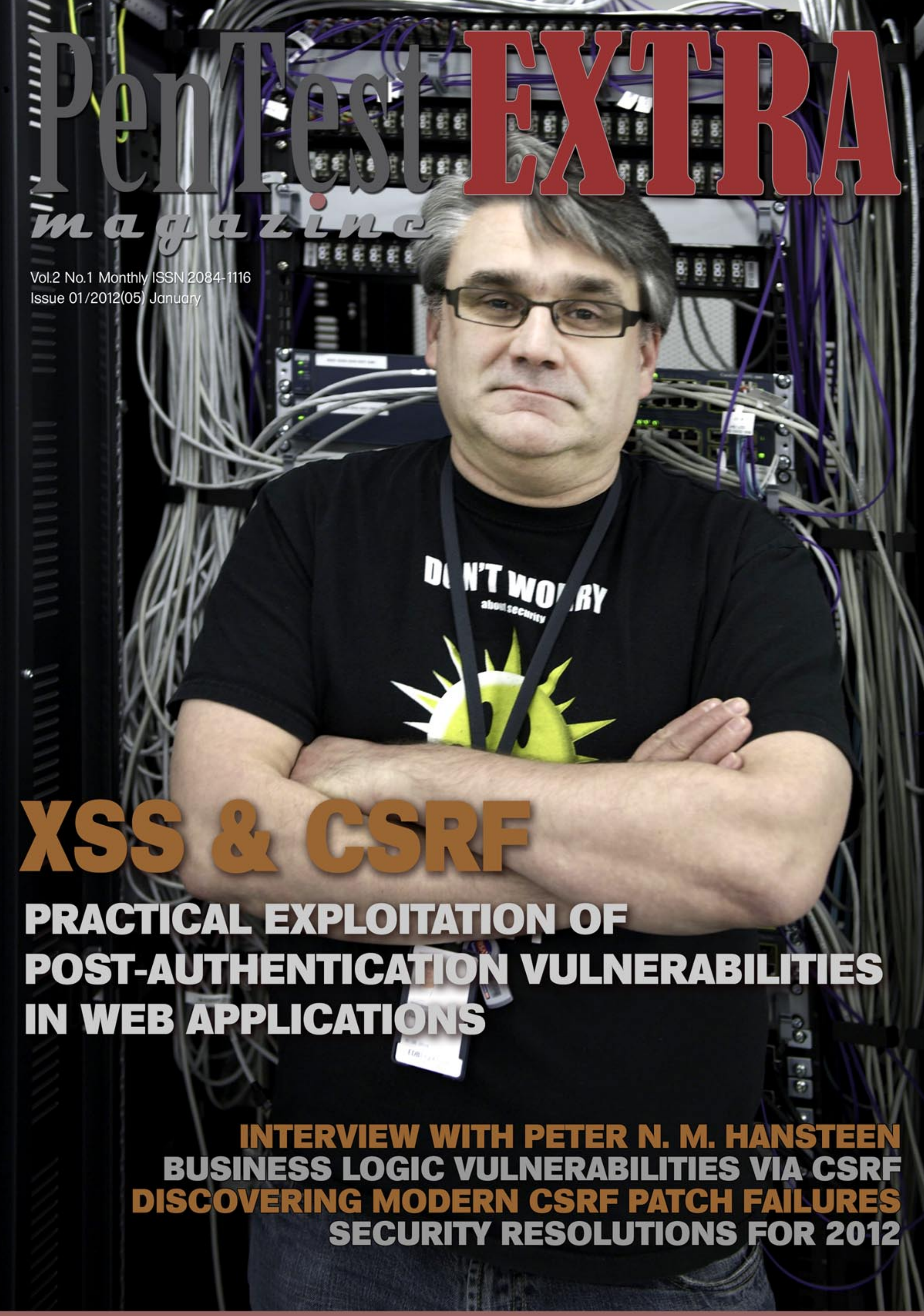
PenTest EXTRA magazine

Vol.2 No.1 Monthly ISSN 2084-1116
Issue 01/2012(05) January



XSS & CSRF

PRACTICAL EXPLOITATION OF POST-AUTHENTICATION VULNERABILITIES IN WEB APPLICATIONS

INTERVIEW WITH PETER N. M. HANSTEEN
BUSINESS LOGIC VULNERABILITIES VIA CSRF
DISCOVERING MODERN CSRF PATCH FAILURES
SECURITY RESOLUTIONS FOR 2012

Interview with Peter N. M. Hansteen

Peter N. M. Hansteen is a consultant, writer and sysadmin from Bergen, Norway. A longtime freenix advocate and during recent years a frequent lecturer and tutor with emphasis on FreeBSD and OpenBSD, author of several articles and The Book of PF (No Starch Press 2007, 2nd edition November 2010). He writes a frequently slashdotted blog at http://bsdly.blogspot.com/.

How did you get your start in the information security field?

Peter N.M. Hansteen: I'll risk sounding a little blunt here, and say that it was really a matter of a series of accidents that lead to, well, a known result. Early on I had what you might call a rather meandering carreer path before I finally started pointing myself in a generally IT-ish direction. Fortunately while I was taking night classes in IT subjects and working a day job in a very junior clerical position at the Norwegian School of Economics here in Bergen, I got an early introduction to the Internet as it was then in the mid to late 1980s. I remember distinctly that a fair number of the machines we encountered at the other side of telnet, archie, ftp and other services ran something slightly exotic called *BSD Unix*.

A few job changes later and I found myself in a position where I was the person in charge of information security and everything IT-ish for myself and about a dozen colleagues. As the inevitable Internet commercialization came around I had a slight edge after some early exposure and hanging around BBSes in the meantime. But then again we had some wonderful security failures too, as far as I can tell not too dangerous and never really breaking anything important, but well, the stories are out there in some form if you poke around USENET archives. Enterprising readers will know where to look.

What drove you to pursue information security?

PNMH: Information security, again, is part of the bigger picture. You want to provide a convenient working environment as well as making sure you keep your colleagues safe from harm, all the while doing your best to implement a regime that protects whatever the organization's assets are. For my own part it all grew out of that motivation. The process was quite gradual. And of course gradually you build up a toolchest. There are invariably applications or entire environments that you would dearly like to do take out of the equation that also happens to be something your client can not be moved to do without. For my own part I ended up with a preference for open source tools in general and OpenBSD in particular. That position evolved in part from various less pleasant experiences with the various proprietary systems, and partly from the rather obvious insight that with open source tools, you actually can check what the tools do and change or enhance any part of the toolchain if you want to.

Then again, whatever you do and how you ever choose to run your security efforts, the security bits have to be integrated in your environment. Basically the tools and procedures need to be part of the normal, ordinary way of going about your business. If your strictly enforced security regime with tools and procedures get in the way of how the organization needs to run its business, your users will find ways to subvert your goals and you may find yourself exposed. It's the *you made the thing foolprof, so they went ahead and created a bigger fool* problem coming back at you.

What do you see as the biggest challenge to information security five years down the road?

PNMH: Well, to start with there are four things we can be absolutely sure will be as problematic five years from now as they are today: Bad design decisions, needlessly growing complexity, implementation bugs, and your trusted users' actions, including your own. For the first three the constant, ongoing code audit of the type the OpenBSD project practices and preaches will give you a head start. But apart from stating the obvious, there are a few other worrying developments that have been happening for a while and have only recently started to come to the general public's at\tention.

One such development is the growing tendency of govenments, even Western ones, to demand the right to peek ever more closely into people's private information, with little or no accountability. The European Union's Data Retention Directive is one such piece of legislation, which mandates that any traffic logs you may be generating for your own needs have to be kept around for longer than any sane techie would think of, just in case law enforcement wants to take a peek. You could of course say that the original intentions were good and point to the so-called war on terror. But we have already seen the motivation morph into the need to catch child molesters, then it got tweaked a little more to be included as a weapon in the decades-old war on drugs and recently it's been found to be vital in the struggle to catch traffic offenders, beaten to the punch only by a very misguided chunk of the media publishing industry, which for good measure seems to be intent on running its own little branch of law enforcement in their very own style.

The same ugly picture includes various national laws that codify warrantless wiretapping and other forms of fine grained surveillance, and there is even legislation on the way that mandates various forms of censorship that may lead to serious technical issues in the name of copyright enforcement. All taken together it looks like a fairly thorny path ahead, and it's worth keeping in mind that all of those things that sound scary enough for individuals pose a real risk for companies too. To some extent we've always had industrial espionage, but to West Europeans at least the idea that your own government could realistically be the ones trying to pry into your confidential information is somewhat new and quite unpleasant.

At the end of the day, bugs of any kind and social engineering will return to bite us, and we won't be rid of either any times. Our adversaries will continue to rely on those techniques. Well designed tools and good code, validated and audited in full public view will help, as will educating your users. Keep in mind too that in this context you, the security professional, are very much a user yourself, with access to elevated privileges that may mean when you do screw up, the situation could escalate into something far more dangerous than run of the mill user's goofs.

Does the OpenBSD version numbering approach confuse people?

PNMH: I suppose it does confuse people that in OpenBSD, the version number is just another identifier, and it gets incremented by exactly 0.1 every six months.

The reason OpenBSD does it that way is that the project has chosen to live by a strict six month development cycle. The development cycle is itself split into roughly four months of introducing new or improved features followed by two months of stabilization leading up to cutting a release and sending it off to production at some never-preannounced date. For the development team this means that large reworks of code will have to be split into chunks that will realistically fit within that timeframe.

The much-ballyhooed and very useful syntax changes that appeared in PF over the OpenBSD 4.6 and 4.7 releases had in fact been works in progress for some years when they hit the tree for general use. For last November's release, 5.0 just happened to be the next increment in line. The release did have some major new features, for PF the prio keyword is the first part of a new traffic shaping engine that will eventually replace the venerable ALTQ when the time comes.

There is kind of a roadmap in place, but the developers have not officially commited to a timetable or specific release when ALTQ is supposed to be replaced. It will happen when the new code is ready and clearly better than the older one. When something new and exciting is committed, I hope to be one of the first to write a blog post about it. My PF tutorials tend to include at least some mention of recent developments, too.

Do you believe all the regulations set forth regarding information security has helped or hindered information security growth?

PNMH: First of all, there is more legislation that's relevant to information security today than there was earlier, and security professionals need to be aware of what rules apply to them. Some legislation may have been beneficial, if for example it was needed in order to codify clear standards of ethical conduct. Basically you need a working knowledge of what rules apply. So the various rules and regulations have made life anything from slightly more complicated to somewhat painful in recent years, depending on where you are and what you do.

If you work in several jurisdictions, you may need to get a lawyer or even a judge to affirm which set of rules apply in each case, and if the precendence of rules is unclear or worse the rules are eve slightly incompatible or unclear, your legal fees could become substantial.

Again it's important to be aware that recent legislation in the US and elsewhere written with the intention of short-circuiting the normal due process rules in certain types of criminal cases, notably those labeled 'terrorist' by the prosecution. Unless those rules are found unconstitutional in a hurry, we should expect to see information security professionals behind bars for indefinite periods soon enough.

Is there a better way to allow root access for remote admins?

PNMH: Heh. There has been a lot of discussion on just what level of immediate access is appropriate for admins when they are in a hurry, but realistically the question boils down to this: What level of exposure to the various threats, including the risk of your own mistakes, is appropriate in your context?

I don't believe there is an easy one size fits all option available. Your analysis of the specific context, with its own set of risks and probabilities and anticipated threat factors dictates what is appropriate.

But reeling back a bit, your question is really about the basic conflict or tradeoff that admins see between convenience on the one hand and security on the other when they need to access critical devices. It's so very convenient to go directly to the maximally permissive settings so you can do anything you like without getting caught up in red tape.

When it comes to what constitues acceptable risk, it really is up to you. If you, after appropriate risk analysis, are confident that logging in to a remote device with the highest possible privilege is appropriate, if you are equally confident that you can effortlessly recover from any mistakes you make while running with maximum privilege and you consider the risk that anyone not formally authorized to reach that level will manage to do so is negigible to non-existent, you are at liberty to go directly to root.

I tend to advocate disallowing direct login to any privileged account, to encourage use of encryption of the strongest practical kind and when appropriate and available, key based authentication or some sort of two factor authentication system. Mainly because I know that I am not infallible, and in some contexts I need a reasonable assurance that anyone attempting unauthorized access would need to expend enough effort that my systems would detect the attempt.

I dislike running with elevated privileges whenever it isn't strictly necessary, mainly because I know that I'm human and will make mistakes, and that configurations can break in unexpected ways. There are, for example, failure modes on some Unix-ish systems that would land you with / as your home directory and no warning that's where you are other than – if you're lucky – a command line prompt that looks subtly different from what you are used to seeing. In those contexts, it's essential to do the right things, and your confidence that you will have *grace under pressure* will be sorely tested.

A large part of the problem is to ensure that any task in the system runs with an appropriate level of privilege. In the OpenBSD project, a lot of work has gone into properly implementing privilege separation in the various daemons. In effect, making sure only those parts of the system that need elevated privilege ever achieve that privilege, and in most cases the program gives up the privilege once the task such as binding to a port below 1024 has been achieved. The most immediately user-visible consequence is that you will find the OpenBSD password database pre-populated with a number of special-purpose users (most of them with names that start with an underscore character '_'), defined specifically to run services at their appropriate privilege levels.

The privilege separated OpenBSD system is out there and available for daily use, and I would encourage your readers to try it out. There are interesting efforts going on in other projects as well, with the main keywords being RBAC or *Role Based Access Control* – essentially a deconstruction of the user authentication and authorization (implemented among other places in the most recent Solaris releases), and from the opposite end of the table, fine grained capabilities models for process privilege separation, with the FreeBSD project's Capsicum project (if I understand correctly to hit mainstream in FreeBSD 9) on my short list of things to look into in the near future.

But the increased complexity that grows naturally from these approaches also means the code and configuration needed to fit the code to your purposes is harder to do correctly, and so we are almost certainly entering dangerous territory for that reason alone. It will take significant development effort to rein in those concepts into something manageable for the average sysadmin, assuming we're also able to squash enough bugs in the process to make the effort worthwhile.

What new concepts or applications are available, or coming available soon, for firewalls?

PNMH: The firewalls concept in its simplest form – and that's what people get hung up on – is rather simplistic. The main decision is to block or pass. Modern firewalls do a lot more of course, including but not restricted to failover and redundancy with CARP and pfsync or VRRP, network address translation and even IPv4 to IPv6 conversion, redirections, load balancing and traffic shaping. The good ones even adapt to network conditions via adaptive state timeouts or can be configured with state tracking tricks that fend off excessive traffic of specific kinds.

And of course there is more, but there is a tendency for news about interesting technical development to drown in marketing hype, so I may be ignoring important work that's going on out there. Personally I think the authpf system – OpenBSD's and PF's non-interactive shell that loads rules on a per user basis – is one type of feature that I think will see a lot more attention and wider use in the future. It's so obviously a good thing to tie what the network lets you do to your user or group identity or to a set of role based criteria.

Come to think of it, most of these advanced firewall features are serisuly under-used and not as well understood in the community at large as we could have liked. But perhaps the identity or role centric setups are the ones with the most scope for interesting development over the next few years, if the added complexity can be managed somehow.

How does BSD pf compare to iptables, ipfw, ipfilter or other firewalls? What is its strength or weakness?

PNMH: The short answer, coming as it would from the author of The Book of PF, is obvously that the other ones suck. But seriously, since I kind of abandoned the other ones in favor of PF at some point, I think it's better to at least start answering the question by describing some of the features that attracted me to PF over the other ones. Then we'll get around to any weak points if we can still remember them after a while.

It's important to remember that PF is developed as an integrated part of OpenBSD, and one of the important design goals has always been that it should be very usable for OpenBSD users. This means that all the features I've touched on earlier are within easy reach directly from your pf.conf configuration file or somewhere equally accessible.

One usability feature I appreciate a lot is called *atomic ruleset load*. It's perhaps easier to explain why this is important if we look at the other ones: iptables and ipfw configurations are actually shell scripts, where each rule is loaded as a separate command. This means that if you press [Ctrl-C] while the script is executing, you have very little control over what rules are actually enabled. More likely than not, some lines of your script were never executed, meaning that your configuration did not load completely, with unpredictable results. IPfilter's developer apparently did not trust the software to keep track of loaded rules by itself and recommended *flushing* previous rules before loading a new configuration.

None of this is necessary with PF – if your rule set is syntactically valid, it will load, completely replacing the previous one. There is no need to flush existing rules, unless you want to make sure you have a a period of 'pass all' to give miscreants a break until you load the next valid rule set, and running a real risk of disrupting valid traffic that (think timeouts due to disappearing redirections) that would have seen no trouble on a clean ruleset load.

If you think this means that PF configurations are totally static, you're wrong. If you need to adjust the contents of your rule set on the fly, your best bet is to create what PF calls an anchor – essentially a named sub-ruleset, and yes, you can have several – where you or applications you write can insert and manipulate rules dynamically, something Apple appear to have used to great effect in their port to MacOS. Apple even wrote some enhancements to the anchor loading code, but unfortunately they wrapped their new bits in #ifdefs with a separate license, so the extended functionality will not easily make it back into the mainstream PF code. You can look up my Call for testing article (see the references at the end) for more details.

And of course for simpler operations like singling out hosts that need special treatment, you can manipulate tables of IP addresses even outside anchors, or you can use state tracking options magic to move IP addresses into tables, and use the tables in your filtering criteria.

From my experience, PF and related tools on OpenBSD provides you with the sanest working environment available for interacting with the TCP/IP stack so you can make your equipment perform the way it's supposed to. None of the other tools come even close, in my opinion, in either admin friendliness or performance.

Will the next intrusion platform be mobile devices?

PNMH: To some extent, or possibly even to a large extent, I think the shift has already happened, in the sense that the focus of would-be intruders is changing more or less in step with the mainstream user and the perceived high-value targets. I'm not suggesting that the installed base of PCs will be going away anytime soon, most of those are well past their use by date anyway, but rather that the Windows PCs that today still make up the largest part of the installed base are destined to become less important over time if current trends continue more or less as we see them today.

Mobile devices are getting a lot of attention these days, and malware targeted at them is of course getting some too. The situation for mobile devices designers today is somewhat parallel to the situation when PCs were introduced to the Internet, but there are important differences.

One such difference between back then and now is that a large part of the PC related business is still aimed squarely at patching or working around security bugs in the most common desktop operating environment. That, and the fact that there are more network-savvy developers out there today than at any time earlier makes me a little hopeful that at least some of the grosser mistakes of PC networking history will not be repeated by mobile device developers. Also, so far we have avoided the monoculture that helped make PCs on the Internet such easy marks. Mobile devices vendors have a real choice in software stacks, and at least the two dominant operating environments in the smartphone space (Apple and Android) are both vaguely Unix-based and use open source components to some degree, which seems to me like their designers are capable of making intelligent decisions.

That said, I'm fairly sure that even in those environments, users and miscreants will find ways to exploit bugs, and some subset of users will always be willing to do things that are simply not smart things to do. One example comes to mind – users of Apple phones decided that their phones ran a system that was unix-like enough that it should be able to accommodate a Secure Shell (ssh) server, and somebody managed to port the software. Only that developer decided to provide a setup with a default password, and there were several reports of phones that were taken over via ssh, thanks to the known default password that the user never bothered to change.

Now I'm geek enough to appreciate the attraction of having a shell login to the phone you carry in your pocket, but (as I noted in a slashdotted blog post at the time) the point here is not that sshd is an insecure piece of software. It isn't. The lack of security comes from not bothering to change your password from a well-known default value.

Something similar is bound to happen again, where a user makes a stupid mistake that has security implications. If we're lucky the damage will be limited to the users's own equipment, but if that user is also a developer and by design or accident inserts exploitable code in other users' mobile devices, the damage could become more widespread.

It's also worth keeping in mind that even if mobile devices seem relatively boring by modern PC standards and may or may not contain useful data, they may still be useful to botnet herders. A typical smartphone today has general processing power at least on par with a run of the mill PC at the time the network dependent malware started turning up on the Microsoft platform, and it's almost certainly on a better network connection than most PCs were back then. If your smartphone doubles as your wallet, all the more reason to pay attention.

How can these mobile devices be protected?

PNMH: If the mobile devices industry indeed manages to avoid making the same mistakes as the PC industry before it, I think we have something of a head start. Over the years what passes for IT security has focused on *enumerating badness* (do read Marcus Ranum's essay linked to in the references for more on that) and in the process diverting attention from the root cause issue that a certain software marketer was, for quite some time, reluctant to even acknowledge that there were bugs to be found in their software.

It may not have been obvious at the time Marcus was writing that essay, but history has taught us that the approach the PC industry took to security at the time – heaping another level of complexity on top of buggy software in the name of security – is not necessarily an improvement in real terms, even if the new layer

References

The references are listed in roughly the order they're mentioned in the text, read them for further treatment of some of the issues I mentioned here.

- The OpenBSD project *http://www.openbsd.org/*
- The FreeSBD project *http://www.freebsd.org/*
- PF tutorial home page *http://home.nuug.no/~peter/pf/*
- The Capsicum Project at Cambridge University, *http://www.cl.cam.ac.uk/research/security/capsicum/*
- How Apple Treats The Gift Of Open Source: The OpenBSD PF Example *http://callfortesting.org/macpf/*
- The Book of PF (second edition) *http://nostarch.com/pf2.htm* or from good bookstores everywhere
- Rickrolled? Get Ready for the Hail Mary Cloud! *http://bsdly.blogspot.com/2009/11/rickrolled-get-ready-for-hail-mary.html* (slashdotted as „The Hail Mary Cloud is Growing, Nov 15 2009, *http://linux.slashdot.org/story/09/11/15/1653228/the-hail-mary-cloud-is-growing*)
- Marcus Ranum: The Six Dumbest Ideas in Computer Security, *http://www.ranum.com/security/computer_security/editorials/dumb/index.html*

somehow provides a workaround for the nasties you know about in the original code. The added complexity most likely means that your debugging gets harder for the next round of problems.

In a way it would be nice if mobile device designers started basing their systems on OpenBSD, which is probably the general purpose system that has been developed and maintained with the most attention to security. I want a phone I can trust, and preferably one that's open enough for qualified developers to hack on. And the same applies tablets and other devices too, of course.

Regardless of what technology the devices are based on, I think a combination of user education and operators paying attention to end user equipment is the way forward. If operators are able to take some of the system administration workload off their end users' hands for a nominal fee, it could turn into a profit center.

It really boils down to a sane system administration regime – don't run any services that are not required for your use case, log properly and pay attention to what your logs say, update your systems at intervals and definitely when security relevant bugs have been fixed.

On the other hand, in addition to user education and the offer of handholding we may need a measure of negative reinforcement – one approach is to mimic the way we treat pets or livestock and their owners. Dogs and computers both are capable of autonomous actions to some extent, so it might be a useful parallel. Dog owners are used to cleaning up the messes their pets make on sidewalks, and if the animal bites somebody, the owner is usually responsible for paying for the damage. Sufficiently stupid behavior with regard to your pet can sometimes earn you a charge of reckless endangerment. I think you can validly argue that a similar regime should apply to owners of fairly damage-capable computing devices.

How can these mobile devices be firewalled?

PNMH: On a technical level, I think that problem is very close to being solved. The existing tools could be adapted fairly easily to fit a roving user scenario (some people are already paying attention), and some of the anticipated developments I mentioned earlier may make the devices even easier to use. But once again, operators and service providers could play a significant role if they manage to come up with useful ways to interact with users' devices. And of course we need to stomp out the snake oil salesmen, if we can't scare them off right away by building sanely constructed devices with trustworthy software.

How do you even know if someone is attempting to access your mobile device or using it to run ssh login attempts against remote systems?

PNMH: On the current crop of devices, I think you'd be blissfully ignorant of any such attempts until either your phone starts doing something unexpected or your next bill turns up with a lot more traffic to pay for than you had expected.

With any of the devices that are vaguely unix-based it shouldn't be very hard to log properly, and once again I think operators should be looking seriously into offering their users some kind of log monitoring and other admin services in order to help run mobile devices sanely. Intelligently designed mobile device management services could become the real differentiator in the telecom operator market. I hope the the operators are paying attention.

And finally, for penetration testers out there, there will always be bugs out there to hunt for and exploit, and if you have a hard time finding those, you can always go for layer 8 or 9 techniques :)

Happy hacking!

By PenTest Team