Framing Signals—Sự trở lại của Portable Shellcode

Erik Bosman
Vrije Universiteit
Amsterdam
erik@minemu.org

Herbert Bos
Vrije Universiteit
Amsterdam
herbertb@cs.vu.nl

Tóm tắt—Xử lý tín hiệu đã là một phần không thể thiếu của các hệ thống UNIX kể từ lần triển khai đầu tiên vào những năm 1970.
Ngày nay, chúng tôi tìm thấy tín hiệu trong tất cả các hư ơ ng vị phổ biến của hệ thống UNIX, bao gồm BSD, Linux, Solaris, Android và Mac OS.
Mặc dù mỗi hư ơ ng vị xử lý tín hiệu theo những cách hơ i khác nhau, như ng cách triển khai rất giống nhau. Trong bài báo này, chúng tôi chỉ ra rằng việc xử lý tín hiệu có thể đư ợc sử dụng như một phư ơ ng pháp tấn công trong khai thác và cửa hậu. Vấn đề đã là một phần của UNIX ngay từ đầu và giờ đây, các biện pháp bảo mật tiên tiến như ASLR, DEP và cookie ngăn xếp đã khiến việc khai thác đơ n giản trở nên khó khăn hơ n nhiều, kỹ thuật của chúng tôi là một trong những kết quả treo thấp nhất có sẵn cho kẻ tấn công.

Cụ thể, chúng tôi mô tả Lập trình định hư ớng Sigreturn (SROP), một kỹ thuật mới để khai thác và mở cửa hậu trong các hệ thống giống như UNIX. Giống như lập trình hư ớng trở lại (ROP), lập trình hư ớng sigreturn xây dựng cái đư ợc gọi là 'máy lạ' có thể đư ợc lập trình bởi những kẻ tấn công để thay đổi hành vi của một quy trình. Để lập trình máy, những kẻ tấn công thiết lập các khung tín hiệu giả và bắt đầu trả về từ các tín hiệu mà nhân không bao giờ thực sự đư ợc gửi đi. Điều này là có thể, bởi vì UNIX lư u trữ các khung tín hiệu trên ngăn xếp của tiến trình.

Lập trình theo định hư ớng Sigreturn rất thú vị đối với những kẻ tấn công, nhà phát triển hệ điều hành và học giả. Đối với những kẻ tấn công, kỹ thuật này rất linh hoạt, với các điều kiện tiên quyết khác với các kỹ thuật khai thác hiện có như ROP. Hơ n nữa, không giống như ROP, các chư ơ ng trình lập trình hư ớng sigreturn có thể mang theo đư ợc. Đối với các nhà phát triển hệ điều hành, kỹ thuật này đư a ra một vấn đề đã xuất hiện ở một trong hai họ hệ điều hành chính ngay từ khi mới thành lập, trong khi các bản sửa lỗi (mà chúng tôi cũng trình bày) là không tầm thư ờng. Từ quan điểm học thuật hơ n, nó cũng thú vị bởi vì chúng tôi chỉ ra rằng lập trình hư ớng sigreturn là Turing hoàn chỉnh.

Chúng tôi chứng minh tính hữu dụng của kỹ thuật này trong ba ứng dụng. Đầu tiên, chúng tôi mô tả việc khai thác một máy chủ web có lỗ hổng trên các bản phân phối Linux khác nhau. Thứ hai, chúng tôi xây dựng một cửa hậu chứng minh khái niệm rất lén lút. Thứ ba, chúng tôi sử dụng SROP để bỏ qua quy trình kiểm tra bảo mật và ký mã của Apple bằng cách xây dựng một ứng dụng có thể thực hiện các cuộc gọi hệ thống tùy ý. Cuối cùng, chúng tôi thảo luận về các kỹ thuật giảm thiểu.

I. GIỚI THIỆU

Xử lý tín hiệu đã là một phần không thể thiếu của các hệ thống UNIX (và giống như UNIX) kể từ lần triển khai đầu tiên của Dennis Ritchie vào đầu những năm 1970. Tín hiệu là một cơ chế cực kỳ mạnh mẽ để gửi thông báo không đồng bộ trực tiếp đến một quy trình hoặc luồng. Chúng đư ợc sử dụng để giết các tiến trình, để báo cho chúng biết rằng bộ hẹn giờ đã hết hạn hoặc để thông báo cho chúng về hành vi đặc biệt. Thiết kế UNIX đã sinh ra rất nhiều "đứa con" giống UNIX trong đó GNU Linux, một số hư ơ ng vị của BSD, Android, iOS/Mac OS X và Solaris có lẽ là những thứ đư ợc biết đến nhiều nhất đang đư ợc sử dụng tích cực ngày nay. Mặc dù mỗi hư ơ ng vị xử lý tín hiệu theo những cách hơ i khác nhau, như ng các triển khai khác nhau đều rất giống nhau.

Chúng tôi chỉ ra rằng việc triển khai có thể đư ợc sử dụng như một phư ơ ng pháp tấn công trong khai thác và cửa hậu, giống như lập trình hư ớng trả về (ROP [24])—mặc dù kỹ thuật này khác. Vấn đề đã tồn tại, theo hiểu biết tốt nhất của chúng tôi mà chư a đư ợc khám phá, đã 40 năm rồi. Hơ n nữa, giờ đây các biện pháp bảo mật tiên tiến như ASLR, DEP và cookie ngăn xếp đang khiến việc khai thác đơ n giản trở nên khó khăn hơ n nhiều, nó có thể đư ợc xếp vào loại quả treo thấp nhất dành cho kẻ tấn công.

Theo truyền thống của 'máy kỳ lạ' [7], chúng tôi mô tả một kỹ thuật để thực thi mã do kẻ tấn công cung cấp trong các tệp nhị phân lành tính. Tuy nhiên, thay vì thực thi shellcode trực tiếp, quay trở lại thư viện C hoặc ghép một chư ơ ng trình bằng ROP, chúng tôi xây dựng cỗ máy kỳ lạ của mình bằng cách trả về giả mạo từ các tín hiệu. Kỹ thuật mà chúng tôi gọi là lập trình hư ớng sigreturn là chung chung và chúng tôi có thể sử dụng nó cho cả khai thác, cửa hậu và proxy gọi hệ thống. Hơ n nữa, chúng tôi chứng minh rằng lập trình hư ớng sigreturn đã hoàn thành Turing.

Ý tư ởng chính đằng sau lập trình hư ớng sigreturn là kẻ tấn công có thể lạm dụng cách mà hầu hết các hệ thống UNIX trả về từ bộ xử lý tín hiệu.

Khi hạt nhân cung cấp tín hiệu, nó sẽ tạm dừng quá trình thực thi bình thư ờng của quy trình và thay đổi ngữ cảnh CPU của không gian ngư ời dùng sao cho trình xử lý tín hiệu thích hợp đư ợc gọi với các đối số phù hợp. Khi trình xử lý tín hiệu này quay trở lại, bối cảnh CPU không gian ngư ời dùng ban đầu đư ợc khôi phục. Cụ thể, một chư ơ ng trình trả về từ trình xử lý bằng cách sử dụng sigreturn, một 'lệnh gọi hệ thống ẩn' trên hầu hết các hệ thống giống UNIX, đọc một khung tín hiệu (struct sigframe) từ ngăn xếp, đư ợc đặt ở đó bởi hạt nhân khi gửi tín hiệu. Khung chứa tất cả thông tin cần thiết để trả về an toàn: giá trị của thanh ghi, con trỏ ngăn xếp, cờ, v.v.

Vấn đề là bất kỳ ai kiểm soát ngăn xếp đều có thể thiết lập khung tín hiệu như vậy. Bằng cách gọi sigreturn, kẻ tấn công có thể xác định trạng thái tiếp theo cho chư ơ ng trình, và như chúng ta sẽ thấy, xâu chuỗi sigreturn và các lệnh gọi hệ thống khác lại với nhau và thực thi mã tùy ý.

Giống như lập trình hư ớng trở lại (ROP), sigreturn ori-

ented programming (SROP) là một kỹ thuật chung và chúng tôi sẽ chỉ ra cách chúng tôi đã sử dụng kỹ thuật này trong khai thác, cửa hậu và proxy cuộc gọi hệ thống cũng như trên nhiều hệ điều hành khác nhau. So với ROP, nó có các điều kiện tiên quyết khác nhau mà trong một số trường hợp sẽ dễ đáp ứng hơn. Chẳng hạn, SROP chỉ cần một tiện ích duy nhất để khai thác và đối với một số bản phân phối Linux, Android và BSD, tiện ích đó luôn hiện diện và tốt hơn nữa là luôn nằm ở một vị trí cố định.
Trong trường hợp đó, những kẻ tấn công thậm chí không cần biết trước phiên bản chính xác của tệp thực thi và tất cả các thư viện, điều này giúp đơn giản hóa đáng kể cuộc tấn công khi không thể trinh sát chi tiết.

Đối với những kẻ tấn công, điều đặc biệt hấp dẫn về SROP so với ROP là khả năng tái sử dụng của nó. Không giống như mã ROP, các chương trình SROP không phụ thuộc nhiều vào nội dung của tệp thực thi. Giống như shellcode truyền thống, điều này cho phép sử dụng lại cùng một mã SROP trên nhiều ứng dụng.
Hơn nữa, kỹ thuật này hoạt động trên các kiến trúc tập lệnh và hệ điều hành rất khác nhau. Ví dụ: chúng tôi đã triển khai SROP thành công trên x86 32 bit và 64 bit, cũng như trên bộ xử lý ARM. Tương tự như vậy, chúng tôi đã thử nghiệm thành công kỹ thuật này trên Linux (các bản phân phối khác nhau), Android, FreeBSD, OpenBSD và Mac OS X/iOS của Apple. Chương trình mà chúng tôi sử dụng để chứng minh tính hoàn thiện của lập trình hướng sigreturn của Turing hoạt động trên Linux 32 bit và 64 bit.

Đóng góp: Trong bài báo này, chúng tôi sẽ mô tả một loại máy lạ mới và giải thích cách lập trình nó bằng lập trình hướng sigreturn. Cụ thể, chúng tôi giới thiệu: 1) một kỹ thuật khai thác chung mới,
được gọi là lập trình hướng ngược si (SROP), trong một số trường hợp không yêu cầu kiến thức trước về ứng dụng nạn nhân và tạo ra 'shellcode' có thể tái sử dụng; 2) một kỹ thuật backdoor mới và lén lút dựa trên

SROP;
3) proxy cuộc gọi hệ thống để vượt qua bảo mật iOS của Apple
người
mẫu; 4) bằng chứng rằng SROP đã hoàn thành
Turing; 5) các kỹ thuật giảm thiểu khả thi.
Ứng dụng: Chúng tôi chứng minh tính thực tế của kỹ thuật khai thác của mình bằng cách sử dụng một lỗ hổng được tìm thấy gần đây trong máy chủ web Asterisk. Chúng tôi cho thấy rằng bằng cách sử dụng SROP, chúng tôi có thể xây dựng một khai thác duy nhất hoạt động cho các phiên bản khác nhau của cùng một chương trình trên các bản phân phối khác nhau của Linux, bao gồm Debian Wheezy (phát hành vào tháng 5 năm 2013), Ubuntu LTS 12.04 (phiên bản Hỗ trợ dài hạn mới nhất của Ubuntu, được phát hành vào năm 2012 và được hỗ trợ trong 5 năm) và Centos 6.3 (được phát hành vào năm 2012 và được hỗ trợ trong 10 năm).

Sau đó, chúng tôi chứng minh khả năng ứng dụng thực tế rộng rãi hơn của kỹ thuật này bằng một cửa hậu tàng hình. Cửa hậu rất khó tìm thấy ngay cả với công cụ pháp y hiện đại. Ví dụ: ngay cả một kết xuất bộ nhớ hoàn chỉnh của quy trình cũng sẽ không tiết lộ bất kỳ hướng dẫn cửa hậu nào, vì tất cả logic được ẩn trong dữ liệu dưới dạng chương trình SROP. Cuối cùng, chúng tôi chỉ ra cách chúng tôi có thể bỏ qua mô hình bảo mật nổi tiếng của Apple bao gồm kiểm tra mã phức tạp và ngăn các ứng dụng độc hại xâm nhập vào App Store. Trong trường hợp của chúng tôi, không cần thêm bất kỳ hướng dẫn độc hại nào vào ứng dụng, vì vậy sẽ cực kỳ khó bị phát hiện. Tuy nhiên, được cung cấp các đầu vào phù hợp, nó sẽ hoạt động như một proxy cuộc gọi hệ thống có thể được lập trình bằng lập trình hướng sigreturn.

Outline: Phần còn lại của bài viết này được tổ chức như sau. Trong Phần II, chúng tôi đặt SROP trong bối cảnh công việc liên quan. Trong Phần III, chúng ta thảo luận về những cỗ máy kỳ lạ và cách lập trình chúng, cũng như vai trò của lập trình hướng về. Xử lý tín hiệu là chủ đề của Phần IV.
Phần V thảo luận chi tiết về kỹ thuật SROP. Chúng tôi thảo luận về kỹ thuật khai thác của chúng tôi trong Phần VI, một cửa hậu lén lút trong Phần VII và proxy cuộc gọi hệ thống iOS của chúng tôi trong Phần VIII. Tính đầy đủ của Turing sẽ được thảo luận trong Phần IX. Trong Phần X, chúng tôi thảo luận về các kỹ thuật giảm thiểu khả thi. Cuối cùng, chúng tôi kết thúc ở Phần XI.

II. CÔNG VIỆC LIÊN QUAN

Công việc của chúng tôi phù hợp với danh mục chung của những cỗ máy kỳ lạ. Thuật ngữ cỗ máy kỳ lạ ban đầu được đặt ra bởi Sergey Bratus và nhanh chóng được các nhà nghiên cứu khác chọn [7], [30]. Đó là một thuật ngữ chung để mô tả các hệ thống trong đó có thể thực hiện được các tính toán mạnh mẽ không mong muốn hoặc không mong muốn. Ví dụ, các nhà nghiên cứu đã chỉ ra rằng có thể sử dụng các cỗ máy kỳ lạ được xây dựng từ các bộ phận phức tạp, chẳng hạn như logic di chuyển ký hiệu ELF trong bộ tải động [26] và thậm chí cả hệ thống con bộ nhớ ảo x86 [9].

Đối với bài báo này, Lập trình hướng trở lại (ROP) là phù hợp nhất [24]. ROP là một kỹ thuật khai thác cho phép kẻ tấn công thực thi mã khi có các biện pháp bảo mật như ngăn xếp không thể thực thi và ký mã. Là điều kiện tiên quyết để khai thác ROP thành công, kẻ tấn công cần giành quyền kiểm soát ngăn xếp và lấy các con trỏ mã hợp lệ.

Sau đó, kẻ tấn công sẽ thao túng địa chỉ trả về để chuyển đến một chuỗi các hướng dẫn kết thúc bằng việc trả về. Khi kẻ tấn công kiểm soát ngăn xếp, cô ấy có thể tiếp tục nhảy tới các tiện ích viết mã và do đó xâu chuỗi một chương trình lại với nhau.

Một biến thể của kỹ thuật tương tự, được gọi là lập trình hướng nhảy, sử dụng các bước nhảy thay vì trả về [3]. Việc tìm kiếm các tiện ích thích hợp và xâu chuỗi chúng lại với nhau rất khó, nhưng các nhà nghiên cứu đã chỉ ra rằng có thể xây dựng các trình biên dịch ROP [23], [13] để thực hiện việc này dễ dàng hơn. Tương tự, các cơ chế bảo vệ hiện đại như ngẫu nhiên hóa không gian địa chỉ [20], [1], [25] khiến việc tìm địa chỉ của các đoạn mã trở nên khó khăn. Mặt khác, được cung cấp một con trỏ mã ban đầu, thường có thể trích xuất các con trỏ mã còn lại cần thiết để khai thác thành công [27]. Ngoài ra, trong Linux và Windows, các tệp thực thi có địa chỉ tải cố định và các thư viện không tương thích với ASLR, có thể dẫn đến các phần của không gian địa chỉ không đổi [22].

Vì ROP và JOP đã nhanh chóng trở nên phổ biến với những kẻ tấn công, nhiều nhóm nghiên cứu đã cố gắng giảm thiểu các vấn đề, chẳng hạn bằng cách cố gắng xóa các tiện ích hữu ích [11], [17], hoán vị thứ tự các chức năng [2], [10], hoặc thiết bị nhị phân dy namic [6], [5]. Ngoài ra, các nhà nghiên cứu đã đề xuất sử dụng mã ngẫu nhiên tại chỗ với chi phí thấp [19] và bằng cách theo dõi lịch sử chi nhánh [18]. Mặc dù không có giải pháp nào trong số này ngăn chặn những kẻ tấn công sử dụng ROP, như ng chúng vẫn tiếp tục nâng cao tiêu chuẩn. Nhiều biện pháp giảm thiểu không hoạt động đối với SROP. Chẳng hạn, các tiện ích đư ợc SROP sử dụng là cần thiết cho hoạt động của tệp nhị phân và không thể xóa đư ợc. Chúng tôi tin rằng SROP có thể là một trong những phư ơ ng pháp tấn công thuận tiện nhất ngay cả đối với các chư ơ ng trình áp dụng tất cả các biện pháp bảo mật thông thư ờng.

Ở mức độ hời hợt, lập trình hư ớng sigreturn chia sẻ một số đặc điểm của ROP. Chúng ta sẽ thấy rằng lập trình hư ớng sigreturn cũng yêu cầu thiết lập các giá trị thích hợp trên ngăn xếp và 'quay lại' giá trị do kẻ tấn công xác định. Như ng kỹ thuật thì khác.

Cụ thể, SROP sử dụng các khung tín hiệu giả và không thực sự phụ thuộc vào việc tìm kiếm các tiện ích và xâu chuỗi chúng lại với nhau.
Cũng vì lý do này, mã SROP có khả năng sử dụng lại tốt hơ n.

Việc sử dụng chính các tín hiệu trong việc khai thác đã đư ợc Michael Zalewski ghi chép kỹ lư ỡng trong: "Delivering Signals for fun and Profit" [32]. Tác giả chỉ ra rằng có nhiều cạm bẫy cần xem xét khi lập trình bộ xử lý tín hiệu. Bản chất ư u tiên của các tín hiệu có thể dẫn đến việc sử dụng cấu trúc dữ liệu khi chúng ở trạng thái không nhất quán, làm hỏng chúng trong quá trình hoặc khiến ứng dụng thực hiện những việc không mong muốn khi ở trạng thái có quyền cao.

Công việc của chúng tôi khác, ở chỗ chúng tôi sử dụng các khung tín hiệu giả như một phần hư ớng dẫn trong một cỗ máy kỳ lạ.

Cuộc gọi không gian ngư ời dùng tư ơ ng tự như cuộc gọi hệ thống sigreturn là longjmp. Trong "Bỏ qua bộ bảo vệ ngăn xếp và lá chắn ngăn xếp" [4], việc ghi đè một con trỏ tới ngữ cảnh ngư ời dùng sau đó đư ợc sử dụng bởi longjmp đư ợc đề cập như một véc tơ khai thác. Tuy nhiên, cần lư u ý rằng sự tồn tại của một con trỏ như vậy trong bối cảnh khai thác là cực kỳ hiếm.

Microsoft Windows không triển khai tín hiệu POSIX.
Cơ chế xử lý lỗi của nó đư ợc thiết kế để tích hợp tốt với xử lý ngoại lệ của C++. Thông qua một cơ chế gọi là Xử lý ngoại lệ có cấu trúc (SEH), nó cho phép các chức năng giải phóng ngăn xếp và thực hiện một số xử lý cho đến khi ngoại lệ đư ợc bắt gặp trong khung ngăn xếp trư ớc đó. Khi các con trỏ của trình xử lý ngoại lệ đư ợc đặt trên ngăn xếp, đây là mục tiêu ghi đè thư ờng xuyên đối với lỗi tràn bộ đệm ngăn xếp trên Windows [12] và một số kỹ thuật giảm thiểu đã đư ợc đề xuất và triển khai [15]. Tuy nhiên, vì SEH trên Windows không quay trở lại trạng thái trư ớc khi trình xử lý đư ợc thực thi nên chúng không thể cung cấp cơ chế cho các khai thác giống như SROP.

Jekyll [31] cung cấp một nỗ lực độc lập để tấn công các thiết bị iOS vốn cũng dựa trên các ứng dụng lành tính trở thành xấu xa. Giống như công việc của chúng tôi, các tác giả cố tình đư a ra các lỗ hổng trong ứng dụng để họ có thể dễ dàng thay đổi luồng điều khiển sau, bằng cách khai thác nó bằng phư ơ ng tiện khai thác ROP. Tư ơ ng tự như vậy, chúng tôi cho thấy rằng giờ đây có thể dễ dàng vư ợt qua tất cả các cơ chế phòng thủ của iOS, bao gồm DEP [14], ALSR [20], [1], hộp cát, đánh giá ứng dụng và ký mã. Không giống như Jekyll cần mang theo các tiện ích của riêng mình, tiện ích dành cho lập trình định hư ớng sigreturn đã có sẵn. Do đó, ngay cả các kỹ thuật phát hiện quét cụ thể chức năng như Jekyll cũng không còn hiệu quả. Cuối cùng, chúng ta có thể sử dụng lập trình hư ớng sigreturn trong kịch bản hậu khai thác. Chẳng hạn, chúng ta có thể sử dụng proxy cuộc gọi hệ thống để bẻ khóa—cho phép kẻ tấn công bẻ khóa thông qua quy trình gốc không phải do chúng viết.

Trong phần tiếp theo, chúng ta sẽ xem xét lại xu hư ớng hiện tại của những kẻ tấn công là chuyển từ việc tiêm shellcode thông thư ờng sang các cuộc tấn công dựa trên việc sử dụng lại mã. Chúng tôi sẽ lập luận rằng vì điều này, việc khai thác ngày càng khó khăn hơ n và SROP có thể là một vũ khí mới hữu ích trong tay kẻ tấn công— loại bỏ một số trở ngại mà kẻ tấn công gặp phải trong các hệ thống hiện đại.

III. VỀ VIỆC CHẾ TẠO MÁY KỲ LẠ (HOẶC: TẠI SAO KHAI THÁC ĐANG KHÓ HƠ N)

Lập trình hư ớng Sigreturn không giống như các kỹ thuật khai thác khác, nơ i những kẻ tấn công thực thi 'mã' nư ớc ngoài trong một chư ơ ng trình hiện có. Cách đây không lâu, làm như vậy là tư ơ ng đối đơ n giản. Thông thư ờng, những kẻ tấn công đã tìm đư ợc lỗ hổng tràn bộ đệm trên ngăn xếp cho phép chúng ghi đè lên địa chỉ trả về. Tất cả những gì họ cần làm là đặt địa chỉ trả về trỏ tới mã máy của chính họ, mã này sẽ đư ợc lư u trữ trong bộ đệm hoặc biến môi trư ờng. Ngay sau khi hàm quay trở lại, chư ơ ng trình sẽ tiếp tục thực thi mã của kẻ tấn công.

1) Các cuộc tấn công sử dụng lại mã (ROP, JOP và ret-into-libc):
Với các biện pháp bảo vệ hiện đại, việc khai thác các lỗi hỏng bộ nhớ truyền thống như vậy, trong đó mã máy đư ợc kẻ tấn công đư a vào và thực thi trực tiếp, đã trở nên hiếm. Cụ thể, các tính năng ngăn chặn thực thi dữ liệu (DEP [14]), hiện diện trong thực tế tất cả các CPU hiện đại và hệ điều hành, mã máy riêng biệt và dữ liệu có thể ghi. Nói cách khác, những kẻ tấn công không còn có thể thực thi trực tiếp shellcode của chúng nữa. Để vư ợt qua các biện pháp bảo mật như vậy, họ phải sử dụng các kỹ thuật khai thác khác nhau. Thay vì tiêm mã máy thông thư ờng, các cuộc tấn công hiện đại thư ờng sử dụng lại logic hoặc mã từ chính tệp thực thi bằng cách sử dụng ret-to-libc [28] hoặc lập trình hư ớng trả về [24]. Bằng cách này, những kẻ tấn công có thể xây dựng các ô tô lạ để thực hiện đấu thầu của chúng.

Xây dựng một máy tự động như vậy là không dễ dàng. Trong vài năm qua, cộng đồng nghiên cứu bảo mật đã đặc biệt tập trung vào các kỹ thuật chung như lập trình hư ớng trở lại (ROP) [24] và lập trình hư ớng nhảy (JOP) [3].

Các kỹ thuật này sử dụng các đoạn mã thực thi có trong chính chư ơ ng trình. Đư ợc xâu chuỗi bởi các hư ớng dẫn luồng điều khiển gián tiếp, các đoạn này kết hợp với nhau thành một chuỗi các hư ớng dẫn cấp thấp để thực hiện những gì kẻ tấn công muốn.

Kẻ tấn công kiểm soát ngăn xếp của ứng dụng do lỗi tràn bộ đệm có thể tìm trong tệp nhị phân của ứng dụng ban đầu để tìm các chuỗi hư ớng dẫn nhỏ thực hiện điều gì đó thú vị và kết thúc bằng lệnh quay lại. Chẳng hạn, một trình tự để thêm hai thanh ghi, hoặc để tải hoặc lư u trữ một thanh ghi, v.v. Những trình tự này đư ợc gọi là 'tiện ích'. Nếu kẻ tấn công quản lý để chuyển hư ớng luồng điều khiển sang một trong các tiện ích, thì tiện ích đó sẽ thực thi phần nhỏ đầu tiên trong mã của kẻ tấn công và sau đó thực hiện trả về. Vì kẻ tấn công kiểm soát ngăn xếp, nên địa chỉ để quay lại cũng nằm dư ới sự kiểm soát của kẻ tấn công. Vì vậy, kẻ tấn công quay trở lại một tiện ích khác. Bằng cách xâu chuỗi các tiện ích lại với nhau, kẻ tấn công có thể thực thi mã tùy ý, đư ợc viết cho một tập lệnh bao gồm các tiện ích và con trỏ ngăn xếp có chức năng như một loại bộ đếm chư ơ ng trình kỳ lạ.

Mặc dù quy trình như vậy nghe có vẻ đơ n giản như ng việc khai thác ROP đang hoạt động thư ờng rất phức tạp. Chẳng hạn, một cuộc tấn công ROP có thể yêu cầu kẻ tấn công thao túng trạng thái của chư ơ ng trình để chuẩn bị cho việc khai thác, bằng cách (1) làm cho nó rò rỉ thông tin về các con trỏ mã (ví dụ: bằng tấn công chuỗi định dạng), (2) sắp xếp các đối tư ợng heap theo một cách cụ thể để mở đư ờng cho việc khai thác con trỏ lơ lửng (sử dụng phong thủy heap tiên tiến [29]), (3) thu thập đủ tiện ích mã hữu ích từ tệp nhị phân để xâu chuỗi chư ơ ng trình cuối cùng của kẻ tấn công (thư ờng giả định rằng các phiên bản chính xác của tệp thực thi và thư viện đã đư ợc biết) và (4) định vị dữ liệu thích hợp trên ngăn xếp và (5) chuyển hư ớng điều khiển sang tiện ích đầu tiên.

Khả năng của một chư ơ ng trình vư ợt ra ngoài đặc điểm kỹ thuật của nó cho phép nó hoạt động như một cỗ máy kỳ lạ [7], [30]. Máy này sau đó có thể bị thao túng, lập trình bởi kẻ tấn công giống như bất kỳ máy lập trình nào khác. Đầu vào của kẻ tấn công giờ đóng vai trò là mã máy (khá đặc biệt) của nó, cho phép cô ta kiểm soát việc thực thi chư ơ ng trình, vư ợt xa mục đích sử dụng của nó.

Điều kẻ tấn công muốn: Một số kỹ thuật để lập trình kết hợp các máy kỳ lạ là rất cụ thể cho ứng dụng. Chúng phụ thuộc rất nhiều vào (các) lỗ hổng tạo thành lỗ hổng trong một ứng dụng đơ n lẻ và không thể tái sử dụng cho lỗi tiếp theo. Những thứ khác, như heap Feng-shui với Javascript [29], áp dụng cho toàn bộ lớp ứng dụng và nỗ lực đáng kể đư ợc dành để làm cho chúng có thể áp dụng trên các hệ thống. Rõ ràng, các kỹ thuật tái sử dụng có giá trị hơ n một thủ thuật chỉ sử dụng đư ợc một lần.

Từ quan điểm của kẻ tấn công, do đó, chúng tôi chắt lọc các mục tiêu sau để xây dựng và sử dụng các máy kỳ lạ:

1) Khả năng sử dụng lại : các kỹ thuật sử dụng lại tốt hơ n so với một lần. Lý tư ởng nhất là chúng tôi muốn có thể sử dụng lại cùng một cách khai thác cho nhiều chư ơ ng trình.
2) Tính đơ n giản: càng cần ít thao tác ư u tiên thì càng tốt. Nếu danh sách các thao tác dài và phức tạp, cuộc tấn công có thể khó thực hiện đối với một số chư ơ ng trình.
3) Tính tổng quát: một kỹ thuật có thể đư ợc sử dụng để khai thác là tốt, như ng thậm chí còn tốt hơ n nếu nó cũng có thể đư ợc sử dụng cho các cửa hậu và các mục đích khác.
4) Đa dạng: càng nhiều lựa chọn khai thác chư ơ ng trình càng tốt. Nếu chúng ta có nhiều vectơ tấn công cho một chư ơ ng trình với các điều kiện tiên quyết khác nhau, xác suất tìm thấy một vectơ phù hợp với chư ơ ng trình mục tiêu sẽ tăng lên.

Như đã giải thích ở trên, một khai thác điển hình trong thực tế bao gồm một chuỗi các kỹ thuật khác nhau, trong đó nhiều mã cụ thể của ứng dụng hơ n sẽ khởi động các phư ơ ng thức khai thác chung hơ n như ROP, và sau đó có thể là mã gốc của kẻ tấn công. Một lần nữa, kỹ thuật càng chung chung thì mã khai thác càng có thể đư ợc sử dụng lại cho các lỗ hổng khác.

2) ROP làm cho các cuộc tấn công trở nên khó khăn hơ n: Như chúng ta đã thấy trư ớc đó, các kỹ thuật khai thác đã phát triển để đáp ứng các biện pháp bảo vệ hiện đang đư ợc sử dụng trên các hệ thống hiện đại: việc thực thi shellcode trực tiếp đã trở nên hiếm và việc khai thác ROP đã trở nên phổ biến. Thật không may cho những kẻ tấn công, ROP không phải là một giải pháp thả xuống dễ dàng như mã shell thông thư ờng trư ớc đây, bởi vì các đoạn mã hữu ích và địa chỉ của chúng khác nhau đối với mỗi mã nhị phân mới mà mã khai thác đang đư ợc viết.

Đúng là các trình biên dịch ROP có thể tự động tạo ra một chuỗi ROP hữu ích cho hầu hết các tệp nhị phân đích, như ng ngay cả với các trình biên dịch ROP hiện đại nhất, lập trình hư ớng trở lại làm phức tạp đáng kể cuộc sống của kẻ tấn công.
Với tràn ngăn xếp truyền thống và không có ngăn chặn thực thi dữ liệu, cùng một cách khai thác có thể rất dễ dàng hoạt động trên nhiều tệp nhị phân có cùng lỗ hổng. Tuy nhiên, bây giờ, kẻ tấn công cần biết chính xác mã nhị phân mà nạn nhân sử dụng để đư a nó vào trình biên dịch ROP hoặc thậm chí có thể xây dựng chuỗi ROP theo cách thủ công. Có thể khó xác định phiên bản chính xác của tệp nhị phân vì ứng dụng đã có một số bản cập nhật có thể đã đư ợc cài đặt hoặc chư a. Trong trư ờng hợp xấu nhất, tệp nhị phân là bản dựng tùy chỉnh có trình biên dịch không xác định và cờ biên dịch không xác định. Ngay cả khi kẻ tấn công có thể xác định phiên bản, thì vẫn còn rất nhiều việc phải làm đối với mọi tệp nhị phân.

Các điều kiện tiên quyết khác cho một cuộc tấn công ROP thành công cũng rất quan trọng. Bên cạnh quyền kiểm soát ngăn xếp và chuyển hư ớng luồng điều khiển ban đầu, tệp thực thi cần tiết lộ địa chỉ của con trỏ mã, tùy thuộc vào ứng dụng, có thể khó.

Trong phần còn lại của bài báo này, chúng ta sẽ thấy rằng lập trình hư ớng sigreturn đạt điểm cao trên cả bốn tiêu chí trên. Mã SROP có thể di động [mục tiêu 1], cuộc tấn công đơ n giản (ví dụ: nó yêu cầu số lư ợng tiện ích tối thiểu mà trong một số hệ thống có thể tìm thấy tại các vị trí cố định) [mục tiêu 2], nó có nhiều ứng dụng [mục tiêu 3], và nó mở rộng tiết mục của những kẻ tấn công với các điều kiện tiên quyết khác nhau [mục tiêu 4].

IV. TRUYỀN TÍN HIỆU TRÊN HỆ THỐNG UNIX

Các tín hiệu đã là một phần không thể thiếu của các hệ thống UNIX kể từ khi bắt đầu [21]. Ban đầu không chỉ là một sự trừu tượng thuận tiện xung quanh các ngắt phần cứng, cơ chế này sau đó đã được điều chỉnh thành một hệ thống chung, có thể nhận thông báo cả từ nhân và từ các quy trình khác. Bằng cách đăng ký chức năng xử lý tín hiệu, một quy trình có thể xử lý các thông báo không đồng bộ bên ngoài luồng điều khiển thông thường của nó.

Lập trình hướng Sigreturn đòi hỏi sự hiểu biết thấu đáo về cách các tín hiệu được xử lý trên các hệ thống UNIX.
Cụ thể, chúng tôi sẽ phóng to cách hệ thống khôi phục trạng thái quy trình khi trở về từ bộ xử lý tín hiệu.

A. Phát tín hiệu

Khi một hạt nhân cung cấp tín hiệu cho một quy trình, luồng thực thi bình thường của quy trình tạm thời bị đình chỉ.
Cụ thể, nhân thay đổi bối cảnh CPU không gian người dùng sao cho chức năng xử lý tín hiệu đã đăng ký trước đó được gọi với các đối số phù hợp. Khi trình xử lý tín hiệu này quay trở lại, bối cảnh CPU không gian người dùng ban đầu được khôi phục.

Theo kiểu UNIX thực sự, điều này được triển khai theo cách tao nhã không yêu cầu sổ sách kế toán ở phía nhân. Bối cảnh người dùng bị treo chỉ đơn giản là được lưu trên ngăn xếp của quy trình và được khôi phục từ ngăn xếp khi bộ xử lý tín hiệu quay trở lại. Ban đầu, điều này được thực hiện đơn giản trong trình xử lý bẫy ngắt (Hình 1). Nhưng sự ra đời của bộ nhớ ảo và bảo vệ bộ nhớ phần cứng đã khiến điều này trở nên bất khả thi, vì bộ xử lý tín hiệu không gian người dùng không còn có thể trực tiếp quay lại quy trình ngắt. Điều này dẫn đến việc giới thiệu lệnh gọi hệ thống sigreturn trong 4.3BSD. Sigreturn lấy đối số đầu tiên làm con trỏ tới bối cảnh người dùng sẽ được khôi phục.

Một đoạn mã tấm bạt lò xo được đặt trong không gian địa chỉ người dùng, trước tiên gọi bộ xử lý tín hiệu và sau đó thực hiện lệnh gọi hệ thống sigreturn để báo hiệu cho hạt nhân, nó sẽ khôi phục bối cảnh không gian người dùng cũ.

Linux làm điều gì đó tương tự như BSD, ngoại trừ việc nó thực thi trực tiếp bộ xử lý tín hiệu. Khi bộ xử lý tín hiệu quay trở lại, nó sẽ quay trở lại một địa chỉ được ghi ở đó bởi kernel. Sơ khai tại địa chỉ này sau đó chịu trách nhiệm gọi sigreturn. Không giống như trên BSD (và iOS/Mac OS X), không có lý do gì để sigreturn. Hạt nhân chỉ cần tải bối cảnh người dùng từ ngăn xếp.

Hình 2 cho thấy phần trên cùng của ngăn xếp của tiến trình Linux 64 bit x86 khi bộ xử lý tín hiệu thoát ra.

B. Ngăn chặn thực thi dữ liệu và sigreturn

Tấm bạt lò xo gọi sigreturn phải nằm trong bộ nhớ không gian của người dùng. Nó từng là tùy chỉnh cho các hạt nhân để viết mã tấm bạt lò xo cùng với bối cảnh người dùng trên ngăn xếp. Trình xử lý tín hiệu trả về sẽ chỉ nhảy tới mã này trên ngăn xếp. Tuy nhiên, điều này yêu cầu một ngăn xếp có thể thực thi được, cho phép chèn shellcode dễ dàng vào ngăn xếp.
Ngày nay, tùy thuộc vào kiến trúc, sigreturn

```
NSIG = 0

# ngắtvectortvect : mov r0 ,
( sp ) ; #
      loadsignalhandlerfunc
                  tion mov
      dvect + [NSIG   2 ] , r 0 ; anh 1 f ;


      NSIG=NSIG+1 # lặp
      lại tổng cộng 20 lần ( đối với 20 tín hiệu )
            . . .
1 :
      # pushtheregisterstate trên ngăn xếp mov r1 , ( sp ) mov
      r2 , ( sp ) mov r3 , ( sp ) mov r4 , ( sp ) # khôi phục
      các register mov r4 , ( sp )+ mov r3 , ( sp )+ mov r2 , ( sp )+ mov
                  ,r 4
                  ,r 3
                  ,r 2
                  ,r 1
                  ,r 0

      rtt
```

Hình 1. Đoạn trích từ s5/signal.s, thói quen ngắt UNIX V6 (thêm ý kiến cho rõ ràng). Chúng tôi thấy rằng ngay cả trong các phiên bản đầu tiên này, trạng thái CPU bị treo vẫn được lưu trữ trên ngăn xếp.

mã tấm bạt lò xo được cung cấp trong trang bộ nhớ thực thi do hạt nhân cung cấp, cho phép hạt nhân biết vị trí của nó hoặc nó nằm ở đâu đó trong libc.

V. LẬP TRÌNH ĐỊNH HƯỚNG SIGRETURN

Trong phần này, chúng tôi thảo luận về các ý tưởng chính đằng sau lập trình hướng sigreturn. Để làm như vậy, trước tiên chúng tôi giới thiệu một số chi tiết về cách Linux và các hương vị UNIX khác trả về từ các tín hiệu và sau đó thảo luận về việc sigreturn dễ bị lạm dụng như thế nào.

A. Gửi tín hiệu trên Linux

Ở đỉnh mới của ngăn xếp, hạt nhân ghi địa chỉ mã sẽ trở thành địa chỉ trả về cho bộ xử lý tín hiệu. Địa chỉ mã này trỏ đến một sơ khai nhỏ không làm gì khác hơn là gọi lệnh gọi hệ thống sigreturn(0) . Cuộc gọi sigreturn() này hoàn tác mọi thứ đã được thực hiện để gọi trình xử lý tín hiệu (thay đổi mặt nạ tín hiệu của quy trình, chuyển đổi ngăn xếp). Do đó, nó khôi phục mặt nạ tín hiệu của quy trình, chuyển đổi ngăn xếp và khôi phục ngữ cảnh của quy trình (các thanh ghi, cờ bộ xử lý) —làm cho quy trình tiếp tục thực thi chính xác tại điểm mà tín hiệu đã làm gián đoạn nó.

B. Lệnh gọi hệ thống sigreturn trên Linux

Mặc dù sigreturn là một lệnh gọi hệ thống, được gọi giống như bất kỳ lệnh gọi nào khác, nhưng điều đặc biệt ở chỗ là không chương trình không gian người dùng nào cần biết về ABI của nó. Hạt nhân chịu trách nhiệm thiết lập

lên ngăn xếp không gian ngư ời dùng theo cách mà cuối cùng nó đư ợc gọi.

Như đã đề cập trư ớc đó, mã tấm bạt lò xo gọi si greturn đã từng nằm trên ngăn xếp trong khung tín hiệu trong trư ờng mã tiền tố , như ng vì việc thực thi tấm bạt lò xo này yêu cầu một ngăn xếp thực thi nên nó không còn đư ợc sử dụng trên các nhân gần đây. Thật thú vị, dấu tích vẫn tồn tại trên Linux i386 để giúp gdb xác định khung tín hiệu. Tuy nhiên, trên thực tế, sơ khai đã chuyển sang đối tư ợng chia sẻ động ảo (vdso), một đoạn mã do nhân cung cấp đư ợc ánh xạ trong không gian địa chỉ của mọi quy trình. Trên x86-64 Linux, sơ khai có mặt trong chính libc. Nó phải đư ợc cung cấp cho kernel khi đăng ký tín hiệu bằng trư ờng sigaction.se_restorer . Trong cả hai trư ờng hợp, địa chỉ gốc sigreturn thư ờng đư ợc chọn ngẫu nhiên bởi ASLR [20], [25].

Khi sigreturn(0) đư ợc gọi, hạt nhân sẽ sử dụng con trỏ ngăn xếp không gian ngư ời dùng để tìm khung tín hiệu mà nó đã lư u trữ ở đó trư ớc đó và tải bối cảnh ngư ời dùng ban đầu vào các thanh ghi của CPU. Bằng cách này, nó khôi phục bối cảnh ban đầu cho quá trình bị gián đoạn.

Hình 2 cho thấy một khung tín hiệu mà nhân Linux 64 bit sẽ đặt trên ngăn xếp. Chúng ta thấy rằng khung chứa các giá trị thanh ghi, bao gồm con trỏ ngăn xếp (RSP), con trỏ lệnh (RIP), các thanh ghi đoạn (CS, FS và GS) và các cờ. Địa chỉ trả về nằm ở đầu khung, theo sau là bối cảnh ngư ời dùng và các thanh ghi.

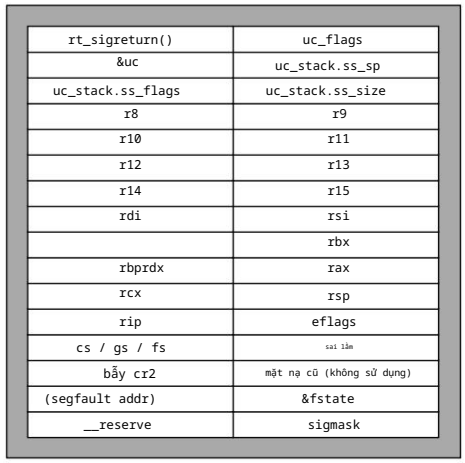
Các thanh ghi thông thư ờng có thể chứa các đối số và chúng tôi cũng sẽ sử dụng chúng theo cách này.

Con trỏ trạng thái dấu phẩy động trỏ đến trạng thái đơ n vị dấu phẩy động đã lư u. Nó chỉ quan trọng nếu có bất kỳ trạng thái dấu phẩy động nào cần khôi phục, nghĩa là, nếu có bất kỳ hoạt động dấu phẩy động nào trư ớc khi tín hiệu đến. Mặt khác, nếu con trỏ là NULL, hạt nhân sẽ giả định rằng không có hoạt động nào như vậy và bỏ qua nó.

Các khung tín hiệu trên các kiến trúc khác trông rất giống nhau, mặc dù ngữ cảnh thanh ghi lư u trữ theo định nghĩa các thanh ghi và giá trị dành riêng cho kiến trúc khác.

C. Sigreturn về hư ơ ng vị UNIX khác

Các hệ thống tư ơ ng tự UNIX và UNIX khác nhau thực hiện quay lại tín hiệu theo những cách hơ i khác nhau. Trên BSD, Mac OSX và iOS, sigreturn cũng tư ơ ng tự, ngoại trừ việc nó là một phần của ABI và nó lấy vị trí của sigframe cấu trúc làm đối số đầu tiên. Trên OpenSolaris không có sigreturn; khôi phục bối cảnh ban đầu xảy ra trong không gian ngư ời dùng và đư ợc xử lý hoàn toàn bởi libc, cung cấp trình bao bọc xung quanh bộ xử lý tín hiệu và khôi phục bối cảnh không gian ngư ời dùng. Mặc dù các hệ điều hành khác cũng cung cấp những con đư ờng khai thác thú vị, như ng chúng tôi sẽ giới hạn bản thân trong các phiên bản khác nhau của Linux, BSD và iOS trong phần còn lại của bài báo, thư ờng sử dụng Linux làm ví dụ minh họa.



Hình 2. Khung tín hiệu giống như trong Linux 86-64

D. Lạm dụng chữ ký

Việc khôi phục bối cảnh của một quá trình bị gián đoạn bằng cách tải một khung ngăn xếp đã lư u trư ớc đó rất thuận tiện vì nó từ bỏ trách nhiệm của hạt nhân trong việc theo dõi các tín hiệu mà nó đã gửi. Tuy nhiên, nó cũng có một như ợc điểm lớn: hạt nhân không theo dõi các tín hiệu mà nó truyền đi. Nói cách khác, không có cách nào để biết liệu một sigreturn có hợp pháp hay không.

Bằng cách thiết lập một khung sigframe cấu trúc chính xác, tải đúng số lệnh gọi hệ thống và thực hiện lệnh gọi hệ thống trong cấu trúc, kẻ tấn công có thể đánh lừa nhân một cách tầm thư ờng để hoạt động giống như một trình xử lý tín hiệu vừa hoàn thành. Trong trư ờng hợp đó, kernel sẽ tải bối cảnh không gian ngư ời dùng do kẻ tấn công tạo ra từ ngăn xếp.

Trong phần còn lại của bài báo này, chúng ta sẽ khám phá những hậu quả không mong muốn của việc có thể thực hiện một sigreturn trên dữ liệu tùy ý.

VI. SROP ĐỂ KHAI THÁC

Như đã đề cập, những kẻ tấn công có thể sử dụng lập trình lập trình theo định hư ớng sigreturn cho các mục đích khác nhau. Đầu tiên, chúng tôi sẽ phác thảo hai kỹ thuật khai thác, một kỹ thuật chung đơ n giản theo sau là một phư ơ ng pháp linh hoạt hơ n cho các quy trình Linux 64 bit. Phư ơ ng pháp thứ hai này phức tạp hơ n như ng có các điều kiện trư ớc yếu hơ n. Trong các phần sau, chúng tôi thảo luận về các cách sử dụng khác của SROP.

A. Khai thác execve() SROP đơ n giản

Kết quả cuối cùng của nhiều khai thác UNIX thư ờng là một shell do kẻ tấn công kiểm soát. Trong phần này, chúng tôi sẽ phác thảo một cách khai thác sẽ gọi hàm execve để khởi động trình bao với các đối số tùy ý.

Để bất kỳ hoạt động khai thác nào thành công, một số điều kiện tiên quyết phải đư ợc đáp ứng. Chẳng hạn, để thực thi shellcode trực tiếp, kẻ tấn công phải có khả năng tải mã của chúng vào bộ nhớ thực thi và chuyển hư ớng điều khiển sang bộ đệm này. ROP yêu cầu con trỏ mã, tiện ích, điều khiển ngăn xếp, điều khiển

chuyển hư ớng dòng chảy, v.v. Tư ơ ng tự, có thể khai thác SROP thành công bằng kỹ thuật này nếu kẻ tấn công đáp ứng các điều kiện tiên quyết sau:

1) Kẻ tấn công phải có quyền kiểm soát con trỏ lệnh (ví dụ: do lệnh trả về trên ngăn xếp bị ghi đè).

2) Con trỏ ngăn xếp phải đư ợc đặt trên dữ liệu do kẻ tấn công kiểm soát và phải cho phép các byte NULL (ví dụ: trong trư ờng hợp tràn). Đối với BSD, Mac OS X và iOS, một con trỏ hàm ghi đè bằng bộ đệm do ngư ời dùng kiểm soát làm đối số đầu tiên cũng có thể xảy ra. Trong trư ờng hợp này, không cần bất kỳ dữ liệu nào do ngư ời dùng kiểm soát trên ngăn xếp.

3) Kẻ tấn công biết địa chỉ của một phần dữ liệu do kẻ tấn công kiểm soát. Đây có thể là ngăn xếp bị ghi đè, như ng không nhất thiết phải ở cùng một vị trí.

4) Kẻ tấn công biết vị trí của mã gọi sigreturn hoặc syscall, trong trư ờng hợp kẻ tấn công có thể kiểm soát thanh ghi CPU chuyển số gọi hệ thống.

B. Tìm tiện ích sigreturn

Như với lập trình hư ớng trả về, SROP cần một số kiến thức về vị trí của mã trong không gian địa chỉ của quy trình. Như ng không giống như ROP, SROP thực sự chỉ cần biết vị trí của một tiện ích duy nhất, đó là lệnh gọi sigreturn.

Trong khai thác của chúng tôi, chúng tôi cũng sử dụng một tiện ích bổ sung thực hiện các cuộc gọi hệ thống tùy ý, như ng tiện ích bổ sung này đư ợc chứa trong tiện ích sigreturn . Đối với điều này, chúng tôi chỉ cần bỏ qua hư ớng dẫn tải cuộc gọi hệ thống. Trong một số trư ờng hợp, chúng tôi có thể kiểm soát thanh ghi CPU chịu trách nhiệm chuyển cuộc gọi hệ thống. Trong trư ờng hợp đó, chúng tôi không cần tiện ích sigreturn và tiện ích syscall là đủ như sẽ đư ợc trình bày chi tiết trong kỹ thuật khai thác thứ hai của chúng tôi.

Hóa ra, thật dễ dàng để tìm thấy những tiện ích này trên một số kiến trúc. Trên FreeBSD 9.2 trên x86-64, có một trang bộ nhớ cố định chứa sigreturn. Linux trên ARM, trư ớc phiên bản 3.11 (bao gồm tất cả các phiên bản Android hiện tại cũng như phiên bản ổn định dài hạn mới nhất (3.10) có bản đồ [vectơ] cố định với các tiện ích sigreturn.

Hơ n nữa, nhiều phiên bản Linux trên x86-64 có bản đồ [vsyscall] cố định với tiện ích syscall & return (xem Phần VI-D).

Nếu sigreturn nằm trong libc và không có tiện ích nào đư ợc biết trư ớc, thì kẻ tấn công có thể cần phải tiết lộ địa chỉ mã libc để có đư ợc tiện ích sigreturn. Trên các hệ thống thực hiện liên kết trư ớc thư viện, các tiện ích sigreturn có thể giống nhau trên toàn hệ thống, điều này cho phép khai thác cục bộ tìm đúng tiện ích. Bảng I hiển thị vị trí của các tiện ích hữu ích trong các Hệ điều hành khác nhau.

C. Thực hiện lời gọi hệ thống

Vì hầu hết các kiến trúc chuyển các tham số gọi hệ thống tới nhân thông qua các thanh ghi (một ngoại lệ đáng chú ý là BSD trên i386), thực hiện sigreturn cho phép chúng tôi thực hiện cuộc gọi hệ thống tùy ý. Bằng cách đặt con trỏ lệnh của khung ngăn xếp của chúng tôi thành địa chỉ của lệnh tòa nhà chọc trời và điền vào các thanh ghi chuyển số cuộc gọi hệ thống và các đối số của nó, chúng tôi tải một cách hiệu quả trạng thái trong đó lệnh gọi hệ thống mà chúng tôi chọn đư ợc thực thi.

Hệ điều hành	Bản đồ bộ nhớ	Gadget i386 [vdso]	
Linux < 3.11	[vectors] 0xffff0000	sigreturn 0xffffe000	Linux < 3.3 x86-64 tòa nhà chọc trời & trả về
Linux ≥ 3.3 x86-64 tòa nhà chọc trời & trả về	[vsyscall] 0xffffffffff600000		
Libc Linux x86-64 sigreturn	Libc FreeBSD 9.2 x86-64 sigreturn	OpenBSD 9.2x86-64 sigreturn	
sigcode trang Mac OS X 10.5 x86-64 sigreturn	Libc iOS ARM sigreturn	Libsystem iOS ARM	
syscall & return Libsystem		0x7fffffff000	

Bảng I

SI GRETURN VÀ SYSCALL GADGETS VÀ VỊ TRÍ CỦA CHÚNG

Vì chúng tôi biết vị trí của một số dữ liệu do kẻ tấn công kiểm soát, giờ đây chúng tôi có thể thực hiện lệnh gọi hệ thống execve tới, chẳng hạn như /system/bin/sh trên Android, sử dụng con trỏ tới dữ liệu của chúng tôi cho tên tệp execve và tham số argv và sử dụng sigreturn & cố định các tiện ích syscall từ trang [vectors] .

D. Khai thác SROP trên Linux x86-64 bằng chuỗi cuộc gọi hệ thống

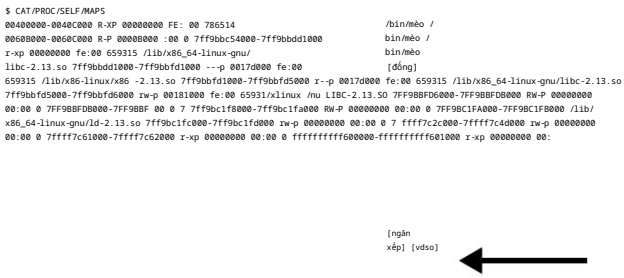
Cho đến bây giờ, chúng tôi đã giả định rằng chúng tôi biết về các vị trí bộ nhớ có dữ liệu do kẻ tấn công kiểm soát và rằng chúng tôi đã biết về một tiện ích sigreturn . Đối với phư ơ ng pháp khai thác tiếp theo, chúng tôi sẽ không phải biết vị trí chính xác của dữ liệu do kẻ tấn công kiểm soát trong không gian địa chỉ và chúng tôi sẽ không yêu cầu bất kỳ kiến thức nào về mã từ chính ứng dụng. Kỹ thuật khai thác này nhắm vào các hệ thống x86-64 chạy nhân Linux cũ hơ n phiên bản 3.3 và sử dụng phư ơ ng pháp này, chúng tôi sẽ khai thác các phiên bản khác nhau của máy chủ Asterisk dễ bị tấn công bằng cách sử dụng cùng một cách khai thác trên các hệ thống khác nhau.

Các điều kiện tiên quyết mới

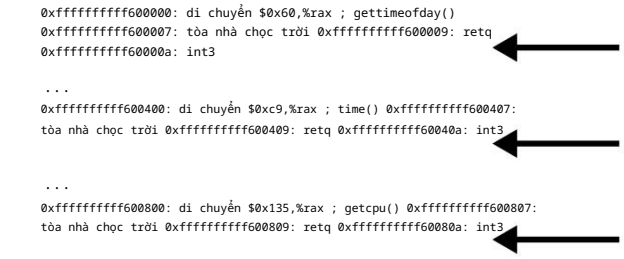
của chúng ta là: • Con trỏ ngăn xếp phải nằm trên dữ liệu do kẻ tấn công kiểm soát và phải cho phép các byte NULL (ví dụ: trong trư ờng hợp tràn).

• Kẻ tấn công phải có một số quyền kiểm soát đối với RAX.

Cụ thể, RAX phải chứa giá trị 15. • Kẻ tấn công phải có quyền kiểm soát con trỏ lệnh RIP (ví dụ do lệnh RET trên ngăn xếp bị ghi đè). • Kẻ tấn công nên biết vị trí của một (bất kỳ !) trang có thể ghi duy nhất trong bộ nhớ. Vị trí hoặc nội dung của mã nhị phân không quan trọng. Nếu kẻ tấn công quản lý để rò rỉ một địa chỉ có thể ghi, thì tệp thực thi thậm chí có thể độc lập về vị trí và danh tính chính xác của tệp nhị phân hoàn toàn không xác định đư ợc. • Hệ thống của mục tiêu triển khai vsyscall riêng, đây là mặc định trên các nhân trư ớc 3.3, chẳng hạn như các nhân



Hình 3. Vsyscall đư ợc ánh xạ trong không gian địa chỉ của một tiến trình



Hình 4. Trang vsyscall chứa tiện ích syscall & ret nhiều lần

đư ợc sử dụng trên Debian 7.0 (2013), Hỗ trợ dài hạn Ubuntu (2012) và CentOS 6 (2012).

- Kẻ tấn công kiểm soát dữ liệu đư ợc gửi đến một tệp hoặc ổ cắm với một số mô tả tập tin đã biết.

Nói chung, cuộc tấn công SROP hoạt động bằng cách liên tục thiết lập các khung tín hiệu giả và quay lại từ "tín hiệu" thành các cuộc gọi hệ thống thông thư ờng và ngư ợc lại. Để giải thích làm thế nào điều này có thể xảy ra, trư ớc tiên chúng tôi mô tả chi tiết về việc thực hiện cuộc gọi hệ thống.

1) Tiện ích hữu ích trên Linux x86-64: vsyscall: Vsyscall là giao diện cuộc gọi hệ thống nhanh dành cho Linux 64 bit. Nó đư ợc tạo ra để tăng tốc các cuộc gọi hệ thống nhạy cảm với thời gian nhất định.
Thay vì sử dụng lệnh tòa nhà chọc trời, một ứng dụng có thể đơ n giản nhảy tới các địa chỉ tĩnh trong một trang tĩnh do nhân thiết lập (xem Hình 3). Hạt nhân đã cung cấp mã không gian ngư ời dùng tại các địa chỉ này đã triển khai time(), gettimeofday() và getcpu(). Bất kỳ dữ liệu đặc quyền nào cần thiết để các lệnh gọi hệ thống này hoàn thành (chẳng hạn như thời gian hiện tại) đều đư ợc cung cấp bởi nhân trong các cấu trúc dữ liệu trong cùng một trang.

Điều này hóa ra lại là một cơ n ác mộng về bảo mật, vì nó cung cấp cho kẻ tấn công rất nhiều tiện ích hữu ích để khai thác tại các địa chỉ cố định, giống hệt nhau cho mọi quy trình đang chạy.
Các tiện ích này bao gồm syscall & ret: một syscall (0f 05) theo sau là một return (c3). Đối với bài báo này, chúng tôi đặc biệt quan tâm đến tiện ích syscall & ret này , bởi vì nó cho phép một ngư ời thực hiện lệnh gọi hệ thống theo lựa chọn, miễn là chúng tôi có thể đặt số lệnh gọi hệ thống thích hợp trong RAX (xem Hình 4).

Mặc dù các tiện ích dư ờng như khác nhau giữa các bản phân phối, như ng các tiện ích hữu ích dư ờng như vẫn giữ nguyên sau ngày cập nhật bảo mật kernel. Nói cách khác, đối với các hệ thống chạy hạt nhân phân phối chứng khoán, sẽ dễ dàng xác định địa chỉ của các tiện ích—thậm chí từ xa! Ngoài ra, vì hạt nhân theo dõi thời gian của đồng hồ treo tư ờng trong cùng một trang với mã thực thi, kẻ tấn công kiên nhẫn có thể chỉ cần đợi biểu diễn nhị phân của thời gian trên đồng hồ treo tư ờng chứa một tiện ích (nhỏ) mà anh ta/cô ta chọn và sau đó chuyển đến (byte ít quan trọng nhất) trư ờng thời gian của đồng hồ treo tư ờng.

Do các rủi ro bảo mật của nó, vsyscall không đư ợc dùng trong Linux 3.1 và theo mặc định, các thói quen không gian ngư ời dùng nhanh hiện chỉ gọi các đối tác cuộc gọi hệ thống bình thư ờng của chúng. Mặc dù điều này đã loại bỏ một số tiện ích có hại, như ng do tính đơ n giản của nó, nó đã dẫn đến việc các tiện ích còn lại, chẳng hạn như syscall & ret có địa chỉ ổn định trên tất cả các bản phân phối. Lý do là tệp C chứa vsyscall đã đư ợc thay thế bằng tệp hợp ngữ chứa một số tiện ích syscall & ret .
Tệp lắp ráp luôn tạo cùng một tệp nhị phân.

Đồng thời, các nhà phát triển Linux đã thêm chế độ mô phỏng vsyscall thứ hai , chế độ này mô phỏng các vsyscall bằng cơ chế dựa trên bẫy. Do đó, chúng tôi không thể sử dụng các tiện ích cuộc gọi hệ thống từ trang này. Tuy nhiên, lư u ý rằng đối với nhiều nhân, đây không phải là cấu hình mặc định1 .
Ngay cả khi không có vị trí cố định cho các tiện ích gọi hệ thống, chúng ta vẫn có thể tìm thấy một vị trí trong chính tệp thực thi—xét cho cùng, chúng ta chỉ cần một tiện ích nhỏ. Trong trư ờng hợp đó, chúng tôi mất tài sản của không cần trinh sát, như ng khai thác vẫn hoạt động. Là một bản phát hành dài hạn, nhân 3.2 hiện có trong nhiều bản phân phối, chẳng hạn như Debian 7.0 (phát hành năm 2013, đư ợc hỗ trợ trong 10 năm) và Hỗ trợ dài hạn Ubuntu 12.04 (phát hành năm 2012 và đư ợc hỗ trợ trong 5 năm)

E. Bootstrapping để thực thi mã tùy ý

Trong phần này, chúng tôi chỉ ra cách sigreturn có thể giúp kẻ tấn công thực thi mã tùy ý. Để làm như vậy, chúng tôi sử dụng một chuỗi các cuộc gọi hệ thống và sigreturns dùng làm ví dụ. Cũng có thể có các trình tự khác mà kẻ tấn công có thể sử dụng để đạt đư ợc hiệu quả tư ơ ng tự.

Điều đầu tiên chúng tôi làm là tạo một khung tín hiệu giả và kích hoạt một sigreturn. Trư ớc tiên, chúng tôi giải thích khung tín hiệu đầu tiên trông như thế nào.

Để thực hiện lệnh gọi hệ thống sigreturn thành công , chúng tôi phải đảm bảo rằng hạt nhân không vư ợt qua bất kỳ giá trị xấu nào trong khung tín hiệu. Yêu cầu đầu tiên là thanh ghi đoạn mã đư ợc khôi phục chính xác. Trên x86-64, khi chạy ở chế độ 64 bit, thanh ghi đoạn mã phải chứa giá trị 0x33. Yêu cầu thứ hai là con trỏ fpstate không phải là con trỏ hoang dã. Fpstate trỏ đến trạng thái đơ n vị dấu phẩy động đã lư u và nếu địa chỉ của nó không hợp lệ hoặc trạng thái dấu phẩy động đã lư u không chính xác, ứng dụng của chúng tôi

[1]Nó chắc chắn không phải là mặc định cho tất cả các nhân trư ớc Linux 3.3. và ngay cả hạt nhân sau 3.3 cho phép một ngư ời khởi động mà không cần vsyscall mô phỏng.

sẽ sụp đổ. Đây có vẻ là một vấn đề, vì giả định của chúng tôi là chúng tôi chưa biết bất kỳ dữ liệu nào do kẻ tấn công kiểm soát trên một địa chỉ đã biết. May mắn thay, khi fpstate là NULL, Linux cho rằng không có hoạt động dấu chấm động nào được sử dụng trước khi tín hiệu đến. Trong trường hợp này, nó xóa trạng thái FPU và sigreturn thành công.

Khi chúng tôi thực hiện sigreturn, chúng tôi có toàn quyền kiểm soát các thanh ghi, cũng như đối với bộ đếm chương trình (RIP.) Nhưng chúng tôi cũng mất mọi cơ hội sử dụng bất kỳ thông tin ASLR nào có trong các thanh ghi vì chúng đã bị ghi đè. Đây là lý do tại sao chúng ta cần địa chỉ của một trang có thể ghi được. Chúng tôi sẽ trỏ con trỏ ngăn xếp trong khung tín hiệu của chúng tôi xuống cuối trang này. Bằng cách điền vào các thanh ghi cần thiết và trỏ bộ đếm chương trình vào tiện ích syscall & ret , chúng ta có thể thiết lập và sau đó thực hiện bất kỳ cuộc gọi hệ thống nào mà chúng ta muốn (xem thêm Hình 5).

Chúng tôi sẽ sử dụng sức mạnh này để thiết lập một cuộc gọi hệ thống read() . Việc đọc sẽ đọc dữ liệu của kẻ tấn công và lưu trữ nó dưới con trỏ ngăn xếp. Khi quá trình đọc kết thúc, dữ liệu của kẻ tấn công sẽ đóng vai trò là địa chỉ trả về. Trong trường hợp của chúng tôi, kẻ tấn công trỏ lại vào tiện ích syscall & ret .

Bước 1 và 2: quay lại và đọc

Tóm lại, trong hai bước đầu tiên, chúng tôi tạo một khung tín hiệu giả về cuộc tấn công, như đã giải thích ở trên. Trong khung tín hiệu, giá trị RSP sẽ trỏ đến trang có thể ghi, giá trị cho RAX sẽ là 0 (biểu thị lệnh gọi hệ thống đọc) và sẽ có các giá trị thích hợp trong các thanh ghi đóng vai trò là đối số cho lệnh gọi hệ thống đọc . Sau đó, kẻ tấn công chuyển hướng luồng điều khiển của chương trình sang tiện ích syscall & ret để thực hiện lệnh gọi hệ thống và đọc dữ liệu của kẻ tấn công trên ngăn xếp mới (trang có thể ghi). Bởi vì kẻ tấn công kiểm soát địa chỉ trả về, nên anh ta có thể chỉ cần quay lại tiện ích syscall & ret —không giống như những gì người ta sẽ làm trong ROP thông thường.

Tại thời điểm này, kẻ tấn công không có lựa chọn nào khác ngoài việc giữ nguyên tất cả các đối số của lệnh gọi hệ thống, vì nhân không thay đổi chúng trong quá trình thực hiện lệnh gọi hệ thống. Nhưng RAX sẽ chứa số byte đã đọc. Điều này rất quan trọng và kẻ tấn công sử dụng nó để chọn lệnh gọi hệ thống tiếp theo để thực hiện. Như đã đề cập trước đó, thanh ghi RAX cho biết lệnh gọi hệ thống nào sẽ được thực hiện.

Bước 3: một NOP cần thiết Cụ

thể, kẻ tấn công chọn đọc 306 byte. Điều này không chỉ cung cấp khá nhiều dữ liệu hiện đang ở một vị trí đã biết, mà 306 còn là số gọi hệ thống cho syncfs(int fd)—bước thứ ba của chúng ta trong quá trình khai thác.

Lệnh gọi hệ thống syncfs() lấy một bộ mô tả tệp làm đối số đầu tiên và xóa tất cả các đĩa thuộc bộ mô tả này.

Vì bộ mô tả tệp của chúng tôi là một ổ cắm, đây thực sự sẽ là một lệnh cấm trả về 0 trong RAX. Kẻ tấn công một lần nữa đảm bảo quay lại tiện ích syscall & ret .

Bước 4: một lần đọc khác để đặt RAX Trên

x86-64, giá trị 0 xảy ra là số cuộc gọi hệ thống để đọc, cho phép kẻ tấn công đọc lại dữ liệu trên ngăn xếp mới—bước thứ tư của chúng ta. Lần này kẻ tấn công làm cho chắc chắn chỉ gửi 15 byte, để giá trị của RAX bây giờ là 15.

Bước 5 và 6: sigreturn để thực thi bất cứ thứ gì chúng ta thích

Như đã đề cập trước đó, 15 cũng là số gọi hệ thống cho sigreturn, vì vậy chúng ta quay lại nơi chúng ta bắt đầu, nhưng có một sự khác biệt quan trọng: có dữ liệu được kiểm soát bởi kẻ tấn công tại một địa chỉ đã biết. Sigreturn tiếp theo , bước thứ năm của chúng ta, một lần nữa tự do tải một lệnh gọi hệ thống tùy ý vào các thanh ghi, điều này cho phép kẻ tấn công làm bất cứ điều gì hắn muốn.

Chẳng hạn, anh ta có thể thực hiện một lệnh gọi hệ thống mprotect và chuyển sang shellcode truyền thống hoặc một execve với các đối số phù hợp để sinh ra một shell.

F. Khai thác máy chủ web Asterisk

Chúng tôi đã thử nghiệm kỹ thuật khai thác của mình trên phiên bản Asterisk gần đây dễ bị tấn công do lỗi phân bổ ngăn xếp không giới hạn (CVE-2012-5976). Lỗ hổng đã được mô tả chuyên sâu tại blog EIP [8]. Khai thác của chúng tôi là hoàn toàn mới và nhắm mục tiêu nhiều nhị phân.

Lỗ hổng phân bổ ngăn xếp không giới hạn xảy ra khi yêu cầu POST HTTP xác thực trước tới bảng điều khiển quản lý web của Asterisk phân bổ dữ liệu bài đăng HTTP trên ngăn xếp.

Nó sử dụng tiêu đề độ dài nội dung do khách hàng gửi để xác định lượng dữ liệu sẽ được phân bổ. Tuy nhiên, nó không kiểm tra xem kích thước này có nằm trong giới hạn hợp lý hay không.

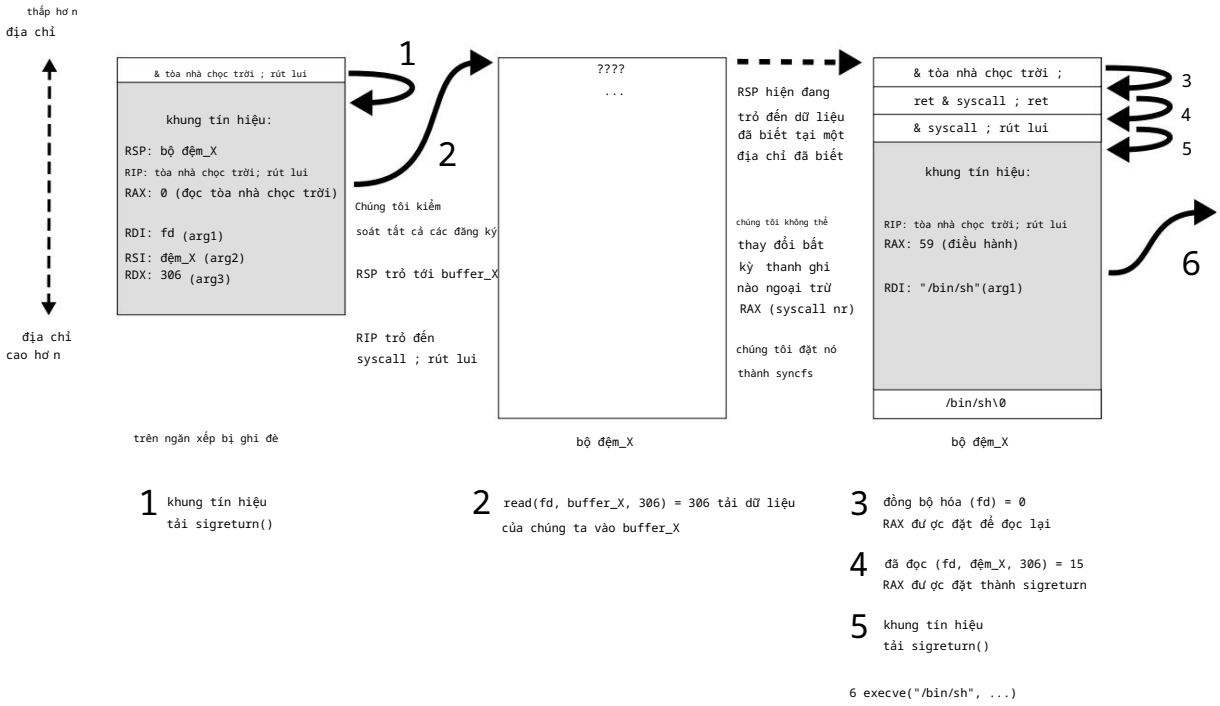
Việc chỉ định độ dài nội dung bằng với kích thước dành riêng cho ngăn xếp cho phép chúng tôi 'nhảy' bằng con trỏ ngăn xếp của mình tới ngăn xếp của luồng thứ hai trong quy trình dấu hoa thị. Sau khi chuyển sang ngăn xếp của luồng thứ hai, chúng ta có thể bắt đầu gửi dữ liệu bài đăng của mình, dữ liệu này sẽ nhanh chóng ghi đè lên ngăn xếp thứ hai.

Bằng cách đảm bảo rằng luồng thứ hai này cũng do chúng tôi khởi tạo, chúng tôi có thể ghi đè lên ngăn xếp của luồng này trong khi nó đang chờ đọc chặn. Khi chúng tôi thực hiện xong việc làm hỏng ngăn xếp, chúng tôi gửi 15 byte tới luồng có ngăn xếp bị hỏng và khi nó trả về đối với tiện ích syscall & ret của chúng tôi , nó gọi sigreturn, khởi động phương thức bootstrap của chúng tôi từ Phần VI-E.

Chúng tôi vẫn cần đáp ứng hai yêu cầu còn lại của mình: chúng tôi cần kiểm soát bộ mô tả tệp và chúng tôi cần biết một trang có thể ghi. Đối với trang có thể ghi, chúng tôi đã đoán một trang trong phần dữ liệu của tệp nhị phân. Theo mặc định, với các tệp thực thi không có vị trí độc lập được biên dịch bằng gcc, phần dữ liệu xuất hiện ngay sau phần mã và phần mã bắt đầu ở độ lệch cố định là 0x40000. Kích thước của mã

phần không thay đổi nhiều giữa các phiên bản khác nhau của Asterisk so với kích thước tuyệt đối của phần dữ liệu, do đó khá dễ đoán một trang có thể ghi trong tệp nhị phân Asterisk.

Để chọn đúng bộ mô tả tệp, chúng tôi mở một số lượng lớn các kết nối đến phần dễ bị tổn thương và gửi cùng một dữ liệu trên tất cả các ổ cắm. Bằng cách chọn một giá trị cao cho bộ mô tả tệp, chúng tôi có thể khá chắc chắn rằng chúng tôi đã chọn một ổ cắm mà chúng tôi đã mở.



Hình 5. Các bước liên quan đến khai thác Linux x86-64 SROP

Chúng tôi đã thử nghiệm cách khai thác này trên ba phiên bản dễ bị tổn thương khác nhau của chương trình Asterisk trên các bản phân phối Linux khác nhau: Debian Wheezy (phát hành vào tháng 5 năm 2013), Ubuntu LTS 12.04 (phiên bản Hỗ trợ dài hạn mới nhất của Ubuntu, phát hành năm 2012 và được hỗ trợ trong 5 năm) và Centos 6.3 (phát hành năm 2012 và được hỗ trợ trong 10 năm).

Việc khai thác hoạt động trên tất cả các bản phân phối Linux mà chúng tôi đã thử. Hơn nữa, khả năng sử dụng lại của khai thác được gợi ý trên một số điều bởi thực tế là mã cho cả ba phiên bản gần như giống hệt nhau. Điểm khác biệt duy nhất là tiện ích syscall & ret tại Centos nằm ở một vị trí hơi khác.

VII. SROP NHƯ CỬA SAU

Một cách sử dụng khả thi khác cho lập trình hướng sigreturn là làm một cửa hậu lén lút. Bằng cách đưa các khung tín hiệu vào không gian địa chỉ của một quy trình và bằng cách tạo thêm một luồng trong một quy trình hoặc chỉ đơn giản là thay thế việc thực thi của một quy trình bằng chính chúng ta, có thể duy trì sự hiện diện trên hệ thống, trong khi có vẻ như đã rời đi.

Các nhà phát triển cửa hậu rất muốn tránh bị phát hiện.
Thật không may, mã shell được đưa vào sẽ có vẻ đáng ngờ khi kết xuất bộ nhớ được xem bằng các công cụ pháp y. Ẩn tất cả logic trong dữ liệu có vẻ lén lút hơn. Mặc dù về nguyên tắc, điều này cũng có thể được thực hiện bằng cách sử dụng ROP, nhưng chúng tôi sẽ chỉ ra rằng đối với lập trình hướng sigreturn, nó có thể được thực hiện theo một cách hoàn toàn chung, hoạt động cho tất cả các quy trình và không yêu cầu trình biên dịch ROP phức tạp.

Đối với ví dụ khai thác của chúng tôi, chúng tôi giả định rằng tiện ích duy nhất có sẵn cho chúng tôi là syscall & ret và tiện ích này được trang vsyscall cung cấp cho chúng tôi dưới dạng tiện ích không phải ASLR. Đối với cửa hậu của chúng tôi, chúng tôi sẽ không còn phụ thuộc vào trang vsyscall nữa. Chúng tôi cũng sẽ không còn yêu cầu kiến trúc phải là x86 64 bit vì chúng tôi không cần sử dụng phương thức bootstrap cụ thể x86-64 để chuyển từ lệnh gọi hệ thống này sang lệnh gọi hệ thống khác, điều này khiến các giá trị trả về của một hệ thống gọi số lệnh gọi hệ thống của hệ thống tiếp theo.

Trong kịch bản cửa hậu của chúng tôi, kẻ tấn công sử dụng ptrace() để đưa một máy lạ vào quy trình nạn nhân. Với ptrace() , việc tìm một tiện ích syscall & ret là chuyện nhỏ vì có thể bẫy các cuộc gọi hệ thống. Chúng tôi cũng sẽ cho rằng chúng tôi có một tiện ích sigreturn() hoàn chỉnh theo ý của chúng tôi.
Trước tiên, tiện ích tải số cuộc gọi hệ thống sigreturn trước khi thực hiện cuộc gọi hệ thống. Có thể dễ dàng tìm thấy tiện ích này bằng cách gửi tín hiệu cho quy trình đã theo dõi mà nó đã đăng ký một trình xử lý, nhắc hạt nhân thiết lập khung tín hiệu với tiện ích sigreturn() ở đầu ngăn xếp.

Sử dụng các tiện ích syscall & ret và sigreturn() của chúng tôi và bằng cách giả mạo một số khung tín hiệu, chúng tôi có thể tạo một chuỗi các lệnh gọi hệ thống. Mỗi khung thiết lập các thanh ghi để thực hiện một cuộc gọi hệ thống nhất định trong khi trỏ con trỏ ngăn xếp vào khung tiếp theo, mỗi khung có một tiện ích sigreturn() ở trên cùng.
Cũng giống như với ROP, theo một nghĩa nào đó, con trỏ ngăn xếp hoạt động như một con trỏ hướng dẫn, chỉ là bây giờ con trỏ ngăn xếp luôn trỏ đến trạng thái ngữ cảnh người dùng hoàn chỉnh. Trong khi tất cả chúng ta

do là thực hiện một số lệnh gọi hệ thống (có thể trong một vòng lặp), việc tạo ra hành vi phức tạp lại đơ n giản một cách đáng ngạc nhiên.

Để chứng minh điều này, chúng tôi đã tạo một tấn automa cửa sau chờ một tệp nhất định đư ợc truy cập. Ví dụ, tệp này có thể là một tệp tối nghĩa trên máy chủ web có thể truy cập công khai. Khi tệp này đư ợc đọc, một ổ cắm TCP của trình nghe đư ợc tạo và nếu ai đó kết nối với ổ cắm này, nó sẽ sinh ra một trình bao đư ợc kết nối với ổ cắm này. Nếu không có ai kết nối trong vòng 5 giây, ngư ời nghe sẽ ngừng nghe cho đến khi tệp kích hoạt đư ợc truy cập lại.

Giả định là một bên ở xa có thể dễ dàng khiến hệ thống thực hiện thao tác đọc trên một tệp hiếm khi đư ợc đọc, ví dụ như một tệp ẩn trong thư mục gốc của tài liệu web.
Chỉ sau khi tập tin này đư ợc truy cập, nó mới có thể tạo kết nối với máy thông qua một ổ cắm, một thứ mà nếu không sẽ rất dễ phát hiện.

Để xây dựng cửa hậu, chúng tôi xâu chuỗi một chuỗi các khung tín hiệu thực hiện lệnh gọi hệ thống, dựa vào ngữ nghĩa chặn lệnh gọi hệ thống cho logic của chúng tôi, như thể hiện ở phía bên trái của Hình 6.

Đặc biệt, chúng tôi sử dụng API inotify . API inotify cung cấp cơ chế giám sát các sự kiện hệ thống tệp và cho phép một ngư ời phát hiện quyền truy cập vào các tệp riêng lẻ hoặc thư mục giám sát. Khi một thư mục đư ợc giám sát, inotify sẽ trả về các sự kiện cho chính thư mục đó và cho các tệp bên trong thư mục. Để xác định những sự kiện nào đã xảy ra, một ứng dụng sẽ đọc từ bộ mô tả tệp inotify. Nếu không có sự kiện nào xảy ra, quá trình đọc sẽ bị chặn (cho đến khi có ít nhất một sự kiện xảy ra). API cho phép chúng tôi đợi nhiều sự kiện: đọc, ghi, đóng, thay đổi thuộc tính, v.v. Đối với cửa hậu của chúng tôi, chúng tôi sẽ đợi bất kỳ lần đọc nào đối với tệp tối nghĩa, như ng chúng tôi có thể dễ dàng đợi các sự kiện khác. Do đó, để đợi một tệp đư ợc truy cập, API inotify cung cấp cho chúng tôi một bộ mô tả tệp có khả năng thực hiện chặn đọc sẽ trả về khi tệp đư ợc đọc. Điều này phục vụ như kích hoạt của chúng tôi.

Khi kết quả đọc trở lại, chúng tôi biết rằng ai đó đã truy cập tệp như ng chúng tôi không thể hoàn toàn chắc chắn rằng đó là trình kích hoạt của chúng tôi hay một sự kiện không liên quan. Để tìm hiểu, chúng tôi sẽ sinh ra một luồng chờ một bên từ xa (chủ cửa hậu) kết nối với ổ cắm của chúng tôi. Nếu không có ai kết nối, chúng tôi cho rằng quyền truy cập tệp không liên quan, hãy đóng ổ cắm và quay lại giám sát quyền truy cập tệp. Nếu có một kết nối, chúng tôi sinh ra một vỏ.

Để thực hiện điều này, chuỗi lệnh gọi hệ thống SROP của cửa hậu tuân theo lệnh chặn đọc của bộ mô tả tệp inotify với lệnh gọi hệ thống clone() (lệnh gọi hệ thống cuối cùng ở cột ngoài cùng bên trái trong Hình 6). Lệnh gọi hệ thống này có một thuộc tính hữu ích: sử dụng clone(), có thể gán một ngăn xếp khác cho tiến trình con, trỏ nó đến một trạng thái khác trong máy tự động của chúng ta. Do đó, trong khi cha tiếp tục chờ tệp kích hoạt, con sẽ chịu trách nhiệm về kết nối cửa hậu. Các hành động của đứa trẻ đư ợc hiển thị trong hai cột còn lại của Hình 6. Đứa trẻ, thiết lập đồng hồ báo thức () và lắng nghe trên một ổ cắm, chặn trên Chấp nhận(). Nếu không có ai kết nối, quá trình này sẽ bị báo động tắt và chúng tôi tiếp tục theo dõi các truy cập tệp của tệp kích hoạt của chúng tôi. Nếu ai đó kết nối, báo thức sẽ đư ợc đặt lại và thông qua một loạt các lệnh gọi hệ thống tiếp theo, một trình bao sẽ đư ợc tạo ra.

Chúng tôi đã triển khai cửa hậu và thử nghiệm nó trên một số bản phân phối Linux, bao gồm cả những bản đã đề cập trư ớc đó, cũng như các biến thể 32 bit. Vì không có shellcode trong bộ nhớ nên rất khó tìm ra backdoor để bảo mật
máy quét.

VIII. CHUYỂN ĐẾN KÝ MÃ MẠCH

Mặc dù trên Linux, thông thư ờng có thể sử dụng ROP hoặc SROP để khởi động shellcode truyền thống hơ n, như ng các hệ thống khác như iOS của Apple iPhone chỉ cho phép mã đã ký để chạy nguyên bản. Bản thân việc có thể chạy mã chư a ký trên các hệ thống đó đã trở thành một mục tiêu. Cộng đồng bẻ khóa thư ờng sử dụng các lỗ hổng kernel để vô hiệu hóa các kiểm tra xác minh các chữ ký này. Một phư ơ ng pháp đã đư ợc thử nghiệm tốt để giành quyền kiểm soát kernel là khai thác các giao diện cuộc gọi hệ thống dễ bị tấn công.
Tuy nhiên, điều này không có nghĩa là bản thân các khai thác này phải đư ợc thực thi từ các quy trình đang chạy mã đã ký.

Trong phần này, chúng tôi sẽ mô tả một kỹ thuật cho proxy cuộc gọi hệ thống sử dụng SROP. Kỹ thuật này cung cấp một phư ơ ng pháp rất chung chung để cung cấp các khai thác hạt nhân từ một quy trình đang chạy mã đã ký.

A. Sigreturn trên iOS

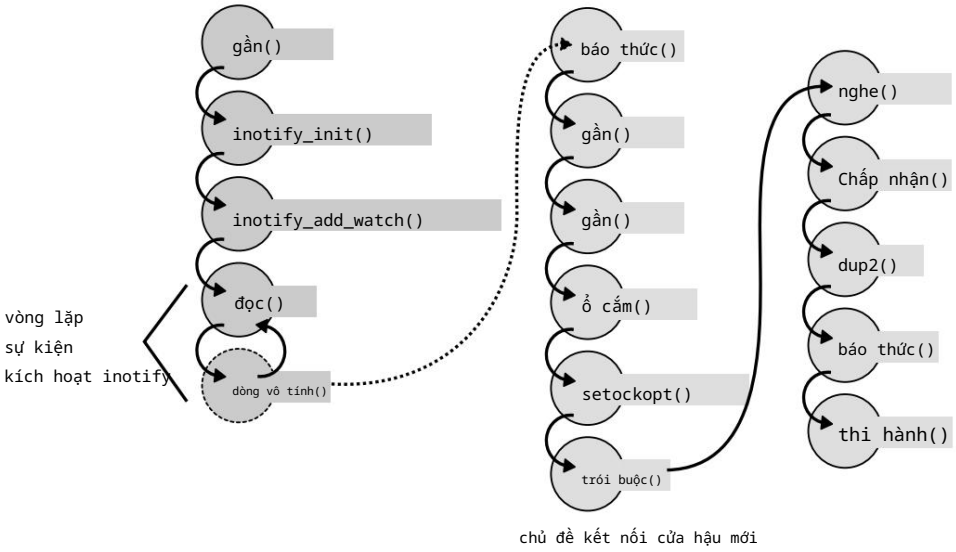
Vì proxy cuộc gọi hệ thống đặc biệt hữu ích cho các hệ thống yêu cầu mã đã ký, chúng tôi đã triển khai tính năng này trên iOS.
Khi nói đến tín hiệu, iOS (giống như Mac OS X) có một chức năng tấm bạt lò xo nhỏ mà từ đó chúng gián tiếp gọi một trình xử lý tín hiệu. Khi trở về từ bộ xử lý tín hiệu này, tấm bạt lò xo tải địa chỉ khung tín hiệu từ ngăn xếp vào r0 (đối số gọi hệ thống đầu tiên) và sau đó gọi sigreturn. Do đó, trong iOS, chúng tôi có thể xác định ngay ba tiện ích hữu ích từ mã xử lý tín hiệu:

1) Sigreturn với con trỏ khung tín hiệu đư ợc tải từ cây rơ m.
2) Sigreturn với con trỏ khung tín hiệu trong lần đầu tiên đối số chức năng.
3) Gọi điện và trả về tiện ích svc 0 ; bx lr Tiện ích thứ

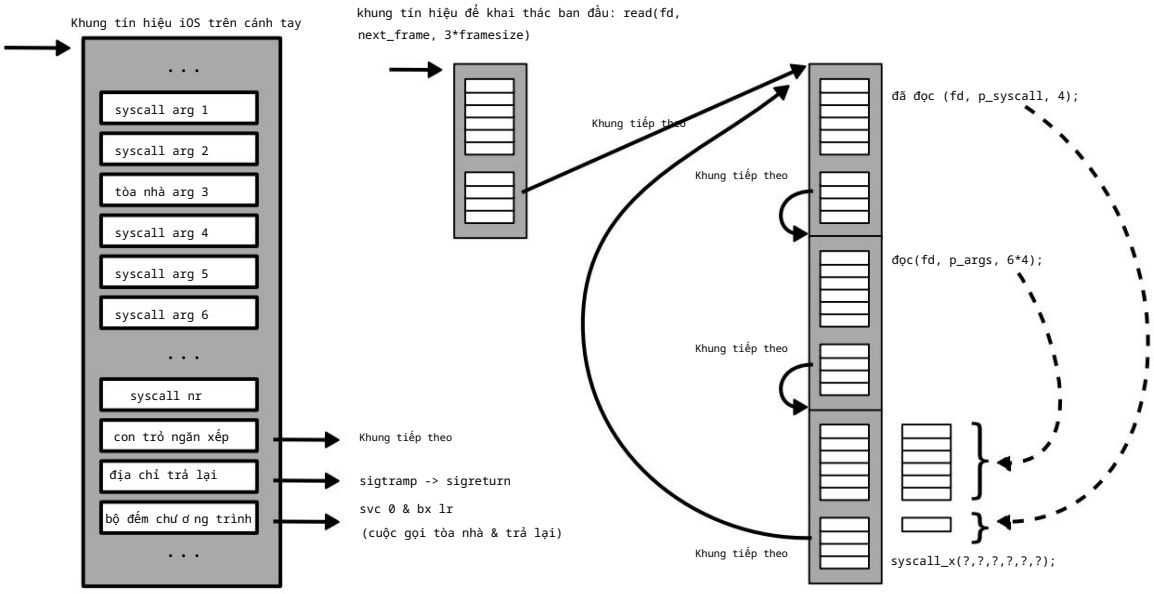
hai rất hữu ích trong trư ờng hợp ghi đè con trỏ hàm. Chúng tôi sẽ sử dụng tiện ích thứ nhất và tiện ích thứ ba cho proxy cuộc gọi hệ thống của chúng tôi. Cần lư u ý rằng, không giống như trên Linux, khung tín hiệu iOS/Mac OSX luôn chứa các con trỏ, do đó, việc thực thi các khung tín hiệu một cách mù quáng mà không biết vị trí của bất kỳ dữ liệu nào là rất khó.

B. Proxy cuộc gọi hệ thống

Mục tiêu của proxy cuộc gọi hệ thống là điều khiển từ xa các cuộc gọi hệ thống đư ợc thực hiện bởi một tiến trình. Trong máy tự động proxy cuộc gọi hệ thống của chúng tôi (Hình 7), chúng tôi tuân theo cùng một mẫu cơ bản như đư ợc mô tả trong Phần VII của chuỗi các cuộc gọi hệ thống. Để khởi động máy tự động, khung tín hiệu ban đầu sẽ di chuyển



Hình 6. Ví dụ về backdoor automaton của chúng tôi. Chủ đề cửa hậu chính thiết lập một danh sách theo dõi inotify . Khi kẻ tấn công khiến một tệp bị truy cập, một luồng con sẽ sinh ra, cho phép kẻ tấn công kết nối với một hệ vỏ cửa hậu.



Hình 7. Một proxy tòa nhà có thể điều khiển đư ợc qua ổ cắm đư ờng ống/tệp/mạng. Khung tín hiệu ban đầu thiết lập lệnh gọi hệ thống đọc để đọc trong một máy tự động tuần hoàn, luân phiên giữa việc đọc số cuộc gọi hệ thống và các đối số của nó từ ổ cắm và thực hiện lệnh gọi hệ thống đã nói.

ngăn xếp và đưa ra lệnh gọi hệ thống đọc trên bộ mô tả tệp do kẻ tấn công kiểm soát, đây có thể là ổ cắm mạng hoặc tệp chẳng hạn. Việc đọc này tải máy tự động lên vị trí mới của ngăn xếp. Bản thân máy tự động là một

vòng lặp của một hoặc nhiều khung tín hiệu thực hiện lệnh gọi hệ thống đọc và một khung tín hiệu sẽ thực hiện lệnh gọi hệ thống tùy ý. Các cuộc gọi hệ thống đọc chịu trách nhiệm điền vào số cuộc gọi hệ thống và các đối số chính xác của chúng trong khung tín hiệu cuối cùng, cho phép kẻ tấn công chỉ cần cung cấp toàn bộ cuộc gọi hệ thống qua một ổ cắm. Các lệnh gọi hệ thống sử dụng con trỏ tới cấu trúc dữ liệu làm đối số có thể được bắt đầu bằng các lệnh gọi mmap và đọc.

IX. GIAO DIỆN CUỘC GỌI HỆ THỐNG LINUX LÀM CHO SROP TURING HOÀN THÀNH

Mặc dù một máy tự động đơn giản xâu chuỗi các lệnh gọi hệ thống bằng cách sử dụng sigreturn là đủ để một cửa hậu thực hiện công việc của nó, nhưng những kẻ tấn công có thể muốn mã hóa logic phức tạp hơn, chẳng hạn như che giấu.

Rõ ràng là nếu chúng ta giữ nguyên nội dung của các khung tín hiệu trong máy tự động của mình trong quá trình thực thi, thì điều tốt nhất chúng ta có thể làm là thực hiện một tập hợp tĩnh các lệnh gọi hệ thống, có thể trong một vòng lặp. Nhưng điều này sẽ thay đổi khi chúng ta cho phép máy tự động của mình ghi lại vào các khung tín hiệu của chính nó, thay đổi các tính toán sắp tới. Trên thực tế, bằng cách sử dụng một máy tự động sửa đổi các đối số hàm và con trỏ ngăn xếp trong các khung ngăn xếp trong tương lai, chúng ta có thể xây dựng một trình thông dịch cho ngôn ngữ hoàn chỉnh Turing.

Ngôn ngữ của chúng tôi có ánh xạ trực tiếp tới “brainfuck”, một ngôn ngữ Turing-đầy đủ nổi tiếng [16]. Về mặt khái niệm, ngôn ngữ của chúng tôi sử dụng ba thanh ghi, bộ đếm chương trình PC, con trỏ bộ nhớ P và một thanh ghi tạm thời được sử dụng để bổ sung 8 bit A. Các thanh ghi này được mô hình hóa như các bộ mô tả tệp. Tệp họ đã mở là /proc/self/mem, trên Linux là một cách đọc và ghi vào không gian địa chỉ của riêng bạn. Việc cộng và trừ các thanh ghi này được thực hiện bằng cách sử dụng lseek. Bộ mô tả tệp PC trỏ đến chương trình ngôn ngữ được giải thích của chúng tôi trong bộ nhớ. Hướng dẫn của nó là địa chỉ của khung tín hiệu thực hiện các hoạt động sau:

1) Bước nhảy (theo sau là phần bù được sử dụng bởi người thân lseek để di chuyển PC).
2) Cộng/trừ con trỏ (theo sau là phần bù được sử dụng bởi một người họ hàng leek để di chuyển P
3) Phép cộng/trừ 8 bit (theo sau là một hằng số để thêm vào byte nằm ở P
4) Nhảy có điều kiện (giống như Nhảy, ngoại trừ việc nó chỉ xảy ra nếu byte tại P bằng 0).
5) Getchar của byte tại P 6)
Putchar của byte tại P 7) Thoát
Hình 8 cho thấy sơ đồ luồng điều

khiển của toàn bộ máy trạng thái. Về mặt cấu trúc, nó có hình dạng giống như một bộ điều phối, có khả năng thực hiện các lệnh của ngôn ngữ. So với brainfuck ban đầu, ngôn ngữ của chúng ta phong phú hơn một chút. Thay vì tăng và giảm, chúng tôi cung cấp thêm tổng quát hơn các số 8 bit.

Gửi lệnh xảy ra bằng cách đọc trên PC, lưu trữ giá trị trong con trỏ ngăn xếp của khung tín hiệu gửi.
Sau đó, khi máy tự động tiến tới một do sigreturn, nó sẽ nhảy tới khung tín hiệu thuộc lệnh mà nó vừa đọc.

Việc đọc tức thời theo hướng dẫn cũng được thực hiện đơn giản bằng cách đọc dữ liệu từ PC. Để thêm con trỏ và để nhảy, chúng tôi sử dụng lseek với đối số SEEK_CUR để di chuyển P và PC tương ứng.

Các bước nhảy có điều kiện được thực hiện giống như một bước nhảy bình thường, ngoại trừ giá trị byte tại P trước tiên được đọc vào byte cao của đối số mô tả tệp cho lseek trên PC. Nếu giá trị tại P không phải là 0, thì lseek sẽ không được thực hiện trên PC mà trên một bộ mô tả tệp rất cao, không tồn tại, khiến lseek thất bại thay vì tìm kiếm, do đó làm cho bước nhảy có điều kiện.

Hoạt động phức tạp nhất hóa ra là việc bổ sung dữ liệu 8 bit của chúng tôi vào bộ nhớ. Đối với điều này, chúng tôi sử dụng bộ đệm 512 byte, mỗi byte chứa đầy modulo 256 của chỉ mục của nó. Để bổ sung, chúng tôi thực hiện tìm kiếm tuyệt đối với thanh ghi A của chúng tôi đến đầu bộ đệm, tiếp theo là 2 tìm kiếm tương đối được cung cấp bởi giá trị hiện tại tại P và ngay lập tức của lệnh hiện tại, kết quả của phép cộng hiện có thể được đọc từ A.

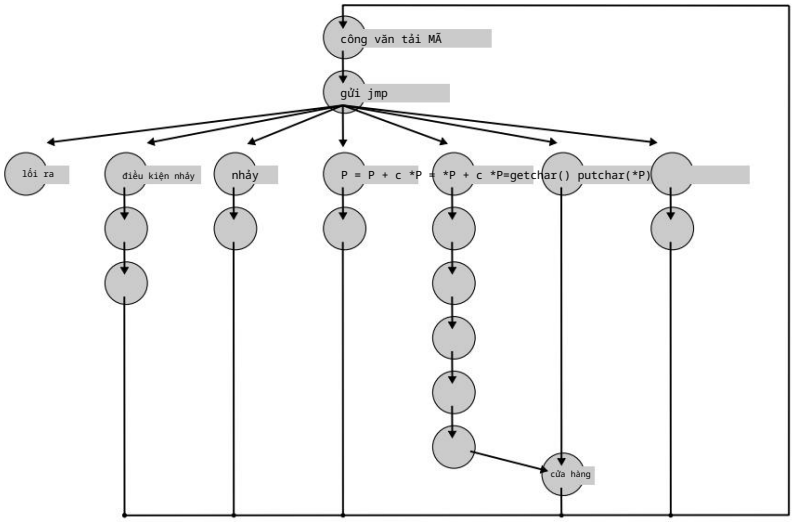
Chúng tôi đã triển khai một trình giả lập cho [16], trình giả lập này trước tiên sẽ dịch nó sang ngôn ngữ máy của chúng tôi và sau đó tiếp tục chạy nó trên máy tự động của chúng tôi.

X. GIẢM THIỂU

Khi triển khai các biện pháp giảm thiểu khai thác, người ta phải cân nhắc sự đánh đổi giữa tổn thất hiệu suất và lợi ích bảo mật.

Chúng tôi nhận ra rằng bản thân chương trình hướng sigreturn không phải là một lỗ hổng có thể khai thác được. Tương tự như ROP, đây là một phương pháp khai thác có thể được sử dụng trong trường hợp có lỗ hổng. Thông thường, ROP có thể được sử dụng thay cho SROP và ngược lại. Ngoài ra, một số biến thể 'SROP phổ quát' mà chúng tôi phát hiện trong Linux trên x86-64 và trên ARM đã được giảm nhẹ trong các nhân Linux gần đây. Trên x86-64, cấu hình mặc định hiện sử dụng mô phỏng vsyscall để loại bỏ các tiện ích tĩnh hữu ích và trên ARM, sigreturn đã bị xóa khỏi trang vectơ tĩnh. Tuy nhiên, điều này không ngăn được chương trình hướng sigreturn được sử dụng như một cửa hậu tàng hình chung. Ngoài ra, nếu có thể tìm thấy tiện ích syscall & return bên trong tệp nhị phân, thì việc sử dụng khai thác SROP chung vẫn dễ dàng hơn so với sử dụng chuỗi ROP cụ thể ở dạng nhị phân. Chúng tôi coi SROP là một trái cây treo thấp cho những người viết khai thác, đáng để giảm thiểu.

Một cách tiếp cận khả thi để loại bỏ lập trình hướng sigreturn như một phương pháp khai thác khả thi sẽ là nhúng một giá trị bí mật do hạt nhân cung cấp vào khung sigreturn.
Khi trở về từ một tín hiệu, hạt nhân sẽ kiểm tra giá trị bí mật này với giá trị mà nó đã ghi trước đó. Nếu



Hình 8. Trình thông dịch hoàn chỉnh Turing của chúng tôi

giá trị khác, hạt nhân có thể chọn để quá trình gặp sự cố. Phư ơ ng pháp này rất giống với stack canaries đã đư ợc áp dụng rộng rãi để bảo vệ chống tràn bộ đệm ngăn xếp. Nó cũng có điểm yếu tư ơ ng tự: Nếu kẻ tấn công có thể rò rỉ giá trị bí mật này, thì kẻ đó có thể sử dụng giá trị đó để giả mạo các khung tín hiệu giả. Rủi ro này có thể đư ợc khắc phục bằng cách làm cho hạt nhân không có giá trị bí mật trong cuộc gọi hệ thống sigreturn, để giá trị chỉ xuất hiện trong không gian ngư ời dùng trong khi trình xử lý tín hiệu đang chạy.

Canary tín hiệu thậm chí có thể là mã xác thực thông điệp mật mã trên khung tín hiệu hoàn chỉnh, để ngăn chặn các sửa đổi tùy ý, ngay cả trong trư ờng hợp rò rỉ bộ nhớ.

Một giải pháp miễn phí có thể là giữ một bộ đếm cho mỗi quy trình trong không gian nhân để theo dõi số lư ợng trình xử lý tín hiệu hiện đang thực thi. Khi gửi tín hiệu, bộ đếm đư ợc tăng lên, trong khi sigreturn làm giảm bộ đếm. Nếu bộ đếm trở nên âm, quá trình này sẽ bị hủy. Mặc dù điều này sẽ hoạt động tốt với các quy trình đơ n luồng, như ng có thể có sự phức tạp với sơ đồ này trong các chư ơ ng trình đa luồng. Giữ một bộ đếm cho mỗi luồng có thể làm hỏng các chư ơ ng trình sử dụng các luồng nhẹ, điều này có thể chuyển ngữ cảnh không gian ngư ời dùng giữa các luồng. Điều này có thể khiến tín hiệu đư ợc gửi trong một luồng và trả về luồng khác. Mặt khác, việc giữ một bộ đếm cho toàn bộ nhóm luồng có thể dẫn đến việc đánh giá quá cao số lư ợng tín hiệu đư ợc gửi khi một trong các luồng thực hiện fork(), không chia sẻ không gian địa chỉ với các luồng khác. Đối với chúng tôi, điều này dư ờng như ít tệ hơ n trong hai tệ nạn.

Cuối cùng, có một câu hỏi là liệu chúng ta có thể thay đổi hành vi của sigreturn mà không phá vỡ các ứng dụng không gian của ngư ời dùng hay không. Các hạt nhân đư ợc cho là có ABI ổn định và vì lý do đó, việc thay đổi hành vi của các cuộc gọi hệ thống không đư ợc thực hiện. Các chư ơ ng trình không gian ngư ời dùng có thể bị hỏng khi chúng phụ thuộc vào hành vi trư ớc đó. Tuy nhiên, sigreturn dư ờng như phải là một trư ờng hợp đặc biệt. Cách hợp pháp duy nhất để gọi sigreturn dư ờng như là khi không gian ngư ời dùng đã đư ợc kernel thiết lập để gọi nó. Ngoài ra, tùy thuộc vào các tính năng của CPU, khung tín hiệu có thể khác nhau, chẳng hạn như các thanh ghi SSE và AVX sẽ đư ợc lư u trên các nền tảng hỗ trợ các tập lệnh này. Vì vậy, mặc dù vị trí của các thanh ghi mục đích chung dư ờng như khá ổn định và có thể truy cập đư ợc từ bên trong bộ xử lý tín hiệu, như ng theo quan điểm của chúng tôi, việc có thể tạo khung ngăn xếp để quay lại theo cách thủ công không đư ợc coi là một phần của ABI.

Tất cả những điều đư ợc xem xét, chúng tôi thực sự cảm thấy rằng việc giảm thiểu chống lại lập trình hư ớng sigreturn như một phư ơ ng pháp khai thác là cần thiết.

XI. PHẦN KẾT LUẬN

Trong bài báo này, chúng ta đã thảo luận về lập trình hư ớng sigreturn, một kỹ thuật mới hoàn chỉnh của Turing để lập trình cho một loại máy lạ mới lạ. Lập trình hư ớng Sigreturn là một kỹ thuật chung, như chúng tôi đã chứng minh bằng cách sử dụng nó để khai thác, mở cửa sau và vư ợt qua chữ ký mã. Hơ n nữa, nó hoạt động trên nhiều hệ điều hành và kiến trúc khác nhau. Đối với một số hệ thống này, lập trình hư ớng sigreturn cho phép khai thác mà không cần bất kỳ kiến thức chính xác nào về tệp thực thi.

Hơ n nữa, việc khai thác có thể tái sử dụng, vì nó hoàn toàn không phụ thuộc nhiều vào quá trình của nạn nhân.

Lập trình hư ớng Sigreturn đại diện cho một kỹ thuật di động, thuận tiện để lập trình mã tùy ý ngay cả trong các máy đư ợc bảo vệ mạnh mẽ. Số lư ợng tiện ích cần thiết là tối thiểu và trong nhiều hệ thống, những tiện ích đó nằm ở một vị trí cố định. Như vậy, kỹ thuật này đư ợc xếp hạng trong số những quả treo thấp nhất hiện có sẵn cho những kẻ tấn công trên các hệ thống UNIX. Điều quan trọng cần nhấn mạnh là ngay cả khi các hạt nhân đư ợc vá để loại bỏ các tiện ích có vị trí cố định này,

tính hữu dụng của backdoor không hề suy giảm. Tóm lại, chúng tôi tin rằng lập trình hư ớng sigreturn là một bổ sung mạnh mẽ cho kho vũ khí của những kẻ tấn công.

SỰ NHÌN NHẬN

Chúng tôi cảm ơ n những ngư ời đánh giá ẩn danh vì phản hồi tuyệt vời của họ. Công việc này đư ợc hỗ trợ bởi dự án ERC StG “Rosetta” và dự án EU FP7 “SYSSEC”.

NGƯ ỜI GIỚI THIỆU

[1] Sandeep Bhatkar, R. Sekar, và Daniel C. DuVarney. Các kỹ thuật hiệu quả để bảo vệ toàn diện khỏi khai thác lỗi bộ nhớ. Trong Kỷ yếu của hội nghị lần thứ 14 về USENIX Security Symposium, SSYM'05, 2005.

[2] Sandeep Bhatkar, R. Sekar, và Daniel C. DuVarney. Các kỹ thuật hiệu quả để bảo vệ toàn diện khỏi khai thác lỗi bộ nhớ. Trong Kỷ yếu hội nghị lần thứ 14 về USENIX Security Symposium - Tập 14, SSYM'05, trang 17–17, Berkeley, CA, USA, 2005. Hiệp hội USENIX.

[3] Tyler Bletsch, Xuxian Jiang, Vince W. Freeh, và Zhenkai Liang. Lập trình theo hư ớng nhảy: một kiểu tấn công tái sử dụng mã mới. Trong Kỷ yếu của Hội nghị chuyên đề ACM lần thứ 6 về Bảo mật thông tin, máy tính và truyền thông, ASIACCS '11, trang 30–40, New York, NY, Hoa Kỳ , 2011. ACM.

[4] Bóng đèn và Kil3r. Bỏ qua StackGuard và Stack Shield. http://www.phrack.org/issues.html?issue=56&id= 5article, tháng 1 năm 2000.

[5] Ping Chen, Hai Xiao, Xiaobin Shen, Xinchun Yin, Bing Mao và Li Xie. Drop: Phát hiện mã độc lập trình hư ớng trở lại. Trong Kỷ yếu của Hội nghị Quốc tế lần thứ 5 về Bảo mật Hệ thống Thông tin, ICISS '09, trang 163–177, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] Lucas Davi, Ahmad-Reza Sadeghi, và Marcel Winandy. Ropdefender: một công cụ phát hiện để bảo vệ chống lại các cuộc tấn công lập trình hư ớng trở lại. Trong Kỷ yếu của Hội nghị Chuyên đề ACM lần thứ 6 về Bảo mật Thông tin, Máy tính và Truyền thông, ASIACCS '11, trang 40–51, New York, NY, USA, 2011. ACM.

[7] Thomas Dullien. Máy khai thác và trạng thái: Lập trình kết hợp 'máy lạ', đư ợc xem lại. Trong Hội nghị Thâm nhập, Miami, FLA, tháng 4 năm 2011.

[8] Brandon Edwards. làm gì? sau đó ai là điện thoại? (khai thác dấu hoa thị liên quan đến cve-2012-5976). http://blog.exodusintel. com/tag/asterisk-khai thác/.

[9] Sergey Bratus Julian Bangert. Trang lỗi giải phóng quân hay đạt đư ợc trong bản dịch. http://events.ccc.de/congress/2012/ Fahrplan/events/5265.en.html.

[10] Chongkyung Kil, Jinsuk Jun, Christopher Bookholt, Jun Xu, và Peng Ning. Hoán vị bố cục không gian địa chỉ (aslp): Dành cho sự ngẫu nhiên hóa chi tiết của phần mềm hàng hóa. Trong Kỷ yếu của Hội nghị Ứng dụng Bảo mật Máy tính Thư ờng niên lần thứ 22, ACSAC '06, trang 339–348, Washington, DC, USA, 2006. IEEE Computer Society.

[11] Jinku Li, Zhi Wang, Xuxian Jiang, Michael Grace, và Sina Bahram. Đánh bại rootkit hư ớng trở lại với hạt nhân "không trả lại". Trong Kỷ yếu hội nghị châu Âu lần thứ 5 về Hệ thống máy tính, EuroSys '10, trang 195–208, New York, NY, USA, 2010. ACM.

[12] David Litchfield. Đánh bại cơ chế chống tràn bộ đệm dựa trên ngăn xếp của máy chủ microsoft windows 2003. 2003.

[13] Vincenzo Lozzo, Tim Kornau, và Ralf-Philipp Weinmann. Mọi ngư ời bình tĩnh, đây là một roppery! Trong Kỷ yếu của BlackHat USA, Las Vegas, USA, 2010.

[14]Microsoft. Ngăn chặn thực thi dữ liệu.

[15] Matt Miller. Ngăn chặn việc khai thác ghi đè seh. Tạp chí Thông tin, 5, 2006.

[16] Đô thị Müller. Brainfuck–một ngôn ngữ lập trình hoàn chỉnh gồm tám lệnh. Có tại địa chỉ Internet http://www.muppetlabs.com/ breadbox/bf/, 1993.

[17] Kaan Onarlioglu, Leyla Bilge, Andrea Lanzi, Davide Balzarotti, và Engin Kirda. G-free: đánh bại chư ơ ng trình định hư ớng trả lại thông qua các tệp nhị phân không có tiện ích. Trong Kỷ yếu của Hội nghị Ứng dụng Bảo mật Máy tính Thư ờng niên lần thứ 26 (ACSAC), ACSAC '10, trang 49–58. ACM, tháng 12 năm 2010.

[18] Vasilis Pappas. kbouncer: Eïn˜C Giảm thiểu rủi ro minh bạch và khoa học. Trong Usenix Security, ngư ời chiến thắng Giải thư ởng Microsoft BlueHat., 2013.

[19] Vasilis Pappas, Michalis Polychronakis, và Angelos D. Keromytis. Phá vỡ các tiện ích: Cản trở chư ơ ng trình hư ớng về lợi nhuận bằng cách sử dụng mã ngẫu nhiên tại chỗ. Trong Proceedings of the 2012 IEEE Symposium on Security and Privacy, SP '12, trang 601–615, Washington, DC, USA, 2012. IEEE Computer Society.

[20] Dự án PaX. Ngẫu nhiên bố trí không gian địa chỉ. http://pax.grsecurity.net/docs/aslr.txt, 2001.

[21] Dennis Ritchie. Phiên bản Unix 6 – signal.s. http://minnie.tuhs.org/cgi-bin/utree.pl?file=V6/usr/source/s5/signal.s, 1975.

[22] Giampaolo Fresi Roglia, Lorenzo Martignoni, Roberto Paleari và Danilo Bruschi. Phẫu thuật trở lại lib (c) đư ợc định dạng ngẫu nhiên. Trong Kỷ yếu của Hội nghị Ứng dụng Bảo mật Máy tính Thư ờng niên 2009, ACSAC '09, trang 60–69, Washington, DC, Hoa Kỳ , 2009. IEEE Computer Society.

[23] Edward J Schwartz, Thanassis Avgerinos, và David Brum ley. Q: Quá trình làm cứng khai thác đư ợc thực hiện dễ dàng. Trong Kỷ yếu của Hội nghị chuyên đề về bảo mật USENIX lần thứ 20, 2011.

[24] Hovav Shacham. Hình học của xác thịt vô tội trên xư ơ ng: return-into-libc mà không cần gọi hàm (trên x86). Trong Kỷ yếu hội nghị ACM lần thứ 14 về Bảo mật máy tính và truyền thông, CCS'07, 2007.

[25] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, và Dan Boneh. Về hiệu quả của ngẫu nhiên hóa không gian địa chỉ. Trong Kỷ yếu hội nghị ACM lần thứ 11 về Bảo mật máy tính và truyền thông (CCS'04), 2004.

[26] Rebecca Shapiro. Việc chăm sóc và cho ăn những cỗ máy kỳ lạ được tìm thấy trong siêu dữ liệu thực thi. http://events.ccc.de/congress/ 2012/Fahrplan/events/5195.en.html.

[27] Kevin Z. Snow, Fabian Monrose, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen và Ahmad-Reza Sadeghi.
Tái sử dụng mã đúng lúc: Về hiệu quả của ngẫu nhiên hóa bố cục không gian địa chỉ chi tiết. Trong Kỷ yếu của Hội nghị chuyên đề về Bảo mật và Quyền riêng tư của IEEE năm 2013, SP '13, trang 574–588, Washington, DC, Hoa Kỳ , 2013. Hiệp hội Máy tính IEEE.

[28] Nhà thiết kế năng lượng mặt trời. Tìm hiểu về ngăn xếp không thể thực thi (và khắc phục). http://seclists.org/bugtraq/1997/Aug/63, tháng 8 năm 1997.

[29] Alexander Sotirov. Heap phong thủy trong javascript. Mũ Đen Châu Âu, 2007.

[30] Julien Vanegue. Thách thức lớn về khai thác tự động, những câu chuyện về những cỗ máy kỳ lạ. Trong hội nghị H2C2, Sao Paulo, Brazil, tháng 10 năm 2013.

[31] Tielei Wang, Kangjie Lu, Long Lu, Simon Chung, và Wenke Lee. Jekyll trên ios: khi các ứng dụng lành tính trở nên xấu xa. Trong Kỷ yếu của hội nghị USENIX lần thứ 22 về Bảo mật, SEC'13, trang 559–572, Berkeley, CA, Hoa Kỳ , 2013. Hiệp hội USENIX.

[32] Michal Zalewski. "cung cấp tín hiệu để giải trí và kiếm lợi nhuận". http://lcamtuf.coredump.cx/signals.txt, 2001.