An rsyslog case study

Bug hunting with CodeQL

Presented by @agustingianni

Approaching the unknown

Methodology

- Give an answer to these questions:
  - **What** does the target do?
    - Read the docs
  - **How** does it do it?
    - Read the code
  - **Where** can we **influence execution**?
    - Query the code

Exploratory queries

```
import cpp

class ReadFunctionCall extends FunctionCall {
    ReadFunctionCall() {
        this.getTarget().getName() = "pread" or
        this.getTarget().getName() = "read" or
        this.getTarget().getName() = "readv" or
        this.getTarget().getName() = "recvfrom" or
        this.getTarget().getName() = "recvmsg" or
        this.getTarget().getName() = "recv"
    }
}

from ReadFunctionCall call
select call.getFile(), call.getEnclosingFunction(), call
```

```
rsRetVal
processPacket(uchar *rcvBuf, ssize_t lenRcvBuf)
{
    smsg_t *pMsg;
    msgConstructWithTime(&pMsg, stTime, ttGenTime);
    MsgSetRawMsg(pMsg, (char*)rcvBuf, lenRcvBuf);
    ratelimitAddMsg(lstn->ratelimiter, multiSub, pMsg);
}
```

```
void MsgSetRawMsg(smsg_t *pThis, char *pszRawMsg,
    size_t lenMsg)
{
    pThis->pszRawMsg = malloc(pThis->iLenRawMsg + 1);
    memcpy(pThis->pszRawMsg, pszRawMsg, pThis->iLenRawMsg);
    pThis->pszRawMsg[pThis->iLenRawMsg] = '\0';
}
```

```
typedef struct msg smsg_t;

struct msg {
    int     offAfterPRI;
    int     offMSG;
    int     iLenRawMsg;
    Uchar  *pszRawMsg;
    Uchar   szRawMsg[CONF_RAWMSG_SIZE];
};
```

Data flow exploration

Where does our data go?

- Our data ends up in a **msg_t** object.
- Hey Siri, err, CodeQL: **where is our data processed?**

```
import cpp

class RAccess extends FieldAccess {
    RAccess() {
        this.getTarget().getName() = "pszRawMsg"
    }
}

class RFunction extends Function {
    RFunction() {
        any(RAccess access).getEnclosingFunction() = this
    }
}

from RFunction access
select access.getFile(), access
```

Results!

CVE-2019-17041

- Heap memory corruption in pmaixforwardedfrom.c
- Reads AIX log messages and converts them into a standard log.
- A specially constructed log message leads to a heap overflow.

```
while(lenMsg && *p2parse != ' ' && *p2parse != ':') {
    --lenMsg;
    ++p2parse;
}
if (lenMsg && *p2parse != ':') {
    ABORT();
}
lenMsg -=1;
memmove(p2parse, p2parse + 1, lenMsg);
```

Variant Analysis

Modeling CVE-2019-17041

- Abstract the seed vulnerability.
  - Use of pszRawMsg inside a loop.
- Translate into a CodeQL query.

```
class RawMessageFieldAccess extends FieldAccess {
    RawMessageFieldAccess() {
        this.getTarget().getName() = "pszRawMsg"
    }
}

from
    DataFlow::Node source,
    DataFlow::Node sink,
    RawMessageFieldAccess access, WhileStmt loop
where
    TaintTracking::localTaint(source, sink) and
    source.asExpr() = access and
    sink.asExpr() = loop.getCondition().getAChild*()
select
    "Loop iterates data from:", source, sink
```

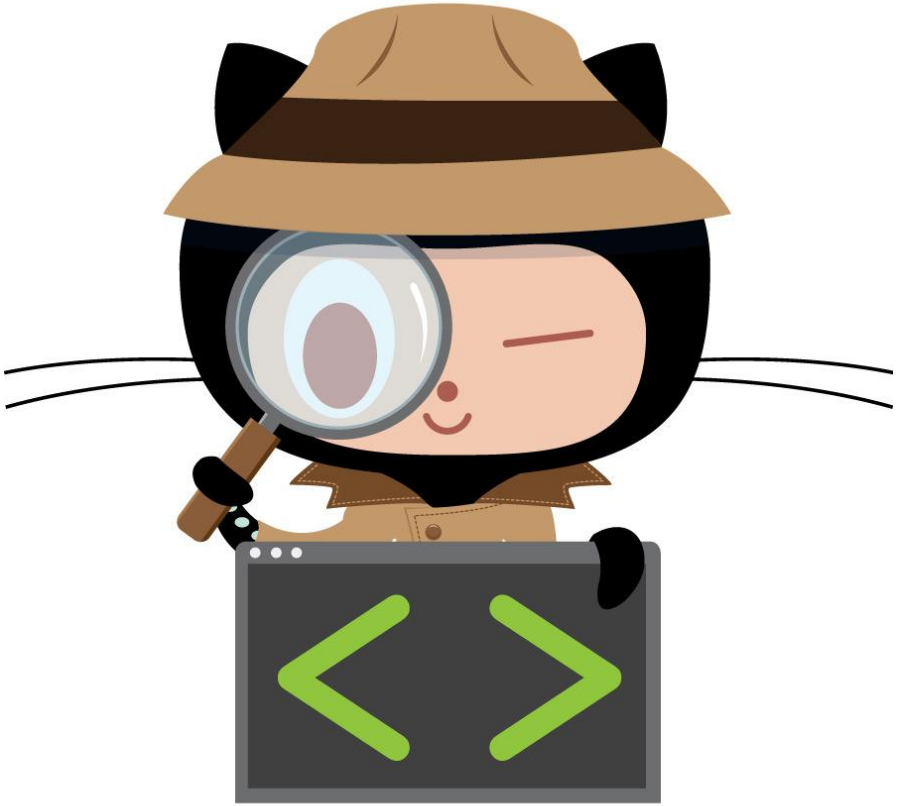
CVE-2019-17042

- Heap memory corruption in pmcisconames.c
- Variant of CVE-2019-17041.

```
while(lenMsg && *p2parse != ' ') {
    --lenMsg;
    ++p2parse;
}

lenMsg -=1;
p2parse +=1;

if(strncasecmp(p2parse, OpeningText, sizeof(OpeningText)-1) != 0) {
    ABORT();
}

lenMsg -=2;
memmove(p2parse, p2parse + 2, lenMsg);
```

Why CodeQL?

- Manual code review
  - Time consuming
  - Large surface to explore
  - Limited time
- Incomplete results
  - Missed anything?
- Does not escale
- Augment your hability



More details @

https://securitylab.github.com/research/bug-hunting-codeql-rsyslog

Questions? Concerns? Comments?

@agustingianni