Mining for Bugs with Graph Database Queries

Fabian Yamaguchi (@fabsx00)
31C3
Hello 31C3!

» It’s been 5 years since my last CCC talk 😊
» Talk back then: mostly about hard earned bugs
» Talk today: my attempts since then to make bug hunting a little less painful.
Pattern Recognition for Vulnerability Discovery

» Finding tiny problems in huge code bases

» Move away from precise but hard to scale methods typical for program analysis

» **Engineering perspective**

Robust (inexact) analysis that recognizes bugs in noisy code bases

» **Assist auditors** in their daily work, don’t try to replace them.
Combining two areas of research

Good old CS

Shiny and new

Graph Databases
Stefan Esser’s libssh2 Bug (Syscan’13)

Store attacker controlled data in 32 bit integer

```c
[...]
if (channelp) {
    /* set signal name (without SIG prefix) */
    uint32_t namelen =
        _libssh2_ntohu32(data + 9 + sizeof("exit-signal"));
    channelp->exit_signal =
        LIBSSH2_ALLOC(session, namelen + 1);

[...]
    memcpy(channelp->exit_signal,
        data + 13 + sizeof("exit_signal"), namelen);
    channelp->exit_signal[namelen] = '\0';
[...]
}
[...]
```
### Stefan Esser’s libssh2 Bug

#### Summation inside allocation

```c
[...]
if (channelp) {
    /* set signal name (without SIG prefix) */
    uint32_t namelen =
    _libssh2_ntohu32(data + 9 + sizeof("exit-signal"));
    channelp->exit_signal =
        LIBSSH2_ALLOC(session, namelen + 1);
    [...]  
    memcpy(channelp->exit_signal,
           data + 13 + sizeof("exit_signal"), namelen);
    channelp->exit_signal[namelen] = '\0';
    [...]
}
[...]
```

---

**From:** Mountain Lion/iOS Vulnerabilities Garage Sale, Syscan’13
Heap-based buffer overflow

```c
[…] if (channelp) {
    /* set signal name (without SIG prefix) */
    uint32_t namelen =
        _libssh2_ntohs(data + 9 + sizeof("exit-signal"));
    channelp->exit_signal =
        LIBSSH2_ALLOC(session, namelen + 1);
    […]
    memcpy(channelp->exit_signal,
        data + 13 + sizeof("exit_signal"), namelen);
    channelp->exit_signal[namelen] = '\0';
    […]
}
 […]
```
Q: How did he find the bug?

- Whitebox fuzzer enhanced with symbolic execution?
- Machine learning powered anomaly detector?
- Theorem proving? Model checking?
Q: How did he find the bug?
   - Whitebox fuzzer enhanced with symbolic execution?
   - Machine learning powered anomaly detector?
   - Theorem proving? Model checking?

A: regular expression for `grep`.

```
'ALLOC[A-Z0-9_]*\s*\([^,]*,[^;]*\[^+\-\[^>\[^;]*\]\s*;'
```
What I think this should tell us

» If used right, even primitive tools like grep are powerful

   *Allow auditor knowledge to guide the analysis!*

» Query is a simple model making auditor knowledge explicit

» High false positive rates are often tolerated in practice
Goal: A Robust Search Engine for Source Code

Source Code -> Robust Parser

Query -> Pattern Recognition

Response
Aspects of code a query should be able to model

» What do statements look like?
» Can we get from one statement to another?
» How do statements affect each other?

» Let's see how compilers deal with this

```c
[...]
if (channelp) {
    /* set signal name (without SIG prefix) */
    uint32_t namelen = 
    _libssh2_ntohu32(data + 9 + sizeof("exit-signal"));
    channelp->exit_signal = 
    LIBSSH2_ALLOC(session, namelen + 1);
[...]
    memcpy(channelp->exit_signal,
           data + 13 + sizeof("exit_signal"), namelen);
    channelp->exit_signal[namelen] = '\0';
[...]
}
[...]
```
void foo()
{
    int x = source();
    if (x < MAX)
    {
        int y = 2 * x;
        sink(y);
    }
}

Typical graph representations for code analysis

AST

CFG

PDG
Representations have different strengths

» None of these representations can “do it all”

» Typical query: *Find a call to foo in a statement where data flow exists to a statement that contains a call to bar.*

» Creates the need to transition from one graph representation to the other. (here: AST -> PDG -> AST)

» Can’t we get a representation to account for all of these aspects and allow transitions?
Idea: Merge representations at statement nodes

» Sub-graphs for statements in AST, CFG and PDG
Describe vulnerabilities as sub-graphs of the code property graph!
Now, let’s just put it in a database and we’re done

» A number of great RE tools use RDBMS backends
  » Tried to map graphs to tables… failed
  » Tried to map graphs to documents … failed
  » Tried to map graphs to graphs. Succeeded immediately ;)

»
Native storage format of RDBMs: Tables

- Relationships can be expressed via “JOIN tables”
- Lookup time proportional to size of JOIN table => inefficient lookups for data with lots of many-to-many relationships
- Well suited to store data that comes in tables in the first place
» Ignore the guy in the suit, this stuff actually makes sense from a technical point of view

» “NoSQL”: Storing and querying databases differently
  » A search for alternatives to the relational database model

» **Graph databases**: NoSQL database that uses graphs as a native storage format
Native storage format of GraphDBs: Property Graphs

Image from: http://markorodriguez.com/2011/08/03/on-the-nature-of-pipes/
Querying property graphs with **Traversals**

Choose a set of start nodes

Describe where to walk from there

If it’s possible to walk the graph according to your description, nodes reached by the walk are returned.

Ex.: Return all objects created by people over the age of 30

» Start nodes: all nodes where “age > 30”

» Follow outgoing ‘created’ edges
Start at all nodes. Filter for “age > 30”

Gremlin traversal:
g.V.filter{ it.age > 30}
.out("created")

Image from: http://markorodriguez.com/2011/08/03/on-the-nature-of-pipes/
Following outgoing “created” edges

**Gremlin traversal:**
g.V.filter{ it.age > 30 } .out("created")

Image from: http://markorodriguez.com/2011/08/03/on-the-nature-of-pipes/
“Stored procedures”: Custom Steps:

Gremlin.defineStep("PeopleWeCanFireNow", [Vertex, Pipe], {
  _().filter{ it.age > 30 }
  .out("created").back(1)
})
Graph databases: a perfect match for social networks!

Example: Facebook Graph Search
Graph databases: a perfect match for social networks!

Example: **Facebook Graph Search**

From: http://actualfacebookgraphsearches.tumblr.com/
Graph databases: a perfect match for social networks!

» Example: Facebook Graph Search

From: http://actualfacebookgraphsearches.tumblr.com/
But they are not limited to storage of useless crap!

» Code property graphs are property graphs by design!
» You can store code property graphs in graph databases!
Allows us to define a domain specific query language for vulnerability discovery!

```
Gremlin.defineStep("PeopleWeCanFireNow", [Vertex, Pipe], {
  _().filter{ it.age > 30 }
  .out("created")
})
```
A platform for code analysis based on code property graphs stored in a graph database

- Extensible query language
- Scriptable via a python interface (python-joern)
- Shell tools for common operations (joern-tools)

Known to work on other people’s machines! ;)

http://mlsec.org/joern
(1) Import code

```python
from joern.all import JoernSteps
query = """
    ...
""
j = JoernSteps()
j.connectToDatabase()
res = j.runGremlinQuery(query)
for r in res: print r
```

(2) Start database server

(3) Run your scripts

(4) Or use the shell utils
Our first test subject: The VLC Media Player

Disclaimer:

» Auditing code for vulnerabilities is NOT about making fun of people.

» VLC developers are doing an exceptionally good job at coding.

» The fact that this is a popular program with a lot of functionality makes it interesting for us.

» Bugs have been reported and should be fixed in the next release.
Importing code and starting the server

```bash
$ joern /home/fabs/sourceCode/vlc-2.1.5/
$ neo4j console
$ firefox http://127.0.0.1:7474
```
Selecting nodes using Apache Lucene

Integrated Text Search Engine for start node selection

```
ls -lh .joernIndex
total 473M
...
[... ] 4.0K Dec 14 14:45 index
[... ] 36M Dec 14 14:45 neostore.nodestore.db
[... ] 233M Dec 14 14:45 neostore.propertystore.db
[... ] 44M Dec 14 14:45 neostore.propertystore.db.strings
[... ] 161M Dec 14 14:45 neostore.relationshipstore.db
...
```
Lucene Queries with joern-lookup

Get all files matching *demux*

```
Lucene Query Language

echo "type:File AND filepath:*demux*" | joern-lookup -c
```

<table>
<thead>
<tr>
<th>Node id</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1015093</td>
<td>filepath:.../modules/demux/tta.c  type:File</td>
</tr>
<tr>
<td>1016218</td>
<td>filepath:.../modules/demux/ty.c   type:File</td>
</tr>
<tr>
<td>1024673</td>
<td>filepath:.../modules/demux/vc1.c  type:File</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Gremlin Queries with `joern-lookup -g`

Get names of all functions in files matching *demux*

```
Custom step

Lucene Query Language

```

```
echo 'queryNodeIndex("type:File AND filepath:*demux*")
  .out().filter{it.type == "Function"}.name' |
joern-lookup -g
```

Control
Demux
Close
Open
parse_chunk_headers
get_chunk_header
...

Gremlin

\[ G = (V, E) \]
"But how do I know what is in the database?"

```bash
echo 'getFunctionsByName("GetAoutBuffer").id'
| joern-lookup -g | tail -n 1
| joern-plot-proggraph -ddg -cfg
```

Plot data flow  Plot control flow

Plot control and data flow for function GetAoutBuffer

Big thanks to Alwin Maier for writing the plotting tools!
Let’s look for a variation of the libssh bug in VLC

Get calls to `malloc` where first argument contains and additive expression and a call to `memcpy` is reached by data flow where the third argument also contains an additive expression and the two additive expression are not equal.

echo 'getCallsTo("malloc").ithArguments("0")
 .sideEffect{cnt = it.code }
 .match{ it.type == "AdditiveExpression"}.statements()
 .out("REACHES")
 .match{ it.type == "CallExpression" &&
       it.code.startsWith("memcpy")}.ithArguments("2")
 .filter{it.code != cnt }
 .match{it.type == "AdditiveExpression"}.id'
 | joern-lookup -g | sort | uniq | joern-location > foo
mp4 looks nice. I have some mp4s.

$ grep 'mp4' foo | joern-code
static int MP4_ReadBox_name( stream_t *p_stream, MP4_Box_t *p_box )
{
    MP4_READBOX_ENTER( MP4_Box_data_name_t );
    p_box->data.p_name->psz_text =
        malloc( p_box->i_size + 1 - 8 ); /* +\0, -name, -size */
    if( p_box->data.p_name->psz_text == NULL )
        MP4_READBOX_EXIT( 0 );

    memcpy( p_box->data.p_name->psz_text, p_peek, p_box->i_size - 8 );
    p_box->data.p_name->psz_text[p_box->i_size - 8] = '\0';
    ...
    MP4_READBOX_EXIT( 1 );
}
“In fact, two of your authors typically start reviewing a new codebase by finding the equivalent of the util/ directory and reading the framework and glue code line by line.”, Dowd, McDonald, Schuh (2006)

» Finding fundamental difficulties in using internal and external APIs can help uncover a lot of vulnerabilities

» Let’s checkout the API for processing of data streams in VLC

» Include/vlc_stream.h
Difficulty: Handling files larger than 4GB

- Size of a stream can be larger than UINT32_MAX

- `sizeof(size_t) == sizeof(uint32_t)` on 32 bit platforms

- Any allocation that depends on stream_Size will need to ensure that the stream_Size fits into a size_t!

```c
/**
 * Get the size of the stream.
 */
static inline int64_t stream_Size( stream_t *s )
{
    uint64_t i_pos;
    stream_Control( s, STREAM_GET_SIZE, &i_pos );
    if( i_pos >> 62 )
        return (int64_t)1 << 62;
    return i_pos;
}
```
Difficulty: Handling files larger than 4GB

» Size of a stream can be larger than \( \text{UINT32\_MAX} \)

» \( \text{sizeof(size\_t)} == \text{sizeof(uint32\_t)} \) on 32 bit

» Any allocation that depends on \( \text{stream\_Size} \) will need to ensure that the \( \text{stream\_Size} \) fits into a \text{size\_t}!

```c
/** *
 * Get the size of the stream
 */
static inline int64_t stream_Size( stream_t *s )
{
    i64_t pos;
    _Control( s, STREAM_GET_SIZE, &i_pos );
    i_pos >> 62
    return (int64_t)1 << 62;
    return i_pos;
}
```
Truncations in allocations

Give me statements containing calls to `stream_Size` and the symbol “int64_t” where data flow exists to statements containing the symbol `malloc`.

```
chunk -> data = malloc ( size )
uint8_t * data = ( uint8_t * ) malloc ( size ) ;
char * p_buffer = ( i_size > 0 ) ? malloc ( i_size ) : NULL ;
psz_update_data = malloc ( i_read + 1 ) ← Updater
uint8_t * p_buf = ( uint8_t * ) malloc ( i_size ) ;
uint8_t * p_unarmored = ( uint8_t * ) malloc ( ( i_size * 3 ) / 4 + 1 ) ;
uint8_t * p_buf = ( uint8_t * ) malloc ( i_size ) ;
```

```
echo 'getCallsTo("stream_Size").statements()
    .filter{ it.code.contains("int64_t") }.out("REACHES")
    .filter{ it.code.contains("malloc")}.code' | joern-lookup -g
```
static bool GetUpdateFile( update_t *p_update )
{
    stream_t *p_stream = NULL;
    char *psz_version_line = NULL;
    char *psz_update_data = NULL;

    p_stream = streamByUrlNew( p_update->p_libvlc, UPDATE_VLC_STATUS_URL );
    if( !p_stream )
    {
        msg_Err( p_update->p_libvlc, "Failed to open %s for reading", UPDATE_VLC_STATUS_URL );
        goto error;
    }

    const int64_t i_read = streamSize( p_stream );
    psz_update_data = malloc( i_read + 1 ); /* terminating \0 */
    if( !psz_update_data )
        goto error;

    if( streamRead( p_stream, psz_update_data, i_read ) != i_read )
    {
        msg_Err( p_update->p_libvlc, "Couldn't download update file %s", UPDATE_VLC_STATUS_URL );
        goto error;
    }
    psz_update_data[i_read] = '\0';

    streamDelete( p_stream );
    p_stream = NULL;
Side-note: Signatures are used to verify updates

```c
signature_packet_t sign;
if( download_signature( VLC_OBJECT( p_udt ), &sign,
    p_update->release.psz_url ) != VLC_SUCCESS )
{
    vlc_unlink( psz_destfile );

dialog_FatalWait( p_udt, _("File could not be verified"),
    _("It was not possible to download a cryptographic signature for "
    "the downloaded file \"%s\". Thus, it was deleted."),
    psz_destfile );
msg_Err( p_udt, "Couldn't download signature of downloaded file" );
goto end;
}

if( memcmp( sign.issuer_longid, p_update->p_pkey->longid, 8 ) )
{
    vlc_unlink( psz_destfile );
msg_Err( p_udt, "Invalid signature issuer" );
dialog_FatalWait( p_udt, _("Invalid signature"),
    _("The cryptographic signature for the downloaded file \"%s\" was "
    "invalid and could not be used to securely verify it. Thus, the "
    "file was deleted."),
    psz_destfile );
goto end;
}

if( sign.type != BINARY_SIGNATURE )
{
void GetUpdateFile( update_t *p_update )
{
    stream_t *p_stream = NULL;
    char *psz_version_line = NULL;
    char *psz_update_data = NULL;

    p_stream = stream_UrlNew( p_update->p_libvlc, UPDATE_VLC_STATUS_URL );
    if( !p_stream )
    {
        msg_Err( p_update->p_libvlc, "Failed to open %s for reading", UPDATE_VLC_STATUS_URL );
        goto error;
    }

    const int64_t i_read = stream_size( p_stream );
    psz_update_data = (char *)malloc( i_read + 1 ); /* terminating '\0' */
    if( !psz_update_data )
        goto error;

    if( stream_dealloc( p_stream, psz_update_data, i_read ) != i_read )
    {
        msg_Err( p_update->p_libvlc, "Couldn't download update file %s", UPDATE_VLC_STATUS_URL );
        goto error;
    }

    psz_update_data[i_read] = '\0';

    stream_Delete( p_stream );
    p_stream = NULL;
Let’s look at the binary straight away

```c
static bool GetUpdateFile( update_t *p_update )
{
    push    ebx
    sub     esp, 8Ch
    mov     ebx, [esp+9Ch+arg_0]
    call    vlc_savecancel
    mov     [esp+9Ch+var_7C], eax
    mov     eax, [ebx+4]
    add     eax, 4
    mov     [esp+9Ch+var_9C], eax
    call    vlc_mutex_lock
    mov     esi, [ebx+4]
    mov     edx, [esi+30h]
    mov     dword ptr [esi+20h], 0
    mov     dword ptr [esi+24h], 0
    mov     dword ptr [esi+28h], 0
    mov     [esp+9Ch+var_9C], edx
    call    free
    mov     ecx, [esi+34h]
    mov     dword ptr [esi+30h], 0
    mov     [esp+9Ch+var_9C], ecx
    call    free
    mov     edi, [ebx+4]
    mov     dword ptr [esi+34h], 0
    mov     ebp, [edi]
    mov     [esp+9Ch+var_98], offset aHttpUpdate_vid ; "http://update.videolan.org/32bit/linux/vlc-1.6.7.tar.gz"
    mov     [esp+9Ch+var_78], edi
    mov     [esp+9Ch+var_9C], ebp
    call    stream.UrlNew ; fabs: GetUpdateFile (inline)
    test    eax, eax
    mov     esi, eax
    jz      loc_73EAF0F0
```
Following the red arrow: Confirms our suspicion
» Edited ...\etc\hosts: update.videolan.org => my web server

» Created a 4294967295 Byte of “A”s (vlc/status-win-x86)

» Attached Debugger and checked for updates in VLC…
Edited \etc\hosts: update.videolan.org => my web server

Created a 4294967295 Byte of “A”s (vlc/status-win-x86)

Attached Debugger and checked for updates in VLC…

That’s convenient!
» Edited `\etc\hosts: update.videolan.org => my web server`

» Created a 4294967295 Byte of “A”s (`vlc/status-win-x86`)

» Attached Debugger and checked for updates in VLC…

Arguing in favor of patching just got a lot easier.
stream_Read is our friend 😊

- Header “Content-Length: 4294967295” triggers bug
- Attacker fully controls amount of data to actually copy
- Data is copied in blocks by stream_Read
- In between copy-operations, stream_Read uses dereferences function pointers!

- ASLR/DEP enabled but limited amount of position dependant code allows for custom ROP chain
- Downsides: multiple threads make stable exploitation difficult
- Demo.
Our second test subject: The Linux Kernel

» Large code base with a lot of users

» Well suited for static analysis as hardware for drivers is typically not available
Defining steps for more complex traversals

» **Unsanitized**

» The traversal returns attacker-controlled sources only if they meet the following two conditions.

  » There exists a path from the source statement to the sink statement in the control flow graph such that no node on the path matches any of the sanitizer descriptions.

  » A variable defined by the source and used by the sink reaches the sink via the control flow path, i.e., the variable is not redefined by any node on the path.

» Pretty complex, so let’s implement it once and the re-use it 😊

From: Modeling and Discovering Vulnerabilities with Code Property Graphs
Example: Query for buffer overflows in write handlers

query1 = "getFunctionASTsByName('*_write*')
  .getArguments('((copy_from_user OR memcpy)', '2')
  .sideEffect{ paramName = 'c(ou)?nt'; }
  .filter{ it.code.matches(paramName) }
  .unsanitized(
    { it._().or(
       _().isCheck('.*' + paramName + '.*'),
       _().codeContains('.*alloc.' + paramName + '.*'),
       _().codeContains('.*min.*')
    )}
  )
  .param( '.*c(ou)?nt.*' )
  .locations()"
query2 = "\"\"
getArguments('(copy_from_user OR memcpy)', '2')
 .filter{ !it.argToCall().toList()[0].code.matches('.*(sizeof|min).*') } 
 .sideEffect{ argument = it.code; } /* store argument */
 .sideEffect{ dstId = it.statements().toList()[0].id; } /* store id of sink */
 .unsanitized({
    it._().or(
     _().isCheck('.*' + Pattern.quote(argument) + '.*') ,
     _().codeContains('.*alloc.*' + Pattern.quote(argument) + '.*'),
     _().codeContains('.*min.*')
    )
  }, { it._().filter{ it.code.contains('copy_from_user')}} }
)
 .filter{ it.id != dstId } /* filter flows from node to self */
 .locations()"""
Running queries 1 and 2

<table>
<thead>
<tr>
<th>Filename</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch/um/kernel/exitcode.c</td>
<td>exitcode_proc_write</td>
</tr>
<tr>
<td>security/smack/smackfs.c</td>
<td>smk_write_rules_list</td>
</tr>
<tr>
<td>drivers/staging/ozwpan/ozcdev.c</td>
<td>oz_cdev_write</td>
</tr>
<tr>
<td>drivers/infiniband/hw/ipath/ipath_diag.c</td>
<td>ipath_diagpkt_write</td>
</tr>
<tr>
<td>drivers/infiniband/hw/qib/qib_diag.c</td>
<td>qib_diagpkt_write</td>
</tr>
<tr>
<td>drivers/scsi/megaraid/megaraid_mm.c</td>
<td>mimd_to_kioc</td>
</tr>
<tr>
<td>drivers/scsi/megaraid.c</td>
<td>megadev_ioctl</td>
</tr>
<tr>
<td>drivers/char/xilinx_.../xilinx_hwicap.c</td>
<td>hwicap_write</td>
</tr>
<tr>
<td>drivers/s390/net/-geth_core_main.c</td>
<td>qeth_snmp_command</td>
</tr>
<tr>
<td>drivers/staging/wlags49_h2/wl_priv.c</td>
<td>wlan_uil_put_info</td>
</tr>
<tr>
<td>arch/ia64/sn/kernel/sn2/sn_hwperf.c</td>
<td>sn_hwperf_ioctl</td>
</tr>
</tbody>
</table>
int qeth_snmp_command(struct qeth_card *card, char __user *udata) {
    struct qeth_cmd_buffer *iob;
    struct qeth_ipa_cmd *cmd;
    struct qeth_snmp_ureq *ureq;
    int req_len;
    struct qeth_arp_query_info qinfo = {0, 0};
    int rc = 0;

    // [...]

    /* skip 4 bytes (data_len struct member) to get req_len */
    if (copy_from_user(&req_len, udata + sizeof(int), sizeof(int)))
        return -EFAULT;

    ureq = memdup_user(udata, req_len + sizeof(struct qeth_snmp_ureq_hdr));
    if (IS_ERR(ureq)) {
        QETH_CARD_TEXT(card, 2, "snmpnhome");
        return PTR_ERR(ureq);
    }

    qinfo.ldata_len = ureq->hdr.data_len;
    qinfo.ldata = kzalloc(qinfo.ldata_len, GFP_KERNEL);
    if (!qinfo.ldata) {
        kfree(ureq);
        return -ENOMEM;
    }

    // [...]
    memcpy(&cmd->data.setadapterparms.data.snmp, &ureq->cmd, req_len);
    // [...]
    return rc;
» Get somebody working in the industry to actually try this!
  » Found about 100 issues in an internal audit at Qualcomm in their kernel code (9 queries)

» Evaluation for paper: audit of the Linux Kernel Mainline
  » Buffer overflows (2 queries)
  » Zero-byte allocation (1 query)
  » Memory mapping bugs (1 query)
  » Memory disclosure (1 query)
### Kernel 0-day 😊

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
<th>Developer Feedback</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Overflow</td>
<td>arch/um/kernel/exitcode.c</td>
<td>Fixed</td>
<td>CVE-2013-4512</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>drivers/staging/ozwpan/ozcdev.c</td>
<td>Fixed</td>
<td>CVE-2013-4513</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>drivers/s390/net/qeth_core_main.c</td>
<td>Fixed</td>
<td>CVE-2013-6381</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>drivers/staging/wlags49_h2/wl_priv.c</td>
<td>Fixed</td>
<td>CVE-2013-4514</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>drivers/scsi/megaraid/megaraid_mm.c</td>
<td>Fixed</td>
<td></td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>drivers/infiniband/hw/ipath/ipath_diag.c</td>
<td>Fixed</td>
<td></td>
</tr>
<tr>
<td>Memory Disclosure</td>
<td>drivers/staging/bcm/Bcmchar.c</td>
<td>Fixed</td>
<td>CVE-2013-4515</td>
</tr>
<tr>
<td>Memory Disclosure</td>
<td>drivers/staging/sb105x/sb_pci_mp.c</td>
<td>Fixed</td>
<td>CVE-2013-4516</td>
</tr>
<tr>
<td>Memory Mapping</td>
<td>drivers/video/au1200fb.c</td>
<td>Fixed</td>
<td>CVE-2013-4511</td>
</tr>
<tr>
<td>Memory Mapping</td>
<td>drivers/video/au1100fb.c</td>
<td>Fixed</td>
<td>CVE-2013-4511</td>
</tr>
<tr>
<td>Memory Mapping</td>
<td>drivers/uio/uio.c</td>
<td>Fixed</td>
<td>CVE-2013-4511</td>
</tr>
<tr>
<td>Memory Mapping</td>
<td>drivers/staging/.../drv_interface.c</td>
<td>Fixed under way</td>
<td></td>
</tr>
<tr>
<td>Zero-byte Allocation</td>
<td>fs/xfs/xfs_ioctl1.c</td>
<td>Fixed</td>
<td>CVE-2013-6382</td>
</tr>
<tr>
<td>Zero-byte Allocation</td>
<td>fs/xfs/xfs_ioctl132.c</td>
<td>Fixed</td>
<td>CVE-2013-6382</td>
</tr>
<tr>
<td>Zero-byte Allocation</td>
<td>drivers/net/wireless/libertas/debugfs.c</td>
<td>Fixed</td>
<td>CVE-2013-6378</td>
</tr>
<tr>
<td>Zero-byte Allocation</td>
<td>drivers/scsi/aacraid/commctrl.c</td>
<td>Fixed</td>
<td>CVE-2013-6380</td>
</tr>
</tbody>
</table>

» 18 vulnerabilities, acknowledged / fixed by developers
Conclusion

» Introduced a system to mine code bases for vulnerabilities
» Built a bridge between program analysis and graph DBs
» Found real exploitable bugs in popular code bases
» Part of a larger effort to explore what we can do with pattern recognition and data mining techniques to discover vulnerabilities
Thank you! Questions?

» Implementation
  » http://mlsec.org/joern (webpage + documentation)
  » http://github.com/fabsx00/joern (main system)
  » http://github.com/fabsx00/python-joern (traversals)
  » http://github.com/fabsx00/joern-tools (shell utils)

» Fabian Yamaguchi
  » fabs@goesec.de
  » http://codeexploration.blogspot.de (my webpage)
  » Twitter: @fabsx00