

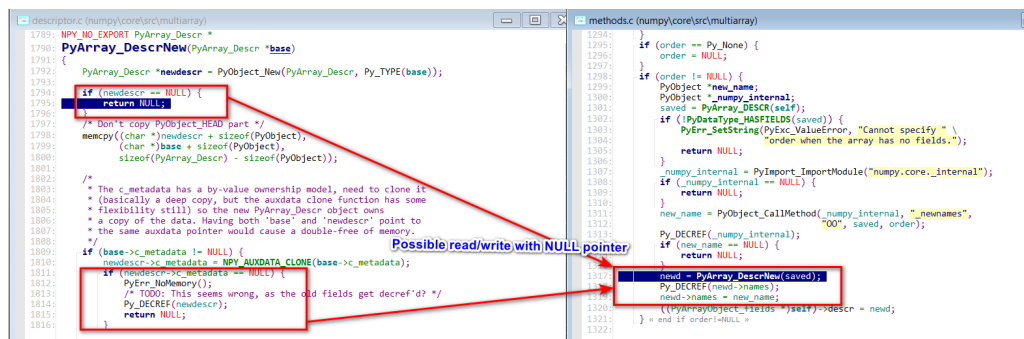
| Benchmark | CVES | #CVE |
|-----------|--|------|
| Numpy | CVE-2021-33430, CVE-2021-41495, CVE-2021-41496, CVE-2021-34141 | 4 |
| Bounter | CVE-2021-41497 | 1 |
| Cvxopt | CVE-2021-41500 | 1 |
| Pyo | CVE-2021-41498, CVE-2021-41499 | 2 |
| Total | | 8 |

| | |
|--------------------|--|
| Subject | NumPy |
| Vulnerability Type | Buffer overflow |
| Input | Integration test |
| Description | <pre> 657: * steals a reference to descr. On failure or descr->subarray, descr will 658: * be decref'd. 659: */ 660: NPY_NO_EXPORT PyObject * 661: PyArray_NewFromDescr_int(662: PyTypeObject *subtype, PyArray_Descr *descr, int nd, 663: npy_intp const *dims, npy_intp const *strides, void *data, 664: int flags, PyObject *obj, PyObject *base, int zeroed, 665: int allow_emptystring) 666: { 667: PyArrayObject_fields *fa; 668: int i; 669: npy_intp nbytes; 670: 671: if (descr->subarray) { 672: PyObject *ret; 673: npy_intp newdims[2*NPY_MAXDIMS]; 674: npy_intp *newstrides = NULL; 675: memcpy(newdims, dims, nd*sizeof(npy_intp)); 676: if (strides) { 677: newstrides = newdims + NPY_MAXDIMS; 678: memcpy(newstrides, strides, nd*sizeof(npy_intp)); 679: } 680: nd = _update_descr_and_dimensions(&descr, newdims, 681: newstrides, nd); 682: ret = PyArray_NewFromDescr_int(683: subtype, descr, 684: nd, newdims, newstrides, data, 685: flags, obj, base, 686: zeroed, allow_emptystring); 687: return ret; 688: } 689: } </pre> <p>external input</p> <p>Fixed size array</p> <p>No boundary check in the two operations of memcpy</p> <p>Affected component: Affect APIs for operations on the high-dimensional array</p> |

| | |
|-----------------|---|
| | <p>Attack vector: Construct a high-dimensional array file (npz); when the dimension is larger than 32, Python will crash when loading the array from the file.</p> <p>Buffer overflow in the operation of the high-dimensional array in NumPy (< 1.19) allows attackers to conduct DoS attacks by carefully constructing a npz file.</p> |
| Taint flow path | numpy.load (Python) -> PyArray_Fromfile -> PyArray_NewFromDescr_int -> memcpy |
| PoC | https://github.com/baltsers/polycruise/tree/main/numpy/vulnerability-1 |
| Status | Confirmed and fixed, CVE assigned: CVE-2021-33430 |
| Issue | https://github.com/numpy/numpy/issues/18939 |

| | |
|--------------------|--|
| Subject | Numpy |
| Vulnerability Type | Buffer overflow |
| Input | Integration test |
| Description | <pre> extern PyArrayObject* array_from_pyobj(const int type_num, npy_intp* dims, const int rank, const int intent, PyObject* obj) { /* * Note about reference counting: * ----- * If the caller returns the array to Python, it must be done with * Py_BuildValue("N",arr). * Otherwise, if obj!=arr then the caller must call Py_DECREF(arr). * * Note on intent(cache,out,...) * ----- * Don't expect correct data when returning intent(cache) array. */ char mess[200]; PyArrayObject* arr = NULL; PyArray_Descr* descr; char typechar; int elsize; if ((intent & F2PY_INTENT_HIDE) ((intent & F2PY_INTENT_CACHE) && (obj==Py_None)) ((intent & F2PY_OPTIONAL) && (obj==Py_None))) { /* intent(cache), optional, intent(hide) */ if (count_negative_dimensions(rank, dims) > 0) { int i; strcpy(mess, "failed to create intent(cache hide) optional array" " -- must have defined dimensions but got ('')."); for(i=0; i<rank; ++i) sprintf(mess+strlen(mess), "%s NPY_INTF_FMT ", dims[i]); strcat(mess, " "); PyErr_SetString(PyExc_ValueError, mess); return NULL; } arr = (PyArrayObject*) PyArray_New(&PyArray_Type, rank, dims, type_num, NULL, NULL, 1, !(intent&F2PY_INTENT_C), NULL); } } </pre> <p>Annotations in the image:</p> <ul style="list-style-type: none"> max of rank is F2PY_MAX_DIMS (40): Points to the <code>rank</code> parameter. strcpy copies 91 characters into buffer "mess": Points to the <code>strcpy</code> call. In its caller "fortran_setattr", values of dims are set to -1. Hence given the format string "%d", ("1",), the max length of this part would be 3*40=120 => 91 + 120 > 200: Points to the <code>dims[i]</code> in the <code>sprintf</code> call. <p>In the f2py module, when creating a high-dimension array through the array API (deliberately construct negative numbers in shape), an unexpected error</p> |

| | |
|------------------------|---|
| | occurs and causes Python to crash down. This allows attackers to conduct DoS attacks by carefully constructing an array with negative values in shape. |
| Taint flow path | numpy.setattr(Python) -> fortran_setattr -> array_from_pyobj |
| PoC | https://github.com/baltsers/polycruise/tree/main/numpy/vulnerability-3 |
| Status | Confirmed, CVE assigned: CVE-2021-41496 |
| Issue | https://github.com/numpy/numpy/issues/19000 |

| | |
|---------------------------|---|
| Subject | Numpy |
| Vulnerability Type | NULL pointer reference |
| Input | Integration test |
| Description |  <p>When constructing a loop to create high-dimension arrays repetitively, and keeping the references of the arrays effective, an error of NULL pointer access happens hence causing the Python to crash down. This allows attackers to conduct DoS attacks by repetitively creating and sort arrays.</p> |
| Taint flow path | numpy.sort (Python) -> array_sort -> PyArray_DescrNew |
| PoC | https://github.com/baltsers/polycruise/tree/main/numpy/vulnerability-2 |
| Status | Confirmed, CVE assigned: CVE-2021-41495 |
| Issue | https://github.com/numpy/numpy/issues/19038 |

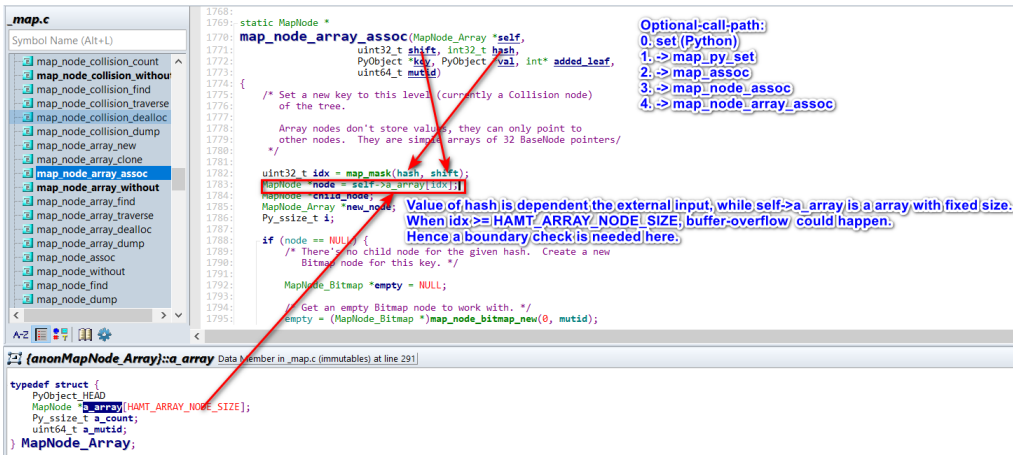
| | |
|--------------------|---|
| Subject | NumPy |
| Vulnerability Type | Incomplete string comparison |
| Input | Integration test |
| Description | <pre> 1725: /* Check for a deprecated Numeric-style typecode */ 1726: /* `Uint` has deliberately weird uppercasing */ 1727: char *dep_tps[] = {"Bytes", "Datetime64", "Str", "Uint"}; 1728: int ndep_tps = sizeof(dep_tps) / sizeof(dep_tps[0]); 1729: for (int i = 0; i < ndep_tps; ++i) { 1730: char *dep_tp = dep_tps[i]; 1731: 1732: if (strcmp(type, dep_tp, strlen(dep_tp)) == 0) { 1733: /* Deprecated 2020-06-09, NumPy 1.20 */ 1734: if (DEPRECATE("Numeric-style type codes are " 1735: "deprecated and will result in " 1736: "an error")) { 1737: goto fail; 1738: } 1739: } 1740: } </pre> <p>Unsecure for incomplete comparison (string comparison without considering terminator), and value of "type" may come from external modules.</p> |
| Taint flow path | numpy.empty(Python) -> PyArray_DescrAlignConverter -> _convert_from_any -> _convert_from_str -> strcmp |
| PoC | pending |
| Status | Confirmed, CVE assigned: CVE-2021-34141 |
| Issue | https://github.com/numpy/numpy/issues/18993 |

| | |
|--------------------|------------------|
| Subject | Numpy |
| Vulnerability Type | Integer overflow |
| Input | Integration test |

| | |
|-----------------|--|
| Description | <pre> if (trim_mode != TrimMode_None && numFractionDigits > 0) { --pCurOut; while (*pCurOut == '0') { --pCurOut; ++bufferSize; --numFractionDigits; } if (trim_mode == TrimMode_LeaveOneZero && *pCurOut == '.') { ++pCurOut; *pCurOut = '0'; --bufferSize; ++numFractionDigits; } ++pCurOut; } /* print the exponent into a local buffer and copy into output buffer */ if (bufferSize > 1) { char exponentBuffer[7]; npy_int32 digits[5]; npy_int32 i, exp_size, count; /* copy the exponent buffer into the output */ count = exp_size + 2; if (count > (npy_int32)bufferSize - 1) { count = (npy_int32)bufferSize - 1; } memcpy(pCurOut, exponentBuffer, count); pCurOut += count; bufferSize -= count; } DEBUG_ASSERT(bufferSize > 0); pCurOut[0] = '\0'; </pre> <p>should check "bufferSize > 0" here</p> <p>Affected branch if int-overflow happens</p> <p>If int-overflow happens, this boundary check is invalid. Hence buf-overflow may happen at the statement of memcpy.</p> |
| Taint flow path | numpy.format_float_scientific (Python) -> Dragon4_PrintFloat_IEEE_binary16 ->Format_floatbits-> FormatScientific -> bufferSize-1 |
| PoC | pending |
| Status | pending |
| Issue | https://github.com/numpy/numpy/issues/18937 |

| | |
|--------------------|------------------------|
| Subject | Bounter |
| Vulnerability Type | NULL pointer reference |
| Input | Integration test |

| | |
|-----------------|---|
| Description | <pre> 59: static int 60: CMS_VARIANT(_init)(CMS_TYPE *self, PyObject *args, PyObject *kwargs) 61: { 62: static char *kwlist[] = {"width", "depth", NULL}; 63: 64: uint32_t w; 65: if (!PyArg_ParseTupleAndKeywords(args, kwargs, "II", kwlist, 66: » » » » &w, &self->depth)) { 67: return -1; 68: } 69: 70: short int hash_length = -1; 71: while (0 != w) 72: hash_length++, w >>= 1; 73: if (hash_length < 0) 74: hash_length = 0; 75: self->width = 1 << hash_length; 76: self->hash_mask = self->width - 1; 77: 78: HyperLogLog_init(&self->hll, 16); 79: 80: self->table = (CMS_CELL_TYPE **) malloc(self->depth * sizeof(CMS_CELL_TYPE *)); 81: 82: int i; 83: for (i = 0; i < self->depth; i++) 84: { 85: self->table[i] = (CMS_CELL_TYPE *) calloc(self->width, sizeof(CMS_CELL_TYPE)); 86: } 87: return 0; 88: } 89: 90: 91: static inline PyObject * 92: CMS_VARIANT(_increment_obj)(CMS_TYPE *self, char *data, Py_ssize_t dataLength, long long increment) 93: { 94: 95: 96: int i; 97: for (i = 0; i < self->depth; i++) 98: { 99: MurmurHash3_x86_32((void *) data, dataLength, i, (void *) &hash); 100: uint32_t bucket = hash & self->hash_mask; 101: buckets[i] = bucket; 102: CMS_CELL_TYPE value = self->table[i][bucket]; 103: if (value < min_value) 104: min_value = value; 105: values[i] = self->table[i][bucket]; 106: 107: if (i == 0) 108: HyperLogLog_add(&self->hll, hash); 109: } </pre> <p>With carefully constructed inputs (when the width of the hash bucket is set large enough), NULL pointer access could happen hence causing the Python to crash down. This allows attackers to conduct DoS attacks by inputting a huge width of hash bucket.</p> <p>malloc and calloc may return NULL, hence cause NULL pointer references</p> |
| Taint flow path | <p>update (Python)-> CMS_Conservative_update -> CMS_Conservative_increment_obj increment (Python)-> CMS_Conservative_increment -> CMS_Conservative_increment_obj</p> |
| PoC | <p>https://github.com/baltsers/polycruise/tree/main/bouncer/vulnerability-1</p> |
| Status | <p>Confirmed and fixed, CVE Assigned: CVE-2021-41497</p> |
| Issue | <p>https://github.com/RaRe-Technologies/bouncer/issues/47</p> |
| Subject | <p>Immutableables</p> |

| | |
|--------------------|---|
| Vulnerability Type | Buffer overflow |
| Input | Integration test |
| Description |  <p>Optional-call-path: 0.->set(Python) 1.->map_py_set 2.->map_assoc 3.->map_node_assoc 4.->map_node_array_assoc</p> <p>Value of hash is dependent the external input, while self->a_array is a array with fixed size. When idx>=HANT_ARRAY_NODE_SIZE, buffer-overflow could happen. Hence a boundary check is needed here.</p> <p>(anonMapNode Array)::a_array Data Member in _map.c (immutables) at line 291</p> <pre>typedef struct { PyObject_HEAD MapNode *a_array[HANT_ARRAY_NODE_SIZE]; Py_ssize_t a_count; uint64_t a_mutid; } MapNode_Array;</pre> |
| Taint flow path | set (Python) -> map_py_set -> map_assoc -> map_node_assoc -> map_node_array_assoc |
| PoC | pending |
| Status | Pending |
| Issue | https://github.com/MagicStack/immutables/issues/67 |

| | |
|---------------------------|---|
| Subject | Japronto |
| Vulnerability Type | Unknown |
| Input | Integration test |
| Description | When passing byte stream context into the server API Response, the server runs abnormally, throws exceptions, and fails to deals with the following requests. This allows attackers to conduct DoS attacks. |
| Taint flow path | - |
| PoC | https://github.com/baltsers/polycruise/tree/main/japronto/vulnerability-1 |
| Status | Pending |
| Issue | https://github.com/squeaky-pl/japronto/issues/183 |

| | |
|--------------------|---|
| Subject | Cvxopt |
| Vulnerability Type | Incomplete string comparison |
| Input | Integration test |
| Description | <pre> static PyObject* spsolve(PyObject *self, PyObject *args, PyObject *kwargs) { spmatrix *B, *X=NULL; cholmod_sparse *Bc=NULL, *Xc=NULL; PyObject *F; cholmod_factor *L; int n, sys=0; #if PY_MAJOR_VERSION >= 3 const char *descr; #else char *descr; #endif char *kwlist[] = {"F", "B", "sys", NULL}; int sysvalues[] = {CHOLMOD_A, CHOLMOD_LDLt, CHOLMOD_LD, CHOLMOD_DLt, CHOLMOD_L, CHOLMOD_Lt, CHOLMOD_D, CHOLMOD_P, CHOLMOD_Pt }; if (!set_options()) return NULL; if (!PyArg_ParseTupleAndKeywords(args, kwargs, "OO i", kwlist, &F, &B, &ys)) return NULL; #if PY_MAJOR_VERSION >= 3 if (!PyCapsule_CheckExact(F) !(descr = PyCapsule_GetName(F))) err_CO("F"); if (strcmp(descr, "CHOLMOD_FACTOR", 14)) PY_ERR_TYPE("F is not a CHOLMOD factor"); L = (cholmod_factor *) PyCapsule_GetPointer(F, descr); #else if (!PyObject_Check(F)) err_CO("F"); descr = PyObject_GetDesc(F); if (descr strcmp(descr, "CHOLMOD_FACTOR", 14)) PY_ERR_TYPE("F is not a CHOLMOD factor"); L = (cholmod_factor *) PyObject_AsVoidPtr(F); #endif </pre> <p>Through carefully modify the name of a Capsule object, the PoC can easily bypass the validation in the spsolve API hence causing unexpected results (e.g., crash down during the execution of PoC). This allows attackers to conduct DoS attacks by construct fake Capsule objects.</p> |
| Taint flow path | spsolve (Python) -> spsolve -> strcmp |
| PoC | https://github.com/baltsers/polycruise/tree/main/cvxopt/vulnerability-4 |
| Status | Confirmed and fixed, CVE Assigned: CVE-2021-41500 |
| Issue | https://github.com/cvxopt/cvxopt/issues/193 |

| | |
|--------------------|--|
| Subject | Cvxopt |
| Vulnerability Type | Incomplete string comparison |
| Input | Integration test |
| Description | <pre> static PyObject* diag(PyObject *self, PyObject *args) { PyObject *F; matrix *d=NULL; cholmod_factor *L; #if PY_MAJOR_VERSION >= 3 const char *descr; #else char *descr; #endif int k, strt, incx=1, incy, nrows, ncols; if (!set_options()) return NULL; if (!PyArg_ParseTuple(args, "O", &F)) return NULL; #if PY_MAJOR_VERSION >= 3 if (!PyCapsule_CheckExact(F) ((descr = PyCapsule_GetName(F)) err_40("F"); if (strcmp(descr, "CHOLMOD FACTOR", 14)) PY_ERR_TYPE("F is not a CHOLMOD factor"); L = (cholmod_factor *) PyCapsule_GetPointer(F, descr); #else if (!PyObject_Check(F)) err_40("F"); descr = PyObject_GetDesc(F); if (!descr strcmp(descr, "CHOLMOD FACTOR", 14)) PY_ERR_TYPE("F is not a CHOLMOD factor"); L = (cholmod_factor *) PyObject_AsVoidPtr(F); #endif </pre> <p>Through carefully modify the name of a Capsule object, the PoC can easily bypass the validation in the diag API hence causing unexpected results (e.g., crash down during the execution of PoC). This allows attackers to conduct DoS attacks by construct fake Capsule objects.</p> |
| Taint flow path | diag(Python) -> diag -> strcmp |
| PoC | https://github.com/baltsers/polycruise/tree/main/cvxopt/vulnerability-1 |
| Status | Confirmed and fixed, CVE Assigned: CVE-2021-41500 |
| Issue | https://github.com/cvxopt/cvxopt/issues/193 |

| | |
|--------------------|--|
| Subject | Cvxopt |
| Vulnerability Type | Incomplete string comparison |
| Input | Integration test |
| Description | <pre> static PyObject* getfactor(PyObject *self, PyObject *args) { PyObject *F; cholmod_factor *Lf; cholmod_sparse *Ls; #if PY_MAJOR_VERSION >= 3 const char *descr; #else char *descr; #endif if (!set_options()) return NULL; if (!PyArg_ParseTuple(args, "O", &F)) return NULL; #if PY_MAJOR_VERSION >= 3 if (!PyCapsule_CheckExact(F) !(descr = PyCapsule_GetName(F))) err_CO("F"); if (strcmp(descr, "CHOLMOD FACTOR", 14)) PY_ERR_TYPE("F is not a CHOLMOD factor"); Lf = (cholmod_factor *) PyCapsule_GetPointer(F, descr); #else if (!PyObject_Check(F)) err_CO("F"); descr = PyObject_GetDescr(F); if (!descr strcmp(descr, "CHOLMOD FACTOR", 14)) PY_ERR_TYPE("F is not a CHOLMOD factor"); Lf = (cholmod_factor *) PyObject_AsVoidPtr(F); #endif </pre> <p>Through carefully modify the name of a Capsule object, the PoC can easily bypass the validation in the getfactor API hence causing unexpected results (e.g., crash down during the execution of PoC). This allows attackers to conduct DoS attacks by construct fake Capsule objects.</p> |
| Taint flow path | getfactor(Python) -> getfactor-> strcmp |
| PoC | https://github.com/baltsers/polycruise/tree/main/cvxopt/vulnerability-2 |
| Status | Confirmed and fixed, CVE Assigned: CVE-2021-41500 |
| Issue | https://github.com/cvxopt/cvxopt/issues/193 |

| | |
|--------------------|--|
| Subject | Cvxopt |
| Vulnerability Type | Incomplete string comparison |
| Input | Integration test |
| Description | <pre> static PyObject* solve(PyObject *self, PyObject *args, PyObject *kwargs) { if (!PyArg_ParseTupleAndKeywords(args, kwargs, "OO iiii", kwlist, &F, &B, &sys, &nrhs, &ldb, &oB)) return NULL; } #if PY_MAJOR_VERSION >= 3 if (!PyCapsule_CheckExact(F) !(descr = PyCapsule_GetName(F))) err_CO("F"); if (strcmp(descr, "CHOLMOD FACTOR", 14)) PY_ERR_TYPE("F is not a CHOLMOD factor"); cholmod_factor *L = (cholmod_factor *) PyCapsule_GetPointer(F, descr); #else if (!PyObject_Check(F)) err_CO("F"); descr = PyObject_GetDescr(F); if (!descr strcmp(descr, "CHOLMOD FACTOR", 14)) PY_ERR_TYPE("F is not a CHOLMOD factor"); cholmod_factor *L = (cholmod_factor *) PyObject_AsVoidPtr(F); #endif </pre> <p>Incomplete comparison (the terminator of descr) - It is a risk as descr comes from external module. Suggest to use sizeof("CHOLMOD FACTOR") instead.</p> <p>Through carefully modify the name of a Capsule object, the PoC can easily bypass the validation in the solve API hence causing unexpected results (e.g., crash down during the execution of PoC). This allows attackers to conduct DoS attacks by construct fake Capsule objects.</p> |
| Taint flow path | solve (Python) -> solve -> strcmp |
| PoC | https://github.com/baltsers/polycruise/tree/main/cvxopt/vulnerability-3 |
| Status | Confirmed and fixed, CVE Assigned: CVE-2021-41500 |
| Issue | https://github.com/cvxopt/cvxopt/issues/193 |

| | |
|--------------------|--|
| Subject | openjpeg2 |
| Vulnerability Type | Incomplete string comparison |
| Input | Integration test |
| Description | <pre> OPJ_BOOL identify_cid(SOCKET connected_socket, char refcid[], FILE *fp) { char *cid; OPJ_BOOL succeed; if(!cid = receive_string(connected_socket)){ fprintf(fp, "Error: error in identify_cid(), while receiving cid from client\n"); return OPJ_FALSE; } succeed = OPJ_FALSE; if(strcmp(refcid, cid, strlen(refcid)) == 0) succeed = OPJ_TRUE; opj_free(cid); return succeed; } </pre> <p>cid is read from socket. Ignoring the terminator could cause incomplete comparison further affected the following control flows</p> |
| Taint flow path | recv -> identify_id -> strcmp |
| PoC | pending |
| Status | Pending |
| Issue | https://github.com/uclouvain/openjpeg/issues/1357 |

| | |
|--------------------|--|
| Subject | libarchive |
| Vulnerability Type | Incomplete string comparison |
| Input | Integration test |
| Description | <pre> int archive_read_open_filenames(struct archive *a, const char **filenames, size_t block_size) { struct read_file_data *mine; const char *filename = NULL; if (filenames) filename = *(filenames++); archive_clear_error(a); do { if (filename == NULL) filename = ""; mine = (struct read_file_data *)calloc(1, sizeof(*mine) + strlen(filename)); if (mine == NULL) goto no_memory; strcpy(mine->filename.m, filename); mine->block_size = block_size; mine->fd = -1; mine->buffer = NULL; mine->st_mode = mine->use_lseek = 0; } while (0); int archive_read_open_filename_W(struct archive *a, const wchar_t *wfilename, size_t block_size) { struct read_file_data *mine = (struct read_file_data *)calloc(1, sizeof(*mine) + wcslen(wfilename) * sizeof(wchar_t)); if (!mine) { archive_set_error(a, ENOMEM, "No memory"); return (ARCHIVE_FATAL); } mine->fd = -1; mine->block_size = block_size; if (wfilename == NULL wfilename[0] == L'\0') { mine->filename_type = FNT_STDIN; } else { #if defined(_WIN32) && !defined(__CYGWIN__) mine->filename_type = FNT_WCS; wcsncpy(mine->filename.w, wfilename); #else /* * POSIX system does not support a wchar_t interface for * open() system call, so we have to translate a wchar_t * filename to multi-byte one and use it. */ struct archive_string fn; archive_string_init(&fn); if (archive_string_append_from_wcs(&fn, wfilename, wcslen(wfilename)) != 0) { if (errno == ENOMEM) archive_set_error(a, ENOMEM, "Can't allocate memory"); else archive_set_error(a, EINVAL, "Failed to convert a wide-character" " filename to a multi-byte filename"); archive_string_free(&fn); free(mine); return (ARCHIVE_FATAL); } mine->filename_type = FNT_MBS; strcpy(mine->filename.m, fn.s); archive_string_free(&fn); } } } </pre> <p>The length should be "strlen (filename) + 1" to ensure the terminator can be copied into the destination</p> <p>(the length should be "wcslen(wfilename) * sizeof(wchar_t) + 1" to ensure the terminator can be copied into the destination)</p> |
| Taint flow path | 1> read -> process_base_block -> process_head_main -> archive_read_open_filenames -> strcmp |

| | |
|--------|---|
| | 2> read -> process_base_block -> process_head_main -> archive_read_open_filenames_w -> strncmp |
| Status | Pending |
| PoC | pending |
| Issue | https://github.com/libarchive/libarchive/issues/1544 |

| | |
|--------------------|---|
| Subject | Pyo |
| Vulnerability Type | Buffer overflow |
| Input | Integration test |
| Description | <pre> int Server_jack_init(Server *self) { int i = 0; char client_name[32]; char name[16]; const char *server_name = "server"; jack_options_t options = JackNullOption; jack_status_t status; int sampleRate = 0; int bufferSize = 0; int nchnls = 0; int total_nchnls = 0; int index = 0; int ret = 0; assert(self->audio_be_data == NULL); PyJackBackendData *be_data = (PyJackBackendData *) PyMem_RawMalloc(sizeof(PyJackBackendData)); self->audio_be_data = (void *) be_data; be_data->activated = 0; strncpy(client_name, self->serverName, 31); Py_BEGIN_ALLOW_THREADS be_data->midi_event_count = 0; </pre> <p>When using the Pyo library with audio type "jack", the server is initialized with an overlong (over 32) string, an error of buffer overflow happens and causes Python to crash down. This allows attackers to conduct DoS attacks by arbitrary constructing a overlong server name.</p> <p>when the length of self->serverName>=31, it is possible that client_name has no terminator. Hence read overflow would happen.</p> |
| Taint flow path | boot (Python) -> Server_boot -> Server_jack_init |
| PoC | https://github.com/baltsers/polycruise/tree/main/pyo/vulnerability-1 |
| Status | Confirmed and fixed, CVE Assigned: CVE-2021-41498 |
| Issue | https://github.com/belangeo/pyo/issues/221 |

| | |
|---------|-----|
| Subject | Pyo |
|---------|-----|

| | |
|-----------------|---|
| Input | Integration test |
| Description | <pre> /* Debug messages should print internal information which might be necessary for debugging internal conditions. */ void Server_debug(Server *self, char * format, ...) { if (self->verbosity & 8) { char buffer[256]; va_list args; va_start (args, format); vsprintf (buffer, format, args); va_end (args); PySys_WriteStdout("Pyo debug: %s", buffer); } } </pre> <p>No boundary check with vsprintf. When a call-path "recstart (Python) -> Server_start_rec -> Server_start_rec_internal -> Server_debug (filename)" Suggest to use vsnprintf instead.</p> <p>After initializing a Pyo server, an arbitrary file name (length > 256) is passed to the recstart API and an error of segment fault happens hence causing Python to crash down. This allows attackers to conduct DoS attacks by deliberately passing on an overlong audio file name.</p> |
| Taint flow path | recstart (Python) -> Server_start_rec -> Server_start_rec_internal -> Server_debug |
| PoC | https://github.com/baltsers/polycruise/tree/main/pyo/vulnerability-2 |
| Status | Confirmed and fixed, CVE Assigned: CVE-2021-41499 |
| Issue | https://github.com/belangeo/pyo/issues/222 |