which can be used in fraudulent online transactions.

This paper presented recommendations relating to technology, process and people, to assist in mitigating unauthorised disclosure of information through point-of-sale transaction receipts in South Africa.

About the author

Dr Suné von Solms is currently employed as a lecturer at the Faculty of Engineering and the Built Environment at the University of Johannesburg, South Africa. She is a computer and electronic engineer, researcher and academic with a keen interest in telecommunication, cyber-security, engineering education and the social and human aspects of engineering.

References

1. Terblanche, Irlon. 'Card swipes comfortably overtake cash'. Moneyweb, 4 November 2013. Accessed August 2015. www.moneyweb.co.za/archive/card-swipes-comfortably-overtake-cash-irlon-terbla/.
2. Goldstuck, Arthur. 'Internet Matters: The Quiet Engine of the South African Economy'. World Wide Worx, May 2012. Accessed April 2015. http://hsf.org.za/resource-centre/focus/focus-66/AGoldstuck.pdf.
3. 'Verified by Visa'. Visa International Service Association. Accessed April 2015. www.visaeurope.com/making-payments/verified-by-visa/.
4. 'MasterCard SecureCode – Enhanced Security For Online Shopping'. MasterCard. Accessed April 2015. www.mastercard.com/za/consumer/securecode.html.
5. 'Sharp increase in bank card fraud'. South African Government News Agency, 25 November 2015. Accessed August 2015. www.sanews.gov.za/business/sharp-increase-bank-card-fraud.
6. 'How criminals steal your credit card info'. Mybroadband, 2 December 2014. Accessed April 2015. http://mybroadband.co.za/news/banking/114873-how-criminals-steal-your-credit-card-info.html.
7. Von Solms, Suné. 'An Investigation into Credit Card Information Disclosure through Point of Sale Purchases'. Proceedings of the 14th International Information Security South Africa Conference, August 2015.
8. 'Card-based payment systems'. Payment Association of South Africa. Accessed September 2015. www.pasa.org.za/more_cardbased.html.
9. 'Card acceptance guidelines for Visa Merchants'. Visa International Service Association, 2015. Accessed August 2015. https://www.visa-asia.com/ap/sg/merchants/include/card_acceptance_guidelines_for_visa_merchants.pdf.
10. 'How Credit Cards Work'. HowStuffWorks.com. Accessed September 2015. http://money.howstuffworks.com/personal-finance/debt-management/credit-card1.htm.
11. 'Security Rules and Procedures – Merchant Edition'. MasterCard, 2015. Accessed August 2015. https://www.mastercard.us/content/dam/mccom/en-us/documents/SPME-manual-February2015.pdf.
12. 'What is Ecommerce?'. Network Solutions, 2015. Accessed September 2015. www.networksolutions.com/education/what-is-ecommerce/.

The secure way to use open source

Steve Mansfield-Devine, editor, *Computer Fraud & Security*

Steve Mansfield-Devine

Open source software long ago shrugged off its 'fringe' image and is now a core component in a great many solutions. These include commercial software, operating systems and bespoke code. But while much of the open source codebase has accrued a reputation for reliability and flexibility, there have also been some high-profile and highly dangerous vulnerabilities discovered in the code. In this interview, Patrick Carey, director of product marketing at Black Duck Software, explains how open source software can open the door to major threats – but also what you can do to enable the safe and effective use of the code.

Deep in the code

Often, open source code is built deep into solutions – which makes it much more serious when problems are discovered. The 2014 flaw that became know as Heartbleed (CVE-2014-0160) – and which quickly acquired its own logo and website – was found in the very popular OpenSSL cryptographic software library. This is widely used to provide secure communication with web, email, instant messaging and, in some cases, Virtual Private Network (VPN) traffic.[1]

According to the website: "The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt

Patrick Carey, Black Duck Software: "Open source has really exploded over the past decade, and become a core component of not only the development of the Linux operating system, but a key component of most applications."

the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users."

Heartbleed arose from a simple programming error in OpenSSL's implementation of the TLS/DTLS heartbeat extension (RFC6520).[2] While most bugs get fixed by new versions of the code, this one had been around for a considerable time. And what made it serious is that it potentially compromised the encryption keys themselves – the crown jewels of public key encryption processes.

The Heartbleed bug was probably the first vulnerability to get its own logo.

The Shellshock bug was made public later the same year.[3] In fact, there turned out to be an entire family of flaws within the Bash shell, so beloved of Unix users. The flaws could lead to the execution of arbitrary commands on a system on which a vulnerable version of Bash was running. Within hours of news of the flaw being made public, hackers had taken advantage of it to commandeer systems in order to build botnets and mount distributed denial of service (DDoS) attacks.

"Heartbleed, Shellshock and other flaws have attracted attention precisely because they occurred in open source software"

In early 2016, TLS security came under attack again when the Drown cross-protocol vulnerability was discovered (CVE-2016-0800).[4] It exploited the fact that modern TLS solutions have legacy support for the obsolete and insecure SSLv2 protocol. At the time of the announcement – accompanied by the by-now inevitable logo and website – only a partial fix was available.[5] As of late March 2016, the website for the bug estimated that 15% of the top one million sites using HTTPS were still vulnerable and around 28% of all sites employing TLS/SSL encryption could be exploited by the bug. This is largely due to server misconfiguration – that is, if the server still allows SSLv2 connections.

Somewhat surprising

Inevitably there were fierce debates about the true danger posed by these flaws. After all, Microsoft's Windows ecosystem has been providing a fertile environment for millions of examples of malware and other exploits for years. Nevertheless, Heartbleed, Shellshock and other flaws have attracted attention precisely because they occurred in open source software.

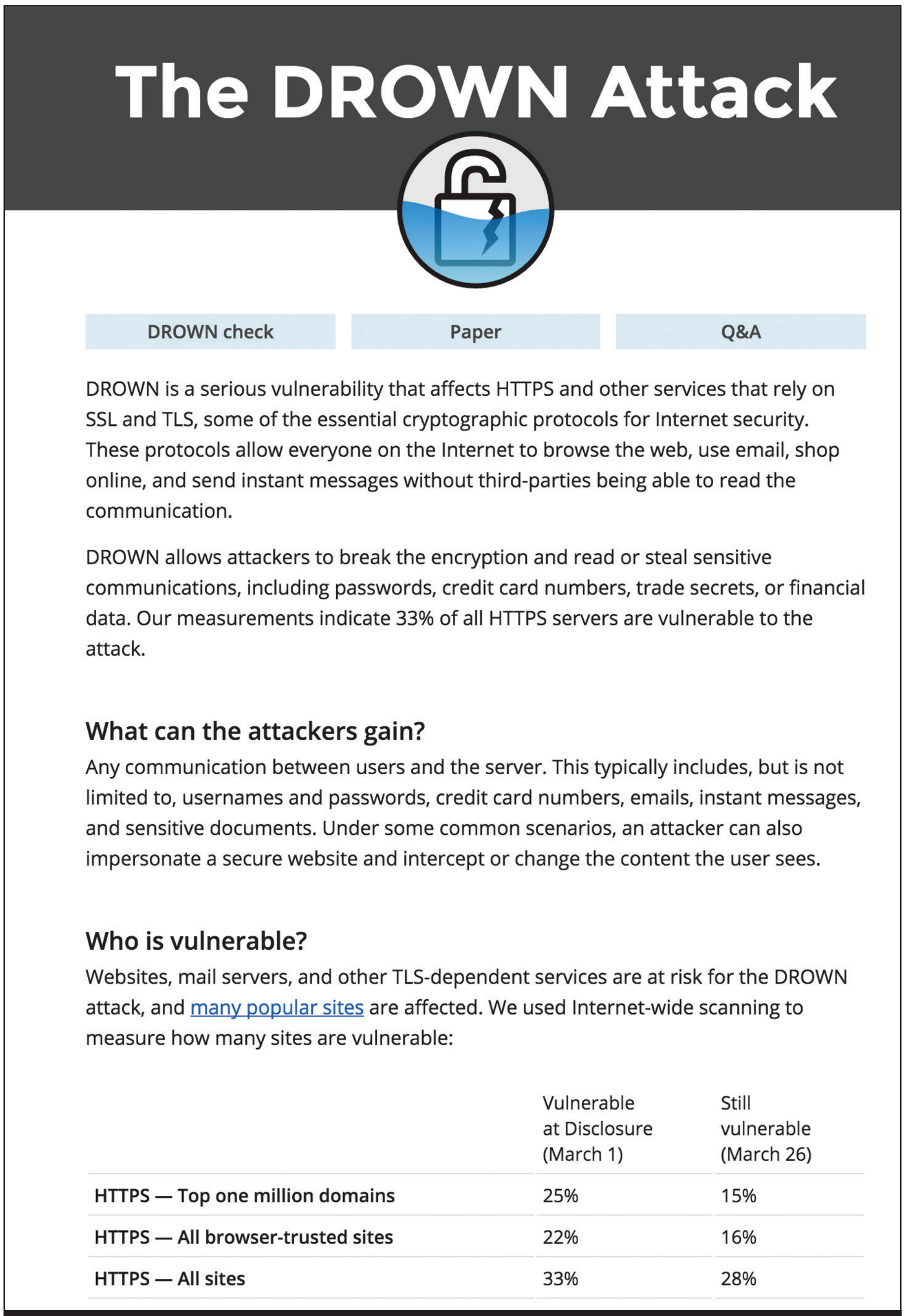
"Actually, there's been a steady increase in reported open source vulnerabilities over the past 15 years, and in fact there were over 6,000 recorded in the past two years alone," says Carey. "We think that's a function of the amount of open source development that is taking place. Open source has really exploded over the past decade, and become a core component of not only the development of the Linux operating system, but a key component of most applications – both commercial applications, and enterprise applications. In fact, Gartner predicts that open source will be 80% of the Global 2000 code base by 2018."

"There's certainly a large and increasing number of security research teams investigating vulnerabilities with both open source and commercial software"

It stands to reason, Carey says, that as the code base grows, so will the number of vulnerabilities discovered. This isn't just a case of more code equals more errors. It's also a side-effect of a growing awareness of security issues and the growth of the information security community.

"There's certainly a large and increasing number of security research teams investigating vulnerabilities with both open source and commercial software," says Carey. "And as they look to give a public service or find vulnerabilities for their own organisations, we're going to see more of these vulnerabilities reported. It's interesting that many of the vulnerabilities that most people would be aware of, like Heartbleed or Shellshock or Ghost – some of the vulnerabilities that were reported over the past few years – were actually found by research teams specifically looking for open source vulnerabilities in different code bases."

While an increase in the number of researchers combing the code base for dangerous flaws is unquestionably a good thing, it's always a double-edged sword. Most researchers are diligent



The DROWN Attack

DROWN is a serious vulnerability that affects HTTPS and other services that rely on SSL and TLS, some of the essential cryptographic protocols for Internet security. These protocols allow everyone on the Internet to browse the web, use email, shop online, and send instant messages without third-parties being able to read the communication.

DROWN allows attackers to break the encryption and read or steal sensitive communications, including passwords, credit card numbers, trade secrets, or financial data. Our measurements indicate 33% of all HTTPS servers are vulnerable to the attack.

What can the attackers gain?

Any communication between users and the server. This typically includes, but is not limited to, usernames and passwords, credit card numbers, emails, instant messages, and sensitive documents. Under some common scenarios, an attacker can also impersonate a secure website and intercept or change the content the user sees.

Who is vulnerable?

Websites, mail servers, and other TLS-dependent services are at risk for the DROWN attack, and many popular sites are affected. We used Internet-wide scanning to measure how many sites are vulnerable:

	Vulnerable at Disclosure (March 1)	Still vulnerable (March 26)
HTTPS — Top one million domains	25%	15%
HTTPS — All browser-trusted sites	22%	16%
HTTPS — All sites	33%	28%

The Drown attack website.

about complying with responsible disclosure practices in which the maintainers or vendors of software are given advanced warning of flaws – and therefore time to fix them and issues patches – before details of the problem are made public. But this isn't always enough. Shellshock demonstrated that attackers will always find victims with unpatched systems.

This, says Carey, "is why companies building software with open source need to remain vigilant and have a plan in place for managing those vulnerabilities."

Many eyes

If you listen to open source evangelists, you'd be forgiven for thinking that major bugs should never arise in the first place. One such evangelist, Eric Raymond, famously said: "Given a large enough beta-tester and co-developer base, almost every problem will be characterised quickly and the fix will be obvious to someone".[6] He then paraphrased this as: "Given enough eyeballs, all bugs are shallow," which has entered computer lore as Linus's Law, named in honour of Linux creator Linus Torvalds.[7]

It doesn't always work that way, however. Often, with open source software, there aren't many eyes overseeing the software, but just a few – typically the same coders who made the mistakes in the first place. After software is released it may get little further attention other than fixing the bugs that are found and made publicly known. And, as Torvalds himself said, in the exchange with Raymond that led to the formulation of the law named after him, finding the errors is the difficult part.

So do the high-profile bugs that were discovered in the past few years represent a busting of the open source myth?

"I don't think so," says Carey. "Any code base is going to be subject to defects, and some of those defects are going to result in security vulnerabilities. I'd say that actually the open source model is working, because there is not only a crowd of developers working to build the open source components, but there's the global crowd of people investigating and testing that open source."

Both open source and proprietary software are liable to flaws. It's unlikely that any significant piece of software has ever been released without some kind of bug in it. And there are plenty of people who test, probe and fuzz software to tease out its weaknesses. Some of those people are security researchers, a proportion of whom are seeking glory while others chase down bug bounties or are simply doing due diligence. And, of course, rather too many of those people are malicious actors looking for weaknesses to exploit.

What's special about open source software, of course, is that these researchers have the source code to work with, increasing the likelihood that they will find the bugs. They may not do that before the code is released, but at least they'll get there eventually. In effect, this community of researchers and developers is acting as those "many eyes", though not necessarily in the manner Raymond originally envisaged. And Carey thinks that organisations or teams that are building software that leverages open source components should look to tap into that global community.

"We think that there's a better model, where advanced insight and review of the security profile or risk profile of these open source projects can factor in very early in the software development life cycle"

"I think many organisations think of open source, and treat it, like they would a commercially available component," says Carey. "But with open source, you don't have an organisation like Microsoft pushing an update to you, and automatically applying that to your system. So organisations need to take on more of the responsibility themselves for monitoring vulnerabilities."

In good faith

Determining for yourself whether open source code is safe to use is simply not feasible. Codes typically employ frameworks and libraries to speed up development, as well as add capabilities that would be difficult for them to create themselves. A great deal of the basic functionality of an application can be derived from these ready-made components. And there are some areas of coding, such as cryptography, where it's unwise for developers to try to do it themselves. But many of these frameworks and libraries are large: it would be difficult and unprofitable to spend time trawling through the code and a full security audit is usually out of the question. So does this mean that developers must take the quality of the code, including its security, on good faith? For Carey, the answer is both yes and no.

"Certainly, most organisations leveraging open source are making that leap of faith today, and the statistics would indicate that that's the majority of organisations building software," he says. "But we actually think there's a better model, where advanced insight and review of the security profile or risk profile of these open source projects, can factor in very early in the software development life cycle [SDLC] – during the requirements phase where open source projects are selected."

This insight comes from taking advantage of vulnerability data that already exist. Open source code is not only "crowd developed", as Carey puts it, but also "crowd tested", which means that important risk information is gathered in a variety of publications and vulnerability databases. So a key step for any organisation developing code that uses open source components is to consult these information sources, preferably as early in the development life cycle as possible. But that's not always easy.

"The vulnerability data is out there," says Carey but adds: "It can be somewhat difficult for organisations to tap into today."

Bringing it all together

To save developers the trouble of picking through the information sources, and potentially missing something important, companies such as Black Duck Software are aggregating the various databases and presenting the data in a more easily accessible form. This starts with cataloguing.

"Even though many organisations are leveraging open source today, the way we see most organisations maintaining or tracking that is still relatively primitive," says Carey. Not all organisations bother to track the vulnerability information, but those that do are doing it via manual methods which, he adds, "essentially boils down to storing a list of open source components and versions in a spreadsheet and asking the developers what they're using."

As any spreadsheet user will know, this is likely to be an error-prone process, especially as, in this instance, it relies on the development teams to reliably report what they're using and keep the list current. Yet having visibility into what open source components are being used is fundamental to being able to maintain visibility into the security and compliance risks.

"If you don't know what's in your code, you don't have any opportunity to stay on top of those risks," says Carey. "So cataloguing is the first step along the journey of managing those open source risks."

Once you have your open source components catalogued, you can then start mapping these against resources such as the National Vulnerability Database, maintained by the US National Institute of Standards and Technology (NIST).[8] Again, doing this manually is likely to be time-consuming and error-prone, which is why services such as Black Duck's help automate the process as well as layering on additional information sources.

Development process

Using external software frameworks and libraries introduces an additional dimension to the software development life cycle. You can't take a 'write, test, ship' attitude because even after your solutions are delivered, the code base on which they're built is likely to change. Some of these changes will be as a result of vulnerabilities being found and patched. But others are likely to be simple updates or upgrades, which can bring new vulnerabilities with them.

The normal processes of static and dynamic analysis will be deployed by organisations during a well-defined testing phase of the SDLC. But once this phase is complete, all modifications to the code stop as the development team moves to deliver the software.

"With open source, it's different," says Carey. "The world continues to change after you ship, so it's important to continue to monitor for new vulnerabilities that get reported after your application goes into production or deploys to customers."

Many of the flaws discovered in open source code have been in place for some time – and are, therefore, included in your code. "Once the application ships, your job in protecting that application doesn't end," Carey adds. "In some ways, it's really just beginning."

"Once the application ships, your job in protecting that application doesn't end. In some ways, it's really just beginning"

This may require a new way of looking at testing. Static testing tools often adopt what's know as a 'white box' approach, looking through the code itself for typical patterns that represent potential vulnerabilities, such as buffer overflows. Dynamic analysis, on the other hand, takes a 'black box' approach, analysing the application from the outside, fuzzing inputs, looking for vulnerabilities in ports and so on.

"We think of open source vulnerability management as having to work outside the box, where you're selecting good, low vulnerability components on the front end," says Carey. "You're constantly cataloguing and verifying the components that you're including in your applications during development and mapping those to currently known vulnerabilities during the development phase. Then, ongoing, you have to continue to monitor for new vulnerabilities – so it's a 'before, during and after' mindset that organisations have to take when trying to manage open source in their applications."

Automation

It clearly makes sense to automate the processes of cataloguing, mapping and monitoring as much as possible. This could be a challenge for individual organisations because of the number of data sources that need to be checked and the variety of formats in which they offer their data. So this is where firms like Black Duck come in, with tools that can trawl code looking for open source components and matching those against vulnerability information.

Some of these tools will simply use manifest files that list code sources used in a software build package. But those will catch only those open source components that are explicitly declared. If the developers have used approaches such as linking to binary repositories, this may not get picked up by the build tools.

A more effective approach uses source code scanning. Unlike static analysis, which looks for coding patterns, this carries out a composition analysis of the application. It looks for files and libraries that are present in the source code development environment that match known open source projects.

"It uses a technique called fingerprinting, where we're looking at the structure of the files and the directories, and comparing that against a database of known projects," explains Carey.

Proactive approach

In a sense, much of this is fairly passive – mostly double-checking that what you're using is secure. But is there a case for companies to take a more proactive approach in actually testing the code they're using, perhaps running penetration tests against the frameworks and libraries, and also getting more involved in feeding back into the open source community in terms of what they find?

"Organisations that are building with open source certainly do have a role to play here," says Carey, "and many of them do. Some of the vulnerabilities that end up getting reported come out of organisations like Google and Facebook and Twitter, who are heavy users of open source, and heavy contributors to the open source community. It can be difficult for enterprises, though, using traditional application security testing tools, to reliably find vulnerabilities and report them. Many of the researchers that are uncovering the vulnerabilities that we've talked about – like Heartbleed and Shellshock – were building specific test harnesses and sometimes using custom test tools, to run specific security and vulnerability tests against software packages. So, absolutely, organisations, if they do uncover vulnerabilities, should be reporting those, and making sure that they make it into databases like the National Vulnerability Database, so the broader community can benefit.

"The lesson here though is that it really is a community effort, so you should both be contributing to it, but also taking advantage of the contributions that the broader community is making, and not solely relying on your in-house testing and tools."

After the fact

Assuming you've created the right environment and have invested in the necessary tools for cataloguing, mapping and

Continued on page 20...

...Continued from page 19
monitoring, how does this help you when a major vulnerability is announced?

"There are really two essential things that are important here," says Carey. "When a vulnerability is disclosed for a piece of open source, one of the issues that you deal with as a developer is that the vulnerability is instantly made public and often, in very short order, so are YouTube videos and blog posts that give instructions on how to exploit it. For people who would be interested in hacking applications and websites, it's like giving them a key that opens hundreds of thousands of doors. Once a vulnerability is disclosed publicly, it's a race between you and potential hackers to see who will find that vulnerability in your code base first. So the earliest possible notification you can get is important to shorten that cycle."

The other factor here is that when a discovery is made about a flawed piece of code, that's not the time to start wondering whether you use it, or where it exists within your organisation.

"That's exactly what happened back in 2014, with Heartbleed, for many organisations," says Carey. "It got publicised very quickly, and many organisations found themselves completely in the dark, and having to answer the question, well, where is this? Are we affected? How many applications? And as you can imagine, if that's the point where you're trying to start the cataloguing process, it's going to take quite a bit of time before you hunt down and fix those vulnerabilities. So the proactive nature of knowing what's in your code on the front end, so that when vulnerabilities are disclosed you have the shortest possible distance to find and fix those within your code, is absolutely essential to maintaining security."

So with all these high-profile vulnerabilities, has the image of open source code taken a battering? Or have they shown how strong open source can be in fixing them?

"I think if you look at the statistics, it would indicate that open source is as strong as ever, and will continue to grow in the marketplace," says Carey. "However, I think this is an opportunity for the open source movement to build on its credibility if we can leverage this crowd-sourced model to help the community, to keep the community safe from vulnerabilities."

About the author

Steve Mansfield-Devine is a freelance journalist specialising in information security and is editor of Computer Fraud & Security *and its sister publication* Network Security. *He also blogs and podcasts on infosecurity issues at Contrarisk.com.*

References

1. Heartbleed. Home page. Accessed Apr 2016. http://heartbleed.com/.
2. 'Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension'. RFC6520. Internet Engineering Task Force, Feb 2012. Accessed Apr 2016. https://tools.ietf.org/html/rfc6520.
3. 'Shellshock (software bug)'. Wikipedia. Accessed Apr 2016. https://en.wikipedia.org/wiki/Shellshock_(software_bug).
4. 'Drown attack'. Wikipedia. Accessed Apr 2016. https://en.wikipedia.org/wiki/DROWN_attack.
5. 'The Drown attack' home page. Accessed Apr 2016. https://drownattack.com/.
6. 'The Cathedral and the Bazaar'. Eric S Raymond. 1999. www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s04.html.
7. 'Linus's Law'. Wikipedia. Accessed Apr 2016. https://en.wikipedia.org/wiki/Linus%27s_Law.
8. 'National Vulnerability Database'. NIST. Accessed Apr 2016. https://nvd.nist.gov/.

EVENTS

1–3 June 2016
CyCon: International Conference on Cyber Conflict
Tallinn, Estonia
http://ccdcoe.org/cycon/

1–3 June 2016
Symposium on Electronic Crime Research
Toronto, Canada
http://apwg.org/apwg-events/ecrime2016/cfp

7–9 June 2016
InfoSecurity Europe
London, UK
www.infosecurityeurope.com

13 June 2016
You Shot the Sheriff
Sao Paulo, Brazil
www.ysts.org

15–16 June 2016
Trust in the Digital World
The Hague, The Netherlands
www.eema.org/event/trust-digital-world-2016/

19–22 June 2016
Applied Cryptography and Network Security
London, UK
http://acns2016.sccs.surrey.ac.uk/

22 June 2016
Identity Management
London, UK
www.whitehallmedia.co.uk/idm/

25 June 2016
BSides Athens
Athens, Greece
http://bit.ly/1lvubhn

27 June–1 July 2016
OWASP AppSec Europe
Rome, Italy
https://2016.appsec.eu/