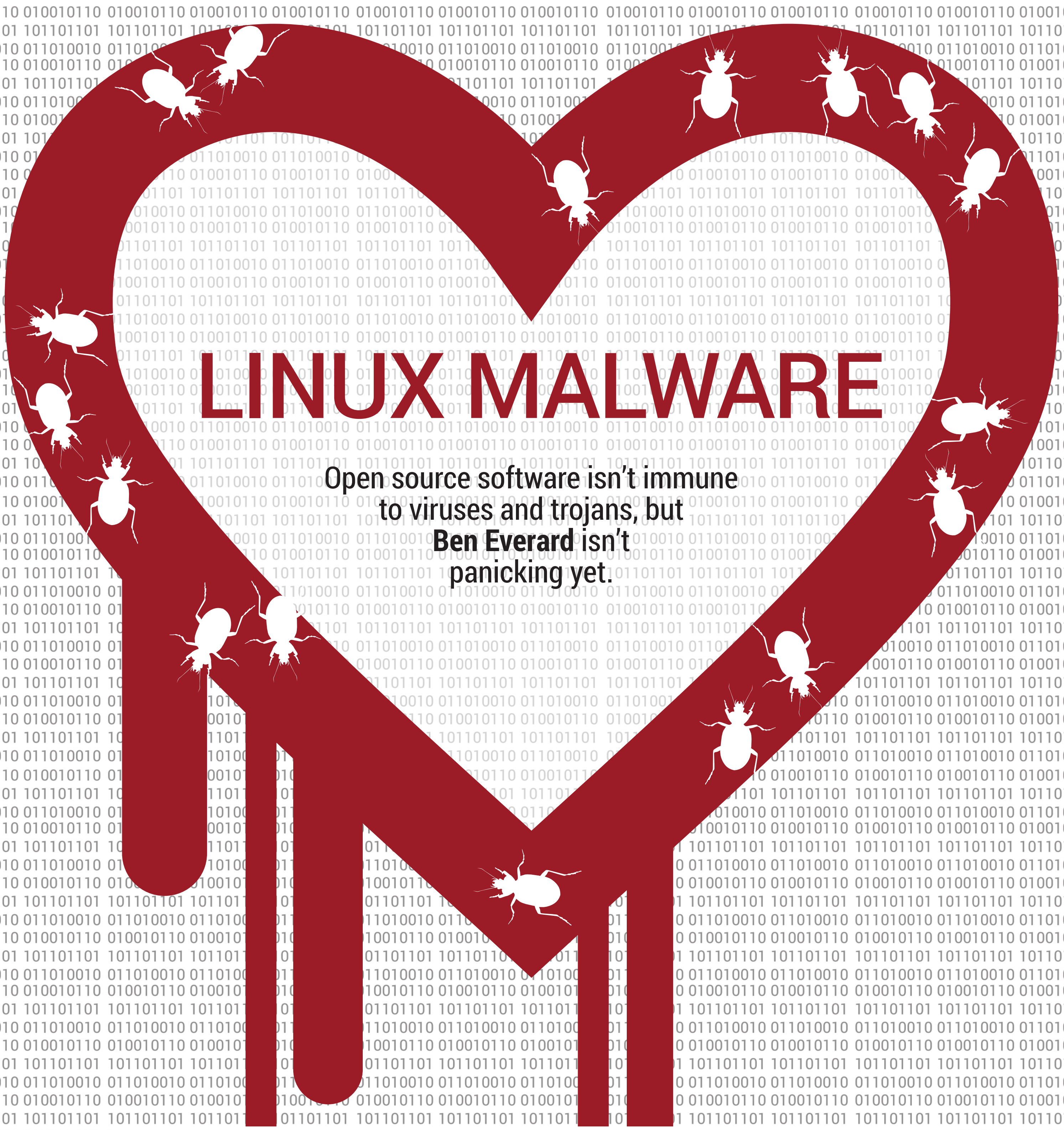


LINUX MALWARE

Open source software isn't immune to viruses and trojans, but **Ben Everard** isn't panicking yet.

Linux is generally considered to be virus-free for all practical purposes. No major distros ship with anti-virus software running, and this is not considered a problem. Most Linux users never install any specific security software, and also never run into any issues. However, as Linux becomes more popular, it also becomes a more attractive target for malware creators. In 2013, we saw the release of the first major piece of malware targeting Linux desktop users in the shape of the Hand of Thief banking trojan. In 2014, the trend has continued with several major Linux malware stories in the news. Is it time to rethink the maxim that Linux is secure?

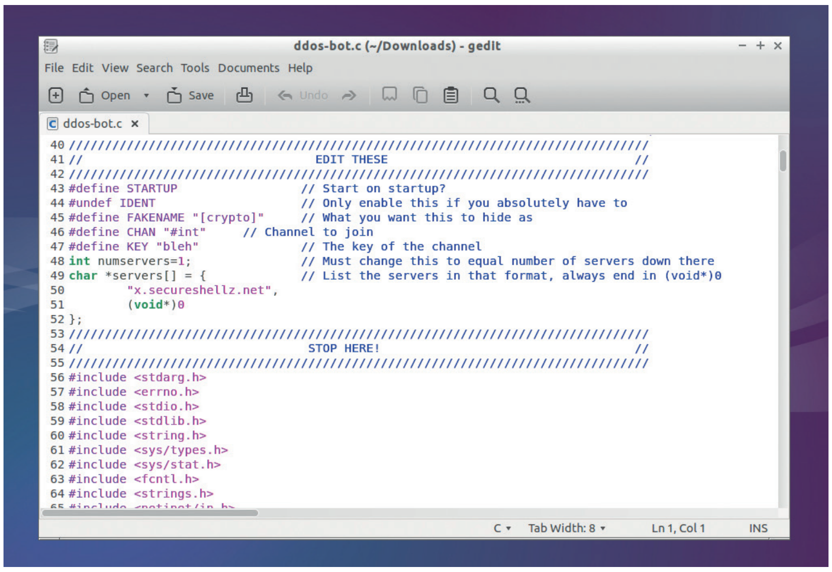
Trojans are pieces of software that hide from the user and steal data from within. Unlike viruses, they don't usually have a mechanism for self-replicating, so they require the user to be tricked into installing them. They're a major problem for people who download software from dubious internet sites. Usually, they attack Windows, but Hand of Thief goes after Linux users. Hand Of Thief is designed to steal information from web browser sessions, specifically login information for internet banking. It can grab data from HTML forms, and other details from the web browser, and relay them back to the attacker. It's also reported to be able to prevent users from accessing anti-virus sites in order to make it harder for them to identify an infection. The Linux version of this trojan has been offered for sale in Russian malware forums for an amount in the region of thousands of pounds, so some people are obviously keen to target Linux users .

Despite being quite powerful once it's installed, Hand of Thief doesn't have a good method of infection. It requires the user to be tricked into installing it manually. This sort of thing is quite familiar to Windows users. For example, many people have received bogus phone calls from people claiming to work for Microsoft saying something along the lines of, 'we have detected a problem with your computer and you need to install some software to fix it'. They will then talk the unsuspecting user into downloading and installing some trojan. This sort of thing is unlikely to work with Linux users both because they tend to be more knowledgeable about computing and because most users would be suspicious of any software that doesn't come from their package manager.

The hard part of Linux malware isn't controlling the system once you're in; it's infecting the system in the first place. This doesn't mean that we Linux users can disregard malware completely though. In 2014 we found that we too were vulnerable to bugs that allow rapid infection of a large number of machines.

Shellshocked

A code execution bug in *Bash* may not immediately seem like a big problem. After all, code execution is the entire point of a shell. However, *Bash* is used to span new sessions in when generating web pages under some server setups. Shellshock – a *Bash* code execution vulnerability announced in September 2014 – allowed attackers to execute code on a server by sending specially crafted HTTP requests that exploited this session spawning.



ddos-bot.c (~/Downloads) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
ddos-bot.c
C Tab Width: 8 Ln 1, Col 1 INS

```
////////////////////////////////////////////////////////////////////////////////
//                              EDIT THESE                                    //
////////////////////////////////////////////////////////////////////////////////
#define STARTUP                 // Start on startup?
#undef IDENT                    // Only enable this if you absolutely have to
#define FAKENAME "[crypto]"     // What you want this to hide as
#define CHAN "#int"     // Channel to join
#define KEY "bleh"              // The key of the channel
int numservers=1;               // Must change this to equal number of servers down there
char *servers[] = {             // List the servers in that format, always end in (void*)0
        "x.secureshellz.net",
        (void*)0
};
////////////////////////////////////////////////////////////////////////////////
//                             STOP HERE!                                     //
////////////////////////////////////////////////////////////////////////////////
#include <stdarg.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <strings.h>
```

After the Shellshock news, crackers tried to exploit our servers; helpfully, they furnished us with commented source code to their malware. It made analysis much easier.

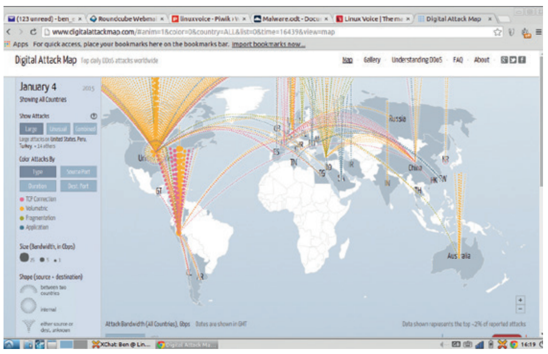
Within a few hours of Shellshock's announcement, malware writers had adapted their code and were scanning huge swathes of the internet for vulnerable servers, trying to infect them using this as a vector.

We have a couple of servers at Linux Voice: one for the website, and one for our internal use. These both run CentOS and were vulnerable to Shellshock (but our server setup wouldn't allow remote execution). We patched them quickly, and didn't have any problems. However, we could see all the attempts to exploit us, and we kept track of the code that attackers were trying to run on our servers. The vast majority tried to use the vulnerability to use **wget** to download some malware onto our server. Seeing these, we grabbed the malware to see what people were trying to run.

All of the malware that targeted us tried to enlist our servers into Distributed Denial of Service (DDOS) botnets. There were bots written in Perl, Python and C (sent as source code to be compiled on the server), but they all worked in roughly the same way.

An important part of any botnet is the command and control setup. This is the mechanism that enables

"The hard part of Linux malware isn't controlling the system; it's infecting it in the first place."



You can view a map of DDOS attacks as they happen at www.digitalattackmap.com, and even scroll back to big attacks in the past.

Embedded malware Inherent insecurity

Embedded devices pose a new type of threat, and one that could be hard to combat unless there's a radical rethink of the way embedded software is maintained. In the past, devices have shipped with software pre-installed, and normally left with the same software for their entire lifetime. This means few embedded devices ever get patched against bugs, few even have their default passwords changed.

Linux can be secure, but it isn't automatically. Unless manufacturers start taking security seriously, and enable users to update their devices, then the internet of things will continue to be an increasingly attractive target for malware developers, regardless of what OS is running it.

Internet of Things When everything is connected, everything is a target

More and more things besides desktop and server computers are being connected to the internet, and anything connected to the internet can get malware. Phones are the most visible aspect of this internet of things (see Android boxout, below), but they're not the only type of device.

Routers and ADSL modems pose a particularly attractive target to malware developers. They're usually left switched on 24 hours a day and permanently connected to the internet, and they sit between people browsing the web and the internet itself, so they're perfectly placed for performing man-in-the-middle attacks.

The first major attack of this type, known as Hydra, happened in 2008. Hydra was an IRC controlled worm that infected Linux systems running on the MIPS architecture (as many routers do) using both a brute force password guesser and an authentication bypass exploit.

Hydra itself wasn't particularly malicious. Infected routers were just used to infect more routers. However, there are many pieces of malware that have since been built based on the Hydra code which are more malicious.

Not all attacks on routers need malware though. One reason for targeting routers is to launch man-in-the-middle attacks people browsing the web, and this can be done quite effectively by altering the router's Domain Name System (DNS) settings. By altering these, an attacker can send all of a router's data through a machine that the attacker controls allowing them to intercept – and change – any of the data. Recently, attackers have used this to target banks in Brazil and Eastern Europe.

The internet of things is rapidly expanding, and there's seemingly no end to the range of devices some manufacturers try to connect to the internet. It goes far beyond routers to more mundane items like televisions and fridges. Although these aren't positioned on the network in such a way as to allow them to run man-in-the-middle attacks like compromised routers can, they could still be used in DDOS botnets.

The trend for embedded Linux devices is increasing, and they're becoming a more and more attractive target for malware. However, few manufacturers make it easy to keep software up to date, and even fewer users actually do. This means that security bugs often sit around unfixed for years. As we saw with Shellshock, attackers can quickly exploit these.

the attacker to communicate with the bots (and ideally with lots of bots concurrently), but at the same time allows the attacker to remain untraceable when the malware gets discovered.

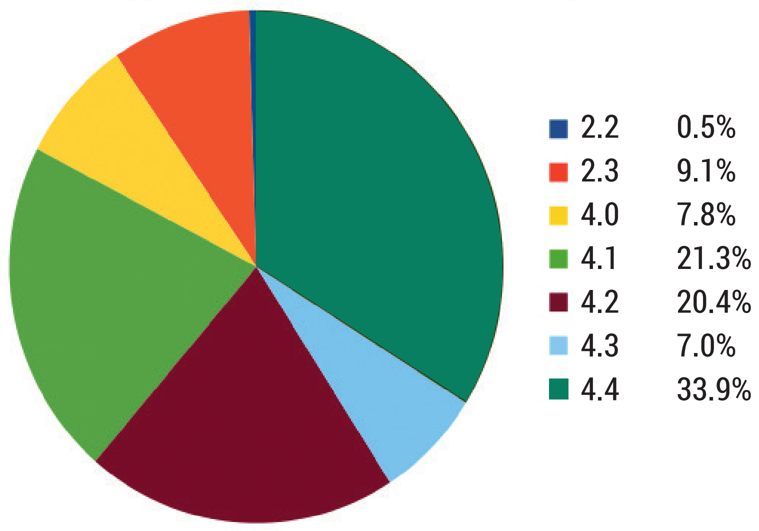
All the bots that we saw used IRC for this. The bots included the details of servers, channels and passwords to connect with. As soon as they ran, they connected to a particular IRC channel, which they listened to, then they acted depending on what was sent on the channel. The people running the botnet could then connect to the same channel through an anonymising proxy and be untraceable.

Helpfully, all the bots were well commented with instructions for use. Here's the comments describing the IRC commands for one bot:

Only about 30% of Android devices are running the latest version of the OS, which means many are vulnerable to known security bugs.

```
The syntax is:
    !<nick> <command>
You send this message to the channel that is defined later in this
code.
Where <nick> is the nickname of the client (which can include
wildcards)
and the command is the command that should be sent.  For
example, if you
want to tell all the clients with the nickname starting with N, to
send you
the help message, you type in the channel:
    !N* HELP
That will send you a list of all the commands.  You can also
specify an
astrick alone to make all client do a specific command:
    !* SH uname -a
There are a number of commands that can be sent to the client:
    TSUNAMI <target> <secs>       = A PUSH+ACK flooder
    PAN <target> <port> <secs>    = A SYN flooder
    UDP <target> <port> <secs>    = An UDP flooder
    UNKNOWN <target> <secs>        = Another non-spoof udp
flooder
    NICK <nick>            = Changes the nick of the client
    SERVER <server>   = Changes servers
    GETSPOOFS              = Gets the current spoofing
    SPOOFS <subnet>  = Changes spoofing to a subnet
    DISABLE                = Disables all packeting from this bot
    ENABLE                 = Enables all packeting from this bot
    KILL                   = Kills the knight
    GET <http address> <save as>  = Downloads a file off the web
    VERSION                = Requests version of knight
    KILLALL                = Kills all current packeting
    HELP                   = Displays this
    IRC <command>       = Sends this command to the server
    SH <command>        = Executes a command
```

Percentage of Android devices by version



By getting all the bots to connect to a single channel, and allowing wildcards in the commands, the controllers can easily launch an attack using a large number of infected computers – this shows that the bot is designed for DDOSing.

The **SH** command gives the controller the power to execute arbitrary code, so the bots could be used for more than DDOSing even though this appears to be their primary purpose. Amassing network bandwidth is usually the aim of server malware.

The Active Threat Level Analysis System (ATLAS) run by Arbour networks monitors DDOS attacks around the world through ISPs sharing their data (see

https://atlas.arbor.net for more information). In 2012, the largest attack reported by Atlas used a peak bandwidth of 100Gbps. By 2014, that had increased to 325Gbps with attacks over 100Gbps occurring almost every month. In other words, DDOSing is becoming big business, and attackers need larger and larger botnets in order to keep up with the competition. Shellshock provided one such source of new servers, but every vulnerability that can be exploited in this way will be. These botnets are then rented out (often by the hour) to whoever has a desire to take a site offline.

Another common goal of malware on Linux servers is as a vector to infect other machines (typically ones running Windows). In this scenario, when a Linux web server is compromised, the attackers then use it to deliver malware to people who visit the websites hosted on the server through so-called drive-by downloads.

Advanced Persistent Threats

Late in 2014, news surfaced of a newly discovered Linux component of the Turla Advanced Persistent Threat (APT). APTs are targeted malware that's designed to get into an organisation and stay there allowing attackers to steal large amounts of data, or monitor activity for a long time. This is the sort of malware used in industrial espionage, or by nation-state spies.

Turla is a suite of APT malware that's been known about for some time. It's thought to be linked to the Russian government, and has been used to spy on governments and militaries around the world. However, up until now, all known components have targeted Windows.

We haven't been able to get hold of a copy of Turla to analyse. However, Turla is based on an older backdoor called cd00r developed by Phenoelit – Turla uses the same method used by cd00r to stay invisible to normal socket monitoring tools while still being contactable from outside for example. A normal TCP connection starts with a three-way handshake. First, the client sends a SYN packet. The server then sends a SYN-ACK packet, and finally, the client responds with an ACK packet. At the end of this handshake, both machines know that they're communicating with a machine that's switched on and working, so they can then start to transfer real data.

From a network security point of view, this means that a computer can always tell which ports on a server are open by sending SYN packets. If they get a SYN-ACK in response then they know that some software is listening on that port. On the server side, Linux **netstat** or **ss** can be used to show which processes are using a particular port.

Cd00r doesn't listen on a port in a normal way. It doesn't respond to SYN packets with SYN-ACK packets. Instead, it sniffs packages that go to a range of ports, and looks for SYN packets being sent to several specified ports in order. For example, the code to trigger cd00r could be SYN packets to port 55, 74, 12, 90 then 45. Once it detected this pattern of SYN packets, it would trigger a piece of code. This is know as port knocking. Turla works in a similar way, but instead of listening for a pattern of SYN packets, it listens for special values in the SYN packets.

This method has allowed Turla to remain undiscovered on Linux for at least four years. If you fancy having a look at how Turla works, the cd00r source code is available at **www.phenoelit.org/fr/tools.html**.

Don't panic!

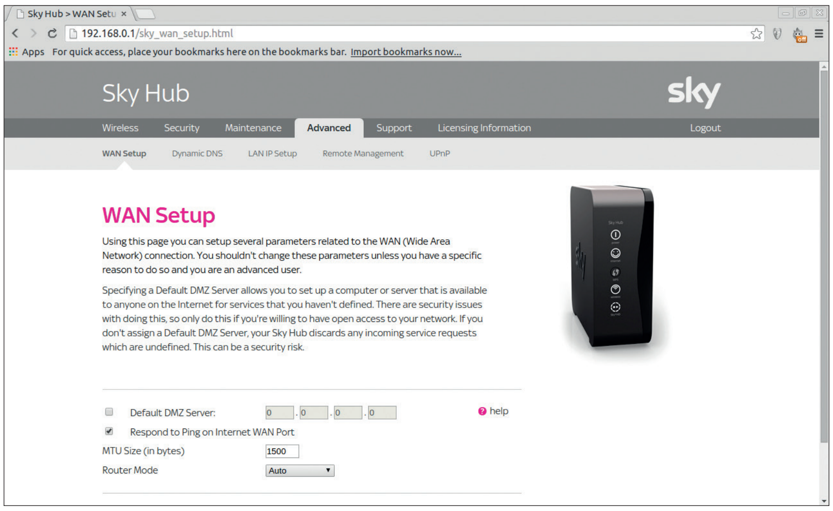
Servers are far more public than most computers, and so are most vulnerable to attack when a weakness is discovered. The popularity of Linux on the server market means that Linux vulnerabilities are exploited heavily once they are discovered.

Desktop computers, on the other hand, are not usually reachable from the internet, and this means attackers have far fewer chances to access them. Linux distros' repositories are our greatest strength in fighting malware on the desktop. As long as you only use trusted repositories, you're unlikely to have any problem with malware. That's true now, and it will remain true for the foreseeable future.

Android Malware Viruses in your pocket

Android – the world's most popular smartphone operating system – is also a distribution of Linux and it s useful case study for how malware can affect this OS. On the desktop, Linux is used much less than Windows or Mac OS X, and has far fewer problems with malware. On phones, Linux (as Android) has a much larger market share than any of the competitors (around 85% of all smartphone users are on Android), and also has the majority of malware (around 97% of smartphone malware is on Android).

The situation isn't as simple as it may first appear though. Smartphone software is typically installed through app stores. Android is the only popular platform that allows appstores other than its own, and it's on these third-party app stores that most of the malware exists. That's not to say there isn't any malware on Google Play Store, but that it's a small proportion of the overall situation.


Sky Hub > WAN Setu

192.168.0.1/sky_wan_setup.html

Apps For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now...

Sky Hub

Wireless | Security | Maintenance | Advanced | Support | Licensing Information | Logout

WAN Setup | Dynamic DNS | LAN IP Setup | Remote Management | UPnP

WAN Setup

Using this page you can setup several parameters related to the WAN (Wide Area Network) connection. You shouldn't change these parameters unless you have a specific reason to do so and you are an advanced user.

Specifying a Default DMZ Server allows you to set up a computer or server that is available to anyone on the Internet for services that you haven't defined. There are security issues with doing this, so only do this if you're willing to have open access to your network. If you don't assign a Default DMZ Server, your Sky Hub discards any incoming service requests which are undefined. This can be a security risk.

☐ Default DMZ Server: 0 . 0 . 0 . 0 help

☑ Respond to Ping on Internet WAN Port

MTU Size (in bytes) 1500

Router Mode Auto

Many routers, like this one, can be configured through an HTML interface. This potentially makes them vulnerable to cross-site request forgery attacks (CSRF).