

# [UNIX] Apache Memory Corruption in Various Architectures

**Source:** <http://www.derkeiler.com/Mailing-Lists/Securiteam/2004-04/0102.html>

---

**From:** SecuriTeam ([support\\_at\\_securiteam.com](mailto:support_at_securiteam.com))

**Date:** 04/26/04

To: [list@securiteam.com](mailto:list@securiteam.com)

Date: 26 Apr 2004 12:54:22 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

-----

Apache Memory Corruption in Various Architectures

---

## SUMMARY

The <<http://www.apache.org/>> Apache Software Foundation's HTTP Server is "an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards".

The Apache HTTP server contains a bug in the way it handles SHA (Secure Hash) contexts. In various processor architectures the possibility exists that memory corruption can occur.

## DETAILS

Vulnerable Systems:

\* Apache HTTP server, all versions as of 1.3.29

The problem lies in the fact that while copying a SHA context, the program calculates the number of bytes to copy based on the size of the unsigned long integer type. In effect, on architectures where the size of an unsigned long is not 32-bit, possible memory corruption can occur. There

## Securiteam: [UNIX] Apache Memory Corruption in Various Architectures

are several places in the code that lead to the same vulnerable section of code:

```
"src/modules/standard/mod_auth.c"
and
"src/modules/standard/mod_auth3.c"
and
"src/modules/standard/mod_auth4.c"
static int authenticate_basic_user(request_rec *r)
{
..
..
    const char *sent_pw;
    char *real_pw;
..
..
    if ((res = ap_get_basic_auth_pw(r, &sent_pw)))
        return res;
..
..
    if (!(real_pw = get_pw(r, c->user, sec->auth_pwfile))) {
        ..
        ..
    }
..
..
    invalid_pw = ap_validate_password(sent_pw, real_pw);
..
..
}
```

While the request\_rec structure is declared in "src/include/httpd.h".

Taking a closer look at the ap\_validate\_password() in the

"src/ap/ap\_check.c" file:

```
API_EXPORT(char *) ap_validate_password(const char *passwd, const char
*hash)
```

```
{
    char sample[120];
..
..
    /* Netscape / SHA1 ldap style strng
    */
    else if (strncmp(hash, AP_SHA1PW_ID, AP_SHA1PW_IDLEN) == 0) {

        ap_sha1_base64(passwd, strlen(passwd), sample);
    }
..
..
}
```

While AP\_SHA1PW\_ID in "src/include/ap\_sha1.h" is defined as:

```
..
```

```
..
#define AP_SHA1PW_ID "{SHA}"
..
..
```

In order for the strcmp hash to be zero, the above should be the password for the ap\_get\_basic\_auth\_pw() function:

```
"src/main/http_pro.c"
API_EXPORT(int) ap_get_basic_auth_pw(request_rec *r, const char **pw)
{
..
..
}
```

The second argument, pw is evaluated inside ap\_validate\_password that is called from inside get\_pw():

```
"src/modules/standard/mod_auth.c"
static char *get_pw(request_rec *r, char *user, char *auth_pwfile)
{
..
..
}
```

Now, the function ap\_validate\_password calls ap\_sha1\_base64(). Taking a closer look shows:

```
"src/ap/ap_sha1.c"
API_EXPORT(void) ap_sha1_base64(const char *clear, int len, char *out)
{
..
..
    AP_SHA1_CTX context;
..
..
    ap_SHA1Init(&context);
    ap_SHA1Update(&context, clear, len);
..
..
}
```

AP\_SHA1\_CTX:

```
"src/ap/ap_sha1.c"
typedef struct {
    AP_LONG digest[5]; /* message digest */
    AP_LONG count_lo, count_hi; /* 64-bit bit count */
    AP_LONG data[16]; /* SHA data buffer */
    int local; /* unprocessed amount in data */
} AP_SHA1_CTX;
```

Taking note of the type of AP\_LONG and looking at ap\_SHA1Init():

```
"src/ap/ap_sha1.c"
```

## Securiteam: [UNIX] Apache Memory Corruption in Various Architectures

```
API_EXPORT(void) ap_SHA1Init(AP_SHA1_CTX *sha_info)
{
    sha_info->digest[0] = 0x67452301L;
    sha_info->digest[1] = 0xefcdab89L;
    sha_info->digest[2] = 0x98badcfeL;
    sha_info->digest[3] = 0x10325476L;
    sha_info->digest[4] = 0xc3d2e1f0L;
    sha_info->count_lo = 0L;
    sha_info->count_hi = 0L;
    sha_info->local = 0;
}

"src/ap/ap_sha1.c"
API_EXPORT(void) ap_SHA1Update(AP_SHA1_CTX *sha_info, const char *buf,
                               unsigned int count)
{
    ..
    ..
    const AP_BYTE *buffer = (const AP_BYTE *) buf;
    ..
    ..
    while (count >= SHA_BLOCKSIZE) {
        ebcDic2ascii((AP_BYTE *)sha_info->data, buffer, SHA_BLOCKSIZE);
        buffer += SHA_BLOCKSIZE;
        count -= SHA_BLOCKSIZE;
        maybe_byte_reverse(sha_info->data, SHA_BLOCKSIZE);
        sha_transform(sha_info);
    }
    ebcDic2ascii((AP_BYTE *)sha_info->data, buffer, count);
    ..
    ..
}
```

Notice the loop with the condition that count has to be greater or equal to the following constant:

```
"src/ap/ap_sha1.c"
..
..
#define SHA_BLOCKSIZE 64
..
..
```

And what happens in `ebcDic2ascii()`:

```
"src/ap/ap_ebcdi.c"
API_EXPORT(void *)
ebcDic2ascii(void *dest, const void *srce, size_t count)
{
    unsigned char *udest = dest;
    const unsigned char *usrce = srce;
```

## Securiteam: [UNIX] Apache Memory Corruption in Various Architectures

```
while (count-- != 0) {
    *udest++ = os_toascii[*usrce++];
}

return dest;
}
```

The above function copies 64 bytes, but the AP\_SHA1\_CTX structure is an array of 16 elements. Take a look at the structure's element declaration:

```
"src/include/ap_sha1.h"
typedef unsigned long AP_LONG; /* a 32-bit quantity */
```

Assuming that an unsigned long is indeed 32-bit (4 bytes), the function will indeed copy only 64 bytes of data into the context structure. However on architectures other than, 80x86, there is no guarantee that the size of unsigned long would be 32-bit. On 64-bit platforms it could very well be possible with some compiler options. When the sizeof(unsigned long) is not 4 bytes, memory corruption can occur in the ebcidc2ascii() function.

### ADDITIONAL INFORMATION

The information has been provided by <mailto:pi3ki31ny@wp.pl> Adam Zabrocki.

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====

=====

### DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.