Attacks on DNS

Cryptography in DNS

Secure design and coding for DNS

D. J. Bernstein
University of Illinois at Chicago

http://cr.yp.to
/talks.html#2009.03.02
/talks.html#2009.03.03
/talks.html#2009.03.04

1996: qmail 0.70.

1997: qmail 1.00.

1998: qmail 1.03.

1996: qmail 0.70.

1997: qmail 1.00.

1998: qmail 1.03.

1999: djbdns (dnscache) 0.60.

2000: djbdns (dnscache) 1.00.

2001: djbdns 1.05.

1996: qmail 0.70.

1997: qmail 1.00.

1998: qmail 1.03.

1999: djbdns (dnscache) 0.60.

2000: djbdns (dnscache) 1.00.

2001: djbdns 1.05.

2007: "Some thoughts on security after ten years of qmail 1.0."

1996: qmail 0.70.

1997: qmail 1.00.

1998: qmail 1.03.

1999: djbdns (dnscache) 0.60.

2000: djbdns (dnscache) 1.00.

2001: djbdns 1.05.

2007: "Some thoughts on security after ten years of qmail 1.0."

$>$ 1000000 of the Internet's SMTP servers are running qmail.

$>$ 4000000 of the Internet's second-level `*.com` names are published by djbdns.

1996: qmail 0.70.

1997: qmail 1.00.

1998: qmail 1.03.

1999: djbdns (dnscache) 0.60.

2000: djbdns (dnscache) 1.00.

2001: djbdns 1.05.

2007: "Some thoughts on security after ten years of qmail 1.0."

$>$ 1000000 of the Internet's SMTP servers are running qmail.
$>$ 4000000 of the Internet's second-level `*.com` names are published by djbdns.

No emergency upgrades, ever.

Some DNS buffer overflows

"CVE-2008-2469: Heap-based buffer overflow in the SPF_dns_resolv_lookup function in Spf_dns_resolv.c in libspf2 before 1.2.8 allows remote attackers to execute arbitrary code via a long DNS TXT record with a modified length field."

"CVE-2008-2357: Stack-based buffer overflow in the split_redraw function in split.c in mtr before 0.73, when invoked with the -p (aka –split) option, allows remote attackers to execute arbitrary code via a crafted DNS PTR record."

"CVE-2008-0530: Buffer overflow in Cisco Unified IP Phone 7940, 7940G, 7960, and 7960G running SCCP and SIP firmware might allow remote attackers to execute arbitrary code via a crafted DNS response."

“CVE-2008-0122: Off-by-one error in the inet_network function in libbind in ISC BIND 9.4.2 and earlier, as used in libc in FreeBSD 6.2 through 7.0-PRERELEASE, allows context-dependent attackers to cause a denial of service (crash) and possibly execute arbitrary code via crafted input that triggers memory corruption.”

“CVE-2007-2434: Buffer overflow in asnsp.dll in Aventail Connect 4.1.2.13 allows remote attackers to cause a denial of service (application crash) or execute arbitrary code via a malformed DNS query.”

“CVE-2007-2362: Multiple buffer overflows in MyDNS 1.1.0 allow remote attackers to (1) cause a denial of service (daemon crash) and possibly execute arbitrary code via a certain update, which triggers a heap-based buffer overflow in update.c; and (2) cause a denial of service (daemon crash) via unspecified vectors that trigger an off-by-one stack-based buffer overflow in update.c."

"CVE-2007-2187: Stack-based buffer overflow in eXtremail 2.1.1 and earlier allows remote attackers to execute arbitrary code via a long DNS response."

"CVE-2007-1866: Stack-based buffer overflow in the dns_decode_reverse_name function in dns_decode.c in dproxy-nexgen allows remote attackers to execute arbitrary code by sending a crafted packet to port 53/udp."

"CVE-2007-1748: Stack-based buffer overflow in the RPC interface in the Domain Name System (DNS) Server Service in Microsoft Windows 2000 Server SP 4, Server 2003 SP 1, and Server 2003

SP 2 allows remote attackers to execute arbitrary code via a long zone name containing character constants represented by escape sequences."

"CVE-2007-1465: Stack-based buffer overflow in dproxy.c for dproxy 0.1 through 0.5 allows remote attackers to execute arbitrary code via a long DNS query packet to UDP port 53."

"CVE-2006-5781: Stack-based buffer overflow in the handshake function in iodine 0.3.2 allows remote attackers to execute arbitrary code via a crafted DNS response."

"CVE-2006-4251: Buffer overflow in PowerDNS Recursor 3.1.3 and earlier might allow remote attackers to execute arbitrary code via a malformed TCP DNS query that prevents Recursor from properly calculating the TCP DNS query length."

"CVE-2006-3441: Buffer overflow in the DNS Client service in Microsoft Windows 2000 SP4, XP SP1 and SP2, and Server 2003 SP1 allows remote attackers to execute arbitrary code via a crafted record response. NOTE: while MS06-041 implies that there is a single issue, there are multiple vectors, and likely multiple vulnerabilities, related to (1) a heap-based buffer overflow in a DNS server response to the client, (2) a DNS server response with malformed ATMA records, and (3) a length miscalculation in TXT, HINFO, X25, and ISDN records."

"CVE-2005-2315: Buffer overflow in Domain Name Relay Daemon (DNRD) before 2.19.1 allows remote attackers to execute arbitrary code via a large number of large DNS packets with the Z and QR flags cleared."

"CVE-2005-0033 Buffer overflow in the code for recursion and glue fetching in BIND 8.4.4 and 8.4.5 allows remote attackers to cause a denial of service (crash) via queries that trigger the overflow in the q_usedns array that tracks nameservers and addresses."

"CVE-2004-1485: Buffer overflow in the TFTP client in InetUtils 1.4.2 allows remote malicious DNS servers to execute arbitrary code via a large DNS response that is handled by the gethostbyname function."

"CVE-2004-1317: Stack-based buffer overflow in doexec.c in Netcat for Windows 1.1, when running with the -e option, allows remote attackers to execute arbitrary code via a long DNS command."

"CVE-2004-0836: Buffer overflow in the mysql_real_connect function in MySQL 4.x before 4.0.21, and 3.x before 3.23.49, allows remote DNS servers to cause a denial of service and possibly execute arbitrary code via a DNS response with a large address length (h_length)."

"CVE-2004-0150: Buffer overflow in the getaddrinfo function in Python 2.2 before 2.2.2, when IPv6 support is disabled, allows remote attackers to execute arbitrary code via an IPv6 address that is obtained using DNS."

"CVE-2003-1377: Buffer overflow in the reverse DNS lookup of Smart IRC Daemon (SIRCD) 0.4.0 and 0.4.4 allows remote attackers to execute arbitrary code via a client with a long hostname.”

“CVE-2002-1219: Buffer overflow in named in BIND 4 versions 4.9.10 and earlier, and 8 versions 8.3.3 and earlier, allows remote attackers to execute arbitrary code via a certain DNS server response containing SIG resource records (RR).”

“CVE-2002-0910: Buffer overflows in netstd 3.07-17 package allows remote DNS servers to execute arbitrary code via a long FQDN reply, as observed in the utilities (1) linux-ftpd, (2) pcnfsd, (3) tftp, (4) traceroute, or (5) from/to.”

“CVE-2002-0906: Buffer overflow in Sendmail before 8.12.5, when configured to use a custom DNS map to query TXT records, allows remote attackers to cause a denial of service and possibly execute arbitrary code via a malicious DNS server."

"CVE-2002-0825: Buffer overflow in the DNS SRV code for nss_ldap before nss_ldap-198 allows remote attackers to cause a denial of service and possibly execute arbitrary code."

"CVE-2002-0698: Buffer overflow in Internet Mail Connector (IMC) for Microsoft Exchange Server 5.5 allows remote attackers to execute arbitrary code via an EHLO request from a system with a long name as obtained through a reverse DNS lookup, which triggers the overflow in IMC's hello response."

"CVE-2002-0684: Buffer overflow in DNS resolver functions that perform lookup of network names and addresses, as used in BIND 4.9.8 and ported to glibc 2.2.5 and earlier, allows remote malicious DNS servers to execute arbitrary code through a subroutine used by functions such as getnetbyname and getnetbyaddr."

"CVE-2002-0651: Buffer overflow in the DNS resolver code used in libc, glibc, and libbind, as derived from ISC BIND, allows remote malicious DNS servers to cause a denial of service and possibly execute arbitrary code via the stub resolvers."

"CVE-2002-0423: Buffer overflow in efingerd 1.5 and earlier, and possibly up to 1.61, allows remote attackers to cause a denial of service and possibly execute arbitrary code via a finger request from an IP address with a long hostname that is obtained via a reverse DNS lookup."

“CVE-2002-0332: Buffer overflows in xtell (xtelld) 1.91.1 and earlier, and 2.x before 2.7, allows remote attackers to execute arbitrary code via (1) a long DNS hostname that is determined using reverse DNS lookups, (2) a long AUTH string, or (3) certain data in the xtell request.”

“CVE-2002-0180: Buffer overflow in Webalizer 2.01-06, when configured to use reverse DNS lookups, allows remote attackers to execute arbitrary code by connecting to the monitored web server from an IP address that resolves to a long hostname.”

“CVE-2002-0163: Heap-based buffer overflow in Squid before 2.4 STABLE4, and Squid 2.5 and 2.6 until March 12, 2002 distributions, allows remote attackers to cause a denial of service, and possibly execute arbitrary code, via compressed DNS responses."

"CVE-2002-0029: Buffer overflows in the DNS stub resolver library in ISC BIND 4.9.2 through 4.9.10, and other derived libraries such as BSD libc and GNU glibc, allow remote attackers to execute arbitrary code via DNS server responses that trigger the overflow in the (1) getnetbyname, or (2) getnetbyaddr functions."

"CVE-2001-0207: Buffer overflow in bing allows remote attackers to execute arbitrary commands via a long hostname, which is copied to a small buffer after a reverse DNS lookup using the gethostbyaddr function."

"CVE-2001-0050: Buffer overflow in BitchX IRC client allows remote attackers to cause a denial of service and possibly execute arbitrary commands via an IP address that resolves to a long DNS hostname or domain name."

"CVE-2001-0029: Buffer overflow in oops WWW proxy server 1.4.6 (and possibly other versions) allows remote attackers to execute arbitrary commands via a long host or domain name that is obtained from a reverse DNS lookup."

"CVE-2001-0011: Buffer overflow in nslookupComplain function in BIND 4 allows remote attackers to gain root privileges."

"CVE-2001-0010: Buffer overflow in transaction signature (TSIG) handling code in BIND 8 allows remote attackers to gain root privileges."

"CVE-2000-0405: Buffer overflow in L0pht AntiSniff allows remote attackers to execute arbitrary commands via a malformed DNS response packet."

"CVE-1999-1321: Buffer overflow in ssh 1.2.26 client with Kerberos V enabled could allow remote attackers to cause a denial of service or execute arbitrary commands via a long DNS hostname that is not properly handled during TGT ticket passing."

"CVE-1999-1060: Buffer overflow in Tetrix TetriNet daemon 1.13.16 allows remote attackers to cause a denial of service and possibly execute arbitrary commands by connecting to port 31457 from a host with a long DNS hostname."

"CVE-1999-0833: Buffer overflow in BIND 8.2 via NXT records."

“CVE-1999-0299: Buffer overflow in FreeBSD lpd through long DNS hostnames.”

“CVE-1999-0101: Buffer overflow in AIX and Solaris ”gethostbyname” library call allows root access through corrupt DNS host names.”

“CVE-1999-0009: Inverse query buffer overflow in BIND 4.9 and BIND 8 Releases.”

More security problems

What we've learned so far:

Attacker easily breaks DNS
through packet forgery,
thanks to bad protocol.

Or through buffer overflows,
thanks to bad software.

But wait, there's more!

"CVE-2008-5077: OpenSSL 0.9.8i and earlier does not properly check the return value from the EVP_VerifyFinal function, which allows remote attackers to bypass validation of the certificate chain via a malformed SSL/TLS signature for DSA and ECDSA keys."

This bug (announced 2009.01) allowed trivial forgery of DNSSEC DSA signatures.

"CVE-2008-5077: OpenSSL 0.9.8i and earlier does not properly check the return value from the EVP_VerifyFinal function, which allows remote attackers to bypass validation of the certificate chain via a malformed SSL/TLS signature for DSA and ECDSA keys."

This bug (announced 2009.01) allowed trivial forgery of DNSSEC DSA signatures.

. . . which was a big deal for the 20 people on Earth who use DNSSEC DSA signatures.

"CVE-2007-2925: The default access control lists (ACL) in ISC BIND 9.4.0, 9.4.1, and 9.5.0a1 through 9.5.0a5 do not set the allow-recursion and allow-query-cache ACLs, which allows remote attackers to make recursive queries and query the cache."

Documentation said that cache was (by default) usable only by local network.

This bug hurt confidentiality: attackers easily see which names have been looked up.

Also hurt availability: e.g., attackers easily use cache as DDoS amplifier.

“2004 Symantec Enterprise Firewall DNSD DNS Cache Poisoning Vulnerability: Dnsd does not ensure that the data returned from a remote DNS server contains related information about the requested records. An attacker could exploit this vulnerability to deny service to legitimate users by redirecting traffic to inappropriate hosts. Man-in-the-middle attacks, impersonation of sites, and other attacks may be possible.”

Nobody had told Symantec about the bailiwick fix.

"CVE-2003-0914: ISC BIND 8.3.x before 8.3.7, and 8.4.x before 8.4.3, allows remote attackers to poison the cache via a malicious name server that returns negative responses with a large TTL (time-to-live) value."

Cache wouldn't allow `microsoft.com` servers to declare an address for `www.google.com`, but would allow `microsoft.com` servers to declare *nonexistence* of `www.google.com`.

"CVE-2001-0497: dnskeygen in BIND 8.2.4 and earlier, and dnssec-keygen in BIND 9.1.2 and earlier, set insecure permissions for a HMAC-MD5 shared secret key file used for DNS Transactional Signatures (TSIG), which allows attackers to obtain the keys and perform dynamic DNS updates."

"Perform dynamic DNS updates" = "change all your DNS data."
Lack of confidentiality of keys compromises integrity of data.

Exploitable on multiuser machines, and on machines where another server has been taken over by an attacker.

Isn't this embarrassing?

Public goal of
computer-security research:
Protection of
the average home computer;
critical-infrastructure computers;
and everything in between.

Isn't this embarrassing?

Public goal of
computer-security research:
Protection of
the average home computer;
critical-infrastructure computers;
and everything in between.

Secret goal of
computer-security research:
Lifetime employment
for computer-security researchers.

ECRYPT is a consortium of European crypto researchers.

eSTREAM is the “ECRYPT Stream Cipher Project.”

2004.11: eSTREAM calls for submissions of stream ciphers. Receives 34 submissions from 97 cryptographers around the world.

2008.04: After two hundred papers and several conferences, eSTREAM selects portfolio of 4 SW ciphers and 4 HW ciphers.

2008.09: 4 SW, 3 HW.

eSTREAM says: The HW ciphers are aimed at "deployment on passive RFID tags or low-cost devices such as might be used in sensor networks. Such devices are exceptionally constrained in computing potential . . . [Keys are] 80 bits which we believe to be adequate for the lower-security applications where such devices might be used."

Obviously these ciphers will be built into many chips over the next 5 or 10 years.

Iain Devlin, Alan Purvis,
"Assessing the security
of key length," 2007:
Buy FPGAs for $3 million;
break 80-bit keys in 1 year.
Or: $165 million; 1 week.

Cost will continue to drop.
Will come within reach
of more and more attackers.

Same story in public-key crypto.
1024-bit RSA will be broken.
160-bit ECC will be broken.

So users will pay us for
96-bit ciphers and 192-bit ECC.
And then those will be broken.
Continue for decades.

So users will pay us for
96-bit ciphers and 192-bit ECC.
And then those will be broken.
Continue for decades.

Success: Lifetime employment!

This is not a new pattern.
Consider, e.g., 56-bit DES,
or 48-bit Mifare CRYPTO1,
or MD5, or Keeloq.
All breakable by brute force,
and often by faster methods.

Naive conclusion
from all these attacks:
“There is no such thing as
100% secure cryptography!
Crypto breaks are inevitable!”

Naive conclusion
from all these attacks:
"There is no such thing as
100% secure cryptography!
Crypto breaks are inevitable!"

This conclusion is unjustified
and almost certainly wrong.

Nobody has found patterns
in output of 256-bit AES.
Most cryptographers think
that nobody ever will.

“AES software leaks keys to cache-timing attacks!”

True, but we know how to eliminate this problem by (for example) bitslicing.

“AES software leaks keys
to cache-timing attacks!”

True, but we know how
to eliminate this problem
by (for example) bitslicing.

“Maybe secret-key crypto is okay,
but large quantum computers will
kill public-key cryptography!”

If large quantum computers are
built, they’ll break RSA and ECC,
but we have replacements.
See PQCrypto workshops.

Enough crypto for this talk.
How about all the rest
of computer security?

Flood of successful attacks,
even more than in crypto.

Conventional wisdom:
We'll never stop the flood.

Viega and McGraw: "Because
there is no such thing as 100%
security, attacks will happen."

Schneier: "Software bugs
(and therefore security flaws)
are inevitable."

Analogy:
“We’ll never build a tunnel
from England to France.”

Why not? “It’s impossible.”
Or: “Maybe it’s possible,
but it’s much too expensive.”

Engineer’s reaction:
How expensive is it?
How big a tunnel *can* we build?
How can we reduce the costs?

Eventually a tunnel *was* built
from England to France.

Here's what I think:

Invulnerable software systems
can and *will* be built,
and will become standard.

Most "security" research today
doesn't aim for invulnerability,
doesn't contribute to it,
and will be discarded once
we have invulnerable software.

Eliminating bugs

Bug: Software feature that violates the user's requirements.

Security hole: Software feature that violates the user's security requirements.

Eliminating bugs

Bug: Software feature that violates the user's requirements.

Security hole: Software feature that violates the user's security requirements.

Every security hole is a bug.

Is every bug a security hole?
No. Many user requirements are not security requirements.

Everyone agrees that
we *can* eliminate all bugs
(and therefore eliminate
all security holes)
in *extremely small* programs.

What about larger programs?

Space-shuttle software:
"The last three versions of the
program—each 420000 lines
long—had just one error each.
The last 11 versions of this
software had a total of 17 errors."

Estimate bug rate of
software-engineering processes
by carefully reviewing code.
(Estimate is reliable enough;
"all bugs are shallow.")

Meta-engineer processes
that have lower bug rates.
Note: progress is *quantified*.

Well-known example:
Drastically reduce bug rate
of typical engineering process
by adding coverage tests.

Example where djbdns did well:
“Don’t parse.”

Typical user interfaces
copy “normal” inputs and
quote “abnormal” inputs.
Inherently bug-prone:
simpler copying is wrong
but passes “normal” tests.
Example (1996 Bernstein):
format-string danger in `logger`.

djbdns’s internal file structures
and program-level interfaces
don’t have exceptional cases.
Simplest code is correct code.

Example where djbdns did badly: integer arithmetic.

In C et al., `a + b` *usually* means exactly what it says, but *occasionally* doesn't.

To detect these occasions, need to check for overflows. Extra work for programmer.

To guarantee sane semantics, extending integer range and failing only if out of memory, need to use large-integer library. Extra work for programmer.

The closest that qmail
has come to a security hole
(Guninski): potential overflow
of an unchecked counter.

Fortunately, counter growth
was limited by memory
and thus by configuration,
but this was pure luck.

Anti-bug meta-engineering:
Use language where `a + b`
means exactly what it says.

Security hole in djbdns (2009.02.25 Dempsky): overflow of an unchecked counter copied into compressed packets.

Decompressing those packets produces incorrect results. Problem for packets that mix data from different sources.

Impact: If administrator of `lsec.be` copies `foo.lsec.be` from an untrusted third party, the third party can control cache entries for `lsec.be`, not just `foo.lsec.be`.

"Large-integer libraries are slow!"

That's a silly objection.
We need invulnerable systems,
and we need them today,
even if they are $10\times$ slower
than our current systems.
Tomorrow we'll make them faster.

Most CPU time is consumed
by a very small portion
of all the system's code.
Most large-integer overheads
are removed by smart compilers.
Occasional exceptions can be
handled manually at low cost.

More anti-bug meta-engineering
examples in my qmail paper:
automatic array extensions;
partitioning variables
to make data flow visible;
automatic updates of
"summary" variables;
abstraction for testability.

"Okay, we can achieve
much smaller bug rates.
But in a large system
we'll still have many bugs,
including many security holes!"

Eliminating code

Measure code rate of
software-engineering processes.

Meta-engineer processes
that spend less code
to get the same job done.
Note: progress is *quantified*.

This is another classic topic
of software-engineering research.
Combines reasonably well
with reducing bug rate.

Example where qmail did well: reusing access-control code.

A story from twenty years ago: My `.forward` ran a program creating a new file in `/tmp`. Surprise: the program was sometimes run under another uid!

How Sendmail handles `.forward`: Check whether user can read it. (Prohibit symlinks to secrets!) Extract delivery instructions. Keep track (often via queue file) of instructions and user.

Many disastrous bugs here.

Kernel already tracks users.
Kernel already checks readability.
Why not reuse this code?

How qmail delivers to a user:
Start `qmail-local`
under the right uid.

When `qmail-local` reads
the user's delivery instructions,
the kernel checks readability.
When `qmail-local` runs a
program, the kernel assigns
the same uid to that program.
No extra code required!

Example where qmail and djbdns did badly: exception handling.

djbdns has thousands of conditional branches. About half are simply checking for temporary errors.

Same for qmail. Easy to get wrong: e.g., "if `ipme_init()` returned `-1`, `qmail-remote` would continue" (fixed in qmail 0.92).

Easily fixed by better language.

More small-code meta-engineering examples in my qmail paper: identifying common functions; reusing network tools; reusing the filesystem.

"Okay, we can build a system with less code, and write code with fewer bugs. But in a large system we'll still have bugs, including security holes!"

Eliminating trusted code

Can architect computer systems
to place most of the code
into *untrusted* prisons.

Definition of "untrusted":
no matter what the code does,
no matter how badly it behaves,
no matter how many bugs it has,
it cannot violate the
user's security requirements.

Measure *trusted* code volume,
and meta-engineer processes
that reduce this volume.
Note: progress is *quantified*.

Warning: “Minimizing privilege” rarely eliminates trusted code.

Every security mechanism, no matter how pointless, says it’s “minimizing privilege.” This is not a useful concept.

qmail and djbdns did very badly here: almost all code is trusted. I spent considerable effort “minimizing privilege”; stupid! This distracted me from eliminating trusted code.

What *are* the user's security requirements?

My fundamental requirement: The system keeps track of sources of data.

When the system is asked for data from source X, it does not allow data from source Y to influence the result.

Example: When I view an account statement from my bank, I am not seeing data from other web pages or email messages.

There is no obstacle to centralization and minimization of source-tracking code.
Can and should be small enough to eliminate all bugs.

Classic military TCBs used very few lines of code to track (e.g.) Top Secret data.
VAX VMM Security Kernel had < 50000 lines of code.

Minor programming problem to support arbitrary source labels instead of Top Secret etc.

"Doesn't the UNIX/Linux kernel already track sources?"

If I log into a system,
the kernel copies my uid
to my login process,
to other processes I start,
to files I create, etc.

But if I transfer data
to another user's processes—
through the network or a file—
the kernel doesn't remember
that I'm the source.

Source tracking today
is implemented by programmers
writing web browsers, mail clients,
PHP scripts, etc.

All of this code is trusted.
All other code in these programs
is also trusted, thanks to
nonexistent internal partitioning.

Your laptop has tens of millions
of lines of trusted code written by
thousands of novice programmers.
A screwup anywhere in that code
can violate security requirements.

"Teach every programmer
how to write secure code."

No, no, no!
If every programmer
is writing trusted code
then the system has
far too much trusted code.

We need new systems
with far less trusted code.
Enforce security in TCB
so that typical programmers
don't have to worry about it.

Imagine a TCB tracking sources.

Alice's process reads Bob's file.
TCB automatically labels
process as /Alice/Bob.

Process creates a file.
TCB automatically labels
file as /Alice/Bob
(and refuses to touch a file
labelled only /Alice).

Joe's process reads the file
and another file from Charlie.
Process then has two labels:
/Joe/Alice/Bob; /Joe/Charlie.

A web-browsing process
that reads from mbna.com
and from nytimes.com
will have both labels.
TCB won't allow process
to write under just one label.

Solution 1: Rewrite browser
in the classic UNIX style,
one process for each page.

Solution 2: Track sources
separately for each variable
inside the web-browsing process.

Suppose a DNS cache receives a packet from the `microsoft.com` servers with `www.google.com` data.

Packet is a string. TCB attaches to the string a `microsoft.com` source label.

Packet-parsing code extracts `www.google.com` information from the packet.

TCB automatically copies the `microsoft.com` source label to derived variables such as the string `www.google.com`.

The cache remembers information in a big associative array.

Post-packet-parsing code tries to store `www.google.com` in the associative array.

Cache policy: only root, `.com`, `.google.com`, `.www.google.com` are allowed to store `www.google.com` information.

TCB enforces this policy: sees `microsoft.com` label, refuses `www.google.com` data.

How much code is required
for a TCB that enforces
source-tracking policy
against all other code?

How many bugs do we expect
in a TCB of this size?
Note: Can afford expensive
techniques to reduce bug rate.

If code volume is small enough,
and bug rate is small enough,
then we will be confident
that sources are tracked.