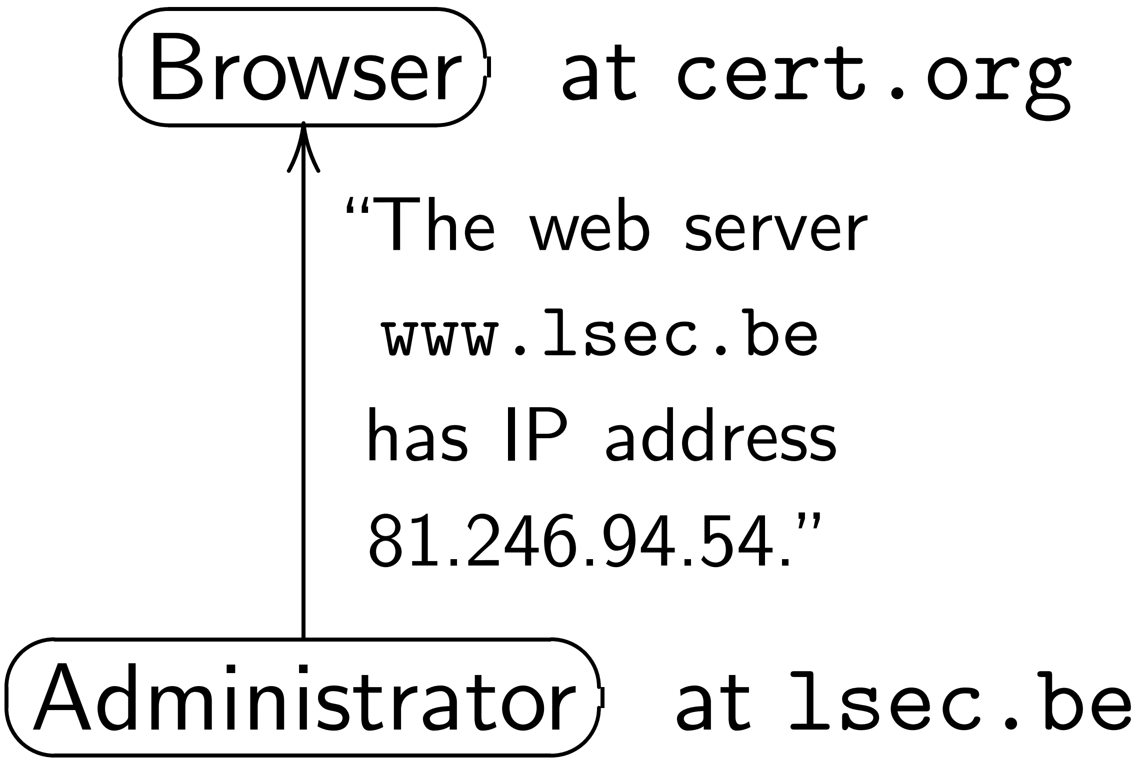
Attacks on DNS

D. J. Bernstein
University of Illinois at Chicago

The Domain Name System

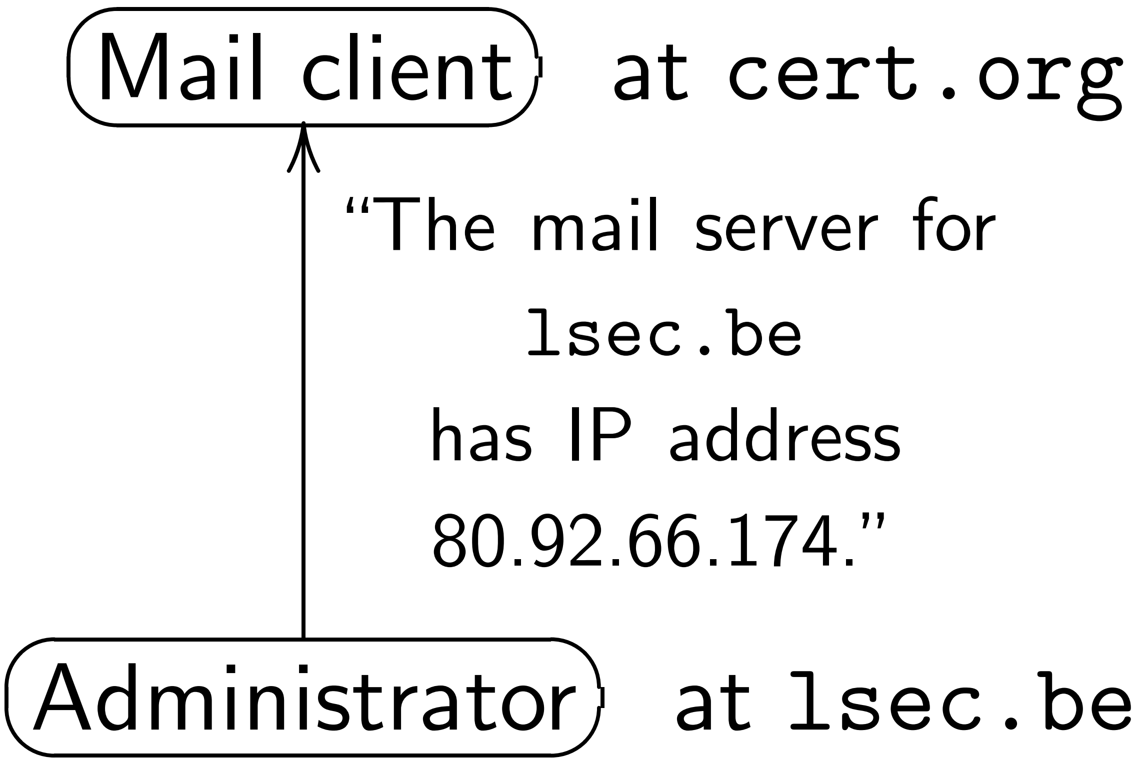
`cert.org` wants to see `http://www.lsec.be`.



Now `cert.org` retrieves web page from IP address 81.246.94.54.

Same for Internet mail.

`cert.org` has mail
to deliver to `someone@lsec.be`.



Now `cert.org`
delivers mail to
IP address 80.92.66.174.

Forging DNS packets

`cert.org` has mail
to deliver to `someone@lsec.be`.

Mail client at `cert.org`

“The mail server for
`lsec.be`
has IP address
157.22.245.20.”

Attacker anywhere on network

Now `cert.org`
delivers mail to
IP address 157.22.245.20,
actually the attacker’s machine.

“Can attackers do that?”

"Can attackers do that?"

— Yes.

"Can attackers do that?"

— Yes.

"Really?"

"Can attackers do that?"
— Yes.

"Really?" — Yes.

“Can attackers do that?”
— Yes.

“Really?” — Yes.

“Don’t the clients check
who’s sending information?”

"Can attackers do that?"
— Yes.

"Really?" — Yes.

"Don't the clients check
who's sending information?"

— Yes, but the attacker
forges the sender address;
as easy as forging address
on a physically mailed postcard.

Real postcard from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged postcard from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

Real packet from administrator:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 80.92.66.174. Hoping to have informed you sufficiently!

Forged packet from attacker:

From: `lsec.be` admin

To: `cert.org` mail client

The mail server for `lsec.be` has IP address 157.22.245.20. Hoping to have informed you sufficiently!

“Is the client always listening for the address of `lsec.be`?”

“Is the client always listening for the address of `lsec.be`?”

— No.

When client wants to know address of `lsec.be`,
it sends a query
to the administrator,
and listens for the response.

Forged `lsec.be` information is effective *if* it arrives at this time.

Many ways for attackers to time forgeries properly:

1. Attack repeatedly.
One of the forgeries will arrive at the right time.

2. Poke the client
to trigger a known lookup.

3. Attack caches
a long time in advance.

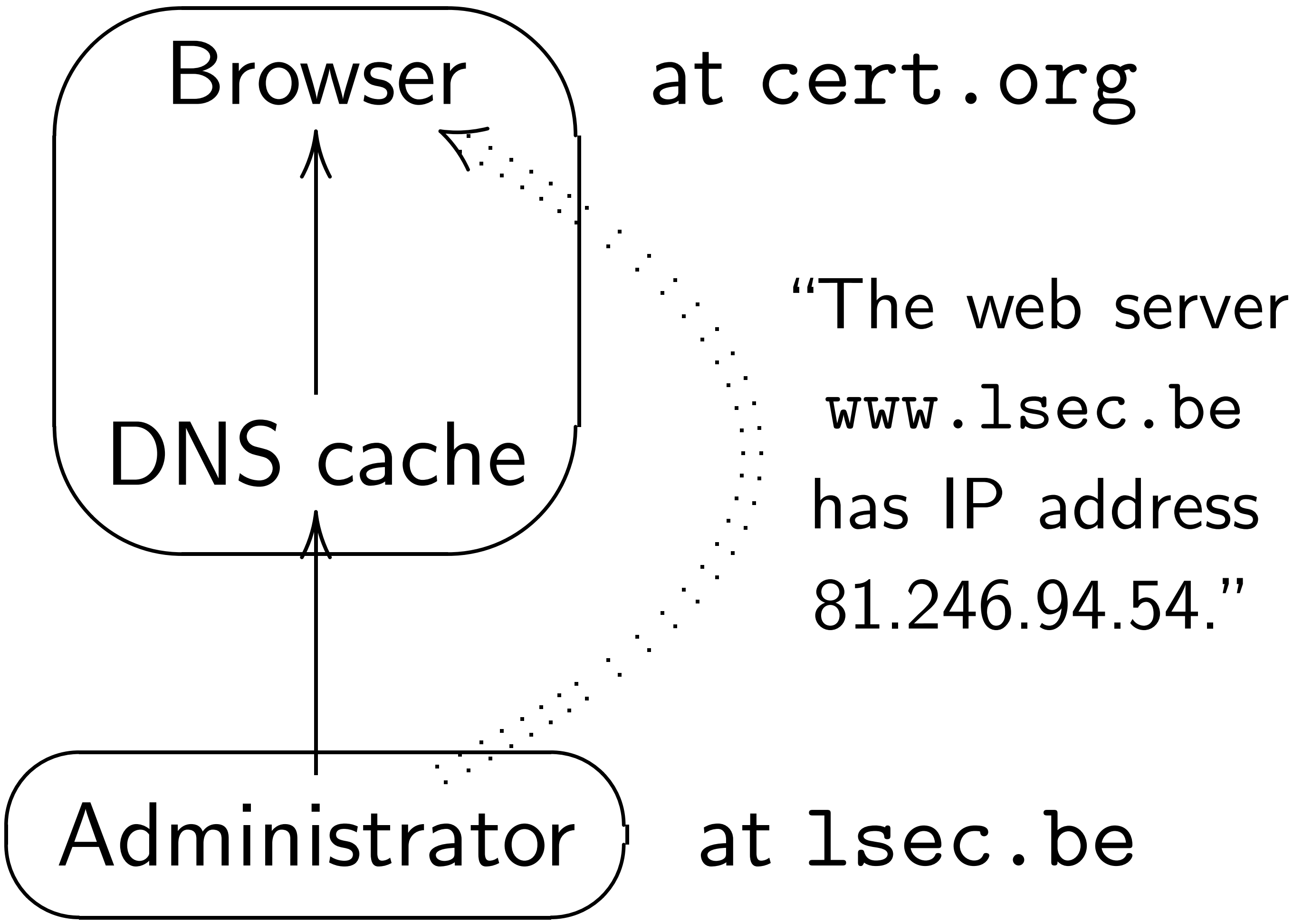
Many ways for attackers to time forgeries properly:

1. Attack repeatedly.
One of the forgeries will arrive at the right time.

2. Poke the client to trigger a known lookup.

3. Attack caches a long time in advance.

4. Easy, succeeds instantly:
Sniff the network.



Browser pulls data from DNS cache at cert.org.

Cache pulls data from administrator *if* it doesn't already have the data.

A typical blind attack:

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees HTTP request, sends web page to browser, waits a moment (for browser to ask cache about `www.lsec.be`), and sends the DNS cache forged data for `www.lsec.be`.

“Doesn’t the attacker have to win a race against the legitimate DNS packets from the administrator at `lsec.be`?”

“Doesn’t the attacker have to win a race against the legitimate DNS packets from the administrator at `lsec.be`?”

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.

2. Mute the legitimate server.

3. Poke-jab-jab-jab-jab-jab.

“Doesn’t the attacker have to win a race against the legitimate DNS packets from the administrator at `lsec.be`?”

— Yes, but many ways for attackers to win race:

1. Deafen the legitimate server.

2. Mute the legitimate server.

3. Poke-jab-jab-jab-jab-jab.

4. Easy, succeeds instantly: Sniff the network.

Typical combined blind attack:

Attacker floods `lsec.be` servers with queries that consume all available CPU time, or floods `lsec.be` network with packets that consume all available network capacity.

Attacker pokes the client to trigger an `lsec.be` lookup. Attacker immediately sends a series of forged packets to the DNS cache.

“What if attacker loses race?”

— Many ways for attacker to continue his attack:

1. He attacks another cache.

2. He attacks another name on the same cache.

3. He attacks the same name on the same cache, sideways.

With any of these approaches, number of cached forgeries increases linearly over time.

Sideways attacks were popularized in 2008 by Dan Kaminsky.

Attacker pokes the client to trigger a DNS lookup for `8675309.lsec.be`.

Attacker forges response for
`8675309.lsec.be`
with extra information about
`www.lsec.be`.

For various performance reasons,
DNS caches are willing
to accept the extra information.

Interlude: types of security

Confidentiality: The attacker cannot see this information.

Integrity: The attacker cannot *silently* modify this information. User doesn't see wrong data.

Availability: The attacker cannot modify this information. User sees the right data.

Attacker flooding a network
is compromising availability.
("Denial of service.")

Attacker successfully forging
DNS packets of `lsec.be`
is compromising integrity.

Attacker stealing email
is compromising confidentiality:
attacker sees the email.
Also compromising availability:
user doesn't see the email.

Lack of availability often
helps compromise integrity:
e.g., flooding a server
can assist in DNS forgeries.

Lack of confidentiality often
helps compromise integrity:
e.g., sniffing DNS queries
makes forgeries trivial.

Lack of integrity often
helps compromise confidentiality:
e.g., forging DNS packets
allows redirecting mail.

etc.

PGP-encrypting your email
can provide confidentiality.
Attacker who steals email
still won’t understand it.

Also integrity.
Attacker can’t modify email.

But it won’t provide availability.
The email silently disappeared!

Retroactively checking integrity
doesn’t restore availability.

What about cookies?

Cache's DNS query packet contains a 16-bit ID.

RFC 1035 (1987): "This identifier is copied [to the] reply and can be used by the requester to match up replies to outstanding queries."

Traditional ID sequence:
1, 2, 3, 4, 5, etc.

Cache discards any reply that has the wrong ID.

"How does the attacker guess the right ID?"

Attacker sets up a web page `supersecuritytools.to`, including an inline image from `www.lsec.be`.

Attacker provides DNS data for `supersecuritytools.to` from his own DNS servers.

Victim asks browser to view `supersecuritytools.to`.

Attacker sees cache's ID for `supersecuritytools.to` DNS query. Attacker then predicts ID for `lsec.be` query.

“ID 47603: SST.to?”

Cache → Attacker

Cache ← Attacker

“ID 47603: SST.to 157.22.245.20 (and please don’t cache this)”

“http://SST.to”

Browser → Attacker

Browser ← Attacker

“... lsec.be ...”

Cache ← Attacker

“ID 47604: lsec.be 157.22.245.20”

Cache ← Attacker

“ID 47604: lsec.be 157.22.245.20”

Cache ← Attacker

“ID 47604: lsec.be 157.22.245.20”

Cache ← Attacker

“ID 47604: lsec.be 157.22.245.20”

Cache ← Attacker

“ID 47604: lsec.be 157.22.245.20”

More recent idea:
"Hey, let's use random IDs! Then the attacker won't be able to forge a packet with the right ID!"

Can use any good stream cipher to expand a short secret key into a long sequence of "random" numbers.
AES-CTR: $\approx$ 10 cycles/byte.
Salsa20/12: $\approx$ 3 cycles/byte.

Output is very hard to predict: attacker has no idea what the next ID will be, even after seeing entire sequence of previous IDs.

Client can randomize
16-bit ID *and*
16-bit UDP source port.

Implemented and advertised
in `djbdns` since 1999,
and in PowerDNS since 2006.

Same feature added 2008.07
in "emergency" upgrade to BIND,
Microsoft DNS, Nominum CNS,
most Cisco products, etc.

New York Times headline:
"WITH SECURITY AT RISK,
A PUSH TO PATCH THE WEB"

Bad news: Ignorant developers often whip up breakable ciphers. See, e.g., Klein's analysis leading to 2007.07.24 "emergency" BIND 9 upgrade:

> In essence, this is a weak version (since the output is 16 bits, as opposed to the traditional 1 bit) of the well studied cryptosystem known by many names: "bilateral stop/go (LFSR) generator", "mutually clock controlled (LFSR) generator" and "mutual (or bilateral) step-1/step-2 (LFSR) generator". ...
>
> The Perl script in Appendix C takes around 10-15 milliseconds ... to extract the internal state from 13-15 consecutive transaction IDs.

Also Klein's 2008.02.06 analysis of IDs in OpenBSD, NetBSD, FreeBSD, MacOS X:

> OpenBSD ported BIND 9 into their code tree, but rolled their own PRNG for the DNS transaction ID field). ... "We decided ... to use a more proven algorithm (LCG, Linear Congruential Generator) instead. Thanks to this wise decision, the BIND 9 shipped with OpenBSD does not have this weakness. The proactive security of OpenBSD strikes again." ... I discovered a serious weakness in OpenBSD's PRNG, which allows an attacker to predict the next transaction ID.

Also Klein's 2007 and 2008 analyses of Microsoft IDs.

Bad news, continued:
Many ways for attackers to beat ID+port randomization, even if it's cryptographic.

1. Attack repeatedly.
"An attacker who makes a few billion random guesses is likely to succeed at least once; tens of millions of guesses are adequate with a colliding attack;" etc.

2. Allocate most UDP ports to other tasks, non-reusably.

Bad news, continued:
Many ways for attackers to beat ID+port randomization, even if it's cryptographic.

1. Attack repeatedly.
"An attacker who makes a few billion random guesses is likely to succeed at least once; tens of millions of guesses are adequate with a colliding attack;" etc.

2. Allocate most UDP ports to other tasks, non-reusably.

3. Easy, succeeds instantly:
Sniff the network.

Colliding attacks on caches
(2001 Bernstein),
aka "birthday attacks":

Attacker triggers many queries
for one name `lsec.be`.
Typical cache allows 200 queries,
using 200 ID+port combinations.

Attacker sends forgeries
to many ID+port combinations
for this name `lsec.be`.

Any ID+port collision succeeds;
i.e., each forgery attempt
has $200/2^{32}$ chance of success.

Port-allocation attacks
(2008.08 Bernstein):

Computer with a DNS cache
usually has more servers.

Attacker convinces those servers
to talk to the attacker on
tens of thousands of UDP ports.
Not all available UDP ports,
but *almost* all.

Computer doesn't let the cache
reuse those UDP ports.
Cache chooses other UDP ports.
Attacker sends DNS forgeries
to *those* UDP ports.

Clients can try to reduce a forgery's success chance: suppress duplicate queries; randomly replace `google.com` by, e.g., `GooGLe.cOm`; remove cache entries in case of doubt; limit caching; ask twice; etc.

Many performance problems. Many interoperability problems. Many bogus security analyses.

Mostly ineffective against smart blind attackers, and all completely ineffective against sniffing attackers.

Who does DNS trust?

What we've learned:
Attackers sniffing the network have trivial control over DNS.
Blind attackers around the world also have some control.

Who does DNS trust?

What we've learned:
Attackers sniffing the network have trivial control over DNS.
Blind attackers around the world also have some control.

What if packet forgeries were magically eliminated?
What if all DNS packets had unforgeable sender addresses?
Who would still control DNS?

Original DNS cache algorithms specified in RFC 1034 allowed any DNS server to control all DNS records.

Cache asks `SST.to` DNS server about `www.SST.to`. Server says: `www.SST.to` has canonical name `www.lsec.be`, which has address `157.22.245.20`.

Cache records address of `www.lsec.be`. Browser later asks about `www.lsec.be`, receives `157.22.245.20`.

The “bailiwick” fix
(1997 BIND; 2003 Microsoft):

The `SST.to` DNS servers
are authorized to control
the name `SST.to`
and names ending `.SST.to`.
Not authorized to control
`www.lsec.be`.

Caches reject `www.lsec.be` data
from the `SST.to` DNS servers.

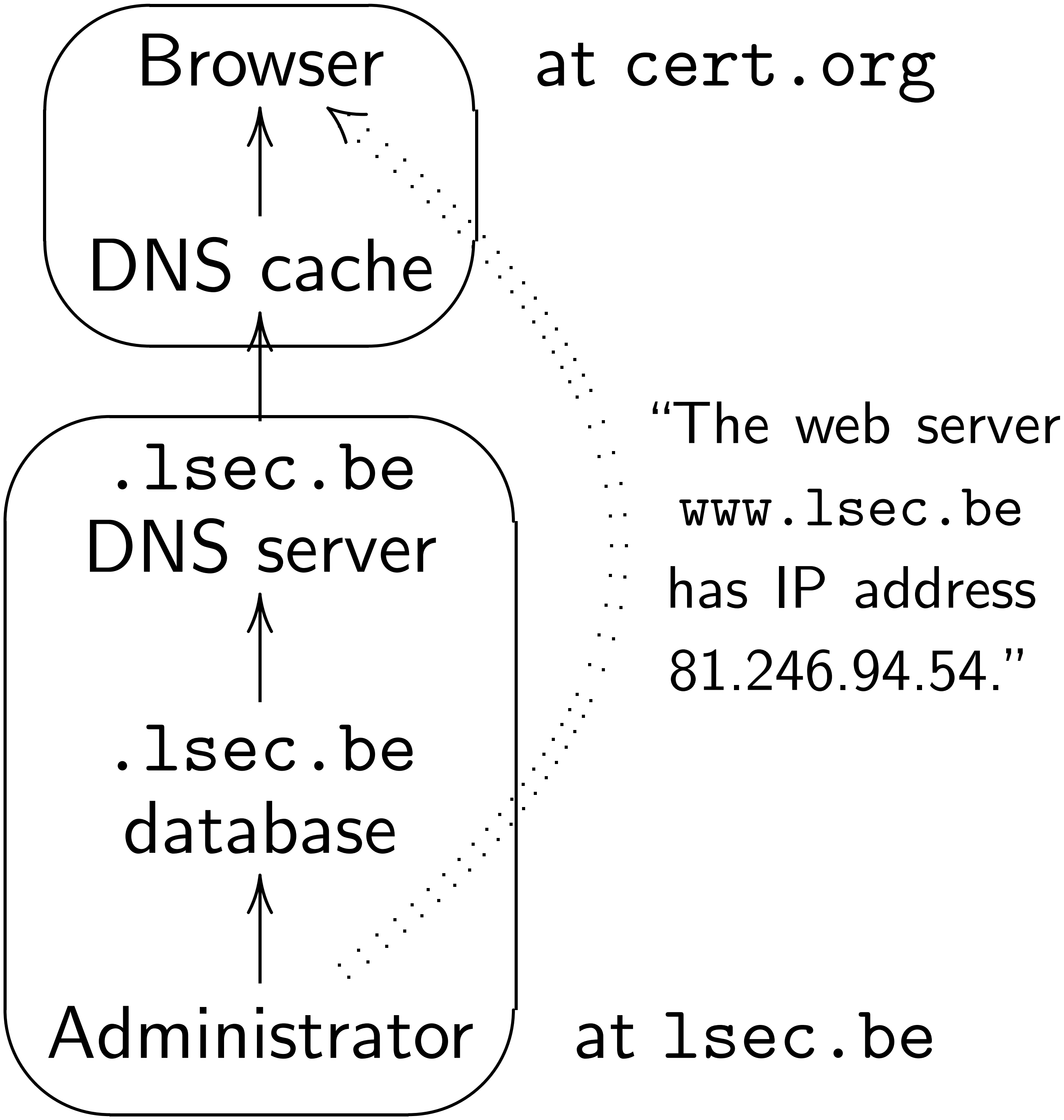
Bugs continue cropping up.
e.g. BIND bug fixed 2003:
`microsoft.com` server can say
that `google.com` has no address.

For performance reasons, administrators sometimes set up third-party DNS servers.

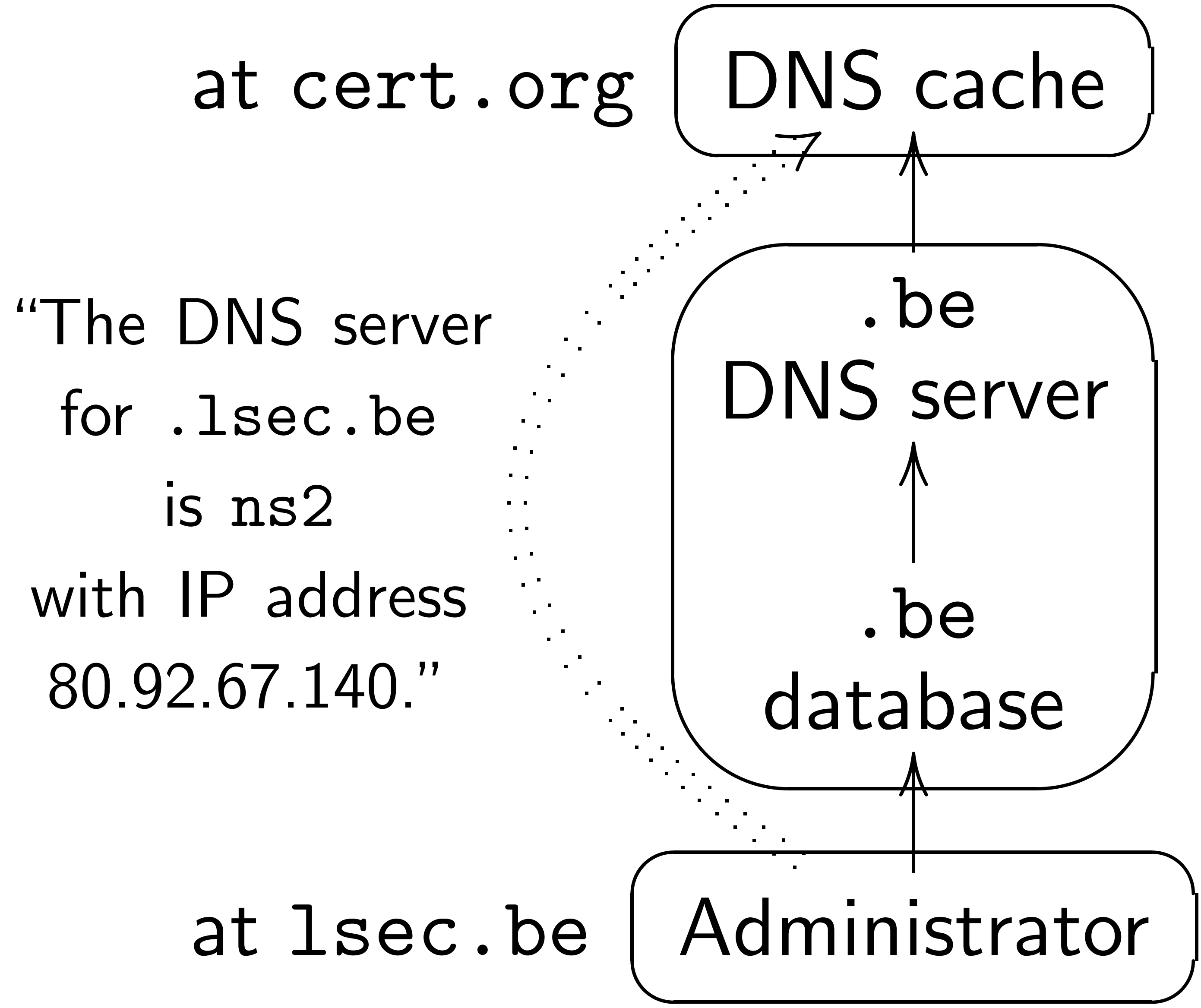
e.g. The `rsa.com` administrator set up two `rsa.com` servers: one of his own computers and a third-party computer.

In 2000, an attacker broke into the third-party server and misdirected `www.rsa.com`.

The `rsa.com` administrator no longer uses third-party servers.


Browser
DNS cache
at cert.org
.lsec.be
DNS server
.lsec.be
database
Administrator
at lsec.be
“The web server
www.lsec.be
has IP address
81.246.94.54.”

DNS cache learns location of `.lsec.be` DNS server from `.be` DNS server:



All packets to/from DNS cache:

God sayeth unto the DNS cache:

"DNS Root `K.Heaven` 193.0.14.129"

"Web `www.lsec.be`?"

193.0.14.129 ← → DNS cache

"DNS `.be brussels` 193.190.135.4"

"Web `www.lsec.be`?"

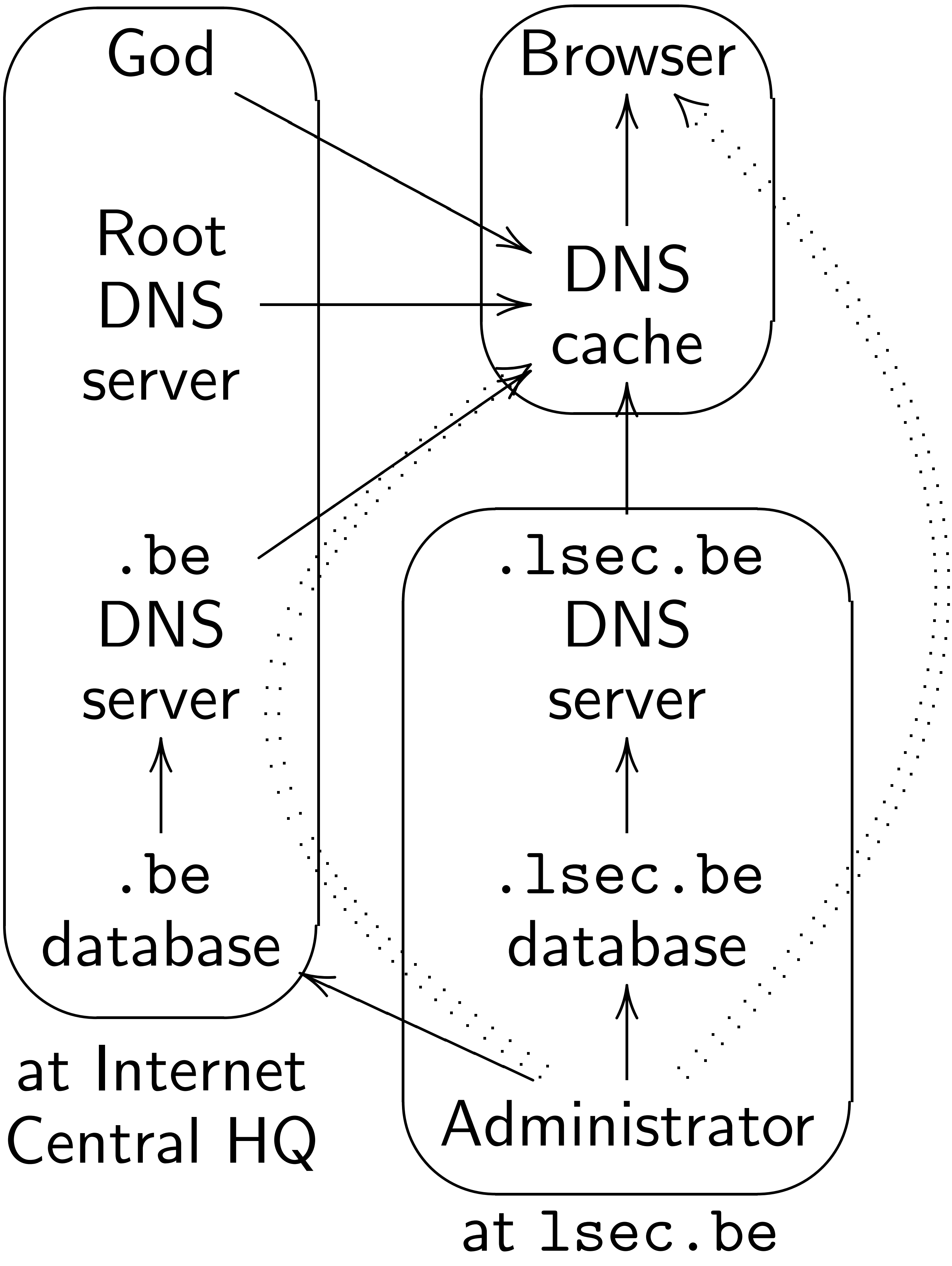
193.190.135.4 ← → DNS cache

"DNS `.lsec.be ns2` 80.92.67.140"

"Web `www.lsec.be`?"

80.92.67.140 ← → DNS cache

"Web `www.lsec.be` 81.246.94.54"


God
Browser
Root
DNS
server
DNS
cache
.be
DNS
server
.lsec.be
DNS
server
.be
database
.lsec.be
database
Administrator
at Internet
Central HQ
at lsec.be

This architecture means that
the `www.lsec.be` address
is controlled
by the DNS root server;
by the `.be` DNS server; and
by the `lsec.be` DNS server.

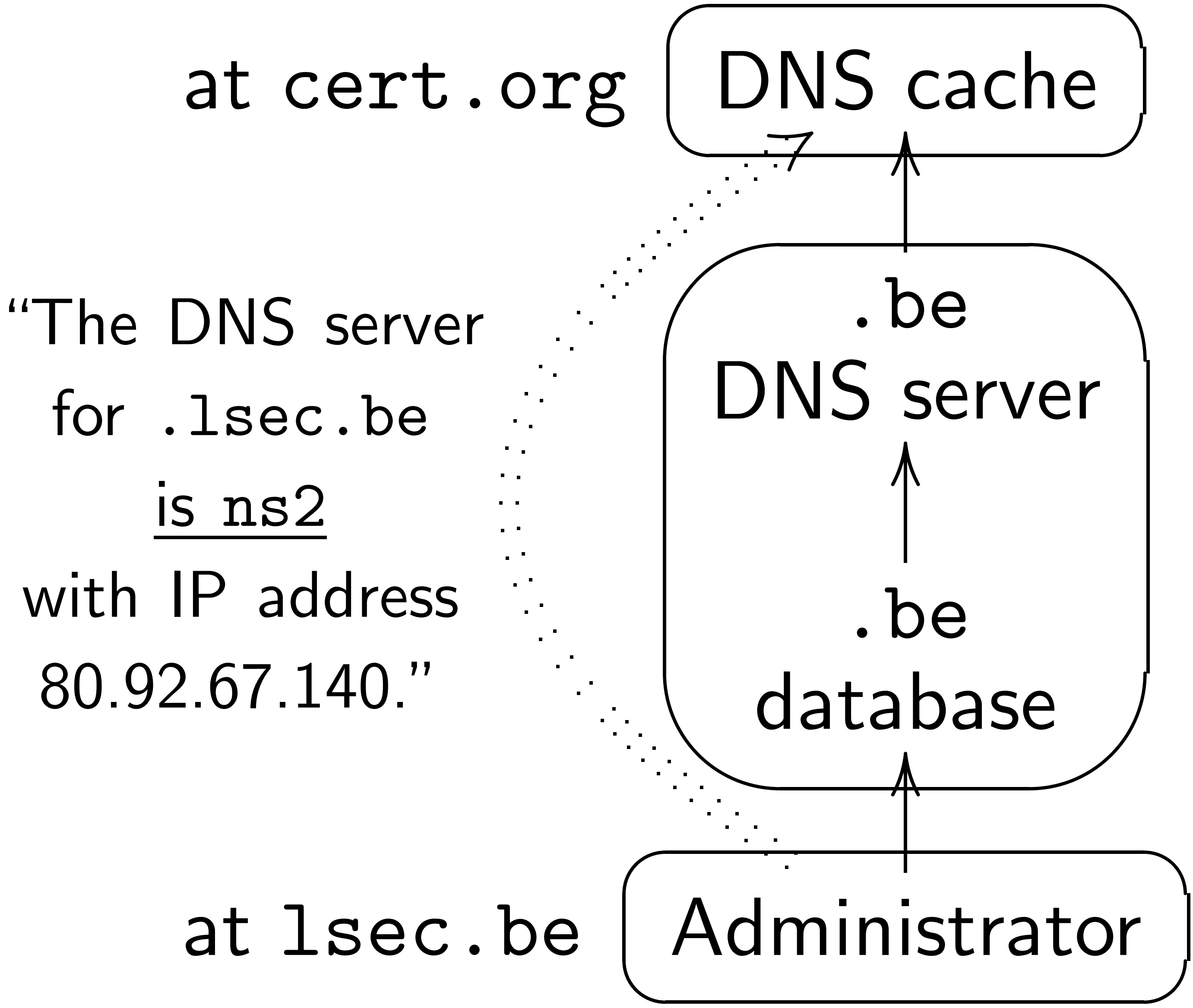
This isn't just the
`lsec.be` DNS server!

e.g. 2001 incident:
An attacker fooled
Internet Central Headquarters
into accepting fake data
for `microsoft.com`.

But wait, there's more!

Recall that the DNS servers

for `lsec.be` have names.



These names can be outside `lsec.be`.

One of the DNS servers for `w3.org` is named `w3csun1.cis.rl.ac.uk`.

One of the DNS servers for `ac.uk` is named `ns.eu.net`.

One of the DNS servers for `eu.net` is named `sunic.sunet.se`.

One of the DNS servers
for `sunet.se` is named
`beer.pilsnet.sunet.se`
and is horribly insecure.

Attacker takes control of
`beer.pilsnet.sunet.se`;
tells DNS cache a fake address
for `sunic.sunet.se`;
tells DNS cache a fake address
for `ns.eu.net`;
tells DNS cache a fake address
for `w3csun1.cis.rl.ac.uk`;
tells DNS cache a fake address
for `w3.org`.

2000 Bernstein: `.com` etc. are controlled by > 200 computers via server-name server trust. Many of these computers run old breakable servers.

Lesson to administrators: Don't use out-of-bailiwick names for DNS servers.

`.com` was then fixed. Eventually `w3.org` was fixed; this example no longer works.

2006 Ramasubramanian–Sirer "Perils of transitive trust": Problem is still widespread.

What's coming up

"Can we detect and eliminate forged packets?" — Second talk: Cryptography in DNS.

"What about buffer overflows and other software problems?" — Third talk: Secure design and coding for DNS.