

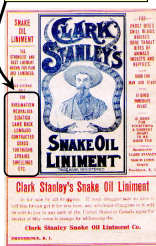
Illusions of Security

An invited talk at the 11th USENIX Security Workshop

Paul Kocher
 President & Chief Scientist,
 Cryptography Research, Inc.
 www.cryptography.com

602 Market St., 5th Floor, San Francisco, CA 94105

"For: Rheumatism, Neuralgia, Sciatica, Lame Back, Lumbago, Contracted Cords, Toothache, Sprains, Swellings, Etc."



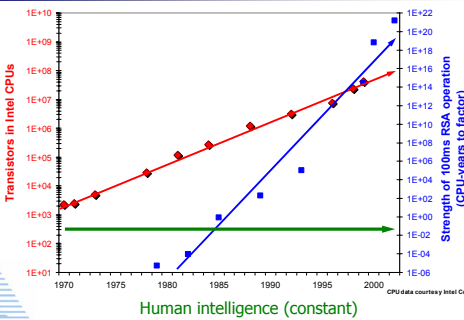
© 2002 Cryptography Research, Inc. All rights reserved. The Cryptography Research logo is a trademark of Cryptography Research, Inc. All trademarks are the property of their respective owners. The information contained in this presentation is provided without any guarantee or warranty whatsoever.

About Cryptography Research

- Specialize in high-risk commercial systems
 - Highly technical; Hands-on + theoretical expertise
 - Crypto, risk management, hardware, networking...
 - Industries: Financial, content, networking, wireless
- Consulting, licensing & research
 - Consulting: Evaluation, implementation, design
 - Licensing: Tamper-resistance/DPA technologies
 - Research: Real attacks & countermeasures
- Emphasis on applied work
 - Practical, reliable solutions to real problems
 - Systems designed by CRI engineers protected >\$40B in 2001
 - Most of our revenue from big companies with real losses



Moore's Law & Security



Security vs. Functionality

- Non-security software tolerates bugs in proportion to complexity → Example: Microsoft Word
- Most security systems have zero-tolerance for flaws → Example: Buffer overflows in rarely-used functions
- Observation: # of security flaws is proportional to complexity squared (# interactions = complexity²) → As complexity [e.g., LOC] increases, the effort to maintain a given level of assurance [e.g., defect count] rises exponentially.

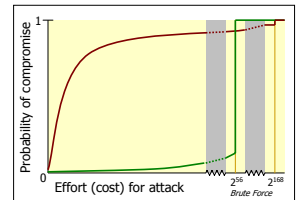
Security ≠ Functionality.
 Moore's Law drives functionality... but security requires demonstrating the absence of undesirable functionality.



© 2002 by Paul Kocher

Measuring Security

- Upper bound?
- Expected/mean?
- Risk curve?
- Against who?
 - Initial vs. repeat attack?
 - Creative adversaries or Uncreative adversaries?



- Most commercial products: Negligible probability of being very secure against creative attackers.
 - Microsoft Windows XP, Linux, X.509 cert parsers, JavaScript, sendmail, garage crypto, web browsers...



Characteristics of Most Flaws

- SPFs that bypass the crypto algorithms
 - Buffer overflows, alg negotiation, scripting, RNGs...
- Interactions between components
 - Incorrect assumptions
 - memcmp timing, bignum limits, system()...
- Inexperienced engineers
 - Use of abstraction without understanding
 - Laziness, overconfidence...
 - Poor practices (obscurity, poor docs, no reviews)
- Complexity without constraints
 - Lack of sandboxing, isolation...

All are major problems in commercial security products.



Evaluation Goals

- Possible results from security tests:
 - [A] Security is provably bad, or
 - [B] Inconclusive
- Typical CRI evaluation goals:
 - Prove that security is bad.
 - Assess the likelihood of additional serious problems.
 - Advise whether a product is worth deploying.
- Attacking is much easier than designing or verifying.
 - Preventing / Testing for single problems is easy.
 - Preventing / Testing for all known problems is hard.
 - Preventing / Testing for all possible problems is impossible.



Evaluation Constraints

CRI median eval: ~80 person-hours / ~\$32K
(Mean time/cost is much higher due to a few huge projects)

Many constraints on the process:

- Time
- Budget
- Technical information
- Evaluator capabilities / experience
- Knowledge of threat model



Homework

- Define target's security objectives
 - Comparative analysis
 - Compare with similar or sketch a design
 - Anything missing? extraneous? confusing?
 - How have similar systems failed?
 - Risk management or impenetrability?
- Implementation details
 - Product architecture
 - Design compromises, unsolvable problems?
 - Underlying technologies
 - Understand all layers of abstraction / layers
 - Perimeters, trust boundaries
 - What is trusted? What crosses boundaries?
 - Algorithms & state machines

Business
Network
Protocol
Crypto
Software
OS
CPU
Microcode
Logic cell
Transistor



Single Points of Failure

Examples

ROM/E2/BIOS contents	Hard disk controllers	Data backup & disaster recovery
Key storage & metadata	Revocation systems	Crypto algorithms & protocols
Error handling / attack detection	Engineering processes	Operating systems & Drivers
Executable program storage	Compilers	CPU execution correctness
Sandboxing	Non-sandboxed code	Hardware features (buses, etc.)
Input validation routines	Passwords & login procedures	Tamper resistance
Software update procedures	Master keys & passwords	Manufacturing / distribution



Collecting Information

What is known about the target?

- Published specifications
- Open literature



Collecting Information

- Published specifications
- Open literature
- Network & bus I/O**

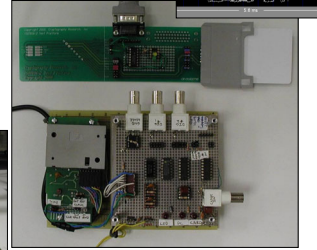
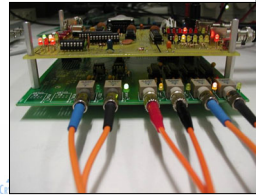


Recording traffic using a SCSI bus analyzer.



Collecting Information

- Published specifications
- Open literature
- Network & bus I/O
- Timing**
- Power consumption**

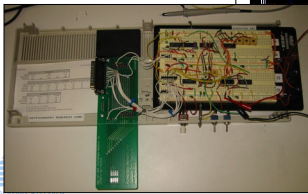
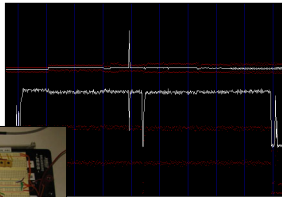


Hardware for monitoring I/O, timing, and power data from smart cards & other crypto chips



Collecting Information

- Published specifications
- Open literature
- Network & bus I/O
- Timing
- Power consumption
- Defective computations**



Collecting Information

- Published specifications
- Open literature
- Network & bus I/O
- Timing
- Power consumption
- Defective computations
- Error messages**
- Failure modes**

Step 2a: Searching with one interval left. Observe if M_1 contains only one interval $[a, M_1]$ or $[M_1, b]$, then choose small integer values r_1, n_1 such that

$$r_1 \geq 2^{2k-1} - \frac{2k}{n_1} \quad (1)$$

and

$$\frac{2k + 2k}{n_1} \leq a_1 < \frac{2k + 2k}{n_1} \quad (2)$$

and the interval $[a_1, M_1]$ is a PCH containing

Step 2b: Narrowing the set of solutions. After a_1 has been found, the set M_1 is computed as

$$M_1 = \bigcup_{i \in \mathbb{Z}} \left\{ \left\lfloor \frac{2k + 2k}{n_1} \right\rfloor + i \cdot \left\lfloor \frac{2k - 1 + 2k}{n_1} \right\rfloor \right\} \quad (3)$$

for all $i, j \in \mathbb{Z}$, M_1 and $a_1 - 2k + 1 \leq i \leq a_1 - 2k$

Step 4: Computing the solution. If M_1 contains only one interval of length l (for $l \leq 2k - 1$), then the interval $[a_1, M_1]$ must contain no solutions $x \geq 2$.

Given a valid RSA signature:
 $S = M^d \pmod{n}$, where $n = p \cdot q$
 and a defective signature S' that is correct mod p and incorrect mod q ,
 then:
 $\rho = \text{GCD}(n, S - S')$
 $q = p / \rho$.



Collecting Information

- Published specifications
- Open literature
- Network & bus I/O
- Timing
- Power consumption
- Defective computations
- Error messages
- Failure modes
- Disk/memory contents**
- Swap files**

```

C:\>dir /ah 777*.*.sys
Volume in drive C: is PCHD_LAPTOP
Volume Serial Number is 0CCB-7738

Directory of C:\

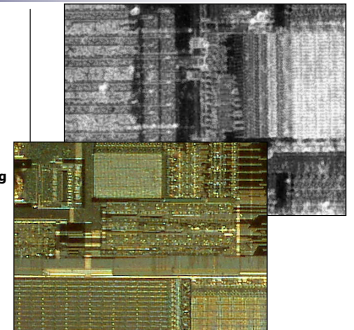
01/30/2002  11:12a         401,588,224 hiberfil.sys
01/30/2002  11:11a         201,326,592 PAGEFILE.SYS
                2 File(s)      602,914,816 bytes
                0 Dir(s)      8,513,707,520 bytes free

C:\>
    
```



Collecting Information

- Published specifications
- Open literature
- Network & bus I/O
- Timing
- Power consumption
- Defective computations
- Error messages
- Failure modes
- Disk/memory contents
- Swap files
- Chip imaging/probing**



Collecting Information

- Published specifications
- Open literature
- Network & bus I/O
- Timing
- Power consumption
- Defective computations
- Error messages
- Failure modes
- Disk/memory contents
- Swap files
- Chip imaging
- RNG seed data**

```
RNG_CreateContext()
(seconds, microseconds) = time of day;
/* Time elapsed since 1970 */
pid = process ID;
ppid = parent process ID;
a = mklcpr(microseconds);
b = mklcpr(pid + seconds + (ppid << 12));
seed = MD5(a, b);
mklcpr(x) /* not cryptographically significant;
shown for completeness */
return ((0xDEECE66D*x+0x2BBB62DC)>>1);
```

Netscape 1.1 seeding process (pseudocode)
From: Goldberg, Ian and Wagner, David, "Randomness and the Netscape Browser", Dr. Dobbs Journal, Jan. 1996



Collecting Information

- Published specifications
- Open literature
- Network & bus I/O
- Timing
- Power consumption
- Defective computations
- Error messages
- Failure modes
- Disk/memory contents
- Swap files
- Chip imaging
- RNG seed data
- Backup / restore**
- Illegal / questionable activities**
 - Dumpster diving
 - Inside jobs
 - Social engineering
 - Physical / network attacks



Collecting Information

- Published specifications
- Open literature
- Network & bus I/O
- Timing
- Power consumption
- Defective computations
- Error messages
- Failure modes
- Disk/memory contents
- Swap files
- Chip imaging
- RNG seed data
- Backup / restore
- Illegal / questionable activities
- Code reviews**
 - Very hard to do in volume
 - Code can be unreviewable

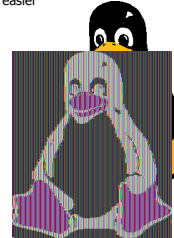
```
bigmath - Microsoft Developer Studio - [F]:bigmath...
File Edit View Insert Project Build Tools Window Help
[Global] [All global members] [bigAdd]
#include <math.h>
void bigAdd(a, b, c)
{
    bigAddition(a, b, c);
    SIGN(c) + SIGN(a);
}
#define LONG_SHIFTSTEP(i, C, A, B) { \
    sum = (long){A[(i)] - (B[(i)]); \
    suh = (long){-(0long)suh > A[(i)]}; \
    (C[(i)] + 0long){(long)suh + carry}; \
    carry = (long){-(C[(i)] > 0long)suh} + suh; \
}
#define LONG_BORROWSTEP(i, C, A) { \
    (C[(i)] + 0long){(long){A[(i)] + carry}; \
    carry = (long){-(C[(i)] > A[(i)]}; \
}
void K_AND_R
void bigsub(a, b, c)
{
    ...
}
```

CryptoLib bug: Error if src=dest and incoming carry (borrow) causes borrow.



Algorithms & Usage

- Proprietary/new algorithms: Guilty until proven innocent...
 - Designers often inexperienced, paranoid, overconfident
 - Security based on a "hard" problem: Are you sure?
 - Cryptanalysis is tedious - other attacks may be easier
- Good algorithms are often used badly
 - Marginal key sizes (e.g., DES)
 - Insecure at 1/2 key length? 1/m trade-off...
 - Multiple use of secrets?
 - Keys storage & management
 - Key lifecycle: Generation, Revocation, Deletion
 - Secret sharing is good, but definitely no cure-all
 - Incorrect computations
 - Bugs, buffer overflows, glitches...
 - Is the correct value used as the key?
 - DES S tables, PGPDisk CAST bug...
 - Stream cipher key reuse
 - CBC adapt. chosen plaintext attacks
 - DES MACs, 3DES with internal CBC...
 - ECB



Protocol Analysis

- Many approaches (intuitive to formal).
- Example:
 - Describe what is supposed to happen
 - E.g.: Client & server negotiate a strong shared key or fail.
 - On three (big) pieces of paper:
 - Chart the protocol flow
 - Include every message that can be sent
 - Error messages, optional messages, etc.
 - List what can be discovered about each cryptographic value.
 - Each crypto step generally reveals something new.
 - List everything (helps catch unintended interactions)
 - Diagram the state machine of each participant
 - Include negotiated options, failure states, crypto, etc.
 - Reconcile possible end states against objectives.
 - Check for missing "free" functionality, excessive complexity...



Common Protocol Weak Spots

- Algorithm negotiation
- Version negotiation (backward + forward)
- Man-in-the-middle
- Message replay (within a session, multiple sessions)
- Message forwarding & impersonation
 - E.g.: A connects to B, who connects to C pretending to be A.
- Certificate handling & validation (or lack thereof)
- Out-of-sequence messages
- Error handling reveals information
- Denial of service
- Timing attacks
- Excessive complexity or lack of defined state machine
- Improper or inadequate use of hash functions
- Inefficiencies (round trips...)
- Redundant information
- Management/debug functions (code upgrades, etc.)



Designing for Security

(Quite Different from Attacking...)

Challenges to address

- Increasing connectedness & complexity
 - Exponential growth in transaction volumes
 - Continuously changing engineering environments
 - Theorists & engineers don't (can't?) talk
 - Users enamored with new features
 - Inadequate security budgets
 - Deeper layers of abstraction
 - Attacks are unpredictable
 - Shrinking ratio of good engineers to problems
- Fraction of data that is well understood is plummeting



10 Things to Do

1 Design for testability

- Verification can be >10X design + implement cost
- Challenge: Testing the "glue"
- Goal: Make problems easy to detect
 - Elegant design security problems visible



10 Things to Do

1 Design for testability

2 Avoid complexity

- Excessive complexity is a security flaw
- All parts should exist for a reason
- 80/20 rule



10 Things to Do

1 Design for testability

2 Avoid complexity

3 Isolate complex components

- Sandboxing
- Component isolation
- Firewalls, verifiers...



10 Things to Do

1 Design for testability

2 Avoid complexity

3 Isolate complex components


4 Check your assumptions

- Goal: Rational paranoia
- Check for:
 - Thorough, clear documentation
 - Third-party verification, if appropriate
 - Solid understanding of your requirements
- Example:
 - Assume that users are lazy and stupid.



10 Things to Do


- 1 Design for testability
- 2 Avoid complexity
- 3 Isolate complex components
- 4 Check your assumptions
- 5 **Spend money rationally**
 - Don't under- or over-spend on security
 - Spend early
 - Hire experienced people



Cryptography Research, Inc. Leader In Advanced Cryptosystems™ 31

10 Things to Do


- 1 Design for testability
- 2 Avoid complexity
- 3 Isolate complex components
- 4 Check your assumptions
- 5 Spend money rationally
- 6 **Focus on interfaces**
 - Most problems are due to unexpected interactions between components designed by different people



Cryptography Research, Inc. Leader In Advanced Cryptosystems™ 32

10 Things to Do


- 1 Design for testability
- 2 Avoid complexity
- 3 Isolate complex components
- 4 Check your assumptions
- 5 Spend money rationally
- 6 Focus on interfaces
- 7 **Be humble**
 - Don't trust yourself – have your work checked
 - It isn't secure just because you can't break it.
 - Have general approaches + specifics reviewed.
 - Use outside resources & technology.
 - Don't expect (or force) others to trust you.



Cryptography Research, Inc. Leader In Advanced Cryptosystems™ 33

10 Things to Do


- 1 Design for testability
- 2 Avoid complexity
- 3 Isolate complex components
- 4 Check your assumptions
- 5 Spend money rationally
- 6 Focus on interfaces
- 7 Be humble
- 8 **Don't re-invent the wheel**
 - Use tried & true designs
 - ... but also beware of interface complexity



Cryptography Research, Inc. Leader In Advanced Cryptosystems™ 34

10 Things to Do


- 1 Design for testability
- 2 Avoid complexity
- 3 Isolate complex components
- 4 Check your assumptions
- 5 Spend money rationally
- 6 Focus on interfaces
- 7 Be humble
- 8 Don't re-invent the wheel
- 9 **Avoid single points of failure**
 - Redundancy increases the odds of survival...



Cryptography Research, Inc. Leader In Advanced Cryptosystems™ 35

10 Things to Do

- 1 Design for testability
- 2 Avoid complexity
- 3 Isolate complex components
- 4 Check your assumptions
- 5 Spend money rationally
- 6 Focus on interfaces
- 7 Be humble
- 8 Don't re-invent the wheel
- 9 Avoid single points of failure
- 10 **Study all layers of the system**
 - Transistors up to business objectives



Cryptography Research, Inc. Leader In Advanced Cryptosystems™ 36

What doesn't work

- Designs by committee
 - Conflicting objectives, no responsibility
- Obscurity
 - Increases cost for initial attack, but not repeat attacks
 - Not useful if design is public (i.e., in most commercial products)
 - Reduces relying party's ability to gain assurance
- Fixed certification standards
 - Standardized evaluations catch standardized attacks...
- Re-use of components with complex interfaces
 - SHABlock(data,len) = good, system("cmd") = bad
- Requirements that push the limits
 - Security: ↓ speed, ↓ features, ↑ cost, ↑ dev. time
- Iterative / evolutionary testing
 - Need security by design: Feedback is often unavailable



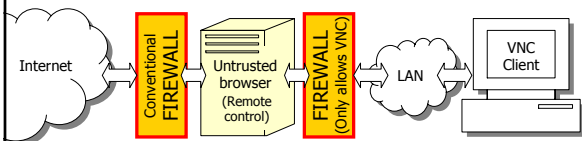
Example: Untrusted browsing

■ Traditional model



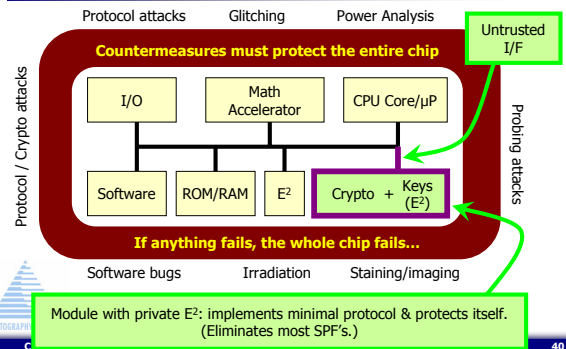
Example: Untrusted browsing

■ Better model



- Could also use compartments
 - Needs HW support (e.g., B1-level access control)

Ex: Tamper Resistant Chip

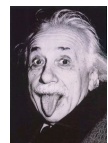


The Future...

- Ongoing challenges:
 - Convincing vendors to spend on prevention
 - Profits from fraud lead to more crime
 - Evangelizing.
 - CRI: Consulting, anti-piracy, licensing/DPA...
 - Moral hazard: Little vendor incentive for security
 - People: Training, Education, Hiring
 - Few understand theory & practice...
 - Few understand multiple levels
 - HW, SW, network, business, transistor, OS, RF...



"The problems that exist in the world today cannot be solved by the level of thinking that created them."



Contact Information

For more information, or to discuss how Cryptography Research, Inc. can help with a security problem, contact myself or Carter Laren:

Paul Kocher
paul@cryptography.com

or

Carter Laren
carter@cryptography.com

www.cryptography.com
Tel: 415-397-0123

