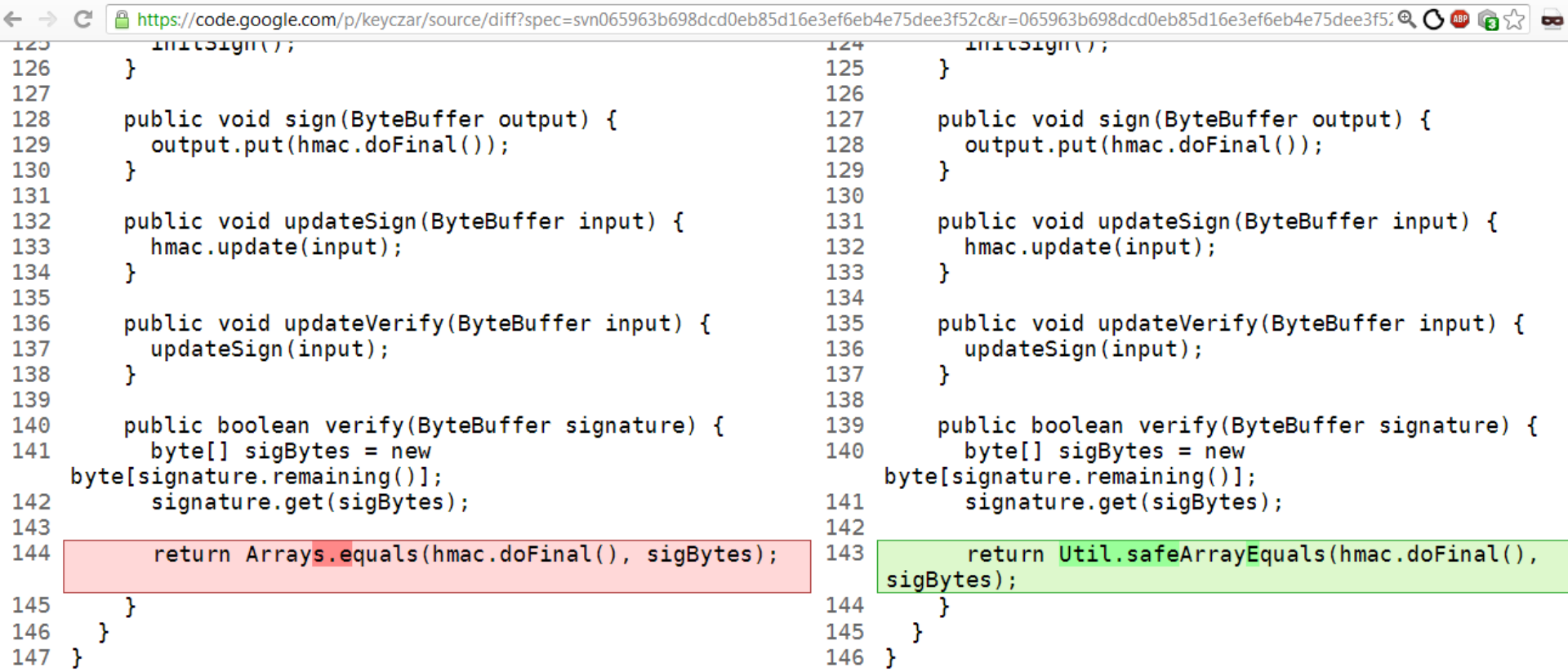
a cryptography coding standard

?

because even experts make mistakes



because even experts make mistakes

If the encoding operation outputs "message too long," output "message too long" and stop. If the encoding operation outputs "intended encoded message length too short," output "RSA modulus too short" and stop.

4. Compare the encoded message *EM* and the second encoded message *EM'*. If they are the same, output "valid signature"; otherwise, output "invalid signature."

PKCS#1 recommendation

```
int RSA_padding_check_PKCS1_type_1(unsigned char *to, int tlen,
        const unsigned char *from, int flen, int num)
    {
    int i,j;
    const unsigned char *p;

    p=from;
    if ((num != (flen+1)) || (*(p++) != 01))
        {
        RSAerr(RSA_F_RSA_PADDING_CHECK_PKCS1_TYPE_1,RSA_R_BLOCK_TYPE_IS_NOT_01);
        return(-1);
        }

    /* scan over padding data */
    j=flen-1; /* one for type. */
    for (i=0; i<j; i++)
        {
        if (*p != 0xff) /* should decrypt to 0xff */
            {
            if (*p == 0)
                { p++; break; }
            else    {
                RSAerr(RSA_F_RSA_PADDING_CHECK_PKCS1_TYPE_1,RSA_R_BAD_FIXED_HEADER_DECRYPT);
                return(-1);
                }
            }
        p++;
        }

    if (i == j)
        {
        RSAerr(RSA_F_RSA_PADDING_CHECK_PKCS1_TYPE_1,RSA_R_NULL_BEFORE_BLOCK_MISSING);
        return(-1);
        }
```

OpenSSL v1.0.1c

lots of mistakes
(SANS top 25 software errors)

Porous Defenses

The weaknesses in this category are related to defensive techniques that are often misused, abused, or just plain ignored.

CWE ID	Name
CWE-306	Missing Authentication for Critical Function
CWE-862	Missing Authorization
CWE-798	Use of Hard-coded Credentials
CWE-311	Missing Encryption of Sensitive Data
CWE-807	Reliance on Untrusted Inputs in a Security Decision
CWE-250	Execution with Unnecessary Privileges
CWE-863	Incorrect Authorization
CWE-732	Incorrect Permission Assignment for Critical Resource
CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CWE-307	Improper Restriction of Excessive Authentication Attempts
CWE-759	Use of a One-Way Hash without a Salt

we need some rules

checklists are simple and effective

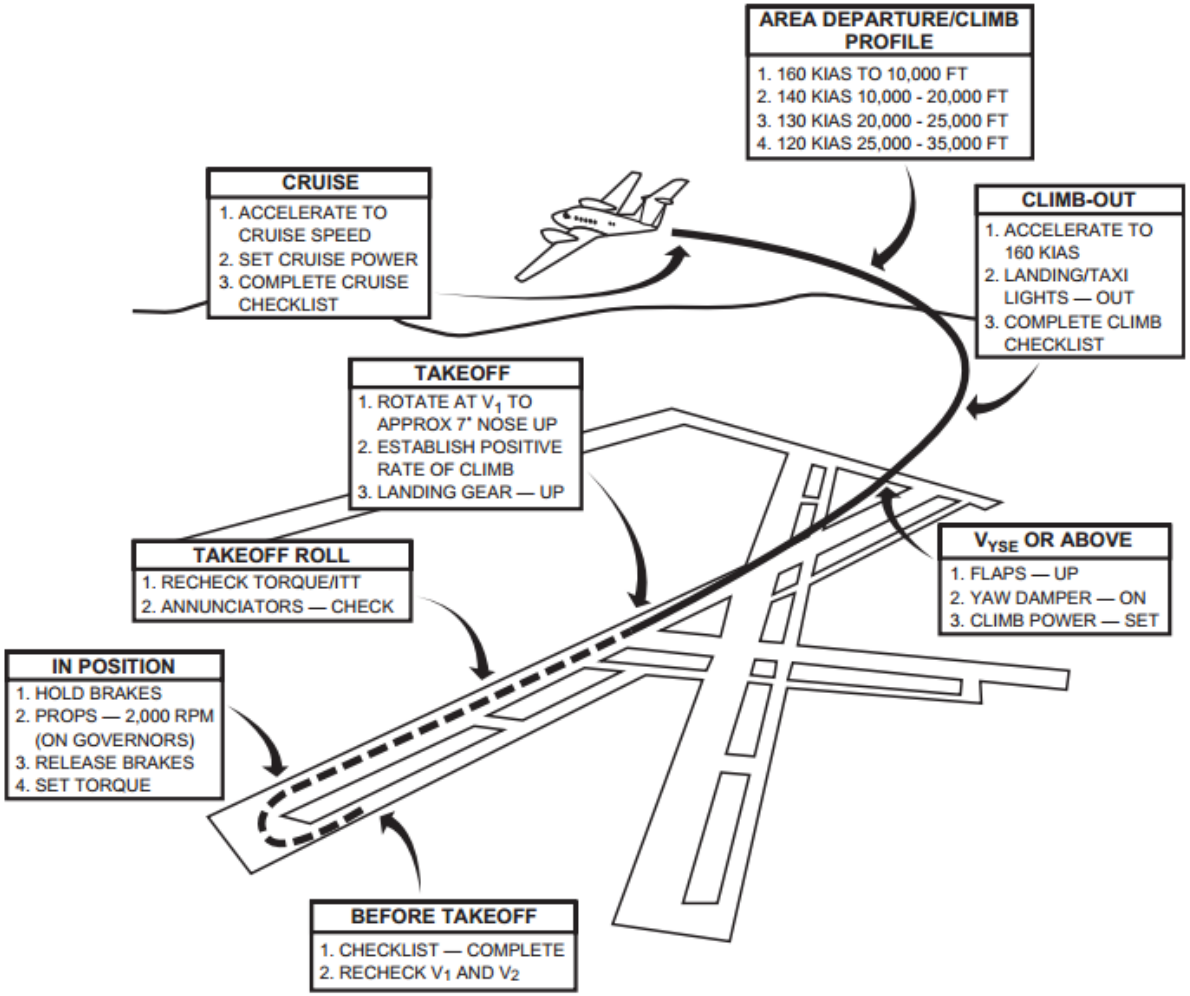


Figure GEN-1. Normal Takeoff and Departure

and familiar to programmers

spinroot.com/p10/

The Power of Ten
10 Rules for Writing Safety Critical Code

1	Restrict to simple control flow constructs.	(details)
2	Give all loops a fixed upper-bound.	(details)
3	Do not use dynamic memory allocation after initialization.	(details)
4	Limit functions to no more than 60 lines of text.	(details)
5	Use minimally two assertions per function on average.	(details)
6	Declare data objects at the smallest possible level of scope.	(details)
7	Check the return value of non-void functions, and check the validity of function parameters.	(details)
8	Limit the use of the preprocessor to file inclusion and simple macros.	(details)
9	Limit the use of pointers. Use no more than two levels of dereferencing per expression.	(details)
10	Compile with all warnings enabled, and use one or more source code analyzers.	(details)

Based on: "The Power of Ten -- Rules for Developing Safety Critical Code," *IEEE Computer*, June 2006, pp. 93-95 (PDF).

and familiar to programmers

JPL DOCID D-60411

Rule Summary

1 Language Compliance	
1	Do not stray outside the language definition.
2	Compile with all warnings enabled; use static source code analyzers.
2 Predictable Execution	
3	Use verifiable loop bounds for all loops meant to be terminating.
4	Do not use direct or indirect recursion.
5	Do not use dynamic memory allocation after task initialization.
*6	Use IPC messages for task communication.
7	Do not use task delays for task synchronization.
*8	Explicitly transfer write-permission (ownership) for shared data objects.
9	Place restrictions on the use of semaphores and locks.
10	Use memory protection, safety margins, barrier patterns.
11	Do not use goto, setjmp or longjmp.
12	Do not use selective value assignments to elements of an enum list.
3 Defensive Coding	
13	Declare data objects at smallest possible level of scope.
14	Check the return value of non-void functions, or explicitly cast to (void).
15	Check the validity of values passed to functions.
16	Use static and dynamic assertions as sanity checks.
*17	Use U32, I16, etc instead of predefined C data types such as int, short, etc.
18	Make the order of evaluation in compound expressions explicit.
19	Do not use expressions with side effects.
4 Code Clarity	
20	Make only very limited use of the C pre-processor.
21	Do not define macros within a function or a block.
22	Do not undefine or redefine macros.
23	Place #else, #elif, and #endif in the same file as the matching #if or #ifdef.
*24	Place no more than one statement or declaration per line of text.
*25	Use short functions with a limited number of parameters.

and familiar to programmers

Secure Coding Guidelines for the Java Programming Language, Version 4.0

- Introduction
- 0 Fundamentals
- 1 Denial of Service
- 2 Confidential Information
- 3 Injection and Inclusion
- 4 Accessibility and Extensibility
- 5 Input Validation
- 6 Mutability
- 7 Object Construction
- 8 Serialization and Deserialization
- 9 Access Control
- Conclusion
- References

and familiar to programmers

Security > Home > **Writing Secure Code**

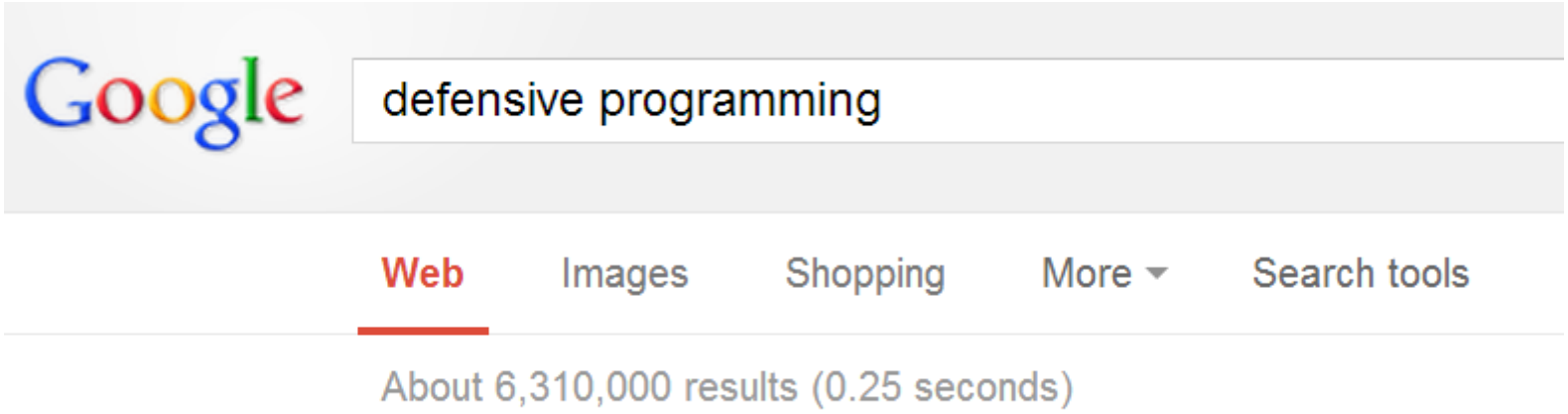
Writing Secure Code

One of the key things that developers can do to help secure their systems is to write code that can withstand attack and use security features properly. This page contains links to best practices and how-to articles on writing secure code.

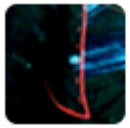
Getting Started

- The Security Development Lifecycle Process
- 10 Security Tips: Defend Your Code with Top Ten Security Tips Every Developer Must Know
- Lessons Learned from Five Years of Building More Secure Software
- Security Compliance as an Engineering Discipline
- An Overview of Security in the .NET Framework
- Web Application Security Fundamentals

plenty of resources



but much less for crypto



JP Aumasson
@aumasson

where can I find the coding rules of OpenSSL (if any)?

Reply Delete Favorite

7:07 PM - 6 Jan 13 · Embed this Tweet

Reply to @aumasson

Shhhhhhh @greenestfield 7m
@aumasson Rule #1 of the OpenSSL coding rules: you don't talk about the OpenSSL coding rules.
Details

Adam Langley @agl__ 5m
@aumasson I just make it look like the surrounding code. I dream someday that we can run indent over it.
Details

the closest I found

nacl.cr.yp.to/internals.html

You can use names other than `hash.c`. You can split your code across several files `*.c` defining various auxiliary functions; the compiled together. You can use external names prefixed by the implementation name: for example, `crypto_hash/sha512/c` `crypto_hash_sha512_core2_iv`, `crypto_hash_sha512_core2_expand`, etc.

Branches

Do not use secret data to control a branch. In particular, do not use the `memcmp` function to compare secrets. Instead use `crypt` `crypto_verify_32`, etc., which perform constant-time string comparisons.

Even on architectures that support fast constant-time conditional-move instructions, always assume that a comparison in C is com conditional move. Compilers can be remarkably stupid.

Array lookups

Do not use secret data as an array index.

Early plans for NaCl would have allowed exceptions to this rule inside primitives specifically labelled `vulnerable`, in particular `crypto_stream_aes128vulnerable`, but subsequent research showed that this compromise was unnecessary.

Dynamic memory allocation

Do not use heap allocators (`malloc`, `calloc`, `sbrk`, etc.) or variable-size stack allocators (`alloca`, `int x[n]`, etc.) in C N

Thread safety

Do not use global variables (i.e., static variables or variables defined outside functions) in C NaCl.

Alignment

Do not assume that the input arrays or output arrays have any particular alignment. If you want to use, e.g., an aligned 16-byte lo

starting this project after I had to write my own crypto coding rules…

inspiration: PTES

www.pentest-standard.org/index.php/Main_Page

Page | Read | View source | View history | Sear

Main Page

Welcome to the Penetration Testing Execution Standard homepage. This will be the ultimate home for the penetrati

For more information on what this standard is, please visit:

- The Penetration Testing Execution Standard: FAQ

High Level Organization of the Standard

- Note: This is the BETA RELEASE. We have had TONS of interest from many members of the security communi where we were at. This effort has been going on since November 2010 and has had over 1800 revisions. The li we are at today.

What we are looking for out of this release:

-Gain help from people who understand the direction of the map and will be willing to document the methods used to branches

-Take feedback and comments form the community on improvements

-Identify the next phase in terms of defining "levels" for each of the sections.

-Create teams to tackle writing our the formal standard

-Create tools to address the gaps identified during the creation of the Standard

-And most of all, put an end to the poorly defined term Penetration Test!

Navigation

Main page
PTES Technical Guideline
In the Media
FAQ

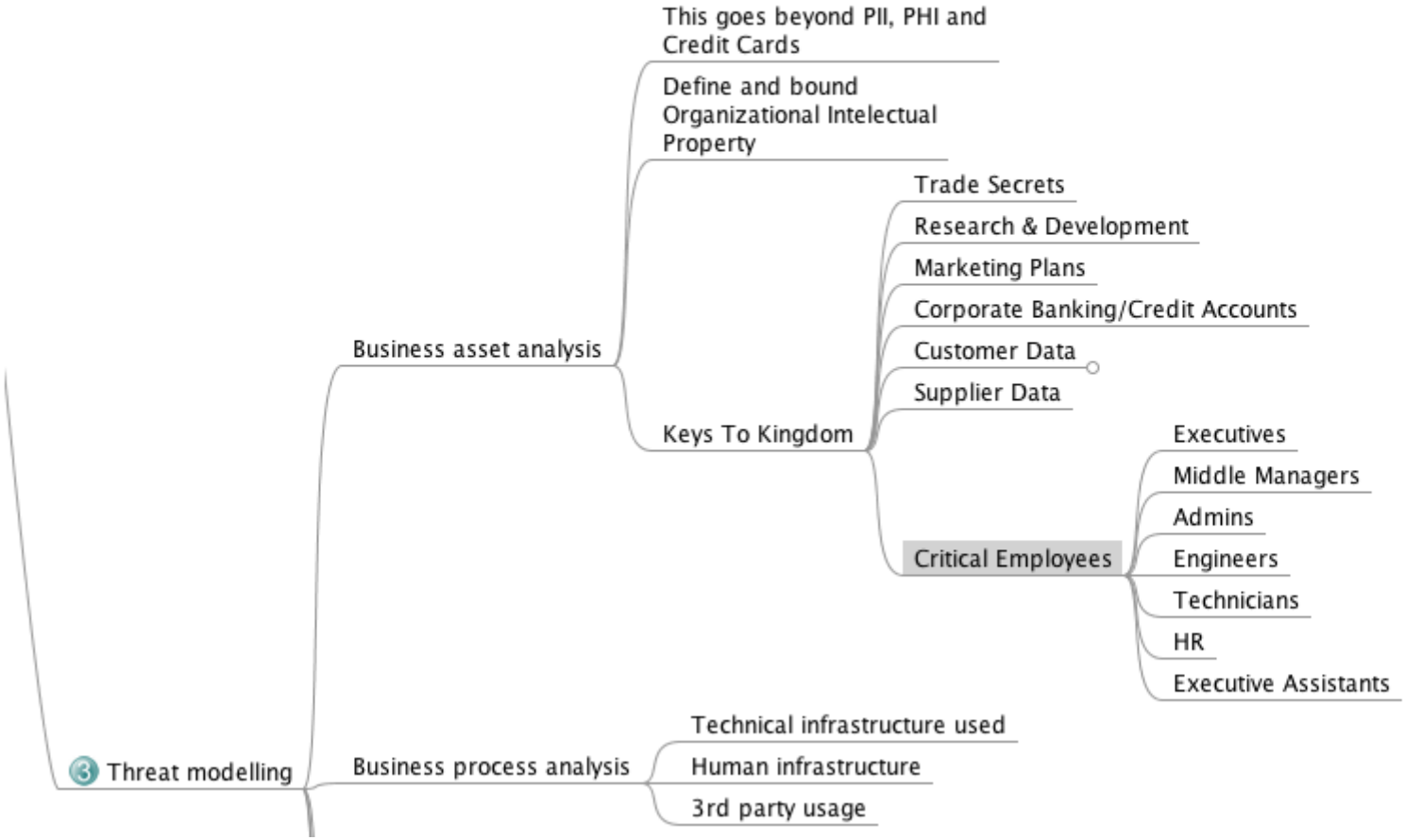
Toolbox

What links here
Related changes
Special pages
Printable version
Permanent link

checklists by experienced professionals

Threat Modeling

This phase details the elements that are part of the threat modelling (based on the intelligence gathered and the pre-engagement information)
Following is an image depicting the main branches of the corresponding mindmap:



similar motivations

Q: Is this going to be a formal standard?

A: We are aiming to create an actual standard so that businesses can have a baseline of what is needed when they get a pentest as well as an understanding of what type of testing they require or would provide value to their business. The lack of standardization now is only hurting the industry as businesses are getting low-quality work done, and practitioners lack guidance in terms of what is needed to provide quality service.

how should this look like for crypto?

list of rules

What: make MAC verification in constant time
Why: MACs could be forged as follows…
How: the following C code performs constant-time…
Example: Keyczar…

What: use a strong pseudorandom generator
Why: weak RNGs may reduce the search space…
How: code to use /dev/urandom, CryptGenRandom…
Reference: factorable.net…

museum of horrors

```
int
crypto_pk_private_sign_digest(....)
{
  char digest[DIGEST_LEN];
  ....
  memset(digest, 0, sizeof(digest));
  return r;
}
```


```
-BEGIN RSA PRIVATE KEY-----
 A DUCKFUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A

FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A
FUCK A DUCKFUCK A DUCKFUCK A DUCKFUCK A
```

PolarSSL Diffie Hellman Key Excha

Bugtraq ID:	46670
:	Design Error
	CVE-2011-1923
te:	Yes
:	No
shed:	Mar 03 2011 12:00AM
ted:	May 18 2012 06:00PM
t:	Larry Highsmith, Subreptio
erable:	Offspark PolarSSL 0.99-pre Offspark PolarSSL 0.14

references

NUMBER 7 — JUNE 26, 1998

RSA Laboratories'

Bulletin

News and advice on data security and cryptography

RSA Laboratories®
A Division of RSA Data Security

Recent Results on PKCS #1: RSA Encryption Standard

Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems

Paul C. Kocher

Cryptography Research, Inc.
607 Market Street, 5th Floor, San Francisco, CA 94105, USA.
E-mail: paul@cryptography.com

Abstract. By carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. Against a vulnerable system, the attack is computationally inexpensive

Cache Attacks and Countermeasures: the Case of AES

(Extended Version)

revised 2005-11-20

Dag Arne Osvik[1], Adi Shamir[2] and Eran Tromer[2]

[1] dag.arne@osvik.no
[2] Department of Computer Science and Applied Mathematics,
Weizmann Institute of Science, Rehovot 76100, Israel
{adi.shamir, eran.tromer}@weizmann.ac.il

ribe several software side-channel attacks based on inter-process leakage throu
's memory cache. This leakage reveals memory access patterns, which can be us
cryptographic primitives that employ data-dependent table lookups. The attac
ed process to attack other processes running in parallel on the same process
methods such as memory protection, sandboxing and virtualization. Some
only the ability to trigger services that perform encryption or MAC using t
as encrypted disk partitions or secure network links. Moreover, we demonstra
type of attack, which requires knowledge of neither the specific plaintexts n
rks by merely monitoring the effect of the cryptographic process on the cach
several such attacks on AES, and experimentally demonstrate their applicabili
to real systems, such as OpenSSL and Linux's dm-crypt encrypted partitions (in the latter case, t

Practical Padding Oracle Attacks

Juliano Rizzo* Thai Duong†

May 25th, 2010

ultimate goals

- become a reference for programmers
- improve state if software security
- bring coding rules to OpenSSL ;-)

what should be in/out?

IN

constant-time execution

padding/format verification

compiler “optimizations”

conformance to specs

choice of algorithms

padding oracles

stack hygiene

randomness

etc. etc.

IN?

SCA mitigation
(smartcards etc.)

masking, hiding, shuffling, etc.

simple countermeasures simple and cheap
effective ones (high-order…) more sophisticated

OUT

fault, microprobing, RE, etc.

not much to do at the code level
(or too invasive/slow)

most effective countermeasures **at HW level**

secure code is useless
if it doesn’t **work**

crypto also fails due to poor **testing**

2 test vectors are not enough

H(0x000000)=0xA74329BF
H(0xFFFFFFFF)=0x76A765E1

the right thing to do?

SUPERCOP's **checksum** checks for

buffers overlap support

off-boundaries writes

determinism

for 1, 2, 3, …, N bytes

easily extended to ciphers, MACs, etc.

lots of other issues to address

encrypt-then/and-MAC

CTR counter format

handling residues

bignum arithmetic

etc.

also:

how to write **reference code**?

less simple than it sounds…

agenda for the discussion?

discuss general **interest** of the project

define **scope** (what should be in/out?)

propose **rules**, **“horrors”**, references

find **volunteers** to edit the wiki ;-)

https://cryptocoding.net/index.php/Cryptography_Coding_Standard

Page | Discussion | Read | Edit | View

Cryptography Coding Standard

Welcome to the Cryptography Coding Standard homepage.

Navigation

Main page
FAQ

Toolbox

What links here
Related changes
Upload file
Special pages
Printable version
Permanent link

This page was last modified on 14 December 2012, at 08:54.

This page has been accessed 70 times.

Privacy policy About Cryptography Coding Standard Disclaimers

www.cryptocoding.net

thank you

a cryptography coding standard