Novell eDirectory/iMonitor Remote Code Execution Security Advisory

08-Sept-2006

Summary

Novell's HTTP Protocol Stack (httpstk) is a component of iMonitor which provides a web-based interface for management of eDirectory, an LDAP service forming the basis for many of the world's largest identity-management deployments. The code fails to check the length of client-supplied HTTP Host request-header (e.g. Host: www.host.com) values before using them to build a formatted URL into an inadequate, statically-sized buffer on the stack. This condition occurs in a call to snprintf() while the server is preparing an HTTP redirect response and can be triggered remotely, before any authentication takes place.

Affected Software

This vulnerability has been confirmed to exist in the following products and corresponding platforms. For additional information, please see the vendor's advisory which is linked from the Remediation section of this document.

Software Title and URL	Version(s)	Platform	Perspective
✘ Novell eDirectory	>= 8.7.3.8	Windows	Remote
✘ Novell eDirectory	>= 8.7.3.8	Linux	Remote
✘ Novell eDirectory	Open Enterprise Server SP2	OES SP2	Remote
✔ Novell eDirectory	N/A (Safe)	Netware	N/A

Impact

Attacker-supplied code can be executed on vulnerable systems with a privilege level equal to the process that loads the httpstk library (the Novell Directory Services process). By default, this is NT_AUTHORITY\SYSTEM on Windows and root on Linux and Solaris.

Credit and Contact

Michael Ligh michael.ligh@mnin.org
Ryan Smith ryan@hustlelabs.com

Details – C++ Pseudo Code

The vulnerable function's logic was examined, and the general structure of the code, depicted in the C++ language, is presented in this section. This code mimics the functions' behavior as accurately as possible in order to illuminate the vulnerability; however, it is only a modest representation of the original source.

The BuildRedirectURL() method calls snprintf() in order to store the user-supplied HTTP Host request-header value into a 64-byte buffer. The assumed intention of this code is to redirect the client to a valid URL on the host that was specified in the request. Under correct circumstances, snprintf()'s length parameter is set to the maximum number of bytes the destination buffer can hold. However, this code utilized the length parameter in order to specify the number of bytes to copy from the Host request-header value, regardless of whether or not the destination buffer is capable of holding it. Therefore, a malicious individual may specify a Host request-header value exceeding 64 bytes, causing a standard stack-based buffer overflow.

```
#define HTTPHDR_HOST_FIELD 211

char szHttp[]  = "HTTP";
char szHttps[] = "HTTPS";
char szHttpS[] = "http%s://";
char szCrlf[]  = "\r\n";
char szS[]     = "s";
char szD[]     = ":%d";
char szS_3[]   = "%s";
BYTE nullbyte  = '\0';

typedef struct SAL_AddrBuf_t {
       short   sin_family;
       u_short sin_port;
       struct  in_addr sin_addr;
       struct  in6_addr sin6_addr;
       char    sa_data[42];
} SAL_AddrBuf;

class HRequest
{
public:
       int SendRedirectRsp(void);
       int SendHeader(int);
       int SendNotFoundRsp(void);
       int SendEndOfContent(void);
       int RspSetHdrValue(char *, char *);
       bool ReqIsSecureChannel(void);
       char *ReqHdrValue(unsigned int);
       SAL_AddrBuf *ReqHostAddress(void);
private:
       int BuildRedirectURL(unsigned int, bool, char *);
       char *path;
       HDR_LOOKUP_TBL *ValueTable;
       unsigned int uint;
       int something;
```

```
        SOCKET sock;
        SAL_AddrBuf name;
};

int HRequest::BuildRedirectURL(unsigned int stackid, bool fl_https,
char *redirect_url)
{
        register char *colon, *crlf;
        register size_t length;
        register unsigned short port; // Original just recycled stackid

        // Stack variables
        SAL_AddrBuf SAL;
        char *szHostHdrValue;
        SAL_AddrBuf *pSAL;
        int retval;

        // Zero-out the local SAL_AddrBuf structure
        memset(&SAL,0,66);

        // Fill in the class' SAL_AddrBuf structure with IP and port
        pSAL = ReqHostAddress();
        SAL.sin_family = pSAL->sin_family;

        // This fills in the redirect port in SAL.sin_port
        retval = PStkEnumTransports(stackid, 2, &Callback, &SAL);
        if ((retval != 0) && (retval != SERR_CALLBACK_CANCELLED)) {
                return(0);
        }

        // Obtain a pointer to the user-supplied HTTP Host-Header value
        szHostHdrValue = ReqHdrValue(HTTPHDR_HOST_FIELD);
        if (szHostHdrValue == NULL) {
                return(SERR_INVALID_REQUEST);
        }

        // Exclude colon and/or CRLF from length of host header value
        colon = strchr(szHostHdrValue, ':');
        if (colon == NULL) {
                crlf = strstr(szHostHdrValue, szCrlf);
                if (crlf == NULL) {
                        length = strlen(szHostHdrValue);
                }
                else {
                        length = crlf - szHostHdrValue;
                }
        }
        else {
                length = colon - szHostHdrValue;
        }

        // Determine if the redirect URL should be https:// or http://
        if (fl_https) {
                redirect_url += sprintf(redirect_url, szHttpS, szS);
        }
        else {
                redirect_url += sprintf(redirect_url, szHttpS, nullbyte);
        }

        // Append the Host-Header value to the redirect URL
        _snprintf(redirect_url, length+1, szS_3, szHostHdrValue);
        redirect_url += length;
```

```
        // Is IPv4
        if (SAL.sin_family == AF_INET) {
                if (retval == ERROR_SUCCESS) {
                        if (SAL.sin_port == 0) {
                                return(SERR_OBJECT_NOT_FOUND);
                        }
                        else {
                                memcpy((void *)&SAL.sin_addr.s_addr,
                                (void *)&pSAL->sin_addr.s_addr, 4);
                        }
                }
        }

        // Is IPv6
        else if (SAL.sin_family == AF_INET6) {
                if (retval == ERROR_SUCCESS) {
                        if (SAL.sin_port == 0) {
                                return(SERR_OBJECT_NOT_FOUND);
                        }
                        else {
                                memcpy((void *)&SAL.sin6_addr.u,
                                (void *)&pSAL->sin6_addr.u, 16);
                        }
                }
        }

        // Convert the port from network byte order to host byte order
        port = ntohs(SAL.sin_port);

        // Append the port to the redirect URL if it is non-standard
        if ((fl_https && port == 443) || (!fl_https && port == 80)) {
                        return(ERROR_SUCCESS);
        }
        sprintf(redirect_url, szD, port);
        return(ERROR_SUCCESS);
}

int HRequest::SendRedirectRsp(void) {

        register int retval;
        register bool fl_https;

        // Stack variables
        char redirect_url[64];
        char *memblock;
        unsigned int stackid;

        // Determine if the connection is operating over SSL
        fl_https = ReqIsSecureChannel();
        if (!fl_https) {
                retval = PStkGetProtocolStackByName(szHttps, &stackid);
        }
        else {
                retval = PStkGetProtocolStackByName(szHttp, &stackid);
        }

        if (retval == ERROR_SUCCESS) {

                // Call this function to begin building the redirect URL
                retval = BuildRedirectURL(stackid, fl_https, redirect_url);

        // Remaining code snipped for brevity
}
```

Details - Disassembly

The following disassembly of httpstk.dlm for Windows, version 201.14.24.0, depicts this vulnerability.

```
HRequest::SendRedirectRsp:

.text:62407079  lea     eax, [ebp+request_url]
.text:6240707C  mov     ecx, esi
.text:6240707E  test    bl, bl
.text:62407080  push    eax
.text:62407081  setz    al
.text:62407084  push    eax
.text:62407085  push    [ebp+stackid]
.text:62407088  call    HRequest::BuildRequestURL
```

```
HRequest::BuildRedirectURL:

.text:62406EF0  push    [ebp+szHostHdrValue]
.text:62406EF3  mov     esi, eax
.text:62406EF5  add     esi, dword ptr [ebp+request_url]
.text:62406EF8  lea     eax, [edi+1]
.text:62406EFB  push    offset aS_3
.text:62406F00  push    eax
.text:62406F01  push    esi
.text:62406F02  call    ds:_snprintf
```

Exploit Design

The stack overflow can be triggered with a GET, POST, or HEAD request to a path on the server that produces an HTTP 302 redirect response (mainly /nds and /dhost), when accompanied by an overly-long Host request-header value. By examining the stack segment around the vulnerable function, one can locate and overwrite the return address (Linux) or structured exception handlers (Windows) to gain control of execution. When exploiting this vulnerability, the shell code should be optimized to not contain NULL bytes, colons, carriage returns, or line feeds.

A proof-of-concept exploit has been developed that will reliably exploit all versions of eDirectory on both Windows and Linux, without the need of fingerprinting. This was accomplished by using SEH overwrites and saved-instruction-pointer overwrites at the same time.

The vulnerability's existence may be determined by sending an overly-long HTTP Host-header value containing a colon in the first few bytes. Versions that are not vulnerable will send a redirect containing an IP address, whereas versions that are vulnerable will send a redirect containing the initial bytes before the colon. Furthermore, a web request to an

invalid resource (e.g. http://host/an_invalid_resource) will result in an error page being returned. This error page has been found to be variable depending on the version of eDirectory that is installed.

Defense

The following Snort rules can *assist* in the detection or prevention of attacks against vulnerable systems, however limitations exist and they should never be used in lieu of patching. False negatives may be high if certain evasion techniques are implemented, including the use of non-standard white space and/or carriage returns. Additionally, it is possible to exploit vulnerable systems over the HttpStk SSL listener (if enabled), and this will go undetected by these rules.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 8028 (msg:"BLEEDING-EDGE Novell
HttpStk Remote Code Execution Attempt /nds"; flow:to_server,established;
content:"/nds"; depth:10; nocase; content:"|0d0a|Host|3a|"; nocase;
content:!"|0d0a|"; within:56; classtype:web-application-attack; sid:20060808;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 8028 (msg:"BLEEDING-EDGE Novell
HttpStk Remote Code Execution Attempt /dhost"; flow:to_server,established;
content:"/dhost"; depth:10; nocase; content:"|0d0a|Host|3a|"; nocase;
content:!"|0d0a|"; within:56; classtype:web-application-attack; sid:20060809;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 8028 (msg:"BLEEDING-EDGE Novell
HttpStk Remote Code Execution Attempt /nds (linewrap)";
flow:to_server,established; content:"/nds"; depth:10; nocase;
content:"|0d0a|Host|3a|"; nocase; content:"|0d0a20|"; within:56; classtype:web-
application-attack; sid:20060810;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 8028 (msg:"BLEEDING-EDGE Novell
HttpStk Remote Code Execution Attempt /dhost (linewrap)";
flow:to_server,established; content:"/dhost"; depth:10; nocase;
content:"|0d0a|Host|3a|"; nocase; content:"|0d0a20|"; within:56;
classtype:web-application-attack; sid:20060811;)
```

Based on information supplied in the Exploit Design section, it is also possible to detect potential weaknesses on one's network using the Nessus security scanner. A plugin for this vulnerability is provided below.

```
if (description)
{
       script_id(999999);
       script_version ("$Revision: 1.0 $");
       name["english"] = "eDirectory overly-long Host request-header overflow";
       script_name(english:name["english"]);
       desc["english"] = "
       Synopsis :
       Arbitrary code can be executed on the remote host
       Description :
       Nessus has determined that the remote host responds similarly
       to a vulnerable version of eDirectory.  The specific vulnerability
       is documented at http://www.mnin.org/advisories/2006_novell_httpstk.pdf.
```

```
        Solution :
        This problem is resolved by applying eDir 8.8.1 ftf for eDirectory
        8.8 or edir8739imon.tgz for eDirectory 8.7.3.8.
        Risk factor : High";
        script_description(english:desc["english"]);
        summary["english"] = "Checks for the eDirectory HTTP host request-header
        value overflow";
        script_summary(english:summary["english"]);
        family["english"] = "Web Servers";
        script_family(english:family["english"]);
        exit(0);
}

include("global_settings.inc");
include("http_func.inc");
include("http_keepalive.inc");

port=get_http_port(default:8028);
if(!get_port_state(port))
exit(0);

hoststr="nessus:" + crap(500);
host=string("Host: ",hoststr,"\r\n");
url_nds=string("GET /nds HTTP/1.1\r\n",host,"\r\n");

sock=open_sock_tcp(port);
if(sock==NULL)
exit(0);

send(socket:sock,data:url_nds);
while(s=recv_line(socket:sock,length:1024)){
if(egrep(pattern:"^Location: https?://nessus",string:s))
  security_hole(port);
}
close(sock);
exit(0);
```

Remediation

FTF packages and additional information will be available on Novell's website by searching for one or more of the following TIDs: 2974592, 2974600, 2974603.

Event Timeline

08-Sep-2006	Located vulnerability
08-Sep-2006	Drafted this advisory
11-Sep-2006	Notified vendor of vulnerability
20-Oct-2006	Vendor to release FTF packages

Attributions

The good and no-good thumb images were purchased from www.istockphoto.com for one dollar.

The vulnerable software was obtained from:
http://www.novell.com

The code snippet was extracted from the disassembly pane of IDA Pro:
http://www.datarescue.com

License