Testing TLS

Hubert Kario
Quality Engineer
7-02-2015

2014

Heartbleed

OpenSSL CCS bug

gotofail

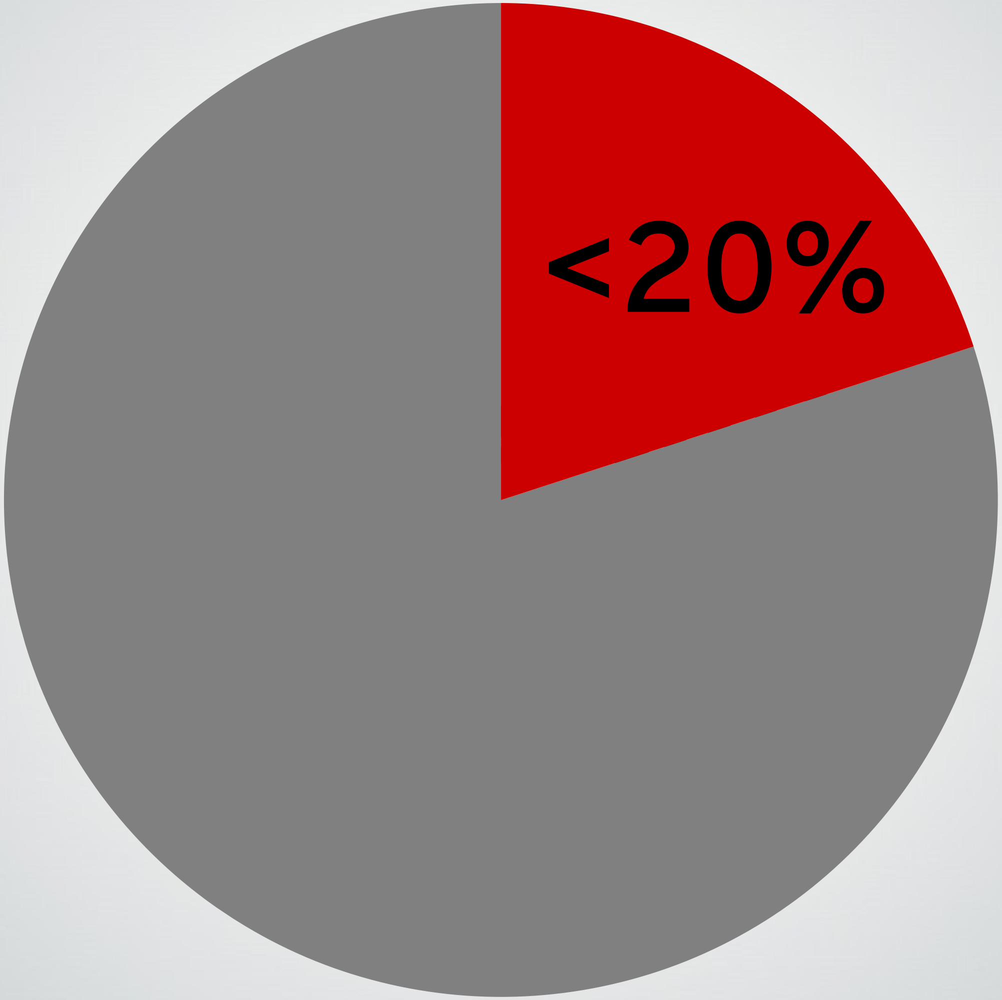
Certificate handling

CVE-2014-6321 in schannel

Testing

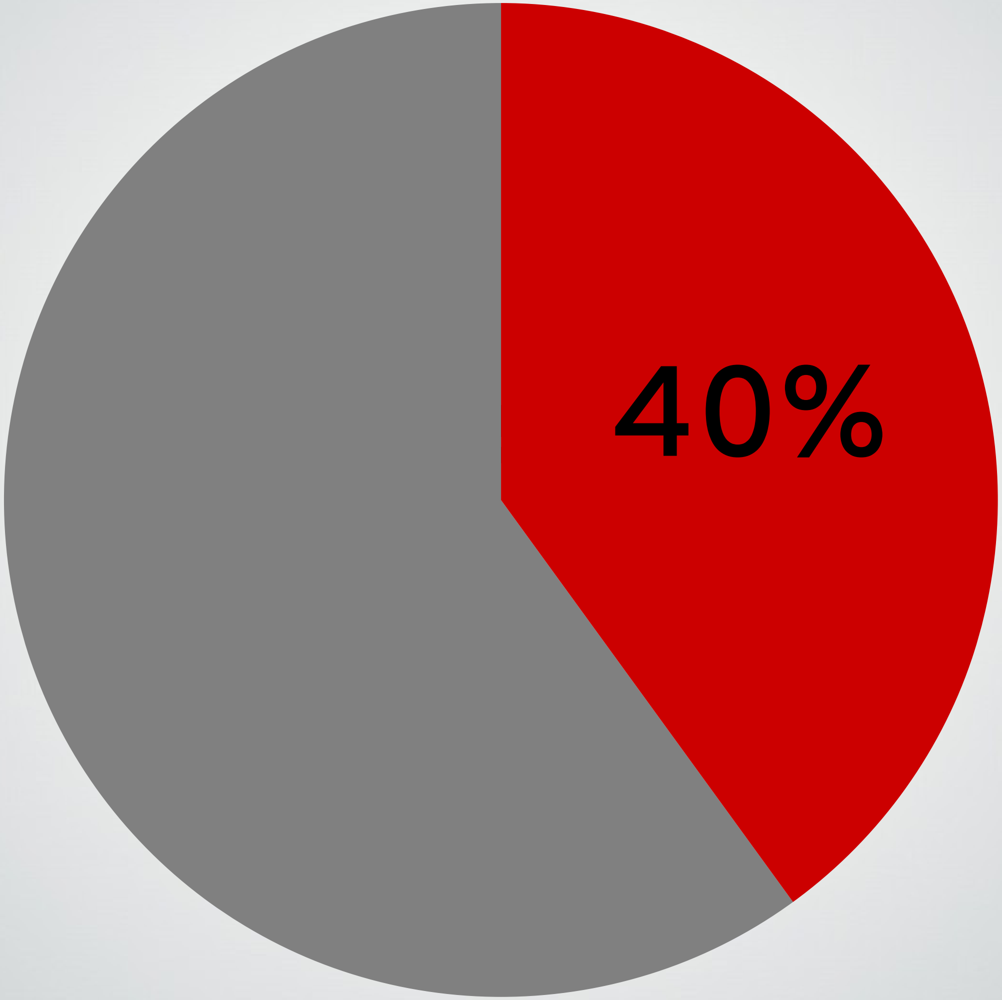
Legacy code

Test plans



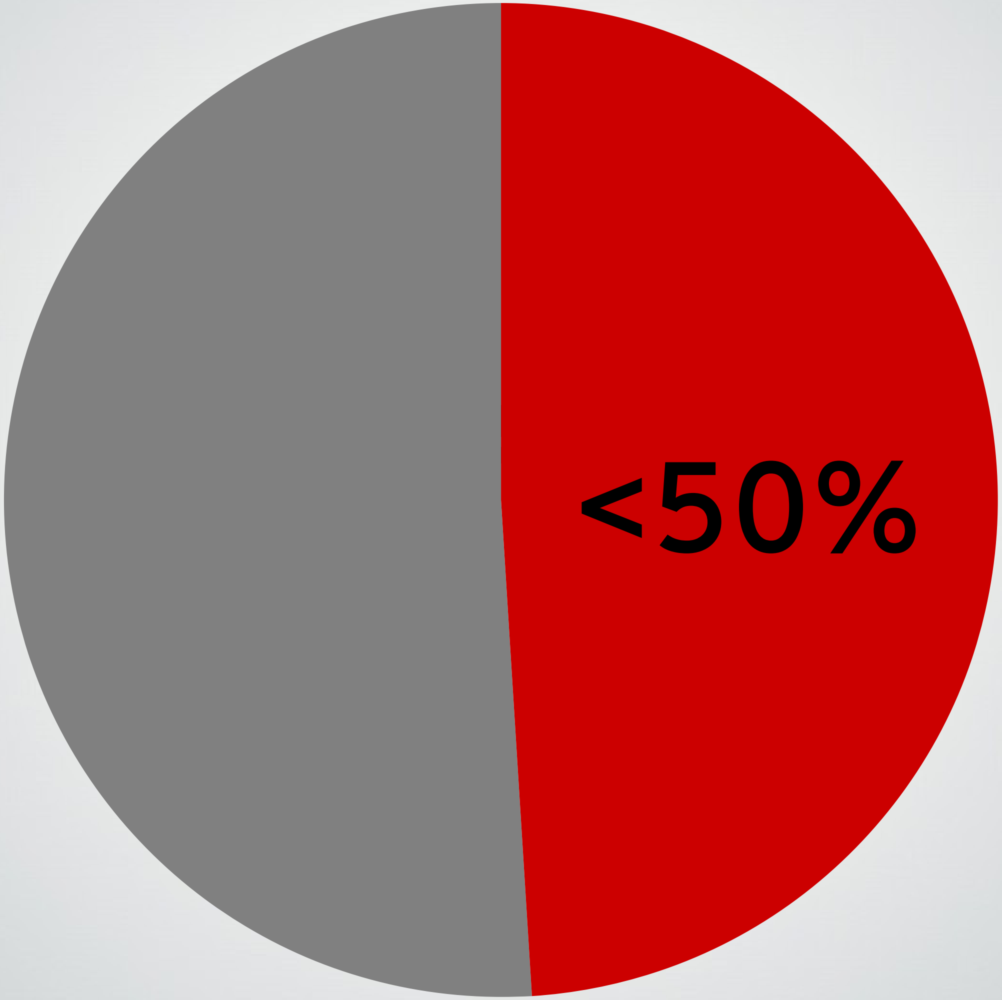
Source: Farooq & Quadri, 2011

Test tools



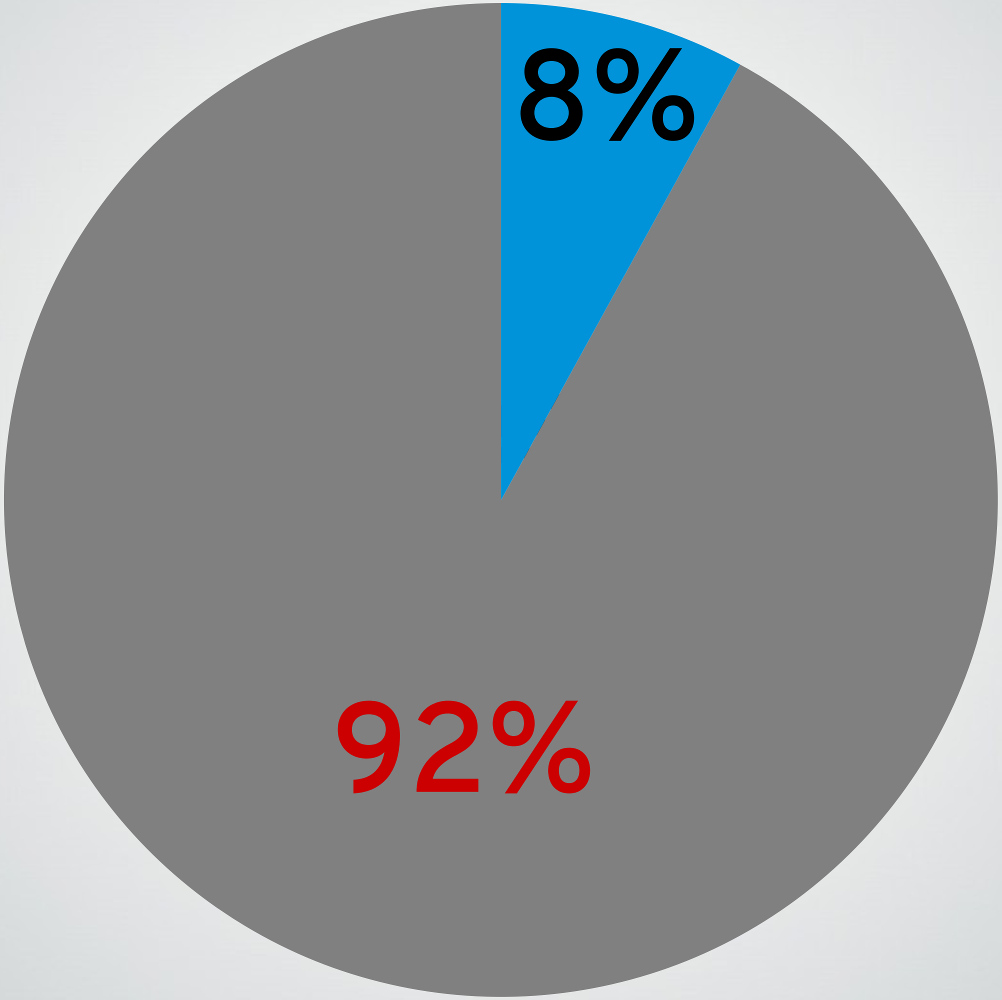
Source: Farooq & Quadri, 2011

Code coverage tools



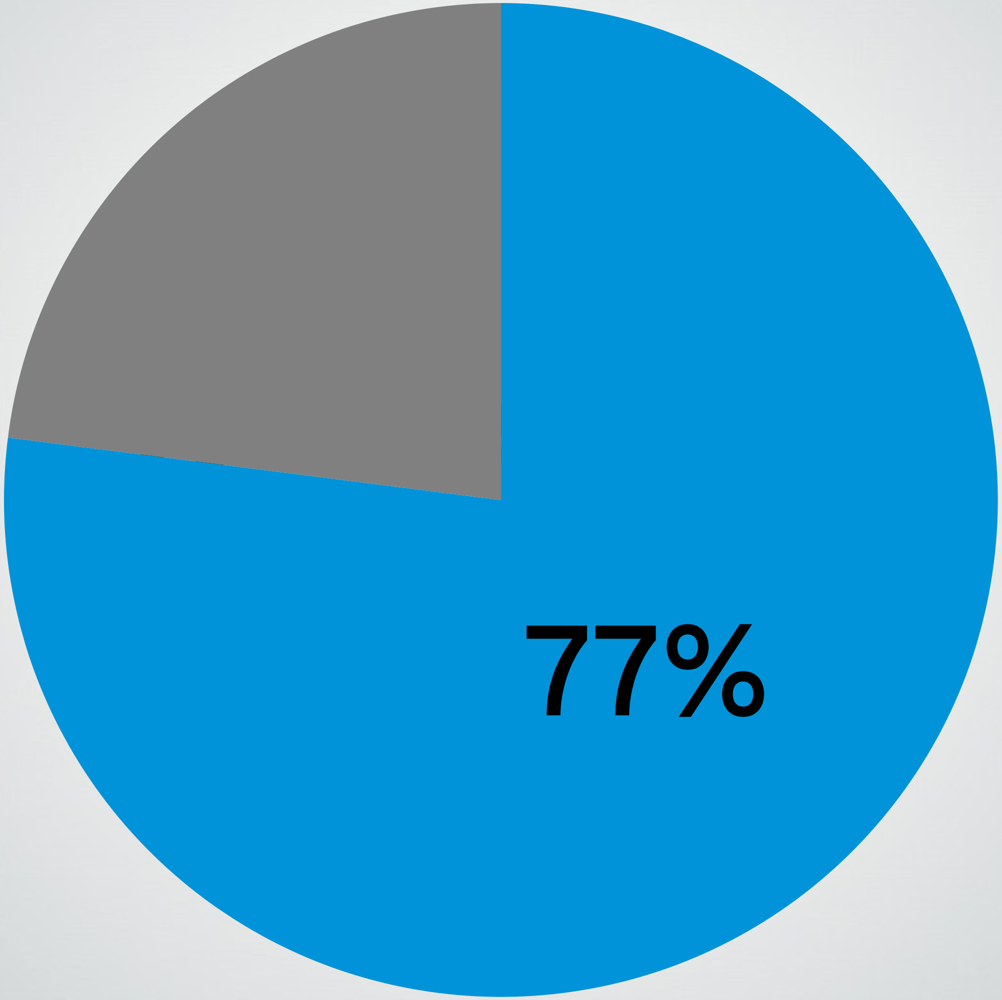
Source: Farooq & Quadri, 2011

Bad error handling



Source: Yan, Luo, Zhuang, Rodrigues, et al, 2014

Unit tests vs bugs

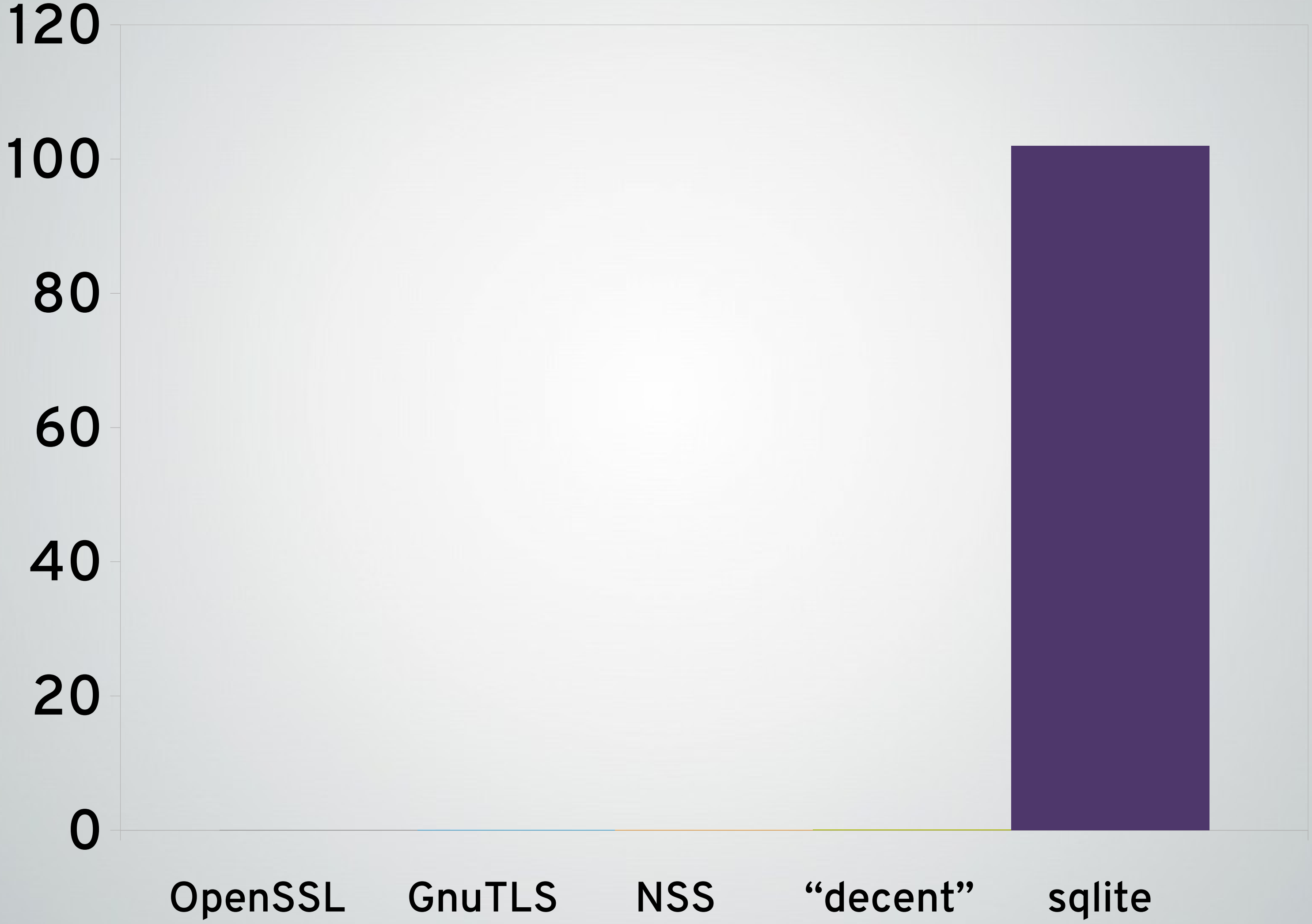


Source: Yan, Luo, Zhuang, Rodrigues, et al, 2014

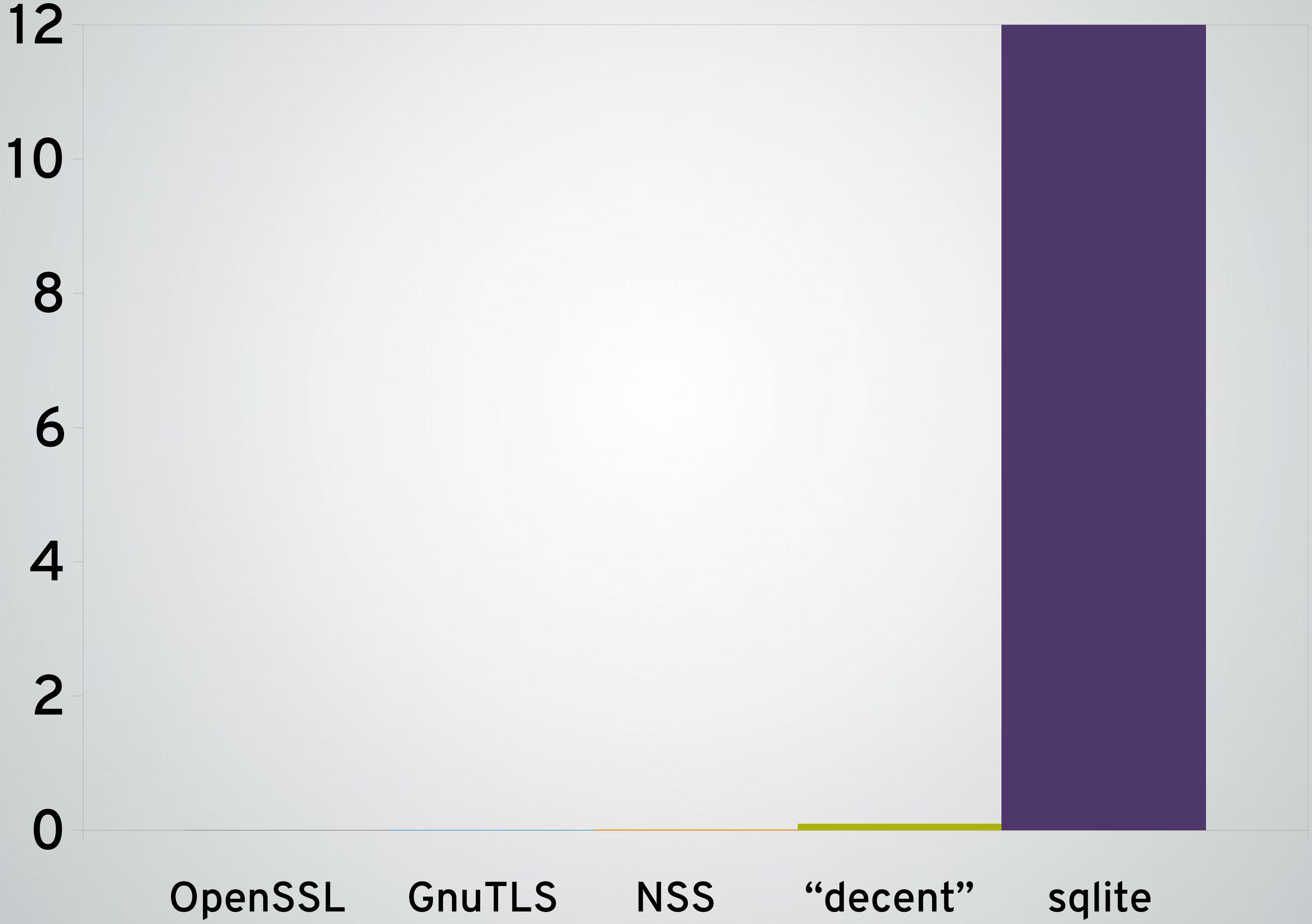
OSS TLS libraries

	OpenSSL	NSS	GnuTLS
Framework	✗	✓	✓
N° tests	100-200	>7000	100-200
Negative tests	✗	✓	✗

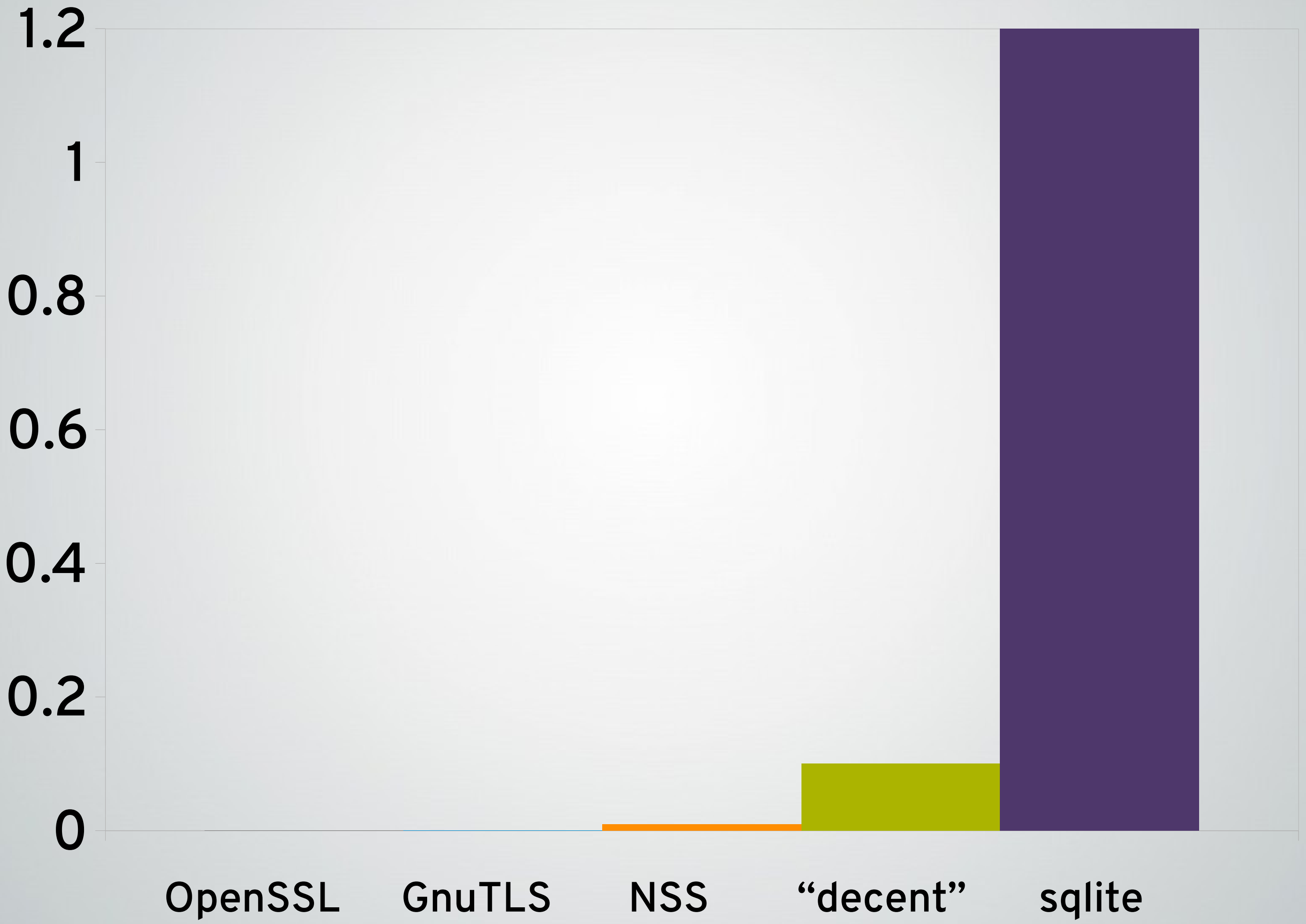
Test coverage



Test coverage



Test coverage



Why is that?

X.509 and ASN.1

Invisible bugs

Libraries and bad data

Fuzzy testing

Fixing the problem

Duplication of effort

Existing fuzzers

TLS testing and fuzzing

Full TLS handshake

```
ClientHello                  -------->
                                                  ServerHello
                                                  Certificate
                                            ServerKeyExchange
                                           CertificateRequest
                            <--------         ServerHelloDone
Certificate
ClientKeyExchange
CertificateVerify
ChangeCipherSpec
Finished                    -------->
                                             ChangeCipherSpec
                            <--------                Finished
ApplicationData            <-------->         ApplicationData
```

TLS RSA handshake

```
ClientHello                  -------->
                                                 ServerHello
                                                 Certificate
                            <--------        ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished                     -------->
                                            ChangeCipherSpec
                            <--------               Finished
ApplicationData            <-------->        ApplicationData
```

Simple fuzzing

```
ClientHello                    -------->
                                                    ServerHello
                                                    Certificate
                               <--------         ServerHelloDone
ClientKeyExchange     ⚡
   ...
   Encrypted PreMasterSecret
ChangeCipherSpec
Finished                       -------->
                                               ChangeCipherSpec
                               <--------                Finished
```

Simple fuzzing

```
ClientHello                --------->
                                                          ServerHello
                                                          Certificate
                                                    ServerKeyExchange
                                              (DHE parameters)
                                                         signature
                          <---------                  ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished                  --------->
                                                    ChangeCipherSpec
                                                             Finished
ApplicationData          <--------->              ApplicationData
```

Message injection

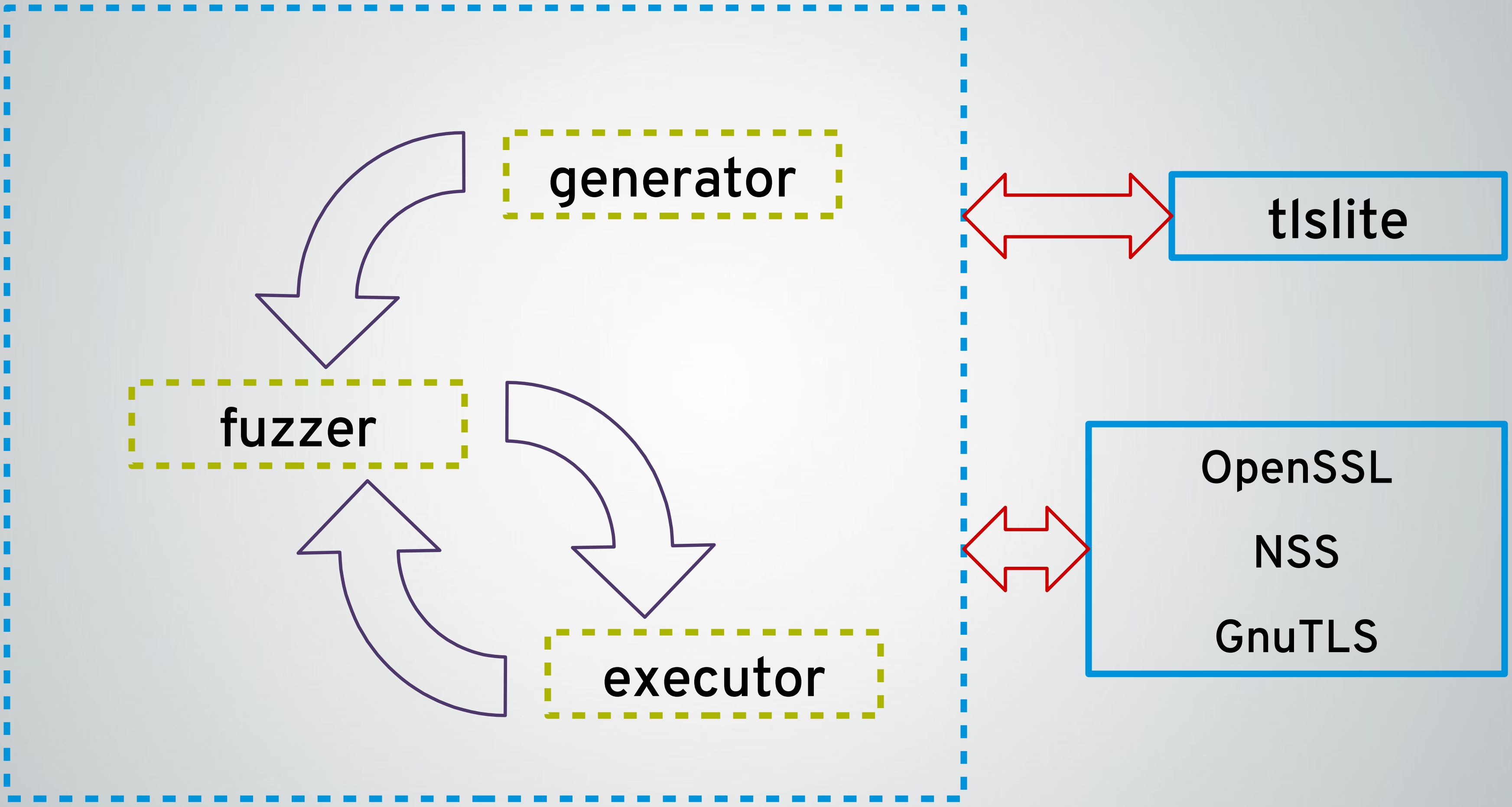
```
ClientHello                 -------->
                                                    ServerHello
                                                    Certificate
                           <--------            ServerHelloDone
Certificate
ClientKeyExchange
ChangeCipherSpec
Finished                   -------->
                                               ChangeCipherSpec
                           <--------                   Finished
ApplicationData           <-------->            ApplicationData
```

Message injection

```
ClientHello              -------->
                                                 ServerHello
                                                 Certificate
                        <--------            ServerHelloDone
ChangeCipherSpec
ClientKeyExchange
ChangeCipherSpec
Finished                 -------->
                                            ChangeCipherSpec
                        <--------                   Finished
ApplicationData         <-------->           ApplicationData
```

tlsfuzzer

Architecture (planned)



Servers first

Continuous Integration

Questions?

Feedback: http://devconf.cz/f/108

Contact: hkario@redhat.com

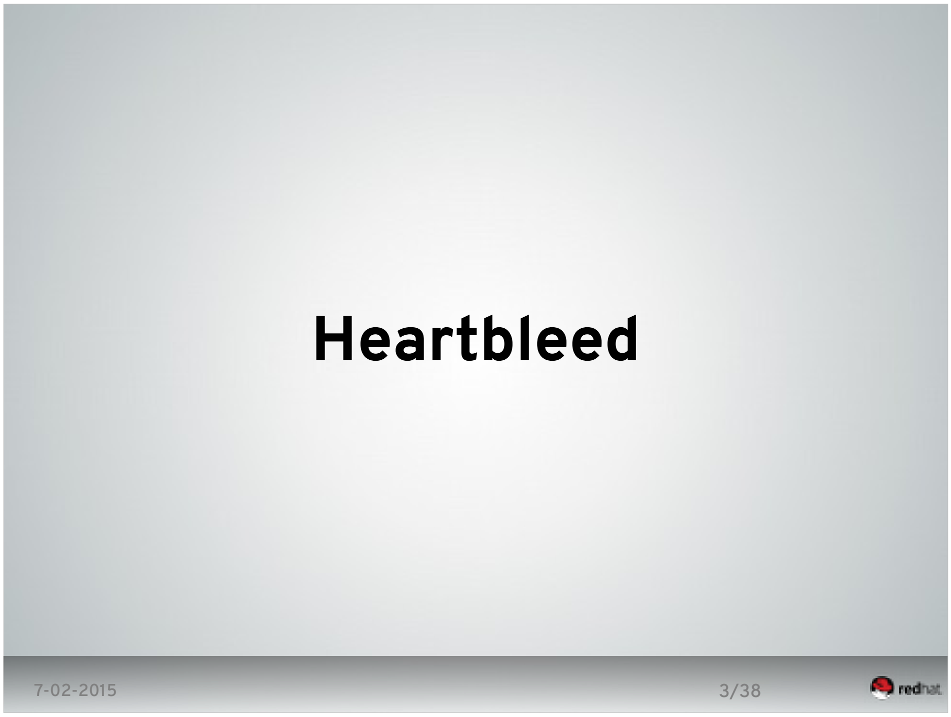
Project: https://github.com/tomato42/tlsfuzzer

Testing TLS

Hubert Kario
Quality Engineer
7-02-2015

2014

“Few” things happened last year. In short: every big cryptographic library had some critical flaws.



Heartbleed

7-02-2015 3/38

Which was an OpenSSL bug in handling rarely used TLS protocol feature

OpenSSL CCS bug

Where the Change Cipher Spec was accepted earlier

gotofail

7-02-2015 5/38

Apple Secure Transport bug in handling signatures of DHE and ECDHE key exchange

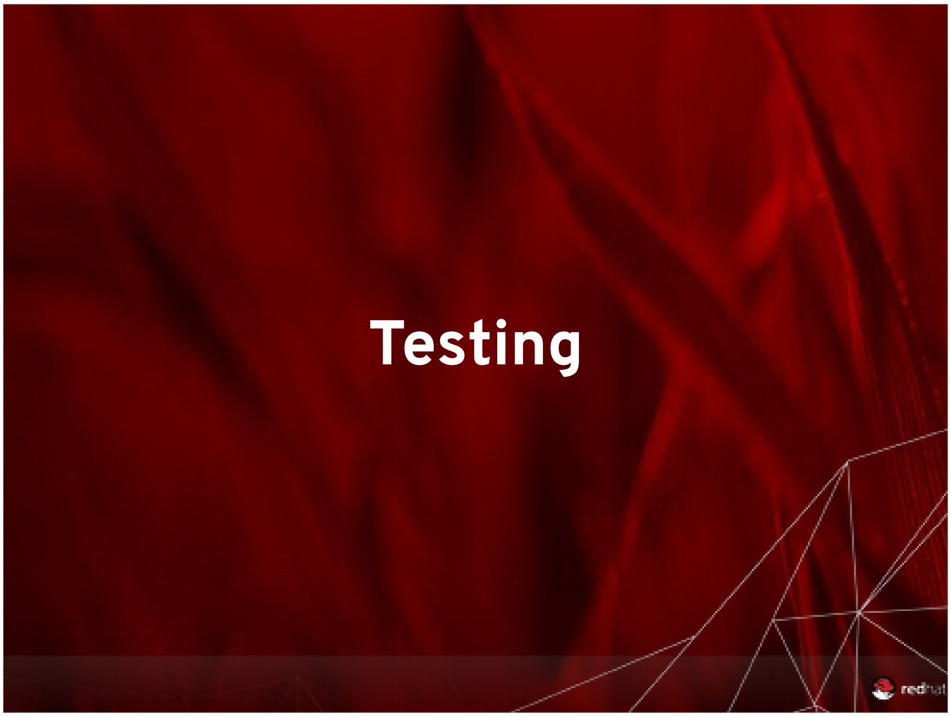
Certificate handling

Various less known bugs in NSS and GnuTLS related to certificate handling. NSS would consider a bad signature to be ok while GnuTLS would consider a certificate to have capabilities it did not have.

CVE-2014-6321 in schannel

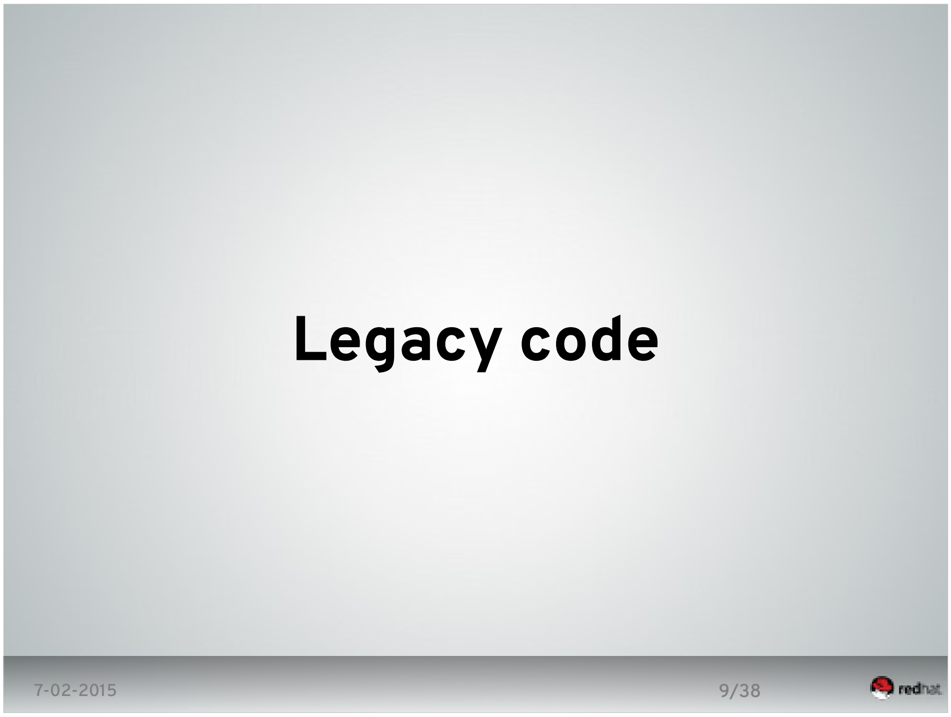
Aka winshock

Microsoft schannel also patched vulnerability in which remote attacker can execute code on server under SYSTEM privileges.



Testing

Why does this happen when those libraries are tested? “I mean, you *are* testing them, don't you?” Yes, *but*...

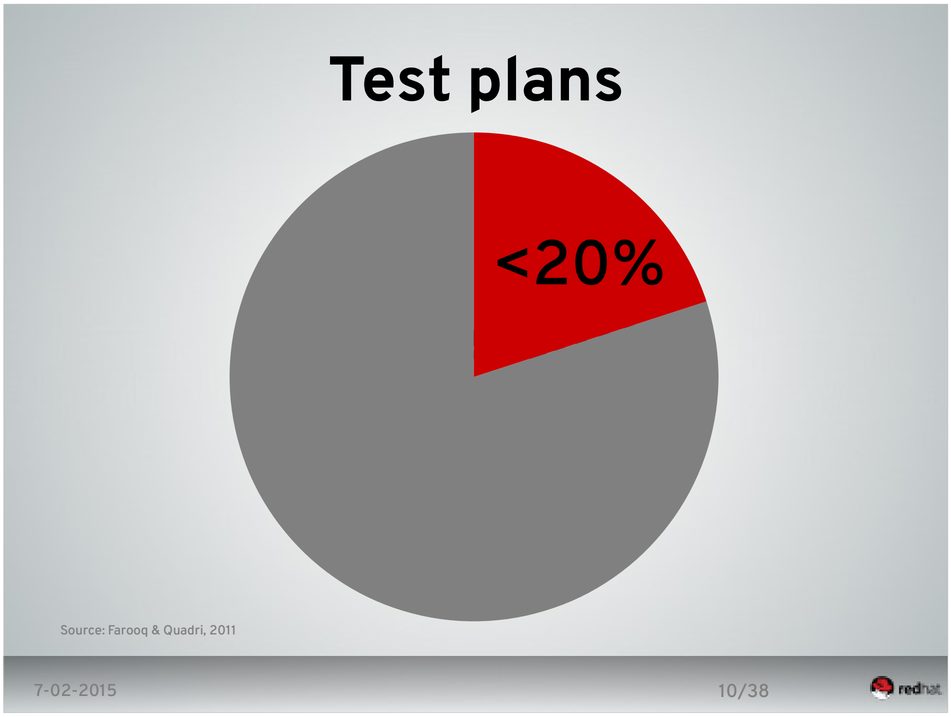


Legacy code

7-02-2015 9/38

Let's be frank, there's a lot of legacy code out there. By legacy code I simply mean code without unit tests. Not old code, not spaghetti code. But code without detailed code coverage. Why? Unknown if it works at all (who have seen code that was committed to repo that would never work?). Hard to refactor. Unknown expected behavior

Working Effectively with Legacy Code, Michael Feathers



Test plans

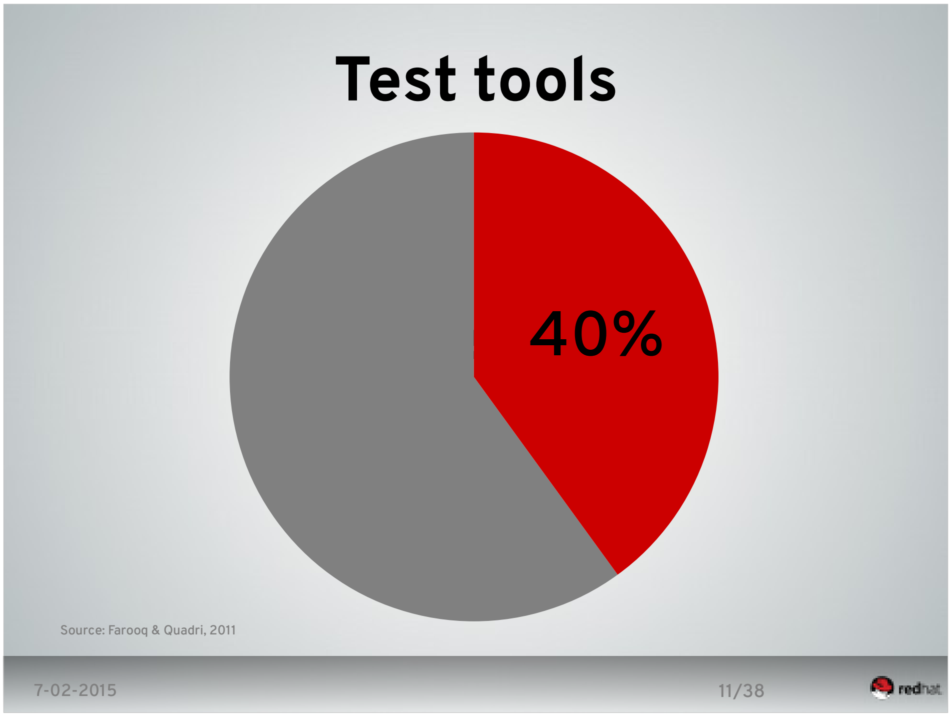
Source: Farooq & Quadri, 2011

7-02-2015

10/38

A study in 2011 took a look on Open Source projects and found out that fewer than 20% use test plans.

Farooq & Quadri, 2011.



Test tools

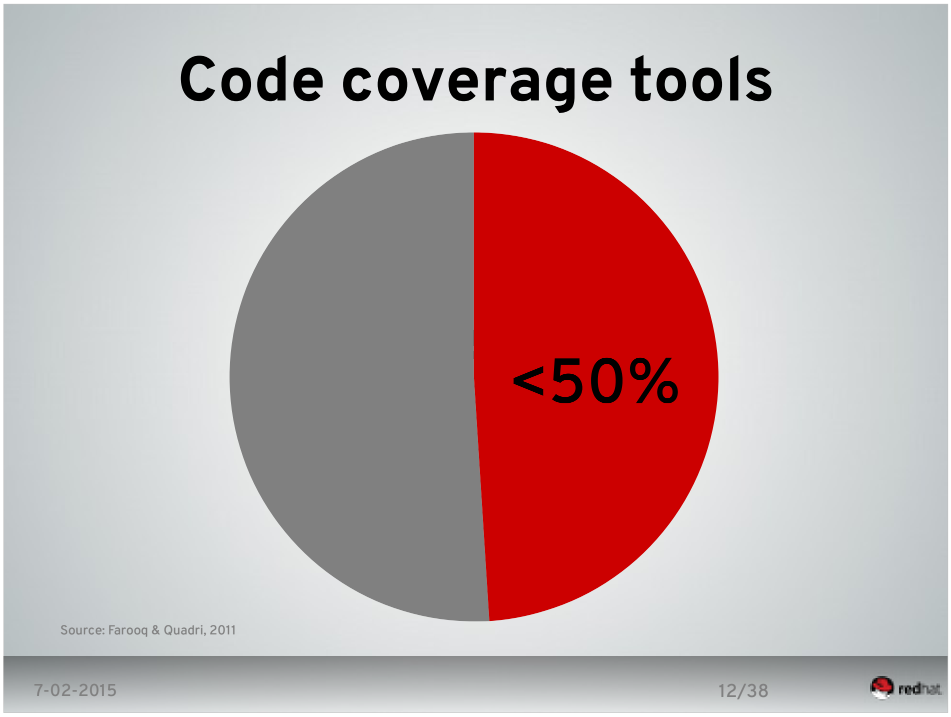
Source: Farooq & Quadri, 2011

7-02-2015

11/38

Only about 40% use test tools.

Farooq & Quadri, 2011.



Code coverage tools

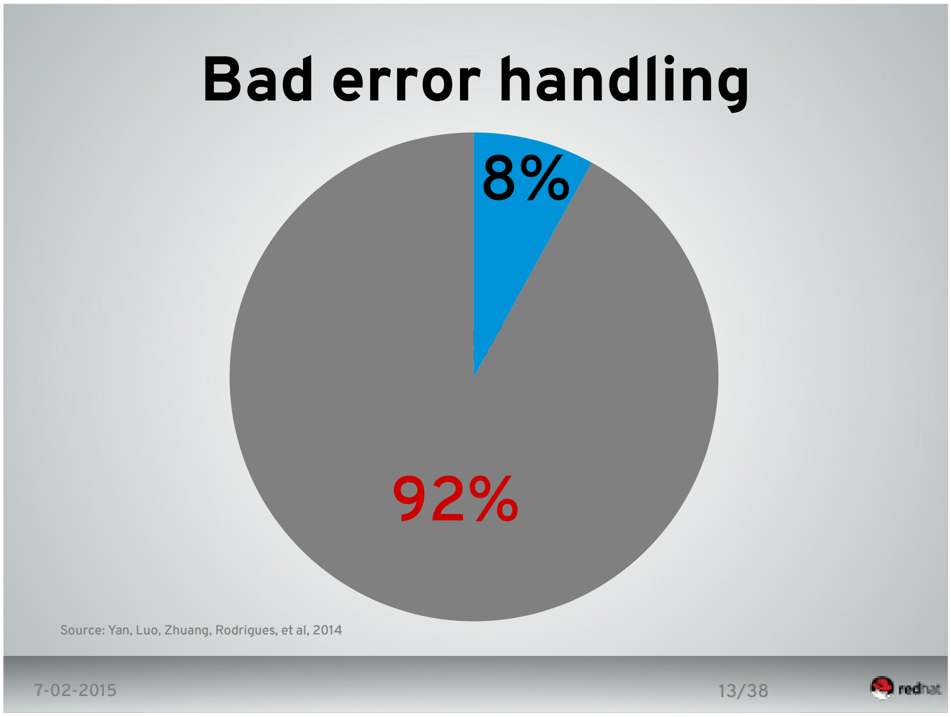
Source: Farooq & Quadri, 2011

7-02-2015

12/38

Less than 50% use code coverage.

Farooq & Quadri, 2011.



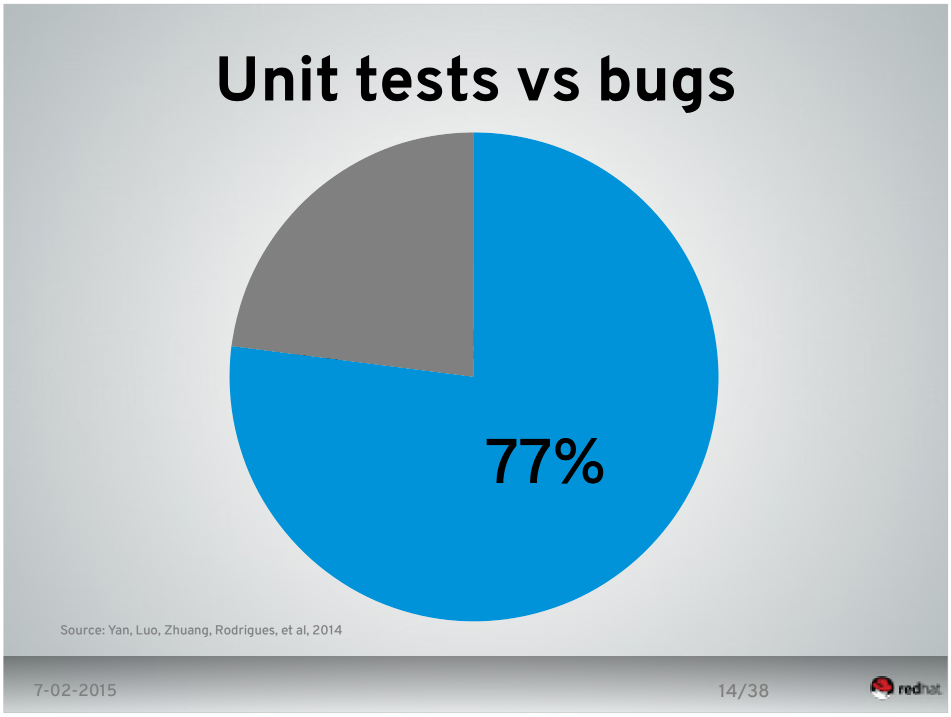
Bad error handling

Source: Yan, Luo, Zhuang, Rodrigues, et al, 2014

7-02-2015 13/38

Last year's study of distributed file systems found out that just 8% of severe bugs (ones that lead to data loss) were caused by logic errors. 92% were caused by wrong or missing error handling.

Yan, Luo, Zhuang, Rodrigues, et al, 2014. .



Unit tests vs bugs

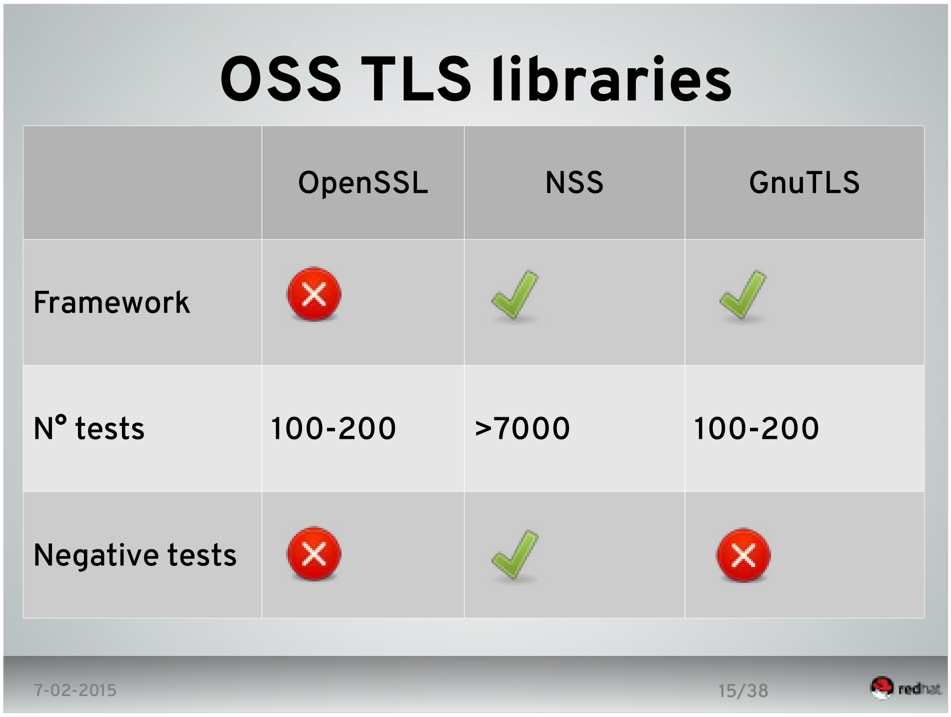
Source: Yan, Luo, Zhuang, Rodrigues, et al, 2014

7-02-2015

14/38

At the same time over 77% were reproducible with unit tests. Most of those that weren't were caused by deficiencies in tooling used for unit tests.

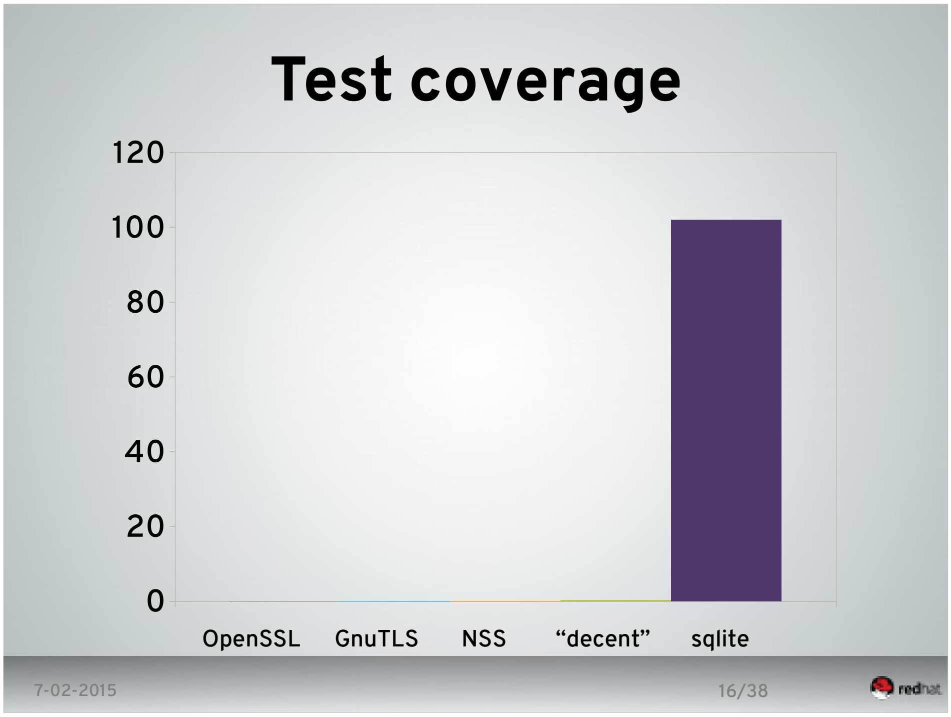
Yan, Luo, Zhuang, Rodrigues, et al, 2014. .



OSS TLS libraries

	OpenSSL	NSS	GnuTLS
Framework	✗	✓	✓
N° tests	100-200	>7000	100-200
Negative tests	✗	✓	✗

How do OSS TLS libraries stake to that? Not well. OpenSSL doesn't follow good testing practice and has minimal test coverage. GnuTLS only adds use of a test framework to that. Only NSS looks good here, but let's compare them to some other pieces of code.

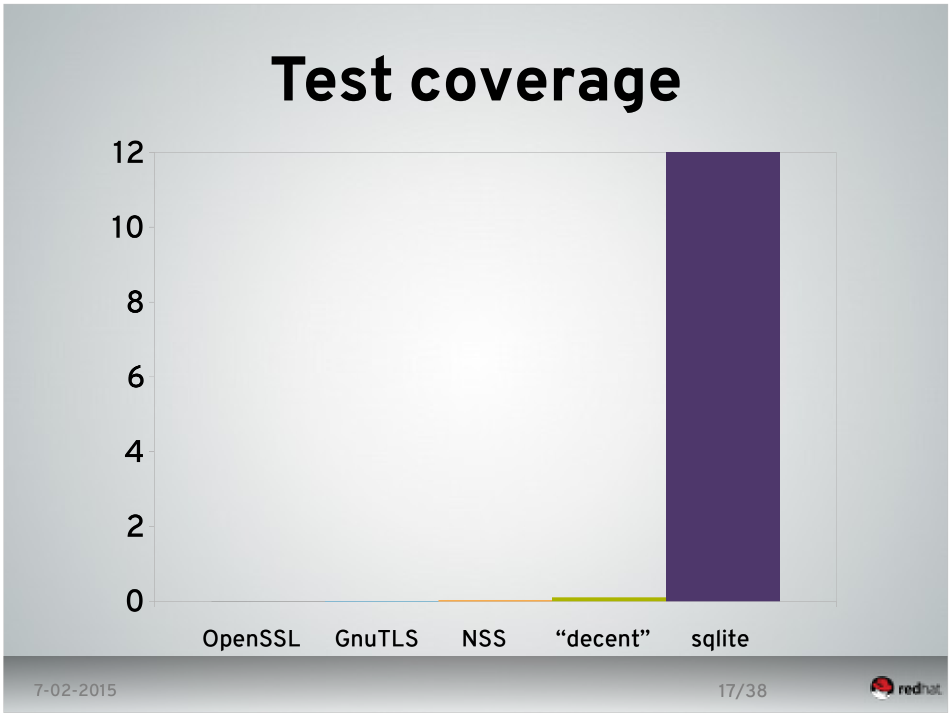


Test coverage

7-02-2015 16/38

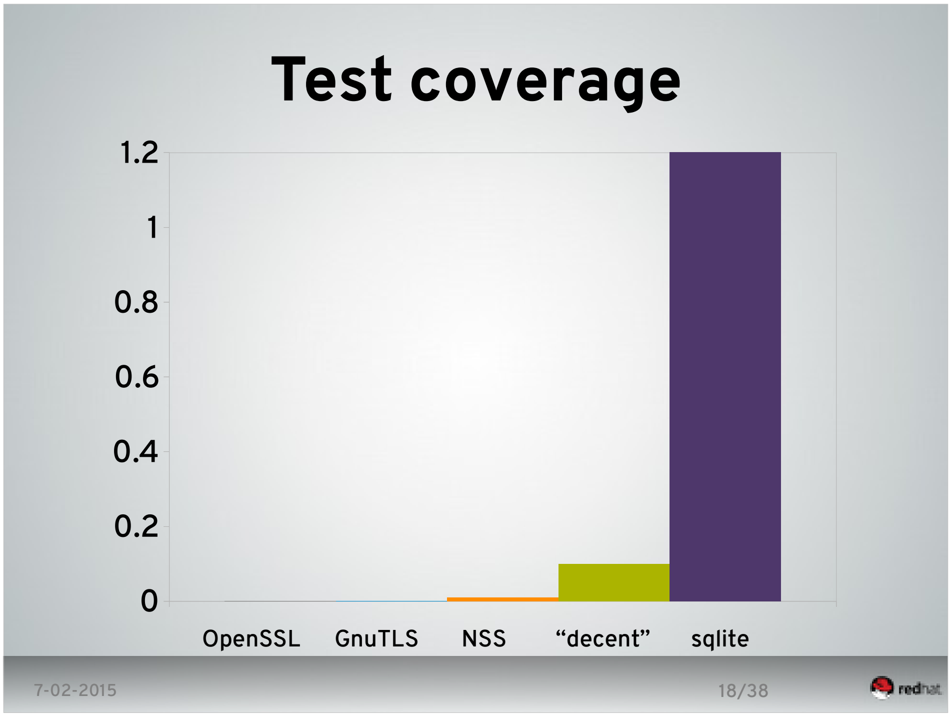
So let's compare the test coverage to other systems. The scale on the left is the ratio of number of test cases to lines of code. Sqlite has 97k LOC and 10M test cases so has ratio of 103.

Hmm, doesn't look like our libraries register on the scale... let's zoom a bit



Test coverage

Still hard to see... let's zoom by a factor of ten again.



Test coverage

7-02-2015 18/38

Ahh, there we go!

What I'd call decent test coverage is one test case for every 3 to 8 lines of code, so one eighth test case per line of code.

NSS has about 670kLOC and 7k test cases so about 0.01 test cases per line of code.

OpenSSL and GnuTLS both have about 500kLOC so they don't register on the scale still, we would have to zoom 3 times more for them to register.

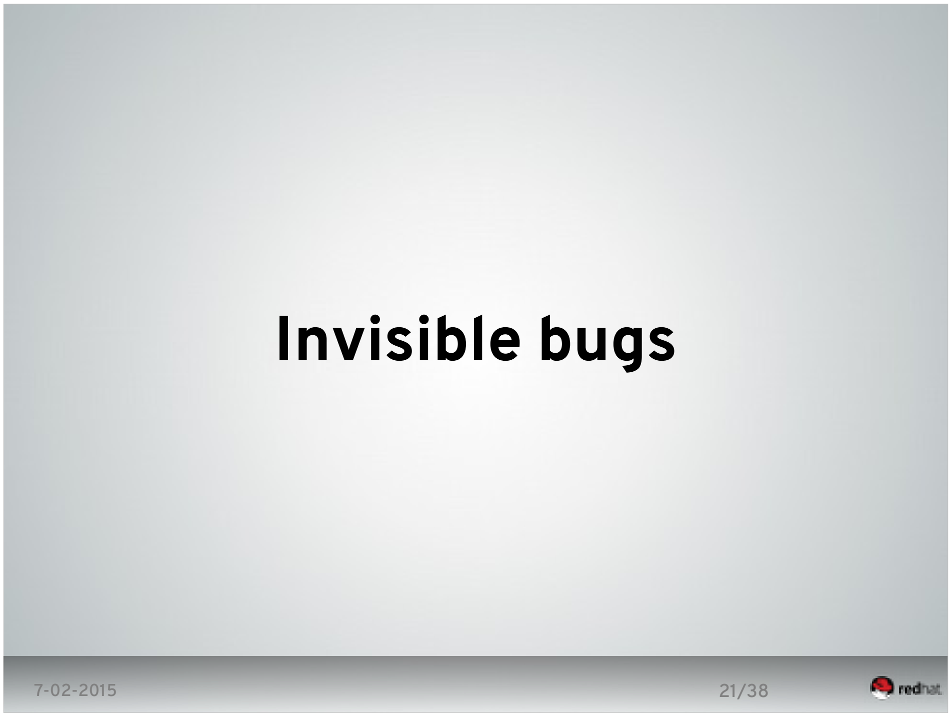
Why is that?

So why is that security libraries don't perform more testing?

X.509 and ASN.1

Both are arcane, support specifying the same information in multiple ways. The most common vulnerability avenue. Thankfully used only for certificates and already extensively tested by fuzzers and test suites.

https://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt



Invisible bugs

Timing server replies when sending invalid requests can recover secret information – plaintext, private keys. Bugs in low level cryptography (especially asymmetric) are far from obvious and require good understanding of maths involved to detect or test.

Libraries and bad data

Other issue with testing is that libraries don't like sending invalid data. That makes it hard to generate data for negative tests and requires creation of parallel implementation of TLS just for testing.

Fuzzy testing

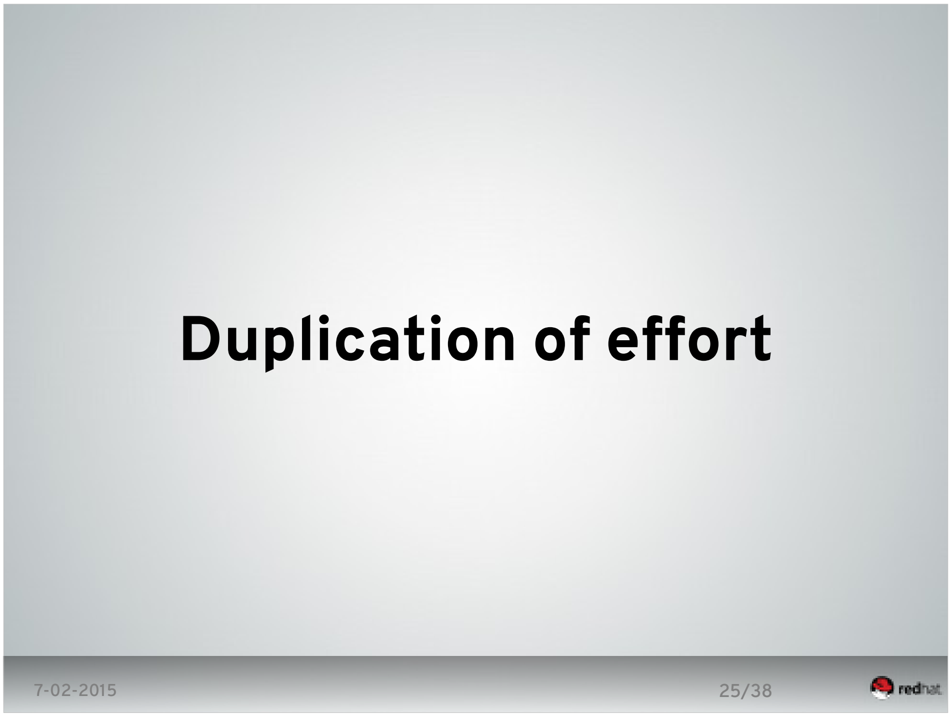
7-02-2015 23/38

Using pure fuzzy testing is problematic as TLS has very large problem space – multiple valid inputs that depend on previous data. After handshake also encrypted and checksummed – requires full TLS implementation in the fuzzer.

So basically a combination of “hard problem” and “understaffed projects”.

Fixing the problem

How can we fix this systemic problem?



Duplication of effort

7-02-2015 25/38

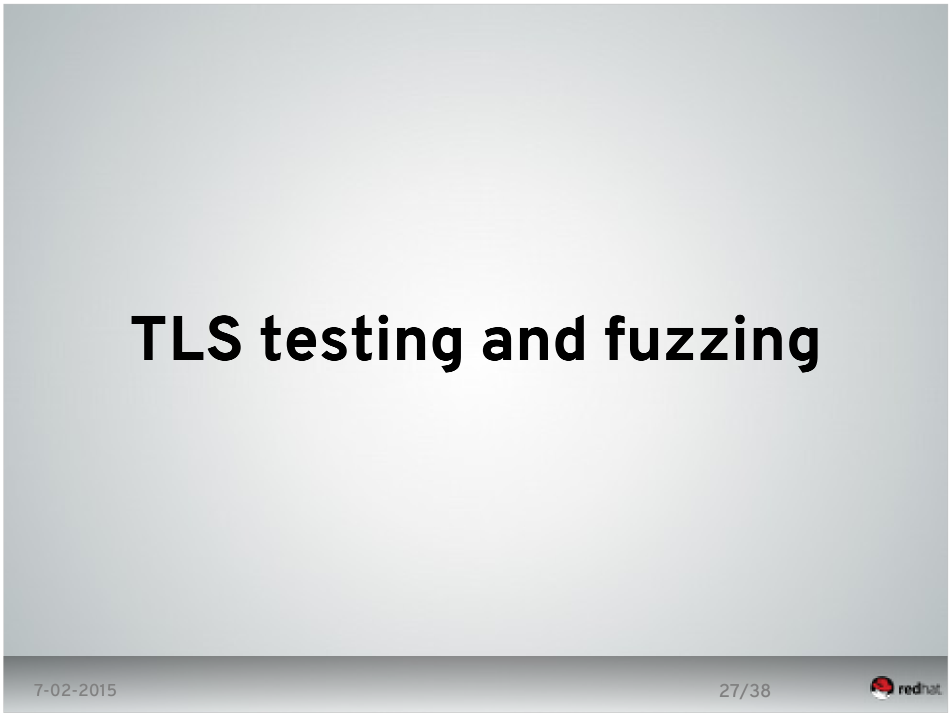
We want to avoid duplication of effort.

The one important point is that TLS is a network protocol in which the client has to advertise its capabilities while the server can select only the features client advertised. This allows for automatic detection of capabilities of the peer, as such we can use a single test tool with multiple implementations.

Existing fuzzers

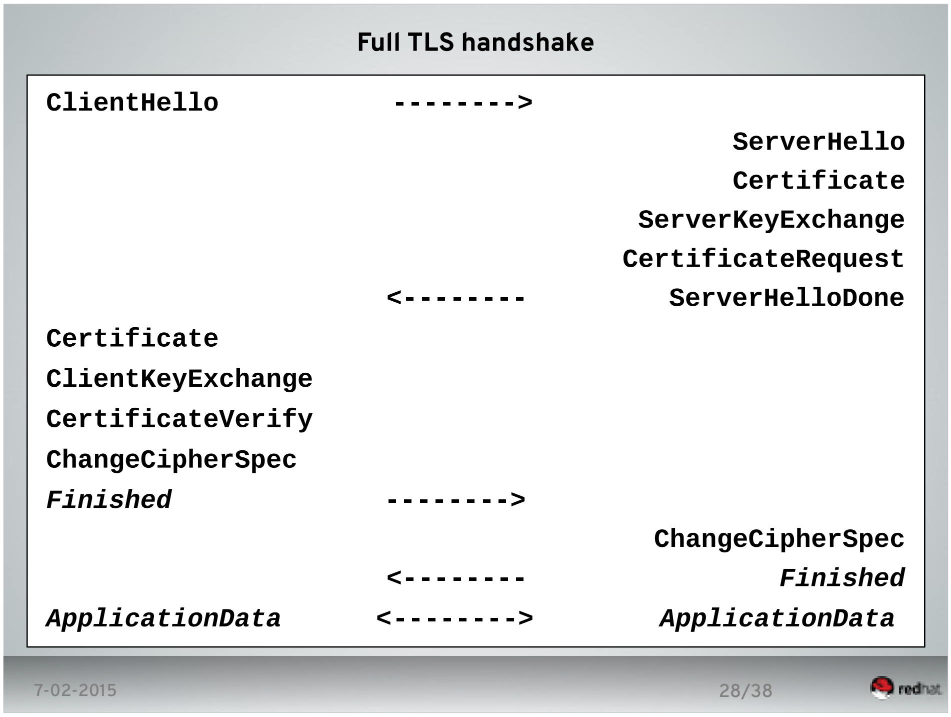
7-02-2015 26/38

So I've looked at existing network protocol fuzzers. Most promising were Sulley and scapy. Unfortunately both suffer from the same problem: vary hard to keep and update state needed to work with encryption. At the same time after testing one message, you have to start a completely new connection. Sulley additionally has no provisions for capturing and testing server output. That makes it completely useless for testing protocol like TLS.



TLS testing and fuzzing

While I'll be talking about fuzzing now, I mostly mean grey or whitebox testing with actual fuzzing being only a cherry on top. The fuzzer will understand the protocol implemented. In general TLS requires the peer to abort on receiving a malformed message. But much more interesting is testing messages which are just slightly wrong, or unexpected. To be able to test them we must have support for doing full TLS handshake and implement significant portion of ciphers.

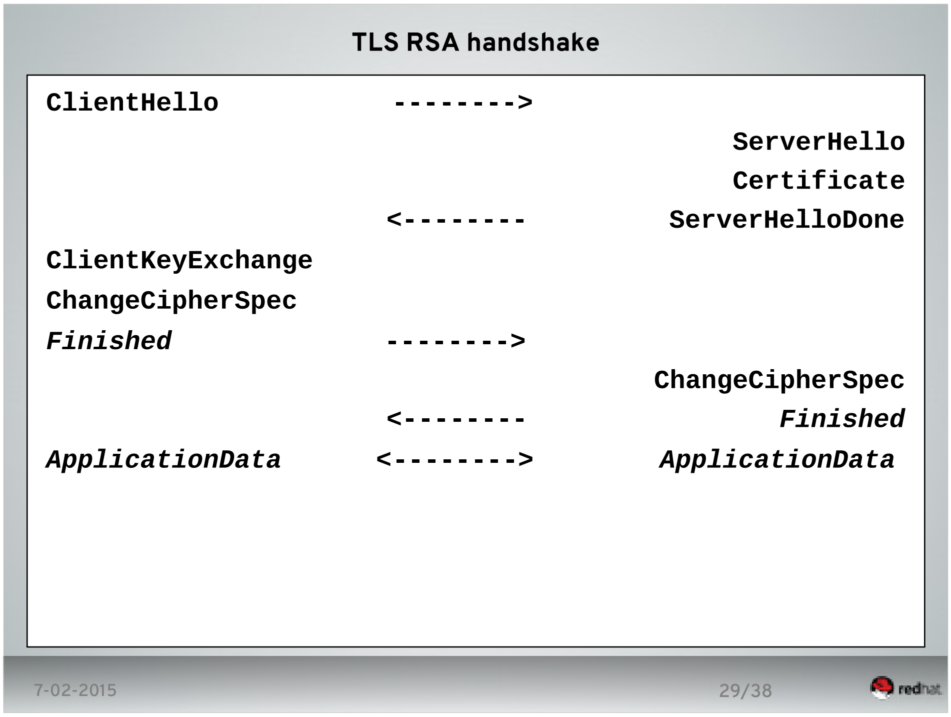


Full TLS handshake

```
ClientHello           -------->
                                            ServerHello
                                            Certificate
                                      ServerKeyExchange
                                     CertificateRequest
                     <--------          ServerHelloDone
Certificate
ClientKeyExchange
CertificateVerify
ChangeCipherSpec
Finished             -------->
                                       ChangeCipherSpec
                     <--------                 Finished
ApplicationData     <-------->          ApplicationData
```

7-02-2015 28/38

A full TLS handshake looks something like this. What is important is that the sent ClientHello at the very beginning will influence our ability to properly decrypt and verify the Finished data. So until we don't receive it we won't know if server *really* accepted our connection.

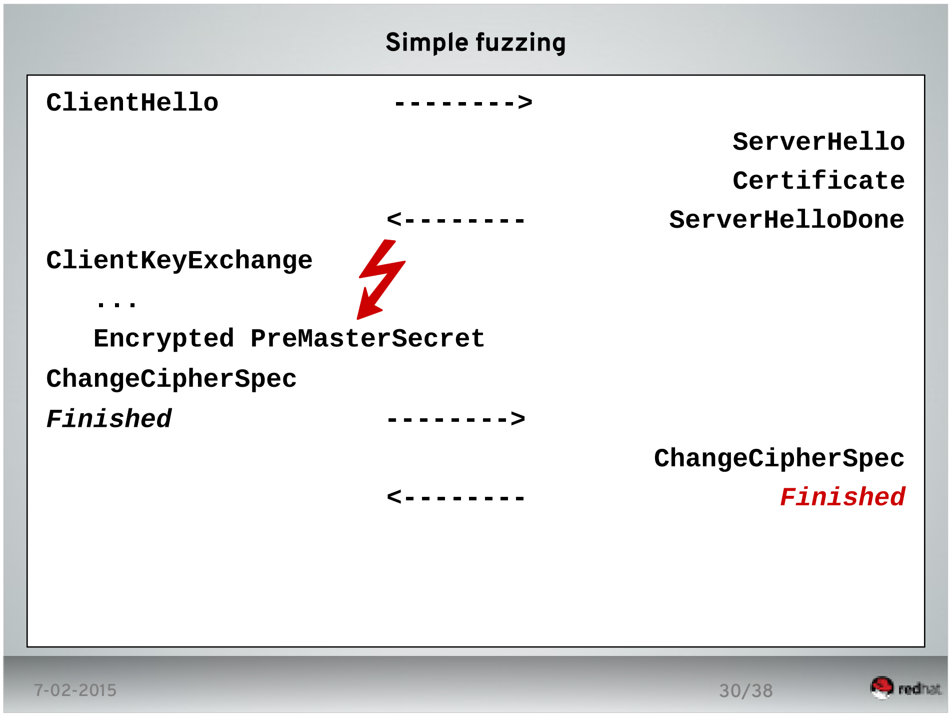


TLS RSA handshake

```
ClientHello            -------->
                                             ServerHello
                                             Certificate
                       <--------         ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished               -------->
                                        ChangeCipherSpec
                       <--------                Finished
ApplicationData       <-------->         ApplicationData
```

7-02-2015 29/38

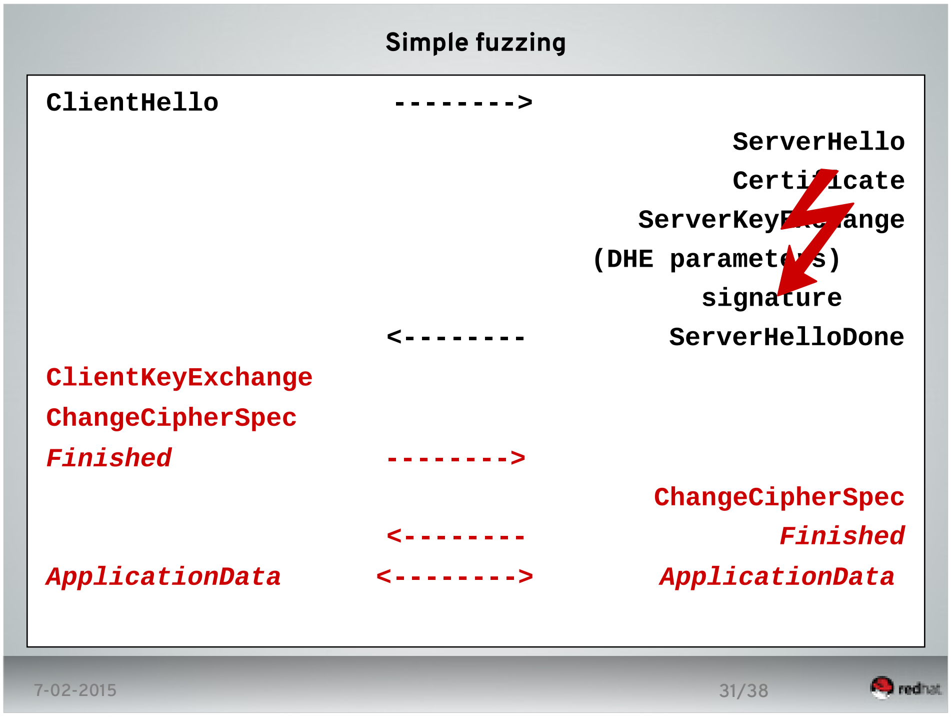
The connection is not so complex every time, for example handshake with RSA key exchange looks like this and exchanges just 9 messages.



Simple fuzzing

```
ClientHello            -------->
                                        ServerHello
                                        Certificate
                       <--------     ServerHelloDone
ClientKeyExchange
   ...
   Encrypted PreMasterSecret
ChangeCipherSpec
Finished               -------->
                                   ChangeCipherSpec
                       <--------           Finished
```

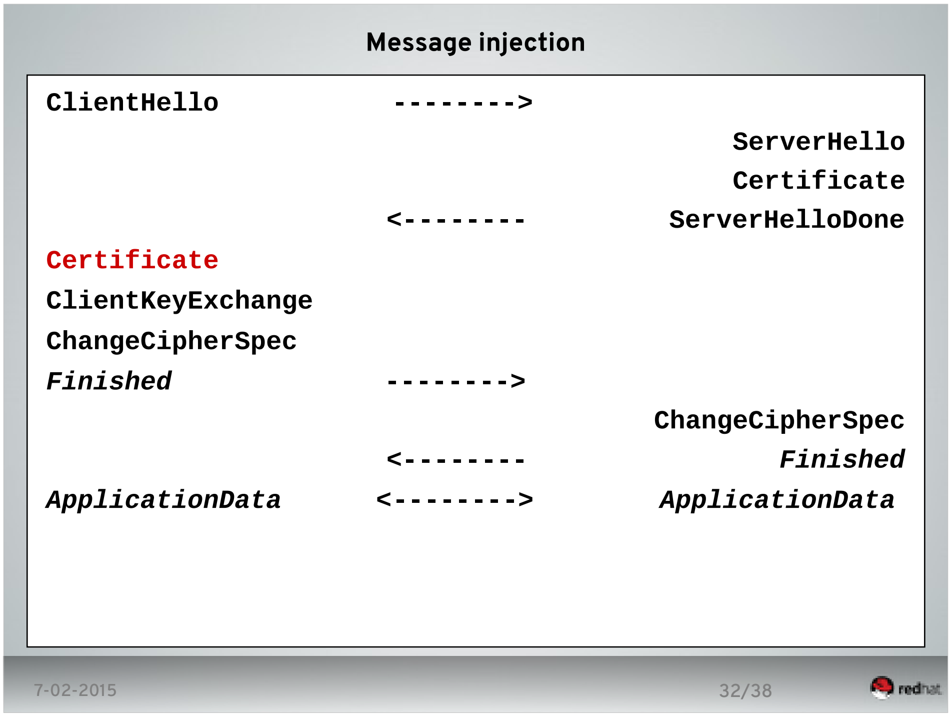
Problem is that if I'm testing the encryption of premaster secret in ClientKeyExchange, I won't know if the server detected the change until I receive the Finished message! Server also won't accept a ClientKeyExchange before it sends a ServerHello message.



Simple fuzzing

```
ClientHello                 -------->
                                                ServerHello
                                                Certificate
                                          ServerKeyExchange
                                       (DHE parameters)
                                             signature
                            <--------        ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished                    -------->
                                           ChangeCipherSpec
                            <--------              Finished
ApplicationData            <-------->        ApplicationData
```

Other kind we can do is insert failures in signatures. For example in DHE key exchange. This is a test for the gotofail bug in Apple Secure Transport.

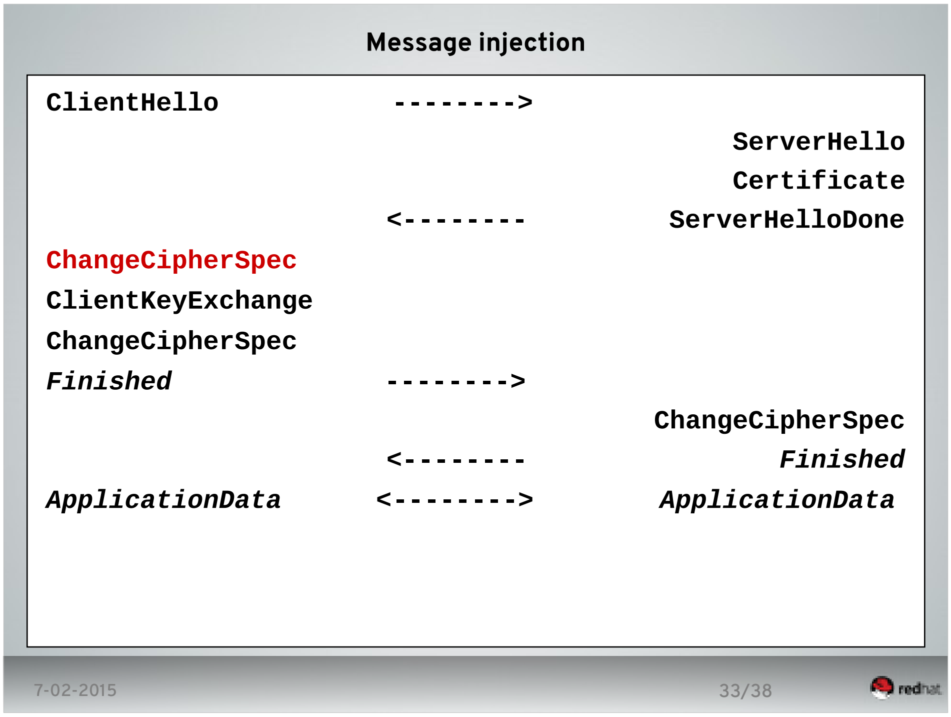


Message injection

```
ClientHello           -------->
                                         ServerHello
                                         Certificate
                      <--------      ServerHelloDone
Certificate
ClientKeyExchange
ChangeCipherSpec
Finished              -------->
                                    ChangeCipherSpec
                      <--------             Finished
ApplicationData      <-------->      ApplicationData
```

We can also inject messages which are not expected by the server. For example, here the server didn't request a certificate (didn't send CertificateRequest) but the client is sending one anyway.

The correct course of action would be to abort the connection. Unfortunately Microsoft schannel not only didn't abort the connection but actually tried to parse the contents of the message. This is what made the vulnerability from last year affect all servers. If they didn't accept the message unconditionally it would be limited to servers using certificate based client authentication.

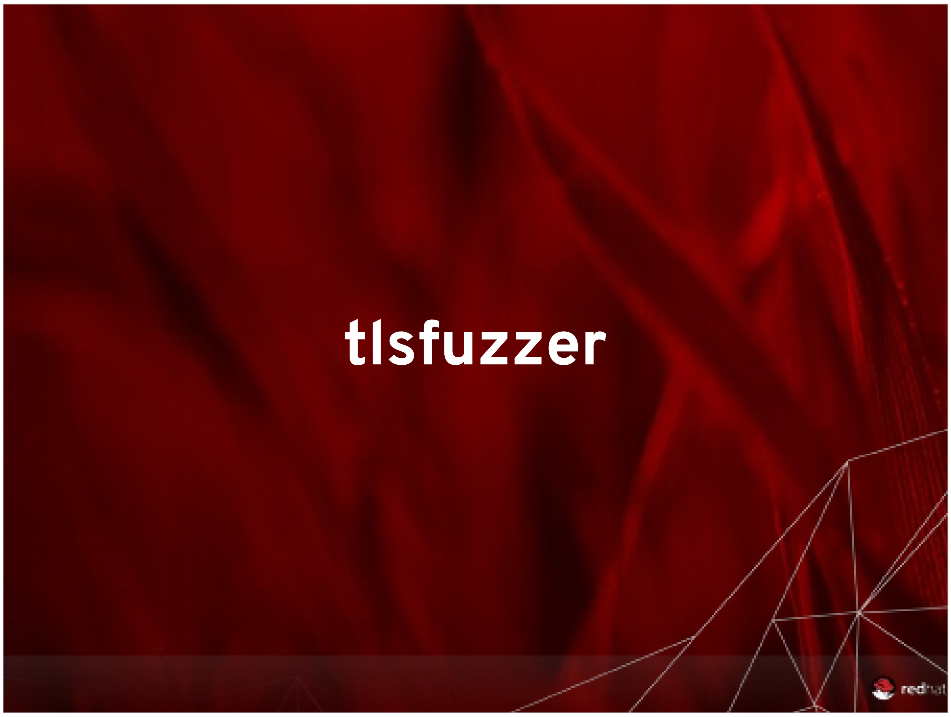


Message injection

```
ClientHello            -------->
                                           ServerHello
                                           Certificate
                       <--------       ServerHelloDone
ChangeCipherSpec
ClientKeyExchange
ChangeCipherSpec
Finished               -------->
                                      ChangeCipherSpec
                       <--------              Finished
ApplicationData       <-------->       ApplicationData
```

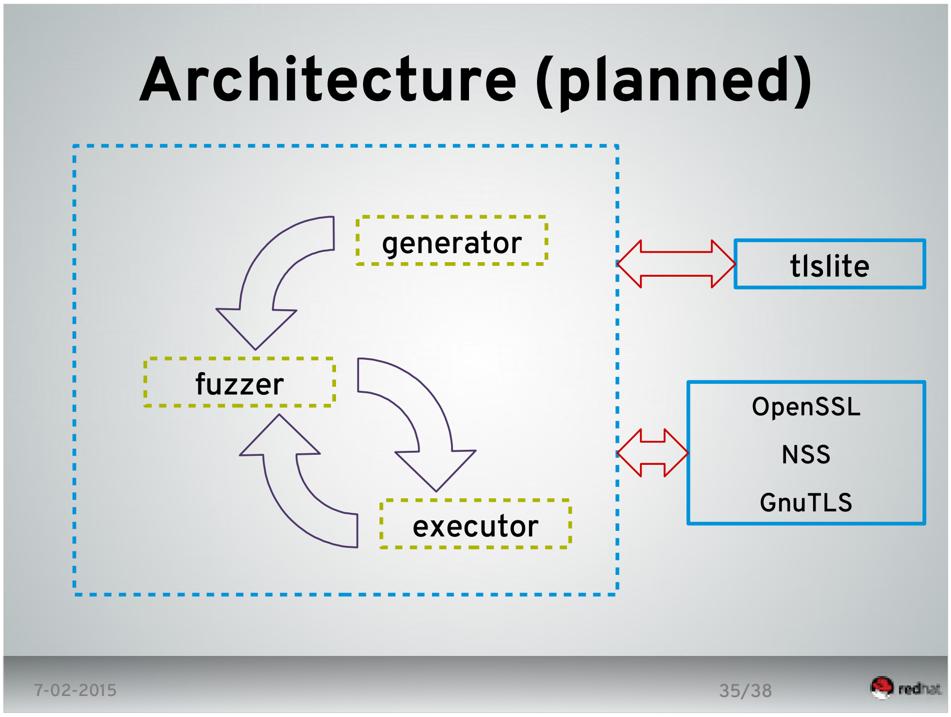
7-02-2015 33/38

On the other hand, if we inject a ChangeCipherSpec message we would detect the OpenSSL CCS bug.



tlsfuzzer

As such, I've started working on a tool that would enable testing, verification and optimally - fuzzing the TLS implementations.



Architecture (planned)

generator

fuzzer

executor

tlslite

OpenSSL

NSS

GnuTLS

7-02-2015 35/38

Major redesign – no available code atm.

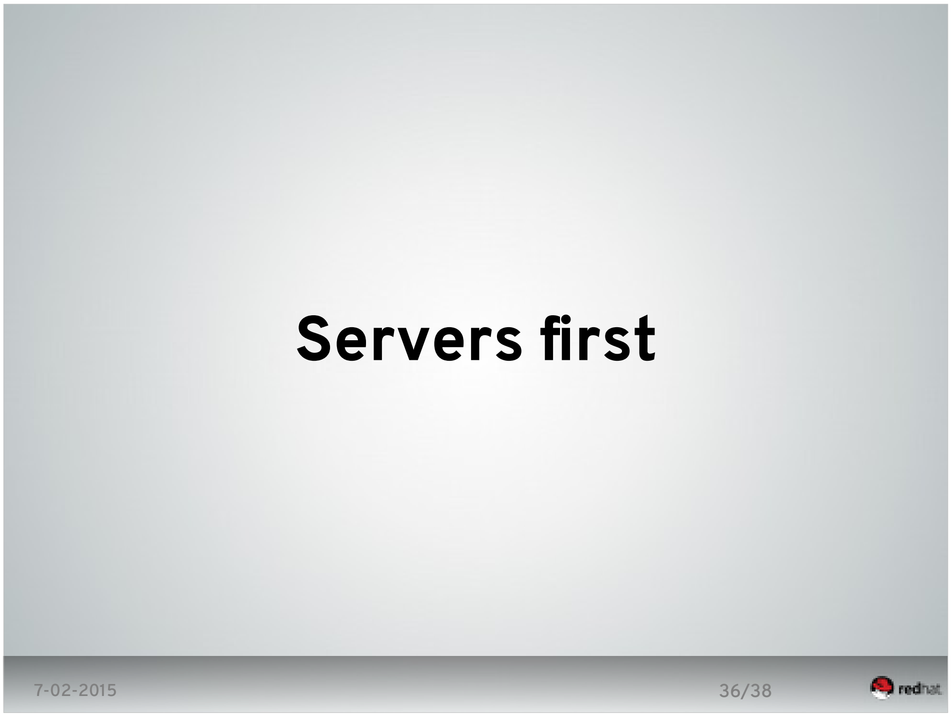
Generator → conversation

Conversation → fuzzer

FuzzedConversation → executer

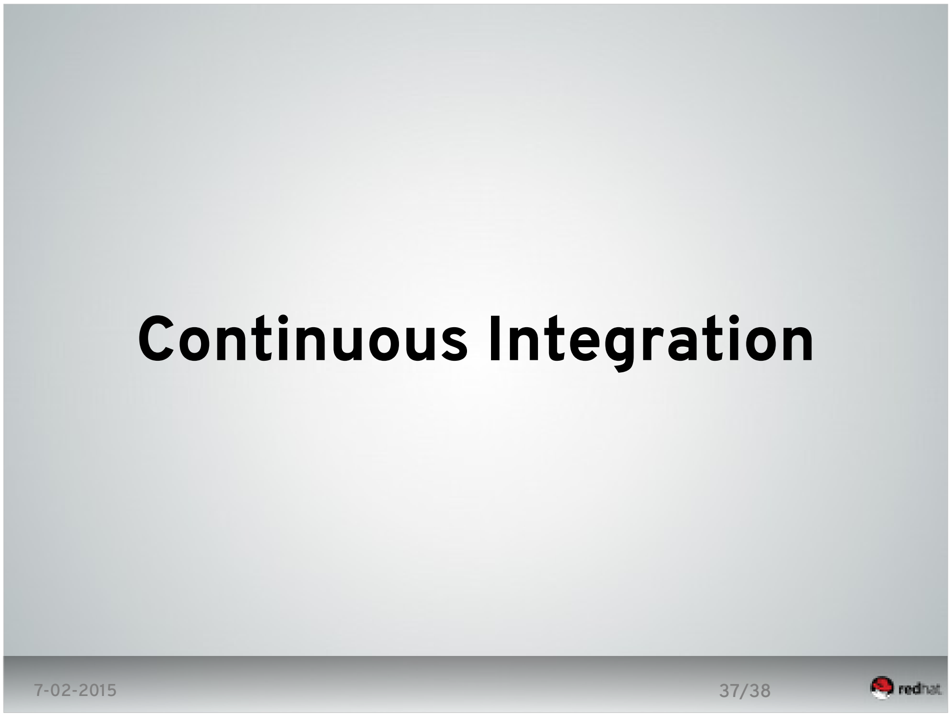
Executer → result

We start with generator, which takes known possible conversations (user created) and creates a test flow (messages to send and expected server messages). This conversation is then sent to fuzzer which sends it first to executor to verify if it is accepted by server. Then it proceeds to change the conversation by mutating the messages inserting new or dropping existing, executing it again. The key point is that fuzzer understands the messages it is changing, so for example, addition of extension with an unassigned ID should not cause change of behavior of server. On the other hand, server sending extension that was not advertised by server should cause connection abort.



Servers first

For now I'll be focusing on testing servers (bigger attack surface, easier automation) but the goal is to test both sides.



Continuous Integration

7-02-2015 37/38

The end goal is to have a system which can be easily used for Continuous Integration of arbitrary TLS libraries.

Because all the test cases have attached expected result it's possible not only to use it as a test for conformance with standards but also to continuously test the implementation for regressions.

At the same time, with high fuzz ratio, it should be possible to find more obscure bugs.

Questions?

Feedback: http://devconf.cz/f/108

Contact: hkario@redhat.com

Project: https://github.com/tomato42/tlsfuzzer