

Advisory 01/2004: 12 x Gaim remote overflows

Source: <http://www.derkeiler.com/Mailing-Lists/Full-Disclosure/2004-01/1052.html>

From: Stefan Esser (*s.esser_at_e-matters.de*)

Date: 01/26/04

Date: Mon, 26 Jan 2004 09:44:42 +0100

To: full-disclosure@lists.netsys.com

e-matters GmbH
www.e-matters.de

-- Security Advisory --

Advisory: 12 x Gaim remote overflows

Release Date: 2004/01/26

Last Modified: 2004/01/26

Author: Stefan Esser [s.esser@e-matters.de]

Application: Gaim <= 0.75

Severity: 12 vulnerabilities were found in the instant messenger GAIM that allow remote compromise

Risk: Critical

Vendor Status: Vendor has fixed in CVS, but feels not ready for release because of problems with HEAD

Reference: <http://security.e-matters.de/advisories/012004.html>

Overview:

Gaim is a multi-protocol instant messaging client for Linux, BSD, MacOS X, and Windows. It is compatible with AIM (Oscar and TOC protocols), ICQ, MSN Messenger, Yahoo, IRC, Jabber, Gadu-Gadu, and Zephyr networks. It is a very popular choice among the users of instant messaging networks and especially in the community of migrated windows users.

While developing a custom add-on, an integer overflow in the handling of AIM DirectIM packets was revealed that could lead to a remote compromise of the IM client. After disclosing this bug to the vendor, they had to make a hurried release because of a change in the Yahoo connection procedure that rendered GAIM useless. Unfortunately at the same time a closer look onto the sourcecode revealed 11 more vulnerabilities.

The 12 identified problems range from simple standard stack

Full-Disclosure: Advisory 01/2004: 12 x Gaim remote overflows

overflows, over heap overflows to an integer overflow that can be abused to cause a heap overflow. Due to the nature of instant messaging many of these bugs require man-in-the-middle attacks between client and server. But the underlying protocols are easy to implement and MIM attacks on ordinary TCP sessions is a fairly simple task.

In combination with the latest kernel vulnerabilities or the habit of users to work as root/administrator these bugs can result in remote root compromises.

Details:

While auditing the Gaim source code the following 12 vulnerabilities were discovered:

Overflows in YMSG protocol (yahoo messenger) handler

- 01) Yahoo Octal-Encoding Decoder Overflow
- 02) Yahoo Octal-Encoding Decoder Out-Of-Bounds Overflow
- 03) Yahoo Web Cookie Parser Overflow
- 04) Yahoo Login Page Name Parser Overflow
- 05) Yahoo Login Page Value Parser Overflow
- 06) Yahoo Packet Parser Overflow

Overflows in oscar protocol (AIM) handler

- 07) AIM/Oscar DirectIM Integer Overflow

Overflows in utility functions
(called in various protocols)

- 08) Quoted Printable Decoder Overflow
- 09) Quoted Printable Decoder Out-Of-Bounds Overflow
- 10) URL Parser Function Overflow
- 11) Extract Info Field Function Overflow

Overflows that do not fit into
the other categories

- 12) HTTP Proxy Connect Overflow

Detailed Bug Description

Full-Disclosure: Advisory 01/2004: 12 x Gaim remote overflows

[01 – Yahoo Octal-Encoding Decoder Overflow]

[02 – Yahoo Octal-Encoding Decoder Out-Of-Bounds Overflow]

When the Yahoo Messenger handler decodes an octal value for email notification functions 2 different kind of overflows can be triggered

Affected version: 0.75 (only)

File: gaim/src/protocols/yahoo/yahoo.c

Function: yahoo_decode()

Code:

```
static char *yahoo_decode(const char *text)
{
    char *converted;
    char *p, *n, *new;

    n = new = g_malloc(strlen (text) + 1);

    for (p = (char *)text; *p; p++, n++) {
        if (*p == '\\') {
            sscanf(p + 1, "%3o\n", (int *)n); <----- [01]
            p += 3; <----- [02]
        }
        else
            *n = *p;
    }

    *n = '\0';
    ...
}
```

The way sscanf is used, it will always write 4 bytes to the allocated buffer. The author did not see the possibility of malformed input like "\1" (the backslash is only a backslash) it is possible to write 1-2 zero bytes over the buffer boundaries. On linux this is exploitable like any heap off by one into the malloc() chunks. The second vulnerability is that no matter how many bytes sscanf() consumes it always increases the pointer with assumed 4 bytes. This can result in overjumping the terminating zero byte and with special prepared memory after the string it is possible to overwrite the heap with an arbitrary amount of bytes.

These bugs are counted as 2 because for most people [02] is not obvious. As an example the vendor fixed only [01] first, because my description was not good enough. Additionally the misuse of sscanf() caused an incompatibility to all big endian platforms.

[03 – Yahoo Web Cookie Parser Overflow]

Full-Disclosure: Advisory 01/2004: 12 x Gaim remote overflows

When parsing the cookies within the HTTP reply header of a yahoo web connection a bufferoverflow can happen.

Affected version: <= 0.75

File: gaim/src/protocols/yahoo/yahoo.c

Function: yahoo_web_pending()

Code:

```
void yahoo_web_pending(gpointer data, gint source, ...
{
    GaimConnection *gc = data;
    GaimAccount *account = gaim_connection_get_account(gc);
    struct yahoo_data *yd = gc->proto_data;
    char buf[1024], buf2[256], *i = buf, *r = buf2;
    int len, o = 0;

    len = read(source, buf, sizeof(buf));
    ...
    while ((i = strstr(i, "Set-Cookie: ")) && 0 < 2) {
        i += strlen("Set-Cookie: ");
        for (;*i != ';'; r++, i++) {
            *r = *i;
        }
        *r=';';
        r++;
        ...
    }
    ...
}
```

Here all cookie data contained in the first 1024 byte of a HTTP reply header is copied into a 256 byte buffer without a size check. Because source and destination buffer are both on the stack and stack layout will most probably result in the smaller buffer overflowing into the smaller one this bug is believed to be not exploitable with the normal stack layout. Note also the typo in the while() condition $0 < 2$ which should be $o < 2$

[04 – Yahoo Login Page Name Parser Overflow]

[05 – Yahoo Login Page Value Parser Overflow]

When parsing the Yahoo Login Webpage the YMSG protocol overflows stachbuffers if the webpage returns oversized values.

Affected version: <= 0.75

File: gaim/src/protocols/yahoo/yahoo.c

Function: yahoo_login_page_hash()

Code:

```
static
```

Full-Disclosure: Advisory 01/2004: 12 x Gaim remote overflows

```
GHashTable *yahoo_login_page_hash(const char *buf,size_t len)
{
    GHashTable *hash = g_hash_table_new_full(g_str_hash, g_s...
    const char *c = buf;
    char *d;
    char name[64], value[64];
    while ((c < (buf + len)) && (c = strstr(c, "<input ") {
        c = strstr(c, "name=\"") + strlen("name=\"");
        for (d = name; *c!=""; c++, d++) <----- [04]
            *d = *c; <-----/
            *d = '\0';
            d = strstr(c, "value=\"") + strlen("value=\"");
            if (strchr(c, '>') < d)
                break;
            for (c = d, d = value; *c!=""; c++, d++) <--- [05]
                *d = *c; <-----/
                *d = '\0';
                g_hash_table_insert(hash, g_strdup(name), g_strdup(value));
            }
        return hash;
    }
```

The content of the yahoo login webpage is trusted although it could be changed with a simple man-in-the-middle attack on the HTTP session. name and value are filled directly from the page without any kind of size check. This results in two independent ways to overflow the stack.

[06 – Yahoo Packet Parser Overflow]

A Yahoo Messenger packet consist of a header and a list of keys with their associated values. When reading an oversized keyname a standard stackoverflow can be triggered. This is most probably the most dangerous discovered vulnerability because the nature of the bug makes it very easy to exploit and additionally a TCP man-in-the-middle attack is NOT needed. It is possible to send a malicious YMSG packet to the yahoo server so that it will be forwarded like any normal message.

Affected version: <= 0.75

File: gaim/src/protocols/yahoo/yahoo.c

Function: yahoo_packet_read()

Code:

```
static void yahoo_packet_read(struct yahoo_packet *pkt,
                             guchar *data, int len)
{
    int pos = 0;

    while (pos + 1 < len) {
```

Full-Disclosure: Advisory 01/2004: 12 x Gaim remote overflows

```
char key[64], *value = NULL, *esc;
int accept;
int x;

struct yahoo_pair *pair = g_new0(struct yahoo_pair, 1);

x = 0;
while (pos + 1 < len) {
    if (data[pos] == 0xc0 && data[pos + 1] == 0x80)
        break;
    key[x++] = data[pos++]; <----- [06]
}
key[x] = 0;
pos += 2;
...
```

Everytime the YMSG handler receives a complete packet it will give it this function to split it into its keys and values.

Because the keyname is copied without any kind of size check into the key variable which is a 64 byte stackbuffer it is very easy to exploit.

NOTE: This bug is also exploitable on systems with non executable stacks, or stack overflow detections as long free() exploits are possible on that platform,

[07 – AIM/Oscar DirectIM Integer Overflow]

Integer Overflow when allocating memory for a directIM packet results in heap overflow. directIM is a client 2 client protocol and therefore does not require a mim.

Affected version: <= 0.74

File: gaim/src/protocols/oscar/ft.c

Function: handlehdr_odc()

Code:

```
static int handlehdr_odc(aim_session_t *sess, aim_...
{
    aim_frame_t fr;
    int ret = 0;
    aim_rxcallback_t userfunc;
    fu32_t payloadlength;
    fu16_t flags, encoding;
    char *snptr = NULL;

    fr.conn = conn;

    /* AAA – ugly */
    aim_bstream_setpos(bs, 20);
    payloadlength = aimbs_get32(bs);
```

```

...

if (payloadlength) {
    char *msg;
    ...

    if (!(msg = calloc(1, payloadlength+1))) { <---- [07]
        free(snptr);
        return -ENOMEM;
    }

    while (payloadlength - recvd) {
        if (payloadlength - recvd >= 1024)
            i = aim_recv(conn->fd, &msg[recvd], 1024);
        else
            ...
    }
}

```

Within this code snippet payloadlength is taken directly from the network and passed to the calloc() function in an unsafe manner.

A user supplied payloadlength of UINT_MAX (0xffffffff) will cause an integer overflow within the second parameter of calloc() and therefore only allocate a 0 byte buffer. Please notice that this bug is not an integer overflow due to the multiplication within calloc() and therefore it is not caught by the recent security patches to calloc() on different platforms. Please also note that calloc(1, 0) will not return a NULL pointer but a pointer into the legal heap on at least all tested platforms (f.e. linux, bsd) On BSD systems this is configureable but it defaults to this behaviour. After allocating the 0 byte buffer aim_recv() is called repeatedly by the while loop to read and overwrite with up to 4GB of data.

[08 – Quoted Printable Decoder Overflow]

[09 – Quoted Printable Decoder Out-Of-Bounds Overflow]

When the MIME decoder decoded a quoted printable encoded string for email notification 2 different kind of overflows can be triggered.

Affected version: 0.75 (only)

File: gaim/src/util.c

Function: quotedp_decode()

Code:

```

void
gaim_quotedp_decode(const char *str, char **ret_str, int ...
{
    char *p, *n, *new;

```

Full-Disclosure: Advisory 01/2004: 12 x Gaim remote overflows

```
n = new = g_malloc(strlen (str) + 1);

for (p = (char *)str; *p; p++, n++) {
    if (*p == '=') {
        sscanf(p + 1, "%2x\n", (int *)n); <----- [08]
        p += 2; <----- [09]
    }
    else if (*p == '_')
        *n = ' ';
    else
        *n = *p;
}

*n = '\0';
...
```

Because these bugs are very similar to [01] and [02] only the vulnerable code snippet is shown here. For an explanation read the yahoo_decode() vulnerability description.

[10 – URL Parser Function Overflow]

At various places this utility function is used to split an url into its parts. Because temporary fixed size stackbuffers are used in an unsafe way a standard stackoverflow can be caused.

Affected version: <= 0.75

File: gaim/src/util.c

Function: gaim_url_parse()

Code:

```
gboolean
gaim_url_parse(const char *url, char **ret_host,
               int *ret_port, char **ret_path)
{
    char scan_info[255];
    char port_str[5];
    int f;
    const char *turl;
    char host[256], path[256];
    int port = 0;
    /* hyphen at end includes it in control set */
    static char addr_ctrl[] = "A-Za-z0-9.-";
    static char port_ctrl[] = "0-9";
    static char page_ctrl[] = "A-Za-z0-9._/!*@&%%?+=^~";

    ...
    g_snprintf(scan_info, sizeof(scan_info),
               "%%%s:%%%s/%%%s", addr_ctrl,
```

```
port_ctrl, page_ct
```

```
f = sscanf(url, scan_info, host, port_str, path); <-- [10]
```

```
...
```

Here `sscanf()` is again used in an unsafe manner. When this function is called with an oversized url, which can be triggered from several protocol handlers in different ways `sscanf()` will overwrite the stackbuffers `host` and `path`. The problem at this point is, that it is only possible to overwrite the buffers with a limited character set which makes exploitation tricky.

[11 – Extract Info Field Function Overflow]

At various places this utility function is called to copy the data between 2 tokens into a fixed size stackbuffer without a size check.

Affected version: ≤ 0.74

File: `gaim/src/util.c`

Function: `gaim_markup_extract_info_field()`

Code:

```
...
const char *p, *q;
char buf[1024];

...
p = strstr(str, start_token);
...
p += strlen(start_token) + skip;
...
q = strstr(p, end_token);

if (q != NULL && (!no_value_token ||
    (no_value_token && strcmp(p, no_value

{
    ...
    if (is_link)
    {
        strcat(dest_buffer, "<br><a href=\"");
        memcpy(buf, p, q - p); <----- [11]
        buf[q - p] = '\0';
    }
    ...
}
```

Here it is obvious that if `q - p` is bigger than 1024 bytes `memcpy()` will overwrite the stack which will result in a standard stack overflow. At the moment this routine is called from within the `get_user_info` functions of the MSN

and YMSG protocol handlers.

[12 – HTTP Proxy Connect Overflow]

When Gaim is setup to use a HTTP proxy for connecting to the server a malicious HTTP proxy can exploit it.

Affected version: <= 0.75

File: gaim/src/proxy.c

Function: http_canread()

Code:

```
static void
http_canread(gpointer data, gint source, GaimInputCondit...
{
    int nlc = 0;
    int pos = 0;
    int minor, major, status, error=0;
    struct PHB *phb = data;
    char inputline[8192], *p;

    gaim_input_remove(phb->inpa);

    while ((nlc != 2) &&
           (read(source, &inputline[pos++], 1) == 1)) {
        if (inputline[pos - 1] == '\n')
            nlc++;
        else if (inputline[pos - 1] != '\r')
            nlc = 0;
    }
    inputline[pos] = '\0';
    ...
}
```

Here the author never thought about the possibility that a proxy server could be malicious. The inputline is read into the 8192 byte buffer byte after byte until a double `\r\n` is found. Because there is no size check at all the buffer will overflow as soon the proxy sends more than 8192 bytes in a line. This bug is exploitable even if stack overwrite protections are in place because it is possible to overwrite the pointer `phb` which points to a struct that contains a callback function which is later called in the function. By overwriting the pointer and so controlling the callback function pointer it is possible to gain control over the instruction pointer before the function is left.

Proof of Concept:

e-matters is not going to release exploit for any of the these vulnerability to the public.

Full-Disclosure: Advisory 01/2004: 12 x Gaim remote overflows

CVE Information:

The Common Vulnerabilities and Exposures project (cve.mitre.org) has assigned the following names to these issues.

CAN-2004-0005: version 0.75 only, buffer overflows:

- [01 – Yahoo Octal-Encoding Decoder Overflow]
- [02 – Yahoo Octal-Encoding Decoder Out-Of-Bounds Overflow]
- [08 – Quoted Printable Decoder Overflow]
- [09 – Quoted Printable Decoder Out-Of-Bounds Overflow]

CAN-2004-0006: 0.75 and earlier, buffer overflows:

- [03 – Yahoo Web Cookie Parser Overflow]
- [04 – Yahoo Login Page Name Parser Overflow]
- [05 – Yahoo Login Page Value Parser Overflow]
- [06 – Yahoo Packet Parser Overflow]
- [10 – URL Parser Function Overflow]
- [12 – HTTP Proxy Connect Overflow]

CAN-2004-0007: 0.74 and earlier, buffer overflows:

- [11 – Extract Info Field Function Overflow]

CAN-2004-0008: 0.74 and earlier, integer overflow:

- [07 – AIM/Oscar DirectIM Integer Overflow]

Disclosure Timeline:

- 04. January 2004 – The Oscar filetransfer bug was sent to the Gaim vendor by email. Within an hour the bug was fixed within the CVS
- 10. January 2004 – Gaim vendor released version 0.75 because of a Yahoo protocol change problem(?)
Some freetime allowed deeper analysis of the new version. This revealed more bugs: 1 fixed in 0.75, some new in 0.75 and some old which are still in 0.75. All these bugs were again mailed to the vendor
- 15. January 2004 – Vendor was contacted with a patch because they had not fixed the bugs yet. Our Patch was applied in the same night
- 16. January 2004 – Vendor-sec was contacted to coordinate the disclosure process. Vendor was asked by email when 0.76 is about to come out and that this should be as soon as possible because the bugfixes were visible in their CVS with explicit commit messaged. No response to this mail until today

Full-Disclosure: Advisory 01/2004: 12 x Gaim remote overflows

- 23. January 2004 – Vendor was notified about public disclosure at the 26th.
- 25. January 2004 – Notification by the vendor that gaim 0.76 releasedate is not planned yet.
- 26. January 2004 – Public Disclosure

Recommendation:

Because there is no official new version out yet, you can download a diff against version 0.75 from

<http://security.e-matters.de/patches/gaim-0.75-fix.diff>

This patch was done by the FreeBSD security team. It is different from the official patches in the Gaim CVS. We suggest that you upgrade as soon as possible, because the explicit commit message into the official CVS tree seems to have leaked. At least it was reported to us that a link to the message on the sourceforge WebCVS was pasted into an IRC channel.

GPG-Key:

http://security.e-matters.de/gpg_key.asc

pub 1024D/75E7AAD6 2002-02-26 e-matters GmbH – Securityteam
Key fingerprint = 43DD 843C FAB9 832A E5AB CAEB 81F2 8110 75E7 AAD6

Copyright 2004 Stefan Esser. All rights reserved.

--

```
-----  
Stefan Esser                               s.esser@e-matters.de  
e-matters Security                         http://security.e-matters.de/  
GPG-Key          gpg --keyserver pgp.mit.edu --recv-key 0xCF6CAE69  
Key fingerprint   B418 B290 ACC0 C8E5 8292 8B72 D6B0 7704 CF6C AE69  
-----
```

```
Did I help you? Consider a gift:           http://wishlist.suspekt.org/  
-----
```