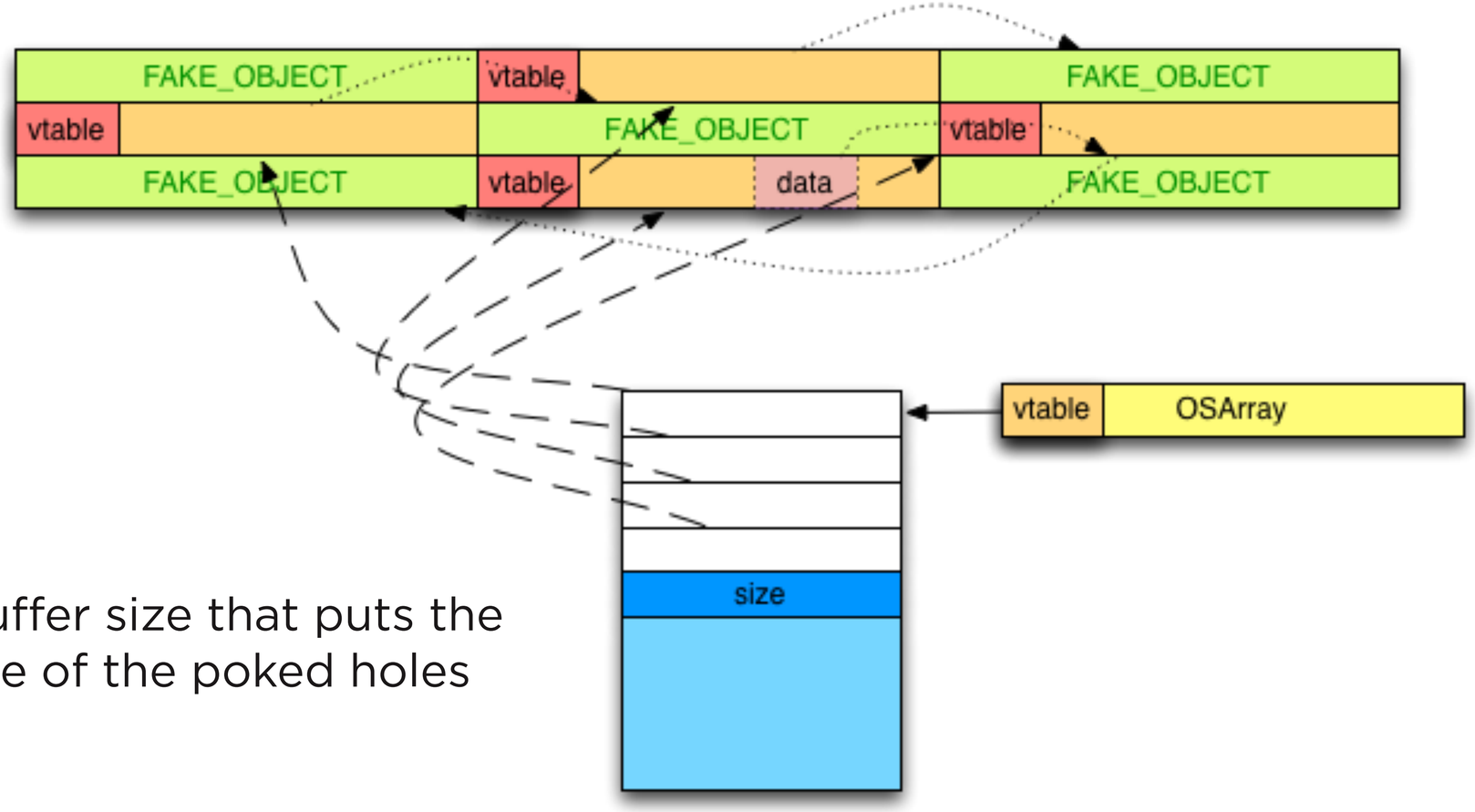


Exploitation of setattrlist() in iOS 6 - Taking Control



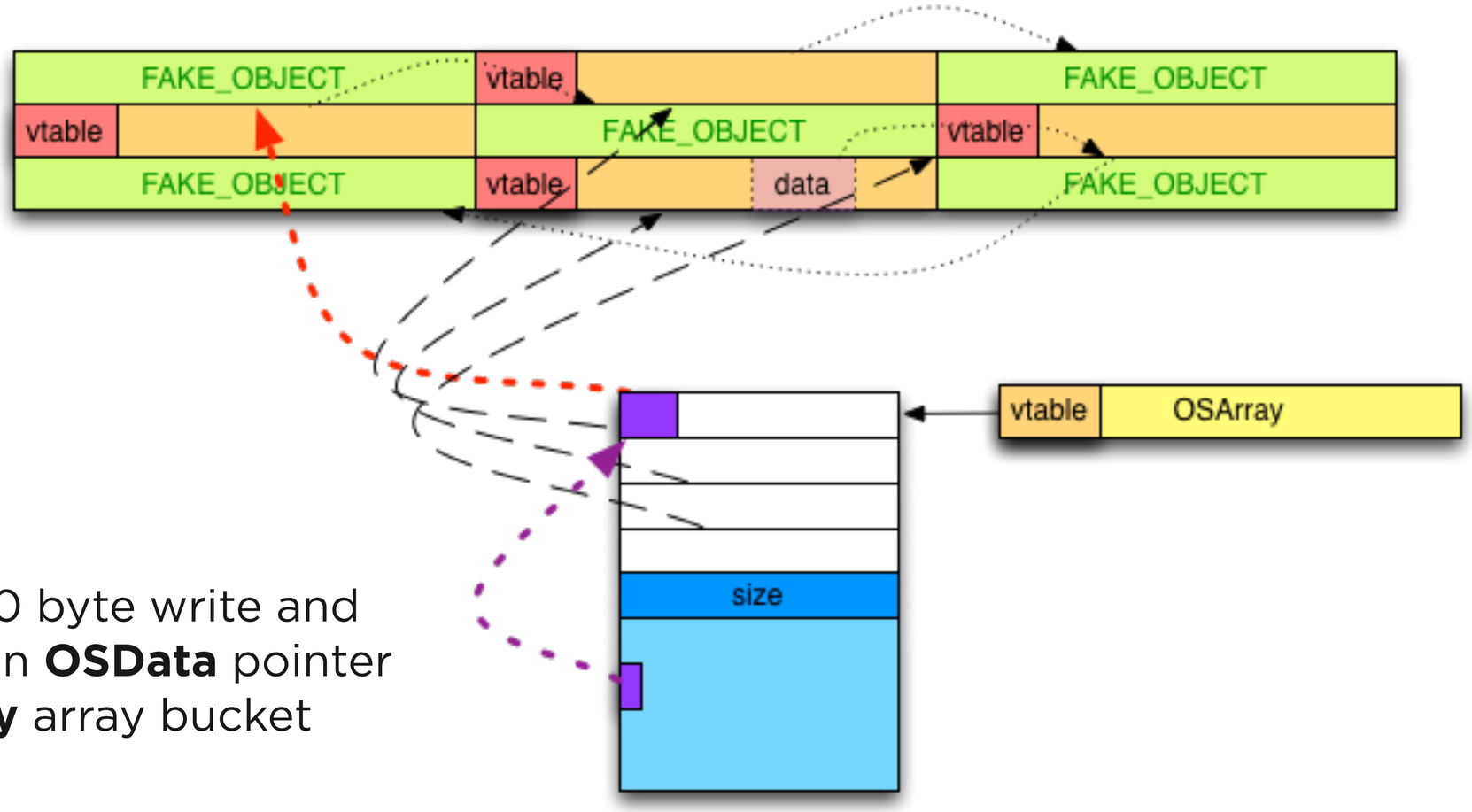
- Heap-Feng-Shui
  - fill kernel heap with **OSData** and **OSArray** objects by opening driver connections to **AppleJPEGDriver** via **io_service_open_extended**() and XML properties
  - ensure that **OSData** object and their data storage buffers are interleaved
  - fill the arrays with pointers to **OSData** objects
  - poke holes in between **OSArray** objects

Exploitation of setattrlist() in iOS 6 - Taking Control



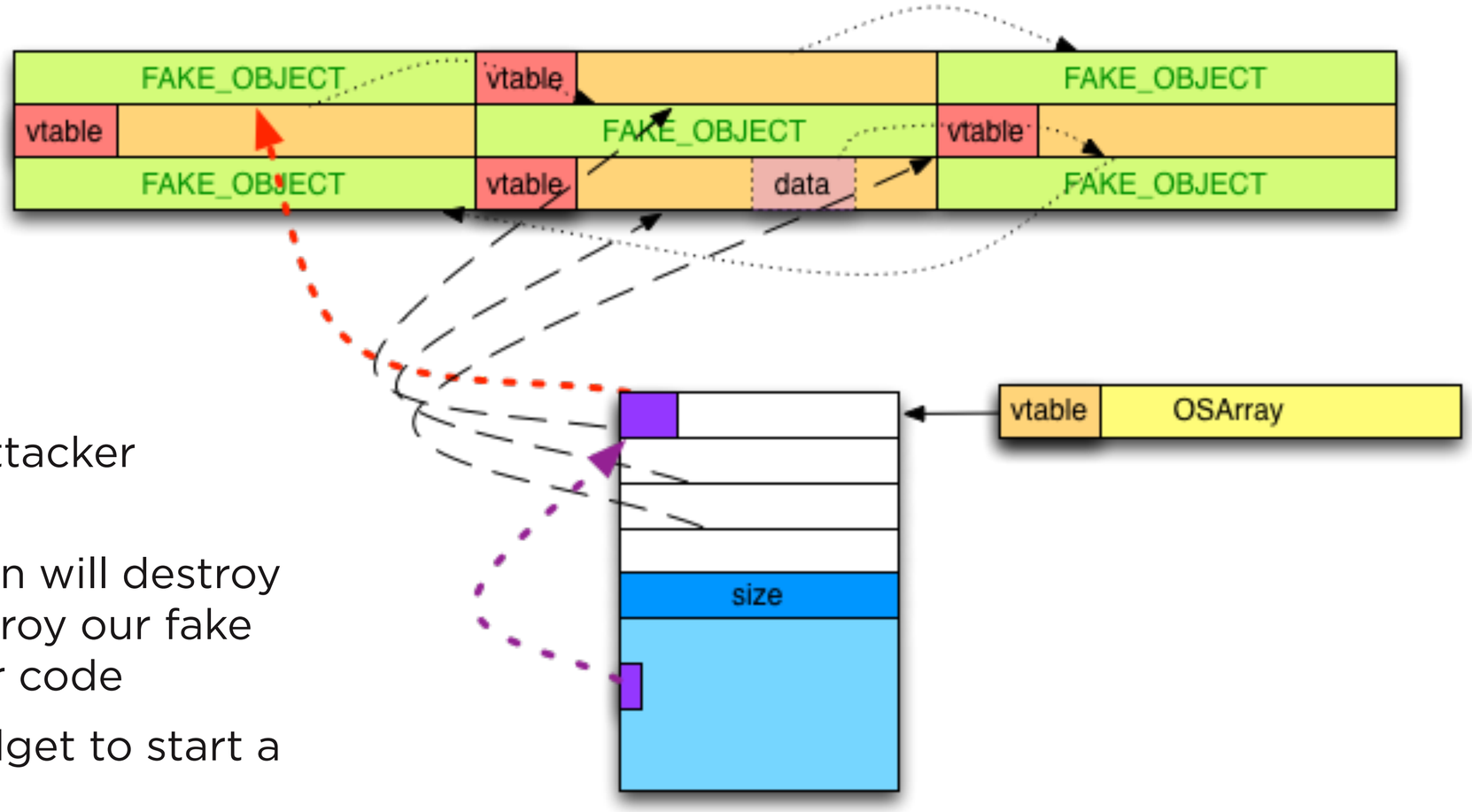
- Memory Corruption
  - call **setattrlist**() with a buffer size that puts the **MALLOC**() buffer into one of the poked holes

Exploitation of setattrlist() in iOS 6 - Taking Control



- Memory Corruption
  - trigger the out of bounds 0 byte write and target the lowest byte of an **OSData** pointer inside an adjacent **OSArray** array bucket

Exploitation of setattrlist() in iOS 6 - Taking Control



- Memory Corruption
  - **OSArray** now contains an attacker controlled fake object
  - closing the driver connection will destroy the **OSArray** and try to destroy our fake object which will trigger our code
  - then e.g. trigger a pivot-gadget to start a ROP chain inside the heap

Exploitation on iOS 7

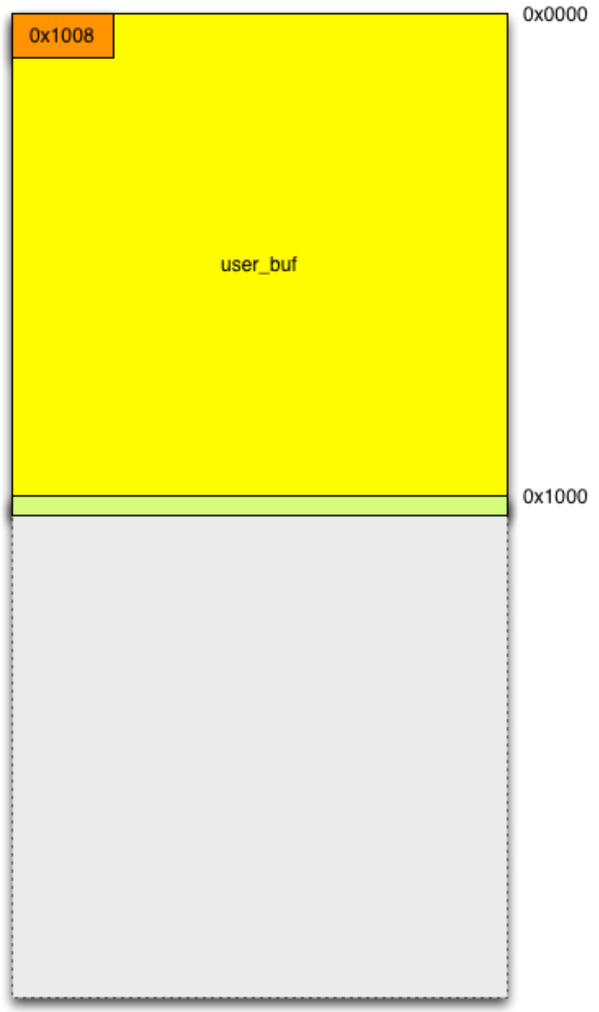
- iOS 6 exploit would work on iOS 7 but we wanted to experiment
- at time of iOS 7 new code was added to the Zone Allocator
- new pagelist feature added unprotected meta data at end of page
  - double linked list, zone back pointer, some counters
- new feature was only used for some zones
- double linked list meant unprotected unlink()
- we wanted to attack this

Exploitation of setattrlist() in iOS 7 - Info Leaking

- we wanted an easier and more stable exploit
- so we just used the **kext_request**() information leak to break KASLR
- this is an API giving back mach-o headers of kernel and KEXT
  - publicly known to be problematic since
    “HITB2012KUL - Mark Dowd and Tarjei Mandt - iOS 6 Security”
  - contained multiple different bugs leaking kernel base address
  - Apple needed multiple attempts to fix it correctly

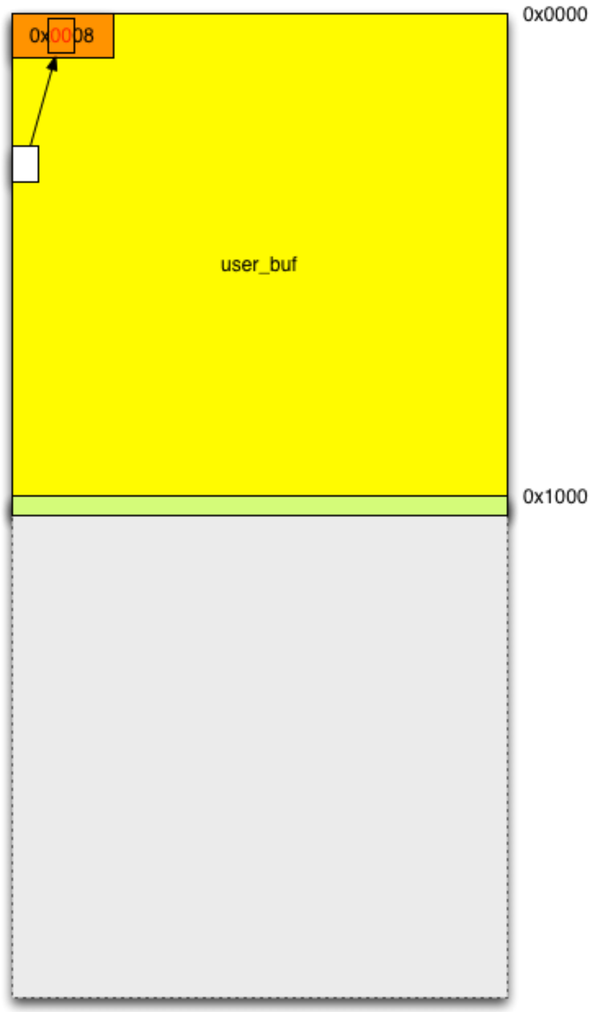
Exploitation of setattrlist() in iOS 7 - Taking Control

- call **setattrlist**() with a buffer size a bit above 0x1000

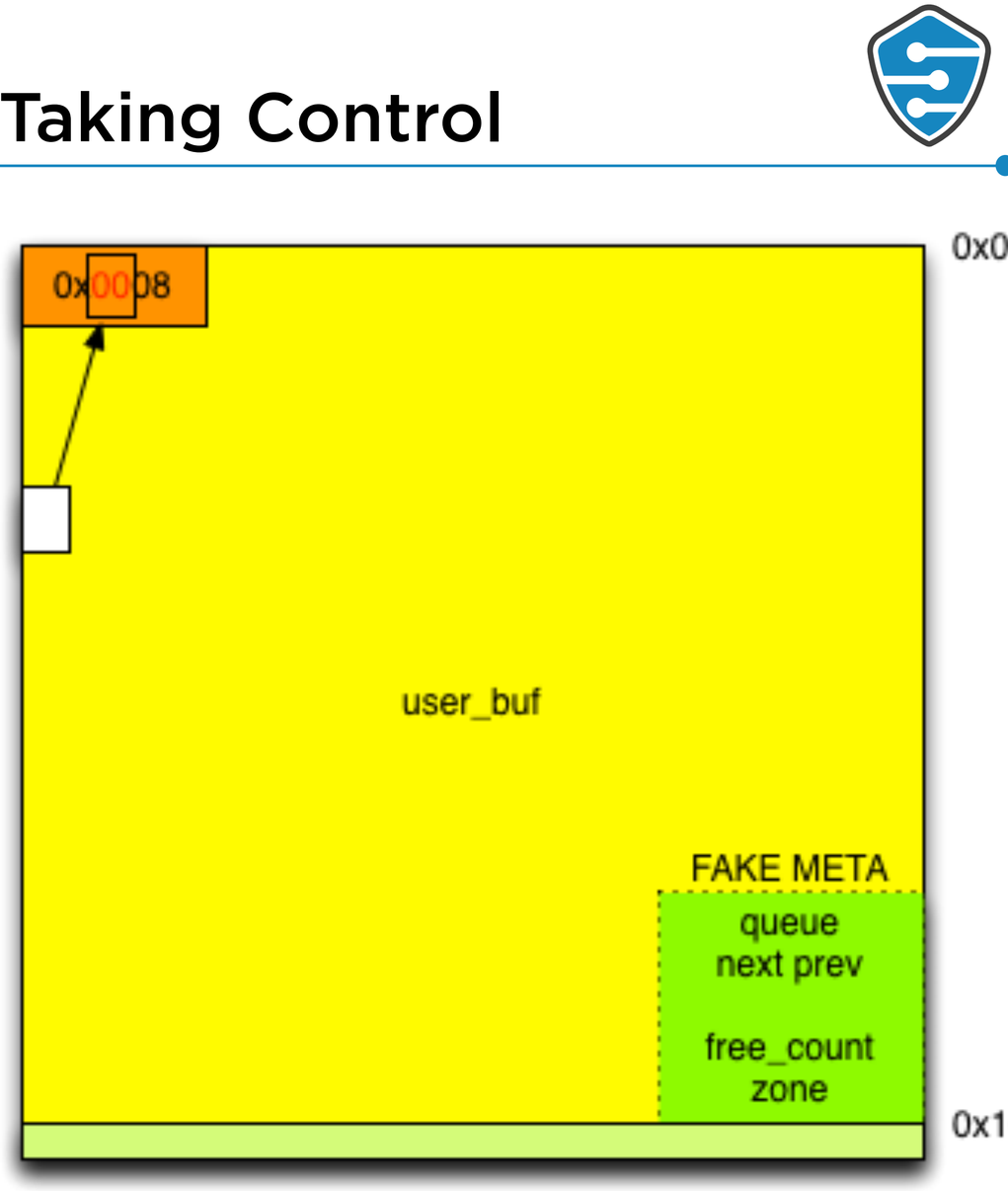


Exploitation of setattrlist() in iOS 7 - Taking Control

- call **setattrlist**() with a buffer size a bit above 0x1000
- trigger the 0 byte write and target the second byte of the **MALLOC**() size field in front of the buffer
- size field becomes very very small
- the **FREE**() will try to put the buffer into a very small zone
- the small zone uses the new pagelist feature



Exploitation of setattrlist() in iOS 7 - Taking Control



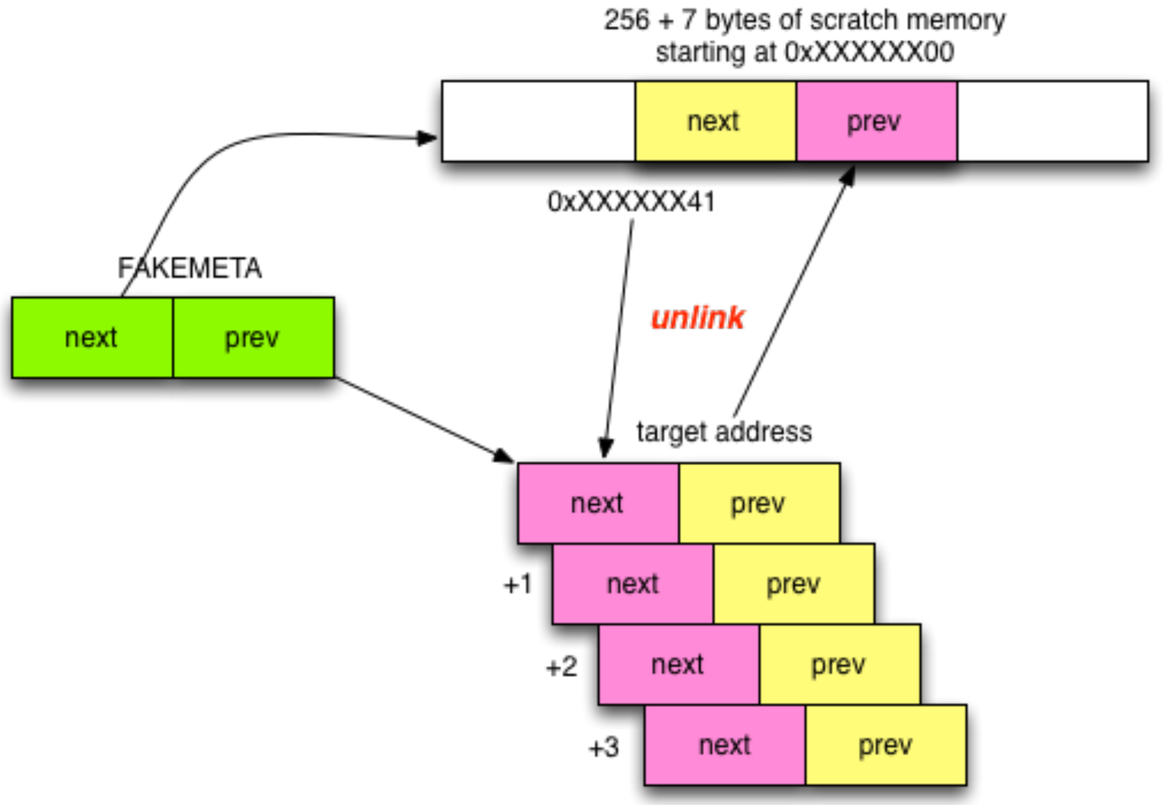
- because target zone uses pagelist feature the end of our page will be used as page metadata
- however the content of that fake metadata is fully controlled by us
- we fully control the forward and next pointer of the double linked list and the counters

Exploitation of setattrlist() in iOS 7 - Taking Control

- trick allocator into removing our page from list of partially used pages and adding it to the list of all free pages
- this will unlink our page from the double linked list which gives us a write anywhere primitive

Exploitation of setattrlist() in iOS 7 - Taking Control

- we setup the pointers like this
  - next: scratch buffer pointer in kernel data (lowest byte will overwrite our target)
  - prev: arbitrary kernel address where we want to write one byte to
- we can repeat this exploit endlessly
- we can write one byte at a time anywhere
- we can write any data structure we want and make the kernel use it



Exploitation on iOS 8

- the iOS 7 exploit stops working in iOS 8
- Apple has protected the unsafe unlink operations
- we can go back to the exploitation technique used in iOS 6
- maybe combine with **io_registry_entry_get_property_bytes**() info leak
- would make the whole exploit way easier

Conclusion

Conclusion

- Apple are sometimes aware of security bugs but don’t fix them correctly
- architectural changes and mitigations in new iOS versions sometimes require reimplementation of exploits
- but sometimes those change make exploitation easier

- more details and POC will be available next week on https://www.antid0te.com
- we are hiring in Singapore … if you are interested in iOS / MacOS contact us

Questions ?

www.antid0te.com