Playing Hide-and-Seek with Hash-DoS

Pascal Junod // HEIG-VD

Insomni'hack 2013, Geneva (Switzerland)

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

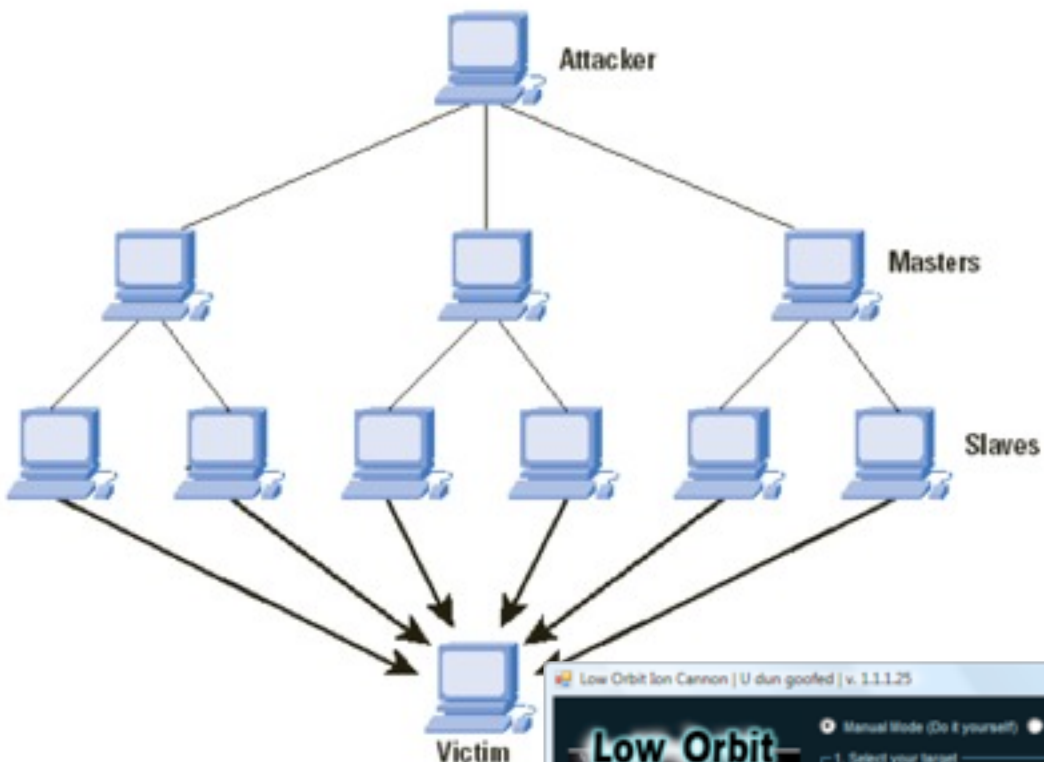
Overview

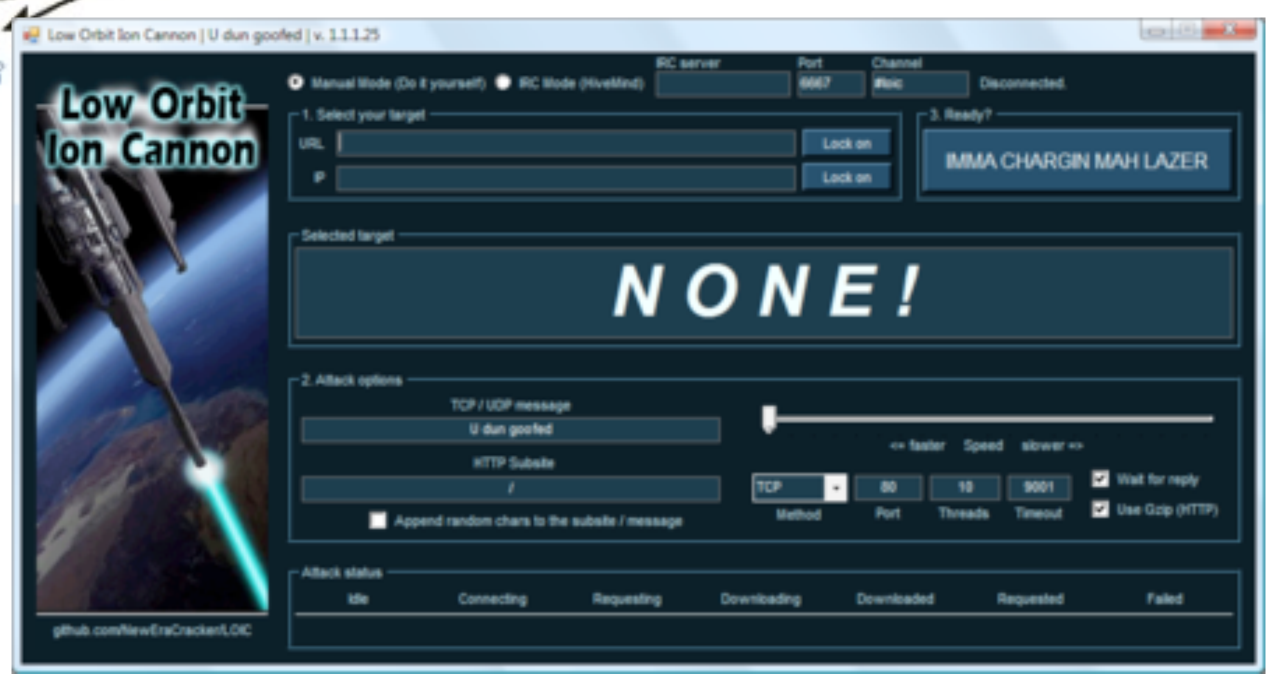
- DoS and Complexity attacks
- Hash DoS
- BTRFS
- Apache

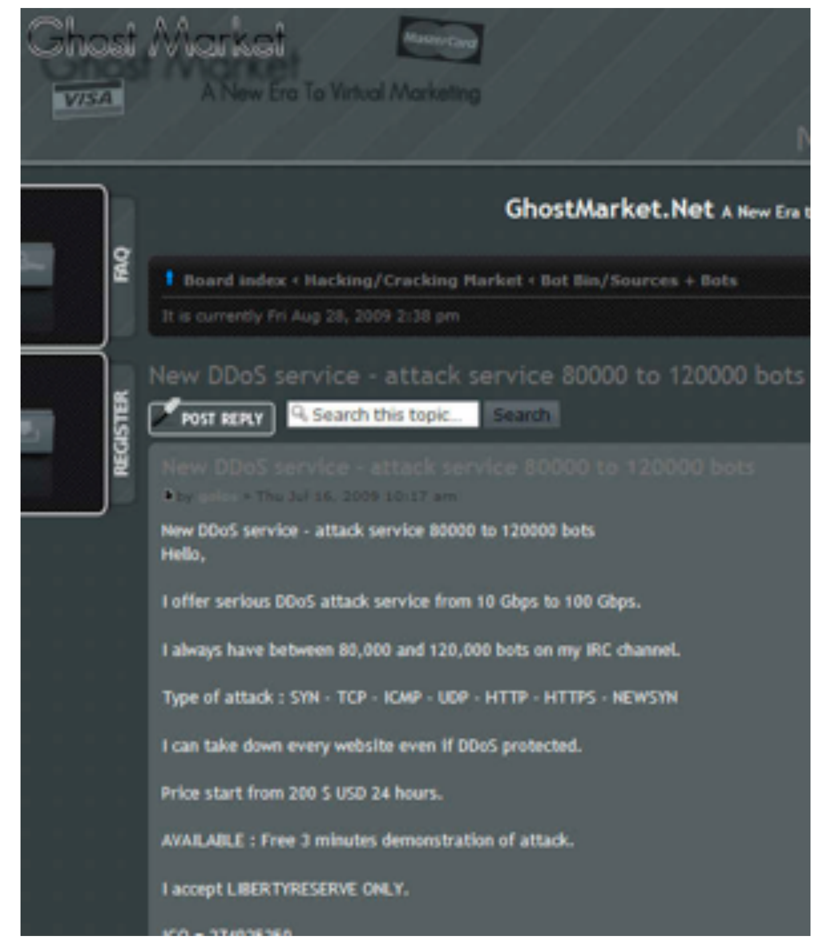
Denial of Service

- Goal is to make a resource unavailable to its intended users.
- Many different ways to do it:
  - (Bad programmers)
  - DDoS with TCP SYN or HTTP queries flood
  - Exploitation of amplification mechanisms
  - Complexity attacks
  - …

DDoS with TCP SYN or HTTP flood

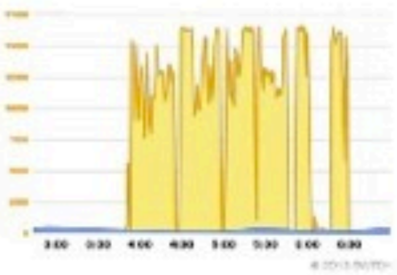






".ch" survives DDoS attack unscathed

January 10, 2013 / Roland Eugster

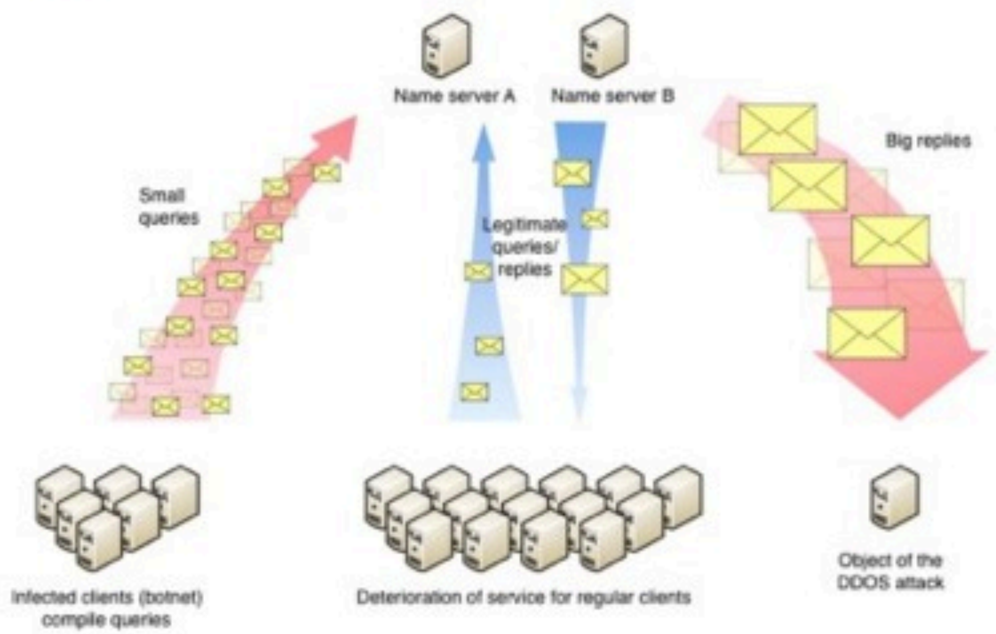


Since this morning (Thursday) all the Swiss name servers have been systematically abused in a bid to stop other websites from operating. "Distributed Denial of Service (DDoS)" attacks are nothing new, but this is the first time that the .ch infrastructure has been abused. Thanks to the high-quality operation of Switzerland's Internet by the SWITCH Foundation and SWITCH's rapid intervention, all .ch websites have remained accessible all the time.

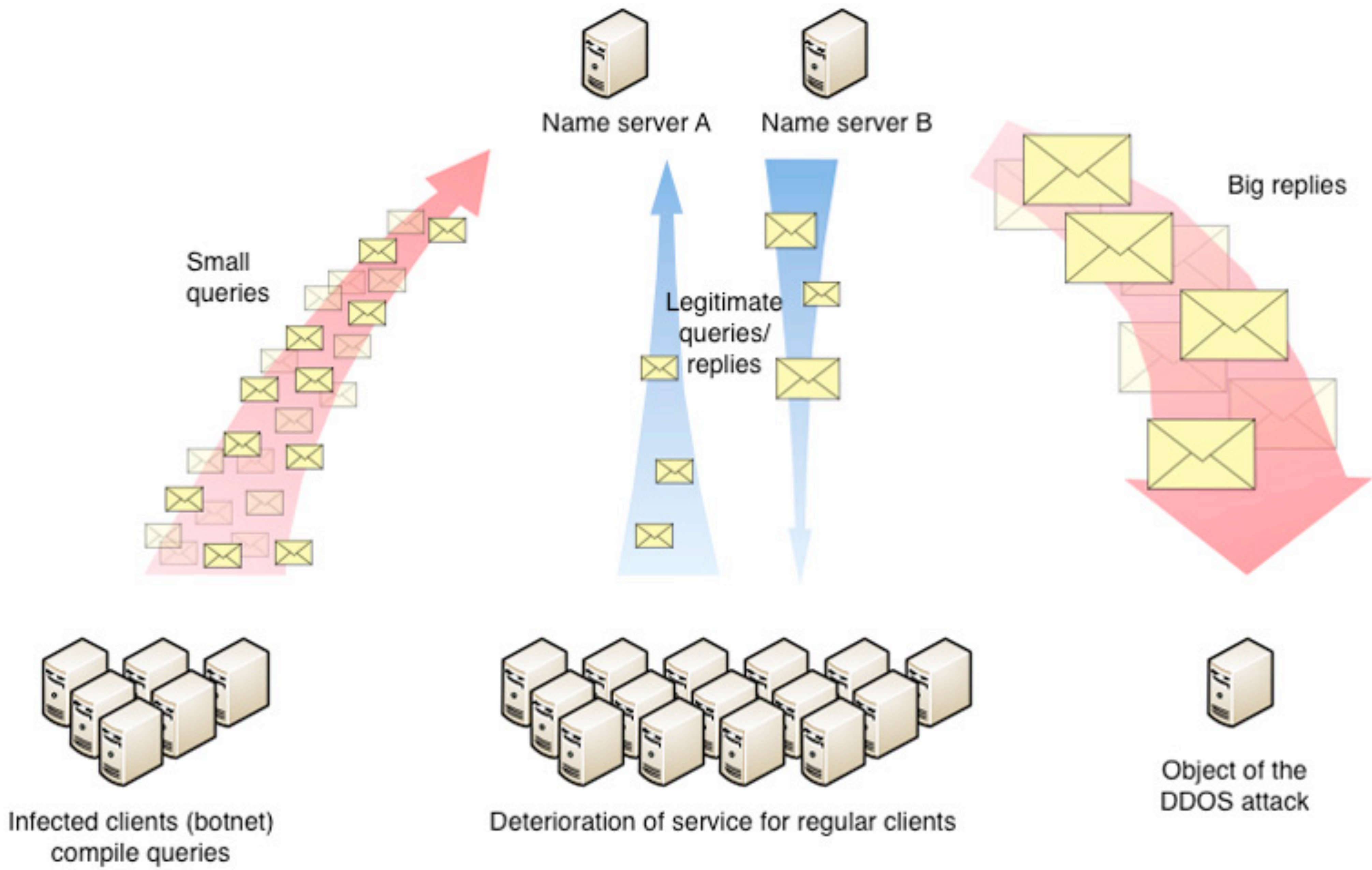
Since 04:00 this morning, all the ".ch" name servers have been attacked by meaningless queries with an intensity that is many times that of the normal network load. The .ch zone has not been the object of the attack but just the means to the end. In abusing the Swiss name servers, the attackers are attempting to prevent various websites in the USA from operating and are setting out to cause damage their operators.

Stable operation guaranteed

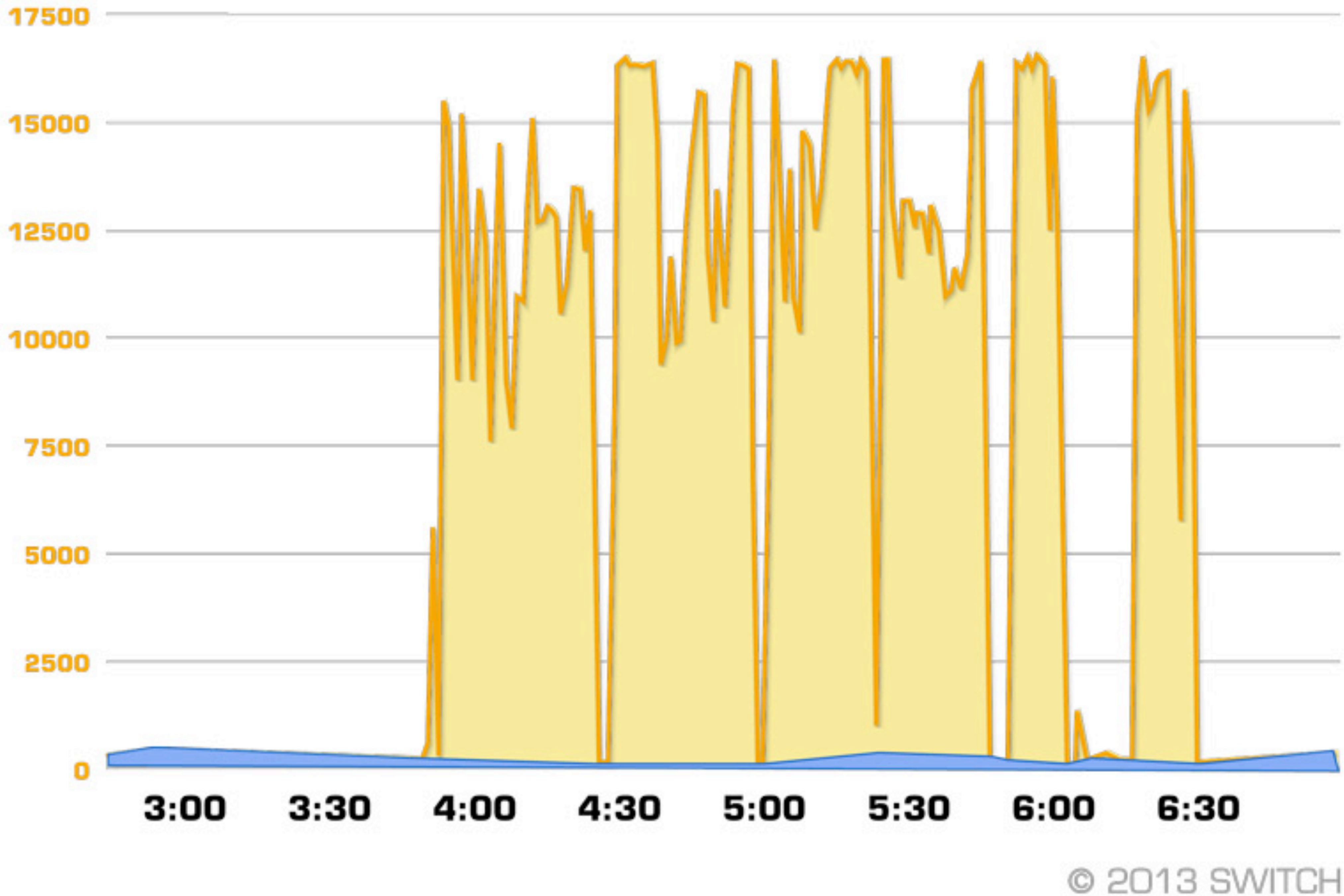
The attack – a standard "Distributed Denial of Service" attack – could have had far-reaching effects had there not been sufficient security precautions in place: if all the name servers are blocked, then no .ch websites can be accessed. Thanks to the rapid intervention of SWITCH's security team, it proved possible to defuse the situation. "We were prepared for such an emergency and were able to activate the necessary filters straight away and thus block the malicious traffic", explains Daniel Stirnimann who is responsible for the name server infrastructure. Since then, the load has been running at the normal level again, even though the attack is still ongoing.



Source: http://www.switch.ch/about/news/2013/ddos.html



Source: http://www.switch.ch/about/news/2013/ddos.html



Source: http://www.switch.ch/about/news/2013/ddos.html

Complexity Attacks

Complexity Attacks

Phrack 1998

```
---[  Phrack Magazine   Volume 8, Issue 53 July 8, 1998, article 13 of 15


-------------------------[  Designing and Attacking Port Scan Detection Tools


--------[  solar designer <solar@false.com>


----[  Introduction

The purpose of this article is to show potential problems with intrusion
detection systems (IDS), concentrating on one simple attack: port scans.

This lets me cover all components of such a simplified IDS.  Also, unlike
the great SNI paper (http://www.secnet.com/papers/IDS.PS), this article
is not limited to network-based tools.  In fact, the simple and hopefully
reliable example port scan detection tool ("scanlogd") that you'll find
at the end is host-based.
```

```
----[  Data Structures and Algorithm Choice

When choosing a sorting or data lookup algorithm to be used for a normal
application, people are usually optimizing the typical case.  However, for
IDS the worst case scenario should always be considered: an attacker can
supply our IDS with whatever data she likes.  If the IDS is fail-open, she
would then be able to bypass it, and if it's fail-close, she could cause
a DoS for the entire protected system.

Let me illustrate this by an example.  In scanlogd, I'm using a hash table
to lookup source addresses.  This works very well for the typical case as
long as the hash table is large enough (since the number of addresses we
keep is limited anyway).  The average lookup time is better than that of a
binary search.  However, an attacker can choose her addresses (most likely
spoofed) to cause hash collisions, effectively replacing the hash table
lookup with a linear search.  Depending on how many entries we keep, this
might make scanlogd not be able to pick new packets up in time.  This will
also always take more CPU time from other processes in a host-based IDS
like scanlogd.

I've solved this problem by limiting the number of hash collisions, and
```


```
Let me illustrate this by an example.  In scanlogd, I'm using a hash table
to lookup source addresses.  This works very well for the typical case as
long as the hash table is large enough (since the number of addresses we
keep is limited anyway).  The average lookup time is better than that of a
binary search.  However, an attacker can choose her addresses (most likely
spoofed) to cause hash collisions, effectively replacing the hash table
lookup with a linear search.  Depending on how many entries we keep, this
might make scanlogd not be able to pick new packets up in time.  This will
also always take more CPU time from other processes in a host-based IDS
like scanlogd.
```


```
more research might be needed.
```

Complexity Attacks

Usenix 2003

Denial of Service via Algorithmic Complexity Attacks

Scott A. Crosby
scrosby@cs.rice.edu

Dan S. Wallach
dwallach@cs.rice.edu

Department of Computer Science, Rice University

Abstract

We present a new class of low-bandwidth denial of service attacks that exploit algorithmic deficiencies in many common applications' data structures. Frequently used data structures have "average-case" expected running time that's far more efficient than the worst case. For example, both binary trees and hash tables can degenerate to linked lists with carefully chosen input. We show how an attacker can effectively compute such input, and we demonstrate attacks against the hash table implementations in two versions of Perl, the Squid web proxy, and the Bro intrusion detection system. Using bandwidth less than a typical dialup modem, we can bring a dedicated Bro server to its knees; after six minutes of carefully chosen packets, our Bro server was dropping as much as 71% of its traffic and consuming all of its CPU. We show how modern universal hashing techniques can yield performance comparable to commonplace hash functions while being provably secure against these attacks.

sume $O(n)$ time to insert n elements. However, if each element hashes to the same bucket, the hash table will also degenerate to a linked list, and it will take $O(n^2)$ time to insert n elements.

While balanced tree algorithms, such as red-black trees [11], AVL trees [1], and treaps [17] can avoid predictable input which causes worst-case behavior, and universal hash functions [5] can be used to make hash functions that are not predictable by an attacker, many common applications use simpler algorithms. If an attacker can control and predict the inputs being used by these algorithms, then the attacker may be able to induce the worst-case execution time, effectively causing a denial-of-service (DoS) attack.

Such algorithmic DoS attacks have much in common with other low-bandwidth DoS attacks, such as stack smashing [2] or the ping-of-death [1], wherein a relatively short message causes an Internet server to crash or misbehave. While a variety of techniques *can* be used to address these DoS attacks, common industrial practice still allows bugs like these

Complexity Attacks

CCC 2011

Efficient Denial of Service Attacks on Web Application Platforms

Alexander "**alech**" Klink
n.runs AG

Julian "**zeri**" Wälde
TU Darmstadt

#hashDoS

December 28th, 2011. 28th Chaos Communication Congress. Berlin, Germany.

php python Ruby

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Hes·so

Complexity Attacks

AppSec 2012

Hash-flooding DoS reloaded: attacks and defenses

Jean-Philippe Aumasson, Kudelski Group
Daniel J. Bernstein, University of Illinois at Chicago
Martin Boßlet, freelancer

Complexity Attacks

CCC 2012

Hash-flooding DoS reloaded: attacks and defenses

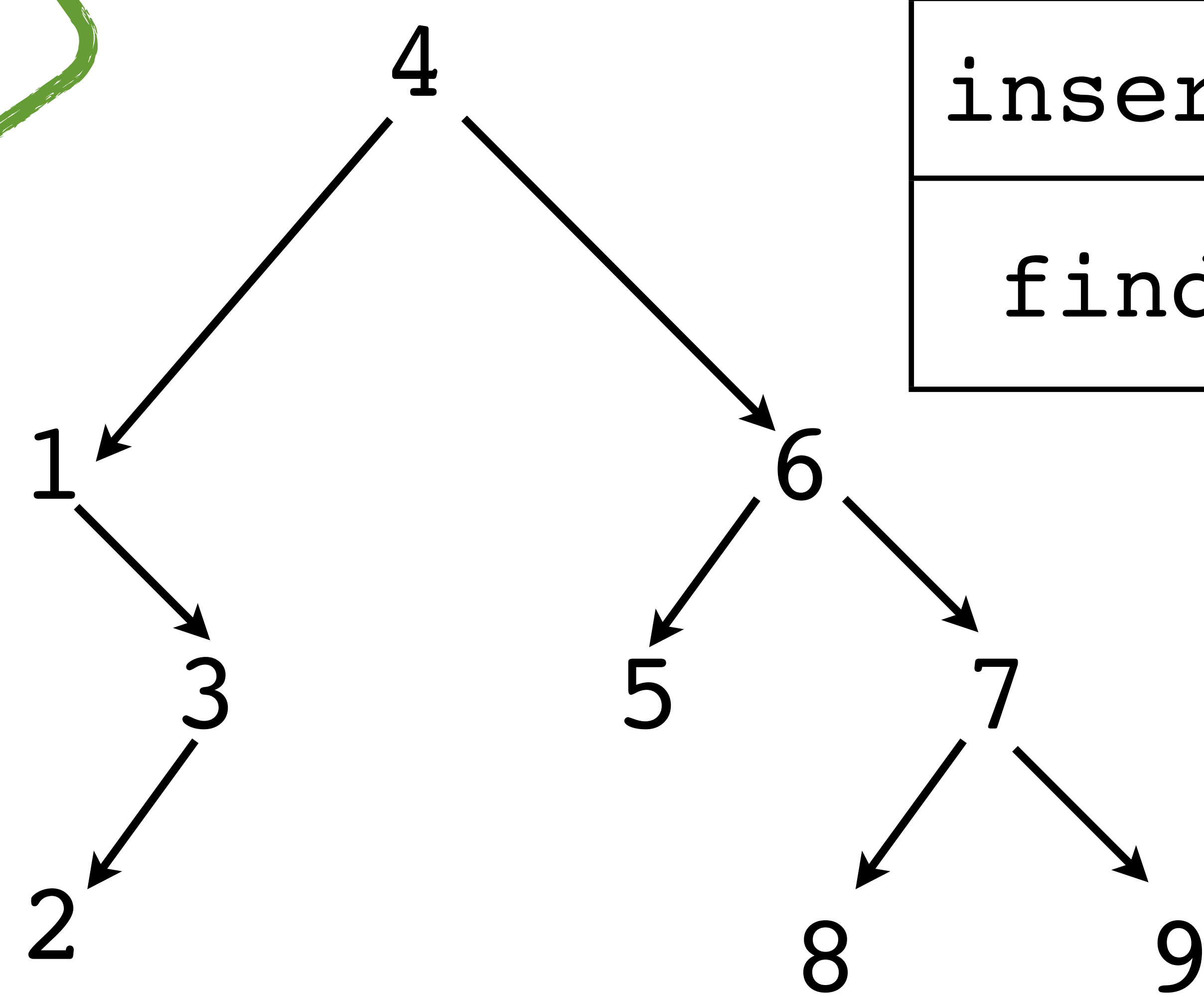
Jean-Philippe Aumasson, Kudelski Security (NAGRA)

D. J. Bernstein, University of Illinois at Chicago & Technische Universiteit Eindhoven

Martin Boßlet, Ruby Core Team

Example: Binary Tree

Average case



`insert()`	$O(\log n)$
`find()`	$O(\log n)$

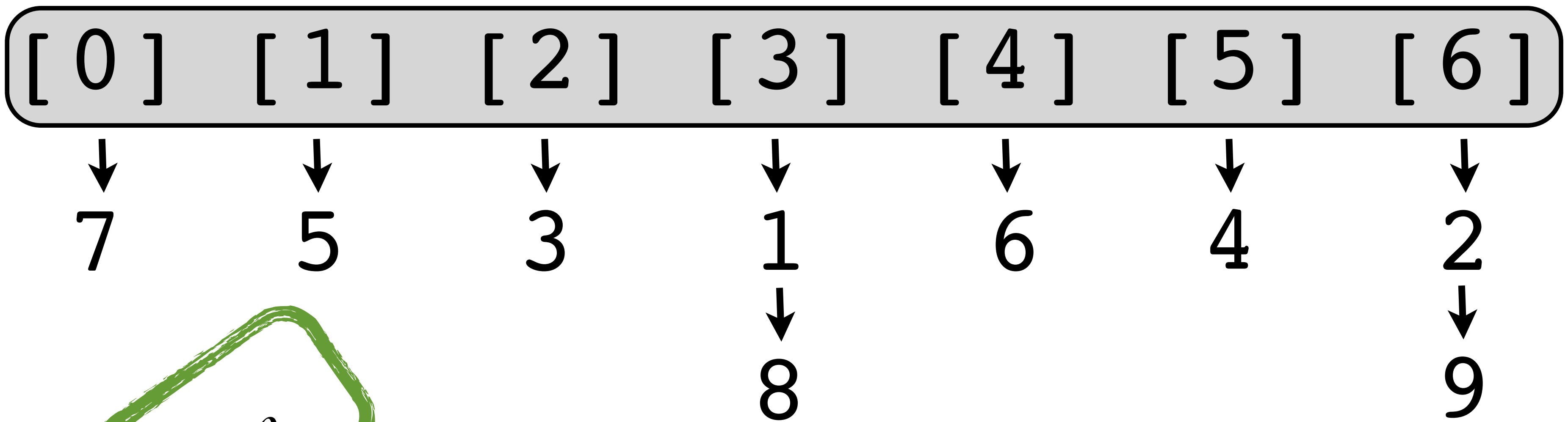
Example: Binary Tree

Worst case

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9

`insert()`	$O(n)$
`find()`	$O(n)$

Example: Hash Table




```
uint8_t hash (uint32_t e)
{
  return (e*3) % 7;
}
```

insert()	$O(1)$
find()	$O(1)$

heig-vd Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Hes·so Haute Ecole Spécialisée de Suisse occidentale

Example: Hash Table

[0] [1] [2] [3] [4] [5] [6]

[4] → 6 → 13 → 20 → 27 → 34 → 41 → 48 → 55

Worst case

```
uint8_t hash (uint32_t e)
{
  return (e*3) % 7;
}
```

insert()	O(n)
find()	O(n)

heig-vd Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Hes·so

Hash DoS

Multi-collisions

- Goal of a Hash-DoS: trigger the worst-case !
- In order to do it, one must find **multi-collisions** on the hash function, i.e., inputs x_1, x_2, x_3, ..., x_n such that $H(x_1) = H(x_2) = H(x_3) = \ldots H(x_n)$
- Most simple hash functions allow this easily.

Randomization

- A possible counter-measure consists in randomizing the hash function, keeping secret the random parameter.
- Not all hash functions and constructions are OK...
- For instance, look at $h := H(x)+r$, where r is the random value and where multi-collisions on $H(.)$ are easy to find.

Better solution

- Use a data structure that has good worst case complexities, like red-black trees.
- Performance loss vs. security loss

Red–black tree

From Wikipedia, the free encyclopedia

> This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(October 2012)*

A **red–black tree** is a type of self-balancing binary search tree, a data structure used in computer science.

The self-balancing is provided by painting each node with one of two colors (these are typically called 'red' and 'black', hence the name of the trees) in such a way that the resulting painted tree satisfies certain properties that don't allow it to become significantly unbalanced. When the tree is modified, the new tree is subsequently rearranged and repainted to restore the coloring properties. The properties are designed in such a way that this rearranging and recoloring can be performed efficiently.

The balancing of the tree is not perfect but it is good enough to allow it to guarantee searching in O(log *n*) time, where *n* is the total number of elements in the tree. The insertion, and deletion operations, along with the tree rearrangement and recoloring are also performed in O(log *n*) time.[1]

Tracking the color of each node requires only 1 bit of information per node because there are only two colors. The tree does not contain any other data specific to its being a red–black tree so its memory footprint is almost identical to classic (uncolored) binary search tree. In many cases the additional bit of information can be stored at no additional memory cost.

Red–black tree

Type	Tree	
Invented	1972	
Invented by	Rudolf Bayer	
Time complexity in big O notation		
	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

Red–black tree

Type	Tree	
Invented	1972	
Invented by	Rudolf Bayer	
Time complexity in big O notation		
	Average	Worst case
Space	O(n)	O(n)
Search	O(log n)	O(log n)
Insert	O(log n)	O(log n)
Delete	O(log n)	O(log n)

Example: nginx

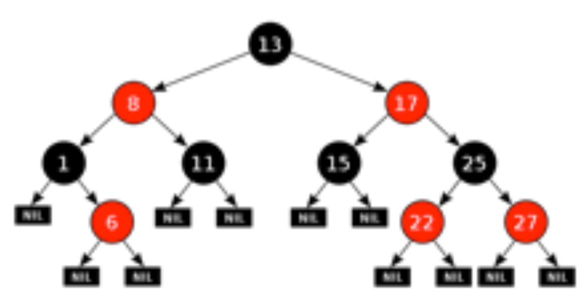
- HTTP server / reverse proxy
- Written by Igor Sysoev since 2002
- High performances, low CPU/memory footprint compared to competitors

Examples: nginx

- All data coming from configuration files are stored in hash tables
  - Very efficient, but not secure

```
#define ngx_hash(key, c)   ((ngx_uint_t) key * 31 + c)
```

- All data coming from outside are stored in red-black trees.
  - A bit less efficient, but mastered worst-case



heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Hes·so

BTRFS

BTRFS

- According to Wikipedia, the «B-tree file system» is a GPL-licensed experimental file-system for Linux.
- Several interesting features à la ZFS

BTRFS

Features

As of Linux 3.6 (released 30 September 2012), Btrfs implements:[22][23]

- Online defragmentation
- Online volume growth and shrinking
- Online block device addition and removal
- Online balancing (movement of objects between block devices to balance load)
- Offline filesystem check
- Online data scrubbing for finding errors and automatically fixing them for files with redundant copies
- RAID0, RAID1, and RAID10
- Subvolumes (one or more separately mountable filesystem roots within each physical partition)
- Transparent compression (zlib and LZO)
- Snapshots (read-only[24] or copy-on-write clones of subvolumes)
- File cloning (copy-on-write on individual files, or byte ranges thereof)
- Checksums on data and metadata (CRC-32C[25])
- In-place conversion (with rollback) from ext3/4 to Btrfs[26]
- File system seeding[27] (Btrfs on read-only storage used as a copy-on-write backing for a writeable Btrfs)
- Block discard support (reclaims space on some virtualized setups and improves wear leveling on SSDs with TRIM)
- Send/receive (saving diffs between snapshots to a binary stream)[28]
- Hierarchical per-subvolume quotas[29]

Planned features include:

- Online filesystem check[30]
- Very fast offline filesystem check[citation needed]
- Parity-based RAID (RAID5 and RAID6)[30]
- Object-level RAID0, RAID1, and RAID10[citation needed]
- Incremental dumps[citation needed]
- Ability to handle swap files and swap partitions[31]
- Data deduplication[4][32]
- Encryption[4][31]

Source: http://en.wikipedia.org/wiki/Btrfs#Features

BTRFS

SUSE Linux Says Btrfs is Ready to Rock

Friday, 07 December 2012 07:04 | Carla Schroder | Exclusive

J'aime 49 Tweet 97 +1 71 Share Submit

The advanced Butter/Better/B-tree Filesystem, Btrfs, is still labeled as experimental in the Btrfs Wiki and on Oracle's Btrfs page, though the Oracle page looks outdated. Btrfs is an advanced copy-on-write filesystem with a lot of great capabilities: snapshotting and rollbacks, checksumming of data and metadata, RAID, volumes and subvolumes, online defragmentation, compression, and online filesystem check and repair. Snapshots are always interesting to me; they're not backups, but a fast way to restore a system to a previous state. With Btrfs users can manage their own snapshots in their home directories. Btrfs supports filesystems up to 16 EiB in size, and files up to 16 EiB as well. (Which may be almost enough to store all the cute kitten photos on the Internet.)

Most distros include Btrfs, and Btrfs has been included in mainline Linux kernels since the 2.6.29 kernel. To use it just install the user-space tools. So what's the story, is it ready for prime time or not?

Btrfs is Ready

Matthias Eckermann, senior product manager at SUSE Enterprise Linux, says that Btrfs is ready for production systems, and as of SUSE Linux Enterprise 11 SP2 Btrfs is officially supported, along with Ext3, ReiserFS, XFS and OCFS2 (Oracle cluster filesystem for Linux).

The idea behind supporting multiple filesystems is to enable customers to choose different filesystems for different workloads. The installation default is good old tried-and-true Ext3. In the release notes SUSE recommends XFS for data, and Btrfs for root filesystems.

Source: http://www.linux.com/news/enterprise/systems-management/677226-suse-linux-says-btrfs-is-ready-to-rock

BTRFS Directory Entry

Directories

Directories are indexed in two different ways. For filename lookup, there is an index comprised of keys:

Directory Objectid	BTRFS_DIR_ITEM_KEY	64 bit filename hash

The default directory hash used is crc32c, although other hashes may be added later on. A flags field in the super block will indicate which hash is used for a given FS.

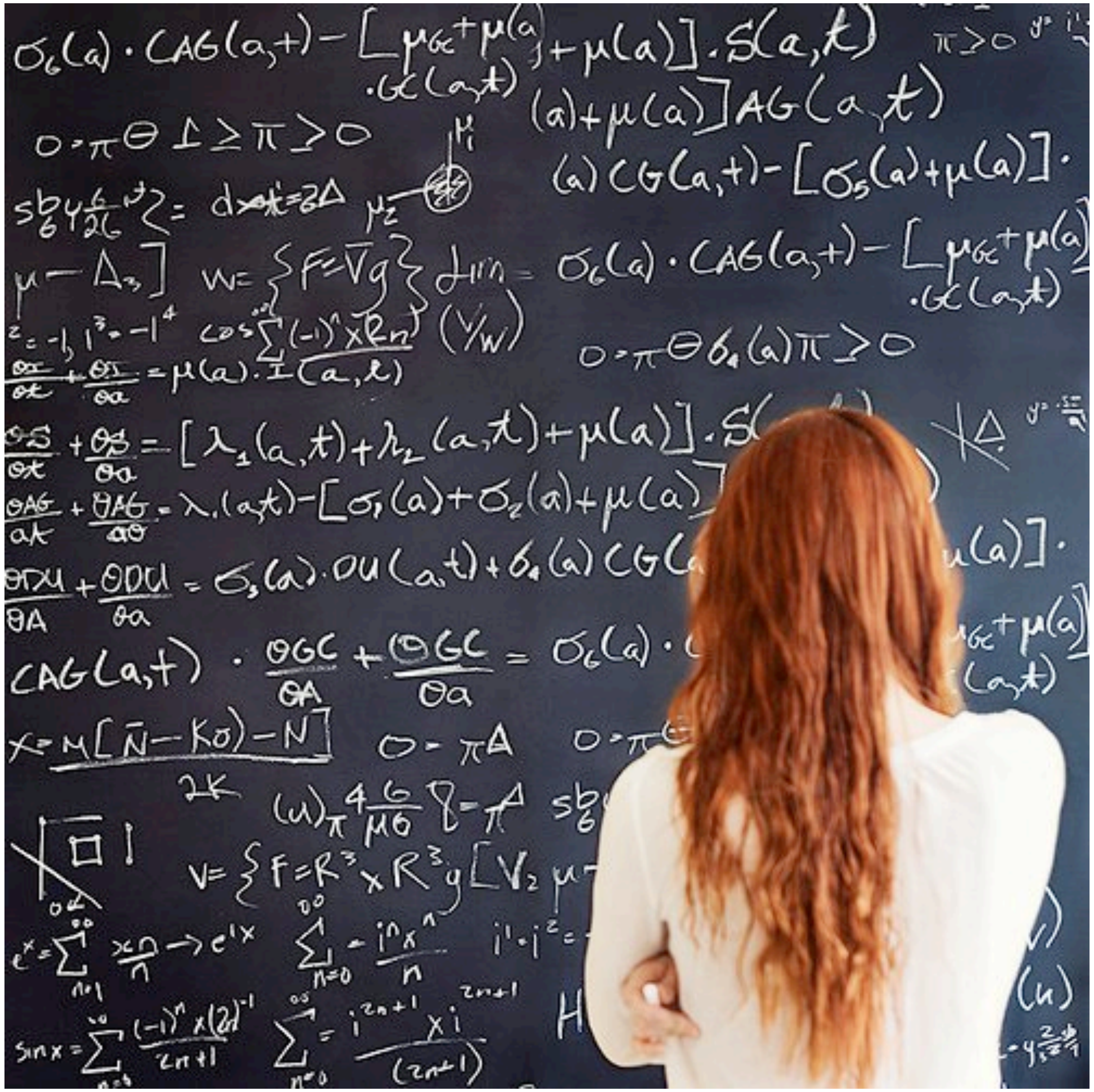
The second directory index is used by readdir to return data in inode number order. This more closely resembles the order of blocks on disk and generally provides better performance for reading data in bulk (backups, copies, etc). Also, it allows fast checking that a given inode is linked into a directory when verifying inode link counts. This index uses an additional set of keys:

Directory Objectid	BTRFS_DIR_INDEX_KEY	Inode Sequence number

The inode sequence number comes from the directory. It is increased each time a new file or directory is added.

Source: https://btrfs.wiki.kernel.org/index.php/Btrfs_design

<math-nerd-session>



CRC32c

- Cyclic Redundancy Check (CRC)
  - Linear error-detecting code
  - Non-cryptographic !
    - (Multi-)collisions are easy to find
    - Pre-images are easy to find

CRCs

- Easily said: CRC(x) is the remainder of the division of x interpreted as a polynomial over GF(2) by the generator polynomial.
- Example:
  - x = `0x77`, $G(x) = x^4+x+1$

CRCs

- `0x77 <=> 0b01110111`
- `0b01110111` <=> $x^6+x^5+x^4+x^2+x+1$

Remainder

$$x^6+x^5+x^4+x^2+x+1 = (x^4+x+1)(x^2+x+1)+x^3+x^2+x$$

Generator polynomial

Quotient, that we forget

- `CRC4(0x77) = 0xE`

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

Hes·so

CRCs & Multi-Collisions

$$x^6+x^5+x^4+x^2+x+1 = (x^4+x+1)(x^2+x+1)+x^3+x^2+x$$

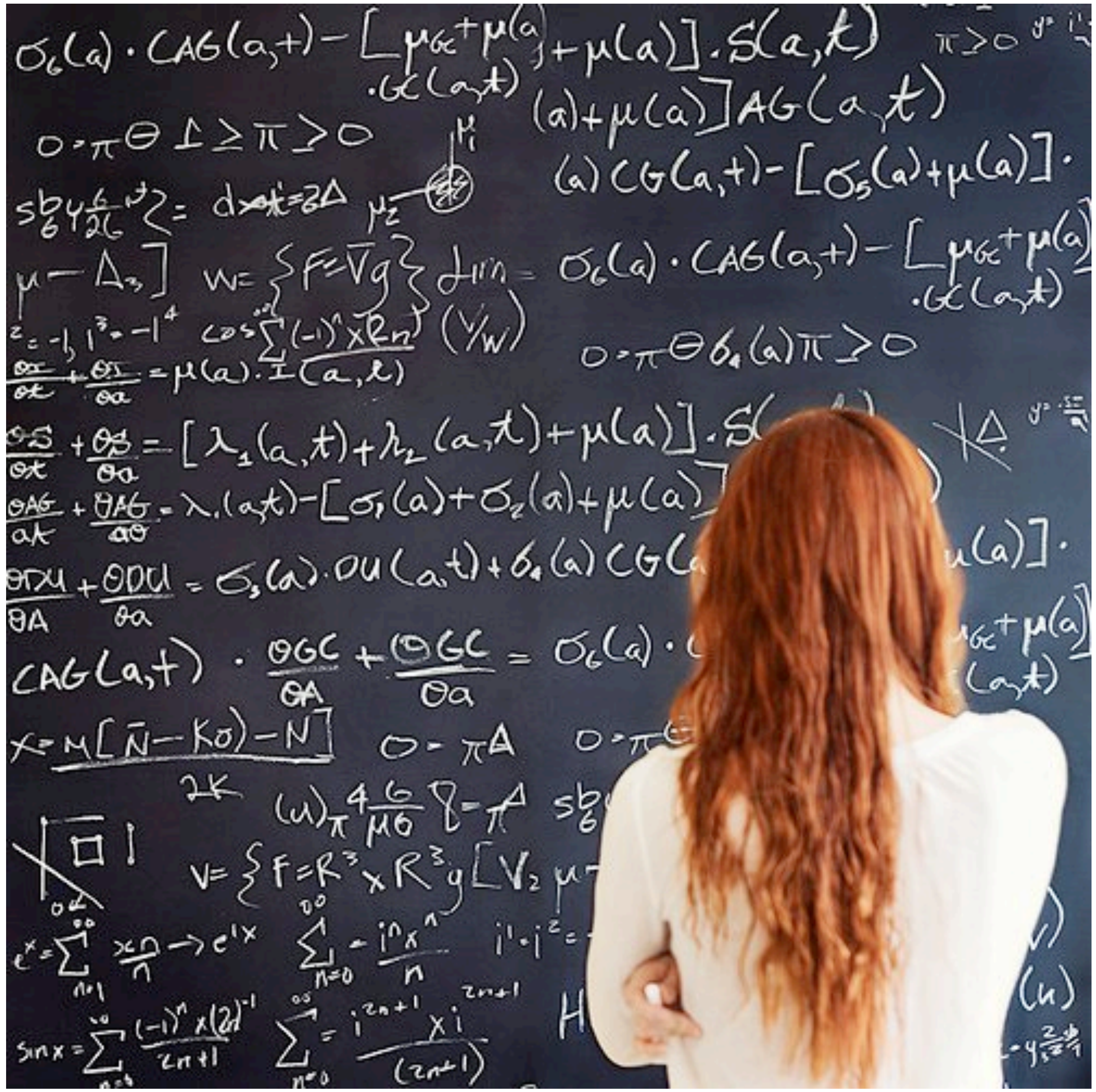
$$x^4+x^3+x^2+1 = (x^4+x+1)1+x^3+x^2+x$$

$$x^5+x^3 = (x^4+x+1)x+x^3+x^2+x$$

$$x^5+x^4+x^3+x+1= (x^4+x+1)(x+1)+x^3+x^2+x$$

...

- `CRC4(0x77) = CRC4(0x1D) = CRC4(0x48) = CRC4(0x3B) = 0xE`



</math-nerd-session>

Back to BTRS

- Idea: let's fill a directory with files whose names collide under CRC32c, and let's see what happens !
- **Demo time**

Results

- You can easily fill a bucket (dedicated CRC32c value), and BTRFS will allow you to create only a very limited number of files whose names collide under CRC32c
- You can also easily DoS the `bash` command line expansion mechanism ;-) !

Results

De Chris Mason
Sujet **Re: [btrfs] security hole disclosure**
Pour Pascal JUNOD
Copie à Chris Mason

Hi Pascal,

First, thanks for contacting me and for the time you've spent looking at Btrfs.

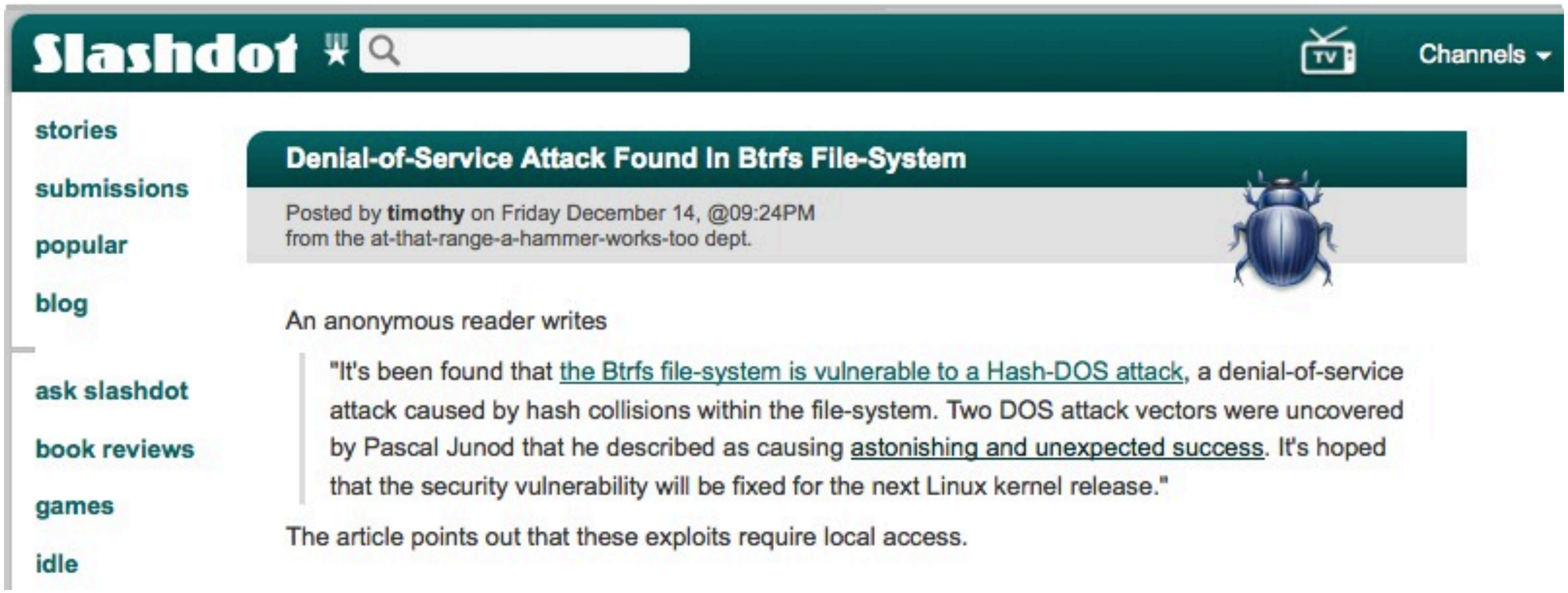
Collisions are a known issue with any of the hash based directories. Ext3's tea hash is a little more involved but it is still possible to do the same kinds of collision based DOS attacks. Other 32bit hashes do work better, but even though they are much more CPU intensive, it is still possible to DOS them.

64 bit hashes would be better, but the application interfaces (readdir,seekdir,telldir) on 32 bit machines don't work well with 64 bit directory offsets. Btrfs is able to get around this at least, we return something other than the hash from readdir.

We could make your exploit more complex by salting the hash, but I've always thought the 32 bit hashes were weak enough that salting alone wasn't a huge benefit. If you have any input there I'm very interested in hearing it.

The impact of the DOS is that a malicious user is able to prevent the creation of specific file names. For this to impact other users on the system, it must be done in directories where both the malicious user and the victim user have permission to create files.

Results



Slashdot

Channels

stories

submissions

popular

blog

ask slashdot

book reviews

games

idle

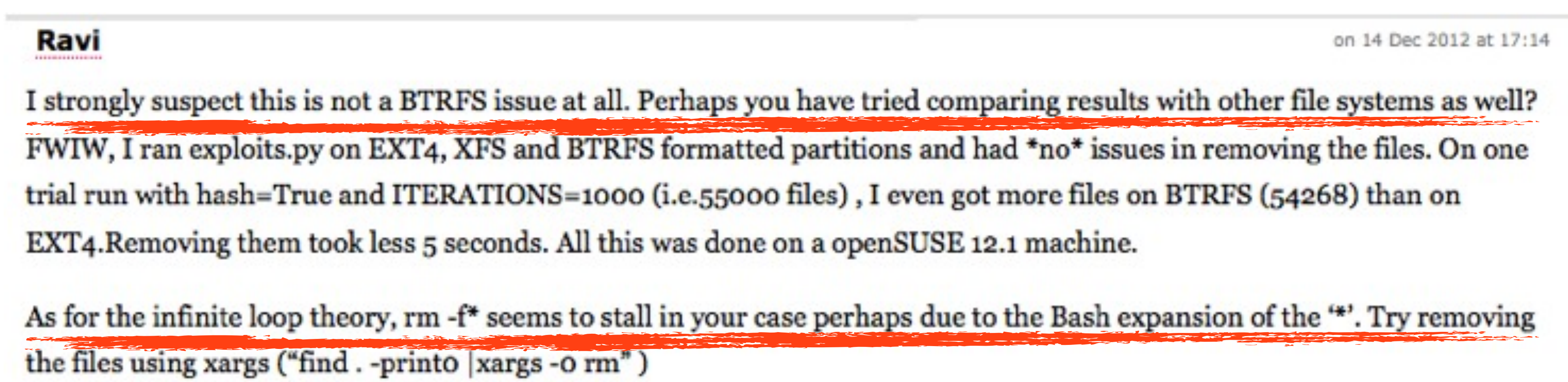
Denial-of-Service Attack Found In Btrfs File-System

Posted by **timothy** on Friday December 14, @09:24PM
from the at-that-range-a-hammer-works-too dept.

An anonymous reader writes

> "It's been found that the Btrfs file-system is vulnerable to a Hash-DOS attack, a denial-of-service attack caused by hash collisions within the file-system. Two DOS attack vectors were uncovered by Pascal Junod that he described as causing astonishing and unexpected success. It's hoped that the security vulnerability will be fixed for the next Linux kernel release."

The article points out that these exploits require local access.



Ravi on 14 Dec 2012 at 17:14

I strongly suspect this is not a BTRFS issue at all. Perhaps you have tried comparing results with other file systems as well? FWIW, I ran exploits.py on EXT4, XFS and BTRFS formatted partitions and had *no* issues in removing the files. On one trial run with hash=True and ITERATIONS=1000 (i.e.55000 files) , I even got more files on BTRFS (54268) than on EXT4.Removing them took less 5 seconds. All this was done on a openSUSE 12.1 machine.

As for the infinite loop theory, rm -f* seems to stall in your case perhaps due to the Bash expansion of the '*'. Try removing the files using xargs ("find . -printo |xargs -o rm")

Source: http://linux.slashdot.org/story/12/12/15/0055217/denial-of-service-attack-found-in-btrfs-file-system

Results

De Chris Mason
Sujet **Re: [btrfs] looping**
Pour Moi <pascal@junod.info>
Copie à Chris Mason

Hi Pascal,

You've found a real problem in btrfs though. Changes since I tested the crc overflow handling have made us not deal with our EOVERFLOW error handling completely right, and there are cases where we force the FS readonly when we shouldn't.

So this is a valid bug, I'm just waiting on some review of my fix, which will get backported to a number of kernels.

-chris

On Sat, Dec 15, 2012 at 04:49:02AM -0700, Pascal Junod wrote:

> Hi Chris,
>
> I have good news for you.
>
> Some people have noticed (in the comments of my blog post) that the CPU is burnt in userland, and not in kernel, a detail that I had unfortunately missed. Some have suggested that actually, btrfs was not looping, but that it was the expansion code of bash. As I generate random filenames, it is likely to have a * or a ? character, and then trigger a complexity attack on the bash command line expansion code, even before going into kernel code.
>
> stracing the rm command has not given anything as output, but I just tried to run my python code on another file-system (ext4), and it loops as well. Argh !
>
> Implementing another way to remove the files (with help of find and xargs feeding rm), I observed an average performance loss of about 10%, which is definitely acceptable. Ironically, the DoS seems to be prevented by the finite number of files mapping to the same key.
>
> Anyway, the filling-bucket "attack" remains still valid, it keeps an open question whether it must be accepted or not. I still tend to think that it is not, as a FS should be robust in every possible situation, including the ones that we don't think about.

Apache

The Apache Case

- Unfortunately, as for nginx, most of the hash tables used in Apache are fed with data coming from configuration files :-(
- Still, one can play a bit !

The Apache Case

pjunod@fedora:~/software/httpd-2.4.3/modules/aaa

File Edit View Search Terminal Help

```c
#define DFLT_NONCE_LIFE apr_time_from_sec(300)
#define NEXTNONCE_DELTA apr_time_from_sec(30)


#define NONCE_TIME_LEN  (((sizeof(apr_time_t)+2)/3)*4)
#define NONCE_HASH_LEN  (2*APR_SHA1_DIGESTSIZE)
#define NONCE_LEN       (int )(NONCE_TIME_LEN + NONCE_HASH_LEN)

#define SECRET_LEN      20


/* client list definitions */

typedef struct hash_entry {
    unsigned long      key;                       /* the key for this entry    */
    struct hash_entry *next;                      /* next entry in the bucket  */
    unsigned long      nonce_count;               /* for nonce-count checking  */
    char               ha1[2*APR_MD5_DIGESTSIZE+1]; /* for algorithm=MD5-sess    */
    char               last_nonce[NONCE_LEN+1]; /* for one-time nonce's      */
} client_entry;

static struct hash_table {
    client_entry  **table;
    unsigned long  tbl_len;
    unsigned long  num_entries;
    unsigned long  num_created;
    unsigned long  num_removed;
    unsigned long  num_renewed;
} *client_list;
```

MD5-sess

- So it seems that Apache uses a hash table (stored in a shared memory segment) to store data related to the `MD5-sess` digest authentication mechanism.
- What is `MD5-sess` ?! Only heard about MD5 digest authentication...

MD5-sess

Overview

[edit]

Digest access authentication was originally specified by RFC 2069 (*An Extension to HTTP: Digest Access Authentication*). RFC 2069 specifies roughly a traditional digest authentication scheme with security maintained by a server-generated nonce value. The authentication response is formed as follows (where HA1, HA2, A1, A2 are names of string variables):

$$\mathrm{HA1} = \mathrm{MD5}\Big(\mathrm{A1}\Big) = \mathrm{MD5}\Big(\mathrm{username} : \mathrm{realm} : \mathrm{password}\Big)$$

$$\mathrm{HA2} = \mathrm{MD5}\Big(\mathrm{A2}\Big) = \mathrm{MD5}\Big(\mathrm{method} : \mathrm{digestURI}\Big)$$

$$\mathrm{response} = \mathrm{MD5}\Big(\mathrm{HA1} : \mathrm{nonce} : \mathrm{HA2}\Big)$$

RFC 2069 was later replaced by RFC 2617 (*HTTP Authentication: Basic and Digest Access Authentication*). RFC 2617 introduced a number of optional security enhancements to digest authentication; "quality of protection" (qop), nonce counter incremented by client, and a client-generated random nonce. These enhancements are designed to protect against, for example, chosen-plaintext attack cryptanalysis.

If the algorithm directive's value is "MD5" or unspecified, then HA1 is

$$\mathrm{HA1} = \mathrm{MD5}\Big(\mathrm{A1}\Big) = \mathrm{MD5}\Big(\mathrm{username} : \mathrm{realm} : \mathrm{password}\Big)$$

If the algorithm directive's value is "MD5-sess", then HA1 is

$$\mathrm{HA1} = \mathrm{MD5}\Big(\mathrm{A1}\Big) = \mathrm{MD5}\Big(\mathrm{MD5}\Big(\mathrm{username} : \mathrm{realm} : \mathrm{password}\Big) : \mathrm{nonce} : \mathrm{cnonce}\Big)$$

If the qop directive's value is "auth" or is unspecified, then HA2 is

$$\mathrm{HA2} = \mathrm{MD5}\Big(\mathrm{A2}\Big) = \mathrm{MD5}\Big(\mathrm{method} : \mathrm{digestURI}\Big)$$

If the qop directive's value is "auth-int", then HA2 is

$$\mathrm{HA2} = \mathrm{MD5}\Big(\mathrm{A2}\Big) = \mathrm{MD5}\Big(\mathrm{method} : \mathrm{digestURI} : \mathrm{MD5}(entityBody)\Big)$$

If the qop directive's value is "auth" or "auth-int", then compute the response as follows:

$$\mathrm{response} = \mathrm{MD5}\Big(\mathrm{HA1} : \mathrm{nonce} : \mathrm{nonceCount} : \mathrm{clientNonce} : \mathrm{qop} : \mathrm{HA2}\Big)$$

If the qop directive is unspecified, then compute the response as follows:

$$\mathrm{response} = \mathrm{MD5}\Big(\mathrm{HA1} : \mathrm{nonce} : \mathrm{HA2}\Big)$$

The above shows that when qop is not specified, the simpler RFC 2069 standard is followed.

MD5-sess

```
If the "algorithm" directive's value is "MD5-sess", then A1 is
calculated only once - on the first request by the client following
receipt of a WWW-Authenticate challenge from the server. It uses the
server nonce from that challenge, and the first client nonce value to
construct A1 as follows:

      A1       = H( unq(username-value) ":" unq(realm-value)
                     ":" passwd )
                     ":" unq(nonce-value) ":" unq(cnonce-value)

This creates a 'session key' for the authentication of subsequent
requests and responses which is different for each "authentication
session", thus limiting the amount of material hashed with any one
key. (Note: see further discussion of the authentication session in
section 3.3.) Because the server need only use the hash of the user
credentials in order to create the A1 value, this construction could
be used in conjunction with a third party authentication service so
that the web server would not need the actual password value. The
specification of such a protocol is beyond the scope of this
specification.
```

Apache & MD5-sess

mod_auth_digest - Apach ×

httpd.apache.org/docs/2.4/ n/mod/mod_auth_digest.html

AuthDigestAlgorithm Directive

Description:	Selects the algorithm used to calculate the challenge and response hashes in digest authentication
Syntax:	`AuthDigestAlgorithm MD5|MD5-sess`
Default:	`AuthDigestAlgorithm MD5`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_auth_digest

The `AuthDigestAlgorithm` directive selects the algorithm used to calculate the challenge and response hashes.

`MD5-sess` is not correctly implemented yet.

mod_auth_digest - Apach ×

httpd.apache.org/docs/2.0/ n/mod/mod_auth_digest.html

AuthDigestAlgorithm Directive

Description:	Selects the algorithm used to calculate the challenge and response hases in digest authentication
Syntax:	`AuthDigestAlgorithm MD5|MD5-sess`
Default:	`AuthDigestAlgorithm MD5`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_digest

The `AuthDigestAlgorithm` directive selects the algorithm used to calculate the challenge and response hashes.

`MD5-sess` is not correctly implemented yet.

Apache & MD5-sess

```
[pjunod@fedora aaa]$ pwd
/home/pjunod/local/httpd/htdocs/aaa
[pjunod@fedora aaa]$ cat .htaccess
AuthType Digest
AuthDigestAlgorithm MD5-sess
AuthName "test"
AuthDigestProvider file
AuthUserFile /home/pjunod/local/httpd/htdocs/
aaa/.htpasswd
AuthDigestNonceLifetime 0
Require user pjunod
```

Apache & MD5-sess

```
[pjunod@fedora aaa]$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /aaa HTTP/1.0

HTTP/1.1 500 Internal Server Error
Date: Thu, 21 Mar 2013 14:25:58 GMT
Server: Apache/2.4.3 (Unix)
Content-Length: 528
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error or
misconfiguration and was unable to complete
your request.</p>
<p>Please contact the server administrator at
 you@example.com to inform them of the time this error occurred,
 and the actions you performed just before this error.</p>
<p>More information about this error may be available
in the server error log.</p>
</body></html>
Connection closed by foreign host.
```

Apache & MD5-sess

```
[pjunod@fedora aaa]$ tail -n1 ~/local/httpd/logs/error_log
[Thu Mar 21 15:25:59.858082 2013] [core:alert] [pid 23526:tid
139779710646016] [client 127.0.0.1:59616] /home/pjunod/local/httpd/
htdocs/aaa/.htaccess: AuthDigestAlgorithm: ERROR: algorithm `MD5-
sess' is not fully implemented
```

Apache & MD5-sess

A string of data specified by the server that contains a reference for the security context that is being established.

```
/*
 * Opaque and hash-table management
 */

/*
 * Generate a new client entry, add it to the list, and return the
 * entry. Returns NULL if failed.
 */
static client_entry *gen_client(const request_rec *r)
{
    unsigned long op;
    client_entry new_entry = { 0, NULL, 0, "", "" }, *entry;

    if (!opaque_cntr) {
        return NULL;
    }

    apr_global_mutex_lock(opaque_lock);
    op = (*opaque_cntr)++;
    apr_global_mutex_unlock(opaque_lock);

    if (!(entry = add_client(op, &new_entry, r->server))) {
        ap_log_rerror(APLOG_MARK, APLOG_ERR, 0, r, APLOGNO(01769)
                      "failed to allocate client entry - ignoring client");
        return NULL;
    }

    return entry;
}
```

Apache & MD5-sess

```
static void note_digest_auth_failure(request_rec *r,
                                     const digest_config_rec *conf,
                                     digest_header_rec *resp, int stale)
```

```
/* Setup opaque */

if (resp->opaque == NULL) {
    /* new client */
    if ((conf->check_nc || conf->nonce_lifetime == 0
         || !strcasecmp(conf->algorithm, "MD5-sess"))
        && (resp->client = gen_client(r)) != NULL) {
        opaque = ltox(r->pool, resp->client->key);
    }
    else {
        opaque = "";                /* opaque not needed */
    }
}
```

Apache & MD5-sess

- In summary:
  - Craft an HTTP query with
    - an `Authorization` field of type `Digest MD5`
    - no `opaque` directive
    - a `AuthDigestNonceLifetime 0` directive *server-side*
- and let's see what happens !

Apache & MD5-sess

```
[Fri Mar 22 19:16:25.832503 2013] [core:notice] [pid
23524:tid 139779852523264] AH00052: child pid 29275 exit
signal Floating point exception (8)
[Fri Mar 22 19:16:25.832554 2013] [core:notice] [pid
23524:tid 139779852523264] AH00052: child pid 29276 exit
signal Floating point exception (8)
[Fri Mar 22 19:16:25.832561 2013] [core:notice] [pid
23524:tid 139779852523264] AH00052: child pid 29277 exit
signal Floating point exception (8)
[Fri Mar 22 19:16:25.832568 2013] [core:notice] [pid
23524:tid 139779852523264] AH00052: child pid 29278 exit
signal Floating point exception (8)
```

Apache & MD5-sess

- In other words, any user can sabotage an Apache installation in a shared web environment
- Possibly, this can be transformed in a remote-only attack if one is able to upload an `.htaccess` file in a way or in another.

Apache & MD5-sess

De Mark J Cox
Sujet **Re: security hole disclosure**
Pour Pascal JUNOD
Copie à Apache Software Foundation HTTP Server Project

Répon

[resent, was missing cc]

I apologise for the delay in responding to you; we give priority to critical and important issues and so lower severity issues and bugs tend to keep getting pushed to the bottom of the pile. In general we would not treat an issue like this as a security threat; it requires a local malicious user; and if your local attacker can craft a .htaccess file then there are likely many other ways they could do so which could cause an Apache DoS. the next step would be to send this as a bug report (or to httpd-dev) for discussion.

Regards, Mark

On Wed, Dec 5, 2012 at 1:37 PM, Pascal Junod <pascal.junod@heig-vd.ch> wrote:

> Dear Apache security team,

He says it's a banana attack :-(


It's an *injustice! It is*!

Apache & MD5-sess

- Still, there is a morale attached to this story:
  - When you have unused code in your project, don't compile it !

RIP
Henri Kudelski
(1969-2013)

Contact

http://crypto.junod.info

pascal@junod.info

@cryptopathe