Using information flow tracking to protect legacy binaries

Slowinska, J.M.

2012

document version
Publisher's PDF, also known as Version of record

Samenvatting

Ondanks dat aanvallen gebaseerd op geheugencorruptie inherent zijn aan het gebruik van de programmeertaal C, is het onwaarschijnlijk dat deze taal snel vervangen zal worden door sterk getypeerde talen met veilige geheugenbeheergaranties [76; 200; 30]. En juist het gebrek aan deze garanties is vaak de oorzaak van serieuze beveiligingsproblemen. Dagelijks [180; 150; 137] worden er meldingen gemaakt van geheugencorruptie-gerelateerde-fouten en zien we aanvallen die veel gebruikte software en kritieke netwerken infecteren [123; 198].

Vanuit het onderzoeksveld zijn verschillende technieken voorgesteld om het probleem aan te pakken. De effectiviteit van deze technieken op productiesoftware blijkt echter beperkt. Oplossing die wel effectief genoeg blijken, zijn vaak niet efficiënt en/of niet toepasbaar op oudere softwaresystemen.

In dit proefschrift onderzoeken wij het probleem van het beschermen van oudere software, geschreven in de programmeer taal C, die vatbaar zijn voor geheugencorruptie aanvallen. We richten ons hierbij op technieken die informatiestromen volgen binnen een programma. Deze technieken zijn aantrekkelijk vanwege de toepasbaarheid op bestaande software en het nauwkeurige inzicht dat ze opleveren in het gedrag van de software terwijl deze draait. Helaas is de toepasbaarheid van de betreffende technieken vaak beperkt tot systemen die niet in productie worden ingezet of omgevingen om malafide software te bestuderen. De reden is namelijk dat de technieken vaak erg traag zijn. In dit proefschrift wordt gezocht naar een balans tussen de hoeveelheid beschikbare informatie die deze technieken opleveren de snelheid waarbij gestreefd wordt naar resultaten binnen een acceptabele tijdsperiode.

De geheugencorrumperende aanvallen kunnen worden onderverdeeld in twee categorieën: (1) aanvallen die de besturingsstroom van het programma zelf veranderen, en waarbij de executie van een programma wordt beïnvloed om geïnjecteerde code van de aanvaller uit te voeren, en (2) aanvallen die niet de besturingsstroom veranderen, maar die in plaats daarvan andere waarden in het geheugen veranderen, zoals de privileges van de gebruiker of een server configuratieparameter.

De onderzoeksgemeenschap heeft het gebruik van technieken om informatiestromen in binaire programma's voorgesteld om bescherming te bieden tegen beide categorieën aanvallen. Een populaire techniek om informatiestromen te volgen is dynamische smet analyse [62; 149], of, in het Engels, "dynamic taint analysis" (DTA).

DTA wordt met succes toegepast om besturingsstroomcompromitterende aanvallen te detecteren. Omdat de gevolgen van aanvallen die de besturingsstroom niet wijzigen net zo ernstig zijn, is er door verschillende projecten getracht een afgeleide techniek van DTA, "limited pointer tainting", toe te passen om beide categorieën te detecteren. Een uitgebreidere versie van limited pointer tainting, "full pointer tracking", heeft toepassingen in het opsporen van informatielekken, zoals het volgen van toetsaanslagen in een computersysteem om keyloggers te detecteren.

Deel 1 Het eerste deel van dit proefschrift beschrijft DTA. We beginnen met het evalueren van de twee genoemde pointer tainting technieken. In de evaluatie kijken we naar de effectiviteit hiervan om niet besturingsstroomcompromitterende aanvallen en keyloggers te detecteren. Vervolgens breiden we DTA uit om aanvallen te analyseren nadat deze zijn gedetecteerd.

In hoofdstuk 3 voeren we een uitgebreide analyse uit naar de problematiek behorende bij pointer tainting technieken op productiesystemen. In deze analyse laten we zien dat deze technieken niet effectief zijn bij de detectie en analyze van malafide programma's die gevoelige informatie van gebruikers stelen en daarnaast ook beperkingen kennen bij andere toepassingen. Het blijft een open probleem om een vorm van pointer tainting succesvol toe te passen op de meest populaire PC architectuur (`x86`) en besturingssysteem (Windows).

In hoofdstuk 4 presenteren we Prospector, een emulator die kan bijhouden welke bytes deelnemen aan een buffer-overschrijvingsaanval (buffer overflow) op de heap of de stack. Wanneer we het protocol herkennen dat het malafide netwerkbericht bevat, dan kunnen we met Prospector een signalement maken dat verschillende vormen van de aanval detecteert. Bij het genereren van het signalement kijken we naar de lengtes van de protocolvelden, in plaats van naar de inhoud. In het gebruik van deze signalementen zijn de aantallen valse meldingen verwaarloosbaar en de aantallen gemiste aanvallen laag, terwijl de signalementen effectief zijn in het filteren van malafide verkeer.

Verder bespreken we in hoofdstuk 5 een nieuw honingpotsysteem ("honeypot"), Hassle. Met Hassle is het mogelijk om signalementen te genereren van aanvallen die plaatsvinden via versleutelde kanalen. De gebruikte techniek plaatst analyse-code tussen het programma en de gebruikte versleutelingsfuncties in bibliotheken, waardoor bijna alle versleutelschema's ondersteund worden zonder dat het programma aangepast hoeft te worden.

Deel 2 In het tweede gedeelte van dit proefschrift kijken we naar technieken die toepasbaar zijn op aanvallen die data wijzigen zonder de besturingsstroom te veranderen. We bespreken een compleet nieuwe oplossing genaamd BodyArmour die het mogelijk maakt om oudere programma's te beschermen tegen beide soorten buffer overflows: zowel de aanvallen die de besturingsstroom veranderen als de aanvallen die weliswaar data corrumperen, maar niet de besturingsstroom. Kort gezegd

volgt BodyArmour de uitvoering van een beschermd programma en zorgt ervoor dat een verwijzing naar een stuk geheugen nooit naar een ander, niet overlappend, stuk geheugen kan verwijzen. Deze simpele regel is effectief tegen beide soorten overflowaanvallen. Ondanks de eenvoud van de regel is de toepassing ervan niet triviaal. Om de regel toe te kunnen passen is veel kennis nodig over het te beschermen programma. Te denken valt aan de locaties van stukken geheugen, de verwijzingen naar deze stukken geheugen en de programmacode die deze stukken geheugen aanspreekt. Daarnaast is er voor de meerderheid van de te beschermen programma's geen symbolische informatie (ook wel bekend als "symbols"), beschikbaar over de gebruikte datastructuren. Als eerste bespreken we daarom hoe wij informatiestromen volgen om deze datastructuren te achterhalen.

In hoofdstuk 6 presenteren we Howard, een framework waarmee wij de gebruikte datastructuren kunnen onttrekken aan programma's waarbij deze informatie niet beschikbaar is gesteld. Howard ontdekt deze datastructuren door te zoeken naar patronen in de benadering van het geheugen door een programma. De doelstelling van Howard is het beschikbaar maken van deze datastructuren aan BodyArmour, maar daarnaast demonstreren wij nog twee andere toepassingen. In de eerste toepassing maken we de gevonden informatie beschikbaar aan bestaande disassemblers en debuggers om zo het reverse engineeren van programma's te vergemakkelijken. In de tweede toepassing laten we zien hoe de informatie gebruikt kan worden om dynamische datastructuren te vinden in programma's. Bij dynamische datastructuren kunt u denken aan lineaire lijsten en binaire bomen.

In hoofdstuk 7 presenteren wij BodyArmour, een toepassing om programma's te beschermen waarvan geen broncode of informatie over de datastructuren beschikbaar is. Gebruikmakende van deze oplossing is het mogelijk om applicaties te beschermen tegen buffer overflows voordat ze gebeuren en voordat we weten of ze ook daadwerkelijk mogelijk zijn. Naast besturingsstroomcompromitterende aanvallen stopt BodyArmour ook niet-besturingsstroomcompromitterende aanvallen. BodyArmour kan werken op twee manieren, aangeduid met BA-veld en de BA-object. Als BodyArmour werkt volgens de BA-veld wijze, dan worden individuele velden in een datastructuur beschermd. Met deze manier is BodyArmour veel preciezer dan huidige oplossingen. Het foutief detecteren van aanvallen in geval van de BA-veld wijze (een valse melding) is mogelijk, maar zeer onwaarschijnlijk. Tijdens onze evaluatie zijn we geen foutieve detecties tegengekomen. Wanneer BodyArmour alleen hele datastructuren beschermd, ingeval van de BA-object wijze, dan is het niet mogelijk om een aanval verkeerd te detecteren. Uit onze evaluatie van BodyArmour concluderen wij dat het een variatie aan verschillende aanvallen zonder al grote vertraging kan voorkomen.