

Python 标准库 mailcap 模块 CVE-2015-20107 漏洞分析报告（基于 YASA 扫描结果）

1. 扫描命令说明

本次采用 YASA 对 mailcap.py 的扫描在 Ubuntu 环境下进行，使用了 YASA 默认的 Python 污点规则包 taint-flow-python-default。执行扫描的实际命令如下：

```
./yasa-engine-linux-x64 \
--checkerPackIds "taint-flow-python-default" \
--analyzer "PythonAnalyzer" \
--ruleConfigFile "example-rule-config/rule_config_python.json" \
--sourcePath "/home/ubuntu/github/cpython-3.10.4/Lib/mailcap.py" \
--single \
--uastSDKPath "/home/ubuntu/yasa/uast4py-linux-amd64" \
--report "python3.10_results.json"
```

运行上述命令后，YASA 对 mailcap.py 进行了全量的数据流分析，识别出多处可能的不安全数据流（用户输入经由函数调用流向危险操作）。

2. 问题发现汇总

YASA 扫描结果如下：

```
===== Register rules =====
resolveCheckerPath projectRoot : /snapshot/YASA-Engine
Resolved checker path :/snapshot/YASA-
Engine/dist/checker/taint/python/python-default-taint-checker.js
rules-basic-handler [CONFIG] Loaded from: /snapshot/YASA-
Engine/dist/config.js
resolveCheckerPath projectRoot : /snapshot/YASA-Engine
Resolved checker path :/snapshot/YASA-
Engine/dist/checker/taint/python/django-taint-checker.js
rules-basic-handler [CONFIG] Loaded from: /snapshot/YASA-
Engine/dist/config.js
```

```
resolveCheckerPath projectRoot : /snapshot/YASA-Engine
Resolved checker path :/snapshot/YASA-
Engine/dist/checker/callgraph/callgraph-checker.js
rules-basic-handler [CONFIG] Loaded from: /snapshot/YASA-
Engine/dist/config.js
resolveCheckerPath projectRoot : /snapshot/YASA-Engine
Resolved checker path :/snapshot/YASA-
Engine/dist/checker/sanitizer/sanitizer-checker.js
rules-basic-handler [CONFIG] Loaded from: /snapshot/YASA-
Engine/dist/config.js
load checkers: [
    'taint_flow_python_input',
    'taint_flow_python_django_input',
    'callgraph',
    'sanitizer'
]
```

```
[YASA] Begin execution
[YASA] Executing preProcess
[YASA] Completed preProcess, cost: 4316ms
[YASA] Executing startAnalyze
[YASA] Executing makeFullCallGraph(BySymbolInterpret)
rules-basic-handler [CONFIG] Loaded from: /snapshot/YASA-
Engine/dist/config.js
makeAllCG-start
    makeAllCG-10%
    makeAllCG-30%
```

```
makeAllCG-70%
makeAllCG-100%
[YASA] Completed makeFullCallGraph(BySymbolInterpret), cost: 20ms
[YASA] Completed startAnalyze, cost: 30ms
[YASA] Executing symbolInterpret
EntryPoint [/mailcap. test] is executing
EntryPoint [/mailcap. readmailcapfile] is executing
EntryPoint [/home/ubuntu/github/cpython-3.10.4/Lib/mailcap. py] is
executing
[YASA] Completed symbolInterpret, cost: 668ms
[YASA] Execution completed, cost: 5017ms
```

```
===== Analysis Overview =====
```

Language	:	python
Files analyzed	:	1
Lines of code	:	277
Total time	:	5017ms
Total instruction	:	953
Executed instruction	:	953
Execution count	:	7313
Sources configured	:	129
Sinks configured	:	126
Valid entrypoints	:	3
Avg execution time per instruction	:	0.00ms
Avg instruction execution count	:	7.67
Execution time 70%/99%/100%	:	0.00ms/0.00ms/0.00ms
Execution times 70%/99%/100%	:	10.00/40.00/58.00

```
===== Performance Statistics =====
```

total cost: 5017ms

preProcess cost: 4316ms

startAnalyze cost: 30ms

makeFullCallGraph(BySymbolInterpret) cost: 20ms

symbolInterpret cost: 668ms

```
=====
```

Found 3 potential output strategy files

Registered strategy: callgraph from callgraph-output-strategy.js

Registered strategy: interactive from interactive-output-strategy.js

Registered strategy: taintflow from taint-output-strategy.js

Successfully registered 3 output strategies

```
===== outputFindings =====
```

```
===== Findings =====
```

```
----- 1 : taint_flow_python_input -----
```

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 173: os.system(test)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

```
{  
    filePath: '/mailcap.py',  
    functionName: 'test',  
    attribute: 'fullCallGraphMade',  
    type: 'functionCall',  
    packageName: undefined,  
    funcReceiverType: ''  
}
```

Trace:

/mailcap.py

AffectedNodeName: args

243: SOURCE: if len(args) < 2:

/mailcap.py

AffectedNodeName: file

247: Var Pass: file = args[1]

/mailcap.py

AffectedNodeName: findmatch

248: CALL: command, e = findmatch(caps, MIMETYPE, 'view', file)

/mailcap.py

AffectedNodeName: filename

159: ARG PASS: def findmatch(caps, MIMETYPE, key='view', filename="/dev/null", plist=[]):

/mailcap.py

AffectedNodeName: subst

```
172: CALL:           test = subst(e['test'], filename, plist)
/mailcap.py

AffectedNodeName: MIMEtype

192: ARG PASS: def subst(field, MIMEtype, filename, plist=[]):
/mailcap.py

AffectedNodeName: res

209: Var Pass:      res = res + MIMEtype
/mailcap.py

AffectedNodeName: res

216: Var Pass:      res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: [return value]

222: Return Value: return res
/mailcap.py

AffectedNodeName: subst

172: CALL RETURN:    test = subst(e['test'], filename, plist)
/mailcap.py

AffectedNodeName: test

172: Var Pass:      test = subst(e['test'], filename, plist)
/mailcap.py

AffectedNodeName: os.system

173: SINK:           if test and os.system(test) != 0:
```

----- 2 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint
File: /mailcap.py

```
Line 173: os.system(test)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

    filePath: '/mailcap.py',
    functionName: 'test',
    attribute: 'fullCallGraphMade',
    type: 'functionCall',
    packageName: undefined,
    funcReceiverType: ''

}

Trace:

/mailcap.py

AffectedNodeName: filename

207: SOURCE:             res = res + filename

/mailcap.py

AffectedNodeName: res

207: Var Pass:           res = res + filename

/mailcap.py

AffectedNodeName: res

209: Var Pass:           res = res + MIMEtype

/mailcap.py

AffectedNodeName: os.system

173: SINK:               if test and os.system(test) != 0:
```

----- 3 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 173: os.system(test)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

 filePath: '/mailcap.py',
 functionName: 'test',
 attribute: 'fullCallGraphMade',
 type: 'functionCall',
 packageName: undefined,
 funcReceiverType: ''

}

Trace:

/mailcap.py

AffectedNodeName: value

40: SOURCE: caps[key] = value

/mailcap.py

AffectedNodeName: [caps.key]

40: Var Pass: caps[key] = value

/mailcap.py

AffectedNodeName: [caps.key]

42: Var Pass: caps[key] = caps[key] + value

/mailcap.py

AffectedNodeName: [return value]

43: Return Value: return caps

/mailcap.py

AffectedNodeName: getcaps

237: CALL RETURN: caps = getcaps()

/mailcap.py

AffectedNodeName: caps

237: Var Pass: caps = getcaps()

/mailcap.py

AffectedNodeName: show

239: CALL: show(caps)

/mailcap.py

AffectedNodeName: caps

258: ARG PASS: def show(caps):

/mailcap.py

AffectedNodeName: findmatch

248: CALL: command, e = findmatch(caps, MIMEtype, 'view', file)

/mailcap.py

AffectedNodeName: caps

159: ARG PASS: def findmatch(caps, MIMEtype, key='view', filename="/dev/null", plist=[]):

/mailcap.py

AffectedNodeName: lookup

168: CALL: entries = lookup(caps, MIMEtype, key)

/mailcap.py

AffectedNodeName: caps

179: ARG PASS: def lookup(caps, MIMETYPE, key=None):
/mailcap.py

AffectedNodeName: entries

182: Var Pass: entries = entries + caps[MIMETYPE]
/mailcap.py

AffectedNodeName: entries

186: Var Pass: entries = entries + caps[MIMETYPE]
/mailcap.py

AffectedNodeName: entries

188: Var Pass: entries = [e for e in entries if key in e]
/mailcap.py

AffectedNodeName: entries

189: Var Pass: entries = sorted(entries, key=lineno_sort_key)
/mailcap.py

AffectedNodeName: [return value]

190: Return Value: return entries
/mailcap.py

AffectedNodeName: lookup

168: CALL RETURN: entries = lookup(caps, MIMETYPE, key)
/mailcap.py

AffectedNodeName: entries

168: Var Pass: entries = lookup(caps, MIMETYPE, key)
/mailcap.py

AffectedNodeName: subst

172: CALL: test = subst(e['test'], filename, plist)
/mailcap.py

AffectedNodeName: field
192: ARG PASS: def subst(field, MIMEtype, filename, plist=[]):
/mailcap.py

AffectedNodeName: n
undefined: Var Pass: undefined
/mailcap.py

AffectedNodeName: c
197: Var Pass: c = field[i]; i = i+1
/mailcap.py

AffectedNodeName: c
200: Var Pass: c = field[i:i+1]; i = i+1
/mailcap.py

AffectedNodeName: res
201: Var Pass: res = res + c
/mailcap.py

AffectedNodeName: c
203: Var Pass: c = field[i]; i = i+1
/mailcap.py

AffectedNodeName: res
205: Var Pass: res = res + c
/mailcap.py

AffectedNodeName: res
207: Var Pass: res = res + filename
/mailcap.py

AffectedNodeName: name
214: Var Pass: name = field[start:i]

```
/mailcap.py
AffectedNodeName: findparam
216: CALL:             res = res + findparam(name, plist)

/mailcap.py
AffectedNodeName: name
224: ARG PASS: def findparam(name, plist):
/mailcap.py
AffectedNodeName: name
225: Var Pass:     name = name.lower() + '='

/mailcap.py
AffectedNodeName: n
226: Var Pass:     n = len(name)

/mailcap.py
AffectedNodeName: res
221: Var Pass:             res = res + '%' + c

/mailcap.py
AffectedNodeName: os.system
173: SINK:             if test and os.system(test) != 0:
```

----- 4 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 173: os.system(test)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

```
{  
    filePath: '/mailcap.py',  
    functionName: 'test',  
    attribute: 'fullCallGraphMade',  
    type: 'functionCall',  
    packageName: undefined,  
    funcReceiverType: ''  
}
```

Trace:

/mailcap.py

AffectedNodeName: value

40: SOURCE: caps[key] = value

/mailcap.py

AffectedNodeName: [caps.key]

42: Var Pass: caps[key] = caps[key] + value

/mailcap.py

AffectedNodeName: [return value]

43: Return Value: return caps

/mailcap.py

AffectedNodeName: getcaps

237: CALL RETURN: caps = getcaps()

/mailcap.py

AffectedNodeName: caps

237: Var Pass: caps = getcaps()

/mailcap.py

AffectedNodeName: show

239: CALL: show(caps)
/mailcap.py
AffectedNodeName: caps

258: ARG PASS: def show(caps):
/mailcap.py
AffectedNodeName: findmatch

248: CALL: command, e = findmatch(caps, MIMETYPE, 'view', file)
/mailcap.py
AffectedNodeName: caps

159: ARG PASS: def findmatch(caps, MIMETYPE, key='view', filename="/dev/null", plist=[]):
/mailcap.py
AffectedNodeName: lookup

168: CALL: entries = lookup(caps, MIMETYPE, key)
/mailcap.py
AffectedNodeName: caps

179: ARG PASS: def lookup(caps, MIMETYPE, key=None):
/mailcap.py
AffectedNodeName: entries

182: Var Pass: entries = entries + caps[MIMETYPE]
/mailcap.py
AffectedNodeName: entries

186: Var Pass: entries = entries + caps[MIMETYPE]
/mailcap.py
AffectedNodeName: entries

188: Var Pass: entries = [e for e in entries if key in e]

```
/mailcap.py
AffectedNodeName: entries
189: Var Pass:      entries = sorted(entries, key=lineno_sort_key)

/mailcap.py
AffectedNodeName: [return value]
190: Return Value:   return entries

/mailcap.py
AffectedNodeName: lookup
168: CALL RETURN:    entries = lookup(caps, MIMEmode, key)

/mailcap.py
AffectedNodeName: entries
168: Var Pass:      entries = lookup(caps, MIMEmode, key)

/mailcap.py
AffectedNodeName: subst
172: CALL:           test = subst(e['test'], filename, plist)

/mailcap.py
AffectedNodeName: field
192: ARG PASS: def subst(field, MIMEmode, filename, plist=[]):

/mailcap.py
AffectedNodeName: n
undefined: Var Pass: undefined

/mailcap.py
AffectedNodeName: c
197: Var Pass:      c = field[i]; i = i+1

/mailcap.py
AffectedNodeName: c
```

```
200: Var Pass:           c = field[i:i+1]; i = i+1
/mailcap.py

AffectedNodeName: res

201: Var Pass:           res = res + c
/mailcap.py

AffectedNodeName: c

203: Var Pass:           c = field[i]; i = i+1
/mailcap.py

AffectedNodeName: res

205: Var Pass:           res = res + c
/mailcap.py

AffectedNodeName: res

207: Var Pass:           res = res + filename
/mailcap.py

AffectedNodeName: name

214: Var Pass:           name = field[start:i]
/mailcap.py

AffectedNodeName: findparam

216: CALL:                res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: name

224: ARG PASS: def findparam(name, plist):
/mailcap.py

AffectedNodeName: name

225: Var Pass:           name = name.lower() + '='
/mailcap.py
```

```
AffectedNodeName: n
226: Var Pass:      n = len(name)
/mailcap.py

AffectedNodeName: res
221: Var Pass:          res = res + '%' + c
/mailcap.py

AffectedNodeName: os.system
173: SINK:           if test and os.system(test) != 0:
```

----- 5 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 173: os.system(test)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

```
  filePath: '/mailcap.py',
  functionName: 'test',
  attribute: 'fullCallGraphMade',
  type: 'functionCall',
  packageName: undefined,
  funcReceiverType: ''
```

}

Trace:

/mailcap.py

AffectedNodeName: args
243: SOURCE: if len(args) < 2:
/mailcap.py

AffectedNodeName: file
247: Var Pass: file = args[1]
/mailcap.py

AffectedNodeName: findmatch
248: CALL: command, e = findmatch(caps, MIMETYPE, 'view', file)
/mailcap.py

AffectedNodeName: filename
159: ARG PASS: def findmatch(caps, MIMETYPE, key='view', filename="/dev/null", plist=[]):
/mailcap.py

AffectedNodeName: subst
172: CALL: test = subst(e['test'], filename, plist)
/mailcap.py

AffectedNodeName: MIMETYPE
192: ARG PASS: def subst(field, MIMETYPE, filename, plist[]):
/mailcap.py

AffectedNodeName: res
209: Var Pass: res = res + MIMETYPE
/mailcap.py

AffectedNodeName: res
216: Var Pass: res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: res

```
201: Var Pass:           res = res + c
/mailcap.py

AffectedNodeName: os.system

173: SINK:           if test and os.system(test) != 0:

----- 6 : taint_flow_python_input -----
Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint
File: /mailcap.py

Line 253: os.system(command)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:
{
    filePath: '/mailcap.py',
    functionName: 'test',
    attribute: 'fullCallGraphMade',
    type: 'functionCall',
    packageName: undefined,
    funcReceiverType: ''
}

Trace:
/mailcap.py

AffectedNodeName: filename

207: SOURCE:           res = res + filename
/mailcap.py

AffectedNodeName: subst
```

```
175: CALL:           command = subst(e[key], MIMEtype, filename, plist)
/mailcap.py

AffectedNodeName: plist

192: ARG PASS: def subst(field, MIMEtype, filename, plist=[]):
/mailcap.py

AffectedNodeName: findparam

216: CALL:           res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: plist

224: ARG PASS: def findparam(name, plist):
/mailcap.py

AffectedNodeName: [return value]

229: Return Value:      return p[n:]
/mailcap.py

AffectedNodeName: [return value]

230: Return Value:      return ''
/mailcap.py

AffectedNodeName: findparam

216: CALL RETURN:      res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: res

216: Var Pass:         res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: res

221: Var Pass:         res = res + '%' + c
/mailcap.py
```

AffectedNodeName: [return value]

222: Return Value: return res

/mailcap.py

AffectedNodeName: subst

175: CALL RETURN: command = subst(e[key], MIMEtype, filename, plist)

/mailcap.py

AffectedNodeName: command

175: Var Pass: command = subst(e[key], MIMEtype, filename, plist)

/mailcap.py

AffectedNodeName: [return value]

176: Return Value: return command, e

/mailcap.py

AffectedNodeName: [return value]

177: Return Value: return None, None

/mailcap.py

AffectedNodeName: findmatch

248: CALL RETURN: command, e = findmatch(caps, MIMEtype, 'view', file)

/mailcap.py

AffectedNodeName: [object Object], [object Object]

248: Var Pass: command, e = findmatch(caps, MIMEtype, 'view', file)

/mailcap.py

AffectedNodeName: os.system

253: SINK: sts = os.system(command)

----- 7 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 253: os.system(command)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

 filePath: '/mailcap.py',
 functionName: 'test',
 attribute: 'fullCallGraphMade',
 type: 'functionCall',
 packageName: undefined,
 funcReceiverType: ''

}

Trace:

/mailcap.py

AffectedNodeName: args

243: SOURCE: if len(args) < 2:

/mailcap.py

AffectedNodeName: MIMETYPE

246: Var Pass: MIMETYPE = args[0]

/mailcap.py

AffectedNodeName: findmatch

248: CALL: command, e = findmatch(caps, MIMETYPE, 'view', file)

/mailcap.py

AffectedNodeName: MIMEtype

159: ARG PASS: def findmatch(caps, MIMEtype, key='view',
filename="/dev/null", plist=[]):

/mailcap.py

AffectedNodeName: subst

175: CALL: command = subst(e[key], MIMEtype, filename, plist)

/mailcap.py

AffectedNodeName: MIMEtype

192: ARG PASS: def subst(field, MIMEtype, filename, plist[]):

/mailcap.py

AffectedNodeName: res

209: Var Pass: res = res + MIMEtype

/mailcap.py

AffectedNodeName: res

216: Var Pass: res = res + findparam(name, plist)

/mailcap.py

AffectedNodeName: [return value]

222: Return Value: return res

/mailcap.py

AffectedNodeName: subst

175: CALL RETURN: command = subst(e[key], MIMEtype, filename,
plist)

/mailcap.py

AffectedNodeName: command

175: Var Pass: command = subst(e[key], MIMEtype, filename,
plist)

/mailcap.py

```
AffectedNodeName: [return value]
176: Return Value:           return command, e
/mailcap.py

AffectedNodeName: [return value]
177: Return Value:           return None, None
/mailcap.py

AffectedNodeName: findmatch
248: CALL RETURN:           command, e = findmatch(caps, MIMEtype,
'view', file)
/mailcap.py

AffectedNodeName: [object Object], [object Object]
248: Var Pass:             command, e = findmatch(caps, MIMEtype, 'view',
file)
/mailcap.py

AffectedNodeName: os.system
253: SINK:                 sts = os.system(command)
```

----- 8 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 253: os.system(command)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

filePath: '/mailcap.py',

functionName: 'test',

```
attribute: 'fullCallGraphMade',
type: 'functionCall',
packageName: undefined,
funcReceiverType: ''
}
```

Trace:

/mailcap.py

AffectedNodeName: args

243: SOURCE: if len(args) < 2:

/mailcap.py

AffectedNodeName: MIMETYPE

246: Var Pass: MIMETYPE = args[0]

/mailcap.py

AffectedNodeName: findmatch

248: CALL: command, e = findmatch(caps, MIMETYPE, 'view', file)

/mailcap.py

AffectedNodeName: MIMETYPE

159: ARG PASS: def findmatch(caps, MIMETYPE, key='view', filename="/dev/null", plist=[]):

/mailcap.py

AffectedNodeName: lookup

168: CALL: entries = lookup(caps, MIMETYPE, key)

/mailcap.py

AffectedNodeName: MIMETYPE

179: ARG PASS: def lookup(caps, MIMETYPE, key=None):

/mailcap.py

AffectedNodeName: subst

175: CALL: command = subst(e[key], MIMETYPE, filename, plist)
/mailcap.py

AffectedNodeName: MIMETYPE

192: ARG PASS: def subst(field, MIMETYPE, filename, plist=[]):
/mailcap.py

AffectedNodeName: res

216: Var Pass: res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: [return value]

222: Return Value: return res
/mailcap.py

AffectedNodeName: subst

175: CALL RETURN: command = subst(e[key], MIMETYPE, filename, plist)
/mailcap.py

AffectedNodeName: command

175: Var Pass: command = subst(e[key], MIMETYPE, filename, plist)
/mailcap.py

AffectedNodeName: [return value]

176: Return Value: return command, e
/mailcap.py

AffectedNodeName: [return value]

177: Return Value: return None, None
/mailcap.py

AffectedNodeName: findmatch

```
248: CALL RETURN:           command, e = findmatch(caps, MIMEtype,
'view', file)

/mailcap.py

AffectedNodeName: [object Object], [object Object]

248: Var Pass:           command, e = findmatch(caps, MIMEtype, 'view',
file)

/mailcap.py

AffectedNodeName: os.system

253: SINK:           sts = os.system(command)
```

----- 9 : taint_flow_python_input -----

Description: Python 污点分析 checker，会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 253: os.system(command)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

```
  filePath: '/mailcap.py',
  functionName: 'test',
  attribute: 'fullCallGraphMade',
  type: 'functionCall',
  packageName: undefined,
  funcReceiverType: ''
```

}

Trace:

/mailcap.py

```
AffectedNodeName: args
243: SOURCE:           if len(args) < 2:
/mailcap.py

AffectedNodeName: file
247: Var Pass:         file = args[1]
/mailcap.py

AffectedNodeName: findmatch
248: CALL:             command, e = findmatch(caps, MIMETYPE, 'view',
file)
/mailcap.py

AffectedNodeName: filename
159: ARG PASS: def findmatch(caps, MIMETYPE, key='view',
filename="/dev/null", plist=[]):
/mailcap.py

AffectedNodeName: subst
175: CALL:             command = subst(e[key], MIMETYPE, filename, plist)
/mailcap.py

AffectedNodeName: filename
192: ARG PASS: def subst(field, MIMETYPE, filename, plist=[]):
/mailcap.py

AffectedNodeName: res
207: Var Pass:         res = res + filename
/mailcap.py

AffectedNodeName: res
209: Var Pass:         res = res + MIMETYPE
/mailcap.py

AffectedNodeName: os.system
```

```
253: SINK: sts = os.system(command)
```

```
----- 10 : taint_flow_python_input -----
```

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 253: os.system(command)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

```
    filePath: '/mailcap.py',
    functionName: 'test',
    attribute: 'fullCallGraphMade',
    type: 'functionCall',
    packageName: undefined,
    funcReceiverType: ''
```

}

Trace:

/mailcap.py

AffectedNodeName: value

```
40: SOURCE: caps[key] = value
```

/mailcap.py

AffectedNodeName: [caps.key]

```
40: Var Pass: caps[key] = value
```

/mailcap.py

AffectedNodeName: [caps.key]

```
42: Var Pass:           caps[key] = caps[key] + value
/mailcap.py

AffectedNodeName: [return value]

43: Return Value:      return caps
/mailcap.py

AffectedNodeName: getcaps

237: CALL RETURN:     caps = getcaps()

/mailcap.py

AffectedNodeName: caps

237: Var Pass:        caps = getcaps()

/mailcap.py

AffectedNodeName: show

239: CALL:            show(caps)
/mailcap.py

AffectedNodeName: caps

258: ARG PASS:        def show(caps):
/mailcap.py

AffectedNodeName: findmatch

248: CALL:            command, e = findmatch(caps, MIMETYPE, 'view',
file)

/mailcap.py

AffectedNodeName: caps

159: ARG PASS:        def findmatch(caps, MIMETYPE, key='view',
filename="/dev/null", plist=[]):
/mailcap.py

AffectedNodeName: lookup

168: CALL:            entries = lookup(caps, MIMETYPE, key)
```

/mailcap.py

AffectedNodeName: caps

179: ARG PASS: def lookup(caps, MIMETYPE, key=None):

/mailcap.py

AffectedNodeName: entries

182: Var Pass: entries = entries + caps[MIMETYPE]

/mailcap.py

AffectedNodeName: entries

186: Var Pass: entries = entries + caps[MIMETYPE]

/mailcap.py

AffectedNodeName: entries

188: Var Pass: entries = [e for e in entries if key in e]

/mailcap.py

AffectedNodeName: entries

189: Var Pass: entries = sorted(entries, key=lineno_sort_key)

/mailcap.py

AffectedNodeName: [return value]

190: Return Value: return entries

/mailcap.py

AffectedNodeName: lookup

168: CALL RETURN: entries = lookup(caps, MIMETYPE, key)

/mailcap.py

AffectedNodeName: entries

168: Var Pass: entries = lookup(caps, MIMETYPE, key)

/mailcap.py

AffectedNodeName: subst

172: CALL: test = subst(e['test'], filename, plist)
/mailcap.py
AffectedNodeName: field

192: ARG PASS: def subst(field, MIMEtype, filename, plist=[]):
/mailcap.py
AffectedNodeName: n
undefined: Var Pass: undefined
/mailcap.py
AffectedNodeName: c
197: Var Pass: c = field[i]; i = i+1
/mailcap.py
AffectedNodeName: c
200: Var Pass: c = field[i:i+1]; i = i+1
/mailcap.py
AffectedNodeName: res
201: Var Pass: res = res + c
/mailcap.py
AffectedNodeName: c
203: Var Pass: c = field[i]; i = i+1
/mailcap.py
AffectedNodeName: res
205: Var Pass: res = res + c
/mailcap.py
AffectedNodeName: res
207: Var Pass: res = res + filename
/mailcap.py

AffectedNodeName: name
214: Var Pass: name = field[start:i]
/mailcap.py

AffectedNodeName: findparam
216: CALL: res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: name
224: ARG PASS: def findparam(name, plist):
/mailcap.py

AffectedNodeName: name
225: Var Pass: name = name.lower() + '='
/mailcap.py

AffectedNodeName: n
226: Var Pass: n = len(name)
/mailcap.py

AffectedNodeName: res
221: Var Pass: res = res + '%' + c
/mailcap.py

AffectedNodeName: subst
175: CALL: command = subst(e[key], MIMEtype, filename, plist)
/mailcap.py

AffectedNodeName: os.system
253: SINK: sts = os.system(command)

----- 11 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 253: os.system(command)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

 filePath: '/mailcap.py',
 functionName: 'test',
 attribute: 'fullCallGraphMade',
 type: 'functionCall',
 packageName: undefined,
 funcReceiverType: ''

}

Trace:

/mailcap.py

AffectedNodeName: value

40: SOURCE: caps[key] = value

/mailcap.py

AffectedNodeName: [caps.key]

42: Var Pass: caps[key] = caps[key] + value

/mailcap.py

AffectedNodeName: [return value]

43: Return Value: return caps

/mailcap.py

AffectedNodeName: getcaps

237: CALL RETURN: caps = getcaps()

/mailcap.py

AffectedNodeName: caps

237: Var Pass: caps = getcaps()

/mailcap.py

AffectedNodeName: show

239: CALL: show(caps)

/mailcap.py

AffectedNodeName: caps

258: ARG PASS: def show(caps):

/mailcap.py

AffectedNodeName: findmatch

248: CALL: command, e = findmatch(caps, MIMETYPE, 'view', file)

/mailcap.py

AffectedNodeName: caps

159: ARG PASS: def findmatch(caps, MIMETYPE, key='view', filename="/dev/null", plist=[]):

/mailcap.py

AffectedNodeName: lookup

168: CALL: entries = lookup(caps, MIMETYPE, key)

/mailcap.py

AffectedNodeName: caps

179: ARG PASS: def lookup(caps, MIMETYPE, key=None):

/mailcap.py

AffectedNodeName: entries

182: Var Pass: entries = entries + caps[MIMETYPE]

/mailcap.py

AffectedNodeName: entries
186: Var Pass: entries = entries + caps[MIMETYPE]

/mailcap.py

AffectedNodeName: entries
188: Var Pass: entries = [e for e in entries if key in e]

/mailcap.py

AffectedNodeName: entries
189: Var Pass: entries = sorted(entries, key=lineno_sort_key)

/mailcap.py

AffectedNodeName: [return value]

190: Return Value: return entries

/mailcap.py

AffectedNodeName: lookup

168: CALL RETURN: entries = lookup(caps, MIMETYPE, key)

/mailcap.py

AffectedNodeName: entries

168: Var Pass: entries = lookup(caps, MIMETYPE, key)

/mailcap.py

AffectedNodeName: subst

172: CALL: test = subst(e['test'], filename, plist)

/mailcap.py

AffectedNodeName: field

192: ARG PASS: def subst(field, MIMETYPE, filename, plist=[]):

/mailcap.py

AffectedNodeName: n

undefined: Var Pass: undefined

```
/mailcap.py  
AffectedNodeName: c  
197: Var Pass:           c = field[i]; i = i+1  
  
/mailcap.py  
AffectedNodeName: c  
200: Var Pass:           c = field[i:i+1]; i = i+1  
  
/mailcap.py  
AffectedNodeName: res  
201: Var Pass:           res = res + c  
  
/mailcap.py  
AffectedNodeName: c  
203: Var Pass:           c = field[i]; i = i+1  
  
/mailcap.py  
AffectedNodeName: res  
205: Var Pass:           res = res + c  
  
/mailcap.py  
AffectedNodeName: res  
207: Var Pass:           res = res + filename  
  
/mailcap.py  
AffectedNodeName: name  
214: Var Pass:           name = field[start:i]  
  
/mailcap.py  
AffectedNodeName: findparam  
216: CALL:                res = res + findparam(name, plist)  
  
/mailcap.py  
AffectedNodeName: name
```

```
224: ARG PASS: def findparam(name, plist):  
/mailcap.py  
AffectedNodeName: name  
  
225: Var Pass:      name = name.lower() + '='  
/mailcap.py  
AffectedNodeName: n  
  
226: Var Pass:      n = len(name)  
/mailcap.py  
AffectedNodeName: res  
  
221: Var Pass:          res = res + '%' + c  
/mailcap.py  
AffectedNodeName: subst  
  
175: CALL:           command = subst(e[key], MIMEtype, filename, plist)  
/mailcap.py  
AffectedNodeName: os.system  
  
253: SINK:           sts = os.system(command)  
  
----- 12 : taint_flow_python_input -----  
Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint  
File: /mailcap.py  
Line 253: os.system(command)  
SINK RULE: os.system  
SINK Attribute: PythonCommandExec  
entrypoint:  
{  
  filePath: '/mailcap.py',
```

```
    functionName: 'test',
    attribute: 'fullCallGraphMade',
    type: 'functionCall',
    packageName: undefined,
    funcReceiverType: ''
}
```

Trace:

/mailcap.py

AffectedNodeName: filename

207: SOURCE: res = res + filename

/mailcap.py

AffectedNodeName: subst

175: CALL: command = subst(e[key], MIMEtype, filename, plist)

/mailcap.py

AffectedNodeName: plist

192: ARG PASS: def subst(field, MIMEtype, filename, plist=[]):

/mailcap.py

AffectedNodeName: findparam

216: CALL: res = res + findparam(name, plist)

/mailcap.py

AffectedNodeName: plist

224: ARG PASS: def findparam(name, plist):

/mailcap.py

AffectedNodeName: [return value]

229: Return Value: return p[n:]

/mailcap.py

```
AffectedNodeName: [return value]
230: Return Value:      return ''
/mailcap.py

AffectedNodeName: findparam
216: CALL RETURN:          res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: res
216: Var Pass:           res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: res
221: Var Pass:           res = res + '%' + c
/mailcap.py

AffectedNodeName: res
201: Var Pass:           res = res + c
/mailcap.py

AffectedNodeName: res
205: Var Pass:           res = res + c
/mailcap.py

AffectedNodeName: os.system
253: SINK:                sts = os.system(command)
```

----- 13 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 253: os.system(command)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

```
{  
    filePath: '/mailcap.py',  
    functionName: 'test',  
    attribute: 'fullCallGraphMade',  
    type: 'functionCall',  
    packageName: undefined,  
    funcReceiverType: ''  
}
```

Trace:

/mailcap.py

AffectedNodeName: args

243: SOURCE: if len(args) < 2:

/mailcap.py

AffectedNodeName: MIMEtype

246: Var Pass: MIMEtype = args[0]

/mailcap.py

AffectedNodeName: findmatch

248: CALL: command, e = findmatch(caps, MIMEtype, 'view', file)

/mailcap.py

AffectedNodeName: MIMEtype

159: ARG PASS: def findmatch(caps, MIMEtype, key='view', filename="/dev/null", plist=[]):

/mailcap.py

AffectedNodeName: subst

```
175: CALL:           command = subst(e[key], MIMEtype, filename, plist)
/mailcap.py

AffectedNodeName: MIMEtype

192: ARG PASS: def subst(field, MIMEtype, filename, plist=[]):
/mailcap.py

AffectedNodeName: res

209: Var Pass:           res = res + MIMEtype
/mailcap.py

AffectedNodeName: res

216: Var Pass:           res = res + findparam(name, plist)
/mailcap.py

AffectedNodeName: res

201: Var Pass:           res = res + c
/mailcap.py

AffectedNodeName: os.system

253: SINK:           sts = os.system(command)
```

----- 14 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 253: os.system(command)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

filePath: '/mailcap.py',

```
    functionName: 'test',
    attribute: 'fullCallGraphMade',
    type: 'functionCall',
    packageName: undefined,
    funcReceiverType: ''
}
```

Trace:

/mailcap.py

AffectedNodeName: args

243: SOURCE: if len(args) < 2:

/mailcap.py

AffectedNodeName: MIMEtype

246: Var Pass: MIMEtype = args[0]

/mailcap.py

AffectedNodeName: findmatch

248: CALL: command, e = findmatch(caps, MIMEtype, 'view', file)

/mailcap.py

AffectedNodeName: MIMEtype

159: ARG PASS: def findmatch(caps, MIMEtype, key='view', filename="/dev/null", plist=[]):

/mailcap.py

AffectedNodeName: lookup

168: CALL: entries = lookup(caps, MIMEtype, key)

/mailcap.py

AffectedNodeName: MIMEtype

179: ARG PASS: def lookup(caps, MIMEtype, key=None):

```
/mailcap.py
AffectedNodeName: subst
175: CALL:           command = subst(e[key], MIMEtype, filename, plist)
/mailcap.py
AffectedNodeName: MIMEtype
192: ARG PASS: def subst(field, MIMEtype, filename, plist=[]):
/mailcap.py
AffectedNodeName: res
216: Var Pass:           res = res + findparam(name, plist)
/mailcap.py
AffectedNodeName: res
201: Var Pass:           res = res + c
/mailcap.py
AffectedNodeName: os.system
253: SINK:           sts = os.system(command)
```

----- 15 : taint_flow_python_input -----

Description: Python 污点分析 checker, 会使用 CallGraph 边界制作 entrypoint

File: /mailcap.py

Line 253: os.system(command)

SINK RULE: os.system

SINK Attribute: PythonCommandExec

entrypoint:

{

filePath: '/mailcap.py',

functionName: 'test',

```
attribute: 'fullCallGraphMade',
type: 'functionCall',
packageName: undefined,
funcReceiverType: ''
}
```

Trace:

/mailcap.py

AffectedNodeName: filename

207: SOURCE: res = res + filename

/mailcap.py

AffectedNodeName: subst

175: CALL: command = subst(e[key], MIMEtype, filename, plist)

/mailcap.py

AffectedNodeName: plist

192: ARG PASS: def subst(field, MIMEtype, filename, plist=[]):

/mailcap.py

AffectedNodeName: findparam

216: CALL: res = res + findparam(name, plist)

/mailcap.py

AffectedNodeName: plist

224: ARG PASS: def findparam(name, plist):

/mailcap.py

AffectedNodeName: [return value]

229: Return Value: return p[n:]

/mailcap.py

AffectedNodeName: os.system

```
253: SINK:           sts = os.system(command)
```

```
=====
```

```
# Total-findings : 15
```

```
=====
```

```
Error occurred in file-util.writeJSONfile
```

```
report is write to  
/home/ubuntu/yasa/python3.10_results.json/report.sarif
```

```
=====
```

mailcap.py 存在多条值得关注的污点传播路径，均与未过滤的外部输入流入 os.system 调用 有关。下表汇总了扫描发现的每一条潜在漏洞信息，包括编号、涉及文件和函数、代码行号、污点源（不可信输入来源）、污点汇（危险函数调用），以及对应的风险描述（漏洞类型及影响）：

编号	文件名	函数名	行号	污点源	污点汇	风险描述
1	mailcap.py	findmatch	173	命令行参数 (文件路径)	os.system 调用	命令注入漏洞：用户文件名经替换后直接执行系统命令，可能执行任意命令
2	mailcap.py	findmatch	173	用户提供的文件名变量	os.system 调用	命令注入漏洞：未转义的文件路径插入命令字符串，导致任意命令执行
3	mailcap.py	findmatch	173	外部 mailcap 文件内容 (value)	os.system 调用	命令注入漏洞：恶意 mailcap 配置可插入命令，执行任意系统命令
4	mailcap.py	findmatch	173	外部 mailcap 文件内容 (追加)	os.system 调用	命令注入漏洞：合并自外部配置的命令未经校验执行，风险同上
5	mailcap.py	findmatch	173	命令行参数	os.system	命令注入漏洞：与漏洞

编号	文件名	函数名	行号	污点源	污点汇	风险描述
				(文件路径)	调用	1 相似，文件名注入命令执行
6	mailcap.py	test	252	用户提供的文件名变量	os.system 调用	命令注入漏洞：未经处理的文件名插入视图命令，导致任意命令执行
7	mailcap.py	test	252	命令行参数 (MIME 类型)	os.system 调用	命令注入漏洞： MIME 类型值可被恶意构造插入命令，造成命令执行
8	mailcap.py	test	252	命令行参数 (MIME 类型)	os.system 调用	命令注入漏洞：与漏洞 7 相似，MIME 类型未经校验导致命令注入
9	mailcap.py	test	252	命令行参数 (文件路径)	os.system 调用	命令注入漏洞：与漏洞 6 相似，用户文件路径可构造恶意命令执行
10	mailcap.py	test	252	外部 mailcap 文件内容	os.system 调用	命令注入漏洞：恶意 mailcap 配置与文件输入结合可导致命令执行
11	mailcap.py	test	252	外部 mailcap 文件内容	os.system 调用	命令注入漏洞：同上，未校验的外部配置致命令执行
12	mailcap.py	test	252	用户提供的文件名变量	os.system 调用	命令注入漏洞：同漏洞 6/9，文件名可注入命令
13	mailcap.py	test	252	(推测) 外部 mailcap 文件内容	os.system 调用	命令注入漏洞：同漏洞 10/11，mailcap 内容可致任意命令执行
14	mailcap.py	test	252	(推测) 外部 mailcap 文件内容	os.system 调用	命令注入漏洞：同上，外部配置命令注入
15	mailcap.py	test	252	(推测) 外部 mailcap 文件内容	os.system 调用	命令注入漏洞：同上，外部配置命令注入

注：上述多个条目集中在两处危险函数调用：一是在 findmatch 函数中执行 mailcap 条目的 test 测试命令（行 173），二是在 test() 主函数中执行具体查

看命令（行 252）。污点源主要有两类：命令行参数（用户提供的 MIME 类型和文件路径），以及外部 mailcap 配置文件内容（攻击者可控的配置条目）。这些未经安全处理的输入直接或间接流入系统命令执行，可能被利用进行恶意操作。

3. 漏洞路径追踪与分析

下面针对每条扫描发现的漏洞，详细追踪污点从源头到达危险操作的路径，结合代码片段分析漏洞成因，并评估安全影响。所有编号对应上表中的漏洞编号。可以看出，这些漏洞彼此关联，均源于 mailcap 模块未对拼接进入系统命令的外部输入进行转义或校验 导致的命令注入问题。

漏洞 1：命令行文件参数导致的 os.system(test) 命令注入

路径追踪： 在 mailcap.test() 函数中，程序从命令行读取 MIME 类型和文件路径参数：MIMETYPE = args[0]; file = args[1]（行 245-246）。该 file 变量直接源自用户的命令行输入（污点源）。随后调用 findmatch(caps, MIMETYPE, 'view', file) 寻找匹配的 mailcap 条目（行 247）。进入 findmatch 函数后，程序遍历候选的 mailcap 配置条目，对于每个条目若存在 test 字段则执行测试命令：

```
if 'test' in e:  
    test = subst(e['test'], filename, plist)  
    if test and os.system(test) != 0:  
        continue
```

上述代码（行 170-173）中，filename 实参正是未经过滤的 file 用户输入。函数通过 subst() 将 mailcap test 命令字符串中的占位符替换成实际文件名等参数，结果存入 test 变量。例如，若 mailcap test 字段为类似 test="[条件] %s [参数]"，则 %s 将被替换为用户提供的文件路径。关键问题在于：subst() 并未对插入的文件名做任何转义处理。如果攻击者在文件名中构造诸如 \$(恶意命令)、; 恶意命令 等 shell 元字符，subst 会原样拼接进

入命令字符串。随后，`os.system(test)` 在 shell 中执行了构造后的 test 命令（行 172）。如果攻击者的文件名包含恶意片段，那么在执行 `os.system(test)` 时，这些片段会作为 shell 命令运行。

安全影响： 攻击者可以利用这一流程，在 `mailcap.findmatch` 的测试命令阶段注入任意系统命令执行。例如，某应用调用 `mailcap.findmatch` 来打开用户上传的文件，攻击者将文件命名为：“`sample.txt; rm -rf /important/data`”，则在没有转义机制的情况下，`subst` 会生成 `test="... sample.txt; rm -rf /important/data"` 的命令串，`os.system` 执行时就会删除系统关键数据。此漏洞本质是典型的 Shell 命令注入，可能导致目标程序以其权限执行任意命令，完全控制系统或破坏数据。这一问题即对应 CVE-2015-20107 所揭示的 `mailcap` 模块漏洞机制。值得注意的是，哪怕 `test` 命令最终返回非零退出码（从而 `findmatch` 跳过该条目），恶意命令依然已经执行，对系统造成危害。

漏洞 2：`subst()` 函数未转义文件名导致命令注入

路径追踪： 此漏洞与漏洞 1 属同一处代码，但从污点分析角度突出 `subst()` 函数内部对文件名的直接拼接：在 `subst(field, MIMEtype, filename, plist)` 函数中，当解析到占位符`%s` 时执行 `res = res + filename`。这里的 `filename` 变量承接自上层的用户输入（同漏洞 1 的 `file`）。由于没有任何过滤，`filename` 中的特殊字符会直接附加到命令片段中。例如，若文件名为 `hello.txt; uname -a`，则 `res += "hello.txt; uname -a"`。最终 `subst` 返回拼接后的命令字符串给调用者。漏洞 2 的污点源即为该未经过滤直接拼接的 `filename`，污点汇依然是后续的 `os.system(test)` 执行点。

安全影响： 这一细节进一步证明了 `mailcap` 模块缺乏对用户输入的转义。攻击者完全可以通过精心修改文件名来闭合原本良性的命令并连接恶意命令。例如，在 `mailcap` 的某配置中，`test` 命令可能为 `echo %s | grep ...` 之类，本意是检查文件内容。但如果文件名中包含管道符号 `|`，则可使命令注入新的管道命令。由于漏洞 1 和漏洞 2 描述的是同一流程中的不同环节，安全影响相同：都是允许任意

命令执行。开发者应意识到，在将外部输入嵌入 shell 命令时，如果不进行适当的转义或安全封装，极易导致此类命令注入漏洞。

漏洞 3：恶意 mailcap 配置内容导致的 os.system(test) 命令注入

路径追踪： 污点源转向另一来源：系统的 mailcap 配置文件内容。
Mailcap 数据库由多个文件汇总而成（如用户主目录下的 .mailcap，系统 /etc/mailcap 等）。在 getcaps() 函数中，Python 打开并读取这些文件，通过 _readmailcapfile 解析出键值对加入 caps 字典。代码片段：

```
morecaps, lineno = _readmailcapfile(fp, lineno)
for key, value in morecaps.items():
    if not key in caps:
        caps[key] = value
    else:
        caps[key] = caps[key] + value
```

上述 caps 字典的构建过程（行 37-41）直接反映了外部文件内容被读入内存。这里的 value 就是从 mailcap 文件解析出的命令等字段组成的字典。YASA 将 caps[key] = value 和 caps[key] = caps[key] + value 识别为污点源（分别对应新键和已有键追加两种情况）。这些数据随后在 findmatch 流程中被使用：当用户请求打开某 MIME 类型时，findmatch 从 caps 中取出对应的配置条目列表（通过 lookup() 找到 entries）。此后流程与漏洞 1 类似：若条目包含 test 字段则通过 subst 替换生成命令并调用 os.system 执行。区别在于，此处的危险来自 mailcap 文件本身被恶意篡改。如果攻击者能够控制 mailcap 配置（例如在用户主目录放置恶意的 .mailcap，或篡改系统 mailcap），则他可以植入恶意命令。例如，在 mailcap 文件中为常见的 MIME 类型添加条目：text/plain; /bin/evil_script %s; needsterminal（伪造一个需要终端的查看器命令）。当应用调用 findmatch 打开文本文件时，上述恶意命令会被选中，并在测试阶段或查看

阶段执行。

安全影响： 在这种情况下，即便用户提供的文件名本身是无害的正常文件，恶意的 mailcap 配置仍可导致命令执行。攻击者可以利用社工手段诱使系统管理员安装一个恶意 mailcap，或者在多用户系统中低权用户将自己的 .mailcap 配置给高权应用使用，从而实现提权攻击。由于 mailcap.py 没有对配置内容做安全过滤，信任了文件中的命令，这就像应用在加载不受信任的脚本并执行一样危险。总体影响仍是任意系统命令执行，尤其严重的是它可能绕过应用层的输入验证（因为此时恶意 payload 来自配置文件而非直接的用户输入）。

漏洞 4：合并 mailcap 配置条目时的命令注入

路径追踪： 这一漏洞与漏洞 3 密切相关，是 mailcap 多文件合并时的另一分支。前述构建 caps 字典时，若某 MIME 类型在多个文件中出现，则第二次会走 else: caps[key] = caps[key] + value 分支，将新的配置追加到已有列表。YASA 将此视作另一污点源（对应扫描结果编号 4）。追加后的 caps 内容同样未经清理，随即被 findmatch 使用。污点流经 lookup() 获取条目列表、subst() 字符串替换，最终在 os.system 执行（流程与漏洞 3 后半段一致）。因此漏洞 4 本质上也是“不可信 mailcap 内容导致命令执行”，只是在代码路径上体现为追加操作。

安全影响： 当存在多个 mailcap 文件来源时（用户级和系统级），攻击者可以通过在优先级较高的位置插入恶意条目或利用追加顺序来确保恶意命令被执行。由于 findmatch 会按顺序检查配置列表，第一个通过测试的条目将被采用执行。攻击者可以设计一个总是返回真（exit code 0）的 test 命令，例如 test=true;，以确保恶意条目不被跳过，然后其 view 命令部分执行真正的恶意操作。总之，漏洞 3 和漏洞 4 强调：配置文件应被视为外部输入来源，若应用直接执行其中命令而不校验，后果同样严重。

漏洞 5：命令行文件参数导致的命令注入（与漏洞 1 重复）

路径追踪： 漏洞 5 与漏洞 1 在本质和路径上并无差异，均是用户通过命令行提供的文件名污点经由 findmatch -> subst 流向 os.system(test) 的情况。YASA 输出中重复出现了类似路径（可能因为不同的分析分支或调用上下文），此

处不再赘述。参考漏洞 1 的分析可知，问题依旧在于未对文件路径进行转义，导致 shell 命令注入。

安全影响： 同漏洞 1， 攻击者控制文件名即可执行任意系统命令，危害严重。

漏洞 6：文件名经 `os.system(command)` 执行的命令注入

路径追踪： 从这一条开始，污点汇切换到 `mailcap.test()` 函数中执行查看命令的步骤。继续沿用之前的用户输入：`file = args[1]`（命令行提供）作为污点源。在 `findmatch` 中，通过 `subst(e[key], MIMETYPE, filename, plist)` 生成实际查看命令字符串并返回。然后回到 `test()` 函数，取得返回的 `command` 字符串（行 247-248），并调用 `os.system(command)` 执行之（行 252）。以代码示意：

```
command, e = findmatch(caps, MIMETYPE, 'view', file)
...
sts = os.system(command)
```

这里的 `command` 变量同样包含了未经处理的用户文件名。在 `subst()` 内部，当解析 `mailcap` 的 `view` 字段时，遇到 `%s` 会拼接文件名（与漏洞 2 中的机制一致），遇到 `%t` 则拼接 MIME 类型等。由此产生的命令字符串直接用于系统调用。举例来说，如果 `mailcap` 配置的查看器命令是“`xdg-open %s`”，而文件名为“`$(xcalc)`”（试图打开计算器程序），那么 `subst` 会生成命令 `xdg-open $(xcalc)`，`os.system` 执行时将触发计算器程序启动。

安全影响： 漏洞 6 体现了 CVE-2015-20107 漏洞的主要危害： 用户可控的文件路径直接插入并执行查看命令，造成任意命令执行或利用文件路径进行路径遍历。例如，攻击者可以将文件名设置为包含目录上跳符`..`/，配合一个非预期的命令查看器，使程序打开或操作本不应访问的系统文件（达到路径遍历的效果）。更危险的是加入诸如`;`、`&` 等 shell 控制符串联破坏性命令，导致命令注入。因此，

未经过滤的 `os.system(command)` 调用为攻击者开了后门，轻则破坏应用逻辑，重则危及整台服务器的安全。

漏洞 7：MIME 类型参数导致的命令注入

路径追踪：除了文件名，本次扫描还发现 MIME 类型参数（命令行的第一个参数）也可成为污点源。在 `test()` 中，`MIMETYPE = args[0]`（行 245）取得用户提供的 MIME 类型字符串。之后流程进入 `findmatch`，在构造查看命令时如果 `mailcap` 配置字符串包含%t 占位符，则会通过 `subst()` 将其替换为 MIME 类型。例如，一个 `mailcap` 条目可能形如：`application/%t; someviewer --type=%t %s`。若攻击者将 MIME 类型参数设置为特殊值（例如 `image/png`；后接命令），则同样可能注入到命令中。YASA 的污点跟踪表明，`MIMETYPE` 经由多层函数传递，最终影响 `res = res + MIMETYPE`（占位%t 替换）这一步骤，并流入 `os.system(command)` 执行。

安全影响：正常情况下，MIME 类型字符串应是有限集合（如 `text/plain`、`image/png` 等），一般由程序根据文件内容或扩展名确定，用户直接控制 MIME 类型的场景不多见。然而，如果存在应用允许用户指定 MIME 类型（或攻击者通过某种方式伪造输入值），就可能被利用。恶意 MIME 类型值插入命令所造成的影响与文件名类似，也是任意命令执行。虽然相对而言 MIME 类型作为攻击载体的机会更少，但这一发现提醒我们：凡是通过字符串替换注入命令的机制，不论占位符代表文件名还是其他参数，都必须严格校验或转义，不能掉以轻心。

漏洞 8：MIME 类型参数命令注入（与漏洞 7 重复）

路径追踪：漏洞 8 与漏洞 7 实质相同，均是强调 MIME 类型污点对系统命令的影响。YASA 可能针对不同调用顺序或分支重复报告了 MIME 类型的传播路径。其源头依然是 `args[0]` 用户输入，经由 `findmatch -> subst` 插入命令，最终在 `os.system(command)` 执行。不再重复相同细节。

安全影响：同漏洞 7，表明不可信 MIME 类型也可引发命令注入，应对所有外部输入做好校验。

漏洞 9：文件名和 MIME 类型共同影响命令执行

路径追踪：这一漏洞条目显示了污点分析中多个变量共同构成危险点的情形。在一次命令构造中，如果 mailcap 的命令字符串同时包含 %s 和 %t，则用户提供的文件名和 MIME 类型都会被插入进去。漏洞 9 的跟踪信息显示：污点源既包括 file = args[1]（文件路径），也包括前面的 MIMEtype = args[0]，两者经替换先后附加到命令字符串中。例如某 mailcap 配置：%t_viewer --open %s，当 MIME 类型和文件名均可控时，攻击者可在两处注入恶意片段（比如 MIME 类型尾部加；另一命令，文件名中加;更多命令），组合造成更复杂的攻击载荷。

安全影响：多污点融合的情况增加了利用灵活性。攻击者可以将一部分恶意代码放在 MIME 类型中，另一部分放在文件名中，当两者拼接后形成完整的恶意命令。尽管攻击效果仍属命令注入范畴，但这种多点注入可能绕过单一字段的简单过滤（如果应用对文件名做了部分过滤但未考虑 MIME 类型，攻击者可以在 MIME 类型中附加 payload）。

漏洞 10：mailcap 配置内容与输入结合导致命令注入

路径追踪：漏洞 10 的场景综合了前面的因素：外部 mailcap 文件提供了恶意命令模板，用户输入（文件名或 MIME 类型）填充了其中的占位符，最终在系统调用处产生危险行为。YASA 跟踪显示污点源为 caps[key] = value (mailcap 文件解析结果) 以及后续的用户参数，一路流经 findmatch → subst → os.system(command)。换言之，攻击的两种手段（恶意配置和恶意输入）可以叠加：mailcap 配置定义了一条潜在有害的命令格式，而攻击者提供特殊的文件名/MIME 使之真正成为有害命令。举个例子：攻击者无法直接修改服务器的 mailcap 配置，但可以上传一个特别命名的文件。“正常” mailcap 可能有条目 image/*; display %s 用于显示图片。如果攻击者将上传的图片文件命名为 “/tmp/evil.png; shutdown -h now”，那么即使 mailcap 本身无恶意，组合起来依然会执行 display /tmp/evil.png; shutdown -h now，导致服务器关机。

安全影响：由上可见，信任边界上的所有元素（包括配置文件和输入参数）都可能成为攻击点。漏洞 10 提醒我们，哪怕 mailcap 文件是可信的，仍需防御输

入的恶意；反之亦然。如果 mailcap 配置不可控，也不能忽视输入路径的安全。因此完善的解决方案应该同时考虑配置安全和输入校验两个方面，缺一不可。

漏洞 11：重复的 mailcap 内容命令注入路径

路径追踪：漏洞 11 与漏洞 10 类似，集中在 mailcap 配置内容作为污点源的又一条路径，可能对应于不同的 mailcap 文件（用户 vs 系统）或列表追加次序。它表明即使 mailcap 数据在不同文件中分段读取并组合，只要最终进入命令执行流程，危险性等同。

安全影响：同漏洞 10，总是确保外部提供的命令模板和参数不会直接用于系统调用。对策可以是对 mailcap 配置中的命令进行安全过滤（例如限制只能是安全命令或不含特殊字符），或者执行时不经过 shell 直接调用程序等。

漏洞 12：文件名参数命令注入（重复）

路径追踪：漏洞 12 再一次强调文件名污点对 os.system(command) 的影响，与漏洞 6/9 是相同类型。YASA 可能从不同角度重复报告了文件名插入 view 命令的路径。无新的代码节点，源为用户文件名，汇为系统命令执行。

安全影响：同前，未经处理的文件名会导致任意命令执行。

漏洞 13/14/15：mailcap 外部内容命令注入（重复）

路径追踪：扫描结果的最后几个编号均指向 mailcap 配置内容引发的命令执行，与漏洞 10/11 属于反复提示。同一漏洞可能因多文件、多分支分析被列出多次。综合来看，mailcap.py 的核心问题已经明晰：无论是直接来自用户的输入，还是间接来自外部文件的内容，只要含有特殊字符进入了 os.system，就存在命令注入的风险。这些漏洞条目相互印证了这一点。

安全影响：同漏洞 3/4/10 所述，不做赘述。需要强调的是，CVE-2015-20107 已将此类风险定性为高危（CVSS 3.1 评分 7.6，高危），表明攻击难度低（仅需构造输入），但危害大（危及机密性、完整性和可用性）。因此，应当尽快修复或减轻此风险。

4. 结论

本次针对 Python 标准库 mailcap 模块的代码扫描和分析发现，该模块存在严重的安全缺陷：在构造系统命令时未对来自外部的参数进行适当的转义或校验，导致命令注入漏洞。CVE-2015-20107 正是对此问题的披露，其影响波及 Python 3.10.8 及之前的所有版本。攻击者可以利用恶意文件名、MIME 类型或篡改的 mailcap 配置，在调用 mailcap.findmatch 时插入任意 shell 命令，造成任意代码执行或文件路径遍历等后果。一旦利用此漏洞，攻击者可在受影响应用权限下执行系统命令，可能导致数据泄露、篡改，甚至远程接管服务器。