Advances in OpenBSD packages

Marc Espie <espie@openbsd.org>, <espie@lse.epita.fr>

September 22, 2018
HTTPS is a lie

Marc Espie <espie@openbsd.org>, <espie@lse.epita.fr>

September 22, 2018
This talk is actually a two-parter

- Security of https in pkg_add (joint LSE / OpenBSD work)
- Security/convenience of building packages (PORTS_PRIVSEP)
pkg tools mechanics

Introduction

- `pkg_add` is used to install and update packages.
- Currently grabs packages through an external `ftp` command.
The problem

More precisely

- each transfer is a new ftp process
- https is slower than http by about 30%
- no resumption, because we have no state
- libtls did provide session resumption, but no serialization
HTTPS

TLS, like most crypto frameworks, uses asymmetric crypto for authentication.
That’s expensive.
Session resumption shunts that by starting with a shared secret.
With session resumption, you get one less packet on a connection start.
And it’s even more useful for slow boxes.
Strategy

- `libtls` must have serialisation for session info
- `ftp` must support this as an option
- `pkg_add` has to pass the option (and create the file)
- `pkg_add` must connect to the same server each time

Same server?

We now use cdn mirrors, (mostly thanks to Job Snyder) so the first connection starts with a redirect.

Solution: `pkg_add` parses redirection messages, so that all connections within a run end up on the same box.
the tls part

Not my work, mostly Joel Sing, Bob Beck, Ted Unangt, Theo de Raadt
Ended up working with a temporary file: `ftp -S session=/dev/fd/n`.

the pkg_add part

Ended up retooling temporary file creation so I could get pure fd (and unlink the file) pass the fd to `ftp`. 
Didn’t work at first

No such file

Turns out perl does something really smart, see $SYSTEM_FD_MAX. Everything above it is marked close-on-exec.
That’s actually what we want.
Just turn it off manually WHEN we want the fd.

Marc Espie <espie@openbsd.org>, <espie@lse.epita.fr> Advances in OpenBSD packages
sub setup_session
{
    my $self = shift;
    $self->{count} = 0;
    local ($>, $));
    my ($uid, $gid, $user) = $self->fetch_id;
    if (defined $uid) {
        $> = $uid;
        $) = "$gid $gid";
    }
    my ($fh, undef) = OpenBSD::Temp::fh_file("session",
        sub { unlink(shift); });
    if (!defined $fh) {
        $self->{state}->fatal("Can’t write session into tmp directory");
    }
    $self->{fh} = $fh; # XXX store the full fh and not the fileno
}

sub ftp_cmd
{
    my $self = shift;
    return $self->SUPER::ftp_cmd." -S session=/dev/fd/".fileno($self->{fh});
}

sub drop_privileges_and_setup_env
{
    my $self = shift;
    $self->SUPER::drop_privileges_and_setup_env;
    # reset the CLOEXEC flag on that one
    fcntl($self->{fh}, F_SETFD, 0);
}
Most mirrors support session resumption. A few do weird things, the Brazilian mirror for instance. Some (ftp.fr) had to get session resumption. Some don’t really support session resumption, mirror.vdms.io for instance.
Note that you can debug this with https://www.ssllabs.com/

<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Renegotiation</td>
<td>Supported</td>
</tr>
<tr>
<td>Secure Client-Initiated Renegotiation</td>
<td>No</td>
</tr>
<tr>
<td>Insecure Client-Initiated Renegotiation</td>
<td>No</td>
</tr>
<tr>
<td>BEAST attack</td>
<td>Not mitigated server-side [more info] TLS 1.0 0x013</td>
</tr>
<tr>
<td>POODLE (SSLv3)</td>
<td>No, SSL 3 not supported [more info]</td>
</tr>
<tr>
<td>POODLE (TLS)</td>
<td>No  [more info]</td>
</tr>
<tr>
<td>Downgrade attack prevention</td>
<td>Yes, TLS_FALLBACK_SCSV supported [more info]</td>
</tr>
<tr>
<td>SSL/TLS compression</td>
<td>No</td>
</tr>
<tr>
<td>RC4</td>
<td>No</td>
</tr>
<tr>
<td>Heartbeat (extension)</td>
<td>No</td>
</tr>
<tr>
<td>Heartbleed (vulnerability)</td>
<td>No [more info]</td>
</tr>
<tr>
<td>Ticketbleed (vulnerability)</td>
<td>No [more info]</td>
</tr>
<tr>
<td>OpenSSL CCS vuln. (CVE-2014-0224)</td>
<td>No  [more info]</td>
</tr>
<tr>
<td>OpenSSL Padding Oracle vuln. (CVE-2016-2107)</td>
<td>No  [more info]</td>
</tr>
<tr>
<td>ROBOT (vulnerability)</td>
<td>No [more info]</td>
</tr>
<tr>
<td>Forward Secrecy</td>
<td>With modern browsers [more info]</td>
</tr>
<tr>
<td>ALPN</td>
<td>Yes h2 http/1.1</td>
</tr>
<tr>
<td>NPN</td>
<td>Yes h2 http/1.1 http/1.0</td>
</tr>
<tr>
<td>Session resumption (caching)</td>
<td>No (IDs assigned but not accepted)</td>
</tr>
</tbody>
</table>
session tickets?

**RFC 5077**

- server won’t/can’t cache session information
- so it sends it to the client
- signed/encrypted by the server for validity
- on resumption, client sends the ticket
- server decodes it and says okay

Marc Espie <espie@openbsd.org>, <espie@lse.epita.fr>  Advances in OpenBSD packages
Bad speed-up

Still slow
HTTPS with session resumption is faster than without but still way slower than http.
Went from 30% slower to 25% slower.

It's all about the Pentium\textsuperscript{WRTT}
- TCP handshake: 3 packets
- TLS handshake: an extra 4 packets
- (with session resumption: 6 packets)
At the TCP level
There’s an extension that allows sessions to start quickly
Needs some kind of cookie to prevent some DOSing.

At the TLS level
People are working on TLS 1.3
(yeah, I know it was approved. Let’s wait for a few CVE first, then we can use it)
The handshakes are coalesced, less packets
There’s an experimental mode called 0-RTT resumption
(Send encrypted info directly along with the ticket)
### Security considerations

- **integrity**: we have it through *signatures*
- **authentication**: likewise
- **confidentiality**: do we care
About signatures

Paranoid signatures

Every package is signed externally (in the gzip header)

<gzip header>
Untrusted comment: verify with openbsd-63-pkg.pub
RWt58k1Iaz/zZG4PGHFxsgGANxma1ETfAIk26os0A2RD6A6Bs38CWzpe16j0O+VsZRCsUGGP85WXUfhfct6AZVNAiyyOzOq= 
date=2018-03-11T14:53:47Z 
key=/etc/signify/openbsd-63-pkg.sec 
algorithm=SHA512/256 
blocksize=65536

b0896feco204c2f9291410c032e503ac6da41c9be49537c0688190fd781c292d 
2427dd3653ed775f372dd71a882edf14e88d36bc547bb1590de1e75e806e2c5e 
aaab821c30bc49cab2aa800283c2d83c256476a5ea7d9d0bca2189f11e578f4 
2bc382177f53383d660511bfc21add474f33868b9a8607775be0b01147311 
2b0666b4c73de1e2bc17d2636a5e3c6e4b9529cbbf0e62c25ec26fc847e828cf 
ba3fa75409fe9184a949756d7b937fe0cd0e70f77817add2dbc5f310473b64 
4d71debbf9107c72e573816c671b14914c4ed40963b8509d954e5a97caad84f4b 
83b7efbf1d0b3bab34c03eecbc925f6f1bc6f90ca65d9b50684bc69451bc8938b 
fc1826f6f1bf521ef3625103f712d8c60d37c30482c9d918a170672c14e 
37c8c986fe46761f528b770598f497f2d15f048ac7a21f723fc01aa985d652e 
aaa580a9fd1c6931eo3a952ac3f25e92fda1422f25b9f779b244b43 
517a7236e5d5b424fd344456444b5e03f710c233a721eca7f8c0f6aff4a6c67 
<actual gzip contents>
controls

- `pkg_add` displays the ts of the quirks (exception) package
- (get quirks thru https explicitly)
- (add expiration timestamps for quirks so that `pkg_add` complains)
- the quirks package contains a cve list of packages that **MUST** be updated.
- people don’t remember that list
In my lab, I work with people on automatic data analysis
- malware classification
- network traffic analysis

So using sophisticated stats techniques to recognize stuff

(they call it Machine Learning these days, but it’s just large stats cooking... mostly)
https is a lie

Encryption does not change packet size

- So I can decode actual GET url lengths
- And `pkg_add` has a predictable order: get stuff in alphabetic order, minus dependencies.
- The only saving grace is the `signify` block size.

Marc Espie <espie@openbsd.org>, <espie@lse.epita.fr>  
Advances in OpenBSD packages
do a few pkg_add ourselves to see the workflow
always gets the list, then quirks
then packages in alphabetical order, with dependencies inserted
... so you retrieve package names lengths
... and match it with the mirrors
... just need a bit of exploration, but it is 100% accurate.
https is a lie

- I can get around url GET lengths
- X-OpenBSD-Padding: A.....A (normal distribution)
- harder to work around package sizes
Using http 1.1 (persistent connection) with byte-ranges is the way to go. 

but

- this only solves the connection packets
- byte-range are helped by signatures (we only need 64K blocks)
- encryption still doesn’t protect lengths
- we would need to enqueue several requests so that the size is unpredictable
There are two modes

- initial installation
- update
On initial installation

- we still need byte-ranges (dependencies)
- so more or less grab 64K
- ... then the rest
- we’ve got the sizes from the list
- stutter a bit, pad stuff with redundant 64K
- so the sizes are all 64K
- ... but for the last one, we get it as two requests
- ... so randomly do other requests in two parts
Signatures have a header
... that header is longer for larger packages
... so we need to fuzz the sizes as well
... basically random sizes 8K-16K
... so we don’t even have to grab the last one twice
... this will work for updates as well
Brittle

Relies on two requests being queued and satisfied as one stream

Better approach

- randomly split each package in packets of reasonable size
- make sure the last packet is large enough
- (smallest package is 550 bytes)
- so basically, you pre-compute a split of the package in packets around 8K long (that can occasionally go down to 500 bytes and up to 16K).
- you make darn sure the last packet is big enough. If it’s not, you enlarge other packets randomly
- ... I haven’t looked at timing yet, but this should be darn unpredictable
Not deployed yet

Details details...

- somewhat complicated (though less than the first version)
- will reuse the "distant" protocol that we use for scp
- needs to handle timeouts manually, because HTTP 1.1 tends to drop you for no reason
dpb state

dpb is fully privilege separated, several identities
- starts as root
- fetches as _pfetch
- builds as _pbuild
- ports tree belongs to user
The problem is when you try to play with it manually (as always with privsep). You end up doing everything as root, because it’s a nightmare otherwise.

**Introducing PORTS_PRIVSEP**

The idea is to use doas to change identity on the fly. and add calls to bsd.port.mk
Why doas

- Basically the security model is still that you have to become root to run `pkg_add` anyway.
- So then you add doas invocations to switch to `_pbuild` or `_pfetch`.
- Once you moved to `_pbuild` or `_pfetch`, you can’t go back, so this gives you security.
- Oh, and `_pfetch` has network access, but `_pbuild` doesn’t.
_PFETCH = ${SUDO} -u ${FETCH_USER}
_PBUILD = ${SUDO} -u ${BUILD_USER}
_MK_READABLE = ${_PBUILD} chmod a+rX
_PMAKE = cd ${.CURDIR} && PKGPATH=${PKGPATH} exec ${_PBUILD} ${MAKE}
_PREDIR = |${_PBUILD} tee >/dev/null
# Some operations will need sudo in privsep mode
_PSUDO = ${SUDO}
_UPDATE_PLIST_SETUP=FAKE_TREE_OWNER=${BUILD_USER} \ 
  PORTS_TREE_OWNER=$$(id -un) ${SUDO}
_INSTALL_CACHE_REPO = ${SUDO} install -d -o ${FETCH_USER} -g $$($(id -g ${FETCH_USER}))
Basic introduction was surprisingly easy, but there were a lot of details to fix. Because it can’t be recursive: if you switch to _pbuild, you can no longer use doas. So you have to be very sure of each operation. (thanks to Solene, Jeremy, Klemens...)
The fine print

- Slightly less secure than dpb, because you trust the infrastructure.
- Requires doas setup (so setuid binaries) within the ports tree.
- Long term solution should be to start as root and drop privs always.
- There are just way more places to tweak
- And then you run large Makefiles as root...
- That’s what base does (apart from the large Makefiles part)
Base uses su.
We will use chroot -u user / cmd
... because it can be prefixed to a command without needing any more tweaks.
Bonus: a second chroot mode, where the ports tree is NOT chroot’d but anything running in it is.
We are trying to limit stuff, basically have doas only run `pkg_add` and delete.

- but doas sucks, because you can’t restrict it to "perl somescript args".
- And we have scripts in the ports tree that are run through perl.
- because not guaranteed execute bit.
- Special treatment for `update_plist` and `update_patches`.
- They are run as root and switch to user and `_pbuild` as needed.
- Because the whole build tree might not be readable.
- Move them to base?
- Paranoia...
People running `dpb` in a chroot will use `proot`.

`proot` sets up everything by default

for manual interaction, you mostly need two scripts/aliases
  - one to chroot
  - another one to edit files prior to edit patches.

but with `PORTS_PRIVSEP` set, you can do everything properly.
Success! and then

Turns out people like having PORTS_PRIVSEP
Even when not in a proot with dpb
So they started setting things up manually
We will have a `fix-permissions` target.
Maybe even run automatically if the checks don’t work?
HTTPS: the same deductive approach can be used for any public web site. Do some web-crawling + predictive model. I’d be very interested to know how other package systems fare. For instance, news sites: article length + assets