Global Information Assurance Certification Paper

Auditing Web Application User Input Handling - case NameSurfer

1. Introduction

The scope of this audit was to research the state of the art of web application user input validation and finally audit a WWW application user input handling. The application in question is NameSurfer, a web user interface for DNS data management. I also wanted to test some tools for web application auditing.

2. The audit target: NameSurfer

NameSurfer is an application to manage Internet Domain Name System (DNS) data, mostly zones, IP addresses and associated names contained in the zones. A zone is a DNS specific term but for the purpose of this audit it is equivalent to a domain, such as sans.org or namesurfer.com.

The application structure is shown in Figure 1:

Web
UI

(Perl)
DB
Core (C++)
DNS
servers
USER
HTTP
HTTPS

FIGURE 1. NameSurfer architecture

There is a Perl based WWW user interface, which interacts with backend core process. The core process handles the actual database manipulation and propagates the changes to actual Internet (or intranet) DNS servers. Thus NameSurfer acts as a hidden primary nameserver for the domains it manages.

NameSurfer is an old product in terms of current Internet years. The first version, structured much like it is today, came out in 1996 and the now audited version was numbered 4.01.

The test system was a NameSurfer binary installation on Red Hat Linux 6.2 with kernel 2.2.14 running on a generic PC hardware. The system doing the audit, i.e acting as a NameSurfer client, was a HP Omnibook 500 laptop running Debian Linux unstable with kernel 2.4.12.

There is a somewhat old NameSurfer evaluation version on-line at http://www.namesurfer.com/evaluation.html.

3. NameSurfer risks

NameSurfer can have several roles in an organization: first of all, it can be an IT department tool for managing the company intranet namespace. Secondly, it can be also used as a tool to manage the company external namespace, i.e. names that are visible to Internet. Thirdly, it can be used in an Internet Service Provider setting to give client organizations remote management access to their own namespace even if the actual name servers are running on the ISP. Each user can be compartmentalized to show only a specified subset of zones and names and to allow allocation of a specified IP address range only.

The test was set up to cover only the simple corporate internet/intranet case, i.e we assume that all users are company staff and all users edit the same zone. This might not be true in all corporate cases and definitely not true in an ISP setting.

In this setup risks are relatively small for the following reasons:

- This is not an application for the masses, i.e for all Internet users.
- All users are authenticated, i.e their identity is known to some degree. The degree depends on the organization account policy. In some organization a phone call might suffice to get access, somewhere else a paper trail may be needed.
- All users are internal to the organization, i.e they can be thought of as friendly. But even in friendly environments the application should be robust enough to be resistant to mistakes and even some deliberate input mangling.
- The application location is typically within the organization security perimeter, i.e in intranet, even if it manages Internet name space.

In more complex environments user compartmentalization issues become more prominent, i.e users should not be able to access data outside their own defined domain. Also in the ISP setting the ISP can not necessarily verify or control the client organization account policies and practices to determine who is actually using the account and if the username and password are stored safely. This may mean the users are not necessarily friendly.

4. Auditing the user input handling

4.1 Testing methodology

As explained in the phase 1 [Vil01] a tool called Spike was selected to help automate the tedious task of testing various strings exhaustively in all possible user input points

Since Spike is free software and more on the proof of concept stage than an actual tool, it was not quite ready for mass testing out of the tar archive.

After compiling and installing the software, Spike auditing process had several steps:

1. Running the supplied web proxy on the client host against the NameSurfer server: `webmitm -i <server-ip> -p <application-port>.` NameSurfer listens on a non-standard port, 8053.
2. Running the browser to URL `http://localhost/` to access NameSurfer via the proxy and doing the actions that need to be tested. The proxy will produce one file (`http-request-[0-9]+.0`) for each HTTP request.
3. The C language test cases need to be generated from the .O files by a Spike script: `makewebfuzz.pl input-file.0 > output-file.c`
4. The `.c` output files need to be compiled to generate actual test programs.
5. Each test program needs to be run against the target server and port: `http_request <target-ip> <target-port> >file.out`
6. Each output file needs to be analysed.

The process is lengthy and since each HTTP request generates it's own test case, there can be hundreds of test cases even in a relatively small application test. Therefore some additional development needed to be done.

To automate steps 3 and 4, I wrote a makefile (see Appendix 1) to help generate the test binaries. Additionally, I wrote a small `runtests` (see Appendix 2) shell script for step 5 to iterate through the test binary running.

As there are possibly hundreds of tests, there are also hundreds of output files. And since this method traps each HTTP request, some requests and output files will be identical (for example for graphics, stylesheets and scripts). To help the auditor, these can be weeded out from the actual analysis process. I wrote a small Perl script `find_dup.pl` (see Appendix 3) that reads each request file and first compares all the requests. For each identical GET/PUT request the output files are compared and if they are identical, the latter output file is renamed with the ending `.dup`. This sameness comparison should not be done before running the tests (i.e based on just the HTTP request GET/PUT line), since sometimes the context, such as the Referer field, may change the request output.

This elimination process worked rather well, for example after one audit run only 55 of 185 output files remained to be analysed. Of course, the application specifics, such as the number of pictures, scripts etc. static elements may change the effectiveness of this tool. Moreover, some output files can be almost identical, i.e differing in timestamp or something like benign, but to be on the safe side these should be still analysed by hand. My script notifies the auditor that it expected some test results to be similar, but some differences were found. This should be a red flag for the auditor.

After this process, there still remains tens, possibly hundreds actual unique output files that need to be analysed by hand. The output files the Spike method produces can be large, in NameSurfer case they ranged from 1kB to 20MB. And since most of the unique result files were in the large end of the size scale, there were hundreds of megabytes of material to be analysed.

The output files were very large because for each modifiable parameter in the request Spike tried 26 different tests and some of those tests were for buffer overflows. So there were normally <URL parameter_count> x 26 HTML pages in the output file. And depending on the application page, the tested parameter could occur several times in the output HTML, which meant the 100 000 byte buffer overflow strings could occur in the output file quite many times.

4.2 Spike tests

The Spike default configuration contained 26 different strings to be used against various user manipulable elements. The strings are (the shorthand n x ‘A’ denotes n times string ‘A’):

1. ../../../../../../../../../../../../etc/hosts%00
2. ../../../../../../../../../../../../etc/passwd%00
3. ../../../../../../../../../../../../etc/shadow%00
4. 999 999 x ’A’
5. 100 000 x ’A’
6. 50 000 x ’A’
7. 25 000 x ’A’
8. 12 000 ’A’
9. 4096 x ’A’
10. 1024 x ’A’
11. 512 x ’ A’
12. 100 x ’A’
13. 64 x ’A’

14. 100 000 x '<'
15. 'sqlattempt1
16. (sqlattempt2)
17. OR 1=1
18. 21 x '%n'
19. ;bob
20. "hihihi
21. |dir
22. |ls
23. 14 x '%25%5c..'
24. !@#$%%^#$%#$@#$%$$@#$%^^**(()
25. %01%02%03%04%0a%0d%0aADSF
26. 128 x '%25n'

The targets Spike "fuzzes" i.e replaces with the above strings are all URL parameters, Cookies and POST parameters. It does not fuzz all HTTP request headers.

4.3 Audited NameSurfer functions

The following NameSurfer functions were audited:

- Zone data handling.
- Host data handling.
- User account handling
- Viewing NameSurfer log.

There were other elements in NameSurfer (for example IP address management functions, raw DNS data uploads etc.), but this core function set alone produced:

- 442 test cases of which unique results were 278. The tests were made in four separate runs and the uniqueness was detected only within a run, so the unique number was somewhat less in reality. But this was the number of hand analysed files.
- The 278 unique cases had 499 megabytes of output.

Additionally, some ten or so extra tests with a modified Spike test set was made to selected queries to detect any new anomalies. These new tests produced two megabytes or so output in total.

This sheer volume of output and the limited time for the certification practical caused the

audit scope to be narrowed from the original plans.

4.4 Actual observations

Tests were run against a domain `example.test`. Some tests were done using the administrator user account, some with a test user account.

4.4.1 Client side validation

Spike as a whole is a test case for client side validation, i.e it tries to manipulate as much user input as possible.

NameSurfer uses a cookie ("lastform") and the Referer:-field to create context for the query, there is no session concept as such.

NameSurfer did not detect the user input manipulation except in the trivial cases (i.e tried to access host or zone that did not exist).

4.4.2 Cross site scripting

Since the purpose of NameSurfer is to present data from database in a HTML format, cross site scripting is a possible problem for NameSurfer.

Since Spike did not have a ready made HTML insertion test, cross site scripting problem was examined by looking at test output files for elements where any user input was directly given for client browser.

Two such places were found:

- Log viewing in two occasions.

  When viewing the log file, NameSurfer displays the zone/domain name in HTML page. And since the zone name is passed in URL and no sanity checks are done, a malicious user can embed script code or a link to malicious script code to the link. This requires the presence of URL encoding possibility (see Section 4.4.8).

  Normal URL: /log-query?ret=zone&zone=example.test&class=in

  URL showing the attack: /log-query?ret=zone&zone=%3Cscript%20SRC%3D%27http://www.malicious.org/ malicious.js%27%3E%3C/script%3E&class=in

  This allows NameSurfer to be used as an attack platform against third parties. However, since users are normally authenticated, the potential attack targets are other NameSurfer users, not all Internet/intranet users in general.

  Additionally, the attacker must get the targets to click the malicious link. Since the zone name in log viewing is never stored in the database, this link must be commu-

nicated to the targets outside NameSurfer, for example by e-mail.

Exactly the same problem is in log date error display, when a user has submitted a malformed date for the log query.

Normal URL: / log?
frame=right&ret=index&to=&zone=zones&from=&class=nsurf&go=Go

Attack URL: / log?frame=right&ret=index&to=&zone=zones&from=%3Cscript%
20SRC%3D%2 7http://www.malicious.org/malicious.js%27%3E%3C/ script%
3E;class=nsurf&go=Go

- NameSurfer specific private HTML resource record.

  For each host, there is a possibility to attach a HTML comment as a NameSurfer private resource record. This record is never propagated to actual outside DNS servers, but is stored in the NameSurfer database.

  This field allows practically any valid HTML content, including malicious code. And since this field is stored in the database, it can be used to attack other NameSurfer users having access to the same data as the attacker.

4.4.3 Direct OS command

Spike has two tests that if used as a parameter to a Perl open() call, might cause the execution of a OS command.

- |ls
- |dir

These did not cause problems to NameSurfer in any tested part of the user interface.

Example 1:

Normal URL: /user?frame=right&ret=user-list&define=password&zone=users

Tried URL: /user?frame=right&ret=user-list&define=|ls&zone=users

Expected response:
Unknown RR type "|ls"

Received response:
Unknown RR type "|ls"

Additional tests later incorporated into Spike tried the effect of backticks with no security implications.

Example 2:

Normal URL: /zone?frame=right&zone=example.test&class=in

Tried URL: /zone?frame=‘pwd‘&zone=example.test&class=in

Expected response:
Unknown frame: ‘pwd‘
<P>

Received response:
Unknown frame: ‘pwd‘
<P>

4.4.4 Direct SQL command

Tests to inject SQL commands into input strings were:

- ‘sqlattempt1
- (sqlattempt2)
- OR 1=1

At a later phase a real SQL command was tried instead of general “sqlattempt” string. No vulnerabilities of this type were found.

Example 1:

Normal URL: /user?name=testuser.users&ret=user-list&zone=users

Tried URL: /user?name=’sqlattempt&ret=user-list&zone=users

Expected response:
The name <B>’sqlattempt1</B> is not known in the zone <B>users</B>.

Received response:
The name <B>’sqlattempt1</B> is not known in the zone <B>users</B>.

Example 2:

Normal URL: /zone?frame=right&zone=example.test&class=in

Tried URL: /zone?frame=right&zone=example.test&class=
0’;insert+into+users+username+values(’foo’);--

Expected response:
Unknown DNS class “0’;insert+into+users+username+values(’foo’);--”

Received response:

Unknown DNS class "0’"
<P>

The output deviation from the expected was due to special handling of the ; character. Additionally, the “OR 1=1” caused various problems in the application because of the embedded space character. See Section 4.4.6 for more discussion on the special characters.

4.4.5 Directory traversal

Directory traversal issues were queried in spike by trying to access extra files by replacing the original URL elements with strings:

- ../../../../../../../../../../../../etc/passwd%00
- ../../../../../../../../../../../../etc/shadow%00
- ../../../../../../../../../../../../etc/hosts%00
- %25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c.. %25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%00

While these did not produce any OS level access in NameSurfer, the null byte caused a Perl error (see Section 4.4.7 for more details).

In additional Spike tests the plain string “../../../../../../../../../../../../etc/hosts” without the null byte was added to the test set and no problems were found.

Example 1:

Normal URL: /zone?frame=right&zone=example.test&class=in

Tried URL: /zone?frame=right&zone=example.test&class=../../../../../../../../../../../etc/hosts

Expected response:
Unknown DNS class “../../../../../../../../../../../../etc/hosts”

Received response:
Unknown DNS class "../../../../../../../../../../../../etc/ hosts"
<P>

4.4.6 Meta characters

The following Spike strings test the system also for meta character input:

- ;bob
- |ls
- |dir

- !@#$%%^#$%#$@#$%$$@#$%^^**(()
- %01%02%03%04%0a%0d%0aADSF
- 100 000 characters ‘<‘
- OR 1=1

Additionally the effect of backticks was tested. Most of the tests passed with flying colours:

Example 1:

> Normal URL: /icons/text
>
> Tried URL: /icons/text?!@#$%%^#$%#$@#$%$$@#$%^^**(()
>
> Expected response:
> The document /icons/text?!@#$%%^#$%#$@#$%$$@#$%^^**(() does not exist.<P>
>
> Received response:
> The document /icons/text?!@#$%%^#$%#$@#$%$$@#$%^^**(() does not exist.<P>

However, the “;” and “ “ characters caused some problems:

Example 2:

> Normal URL: /node?name=mail.example.test&ret=zone&zone=example.test&class=in
>
> Problematic URL: /node?name=;bob&ret=zone&zone=example.test&class=in
>
> Expected response:
> The name <B>;bob</B> is not known in the zone <B>example.test</B>
>
> Received response:
> The name <B></B> is not known in the zone <B>example.test</B>

The problem is that the “;” character in the middle of URL is handled as a parameter value terminator. However, this does not seem to be an exploitable security problem.

Example 3:

> Normal URL: /zone?frame=left&ret=index&zone=example.test&class=in
>
> Problematic URL: /zone?frame=OR 1=1&ret=index&zone=example.test&class=in

Expected response:
HTTP/1.0 400 Bad Request

Received response:
HTTP/1.0 200 OK
...
The required argument <B>zone</B> is missing from the URL or form data.

Here also the space character is a terminator, in this case it is the whole URL terminator and no value parameters are read after the space. This type of request line is in violation of HTTP specification (RFC2616, chapter 5.1) and the server should probably respond with error code “400 Bad Request” instead of success.

4.4.7 Null character

Null character input was also tested with directory traversal strings:

- ../../../../../../../../../../../etc/passwd%00
- ../../../../../../../../../../../etc/shadow%00
- ../../../../../../../../../../../etc/hosts%00
- %25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c.. %25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%00

Additionally a string “users%00no” was later added to test set to determine whether a null character in the middle of the string caused any extra problems.

The null byte caused a Perl error in two of the UI scripts:

Example 1:

Normal URL: /index?frame=left

Problematic URL: /index?frame=../../../../../../../../../../../etc/hosts%00

Expected response:
Unknown frame ../../../../../../../../../../../etc/hosts

Received response:
Unknown frame: ../../../../../../../../../../ ../etc/hosts at /usr/local/namesurfer/webui/frames.pl line 84, <GEN2217> chunk 13

Example 2:

Normal URL: /zone?frame=left&ret=index&zone=example.test&class=in

Problematic URL:
/zone?frame=left&ret=index&zone=example.test&class=%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c. .%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%2 5%5c..%00

Expected response:
Unknown DNS class "%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\.."

Received response:
Unknown DNS class "%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\..%\.. at /usr/local/ namesurfer/lib/util.pl line 465, <GEN2109> chunk 13.

Example 3:

Normal URL: / log?
frame=right&ret=index&to=&zone=zones&from=&class=nsurf&go=G o

Problematic URL: /log?frame=right&ret=index&to=&zone=zones&from=../../../../../../../../../../../etc/shadow%00&class=nsurf&go=Go

Expected response:
Cannot understand the date <B>../../../../../../../../../../../ ../etc/shadow</B>

Received response:
Cannot understand the date: <B>../../../../../../../../../../../../etc/shadow at /usr/local/namesurfer/webui/bw3.pl line 422, <GEN27967> chunk 13.

The result of this problem being:

- Disclosure of the Unix platform.
- Disclosure of NameSurfer installation path.
- Disclosure of Perl language.

The null byte in the middle of the string caused exactly the same problems but no new ones.

4.4.8 URL encoding

Tests utilizing encoded characters in input strings were:

- ../../../../../../../../../../../etc/hosts%00
- ../../../../../../../../../../../etc/passwd%00
- ../../../../../../../../../../../etc/shadow%00
- %n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n%n

- %25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%25%5c..%00
- %01%02%03%04%0a%0d%0aADSF
- %25n

As previously found, NameSurfer accepts %xx encoded data as input. This has caused NameSurfer to be vulnerable to null byte attack (see Section 4.4.7) and cross site scripting (Section 4.4.2).

4.4.9 Buffer overflow

Spike had several test to determine system vulnerability for buffer overflows. These were:

- 999 999 characters ‘A’
- 100 000 characters ‘A’
- 50 000 characters ‘A’
- 25 000 characters ‘A’
- 12 000 characters ‘A’
- 4096 characters ‘A’
- 1024 characters ‘A’
- 512 characters ‘A’
- 100 characters ‘A’
- 64 characters ‘A’
- 100 000 characters ‘<‘

NameSurfer was not vulnerable to buffer overflows, all length user input was handled correctly. This is not surprising, since NameSurfer user interface is made in Perl, where the interpreter itself handles memory allocation and thus avoids buffer overflows [PERLRISKS].

Example 1:

Normal URL: /user?name=testuser.users&frame=right&ret=user-list&zone=users

Tried URL: / user?
name=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA&frame=right&ret=user -
list&zone=users

Expected response:
Domain name label too long
<P>

Received response:
Domain name label too long
<P>

4.4.10 Binary data

In addition to the null testing Spike has one test to determine whether non-printable data causes problems in the application:

- %01%02%03%04%0a%0d%0aADSF

Two vulnerable spots were found, where this string was displayed as is to the user. They were in exactly the same spots as the cross site scripting problems in log viewing (see Section 4.4.2). This also required the presence of URL encoding (see Section 4.4.8).

Example:

Normal URL: /log-query?ret=zone&zone=example.test&class=in

Tried URL: /log-query?ret=zone&zone=%01%02%03%04%0a%0d%
0aADSF&class=in

Expected response:
<TITLE>NameSurfer - Change log(%01%02%03%04%0a%0d%0aADSF)
</TITLE>

Received response:
<TITLE>NameSurfer - Change log(\01\02\03\04

ADSF)</TITLE>

Here it can be seen that the URL encoded strings have been interpreted and displayed as is to the browser. \01 etc. are a shorthand notation for the corresponding binary characters that would not be otherwise visible in this document. The sequence %0a%0d%0a, meaning “\n\r\n” can be seen also in this paper version.

5. Conclusions

5.1 Security of NameSurfer user input handling

Overall, there were several problems with NameSurfer's capabilities to handle unexpected user input. However, most of these problems did not lead to exploitable security vulnerabilities in our tests.

The found problems in the order of severity are:

1. Cross site scripting
2. Null byte / binary data
3. URL encoding
4. Special character handling

Fixing the first two problems should be a priority and can be accomplished in just a few hours. However, the actual problem with the whole problem list is inadequate user input filtering. Correcting this underlying problem may be a somewhat larger issue and may take several days to:

- (re)think and (re)design a user input filter for NameSurfer
- implement and document it
- re-run some of the now-run tests to verify correct behaviour.

It is also clearly visible that user input handling testing has not been a part of the NameSurfer functional testing. This is probably something that needs to be done as well to make sure any changes in the user interface code do not break the security.

Going back to the risks and the use case explored in Section 3, the following statement can be made. In its present state NameSurfer v.4.01 user input handling is tight enough to be used in corporate intranets, but for non-authenticated or ISP use at least the two topmost problems in the problem list above should be fixed.

Of course, NameSurfer may have other security problems outside the scope of this audit, so the above mentioned statement cannot be generalized to the whole of NameSurfer.

5.2 Evaluation of the audit method

The OWASP list is an excellent and well thought of list of different types of problems in user input handling. With the addition of buffer overflow testing it forms a solid checklist of input handling issues.

The practical audit method used leaves much to be desired, however.

The good point in using Spike was the automated testing and the exhaustiveness of it. It saves a lot of typing and definitely goes through every element in the requests.

However, at the same time Spike transforms one time consuming element to another time consuming element. With manual proxy based auditing the time goes to typing various strings by hand (or doing copy & paste). With Spike this is turned to going manually through megabytes of output files, which is also very time consuming.

Therefore, if Spike is to be used, some analysis tools are needed to semi-automatically go through the files to detect anomalies. This may not be trivial, though. Besides, different applications may require different analysis tools.

Additionally, looking at the OWASP categories and the list of Spike tests in Section 4.2 we can see that the Spike test set is not currently complete enough. This is not a real problem, since Spike is open source software and the piece of C-code responsible for generating the test code is not hard to understand even for a novice C programmer. However, to make editing the code and adding tests easier, the tests should be split into a completely separate .h or .c file.

The following additional strings should be tested by Spike:

- Directory traversal without the null byte.
- a Windows-style directory traversal with command.com etc. execution
- <h1>foo</h1> and/or <script Language=JavaScript>alert('Gotcha!');</script> for cross site scripting detection
- Better SQL tests, although doing a good test requires knowledge of the application.
- A null byte test, where the null byte is in the middle of the string
- Various meta character tests, such as
    - backticks and quotation marks in pairs
    - backslashes as themselves
    - tilde, braces and brackets

The Spike test set was modified to do some of these at the last test run.

Spike does not fuzz all the HTTP request headers either. It should be modified to test for example the Accept-Language header as well.

Overall, it seems that Spike is currently not the tool for doing the full input handing audit, but a reasonable tool to do some automatic tests and then do additional selected tests with a hand driven proxy.

It also needs to be stressed, that this method concentrated only on the user input handling of the web application. There are several other areas in web application audit, as well as in the whole web service security.

6. References

[PERLRISKS] http://www.perl.org/phbs/reduce-risks.html

[RFC2616] Fielding et al. Hypertext Transfer Protocol -- HTTP/1.1. Request For Comments 2616. 1999.

[Vil01] Viljanen, L. Auditing Web Application User Input Handling. Assignment 1 of GSNA Certification.

Appendix 1: Spike tests Makefile

```
#
# Makefile for making SPIKE tests
# We assume SPIKE has been compiled already
#

# All webmitm generated browses are in http_request-XX.0 files
# Test case sources will be in http_request-XX.c files
# All executable tests will be in http_request-XX
#

# Usage: make all
#
# - ladybug 21.11.2001

CC = gcc

FUZZER = bin/makewebfuzz.pl

# These are the needed SPIKE files

SPIKESRC = src/webfuzzpostlude.c src/webfuzzprelude.c
SPIKEOBJS = objs/spike.o objs/listener.o objs/tcpstuff.o
SPIKEHDRS = include/spike.h include/listener.h include/tcpstuff.h include/hdebug.h

SRC := $(patsubst %.0,%.c,$(wildcard *.0))
TARGETS := $(patsubst %.0,%,$(wildcard *.0))

$(SRC): %.c : %.0 $(SPIKESRC)
        $(FUZZER) $< >$@

$(TARGETS): % : %.c
        $(CC) -Iinclude -o $@ $(SPIKEOBJS) $<

clean:
        rm $(SRC) $(TARGETS)

realclean:
        rm -f *.0 $(SRC) $(TARGETS)

all: $(TARGETS)
```

Appendix 2: runtests script for Spike

```
#!/bin/sh
#
# A script to run SPIKE generated tests
# We assume tests are named http_request-X...
#

# Usage:
#       ./runtests <host-ip> <port>
#

# - ladybug 30.11.2001
#

if [ $# -ne 2 ]
then
        echo "Usage: runtests <host-ip> <port>"
        exit 1;
fi

host=$1
port=$2

for i in ‘ls http_request* | sort -t- -n -k2‘
do
        if [ -x $i ]
        then
                ./$i $host $port > $i.out
        fi
done

exit 0
```

Appendix 3: find_dup.pl

```
#!/usr/bin/perl
#
# Usage: ./find_dup.pl
#
# Reads spike output and finds duplicate output files
# We assume http_request-*.0 and *.out files
#
# Author: ladybug, 3.12.2001
#

use File::Glob ':glob';

$inputending = ".0";
$outputending = ".out";
$dupending = ".dup.";
$namebase = "http_request-";

#
# Step 1: read all *.0 files and determine duplicate URLs
#

my @files = bsd_glob('*' . $inputending);
if (GLOB_ERROR)
{
   die("Couldn't get list of $inputending files: $!");
}

foreach my $file (@files)
{
   # Open file, read first line
   open (OFILE, "$file") || die ("can't open $file: $!");
   $get = readline(OFILE);

   # Get request number from file name
   ($name, $numpart) = split($namebase,$file);
   ($num, $end) = split(/_/,$numpart);

   # HTTP Request format: ACTION URL HTTP/1.1
   # Put into hash with request numbers as value
    ($action, $url, $http) = split(/ /, $get);
    $actionurl = join(' ', $action, $url);
    if (exists $URLS{$actionurl})
    {
      $value = $URLS{$actionurl};
```

```
      $URLS{$actionurl} = $value . " " . $num;
    }
    else {

      $URLS{$actionurl} = $num;
    }
   close(OFILE);
}

# Sort numerically and print
foreach my $key (keys %URLS)
{
   $newval = join(' ', sort {$a <=> $b} split(' ', $URLS{$key}));
   $URLS{$key} = $newval;
   print STDOUT $key . ": " . $newval . "\n";
}

#
# Step 2: Check if URL duplicate output files are really equivalent,
#        hidden parameters may change the behaviour/output.
#        If real duplicates, rename files to .dup.XXX
#
# Note: simple time stamp change will cause output files being different!
#

foreach $key (keys %URLS)
{
   @numbers = split(' ', $URLS{$key});
   $refnumber = $numbers[0];
   $firstfile = $namebase . $refnumber . $outputending;
   foreach $num (1 .. $#numbers)
   {
      $diffnum = $numbers[$num];
      $otherfile = $namebase . $diffnum . $outputending;
      next unless -f $otherfile;
      # Ha, don't reinvent the wheel...
      $outcount = `diff $firstfile $otherfile | wc -l`;
      chop($outcount);
      $outcount =~ tr/ //d;
      # If real duplicates, rename
      if ($outcount == 0)
      {
         print STDOUT "$firstfile $otherfile duplicates (diff $outcount). Renaming...\n";
         rename $otherfile, $otherfile . $dupending . $refnumber
            || die "Couldn't rename $otherfile: $!";
      }
```

```
        else {
            print STDOUT "$firstfile $otherfile not real duplicates, check diffs\n";
        }
    }
}
```