

A Buffer Overflow Story

From Responsible Disclosure
to Closure

Agenda

1. Why the research was carried out, and how
2. Scope of the research
3. The responsible disclosure process
4. Observed fix
5. A surprise
6. Community response
7. More community response...
 - Conclusion
 - Questions/Answers/Shrugs


[stories](#)
[recent](#)
[popular](#)
[ask slashdot](#)
[book reviews](#)
[games](#)
[idle](#)
[yro](#)
[cloud](#)
[hardware](#)
[linux](#)
[management](#)
[mobile](#)
[science](#)
[security](#)

NIST Announces Round 1 Candidates For SHA-3 Competition

Posted by **Soulskill** on Sunday December 21 2008, @02:27PM
from the time-to-pick-them-apart dept.

[jd](#) writes

"NIST has [announced the round 1 candidates](#) for the [Cryptographic Hash Algorithm Challenge](#). Of the 64 who submitted entries, 51 were accepted. Of those, in mere days, one has been definitely broken, and three others are believed to have been. At this rate, it won't take the [couple of years NIST was reckoning](#) to whittle down the field to just one or two. (In comparison, the European Union version, NESSIE, received [just one](#) cryptographic hash function for its contest. One has to wonder if NIST and the crypto experts are so concerned about being overwhelmed with work for this current contest, why they all but ignored the European effort. A self-inflicted wound might hurt, but it's still self-inflicted.) Popular wisdom has it that no product will have any support for any of these algorithms for years — if ever. Of course, popular wisdom is ignoring all Open Source projects that support cryptography (including the Linux kernel) which could add support for any of these tomorrow. Does it really matter if the algorithm is found to be flawed later on, if most of these packages support algorithms known to be flawed today? Wouldn't it just be geekier to have passwords in Blue Midnight Wish or SANDstorm rather than boring old MD5, even if it makes no practical difference whatsoever?"

Ads Disabled

Thanks ag
great!

FormOfA

[Comment](#)

[Reply to "I](#)

[DoofusOfA](#)

[Reply to "I](#)

[Delete All](#)

Karma: G

[Journal](#)

Why the research was carried out and how

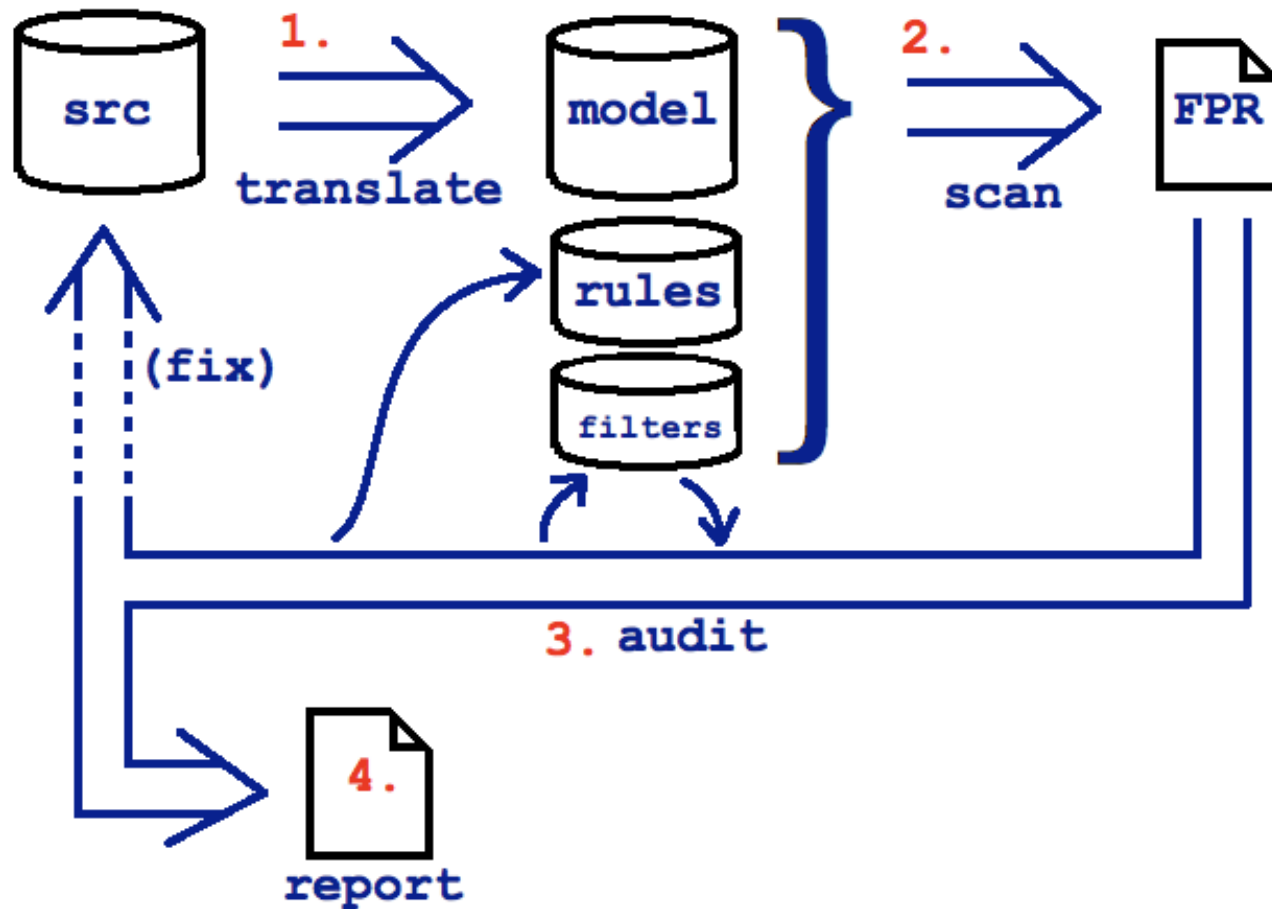
Marcus Ranum:

"Crypto code, as an underlying routine that's linked all over the place, has to be pretty much perfect."

Why the research was carried out and how

- Fortify SCA, a static analyzer

Why the research was carried out and how



How...



Fortify Audit Workbench

- A quick live demo

How...

```
$ ls
README_Reference.txt
md6_mode.c
inttypes.h
md6_nist.c
md6.h
md6_nist.h
md6_compress.c
stdint.h
$
```

- Compilation \equiv Translation

How...

```
$ gcc -c *.c  
$
```

- Compilation \equiv Translation

How...

```
$ sourceanalyzer -b md6 \  
gcc -c *.c  
$
```

- Compilation \equiv Translation
- Create a model called “md6”

How...

```
$ sourceanalyzer -b md6 \
    -scan \
    -f md6.fpr
$
```

- Analyze the model with SCA analyzers
- Print the results in a file

How...

```
$ auditworkbench md6.fpr  
$
```

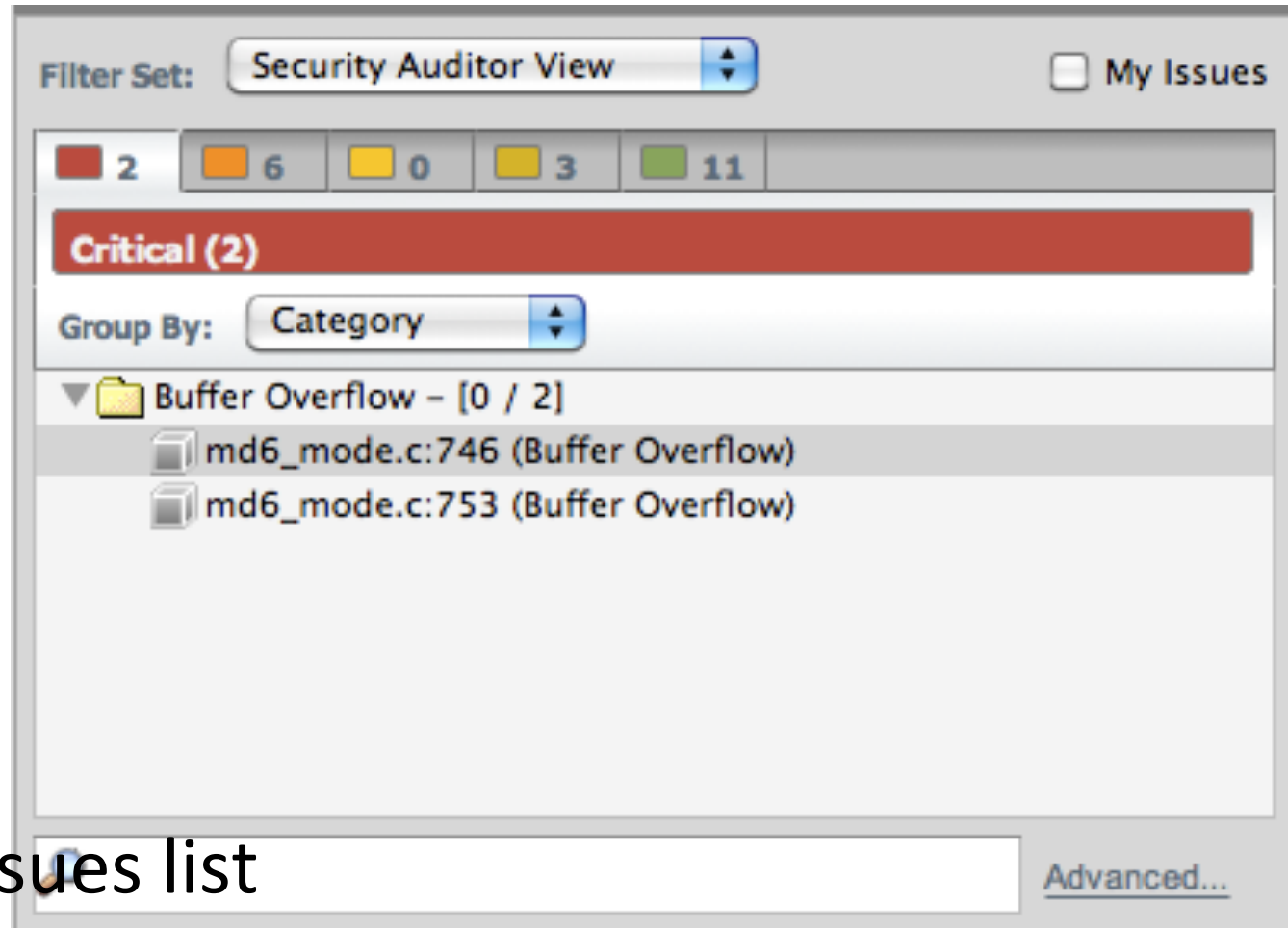
- Open the file in the graphical AWB

How...

The screenshot displays the Fortify Audit Workbench interface. At the top, there are navigation tabs: Summary, Audit Guide, Scan, and Reports. The main window is titled 'md6_mode.c'. On the left, a 'Filter Set' dropdown is set to 'Security Auditor View'. Below it, a summary bar shows 'Critical (2)' issues. A tree view under 'Buffer Overflow - [0 / 2]' lists two issues: 'md6_mode.c:746 (Buffer Overflow)' and 'md6_mode.c:753 (Buffer Overflow)'. The central pane shows the source code for 'md6_mode.c', with line 746 highlighted: `st->hashval[i] = 0;`. The right pane shows the 'Analysis Evidence' for the selected issue, including the title 'md6.h:217 - Buffer hashval Declared' and details: 'md6_mode.c:746 - Assignment to st->hashval', 'Buffer Size: 64 bytes', 'Write Length: 128 bytes', and '[var 0] i: 127'. Below this, there are tabs for 'Summary', 'Details', 'Recommendations', and 'Diagram'. The 'Abstract' section states: 'The function trim_hashval() in md6_mode.c writes outside the bounds of hashval on line 746, which could corrupt data, cause the program to crash, or lead to the execution of malicious code.' The 'Explanation' section describes buffer overflow as a common security vulnerability where data is written into an undersized stack buffer, overwriting return pointers and potentially taking control of the program.

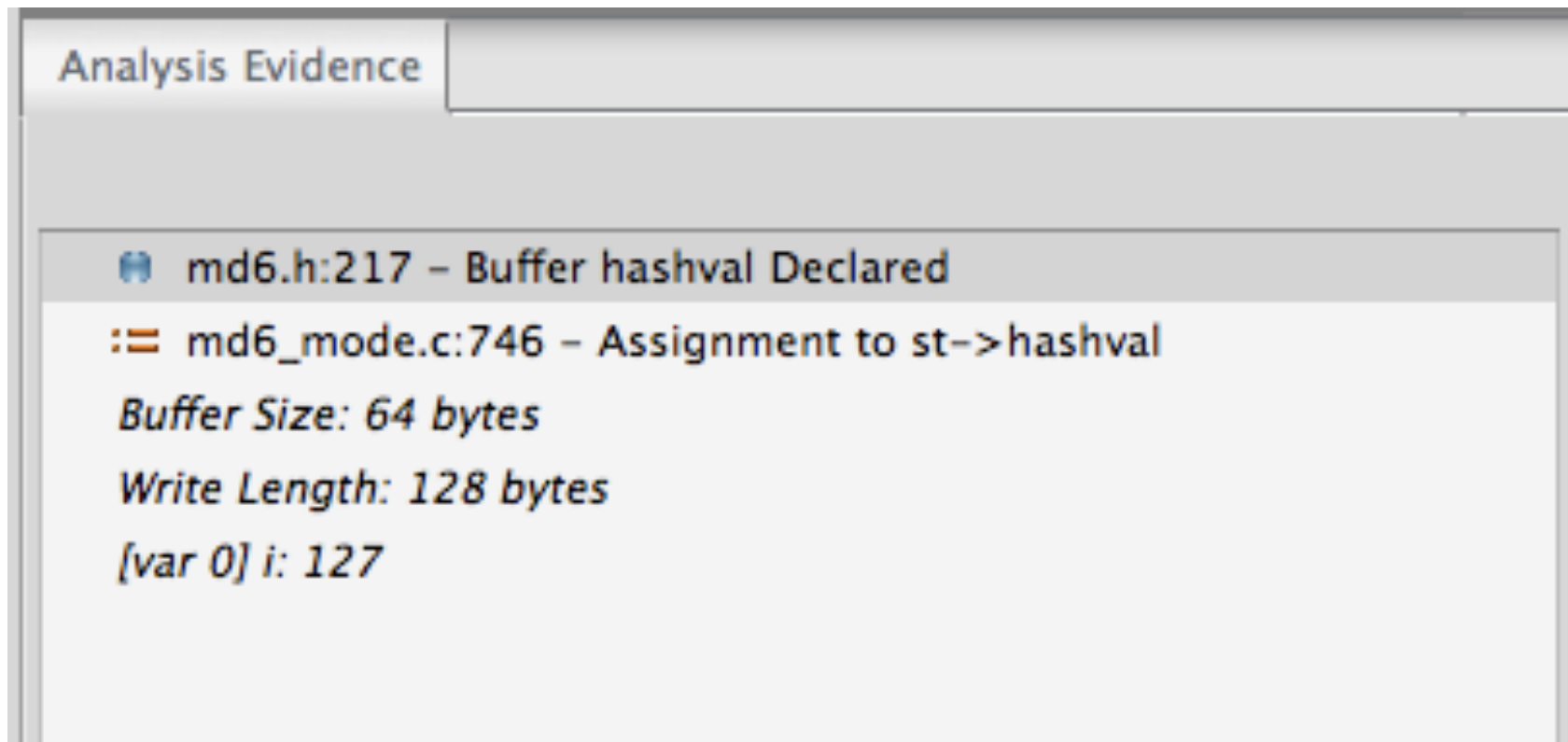
- Open the file in the graphical AWB

How...



- Issues list
- Filtering, grouping

How...



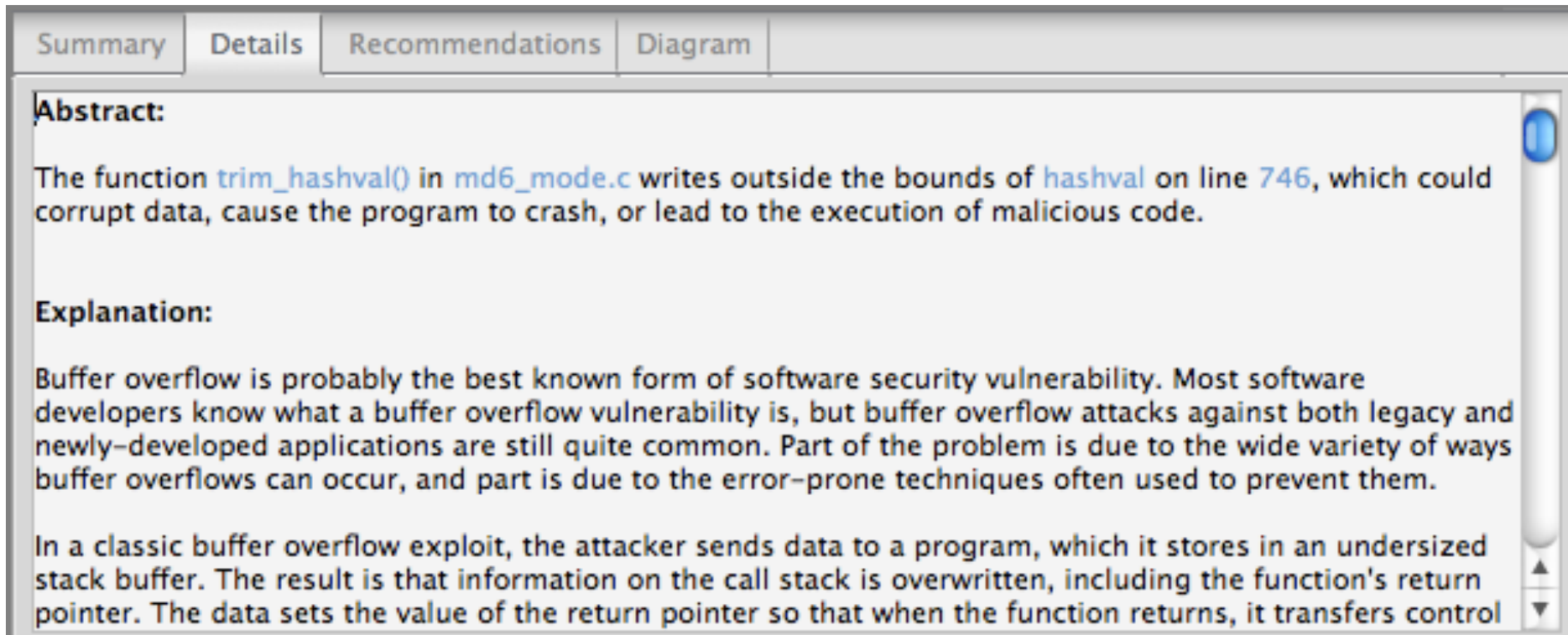
- Evidence from the analyzer
- Step through dangerous sequence

How...

```
md6_mode.c ✖
742     st->hashval[i] = st->hashval[c*(w/8)-full_or_partial_bytes+i];
743
744     /* zero out following bytes */
745     for ( i=full_or_partial_bytes; i<c*(w/8); i++ )
746         st->hashval[i] = 0;
747
748     /* shift result left by (8-bits) bit positions, per byte, if needed */
749     if (bits>0)
750     { for ( i=0; i<full_or_partial_bytes; i++ )
751       { st->hashval[i] = (st->hashval[i] << (8-bits));
752         if ( (i+1) < c*(w/8) )
753             st->hashval[i] |= (st->hashval[i+1] >> bits);
754       }
755     }
756 }
757
758
759 /* Final -- no more data; finish up and produce hash value.
760 */
761
```

- Contextual view of the source code

How...



Summary Details Recommendations Diagram

Abstract:

The function `trim_hashval()` in `md6_mode.c` writes outside the bounds of `hashval` on line 746, which could corrupt data, cause the program to crash, or lead to the execution of malicious code.

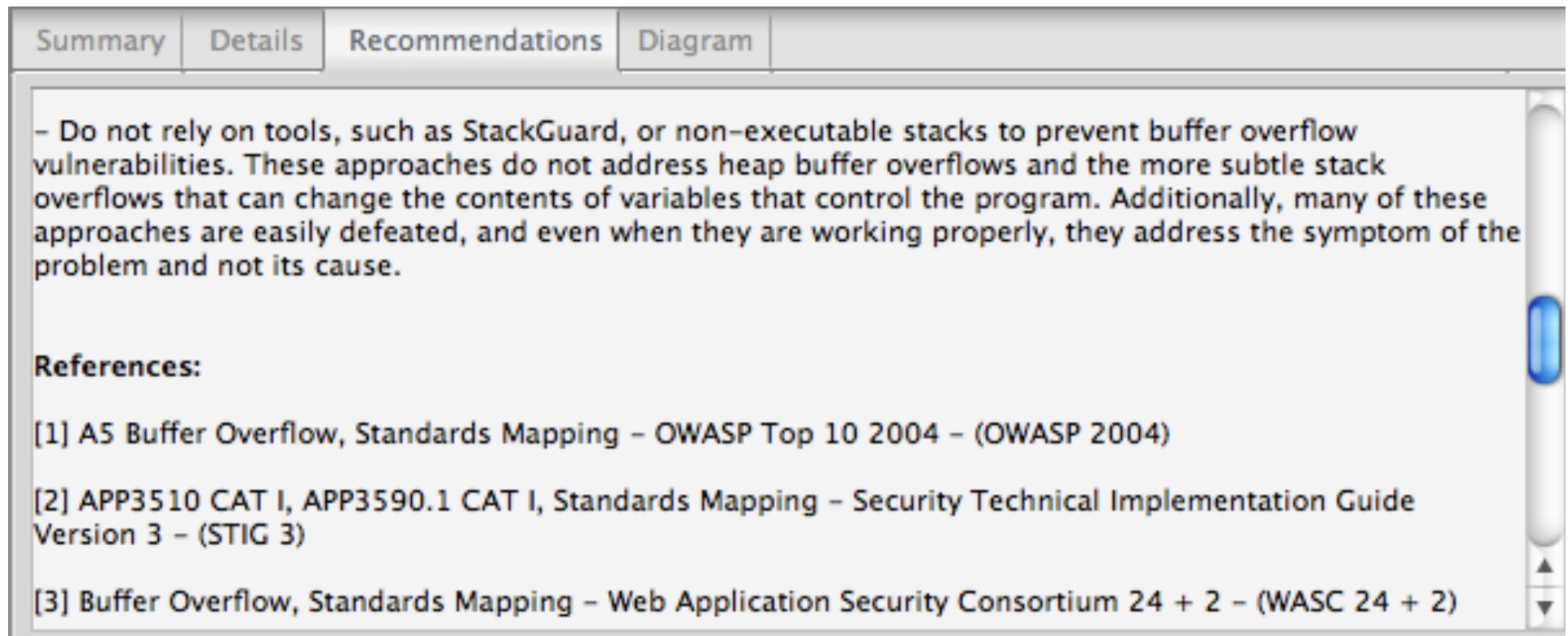
Explanation:

Buffer overflow is probably the best known form of software security vulnerability. Most software developers know what a buffer overflow vulnerability is, but buffer overflow attacks against both legacy and newly-developed applications are still quite common. Part of the problem is due to the wide variety of ways buffer overflows can occur, and part is due to the error-prone techniques often used to prevent them.

In a classic buffer overflow exploit, the attacker sends data to a program, which it stores in an undersized stack buffer. The result is that information on the call stack is overwritten, including the function's return pointer. The data sets the value of the return pointer so that when the function returns, it transfers control

- Vulnerability description
- Suitable for a developer, new to security

How...



The screenshot shows a web application security tool interface with four tabs: Summary, Details, Recommendations, and Diagram. The Recommendations tab is active, displaying a recommendation for buffer overflow vulnerabilities. The text reads: '- Do not rely on tools, such as StackGuard, or non-executable stacks to prevent buffer overflow vulnerabilities. These approaches do not address heap buffer overflows and the more subtle stack overflows that can change the contents of variables that control the program. Additionally, many of these approaches are easily defeated, and even when they are working properly, they address the symptom of the problem and not its cause.' Below this text is a section titled 'References:' with three numbered references: [1] A5 Buffer Overflow, Standards Mapping - OWASP Top 10 2004 - (OWASP 2004); [2] APP3510 CAT I, APP3590.1 CAT I, Standards Mapping - Security Technical Implementation Guide Version 3 - (STIG 3); and [3] Buffer Overflow, Standards Mapping - Web Application Security Consortium 24 + 2 - (WASC 24 + 2).

Summary Details Recommendations Diagram

- Do not rely on tools, such as StackGuard, or non-executable stacks to prevent buffer overflow vulnerabilities. These approaches do not address heap buffer overflows and the more subtle stack overflows that can change the contents of variables that control the program. Additionally, many of these approaches are easily defeated, and even when they are working properly, they address the symptom of the problem and not its cause.

References:

[1] A5 Buffer Overflow, Standards Mapping - OWASP Top 10 2004 - (OWASP 2004)

[2] APP3510 CAT I, APP3590.1 CAT I, Standards Mapping - Security Technical Implementation Guide Version 3 - (STIG 3)

[3] Buffer Overflow, Standards Mapping - Web Application Security Consortium 24 + 2 - (WASC 24 + 2)

- Vulnerability recommendations
- Suitable for auditor, architect...

2. Scope of the research

Joy Forsythe:

"We're not hard-core cryptographers, so we decided to take a look at the reference implementations."

2. Scope of the research

Joy Forsythe:

"We're not hard-core cryptographers, so we decided to take a look at the reference implementations."

Brian Chess:

"The purpose here is not to judge the algorithm based on some implementation errors, it's to get the implementation errors fixed so that the best algorithm is selected and so that mistakes aren't propagated into production code"

3. The responsible disclosure process

Ron Rivest:

"We greatly appreciate your careful review of our submitted code, and your reporting your concerns to us. I'd like to have them reviewed quickly, and have any necessary fixes issued promptly."

From: "Doug Held" <dheld@fortify.com>
Date: 23 December 2008 21:09:07 EET
To: <rivest@mit.edu>
Subject: Possible buffer overflow in MD6 reference implementation

Hello, **Dear Professor...**

I've taken a look at your proposed MD6 implementation (from <http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/MD6.zip>) and I may have found a buffer overflow vulnerability.

In md6_mode.c, the memcpy() on line 611 copies the following length:
 $md6_c * (w / 8)$
 $= 16 * (64 / 8)$
 $= 128 \text{ bytes}$

into the buffer st->hashval which I believe is defined on line 217 of md6.h:
 $(md6_c / 2) * (md6_w / 8)$
 $= (16 / 2) * (64 / 8)$
 $= 64 \text{ bytes.}$

It's entirely possible that I've made a mistake, as I'm taking an extremely narrow look at your implementation and I'm also new to C.

If I've wasted your time I'd appreciate you letting me know. If you'd find it interesting I have twelve additional findings that I'd be happy to share.

Kind Regards,
Douglas Held

4. Observed fix

- MD6 was updated before the cutoff, with a doubling of the buffer size.

4. Observed fix

----- Original Message -----

From: Ronald L. Rivest <rivest@mit.edu>

To: Doug Held; Joy Forsythe

Cc: Jayant Krishnamurthy <jayantkrishnamurthy@example.com>

Sent: Thu Jan 15 07:52:16 2009

Subject: Re: Possible buffer overflow in MD6 reference implementation

Hi Doug and Joy --

I'm writing to thank you again for bringing this issue of a buffer overflow in our MD6 code to our attention!

We have this week filed an update to our code to NIST. (The deadline for updates is today.) It adjusts the size of the "hashval" buffer to be twice as big, which fixes the main problem you detected. (The other issue regarding static allocation of an array we have not

5. Surprise

- Brian Chess noticed on someone's blog that the Conficker worm ***was now updated with the patch.***

5. Surprise

- I guess the bad guys read the news, too.

5. Surprise

- I guess the bad guys read the news, too.
- Kind of have to wonder whether I played this right...?

5. Surprise

- I guess the bad guys read the news, too.
- Kind of have to wonder whether I played this right...?
- *Whose side are we on??*

The Public Disclosure

- Fortify Security Research blog
 - <http://blog.fortify.com>
- Slashdot article submission

Subscribe

- Feedburner (RSS)
- Local (RSS)
- Local (RSS v2.0)
- Local (Atom)

Recent Posts

- ▶ [Back to All Posts](#)

Categories

- ▶ Fortify
- ▶ Healthcare
- ▶ News
- ▶ Random
- ▶ Research
- ▶ Vulnerabilities-Breaches

About

Fortify Software

Fortify Resources

Software Security Resources
Products and Services

Recommended Blogs

- ▶ Justice League
- ▶ Schneier on Security
- ▶ SecurityFix
- ▶ CERIAS
- ▶ Security Curve

Friday, 20 February 2009

SHA-3 Round 1: Buffer Overflows

[« Gartner Magic Quadrant for Static Analysis | Main | SHA-3 Analysis Details »](#)

NIST is currently holding a [competition](#) to choose a design for the SHA-3 algorithm (Bruce Schneier has a good description of [secure hashing algorithms and why this is important](#)). The reference implementations of a few of the contestants have bugs in them that could cause crashes, performance problems, or security problems if they are used in their current state. Based on our bug reports, some of those bugs have already been fixed. Here's the full story:

The main idea behind the competition is to have the cryptographic community weed out the less secure algorithms and choose from the remainder. A couple of us at Fortify (thanks to Doug Held for his help) decided to do our part. We're not hard-core cryptographers, so we decided to take a look at the reference implementations.

This competition is to pick an algorithm, but all of the submissions had to include a C implementation, to demonstrate how it works and test the speed, which will be a factor in the final choice. We used Fortify SCA to audit the [42 projects accepted into Round 1](#). We were impressed with the overall quality of the code, but we did find significant issues in a few projects, including buffer overflows in two of the projects. We have emailed the submission teams with our findings and one team has already corrected their implementation.

Confirmed issues:

Implementation	Buffer Overflow	Out-of-bounds Read	Memory Leak	Null Dereference
Blender	1	0	0	0
Crunch	0	0	0	4
FSB	0	0	3	11
MD6	3	2	0	0
Vortex	0	0	1	15

Subscribe

- Feedburner (RSS)
- Local (RSS)
- Local (RSS v2.0)
- Local (Atom)

Recent Posts

- ▶ Back to All Posts

Categories

- ▶ Fortify
- ▶ Healthcare
- ▶ News
- ▶ Random
- ▶ Research
- ▶ Vulnerabilities-Breaches

About

Fortify Software

Fortify Resources

Software Security Resources
Products and Services

Recommended Blogs

- ▶ Justice League
- ▶ Schneier on Security
- ▶ SecurityFix
- ▶ CERIAS
- ▶ Security Curve

Friday, 20 February 2009

SHA-3 Round 1: Buffer Overflows

[« Gartner Magic Quadrant for Static Analysis](#) | [Main](#) | [SHA-3 Analysis Details](#) [»](#)

NIST is currently holding a [competition](#) to choose a design for the SHA-3 algorithm (Bruce Schneier has a good description of [secure hashing algorithms and why this is important](#)). The reference implementations of a few of the contestants have bugs in them that could cause crashes, performance problems, or security problems if they are used in their current state. Based on our bug reports, some of those bugs have already been fixed. Here's the full story:

The main idea behind the competition is to have the cryptographic community weed out the less secure algorithms and choose from the remainder. A couple of us at Fortify (thanks to Doug Held for his help) decided to do our part. We're not hard-core cryptographers, so we decided to take a look at the reference implementations.

This competition is to pick an algorithm, but all of the submissions had to include a C implementation, to demonstrate how it works and test the speed, which will be a factor in the final choice. We used Fortify SCA to audit the [42 projects accepted into Round 1](#). We were impressed with the overall quality of the code, but we did find significant issues in a few projects, including buffer overflows in two of the projects. We have emailed the submission teams with our findings and one team has already corrected their implementation.

Confirmed issues:

Implementation	Buffer Overflow	Out-of-bounds Read	Memory Leak	Null Dereference
Blender	1	0	0	0
Crunch	0	0	0	4
FSB	0	0	3	11
MD6	3	2	0	0
Vortex	0	0	1	15

Subscribe

- Feedburner (RSS)
- Local (RSS)
- Local (RSS v2.0)
- Local (Atom)

Recent Posts

- ▶ [Back to All Posts](#)

Categories

- ▶ Fortify
- ▶ Healthcare
- ▶ News
- ▶ Random
- ▶ Research
- ▶ Vulnerabilities-Breaches

About

Fortify Software

Fortify Resources

Software Security Resources
Products and Services

Recommended Blogs

- ▶ Justice League
- ▶ Schneier on Security
- ▶ SecurityFix
- ▶ CERIAS
- ▶ Security Curve

Friday, 20 February 2009

SHA-3 Round 1: Buffer Overflows

[« Gartner Magic Quadrant for Static Analysis | Main | SHA-3 Analysis Details »](#)

NIST is currently holding a [competition](#) to choose a design for the SHA-3 algorithm (Bruce Schneier has a good description of [secure hashing algorithms and why this is important](#)). The reference implementations of a few of the contestants have bugs in them that could cause crashes, performance problems, or security problems if they are used in their current state. Based on our bug reports, some of those bugs have already been fixed. Here's the full story:

The main idea behind the competition is to have the cryptographic community weed out the less secure algorithms and choose from the remainder. A couple of us at Fortify (thanks to Doug Held for his help) decided to do our part. We're not hard-core cryptographers, so we decided to take a look at the reference implementations.

This competition is to pick an algorithm, but all of the submissions had to include a C implementation, to demonstrate how it works and test the speed, which will be a factor in the final choice. **We used Fortify SCA to audit the [42 projects accepted into Round 1](#). We were impressed with the overall quality of the code, but we did find significant issues in a few projects, including buffer overflows in two of the projects. We have emailed the submission teams with our findings and one team has already corrected their implementation.**

Confirmed issues:

Implementation	Buffer Overflow	Out-of-bounds Read	Memory Leak	Null Dereference
Blender	1	0	0	0
Crunch	0	0	0	4
FSB	0	0	3	11
MD6	3	2	0	0
Vortex	0	0	1	15

Subscribe

- Feedburner (RSS)
- Local (RSS)
- Local (RSS v2.0)
- Local (Atom)

Recent Posts

- ▶ [Back to All Posts](#)

Categories

- ▶ Fortify
- ▶ Healthcare
- ▶ News
- ▶ Random
- ▶ Research
- ▶ Vulnerabilities-Breaches

About

Fortify Software

Fortify Resources

Software Security Resources
Products and Services

Recommended Blogs

- ▶ Justice League
- ▶ Schneier on Security
- ▶ SecurityFix
- ▶ CERIAS
- ▶ Security Curve

Friday, 20 February 2009

SHA-3 Round 1: Buffer Overflows

[« Gartner Magic Quadrant for Static Analysis | Main | SHA-3 Analysis Details »](#)

NIST is currently holding a [competition](#) to choose a design for the SHA-3 algorithm (Bruce Schneier has a good description of [secure hashing algorithms and why this is important](#)). The reference implementations of a few of the contestants have bugs in them that could cause crashes, performance problems, or security problems if they are used in their current state. Based on our bug reports, some of those bugs have already been fixed. Here's the full story:

The main idea behind the competition is to have the cryptographic community weed out the less secure algorithms and choose from the remainder. A couple of us at Fortify (thanks to Doug Held for his help) decided to do our part. We're not hard-core cryptographers, so we decided to take a look at the reference implementations.

This competition is to pick an algorithm, but all of the submissions had to include a C implementation, to demonstrate how it works and test the speed, which will be a factor in the final choice. We used Fortify SCA to audit the [42 projects accepted into Round 1](#). We were impressed with the overall quality of the code, but **we did find significant issues in a few projects, including buffer overflows in two of the projects. We have emailed the submission teams with our findings and one team has already corrected their implementation.**

Confirmed issues:

Implementation	Buffer Overflow	Out-of-bounds Read	Memory Leak	Null Dereference
Blender	1	0	0	0
Crunch	0	0	0	4
FSB	0	0	3	11
MD6	3	2	0	0
Vortex	0	0	1	15

[stories](#)[recent](#)[popular](#)[ask slashdot](#)[book reviews](#)[games](#)[idle](#)[yro](#)[cloud](#)[hardware](#)[linux](#)[management](#)[mobile](#)[science](#)[security](#)[storage](#)

Security Review Summary of NIST SHA-3 Round 1

Posted by [timothy](#) on Sunday February 22 2009, @07:15PM
from the works-in-progress dept.

[FormOfActionBanana](#) writes

"The security firm [Fortify Software](#) has undertaken an automated code review of the [NIST SHA-3 round 1 contestants \(previously Slashdotted\)](#) reference implementations. After a followup audit, the team is now [reporting summary results](#). According to the blog entry, 'This just emphasizes what we already knew about C, even the most careful, security conscious developer messes up memory management.' Of particular interest, Professor [Ron Rivest's](#) (the "R" in RSA) [MD6](#) team has already corrected a buffer overflow pointed out by the Fortify review. Bruce Schneier's Skein, [also previously Slashdotted](#), came through defect-free."

This story has **146 Comments**

Read similar stories with these tags

 encryption security technology Itroll lobscurity story

You may also like to read

Ads Disable

Thanks a
great!

6. Community response



6. And more community response



6. Community response

"It's not security"

Hardly "memory management" (Score:3, Insightful)

by SoapBox17 (1020345) on Sunday February 22 2009, @07:57PM (#26950787) Homepage

From TFA (and TFS):

| This just emphasizes what we already knew about C, even the most careful, security conscious developer messes up memory management.

This doesn't follow from TFA. The blog points out two instances of buffer overflows. The first one you could argue they messed up "memory management" because they used the wrong bounds for their array in several places... but they don't sound very "careful" or "security conscious" since checking to make sure you understand the bounds of the array you're using is pretty basic.

But that's not what bothered me. The second example is a typo where TFA says someone entered a "3" instead of a "2". In what dimension is mis-typing something "messing up memory management"? That just doesn't follow.

Re: Exactly, how are we supposed to trust their algorithm that is barely understandable by 1 in 400 million people when they cannot get a simple C

6. Community response

"It's not security"

In defense of C (Score:4, Insightful)

by phantomfive (622387) on Sunday February 22 2009, @08:18PM (#26950937) Journal

The summary is kind of a troll, since most of the submissions actually managed to get through without ANY buffer overflows.

Buffer overflows are not hard to avoid, they are just something that must be tested. If you don't test, you are going to make a mistake, but they are easy to find with a careful test plan or an automated tool. Apparently those authors who had buffer overflows in their code didn't really check for them.

C is just a tool, like any other, and it has tradeoffs. The fact that you are going to have to check for buffer overflows is just something you have to add to the final estimate of how long your project will take. But C gives you other advantages that make up for it. Best tool for the job, etc.

7. Community response

Blame the language!

7. Community response

Blame the language!

XOR

Blame the developer!

7. Community response

Blame the language!

ANSI C (Score:4, Insightful)

by chill (34294) on Sunday February 22 2009, @07:23PM (#26950515) Journal

That is what they get for mandating the code be in ANSI C. How about allowing reference implementation in SPARK, ADA or something else using design-by-contract. After all, isn't something as critical as a international standard for a hash function the type of software d-b-c was meant for?

7. Community response

Blame the language!

C isn't the problem, it is really... (Score:3, Insightful)

by MobyDisk (75490) on Sunday February 22 2009, @08:46PM (#26951135) [Homepage](#)

I suspect the problem is related to the poor coding practices used in academia. I see college professors who write code that barely compiles in GCC without a bunch of warnings about anachronistic syntax. Some of the C code used constructs that are unrecognizable to someone who learned the language within the past 10 years, and is completely type unsafe.

I can't tell much from the code on the link, but I do see `#define` used for constants which is no longer appropriate (yet is EXTREMELY common to see). C99 had the `const` keyword in it, probably even before that.

7. Community response

Blame the language!


Comment: C++ Coder at Sun, 22 Feb 2:07 PM

I think the key was Name of a security developer *AND HIS TEAM*. What do we know of them? **Apaprently** they **AREN'T** the most security concious developers. C is by far the most powerful flexible language, and a **truely aware team can write secure code. To attack C is ludicrous.** Yeah! Lets write everything in JAVA! or interpret/babysit everyone! This isn't the answer. If we **truely** make security a focus, C coders can be trained to do things right. Unfortunatly now, usually the focus is features, minimal budget, and short schedules. Note security wasn't listed.

7. Community response

Blame the language!

Re: this is why... (Score:3, Insightful)

by tepples (727027)  <{/slash2006} {at} {pineight.com}> on Monday February 23 2009, @02:57AM (#26953863) Homepage Journal

If you're still writing unmanaged code, you get what you deserve. It's 2009, not 1989.

Try running managed code in the 4 MB RAM of a widely deployed handheld computer. Now try making that managed code time-competitive and space-competitive with an equivalent program in C++ compiled to a native binary.

Re:ANSI C (Score:5, Insightful)

by Anonymous Coward on Sunday February 22 2009, @07:44PM (#26950683)

What did they get? You realize this is just an ad for Fortify, right? Out of 42 projects, they found 5 with memory management issues using their tool. Maybe instead of switching to SPARK, the 5 teams that fucked up could ask the 37 that didn't for some tips on how to write correct C.

Re: Mod parent up. Thanks. Annoying SLASHVERTISEMENT!

dandelion orchard (Score:3, Insightful)

by epine (68316) on Sunday February 22 2009, @10:30PM (#26951935)

This just emphasizes what we already knew about C, even the most careful, security conscious developer messes up memory management.

I know nothing of the sort. How about asking some developers who have a history of getting both the security and the memory management correct which intellectual challenge they lose the most sleep over?

The OpenBSD team has a history of strength in both areas. I suspect most of these developers would laugh out loud at the intellectual challenge of the memory management required to implement a hash function. It's about a hundred thousand lines of code short of where the OpenBSD team gets grey hair over memory management problems in the C language.

I wouldn't hire a programmer who can't get memory management right to take on any significant intellectual challenge. It's just a way to feel good about yourself without having the aptitude to cut your way out of a wet paper bag.

90% of software development projects are not aptitude driven. Let's stop fooling ourselves into thinking that the languages that work well in those contexts having anything to offer those of us dealing with the other 10%

Memory management is a subcase of resource management with a particularly harsh way of delivering the news: you suck. A memory managed language deprives the environment of so many golden opportunities to deliver this message, despite the fact that you still suck. By the time you don't suck, you've ceased to regard unmanaged memory as a core intellectual challenge, and trained yourself to work within an idiom where you hardly ever get it wrong anyway.

7. Community response

Blame the language!

The developer simply mistyped, using 3 instead of 2 for the array access. This issue was probably not caught because it would not be exposed without a very large input. The other issues we found were memory leaks and null dereferences from memory allocation.

This just emphasizes what we already knew about C, even the most careful, security conscious developer messes up **memory management**. Some of you are saying, so what? These are reference implementations and this is only Round 1. There are a few problems with that thought.

Reference implementations don't disappear, they serve as a starting point for future implementations or are used directly. A [bug in the RSA reference implementation](#) was responsible for vulnerabilities in OpenSSL and two separate SSH implementations. They can also be used to design hardware implementations, using buffer sizes to decide how much silicon should be used.

7. Community response

Blame the language!

In defense of C (Score:4, Insightful)

by phantomfive (622387) on Sunday February 22 2009, @08:18PM (#26950937) Journal

The summary is kind of a troll, since most of the submissions actually managed to get through without ANY buffer overflows.

Buffer overflows are not hard to avoid, they are just something that must be tested. If you don't test, you are going to make a mistake, but they are easy to find with a careful test plan or an automated tool. Apparently those authors who had buffer overflows in their code didn't really check for them.

C is just a tool, like any other, and it has tradeoffs. The fact that you are going to have to check for buffer overflows is just something you have to add to the final estimate of how long your project will take. But C gives you other advantages that make up for it. Best tool for the job, etc.

7. Community response


Blame the developer!

[memory leak... WTF???](#) (Score:0) why the fuck dynamic memory allocation is used in a C implementation of a hashing algorithm?

7. Community response

Blame the language!

Re:dandelion orchard (Score:3, Insightful)

by Luke has no name (1423139)  <`moc.liamg' `ta' `emanonsahekul"> on Monday February 23 2009, @01:23AM (#26953323)

What I get out of this:

"We're going to give you more shit to think about by making you use C. if you can't deal with all the stupid shit C throws at you, you suck."

Which is a shit argument. Just use a better language that gives people less to worry about, and develop from there. Having to debug the shit out of a program for obscure memory management issues shouldn't be a test of your competence. You should be able to focus on the task at hand, nothing else.

[Parent](#)

8. Community response

- It's only a reference implementation
- It's only a competition

Comment: Nick at Sun, 22 Feb 1:05 PM

Dyanmo is also wrong;

The point here is analyzing the underlying algorithms, not some arbitrary pseudocode implementation.

Why not analyze these functions for something that /matters/? Like collision resistance, differential cryptanalysis resistance, hash length extension attacks, etc.?

Oh, right, because that requires actual knowledge of cryptography. You fail.

Comment: PaulWay at Sun, 22 Feb 4:58 PM

I think some commenters are missing the point here – these code tests are only one of the tests being applied to the algorithms. There is, of course, exhaustive cryptanalysis being applied to them in parallel to check for the more fundamental problems in the algorithms. However, testing the reference code for standard coding errors is vital as reference code `_does_` get used a lot, even where faster or more secure implementations might exist elsewhere. Having this kind of code audit is vital to the security process but it is by no means the only step.

Comment: Hacksaw at Tue, 24 Feb 4:23 AM

mandolin: ORLY? Why do people think great mathematical skills mean you're also a great programmer? A mathematician may be good at designing concepts and high-level algorithms but actual programming is frequently too low-level. It's only human to overestimate your skills in one area you know a bit about if you excel in a related area.

Writing ***correct*** code has nothing to do with ***security***. Of course, an incorrectly implemented, yet securely designed algorithm, will likely compromise security but correctly implemented code is not secure per se at all. You wade deep in snake oil already – or just fail horribly at human language – if you use these terms as loosely.

I can tell that most people coding in C I've known, have never read any of the C standards. They only practiced learning by doing and sooner or later they'll become totally ignorant and arrogant towards the standards, so that whenever someone points out what they're doing is either not portable or obviously not standards-conform, these guys frequently ignore you, even if you provide a correction, or worse try to offend you and bash the standards for being stupid or whatever.

Schwern: C has never been a portable assembly language. Unfortunately, many people see it this way and try to use the same "tricks" that were acceptable in assembly language on a specific machine but a strictly forbidden in C. I see a parallel between modern languages and modern politics. C is one of these languages which give you a lot of freedom but also responsibility. Nowadays nobody wants to take responsibility anymore but everyone is crying for security despite knowing that perfect security exists nowhere except in theory perhaps. Likewise, this also means less freedom. I'm not saying this means anything in this context, I just find it interesting that this parallel exists.

By the way, you can do high-level programming in C, too. It'll simply take more effort but the result is likely more readable and maintainable. There are no rules that you have to obfuscate your code, use unintelligible identifiers, use brain-dead library functions etc. People who do this without any need and are even proud of it are simply abusing the freedom given to them by C.

lang: Then why are they already providing optimized code for MD6?

Last but not least, it's fairly pathetic to see these professionals don't seem to use verification tools that have been around for years.

only a reference implementation

Comment: [Brian](#) at Sun, 22 Feb 4:31 PM

Bugs in the reference implementations have the potential to effect performance. Under-allocating a buffer can also skew the cost estimate for an embedded system. In other words, these bugs have the potential to effect the outcome of the competition.

The purpose here is not to judge the algorithm based on some implementation errors, it's to get the implementation errors fixed so that the best algorithm is selected and so that mistakes aren't propagated into production code.

only a reference implementation

Comment: Matt at Mon, 23 Feb 7:04 AM

Sorry to all those in the ivory tower but in the real world the reference implementations often show up in real code (especially open source). The purpose of the competition is not to produce code snippets for use in real code but that is what happens. The comments of Brian, Brian and DaveM are spot on. In addition this is not just Fortify grandstanding but rather them providing valuable feedback to an important program. I personally appreciate this sort of industry involvement in the process. Correcting these defects early in the process is critical to insuring that clean, safe and secure code propagates in to production software.

The last word on: “It’s only a reference implementation”

Comment: [Marcus Ranum](#) at Sat, 21 Feb 11:05 AM

Good write-up. You're absolutely correct, too – crypto code, as an underlying routine that's linked all over the place, has to be pretty much perfect.

It's only a competition

Re:ANSI C (Score:3, Interesting)

by RichardJenkins (1362463) on Sunday February 22 2009, @07:29PM (#26950549)

Why do they even have a reference implementation for a hash function in a programming language? Wouldn't just defining the function mathematically be less error prone and just as effective?

[Parent](#)

The reason is in the summary... (Score:5, Insightful)

by pathological liar (659969) on Sunday February 22 2009, @07:36PM (#26950607)

... because implementation is where people screw up.

Re: 1. Because we, rank and file developers, have to use it afterward (and some of us write in C or C-derived languages, like oh, I don't know,

Re: Ummmm... because it wouldn't be a reference implementation if it wasn't actually implemented?

Re:ANSI C (Score:4, Insightful)

by John Hasler (414242) on Sunday February 22 2009, @07:42PM (#26950665)

Presumably one of the things they want to evaluate is performance.

Re:ANSI C (Score:4, Insightful)

by IversenX (713302) on Sunday February 22 2009, @07:44PM (#26950681) [Homepage](#)

Because you can't compile a mathematical definition.

If we imagine that the hash function came only as a mathematical definition, how would you test your new implementation in LangOfTheWeek is correct?

Reference implementation (Score:4, Informative)

by rgmoore (133276) <glandauer@charter.net> on Sunday February 22 2009, @07:49PM (#26950723) [Homepage](#)

In a word, no. A reference implementation is supposed to be a working version of the code, not just a mathematical description. With a working version, it's possible to do things like test its real world performance or cut and paste directly into a program that needs to use the function. That's obviously only possible if you have a version that works on real-world processors.

Comment: lang at Mon, 23 Feb 12:57 PM

It is certainly appreciated that work is put in to improve the implementations during the competition (my group did something similar for the Java parts of AES, so I know how much work it can be).

However I think it is not really efficient at this stage to insist on secure programming for submission implementations. For the simple reason that there are 42 submissions, and 41 of those will be thrown away, more or less. There isn't much point in making the 41 secure; better off to save the energy until "the one" is found. Then concentrate the energy, no?

The developer simply mistyped, using 3 instead of 2 for the array access. This issue was probably not caught because it would not be exposed without a very large input. The other issues we found were memory leaks and null dereferences from memory allocation.

This just emphasizes what we already knew about C, even the most careful, security conscious developer messes up memory management. Some of you are saying, so what? These are reference implementations and this is only Round 1. There are a few problems with that thought.

Reference implementations don't disappear, they serve as a starting point for future implementations or are used directly. A [bug in the RSA reference implementation](#) was responsible for vulnerabilities in OpenSSL and two separate SSH implementations. They can also be used to design hardware implementations, using buffer sizes to decide how much silicon should be used.

Re: I think that since only 5 of the 42 projects garnered your attention that a better quote to include in the summary would have been: We were

Re:Disclaimer

by FormOfActionBanana (966779) <slashdot2@douglasheld.net> on Sunday February 22 2009, @08:36PM (#26951075) Homepage

Yeah, both very good points.

People do make mistakes. Even geniuses, when they're trying really hard to be careful. Personally, I see recognizing that as a validation for code review (including the automated code review that I do).

I want the winning entry for this competition to be flawless to the extent that's feasible. Right now, my job includes finding SHA-1 for cryptographic key generation, and telling people to replace that with something better. I don't want to be pulling out SHA-3 in a couple years, too.

Comment: kilmo at Sat, 21 Feb 4:10 PM

Why?

These reference code is NOT the code that will be used in real life applications. It was written for a specific input, and no one should ever use it in any application...

A complete waste of time...

Comment: Dynamo at Sun, 22 Feb 12:58 PM

Kilmo is really wrong on this. Reference code lasts and lasts and lasts. In fact, people are reluctant to change it, precisely because they feel that it has been looked at carefully.

Please continue the good work.

9. Conclusions

- Humans still don't get it right.
 - Civilization is not ready to move on from C.
 - Pointing fingers is unlikely to help.

9. Conclusions

- Humans still don't get it right.
 - Civilization is not ready to move on from C.
 - Pointing fingers is unlikely to help.
- Set good examples for:
 - responsible disclosure
 - helping to fix problems
 - improving processes

Questions



Douglas Held

Fortify, an HP Company

douglas.held@hp.com