**BISTRO: Binary Component Extraction and Embedding for Software Security Applications**

Zhui Deng, Xiangyu Zhang & Dongyan Xu - Lecture Notes in Computer Science ((LNSC,volume 8134)), pp 200–218

BISTRO: Trích xuất và Nhúng Thành phần Nhị phân

Giới thiệu: BISTRO là một hệ thống cho phép trích xuất và nhúng các thành phần chức năng từ các chương trình nhị phân mà không cần thông tin biểu tượng, thông tin di chuyển, hoặc sự hỗ trợ của trình biên dịch. BISTRO đảm bảo tính chính xác về chức năng của cả thành phần được trích xuất và chương trình nhị phân được mở rộng (với thành phần được nhúng) bằng cách vá chúng một cách hệ thống.

Tổng quan: BISTRO bao gồm hai thành phần chính: trích xuất nhị phân và mở rộng nhị phân. Trích xuất nhị phân chịu trách nhiệm trích xuất một thành phần chức năng được chỉ định từ một chương trình nhị phân gốc, bao gồm cả mã và dữ liệu của thành phần chức năng. Mở rộng nhị phân chịu trách nhiệm mở rộng chương trình nhị phân mục tiêu để tạo “không gian” cho việc nhúng một thành phần chức năng. Kết quả là một chương trình nhị phân “mở rộng” P’ = P + c sẵn sàng để thực thi.

Giả Định: Có một số giả định được đặt ra, bao gồm việc người dùng, chứ không phải BISTRO, sẽ xác định tính phù hợp về mặt ngữ nghĩa khi nhúng thành phần c vào chương trình mục tiêu P. Hơn nữa, họ sẽ quyết định vị trí cụ thể để chèn c. Việc này có thể được thực hiện thông qua kỹ thuật đảo ngược trên P. Một giả định khác là việc xác định c trong chương trình nhị phân gốc Q được thực hiện trước thông qua các kỹ thuật thủ công hoặc tự động. Cuối cùng, giả định rằng các chương trình nhị phân có thể được phân tích đúng cách (ví dụ, bởi IDA-Pro) trước khi được chuyển đến BISTRO. Các giả định này hỗ trợ việc thử nghiệm với số lượng lớn chương trình nhị phân thực tế, không cần xử lý các chương trình nhị phân bị mã hóa hoặc tự sửa đổi.

Thuật toán cơ bản: Mô tả thuật toán cơ bản được sử dụng bởi BISTRO để thực hiện việc kéo giãn hoặc co lại file nhị phân.

Giả định: Các giả định được đặt ra cho thuật toán, bao gồm không có chuyển giao điều khiển gián tiếp và dữ liệu toàn cục được tham chiếu trực tiếp trong một lệnh.

Thách thức thực tế: Đề cập đến các thách thức khi áp dụng thuật toán này vào phần mềm quy mô lớn thực tế và cách BISTRO giải quyết chúng.

Kết quả đánh giá: Cung cấp thông tin về hiệu suất và độ chính xác của BISTRO khi áp dụng thuật toán này vào các ứng dụng Windows thực tế và mã nguồn mở.

Xử lý Chuyển Tiếp - Gián Tiếp. Xử lý các lệnh nhảy và gọi gián tiếp là một trong những thách thức chính trong thiết kế của BISTRO. Khó khăn nằm ở việc mục tiêu nhảy/gọi không thể biết trước một cách tĩnh và do đó khó để vá. Giải pháp đơn giản là xác định và vá bất kỳ giá trị hằng số nào trong nhị phân mà có vẻ là mục tiêu nhảy/gọi. Nhưng điều này không an toàn vì những giá trị đó có thể không phải là mục tiêu nhảy/gọi.

Nếu một nhị phân có bảng tái định vị và nó không thực hiện bất kỳ quản lý bố cục không gian địa chỉ tự quản lý nào như thông qua một packer, bảng tái định vị sẽ cung cấp vị trí của tất cả các giá trị hằng số là mục tiêu nhảy/gọi cho BISTRO để vá chúng, do đó dẫn đến một giải pháp âm thanh và hoàn chỉnh cho việc kéo dãn/thu nhỏ nhị phân. Tuy nhiên, bảng tái định vị có thể không có mặt hoặc chứa các mục nhập giả mạo trong nhị phân di sản và phần mềm độc hại. Do đó, cho phần còn lại của bài báo, chúng tôi không giả định sự hiện diện của bảng tái định vị trong thiết kế và đánh giá của chúng tôi. Tiếp theo, chúng tôi sẽ mô tả cách xử lý các cuộc gọi gián tiếp trong Mục 4.1 và các nhảy gián tiếp trong Mục 4.2.

4.1 Xử lý Cuộc Gọi Gián Tiếp Các cuộc gọi gián tiếp rất phổ biến trong nhị phân hiện đại để tận dụng tính linh hoạt của con trỏ hàm. Chúng tôi đã thảo luận về khó khăn của việc xử lý con trỏ hàm ngay từ đầu Mục 4. Thực tế, có một tình huống thách thức hơn, trong đó một nhị phân có thể chuyển địa chỉ hàm của nó cho các hàm thư viện bên ngoài gọi lại các hàm được cung cấp (ví dụ, một hàm người dùng cmp() được cung cấp làm tham số cho một hàm thư viện bên ngoài qsort()). Trong trường hợp này, nếu một điểm nhập hàm đã thay đổi do kéo dãn hoặc thu nhỏ, việc gọi nó sẽ nằm ngoài phạm vi của nhị phân và do đó nằm ngoài tầm kiểm soát của chúng tôi. Rất khó để vá các tham số con trỏ hàm gọi lại trước khi chúng được chuyển cho thư viện để sử dụng vì hai lý do. Đầu tiên, một con trỏ hàm có thể không trực tiếp xuất hiện như một tham số. Nó có thể là thành viên của một cấu trúc được chuyển cho một thư viện bên ngoài. Nó thậm chí có thể yêu cầu một số lớp gián tiếp con trỏ để truy cập giá trị của nó. Vá điều đó là thách thức. Thứ hai, đối với nhiều hàm thư viện bên ngoài, chúng tôi không thể giả định sự có mặt của định nghĩa nguyên mẫu của họ, do đó khó biết loại tham số của họ.

Để xử lý các cuộc gọi gián tiếp bao gồm các hàm gọi lại, chúng tôi đề xuất kéo dãn nhị phân mục tiêu để tạo ra các lỗ nhỏ tại điểm nhập của mỗi hàm có thể là mục tiêu của một cuộc gọi gián tiếp. Những lỗ này được gọi là neo; chúng không nên được di chuyển trong quá trình kéo dãn/thu nhỏ. Bên trong một neo, chúng tôi đặt một lệnh nhảy chuyển đến địa chỉ mới được ánh xạ của nó trong nhị phân đã kéo dãn/thu nhỏ, đó là điểm nhập mới của hàm. Như vậy, chúng tôi không cần phải xác định hoặc vá bất kỳ con trỏ.

4.1

Thách thức: Xử lý các cuộc gọi gián tiếp trong mã nhị phân là một thách thức lớn do không thể biết trước được đích đến của cuộc gọi.

Giải pháp Anchor: BISTRO giải quyết vấn đề này bằng cách tạo ra các “anchor” tại điểm nhập của mỗi hàm có thể là mục tiêu của cuộc gọi gián tiếp, không di chuyển trong quá trình co giãn mã nhị phân.

Thuật toán Anchor-Based: Thuật toán này chia quá trình co giãn thành hai giai đoạn, tạo ra không gian cho các đoạn mã được chèn vào hoặc loại bỏ, sau đó thêm các anchor.

Tối ưu hóa Anchor: Chỉ tạo anchor cho một tập hợp con các hàm có thể là mục tiêu của cuộc gọi gián tiếp, giảm thiểu số lượng anchor cần thiết.

4.2.

Xử lý Nhảy Gián Tiếp Nhảy gián tiếp khác với gọi gián tiếp vì mục tiêu nhảy có thể không phải là điểm nhập của hàm, mà có thể ở bất cứ đâu trong mã nhị phân. Nếu chúng ta áp dụng phương pháp neo, sẽ cần quá nhiều neo. Người ta có thể sử dụng một số phỏng đoán như việc nhảy gián tiếp thường nhận mục tiêu từ bảng nhảy và do đó chỉ cần sửa đổi các mục trong bảng nhảy. Tuy nhiên, điều này không an toàn vì khó xác định ranh giới của bảng nhảy. Một bảng nhảy có thể không phân biệt được với dữ liệu thông thường. Do đó, chúng tôi đề xuất một phương pháp khác. Cụ thể, chúng tôi chèn một đoạn mã ngay trước mỗi lệnh nhảy gián tiếp để dịch địa chỉ mục tiêu nhảy sang địa chỉ đã được ánh xạ trong mã nhị phân đã được mở rộng tại thời gian chạy, như trong ví dụ dưới đây.

Lưu ý rằng ví dụ chỉ để minh họa. Trong thực tế, chúng tôi sử dụng băm hoàn hảo cho việc tra cứu địa chỉ và bảo toàn thanh ghi cờ trong quá trình dịch. Vì ánh xạ từng byte sang byte đã được tính toán hoàn chỉnh trong Thuật toán 1, bất kỳ mục tiêu nhảy gián tiếp nào cũng có thể được dịch và xử lý đúng cách bằng phương pháp này. Quan sát rằng cần thêm các lệnh bổ sung để thực hiện dịch. Chúng tôi có thể dễ dàng xử lý điều này bằng cách mở rộng mã nhị phân để chứa những lệnh đó.

Cắt Giảm Bộ Đích Nhánh Mặc dù việc dịch sử dụng ánh xạ hoàn chỉnh đảm bảo an toàn, nó cũng gây ra đáng kể tiêu hao bộ nhớ. Mỗi byte trong mã nhị phân gốc đòi hỏi 4 byte để biểu diễn địa chỉ đã được ánh xạ của nó. Trên thực tế, chúng ta chỉ cần một tập hợp con của ánh xạ: mã nhị phân đã được mở rộng/mở rộng sẽ an toàn miễn là ánh xạ chứa dịch cho mọi mục tiêu nhảy gián tiếp có thể có.

Chúng tôi xây dựng tập hợp con với tiêu chí sau: bất kỳ giá trị dữ liệu bốn byte nào hoặc bất kỳ toán hạng bốn byte tức thì nào trong một lệnh không được sử dụng trực tiếp làm địa chỉ được coi là mục tiêu nhảy gián tiếp có thể có, nếu giá trị đó nằm trong phạm vi của một phần mã. Chúng tôi tiếp tục cắt giảm tập hợp bằng cách loại bỏ các giá trị chỉ đến giữa một lệnh hiện có. Lưu ý rằng chiến lược này an toàn cho nhảy dài/set vì mục tiêu nhảy của chúng được thu thập tại thời gian chạy.

Bộ Dịch Băm Hoàn Hảo Thách thức còn lại là đạt được dịch nhanh. Lưu ý rằng sau khi cắt giảm, bộ đích nhảy trở thành một tập hợp thưa thớt trong không gian địa chỉ. Là một sự thỏa hiệp giữa tiêu hao bộ nhớ và tiêu hao thời gian chạy, chúng tôi chọn sử dụng băm hoàn hảo cho việc dịch. Một hàm băm hoàn hảo ánh xạ một tập hợp các khóa sang một tập hợp khác các giá trị số nguyên mà không có bất kỳ va chạm nào. Nó đảm bảo thời gian dịch O(1). Chúng tôi sử dụng gperf để tạo hàm băm hoàn hảo cho tập hợp mục tiêu nhảy và biên dịch nó thành một tệp .obj có thể liên kết được có thể nhúng vào mã nhị phân mục tiêu thông qua BISTRO.

Một hàm băm hoàn hảo có thể yêu cầu nhiều không gian hơn N khóa để đạt được thời gian dịch O(1). Trong thực tế, chúng tôi thấy kích thước của hàm băm hoàn hảo được tạo ra là chấp nhận được. Ví dụ, đối với 3635 mục tiêu nhảy của Adobe Reader, hàm băm được tạo ra khoảng 152KB, khoảng 11% kích thước của tệp Adobe Reader.

4.2

Mục tiêu: Xử lý các nhảy gián tiếp trong việc kéo dài/ thu nhỏ file nhị phân.

Thách thức: Khó xác định mục tiêu nhảy gián tiếp vì chúng có thể ở bất cứ đâu trong file nhị phân.

Giải pháp: Sử dụng đoạn mã trước mỗi lệnh nhảy gián tiếp để dịch địa chỉ mục tiêu nhảy sang địa chỉ mới trong file nhị phân đã được kéo dài.

Cải tiến: Sử dụng bảng băm hoàn hảo (perfect hashing) để tra cứu địa chỉ mục tiêu một cách nhanh chóng và chính xác khi thực thi.

5.

Xử lý Tham Chiếu Dữ Liệu: Khi thực hiện việc kéo dãn hoặc rút gọn nhị phân, có thể xảy ra việc di chuyển các mục dữ liệu. Điều quan trọng là đảm bảo tính chính xác của các lệnh tham chiếu đến dữ liệu đó.

Tiếp Cận Với Binary Mục Tiêu: Đối với binary mục tiêu, thường phức tạp và liên quan đến nhiều tham chiếu dữ liệu toàn cục. Để xử lý vấn đề này một cách hiệu quả, dữ liệu trong binary được nhóm thành các khối dữ liệu liên tục. Nếu một khối dữ liệu có thể được truy cập gián tiếp, chúng ta sẽ đảm bảo rằng khối đó không được di chuyển bằng cách sử dụng neo (anchors) để giữ chúng ở vị trí ban đầu.

Tiếp Cận Với Thành Phần Được Nhúng: Ngược lại, các mục dữ liệu được trích xuất như một phần của thành phần nhúng có khả năng cao sẽ được di chuyển. Ví dụ, nếu chúng được phân tán rải rác trong không gian địa chỉ, công cụ trích xuất BISTRO sẽ thu gọn chúng vào một khối liên tục, gây ra việc di chuyển.

Phương Pháp Dịch Địa Chỉ Tham Chiếu Dữ Liệu: Chúng ta áp dụng một phương pháp tương tự như kỹ thuật dịch địa chỉ nhảy gián tiếp để dịch địa chỉ tham chiếu dữ liệu. Trước khi dịch, chúng ta thêm một so sánh để tránh dịch các truy cập bộ nhớ ngăn xếp hoặc heap. Theo kinh nghiệm của chúng tôi, chỉ có 2% truy cập bộ nhớ động cần được dịch. Chúng tôi cũng sử dụng quét lỗ hổng tĩnh để xác định các tham chiếu chắc chắn truy cập vào ngăn xếp và tránh can thiệp vào chúng hoàn toàn.

6.

Hiệu Suất BISTRO: Đánh giá hiệu suất của BISTRO bằng cách mở rộng các ứng dụng Windows thực tế và các binary của SPEC CPU 2000. Kết quả cho thấy overhead về không gian và thời gian chạy là tối thiểu.

Thử Nghiệm và Kết Quả: Thử nghiệm trên laptop Dell Inspiron 15R với CPU Intel Core i5 và 4GB RAM chạy Windows 7 SP1. Các ứng dụng được mở rộng không có sự chậm trễ đáng kể khi sử dụng.

Chiến Lược Pruning: Sử dụng chiến lược pruning để giảm số lượng mục tiêu nhảy gián tiếp và điểm neo dữ liệu, làm giảm overhead về bộ nhớ.

Kết Luận: BISTRO hiệu quả trong việc xử lý các binary không thể di chuyển, với overhead thấp về không gian và thời gian chạy, đồng thời duy trì chức năng của binary gốc.

7.

Thách thức: Phần này thảo luận về những thách thức đặc biệt khi thực hiện việc trích xuất và nhúng thành phần nhị phân, đặc biệt là khi không có thông tin tái định vị hoặc bảng tái định vị có thể bị thiếu hoặc chứa các mục nhập giả mạo trong các tệp nhị phân cũ hoặc malware.

Giải pháp: BISTRO giải quyết những thách thức này bằng cách tạo ra các “neo” (anchors) tại điểm nhập của các hàm có thể là mục tiêu của các lệnh gọi gián tiếp, và sử dụng một thuật toán băm hoàn hảo để dịch địa chỉ mục tiêu của các lệnh nhảy gián tiếp một cách động.

Hiệu suất: Các kết quả đánh giá cho thấy BISTRO hoạt động hiệu quả với chi phí thời gian và không gian thấp, ngay cả khi áp dụng cho các ứng dụng Windows lớn và phức tạp.

Giả định: BISTRO dựa trên một số giả định như việc người dùng sẽ xác định trước tính phù hợp về mặt ngữ nghĩa của việc nhúng thành phần c vào chương trình mục tiêu P, và việc xác định c trong tệp nhị phân gốc Q được thực hiện trước thông qua kỹ thuật thủ công hoặc tự động.

9.

Chúng tôi đã phát triển một cặp nguyên tắc xử lý chương trình nhị phân mới được gọi là BISTRO để trích xuất và đóng gói lại một thành phần chức năng từ một chương trình nhị phân; và để nhúng một thành phần chức năng vào một chương trình nhị phân mục tiêu. Chúng tôi giải quyết các thách thức của việc vá các hướng dẫn chuyển điều khiển và tham chiếu dữ liệu để bảo toàn ngữ nghĩa của cả thành phần được trích xuất và chương trình nhị phân đã được kéo dãn, đặc biệt là các cuộc gọi và nhảy gián tiếp. BISTRO gây ra thời gian chạy thấp (trung bình 1.9%) và không gian bộ nhớ thấp (trung bình 11%). Các hoạt động trích xuất và nhúng là hiệu quả cao, với thời gian dưới 1.5 giây cho hầu hết các trường hợp. Chúng tôi đã áp dụng BISTRO vào hai kịch bản ứng dụng bảo mật, chứng minh tính hiệu quả, độ chính xác và tính linh hoạt của nó.

Tóm tắt của tóm tắt:

BISTRO (Binary componenT extracTor and Re-binder) là một hệ thống đột phá cho phép trích xuất và nhúng các thành phần chức năng từ/vào các chương trình nhị phân phức tạp một cách chính xác, mà không cần thông tin biểu tượng, thông tin di chuyển hay sự hỗ trợ của trình biên dịch. Điều này rất quan trọng vì nhiều chương trình nhị phân cũ và phần mềm độc hại thường thiếu những thông tin này.

BISTRO bao gồm hai thành phần chính: trích xuất nhị phân và mở rộng nhị phân. Trích xuất nhị phân có nhiệm vụ trích xuất một thành phần chức năng cụ thể từ chương trình nhị phân gốc, bao gồm cả mã và dữ liệu của thành phần đó. Mở rộng nhị phân có trách nhiệm kéo giãn chương trình nhị phân mục tiêu để tạo ra không gian cần thiết cho việc nhúng thành phần chức năng trích xuất được.

BISTRO dựa trên một số giả định như người dùng sẽ xác định tính phù hợp về ngữ nghĩa khi nhúng thành phần vào chương trình mục tiêu, quyết định vị trí chèn thành phần, và xác định thành phần cần trích xuất từ chương trình gốc trước đó bằng kỹ thuật thủ công hoặc tự động. BISTRO cũng giả định rằng các chương trình nhị phân có thể được phân tích đúng cách trước khi đưa vào hệ thống.

Một trong những thách thức lớn nhất của BISTRO là xử lý các cuộc gọi và nhảy gián tiếp, trong đó mục tiêu cuộc gọi/nhảy không thể xác định trước một cách tĩnh. Điều này rất phổ biến trong các chương trình nhị phân hiện đại để tận dụng tính linh hoạt của con trỏ hàm và bảng nhảy. Nếu không xử lý đúng, sẽ dẫn đến thay đổi ngữ nghĩa của chương trình.

Để giải quyết vấn đề cuộc gọi gián tiếp, BISTRO sử dụng phương pháp tạo "neo" (anchors) tại điểm nhập của mỗi hàm có khả năng là mục tiêu gọi gián tiếp. Các neo này không được di chuyển trong quá trình kéo dãn/thu nhỏ. Bên trong mỗi neo là một lệnh nhảy ngắn chuyển đến địa chỉ mới của hàm sau khi kéo dãn/thu nhỏ. Nhờ vậy, BISTRO không cần phải xác định hay vá các con trỏ hàm.

Đối với vấn đề nhảy gián tiếp, BISTRO sử dụng một đoạn mã nhỏ được chèn ngay trước mỗi lệnh nhảy gián tiếp để dịch địa chỉ mục tiêu nhảy sang địa chỉ mới tại thời điểm chạy. Để tối ưu hóa, BISTRO chỉ dịch một tập hợp con các địa chỉ mục tiêu nhảy gián tiếp có khả năng, thông qua phân tích tĩnh. Ngoài ra, BISTRO sử dụng bảng băm hoàn hảo (perfect hashing) cho việc tra cứu và dịch địa chỉ nhanh chóng, hiệu quả.

Một vấn đề khác BISTRO phải đối mặt là xử lý các tham chiếu dữ liệu khi có sự di chuyển của các khối dữ liệu do việc kéo dãn/thu nhỏ chương trình. Với chương trình nhị phân mục tiêu, BISTRO nhóm dữ liệu thành các khối liên tục và sử dụng neo để giữ nguyên vị trí các khối có thể truy cập gián tiếp, không cho phép di chuyển chúng.

Với thành phần được nhúng, dữ liệu thường bị di chuyển, nên BISTRO áp dụng phương pháp dịch địa chỉ tham chiếu dữ liệu tương tự kỹ thuật dịch địa chỉ nhảy gián tiếp. Trước mỗi lệnh tham chiếu dữ liệu, BISTRO chèn đoạn mã kiểm tra và dịch địa chỉ tham chiếu nếu cần thiết. Ngoài ra, BISTRO sử dụng kỹ thuật quét lỗ hổng tĩnh để xác định và tránh can thiệp vào các tham chiếu truy cập ngăn xếp hoặc bộ nhớ heap.

Kết quả đánh giá cho thấy BISTRO gây ra overhead rất nhỏ về không gian (trung bình 11%) và thời gian chạy (trung bình 1.9%) khi áp dụng cho các ứng dụng Windows thực tế lớn và SPEC CPU 2000. Quá trình trích xuất và nhúng thành phần diễn ra nhanh chóng, trong vòng dưới 1.5 giây cho hầu hết các trường hợp.

BISTRO đã được áp dụng thành công trong hai kịch bản ứng dụng bảo mật khác nhau, chứng minh tính hiệu quả, độ chính xác và sự linh hoạt của nó trong việc xử lý các chương trình nhị phân phức tạp, không thể di chuyển. Điều này cho thấy tiềm năng to lớn của BISTRO trong lĩnh vực bảo mật phần mềm và phân tích nhị phân.

Mặc dù BISTRO đặt ra một số giả định nhất định, nhưng nó đã giải quyết được những thách thức lớn trong việc trích xuất và nhúng thành phần từ/vào các chương trình nhị phân phức tạp, đặc biệt là khi thiếu thông tin tái định vị hoặc bảng tái định vị bị thiếu/giả mạo trong các tệp nhị phân cũ hay phần mềm độc hại.

Các kỹ thuật chính được sử dụng bao gồm:

- Tạo neo (anchors) để xử lý cuộc gọiCác kỹ thuật chính được sử dụng bao gồm:

- Tạo neo (anchors) để xử lý cuộc gọi gián tiếp

- Dịch địa chỉ tại thời gian chạy để xử lý nhảy gián tiếp, sử dụng bảng băm hoàn hảo để tối ưu

- Nhóm và sử dụng neo cho dữ liệu, dịch địa chỉ tham chiếu dữ liệu

Nhờ những kỹ thuật này, BISTRO có thể trích xuất và nhúng các thành phần chức năng một cách chính xác, bảo toàn ngữ nghĩa của cả thành phần và chương trình gốc. Điều này mở ra nhiều khả năng ứng dụng của BISTRO trong lĩnh vực bảo mật phần mềm.

Tóm lại, BISTRO là một hệ thống đột phá, cung cấp một cách tiếp cận mới và hiệu quả để thao tác với các chương trình nhị phân phức tạp mà trước đây rất khó thực hiện do thiếu thông tin biên dịch và tái định vị.