Design Categories: **Defense in Depth** and **Monitoring/Traceability**

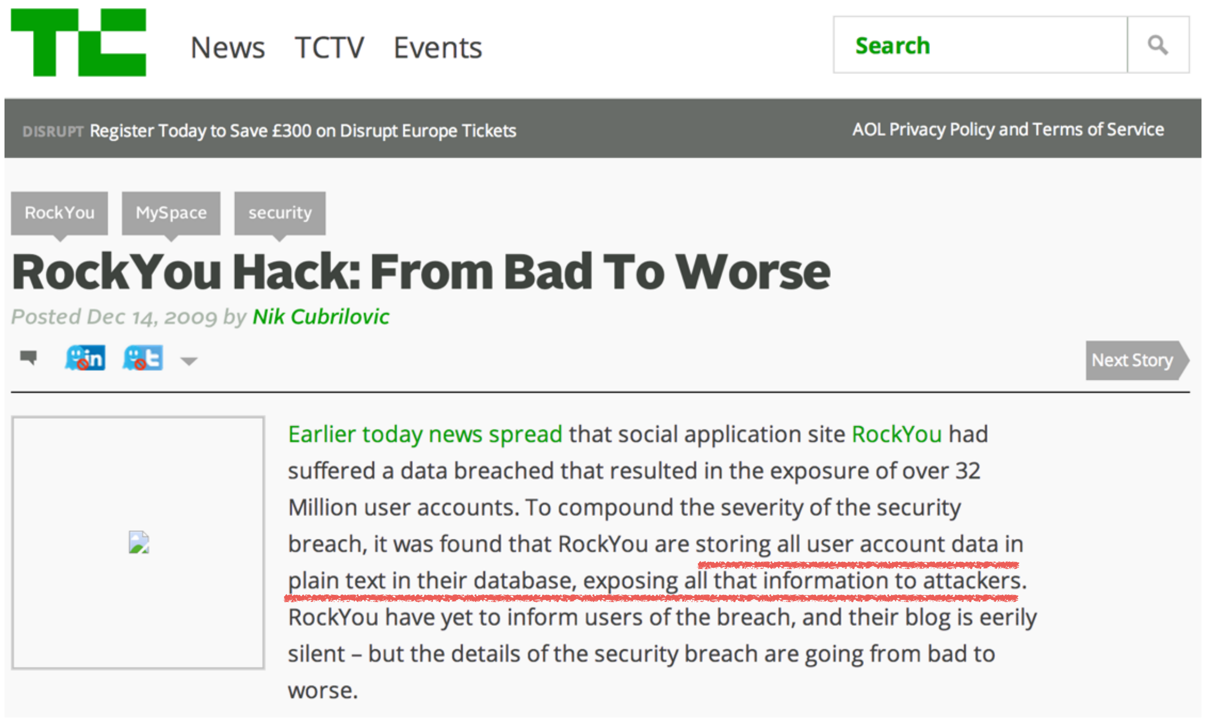
Defense in Depth (DiD)

- **Security by diversity**
  - If one layer is broken, there is another of a materially different character that needs to be bypassed
  - **Categories**: Prevention/Mitigation

- **Example**: Do *all of the following*, not just one
  - Use a firewall for preventing access via non-web ports
  - Encrypt account data at rest
  - Use a safe language for avoiding low-level vulnerabilities

Failure: Authentication Bypass

- **(Poor) passwords can be guessed**
  - bypassing authentication process intent
- **Passwords can be stolen**
  - **Defense in depth**: Should encrypt the password database
    - Assumes that compromise is possible, and thus requires additional defense


News
TCTV
Events
Search
DISRUPT Register Today to Save £300 on Disrupt Europe Tickets
AOL Privacy Policy and Terms of Service
RockYou
MySpace
security
Next Story

RockYou Hack: From Bad To Worse

Posted Dec 14, 2009 by Nik Cubrilovic

Earlier today news spread that social application site RockYou had suffered a data breached that resulted in the exposure of over 32 Million user accounts. To compound the severity of the security breach, it was found that RockYou are storing all user account data in plain text in their database, exposing all that information to attackers. RockYou have yet to inform users of the breach, and their blog is eerily silent – but the details of the security breach are going from bad to worse.

DiD: Use community resources

- **Use hardened code**, perhaps from other projects
  - E.g., **crypto libraries**
  - But make sure it meets your needs (*test it*; cf. Heartbleed!)

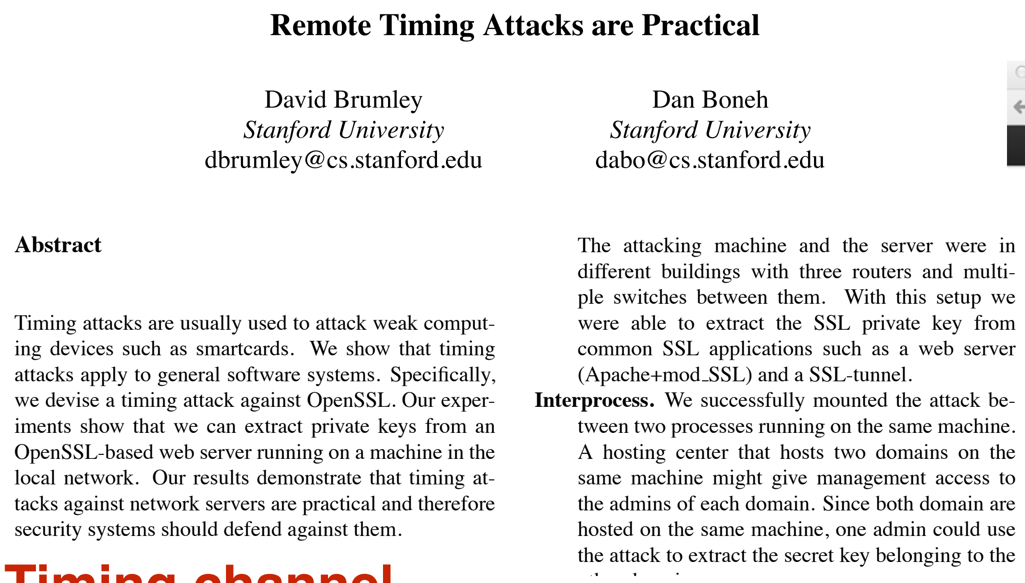
- **Vet designs publicly**: *No security by obscurity!*

- **Stay up on recent threats and research**
  - **NIST** for **standards**
  - **OWASP**, **CERT**, **Bugtraq** for **vulnerability reports**
  - **SANS** *Newsbites* for **latest top threats**
  - Academic and industry **conferences and journals** for **longer term trends**, **technology**, and **risks**

Failure: Broken Crypto Impl.

Getting crypto right is hard

Use vetted implementations and algorithms



Remote Timing Attacks are Practical

David Brumley
Stanford University
dbrumley@cs.stanford.edu

Dan Boneh
Stanford University
dabo@cs.stanford.edu

Abstract

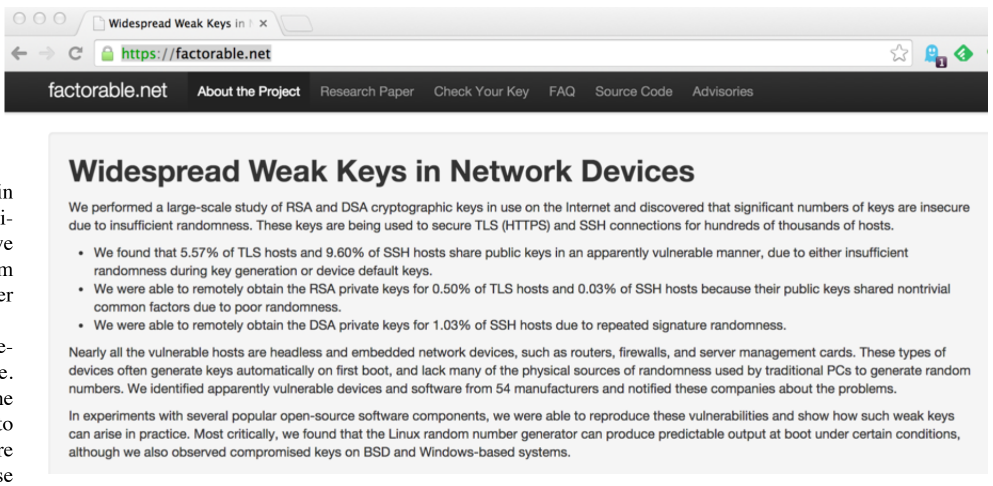
Timing attacks are usually used to attack weak computing devices such as smartcards. We show that timing attacks apply to general software systems. Specifically, we devise a timing attack against OpenSSL. Our experiments show that we can extract private keys from an OpenSSL-based web server running on a machine in the local network. Our results demonstrate that timing attacks against network servers are practical and therefore security systems should defend against them.

The attacking machine and the server were in different buildings with three routers and multiple switches between them. With this setup we were able to extract the SSL private key from common SSL applications such as a web server (Apache+mod_SSL) and a SSL-tunnel.

Interprocess. We successfully mounted the attack between two processes running on the same machine. A hosting center that hosts two domains on the same machine might give management access to the admins of each domain. Since both domain are hosted on the same machine, one admin could use the attack to extract the secret key belonging to the

Timing channel

USENIX Security'03



Widespread Weak Keys in Network Devices

We performed a large-scale study of RSA and DSA cryptographic keys in use on the Internet and discovered that significant numbers of keys are insecure due to insufficient randomness. These keys are being used to secure TLS (HTTPS) and SSH connections for hundreds of thousands of hosts.

- We found that 5.57% of TLS hosts and 9.60% of SSH hosts share public keys in an apparently vulnerable manner, due to either insufficient randomness during key generation or device default keys.
- We were able to remotely obtain the RSA private keys for 0.50% of TLS hosts and 0.03% of SSH hosts because their public keys shared nontrivial common factors due to poor randomness.
- We were able to remotely obtain the DSA private keys for 1.03% of SSH hosts due to repeated signature randomness.

Nearly all the vulnerable hosts are headless and embedded network devices, such as routers, firewalls, and server management cards. These types of devices often generate keys automatically on first boot, and lack many of the physical sources of randomness used by traditional PCs to generate random numbers. We identified apparently vulnerable devices and software from 54 manufacturers and notified these companies about the problems.

In experiments with several popular open-source software components, we were able to reproduce these vulnerabilities and show how such weak keys can arise in practice. Most critically, we found that the Linux random number generator can produce predictable output at boot under certain conditions, although we also observed compromised keys on BSD and Windows-based systems.

Poor randomness

USENIX Security'12

Monitoring and Traceability

- **If you are attacked, how will you know it?**
  - Once you learn, **how will you discern the cause?**

- Software must be designed to **log relevant operational information**
  - What to log? E.g., events handled, packets processed, requests satisfied, …
  - **Category:** Detection and Recovery

- **Log aggregation**: Correlate activities of multiple applications when diagnosing a breach
  - E.g., `splunk` log aggregator