UKUUG

news@UK

The Newsletter of UKUUG, the UK's Unix and Open Systems Users Group
Published electronically at http://www.ukuug.org/newsletter/

Volume 18, Number 1 **ISSN 0965-9412** **March 2009**

Contents

News from the Secretariat

Jane Morrison

Thank you to everyone who has kindly sent in their subscription payments so promptly. We have received a number of early payments. Those remaining outstanding will be chased this month and any not paid at the end of April will not receive the next issue (June) Newsletter.

The two Perl tutorials (organised jointly by UKUUG and O'Reilly) which took place at the end of February were fully booked and highly successful. We are considering running them again towards the end of the year.

We now have everything in place for the UKUUG Spring Conference and Tutorials, being held in London (24th – 26th March). We have again received kind sponsorship from Google which allows us to offer the Conference Dinner at reduced rates for all Conference delegates. The event is also sponsored by Bytemark as well as by UKUUG Gold sponsor members Novell, IBM and SUN Microsystems.

Full information about the Srping Conference including programme details and an online booking form can be found at:
http://www.ukuug.org/events/spring2009/

Remember that by attending the Spring event you can:

- Keep abreast with new and emerging technologies
- Learn hands-on skills at the Kerberos training tutorial
- Network with some of the people who are responsible for developing critical applications such as OpenLDAP, Kerberos and openQRM
- Play a part in the UK Unix community – build up informal relationships that can be invaluable in problem solving
- Benefit from the experience of delegates with similar interests
- Keep staff happy and feeling valued
- Make valuable contacts with sponsor organisations such as Google, Bytemark, Novell and Sun
- Secure essential O'Reilly textbooks at a significant discount to the cover price

Plans are also afoot to organise a Summer conference (more details will be coming soon). Announcements of the forthcoming OpenTech event in July and the EuroBSDCon in September can be found elsewhere in this Newsletter. We are also investigating possible Autumn tutorials on DNS and RT.

The next Newsletter will be the June issue and the copy date is: Friday 22nd May.

For any comments about past or future events, or if you have something to say about the UKUUG or this Newsletter please contact:
newsletter@ukuug.org

Chairman's report

Paul Waring

Report on legal action with the British Standards Institute

Back in April, UKUUG sought a Judicial Review of BSI's decision to vote "yes" to the fast tracking of the DIS29500 (OOXML) porposed standard in ISO. The original application was rejected, but UKUUG appealed as we felt that the judge had not understood the arguments presented and wished to present our case at an oral hearing. Unfortunately, by the time this date came round it was clear that any decision by the court would not affect the fast-tracking of the standard, and permission was granted to discontinue the case. Some time later, a costs order was made against UKUUG to pay £18,500 to BSI – a fraction of

the total costs claimed, part of which was described by the judge as “grotesquely disproportionate”. As UKUUG paid the costs order in full by the deadline, the case is now closed.

Although UKUUG discontinued the case, we built up a good working relationship with other groups and obtained positive media coverage as a result of the case. Part of the costs of the case were also met by individual donors who contributed to the OOXML Fighting Fund and I would like to express our thanks on behalf of UKUUG to everyone who donated towards this, and to Phil Hands and Alain Williams who spent a lot of time dealing with the case.

Spring 2009

Our annual Spring conference is fast approaching – with a one day tutorial, seventeen talks, a pub social and a conference dinner (on HMS Belfast, no less) it’s shaping up to be another enjoyable event in the UKUUG calendar. I look forward to seeing many of you there.

Get involved

UKUUG exists to serve its members, so if you have any ideas for future events or would like to get involved with one of our working groups, please do let us know – our email address is
`council@ukuug.org`

Open Tech 2009 announcement

Open Tech 2009 is an informal, low cost, one-day conference on slightly different approaches to technology, democracy and community. Thanks to 4IP for sponsoring the event.

What do we need?

- Proposals from people who want to give a presentation, run a panel, organise a tutorial, or run a demo of something new and interesting on something that they think matters or getting people to help.
- Publicity – please blog this announcement, write a newspaper article, forward to mailing lists, and tell your friends!

What topics do we hope to cover?

- Mashups, open data and security
- Disaster politics and technology
- Future of media distribution
- Community engagement
- Democracy 2.0
- Highlights, lowlights and lessons learnt
- Long term thinking on big problems and massive opportunities
- Tutorials and Workshops – share what you know

If you’ve got an interesting proposal that doesn’t fit into any of the categories above, please send it in anyway!

What have we already got talks or sessions about?

- ID, surveillance and data-sharing
- mySociety
- International disaster management technologies

We're still looking for lots more talks on all our topics, so if you want to offer something, we're waiting to hear your ideas.

For further details of this event, including how to submit a proposal for a talk or presentation and how to register for the event, please see the event web site at
http://www.ukuug.org/events/opentech2009/

EuroBSDCon: September 18th – 20th, Cambridge

The Berkeley Software Distribution (BSD) family of computer operating systems is derived from software developed at the University of California at Berkeley. The various family members (DragonflyBSD, FreeBSD, NetBSD and OpenBSD, among others) are extensively used both for embedded appliances and for large internet servers and have an excellent reputation for stability and state-of-the-art technology. BSD-derived software is a driving force for IT research and development and is well-received as a building block in commercial software due to its unique license scheme.

The ninth European BSD conference is a great opportunity to present new ideas to the community and to meet some of the developers behind the different BSDs.

The two day conference program (September 19 – 20) will be complemented by a tutorial day preceding the conference (Sept 18).

Call for Papers

The Conference is inviting authors to submit innovative and original papers not submitted to other European conferences on the applications, architecture, implementation, performance and security of BSD-derived operating systems. Investigations on economic aspects regarding the operation of BSD systems are also welcome. Topics of interest for the EuroBSD Conference 2009 include, but are not limited to:

- embedded application development and deployment
- device drivers
- security and safe coding practices
- methods others should know about
- system administration: techniques and tools of the trade
- operational and economic aspects

Call for Tutorial Proposals

Selected tutorials on practical and problem-solving aspects of BSD-derived operating systems will be offered on the day before the Conference. The tutorials will be presented by speakers who have wide experience in developing and administering the different BSDs. Potential tutorial themes could include, but are not limited to:

- Safe coding practices to provide secure solutions
- System load testing and tuning
- BSD in a large network
- Solving sets of problems

For all other details, submission dates, and details of sponsorship opportunities, please see the conference website:
http://www.ukuug.org/events/eurobsdcon2009/

BCS Open Source events

We have received the following information on upcoming BCS OSSG events from Mark Elkins (OSSG Chair).

Ivan Ristić – Open Source Security – London 30th March 2009

Ivan Ristić will give a talk on Open Source Security for the Open Source Specialist Group (OSSG) on Monday 30th March 2009 at 6pm.

Ivan Ristić is an open source advocate, entrepreneur, writer, programmer and web security specialist. He is the principal author of ModSecurity, the open source web application firewall, and the author of Apache Security, a concise yet comprehensive web security guide for the Apache web server.
`http://www.ivanristic.com`

For further information please visit
`http://ossg.bcs.org/`

OpenStreetMap and CloudMade - Open Licensed Geo Data - London 27th April 2009

Cloudmade will give a talk about the OpenStreetMap project, how open licensed geo-data is being created in an open community, and how it can be used, on 27th April 2009 at 6pm.

Open Source CMS for Social Networking Workshop - London 30th April 2009

The Open Source Specialist Group (OSSG) will be holding a one day workshop on Open Source Content Management Systems (CMS) for Social Networking from 10:30am on Thursday 30th April 2009.

Papers and posters for presentation are being sought along with a call for those just wishing to attend and participate. It is envisaged that the workshop will compare some of the leading Open Source CMS such as Drupal, Joomla, and Plone for social networking purposes.

OSSG AGM 2009 - London 14th May 2009

The Annual General Meeting (AGM) of the Open Source Specialist Group (OSSG) will be held on 14th May 2009 at 6pm.

An event (to be confirmed) is due to commence immediately after the business of the AGM has been concluded.

These events will all take place at BCS Central London Offices, First Floor, The Davidson Building, 5 Southampton Street, London WC2E 7HA.

A Year of Birthdays

Peter H Salus

Every year is the anniversary of lots of things. 2009 is a year of great significance for technology.

Most of you know that it's the 200th anniversary of Darwin's birth; the 150th of the publication of *Origin of Species*. It's also the 250th anniversary of the founding of the British Museum and the Botanical Gardens at Kew. And the 500th of Henry VIII.

More recently, 1959 saw the birth of the IBM 1620, a transistorized version of the 650. It had drum memory and ran FORTRAN II. The 1620 cost $85,000 and up. Fifty years ago.

But 1969 was a blockbuster.

Mankind set foot on the moon. UNIX was invented. The ARPAnet came into being.

And, on 28 December, Linus Torvalds was born.

I'm planning to write about these things this year, but this essay will concern something that came earlier in 1969: RFC 1.

There may be a reader among you who is unfamiliar with the RFCs – the Requests for Comment, the standards of the ARPAnet, now the Internet. As I write this RFC 5469 has become available. Just under a dozen per month for 40 years.

But in 1969 there was nothing. Elmer Shapiro of SRI (the Stanford Research Institute) suggested that the Networking Working Group document its discussions. Most of those early discussions were about connecting hardware and such. When the graduate students from the West Coast met the "professionals" from BBN in Massachusetts, both groups were quite surprised.

The students expected a group of Ph.D.s who would roughshod over them. The BBN engineers expected the students to have readied everything for the moment the hardware was delivered.

Oh, hardware. Well, of the four sites designated, UCLA was running a Sigma 7 (an SDS [Scientific Data Systems] computer); SRI was running an SDS 940 (a machine designed for time-sharing, later renamed the XDS 940 when Xerox purchased SDS); the University of California, Santa Barbara had an IBM 360; and the University of Utah had a DEC PDP 10. Following a suggestion from Wesley Clark, each site would get a minicomputer, which it would connect to its main machine. The minis would talk to each other. They would be IMPs – Interface Message Processors. In many ways, they were the first routers.

The mini would be a Honeywell 516, which cost under $10,000. BBN chose the Model 516 because of its fast cycle time and powerful instruction set.

But let's go back to software: to RFC 1, 7 April 1969. Written by Stephen Crockett, one of the UCLA graduate students, it was called "Host Software."

The most interesting piece of RFC 1, today, may be the "Summary of IMP Software." It begins:

```
Information is transmitted from HOST to HOST in bundles called
messages.  A message is any stream of not more than 8080 bits,
together with its header.  The header is 16 bits and contains the
following information:

        Destination     5 bits
        Link            8 bits
        Trace           1 bit
        Spare           2 bits

The destination is the numerical code for the HOST to which the
message should be sent.  The trace bit signals the IMPs to record
status information about the message and send the information back to
the NMC (Network Measurement Center, i.e., UCLA).  The spare bits are
unused.
```

Think of it! 16 bits for the header. Five bits for addressing. A total of 32 possible addresses. By the summer, the IWG had decided on six-bit addressing. But then, who knew what might happen?

Looking back those four decades, it's really hard to fault anyone's vision. In 1971 – two years later – Alex McKenzie at BBN wrote to Nico Haberman at Carnegie-Mellon University, inviting CMU to become a host on the ARPAnet, envisaging a "user community of perhaps 2000" spread over fourteen sites. Today's (worldwide) user community is close to a million times that.

I admit to being full of nostalgia about those things. My first modem, in 1976, was a plush-lined receptacle for a telephone receiver. It connected my DECwriter II at 110 baud to Toronto's IBM 360. There were 63 hosts on the net. #43 was in Peter Kirstein's lab at UCL. True adventures. The next year, my lab was upgraded to a 300 baud connexion to the IBM 370.

But I'd like to offer a party cracker to all of those involved in where we are 40 years later. Davies and others at Rutherford; Kleinrock and students at UCLA; Frank Heart and his team at BBN; and many, many others.

The US Government put in just over $1 million in 1969. Thanks. That's a great rate of return!

* * *

Yes. The SCO Group bankruptcy drags on. Yes, I still hope to survive to reach the "end" of it all. But who knows? Think of Jarndyce and Jarndyce in *Bleak House*. Or "the Law's delay" in *Hamlet*. Sigh.

And now, Microsoft announces (according to *The Wall Street Journal*):

> *Microsoft Corp. said it hired a former Wal-Mart Stores Inc. executive to help the company open its own retail stores, a strategy shift that borrows from the playbook of rival Apple Inc. The Redmond, Wash., company said it hired David Porter, most recently the head of world-wide product distribution at DreamWorks Animation SKG, as corporate vice president of retail stores for Microsoft. In a statement, Microsoft said the first priority of Mr. Porter, who is also a 25-year veteran of Wal-Mart, will be to define where to place the Microsoft stores and when to open them. A Microsoft spokesman said the company's current plans are for a "small number" of stores. It remains to be seen whether the effort can add some pizzazz to Microsoft's unfashionable image, which Apple has sought to reinforce with ads that mock its competitor. Mr. Porter, in a statement, said there are "tremendous opportunities" for Microsoft to create a "world-class shopping experience" for the company's customers. "The purpose of opening these stores is to create deeper engagement with consumers and continue to learn firsthand about what they want and how they buy," Microsoft said in a statement.*

I find this hilarious and bizarre.

Of course, I don't expect either an M$ press release nor the *Journal* to reveal any knowledge of history. But this will be Microsoft's second charge into retail territory. I can recall the microsoftSF outlet in San Francisco's Metreon shopping centre, back in 1999. MircosoftSF had t-shirts and other clothing, desk accessories and "office supplies" – all with the Microsoft logo. The San Francisco shop closed in 2001. It also had "lifestyle areas" where Microsoft showed off its "technology in action." I can see it now. "Yes, ma'am. This little box contains all the bits your computer will ever need." But it shows how Ballmer's thoughts turn to sales and marketing rather than technology and development when things get rough. Go to the malls. Do you see all those Linux Shops? Look at the high streets! Every village in Cornwall with its Ubuntu store. Red Hat in the highlands!

Bah!

* * *

Hey! Wait! Will each Microsoft store have its own Mr. Clippy to help?

The Murky Issue of Changing Process Identity: revising "setuid demystified"

Dan Tsafrir, Dilma da Silva and David Wagner

Dropping unneeded process privileges promotes security but is notoriously error-prone because of confusing set*id system calls with unclear semantics and subtle portability issues. To make things worse, existing recipes to accomplish the task are lacking, related manuals can be misleading, and the associated kernel subsystem might contain bugs. We therefore proclaim the system as untrustworthy when it comes to the subject matter, and we suggest a defensive, easy-to-use solution that addresses all concerns.

Whenever you run a program, it assumes your identity and you lend it all your power: Whatever you're allowed to do, it too is allowed. This includes deleting your files, killing your other programs, changing your password, and retrieving your mail, for example. Occasionally, you need to write programs that enhance the power of others. Consider, for example, a Mahjongg game that maintains a high-score file. Of course, making the file writeable by all is not a very good idea if you want to ensure that no one cheats, so Mahjongg must somehow convey to players the ability to update the file in a controlled manner. In UNIX systems this is done as follows: When a game ends, if the score is high enough, Mahjongg

temporarily assumes the identity of the file's owner, makes the appropriate modifications, and switches back to the identity of the original player.

Many standard utilities work this way, including `passwd` and `chsh` (which update `/etc/passwd`), `xterm` (which updates `utmp` usage information), `su` (which changes user), `sudo` (which acts as root), and X (which accesses interactive devices). The common feature of these tools is that they know their real identity is of a nonprivileged user, but they have the ability to assume a privileged identity when required. (Note that "privileged" doesn't necessarily mean root; it merely means some other identity that has the power to do what the real user can't.) Such executables are collectively referred as "setuid programs," because (1) they must be explicitly associated with a "setuid bit" (through the `chmod` command) and (2) they pull off the identity juggling trick through the use of set*id system calls (`setuid(2)`, `setreuid(2)`, and all their friends).

There's another, often overlooked, type of program that can do identity juggling but does *not* have an associated setuid bit. These start off as root processes and use set*id system calls to change their identity to that of an ordinary nonprivileged user. Examples include the login program, the cron daemon (which runs user tasks at a specified time), daemons providing service to remote users by assuming their identity (sshd, telnetd, nfs, etc.), and various mail server components.

Both types of programs share a similar philosophy: To reduce the chances of their extra powers being abused, they attempt to obey the principle of least privilege, which states that "every program and every user of the system should operate using the least set of privileges necessary to complete the job" [16]. For setuid programs this translates to:

- minimizing the number and duration of the time periods at which the program temporarily assumes the privileged identity, to reduce the negative effect that programming mistakes might have (e.g., mistakenly removing a file as root can have far greater negative implications than doing it when the nonprivileged identity is in effect), and
- permanently giving up the ability to assume the privileged identity as soon as it's no longer needed, so that if an attacker gains control (e.g., through a buffer overflow vulnerability), the attacker can't exploit those privileges.

The principle of least privilege is a simple and sensible rule. But when it comes to identity-changing programs (in the immortal words of The Essex [7] or anybody who ever tried to lose weight [14]) it's easier said than done. Here are a few quotes that may explain why it's at least as hard as doing a diet: Chen et al said that "for historical reasons, the uid-setting system calls are poorly designed, insufficiently documented, and widely misunderstood" and that the associated manuals "are often incomplete or even wrong" [2]. Dean and Hu observed that "the setuid family of system calls is its own rat's nest; on different UNIX and UNIX-like systems, system calls of the same name and arguments can have different semantics, including the possibility of silent failures" [3]. Torek and Dik concluded that "many years after the inception of setuid programs, how to write them is still not well understood by the majority of people who write them" [17]. All these deficiencies have made the setuid mechanism the source of many security vulnerabilities.

It has been more than 30 years since Dennis Ritchie introduced the `setuid` mechanism [15] and more than 20 years since people started publishing papers about how to correctly write setuid programs [1]. The fact that this article has something new to say serves as an unfortunate testament that the topic is not yet resolved. Our goal in this paper is to provide the equivalent of a magical diet pill that effortlessly makes you slim (or at least lays the foundation for this magic). Specifically, we design and implement an intuitive change-identity algorithm that abstracts away the many pitfalls, confusing details, operating-system-specific behavior, and portability issues. We build on and extend the algorithm proposed by Chen et al.[2], which neglected to factor in the role that supplementary groups play in forming an identity. Our code is publicly available [18]. It was extensively tested on Linux 2.6.22, FreeBSD 7.0-STABLE, OpenSolaris, and AIX 5.3. We warn that, given the history of subtle pitfalls in the set*id syscalls, it may be prudent for developers to avoid relying upon our algorithm until it has been subject to careful review

by others.

User Identity vs.Process Identity

Before attempting to securely switch identities, we need to define what the term "identity" means. In this context, we found it productive to make a distinction between two types of identities: that of a user and that of a process. The user's credentials include the user ID (uid), the user's primary group (gid), and an additional array of supplementary groups (sups). Collectively, they determine which system resources the user can access. In particular, a zero uid is associated with the superuser (root) who can access all resources. We define the `ucred_t` type to represent a user by aggregating these three fields, as follows:

```
typedef struct supplementary_groups {
  gid_t *list; // sorted ascending, no duplicates
  int size;    // number of entries in 'list'
} sups_t;

typedef struct user_credentials {
  uid_t    uid;
  gid_t    gid;
  sups_t   sups;
} ucred_t;
```

Things are a bit more complicated when it comes to the corresponding process credentials. Each process has three user IDs: real (ruid), effective (euid), and saved (suid). The real uid identifies the "owner" of the process, which is typically the executable's invoker. The effective uid represents the identity in effect, namely, the one used by the OS (operating system) for most access decisions. The saved uid stores some previous user ID, so that it can be restored (copied to the euid) at some later time with the help of set*uid system calls. Similarly, a process has three group IDs: rgid, egid, and sgid. We define the `pcred_t` type to encapsulate the credentials of a process:

```
typedef struct user_ids  {uid_t r, e, s;} uids_t;
typedef struct group_ids {gid_t r, e, s;} gids_t;

typedef struct process_credentials {
  uids_t uids;  // uids.r = ruid, uids.e = euid, uids.s = suid
  gids_t gids;  // gids.r = rgid, gids.e = egid, gids.s = sgid
  sups_t sups;
} pcred_t;
```

Supplementary groups can be queried with the help of the `getgroups` system call. The ruid, euid, rgid, and egid of a process can be retrieved with `getuid`, `geteuid`, `getgid`, and `getegid`, respectively. The ways to find out the values of suid and sgid are OS-specific.

In Linux, each process also has an fsuid and an fsgid, which are used for access control to the file system. Normally, these are equal to the euid and egid, respectively, unless they are explicitly changed [11]. As this rarely used feature is Linux-specific, it is not included in the aforementioned data structures. To ensure correctness, our algorithm never manipulates the fsuid or fsgid, ensuring that (if programs rely only upon our interface for manipulating privileges) the fsuid and fsgid will always match the euid and egid.

The benefit of differentiating between user and process identities is that the former is more convenient to work with, easier to understand, better captures the perception of programmers regarding identity, and typically is all that is needed for programmers to specify what kind of an identity they require. In other words, the notions of real, effective, and saved IDs are not important in their own right; rather, they are simply the technical means by which identity change is made possible. Note, however, that "user" isn't an abstraction that is represented by any kernel primitive: The kernel doesn't deal with users; it deals with processes. It is therefore the job of our algorithm to internally use `pcred_t` and provide the appropriate mappings.

Rules of Identity Juggling

Identity propagation and split personalities

The second thing one has to consider when attempting to correctly switch identities is the manner by which processes initially get their identity. When a user *rik* logs in, the login program forks a process P

and sets things up such that (1) P's three uids hold *rik*'s uid, (2) P's three gids hold *rik*'s primary group, and (3) P's supplementary array is populated with the gids of the groups to which *rik* belongs. The process credentials are then inherited across `fork`. They are also inherited across `exec`, unless the corresponding executable E has its setuid bit set, in which case the effective and saved uids are set to be that of E's owner (but the real uid remains unchanged). Likewise, if E is setgid, then the saved and effective groups of the new process are assigned with E's group.

Conversely, the supplementary array is *always* inherited as is, even if E's setuid/setgid bits are set. Notice that this can lead to a bizarre situation where E is running with a split personality: the effective user and group are of E's owner, whereas the supplementary groups are of E's invoker. This isn't necessarily bad (and in fact constitutes the typical case), but it's important to understand that this is what goes on.

User id juggling

Since access control is based on the effective user ID, a process gains privilege by assigning a privileged user ID to its euid, and drops privilege by removing it. To drop privilege temporarily, a process removes the privileged user ID from its euid but stores it in its saved ID; later, the process may restore privilege by copying this value back to the euid. To drop privilege permanently, a process removes the privileged user ID from all three uids. Thereafter, the process can never restore privilege.

Roughly speaking, there typically exists some technical way for a process to copy the value from one of its three uids to another, and thus perform the uid juggling as was just described. If the process is nonroot (uid != 0), then / that's all it can do (juggle back and forth between the real and saved uids). Root, however, can assume any identity.

Primary group juggling

The rules of changing gids are identical, with the exception that egid=0 doesn't convey any special privileges: Only if euid=0 can the process set arbitrary gids.

Supplementary groups juggling

The rules for changing supplementary groups are much simpler: If a process has euid=0, it can change them however it likes through the `setgroups` system call. Otherwise, the process is forbidden from using `setgroups` and is stuck with the current setting. The implications for setuid programs are interesting. If the setuid program drops privileges (assuming the identity of its invoker), then the supplementary groups will already be set appropriately. However, until that happens, the program will have a split personality. A setuid-root program can set the supplementary groups to match its privileged identity, if it chooses. However, nonroot setuid programs cannot: They will suffer from a split personality for as long as they maintain their privileged identity, and there's simply no way around it. As a result, nonroot setuid programs might run with extra privileges that their creators did not anticipate.

Messiness of setuid system calls

Several standard set*id system calls allow programmers to manipulate the real, effective, and saved IDs, in various ways. To demonstrate their problematic semantics, we focus on only `setuid(2)` through an example of a vulnerability found in a mainstream program. Googling the word "setuid" with "vulnerability" or "bug" immediately brings up many examples that are suitable for this purpose. But to also demonstrate the prevalence of the problem, we attempted to find a new vulnerability. Indeed, the first program we examined contained one.

Exim is a popular mail server that is used by default in many systems [5]. Figure 1 shows the function exim uses to drop privileges permanently, taken from the latest version available at the time of this writing [6]. It implicitly assumes that calling setuid will update all three uids, so that all privileges are permanently relinquished. This assumption indeed holds for some OSes (e.g., FreeBSD). But if the effective ID is nonzero (which may be the case according to the associated documentation) then the assumption doesn't hold for Linux, Solaris, and AIX, as the semantics of setuid under these circumstances dictate that only the euid will be updated, leaving the ruid and suid unchanged. Consequently, if exim is compromised, the attacker can restore exim's special privileges and, for example, obtain uncontrolled access to all mail in the system.

Although this particular vulnerability isn't nearly as dangerous as some previously discovered setuid bugs, it does successfully highlight the problematic system call behavior, which differs not only between OSes but also according to the current identity.

Figure 1: *Exim's code to permanently change identity contains a vulnerability*

```
/*
 * This function sets a new uid and gid permanently, optionally calling
 * initgroups() to set auxiliary groups. There are some special cases when
 * running Exim in unprivileged modes. In these situations the effective
 * uid will not be root; [...]
 */
void exim_setugid(uid_t uid, gid_t gid, BOOL igflag, uschar *msg)
{
  uid_t euid = geteuid();
  gid_t egid = getegid();

if (euid == root_uid || euid != uid || egid != gid || igflag) {

if (igflag) {
    /* do some supplementary groups handling here */ ...
}
if (setgid(gid) < 0 || setuid(uid) < 0) {
    /* PANIC! */ ...
}
    }
  }
}
```

Safely Dropping Privileges

Equipped with a good understanding of the subject, we go on to develop an algorithm to safely drop privileges permanently. We do so in a top-down manner, making use of the `ucred_t` and `pcred_t` types previously defined. Figure 2 shows the algorithm. Its input parameter specifies the target identity; the algorithm guarantees to permanently switch to the target identity or clearly indicate failure. The algorithm works by first changing the supplementary groups, then changing the gids and changing the uids (in that order), and, finally, checking that the current identity matches the target identity.

Error handling

There are two ways to indicate failure, depending on how the macros `DO_CHK` and `DO_SYS` are defined:

```
#ifdef LIVING_ON_THE_EDGE
#define DO_SYS(call) if((call) == -1 ) return -1 /* do system call   */
#define DO_CHK(expr) if(!(expr)     ) return -1 /* do boolean check */
#else
#define DO_SYS(call) if((call) == -1 ) abort()   /* do system call   */
#define DO_CHK(expr) if(!(expr      ) abort()   /* do boolean check */
#endif
```

But although reporting failure through return values is possible, we advise against it, as it might leave the identity in an inconsistent state. Thus, when an identity change fails in the middle, programmers should either abort or really know what they're doing.

Input check

The `ucred_is_sane` function checks the validity of the input parameter. It is implemented as follows:

```
long nm = sysconf(_SC_NGROUPS_MAX);
return (nm >= 0) && (nm >= uc->sups.size) && (uc->sups.size >= 0) &&
   uc->uid != (uid_t)-1 &&
   uc->gid != (gid_t)-1;
```

The maximal size of the supplementary groups may differ between systems, but it can be queried in a standard way. We also check that the user and group IDs aren't -1, because this has special meaning for

several set*id system calls (“ignore”).

Verification

The first chunk of code in Figure 2 is responsible for setting the supplementary groups to `uc->sups`, the three gids to `g`, and the three uids to `u`. Setting the uids last is important, because afterward the process might lose its privilege to change its groups. Setting supplementary groups before primary groups is also important, for reasons to become clear later on. The remainder of the function verifies that all of these operations successfully changed our credentials to the desired identity. This policy is required in order to prevent mistakes in the face of the poorly designed set*id interface (e.g., this policy would have prevented the exim vulnerability), to protect against possible related kernel bugs [2] or noncompliant behaviour (see below) and to defend against possible future kernel changes. These reasons, combined with the fact that having the correct identity is crucial in terms of security, provide good motivation for our untrusting approach.

Figure 2: *Permanently switching identity and verifying the correctness of the switch*

```
int drop_privileges_permanently(const ucred_t *uc /*target identity*/)
{
   uid_t u = uc->uid;
   gid_t g = uc->gid;
   pcred_t pc;
   DO_CHK(ucred_is_sane(uc)               );
   DO_SYS(set_sups(&uc->sups )        );
   DO_SYS(set_gids(g/*real*/, g/*effective*/, g/*saved*/ ) );
   DO_SYS(set_uids(u/*real*/, u/*effective*/, u/*saved*/ ) );
   DO_SYS(get_pcred(&pc)             );
   DO_CHK(eql_sups (&pc.sups, &uc->sups)           );
   DO_CHK(g == pc.gids.r && g == pc.gids.e && g == pc.gids.s );
   DO_CHK(u == pc.uids.r && u == pc.uids.e && u == pc.uids.s );
   free( pc.sups.list );
#if defined(__linux__)
   DO_SYS( get_fs_ids( &u, &g)          );
   DO_CHK( u == uc->uid && g == uc->gid );
#endif
   return 0;   /* success */
}
```

Querying process identity

The `get_pcred` function we implement fills the memory pointed to by the `pcred_t` pointer it gets. We get the ruid, rgid, euid, and egid with the help of the standard system calls `getuid`, `getgid`, `geteuid`, and `getegid`, respectively. Unfortunately, there’s no standard way to retrieve saved IDs, so we use whatever facility the OS makes available, as shown in Figure 3. The `getresuid` and `getresgid` nonstandard system calls are the easiest to use and the most popular among OSes. AIX’s `getuidx` and `getgidx` also have easy semantics, whereas with Solaris the programmer must resort to using Solaris’s `/proc` interface[10].

The supplementary groups are retrieved with the help of the standard getgroups system call. To allow for easy comparison of supplementary arrays, we normalise the array by sorting it and by removing duplicate entries, if any exist. The array is malloc ed, and it should therefore be free d later on.

linux filesystem ids

In Linux, the fsuid is supposed to mirror the euid, as long as setfsuid isn’t explicitly used [11], and the same goes for fsgid and egid. However, there has been at least one kernel bug that violated this invariant[2]. Therefore, in accordance with our defensive approach, the algorithm in Figure 2 explicitly verifies that the fs-invariant indeed holds.

Figure 3: *Getting the saved uid and gid is an OS-dependent operation*

```
int get_saved_ids(uid_t *suid, gid_t *sgid)
{
#if defined(__linux__)    || defined(__HPUX__)     ||\
    defined(__FreeBSD__) || defined(__OpenBSD__) || defined(__DragonFly__)
```

```
    uid_t ruid, euid;
    gid_t rgid, egid;
    DO_SYS( getresuid(&ruid, &euid, suid) );
    DO_SYS( getresgid(&rgid, &egid, sgid) );
#elif defined(_AIX)
    DO_SYS( *suid = getuidx(ID_SAVED) );
    DO_SYS( *sgid = getgidx(ID_SAVED) );
#elif defined(__sun__) || defined(__sun)
    prcred_t p; /* prcred_t is defined by Solaris */
    int fd;
    DO_SYS( fd = open("/proc/self/cred", O_RDONLY));
    DO_CHK( read(fd, &p, sizeof(p)) == sizeof(p));
    DO_SYS( close(fd)                              );
    *suid = p.pr_suid;
    *sgid = p.pr_sgid;
#else
#error "need to implement, notably: __NetBSD__, __APPLE__, __CYGWIN__"
#endif
    return 0;
}
```

As there is no `getfsuid` or `getfsgid`, our implementation of `get_fs_ids` is the C equivalent of

```
  grep Uid /proc/self/status | awk '{print $5}' # prints fsuid
  grep Gid /proc/self/status | awk '{print $5}' # prints fsgid
```

Setting uids and gids

The POSIX-standard interfaces for setting IDs are tricky, OS-dependent, and offer no way to directly set the saved IDs. Consequently, nonstandard interfaces are preferable, if they offer superior semantics. This is the design principle underlying our implementation of `set_uids` and `set_gids`. The implementation is similar in spirit to the code in Figure 3, but it is complicated by the fact that nonprivileged processes are sometimes not allowed to use the preferable interface, in which case we fall back on whatever is available.

Specifically, all OSes that support `getresuid` (see Figure 3) also support `setresuid` and `setresgid`. These offer the clearest and most consistent semantics and can be used by privileged and nonprivileged processes alike. (Of course the usual restrictions for nonprivileged processes still apply, namely, each of the three parameters must be equal to one of the three IDs of the process.) In Solaris, only root can use the `/proc` interface for setting IDs[10], so with nonroot processes we naively use `seteuid` and `setreuid` (and their gid counterparts) and hope for the best: the verification part in Figure 2 will catch any discrepancies. In AIX, `setuidx` and `setgidx` are the clearest and most expressive, and they can be used by both root and nonroot processes[13]. However, AIX is very restrictive: a nonroot process can only change its effective IDs, so dropping privileges permanently is impossible for nonroot processes; also, root processes are allowed to set euid, euid/ruid, or euid/ruid/suid, but only to the same value.

Supplementary groups caveats

Recall that nonroot processes are not allowed to call setgroups. Therefore, to avoid unnecessary failure, `setgroups` is only invoked if the current and target supplementary sets are unequal, as shown in Figure 4. (Disregard the FreeBSD chunk of code for the moment.) Additionally, recall that after setting the supplementary groups in Figure 2, we verify that this succeeded by querying the current set of supplementary groups and checking that it matches the desired value. In both cases the current and target supplementary sets must be compared. But, unfortunately, this isn't as easy as one would expect.

Figure 4: *Setting supplementary groups, while trying to avoid failure of nonroot processes and accommodating noncompliant behaviour of FreeBSD*

```
  int set_sups(const sups_t *target_sups)
  {
     sups_t targetsups = *target_sups;
  #ifdef __FreeBSD__
     gid_t arr[ targetsups.size + 1 ];
     memcpy(arr+1, targetsups.list, targetsups.size * sizeof(gid_t) );
```

```
    targetsups.size        = targetsups.size + 1;
    targetsups.list        = arr;
    targetsups.list[0] = getegid();
#endif
    if( geteuid() == 0 ) { // allowed to setgroups, let's not take any chances
        DO_SYS( setgroups(targetsups.size, targetsups.list) );
    }
    else {
        sups_t cursups;
        DO_SYS( get_sups( &cursups) );
        if( ! eql_sups( &cursups, &targetsups) ) // this will probably fail... :(
            DO_SYS( setgroups(targetsups.size, targetsups.list) );
        free( cursups.list );
    }
    return 0;
}
```

The POSIX standard specifies that "it is implementation-defined whether getgroups also returns the effective group ID in the grouplist array" [9]. This seemingly harmless statement means that if the egid is in fact found in the list returned by getgroups, there's no way to tell whether this group is actually a member of the supplementary group list. In particular, there is no reliable, portable way to get the current list of supplementary groups. As a result, our code for comparing the current and target supplementary sets (see `eql_sups` in Figure 5, which is used in Figure 2 and Figure 4) assumes that they match even if the current supplementary set contains the egid and the target supplementary set doesn't. This isn't completely safe, but it's the best we can do, and it's certainly better than not comparing at all.

Figure 5: *When comparing the current supplementary array to the target array, we ignore the egid if it's included in the former*

```
bool eql_sups(const sups_t *cursups, const sups_t *targetsups)
{
  int        i, j, n = targetsups->size;
  int        diff = cursups->size - targetsups->size;
  gid_t egid = getegid();
  if( diff > 1 || diff < 0 ) return false;
  for(i=0, j=0; i < n; i++, j++)
      if( cursups->list[j] != targetsups->list[i] ) {
          if( cursups->list[j] == egid ) i--; // skipping j
          else return false;
      }
  // If reached here, we're sure i==targetsups->size. Now, either
  // j==cursups->size (skipped the egid or it wasn't there), or we didn't
  // get to the egid yet because it's the last entry in cursups
  return j == cursups->size ||
      (j+1 == cursups->size && cursups->list[j] == egid);
}
```

Kernel designers might be tempted to internally represent the egid as just another entry in the supplementary array, as this can somewhat simplify the checking of file permissions. Indeed, instead of separately comparing the file's group against (1) the egid of the process and (2) its supplementary array, only the latter check is required. The aforementioned POSIX rule that allows getgroups to also return the egid reflects this fact. But POSIX also explicitly states that "set[*]gid function[s] shall not affect the supplementary group list in any way" [12]. And, likewise, `setgroups` shouldn't affect the egid. So such a design decision, if made, must be implemented with care.

The FreeBSD kernel has taken this decision and designated the first entry of the supplementary array to the egid of the process. But the implementers weren't careful enough, or didn't care about POSIX semantics[4]. When trying to understand why the verification code in Figure 2 sometimes fails in FreeBSD, we realized that the kernel ignores the aforementioned POSIX rules and makes no attempt to mask the internal connection between egid and the supplementary array. Thus, when changing the array through setgroups, the egid becomes whatever happens to be the first entry of the array. Likewise, when setting the egid (e.g. through setegid), the first entry of the array changes accordingly, in clear violation of POSIX. The code in the beginning of Figure 4 accommodates this noncompliant behavior. Additionally, whenever

we need to set the egid, we always make sure to do it after setting the supplementary groups, not before (see Figure 2).

Temporarily dropping and restoring privileges

Our implementation also includes functions to temporarily drop privileges and to restore them. They are similar to Figure 2 in that they accept a "target identity" `ucred_t` argument, they treat supplementary groups identically, and they verify that the required change has indeed occurred. When dropping privileges temporarily, we change only the euid/egid if we can help it (namely, if the values before the change are present in the real or saved IDs, which means restoration of privileges will be possible). Otherwise we attempt to copy the current values to the saved IDs before making the change. (Unfortunately, this will fail on AIX for nonroot processes.) The algorithm that restores privileges performs operations in the reverse order: first restoring uids, and only then restoring groups; saved and real IDs are unaffected.

Caution!

Identity is typically shared among threads of the same application. Consequently, our code is not safe in the presence of any kind of multithreading: Concurrent threads should be suspended, or else they run the risk of executing with an inconsistent identity. Likewise, signals should be blocked or else the corresponding handlers might suffer from the same deficiency.

The algorithms described in this article do not take into account any capabilities system the OS might have (e.g., "POSIX capabilities" in Linux [8]). Capabilities systems, if used, should be handled separately.

Conclusion

Correctly changing identity is an elusive, OS-dependent, error-prone, and laborious task. We therefore feel that it is unreasonable and counterproductive to require every programmer to invent his or her own algorithm to do so, or to expect programmers to become experts on these pitfalls. We suggest that the interests of the community would be better served by a unified solution for managing process privileges, and we propose the approach outlined in this article as one possible basis for such a solution. Our code is publicly available [18]. We welcome suggestions, bug reports, and extensions.

References

[1] M.Bishop, "How to Write a Setuid Program," ;login 12(1) (Jan./Feb. 1987).

[2] H.Chen, D.Wagner, and D.Dean, "Setuid Demystified," in 11th USENIX Security Symp., pp.171-190 (Aug. 2002).

[3] D.Dean and A.J.Hu, "Fixing Races for Fun and Profit: How to Use Access(2)," in 13th USENIX Security Symp., pp.195-206 (Aug. 2004).

[4] R.Ermilov, R.Watson, and B.Evans, [CFR] ucred.cr_gid, thread from the FreeBSD-current mailing list (June 2001):
http://www.mail-archive.com/freebsd-current@freebsd.org/msg28642.html

[5] Exim Internet mailer:
http://www.exim.org/

[6] `Exim-4.69/src/exim.c`, source code of exim 4.69:
ftp://ftp.exim.org/pub/exim/exim4/exim-4.69.tar.gz

[7] W.Linton and L.Huff, "Easier Said Than Done," performed by The Essex (July 1963):
http://www.youtube.com/watch?v=tgJ1ssTJtnA

[8] `man capabilities(7)` Linux man page – overview of Linux capabilities:
http://linux.die.net/man/7/capabilities

[9] `man getgroups(2)` the Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004 edition:
http://www.opengroup.org/onlinepubs/000095399/functions/getgroups.html

[10] `proc(4)` Solaris 10 reference manual collection:
http://docs.sun.com/app/docs/doc/816-5174/proc-4?l=en&a=view

[11] `man setfsuid(2)` Linux man page:
http://linux.die.net/man/2/setfsuid

[12] `man setgid(2)` the Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004 edition:
http://www.opengroup.org/onlinepubs/000095399/functions/setgid.html

[13] `man setuidx` AIX Technical Reference: Base Operating System and Extensions, Volume 2:
http://publib.boulder.ibm.com/infocenter/systems/topic/com.ibm.aix.basetechref/doc/basetrf2/setuid.htm

[14] Nerd Gurl, "Why Can't I Ever Achieve My Goals?" Yahoo! Answers (Jan. 2008):
http://answers.yahoo.com/question/index?qid=20080101143342AAQ1jbO

[15] D.M.Ritchie, Protection of Data File Contents, Patent No. 4135240 (July 1973):
http://www.google.com/patents?vid=USPAT4135240

[16] J.H.Saltzer and M.D.Schroeder, "The Protection of Information in Computer Systems", Proc.of the IEEE 63(9), 1278-1308 (Sept. 1975).

[17] C.Torek and C.H.Dik, Setuid mess (Sept. 1995):
http://yarchive.net/comp/setuid_mess.html

[18] D.Tsafrir, D.Da Silva, and D.Wagner, "Change Process Identity":
http://www.research.ibm.com/change-process-identity
http://code.google.com/p/change-process-identity

Originally published in ***;login: The USENIX Magazine,*** *vol. 33, no. 3 (Berkeley, CA: USENIX Association, 2008). Reprinted by kind permission.*

Mac OS X For Unix Geeks

Ernest Rothman, Brian Jepson and Rich Rosen

O'Reilly Media
ISBN: 978-0-596-52062-5
426pp.
£ 21.99
Published: 15th September 2008

reviewed by Gavin Inglis

Success in publishing depends upon knowing your potential readers and meeting their needs. Seldom has a book illustrated this principle so clearly as *Mac OS X For UNIX Geeks.* Within the first page we learn how to change the behaviour of `bash` to conform to the POSIX 1003.1 standard. By this point the casual browser knows exactly whether they will purchase the book or not.

The target market is, of course, Linuxheads and old school console jockeys who are attracted to the white and silver gleam of the Apple Mac, but who just don't feel *comfortable* with a computer unless they can get beneath the gloss and mess about with the bits that the man in the shop would really rather they didn't.

For this kind of reader, the book is pitched just right. Part I alone contains enough material to fill the average technical book. It begins with the Terminal application: its differences from xterm, customising it, grouping it, or throwing it away entirely and choosing an alternative.

A useful piece of orientation follows in the chapter on files. From subtleties of resource forks to a grand tour of the file system, this explains enough to reassure the Linux user who feels adrift in a familiar but alien world. Nearby is a discussion of the Mac's much maligned implementation of X11, which comes installed as standard with Leopard.

For those not ready to leave behind the joys of Linux, Part I concludes with a round-up of options for squeezing other operating systems onto Mac hardware: virtualisation, dual boot and emulation. Even old faithful MS-DOS games get a look in.

Part II has a tighter focus and contains fifty pages of nitty-gritty on compiling software, headers, libraries and frameworks. The specifics here are likely to solve many build problems. Part III concerns package management. After an intro to Fink and MacPorts, the emphasis is on actually constructing packages and the options available for distributing your own software.

Finally Part IV considers using a Mac OS X machine as a server. The topics one would expect are all covered: file and printer sharing, database options, firewalls and mail handling. A very brief section on programming languages is tacked on the end.

The changes in this fourth edition largely concern Mac OS X 10.5 (Leopard). Where the system behaviour has changed, the difference is explained clearly in the body of the text. There remains the odd reference to earlier versions, but the emphasis is on Tiger and Leopard. It's worth upgrading from a previous edition if the little details mean a lot to you.

Between the wider subjects mentioned here, a lot of bitty little topics are covered in a variable amount of detail. Any Mac-positive UNIX geek who doesn't learn something of interest from *Mac OS X For UNIX Geeks* is probably writing books like this themselves. Or working for Apple.

Learning Rails

Simon St. Laurent and Edd Dumbill

O'Reilly Media
ISBN: 978-0-596-51877-6
442pp.
£ 24.99
Published: 28th November 2008

reviewed by Mike Smith

Hello. It's been a while since my last book review, so good to be back, and I hope you're all having a reasonable 2009 so far. For those who don't remember, I'm an infrastructure person (formerly a UNIX SysAdmin), who dabbles with programming from time to time . . . but I am certainly no developer. I've always wondered what's so good about Ruby, and Rails, but never got into it. I know (knew, but the time you read this, I hope) virtually nothing about either so I was interested to work out whether "Learning Rails" is good for the absolute beginner like myself; and whether I'd be able to pick up Ruby as well as Rails from a book like this.

In case there are others reading with similarly little knowledge of this area, I slight diversion:

1. Ruby is the programming language, originating in the 90's in Japan. It has concepts from many other languages, including SmallTalk (which I learnt myself at Uni but have completely forgotten!), Perl, C, Python etc. The first book published in English is now available for free, (on `ruby-doc.org`), but there's also an O'Reilly book, The Ruby Programming Language, co-authored by the man himself, Yukihiro Matsumoto. It looks to me like a bit of a modern classic in its new form (it was a Nutshell in a previous guise) and has been compared to K&R's The C Programming Language: i.e. the definitive text.

2. Rails is a framework, which to the layman like me is an environment and set of libraries which make it easier to develop applications that obey particular good-practice (such as the Model-View-Controller architecture used here and elsewhere). It was created by David Heinemeier Hansson in 2003, and version 2.2 was released in November 2008 (.. the buzz at the moment is all about 2.3rc1 and I suspect that by the time you read this 2.3 will be out).

Back to the book: The emphasis is on the practical and pragmatic; and it suits me down to the ground. We get started by looking at three options for running Rails: An online environment (Heroku), InstantRails (which is a packaged up Windows installation) and finally the command-line installation of individual RoR components (could be on Linux, Mac or Doze). From there we're straight on to creating our first Hello World web application.

I worked through the initial examples, and uncovered some careless mistakes. I've submitted these to the errata website, but having previously worked on an O'Reilly manuscript (the SSH book), there's no excuse for this – it should have been caught prior to publication simply by someone doing what I did.

Having learnt how Rails goes about rendering the presentation layer, using Views and Layouts, we then move over to Controllers and Models. This is where we find the actual Ruby code, and we build a second application to explore this area. This only takes a moment or two using Rails' scaffolding. (You see, talking like a pro already!)

Each chapter has a little quiz at the end. I'm not usually a fan of this (reminds me too much of text books and therefore exams), but in this case it does its job – it makes you stop and think, and acts as a refresher.

A few more chapters in, and we're adding validation code, handling file uploads and discussing RESTful apps. Then we're working with multiple database tables and adding relationships between them; Standard relational database stuff, but without the SQL!

Again with its pragmatic approach, and we're a little more than half way through the book by now, we look at debugging techniques and setting up test frameworks (unit testing, functional testing and integration.)

After that brief intermission we return to some interesting web-related coding topics: Sessions, Cookies and a foray into Ajax. I didn't work through these later examples from scratch, but downloaded them from the book's supporting website. I had at some point upgraded to the latest version of Rails and I couldn't then launch the examples (no errors, so I had to guess what was up . . . some clues were provoked by running rake, Rails' version of the make command). I worked it out, and simply reinstalled an earlier version of Rails by doing a "`gem install -v=2.1.0 rails`". That did the trick.

The book is rounded off with some brief appendices covering Ruby itself, relational databases, regular expressions and a quick reference of Helper Methods.

Just to mention, there's a Facebook page for the book too. Only the two authors and one other fan currently present. Oh dear. Though they have uploaded a demonstration video:
`http://www.facebook.com/pages/Learning-Rails/56672626496`

One final comment: I've been playing with Amazon's Cloud (EC2 and S3) of late, and Amazon's AWS tools are available as Ruby gems too, so may be worth experimenting with RoR there.

So overall this book did absolutely give me that quick introduction to what Rails is all about. I picked up a few Ruby factoids along the way, but I think O'Reilly's other book on this will be a worthwhile investment to get a full appreciation of this object-oriented language. I liked the style of Learning Rails too – none of that high-brow theory and philosophical architectural discussion. We just get on and do it. And with these basics under my belt, I feel confident I understand enough to take the next steps, just referring to online resources when I need to. That, I think, is a good testament to this particular book.

SQL in a Nutshell

Kevin Kline, Daniel Kline and Brand Hunt

O'Reilly Media
ISBN: 978-0-596-51884-4
591pp.
£ 31.99
Published: 2nd December 2008

reviewed by Raza Rizvi

At the outset I have to say that I cannot claim to have read this book from cover to cover. As the author(s) freely state, this is a reference work and not something designed to teach SQL or database design, other than a very brief explanation of the syntax. It is a very comprehensive reference work covering every command in the SQL2003 standard as implemented in the most popular web databases (MySQL, Postgres, Microsoft SQL, and Oracle). That is every command, with every nuance and every variation in those 4 database implementations as well as the standard.

So then this is not a book for anyone working with just a single database, nor for those starting out with database development where other O'Reilly texts can provide the background and context in which the SQL commands are best used. It is definitely a boon for web programmers, contractors, and for those people migrating from one platform to another. To have the definitions of the operators to hand, with inline code snippets, and sensible descriptions is great, but to have it for multiple implementations is a work of genius and obvious sweat.

The author(s) clearly mark important points (like permission persistence after a `REVOKE` in MySQL, or case sensitivity options for searches in the `SELECT WHERE` clause), and you can see the overall divergence from the standard in a summary table at the start of chapter 3 (which deals with the command reference and makes up some 400 pages of the book). Platform specific extensions are included in chapter 4.

If you have to work with more than one database implementation, you really should end up with a well-thumbed copy of this book on your desk if you take any pride in your work.

Web Security Testing Cookbook: Systematic Techniques to Find Problems Fast

Paco Hope and Ben Walther

O'Reilly Media
ISBN: 978-0-596-51483-9
312pp.
£ 30.99
Published: 28th October 2008

reviewed by Raza Rizvi

'Systematic Techniques to Find Problems Fast' they say right up top on the front cover. Indeed easily a good third of this book shows you how to automate a variety of attacks on your web site using testing tools. But this isn't a book about hacking for hackers, it is written for the programmers who are developing web services, to open their eyes to the sort of testing they should be including as part of their code release routines.

It isn't an advert for a multi-thousand dollar testing toolkit but a sensible description of a kitbag of public domain utilities that can be stitched together to form a credible arsenal of weapons exploiting the common mistakes, misunderstandings and oversights to which web code is susceptible in the 'release it now' deployment lifecycles a commercial environment has.

An elegantly simple description of the security testing and web applications in chapter 1 starts the book. The authors make it clear that although this is a 'cookbook', it focuses on the essence of how one performs security tests rather than providing the absolute minutiae of process – a focus on the menu rather than the ingredients if you like. The tools to be used are introduced in turn on chapter 2, following the convention used in the majority of the book of Problem, Solution, and Discussion.

The readership is educated how to understand the flow of information from the web server to the web browser – not just the source code that makes up the web page but also the information that is used by the browser to interpret the HTML or Javascript. Following on into chapter 4 with a description of the encoding of data, the authors take time to visually point out common misconceptions and provide snippets of information that might otherwise be overlooked – for example that a replay attack might allow someone to subvert account security without knowing the actual password if they use a provided hash of the password.

With the 'basic' background adequately covered, the second part of the book, from chapters 5 through to chapter 8, gets down to manual tweaking of the input one can provide to a website – from URL and XML manipulation to cookie modifications. It could take you a good week (of 'oohs and aahs') to apply all the techniques described but it is time well spent so that one understands the kind of things one can do

before you unleash some automation. This is covered from chapter 6 onwards and is obviously far more productive for anything other than the very smallest site. The approaches explained allow you to focus testing in individual parts of the web site and its applications using the tools from chapter 2, including the fiendishly useful cURL as an alternative to PERL.

Now we have both an understanding of the interaction of the browser and web server, and the means to automate the testing, we are ready for the final part of the book which leads off with another set of data input attacks (like using URLs out of the intended sequence or subverting password recovery procedures). The examples are all highly interesting reading in their own right and quite rightly put the fear of code abuse into your head. Pity then the AJAX programmer for the examples to this stage have largely used traditional HTML but AJAX (which sits atop Javascript) means local data manipulation in the client-side code has to be considered. The book rounds off by considering attacks that use more than one of the methods described in the book to ensure that data or information is undermined (e.g. guessing usernames and passwords or using Cross Site Scripting).

I expected the book to have a final chapter to preach to me about the need for security, but then I remembered that in chapter 1 I was told that this was a book of 'how to do things' rather than a 'why you need security'. By the end, frankly, it was clear why I needed this book, and I was duly grateful.

Desktop GIS: Mapping the Planet with Open Source Tools

Gary Sherman

Pragmatic Bookshelf
ISBN: 978-1-934356-06-7
368pp.
£ 21.99
Published: 28th October 2008

reviewed by Greg Matthews

This book has already been reviewed in the UKUUG Newsletter and I have been asked to give a second opinion. I should point out at this point that I know very little about GIS although it is used extensively by the scientists in my organisation.

The book can be used as a simple introduction to basic GIS techniques and applications and it was in this way that I approached it. The early chapters are devoted to introducing GIS to an amateur audience who might be interested in getting more use out of existing their GPS equipment for example. Some time is spent on data formats and projection and coordinate systems but it is to the author's credit that these dry topics are covered clearly and adequately without descending into impenetrable technical jargon.

The main thrust of this book is to encourage the use of open source tools for GIS work and I must admit that I was pleasantly surprised by the maturity of the available software. Gary Sherman is actually a senior developer for the Quantum GIS (QGIS) project which is quickly approaching its 1.0 release. There is scant mention of any commercial tools other than in passing when discussing data formats or similar. If you are used to working with ARC GIS or some other commercial software, you will not find a detailed comparison of features or equivalent working methods but it may still be worth getting this book for the in depth review of the various FOSS tools that are available.

There is a chapter on spatial databases but it is really only an introduction to what is a big topic. At the end of the book are no less than four appendices covering the various tools that are used to illustrate the examples throughout the book.

On the whole the book strikes a neat balance between a tutorial on GIS with its possible applications for an interested non-professional, and a technical 'howto' detailing the use of specific open source tools to accomplish common tasks. Highly recommended to anyone with an interest in mapping.

Contributors

Gavin Inglis works in Technical Infrastructure for the EDINA National Data Centre. His interests include punk music, Egyptian mythology and complicated diagrams.

Greg Matthews is a senior systems administrator and manages the Unix and Linux support team for the Natural Environment Research Council.

Jane Morrison is Company Secretary and Administrator for UKUUG, and manages the UKUUG office at the Manor House in Buntingford. She has been involved with UKUUG administration since 1987. In addition to UKUUG, Jane is Company Secretary for a trade association (Fibreoptic Industry Association) that she also runs from the Manor House office.

Raza Rizvi normally worries about network based security.

Peter H Salus has been (inter alia) the Executive Director of the USENIX Association and Vice President of the Free Software Foundation. He is the author of *A Quarter Century of Unix* (1994) and other books.

Dilma da Silva is a researcher at the IBM T.J. Watson Research Center in New York. She manages the Advanced Operating Systems group. Prior to joining IBM, she was an Assistant Professor at University of Sao Paulo, Brazil. Her research in operating systems addresses the need for scalable and customizable system software.

Mike Smith works in the Chief Technology Office of a major European listed outsourcing company, setting technical strategy and working with hardware and software vendors to bring innovative solutions to its clients. He has over 15 years in the industry, including mid-range technical support roles and has experience with AIX, Dynix/ptx, HP-UX, Irix, Reliant UNIX, Solaris and of course Linux.

Dan Tsafrir is a postdoctoral researcher at the IBM T.J. Watson Research Center in New York. He is a member of the advanced operating systems group and is interested in various aspects of operating systems.

David Wagner is a professor in the computer science division at the University of California at Berkeley. He studies computer security, cryptography, and electronic voting.

Paul Waring is chairman of UKUUG and currently trying to complete the “final” draft of his MPhil thesis whilst working for an insurance intermediary. He is also responsible for organising the UKUUG Spring conference in 2009.

Contacts

Paul Waring
UKUUG Chairman
Manchester

John M Collins
Council member
Welwyn Garden City

Phil Hands
Council member
London

Holger Kraus
Council member
Leicester

Niall Mansfield
Council member
Cambridge

John Pinner
Council member
Sutton Coldfield

Howard Thomson
Treasurer; Council member
Ashford, Middlesex

Jane Morrison
UKUUG Secretariat
PO Box 37
Buntingford
Herts
SG9 9UQ
Tel: 01763 273475
Fax: 01763 273255
office@ukuug.org

Sunil Das
UKUUG Liaison Officer
Suffolk

Roger Whittaker
Newsletter Editor
London

Alain Williams
UKUUG System Administrator
Watford

Sam Smith
Events and Website
Manchester