Rebuilding the airplane in Flight

Eric Rescorla

ekr@rtfm.com

Real World Cryptography 2013

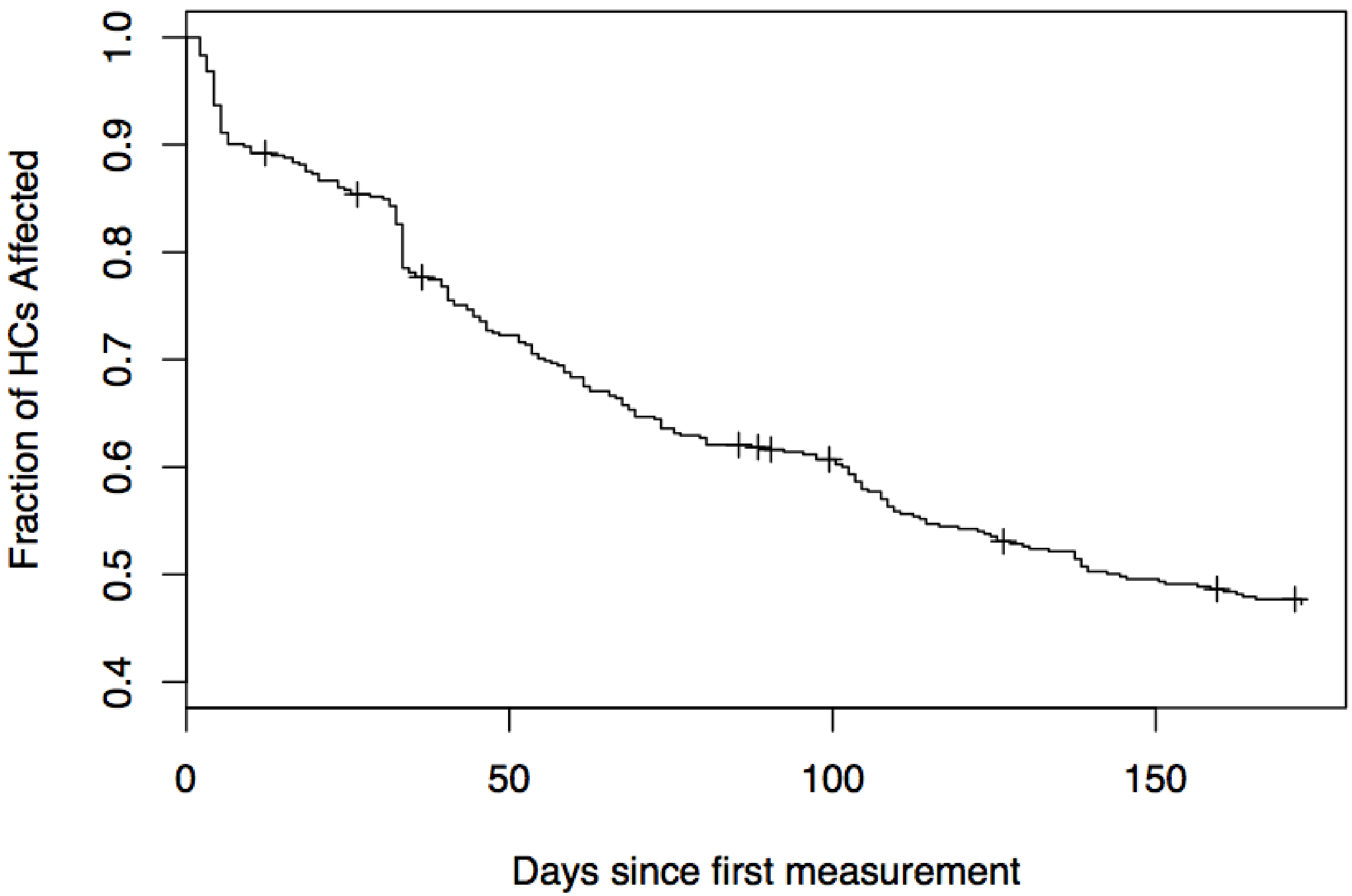
So you have a secure protocol....

- Now what happens when you want to change something?
  - A new algorithm comes out
  - Someone finds a problem
- This is where life gets hard
  - I already have all this old *stuff* in the field
  - It's not really going anywhere
  - How do I work around this?

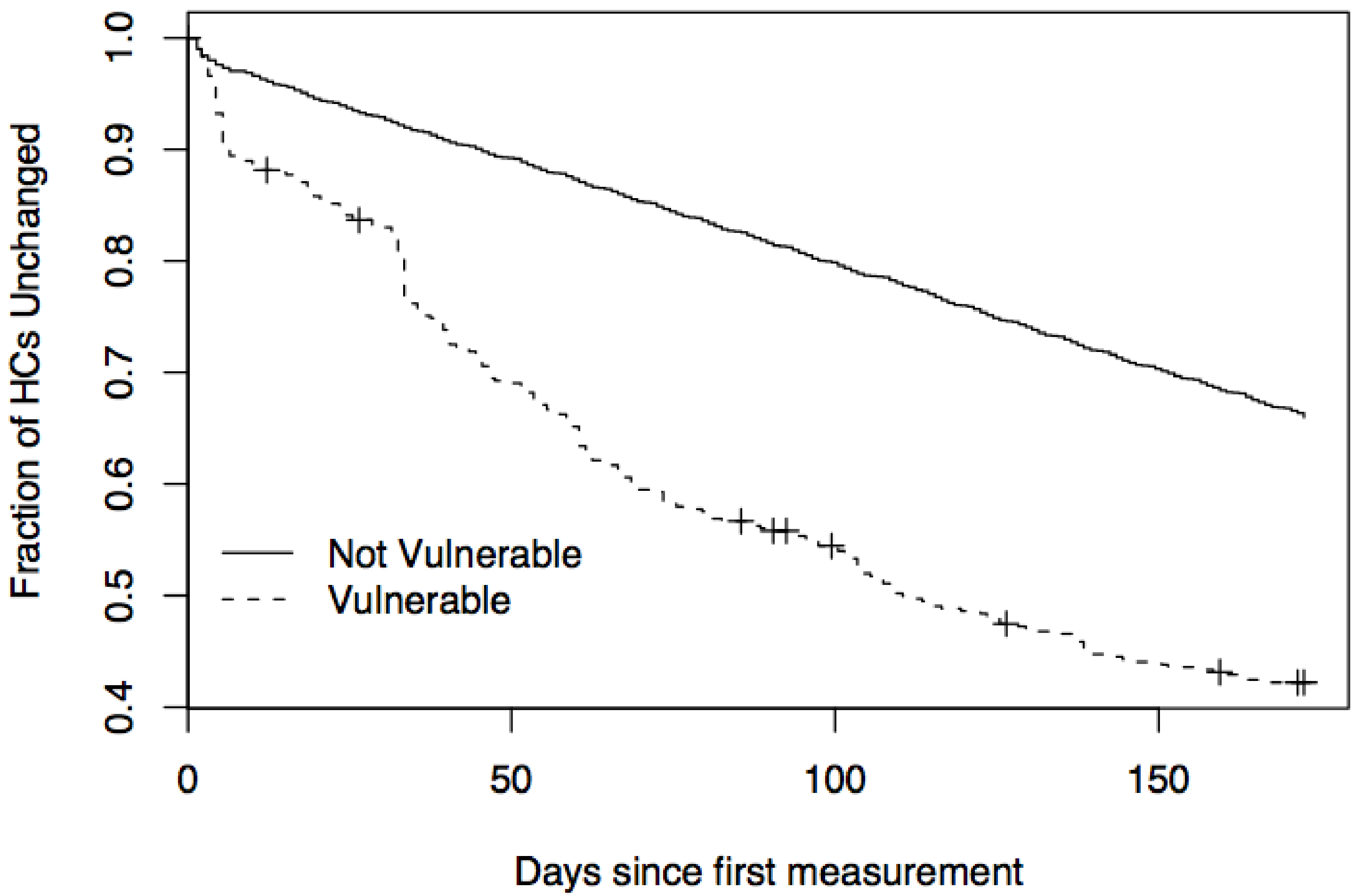
The Best (Worst?)-Case Scenario: Debian PRNG

- In 2006, Debian version of OpenSSL patched to fix Valgrind warnings
  - Accidentally wipes out nearly all entropy in PRNG (≈ 16 bits left)
- Noticed in 2008 by Luciano Bello
- About 1% of servers had predictable private keys
  - Easily remotely detectable
  - Completely breaks RSA cipher suites against passive attack
  - Breaks DHE cipher suites against active attack or fancy passive attack [YRS$^+$09]
- Imperative that servers fix
  - Fix was compatible and easy (get a new certificate)

How fast did people fix affected servers? [YRS$^+$09]



Certificate churn versus natural replacement rate



Upgrading Options

- Don't upgrade

Upgrading Options

- ~~Don't upgrade~~
- Forklift upgrade

Upgrading Options

- ~~Don't upgrade~~
- ~~Forklift upgrade~~
- Parallel universes

Upgrading Options

- ~~Don't upgrade~~
- ~~Forklift upgrade~~
- ~~Parallel universes~~
- Version negotiation

Negotiating Protocol Versions

- All agents start out supporting version n
- We want to introduce $n+1$ while retaining backward compatibility
  - Need some kind of version negotiation mechanism
- Problems
  - How to negotiate securely?
  - When can you discard version n? (probably never)

Client Version	Server Version	
	n	$n/n+1$
n	n	n
$n/n+1$	n	$n+1$

Table 1: Desired Negotiation Outcome

Example: SSL/TLS Negotiation Mechanisms

- Version number
- Cipher suites
- “certificate types” field in CertificateRequest
- Extensions (published in 2003) [BWNH+03]

Something that went fairly well: transition to AES

- SSL/TLS, S/MIME, IPsec etc. were designed before AES
  - Typical ciphers supported: DES, 3DES, RC4, RC2
- AES protocol support added rapidly
  - AES [NIS01] published in 2001
  - AES for TLS published in 2002 [Cho02]
  - AES for CMS published in 2003 [Sch03]
  - AES for IPsec published in 2003 [FGK03]
- Implementations kept pace with standardization
  - OpenSSL added AES for TLS and S/MIME in 2002

Why did AES deployment go as well as it did?

- Protocols were designed for encryption algorithm agility
  - The one thing everyone knew they needed to be able to replace
  - AES has nearly exactly the same "API" as DES
    - Except for block and key size
  - In many cases implementations automatically negotiated AES support
- AES filled a real gap
  - 3DES obviously too slow
  - Concerns about security of RC4
- Very strong push by USG
  - Many applications where AES is required


WHAT IF I TOLD YOU
AMAZON AND GOOGLE USE RC4
memegenerator.net

SSL/TLS AES deployment far from universal

- Many popular sites still prefer RC4 to AES
  - Examples: Google, Wells Fargo, Amazon
  - As of 2011 AES still only supported in $< 65\%$ of servers[Ris11]
- RC4 is 1.5-2x faster than AES-CBC
- No *practical* known security problems with RC4
  - Though lots of concerns about apparent non-randomness
  - Especially with the initial bytes

So AES GCM should be easy, right?

- Not just a drop-in with the same "API"
  - SSL/TLS was designed with encryption and MAC as separate primitives
  - This is fixed in TLS 1.2 [DR08]
- TLS 1.2 deployment status is still minimal
  - OpenSSL 1.0.1 (2012)
  - iOS 5.0
  - In process for NSS (Chrome, Firefox)
- But deployment will happen automatically (eventually)

Something less clean: transition from MD5 to SHA-1

- MD5 collisions are a threat to certificates [SSA+09]
  - Though countermeasures (sequence number randomization) are available
- Essentially all SSL/TLS stacks already supported SHA-1 certificates
- CAs just had to stop using MD5
  - Mostly transparent to servers

A certificate with a strong hash doesn't help the certificate holder

- Threat is an attacker getting a certificate in my name
  - The existence of a weak certificate for me doesn't help them
  - Modifying that certificate requires a second preimage attack
    - Existing attacks involve (easier) collision-finding
- I want *relying parties* to stop accepting weak hashes
  - But my actions don't really affect that
  - Chrome and Firefox finally turned off MD5 in 2012!
- Classic collective action problem

(from Mozilla Memes)

What about SHA-2?

- Many browsers *don't* support SHA-x
- Need to negotiate it in the handshake SSL/TLS handshake
- This turns out to be a huge hassle
  - Design finished in 2008
  - Only starting to roll out now

Negotiating Certificate Digests

- This should be easy
  - Certificates have a hash algorithm field
  - TLS has negotiable cipher suites
    - They have a digest in them
    - E.g., `TLS_RSA_WITH_RC4_128_SHA`
- TLS cipher suites don't control the certificate digest
  - No way for clients to indicate that they support SHA-256
  - So only safe to send MD5 and SHA-1 certs
- Solution: signature_algorithms extension
  - Indicates which signature and digest algorithms each side supports

Replacing the TLS PRF

- TLS before 1.2 had a hardwired internal PRF
  - Used for key generation and handshake integrity check
  - Based on MD5 and SHA-1 XORed together
- This is probably safe
  - But still pretty scary
- TLS 1.2 has a negotiable PRF
  - Tied to the cipher suite
  - Default is SHA-256
- Note: security of the handshake is now no stronger than HMAC-SHA256

Deployment model for SHA-2 with SSL/TLS

- Authenticating parties need *two* certificates
  - One for SHA-1
  - One for SHA-x
  - Until all relying parties support SHA-x
- Relying parties need to support SHA-1 *and* SHA-x
  - Until nearly all authenticating parties have SHA-x certificates
- Confusion over SHA-2 vs. SHA-3 doesn't help here
- This is more or less the same story as ECDSA versus RSA

Countermeasures versus fixes: Predictable IV attacks [Moe]

- Scenario: Attacker can observe ciphertext and inject his own plaintext
  - He observes a block C_i and wants to verify his guess X for its value
- Attacker sees a record with trailing block B
  - This means that B is the IV for the next block
- Attacker injects $C_{i-1} \oplus B \oplus X$ as plaintext
  - Victim encrypts $B \oplus C_{i-1} \oplus B \oplus X = C_{i-1} \oplus X$
  - If result is C_i then the guess was correct

Limitations of Predictable IV Attacks

- This has been known for years
- Need tight control of the channel
  - Didn't seem likely except for VPN settings
- Need to guess an entire block at a time
  - Not easy!
- This all doesn't sound very serious
  - TLS WG duly fixed TLS [DR06]
  - But practically nobody implemented it

Predictions are hard... especially about the future

- Rizzo/Duong "BEAST" paper changed people's perceptions of the risk
  - New technique for byte-by-byte guessing
  - New threat vector via Web technologies (WebSockets and Java)
- But this was fixed in TLS 1.1
  - So we'll just deploy TLS 1.1, right?
  - Well sort of...
- People are deploying TLS 1.1
  - But most servers still don't have it
  - And active downgrade attacks create a problem

(Mostly) Compatible Countermeasures

- Server side: move to RC4
  - Nearly all clients support RC4
  - Auditors actually *require* this in some cases
- Client side: 1/n+1 splitting
  - Victim does a write of n bytes
  - SSL stack encrypts it as two records
    - 1 bytes and $n - 1$ bytes
- This prevents the Rizzo/Duong attack
  - But breaks some servers
  - This time it's HTTP stacks not TLS stacks
  - We're still tracking down broken down implementations

Worse is better?

- New version deployment is almost never universal
  - IE 7 still has around 5% market share
- Options
  - Refuse to communicate with old versions
    - You broke the Internet!
  - Figure out some kind of countermeasure
- But countermeasures reduce the incentive to fix...

Summary

- Many of the extension points aren't
  - Code (or standards) which hasn't been tested doesn't work
  - ... any new primitive needs to look exactly like an existing primitive
- Changes in only one side are easier
  - But this generally precludes protocol/algorithm changes
  - And needed anyway to support older peers
- Hard to evaluate the security impact of cryptographic issues
  - Cryptographers tend to work in "abstract" environments
  - The real protocol is more complicated
  - COMSEC engineers don't understand the crypto well enough
- Incentives favor interoperability over security

References

[BWNH+03] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. RFC 3546, Internet Engineering Task Force, June 2003.

[Cho02] P. Chown. Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS). RFC 3268, Internet Engineering Task Force, June 2002.

[DR06] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, Internet Engineering Task Force, April 2006.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, Internet Engineering Task Force, August 2008.

[FGK03] S. Frankel, R. Glenn, and S. Kelly. The AES-CBC Cipher Algorithm and Its Use with IPsec. RFC 3602, Internet Engineering Task Force, September 2003.

[Moe] Bodo Moeller. Security of CBC Ciphersuites in SSL/TLS: Problems

and Countermeasures.
`http://www.openssl.org/~bodo/tls-cbc.txt`.

[NIS01] NIST. Specification for the Advanced Encryption Standard (AES), nov 2001. FIPS PUB 197.

[Ris11] Ivan Ristić. State of SSL, April 2011. `file:///Users/ekr/Downloads/Qualys_SSL_Labs-State_of_SSL_InfoSec_World_April_2011.pdf`.

[Sch03] J. Schaad. Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS). RFC 3565, Internet Engineering Task Force, July 2003.

[SSA+09] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In *Advances in Cryptology – CRYPTO'09*, 2009.

[YRS+09] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In Anja Feldmann and Laurent Mathy, editors, *Proceedings of IMC 2009*, pages 15–27. ACM Press, November 2009.