



Reasoning® Inspection Service

Reasoning  
Inspection Service  
Defect Data

Apache  
2.1-dev  
OPEN SOURCE

*30-Nov-2004*

Discovery Mapping Analytics™ is a mark of:



P.O. Box 478

Menlo Park, CA 94026-0478

+1 650-324-2510

[www.reasoning.com](http://www.reasoning.com)

# INTRODUCTION

---

## Reasoning™ Inspection Service

Reasoning Inspection Services for Java, C, and C++ provide essential expertise that boosts the productivity of development teams by finding software defects faster, earlier, and at a far lower cost than traditional approaches. The Reasoning service provides many of the benefits of a manual code review, but in significantly less time and at a dramatically lower cost. Reasoning detects and diagnoses defects well before they become discernible problems and provides a roadmap to the exact location for remedy and resolution.

Reasoning achieves these ends by automating software inspection, which enhances the organization's existing QA processes and bolsters efforts to provide easy-to-support, high-quality software. As a result, Reasoning enables fast time to ROI and permits software developers to focus on their core competency - software development.

## Deliverables

Reasoning provides two types of reports at the conclusion of the inspection process:

- The Defect Data report which makes defect analysis and repair simple by identifying the type and location of every defect and describing the circumstances under which they will occur. By providing this map to discovered defects, development time can be spent more effectively.
- The Defect Metrics report is designed for development managers. Providing a better insight into to problem areas within an application, it allows managers to better plan testing and development efforts.

This is the Defect Data report.

# SUMMARY DEFECT REPORT

---

## Inventory Summary

Reasoning inspected 360 user files in the Apache 2.1-dev 1/31/2003 code, including all the source files.

Total Number of Source Files:	169
Number of User Include Files:	191
<b>Total Number of User Files Processed:</b>	<b>360</b>
Total LOC of Source Files:	58,944
Number LOC User Include Files:	17,264
<b>Total LOC in Project:</b>	<b>76,208</b>

## Defect Summary

The column *Defect Instances* in the table below details, per defect class, how many defects there are in the application.

The column *Files Affected* details, per defect class, the number of files in the application that have one or more defects.

Inspection Class	Defect Instances	Files Affected
Memory Leak <i>Reference to allocated memory is lost</i>	0	0
NULL Pointer Dereference <i>Expression dereferences a NULL pointer</i>	29	20
Bad Deallocation <i>Deallocation is inappropriate for type of data</i>	0	0
Out of Bounds Array Access <i>Expression accesses a value beyond the array</i>	0	0
Uninitialized Variable <i>Variable is not initialized prior to use</i>	2	2
<b>Total Defect Instances</b>	<b>31</b>	<b>---</b>

## DETAILED DEFECT REPORT

---

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b>	1
<b>LOCATION:</b>	httpd-2.1/modules/aaa/mod_auth_basic.c : 291		
<b>DESCRIPTION</b>	The local pointer variable <b>current_provider</b> , declared on line <b>235</b> , and assigned on line <b>257</b> , may be NULL where it is dereferenced on line <b>291</b> .		
<b>PRECONDITIONS</b>	<p>The conditional expression (<b>res</b>) on line <b>253</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>!current_provider</b>) on line <b>264</b> evaluates to <b>true</b> AND</p> <p>The conditional expression (<b>!provider    !provider-&gt;check_password</b>) on line <b>268</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>auth_result != AUTH_USER_NOT_FOUND</b>) on line <b>282</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>!conf-&gt;providers</b>) on line <b>287</b> evaluates to <b>false</b>.</p>		

### CODE FRAGMENT

```

228 static int authenticate_basic_user(request_rec *r)
229 {
230     auth_basic_config_rec *conf = ap_get_module_config(r->per_dir_config,
231                                                     &auth_basic_module);
232     const char *sent_user, *sent_pw, *current_auth;
233     int res;
234     authn_status auth_result;
235     authn_provider_list *current_provider;
236
237     /* Are we configured to be Basic auth? */
238     current_auth = ap_auth_type(r);
239     if (!current_auth || strcasecmp(current_auth, "Basic")) {
240         return DECLINED;
241     }
242
243     /* We need an authentication realm. */
244     if (!ap_auth_name(r)) {
245         ap_log_rerror(APLOG_MARK, APLOG_ERR,
246                     0, r, "need AuthName: %s", r->uri);
247         return HTTP_INTERNAL_SERVER_ERROR;
248     }
249
250     r->ap_auth_type = "Basic";
251
252     res = get_basic_auth(r, &sent_user, &sent_pw);
253     if (res) {
254         return res;
255     }
256
257     current_provider = conf->providers;
258     do {
259         const authn_provider *provider;
260
261         /* For now, if a provider isn't set, we'll be nice and use the file
262          * provider.
263          */
264         if (!current_provider) {
265             provider = ap_lookup_provider(AUTHN_PROVIDER_GROUP,
266                                         AUTHN_DEFAULT_PROVIDER, "0");

```

```

267
268     if (!provider || !provider->check_password) {
269         ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r,
270                     "No Authn provider configured");
271         auth_result = AUTH_GENERAL_ERROR;
272         break;
273     }
274 }
275 else {
276     provider = current_provider->provider;
277 }
278
279 auth_result = provider->check_password(r, sent_user, sent_pw);
280
281 /* Something occurred. Stop checking. */
282 if (auth_result != AUTH_USER_NOT_FOUND) {
283     break;
284 }
285
286 /* If we're not really configured for providers, stop now. */
287 if (!conf->providers) {
288     break;
289 }
290
291 current_provider = current_provider->next;
292 } while (current_provider);
293
294 if (auth_result != AUTH_GRANTED) {
295     int return_code;
296
297     /* If we're not authoritative, then any error is ignored. */
298     if (!(conf->authoritative) && auth_result != AUTH_DENIED) {
299         return DECLINED;
300     }

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID 2****LOCATION:** httpd-2.1/modules/filters/mod\_include.c : 1406**DESCRIPTION** The local pointer variable **echo\_text**, declared on line **1368**, and assigned on line **1402**, may be NULL where it is dereferenced on line **1406**. This NULL pointer dereference only happens in an Out Of Memory context.**PRECONDITIONS** The conditional expression (**ctx->flags & FLAG\_PRINTING**) on line **1376** evaluates to **true** ANDThe conditional expression (**tag\_val == NULL**) on line **1379** evaluates to **false** ANDThe conditional expression (**!strcmp(tag, "var")**) on line **1387** evaluates to **true** ANDThe conditional expression (**val**) on line **1393** evaluates to **true** ANDThe case statement **E\_ENTITY** on line **1401** is executed.**CODE FRAGMENT**

```

1362 static int handle_echo(include_ctx_t *ctx, apr_bucket_brigade **bb,
1363                        request_rec *r, ap_filter_t *f, apr_bucket *head_ptr,
1364                        apr_bucket **inserted_head)
1365 {
1366     char          *tag          = NULL;
1367     char          *tag_val      = NULL;
1368     const char    *echo_text    = NULL;
1369     apr_bucket    *tmp_buck;
1370     apr_size_t    e_len;
1371     enum {E_NONE, E_URL, E_ENTITY} encode;
1372
1373     encode = E_ENTITY;
1374
1375     *inserted_head = NULL;
1376     if (ctx->flags & FLAG_PRINTING) {
1377         while (1) {
1378             ap_ssi_get_tag_and_value(ctx, &tag, &tag_val, 1);
1379             if (tag_val == NULL) {
1380                 if (tag != NULL) {
1381                     return 1;
1382                 }
1383                 else {
1384                     return 0;
1385                 }
1386             }
1387             if (!strcmp(tag, "var")) {
1388                 conn_rec *c = r->connection;
1389                 const char *val =
1390                     get_include_var(r, ctx,
1391                                   ap_ssi_parse_string(r, ctx, tag_val, NULL,
1392                                                       MAX_STRING_LEN, 0));
1393                 if (val) {
1394                     switch(encode) {
1395                         case E_NONE:
1396                             echo_text = val;
1397                             break;
1398                         case E_URL:
1399                             echo_text = ap_escape_uri(r->pool, val);
1400                             break;
1401                         case E_ENTITY:
1402                             echo_text = ap_escape_html(r->pool, val);
1403                             break;

```

```
1404     }
1405
1406     e_len = strlen(echo_text);
1407     tmp_buck = apr_bucket_pool_create(echo_text, e_len,
1408                                     r->pool, c-
>bucket_alloc);
1409 }
1410 else {
1411     include_server_config *sconf=
1412         ap_get_module_config(r->server->module_config,
1413                             &include_module);
1414     tmp_buck = apr_bucket_pool_create(sconf->undefinedEcho,
1415                                     sconf->undefinedEchoLen,
1416                                     r->pool, c-
>bucket_alloc);
```

**DEFECT CLASS:** Null Pointer Dereference **DEFECT ID 3**

**LOCATION:** httpd-2.1/modules/filters/mod\_include.c : 2823

**DESCRIPTION** The local pointer variable **tag**, declared on line **2797**, and assigned on line **2816**, may be NULL where it is dereferenced on line **2823**. A similar error can be found on line 2827.

**PRECONDITIONS** The conditional expression (**ctx->flags & FLAG\_PRINTING**) on line **2814** evaluates to **true** AND

The conditional expression (**tag == NULL**) on line **2817** evaluates to **true** AND

The conditional expression (**tag\_val == NULL**) on line **2817** evaluates to **false** AND

The conditional expression (**tag\_val == NULL**) on line **2820** evaluates to **false**.

**CODE FRAGMENT**

```

2793 static int handle_set(include_ctx_t *ctx, apr_bucket_brigade **bb,
2794                       request_rec *r, ap_filter_t *f, apr_bucket *head_ptr,
2795                       apr_bucket **inserted_head)
2796 {
2797     char *tag      = NULL;
2798     char *tag_val  = NULL;
2799     char *var      = NULL;
2800     apr_bucket *tmp_buck;
2801     char *parsed_string;
2802     request_rec *sub = r->main;
2803     apr_pool_t *p = r->pool;
2804
2805     /* we need to use the 'main' request pool to set notes as that is
2806      * a notes lifetime
2807      */
2808     while (sub) {
2809         p = sub->pool;
2810         sub = sub->main;
2811     }
2812
2813     *inserted_head = NULL;
2814     if (ctx->flags & FLAG_PRINTING) {
2815         while (1) {
2816             ap_ssi_get_tag_and_value(ctx, &tag, &tag_val, 1);
2817             if ((tag == NULL) && (tag_val == NULL)) {
2818                 return 0;
2819             }
2820             else if (tag_val == NULL) {
2821                 return 1;
2822             }
2823             else if (!strcmp(tag, "var")) {
2824                 var = ap_ssi_parse_string(r, ctx, tag_val, NULL,
2825                                         MAX_STRING_LEN, 0);
2826             }
2827             else if (!strcmp(tag, "value")) {
2828                 if (var == (char *) NULL) {
2829                     ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r,
2830                                 "variable must precede value in set directive in
2831                                 %s",
2832                                 r->filename);
2833                     CREATE_ERROR_BUCKET(ctx, tmp_buck, head_ptr,
2834                                         *inserted_head);

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 4**LOCATION:** httpd-2.1/modules/filters/mod\_include.c : 2952**DESCRIPTION** The local pointer variable **tmp\_bkt**, declared on line **2947**, and assigned on line **2949**, may be NULL where it is dereferenced on line **2952**. This NULL pointer dereference only happens in an Out Of Memory context.**PRECONDITIONS** The conditional expression **(dptr != APR\_BRIGADE\_SENTINEL(\*bb) && !APR\_BUCKET\_IS\_EOS(dptr))** on line **2931** evaluates to **true** ANDThe conditional expression **((ctx->state == PRE\_HEAD) || (ctx->state == PARSE\_HEAD))** on line **2933** evaluates to **true** ANDThe conditional expression **(!APR\_STATUS\_IS\_SUCCESS(ctx->status))** on line **2938** evaluates to **false** ANDThe conditional expression **((do\_cleanup) && (!APR\_BRIGADE\_EMPTY(ctx->ssi\_tag\_brigade))** on line **2946** evaluates to **true** ANDThe function **apr\_bucket\_immortal\_create**, called on line **2949**, returns NULL.**CODE FRAGMENT**

```

2913 static apr_status_t send_parsed_content(apr_bucket_brigade **bb,
2914                                         request_rec *r, ap_filter_t *f)
2915 {
2916     include_ctx_t *ctx = f->ctx;
2917     apr_bucket *dptr = APR_BRIGADE_FIRST(*bb);
2918     apr_bucket *tmp_dptr;
2919     apr_bucket_brigade *tag_and_after;
2920     apr_status_t rv = APR_SUCCESS;
2921
2922     if (r->args) { /* add QUERY stuff to env cause it ain't yet
*/
2923         char *arg_copy = apr_pstrdup(r->pool, r->args);
2924
2925         apr_table_setn(r->subprocess_env, "QUERY_STRING", r->args);
2926         ap_unescape_url(arg_copy);
2927         apr_table_setn(r->subprocess_env, "QUERY_STRING_UNESCAPED",
2928                       ap_escape_shell_cmd(r->pool, arg_copy));
2929     }
2930
2931     while (dptr != APR_BRIGADE_SENTINEL(*bb) && !APR_BUCKET_IS_EOS(dptr)) {
2932         /* State to check for the STARTING_SEQUENCE. */
2933         if ((ctx->state == PRE_HEAD) || (ctx->state == PARSE_HEAD)) {
2934             int do_cleanup = 0;
2935             apr_size_t cleanup_bytes = ctx->parse_pos;
2936
2937             tmp_dptr = find_start_sequence(dptr, ctx, *bb, &do_cleanup);
2938             if (!APR_STATUS_IS_SUCCESS(ctx->status)) {
2939                 return ctx->status;
2940             }
2941
2942             /* The few bytes stored in the ssi_tag_brigade turned out not to
2943              * be a tag after all. This can only happen if the starting
2944              * tag actually spans brigades. This should be very rare.
2945              */
2946             if ((do_cleanup) && (!APR_BRIGADE_EMPTY(ctx->ssi_tag_brigade))) {
2947                 apr_bucket *tmp_bkt;
2948
2949                 tmp_bkt = apr_bucket_immortal_create(ctx->start_seq,
2950                                                     cleanup_bytes,

```

```
2951                                     r->connection-
>bucket_alloc);
2952 APR_BRIGADE_INSERT_HEAD(*bb, tmp_bkt);
2953     apr_brigade_cleanup(ctx->ssi_tag_brigade);
2954 }
2955
2956 /* If I am inside a conditional (if, elif, else) that is false
2957  * then I need to throw away anything contained in it.
2958  */
2959 if (!(ctx->flags & FLAG_PRINTING) && (tmp_dptra != NULL) &&
2960     (dptra != APR_BRIGADE_SENTINEL(*bb))) {
2961     while ((dptra != APR_BRIGADE_SENTINEL(*bb)) &&
2962           (dptra != tmp_dptra)) {
```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b> 5
<b>LOCATION:</b>	httpd-2.1/modules/generators/mod_cgi.c : 716	
<b>DESCRIPTION</b>	The pointer expression <b>dbuf</b> used in the <b>index</b> expression <b>dbuf + dbpos</b> on line <b>716</b> may be NULL. Although this is not an immediate dereference of a NULL pointer, the expression <b>dbuf</b> is not a valid pointer. A similar error can be found on line 735.	
<b>PRECONDITIONS</b>	<p>The conditional expression (<b>conf-&gt;logname</b>) on line <b>671</b> evaluates to <b>true</b> AND</p> <p>The conditional expression (<b>APR_BUCKET_IS_EOS(bucket)</b>) on line <b>689</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>APR_BUCKET_IS_FLUSH(bucket)</b>) on line <b>695</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>child_stopped_reading</b>) on line <b>700</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>conf-&gt;logname &amp;&amp; dbpos &lt; conf-&gt;bufbytes</b>) on line <b>707</b> evaluates to <b>true</b>.</p>	

**CODE FRAGMENT**

```

570 static int cgi_handler(request_rec *r)
571 {
...
577     char *dbuf = NULL;
...
671     if (conf->logname) {
672         dbuf = apr_palloc(r->pool, conf->bufbytes + 1);
673         dbpos = 0;
674     }
675     do {
676         apr_bucket *bucket;
677
678         rv = ap_get_brigade(r->input_filters, bb, AP_MODE_READBYTES,
679                             APR_BLOCK_READ, HUGE_STRING_LEN);
680
681         if (rv != APR_SUCCESS) {
682             return rv;
683         }
684
685         APR_BRIGADE_FOREACH(bucket, bb) {
686             const char *data;
687             apr_size_t len;
688
689             if (APR_BUCKET_IS_EOS(bucket)) {
690                 seen_eos = 1;
691                 break;
692             }
693
694             /* We can't do much with this. */
695             if (APR_BUCKET_IS_FLUSH(bucket)) {
696                 continue;
697             }
698
699             /* If the child stopped, we still must read to EOS. */
700             if (child_stopped_reading) {
701                 continue;
702             }
703
704             /* read */
705             apr_bucket_read(bucket, &data, &len, APR_BLOCK_READ);

```

```
706
707     if (conf->logname && dbpos < conf->bufbytes) {
708         int cursize;
709
710         if ((dbpos + len) > conf->bufbytes) {
711             cursize = conf->bufbytes - dbpos;
712         }
713         else {
714             cursize = len;
715         }
716         memcpy(dbuf + dbpos, data, cursize);
717         dbpos += cursize;
718     }
719
720     /* Keep writing data to the child until done or too much time
721      * elapses with no progress or an error occurs.
722      */
723     rv = apr_file_write_full(script_out, data, len, NULL);
724
725     if (rv != APR_SUCCESS) {
726         /* silly script stopped reading, soak up remaining message *
```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b> 6
<b>LOCATION:</b>	httpd-2.1/modules/generators/mod_cgi.c : 749	
<b>DESCRIPTION</b>	The local pointer variable <b>b</b> , declared on line <b>580</b> , and assigned on line <b>748</b> , may be NULL where it is dereferenced on line <b>749</b> . This NULL pointer dereference only happens in an Out Of Memory context. Similar errors can be found on lines 751, 819 and 821.	
<b>PRECONDITIONS</b>	<p>The conditional expression (<b>APR_BUCKET_IS_EOS(bucket)</b>) on line <b>689</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>APR_BUCKET_IS_FLUSH(bucket)</b>) on line <b>695</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>child_stopped_reading</b>) on line <b>700</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>script_in &amp;&amp; !nph</b>) on line <b>742</b> evaluates to <b>true</b> AND</p> <p>The function <b>apr_bucket_pipe_create</b>, called on line <b>748</b>, returns NULL.</p>	

**CODE FRAGMENT**

```

570 static int cgi_handler(request_rec *r)
571 {
...
580     apr_bucket *b;
...
681     if (rv != APR_SUCCESS) {
682         return rv;
683     }
684
685     APR_BRIGADE_FOREACH(bucket, bb) {
686         const char *data;
687         apr_size_t len;
688
689         if (APR_BUCKET_IS_EOS(bucket)) {
690             seen_eos = 1;
691             break;
692         }
693
694         /* We can't do much with this. */
695         if (APR_BUCKET_IS_FLUSH(bucket)) {
696             continue;
697         }
698
699         /* If the child stopped, we still must read to EOS. */
700         if (child_stopped_reading) {
701             continue;
702         }
703
704         /* read */
705         apr_bucket_read(bucket, &data, &len, APR_BLOCK_READ);
...
723         rv = apr_file_write_full(script_out, data, len, NULL);
724
725         if (rv != APR_SUCCESS) {
726             /* silly script stopped reading, soak up remaining message */
727             child_stopped_reading = 1;
728         }
729     }
730     apr_brigade_cleanup(bb);

```

```

731     }
732     while (!seen_eos);
733
734     if (conf->logname) {
735         dbuf[dbpos] = '\0';
736     }
737     /* Is this flush really needed? */
738     apr_file_flush(script_out);
739     apr_file_close(script_out);
740
741     /* Handle script return... */
742     if (script_in && !nph) {
743         conn_rec *c = r->connection;
744         const char *location;
745         char sbuf[MAX_STRING_LEN];
746         int ret;
747
748         b = apr_bucket_pipe_create(script_in, c->bucket_alloc);
749         APR_BRIGADE_INSERT_TAIL(bb, b);
750         b = apr_bucket_eos_create(c->bucket_alloc);
751         APR_BRIGADE_INSERT_TAIL(bb, b);
752
753         if ((ret = ap_scan_script_header_err_brigade(r, bb, sbuf))) {
754             return log_script(r, conf, ret, dbuf, sbuf, bb, script_err);
755         }
756
757         location = apr_table_get(r->headers_out, "Location");
758
759         if (location && location[0] == '/' && r->status == 200) {

```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b> 7
<b>LOCATION:</b>	httpd-2.1/modules/http/http_core.c : 220	
<b>DESCRIPTION</b>	The local pointer variable <b>e</b> , declared on line <b>149</b> , and assigned on line <b>218</b> , may be NULL where it is dereferenced on line <b>220</b> . This NULL pointer dereference only happens in an Out Of Memory context. A similar error can be found on line 234.	
<b>PRECONDITIONS</b>	<p>The conditional expression (<b>APR_BUCKET_IS_EOS(e)</b>) on line <b>162</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>APR_BUCKET_IS_FLUSH(e)</b>) on line <b>167</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>e-&gt;length == (apr_size_t)-1</b>) on line <b>170</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>bytes &gt; 0</b>) on line <b>208</b> evaluates to <b>true</b> AND</p> <p>The function <b>apr_bucket_transient_create</b>, called on line <b>218</b>, returns NULL.</p>	

**CODE FRAGMENT**

```

143 static apr_status_t chunk_filter(ap_filter_t *f, apr_bucket Brigade *b)
144 {
...
149     apr_bucket *e;
...
152     for (more = NULL; b; b = more, more = NULL) {
153         apr_off_t bytes = 0;
154         apr_bucket *eos = NULL;
155         apr_bucket *flush = NULL;
...
159         char chunk_hdr[20]; /* enough space for the sprintf below */
160
161         APR_BRIGADE_FOREACH(e, b) {
162             if (APR_BUCKET_IS_EOS(e)) {
163                 /* there shouldn't be anything after the eos */
164                 eos = e;
165                 break;
166             }
167             if (APR_BUCKET_IS_FLUSH(e)) {
168                 flush = e;
169             }
170             else if (e->length == (apr_size_t)-1) {
171                 /* unknown amount of data (e.g. a pipe) */
172                 const char *data;
173                 apr_size_t len;
174
175                 rv = apr_bucket_read(e, &data, &len, APR_BLOCK_READ);
176                 if (rv != APR_SUCCESS) {
177                     return rv;
178                 }
179                 if (len > 0) {
...
185                     bytes += len;
186                     more = apr_brigade_split(b, APR_BUCKET_NEXT(e));
187                     break;
188                 }
189                 else {
...
194                     continue;
195                 }
196             }
197             else {

```

```

198         bytes += e->length;
199     }
200 }
...
208 if (bytes > 0) {
209     apr_size_t hdr_len;
...
215     hdr_len = apr_snprintf(chunk_hdr, sizeof(chunk_hdr),
216                          "%qx" CRLF, (apr_uint64_t)bytes);
217     ap_xlate_proto_to_ascii(chunk_hdr, hdr_len);
218     e = apr_bucket_transient_create(chunk_hdr, hdr_len,
219                                   c->bucket_alloc);
220     APR_BRIGADE_INSERT_HEAD(b, e);
221
222     /*
223     * Insert the end-of-chunk CRLF before an EOS or
224     * FLUSH bucket, or appended to the brigade
225     */
226     e = apr_bucket_immortal_create(ASCII_CRLF, 2, c->bucket_alloc);
227     if (eos != NULL) {
228         APR_BUCKET_INSERT_BEFORE(eos, e);
229     }
230     else if (flush != NULL) {

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 8**LOCATION:** httpd-2.1/modules/http/http\_protocol.c : 830**DESCRIPTION** The local pointer variable **e**, declared on line **766**, and assigned on line **828**, may be NULL where it is dereferenced on line **830**. This NULL pointer dereference only happens in an Out Of Memory context. Similar errors can be found on lines 832, 849, 851, 870, 888, 890, 921, 923, 933, 942, 952, 988, 990, 1000, 1058, 1060 and 1563.**PRECONDITIONS** The conditional expression (**!ctx**) on line **776** evaluates to **true** AND  
The conditional expression (**tenc**) on line **799** evaluates to **false** AND  
The conditional expression (**lenp**) on line **804** evaluates to **true** AND  
The conditional expression (**conversion\_error**) on line **821** evaluates to **true** AND  
The function **ap\_bucket\_error\_create**, called on line **828**, returns NULL.**CODE FRAGMENT**

```

762 apr_status_t ap_http_filter(ap_filter_t *f, apr_bucket Brigade *b,
763                             ap_input_mode_t mode, apr_read_type_e block,
764                             apr_off_t readbytes)
765 {
766     apr_bucket *e;
767     http_ctx_t *ctx = f->ctx;
768     apr_status_t rv;
769     apr_off_t totalread;
770
771     /* just get out of the way of things we don't want. */
772     if (mode != AP_MODE_READBYTES && mode != AP_MODE_GETLINE) {
773         return ap_get_brigade(f->next, b, mode, block, readbytes);
774     }
775
776     if (!ctx) {
777         const char *tenc, *lenp;
778         f->ctx = ctx = apr_palloc(f->r->pool, sizeof(*ctx));
779         ctx->state = BODY_NONE;
780         ctx->remaining = 0;
781         ctx->limit_used = 0;
782         ctx->eos_sent = 0;
783
784         ...
789         if (!f->r->proxyreq) {
790             ctx->limit = ap_get_limit_req_body(f->r);
791         }
792         else {
793             ctx->limit = 0;
794         }
795
796         tenc = apr_table_get(f->r->headers_in, "Transfer-Encoding");
797         lenp = apr_table_get(f->r->headers_in, "Content-Length");
798
799         if (tenc) {
800             if (!strcasecmp(tenc, "chunked")) {
801                 ctx->state = BODY_CHUNK;
802             }
803         }
804         else if (lenp) {
805             int conversion_error = 0;
806             char *endstr;
807
808             ctx->state = BODY_LENGTH;
809             errno = 0;

```

```

810         ctx->remaining = strtol(lenp, &endstr, 10);    /* we depend on ANSI
*/
...
817     if (errno || (endstr && *endstr) || (ctx->remaining < 0)) {
818         conversion_error = 1;
819     }
820
821     if (conversion_error) {
822         apr_bucket_brigade *bb;
823
824         ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, f->r,
825                     "Invalid Content-Length");
826
827         bb = apr_brigade_create(f->r->pool, f->c->bucket_alloc);
828         e = ap_bucket_error_create(HTTP_REQUEST_ENTITY_TOO_LARGE, NULL,
829                                   f->r->pool, f->c->bucket_alloc);
830         APR_BRIGADE_INSERT_TAIL(bb, e);
831         e = apr_bucket_eos_create(f->c->bucket_alloc);
832         APR_BRIGADE_INSERT_TAIL(bb, e);
833         ctx->eos_sent = 1;
834         return ap_pass_brigade(f->r->output_filters, bb);
835     }
836
837     /* If we have a limit in effect and we know the C-L ahead of
838     * time, stop it here if it is invalid.
839     */
840     if (ctx->limit && ctx->limit < ctx->remaining) {

```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b> 9
<b>LOCATION:</b>	httpd-2.1/modules/mappers/mod_negotiation.c : 1157	
<b>DESCRIPTION</b>	The local pointer variable <b>segend</b> , declared on line <b>1152</b> , and assigned on line <b>1156</b> , may be NULL where it is dereferenced on line <b>1157</b> . Similar errors can be found on lines 1172 and 1173.	
<b>PRECONDITIONS</b>	<p>The conditional expression (<b>sub_req-&gt;finfo.filetype != APR_REG</b>) on line <b>1114</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>!exception_list</b>) on line <b>1138</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>*segstart &amp;&amp; nexcept</b>) on line <b>1154</b> evaluates to <b>true</b> AND</p> <p>The conditional expression (<b>!(segend = strchr(segstart, '.'))</b>) on line <b>1155</b> evaluates to <b>true</b> AND</p> <p>The function <b>strchr</b>, called on line <b>1156</b>, returns NULL.</p>	

**CODE FRAGMENT**

```

1039 static int read_types_multi(negotiation_state *neg)
1040 {
...
1114     if (sub_req->finfo.filetype != APR_REG)
1115         continue;
1116
1117     /* If it has a handler, we'll pretend it's a CGI script,
1118     * since that's a good indication of the sort of thing it
1119     * might be doing.
1120     */
1121     if (sub_req->handler && !sub_req->content_type) {
1122         ap_set_content_type(sub_req, CGI_MAGIC_TYPE);
1123     }
1124
1125     /*
1126     * mod_mime will always provide us the base name in the
1127     * ap-mime-exception-list, if it processed anything. If
1128     * this list is empty, give up immediately, there was
1129     * nothing interesting. For example, looking at the files
1130     * readme.txt and readme.foo, we will throw away .foo if
1131     * it's an insignificant file (e.g. did not identify a
1132     * language, charset, encoding, content type or handler,)
1133     */
1134     exception_list =
1135         (apr_array_header_t *)apr_table_get(sub_req->notes,
1136                                             "ap-mime-exceptions-list");
1137
1138     if (!exception_list) {
1139         ap_destroy_sub_req(sub_req);
1140         continue;
1141     }
1142
1143     /* Each unrecognized bit better match our base name, in sequence.
1144     * A test of index.html.foo will match index.foo or index.html.foo,
1145     * but it will never transpose the segments and allow index.foo.html
1146     * because that would introduce too much CPU consumption. Better that
1147     * we don't attempt a many-to-many match here.
1148     */
1149     {
1150         int nexcept = exception_list->nelts;
1151         char **cur_except = (char**)exception_list->elts;

```

```
1152         char *segstart = filp, *segend, saveend;
1153
1154         while (*segstart && nexcept) {
1155             if (!(segend = strchr(segstart, '.')))
1156                 segend = strchr(segstart, '\\0');
1157             saveend = *segend;
1158             *segend = '\\0';
1159
1160 #ifdef CASE_BLIND_FILESYSTEM
1161             if (strcasecmp(segstart, *cur_except) == 0) {
1162 #else
1163             if (strcmp(segstart, *cur_except) == 0) {
1164 #endif
1165                 --nexcept;
1166                 ++cur_except;
1167             }
```

**DEFECT CLASS:** Null Pointer Dereference **DEFECT ID** 10

**LOCATION:** httpd-2.1/modules/mappers/mod\_negotiation.c : 2495

**DESCRIPTION** The local pointer variable **arr**, declared on line **2349**, and assigned on line **2365**, may be NULL where it is dereferenced on line **2495**. This NULL pointer dereference only happens in an Out Of Memory context.

**PRECONDITIONS** The conditional expression (**neg->send\_alternates && neg->avail\_vars->nelts**) on line **2364** evaluates to **true** AND

The function **apr\_array\_make**, called on line **2365**, returns NULL AND

The conditional expression (**neg->send\_alternates && neg->avail\_vars->nelts**) on line **2494** evaluates to **true**.

**CODE FRAGMENT**

```

2336 static void set_neg_headers(request_rec *r, negotiation_state *neg,
2337                             int alg_result)
2338 {
2339     apr_array_header_t *arr;
2340
2341     if (neg->send_alternates && neg->avail_vars->nelts)
2342         arr = apr_array_make(r->pool, max_vlist_array, sizeof(char *));
2343     else
2344         arr = NULL;
2345
2346     if (neg->send_alternates && neg->avail_vars->nelts) {
2347         arr->nelts--; /* remove last comma */
2348         apr_table_mergen(hdrs, "Alternates",
2349                         apr_array_pstrcat(r->pool, arr, '\0'));
2350     }
2351
2352     if (neg->is_transparent || vary_by_type || vary_by_language ||
2353         vary_by_language || vary_by_charset || vary_by_encoding) {
2354         apr_table_mergen(hdrs, "Vary", 2 + apr_pstrcat(r->pool,
2355             neg->is_transparent ? ", negotiate" : "",
2356             vary_by_type ? ", accept" : "",

```

**DEFECT CLASS:** Null Pointer Dereference **DEFECT ID** 11

**LOCATION:** httpd-2.1/modules/mappers/mod\_negotiation.c : 2939

**DESCRIPTION** The local pointer variable **e**, declared on line **2880**, and assigned on line **2938**, may be NULL where it is dereferenced on line **2939**.

**PRECONDITIONS** The conditional expression **(best->body)** on line **2876** evaluates to **true** AND

The conditional expression **(r->method\_number != M\_GET && r->method\_number != M\_POST)** on line **2887** evaluates to **false** AND

The conditional expression **((res = ap\_meets\_conditions(r)) != OK)** on line **2926** evaluates to **false** AND

The conditional expression **((res = ap\_discard\_request\_body(r)) != OK)** on line **2930** evaluates to **false** AND

The function **apr\_bucket\_eos\_create**, called on line **2938**, returns NULL.

**CODE FRAGMENT**

```

2857 static int handle_map_file(request_rec *r)
2858 {
2859     .
2860     .
2861     .
2876     if (best->body)
2877     {
2878         conn_rec *c = r->connection;
2879         apr_bucket_brigade *bb;
2880         apr_bucket *e;
2881
2882         ap_allow_standard_methods(r, REPLACE_ALLOW, M_GET, M_OPTIONS, M_POST,
-1);
2883         /*if (r->method_number == M_OPTIONS) {
2884             *   return ap_send_http_options(r);
2885             * }
2886             */
2887         if (r->method_number != M_GET && r->method_number != M_POST) {
2888             return HTTP_METHOD_NOT_ALLOWED;
2889         }
2890
2891         /* ### These may be implemented by adding some 'extra' info
2892            *   of the file offset onto the etag
2893            *   ap_update_mtime(r, r->finfo.mtime);
2894            *   ap_set_last_modified(r);
2895            *   ap_set_etag(r);
2896            */
2897         apr_table_setn(r->headers_out, "Accept-Ranges", "bytes");
2898         ap_set_content_length(r, best->bytes);
2899
2900         /* set MIME type and charset as negotiated */
2901         if (best->mime_type && *best->mime_type) {
2902             if (best->content_charset && *best->content_charset) {
2903                 ap_set_content_type(r, apr_pstrcat(r->pool,
2904                                                     best->mime_type,
2905                                                     "; charset=",
2906                                                     best->content_charset,
2907                                                     NULL));
2908             }
2909             else {
2910                 ap_set_content_type(r, apr_pstrdup(r->pool, best->mime_type));
2911             }
2912         }
2913     }

```

```

2914     /* set Content-language(s) as negotiated */
2915     if (best->content_languages && best->content_languages->nelts) {
2916         r->content_languages = apr_array_copy(r->pool,
2917                                             best->content_languages);
2918     }
2919
2920     /* set Content-Encoding as negotiated */
2921     if (best->content_encoding && *best->content_encoding) {
2922         r->content_encoding = apr_pstrdup(r->pool,
2923                                         best->content_encoding);
2924     }
2925
2926     if ((res = ap_meets_conditions(r)) != OK) {
2927         return res;
2928     }
2929
2930     if ((res = ap_discard_request_body(r)) != OK) {
2931         return res;
2932     }
2933     bb = apr_brigade_create(r->pool, c->bucket_alloc);
2934     e = apr_bucket_file_create(map, best->body,
2935                               (apr_size_t)best->bytes, r->pool,
2936                               c->bucket_alloc);
2937     APR_BRIGADE_INSERT_TAIL(bb, e);
2938     e = apr_bucket_eos_create(c->bucket_alloc);
2939     APR_BRIGADE_INSERT_TAIL(bb, e);
2940
2941     return ap_pass_brigade(r->output_filters, bb);
2942 }
2943
2944 if (r->path_info && *r->path_info) {
2945     /* remove any path_info from the end of the uri before trying
2946      * to change the filename.  r->path_info from the original
2947      * request is passed along on the redirect.

```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b>	12
<b>LOCATION:</b>	httpd-2.1/os/unix/unixd.c : 382		
<b>DESCRIPTION</b>	The local variable <b>args</b> , passed in as an argument on line <b>320</b> , may be NULL where it is dereferenced on line <b>382</b> . A similar error can be found on line 383.		
<b>PRECONDITIONS</b>	The conditional expression <b>(!unixd_config.suexec_enabled)</b> on line <b>331</b> evaluates to <b>false</b> AND		
	The conditional expression <b>(!execuser    !execgroup)</b> on line <b>353</b> evaluates to <b>false</b> AND		
	The conditional expression <b>(args)</b> on line <b>358</b> evaluates to <b>false</b> AND		
	The conditional expression <b>(apr_procattr_cmdtype_set(attr, APR_PROGRAM) != APR_SUCCESS)</b> on line <b>376</b> evaluates to <b>false</b> .		

**CODE FRAGMENT**

```

318 static apr_status_t ap_unix_create_privileged_process(
319     apr_proc_t *newproc, const char *progrname,
320     const char * const *args,
321     const char * const *env,
322     apr_procattr_t *attr, ap_unix_identity_t *ugid,
323     apr_pool_t *p)
324 {
325     int i = 0;
326     const char **newargs;
327     char *newprogrname;
328     char *execuser, *execgroup;
329     const char *argv0;
330
331     if (!unixd_config.suexec_enabled) {
332         return apr_proc_create(newproc, progrname, args, env, attr, p);
333     }
334
335     argv0 = ap_strrchr_c(progrname, '/');
336     /* Allow suexec's "/" check to succeed */
337     if (argv0 != NULL) {
338         argv0++;
339     }
340     else {
341         argv0 = progrname;
342     }
343
344
345     if (ugid->userdir) {
346         execuser = apr_psprintf(p, "%ld", (long) ugid->uid);
347     }
348     else {
349         execuser = apr_psprintf(p, "%ld", (long) ugid->uid);
350     }
351     execgroup = apr_psprintf(p, "%ld", (long) ugid->gid);
352
353     if (!execuser || !execgroup) {
354         return APR_ENOMEM;
355     }
356
357     i = 0;
358     if (args) {
359         while (args[i]) {
360             i++;
361         }
362     }

```

```

363     /* allocate space for 4 new args, the input args, and a null terminator */
364     newargs = apr_palloc(p, sizeof(char *) * (i + 4));
365     newprogname = SUEXEC_BIN;
366     newargs[0] = SUEXEC_BIN;
367     newargs[1] = execuser;
368     newargs[2] = execgroup;
369     newargs[3] = apr_pstrdup(p, argv0);
370
371     /*
372     ** using a shell to execute suexec makes no sense thus
373     ** we force everything to be APR_PROGRAM, and never
374     ** APR_SHELLCMD
375     */
376     if(apr_procattr_cmdtype_set(attr, APR_PROGRAM) != APR_SUCCESS) {
377         return APR_EGENERAL;
378     }
379
380     i = 1;
381     do {
382         newargs[i + 3] = args[i];
383     } while (args[i++]);
384
385     return apr_proc_create(newproc, newprogname, newargs, env, attr, p);
386 }

```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b>	13
<b>LOCATION:</b>	httpd-2.1/server/config.c : 1048		
<b>DESCRIPTION</b>	The local variable <b>current</b> , passed in as an argument on line <b>1018</b> , may be NULL where it is dereferenced on line <b>1048</b> .		
<b>PRECONDITIONS</b>	<p>The conditional expression <b>(!memcmp(l, "&lt;/", 2) &amp;&amp; (strcasecmp(l + 2, bracket) == 0) &amp;&amp; (*curr_parent == NULL))</b> on line <b>1029</b> evaluates to <b>false</b> AND</p> <p>The conditional expression <b>(retval != NULL)</b> on line <b>1036</b> evaluates to <b>false</b> AND</p> <p>The conditional expression <b>curr_parent != NULL</b> on line <b>1039</b> evaluates to <b>false</b> AND</p> <p>The conditional expression <b>sub_tree == NULL</b> on line <b>1043</b> evaluates to <b>true</b> AND</p> <p>The conditional expression <b>current != NULL</b> on line <b>1043</b> evaluates to <b>false</b>.</p>		

**CODE FRAGMENT**

```

1015 AP_DECLARE(const char *) ap_build_cont_config(apr_pool_t *p,
1016                                               apr_pool_t *temp_pool,
1017                                               cmd_parms *parms,
1018                                               ap_directive_t **current,
1019                                               ap_directive_t **curr_parent,
1020                                               char *orig_directive)
1021 {
1022     char l[MAX_STRING_LEN];
1023     char *bracket;
1024     const char *retval;
1025     ap_directive_t *sub_tree = NULL;
1026
1027     bracket = apr_pstrcat(p, orig_directive + 1, ">", NULL);
1028     while (!(ap_cfg_getline(l, MAX_STRING_LEN, parms->config_file))) {
1029         if (!memcmp(l, "</", 2)
1030             && (strcasecmp(l + 2, bracket) == 0)
1031             && (*curr_parent == NULL)) {
1032             break;
1033         }
1034         retval = ap_build_config_sub(p, temp_pool, l, parms, current,
1035                                     curr_parent, &sub_tree);
1036         if (retval != NULL)
1037             return retval;
1038
1039         if (sub_tree == NULL && curr_parent != NULL) {
1040             sub_tree = *curr_parent;
1041         }
1042
1043         if (sub_tree == NULL && current != NULL) {
1044             sub_tree = *current;
1045         }
1046     }
1047
1048     *current = sub_tree;
1049     return NULL;
1050 }

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 14**LOCATION:** httpd-2.1/server/config.c : 1480**DESCRIPTION** The pointer expression **pattern** used in the **index** expression **pattern++ = '\0'** (1) times on line **1480** may be NULL. Although this is not an immediate dereference of a NULL pointer, the expression **pattern++ = '\0'** is not a valid pointer.**PRECONDITIONS** The conditional expression **((ap\_server\_pre\_read\_config->nelts || ap\_server\_post\_read\_config->nelts) && !(strcmp(fname, ap\_server\_root\_relative(p, SERVER\_CONFIG\_FILE))))** on line **1458** evaluates to **false** ANDThe conditional expression **(ispatt || ap\_is\_rdirectory(p, fname))** on line **1466** evaluates to **true** ANDThe conditional expression **(ispatt)** on line **1475** evaluates to **true** ANDThe function **ap\_strchr**, called on line **1476**, returns NULL.**CODE FRAGMENT**

```

1443 AP_DECLARE(void) ap_process_resource_config(server_rec *s, const char *fname,
1444                                             ap_directive_t **conftree,
1445                                             apr_pool_t *p,
1446                                             apr_pool_t *ptemp)
1447 {
1448     cmd_parms parms;
1449     apr_info_t finfo;
1450     const char *errmsg;
1451     ap_configfile_t *cfp;
1452     int ispatt;
1453
1454     /* XXX: lstat() won't work on the wildcard pattern...
1455      */
1456
1457     /* don't require conf/httpd.conf if we have a -C or -c switch */
1458     if ((ap_server_pre_read_config->nelts
1459         || ap_server_post_read_config->nelts)
1460         && !(strcmp(fname, ap_server_root_relative(p, SERVER_CONFIG_FILE)))) {
1461         if (apr_lstat(&finfo, fname, APR_FINFO_TYPE, p) != APR_SUCCESS)
1462             return;
1463     }
1464
1465     ispatt = apr_fnmatch_test(fname);
1466     if (ispatt || ap_is_rdirectory(p, fname)) {
1467         apr_dir_t *dirp;
1468         apr_info_t dirent;
1469         int current;
1470         apr_array_header_t *candidates = NULL;
1471         fnames *fnew;
1472         apr_status_t rv;
1473         char errmsg[120], *path = apr_pstrdup(p, fname), *pattern = NULL;
1474
1475         if (ispatt) {
1476             pattern = ap_strchr(path, '/');
1477
1478             AP_DEBUG_ASSERT(pattern != NULL); /* path must be absolute. */
1479
1480             *pattern++ = '\0';
1481
1482             if (apr_fnmatch_test(path)) {

```

```
1483         fprintf(stderr, "%s: wildcard patterns not allowed in Include
1484         "
1485         "%s\n", ap_server_argv0, fname);
1486     }
1487     exit(1);
1488     if (!ap_is_rdirectory(p, path)){
1489         fprintf(stderr, "%s: Include directory '%s' not found",
1490         ap_server_argv0, path);
```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b> 15
<b>LOCATION:</b>	httpd-2.1/server/core.c : 3377	
<b>DESCRIPTION</b>	The local pointer variable <code>e</code> , declared on line <b>3241</b> , and assigned on line <b>3376</b> , may be NULL where it is dereferenced on line <b>3377</b> . This NULL pointer dereference only happens in an Out Of Memory context. Similar errors can be found on lines 3579, 3612 and 3635.	
<b>PRECONDITIONS</b>	<p>The conditional expression <code>((r-&gt;used_path_info != AP_REQ_ACCEPT_PATH_INFO) &amp;&amp; r-&gt;path_info &amp;&amp; *r-&gt;path_info)</code> on line <b>3287</b> evaluates to <b>false</b> AND</p> <p>The conditional expression <code>(r-&gt;method_number != M_GET)</code> on line <b>3305</b> evaluates to <b>false</b> AND</p> <p>The conditional expression <code>((status = apr_file_open(&amp;fd, r-&gt;filename, APR_READ   APR_BINARY #if APR_HAS_SENDFILE  ((d-&gt;enable_sendfile == ENABLE_SENDFILE_OFF) ? 0 : APR_SENDFILE_ENABLED) #endif, 0, r-&gt;pool)) != APR_SUCCESS)</code> on line <b>3319</b> evaluates to <b>false</b> AND</p> <p>The conditional expression <code>((errstatus = ap_meets_conditions(r)) != OK)</code> on line <b>3335</b> evaluates to <b>false</b> AND</p> <p>The function <code>apr_bucket_eos_create</code>, called on line <b>3376</b>, returns NULL.</p>	

**CODE FRAGMENT**

```

3237 static int default_handler(request_rec *r)
3238 {
3239     conn_rec *c = r->connection;
3240     apr_bucket_brigade *bb;
3241     apr_bucket *e;
...
3287     if ((r->used_path_info != AP_REQ_ACCEPT_PATH_INFO) &&
3288         r->path_info && *r->path_info)
3289     {
3290         /* default to reject */
3291         ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r,
3292                     "File does not exist: %s",
3293                     apr_pstrcat(r->pool, r->filename, r->path_info,
NULL));
3294         return HTTP_NOT_FOUND;
3295     }
...
3305     if (r->method_number != M_GET) {
...
3314         return HTTP_METHOD_NOT_ALLOWED;
3315     }
3316 }
...
3319     if ((status = apr_file_open(&fd, r->filename, APR_READ | APR_BINARY
3320 #if APR_HAS_SENDFILE
3321         | ((d->enable_sendfile == ENABLE_SENDFILE_OFF)
3322           ? 0 : APR_SENDFILE_ENABLED)
3323 #endif
, 0, r->pool)) != APR_SUCCESS) {
3324         ap_log_rerror(APLOG_MARK, APLOG_ERR, status, r,
3325                     "file permissions deny server access: %s", r-
>filename);
3326     }
3327     return HTTP_FORBIDDEN;
3328 }
...

```

```
3335     if ((errstatus = ap_meets_conditions(r)) != OK) {
3336         apr_file_close(fd);
3337         return errstatus;
3338     }
3339     ...
3376     e = apr_bucket_eos_create(c->bucket_alloc);
3377     APR_BRIGADE_INSERT_TAIL(bb, e);
3378
3379     return ap_pass_brigade(r->output_filters, bb);
3380 }
3381 else {
3382     /* unusual method (not GET or POST) */
3383     if (r->method_number == M_INVALID) {
3384         ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r,
3385             "Invalid method in request %s", r->the_request);
3386         return HTTP_NOT_IMPLEMENTED;
3387     }
3388 }
```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b> 16
<b>LOCATION:</b>	httpd-2.1/server/protocol.c : 1258	
<b>DESCRIPTION</b>	The local pointer variable <b>flush</b> , declared on line <b>1256</b> , and assigned on line <b>1256</b> , may be NULL where it is dereferenced on line <b>1258</b> . This NULL pointer dereference only happens in an Out Of Memory context. Similar errors can be found on lines 1270 and 1291.	
<b>PRECONDITIONS</b>	<p>The conditional expression (<b>e-&gt;length == (apr_size_t)-1</b>) on line <b>1236</b> evaluates to <b>true</b> AND</p> <p>The conditional expression (<b>rv == APR_SUCCESS</b>) on line <b>1245</b> evaluates to <b>false</b> AND</p> <p>The conditional expression (<b>APR_STATUS_IS_EAGAIN(rv)</b>) on line <b>1250</b> evaluates to <b>true</b> AND</p> <p>The conditional expression (<b>e != APR_BRIGADE_FIRST(b)</b>) on line <b>1254</b> evaluates to <b>true</b> AND</p> <p>The function <b>apr_bucket_flush_create</b>, called on line <b>1256</b>, returns NULL.</p>	

**CODE FRAGMENT**

```

1212 AP_CORE_DECLARE_NONSTD(apr_status_t) ap_content_length_filter(
1213     ap_filter_t *f,
1214     apr_bucket_brigade *b)
1215 {
1216     if (e->length == (apr_size_t)-1) {
1217         apr_size_t len;
1218         const char *ignored;
1219         apr_status_t rv;
1220
1221         /* This is probably a pipe bucket.  Send everything
1222          * prior to this, and then read the data for this bucket.
1223          */
1224         rv = apr_bucket_read(e, &ignored, &len, eblock);
1225         if (rv == APR_SUCCESS) {
1226             /* Attempt a nonblocking read next time through */
1227             eblock = APR_NONBLOCK_READ;
1228             r->bytes_sent += len;
1229         }
1230         else if (APR_STATUS_IS_EAGAIN(rv)) {
1231             /* Output everything prior to this bucket, and then
1232              * do a blocking read on the next batch.
1233              */
1234             if (e != APR_BRIGADE_FIRST(b)) {
1235                 apr_bucket_brigade *split = apr_brigade_split(b, e);
1236                 apr_bucket *flush = apr_bucket_flush_create(r->connection-
1237 >bucket_alloc);
1238
1239                 APR_BRIGADE_INSERT_TAIL(b, flush);
1240                 rv = ap_pass_brigade(f->next, b);
1241                 if (rv != APR_SUCCESS || f->c->aborted) {
1242                     apr_brigade_destroy(split);
1243                     return rv;
1244                 }
1245                 b = split;
1246                 e = APR_BRIGADE_FIRST(b);
1247
1248                 ctx->data_sent = 1;
1249             }
1250         }
1251     }
1252 }

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 17**LOCATION:** httpd-2.1/server/protocol.c : 1313

**DESCRIPTION** The local pointer variable **b**, declared on line **1308**, and assigned on line **1312**, may be NULL where it is dereferenced on line **1313**. This NULL pointer dereference only happens in an Out Of Memory context. Similar errors can be found on lines 1401 and 1571.

**PRECONDITIONS** The function **apr\_bucket\_file\_create**, called on line **1312**, returns NULL.

**CODE FRAGMENT**

```

1302 AP_DECLARE(apr_status_t) ap_send_fd(apr_file_t *fd, request_rec *r,
1303                                     apr_off_t offset, apr_size_t len,
1304                                     apr_size_t *nbytes)
1305 {
1306     conn_rec *c = r->connection;
1307     apr_bucket_brigade *bb = NULL;
1308     apr_bucket *b;
1309     apr_status_t rv;
1310
1311     bb = apr_brigade_create(r->pool, c->bucket_alloc);
1312     b = apr_bucket_file_create(fd, offset, len, r->pool, c->bucket_alloc);
1313     APR_BRIGADE_INSERT_TAIL(bb, b);
1314
1315     rv = ap_pass_brigade(r->output_filters, bb);
1316     if (rv != APR_SUCCESS) {
1317         *nbytes = 0; /* no way to tell how many were actually sent */
1318     }
1319     else {
1320         *nbytes = len;
1321     }
1322
1323     return rv;

```

**DEFECT CLASS:** Null Pointer Dereference

**DEFECT ID** 18

**LOCATION:** httpd-2.1/server/util\_filter.c : 585

**DESCRIPTION** The local pointer variable **b**, declared on line **582**, and assigned on line **584**, may be NULL where it is dereferenced on line **585**. This NULL pointer dereference only happens in an Out Of Memory context.

**PRECONDITIONS** The function **apr\_bucket\_flush\_create**, called on line **584**, returns NULL.

**CODE FRAGMENT**

```
580 AP_DECLARE(apr_status_t) ap_fflush(ap_filter_t *f, apr_bucket_brigade *bb)
581 {
582     apr_bucket *b;
583
584     b = apr_bucket_flush_create(f->c->bucket_alloc);
585     APR_BRIGADE_INSERT_TAIL(bb, b);
586     return ap_pass_brigade(f, bb);
587 }
```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b> 19
<b>LOCATION:</b>	httpd-2.1/server/util_script.c : 695	
<b>DESCRIPTION</b>	The pointer expression <b>p</b> used in the <b>assignment</b> expression <b>p - strs-&gt;curpos</b> on line <b>695</b> may be NULL. Although this is not an immediate dereference of a NULL pointer, the expression <b>p - strs-&gt;curpos</b> may not be a valid length.	
<b>PRECONDITIONS</b>	The conditional expression <b>(!strs-&gt;curpos    !*strs-&gt;curpos)</b> on line <b>688</b> evaluates to <b>false</b> AND The conditional expression <b>(p)</b> on line <b>691</b> evaluates to <b>false</b> AND The function <b>ap_strchr_c</b> , called on line <b>694</b> , returns NULL.	

**CODE FRAGMENT**

```

682 static int getsfunc_STRING(char *w, int len, void *pvastrs)
683 {
684     struct vastrs *strs = (struct vastrs*) pvastrs;
685     const char *p;
686     int t;
687
688     if (!strs->curpos || !*strs->curpos)
689         return 0;
690     p = ap_strchr_c(strs->curpos, '\n');
691     if (p)
692         ++p;
693     else
694         p = ap_strchr_c(strs->curpos, '\0');
695     t = p - strs->curpos;
696     if (t > len)
697         t = len;
698     strncpy (w, strs->curpos, t);
699     w[t] = '\0';
700     if (!strs->curpos[t]) {
701         ++strs->arg;
702         strs->curpos = va_arg(strs->args, const char *);
703     }
704     else
705         strs->curpos += t;

```

**DEFECT CLASS:** Null Pointer Dereference **DEFECT ID** 20

**LOCATION:** httpd-2.1/src/lib/apr/memory/unix/apr\_pools.c : 826

**DESCRIPTION** The local variable **parent**, passed in as an argument on line **810**, may be NULL where it is dereferenced on line **826**.

**PRECONDITIONS** The conditional expression **(!parent)** on line **819** evaluates to **true** AND  
 The conditional expression **(!abort\_fn && parent)** on line **822** evaluates to **false** AND  
 The conditional expression **(allocator == NULL)** on line **825** evaluates to **true**.

**CODE FRAGMENT**

```

809 APR_DECLARE(apr_status_t) apr_pool_create_ex(apr_pool_t **newpool,
810 apr_pool_t *parent,
811 apr_abortfunc_t abort_fn,
812 apr_allocator_t *allocator)
813 {
814     apr_pool_t *pool;
815     apr_memnode_t *node;
816
817     *newpool = NULL;
818
819     if (!parent)
820         parent = global_pool;
821
822     if (!abort_fn && parent)
823         abort_fn = parent->abort_fn;
824
825     if (allocator == NULL)
826         allocator = parent->allocator;
827
828     if ((node = allocator_alloc(allocator,
829                               MIN_ALLOC - APR_MEMNODE_T_SIZE)) == NULL) {
830         if (abort_fn)
831             abort_fn(APR_ENOMEM);
832
833         return APR_ENOMEM;
834     }
835
836     node->next = node;

```

**DEFECT CLASS:** Null Pointer Dereference

**DEFECT ID** 21

**LOCATION:** httpd-2.1/src/lib/apr/misc/unix/otherchild.c : 137

**DESCRIPTION** The local pointer variable **cur**, declared on line **126**, and assigned on line **128**, may be NULL where it is dereferenced on line **137**.

**PRECONDITIONS** The conditional expression (**cur**) on line **129** evaluates to **false**.

**CODE FRAGMENT**

```
124 APR_DECLARE(void) apr_proc_other_child_unregister(void *data)
125 {
126     apr_other_child_rec_t *cur;
127
128     cur = other_children;
129     while (cur) {
130         if (cur->data == data) {
131             break;
132         }
133         cur = cur->next;
134     }
135
136     /* segfault if this function called with invalid parm */
137     apr_pool_cleanup_kill(cur->p, cur->data, other_child_cleanup);
138     other_child_cleanup(data);
139 }
```

<b>DEFECT CLASS:</b>	Null Pointer Dereference	<b>DEFECT ID</b> 22
<b>LOCATION:</b>	httpd-2.1/src/lib/apr/tables/apr_hash.c : 434	
<b>DESCRIPTION</b>	The local pointer variable <b>new_vals</b> , declared on line <b>396</b> , and assigned on line <b>427</b> , may be NULL where it is dereferenced on line <b>434</b> . This NULL pointer dereference only happens in an Out Of Memory context. Similar errors can be found on lines 435, 436, 437, 438, 439, 461, 462, 463, 464, 465 and 466.	
<b>PRECONDITIONS</b>	The conditional expression ( <b>base-&gt;count + overlay-&gt;count</b> ) on line <b>426</b> evaluates to <b>true</b> AND The function <b>apr_palloc</b> , called on line <b>427</b> , returns NULL AND The for loop on line <b>431</b> is executed with <b>k &lt;= base-&gt;max</b> evaluates to <b>true</b> AND The for loop on line <b>432</b> is executed with <b>iter</b> evaluates to <b>true</b> .	

**CODE FRAGMENT**

```

384 APR_DECLARE(apr_hash_t *) apr_hash_merge(apr_pool_t *p,
385                                           const apr_hash_t *overlay,
386                                           const apr_hash_t *base,
387                                           void * (*merger)(apr_pool_t *p,
388                                                         const void *key,
389                                                         apr_ssize_t klen,
390                                                         const void *h1_val,
391                                                         const void *h2_val,
392                                                         const void *data),
393                                           const void *data)
394 {
395     apr_hash_t *res;
396     apr_hash_entry_t *new_vals = NULL;
397     apr_hash_entry_t *iter;
398     apr_hash_entry_t *ent;
399     unsigned int i,j,k;
400
401     #ifdef POOL_DEBUG
402     /* we don't copy keys and values, so it's necessary that
403     * overlay->a.pool and base->a.pool have a life span at least
404     * as long as p
405     */
406     if (!apr_pool_is_ancestor(overlay->pool, p)) {
407         fprintf(stderr,
408                "apr_hash_overlay: overlay's pool is not an ancestor of p\n");
409         abort();
410     }
411     if (!apr_pool_is_ancestor(base->pool, p)) {
412         fprintf(stderr,
413                "apr_hash_overlay: base's pool is not an ancestor of p\n");
414         abort();
415     }
416     #endif
417
418     res = apr_palloc(p, sizeof(apr_hash_t));
419     res->pool = p;
420     res->count = base->count;
421     res->max = (overlay->max > base->max) ? overlay->max : base->max;
422     if (base->count + overlay->count > res->max) {
423         res->max = res->max * 2 + 1;
424     }
425     res->array = alloc_array(res, res->max);
426     if (base->count + overlay->count) {
427         new_vals = apr_palloc(p, sizeof(apr_hash_entry_t) *
428                             (base->count + overlay->count));

```

```
429     }
430     j = 0;
431     for (k = 0; k <= base->max; k++) {
432         for (iter = base->array[k]; iter; iter = iter->next) {
433             i = iter->hash & res->max;
434             new_vals[j].klen = iter->klen;
435             new_vals[j].key = iter->key;
436             new_vals[j].val = iter->val;
437             new_vals[j].hash = iter->hash;
438             new_vals[j].next = res->array[i];
439             res->array[i] = &new_vals[j];
440             j++;
441         }
442     }
443     for (k = 0; k <= overlay->max; k++) {
```

**DEFECT CLASS:** Null Pointer Dereference

**DEFECT ID** 23

**LOCATION:** httpd-2.1/src/lib/apr-util/buckets/apr\_brigade.c : 109

**DESCRIPTION** The local pointer variable **b**, declared on line **103**, and assigned on line **105**, may be NULL where it is dereferenced on line **109**. This NULL pointer dereference only happens in an Out Of Memory context.

**PRECONDITIONS** The function **apr\_palloc**, called on line **105**, returns NULL.

**CODE FRAGMENT**

```
100  APU_DECLARE(apr_bucket_brigade *) apr_brigade_create(apr_pool_t *p,  
101                                                         apr_bucket_alloc_t *list)  
102  {  
103      apr_bucket_brigade *b;  
104  
105      b = apr_palloc(p, sizeof(*b));  
106      b->p = p;  
107      b->bucket_alloc = list;  
108  
109      APR_RING_INIT(&b->list, apr_bucket, link);  
110  
111      apr_pool_cleanup_register(b->p, b, brigade_cleanup, apr_pool_cleanup_null);  
112      return b;  
113  }
```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 24**LOCATION:** httpd-2.1/src/lib/apr-util/buckets/apr\_brigade.c : 424**DESCRIPTION** The local pointer variable **e**, declared on line **406**, and assigned on line **423**, may be NULL where it is dereferenced on line **424**. This NULL pointer dereference only happens in an Out Of Memory context. Similar errors can be found on lines 429, 439, 482, 491 and 570.**PRECONDITIONS** The conditional expression (**nbyte > remaining**) on line **418** evaluates to **true** AND  
The conditional expression (**flush**) on line **422** evaluates to **true** AND  
The function **apr\_bucket\_transient\_create**, called on line **423**, returns NULL.**CODE FRAGMENT**

```

401  APU_DECLARE(apr_status_t) apr_brigade_write(apr_bucket_brigade *b,
402                                             apr_brigade_flush flush,
403                                             void *ctx,
404                                             const char *str, apr_size_t nbyte)
405  {
406      apr_bucket *e = APR_BRIGADE_LAST(b);
407      apr_size_t remaining = APR_BUCKET_BUFF_SIZE;
408      char *buf = NULL;
409
410      if (!APR_BRIGADE_EMPTY(b) && APR_BUCKET_IS_HEAP(e)) {
411          apr_bucket_heap *h = e->data;
412
413          /* HEAP bucket start offsets are always in-memory, safe to cast */
414          remaining = h->alloc_len - (e->length + (apr_size_t)e->start);
415          buf = h->base + e->start + e->length;
416      }
417
418      if (nbyte > remaining) {
419          /* either a buffer bucket exists but is full,
420           * or no buffer bucket exists and the data is too big
421           * to buffer. In either case, we should flush. */
422          if (flush) {
423              e = apr_bucket_transient_create(str, nbyte, b->bucket_alloc);
424              APR_BRIGADE_INSERT_TAIL(b, e);
425              return flush(b, ctx);
426          }
427          else {
428              e = apr_bucket_heap_create(str, nbyte, NULL, b->bucket_alloc);
429              APR_BRIGADE_INSERT_TAIL(b, e);
430              return APR_SUCCESS;
431          }
432      }
433      else if (!buf) {
434          /* we don't have a buffer, but the data is small enough

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 25**LOCATION:** httpd-2.1/src/lib/apr-util/dbm/sdbm/sdbm.c : 137**DESCRIPTION** The local pointer variable **db**, declared on line **131**, and assigned on line **136**, may be NULL where it is dereferenced on line **137**. This NULL pointer dereference only happens in an Out Of Memory context. Similar errors can be found on lines 139, 147, 157, 169, 173, 179, 189, 201, 203, 204, 205 and 206.**PRECONDITIONS** The function **malloc**, called on line **136**, returns NULL.**CODE FRAGMENT**

```

128 static apr_status_t prep(apr_sdbm_t **pdb, const char *dirname, const char
    *pagname,
129                          apr_int32_t flags, apr_fileperms_t perms, apr_pool_t
    *p)
130 {
131     apr_sdbm_t *db;
132     apr_status_t status;
133
134     *pdb = NULL;
135
136     db = malloc(sizeof(*db));
137     memset(db, 0, sizeof(*db));
138
139     db->pool = p;
140
141     /*
142     * adjust user flags so that WRONLY becomes RDWR,
143     * as required by this package. Also set our internal
144     * flag for RDONLY if needed.
145     */
146     if (!(flags & APR_WRITE)) {
147         db->flags |= SDBM_RDONLY;

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 26**LOCATION:** httpd-2.1/src/lib/apr-util/hooks/apr\_hooks.c : 195**DESCRIPTION** The local pointer variable **pTail**, declared on line **171**, and assigned on line **171**, may be NULL where it is dereferenced on line **195**.**PRECONDITIONS** The for loop on line **173** is executed with **nTotal < nItems** evaluates to **false**.**CODE FRAGMENT**

```

167 static TSort *tsort(TSort *pData,int nItems)
168 {
169     int nTotal;
170     TSort *pHead=NULL;
171     TSort *pTail=NULL;
172
173     for(nTotal=0 ; nTotal < nItems ; ++nTotal) {
174         int n,i,k;
175
176         for(n=0 ; ; ++n) {
177             if(n == nItems)
178                 assert(0); /* we have a loop... */
179             if(!pData[n].pNext && !pData[n].nPredecessors)
180                 break;
181         }
182         if(pTail)
183             pTail->pNext=&pData[n];
184         else
185             pHead=&pData[n];
186         pTail=&pData[n];
187         pTail->pNext=pTail; /* fudge it so it looks linked */
188         for(i=0 ; i < nItems ; ++i)
189             for(k=0 ; pData[i].ppPredecessors[k] ; ++k)
190                 if(pData[i].ppPredecessors[k] == &pData[n]) {
191                     --pData[i].nPredecessors;
192                     break;
193                 }
194     }
195     pTail->pNext=NULL; /* unfudge the tail */
196     return pHead;
197 }

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 27**LOCATION:** httpd-2.1/src/lib/apr-util/misc/apr\_reslist.c : 303**DESCRIPTION** The local pointer variable **rl**, declared on line **285**, and assigned on line **293**, may be NULL where it is dereferenced on line **303**. This NULL pointer dereference only happens in an Out Of Memory context. A similar error can be found on line 304.**PRECONDITIONS** The conditional expression **(min >= smax || min >= hmax || smax > hmax || ttl < 0)** on line **289** evaluates to **false**.**CODE FRAGMENT**

```

276  APU_DECLARE(apr_status_t) apr_reslist_create(apr_reslist_t **reslist,
277                                               int min, int smax, int hmax,
278                                               apr_interval_time_t ttl,
279                                               apr_reslist_constructor con,
280                                               apr_reslist_destructor de,
281                                               void *params,
282                                               apr_pool_t *pool)
283  {
284      apr_status_t rv;
285      apr_reslist_t *rl;
286
287      /* Do some sanity checks so we don't thrash around in the
288       * maintenance routine later. */
289      if (min >= smax || min >= hmax || smax > hmax || ttl < 0) {
290          return APR_EINVAL;
291      }
292
293      rl = apr_palloc(pool, sizeof(*rl));
294      rl->pool = pool;
295      rl->min = min;
296      rl->smax = smax;
297      rl->hmax = hmax;
298      rl->ttl = ttl;
299      rl->constructor = con;
300      rl->destructor = de;
301      rl->params = params;
302
303      APR_RING_INIT(&rl->avail_list, apr_res_t, link);
304      APR_RING_INIT(&rl->free_list, apr_res_t, link);
305
306      rv = apr_thread_mutex_create(&rl->listlock, APR_THREAD_MUTEX_DEFAULT,
307                                 pool);
308      if (rv != APR_SUCCESS) {
309          return rv;
310      }
311      rv = apr_thread_cond_create(&rl->avail, pool);
312      if (rv != APR_SUCCESS) {
313          return rv;

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 28**LOCATION:** httpd-2.1/support/ab.c : 1478**DESCRIPTION** The pointer expression **part** used in the **index** expression (**part + strlen("HTTP/1.x\_")**) on line **1478** may be NULL. Although this is not an immediate dereference of a NULL pointer, the expression (**part + strlen("HTTP/1.x\_")**) is not a valid pointer.**PRECONDITIONS** The conditional expression (**r == 0 && APR\_STATUS\_IS\_EOF(status)**) on line **1377** evaluates to **false** ANDThe conditional expression (**status != APR\_SUCCESS**) on line **1383** evaluates to **false** ANDThe conditional expression (**!c->goheader**) on line **1400** evaluates to **true** ANDThe conditional expression (**!s**) on line **1434** evaluates to **false** ANDThe function **strstr**, called on line **1477**, returns NULL.**CODE FRAGMENT**

```

1352 static void read_connection(struct connection * c)
1353 {
1354     apr_size_t r;
1355     apr_status_t status;
1356     char *part;
1357     char respcode[4];          /* 3 digits and null */
1358
1359     r = sizeof(buffer);
1360
1361     status = apr_recv(c->aprsock, buffer, &r);
1362     if (APR_STATUS_IS_EAGAIN(status))
1363         return;
1364     1377 else if (r == 0 && APR_STATUS_IS_EOF(status)) {
1365         good++;
1366         close_connection(c);
1367         return;
1368     }
1369     /* catch legitimate fatal apr_recv errors */
1370     1383 else if (status != APR_SUCCESS) {
1371         err_except++; /* XXX: is this the right error counter? */
1372         /* XXX: Should errors here be fatal, or should we allow a
1373          * certain number of them before completely failing? -aaron */
1374         apr_err("apr_recv", status);
1375     }
1376
1377     totalread += r;
1378     if (c->read == 0) {
1379         c->beginread = apr_time_now();
1380     }
1381     c->read += r;
1382
1383     1400 if (!c->goheader) {
1384         char *s;
1385         int l = 4;
1386         apr_size_t space = CBUFSIZE - c->cbx - 1; /* -1 allows for \0 term */
1387         int tocopy = (space < r) ? space : r;
1388     #ifdef NOT_ASCII
1389
1390     #else
1391         memcpy(c->cbuff + c->cbx, buffer, space);
1392     #endif
1393     }
1394     #endif NOT_ASCII

```

```
1434     if (!s) {
1452     }
1453     else {
1476         /* check response code */
1477         part = strstr(c->cbuff, "HTTP");          /* really HTTP/1.x_ */
1478         strncpy(respcode, (part + strlen("HTTP/1.x_")), 3);
1479         respcode[3] = '\0';
1480         if (respcode[0] != '2') {
1481             err_response++;
1482             if (verbosity >= 2)
1483                 printf("WARNING: Response code not 2xx (%s)\n", respcode);
1484         }
1485         else if (verbosity >= 3) {
1486             printf("LOG: Response code = %s\n", respcode);
1487         }
1488         c->gotheader = 1;

```

**DEFECT CLASS:** Null Pointer Dereference**DEFECT ID** 29**LOCATION:** httpd-2.1/support/ab.c : 1631

**DESCRIPTION** The local pointer variable **buff**, declared on line **1630**, and assigned on line **1630**, may be NULL where it is dereferenced on line **1631**. This NULL pointer dereference only happens in an Out Of Memory context. A similar error can be found on line 1632.

**PRECONDITIONS** The conditional expression (**posting == 1**) on line **1629** evaluates to **true** AND The function **malloc**, called on line **1630**, returns NULL.

**CODE FRAGMENT**

```

1554 static void test(void)
1555 {
...
1629     if (posting == 1) {
1630         char *buff = (char *) malloc(postlen + reqlen + 1);
1631         strcpy(buff, request);
1632         strcpy(buff + reqlen, postdata);
1633         request = buff;
1634     }
1635
1636 #ifdef NOT_ASCII
1637     inbytes_left = outbytes_left = reqlen;
1638     status = apr_xlate_conv_buffer(to_ascii, request, &inbytes_left,
1639                                   request, &outbytes_left);
1640     if (status || inbytes_left || outbytes_left) {
1641         fprintf(stderr, "only simple translation is supported (%d/%u/%u)\n",

```

**DEFECT CLASS:** Uninitialized Variable**DEFECT ID** 30**LOCATION:** httpd-2.1/support/htdbm.c : 358**DESCRIPTION** The local variable **cpw**, declared on line **327**, is used on line **358**, before **cpw** has been initialized.**PRECONDITIONS** The case default on line **355** is executed.**CODE FRAGMENT**

```

325 static apr_status_t htdbm_make(htdbm_t *htdbm)
326 {
327     char cpw[MAX_STRING_LEN];
328     char salt[9];
329
330     switch (htdbm->alg) {
331         case ALG_APSHA:
332             /* XXX cpw >= 28 + strlen(shal) chars - fixed len SHA */
333             apr_shal_base64(htdbm->userpass, strlen(htdbm->userpass), cpw);
334             break;
335
336         case ALG_APMD5:
337             (void) srand((int) time((time_t *) NULL));
338             to64(&salt[0], rand(), 8);
339             salt[8] = '\0';
340             apr_md5_encode((const char *)htdbm->userpass, (const char *)salt,
341                          cpw, sizeof(cpw));
342             break;
343         case ALG_PLAIN:
344             /* XXX this len limitation is not in sync with any HTTPd len. */
345             apr_cpystn(cpw, htdbm->userpass, sizeof(cpw));
346             break;
347         #if APR_HAVE_CRYPT_H
348             case ALG_CRYPT:
349                 (void) srand((int) time((time_t *) NULL));
350                 to64(&salt[0], rand(), 8);
351                 salt[8] = '\0';
352                 apr_cpystn(cpw, (char *)crypt(htdbm->userpass, salt), sizeof(cpw)
- 1);
353                 fprintf(stderr, "CRYPT is now deprecated, use MD5 instead!\n");
354             #endif
355         default:
356             break;
357     }
358     htdbm->userpass = apr_pstrdup(htdbm->pool, cpw);
359     return APR_SUCCESS;
360 }

```

**DEFECT CLASS:** Uninitialized Variable**DEFECT ID** 31**LOCATION:** httpd-2.1/support/htdigest.c : 131**DESCRIPTION** The local variable **ch**, declared on line **124**, is used on line **131**, before **ch** has been initialized.**PRECONDITIONS** The conditional expression **i < (n - 1)** on line **127** evaluates to **false**.**CODE FRAGMENT**

```

121 static int get_line(char *s, int n, apr_file_t *f)
122 {
123     register int i = 0;
124     char ch;
125     apr_status_t rv = APR_EINVAL;
126
127     while (i < (n - 1) &&
128           ((rv = apr_file_getc(&ch, f)) == APR_SUCCESS) && (ch != '\n')) {
129         s[i++] = ch;
130     }
131     if (ch == '\n')
132         s[i++] = ch;
133     s[i] = '\0';
134
135     if (rv != APR_SUCCESS)
136         return 1;
137
138     return 0;
139 }
```

## A. UNDERSTANDING DEFECT CLASS DESCRIPTIONS

---

This Appendix provides a detailed explanation of each of the Reasoning defect classes for C/C++. These examples assist the client in evaluating the impact of each of the defects listed in the Detailed Defect Report.

For each defect class, this document provides:

1. a short description of the class;
2. the likely impact of the defect;
3. advice on how the defect can be repaired;
4. an example code fragment that explains the defect in detail. Note that the example code fragments are *not* from the inspected application.

In general, *data corruption* is considered the worst impact. Data corruption can go unnoticed for weeks while damage continues to accumulate.

When a *program exception* occurs, there is better evidence that something went wrong. Even in this case, the failure sometimes goes unnoticed, for example if the program is a cgi-bin program that runs frequently for a short period of time. However, in most other situations, a program exception is a fatal error that leads to immediate program termination. In an embedded system or "daemon" process, such a failure could be catastrophic for the overall system.

When *unpredictable results* happen, they may eventually result in data corruption and/or program exceptions, but they may also go unnoticed. If they finally cause an observable error to occur, a large effort is usually required to track down exactly where the error occurs, because the reduction process to track the error down often makes the error fail to recur. Certain defects in one application may cause other programs on the same platform, or even the operating system, to fail, often after some time has passed. These defects are particularly difficult to track down.

Finally, over 70% of the effort spent on most C/C++ applications is spent in maintenance. Even when not directly preventing failures, the removal of defects may greatly reduce the cost of:

- extending the system with new functionality, such as web-enabling it;
- finding the root cause of a failure;
- training new staff and consultants; and
- rewriting or replacing the system.

In the descriptions of the defect classes, as much context as reasonably possible is provided. It is beyond the scope of this document, however, to explain everything that is needed to understand fully how to program in C/C++. The following publications may help to gain a deeper understanding:

- Kernighan & Ritchie, *The C Programming Language*, 2nd Edition, Bell Telephone Laboratories, Inc., 1988.
- Bjarne Stroustrup, *The C++ Programming Language*, 3rd Edition, AT&T, 1997.
- Steve McConnell, *Code Complete*, Microsoft Press, 1993.

- Andrew Koenig, *C Traps and Pitfalls*, AT&T Bell Telephone Labs, 1989.
- P.J. Plauger, *The Standard C library*, Prentice Hall, 1992.
- Scott Meyers, *Effective C++*, 2nd Edition, Addison Wesley, 1998.

# Memory Leak

## DESCRIPTION

*Memory leak* refers to the loss of available memory space that occurs when dynamic data (memory allocated on the heap by calling any of the standard C library routines `malloc()`, `calloc()`, `realloc()`, `strdup()` or the C++ operator `new`) is no longer used but never deallocated (by calling `free()`, `realloc()` or the C++ operator `delete`).

## IMPACT

Each time the leak occurs, the application drains the available memory pool. On some systems, memory may be allocated from a global pool, in which case the loss of available memory may affect the entire system, not just the application that caused it. Even on virtual-memory systems, where each process has its own protected address space, the gradual increase in application size can result in performance degradation that affects the entire system.

Depending on how long the application runs, how frequently the leak occurs, and the amount of available (including virtual) memory, memory leaks will sooner or later cause performance degradation of the application, and potentially of the entire system.

Eventually, the performance degradation may lead to a fatal out-of-memory condition. This condition may be encountered by an application unrelated to the one that caused the memory leak.

Understanding the application is important to rating this defect. For example, if this defect occurs in a daemon, the impact is almost always high, but if it occurs in a CGI script, the impact is usually negligible.

## REPAIR

A deallocation corresponding to each allocation is not always necessary, but it is a safe programming practice. For example, it is not necessary to deallocate dynamic data that is allocated in `main()`, since all dynamic data is freed after `main()` finishes, but a conservative programmer might include the deallocation anyway.

**EXAMPLE**

```
int check_msg() {
    msg_t *msg;
    int status = RET_FAIL;
    ...
    msg = (msg_t *)calloc(1, sizeof(*msg));
    if (!msg) {
        errno = OUTFOFMEMORY;
        return status;
    }
    err = Readmsg(msg);
    if (err == -1)
        return status;
    ...
}
```

In this example, memory space is allocated for a new `msg`. One possible execution path, however, is where the `Readmsg()` call fails; this leads to a return without the memory allocated for `msg` being freed. If this function is called frequently, and `Readmsg()` frequently fails, this could result in a large number of memory leaks. Eventually, this can lead to a fatal out-of-memory program exception.

## NULL Pointer Dereference

### DESCRIPTION

A `NULL` pointer is a pointer that refers to a specific, invalid memory address. A *dereference* means following a pointer to the memory location it refers to and accessing the data at that location. Thus, a *NULL pointer dereference* refers to an attempt to access data at this invalid address.

This defect class also reports situations where a `NULL` pointer is used in an assignment. Even though the assignment is not a defect by itself, a subsequent dereference will cause a program exception. The defect is reported at the assignment, because this is the earliest point in the code where the problem could be identified.

### IMPACT

On most general-purpose computing platforms (such as Windows or UNIX), a `NULL` pointer dereference usually causes a program exception. On systems without memory management hardware, such references may not be detected at all.

### REPAIR

To repair these defects, an if-statement checking for `NULL` values should be placed around the statements that dereference the pointers. Appropriate error-recovery should also be provided for the situations where the pointers are `NULL`.

### EXAMPLE

```
char* ptr = strrchr(tmp_eq, '-');
int chan_num = atoi(ptr+1);
```

The string function `strrchr()` returns a pointer to the last occurrence of '-' in the string `tmp_eq`, or `NULL` if the character is not present. The next line of code will cause a program exception if `ptr` is `NULL`, because `atoi()` does not accept a `NULL` pointer. A check should be performed before the call to `atoi()` to ensure that `ptr` is not `NULL`.

## Bad Deallocation

### DESCRIPTION

*Bad deallocation* refers to the use of an inappropriate memory release operation (standard C library routines `free()` or `realloc()`, or C++ operators `delete` or `delete[]`) for deallocating memory, or to the deallocation of memory that was never explicitly allocated.

### IMPACT

Depending on the compiler and the specific operation, the impact of this defect may range from no effect at all, to unexpected results, a memory leak, memory corruption, or a program exception.

Specifically:

- The use of `delete` on memory allocated with `new[]` may be a memory leak, because only the first element of the array is released, not the entire array;
- The use of `delete` on memory allocated with `malloc()`, `calloc()` or `realloc()`, or `strdup()` may cause memory corruption, or a program exception;
- The use of `free()` or `realloc()` on memory allocated with `new` may cause memory corruption, or a program exception;
- The use of `free()` or `realloc()` on memory that is not heap-allocated may cause memory corruption, or a program exception.

### REPAIR

Memory allocated with `malloc()`, `calloc()`, `realloc()`, or `strdup()`, should be deallocated only with `free()` or `realloc()`. Similarly, memory allocated with `new` should be deallocated with `delete`, and memory allocated with `new []` should be deallocated with `delete []`. Furthermore, `free()` and `delete` should be applied to the exact same pointer value that was returned by the corresponding allocating expression.

### EXAMPLE 1

```
DS3_NOC noct3msg;
memset((void *)&noct3msg, 0, sizeof(DS3_NOC));
...
delete (&noct3msg);
```

In this example, a local variable is initialized to all 0's using the `memset()` function. One of the rules in C++ is that the `delete` operator can only be applied to a pointer value that was previously obtained from the `new` operator. Use of the `delete` operator in the Example 1 will result in a program exception.

**EXAMPLE 2**

```
char* buffer = new char[100];
...
buffer++;
delete [] buffer;
```

In this example, a new character array called `buffer` is allocated, and later the base pointer to that array is incremented. When the memory is deallocated, `buffer` no longer points to the beginning of the allocated block. This will often lead to a program exception, or corruption of the heap.

**EXAMPLE 3**

```
char* p = new char[100];
...
delete p;
```

The problem in this example is the missing `[]` on the `delete` operator. Depending on the compiler, this can result in a program exception, or corruption of the heap.

**EXAMPLE 4**

```
void parse_message(char *msg) {
    char formatstring[100];
    ...
    free(formatstring);
    return;
}
```

The character array `formatstring` in this example is stack-allocated. It is inappropriate to call `free()` on stack-allocated memory. Depending on the memory manager, this can result in a program exception, or corruption of the heap.

## Out of Bounds Array Access

### DESCRIPTION

An *out-of-bounds array access* refers to a defect where an array index expression is not within the upper and lower bounds of the array.

### IMPACT

An out-of-bounds array access defect can cause data corruption or lead to a program exception.

### REPAIR

Array indexing needs to be guarded against out-of-bounds defects.

For situations where one array is copied into another, this type of defect can be repaired by adjusting array sizes.

For situations where index expressions are used to access arrays, an if-statement check on the index expression may suffice. In case the index expression is controlled by a loop construct, the terminating value/condition should be changed to be within the size of the array, and the index variable should be checked for manipulations within the loop that can cause an out-of-bounds access.

In case the index variable controlled by a loop is used outside the loop (which is in itself questionable programming practice), one should be aware that the value of the index variable may be outside the range specified by the loop construct.

The two most common programming mistakes are (1) using the wrong inequality test on the loop conditional (generally,  $\leq$  when it should have been  $<$ ), and (2) using the final value of the loop index variable *after* the loop to index the array, when often the loop is written such that the final value is beyond the end of the array. The following two examples demonstrate each of these problems.

### EXAMPLE 1

```
#define TABLE_SIZE 20
int PowerOf2[TABLE_SIZE];

void initTable() {
    int j;

    PowerOf2[0] = 1;
    for (j = 1; j <= TABLE_SIZE; j++)
        PowerOf2[j] = PowerOf2[j-1] * 2;
}
```

In this example, the problem is that the conditional expression in the for-loop terminates when  $j > \text{TABLE\_SIZE}$ . During the last iteration of the loop,  $j == \text{TABLE\_SIZE}$ , which means that the assignment indexes one beyond the end of the array.

**EXAMPLE 2**

```

#define MAX_PATH
void MungeFilePath(char dos_path[])
{
    int i;
    char pathName[MAX_PATH + 1];    /* room for trailing '\0' */

    /* Convert MS-DOS path characters to UNIX form, add a
     * trailing '/' if necessary to prepare for later
     * concatenation with the file name.
     */

    for (i = 0; i < MAX_PATH && dos_path[i] != '\0'; i++) {
        if (dos_path[i] == '\\')
            pathName[i] = '/';
        else
            pathName[i] = dos_path[i];
    }
    if (pathName[i - 1] != '/')
        pathName[i++] = '/';
    pathName[i] = '\0';
    ...
}

```

This code fragment actually has two types of array bounds violations, both after the loop. The first defect occurs when the original path is `MAX_PATH` characters long and the last character stored in `pathName` is not a `'/'`. In this case, the logic is to append a `'/'` character, followed by a NUL character. Even though the programmer made the array one larger to hold the trailing NUL character, that is insufficient when both a `'/'` and a NUL character must be appended.

The second bug occurs when the initial string is zero length (i.e., the first character in `dos_path` is a NUL character). In this case, the first loop does not execute at all and the if-statement tests `pathName[i - 1]`. However, in this case `i` is zero, and this expression accesses a memory location before the beginning of the array!

**EXAMPLE 3**

```

int color[50];
...
for (i=0; i<50; i+=3) {
    color[i]= colorSet(i, ...);
    color[i+1]= colorSet(i, ...);
    color[i+2]= colorSet(i, ...);
}

```

An array declared as `a[n]` has `n` elements, indexed from 0 to `n-1`. In this example, `i` is incremented by 3 each time through the for-loop. During the last iteration through the loop, its value is 48. When `i=48`, `color[i+2]` accesses the out-of-bounds element `color[50]`.

## Uninitialized Variable

### DESCRIPTION

*Uninitialized variable* refers to a defect where local and dynamic variables are not explicitly initialized prior to use. Note that the ANSI standard requires that *global* and *static* variables are initialized (*ints* to 0, *floats* to 0.0 and *pointers* to `NULL`); therefore this defect is not reported for variables with either of these storage classes.

### IMPACT

Usage of uninitialized variables can cause unpredictable results in the program (because the value of the variable is essentially random), and in the worst case, a program exception.

### REPAIR

To repair this type of defect, the variable must be initialized to an appropriate value.

### EXAMPLE

```
int fun() {
    int len;
    if ((to - from) > 86400) {
        now = localtime(&from);
        len = strftime(&OdateToStr, 64, "%m/%d/%Y - ", now);
        now = localtime(&to);
        len = strftime(&OdateToStr[len], 64, "%m/%d/%Y", now);
    } else {
        now = localtime(&from);
        len = strftime(&OdateToStr[len], 64, "%m/%d/%Y", now);
    }
}
```

In the example above, the stack-allocated variable `len` is uninitialized when used in the `else` clause of the conditional. The `then` clause of the conditional is fine, but if the `else` path is taken, then the `len` used in the array access is an uninitialized variable. This may result in a program exception if the random value in `len` causes an out-of-bounds array read.





P.O. Box 478

Menlo Park, CA 94026-0478

+1 650-324-2510

[www.reasoning.com](http://www.reasoning.com)