Robustness in Error Handling in BIND 10

Shane Kerr, ISC

shane@isc.org

T-DOSE 2009

DNS Servers 101

101

adj. (chiefly US) Basic, beginner, starting from scratch.

DNS

Turns a name:
WWW.T-DOSE.ORG

Into a number:
217.21.240.105

DNS Query

```
shane@shane-asus-laptop:~$ sudo tcpdump -n -i wlan0 -s 0 -t port 53
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan0, link-type EN10MB (Ethernet), capture size 65535 bytes
IP 172.29.1.211.47670 > 172.29.1.3.53: 40643+ A? www.t-dose.org. (32)
IP 172.29.1.3.53 > 172.29.1.211.47670: 40643 1/3/0 A 217.21.240.105 (129)
IP 172.29.1.211.39821 > 172.29.1.3.53: 22766+ AAAA? www.t-dose.org. (32)
IP 172.29.1.3.53 > 172.29.1.211.39821: 22766 0/1/0 (104)
```

DNS Resolvers

Stub Resolvers:

- Send DNS Query to Recursive Resolvers

Recursive Resolvers:

- Receive DNS Query from Stub Resolvers (or anyone)
- Send DNS Query to Authoritative Servers

DNS Servers

Either:

- Recursive resolver
- Authoritative server
- Both (icky)

DNS Servers Summary

1. DNS is useful fo[illegible]ning computer names into n[illegible]
2. DNS queries [illegible] a simple UDP packet excha[illegible]
3. DNS resolvers [illegible]ner stub resolvers or recursive r[illegible]
4. DNS servers a[illegible]sive resolvers, authoritative servers, or both.

This Talk

Introductions

- BIND 9
- BIND 10
- ISC
- Me

History

https://www.isc.org/software/bind/history

Engineering BIND 9 Security

State-of-the art (circa 2000)

- C programming language (for performance)
- Multi-threaded (for scaling)
- Design & code standards
- Code reviews
- chroot(), setuid(), system-specific features (like Linux capabilities)
- Design-by-contract

Design-by-Contract

- Technique from Bertrand Meyer
  - Built into Eiffel programming language
- Object Oriented approach
- Defines explicit conditions, the "Contract"
  - Preconditions
  - Postconditions
  - Invariants
- Makes writing deterministic systems possible

Contract violations raise exceptions

Design-by-Contract in BIND 9

- C is a mess:
  - C has no preconditions.
  - C has no postconditions.
  - C has no objects, must less invariants.

- BIND 9's approach:

  REQUIRE(), ENSURE(), INVARIANT(), INSIST()

Programmer errors cause BIND 9 to exit

BIND 9 Security Report

Not Too Shabby

https://www.isc.org/advisories/bind

13 advisories in 9 years

- 1 non-issue
- 1 failure to check return from library call
- 1 simple programmer error
- 2 serious design problems
- 8 design-by-contract errors

So, 2/3 of the actual security problems were aggravated due to design-by-contract

BIND 10

- Modularity
- Well-defined APIs and libraries
- Customization
- Cluster support
- Better runtime control
- Resilience

Modern Languages

Requirements:

- Mainstream language
- C-like performance
- Supports exceptions

Selection:

- Python (when possible)
- C++ (when necessary)

Modern Languages: Benefits

- Objects
- Well-tested libraries
  - End of NIH syndrome!
- Garbage collection (in Python)
- Compact, easier-to-read code
- Exceptions

Without Exceptions: Checking Return Values

```
fp = fopen("foo.txt", "w");
if (fp == NULL) {
    return ERROR_CODE;
}
for (i=0; i<10; i++) {
    if (fprintf(fp, "%d\n", i) < 0) {
        fclose(fp);
        remove("foo.txt");
        return ERROR_CODE;
    }
}
if (fclose(fp) != 0) {
    remove("foo.txt");
    return ERROR_CODE;
}

return SUCCESS_CODE;
```

With Exceptions

```
try:
    fp = open("bar.txt", "w")
    for i in range(10):
        fp.write("%d\n" % I)
    fp.close()
    return SUCCESS_CODE
except IOError, e:
    if fp:
        fp.close()
    os.remove("bar.txt")
    return ERROR_CODE
```

Passing Exceptions Up the Stack

```
try {
    foo();
} catch (const char *e) {
    puts(e);
}

void foo(void)
{
    bar();
}
void bar(void)
{
    throw "exception going through foo()";
}
```

Exceptions: Wholesome Goodness

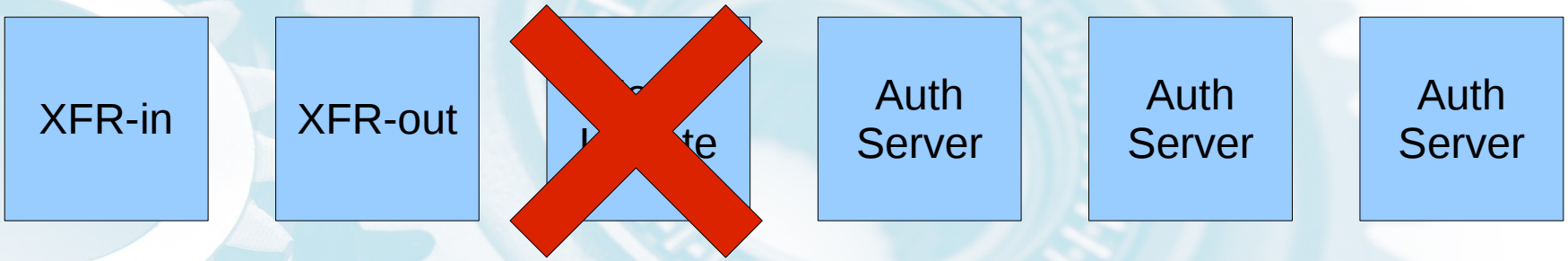
- Makes code easier to read
- Minimize unchecked errors (one of our BIND 9 security bugs)
- Allows errors to be handled as close to problem as makes sense

  Including design-by-contract errors!

- Some care required - not a silver bullet

Multiple Processes

- Separate process per type of task
- Multiple process for some tasks



Limitations of Techniques

- Analysis required for each piece of code
- Process failures always have a cost
- Shared data can affect all processes
- Failure at some operations will have long-term effects
- Still, large classes of failures will be eliminated

BIND 10 Robustness

Old techniques:

- Design & code standards
- Code reviews
- `chroot()`, `setuid()`, system-specific features (like Linux capabilities)
- Design-by-contract

BIND 10 Robustness

New approaches:

- Modern languages (standard libraries, garbage collection, simpler code)
- Exceptions
- Multiple processes

BIND 10 will be significantly more robust than BIND 9.

Questions & Answers