

Laika BOSS: File-Centric Intrusion Detection System



PRESENTED BY

Wellington Lee, Cybersecurity R&D

File-Centric IDS

Laika BOSS

Examples

Why Laika BOSS?

Installation

Laika BOSS Development

IDS that analyzes files (objects transferred through network)

- Scan
 - (yara, xor-search, pdfrate, etc)
- Decode
 - (zip, MIME, OLE, etc)
- Metadata
 - (document author, PE section name, image size, etc)

Benefits

- Address attacks visible in network payloads vs. network protocols
- Consistency/re-use of content (mail, web, bulk malware, analyst driven)
- Ease of detection content creation/deployment

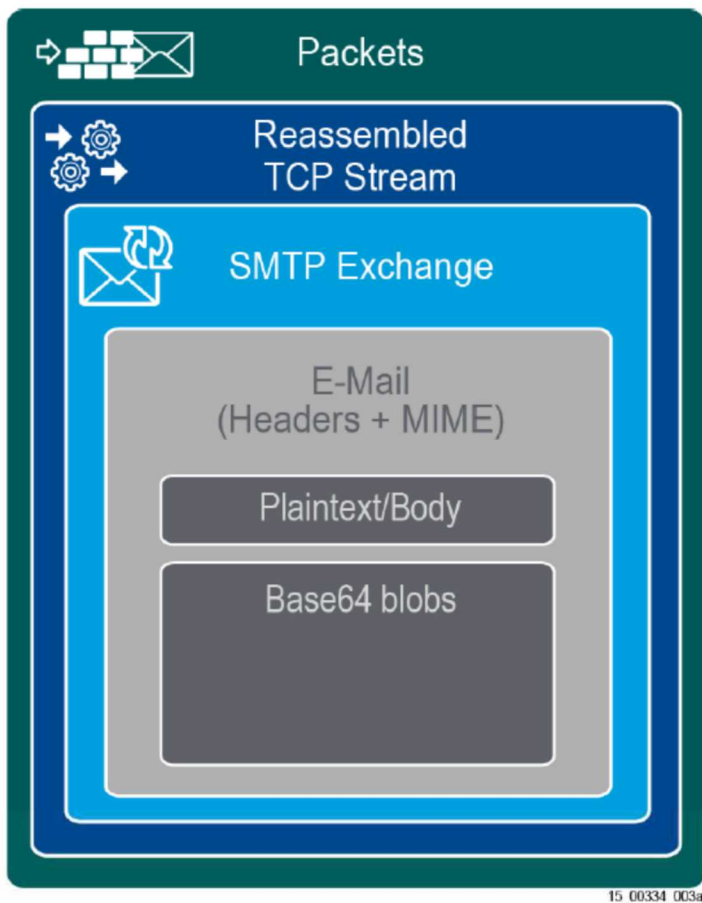


Figure 3: Traditional NIDS view of a malicious email with payload hidden through encoding

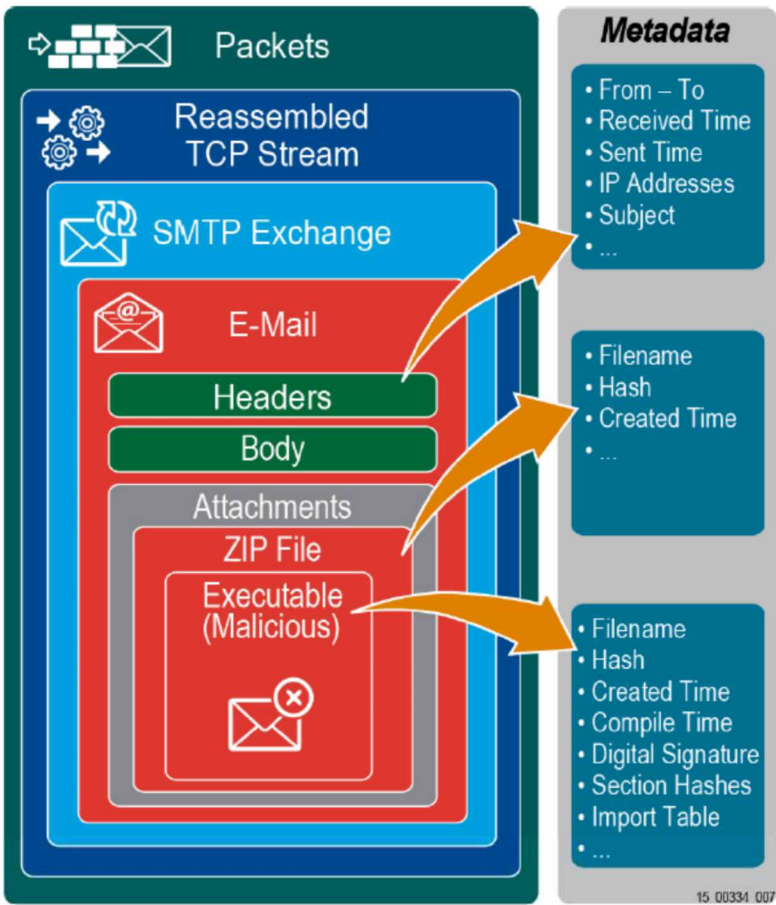


Figure 4: Laika IDS view of an exploded email, showing a malicious executable contained within a ZIP file and extracted metadata

5 Laika BOSS Overview

File Centric IDS

Designed for email, web, bulk file, and analyst directed scanning

Modules for each file analyzer

Implemented in Python

Relies heavily on Yara for signature matching, configuration

Highly Scalable

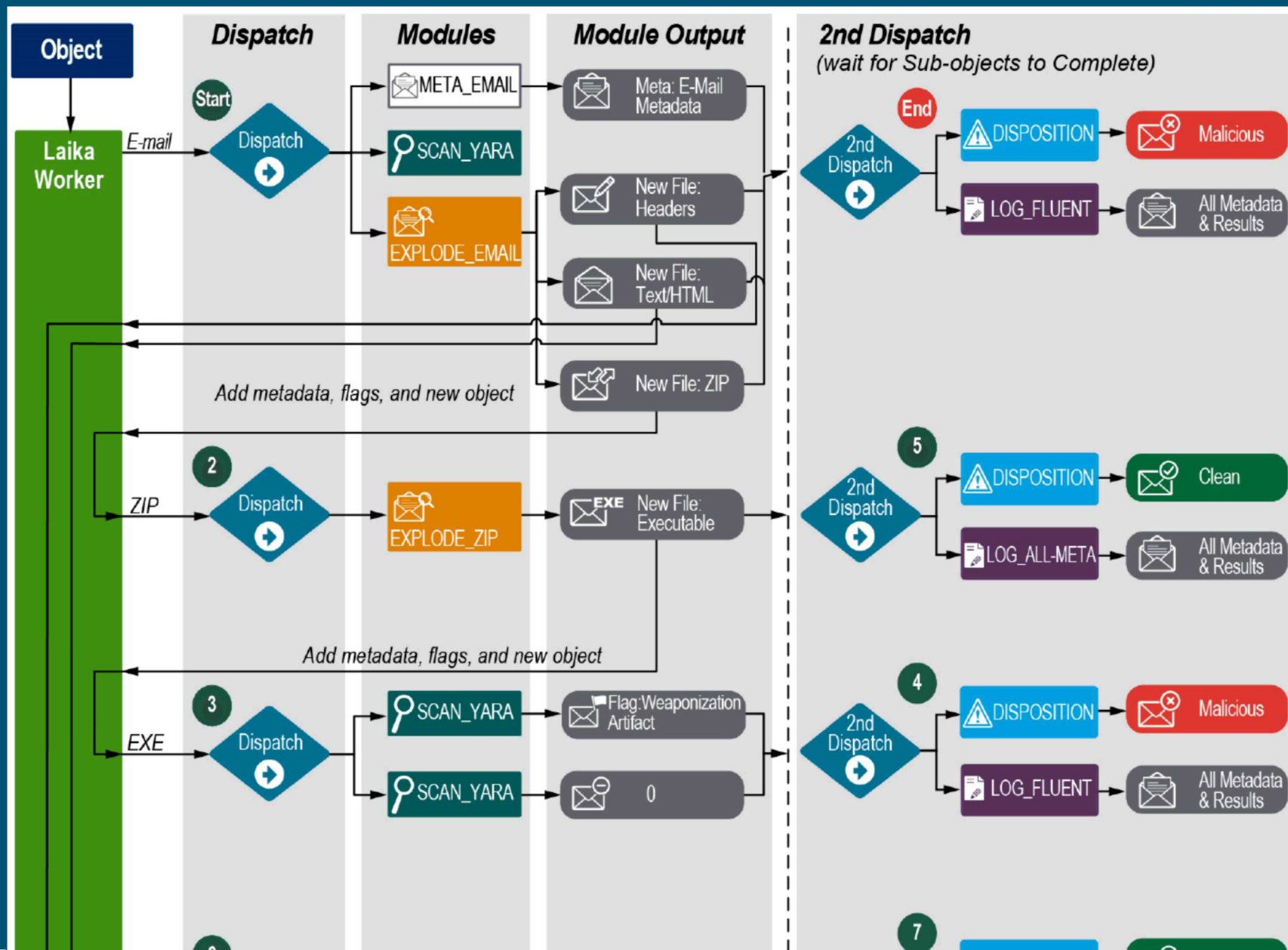
Inline (Blocking Mode) Capable

Strong Configurability, Great Transparency, Good Enough Performance

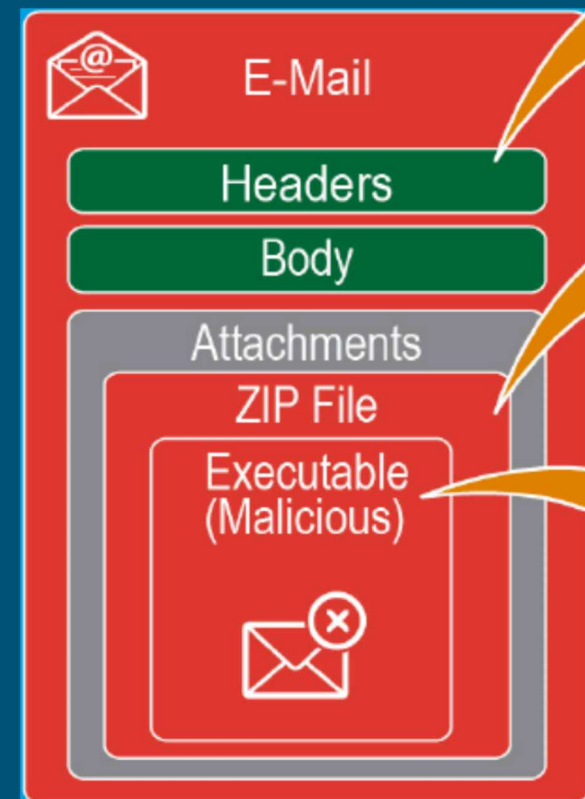
Open Source by Lockheed Martin



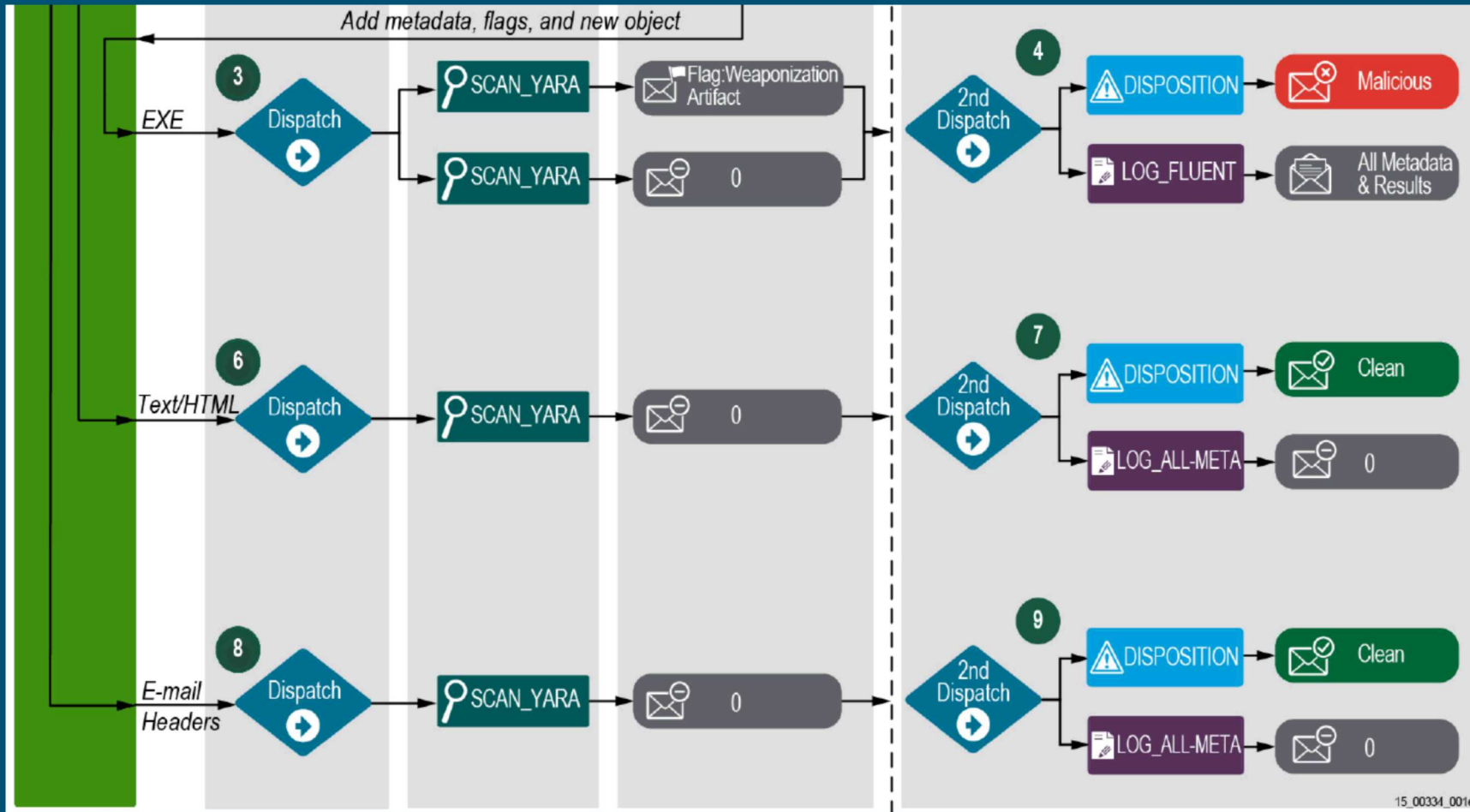
Laika BOSS Recursive Scanning



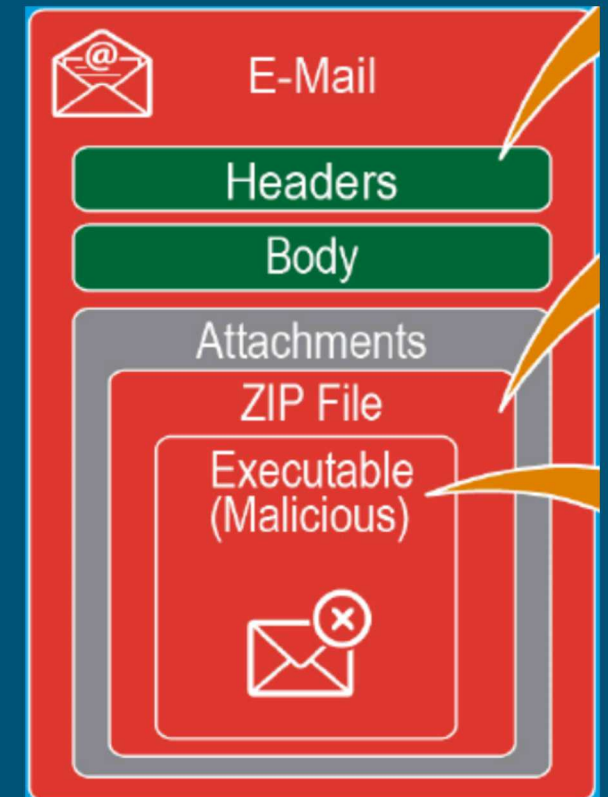
Malicious PE in Zip as Email Attachment



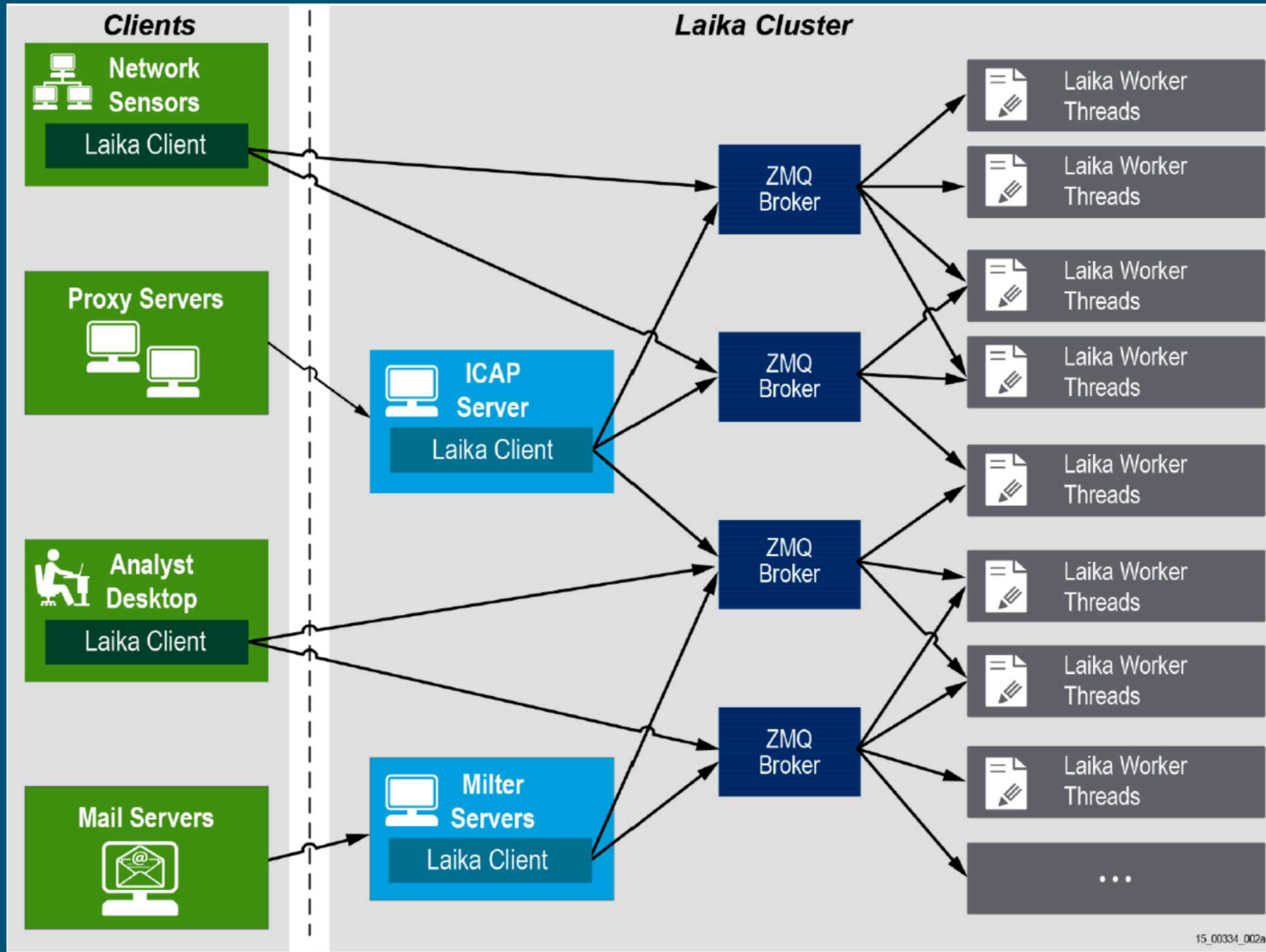
Laika BOSS Recursive Scanning (cont.)



Malicious PE in Zip as Email Attachment



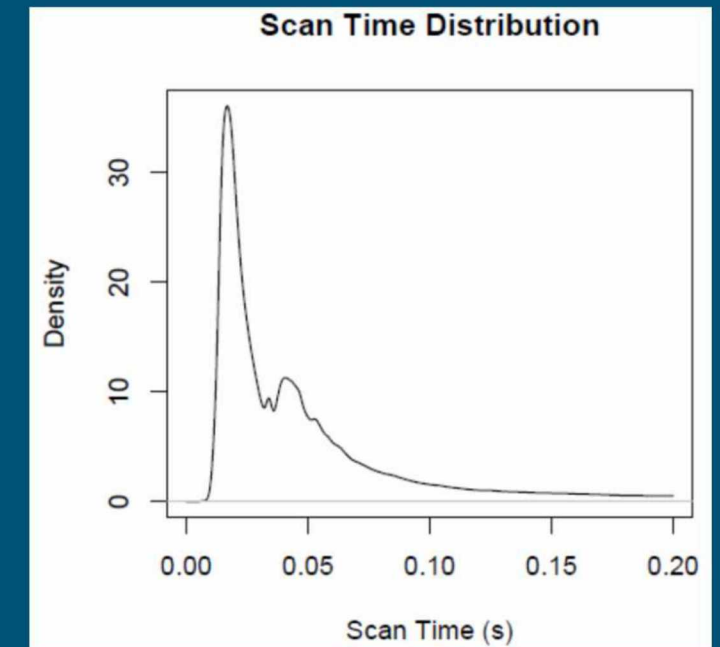
Laika BOSS Scalability, Versatility, Performance



Example Enterprise Deployment

16 servers x
24 cores / server

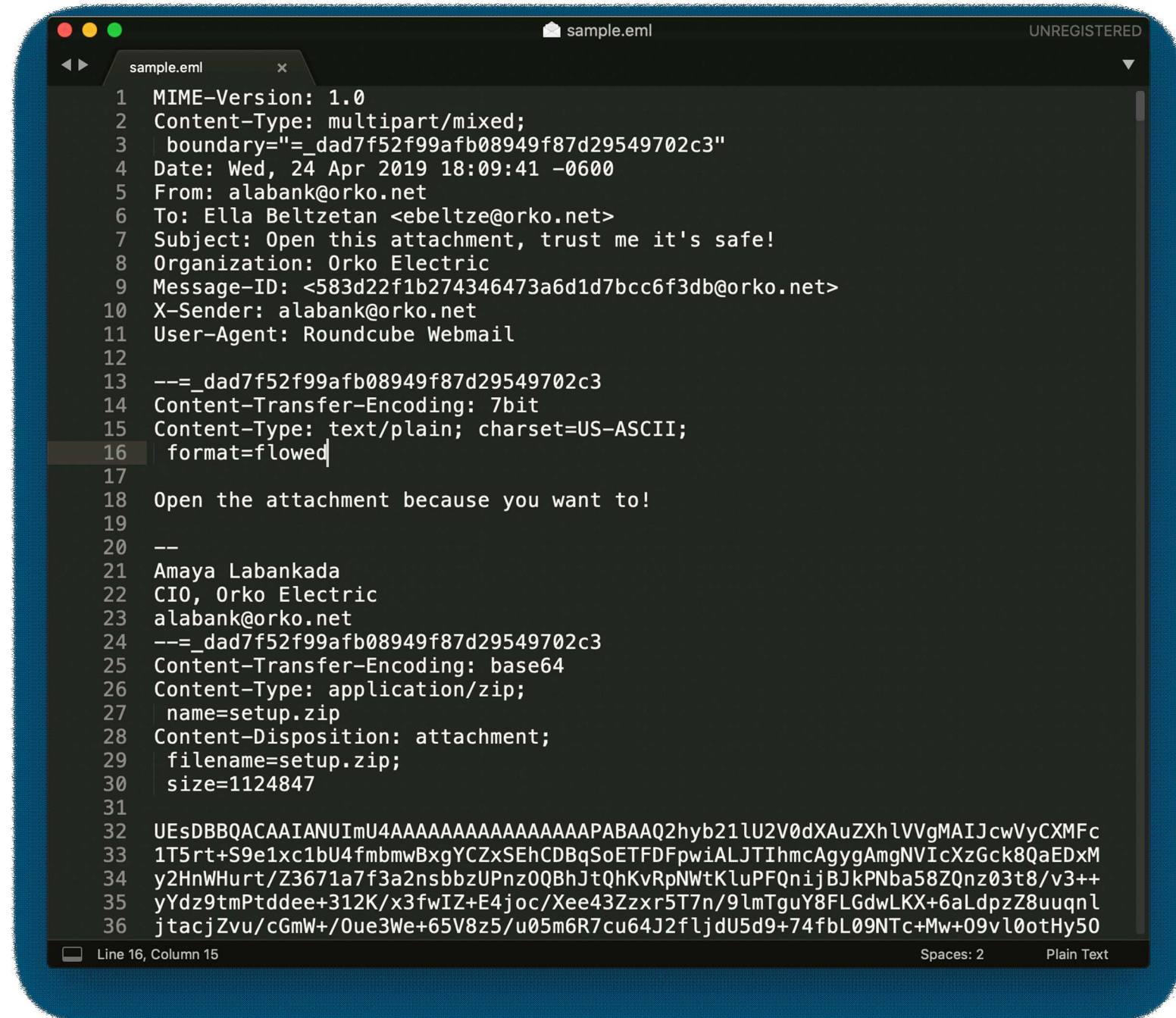
384 scanning cores



Credit: Lockheed Martin Laika BOSS whitepaper

Zip with Exe

Email -> Zip -> Exe



The screenshot shows a webmail interface for 'sample.eml'. The email header includes fields for MIME-Version, Content-Type, Date, From, To, Subject, Organization, Message-ID, X-Sender, and User-Agent. The body of the email contains a text message followed by a multipart boundary separator. The second part of the multipart message is a zip file attachment with the filename 'setup.zip' and a size of 1124847 bytes. The email is displayed in a dark-themed editor with line numbers on the left and status information at the bottom.

```
1 MIME-Version: 1.0
2 Content-Type: multipart/mixed;
3   boundary="=_dad7f52f99afb08949f87d29549702c3"
4 Date: Wed, 24 Apr 2019 18:09:41 -0600
5 From: alabank@orko.net
6 To: Ella Beltzetan <ebeltze@orko.net>
7 Subject: Open this attachment, trust me it's safe!
8 Organization: Orko Electric
9 Message-ID: <583d22f1b274346473a6d1d7bcc6f3db@orko.net>
10 X-Sender: alabank@orko.net
11 User-Agent: Roundcube Webmail
12
13 --=_dad7f52f99afb08949f87d29549702c3
14 Content-Transfer-Encoding: 7bit
15 Content-Type: text/plain; charset=US-ASCII;
16   format=flowed
17
18 Open the attachment because you want to!
19
20 --
21 Amaya Labankada
22 CIO, Orko Electric
23 alabank@orko.net
24 --=_dad7f52f99afb08949f87d29549702c3
25 Content-Transfer-Encoding: base64
26 Content-Type: application/zip;
27   name=setup.zip
28 Content-Disposition: attachment;
29   filename=setup.zip;
30   size=1124847
31
32 UEsDBBQACAAIANUIU4AAAAAAAAAAAAAAAAAPABAAQ2hyb21lU2V0dXAuZXhlLVVgMAIJcwVyCXMFc
33 1T5rt+S9e1xc1bU4fmbmwBxgYCZxSEhCDBqSoETDFDfwiALJTIhmcAgygAmgNVicXzGck8QaEDxM
34 y2HnWHurt/Z3671a7f3a2nsbbzUPnz0QBhJtQhKvRpNWtKluPFQnijBJkPNba58ZQnz03t8/v3++
35 yYdz9tmPtddee+312K/x3fwIZ+E4joc/Xee43Zzxr5T7n/9lmtGuY8FLGdwLKX+6aLdpzZ8uuqnI
36 jtacjZvu/cGmW+/0ue3We+65V8z5/u05m6R7cu64J2fljdU5d9+74fbL09NTc+Mw+09vI0otHy50
```

Line 16, Column 15 Spaces: 2 Plain Text

Relevant Modules

META_EMAIL:

- Metadata on email

```
"META_EMAIL": {  
    "Attachments": [  
        "setup.zip"  
    ]  
},
```

EXPLODE_EMAIL:

- Extract out parts and attachments of email

EXPLODE_ZIP:

- Extract files from zip archives

```
"EXPLODE_ZIP": {  
    "Total_Files": 3,  
    "Unzipped": 3  
},
```

Relevant Modules (cont.)

META_PE:

- Extract metadata from PE files

```
"META_PE": {  
  "Subsystem": "IMAGE_SUBSYSTEM_WINDOWS_GUI",  
  "Image Characteristics": [  
    "IMAGE_FILE_32BIT_MACHINE",  
    "IMAGE_FILE_EXECUTABLE_IMAGE"  
  ],  
  "Timestamp": 1552947293,  
  "Stack Commit Size": 4096,  
  "ImageBase": "0x400000",  
  "Heap Commit Size": 4096,  
  "Heap Reserve Size": 1048576,  
  "Machine Type": {  
    "Type": "IMAGE_FILE_MACHINE_I386",  
    "Id": 332  
  },  
  "Rich Header": {  
    "Checksum": 652346176,  
    "Hashes": {  
      "SHA1": "4ebbf2e70992243e0fad5a10eb5dd74afa790e10",  
      "SHA256": "77079f70f0a4b3326a7f3f8ded4d73587512cfc63739d6e10f98dbe81bcbd4bc",  
    }  
  }  
}
```



```
rule type_is_email
{
    meta:
        scan_modules = "META_EMAIL EXPLODE_EMAIL"
        file_type = "eml"
    strings:
        $from = "From "
        $received = "\x0aReceived:"
        $return = "\x0aReturn-Path:"
    condition:
        (not ext_sourceModule contains "EXPLODE_EMAIL") and
        (($from at 0) or
        ($received in (0 .. 2048)) or
        ($return in (0 .. 2048)))
}
```



```
rule type_is_zip
{
    meta:
        scan_modules = "EXPLODE_ZIP(filelimit=1000)"
        file_type = "zip"
    condition:
        uint32(0) == 0x04034b50 and not uint32(4) == 0x00060014
}
```

```
rule type_is_mz
{
    meta:
        scan_modules = "META_PE"
        file_type = "pe"
    condition:
        uint16(0) == 0x5a4d
        and not ext_sourceModule contains "META_PE"
}
```

Why Laika BOSS?

Community Platform/Sharing Vision

- Community development, support of core framework
- Modules can be shared at all levels

Weaknesses

- Python 2
- Often not pythonic (ex. test cases)
- Submissions API, queueing, logging, etc. could be more modular

Alternatives

- stoQ <https://stoq.punchcyber.com/>
- Strelka <https://github.com/target/strelka>
- File Scanning Framework (FSF) <https://github.com/EmersonElectricCo/fsf>

Installation

<https://github.com/lmco/laikaboss#getting-started>

Install dependencies (depends slightly on Linux version)

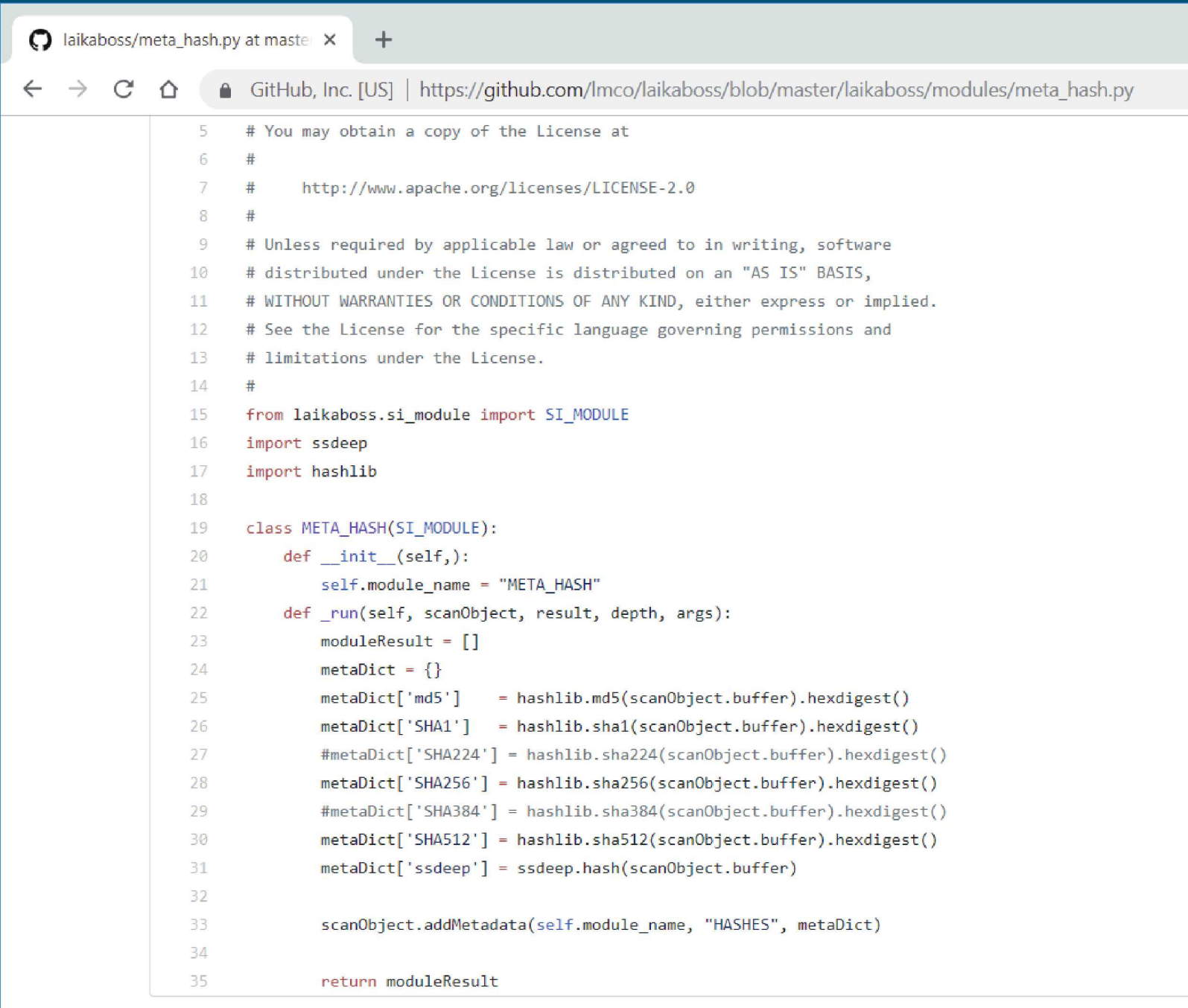
Download and use Laika BOSS

Modules have 3 primary actions

- Extract Metadata (META_*)
- Decode subfiles (EXPLODE_*)
- Flag (~alert) conditions (SCAN_*)

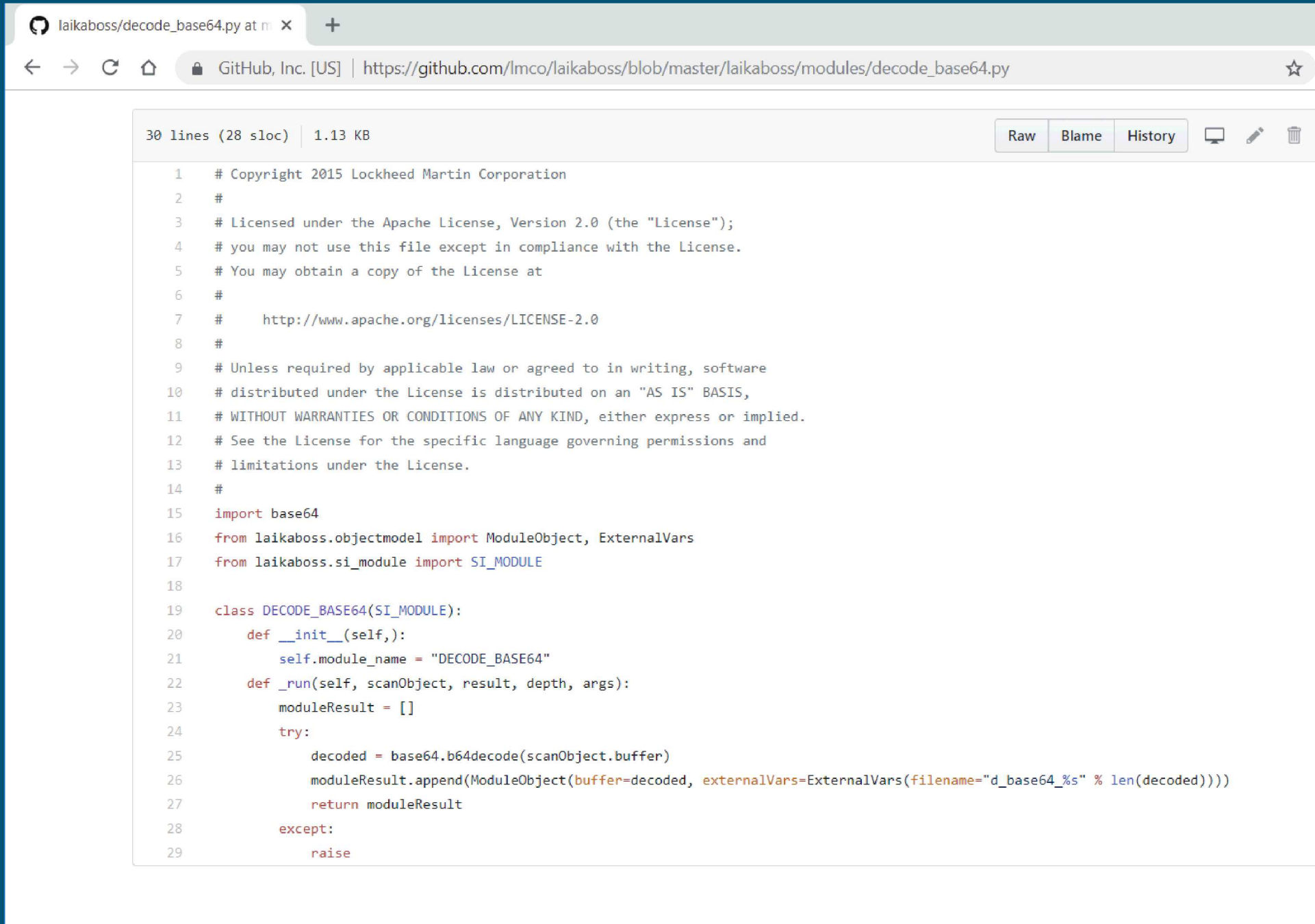
Secondary Actions

- Format/Log/Store data (LOG_*, STORE_*)
- Correlations/Lookups (LOOKUP_*)



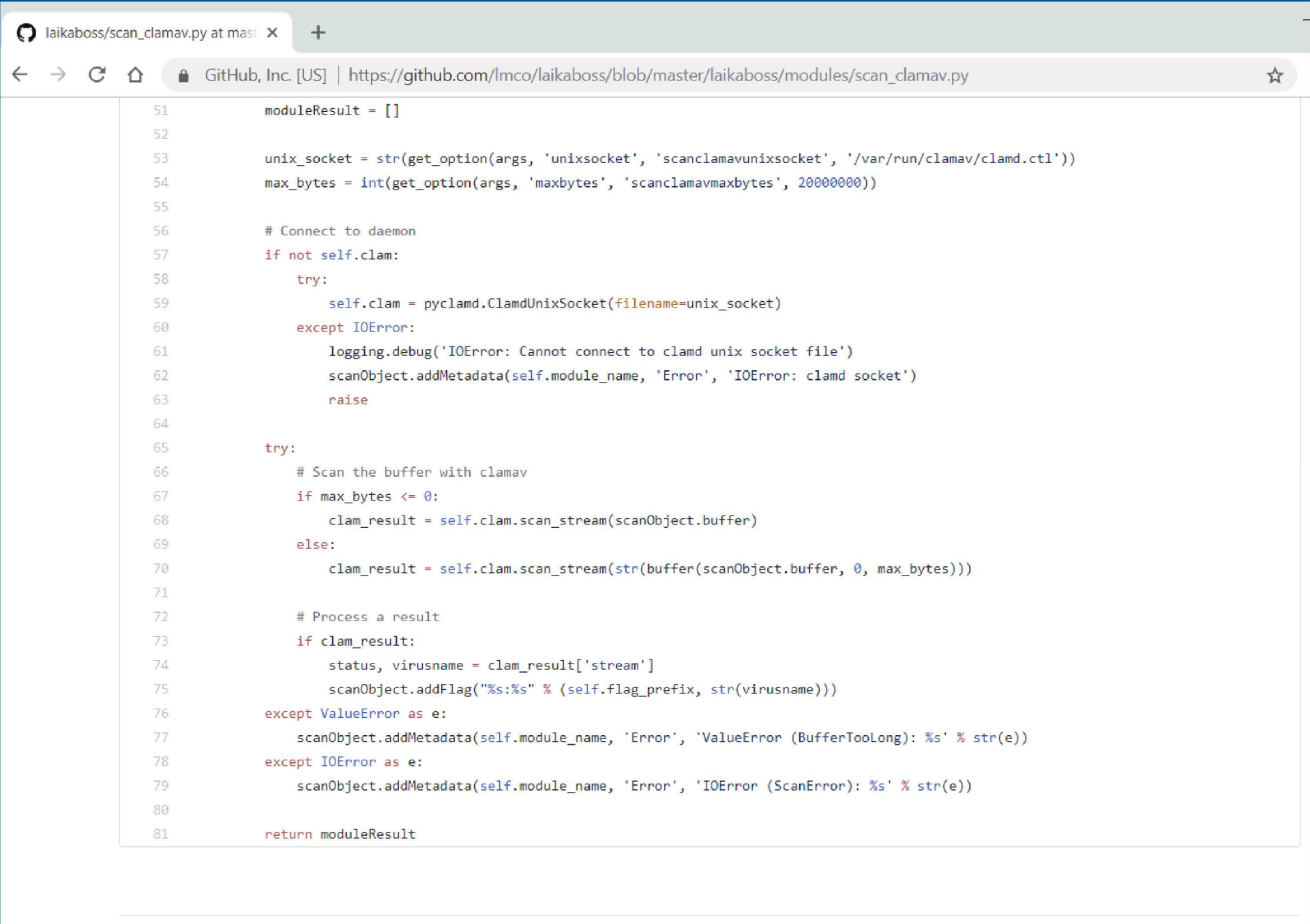
A screenshot of a web browser displaying a GitHub repository file. The browser's address bar shows the URL `https://github.com/lmco/laikaboss/blob/master/laikaboss/modules/meta_hash.py`. The page title is `laikaboss/meta_hash.py at master`. The code is a Python file with line numbers 5 through 35. It contains a license notice, imports for `SI_MODULE`, `ssdeep`, and `hashlib`, and a `META_HASH` class with `__init__` and `_run` methods. The `_run` method calculates various hashes (md5, SHA1, SHA224, SHA256, SHA384, SHA512, and ssdeep) for a given scan object and adds them as metadata.

```
5 # You may obtain a copy of the License at
6 #
7 # http://www.apache.org/licenses/LICENSE-2.0
8 #
9 # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the License for the specific language governing permissions and
13 # limitations under the License.
14 #
15 from laikaboss.si_module import SI_MODULE
16 import ssdeep
17 import hashlib
18
19 class META_HASH(SI_MODULE):
20     def __init__(self,):
21         self.module_name = "META_HASH"
22     def _run(self, scanObject, result, depth, args):
23         moduleResult = []
24         metaDict = {}
25         metaDict['md5'] = hashlib.md5(scanObject.buffer).hexdigest()
26         metaDict['SHA1'] = hashlib.sha1(scanObject.buffer).hexdigest()
27         #metaDict['SHA224'] = hashlib.sha224(scanObject.buffer).hexdigest()
28         metaDict['SHA256'] = hashlib.sha256(scanObject.buffer).hexdigest()
29         #metaDict['SHA384'] = hashlib.sha384(scanObject.buffer).hexdigest()
30         metaDict['SHA512'] = hashlib.sha512(scanObject.buffer).hexdigest()
31         metaDict['ssdeep'] = ssdeep.hash(scanObject.buffer)
32
33         scanObject.addMetadata(self.module_name, "HASHES", metaDict)
34
35         return moduleResult
```



The screenshot shows a web browser window with a single tab titled "laikaboss/decode_base64.py at m x". The address bar displays the URL "https://github.com/lmco/laikaboss/blob/master/laikaboss/modules/decode_base64.py". The page content shows a Python file with 30 lines (28 sloc) and a size of 1.13 KB. The file content is as follows:

```
1  # Copyright 2015 Lockheed Martin Corporation
2  #
3  # Licensed under the Apache License, Version 2.0 (the "License");
4  # you may not use this file except in compliance with the License.
5  # You may obtain a copy of the License at
6  #
7  #     http://www.apache.org/licenses/LICENSE-2.0
8  #
9  # Unless required by applicable law or agreed to in writing, software
10 # distributed under the License is distributed on an "AS IS" BASIS,
11 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
12 # See the license for the specific language governing permissions and
13 # limitations under the License.
14 #
15 import base64
16 from laikaboss.objectmodel import ModuleObject, ExternalVars
17 from laikaboss.si_module import SI_MODULE
18
19 class DECODE_BASE64(SI_MODULE):
20     def __init__(self,):
21         self.module_name = "DECODE_BASE64"
22     def _run(self, scanObject, result, depth, args):
23         moduleResult = []
24         try:
25             decoded = base64.b64decode(scanObject.buffer)
26             moduleResult.append(ModuleObject(buffer=decoded, externalVars=ExternalVars(filename="d_base64_%s" % len(decoded))))
27             return moduleResult
28         except:
29             raise
```



The screenshot shows a web browser window with a single tab titled "laikaboss/scan_clamav.py at mast...". The address bar displays "GitHub, Inc. [US] | https://github.com/lmco/laikaboss/blob/master/laikaboss/modules/scan_clamav.py". The main content area shows a Python code file with line numbers 51 through 81. The code defines a function that interacts with a ClamAV daemon via a Unix socket, scans a buffer, and processes the results, adding metadata and flags to a scan object.

```
51     moduleResult = []
52
53     unix_socket = str(get_option(args, 'unixsocket', 'scanclamavunixsocket', '/var/run/clamav/clamdctl'))
54     max_bytes = int(get_option(args, 'maxbytes', 'scanclamavmaxbytes', 20000000))
55
56     # Connect to daemon
57     if not self.clam:
58         try:
59             self.clam = pyclamd.ClamdUnixSocket(filename=unix_socket)
60         except IOError:
61             logging.debug('IOError: Cannot connect to clamd unix socket file')
62             scanObject.addMetadata(self.module_name, 'Error', 'IOError: clamd socket')
63             raise
64
65     try:
66         # Scan the buffer with clamav
67         if max_bytes <= 0:
68             clam_result = self.clam.scan_stream(scanObject.buffer)
69         else:
70             clam_result = self.clam.scan_stream(str(buffer(scanObject.buffer, 0, max_bytes)))
71
72     # Process a result
73     if clam_result:
74         status, virusname = clam_result['stream']
75         scanObject.addFlag("%s:%s" % (self.flag_prefix, str(virusname)))
76     except ValueError as e:
77         scanObject.addMetadata(self.module_name, 'Error', 'ValueError (BufferTooLong): %s' % str(e))
78     except IOError as e:
79         scanObject.addMetadata(self.module_name, 'Error', 'IOError (ScanError): %s' % str(e))
80
81     return moduleResult
```

Dispatching

Use Yara signatures to determine which modules to run on a given (sub)file

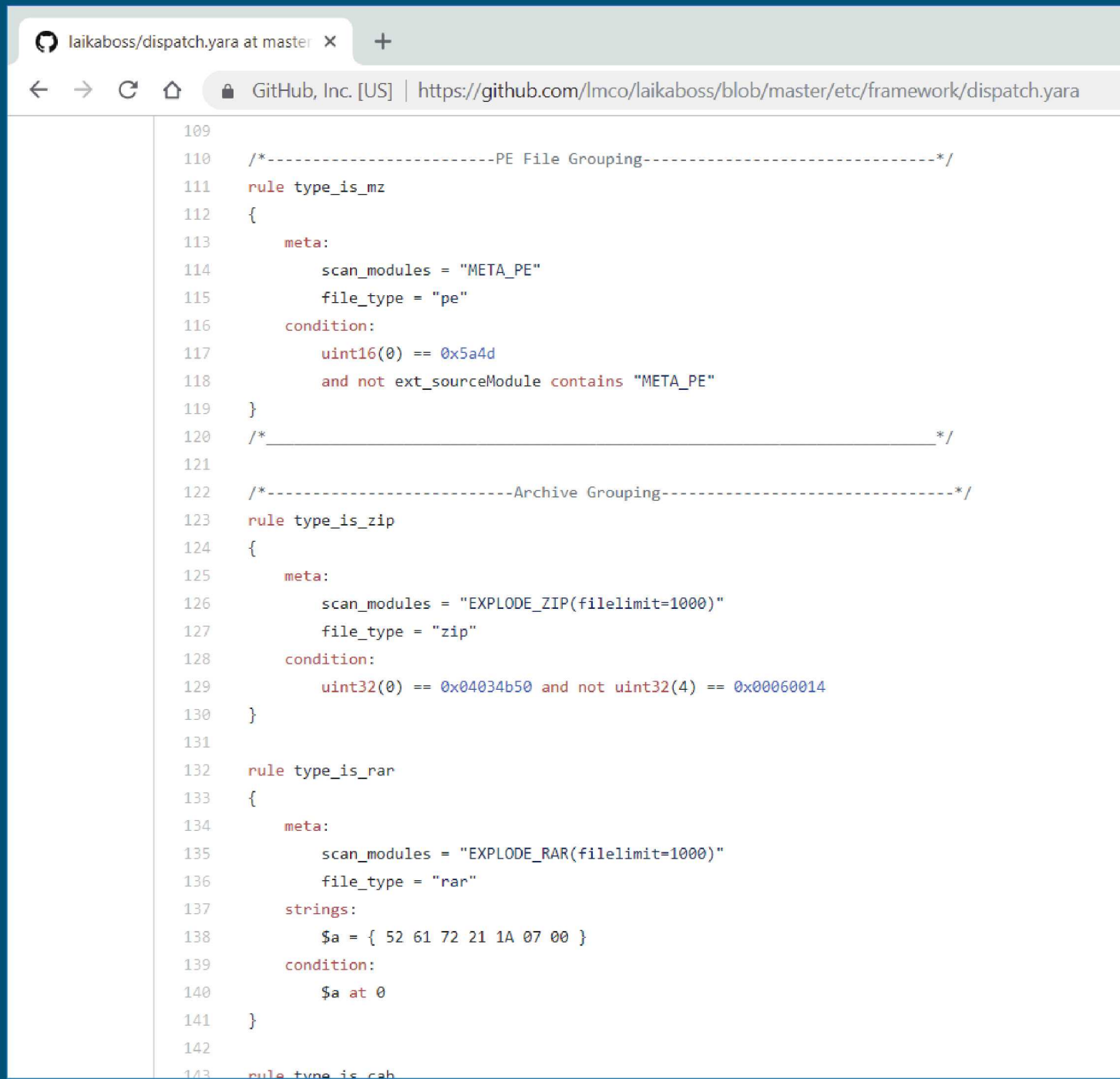
Include some metadata to increase flexibility

Primary Dispatch (dispatch.yara)

- Runs on files
- Most signatures based on file magic number

Secondary Dispatch (conditional-dispatch.yara)

- Runs on concatenation of flags (allows running modules based on detections)
- Most signatures based on metadata or flags



```
109
110 /*-----PE File Grouping-----*/
111 rule type_is_mz
112 {
113     meta:
114         scan_modules = "META_PE"
115         file_type = "pe"
116     condition:
117         uint16(0) == 0x5a4d
118         and not ext_sourceModule contains "META_PE"
119 }
120 /*-----*/
121
122 /*-----Archive Grouping-----*/
123 rule type_is_zip
124 {
125     meta:
126         scan_modules = "EXPLODE_ZIP(filelimit=1000)"
127         file_type = "zip"
128     condition:
129         uint32(0) == 0x04034b50 and not uint32(4) == 0x00060014
130 }
131
132 rule type_is_rar
133 {
134     meta:
135         scan_modules = "EXPLODE_RAR(filelimit=1000)"
136         file_type = "rar"
137     strings:
138         $a = { 52 61 72 21 1A 07 00 }
139     condition:
140         $a at 0
141 }
142
143 rule type_is_cab
```

Concise tags indicative of specific conditions

Should be related to potential maliciousness, often loosely

- But not necessarily high fidelity
- Sometimes closely tied to scanning errors, etc.

Examples

- yr:backdoor_abc
- pdf:url
- rar:BYTE_LIMITED_EXCEEDED
- pdf:PDFNotImplementedError
- x509:nfo:self_signed_cert

Middle ground between metadata and binary alert

Allows for abstraction of conditions and actions

- Module/Signature authors focus on identifying relevant conditions
 - Not required to know and maintain alert/block worthiness

Special roles in LB:

- Provided as an input for conditional dispatching
- Use by Dispositioner to determine disposition (action: alert or block)

Dispositioner

Module used to convert flags into end action (alert or block)

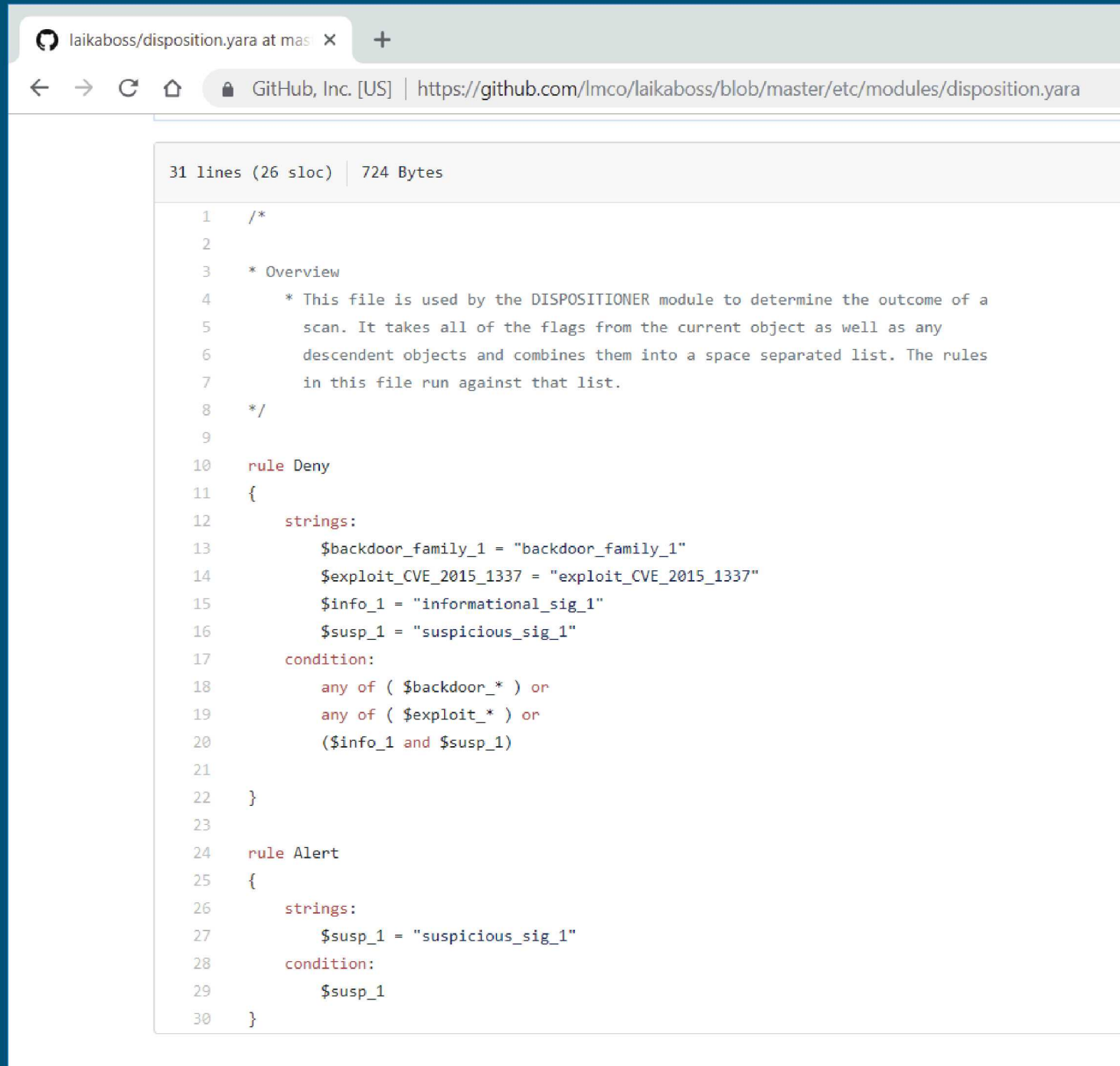
- Only flags, not metadata generally or file content

Configured using yara

- yara runs on space separated list of flag rollup (all flags from children objects)

Supports arbitrary outcomes

- Recommend one alert level per alerting class
- “Deny” used by clients (milter) to block content



The screenshot shows a web browser window with a single tab titled "laikaboss/disposition.yara at mas". The address bar displays "GitHub, Inc. [US] | https://github.com/lmco/laikaboss/blob/master/etc/modules/disposition.yara". The main content area shows a YARA rule file named "disposition.yara". At the top of the file view, it indicates "31 lines (26 sloc) | 724 Bytes". The code is a YARA rule with two main rules: "Deny" and "Alert". The "Deny" rule has a "strings" section with four variables: "\$backdoor_family_1", "\$exploit_CVE_2015_1337", "\$info_1", and "\$susp_1". Its "condition" is defined as "any of (\$backdoor_*) or any of (\$exploit_*) or (\$info_1 and \$susp_1)". The "Alert" rule has a "strings" section with one variable: "\$susp_1", and its "condition" is simply "\$susp_1".

```
1  /*
2
3  * Overview
4      * This file is used by the DISPOSITIONER module to determine the outcome of a
5      scan. It takes all of the flags from the current object as well as any
6      descendent objects and combines them into a space separated list. The rules
7      in this file run against that list.
8  */
9
10 rule Deny
11 {
12     strings:
13         $backdoor_family_1 = "backdoor_family_1"
14         $exploit_CVE_2015_1337 = "exploit_CVE_2015_1337"
15         $info_1 = "informational_sig_1"
16         $susp_1 = "suspicious_sig_1"
17     condition:
18         any of ( $backdoor_* ) or
19         any of ( $exploit_* ) or
20         ( $info_1 and $susp_1 )
21 }
22
23
24 rule Alert
25 {
26     strings:
27         $susp_1 = "suspicious_sig_1"
28     condition:
29         $susp_1
30 }
```




Questions?

wkleee@sandia.gov