Eindwerk SN: Beveiligingssite

Adriaan Dens (r0257589)
Jan Weyens (s0205128)
Klaas Janssens (r0429246)
Quinten Van Der Auwera (r0258371)

7 maart 2014

Inhoudsopgave

Hoofdstuk 1

Inleiding

In het vijfde semester van de opleiding Toegepaste Informatica aan de Katholieke Hogeschool Leuven krijgt men het vak "Beveiliging". Dit voornamelijk theoretisch vak geeft de studenten een inleiding tot veiligheidsproblemen in de Informatica. Ons eindwerk richt er zich op een plaats te voorzien om deze theorie in de praktijk om te zetten en om informatie te voorzien voor studenten die meer willen weten.

Hoofdstuk 2

Problemen

2.1 Veilig nakijken en uitvoeren van onbekende C code

Voor de oefeningen over exploits in gecompileerde programma's willen we nakijken of de student succesvol het programma heeft kunnen exploiten. Hiervoor laten we toe dat de student zijn C code uploadt via de website waarna het zal nagekeken worden door een intern script.

De vraag is natuurlijk hoe je dit veilig kunt doen. De student kan immers schrijven wat hij wil in het programma dat hij uploadt en uitgevoert zal worden achter de schermen. Oplossingen hiervor zijn *Operating System level Virtualization* of Containers en volledige virtualisatie of Virtuele Machines.

2.1.1 Containers of VMs

Het verschil tussen beiden dat is het verschil tussen *Operating System level Virtualization* en virtuele machines is dat bij een OS-level virtualisatie de containers gebruik maken van de kernel van de host terwijl er bij virtuele machines een strikte scheiding is tussen twee machines, elk gebruikt zijn eigen kernel.

Het voordeel van virtuele machines is daardoor duidelijk, je kan namelijk een Windows VM maken en daarnaast een OS X VM draaien omdat deze toch geen kernel delen. Dit is niet mogelijk met OS-level virtualisatie. Containers hebben dan weer het zeer snel opstart doordat het gebruik maakt van de kernel van het hostsysteem.

Onze voorkeur gaat hierdoor uit naar OS-level virtualisatie omdat we enkel Linux machines nodig hebben en omdat we (zeer) snel een nieuwe container willen kunnen bouwen, gebruiken en dan terug verwijderen.

Er zijn enkele verschillende praktische implementaties van OS-level virtualisatie, ik bespreek hieronder de meest relevante voor onze probleemstelling. Chroots worden bijvoorbeeld niet besproken omdat je hiermee geen limitaties kunt opleggen voor geheugen- en cpugebruik.

LXC LXC of Linux Containers maakt gebruik van cgroups (control groups) en namespaces om OS-level virtualisatie toe te laten. LXC maakt dus gebruik van de Linux kernel om virtualisatie aan te bieden.

Docker Docker gebruikt LXC om virtualisatie aan te bieden en is geprogrammeerd in Go [6]. Het voordeel van Docker is dat er een grote repository bestaat met beschikbare templates zodat je zelf geen LXC template meer hoeft te maken [5].

FreeBSD Jails Jails kunnen gezien worden als een uitbreiding van de mogelijkheden van een chroot en een verbetering van de veiligheid die chroots aanbieden [1].

OpenVZ Het OpenVZ project werd gestart in 2005 en bevat heel veel features om de virtuele omgeving te controleren. In tegenstelling tot LXC maakt deze niet altijd gebruik van de Linux Kernel voor de virtualisatie.

Onze keuze valt op OpenVZ voornamelijk omdat de andere opties afvallen voor één of andere reden. LXC bijvoorbeeld is momenteel druk bezig met de release candidate van versie 1.0.0. [2]. Hierdoor lijkt het ons ongeschikt om deze al in een omgeving te gebruiken waar beveiliging een prioriteit is. Hetzelfde geldt eigenlijk voor Docker dat momenteel op versie 0.8 zit [3] en zichzelf nog niet klaar beschouwt voor productie [4]. We kozen ook niet voor Jails omdat niemand ervaring heeft met BSD en omdat we spijtig genoeg niet al teveel tijd hebben om dit eindwerk te maken en we liever kiezen voor een Linux oplossing omdat we dan naar alle waarschijnlijk minder tijd gaan besteden aan futiliteiten.

2.1.2 Praktisch

Als onze website de code toegestuurd krijgt, stuurt deze de code en wat metadata (compile flags, userid) door naar een interne daemon. Deze daemon (initieel geschreven in Perl) ontvangt de data en start een nieuwe thread op die een VE opstart, de code compileert en runt en daarna het resultaat wegschrijft naar de DB.

Indien er genoeg tijd is kan er gewerkt worden met een queue van idle VE's die gebruikt kunnen worden door de threads wat het gehele plaatje aanzienlijk zou moeten versnellen. De code kan ook best naar een lagere taal herschreven worden, C bijvoorbeeld.

Bibliografie

[1] FreeBSD. Chapter 15. jails - introduction. Beschikbaar op `http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails-intro.html`, 2014. Bezocht op 16/02/2014.

[2] Stéphane Graber. Daily builds of lxc. Beschikbaar op `https://launchpad.net/~ubuntu-lxc/+archive/daily`, 2014. Bezocht op 15/02/2014.

[3] Solomon Hykes. Docker 0.8: Quality, new builder features, btrfs, osx support. Beschikbaar op `http://blog.docker.io/2014/02/docker-0-8-quality-new-builder-features-btrfs-storage-osx-support/`, 2014. Bezocht op 15/02/2014.

[4] Docker Inc. About docker. Beschikbaar op `https://www.docker.io/learn_more/`, 2014. Bezocht op 15/02/2014.

[5] Docker Inc. Docker index. Beschikbaar op `https://index.docker.io/`, 2014. Bezocht op 15/02/2014.

[6] Docker Inc. dotcloud/docker. Beschikbaar op `https://github.com/dotcloud/docker`, 2014. Bezocht op 15/02/2014.