



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

## Interested in learning more?

Check out the list of upcoming events offering  
"Auditing Systems, Applications, and the Cloud (Audit 507)"  
at <http://www.giac.org/registration/gсна>

## **Tabula rasa: Auditing RobinHood under BeOS**

© SANS Institute 2000 - 2002, Author retains full rights.

## Topic

The application I have audited is a web server, *Robin Hood 1.1*, running under the *BeOS 5.0 Personal Edition* Operating System. See Appendices A and B for details on both software products. They were chosen because both were relatively new, free of charge and undertested.

I shall concentrate my efforts in two areas: Auditing the application, RobinHood, as a web server and auditing the TCP/IP stack.

## Current State of Practice

### *Auditing a web server*

While there exist several checklists for particular web-servers (e.g. Microsoft's IIS [ISS00] [ISS01], Netscape Enterprise Server [ENTE99] and Apache [APAC00]), there exists no checklist for BeOS and RobinHood, since they represent a negligible proportion of the overall OS and web server market. I searched [www.bedope.com](http://www.bedope.com), [www.beforever.com](http://www.beforever.com), [news.begroovy.com](http://news.begroovy.com), and [www.beoscentral.com](http://www.beoscentral.com).

Moreover, there is a paucity of *general auditing* checklists for auditing web servers. A notable exception is Rhoades' SANS course book [RHOA01] on auditing web-based applications, which covers almost all the bases.

I shall first give a short overview of the specific server checklists I found on the Internet using [www.google.com](http://www.google.com) and search for "<server> checklist". Then I shall illustrate via a checklist of checklists how few bases these checklists cover to be useful for auditing purposes. I shall go briefly over Rhoades checklist, comment on the objectivity/subjectivity of his points and the procedures to check them. I will then suggest a few improvement of Rhoades' work: more details in the audit areas he identified and a few additions to areas he neglected. Finally, I shall point out directions of future improvements.

### Overview

Three products – Apache, IIS and Enterprise – have cornered the market with roughly a 90% share market share [Table 1]. Two operating systems, Windows and Unix variants, represent between ninety [SURV01, Fig. 2] and ninety-five [ZOEB99, Table 1] percent of OS server used on the Internet. This is a plausible explanation for the lack of *general* checklists – no market and no demand.

Server	April 2001	% (rounded)	% (cum.)
Apache	1,7932,251	63	63
Microsoft-IIS	5,916,724	21	84

Netscape-Enterprise	1,762,872	6	90
Zeus	779,209	3	93
Rapidsite	402,829	1	94
AOLserver	272,815	1	95
thttpd	369,930	1	96
tigershark	200,620	<1	97
WebSitePro	119,586	<1	98
ConcentricHost-Ashurbanipal	106,443	<1	99

**Table 1: Web server market breakdown<sup>1</sup>**

## Microsoft IIS

The Microsoft IIS 4.0 Security checklist

(<http://www.microsoft.com/technet/security/iischk.asp>) consists of six sections:

1. *General information*

This is information about the physical installation of the IIS server.  
Five fields record who set up what, where and when.

2. *Background work*

These are steps to ensure that you are adequately prepared to secure IIS.

These four steps include reading the corporate security policy, configuring hardware to meet security policy, reading the IIS4 Resource Kit Security Chapter and subscribing to Microsoft Security Notification Service. Curiously enough, there is not detail provided for configuring your hardware.

3. *Windows NT 4.0 settings*

These are steps to secure the Windows NT 4.0 operating system.

There are thirty-five points in total. Most of them are good general advice for Windows NT systems in general (disabling guest accounts, removing net shares, prevent unauthenticated access to the registry, etc). Two advisories stand out because they are somewhat rare: One calls for the removal of any unused ODBC / OLE – DB Data source and drivers; the other for the synchronization of the time clocks on all web servers for log synchronization in case of intrusions.<sup>2</sup>

---

<sup>1</sup> See Netcraft.com for more details: <http://www.netcraft.co.uk/survey/>

<sup>2</sup> See <http://www.microsoft.com/technet/security/iischk.asp#31>

#### 4. *Internet Information Server 4.0 Settings*

These are the steps to audit and secure the IIS server proper.

There are twenty-one points in total, four of which are not necessarily specific to IIS.<sup>3</sup> These are

- IIS-1) Install minimal Internet services required using the Service Configuration Manager
- IIS-2) Set appropriate authentication methods
- IIS-3) Lock down Microsoft Certificate Server ASP Enrollment pages
- IIS-4) Disable or remove unneeded COM Components

The remaining seventeen points consists of the following:

- IIS-5) Set appropriate virtual directory permissions and partition Web application space
- IIS-6) Executable content validated for trustworthiness
- IIS-7) Set IP Address/DNS Address restrictions
- IIS-8) Set up Secure Sockets Layer
- IIS-9) Migrate new Root Certificates to IIS
- IIS-10) Remove Non-trusted Root Certificates
- IIS-11) Set Appropriate IIS Log file ACLs
- IIS-12) Logging enabled
- IIS-13) Index Server only indexing documentation
  
- IIS-14) Remove the iisadmpwd vdir
- IIS-15) Remove Used Script Mappings
- IIS-16) Disable RDS support
- IIS-17) Disable or remove all sample applications
- IIS-18) Check <FORM> input
- IIS-19) Disable calling the command shell with #exec
- IIS-20) Disable 'Parent Paths'
- IIS-21) Disable IP Address in Content-Location

#### 5. *Installing Scanner / Intrusion Software*

Microsoft recommends running a security scanner regularly. They also provide a rather nifty page, listing trusted 'security partners' the products of which, presumably, you should considered purchasing.<sup>4</sup>

#### 6. *Update the Emergency Repair Disk*

Microsoft recommends updating the ERD regularly.

---

<sup>3</sup> See <http://www.microsoft.com/technet/security/iischk.asp#IIS1>

<sup>4</sup> See <http://www.microsoft.com/technet/security/partners/default.asp>

## Netscape Enterprise Server

The Netscape Enterprise Server checklist

(<http://www.usenix.org/sage/sysadmins/solaris/webservers/netscape.html>) consists of four sections: *Installation*, *Configuration*, *Using SSL* and *maintenance*. The sections' thrusts are somewhat different from the Microsoft IIS one above. The vast majority (75%) of the checklist is devoted to the Enterprise Server, with just one point, *Install & secure the host operating system*, referring to the operating system. The minutiae of that point are quite extensive, though.

The sixteen point consist of the following:

- NES-1) Set permissions for web server directories and files
- NES-2) Disable symbolic links
- NES-3) Configure a controlled CGI area
- NES-4) Disable automatic directory listings
- NES-5) Disable the "exec" form of server side includes
- NES-6) Delete all unneeded files from the HTML document tree.
- NES-7) Delete all unapproved CGI Scripts.
- NES-8) Restrict types of operations (e.g., PUT and POST)
- NES-9) Configure server auditing
- NES-10) Configure appropriate access controls/authentication mechanisms
- NES-11) Ensure that a security banner is displayed on the home page
- NES-12) Generating a Public and Private key pair for the server
- NES-13) Requesting a server certificate
- NES-14) Installing the server certificate
- NES-15) Installing the CA certificate
- NES-16) Configuring your server to use SSL

## Apache

The Apache Web Server checklist

(<http://www.usenix.org/sage/sysadmins/solaris/webservers/apache.html>) consists of three sections: *Installation*, *Configuration*, and *Maintenance*. There is no version number specified, but a perusal of the Apache archives of March 1998 positions it be 1.2.4 or later.<sup>5</sup>

The sections' thrusts are similar to Netscape Enterprise Server, which is not surprising inasmuch as both checklists originated from the same source. About half of the checklist is devoted to the Apache web server proper, with just one point, *Install & secure the host*

---

<sup>5</sup> See [http://httpd.apache.org/info/in\\_the\\_news\\_1999.html](http://httpd.apache.org/info/in_the_news_1999.html)

*operating system*, referring to the operating system. The operating system is Solaris 2.x and again as above, the discussion is quite exhaustive.

- APA-1) Set permissions for web server directories and files
- APA-2) Delete all unapproved CGI scripts
- APA-3) Delete unneeded files from the HTML document tree
- APA-4) Make working copies of server configuration files
- APA-5) Set a server name
- APA-6) Disable automatic directory listings
- APA-7) Disable symbolic links
- APA-8) Configure server auditing
- APA-9) Configure access control & authentication
- APA-10) Disable the exec form of server side includes
- APA-11) Restrict remote operations (e.g., PUT and POST)
- APA-12) Provide a security banner for the home page

### Auditing web-based applications in general

Rhoades' procedures are more comprehensive than either three specialized checklists [RHOA01]. His taxonomy is logical, granular and far more systematic than anything else I could find. All three specialized checklists are inadequate inasmuch that they only cover parts of Rhoades auditing checklist. It is not that surprising that the specialized checklists are not comprehensive. They were meant as *install, configuration and lockdown* guides, with no further provisions for auditing of *functionality*. This approach downplays the inherent faults in software. Commercial Off-The-Shelf (COTS) software is said to exhibit the same software fault density for the past twenty years: 0.5-2 fault/K line of source code.<sup>6</sup> Windows NT 4.0, for instance, has 35 million lines of source code. A thorough auditing procedure must include black box testing. I will return to this at the end of the auditing process.

Rhoades' auditing checklist (selected points)	Checklist for Microsoft IIS	Checklist for Netscape Enterprise	Checklist for Apache Server
<b>OS Security</b>	✓	✓	✓
<b>Web server security</b>			
Server weaknesses	✓		
Default material	✓	✓	✓
Configuration issues	✓	✓	✓
<b>Web server output</b>			
HTTP header			
HTML & JavaScript			

---

<sup>6</sup> See <http://www.rstcorp.com/presentations/orlando98/sld003.htm>

Encryption	✓	✓	
Error Messages			
Caching			
<b>Authentication</b>	◆		✱
Multiple Authentication disorder			
Sign-in			
Sign-off			
<b>Session Issues</b>			
Session tracking	✱		
IP hopping & session cloning			
Concurrency			
Time-outs			
<b>Transaction issues</b>			
Unexpected User Input	✱		
Hidden FORM elements			
GET vs. POST			
JavaScript Filters			
<b>Testing</b>		**	**

**Table 2: Checklist of checklists**

- ◆ advice is to “set appropriate authentication methods”
- ✱ advice is to “check <FORM> input”
- ✱ advice is to “configure access control and authentication
- \*\* advice is to check logs, archive them and do backups, no tools are mentioned

## Evaluation

Let me know evaluate Rhoades’ approach in more detail. I shall list the audit points, comment on the objectivity / subjectivity and indicate how compliance can be checked. I made a few improvements over Rhoades’ methods – standardization of presentation and detail, for one.

### OS Security

**Point:** Are the latest OS service packs / patches installed?

**Criterion:** Objective

**Type:** Binary

**Compliance:** Moderate: Some user expertise required in gathering and interpreting information

1. OS type

- a. NT: Go to <http://windowsupdate.microsoft.com> to let an ActiveX control check it for you
- b. Unix / Unix-like:
  - i. Execute `uname -a`
  - ii. Check with your vendor whether you have the latest kernel version / patches



2. If you do not have the latest version, you may have OS security problems<sup>7</sup>
3. Check with [www.cert.org](http://www.cert.org), [icat.nist.gov/icat.cfm](http://icat.nist.gov/icat.cfm) and [www.securityfocus.org](http://www.securityfocus.org) for severity of bugs
4. Root exploit bugs means the system fails the audit. If you found this, hackers will, too. Everything else is of concern and should be fixed immediately.

## Web server security

### *Server weaknesses*

- Point:** Are the latest service packs / patches installed?
- Criterion:** Objective
- Type:** Subjective: root access weaknesses are serious, DOS may be too
- Compliance:** Moderate: Some user expertise needed in gathering and interpreting information
1. Microsoft IIS:
    - a. Go to [www.microsoft.com/technet/security/current.asp](http://www.microsoft.com/technet/security/current.asp)
    - b. Use IIS\_PROMISC (<http://unsekure.com.br>) , a free perl-based auditing tool for IIS
  2. Apache:
    - a. Execute `httpd -v`
    - b. Check patches/upgrades at [www.apache.org](http://www.apache.org)
  3. Unknown web server:
    - a. If publicly accessible: Submit web server IP to <http://www.web-caching.com/cacheability.html>
    - b. If not publicly accessible: Read the documentation that came with the product
    - c. Check Server line for identifying information
    - d. Check patches/upgrades at vendor's site
  4. If you do not have the latest patches/version, you may be vulnerable.
  5. Check with [www.cert.org](http://www.cert.org) and [www.securityfocus.org](http://www.securityfocus.org) for severity of bugs
  6. Root exploit bugs mean that the web server fails the audit. If you found this, hackers will, too. Everything else is of concern and should be fixed ASAP.

### *Default material*

- Point:** Are default items with known vulnerabilities installed?
- Criterion:** Objective
- Type:** Binary

---

<sup>7</sup> I say 'may' because NT Service Pack 3 reportedly introduced more bugs than it plugged.

**Compliance:** Moderate: Some manual expertise for researching vulnerabilities

1. Check [www.securityfocus.com](http://www.securityfocus.com) or [www.cert.org](http://www.cert.org) for vulnerabilities in items that are in typical default installation. See also Stein for very good explanation on the pitfalls of cgi scripts [STEI00, chapter 6]
2. Search for those items on your web server.
3. If you find a match, you have default vulnerabilities.
4. Root exploit bugs mean that the web server fails the audit. If you found this, hackers will, too. Everything else is of concern and should be fixed ASAP

### ***Configuration issues***

**Point:** Are interpreters such as perl, java, etc in the web document root path?

**Criterion:** Objective

**Type:** Range of harmless executables that take no user input to full-blown interpreters

**Compliance:** Moderate: Some manual expertise in ascertaining whether executables are interpreters

1. Check for executable interpreters in web document root path
  - a. NT: `dir *.exe`
  - b. Unix: `ls -la | grep '^[/^d]x'`
2. If search returns some listings, you have executables in your path and you may be at risk
3. If these are interpreters that are callable, the web server fails the audit. If you found this, hackers will, too. Everything else should be carefully investigated.

**Point:** Are directories indexable?

**Criterion:** Objective

**Type:** Binary

**Compliance:** Moderate: requires some manual expertise

1. Locate default page for particular directory (normally index.htm, index.html, default.html, etc)
2. Rename to <filename>.bak
3. Pull up <http://www.site.com/dir> (assuming directory *dir*) and see whether you get directory listings
4. Alternative: Download BlackWidow ( <http://softbytelabs.com/BlackWidow/> ) and scan website and see whether you get your directory listings returned
5. If the web server's directories are indexable, it fails the audit. If you found this, hackers will, too. This is easy to fix.

## **Web server output**

### ***HTTP header***

**Point:** Is identifying information about the server, software, etc. unnecessarily returned in the HTTP header?  
**Criterion:** Objective  
**Type:** Binary  
**Compliance:** Moderate: Automatic checking through software, but interpretation of header fields requires a little expertise. See Stevens and Yeager for details [STEV96, 166] [YEA96, 32-35]

1. If the page is publicly available:
  - a. Make a HTTP request for a page
    - i. NT: Use Netcraft ( [www.netcraft.com](http://www.netcraft.com) ) to check
    - ii. Unix: Use wget ( <http://sunsite.dk/wget/> ) to check
2. If the page is not publicly available
  - a. Run a packet sniffer
  - b. Make a HTTP request for a page
3. Check if SERVER field returns identifying information
4. In almost all cases, this is unnecessary information and the web server fails the audit if this is revealed.

### ***HTML & JavaScript***

**Point:** Is identifying information about the server, software, and passwords unnecessarily returned?  
**Criterion:** Subjective  
**Type:** Range from legitimate debugging information, unavoidable specifications to Easter eggs planted by programmers  
**Compliance:** Hard: Requires source code review of individual HTML code and good understanding of mechanisms involved.

1. Pull up a HTML page
2. Look for keywords such as “Password”, “UserName”, “Login”, “passwd”, etc.
3. If you find information that does not pertain to the display of information (which is what HTML is designed for), that is poor design and open potential security holes. If you find any of those keywords (or their synonyms), the web server fails the audit. If you are in doubt, have an experienced JavaScript programmer review the code.

### ***Encryption***

**Point:** What are the weakest SSL ciphers allowed by the web site?  
**Criterion:** Objective  
**Type:** Range from no encryption, MD5 message authentication only to triple DES, 168-bit encryption with SHA-1 message authentication<sup>8</sup>

---

<sup>8</sup> See <http://developer.netscape.com/docs/manuals/security/sslin/index.htm> for more information

**Compliance:** Moderate: Automatic checking through software, but need to know what output means (cipher identification)

1. Use <http://www.netcraft.com/sslwhats/> to check supported ciphers
2. Check output against <http://developer.netscape.com/docs/manuals/security/sslin/contents.htm#1046261> , Table 1. If you find any cipher under “exportable cipher suites” or “weak cipher suites”, your server allows for too weak keys.
3. If your web server allows for weak ciphers:
  - a. Do you have for any reason support old browsers? If not, the web server fails the audit.
  - b. Do you market to the minuscule number of countries that are not allowed to use strong encryption? If not, the web server fails the audit.

### ***Error Messages***

**Point:** Are error messages giving back unnecessary information?

**Criterion:** Subjective

**Type:** Range from non-descript message to very detailed info about server, third party software, code that is executed, etc.<sup>9</sup>

**Compliance:** Hard: *Black box testing* is necessary to induce errors.

1. See Rhoades [RHOA01, 109] and SQAtester (<http://www.sqatester.com/testingtips/blackbox/index.htm> ) for some hints
  - a. Large input (>2000 chars) in text fields
  - b. Metacharacters like ; , > ? / # \$ @ % in text fields
  - a. JavaScript in text fields: <script>windows.alert(“Cross site scripting possible”)</script>
2. Alternatively, look for vulnerabilities in product at [www.securityfocus.com](http://www.securityfocus.com) and [icat.nist.gov](http://icat.nist.gov) and apply the exploit.
3. Any output that breaks or circumvents the normal process flow is suspect and the web server fails the audit.

### ***Caching***

**Point:** Is data permitted to be cached at the proxy or at the browser?

**Criterion:** Subjective

**Type:** Range from last viewed information to very detailed info login, password, credit cards, etc.

**Compliance:** Moderate: User expertise in interpretation required

1. Run packet sniffer
2. Set browser to support only HTTP 1.0
3. Call up page

---

<sup>9</sup> Go to [www.jcupid.com](http://www.jcupid.com) and login with a valid username but no password to see informative error messages

4. Check HTTP header fields<sup>10</sup>

Check that field **Expires** is set to a date in the not-too-distant future

For sensitive information, the date should not be set more than a few minutes into the future. If it does, the web server fails the audit.

5. Set browser to support only HTTP 1.1

6. Call up page

7. Check HTTP header fields<sup>11</sup>

Check that field **Cache-Control** is set to *no-cache*

If it is not set to *no-cache*, the web server fails the audit

OR

1. Call up page where you have to authenticate and authenticate

2. Leave the authenticated area and browse

3. Go back to same page you had to authenticate for.

4. If you don't have to re-authenticate, your credentials were cached. The web server fails the audit.

## Authentication

### *Multiple Authentication disorder*

**Point:** If multiple levels of authentication are present (application, session, transaction level), are they synchronized? If not, is authentication determined by the weakest credentials?

**Criterion:** Objective

**Type:** Binary

**Compliance:** Hard: Requires systematic, fully factorial experiment for exhaustive testing by type, if not by instance.

1. Gather two or three password/logins topples with different access privilege levels, i.e. one has root privileges, once can maybe search and change files, the other one can only read
2. At every authentication step, mix the credentials and see whether you manage to perform actions reserved for higher-privilege credentials
3. If you can, you have authentication disorder and the web server fails the audit.

### *Sign-in*

---

<sup>10</sup> See [http://www.mnot.net/cache\\_docs/](http://www.mnot.net/cache_docs/) for details on HTTP headers and caching

<sup>11</sup> See [http://www.mnot.net/cache\\_docs/](http://www.mnot.net/cache_docs/) for details on HTTP headers and caching

### **User Name and Password Harvesting**

- Point:** Does failed sign-in reveal which piece (login, password) of account was incorrect or which state (suspended, lockout) the account is in?
- Criterion:** Objective
- Type:** Binary
- Compliance:** Hard: Requires systematic, fully factorial experiment for exhaustive testing
1. With  $n$  binary (correct, incorrect) criteria (login, password, account status, etc), you will have  $2^n$  test logins.
  2. Make a matrix of criteria combinations and record output messages when using particular set of criteria [RHOA01, 58]
  3. If output is not consistent, harvesting may be possible. The web server fails the audit.

### **Brute Force Password Guessing and Account Lockouts**

- Point:** Does web site enforce lockout after numerous sequential failed login attempts?
- Criterion:** Objective
- Type:** Binary
- Compliance:** Easy: Beginners can do this
1. Attempt to login repeatedly (at least five times) with an invalid user/password combination.
  2. If you can keep on doing this, lockouts are not enforced. You may be subjected to a brute force attack.
  3. If you cannot keep on doing this, lockouts are enforced. You may be subjected to a denial of service attack.
  4. This is an inherent risk of authenticated web services via login/password. The web server cannot fail this test, but it is an area to watch out for. The best you can do is to mitigate the effects. See Rhoades [RHOA01, 60-64]

### **Resource exhaustion**

- Point:** Can a malicious third-party cause a denial of service by exhausting the server's session ID resources?
- Criterion:** Objective
- Type:** Resource allocation must be determined on case-by-case basis by user volume, machine capacity
- Compliance:** Hard:
1. Find page where server assigns session id for user. The exact point in the authentication process varies, it can come before, during or after user authentication. The critical points in time are before and during user authentication [RHOA01, 73; 82].  
To ascertain session ID:
    - a. Observe URL for session ID

- b. Put a packet sniffer and look at HTTP referrer field in client request for session ID
  - c. Look at HTML code for hidden form elements with keywords like “session”
2. Depending on how the session ID is transmitted
  - a. Hit *Reload* button on browser multiple times in short order in attempt to exhaust resources
  - b. Write script that repeatedly uses the same session ID in short time in attempt to exhaust resources
3. If you are unable to establish a further session, you are vulnerable to resource exhaustion.
4. This is an inherent risk of offering web services. The web server cannot fail this test, but it is an area to watch out for.

### ***Sign-off***

- Point:** Do web server settings allow client browser to cache credentials after sign-off?
- Criterion:** Objective
- Type:** Binary
- Compliance:** Easy: Beginner can do this
1. Sign on, perform some transaction, then sign off
  2. Click on *Back* button in Browser
  3. Click on unvisited link to sensitive information and check whether you can reach it
  4. Repeat once
  5. If you are able to follow the link, the sign-off process is flawed. The web server fails the audit.

## **Session Issues**

### ***Session tracking (no cookies)***

- Point:** Can the session ID be captured or guessed and re-used for session cloning?
- Criterion:** Subjective: inherent risk
- Type:** Binary
- Compliance:** Hard: Requires user and programming skills
1. Find page where server assigns session id for user. The exact point in the authentication process varies, it can come before, during or after user authenticates [RHOA01,73; 82].  
To ascertain session ID:
    - a. Observe URL for session ID
    - b. Put a packet sniffer and look at HTTP *Referrer* field<sup>12</sup> in client request for session ID

---

<sup>12</sup> See <http://www.ics.uci.edu/pub/ietf/http/rfc1945.html#Referer>

- c. Put a packet sniffer and look for HTTP *WWW-Authenticate* field (packet from server) and a HTTP *Authorization* field (packet from client)<sup>13</sup> in the header [YEA96, 293-294]
  - d. Look at HTML code for hidden form elements with keywords like “session”, “session”, etc.
2. Write script or program that uses the same session ID in short time
3. If you can, then session ID can be reused, creating a potential security hazard
4. Alternative: Use Hijacking Suite (<http://www.pkcrew.org/tools/hjksuite>) to try to reuse session ID
5. This is an inherent risk of using unsigned session IDs. The web server fails the test, though, by Rhoades’ sensible standards [RHOA01, 96], if session ID is any of the following:
  - a. not random
  - b. short
  - c. relates to user information
  - d. does not expire
  - e. for sensitive data: not sent over a secure path (like SSL)

### ***Session tracking (with cookies)***

- Point:** Does the server use unsafe cookies, enabling session ID re-use?
- Criterion:** Objective
- Type:** Range varies by needs. At the very least, cookies should be marked SECURE, non-persistent and reasonably limited in their path and domain [RHOA01, 92]
- Compliance:** Moderate: Requires some expertise in interpreting packet sniffer output
1. Run packet sniffer and look at packets from web server with HTTP header field *Set-Cookie*<sup>14</sup>
  2. Check that cookie field data
    - a. **expires** is a reasonable date, depending on your security needs. If no date is indicated, the cookie will expire at the end of the session
    - b. **domain** is specific: \*.com is not specific enough, company.com probably is. The default value of domain is the host name of the server that generated the cookie response.
    - c. **path** is specific: / is probably too vague. If the path is not specified, it is assumed to be the same path as the document being described by the header that contains the cookie.
    - d. **secure** is set to YES which indicates HTTPS-only (encrypted) traffic. This is the one must setting for security

---

<sup>13</sup> See <http://www.ics.uci.edu/pub/ietf/http/rfc1945.html#WWW-Authenticate>

<sup>14</sup> See [http://home.netscape.com/newsref/std/cookie\\_spec.html](http://home.netscape.com/newsref/std/cookie_spec.html)



reasons. If secure is not specified, a cookie is considered safe to be sent in the clear over unsecured channels

3. Alternative: Use WinPatrol for NT ( [www.winpatrol.com](http://www.winpatrol.com) ) to check cookies
4. If the cookies properties do not meet the above-mentioned points, your session ID may be easily re-used. For practical purposes, the web server fails the audit if points a,b and d are not met. Point c may be too much of a hassle for the real world.

### ***IP Hopping & Session Cloning***

**Point:** Can the user change IP address in the middle of the session without re-authenticating?

**Criterion:** Objective

**Type:** Binary

**Compliance:** Moderate: Need configuration and user skills

1. Sign on and perform some transaction
2. Reconfigure internet adapter and route  
Assume adapter eth0, gateway 128.16.24.1, new IP address 128.16.24.109, class C network
  - a. Unix:
    - i. `ifconfig eth0 128.16.24.109 netmask 255.255.255.0 broadcast 128.16.24.255`
    - ii. `route add default gw 128.16.24.1`
  - b. Windows NT:
    - i. Open *Control Panels – Network*
    - ii. In Edit box *IP address*, type 128.16.24.109
    - iii. In Edit box *Subnet Mask*, type 255.255.255.0
    - iv. In Edit box *Default Gateway*, type 128.16.24.1
    - v. Click *Apply*
3. Continue with transaction – if you are successful, IP hopping works and the web server fails the audit.

### ***Concurrency***

**Point:** Can a single authentication credential be used for multiple sessions at the same time?

**Criterion:** Objective

**Type:** Binary

**Compliance:** Easy: Beginners can do this

1. Log on with user name and password from two different systems
2. If you are successful, you have concurrency problems. The web server fails the audit.

### ***Session time-outs***

**Point:** Are session time-outs supported?

**Criterion:** Objective

**Type:** Binary

**Compliance:** Easy: Beginners can do this

1. Log on with user name and password
2. Don't do anything for at least ten minutes
3. Perform a transaction – if you are successful, time-outs need to be set. The web server fails the audit.

### Transaction issues

#### *Unexpected User Input (Cross-site scripting)*

**Point:** Can the application withstand all kinds of user input without creating a security hole?

**Criterion:** Objective

**Type:** Range from returned detailed info about server, third party software, code that is executed to root shell access.

**Compliance:** Hard: *Black-box testing* is necessary to gauge robustness.

1. See Rhoades [RHOA01, 109] and SQAtester (<http://www.sqatester.com/testingtips/blackbox/index.htm>) for some hints on manual input
  - a. Large input (>2000 chars) in text fields
  - b. Metacharacters like ; , > ? / # \$ @ % in text fields
  - c. JavaScript in text fields: <script>windows.alert("Cross site scripting possible")</script>
2. Use automated tool like
  - a. *AppScan* ([www.sanctuminc.com](http://www.sanctuminc.com)) - very expensive (\$10,000 - \$15,000 license) to check
  - b. *twwwwscan* ([search.iland.co.kr/twwwscan](http://search.iland.co.kr/twwwscan)) – freeware for NT and Unix
3. If you manage to crash the server or see unusual responses, you have an input validation problem. The web server fails the audit.

#### *Hidden FORM elements*

**Point:** Do web pages contain HTML <FORM> elements called “hidden”?

**Criterion:** Objective

**Type:** Binary

**Compliance:** Moderate: requires some user search expertise

1. Save HTML document
2. Open saved document in editor
3. Look for TYPE=”HIDDEN”
4. If you find some, you are potentially vulnerable. To assess impact, you will have to do some black box testing. HIDDEN fields are bad programming style – the web server fails the audit to be on the safe side.

#### *GET vs. POST*

**Point:** Do forms use GET method to pass parameters to web server?

**Criterion:** Objective

**Type:** Binary

**Compliance:** Moderate: Requires some user expertise in interpretation

1. Pull up web page containing the form
2. Check for interaction method
  - a. Netscape:
    - i. Go to *View – Page Info*
    - ii. Check if method used is GET
  - b. Internet Explorer:
    - i. Go to *View – Source*
    - ii. Search for all occurrences of “action”
    - iii. In the HTML line that contains “action”, check whether “method” used is GET
3. If you find GET, you are unnecessarily exposing user info. The web server fails the audit.

### ***Client-side JavaScript user input manipulation***

**Point:** Do forms do client-side manipulation or filtering of user input?

**Criterion:** Objective

**Type:** Binary

**Compliance:** Hard: Requires JavaScript programming knowledge

1. Pull up HTML document containing form
2. Save HTML document and open in editor
3. Search for keyword <SCRIPT>
4. If found, search for keywords like “check”, “validation”, “cookie”, “compliance”, “verification” etc.
5. Read the code section to see whether manipulation is possible.
6. User input validation should be done server-side. If it is not, web server fails the audit.

### ***Suggested improvement to checklist***

Rhoades checklist is systematic and comprehensive. It has one blind spot, however: It neglects auditing the layers below the Session layer. The black highlighted layers are audited directly; the red highlighted ones are not [Figure 1]:

HTTP, FTP	Application Layer	Application Layer Presentation Layer Session Layer
TCP, UDP	Transport Layer	Transport Layer
IP	Internet Layer	Network Layer
Ethernet, PPP	Network Access Layer	Data Link Layer Physical Layer
TCP/IP		OSI

Figure 1: TCP IP Stack

This is an oversight, since denial of service attacks, server crashes, root exploits can be launched against machines with faulty, incomplete, non-standard stack implementation. Therefore, I shall expand the checklist by using two tools:

1. snot (<http://www.geocities.com/sniph00> )
2. ISIC – IP Stack Integrity Checker (<http://www.packetfactory.net/Projects/ISIC> )

to test the TCP /IP stack. Specifically, *snort* parses snort rules ([www.snort.org](http://www.snort.org)) and proceeds to blast the stack of the target machine with packets crafted according to those rules. I will use the default rule base in the snort package which is quite comprehensive. It includes over a thousand rules, representing many types of exploits and attacks (dos, icmp, web-misc, telnet, etc.). While snort is mostly used to test intrusion detection suites, I shall use it here more generically to see whether any of these packets affect the network stack in a negative way.

ISIC consists of five separate executables: *esic* (Ethernet), *isic* (IP), *tcpsic* (TCP), *udpsic* (UDP) and *icmpsic* (ICMP). The great strength of ISIC-generated frames, packets and segments is the *controlled, probabilistic randomization* of flags, header length, fragmentation, options, size, etc. ISIC can be described as an out-of-bound traffic generator. Out-of-bound (or out-of-spec) traffic is network traffic with invalid or unusual options set. See Northcutt for a sampling of such traffic [NOR01, 343-359].

For instance, running *tcpsic -s 1.2.3.4 -d 1.2.3.5 -p 100 -I100 -T30* will send out 100 segments to 1.2.3.5 to random dest ports from 1.2.3.4 with random source port with following header option field distribution:

```
[root@Dan00 /]# tcpsic -s 1.2.3.4 -d 1.2.3.5 -p 100 -I100 -T30
Compiled against Libnet 1.0.2
```

```
Installing Signal Handlers.  
Seeding with 3187  
No Maximum traffic limiter  
Using random source ports.  
Using random destination ports.  
Bad IP Version      = 10%      IP Opts Pcnt      = 100%  
Frag'd Pcnt = 30%      Urg Pcnt      = 30%  
Bad TCP Cksm       = 10%      TCP Opts Pcnt    = 30%  
Wrote 100 packets in 0.02s @ 5777.34 pkts/s
```

This is as close to randomized black box testing of the network stack as you will get.

So the two additional auditing tests I will run are as follows:

### *Network stack auditing with attack packets*

**Point:** Is the BeOS network stack vulnerable to common specific attack packets?  
**Criterion:** Objective  
**Type:** Binary  
**Compliance:** Moderate: Requires installation of snot and some user skills

1. Install snot from [www.geocities.com/sniph00](http://www.geocities.com/sniph00)
2. Assuming destination host *1.2.3.4* and rule file *snorrules.txt* with 1100 rules:  
Execute *snot -r snorrules.txt -d 1.2.3.4 -n 1100*
3. Test if you can:
  - a. Pull up index.html in the top directory
  - b. Execute *cgi-bin/test-cgi*
4. If either test fails, the web server fails the audit

### *Network stack auditing with randomized traffic*

**Point:** Is the BeOS network stack vulnerable to out-of-bounds traffic?  
**Criterion:** Objective  
**Type:** Binary  
**Compliance:** Moderate: Requires installation of ISIC and some user skills

1. Install ISIC from [www.packetfactory.net/Projects/ISIC](http://www.packetfactory.net/Projects/ISIC)
2. Assuming the following:

Host	Source	Destination
IP	1.2.3.5	1.2.3.4
MAC address	00-11-00-00-00-00	00-22-00-00-00-00

3. Execute
  - a. Data Link Layer:

*esic -i eth0 -s 00-11-00-00-00-00 -d 00-22-00-00-00-00 -p 1 -c 10000<sup>15</sup>*

b. Network Layer:

*isic -s 1.2.3.5 -d 1.2.3.4 -p 10000 -F100 -V100 -I100*

c. Transport Layer:

i. *tcpsic -s 1.2.3.5 -d 1.2.3.4 -I100 -T100 -u100 -t100 -p 1000 -m1*

ii. *udpsic -s 1.2.3.5 -d 1.2.3.4 -I100 -F100 -U100 -V100 -m1*

d. Network/Transport Layer<sup>16</sup>:

*icmpsic -s 1.2.3.5 -d 1.2.3.4 -F100 -V100 -I100 -i100 -p 1000*

4. Using a browser, test if you can:

a. Pull up <http://1.2.3.4/index.html> in the top directory

b. Execute <http://1.2.3.4/cgi-bin/test.cgi>

5. If either test fails, the web server fails the audit

I will now move on to assignment 2, the application of the audit techniques to BeOS and RobinHood.

## Conducting the audit

I will audit the following items of my checklist:

1. Network stack auditing with attack packets
  - a. Is the BeOS network stack vulnerable to common specific attack packets?
2. Network stack auditing with randomized traffic
  - a. Is the BeOS network stack vulnerable to out-of-bounds traffic?
3. OS Security
  - a. Are the latest OS service packs / patches installed?
4. Web server security
  - a. Are the latest service packs / patches installed?
  - b. Are default items with known vulnerabilities installed?
5. Web server configuration issues
  - a. Are directories indexable?
  - b. Are interpreters such as perl, java, etc in the web document root path
6. HTTP header

---

<sup>15</sup> There is a bug in *esic*; *-p* actually sets the length, not the protocol type.

<sup>16</sup> Stevens considers ICMP to be part of the network layer. However, the response to ICMP packets may be generated at the network, transport, or application layer [STEV94, 6:69]

- a. Is identifying information about the server, software, etc. unnecessarily returned in the HTTP header?
7. Error Messages
  - a. Are error messages giving back unnecessary information?
8. Caching
  - a. Is data permitted to be cached at the proxy or at the browser?
9. IP Hopping & Session Cloning
  - a. Can the user change IP address in the middle of the session without re-authenticating?
10. Session time-outs
  - a. Are session time-outs supported?

My procedure for the first two tests will be the following: A before/after screenshot of the host running BeOS and RobinHood and a shot of the host doing the testing. The ‘before’ screen shot will be the same for all tests [Figure 2]. The ‘after’ screenshot of the BeOS host will contain at a minimum:

1. The window *Robin Hood Console*, which is the control center for RobinHood. The Console logs any interactions the web server has to stimuli.
2. The window *Terminal 2* shows the output of the netstat command: RobinHood is seen to be running on port 80.
3. Any anomaly I can find.

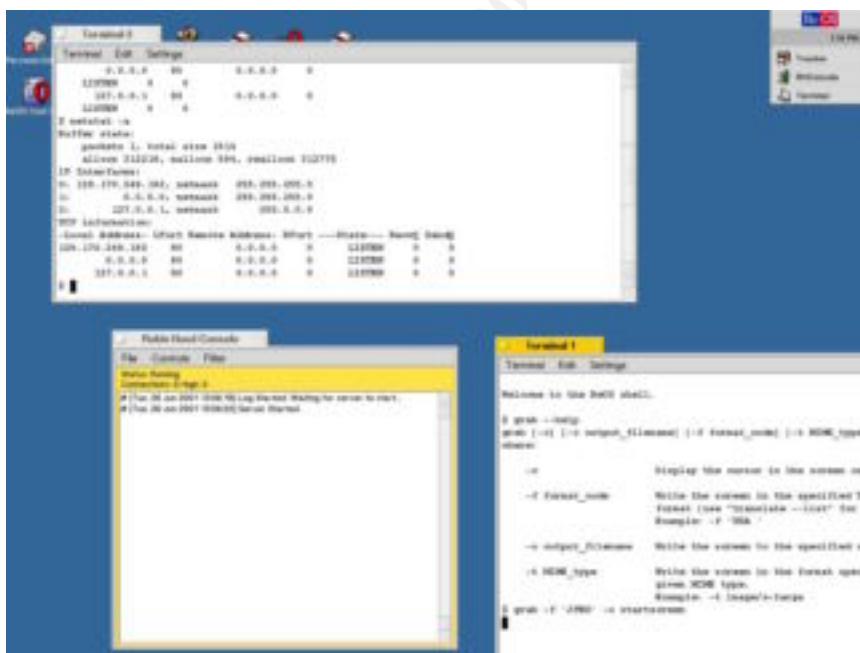


Figure 2: BeOS and Robin Hood initial startup screen

✓ Network stack auditing with attack packets

Executing `snot -r snortrules.txt -d 129.170.249.192 -n 1100` on my NT box [Figure 3 (snipped)] gives me the following output:

```

92:80
TCP - "FTP satan scan" - 131.194.202.102:15146 -> 129.170.249.192:21
TCP - "WEB-IIS exec-src access" - 94.149.97.129:7474 -> 129.170.249.192:80
TCP - "WEB-MISC moreover shopping cart directory traversal" - 82.55.12.70:23359
-> 129.170.249.192:80

G:\Program Files\snot>snot -r snortrules.txt -d 129.170.249.192 -n1100
snot 0.91 (alpha) by sniph (sniph80@yahoo.com)

-----
Rulefile           : snortrules.txt
Source Address     : Use rules file
Dest Address       : 129.170.249.192
Number of Packets  : 1100
Delay (max seconds): No Delay
Interface          : 0

[Parse Rules - Completed parsing 1866 rules - Sending now]

ICMP - "ICMP Information Request (Undefined Code!)" - 199.54.65.246 -> 129.170.2
49.192
^C
G:\Program Files\snot>

```

Figure 3: Test 1 tester screen: snot execution on NT

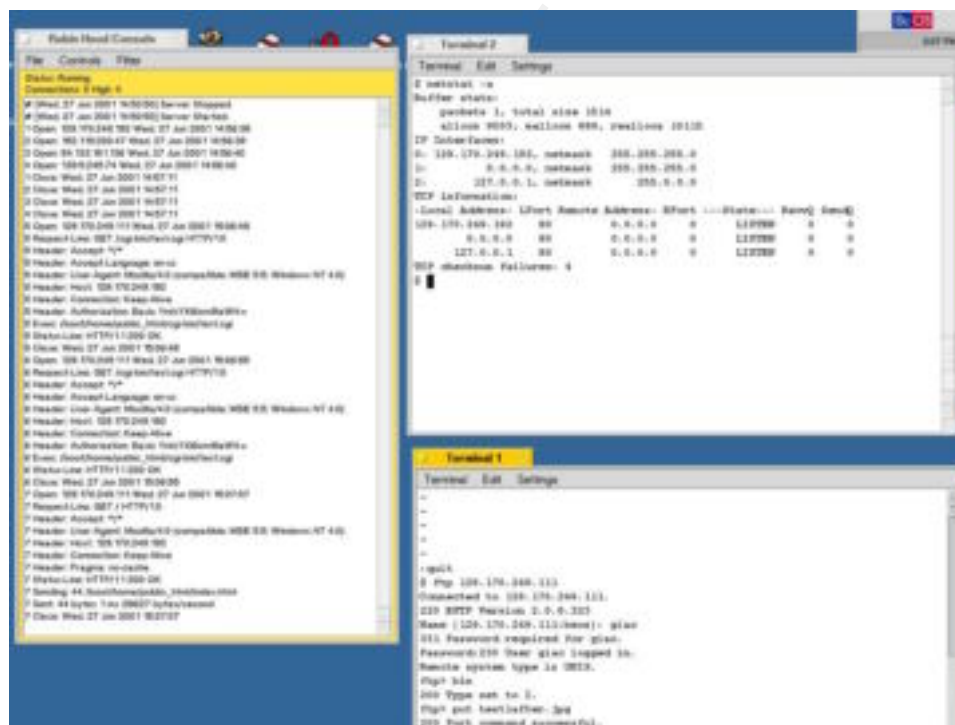


Figure 4: Test 1 'after' screen shot



From the output on the browser (not shown) and the output on the Console, both `index.html` and `cgi-bin/test.cgi` were properly display. The web server **passes** this test.

× **Network stack auditing with random traffic**

A. Executing `esic -I eth0 -d 00-e0-29-74-dd-90 -p 1 -c10000` on my Linux box [Figure 5] gives me the output shown below [Figure 6].

From the output on the browser (not shown) and the output on the Console, both `index.html` and `cgi-bin/test.cgi` were properly display. The web server **passes** this subtest.

B. Executing `isic -s 129.170.249.122 -d 129.170.249.192 -p10000 -F100 -V100 -I100` on my Linux box [Figure 7] gives me the output shown below [Figure 8].

From the output on the browser (not shown) and the output on the Console, both `index.html` and `cgi-bin/test.cgi` were properly display. The web server **passes** this subtest.

C1. Executing `tcpsic -s 129.170.249.122 -d 129.170.249.192 -I100 -T100 -u100 -t100 -p1000 -m10` on my Linux box [Figure 9] gives me the output shown below [Figure 10, Figure 11].

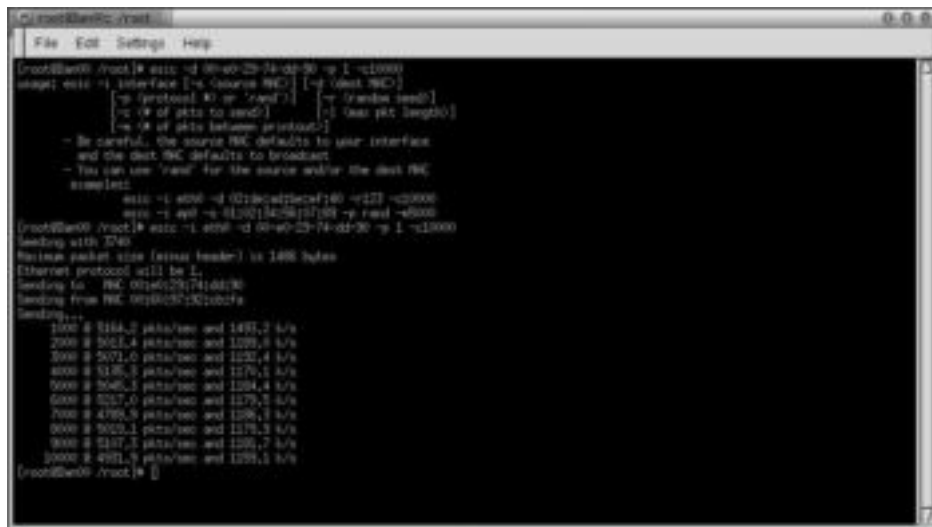
From the output on the browser (not shown, but a 404 error both times) and the output on the Console, neither `index.html` nor `cgi-bin/test.cgi` were properly display. Furthermore, the output screenshots show that the `net_server` [Figure 10], specifically the `ether_reader` [Figure 11], has crashed. The host is unreachable and requires a reboot. The web server **fails** this subtest

C2. Executing `udpsic -s 129.170.249.122 -d 129.170.249.192 -I100 -F100 -U100 -V100 -m10` on my Linux Box [Figure 12] gives me the output shown below [Figure 13].

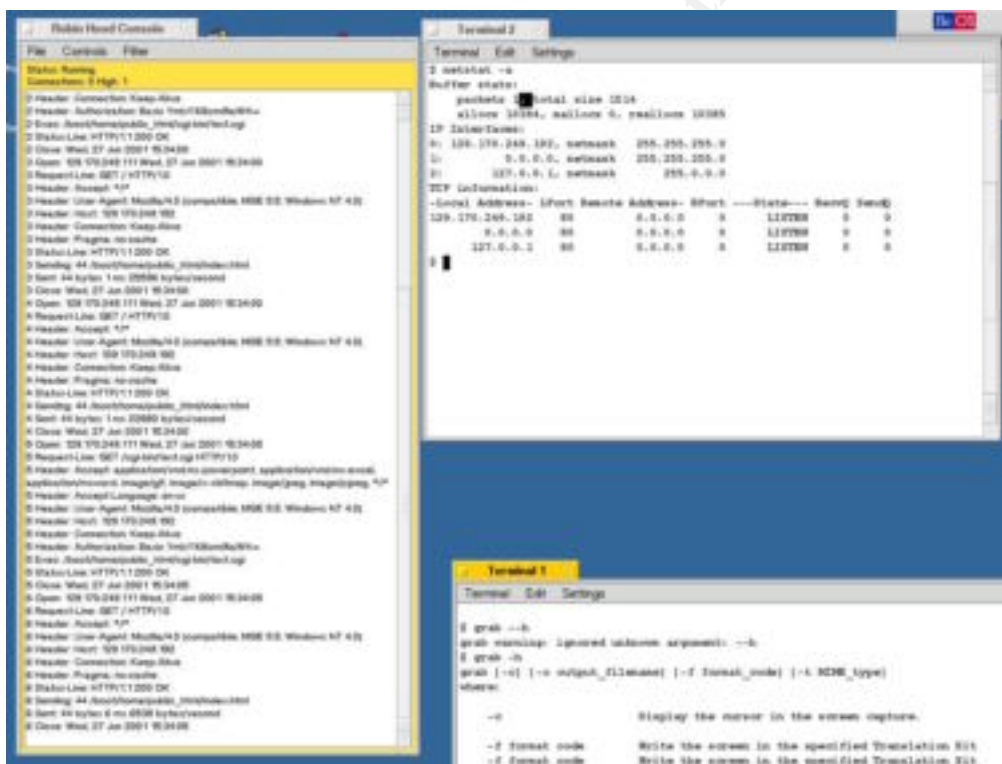
From the output on the browser (not shown) and the output on the Console, both `index.html` and `cgi-bin/test.cgi` were properly display. The web server **passes** this subtest.

D. Executing `icmpsic -s 129.170.249.122 -d 129.170.249.192 -F100 -V100 -I100 -i100 -p1000` on my Linux Box [Figure 14] gives me the output shown below [Figure 15].

From the output on the browser (not shown) and the output on the Console, both `index.html` and `cgi-bin/test.cgi` were properly display. The web server **passes** this subtest.



**Figure 5: Test 2 (esic) tester**



**Figure 6: Test 2 (esic) 'after' screen shot**

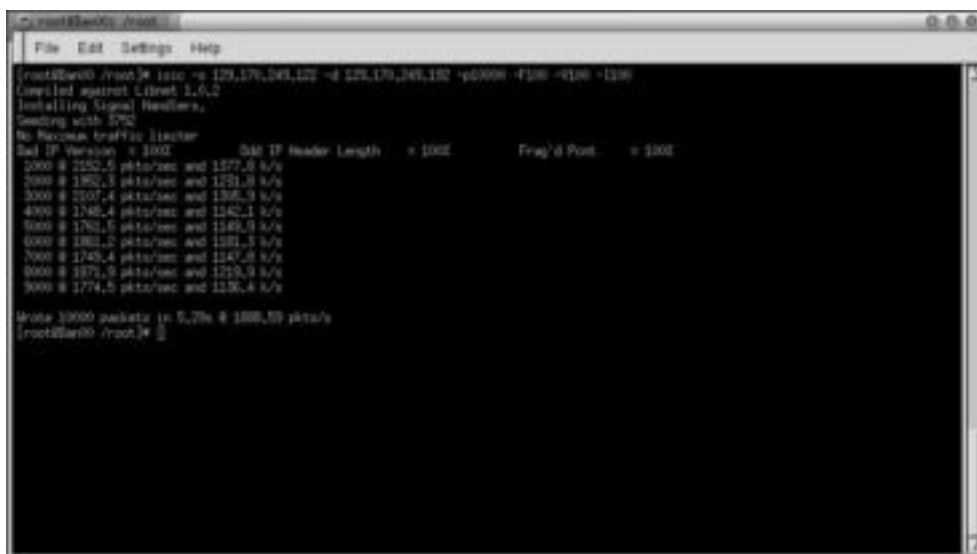


Figure 7: Test 2 (isic) tester screen shot

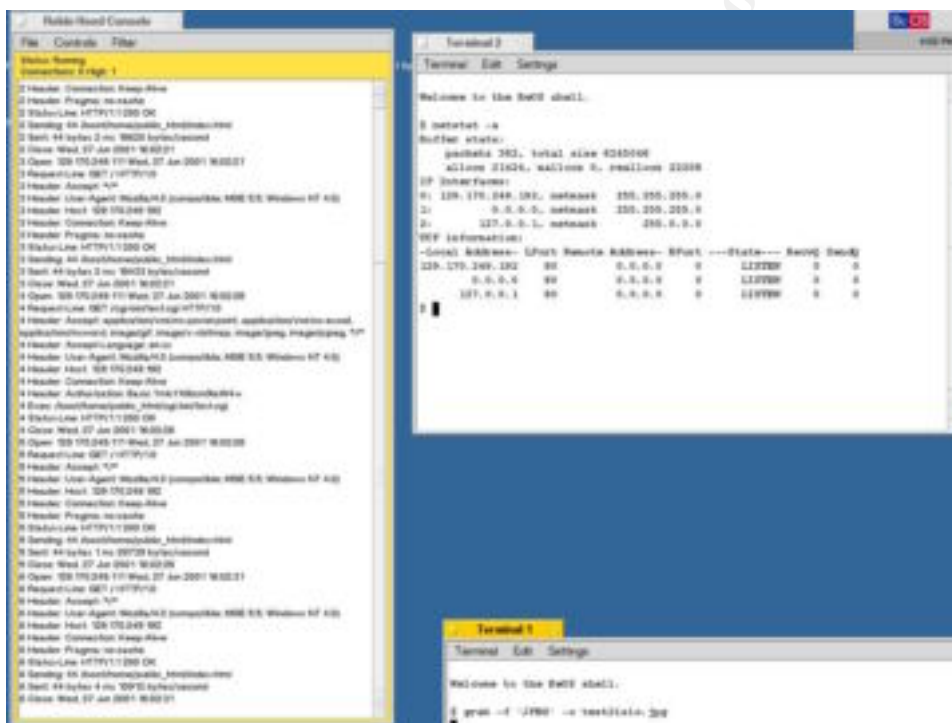


Figure 8: Test 2 (isic) 'after' screen shot

```

root@lan00: /root# ./tcpsic
File Edit Settings Help
[root@lan00 /root]# tcpsic -s 129,170,249,122 -d 129,170,249,132 -i100 -t100 -d90 -t100 -p1000 -m10
Compiled against Libnet 1.0.2
Installing Signal Handlers.
Seeding with 1232
Maximum traffic rate = 10.00 k/s
Using random source ports.
Using random destination ports.
Bad IP Version = 100%      IP Opt Port = 100%
Frag'd Port = 100%      Urg Port = 100%
Bad TCP Class = 100%      TCP Opt Port = 100%

Wrote 1000 packets in 19.37s @ 50.07 p/s
[root@lan00 /root]#
    
```

Figure 9: Test 2 (tcpsic) tester screen shot

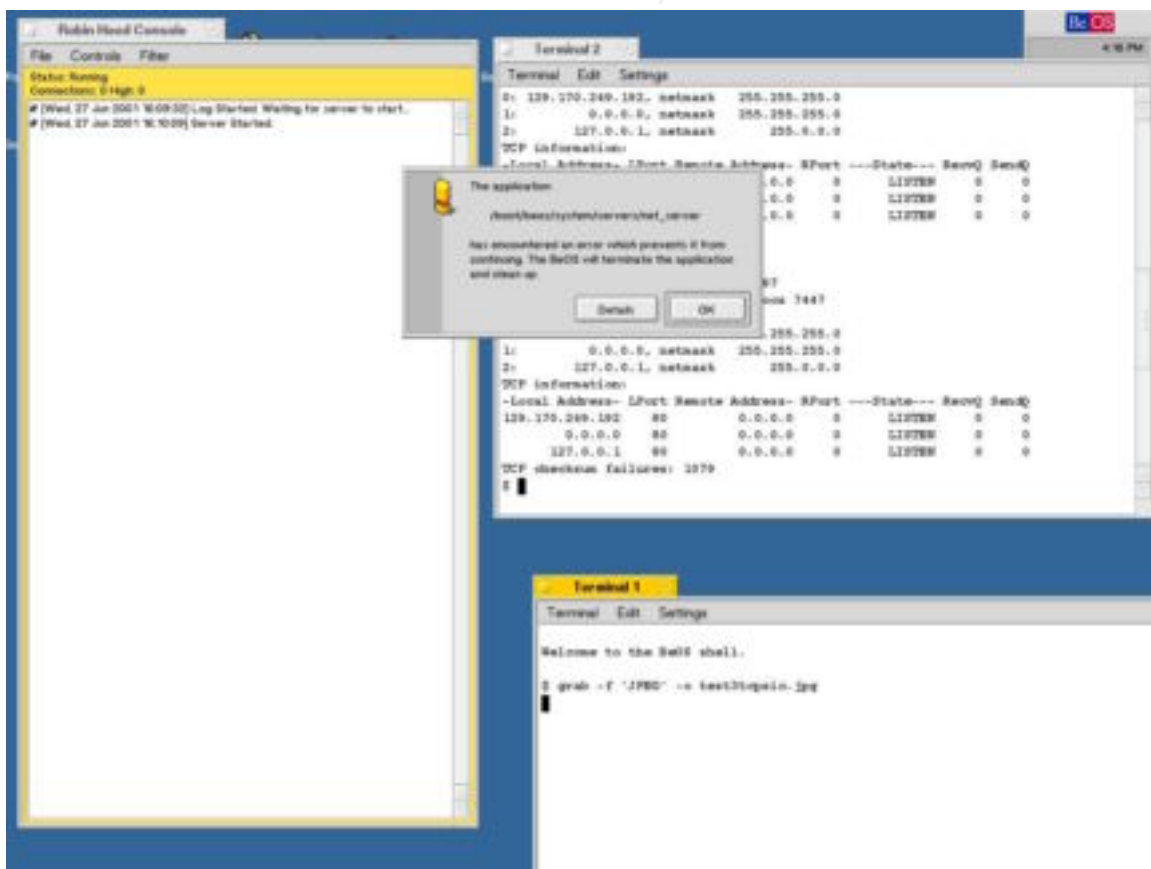


Figure 10: Test 2 (tcpsic) 'after' screen shot 1

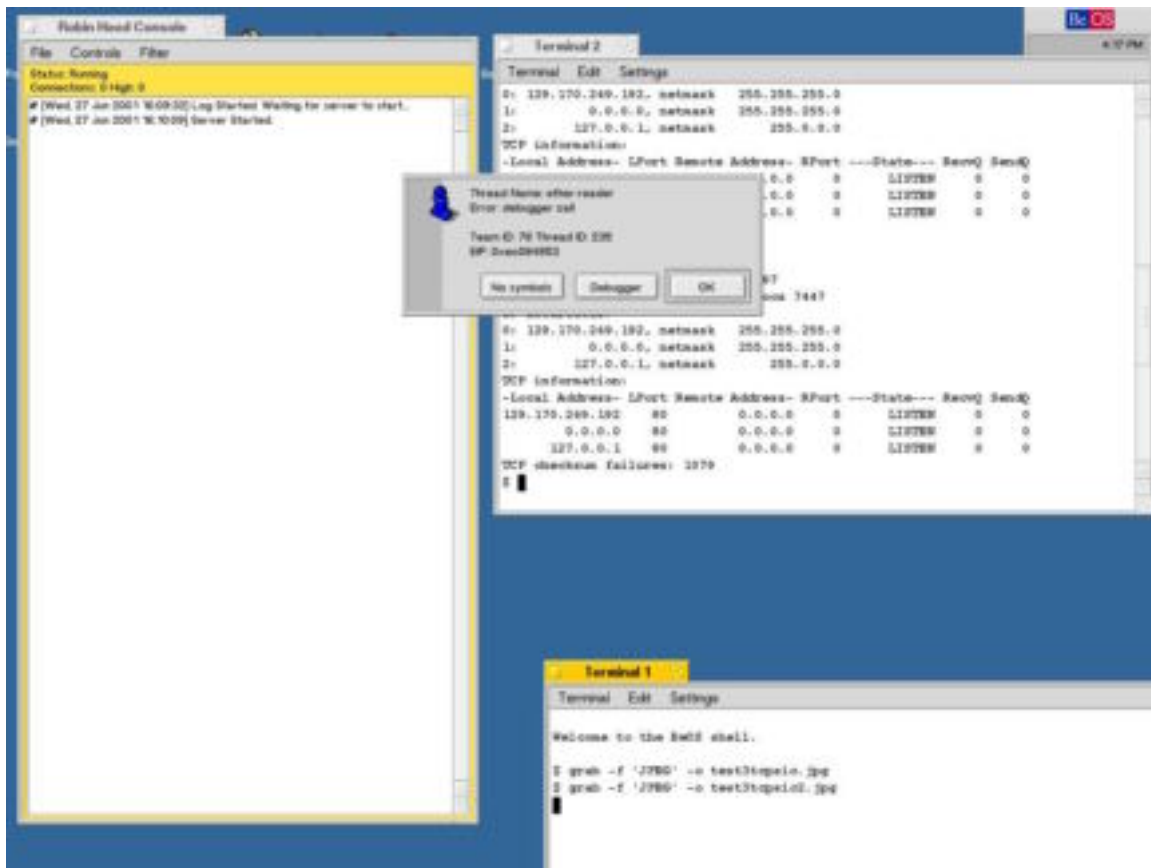


Figure 11: Test 2 (tcpinfo) 'after' screenshot test 2

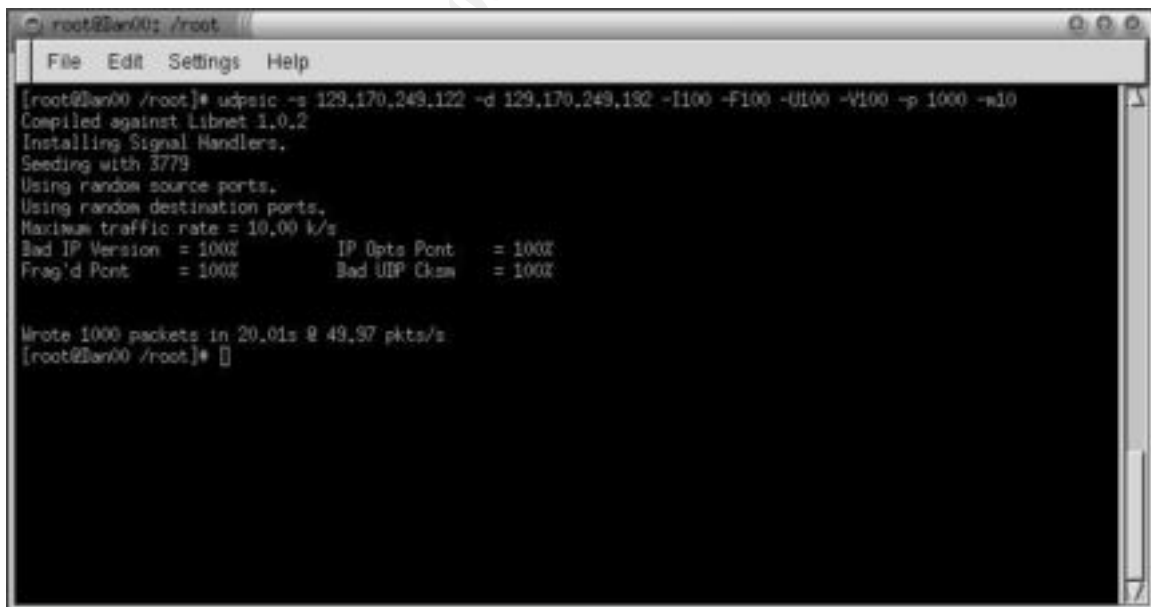


Figure 12: Test 2 (udpsic) tester screen shot

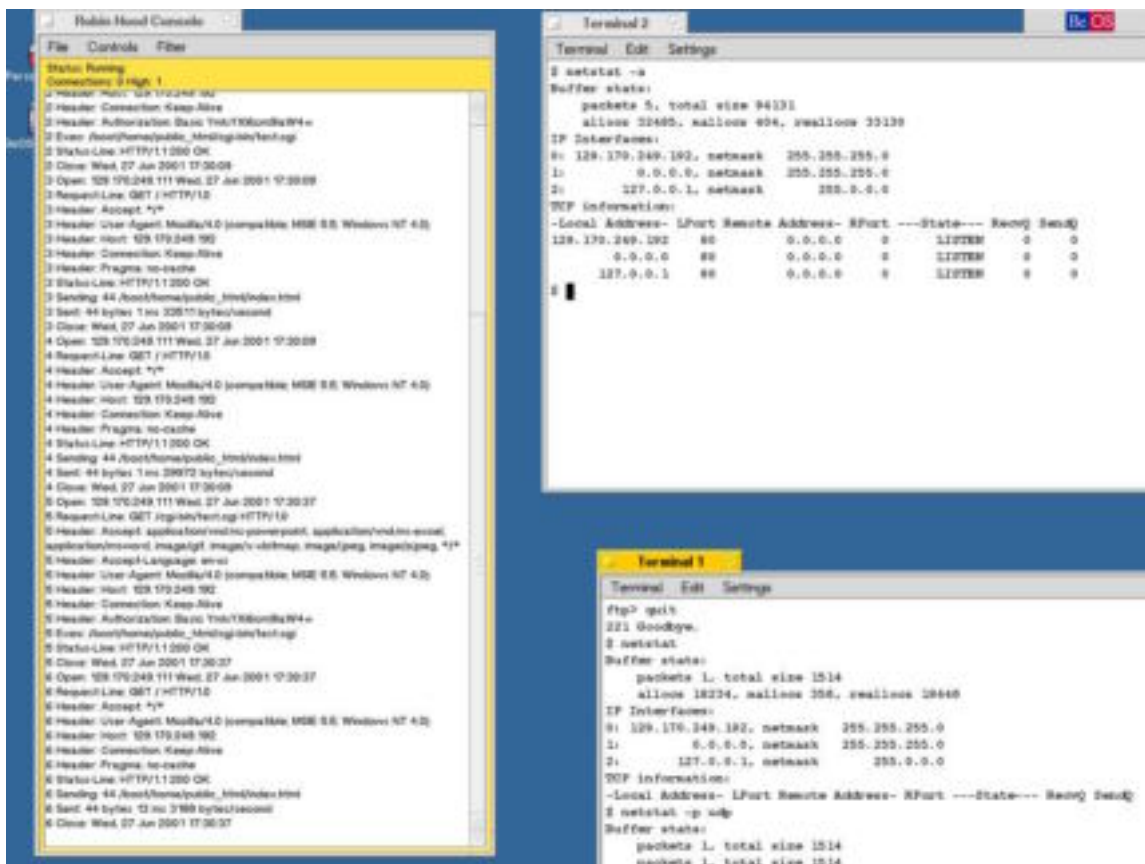


Figure 13: Test 2 (udpsic) 'after' screen shot

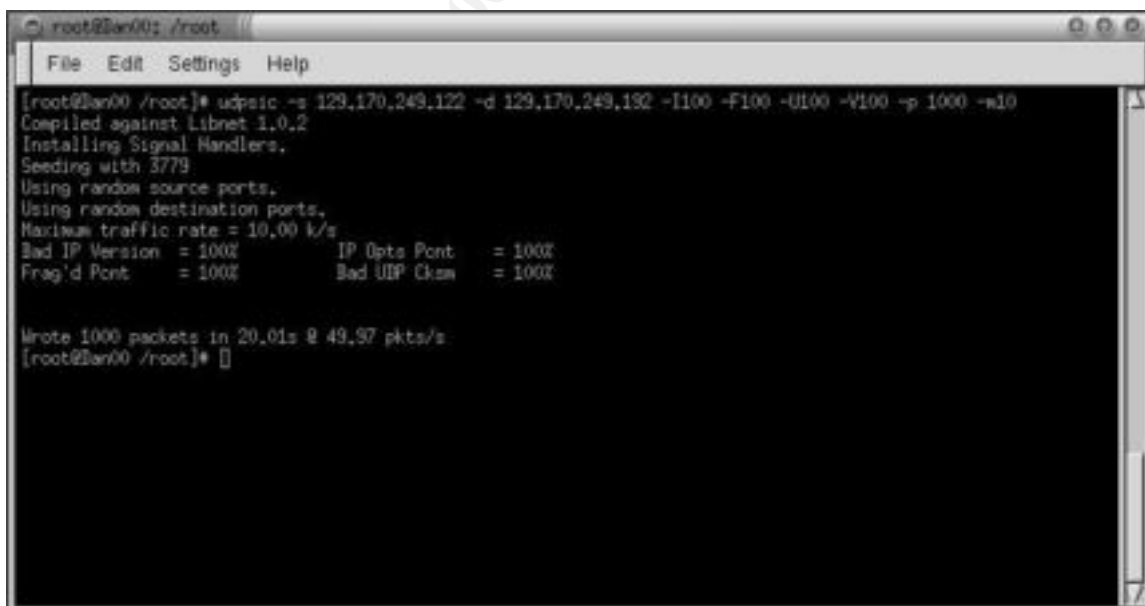


Figure 14: Test 2 (icmppsic) tester screen shot

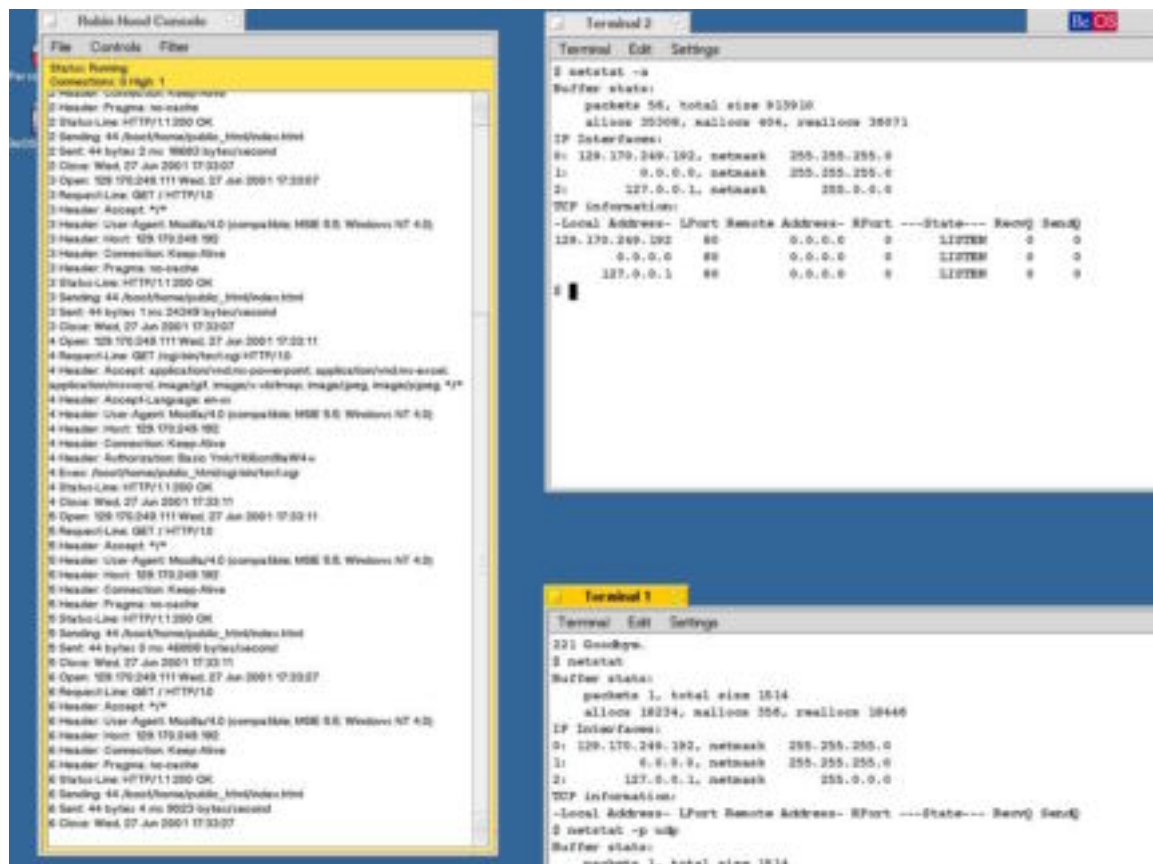


Figure 15: Test 2 (icmcsic) 'after' screen shot

### ✓ OS Security

1. Executing `uname -a` gives me *BeOS 5.0 10000009 BePC*.
2. Checking at <http://www.be.com/support/updates/index.html> : Latest version is BeOS 5.0.3 and there is a FTP server vulnerability in BeOS 5.0
3. Double-checking at
  - a. [icat.nist.gov/icat.cfm](http://cat.nist.gov/icat.cfm) for FTP server vulnerability severity: nothing found
  - b. [www.securityfocus.com](http://www.securityfocus.com) for FTP server vulnerability severity: nothing found
  - c. [www.cert.org](http://www.cert.org) for FTP server vulnerability severity: nothing found
4. Since it is not obviously a root exploit, the OS **passes** the audit. However, the patch should be applied as soon as possible.

### ✓ Web Server Security

#### A. Server weakness

1. Reading Robin Hood documentation – version is 1.1 10/5/99
2. Checking at
  - a. [icat.nist.gov/icat.cfm](http://cat.nist.gov/icat.cfm) : Buffer overflow found [Figure 16]



- b. [www.securityfocus.com](http://www.securityfocus.com): Buffer overflow vulnerability found at <http://www.securityfocus.com/frames/?content=/vdb/bottom.html%3Fvid%3D1944>
    - c. [www.cert.org](http://www.cert.org): nothing found
  3. Since this is not a root exploitable, the web server **passes** this test.

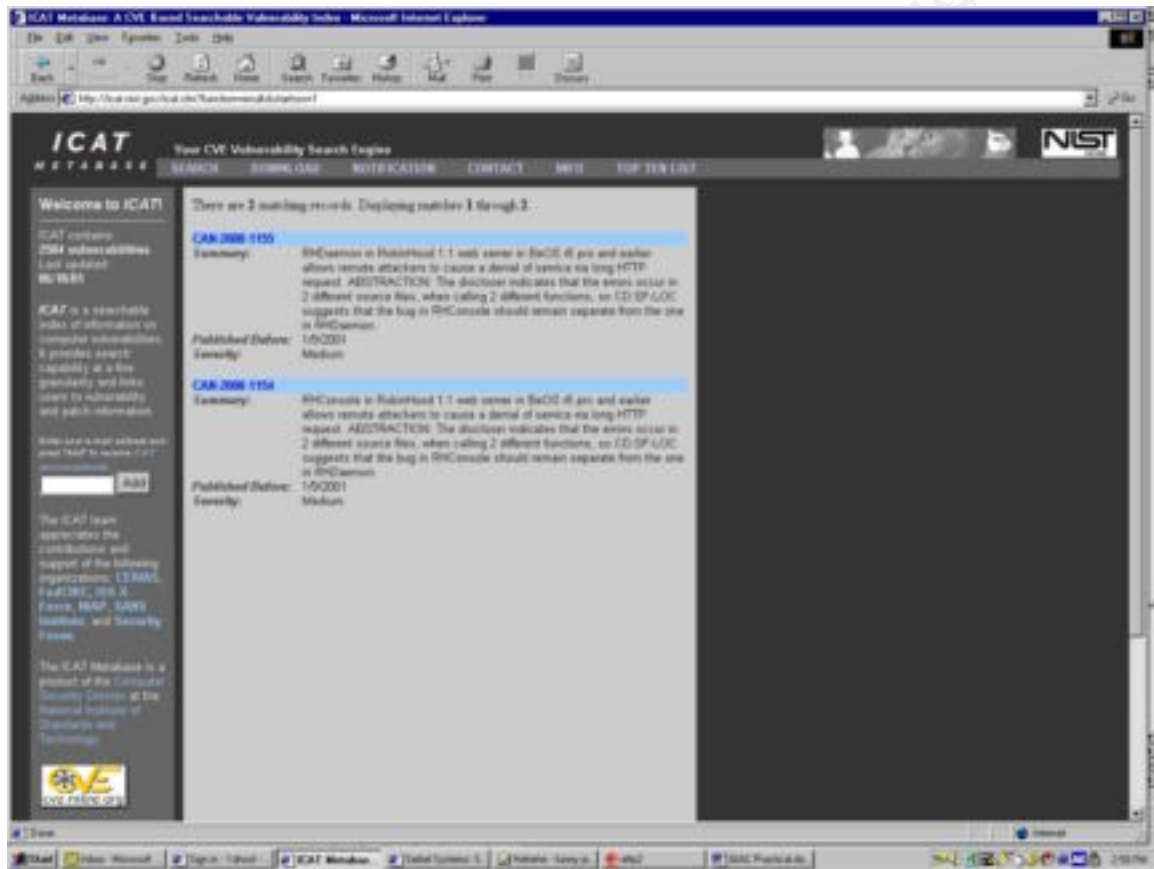


Figure 16: Test 4 ICAT search screen

## B. Default vulnerabilities

1. Checking Robin Hood 1.1 default vulnerabilities at
  - a. [icat.nist.gov/icat.cfm](http://icat.nist.gov/icat.cfm) : Buffer overflow found [Figure 16]
  - b. [www.securityfocus.com](http://www.securityfocus.com): Buffer overflow vulnerability found at <http://www.securityfocus.com/frames/?content=/vdb/bottom.html%3Fvid%3D1944>
  - c. [www.cert.org](http://www.cert.org): nothing found
2. After reading the vulnerability report, the bugs are on source code level in essential modules. The web server would not work without those modules, so it will not do any good to disable them.
3. Since these are not root exploitable, the web **server** passes the test.



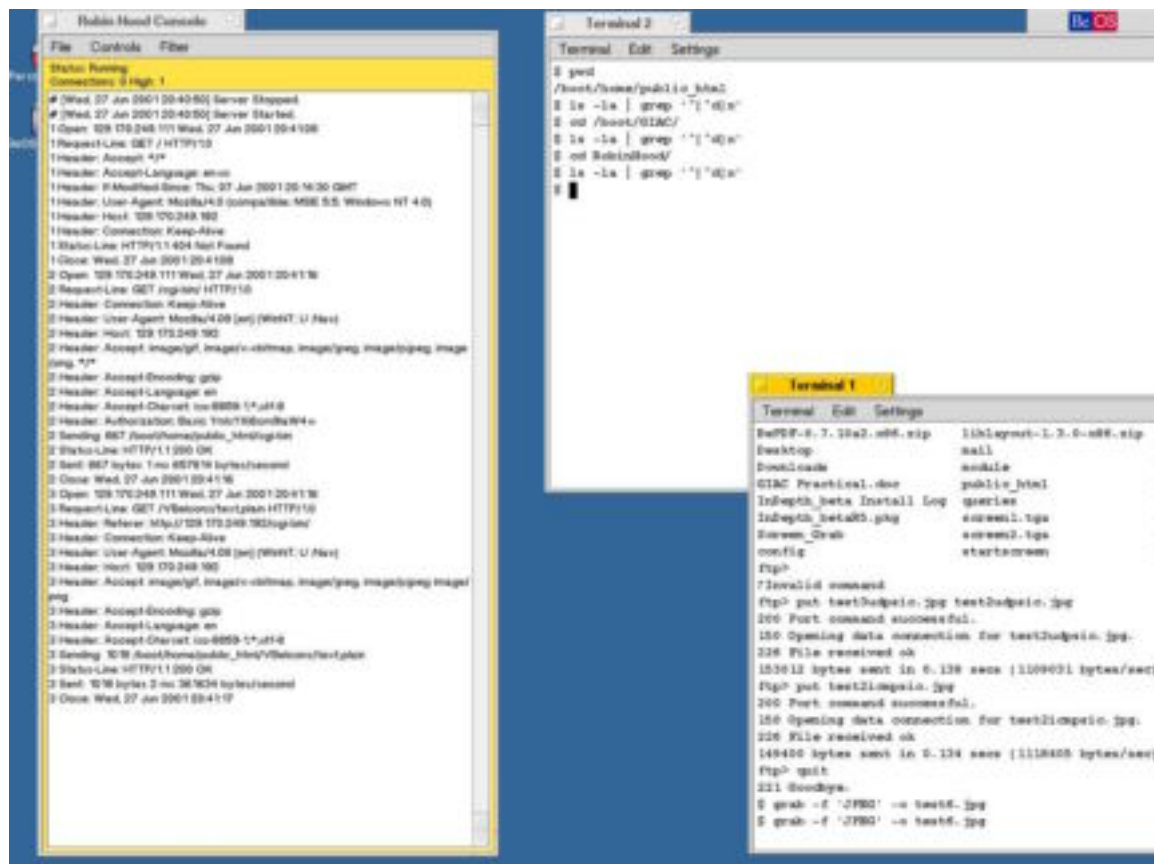
× **Web server configuration issues**

A. Callable Interpreters

1. Executing `ls -la | grep '[^/]x'` in web root directory `public_html`
2. No executable was found [Figure 17, Terminal 2]. The web server **passes** this subtest.

B. Indexable directory

1. Renamed `index.html` to `index.html.bak` in web root directory `public_html`
2. Called up
  - a. <http://129.170.249.192>
  - b. <http://129.170.249.192/cgi-bin>
3. The request was denied for 2a [Figure 17, Console request 1]. The request was granted and directory listing returned for 2b [Figure 17, Console request 2-3].
4. Since the directory of the cgi-bin directory was returned (after successful authentication), the web server **fails** the test [Figure 18].



```
Robin Hood Console
File Controls Filter
Status: Running
Connections: 9 High: 1
# [Wed, 27 Jun 2001 20:40:50] Server Stopped
# [Wed, 27 Jun 2001 20:40:50] Server Started
1 Open: 129.170.248.111 Wed, 27 Jun 2001 20:41:08
1 Request-Line: GET / HTTP/1.0
1 Header: Accept */*
1 Header: Accept-Language: en-us
1 Header: If-Modified-Since: Thu, 27 Jun 2001 20:16:30 GMT
1 Header: User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 4.0)
1 Header: Host: 129.170.248.100
1 Header: Connection: Keep-Alive
1 Status-Line: HTTP/1.1 404 Not Found
1 Close: Wed, 27 Jun 2001 20:41:08
2 Open: 129.170.248.111 Wed, 27 Jun 2001 20:41:16
2 Request-Line: GET /cgi-bin/ HTTP/1.0
2 Header: Connection: Keep-Alive
2 Header: User-Agent: Mozilla/4.0 (en) (WinNT; U; Next)
2 Header: Host: 129.170.248.100
2 Header: Accept: image/gif, image/x-bitmap, image/png, image/jpeg, image
2 Header: Accept-Encoding: gzip
2 Header: Accept-Language: en
2 Header: Accept-Charset: iso-8859-1*,utf-8
2 Header: Authorization: Basic TmlhRG80bmR4W4=
2 Sending: 867 /root/home/public_html/cgi
2 Status-Line: HTTP/1.1 200 OK
2 Sent: 867 bytes 1 ms 857816 bytes/sec
2 Close: Wed, 27 Jun 2001 20:41:16
3 Open: 129.170.248.111 Wed, 27 Jun 2001 20:41:16
3 Request-Line: GET /VBrowsers/testplan HTTP/1.0
3 Header: Referer: http://129.170.248.100/cgi-bin/
3 Header: Connection: Keep-Alive
3 Header: User-Agent: Mozilla/4.0 (en) (WinNT; U; Next)
3 Header: Host: 129.170.248.100
3 Header: Accept: image/gif, image/x-bitmap, image/png, image/jpeg, image
3 Header: Accept-Encoding: gzip
3 Header: Accept-Language: en
3 Header: Accept-Charset: iso-8859-1*,utf-8
3 Sending: 1018 /root/home/public_html/VBrowsers/testplan
3 Status-Line: HTTP/1.1 200 OK
3 Sent: 1018 bytes 2 ms 361634 bytes/sec
3 Close: Wed, 27 Jun 2001 20:41:17

Terminal 2
Terminal Edit Settings
$ pwd
/root/home/public_html
$ ls -la | grep "\.gif$"
$ cd /root/GIAC/
$ ls -la | grep "\.gif$"
$ cd RobinHood/
$ ls -la | grep "\.gif$"
$

Terminal 1
Terminal Edit Settings
testplan-1.7.12a2.m86.zip liblognet-1.3.0-m86.zip
backtop mail
download module
GIAC Practical.doc public_html
InDepth_beta Install Log queries
InDepth_beta05.jpg screen1.jpg
Screen_Grab screen2.jpg
config startscreen
ftp>
?Invalid command
ftp> put testbudpin.jpg testbudpin.jpg
200 Port command successful.
150 Opening data connection for testbudpin.jpg.
226 File received ok
153612 bytes sent in 0.138 secs (1109431 bytes/sec)
ftp> put test2imgpin.jpg
200 Port command successful.
150 Opening data connection for test2imgpin.jpg.
226 File received ok
149400 bytes sent in 0.134 secs (1118405 bytes/sec)
ftp> quit
221 Goodbye.
$ grab -f '2F80' -o test5.jpg
$ grab -f '2F80' -o test6.jpg
```

Figure 17: Test 5: Configuration issues

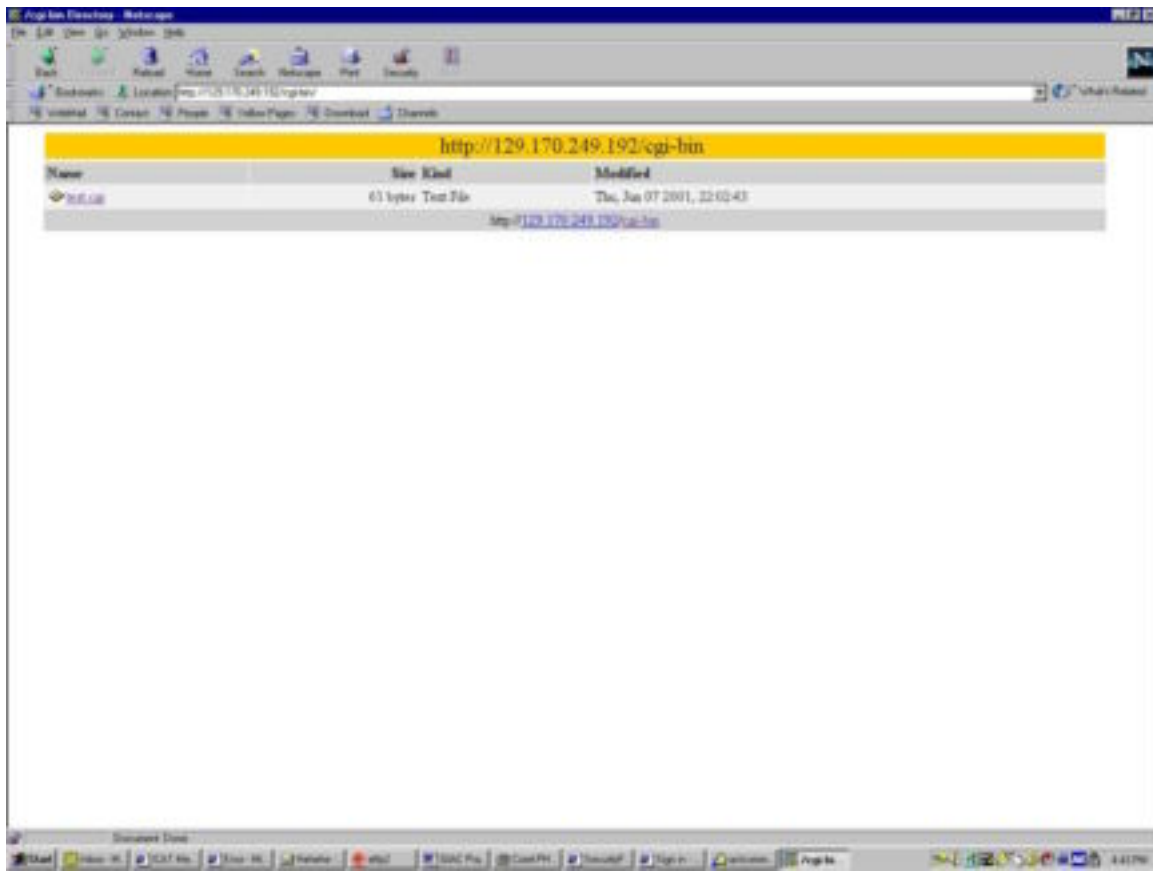


Figure 18: Test 5: indexable directory

× HTTP header

1. Called up *index.html*
2. Viewed the session in a packet sniffer and the server field was RobinHood [Figure 19].
3. Too much information was revealed: The web server **fails** this test.

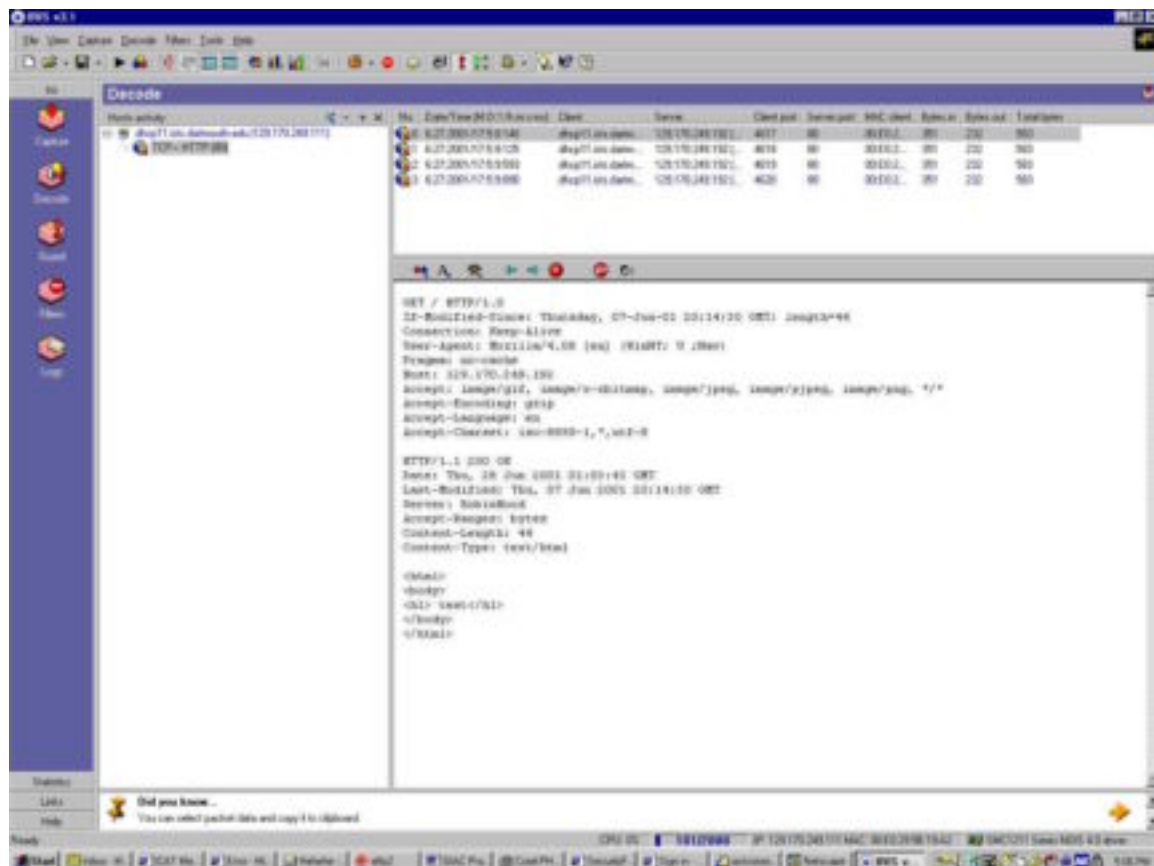


Figure 19: Test 6: HTTP header

### ✓ Error messages

1. Ran exploit found at <http://www.securityfocus.com/frames/?content=/vdb/bottom.html%3Fvid%3D1944>
2. The web server did not crash – it even detected the buffer overflow exploit [Figure 20].
3. Called cgi-bin/test.cgi and authenticated with >5000 byte input
4. Server crashed, but no revealing error messages returned [Figure 21].
5. For now and for this limited black-box testing, the web server **passes** this test.

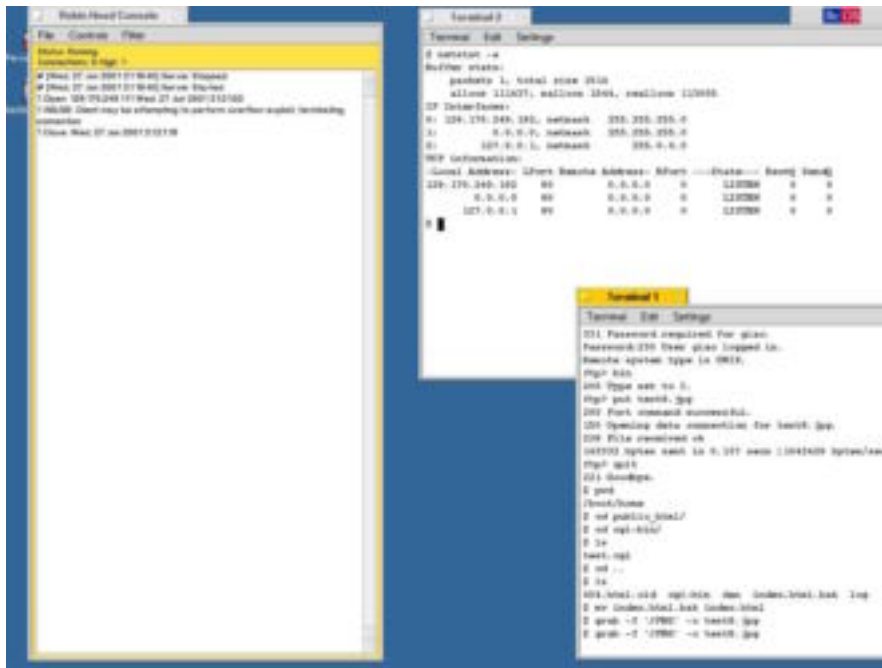


Figure 20: Test 8: failed GET buffer overflow exploit attempt

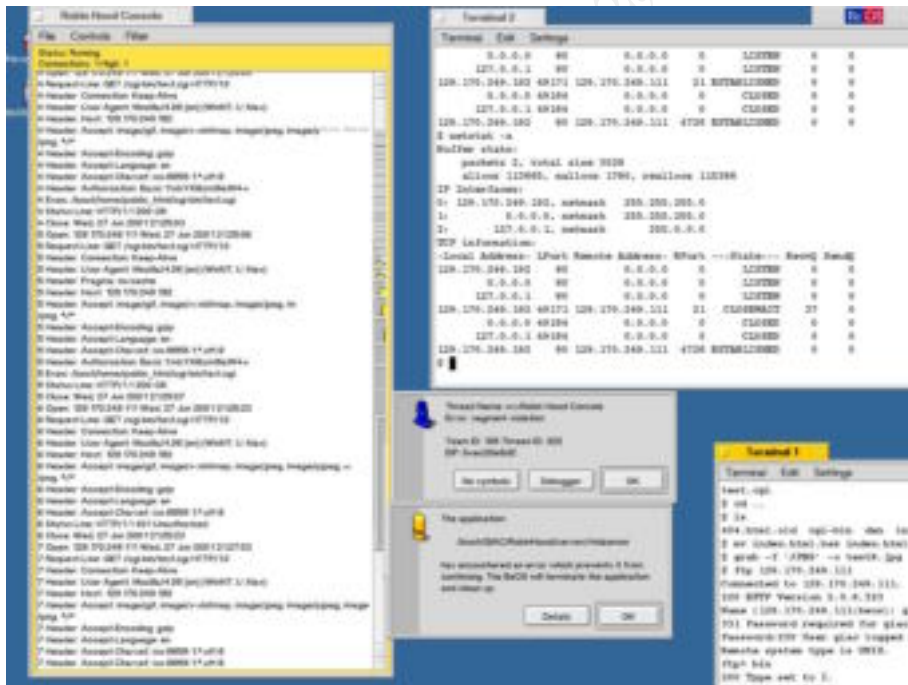


Figure 21: Test 8: Authentication buffer overflow success

## × Caching

### 1. Run packet sniffer

2. execute *telnet 129.170.249.192 80*
3. Type GET / HTTP/1.0 , then press ENTER twice
4. Type GET / HTTP/1.1, then press ENTER twice
5. Call cgi-bin/test.cgi and authenticate.
6. Go to a different web site
7. Call cgi-bin/test.cgi again and see if you have to re-authenticate
8. Since you do not have to re-authenticate [Figure 24], the web server **fails** the test.  
As an aside: The server does not support HTTP 1.1, only HTTP 1.0 [Figure 23].  
No *Expires* field is set in HTTP 1.0 [Figure 22]. This means that data is cached until the browser is closed.

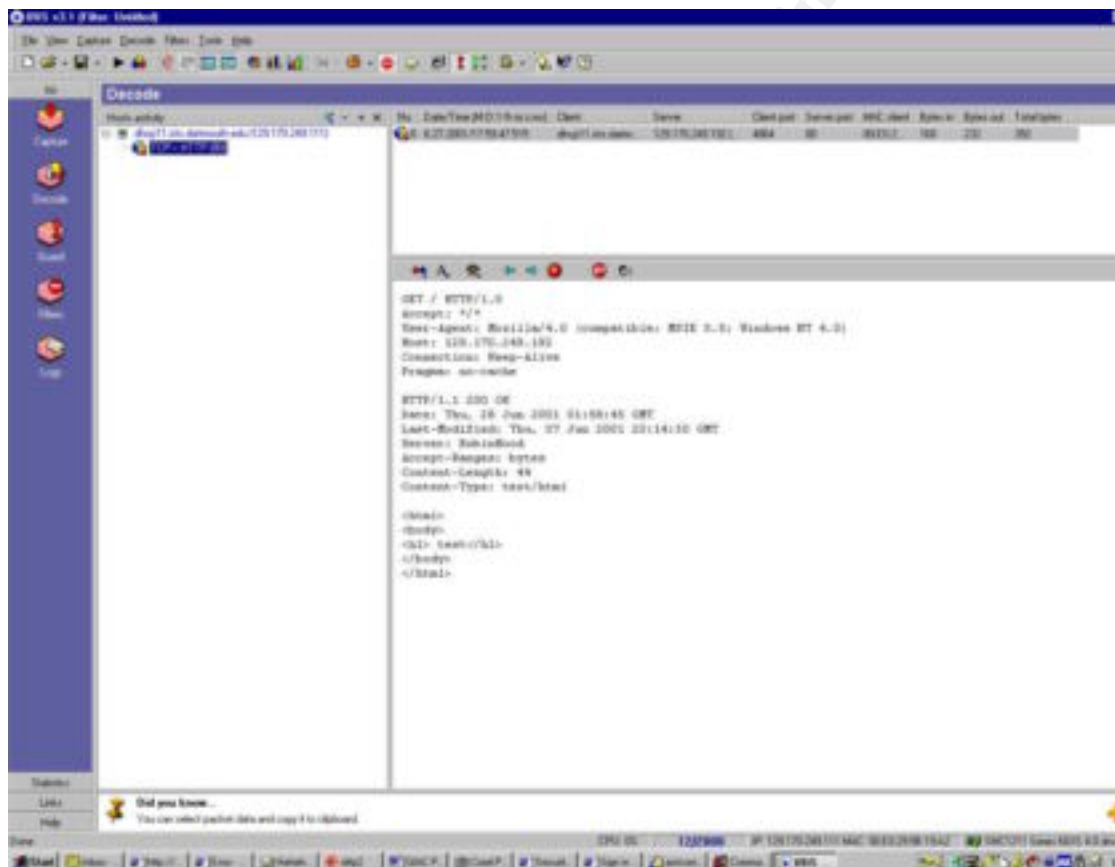


Figure 22: Test 8: HTTP 1.0 request

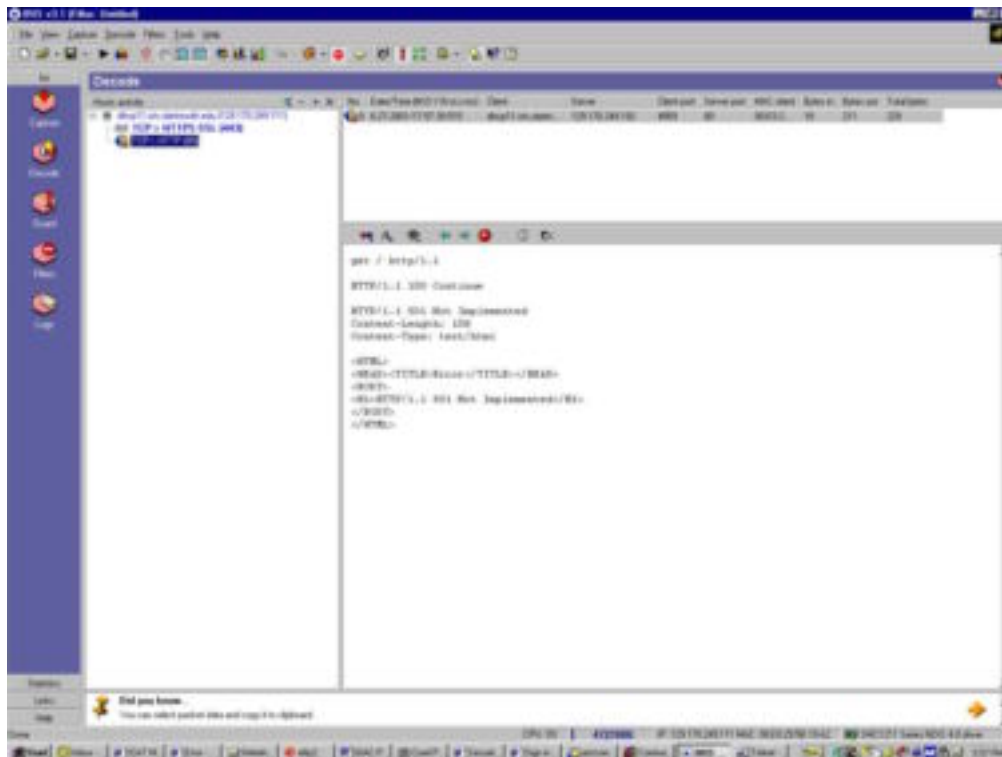


Figure 23: Test 9: HTTP 1.1 request

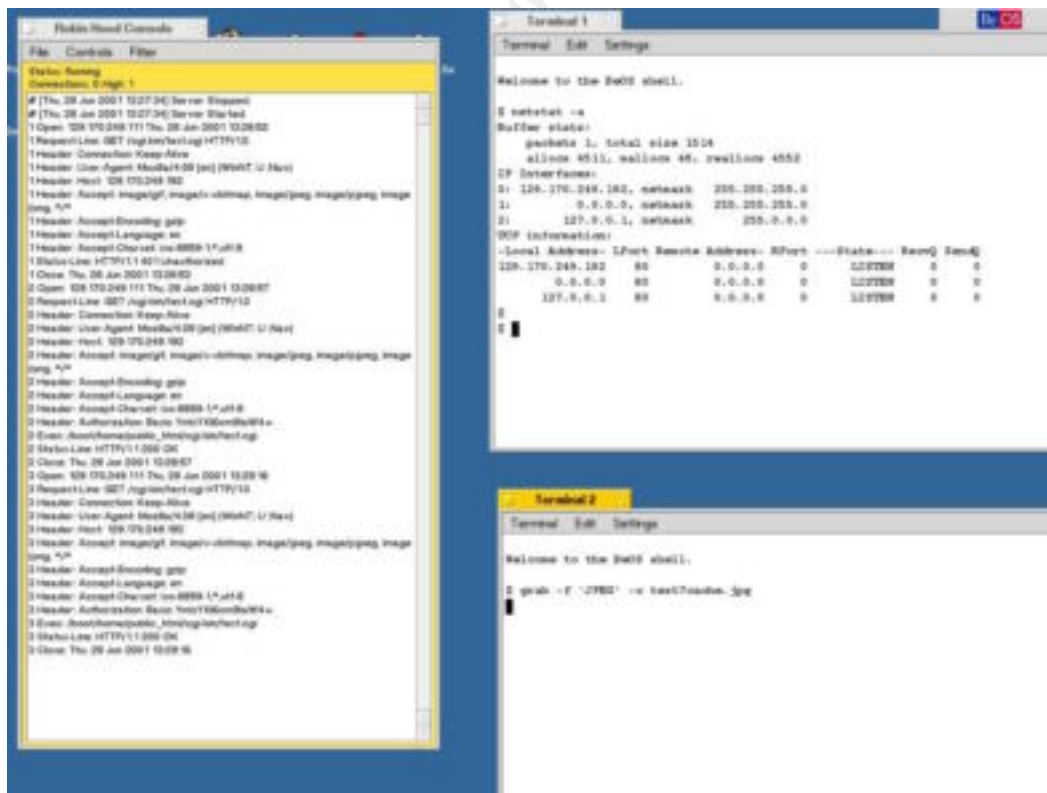


Figure 24: Test 9: no re-authentication needed

× IP hopping

1. Call *cgi-bin/test.cgi* and authenticate
2. Change the IP configuration
3. Call *cgi-bin/test.cgi* again
4. The page was displayed even though the IP address changed between transactions [Figure 25, request 2 line 1, request 3 line 1]. The web server **fails** this test.

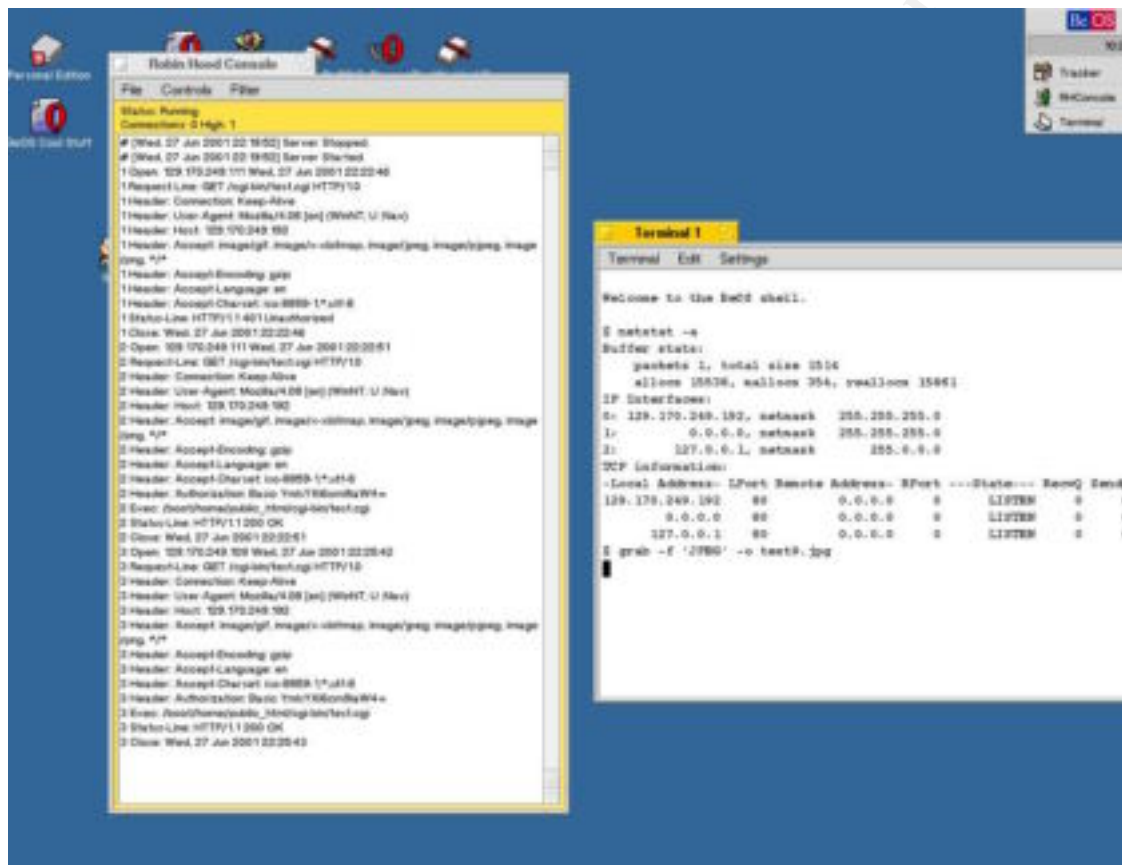


Figure 25: Test 9: IP hopping success

× Session time-outs

1. Pull up *cgi-bin/test.cgi*
2. Drink some tea for twelve minutes
3. Call *cgi-bin/test.cgi* again
4. The page is displayed, no session time out detected [Figure 26]. The web server **fails** this test.



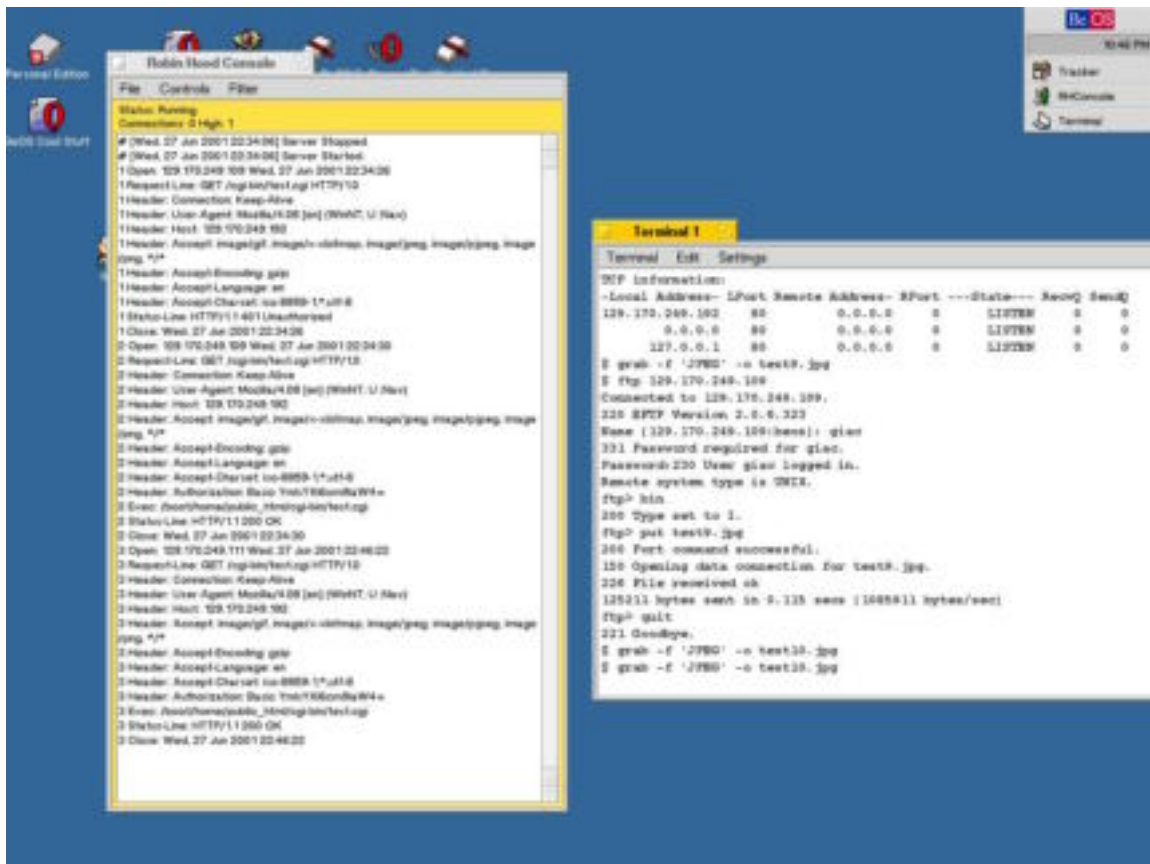


Figure 26: Test 10: Session time-out did not occur

## Evaluating the audit

The score card for Robin Hood 1.1 is passing 4 out of 10 test fields:

Network stack auditing with attack packets	PASSED
Network stack auditing with randomized traffic	FAILED
OS Security	PASSED
Web server security	PASSED
Web server configuration issues	FAILED
HTTP header	FAILED
Error Messages	PASSED
Caching	FAILED
IP Hopping & Session Cloning	FAILED
Session time-outs	FAILED

Table 3: RobinHood auditing scorecard

I will briefly comment on the specific audit points, then on the audit in general :

**Network stack auditing with attack packets:** Most of these attacks were designed to audit intrusion detection systems and for specific exploits against specific applications. This test is useful for popular systems, but of limited value to BeOS and RobinHood. Nevertheless, there are some generic attack packets that are applicable, as well. For systems such as Linux with active ipchains<sup>17</sup>, however, this is a very good test, since it will pin point the holes in the firewall / filtering configuration.

**Network stack auditing with random traffic:** This is a very useful test since it is a practical example of bona-fide black-box testing. Auditing the integrity of the network stack (transport layer and below) is a neglected subject that can smoke out non-standard, buggy implementations. The application and session layers are not tested, though. This should be improved upon, as I will elaborate further in directions for future work. I would recommend this test for all systems, as it will yield interesting insights.

**OS Security:** This is a standard test that should be done routinely (i.e. check if latest patch is applied). The limitation is that you do not have any guarantees that the patch is not buggier than its predecessor. For popular OSs, there are very thorough audits<sup>18</sup> and hardening scripts<sup>19</sup> available.

**Web server security:** Like the OS security test, this is a standard test that should be done routinely (i.e. check if latest patch is applied). The limitation is that you do not have any guarantees that the patch is not buggier than its predecessor. For popular web servers, there are thorough audits<sup>20</sup> and hardening scripts<sup>21</sup> available. For an excellent explanation of general issues, see Stein [STEIN00, chapter 10].

**Web server configuration:** The search for executable interpreters is a valuable test that should be done with some care. If an interpreter is found, much malign activity is possible. Indexable directories offer too much information, since many executable scripts have vulnerabilities in them that allow for machinations that were unintended. For BeOS and RobinHood, this is not much of a problem, since the installed user base is so small, as well as the app and the OS. NT machines, with their software complexity, should have their web server configuration very carefully audited. Ideally, they should use a hardening script, available on the Internet.

**HTTP header:** A simple but effective test. It is important to audit for these little clues; why give a potential attacker more information than he actually needs? The first step to a successful exploit is reconnaissance. By keeping as much information on a need-to-know basis, you can make it harder for potential attackers to penetrate your system. This is

---

<sup>17</sup> See <http://www.linuxdoc.org/HOWTO/IPCHAINS-HOWTO.html> for details

<sup>18</sup> See step-by-step tutorials and guidelines at <http://www.cert.org/security-improvement/>

<sup>19</sup> For Linux, see Bastille Linux at <http://www.bastille-linux.org/> .

For Solaris, see YASSP at <http://www.yassp.org/>

<sup>20</sup> See <http://www.cert.org/security-improvement/modules/m11.html> for details

<sup>21</sup> For IIS, see

[http://ehampshire.com/computer/Technical%20Reference%20NT%20IIS%20205\\_0%20and%20Win2K%20Hardening%20Configuration.php3](http://ehampshire.com/computer/Technical%20Reference%20NT%20IIS%20205_0%20and%20Win2K%20Hardening%20Configuration.php3) for details

especially valuable for popular web servers where a plethora of exploits are downloadable, once a positive web server version ID is made.

**Error messages:** This is an important test, but hard to implement correctly because of the vast option space for generating error conditions. The web server can never assuredly pass this test, since you cannot prove, even in principle, program validity (it is a version of the halting problem<sup>22</sup>). However, for the most popular cases, the error messages should be carefully studied so as to ensure that they give out as little information as possible.

**Caching:** Again, a simple, but effective test. Sensitive data like authentication, login/password should not be cached at all. The minor inconvenience of having to re-authenticate yourself a couple of times more are far outweighed by the security benefits (a simple person A bathroom break offer ample possibilities for mischief from colleague B, since 80% of attackers are insiders). There is no reason why the web server should not set these header fields to sensible values, it is mere oversight if they aren't.

**IP Hopping:** This is a very valuable test. The possibility of IP hopping and session cloning is a subtle form of authentication disorder, since IP address is much weaker than login/password. It is somewhat harder to plug and requires some detail knowledge of web server configuration, but should be addressed right away because of the very real combination of user error and session cloning: Users download an unknown \*.exe file and unwittingly install a trojan. This happens all the time. If that trojan captures and transmits the session ID to the malicious hacker, much damage can be done if the web server allows for IP hopping.

**Session time-outs:** A simple, yet effective test and again, a must. As with caching, the operating principle of secrets should be that they are short-lived. The test I ran is admittedly not very precise on my system since I do not have authentication beyond the simple realm model that BeOS offers. It should be run, however, on any serious web server that offers user accounts, e-commerce, etc.

My ten-point auditing checklist, as executed, covers enough bases for a hobbyist running a web server. The standards are pretty stringent, but rightly so – you do not want public relation fiascos of data stolen, hosts hacked, etc which plagued unsafe web servers. If the ten tested points are passed, you can feel somewhat confident that you have at least raised the hurdle for attackers, and, as a bonus, gained a deeper understanding of the web server you are running. I did not address enough user input issues on the application and session layer enough in the performed audit, since my setup did not allow for much user interaction (except for realms authentication). Thus, as conducted, user input validation was neglected for RobinHood. For the setup that I was running - serving static web pages and running cgi-scripts - the ten point audit suite was more than adequate, for it addressed the OS, the web server, the network stack, the application, the scripts and the meta-information returned.

---

<sup>22</sup> See <http://www.cs.washington.edu/homes/csk/halt.html> for short discussion

The complete auditing checklist with some 27 points requires expertise to implement, but if done thoroughly and systematically, should give you a high degree of certainty that your web server would be a hard nut to crack. This audit is *production strength* (even more so if you implement the ideas proposed in future directions); and I would sign my name to any web server that passes these 27 tests.

I would like to stress that the single best thing you can do is to stay abreast of the vulnerabilities and apply the latest patches. There are three year old serious BIND vulnerabilities that are still exploitable because system administrator are not diligent about this point. The second-best thing to do is to simulate user input. Black-box testing of user input is very hard to do correctly, but some time should be put into it. My additions to Rhoades's checklist with the TCP/UDP/IP/ICMP traffic generation is a start, but, as I said, more effort should be put into application and session layer level testing and, if possible, to thorough application review on a functional and source code level. I will discuss this more in detail below.

## Directions for future work

### ***Shortcoming of general audit processes***

As we move to component-based and service-based applications delivered over the Internet (Microsoft's .NET initiative comes to mind), the auditing process should contain a specific part, tailored to the application / operating system in question, and a *generic part which should be the standard baseline in any audit of network-aware software*. Right now, I find the audit checklists (even Rhoades', although it is a very good start) too constricted to specific versions of specific applications, with some continuity between version of the same application and very little among different vendor applications. This critique applies to auditing web-based applications in general, not just web servers.

Secondly, *automated black box testing tools* need to be more refined. ISIC is a good beginning for network stack integrity testing transport layers and below, but there is a dire need for application and session layer level testing. Right now, these services are few and very expensive – a tool suite for this purpose could be written by an up-and-coming security consultant. It would not require some effort – identification of form input fields, transmission methods (GET vs. POST) and a good random input generator producing metacharacters, large inputs, etc., but would facilitate testing of applications where the source code is not available.

Thirdly, there should be a *widely available network stack validation suite* to test the different TCP/IP implementations. ISIC is a good start, but it would be nicer if I had a suite that tested the stacks according to the RFC requirements.

Nmap, the popular network scanner ([www.insecure.org/nmap](http://www.insecure.org/nmap)), has an option: OS identification through TCP/IP stack fingerprinting. Novak gives a succinct overview of the nine tests that nmap runs [NOV01, 11-16]. Custom segments and packets are sent to a host. The network stacks of different OSs react differently to the same stimuli, for it

turns out that no developer implements the TCP / IP stack according to RFC 791<sup>23</sup> (IP), 792<sup>24</sup> (ICMP) and 793<sup>25</sup> (TCP). Even if they did, the RFCs are not specific on certain points (default TTL is not specified, for instance). The answers of the host can be viewed as the “fingerprint” of the operating system, specifically, the TCP /IP stack, and can be checked against a database of known OS fingerprints.<sup>26</sup>

The fact that OS fingerprinting works is more of a bug in the stack implementation than a feature of nmap and like-minded programs. OS identification is unnecessary and dangerous information for a security point of view and should be actively foiled using either cloak<sup>27</sup> tactic or a stricter adherence to the RFCs<sup>28</sup>. Both techniques would make it more difficult to ascertain the OS running on the system – a major security enhancement.

## ***Future improvements of the audit***

### **Flagging of unsafe code and library calls**

There are two general risks in computer networks. Unavoidable and avoidable ones. Unavoidable ones are inherent risks of offering Internet services; the most notorious risk you run are denial of service attacks. There is no way to guard against this – it is simply a risk you have to live with, like the risk of an earthquake should you chose to live on the San Andrea fault line in California.

Avoidable risks can be mitigated by taking the proper precautions. Of all avoidable risks, buffer overflows of heap and stacks are probably the most serious, giving skilled attackers root access. In recent years, attacks that exploit buffer overflow bugs have accounted for approximately half of all reported CERT advisories [BAR01]:

---

<sup>23</sup> See <http://www.freessoft.org/CIE/RFC/791/index.htm> for details

<sup>24</sup> See <http://www.freessoft.org/CIE/RFC/792/index.htm> for details

<sup>25</sup> See <http://www.freessoft.org/CIE/RFC/793/index.htm> for details

<sup>26</sup> For a good overview on the nmap techniques, see <http://www.insecure.org/nmap/nmap-fingerprinting-article.txt> for details.

<sup>27</sup> See Cohen's Deception Toolkit at [all.net/dtk/index.html](http://all.net/dtk/index.html)

<sup>28</sup> For instance, the specific responses of TCP to certain flags are given in RFC 793, Northcutt lists the basics [NOR01, 344].

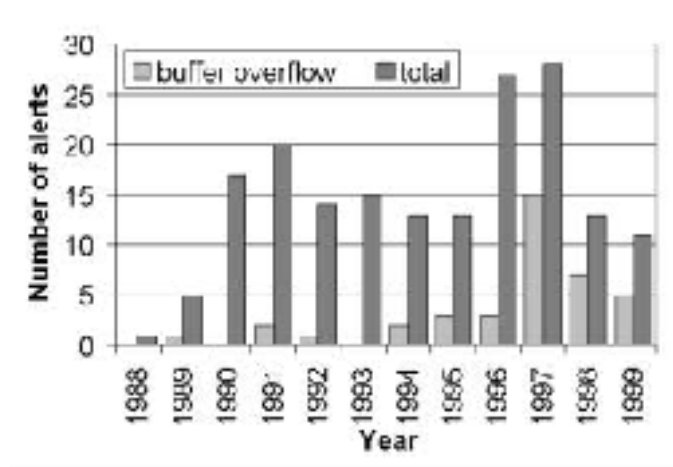


Figure 27: CERT buffer overflow proportion [BAR01]

They can also be avoided using one of two techniques, depending on whether the software you run is open source or not. The basic precaution should be to have a kernel that eliminates these vulnerabilities on principle.

1. Kernel patches
  - a. NT/2000: SecureStack 1.0 at [www.securewave.com/html/secure\\_stack.html](http://www.securewave.com/html/secure_stack.html)
  - b. A patch to eliminate executable stacks on Linux at [www.openwall.com/linux](http://www.openwall.com/linux) [NOR01, 297]

In addition, if the software is open source, you can run a source code checker to flag potential unsafe code. Matt Bishop's SANS course book on how to write secure programs is a good theoretical introduction, but you will want to use an automated static tool.

1. Static, compile-time tools:
  - a. Flawfinder at [www.dwheeler.com/flawfinder](http://www.dwheeler.com/flawfinder)
  - b. RATS at [www.securesw.com/news.html](http://www.securesw.com/news.html)
  - c. Stackguard 7.0 at [www.immunix.org](http://www.immunix.org)

A new approach that does not even require access to the source code is run-time detection, substitution of dangerous function calls and stack integrity checking using loadable kernel modules. This concept is explained in Barabatloo et al.'s paper "Transparent Run-Time Defense Against Stack Smashing Attacks" [BAR01]. Since many bounds cannot be deduced at compile time, this is a valuable new approach.

## Appendix A : BeOS

From the company's, Be Incorporated, website in Menlo Park, CA: "Be Incorporated, founded in 1990 by Jean-Louis Gassée, is a software company focused on delivering an

operating system designed for Internet appliances that deliver information, entertainment and rich Internet experiences to consumers”. ( [www.be.com/aboutbe](http://www.be.com/aboutbe) )

I am using the individual BeOS 5 Personal Edition license for personal, non-commercial use only, available at <http://www.be.com/products/freebeos/> .

These specifications can be found at [www.be.com/products/freebeos/beosspecs.html](http://www.be.com/products/freebeos/beosspecs.html) :

### **Internet Services**

#### *PPP*

Allows connections to Internet Service Providers using the standard point-to-point protocol.

#### *TCP/IP-native*

BeOS speaks the Internet's language and is fully Internet-compatible and compliant right out of the box.

#### *Internet tools*

Comes with built-in Web server, Web browser, POP3 e-mail client, ftp client, ftp server, telnet client, and telnet server, all of which are removable or replaceable by third-party software.

### **Media Services**

#### *Formats*

Allows enabled applications to read and write to files in standard data formats, including QuickTime, AVI, MPEG-1, JPEG, TIFF, BMP, Targa, PNG, PPM, WAV, AIFF, and AU. Supports plug-ins for other formats.

#### *Microsecond resolution*

Allows consistent, accurate, reliable playback and recording of digital media that is tracked down to 1/1,000,000 sec. Result: immediate responsiveness.

### **Graphics System**

#### *Anti-aliased fonts*

Anti-aliased outline fonts are standard, providing smooth text onscreen, as well as on paper.

#### *OpenGL*

Provides industry-standard, high-resolution 3D graphics and rendering.

#### *Direct-access graphics*

Allows your video card to draw images directly from your computer's memory. You'll see fast video and high frame rates in animated sequences.



*Unicode fonts support*

Allows display of languages with complex characters, such as Japanese, Russian, or Hebrew.

**File System**

*64-bit file system*

Allows BeOS to manage disks and files millions of gigabytes in size.

*Journaling*

Tracks all file system changes dynamically, speeding recovery from conditions such as power losses, and providing quick boot-up in under 20 seconds. Protects your hard disk so power failures won't corrupt it.

*File system support*

Plug-in-based support allows read/write access to files created under systems such as HFS (Mac OS), FAT16, FAT32, vFAT, and ISO-9660. Third-party support is available for ext2, NTFS, and NFS.

**Kernel Services**

*Symmetric multiprocessing*

Supports 1, 2, 4, or 8 processors, automatically, without reconfiguration. Doubles your application speed each time you double the number of processors.

*Pervasive multithreading*

Designed to be threaded at every level of the OS to make the most of your CPU's power. Allows the system to respond to user input even when busy with other tasks.

*Virtual memory*

Extends memory by swapping less-used code to disk.

*Protected memory*

Runs each application in its own, isolated memory space. If an application crashes, other loaded applications and the operating system are undisturbed.

*Low-latency kernel services*

250-microsecond (250/1,000,000 sec.) latency for scheduling and timer events ensures accuracy and high system responsiveness.

*Dynamic drivers*

Drivers load and unload dynamically as needed, reducing demands on kernel memory.

**Additional Features**



#### *Hardware support*

Extensive support for a wide and growing variety of graphics, sound, network and SCSI cards, as well as USB, video capture cards, and digital cameras. Complete list of supported hardware at [www.be.com](http://www.be.com).

#### *Applications included*

Web browser, e-mail client, media player, TV viewer, contact manager, graphical archive extractor (expander), utilities, translators, screensaver, 3D audio mixer and other application demos, integrated development environment with source-level debugger.

#### *Software Valet*

Easily manages downloaded software, from installation to updating.

#### *UNIX/POSIX compatibility*

Fully functional POSIX layer allows a wide range of POSIX applications to compile and run on BeOS. Also included is a powerful, UNIX-style shell and windowed terminal program.

#### *Localization support*

Supports inline input of languages with special requirements, such as Japanese. Plug-in architecture facilitates support for virtually any other language.

## **Appendix B: Robin Hood**

RobinHood is a free HTTP/1.1 web server, available for download at <http://bebits.com/app/322>.

It is an original work designed by Jeff Kloss, [joek@be.com](mailto:joek@be.com), and published under the General Public License (GPL), see <http://www.gnu.org/copyleft/gpl-faq.html> for details.

The specifications can be found at <http://bebits.com/app/322> :

#### *HTTP/1.1 Compliant*

#### *Dynamically loaded add-on modules*

Modules are invoked by MIME type:

- *File Handler* - File transfer.
- *CGI Handler* - Invokes CGI scripts
- *SSI Handler* - Process Server Side Includes
- *Directory Handler* - Displays directory listings with Tracker icons
- *Redirect Handler* - Redirection of resources
- *404 Handler* - Custom "404 File Not Found" generator.

#### *Virtual Hosts support*

### *Virtual Resources*

#### *Multiport capacity*

#### *Basic Authenticaon support*

- User defined Realms.
- Uses file permissions to control access.

#### *CGI 1.1 support*

- Supports Parsed Header Output as well as Non-Parsed Header Output
- Works with shell scripts, Perl, and compiled CGIs.
- Full CGI environment variable support

#### *SSI support*

- Supports: config, include, echo, fsize, flastmod, and exec.

## References

- [APAC00] Vandenberg, Paul; Wyess, Susan (26 March 1998). *Apache Web Server Checklist*.  
<http://www.usenix.org/sage/sysadmins/solaris/webservers/apache.html> .13 Jun 2001
- [BAR01] Baratloo, Arash; Singh, Navjot and Tsai, Timothy (2000). *Transparent Run-Time Defense Against Stack Smashing Attacks*.  
<http://www.research.avayalabs.com/project/libsafe/doc/usenix00/paper.html> . 24 Jun 2001
- [ENTE99] Vandenberg, Paul; Wyess, Susan (26 March 1998). *Netscape Enterprise Server Checklist*.  
<http://www.usenix.org/sage/sysadmins/solaris/webservers/netscape.html> . 12 Jun. 2001
- [HACK99] Hacker, Scott. *The BeOS Bible*. Peachpit Press. Berkeley 1999
- [IIS00] Microsoft Corporation (15 Mar. 2000). *Microsoft Internet Information Server 4.0 Security Checklist*.  
<http://www.microsoft.com/technet/security/iischk.asp> . 12 Jun. 2001
- [IIS01] Purdie, Leigh and Corda, George (28 Mar 2001). *INTERNET INFORMATION SERVER 4.0 SECURITY Graded Security Configuration Document*. <http://packetstorm.securify.com/papers/NT/IIS4Config.htm> . 12 Jun. 2001
- [NOR01] Northcutt, Stephen and others (eds). *Intrusion Signatures and Analysis*.

New Riders Publishing. Indianapolis 2001

- [NOV01] Novak, Judy. *Network Traffic Analysis Using tcpdump*. John Hopkins University Applied Physics Laboratory. PDF file. Maryland 2001
- [RHOA01] Rhoades, David. *How to Audit Web-based Applications*. SANS Track 7 - LevelTwo Auditing Networks, Perimeters and Systems Series. Baltimore 2001
- [STEI00] Stein, Lincoln (24 Mar. 2000). *The World Wide Web Security FAQ* (v. 2.0.1). <http://www.w3.org/Security/Faq/> . 12 Jun. 2001.
- [STEV94] Stevens, Richard. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley Professional Computing Series. Reading, MA. 1994.
- [STEV96] Stevens, Richard. *TCP/IP Illustrated, Volume 3: TCP for Transactions, HTTP, NNTP and the Unix© Domain Protocols*. Addison-Wesley Professional Computing Series. Reading, MA. 1994.
- [SURV01] Survey.com . *Enterprise Adoption of Linux*.  
[http://www.survey.com/bidw/description\\_linux.html](http://www.survey.com/bidw/description_linux.html) . 12 Jun. 2001
- [WALK00] Walker, J. (20 Mar. 2000). *The Columbia Guide to Online Style*.  
[http://www.columbia.edu/cu/cup/cgos/idx\\_basic.html](http://www.columbia.edu/cu/cup/cgos/idx_basic.html) . 12 Jun. 2001
- [WEND01] Wendt, Carla. *Network, Perimeter, and System Audit Review*. SANS Track 7 - LevelTwo Auditing Networks, Perimeters and Systems Series. Baltimore 2001
- [YEA96] Yeager, Nancy J. and McGrath, Robert E. *WebServer Technology – The Advanced Guide for World Wide Web Information Providers*. Morgan Kaufmann Publishers. San Francisco 1996
- [ZOE99] Zobebelein, Hans (April 1999). *The Internet Operating System Counter*.  
<http://www.leb.net/hzo/ioscount/data/r.9904.txt> . 12 Jun 2001