

Phần mềm độc hại

Khoa học dữ liệu

Phát hiện và phân bổ tấn công



Joshua Saxe với Hillary Sanders

Lời nói đầu của Anup Ghosh, Tiến sĩ



Khoa học dữ liệu phần mềm độc hại

Phần mềm độc hại

Khoa học dữ liệu

Tại tack D etectionand
Ghi công

bởi Joshua Saxe
với Hillary Sanders



San Francisco

Khoa học dữ liệu phần mềm đặc hại. Bản quyền © 2018 của Joshua Saxe với Hillary Sanders.

Đã đăng ký Bản quyền. Không được sao chép hoặc truyền bát kỳ phần nào của tác phẩm này dưới bất kỳ hình thức nào hoặc bằng bất kỳ phương tiện nào, điện tử hoặc cơ học, bao gồm sao chụp, ghi âm hoặc bằng bất kỳ hệ thống lưu trữ hoặc truy xuất thông tin nào mà không có sự cho phép trước bằng văn bản của chủ sở hữu bản quyền và nhà xuất bản.

ISBN-10: 1-59327-859-4

ISBN-13: 978-1-59327-859-5

Nhà xuất bản: William Pollock

Biên tập sản xuất: Laurel Chun

Minh họa bìa: Jonny Thomas

Thiết kế nội thất: Octopod Studios

Biên tập phát triển: Annie Choi và William Pollock

Người đánh giá kỹ thuật: Gabor Szappanos

Biên tập: Barton Reed

Sáng tác: Laurel Chun

Người soát lỗi: James Fraleigh

Người lập chỉ mục: BIM Creatives, LLC

Để biết thông tin về phân phối, bản dịch hoặc bán số lượng lớn, vui lòng liên hệ trực tiếp với No Starch Press, Inc.:

Không có Starch Press, Inc.

245 Đường số 8, San Francisco, CA 94103

điện thoại: 1.415.863.9900; info@nostarch.com

www.nostarch.com

Thư viện Quốc hội Số kiểm soát: 2018949204

No Starch Press và logo No Starch Press là các nhãn hiệu đã đăng ký của No Starch Press, Inc. Các sản phẩm và tên công ty khác được đề cập ở đây có thể là nhãn hiệu của các chủ sở hữu tương ứng. Thay vì sử dụng biểu tượng nhãn hiệu với mỗi lần xuất hiện tên đã đăng ký nhãn hiệu, chúng tôi chỉ sử dụng tên theo cách biên tập và vì lợi ích của chủ sở hữu nhãn hiệu, không có ý định vi phạm
Nhãn hiệu.

Thông tin trong cuốn sách này được phân phối trên cơ sở "Nguyên trạng", không có bảo đảm. Mặc dù mọi biện pháp phòng ngừa đã được thực hiện trong quá trình chuẩn bị cho tác phẩm này, cả tác giả và No Starch Press, Inc. thông tin chưa trong đó.

Gửi tới Alen Capalik,
vì đã đưa tôi trở lại với máy tính sau một thời gian dài gián đoạn

Giới thiệu về tác giả

Joshua Saxe là Nhà khoa học dữ liệu trưởng tại nhà cung cấp bảo mật lớn Sophos, nơi ông lãnh đạo một nhóm nghiên cứu khoa học dữ liệu bảo mật. Ông cũng là nhà phát minh chính của công cụ phát hiện phần mềm độc hại dựa trên mạng thần kinh của Sophos, giúp bảo vệ hàng chục triệu khách hàng của Sophos khỏi bị lây nhiễm phần mềm độc hại. Trước khi gia nhập Sophos, Joshua đã có 5 năm lãnh đạo các dự án nghiên cứu dữ liệu bảo mật do DARPA tài trợ cho chính phủ Hoa Kỳ.

Hillary Sanders là kỹ sư phần mềm cao cấp và nhà khoa học dữ liệu tại Sophos, nơi cô đóng vai trò quan trọng trong việc phát minh và sản xuất các công nghệ mạng thần kinh, máy học và phân tích sự tương đồng của phần mềm độc hại. Trước khi gia nhập Sophos, Hillary là nhà khoa học dữ liệu tại Premise Data Corporation. Cô ấy là diễn giả thường xuyên tại các hội nghị bảo mật, đã có các bài nói chuyện về khoa học dữ liệu bảo mật tại Blackhat USA và BSides Las Vegas. Cô học Thống kê tại UC Berkeley.

Giới thiệu về Người đánh giá kỹ thuật

Gabor Szappanos tốt nghiệp Đại học Eotvos Lorand ở Budapest với bằng vật lý. Công việc đầu tiên của anh là phát triển phần mềm và phần cứng chẩn đoán cho các nhà máy điện hạt nhân tại Viện Nghiên cứu Máy tính và Tự động hóa. Gabor bắt đầu công việc chống vi-rút vào năm 1995 và gia nhập VirusBuster vào năm 2001, nơi anh chịu trách nhiệm xử lý vi-rút macro và phần mềm độc hại tập lệnh; năm 2002, ông trở thành trưởng phòng thí nghiệm virus. Từ năm 2008 đến 2016, anh là thành viên ban giám đốc của Tổ chức tiêu chuẩn kiểm tra chống phần mềm độc hại (AMTSO) và vào năm 2012, anh gia nhập Sophos với tư cách là Nhà nghiên cứu phần mềm độc hại chính.

Nội dung tóm tắt

Lời tựa	xvii
Sự nhìn nhận	xix
Giới thiệu	xxi
Chương 1: Phân tích phần mềm độc hại tĩnh cơ bản	1
Chương 2: Ngoài phân tích tĩnh cơ bản: Thảo gỡ x86	11
Chương 3: Giới thiệu tóm tắt về Phân tích động	25
Chương 4: Xác định Chiến dịch Tấn công Sử dụng Mạng Phần mềm độc hại	35
Chương 5: Phân tích mã chia sẻ	59
Chương 6: Tìm hiểu về Trình phát hiện phần mềm độc hại dựa trên máy học.	89
Chương 7: Đánh giá Hệ thống Phát hiện Phần mềm độc hại	119
Chương 8: Xây dựng Máy dò học máy	127
Chương 9: Trực quan hóa Xu hướng Phần mềm độc hại	155
Chương 10: Khái niệm cơ bản về Deep Learning.	175
Chương 11: Xây dựng Trình phát hiện phần mềm độc hại mạng nơ-ron bằng Keras	199
Chương 12: Trở thành nhà khoa học dữ liệu.	215
Phụ lục: Tổng quan về Bộ dữ liệu và Công cụ	221
Mục lục	233

Nội dung chi tiết

Lời nói đầu của Anup Ghosh	xvii
Sự nhìn nhận	xix
Giới thiệu	xxi
Khoa học dữ liệu là gì?	xxii
Tại sao khoa học dữ liệu quan trọng đối với bảo mật	xxii
Áp dụng Khoa học Dữ liệu cho Phần mềm độc hại.	xxiii
Ai nên đọc cuốn sách này?	xxiv
Về cuốn sách này	xxiv
Cách sử dụng Mã mẫu và Dữ liệu	xxv
 1	
Phân tích phần mềm độc hại tinh cơ bản	1
Định dạng thực thi di động của Microsoft Windows	2
Tiêu đề PE	3
Tiêu đề tùy chọn.	3
Phản tiêu đề.	4
Phân tích Định dạng PE Sử dụng pefile	5
Kiểm tra hình ảnh phần mềm độc hại.	7
Kiểm tra chuỗi phần mềm độc hại.	số 8
Sử dụng chuỗi Program	số 8
Phân tích chuỗi của bạn Dump	9
Bản tóm tắt	10
 2	
Ngoài phân tích tinh cơ bản: Tháo gỡ x86	11
Phương pháp tháo gỡ.	12
Khái niệm cơ bản về ngôn ngữ hợp ngữ x86.	12
Thanh ghi CPU.	13
Hướng dẫn số học	15
Hướng dẫn di chuyển dữ liệu.	15
Tháo rời ircbot.exe Sử dụng pefile và capstone	20
Các yếu tố hạn chế phân tích tinh.	21
đóng gói	21
Làm xáo trộn tài nguyên	22
Kỹ thuật chống tháo gỡ	22
Dữ liệu được tái xuống động.	22
Bản tóm tắt	23

3

Giới thiệu tóm tắt về phân tích động	25
Tại sao nên sử dụng Phân tích động?	26
Phân tích động cho khoa học dữ liệu phần mềm độc hại.	26
Công cụ cơ bản để phân tích động.	27
Hành vi phần mềm độc hại điển hình.	27
Đang tải tệp trên malwr.com.	27
Phân tích kết quả trên malwr.com.	28
Hạn chế của Phân tích động cơ bản.	33
Bản tóm tắt	34

4

Xác định các chiến dịch tấn công	
Sử dụng mạng phần mềm độc hại	35
Nút và Cạnh	37
Mạng lưỡng cực	37
Trực quan hóa mạng phần mềm độc hại.	39
Vấn đề biến dạng.	39
Thuật toán hướng lực.	40
Xây dựng mạng với NetworkX	40
Thêm Nút và Cạnh.	41
Thêm thuộc tính	42
Lưu mạng vào đĩa	42
Trực quan hóa mạng với GraphViz.	43
Sử dụng Tham số để Điều chỉnh Mạng.	44
Công cụ dòng lệnh GraphViz.	44
Thêm Thuộc tính Hình ảnh cho Nút và Cạnh.	48
Xây dựng mạng phần mềm độc hại.	51
Xây dựng mạng lưới quan hệ hình ảnh được chia sẻ.	54
Bản tóm tắt	58

5

Phân tích mã được chia sẻ	59
Chuẩn bị mẫu để so sánh bằng cách trích xuất các tính năng.	62
Cách thức hoạt động của các mô hình túi tính năng.	62
N-Gram là gì?	63
Sử dụng Chỉ số Jaccard để định lượng tính tương đồng.	64
Sử dụng ma trận tương tự để đánh giá các phương pháp ước tính mã chia sẻ phần mềm độc hại.	66
Sự giống nhau dựa trên trình tự hướng dẫn.	67
Tính tương tự dựa trên chuỗi.	70
Nhập độ tương tự dựa trên bảng địa chỉ	71
Tương tự dựa trên lệnh gọi API động	72
Xây dựng một biểu đồ tương tự.	73
Mở rộng quy mô so sánh tương tự	77
Tóm tắt Minhash.	77
Minhash chuyên sâu.	78
Xây dựng Hệ thống Tìm kiếm Tương tự Phần mềm độc hại Liên tục.	79
Chạy Hệ thống tìm kiếm sự giống nhau.	85
Bản tóm tắt	87

6

Hiểu về Machine Learning-Based	
Trình phát hiện phần mềm độc hại	89
Các bước để xây dựng bộ phát hiện dựa trên máy học.	90
Thu thập các ví dụ đào tạo.	91
Trích xuất các tính năng.	91
Thiết kế các tính năng tốt.	92
Hệ thống học máy đào tạo.	92
Kiểm tra hệ thống học máy.	93
Hiểu không gian tính năng và ranh giới quyết định.	93
Điều gì khiến người mẫu trở nên tốt hay xấu: Trang bị thừa và trang phục thiếu.	98
Các loại thuật toán học máy chính.	101
Hồi quy logistic.	102
K-Hàng xóm gần nhất.	105
Cây quyết định	109
Khu rừng ngẫu nhiên	115
Bản tóm tắt	117

7

Đánh giá hệ thống phát hiện phần mềm độc hại	119
Bốn kết quả phát hiện có thể xảy ra.	120
Tỷ lệ Tích cực Đúng và Sai.	120
Mối quan hệ giữa tỷ lệ tích cực đúng và sai.	121
Đường cong ROC.	123
Xem xét tỷ lệ cơ bản trong đánh giá của bạn.	124
Tỷ lệ cơ bản ảnh hưởng đến độ chính xác như thế nào.	124
Ước tính độ chính xác trong môi trường triển khai.	125
Bản tóm tắt	126

số 8

Xây dựng máy dò học máy	127
Thuật ngữ và khái niệm.	128
Xây dựng bộ phát hiện dựa trên cây quyết định dò chơi	129
Đào tạo Trình phân loại cây quyết định của bạn.	130
Trực quan hóa cây quyết định.	131
Hoàn thành mã mẫu.	133
Xây dựng Trình phát hiện học máy trong thế giới thực với sklearn	134
Trích xuất tính năng trong thế giới thực.	134
Tại sao bạn không thể sử dụng tất cả các tính năng có thể.	137
Sử dụng thủ thuật băm để nén các tính năng.	138
Xây dựng Máy dò Sức mạnh Công nghiệp	141
Trích xuất các tính năng.	141
Đào tạo máy dò.	142
Chạy Trình phát hiện trên các tập dữ liệu mới.	144
Những gì chúng tôi đã thực hiện cho đến nay.	144
Đánh giá hiệu suất của máy dò của bạn.	146
Sử dụng các đường cong ROC để đánh giá hiệu quả của máy dò.	147
Tính toán các đường cong ROC.	147
Tách dữ liệu thành các tập huấn luyện và kiểm tra.	148

Tính toán đường cong ROC.	149
Xác thực chéo.	150
Bước tiếp theo	153
Bản tóm tắt	154
9	
Trực quan hóa xu hướng phần mềm độc hại	155
Tại sao trực quan hóa dữ liệu phần mềm độc hại lại quan trọng.	156
Hiểu Tập dữ liệu về phần mềm độc hại của chúng tôi	158
Đang tải dữ liệu vào gấu trúc.	158
Làm việc với DataFrame của gấu trúc.	159
Lọc dữ liệu bằng điều kiện.	161
Sử dụng matplotlib để trực quan hóa dữ liệu.	162
Vẽ sơ đồ mối quan hệ giữa kích thước và khả năng phát hiện phần mềm độc hại.	162
Sơ đồ tỷ lệ phát hiện ransomware.	164
Vẽ sơ đồ Ransomware và tỷ lệ phát hiện sâu.	165
Sử dụng seaborn để trực quan hóa dữ liệu.	168
Sơ đồ phân phối các phát hiện chống vi-rút.	169
Tạo một âm mưu vĩ cầm.	172
Bản tóm tắt	174
10	
Khái niệm cơ bản về học sâu	175
Học sâu là gì?.....	176
Mạng lưới thần kinh hoạt động như thế nào.	177
Giải phẫu của một tế bào thần kinh.	177
Một mạng lưới các tế bào thần kinh.	180
Định lý xấp xỉ phổ quát	181
Xây dựng mạng lưới thần kinh của riêng bạn.	182
Thêm một nơ-ron khác vào mạng.	186
Tạo tính năng tự động.	188
Đào tạo mạng lưới thần kinh.	189
Sử dụng Backpropagation để tối ưu hóa mạng lưới thần kinh.	190
Bùng nổ đường dẫn.	192
Độ đặc biến mất.	192
Các loại mạng lưới thần kinh.	193
Mạng thần kinh chuyển tiếp nguồn cấp dữ liệu.	193
Mạng thần kinh tích chập.	193
Mạng thần kinh mã hóa tự động.	194
Mạng đối thủ sáng tạo	195
Mạng thần kinh tái phát	196
ResNet	196
Bản tóm tắt	197
11	
Xây dựng Trình phát hiện phần mềm độc hại mạng nơ-ron với Keras	199
Xác định Kiến trúc của Mô hình.	200
Biên dịch Mô hình.	202

Đào tạo Người mẫu.	203
Trích xuất các tính năng.	203
Tạo Trình tạo dữ liệu	204
Kết hợp dữ liệu xác nhận.	207
Lưu và tải mô hình.	209
Đánh giá mô hình.	209
Tăng cường quá trình đào tạo mô hình với các cuộc gọi lại.	211
Sử dụng Gọi lại tích hợp.	212
Sử dụng Gọi lại tùy chỉnh	213
Bản tóm tắt	214

12

Trở thành nhà khoa học dữ liệu Con đường trở thành	215
--	-----

nà khoa học dữ liệu bảo mật.	216
Một ngày trong cuộc đời của một nhà khoa học dữ liệu bảo mật.....	216
Các đặc điểm của một nhà khoa học dữ liệu bảo mật hiệu quả. : 218	218
Tư duy cởi mở.	218
Trí tuệ vô biên.	218
Nỗi ám ảnh với kết quả.	219
Chủ nghĩa hoài nghi về kết quả.	219
Đi đâu từ đây	219

xuột thừa

Tổng quan về Bộ dữ liệu và Công cụ	221
------------------------------------	-----

Tổng quan về Bộ dữ liệu.	222
Chương 1: Phân tích phần mềm độc hại tĩnh cơ bản	222
Chương 2: Ngoài phân tích tĩnh cơ bản: Tháo gỡ x86	222
Chương 3: Giới thiệu tóm tắt về Phân tích động	222
Chương 4: Xác định Chiến dịch Tấn công Sử dụng Mạng Phần mềm độc hại	222
Chương 5: Phân tích mã chia sẻ	223
Chương 6: Tìm hiểu về Trình phát hiện phần mềm độc hại dựa trên máy học và Chương 7: Đánh giá hệ thống phát hiện phần mềm độc hại	223
Chương 8: Xây dựng Máy dò học máy	224
Chương 9: Trực quan hóa Xu hướng Phần mềm độc hại	224
Chương 10: Khái niệm cơ bản về Deep Learning.	224
Chương 11: Xây dựng Trình phát hiện phần mềm độc hại mạng nd-ron bằng Keras	224
Chương 12: Trở thành nhà khoa học dữ liệu.	224
Hướng dẫn triển khai công cụ	225
Trực quan hóa mạng tên máy chủ được chia sẻ.	225
Trực quan hóa mạng hình ảnh được chia sẻ.	226
Trực quan hóa tính tương tự của phần mềm độc hại	227
Hệ thống Tim kiếm Tương tự Phần mềm độc hại	229
Hệ thống phát hiện phần mềm độc hại Machine Learning.	230

Mục lục	233
----------------	-----

Lời tựa

Chúc mừng bạn đã chọn Khoa học dữ liệu phần mềm độc hại. Bạn đang trên đường trang bị cho mình những kỹ năng cần thiết để trở thành một chuyên gia an ninh mạng. Trong cuốn sách này, bạn sẽ tìm thấy phần giới thiệu tuyệt vời về khoa học dữ liệu khi áp dụng cho phân tích phần mềm độc hại, cũng như các kỹ năng và công cụ cần thiết mà bạn cần để thành thạo về nó.

Có nhiều việc làm trong lĩnh vực an ninh mạng hơn số ứng viên đủ điều kiện, vì vậy tin tốt là an ninh mạng là một lĩnh vực tuyệt vời để tham gia. Tin xấu là các kỹ năng cần thiết để duy trì hiện tại đang thay đổi nhanh chóng. Như thường lệ, sự cần thiết là mẹ của phát minh. Với nhu cầu về các chuyên gia an ninh mạng lành nghề cao hơn nhiều so với nguồn cung, các thuật toán khoa học dữ liệu đang lắp đầy khoảng trống bằng cách cung cấp những hiểu biết và dự đoán mới về các mối đe dọa đối với mạng. Mô hình đồng hồ truyền thống theo dõi dữ liệu mạng đang nhanh chóng trở nên lỗi thời khi khoa học dữ liệu

đang ngày càng được sử dụng để tìm các mẫu mồi đe dọa trong hàng terabyte dữ liệu. Và cảm ơn Chúa vì điều đó, bởi vì việc theo dõi một màn hình cảnh báo cũng thú vị như theo dõi hệ thống camera giám sát của một bãi đậu xe.

Vậy khoa học dữ liệu chính xác là gì và nó áp dụng như thế nào cho bảo mật? Như bạn sẽ thấy trong phần Giới thiệu, khoa học dữ liệu áp dụng cho bảo mật là nghệ thuật và khoa học sử dụng máy học, khai thác dữ liệu và trực quan hóa để phát hiện các mối đe dọa đối với mạng. Mặc dù bạn sẽ tìm thấy rất nhiều lời cung điệu xung quanh việc học máy và trí tuệ nhân tạo do tiếp thị thúc đẩy, nhưng trên thực tế, có những trường hợp sử dụng rất tốt cho những công nghệ này đang được sản xuất ngày nay.

Chẳng hạn, khi phát hiện phần mềm độc hại, cả khối lượng việc sản xuất phần mềm độc hại và chi phí cho kẻ thù trong việc thay đổi chữ ký phần mềm độc hại đã khiến các phương pháp tiếp cận phần mềm độc hại chỉ dựa trên chữ ký đã trở nên lỗi thời. Thay vào đó, các công ty chống vi-rút hiện đang đào tạo mạng thần kinh hoặc các loại thuật toán học máy khác trên bộ dữ liệu phần mềm độc hại rất lớn để tìm hiểu đặc điểm của chúng, để có thể phát hiện các biến thể mới của phần mềm độc hại mà không cần phải cập nhật mã hình hàng ngày. Sự kết hợp giữa tính năng phát hiện dựa trên chữ ký và máy học cung cấp độ tuổi phù hợp cho cả phần mềm độc hại đã biết và chưa biết. Đây là một chủ đề mà cả Josh và Hillary đều là chuyên gia và từ đó họ phát biểu từ kinh nghiệm sâu sắc.

Nhưng phát hiện phần mềm độc hại chỉ là một trường hợp sử dụng cho khoa học dữ liệu. Trên thực tế, khi tìm kiếm các mối đe dọa trên mạng, các đối thủ tinh vi ngày nay thường sẽ không bỏ các chương trình thực thi. Thay vào đó, họ sẽ khai thác phần mềm hiện có để truy cập ban đầu và sau đó tận dụng các công cụ hệ thống để chuyển từ máy này sang máy khác bằng cách sử dụng các đặc quyền của người dùng có được thông qua khai thác. Từ quan điểm của đối thủ, cách tiếp cận này không để lại các hiện vật như phần mềm độc hại mà phần mềm chống vi-rút sẽ phát hiện. Tuy nhiên, một hệ thống ghi nhật ký diểm cuối tốt hoặc hệ thống phát hiện và phản hồi diểm cuối (EDR) sẽ nắm bắt các hoạt động ở cấp hệ thống và gửi phép đo từ xa này tới đám mây, từ đó các nhà phân tích có thể có gắng ghép các dấu vết kỹ thuật số của kẻ xâm nhập lại với nhau. Quá trình rà soát các luồng dữ liệu khổng lồ này và liên tục tìm kiếm các kiểu xâm nhập là một vấn đề rất phù hợp với khoa học dữ liệu, cụ thể là khai thác dữ liệu bằng các thuật toán thống kê và trực quan hóa dữ liệu. Bạn có thể mong đợi ngày càng nhiều Trung tâm điều hành bảo mật (SOC) áp dụng công nghệ khai thác dữ liệu và trí tuệ nhân tạo. Đó thực sự là cách duy nhất để chọn lọc thông qua các tập dữ liệu khổng lồ về các sự kiện hệ thống để xác định các cuộc tấn công thực sự.

An ninh mạng đang trải qua những thay đổi lớn trong công nghệ và hoạt động của nó và khoa học dữ liệu đang thúc đẩy sự thay đổi. Chúng tôi may mắn có được những chuyên gia như Josh Saxe và Hillary Sanders không chỉ chia sẻ kiến thức chuyên môn của họ với chúng tôi mà còn thực hiện điều đó theo cách hấp dẫn và dễ tiếp cận. Đây là cơ hội của bạn để tìm hiểu những gì họ biết và áp dụng nó vào công việc của riêng bạn để bạn có thể đón đầu những thay đổi của công nghệ và những kẻ thù mà bạn có nhiệm vụ đánh bại.

Anup K. Ghosh, tiến sĩ
Người sáng lập, Invincea, Inc
Washington DC

Sự nhìn nhận

Xin cảm ơn Annie Choi, Laurel Chun và Bill Pollock tại No Starch Press và người biên tập của tôi, Bart Reed. Công bằng mà nói, họ nên được coi là đồng tác giả của cuốn sách này. Xin gửi lời cảm ơn trước tới các nhân viên chịu trách nhiệm in ấn, vận chuyển và bán các bản sao của cuốn sách này cũng như các kỹ sư chịu trách nhiệm lưu trữ, truyền tải và kết xuất kỹ thuật số của nó. Cảm ơn Hillary Sanders vì đã đưa những tài năng đáng chú ý của cô ấy vào dự án đúng lúc họ cần. Xin cảm ơn Gabor Szappanos vì bài đánh giá kỹ thuật xuất sắc và chính xác của anh ấy.

Cảm ơn con gái Maya hai tuổi của tôi, người mà tôi rất vui được chia sẻ rằng dự án này đã bị chậm lại đáng kể. Xin cảm ơn Alen Capalik, Danny Hillis, Chris Greamo, Anup Ghosh và Joe Levy vì sự cố vấn của họ trong 10 năm qua. Xin gửi lời cảm ơn sâu sắc tới Cơ quan Dự án Nghiên cứu Tiên tiến Quốc phòng (DARPA) và Timothy Fraser vì đã hỗ trợ nghiên cứu mà phần lớn cuốn sách này dựa vào. Cảm ơn Mandiant và Mila Parkour đã thu thập và quản lý các mẫu phần mềm độc hại APT1 được sử dụng cho mục đích trình diễn trong cuốn sách này. Xin gửi lời cảm ơn sâu sắc tới các tác giả của Python, NetworkX, matplotlib, numpy, sklearn, Keras, seaborn, pefile, icoutils, malwr.com, CuckooBox, capstone, pandas và sqlite vì những đóng góp của bạn cho phần mềm khoa học dữ liệu và bảo mật mã nguồn mở và miễn phí.

Vô cùng biết ơn cha mẹ tôi, Maryl Gearhart và Geoff Saxe, vì đã giới thiệu tôi với máy tính, vì đã bao dung cho giai đoạn hacker tuổi teen của tôi (và tắt cả những hành vi phạm pháp kéo theo), cũng như vì tình yêu và sự hỗ trợ vô bờ bến của họ. Cảm ơn Gary Glickman vì tình yêu và sự hỗ trợ không thể thiếu của anh ấy. Cuối cùng, xin cảm ơn Ksenya Gurshtein, người bạn đời của tôi, đã hỗ trợ tôi hoàn toàn và không do dự trong nỗ lực này.

Joshua Saxe

Cảm ơn Josh, vì đã bao gồm tôi trong việc này! Cảm ơn Ani Adhikari vì đã trở thành một giáo viên tuyệt vời. Cảm ơn Jacob Michelini, vì ông ấy thực sự muốn có tên mình trong một cuốn sách.

Hillary Sanders

Giới thiệu



Nếu bạn đang làm việc trong lĩnh vực bảo mật, rất có thể bạn đang sử dụng khoa học dữ liệu nhiều hơn bao giờ hết, ngay cả khi bạn có thể không nhận ra điều đó. Ví dụ: sản phẩm chống vi-rút của bạn sử dụng thuật toán khoa học dữ liệu để phát hiện phần mềm độc hại. Nhà cung cấp tường lửa của bạn có thể có các thuật toán khoa học dữ liệu phát hiện hoạt động mạng đáng ngờ. Phần mềm quản lý sự kiện và thông tin bảo mật (SIEM) của bạn có thể sử dụng khoa học dữ liệu để xác định các xu hướng đáng ngờ trong dữ liệu của bạn. Dù rõ ràng hay không, toàn bộ ngành bảo mật đang hướng tới việc kết hợp nhiều khoa học dữ liệu hơn vào các sản phẩm bảo mật.

Các chuyên gia bảo mật CNTT nâng cao đang kết hợp các thuật toán máy học tùy chỉnh của riêng họ vào quy trình làm việc của họ. Ví dụ: trong các bài báo và bài thuyết trình tại hội nghị gần đây, các nhà phân tích bảo mật tại Target, Mastercard và Wells Fargo đều mô tả việc phát triển khoa học dữ liệu tùy chỉnh

các công nghệ mà họ sử dụng như một phần trong quy trình bảo mật của họ.¹ Nếu bạn chưa tham gia vào nhóm khoa học dữ liệu, thì không có thời điểm nào tốt hơn để nâng cấp các kỹ năng của bạn để đưa khoa học dữ liệu vào thực tiễn bảo mật của bạn.

Khoa học dữ liệu là gì?

Khoa học dữ liệu là một tập hợp các công cụ thuật toán ngày càng phát triển cho phép chúng ta hiểu và đưa ra dự đoán về dữ liệu bằng cách sử dụng số liệu thống kê, toán học và trực quan hóa dữ liệu thống kê nghệ thuật. Có nhiều định nghĩa cụ thể hơn, nhưng nói chung, khoa học dữ liệu có ba thành phần phụ: học máy, khai thác dữ liệu và trực quan hóa dữ liệu.

Trong bối cảnh bảo mật, các thuật toán học máy học từ dữ liệu đào tạo để phát hiện các mối đe dọa mới. Những phương pháp này đã được chứng minh là có thể phát hiện phần mềm độc hại nằm dưới tầm ngắm của các kỹ thuật phát hiện truyền thống như chữ ký. Các thuật toán khai thác dữ liệu tìm kiếm dữ liệu bảo mật để tìm các mẫu thú vị (chẳng hạn như mối quan hệ giữa các tác nhân đe dọa) có thể giúp chúng tôi phân biệt các chiến dịch tấn công nhằm mục tiêu vào các tổ chức của chúng tôi. Cuối cùng, trực quan hóa dữ liệu biến dữ liệu dạng bảng vô trùng thành định dạng đồ họa để giúp mọi người dễ dàng phát hiện ra các xu hướng thú vị và đáng ngờ. Tôi trình bày chi tiết cả ba lĩnh vực trong cuốn sách này và chỉ cho bạn cách áp dụng chúng.

Tại sao khoa học dữ liệu quan trọng đối với bảo mật

Khoa học dữ liệu cực kỳ quan trọng đối với tương lai của an ninh mạng vì ba lý do: thứ nhất, bảo mật là tất cả về dữ liệu. Khi tìm cách phát hiện các mối đe dọa trên mạng, chúng tôi đang phân tích dữ liệu ở dạng tệp, nhật ký, gói mạng và các tệp khác. Theo truyền thống, các chuyên gia bảo mật không sử dụng các kỹ thuật khoa học dữ liệu để phát hiện dựa trên các nguồn dữ liệu này. Thay vào đó, họ đã sử dụng hàm băm tệp, các quy tắc được viết tay chỉnh như chữ ký và các heuristic được xác định thủ công. Mặc dù các kỹ thuật này có giá trị riêng, nhưng chúng yêu cầu các kỹ thuật thủ công cho từng loại tấn công, đòi hỏi quá nhiều công việc thủ công để theo kịp bối cảnh mối đe dọa mạng đang thay đổi. Trong những năm gần đây, các kỹ thuật khoa học dữ liệu đã trở nên quan trọng trong việc cung cấp khả năng phát hiện các mối đe dọa của chúng ta.

Thứ hai, khoa học dữ liệu rất quan trọng đối với an ninh mạng vì số lượng các cuộc tấn công mạng trên internet đã tăng lên đáng kể. Lấy sự phát triển của thế giới ngầm phần mềm độc hại làm ví dụ. Trong năm 2008, có khoảng 1 triệu tập tin thực thi phần mềm độc hại duy nhất được cộng đồng bảo mật biết đến. Đến năm 2012 là 100 triệu. Khi cuốn sách này được xuất bản vào năm 2018, đã có hơn 700 triệu tệp thực thi độc hại được cộng đồng bảo mật biết đến (<https://www.av-test.org/en/statistics/malware/>) và con số này có khả năng phát triển.

1. Mục tiêu (<https://www.rsaconference.com/events/us17/agenda/sessions/6662-applied-machine-learning-defeat-modern-malicious>), Mastercard (<https://blogs.wsj.com/cio/15/11/2017/tri-tu-pham-tao-transforms-hacker-arsenal/>), và Wells Fargo (<https://blogs.wsj.com/cio/2017/11/16/buoi-sang-tai-xuong-dau-tien-ai-cung-cap-cac-cuoc-tan-cong-mang-duoc-phat-hien/>).

Do số lượng lớn phần mềm độc hại, các kỹ thuật phát hiện thủ công như chữ ký không còn là phương pháp hợp lý để phát hiện tất cả các cuộc tấn công mạng. Do các kỹ thuật khoa học dữ liệu tự động hóa phần lớn công việc phát hiện các cuộc tấn công mạng và giảm đáng kể mức sử dụng bộ nhớ cần thiết để phát hiện các cuộc tấn công như vậy nên chúng hứa hẹn rất nhiều trong việc bảo vệ mạng và người dùng khi các mối đe dọa mạng ngày càng gia tăng.

Cuối cùng, khoa học dữ liệu quan trọng đối với bảo mật vì khoa học dữ liệu là xu hướng kỹ thuật của thập kỷ, cả bên trong và bên ngoài ngành bảo mật, và nó có thể sẽ duy trì như vậy trong thập kỷ tới. Thật vậy, bạn có thể đã thấy các ứng dụng của khoa học dữ liệu ở khắp mọi nơi-trong trợ lý giọng nói cá nhân (Amazon Echo, Siri và Google Home), ô tô tự lái, hệ thống đề xuất quảng cáo, công cụ tìm kiếm trên web, hệ thống phân tích hình ảnh y tế và theo dõi hoạt động thể chất. Ứng dụng.

Chúng ta có thể mong đợi các hệ thống dựa trên khoa học dữ liệu sẽ có tác động lớn đến các dịch vụ pháp lý, giáo dục và các lĩnh vực khác. Vì khoa học dữ liệu đã trở thành yếu tố quyết định chính trong toàn cảnh kỹ thuật nên các trường đại học, công ty lớn (Google, Facebook, Microsoft và IBM) và các chính phủ đang đầu tư hàng tỷ đô la để cải tiến các công cụ khoa học dữ liệu. Nhờ những khoản đầu tư này, các công cụ khoa học dữ liệu sẽ trở nên hiệu quả hơn trong việc giải quyết các vấn đề phát hiện tấn công khó.

Áp dụng khoa học dữ liệu cho phần mềm độc hại

Cuốn sách này tập trung vào khoa học dữ liệu vì nó áp dụng cho phần mềm độc hại, mà chúng tôi định nghĩa là các chương trình thực thi được viết với mục đích xấu, vì phần mềm độc hại tiếp tục là phương tiện chính mà các tác nhân đe dọa có được chở đứng trên mạng và sau đó đạt được mục tiêu của chúng. Ví dụ: trong thảm họa phần mềm độc hại đã xuất hiện trong những năm gần đây, những kẻ tấn công thường gửi cho người dùng các tệp đính kèm email độc hại để tải các tệp thực thi ransomware (phần mềm độc hại) về máy của người dùng, sau đó mã hóa dữ liệu của người dùng và yêu cầu họ trả tiền chuộc để giải mã dữ liệu. Mặc dù những kẻ tấn công lành nghề làm việc cho chính phủ đôi khi tránh sử dụng phần mềm độc hại hoàn toàn để lọt vào tầm ngắm của các hệ thống phát hiện, nhưng phần mềm độc hại vẫn tiếp tục là công nghệ hỗ trợ chính trong các cuộc tấn công mạng ngày nay.

Bằng cách tập trung vào một ứng dụng cụ thể của khoa học dữ liệu bảo mật thay vì cố gắng đề cập rộng rãi đến khoa học dữ liệu bảo mật, cuốn sách này nhằm mục đích trình bày kỹ lưỡng hơn cách các kỹ thuật khoa học dữ liệu có thể được áp dụng cho một vấn đề bảo mật lớn.

Bằng cách hiểu về khoa học dữ liệu phần mềm độc hại, bạn sẽ được trang bị tốt hơn để áp dụng khoa học dữ liệu vào các lĩnh vực bảo mật khác, như phát hiện các cuộc tấn công mạng, email lừa đảo hoặc hành vi đáng ngờ của người dùng.

Thật vậy, hầu hết tất cả các kỹ thuật bạn sẽ học trong cuốn sách này đều áp dụng cho việc xây dựng các hệ thống tình báo và phát hiện khoa học dữ liệu nói chung, không chỉ dành cho phần mềm độc hại.

Ai nên đọc cuốn sách này?

Cuốn sách này hướng tới các chuyên gia bảo mật, những người quan tâm đến việc tìm hiểu thêm về cách áp dụng khoa học dữ liệu vào các vấn đề bảo mật máy tính. Nếu bảo mật máy tính và khoa học dữ liệu là mới đối với bạn, bạn có thể thấy mình phải tra cứu các thuật ngữ để cung cấp cho mình một chút ngữ cảnh, nhưng bạn vẫn có thể đọc thành công cuốn sách này. Nếu bạn chỉ quan tâm đến khoa học dữ liệu chứ không quan tâm đến bảo mật, thì cuốn sách này có lẽ không dành cho bạn.

Về cuốn sách này

Phần đầu tiên của cuốn sách bao gồm ba chương đề cập đến các khái niệm kỹ thuật đảo ngược cơ bản cần thiết để hiểu các kỹ thuật khoa học dữ liệu phần mềm độc hại được thảo luận sau trong cuốn sách. Nếu bạn chưa quen với phần mềm độc hại, trước tiên hãy đọc ba chương đầu tiên. Nếu bạn đã thành thạo kỹ thuật đảo ngược phần mềm độc hại, bạn có thể bỏ qua các chương này.

- Chương 1: Phân tích phần mềm độc hại tĩnh cơ bản bao gồm công nghệ phân tích tĩnh các cách để chọn ra các tệp phần mềm độc hại và khám phá cách chúng đạt được mục đích độc hại trên máy tính của chúng tôi.
- Chương 2: Ngoài phân tích tĩnh cơ bản: x86 Disassembly cung cấp cho bạn một cái nhìn tổng quan ngắn gọn về hợp ngữ x86 và cách tháo rời và thiết kế ngược phần mềm độc hại.
- Chương 3: Giới thiệu tóm tắt về Phân tích động kết thúc phần
phần kỹ thuật đảo ngược của cuốn sách bằng cách thảo luận về phân tích động, liên quan đến việc chạy phần mềm độc hại trong môi trường được kiểm soát để tìm hiểu về hành vi của nó.

Hai chương tiếp theo của cuốn sách, Chương 4 và 5, tập trung vào phân tích mối quan hệ của phần mềm độc hại, bao gồm việc xem xét những điểm tương đồng và khác biệt giữa các bộ sưu tập phần mềm độc hại để xác định các chiến dịch phần mềm độc hại chống lại tổ chức của bạn, chẳng hạn như chiến dịch phần mềm tống tiền do một nhóm mã độc kiểm soát. tội phạm mạng hoặc một cuộc tấn công có chủ đích, phối hợp vào tổ chức của bạn. Các chương độc lập này dành cho những độc giả không chỉ quan tâm đến việc phát hiện phần mềm độc hại mà còn quan tâm đến việc trích xuất thông tin tình báo về mối đe dọa có giá trị để tìm hiểu xem ai đang tấn công mạng của họ. Nếu bạn ít quan tâm đến thông tin tình báo về mối đe dọa và quan tâm nhiều hơn đến việc phát hiện phần mềm độc hại dựa trên khoa học dữ liệu, thì bạn có thể yên tâm bỏ qua các chương này.

- Chương 4: Nhận diện Chiến dịch Tấn công Sử dụng Mạng Malware
chỉ cho bạn cách phân tích và trực quan hóa phần mềm độc hại dựa trên các thuộc tính được chia sẻ, chẳng hạn như tên máy chủ mà chương trình phần mềm độc hại gọi ra.
- Chương 5: Shared Code Analysis giải thích cách nhận dạng và hình ảnh
đánh giá mối quan hệ mã được chia sẻ giữa các mẫu phần mềm độc hại, điều này có thể giúp bạn xác định xem các nhóm mẫu phần mềm độc hại đến từ một hay nhiều nhóm tội phạm nhỏ.

Bốn chương tiếp theo bao gồm mọi thứ bạn cần biết để hiểu rõ, áp dụng và triển khai các hệ thống phát hiện phần mềm độc hại dựa trên máy học. Các chương này cũng cung cấp nền tảng để áp dụng học máy vào các bối cảnh bảo mật khác.

- Chương 6: Tìm hiểu phần mềm độc hại dựa trên máy học
Máy dò là phần giới thiệu dễ tiếp cận, trực quan và phi toán học đối với các khái niệm máy học cơ bản. Nếu bạn đã từng học máy, chương này sẽ giúp bạn ôn lại một cách thuận tiện.
- Chương 7: Đánh giá Hệ thống Phát hiện Phần mềm độc hại chỉ cho bạn cách đánh giá độ chính xác của hệ thống máy học bằng các phương pháp thống kê cơ bản để bạn có thể chọn phương pháp tốt nhất có thể.
- Chương 8: Xây dựng Machine Learning Detectors giới thiệu mở nguồn công cụ máy học mà bạn có thể sử dụng để xây dựng hệ thống máy học của riêng mình và giải thích cách sử dụng chúng.
- Chương 9: Trực quan hóa xu hướng phần mềm độc hại bao gồm cách trực quan hóa dữ liệu về mối đe dọa của phần mềm độc hại để tiết lộ các chiến dịch và xu hướng tấn công bằng Python và cách tích hợp trực quan hóa dữ liệu vào quy trình làm việc hàng ngày của bạn khi phân tích dữ liệu bảo mật.

Ba chương cuối cùng giới thiệu về học sâu, một lĩnh vực học máy nâng cao liên quan đến toán học nhiều hơn một chút. Học sâu là một lĩnh vực phát triển nóng trong khoa học dữ liệu bảo mật và các chương này cung cấp đủ kiến thức để bạn bắt đầu.

- Chương 10: Khái niệm cơ bản về Deep Learning bao gồm các khái niệm cơ bản làm nền tảng cho deep learning.
- Chương 11: Xây dựng bộ phát hiện phần mềm độc hại mạng nơ-ron bằng Keras giải thích cách triển khai các hệ thống phát hiện phần mềm độc hại dựa trên học sâu trong Python bằng các công cụ nguồn mở.
- Chương 12: Trở thành Nhà khoa học dữ liệu kết thúc cuốn sách bằng cách chia sẻ các con đường khác nhau để trở thành nhà khoa học dữ liệu và những phẩm chất có thể giúp bạn thành công trong lĩnh vực này.
- Phụ lục: Tổng quan về Bộ dữ liệu và Công cụ mô tả việc triển khai dữ liệu và công cụ ví dụ đi kèm với cuốn sách.

Cách sử dụng mã mẫu và dữ liệu

Không có cuốn sách lập trình hay nào hoàn chỉnh nếu không có mã mẫu để bạn tự học và mở rộng. Mã mẫu và dữ liệu đi kèm với mỗi chương của cuốn sách này và được mô tả đầy đủ trong phần phụ lục. Tất cả các mã nhằm mục tiêu Python 2.7 trong môi trường Linux. Để truy cập mã và dữ liệu, bạn có thể tải xuống máy ảo VirtualBox Linux, có mã, dữ liệu và các công cụ nguồn mở hỗ trợ, tất cả đều được thiết lập và sẵn sàng hoạt động,

và chạy nó trong môi trường VirtualBox của riêng bạn. Bạn có thể tải xuống dữ liệu đi kèm của cuốn sách tại <http://www.malwaredatascience.com/> và bạn có thể tải xuống VirtualBox miễn phí tại https://www.virtualbox.org/wiki/Tải_xuống. Mã này đã được thử nghiệm trên Linux, nhưng nếu bạn thích làm việc bên ngoài Linux VirtualBox, thì mã tương tự cũng sẽ hoạt động gần như tốt trên MacOS và ở mức độ thấp hơn trên các máy Windows.

Nếu bạn muốn cài đặt mã và dữ liệu trong môi trường Linux của riêng mình, bạn có thể tải xuống tại đây: <http://www.malwaredatascience.com/>.

Bạn sẽ tìm thấy một thư mục cho từng chương trong kho lưu trữ có thể tải xuống và trong thư mục của mỗi chương có các thư mục mã/ và dữ liệu/ chứa mã và dữ liệu tương ứng. Các tệp mã tương ứng với danh sách chương hoặc phần, tùy theo điều kiện nào hợp lý hơn đối với ứng dụng hiện tại. Một số tệp mã hoàn toàn giống với danh sách, trong khi những tệp khác đã được thay đổi một chút để giúp bạn thao tác dễ dàng hơn với các tham số và các tùy chọn khác. Các thư mục mã đi kèm với các tệp pip tests.txt , cung cấp các thư viện nguồn mở mà mã trong chương đó phụ thuộc vào để chạy. Để cài đặt các thư viện này trên máy của bạn, chỉ cần nhập pip -r tests.txt trong thư mục code/ của mỗi chương .

Bây giờ bạn đã có quyền truy cập vào mã và dữ liệu của cuốn sách này, hãy bắt đầu.

1

B a sic Static Malware Phân tích



Trong chương này, chúng ta xem xét những kiến thức cơ bản về phân tích phần mềm độc hại tĩnh.

Phân tích tĩnh được thực hiện bằng cách phân tích mã được phân tách của tệp chương trình, hình ảnh đồ họa, chuỗi có thể in và các tài nguyên trên đĩa khác. Nó đề cập đến kỹ thuật đảo ngược mà không thực sự chạy chương trình. Mặc dù các kỹ thuật phân tích tĩnh có những thiếu sót nhưng chúng có thể giúp chúng tôi hiểu được nhiều loại phần mềm độc hại. Thông qua kỹ thuật đảo ngược cẩn thận, bạn sẽ có thể hiểu rõ hơn về lợi ích mà các tệp nhị phân phần mềm độc hại mang lại cho kẻ tấn công sau khi chúng chiếm hữu mục tiêu, cũng như cách kẻ tấn công có thể che giấu và tiếp tục tấn công vào máy bị nhiễm. Như bạn sẽ thấy, chương này kết hợp các mô tả và ví dụ. Mỗi phần giới thiệu một kỹ thuật phân tích tĩnh và sau đó minh họa ứng dụng của nó trong phân tích thực tế.

Tôi bắt đầu chương này bằng cách mô tả định dạng tệp Portable Executable (PE) được sử dụng bởi hầu hết các chương trình Windows, sau đó kiểm tra cách sử dụng thư viện Python phô biến pefile để phân tích tệp nhị phân phần mềm độc hại trong thế giới thực. Sau đó, tôi mô tả các kỹ thuật như phân tích nhập khẩu, phân tích hình ảnh đồ họa và phân tích chuỗi. Trong mọi trường hợp, tôi sẽ chỉ cho bạn cách sử dụng các công cụ nguồn mở để áp dụng kỹ thuật phân tích cho phần mềm độc hại trong thế giới thực. Cuối cùng, ở cuối chương này, tôi giới thiệu những cách phần mềm độc hại có thể gây khó khăn cho các nhà phân tích phần mềm độc hại và thảo luận một số cách để giảm thiểu những vấn đề này.

Bạn sẽ tìm thấy mẫu phần mềm độc hại được sử dụng trong các ví dụ trong chương này ở dữ liệu của cuốn sách này trong thư mục /ch1. Để minh họa các kỹ thuật được thảo luận trong chương này, chúng tôi sử dụng ircbot.exe, bot Tiếng chuyện chuyển tiếp qua Internet (IRC) được tạo để sử dụng thử nghiệm, làm ví dụ về các loại phần mềm độc hại thường thấy trong thực tế. Do đó, chương trình được thiết kế để lưu trú trên máy tính mục tiêu trong khi được kết nối với máy chủ IRC, sau ircbot

.exe chiếm được mục tiêu, kẻ tấn công có thể điều khiển máy tính mục tiêu thông qua IRC, cho phép chúng thực hiện các hành động như bật webcam để chụp và lén lút trích xuất nguồn cấp dữ liệu video về vị trí thực của mục tiêu, chụp ảnh màn hình máy tính để bàn, trích xuất tệp từ máy mục tiêu, v.v. Trong suốt chương này, tôi trình bày cách các kỹ thuật phân tích tĩnh có thể tiết lộ các khả năng của phần mềm độc hại này.

Định dạng thực thi động của Microsoft Windows

Để thực hiện phân tích phần mềm độc hại tĩnh, bạn cần hiểu định dạng Windows PE, định dạng này mô tả cấu trúc của các tệp chương trình Windows hiện đại, chẳng hạn như tệp .exe, .dll và .sys, đồng thời xác định cách chúng lưu trữ dữ liệu. Các tệp PE chứa các hướng dẫn x86, dữ liệu như hình ảnh và văn bản cũng như siêu dữ liệu mà chương trình cần để chạy.

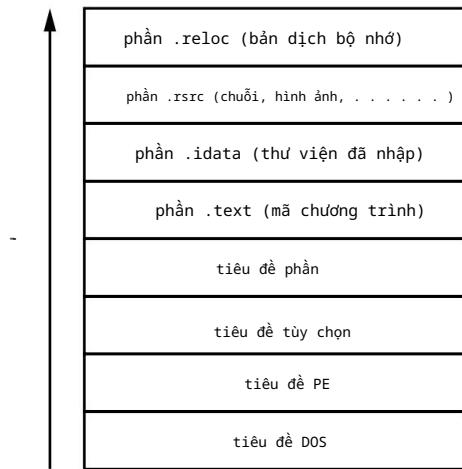
Định dạng PE ban đầu được thiết kế để thực hiện những việc sau:

Cho Windows biết cách tải chương trình vào bộ nhớ Định dạng PE mô tả đoạn nào của tệp sẽ được tải vào bộ nhớ và ở đâu. Nó cũng cho bạn biết vị trí trong mã chương trình Windows sẽ bắt đầu thực thi chương trình và thư viện mã liên kết đóng nào sẽ được tải vào bộ nhớ.

Cung cấp phương tiện (hoặc tài nguyên) mà một chương trình đang chạy có thể sử dụng trong quá trình thực thi. Các tài nguyên này có thể bao gồm các chuỗi ký tự giống như chuỗi ký tự trong hộp thoại GUI hoặc đầu ra của bảng điều khiển, cũng như hình ảnh hoặc video.

Cung cấp dữ liệu bảo mật chẳng hạn như chữ ký mã kỹ thuật số Windows sử dụng dữ liệu bảo mật đó để đảm bảo rằng mã đến từ một nguồn đáng tin cậy.

Định dạng PE hoàn thành tất cả những điều này bằng cách tận dụng chuỗi cấu trúc được hiển thị trong Hình 1-1.



Hình 1-1: Định dạng tệp PE

Như hình minh họa, định dạng PE bao gồm một loạt tiêu đề cho hệ điều hành biết cách tải chương trình vào bộ nhớ. Nó cũng bao gồm một loạt các phần chứa dữ liệu chương trình thực tế. Windows tải các phần vào bộ nhớ sao cho độ lệch bộ nhớ của chúng tương ứng với vị trí chúng xuất hiện trên đĩa. Hãy khám phá cấu trúc tệp này chi tiết hơn, bắt đầu với tiêu đề PE. Chúng ta sẽ bỏ qua phần thảo luận về tiêu đề DOS, đây là di tích của hệ điều hành Microsoft DOS từ những năm 1980 và chỉ xuất hiện vì lý do tương thích.

Tiêu đề PE

Hiển thị ở dưới cùng của Hình 1-1, phía trên tiêu đề DOS u, là tiêu đề PE v, xác định các thuộc tính chung của chương trình như mã nhị phân, hình ảnh, dữ liệu nén và các thuộc tính chương trình khác. Nó cũng cho chúng ta biết chương trình được thiết kế cho hệ thống 32 hay 64 bit. Tiêu đề PE cung cấp thông tin ngữ cảnh cơ bản nhưng hữu ích cho nhà phân tích phần mềm độc hại. Ví dụ: tiêu đề bao gồm trường dấu thời gian có thể tiết lộ thời gian mà tác giả phần mềm độc hại đã biên dịch tệp. Điều này xảy ra khi tác giả phần mềm độc hại quên thê trường này bằng một giá trị không có thật, điều mà họ thường làm.

Tiêu đề tùy chọn

Tiêu đề tùy chọn w thực sự phổ biến trong các chương trình thực thi PE ngày nay, trái ngược với tên gọi của nó. Nó xác định vị trí của điểm vào của chương trình trong tệp PE, tham chiếu đến lệnh đầu tiên mà chương trình chạy sau khi được tải. Nó cũng xác định kích thước của dữ liệu mà Windows tải vào bộ nhớ khi nó tải tệp PE, hệ thống con Windows, mục tiêu chương trình (chẳng hạn như Windows GUI hoặc Windows

dòng lệnh) và các chi tiết cấp cao khác về chương trình. Thông tin trong tiêu đề này có thể chứng minh là vô giá đối với các kỹ sư đảo ngược, bởi vì điểm đầu vào của chương trình cho họ biết nơi bắt đầu kỹ thuật đảo ngược.

Tiêu đề mục

Tiêu đề phần x mô tả các phần dữ liệu có trong tệp PE. Một phần trong tệp PE là một đoạn dữ liệu sẽ được ánh xạ vào bộ nhớ khi hệ điều hành tải chương trình hoặc sẽ chứa các hướng dẫn về cách chương trình sẽ được tải vào bộ nhớ. Nói cách khác, một phần là một chuỗi các byte trên đĩa sẽ trở thành một chuỗi byte liền kề trong bộ nhớ hoặc thông báo cho hệ điều hành về một số khía cạnh của quá trình tải.

Các tiêu đề của phần cũng cho Windows biết những quyền mà nó sẽ cấp cho các phần, chẳng hạn như liệu chương trình có thể đọc, ghi hoặc thực thi chúng khi chương trình đang thực thi hay không. Ví dụ: phần .text chứa mã x86 thường sẽ được đánh dấu là có thể đọc và thực thi nhưng không thể ghi để ngăn mã chương trình vô tình tự sửa đổi trong quá trình thực thi.

Một số phần, chẳng hạn như .text và .rsrc, được mô tả trong Hình 1-1. Chúng được ánh xạ vào bộ nhớ khi tệp PE được thực thi. Các phần đặc biệt khác, chẳng hạn như phần .reloc, không được ánh xạ vào bộ nhớ. Chúng ta cũng sẽ thảo luận về những phần này. Hãy xem qua các phần được hiển thị trong Hình 1-1.

Phần .text

Mỗi chương trình PE chứa ít nhất một phần mã x86 được đánh dấu là có thể thực thi được trong tiêu đề phần của nó; những phần này hầu như luôn được đặt tên là .text y. Chúng ta sẽ phân tách dữ liệu trong phần .text khi thực hiện phân tách chương trình và kỹ thuật đảo ngược trong Chương 2.

Phần .idata

Phần .idata z, còn được gọi là nhập, chứa Bảng Địa chỉ Nhập (IAT), liệt kê các thư viện được liên kết động và các chức năng của chúng. IAT là một trong những cấu trúc PE quan trọng nhất cần kiểm tra khi ban đầu tiếp cận tệp nhị phân PE để phân tích vì nó cho thấy thư viện gọi một chương trình tạo ra, do đó có thể phản bội chức năng cấp cao của phần mềm độc hại.

Các phần dữ liệu

Các phần dữ liệu trong tệp PE có thể bao gồm các phần như .rsrc, .data và .rdata, lưu trữ các mục như hình ảnh con trỏ chuột, giao diện nút, âm thanh và phương tiện khác được chương trình sử dụng. Ví dụ: phần .rsrc { trong Hình 1-1 chứa các chuỗi ký tự có thể in được mà chương trình sử dụng để hiển thị văn bản dưới dạng chuỗi.

Thông tin trong phần .rsrc (tài nguyên) có thể rất quan trọng đối với phần mềm độc hại các nhà phân tích bởi vì bằng cách kiểm tra các chuỗi ký tự có thể in, hình ảnh đồ họa và các nội dung khác trong tệp PE, họ có thể thu được manh mối quan trọng về chức năng của tệp. Trong "Kiểm tra hình ảnh phần mềm độc hại" trên trang 7, bạn sẽ tìm hiểu cách sử dụng bộ công cụ icoutils (bao gồm icotool và wrestool) để trích xuất hình ảnh đồ họa từ các phần tài nguyên của tệp nhị phân phần mềm độc hại. Sau đó, trong "Kiểm tra các chuỗi phần mềm độc hại" ở trang 8, bạn sẽ tìm hiểu cách trích xuất các chuỗi có thể in được từ các phần tài nguyên phần mềm độc hại.

Phần .reloc

Mã nhị phân PE không độc lập với vị trí, có nghĩa là nó sẽ không thực thi chính xác nếu nó được di chuyển từ vị trí bộ nhớ dự kiến sang vị trí bộ nhớ mới. Phần .reloc | giải quyết vấn đề này bằng cách cho phép mã được di chuyển mà không bị hỏng. Nó yêu cầu hệ điều hành Windows chuyển địa chỉ bộ nhớ muốn trong mã của tệp PE nếu mã đã được chuyển để mã vẫn chạy chính xác. Các bản dịch này thường liên quan đến việc cộng hoặc trừ một phần bù từ một địa chỉ bộ nhớ.

Mặc dù phần .reloc của tệp PE có thể chứa thông tin bạn muốn sử dụng trong quá trình phân tích phần mềm độc hại, nhưng chúng tôi sẽ không thảo luận thêm về phần này trong cuốn sách này vì trọng tâm của chúng tôi là áp dụng máy học và phân tích dữ liệu cho phần mềm độc hại, chứ không phải loại của kỹ thuật đảo ngược hardcore liên quan đến việc xem xét chuyển vị trí.

Phân tích định dạng PE bằng pefile

Mô -đun pefile Python, được viết và duy trì bởi Ero Carrera, đã trở thành một thư viện phân tích phần mềm độc hại tiêu chuẩn ngành để phân tích các tệp PE. Trong phần này tôi hướng dẫn các bạn cách sử dụng pefile để mở xé ircbot.exe. Có thể tìm thấy tệp ircbot.exe trên máy ảo đi kèm cuốn sách này trong thư mục ~/malware_data_science/ch1/data. Liệt kê 1-1 giả định rằng ircbot.exe nằm trong thư mục làm việc hiện tại của bạn.

Nhập thông tin sau để cài đặt thư viện pefile để chúng tôi có thể nhập nó trong Python:

\$ pip cài đặt pefile

Bây giờ, hãy sử dụng các lệnh trong Liệt kê 1-1 để khởi động Python, nhập tệp pefile mô -đun, mở và phân tích cú pháp tệp PE ircbot.exe bằng cách sử dụng pefile.

\$ trấn
>>> nhập pefile
>>> pe = pefile.PE("ircbot.exe")

Liệt kê 1-1: Đang tải mô -đun pefile và phân tích cú pháp tệp PE (ircbot.exe)

Chúng tôi khởi tạo pefile.PE, đây là lớp lõi được triển khai bởi mô-đun PE. Nó phân tích các tệp PE để chúng tôi có thể kiểm tra các thuộc tính của chúng. Bằng cách gọi hàm tạo PE, chúng tôi tải và phân tích tệp PE đã chỉ định, đó là ircbot.exe trong ví dụ này. Bây giờ chúng ta đã tải và phân tích cú pháp tệp của mình, hãy chạy mã trong Liệt kê 1-2 để lấy thông tin từ các trường PE của ircbot.exe .

```
# dựa trên mã ví dụ của Ero Carrera (tác giả thư viện pefile)
cho phần trong pe.sections:
    in (phần. Tên, hex (phần. VirtualAddress),
        hex(section.Misc_VirtualSize), section.SizeOfRawData )
```

Liệt kê 1-2: Lặp lại qua các phần của tệp PE và in thông tin về chúng

Liệt kê 1-3 cho thấy đầu ra.

```
(''.text\x00\x00\x00', '0x1000', '0x32830', w207360)
(''.rdata\x00\x00', '0x340000', '0x427a', 17408)
(''.data\x00\x00\x00', '0x390000', '0x5cff8', 10752)
(''.idata\x00\x00', '0x960000', '0xbb0', 3072)
(''.reloc\x00\x00', '0x970000', '0x211d', 8704)
```

Liệt kê 1-3: Kéo dữ liệu phần tử ircbot.exe bằng mô-đun pefile của Python

Như bạn có thể thấy trong Liệt kê 1-3, chúng tôi đã lấy dữ liệu từ năm giây khác nhau các phần của tệp PE: .text, .rdata, .data, .idata và .reloc. Đầu ra được đưa ra dưới dạng năm bộ, một bộ cho mỗi phần PE được kéo. Mục nhập đầu tiên trên mỗi dòng xác định phần PE. (Bạn có thể bỏ qua chuỗi \x00 byte null, vốn chỉ đơn giản là các dấu kết thúc chuỗi null kiểu C.) Các trường còn lại cho chúng tôi biết mức sử dụng bộ nhớ của từng phần sau khi được tải vào bộ nhớ và nó sẽ được tìm thấy ở đâu trong bộ nhớ một lần nạp vào.

Ví dụ: 0x1000 là địa chỉ bộ nhớ ảo cơ sở nơi các giây này sẽ được tải. Hãy coi đây là địa chỉ bộ nhớ cơ sở của phần. 0x32830 trong trường kích thước ảo chỉ định dung lượng bộ nhớ mà phần này yêu cầu sau khi được tải. 207360 trong trường thứ ba biểu thị lượng dữ liệu mà phần sẽ chiếm trong đoạn bộ nhớ đó .

Ngoài việc sử dụng pefile để phân tích cú pháp các phần của chương trình, chúng ta cũng có thể sử dụng nó để liệt kê các tệp DLL mà tệp nhị phân sẽ tải, cũng như hàm gọi nó sẽ yêu cầu trong các tệp DLL đó. Chúng tôi có thể làm điều này bằng cách xuất IAT của tệp PE. Liệt kê 1-4 cho thấy cách sử dụng pefile để xuất IAT cho ircbot.exe.

```
$ trấn
pe = pefile.PE("ircbot.exe")
cho mục nhập trong pe.DIRECTORY_ENTRY_IMPORT:
    in entry.dll
    cho chức năng trong entry.imports:
        in '\t',function.name
```

Liệt kê 1-4: Trích xuất nhập khẩu từ ircbot.exe

Liệt kê 1-4 sẽ tạo ra đầu ra như trong Liệt kê 1-5 (được cắt ngắn cho ngắn gọn).

KERNEL32.DLL

```

Nhận thời gian cục bộ
thoát chủ đề
ĐóngXử lý
GhiTệp
Tạo FileA
Quá trình thoát
TạoProcessA
GetTickCount
GetModuleFileNameA

--snip--

```

Liệt kê 1-5: Nội dung IAT của ircbot.exe, hiển thị các chức năng thư viện được phần mềm độc hại này sử dụng

Như bạn có thể thấy trong Liệt kê 1-5, điều ra này rất có giá trị đối với sis phân tích phần mềm độc hại vì nó liệt kê một loạt các chức năng mà phần mềm độc hại khai báo và sẽ tham chiếu. Ví dụ: một vài dòng đầu tiên của kết quả cho chúng tôi biết phần mềm độc hại sẽ ghi vào tệp bằng WriteFile , mở tệp bằng CreateFileA gọi , và tạo các tiến trình mới sử dụng CreateProcessA . Mặc dù đây là thông tin khá cơ bản về phần mềm độc hại, nhưng đây là bước khởi đầu để hiểu chi tiết hơn về hành vi của phần mềm độc hại.

Kiểm tra hình ảnh phần mềm độc hại

Để hiểu cách phần mềm độc hại có thể được thiết kế để đánh lừa mục tiêu, hãy xem các biểu tượng có trong phần .rsrc của nó . Ví dụ: các tệp nhị phân của phần mềm độc hại thường được thiết kế để lừa người dùng nhập vào chúng bằng cách giả dạng tài liệu Word, trình cài đặt trò chơi, tệp PDF, v.v. Bạn cũng tìm thấy hình ảnh trong phần mềm độc hại gợi ý các chương trình mà chính kẻ tấn công quan tâm, chẳng hạn như các công cụ tấn công mạng và chương trình do kẻ tấn công chạy để điều khiển từ xa các máy bị xâm nhập. Tôi thậm chí đã nhìn thấy các tệp nhị phân chứa các biểu tượng trên màn hình của các chiến binh thánh chiến, hình ảnh của các nhân vật hoạt hình trên mạng trông có vẻ xấu xí và hình ảnh của súng trường Kalashnikov. Đối với phân tích hình ảnh mẫu của chúng tôi, hãy xem xét một mẫu phần mềm độc hại mà công ty bảo mật Mandiant đã xác định là do một nhóm tin tặc do nhà nước Trung Quốc tài trợ tạo ra. Bạn có thể tìm thấy phần mềm độc hại mẫu này trong thư mục dữ liệu của chương này dưới tên fakepdfmalware.exe. Mẫu này sử dụng biểu tượng Adobe Acrobat để đánh lừa người dùng nghĩ rằng đó là tài liệu Adobe Acrobat, trong khi thực tế đó là tệp thực thi PE độc hại.

Tước khi chúng ta có thể trích xuất các hình ảnh từ tệp nhị phân fakepdfmalware.exe bằng cách sử dụng wrestool của công cụ dòng lệnh Linux , trước tiên chúng ta cần tạo một thư mục để chứa các hình ảnh mà chúng ta sẽ trích xuất. Liệt kê 1-6 cho thấy cách thực hiện tất cả điều này.

```

$ mkdir hình ảnh
$ wrestool -x fakepdfmalware.exe -output=images
$ icotool -x -o hình ảnh hình ảnh/*.ico

```

Liệt kê 1-6: Các lệnh Shell trích xuất hình ảnh từ một mẫu phần mềm độc hại

Trước tiên, chúng tôi sử dụng hình ảnh mkdir để tạo thư mục chứa tệp đã giải nén hình ảnh. Tiếp theo, chúng tôi sử dụng wrestool để trích xuất tài nguyên hình ảnh (-x) từ fakepdfmalware.exe sang /images , sau đó sử dụng icotool để trích xuất (-x) và chuyển đổi (-o) bất kỳ tài nguyên nào ở định dạng biểu tượng Adobe .ico thành đồ họa .png . rằng chúng ta có thể xem chúng bằng các công cụ xem ảnh tiêu chuẩn. Nếu bạn chưa cài đặt wrestool trên hệ thống của mình, bạn có thể tải xuống tại <http://www.hongnu.org/icoutils/>.

Khi bạn đã sử dụng wrestool để chuyển đổi hình ảnh trong tệp thực thi đích có thể sang định dạng PNG, bạn sẽ có thể mở chúng trong trình xem ảnh yêu thích của mình và xem biểu tượng Adobe Acrobat ở các độ phân giải khác nhau. Như ví dụ của tôi ở đây chúng mình, việc trích xuất hình ảnh và biểu tượng từ tệp PE tương đối đơn giản và có thể nhanh chóng tiết lộ thông tin thú vị và hữu ích về các tệp nhị phân phần mềm độc hại. Tương tự, chúng ta có thể dễ dàng trích xuất các chuỗi có thể in được từ phần mềm độc hại để biết thêm thông tin, điều mà chúng ta sẽ thực hiện tiếp theo.

Kiểm tra chuỗi phần mềm độc hại

Chuỗi là chuỗi các ký tự có thể in được trong một chương trình nhị phân. Các nhà phân tích phần mềm độc hại thường dựa vào các chuỗi trong một mẫu độc hại để hiểu nhanh những gì có thể xảy ra bên trong nó. Các chuỗi này thường chứa những thứ như lệnh HTTP và FTP tải xuống các trang web và tệp, địa chỉ IP và tên máy chủ cho bạn biết phần mềm độc hại kết nối với địa chỉ nào và những thứ tương tự. Đôi khi, ngay cả ngôn ngữ được sử dụng để viết các chuỗi cũng có thể gợi ý về quốc gia xuất xứ của mã nhị phân phần mềm độc hại, mặc dù điều này có thể bị làm giả. Bạn thậm chí có thể tìm thấy văn bản trong một chuỗi giải thích bằng leetspeak mục đích của tệp nhị phân độc hại.

Chuỗi cũng có thể tiết lộ thêm thông tin kỹ thuật về nhị phân. Vì ví dụ, bạn có thể tìm thấy thông tin về trình biên dịch được sử dụng để tạo nó, ngôn ngữ lập trình mà tệp nhị phân được viết bằng, tập lệnh nhúng hoặc HTML, v.v. Mặc dù các tác giả phần mềm độc hại có thể làm xáo trộn, mã hóa và nén tất cả các dấu vết này, nhưng ngay cả các tác giả phần mềm độc hại nâng cao cũng thường để lại ít nhất một số dấu vết, điều này khiến việc kiểm tra các kết xuất chuỗi khi phân tích phần mềm độc hại trở nên đặc biệt quan trọng.

Sử dụng chương trình chuỗi

Cách tiêu chuẩn để xem tất cả các chuỗi trong một tệp là sử dụng chuỗi công cụ dòng lệnh, sử dụng cú pháp sau:

\$ chuỗi tệp đường dẫn | ít hơn

Lệnh này in tất cả các chuỗi trong một tệp vào thiết bị đầu cuối, từng dòng một. Thêm | less ở cuối ngăn các chuỗi chỉ cuộn qua thiết bị đầu cuối. Theo mặc định, lệnh strings tìm tất cả các chuỗi có thể in được với độ dài tối thiểu là 4 byte, nhưng bạn có thể đặt độ dài tối thiểu khác và thay đổi nhiều tham số khác, như được liệt kê trong trang hướng dẫn sử dụng lệnh. Tôi khuyên bạn chỉ nên sử dụng độ dài chuỗi tối thiểu mặc định là 4,

nhưng bạn có thể thay đổi độ dài chuỗi tối thiểu bằng tùy chọn -n . Ví dụ: chuỗi -n 10 filepath sẽ chỉ trích xuất các chuỗi có độ dài tối thiểu là 10 byte.

Phân tích kết xuất chuỗi của bạn

Bây giờ chúng ta đã loại bỏ các chuỗi có thể in được của chương trình phần mềm độc hại, thách thức là hiểu ý nghĩa của các chuỗi này. Ví dụ: giả sử chúng ta kết xuất các chuỗi vào tệp ircbotstring.txt cho ircbot.exe mà chúng ta đã khám phá trước đó trong chương này bằng thư viện pefile , như sau:

```
$ chuỗi ircbot.exe > ircbotstring.txt
```

Nội dung của ircbotstring.txt chứa hàng nghìn dòng văn bản, nhưng một số dòng trong số này sẽ nổi bật. Ví dụ, Liệt kê 1-7 hiển thị một loạt các dòng được trích xuất từ kết xuất chuỗi bắt đầu bằng từ TẢI XUỐNG.

```
[TẢI XUỐNG]: URL không hợp lệ hoặc Lỗi DNS: %s.
[TẢI XUỐNG]: Cập nhật không thành công: Lỗi khi thực thi tệp: %s.
[TẢI XUỐNG]: Đã tải %.1fKB xuống %s @ %.1fKB/giây. Đang cập nhật.
[TẢI XUỐNG]: Đã mở: %s.
--snip--
[TẢI XUỐNG]: Đã tải xuống %.1f KB thành %s @ %.1f KB/giây.
[TẢI XUỐNG]: CRC không thành công (%d != %d).
[DOWNLOAD]: Kích thước tệp không chính xác: (%d != %d).
[TẢI XUỐNG]: Cập nhật: %s (đã chuyển %dKB).
[TẢI XUỐNG]: Tải xuống tệp: %s (đã chuyển %dKB).
[TẢI XUỐNG]: Không thể mở tệp: %s.
```

Liệt kê 1-7: Đầu ra chuỗi hiển thị bằng chứng cho thấy phần mềm độc hại có thể tải xuống các tệp do kẻ tấn công chỉ định vào máy mục tiêu

Những dòng này chỉ ra rằng ircbot.exe sẽ có tải xuống các tệp do kẻ tấn công chỉ định vào máy mục tiêu.

Hãy thử phân tích một cái khác. Kết xuất chuỗi được hiển thị trong Liệt kê 1-8 chỉ ra rằng ircbot.exe có thể hoạt động như một máy chủ web lắng nghe trên máy mục tiêu để tìm các kết nối từ kẻ tấn công.

NHẬN

HTTP/1.0 200 OK

Máy chủ: myBot

Kiểm soát bộ đếm: không có bộ đếm, không lưu trữ, tuổi tối đa = 0

pragma: không có bộ đếm

Loại nội dung

Độ dài nội dung: %i

Phạm vi chấp nhận: byte

Ngày: %s %s GMT

Sửa lần cuối: %s %s GMT

Hết hạn: %s %s GMT

Kết nối: đóng

HTTP/1.0 200 OK

Máy chủ: myBot

Kiểm soát bộ đệm: không có bộ đệm, không lưu trữ, tuổi tối đa = 0
 pragma: không có bộ đệm
 Loại nội dung
 Phạm vi chấp nhận: byte
 Ngày: %s %s GMT
 Sửa lần cuối: %s %s GMT
 Hết hạn: %s %s GMT
 Kết nối: đóng
 HH:mm:ss
 dd, d MMM yyyy
 ứng dụng/octet-stream
 văn bản/html

Lỗi kẽ 1-8: Đầu ra chuỗi cho thấy phần mềm độc hại có máy chủ HTTP mà kẻ tấn công có thể kết nối

Lỗi kẽ 1-8 cho thấy nhiều bản soạn sẵn HTTP khác nhau được sử dụng bởi ircbot.exe để thực hiện một máy chủ HTTP. Có khả năng máy chủ HTTP này cho phép kẻ tấn công kết nối với máy đích thông qua HTTP để ra lệnh, chẳng hạn như lệnh chụp ảnh màn hình màn hình của nạn nhân và gửi lại cho kẻ tấn công. Chúng tôi thấy bằng chứng về chức năng HTTP trong toàn bộ danh sách. Ví dụ: phương thức GET yêu cầu dữ liệu từ tài nguyên internet.

Dòng HTTP/1.0 200 OK là một chuỗi HTTP trả về mã trạng thái 200, cho biết rằng mọi việc diễn ra suôn sẻ với một giao dịch mạng HTTP và Máy chủ: myBot cho biết rằng tên của máy chủ HTTP là myBot, một quà tặng mà ircbot.exe có máy chủ HTTP tích hợp.

Tất cả thông tin này đều hữu ích trong việc hiểu và ngăn chặn một mẫu phần mềm độc hại hoặc chiến dịch độc hại cụ thể. Ví dụ: biết rằng mẫu phần mềm độc hại có máy chủ HTTP xuất ra một số chuỗi nhất định khi bạn kết nối với máy chủ đó cho phép bạn quét mạng của mình để xác định các máy chủ bị nhiễm.

Bản tóm tắt

Trong chương này, bạn đã có cái nhìn tổng quan cấp cao về phân tích phần mềm độc hại tĩnh, bao gồm việc kiểm tra một chương trình phần mềm độc hại mà không thực sự chạy nó. Bạn đã tìm hiểu về định dạng tệp PE xác định các tệp .exe và .dll của Windows, đồng thời bạn đã học cách sử dụng thư viện Python pefile để phân tích tệp nhị phân ircbot.exe của phần mềm độc hại trong thế giới thực. Bạn cũng đã sử dụng các kỹ thuật phân tích tĩnh như phân tích hình ảnh và phân tích chuỗi để trích xuất thêm thông tin từ các mẫu phần mềm độc hại. Chương 2 tiếp tục thảo luận về phân tích phần mềm độc hại tĩnh với trọng tâm là phân tích mã hợp ngữ có thể được phục hồi từ phần mềm độc hại.

2

Beyond Basic Static Analysis: x86 Disassembly



Để hiểu thấu đáo về một chương trình độc hại, chúng ta thường cần vượt qua phân tích tĩnh cơ bản về các phần, chuỗi, nhập và hình ảnh của nó. Điều này liên quan đến kỹ thuật đảo ngược mã hợp ngữ của chương trình. Thực vậy, tháo gỡ và kỹ thuật đảo ngược nằm ở trung tâm của phân tích tĩnh sâu các mẫu phần mềm độc hại.

Vì kỹ thuật đảo ngược là một nghệ thuật, thủ công kỹ thuật và khoa học nên việc khám phá kỹ lưỡng nằm ngoài phạm vi của chương này. Mục tiêu của tôi ở đây là giới thiệu cho bạn về kỹ thuật đảo ngược để bạn có thể áp dụng nó vào khoa học dữ liệu phần mềm độc hại. Hiểu phương pháp này là điều cần thiết để áp dụng thành công máy học và phân tích dữ liệu cho phần mềm độc hại.

Trong chương này, tôi bắt đầu với các khái niệm mà bạn sẽ cần để hiểu về việc tháo gỡ x86. Ở phần sau của chương này, tôi sẽ chỉ ra cách mà các tác giả phần mềm độc hại cố gắng vượt qua quá trình tháo gỡ và thảo luận về các cách để giảm thiểu các thao tác chống phân tích và chống phát hiện này. Nhưng trước tiên, chúng ta hãy điểm lại một số phương pháp tháo gỡ phổ biến cũng như kiến thức cơ bản về hợp ngữ x86.

Phương pháp tháo gỡ

Gỡ bỏ là quá trình chúng tôi dịch mã nhị phân của phần mềm độc hại sang ngôn ngữ hợp ngữ x86 hợp lệ. Tác giả phần mềm độc hại thường viết các chương trình phần mềm độc hại bằng ngôn ngữ cấp cao như C hoặc C++, sau đó sử dụng trình biên dịch để biên dịch mã nguồn thành mã nhị phân x86. Hợp ngữ là biểu diễn mà con người có thể đọc được của mã nhị phân này. Do đó, việc phân tích một chương trình phần mềm độc hại thành ngôn ngữ hợp ngữ là cần thiết để hiểu cách hoạt động của nó ở cốt lõi.

Thật không may, việc tháo gỡ không dễ dàng vì các tác giả phần mềm độc hại thường sử dụng các thủ thuật để cản trở các kỹ sư đảo ngược. Trên thực tế, việc tháo gỡ hoàn hảo khi đối mặt với sự che giấu có chủ ý là một vấn đề chưa được giải quyết trong khoa học máy tính. Hiện tại, chỉ tồn tại các phương pháp gần đúng, dễ bị lỗi để phân tách các chương trình như vậy.

Ví dụ, hãy xem xét trường hợp mã tự sửa đổi hoặc mã nhị phân tự sửa đổi khi nó thực thi. Cách duy nhất để tách mã này đúng cách là hiểu logic chương trình mà mã tự sửa đổi, nhưng điều đó có thể cực kỳ phức tạp.

Bởi vì việc tháo gỡ hoàn hảo hiện tại là không thể, chúng ta phải sử dụng các phương pháp không hoàn hảo để hoàn thành nhiệm vụ này. Phương pháp chúng tôi sẽ sử dụng là phân tách tuyến tính, bao gồm việc xác định chuỗi byte liền kề trong tệp Có thể thực thi di động (PE) tương ứng với mã chương trình x86 của nó và sau đó giải mã các byte này. Hạn chế chính của cách tiếp cận này là nó bỏ qua sự tinh tế về cách giải mã các lệnh bởi CPU trong quá trình thực thi chương trình. Ngoài ra, nó không tính đến các mã độc gây nhiễu khác nhau mà các tác giả phần mềm độc hại đãi khi sử dụng để làm cho chương trình của họ khó phân tích hơn.

Các phương pháp khác của kỹ thuật đảo ngược, mà chúng tôi sẽ không đề cập ở đây, là các phương pháp tháo gỡ phức tạp hơn được sử dụng bởi các trình giả lập tháo gỡ cấp độ công nghiệp như IDA Pro. Các phương pháp nâng cao hơn này thực sự mô phỏng hoặc lập luận về việc thực thi chương trình để khám phá ra hướng dẫn hợp ngữ nào mà chương trình có thể đạt được do một loạt các nhánh có điều kiện.

Mặc dù kiểu tháo gỡ này có thể chính xác hơn so với tháo gỡ tuyến tính, nhưng nó sử dụng nhiều CPU hơn so với các phương pháp tháo gỡ tuyến tính, khiến nó ít phù hợp hơn cho các mục đích khoa học dữ liệu, nơi tập trung vào việc tháo gỡ hàng nghìn hoặc thậm chí hàng triệu chương trình.

Tuy nhiên, trước khi bạn có thể bắt đầu phân tích bằng cách tháo gỡ tuyến tính, bạn sẽ cần xem lại các thành phần cơ bản của hợp ngữ.

Khái niệm cơ bản về hợp ngữ x86

Hợp ngữ là ngôn ngữ lập trình cấp thấp nhất mà con người có thể đọc được cho một kiến trúc nhất định và ngôn ngữ này ánh xạ gần với định dạng lệnh nhị phân của một kiến trúc CPU cụ thể. Một dòng hợp ngữ hầu như luôn tương đương với một lệnh CPU. Vì quá trình lắp ráp ở mức thấp nên bạn thường có thể truy xuất nó dễ dàng từ tệp nhị phân của phần mềm độc hại bằng cách sử dụng các công cụ phù hợp.

Đạt được trình độ cơ bản trong việc đọc mã phần mềm độc hại x86 đã phân tách là dễ dàng hơn bạn có thể nghĩ. Điều này là do hầu hết mã hợp ngữ của phần mềm độc hại dành phần lớn thời gian gọi vào hệ điều hành thông qua các thư viện liên kết động (DLL) của hệ điều hành Windows, được tải vào bộ nhớ chương trình khi chạy. Các chương trình phần mềm độc hại sử dụng DLL để thực hiện hầu hết các công việc thực tế, chẳng hạn như sửa đổi sổ đăng ký hệ thống, di chuyển và sao chép tệp, tạo kết nối mạng và giao tiếp qua các giao thức mạng, v.v. Do đó, việc tuân theo mã hợp ngữ của phần mềm độc hại thường liên quan đến việc hiểu cách thực hiện các lời gọi hàm từ hợp ngữ và hiểu những lời gọi DLL khác nhau thực hiện. Tất nhiên, mọi thứ có thể trở nên phức tạp hơn nhiều, nhưng biết nhiều điều này có thể tiết lộ nhiều điều về phần mềm độc hại.

Trong các phần tiếp theo tôi giới thiệu một số khái niệm hợp ngữ quan trọng. Tôi cũng giải thích một số khái niệm trừu tượng như luồng điều khiển và đồ thị luồng điều khiển. Cuối cùng, chúng tôi tháo rời chương trình ircbot.exe và khám phá cách quy trình lắp ráp và điều khiển của nó có thể giúp chúng tôi hiểu rõ hơn về mục đích của nó.

Có hai phương ngữ chính của lắp ráp x86: Intel và AT&T. Trong cuốn sách này, tôi sử dụng cú pháp Intel, cú pháp này có thể được lấy từ tất cả các trình dịch ngược chính và là cú pháp được sử dụng trong tài liệu chính thức của Intel về CPU x86.

Hãy bắt đầu bằng cách xem các thanh ghi CPU.

Thanh ghi CPU

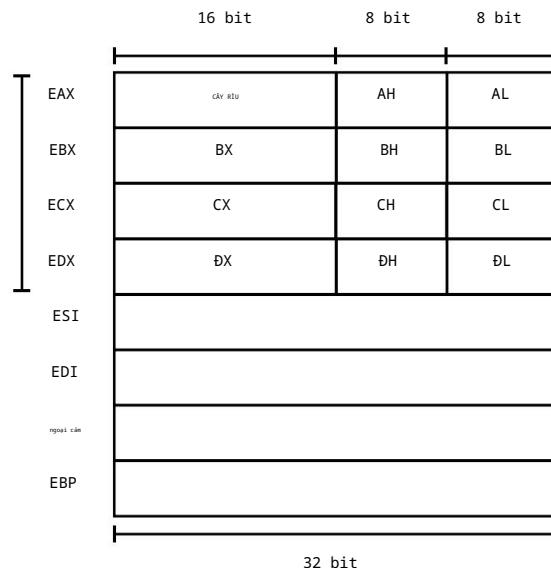
Các thanh ghi là các đơn vị lưu trữ dữ liệu nhỏ trên đó các CPU x86 thực hiện tính toán. Bởi vì các thanh ghi nằm trên chính CPU, nên việc truy cập thanh ghi nhanh hơn nhiều so với truy cập bộ nhớ. Đây là lý do tại sao các hoạt động tính toán cốt lõi, chẳng hạn như hướng dẫn kiểm tra điều kiện và số học, tất cả các thanh ghi mục tiêu. Đó cũng là lý do tại sao CPU sử dụng các thanh ghi để lưu trữ thông tin về trạng thái của các chương trình đang chạy. Mặc dù nhiều thanh ghi có sẵn cho các lập trình viên hợp ngữ x86 có kinh nghiệm, chúng ta sẽ chỉ tập trung vào một vài thanh ghi quan trọng ở đây.

Sổ đăng ký mục đích chung

Các thanh ghi mục đích chung giống như không gian đầu cho các lập trình viên hợp ngữ. Trên hệ thống 32 bit, mỗi thanh ghi này chứa 32, 16 hoặc 8 bit không gian mà chúng ta có thể thực hiện các phép toán số học, phép toán theo bit, phép toán hoán đổi thứ tự byte, v.v.

Trong các quy trình công việc tính toán thông thường, các chương trình di chuyển dữ liệu vào các thanh ghi từ bộ nhớ hoặc từ các thiết bị phần cứng bên ngoài, thực hiện một số thao tác trên dữ liệu này, sau đó di chuyển dữ liệu trở lại bộ nhớ để lưu trữ. Ví dụ: để sắp xếp một danh sách dài, một chương trình thường lấy các mục trong danh sách từ một mảng trong bộ nhớ, so sánh chúng trong các thanh ghi, sau đó ghi kết quả so sánh trở lại bộ nhớ.

Để hiểu một số đặc điểm của thanh ghi mục đích chung
mô hình trong kiến trúc Intel 32-bit, hãy xem Hình 2-1.



Hình 2-1: Các thanh ghi trong kiến trúc x86

Trục dọc hiển thị bố cục của các thanh ghi mục đích chung và trục hoành cho biết cách chia nhỏ EAX, EBX, ECX và EDX.

EAX, EBX, ECX và EDX là các thanh ghi 32 bit có các thanh ghi 16 bit nhỏ hơn bên trong chúng: AX, BX, CX và DX. Như bạn có thể thấy trong hình, các thanh ghi 16 bit này có thể được chia nhỏ thành các thanh ghi 8 bit trên và dưới: AH, AL, BH, BL, CH, CL, DH và DL. Mặc dù đôi khi rất hữu ích khi giải quyết các phân mục trong EAX, EBX, ECX và EDX, nhưng hầu như bạn sẽ thấy các tham chiếu trực tiếp đến EAX, EBX, ECX và EDX.

Thanh ghi luồng ngăn xếp và điều khiển

Các thanh ghi quản lý ngăn xếp lưu trữ thông tin quan trọng về ngăn xếp chương trình, chịu trách nhiệm lưu trữ các biến cục bộ cho các hàm, các đối số được truyền vào các hàm và thông tin điều khiển liên quan đến luồng điều khiển chương trình. Hãy xem qua một số thanh ghi này.

Nói một cách đơn giản, thanh ghi ESP trả tới định nghĩa ngăn xếp cho chức năng hiện đang thực thi, trong khi thanh ghi EBP trả tới đáy ngăn xếp cho chức năng hiện đang thực thi. Đây là thông tin quan trọng đối với các chương trình hiện đại, bởi vì nó có nghĩa là bằng cách tham chiếu dữ liệu liên quan đến ngăn xếp thay vì sử dụng địa chỉ tuyệt đối của nó, mã thủ tục và mã hướng đối tượng có thể truy cập các biến cục bộ một cách hiệu quả và duyên dáng hơn.

Mặc dù bạn sẽ không thấy các tham chiếu trực tiếp đến thanh ghi EIP trong mã hợp ngữ x86, nhưng nó rất quan trọng trong phân tích bảo mật, đặc biệt là trong bối cảnh nghiên cứu lỗ hổng và phát triển khai thác lỗ tràn bộ đệm. Điều này là do EIP chứa địa chỉ bộ nhớ của lệnh đang thực hiện. Những kẻ tấn công có thể sử dụng khai thác tràn bộ đệm để gián tiếp làm hỏng giá trị của thanh ghi EIP và kiểm soát việc thực thi chương trình.

Ngoài vai trò khai thác, EIP cũng rất quan trọng trong việc phân tích mã độc do phần mềm độc hại triển khai. Sử dụng trình gõ lỗi, chúng tôi có thể kiểm tra giá trị của EIP bất kỳ lúc nào, điều này giúp chúng tôi hiểu phần mềm độc hại mã nào đang thực thi tại bất kỳ thời điểm cụ thể nào.

EFLAGS là một thanh ghi trạng thái chứa các cờ CPU, là các bit lưu trữ thông tin trạng thái về trạng thái của chương trình hiện đang thực thi. Thanh ghi EFLAGS là trung tâm của quá trình tạo các nhánh có điều kiện hoặc các thay đổi trong luồng thực thi do kết quả của logic chương trình kiểu if/then, trong các chương trình x86. Cụ thể, bắt cứ khi nào một chương trình hợp ngữ x86 kiểm tra xem một giá trị nào đó lớn hơn hay nhỏ hơn 0 và sau đó nhảy tới một hàm dựa trên kết quả của phép thử này, thanh ghi EFLAGS sẽ đóng vai trò hỗ trợ, như được mô tả chi tiết hơn trong "Khối cơ bản và Điều khiển Flow Graphs" ở trang 19.

Hướng dẫn số học

Hướng dẫn hoạt động trên các thanh ghi mục đích chung. Bạn có thể thực hiện các phép tính đơn giản với các thanh ghi đa năng bằng cách sử dụng các lệnh số học. Ví dụ: add, sub, inc, dec và mul là những ví dụ về hướng dẫn số học mà bạn sẽ gặp thường xuyên trong kỹ thuật đảo ngược phần mềm độc hại.

Bảng 2-1 liệt kê một số ví dụ về các lệnh cơ bản và cú pháp của chúng.

Bảng 2-1: Các lệnh số học

Hướng dẫn	Sự miêu tả
thêm ebx, 100	Thêm 100 vào giá trị trong EBX và sau đó lưu trữ kết quả trong EBX
phụ ebx, 100	Trừ 100 từ giá trị trong EBX và sau đó lưu trữ kết quả trong EBX
tăng đoán ah	Tăng giá trị trong AH lên 1
đè can	Giảm giá trị trong AL đi 1

Lệnh add cộng hai số nguyên và lưu kết quả trong toán hạng đầu tiên được chỉ định, cho dù đây là vị trí bộ nhớ hay thanh ghi theo cú pháp sau. Hãy ghi nhớ chỉ một đối số có thể là một vị trí bộ nhớ. Lệnh phụ tương tự như phép cộng, ngoại trừ phép trừ các số nguyên. Lệnh inc tăng giá trị nguyên của thanh ghi hoặc vị trí bộ nhớ, trong khi lệnh dec giảm giá trị nguyên của thanh ghi hoặc vị trí bộ nhớ.

Hướng dẫn di chuyển dữ liệu

Bộ xử lý x86 cung cấp một bộ hướng dẫn mạnh mẽ để di chuyển dữ liệu giữa các thanh ghi và bộ nhớ. Các hướng dẫn này cung cấp các cơ chế cơ bản cho phép chúng tôi thao tác dữ liệu. Lệnh di chuyển bộ nhớ chính là lệnh mov. Bảng 2-2 cho thấy cách bạn có thể sử dụng lệnh mov để di chuyển dữ liệu.

Bảng 2-2: Hướng dẫn di chuyển dữ liệu

Hướng dẫn	Sự miêu tả
di chuyển ebx, eax	Di chuyển giá trị trong thanh ghi EAX vào thanh ghi EBX
di chuyển eax, [0x12345678]	Di chuyển dữ liệu tại địa chỉ bộ nhớ 0x12345678 vào thanh ghi EAX
mov edx, 1	Chuyển giá trị 1 vào thanh ghi EDX
mov [0x12345678], eax	Di chuyển giá trị trong EAX vào vị trí bộ nhớ 0x12345678

Liên quan đến lệnh mov, lệnh lea tải địa chỉ bộ nhớ tuyệt đối được chỉ định vào thanh ghi được sử dụng để nhận con trỏ tới vị trí bộ nhớ. Ví dụ: lea edx, [esp-4] trừ 4 khỏi giá trị trong ESP và tải giá trị kết quả vào EDX.

Hướng dẫn ngăn xếp

Ngăn xếp trong tổ hợp x86 là một cấu trúc dữ liệu cho phép bạn đẩy và bật các giá trị lên và xuống của nó. Điều này tương tự như cách bạn thêm và bớt các đĩa trên và ngoài của chồng đĩa.

Bởi vì luồng điều khiển thường được thể hiện thông qua các lời gọi hàm kiểu C trong hợp ngữ x86 và bởi vì những lời gọi hàm này sử dụng ngăn xếp để truyền các đối số, cấp phát các biến cục bộ và ghi nhớ phần nào của chương trình sẽ quay lại sau khi một hàm kết thúc thực thi, nên ngăn xếp và luồng điều khiển cần được hiểu cùng nhau.

Lệnh đẩy đẩy các giá trị vào ngăn xếp chương trình khi lập trình viên chuyên nghiệp muốn lưu một giá trị thanh ghi vào ngăn xếp và lệnh pop xóa các giá trị khỏi ngăn xếp và đặt chúng vào một thanh ghi được chỉ định.

Lệnh đẩy sử dụng cú pháp sau để thực hiện các thao tác của nó:

đẩy 1

Trong ví dụ này, chương trình trả con trỏ ngăn xếp (thanh ghi ESP) đến một địa chỉ bộ nhớ mới, do đó nhường chỗ cho giá trị (1), giá trị này hiện được lưu trữ ở vị trí trên cùng của ngăn xếp. Sau đó, nó sao chép giá trị từ đối số vào vị trí bộ nhớ mà CPU vừa tạo chỗ cho phần trên cùng của ngăn xếp.

Hãy đổi chiều điều này với pop:

pop eax

Chương trình sử dụng cửa sổ bật lên để lấy giá trị trên cùng ra khỏi ngăn xếp và chuyển nó vào một thanh ghi đã chỉ định. Trong ví dụ này, pop eax bật giá trị cao nhất ra khỏi ngăn xếp và di chuyển nó vào eax.

Một chi tiết không trực quan nhưng quan trọng cần hiểu về ngăn xếp x86 là nó phát triển xuống trong bộ nhớ, sao cho giá trị cao nhất trên ngăn xếp thực sự được lưu trữ tại địa chỉ thấp nhất trong bộ nhớ ngăn xếp. Cái này

trở nên rất quan trọng cần nhớ khi bạn phân tích mã hợp ngữ tham chiếu dữ liệu được lưu trữ trên ngăn xếp, vì nó có thể nhanh chóng gây nhầm lẫn trừ khi bạn biết cách bố trí bộ nhớ của ngăn xếp.

Bởi vì ngăn xếp x86 phát triển xuống trong bộ nhớ, khi lệnh đẩy phân bổ không gian trên ngăn xếp chương trình cho một giá trị mới, nó sẽ giảm giá trị của ESP để nó trở đến một vị trí thấp hơn trong bộ nhớ và sau đó sao chép một giá trị từ thanh ghi đích vào vị trí bộ nhớ đó, bắt đầu từ địa chỉ trên cùng của ngăn xếp và lớn dần lên. Ngược lại, lệnh pop thực sự sao chép giá trị trên cùng của ngăn xếp và sau đó tăng giá trị của ESP để nó trở đến một vị trí bộ nhớ cao hơn.

Hướng dẫn luồng điều khiển

Luồng điều khiển của chương trình x86 xác định mạng các trình tự thực thi lệnh có thể có mà một chương trình có thể thực thi, tùy thuộc vào dữ liệu, tương tác của thiết bị và các đầu vào khác mà chương trình có thể nhận được. Hướng dẫn luồng điều khiển xác định luồng điều khiển của chương trình. Chúng phức tạp hơn các hướng dẫn ngăn xếp nhưng vẫn khá trực quan. Bởi vì luồng điều khiển thường được thể hiện thông qua các lệnh gọi hàm kiểu C trong hợp ngữ x86, ngăn xếp và luồng điều khiển có liên quan chặt chẽ với nhau. Chúng cũng có liên quan với nhau vì các lệnh gọi hàm này sử dụng ngăn xếp để truyền đối số, phân bổ các biến cục bộ và ghi nhớ phần nào của chương trình sẽ quay lại sau khi một hàm kết thúc thực thi.

Các hướng dẫn luồng điều khiển gọi và rút lại là quan trọng nhất về cách các chương trình gọi các hàm trong hợp ngữ x86 và cách các chương trình quay trở lại từ các hàm sau khi các hàm này được thực thi xong.

Lệnh gọi gọi một hàm. Hãy coi đây là một hàm bạn có thể viết bằng ngôn ngữ cấp cao hơn như C để cho phép chương trình quay lại lệnh sau khi lệnh gọi được gọi và hàm đã thực thi xong. Bạn có thể gọi lệnh gọi bằng cú pháp sau, trong đó địa chỉ biểu thị vị trí bộ nhớ nơi bắt đầu mã của hàm:

địa chỉ cuộc gọi

Lệnh gọi thực hiện hai việc. Đầu tiên, nó đẩy địa chỉ của lệnh sẽ thực thi sau khi lệnh gọi hàm quay trở lại trên cùng của ngăn xếp để chương trình biết địa chỉ nào sẽ quay trở lại sau khi hàm được gọi kết thúc thực thi. Thứ hai, cuộc gọi thay thế giá trị hiện tại của EIP bằng giá trị được chỉ định bởi toán hạng địa chỉ. Sau đó, CPU bắt đầu thực hiện tại vị trí bộ nhớ mới được chỉ ra bởi EIP.

Giống như lời gọi bắt đầu một lời gọi hàm, lệnh ret hoàn thành nó. Bạn có thể sử dụng lệnh ret của riêng nó và không có bất kỳ tham số nào, như được hiển thị ở đây:

rút lui

Khi được gọi, ret bật giá trị trên cùng ra khỏi ngăn xếp, mà chúng tôi mong đợi là giá trị bộ đếm chương trình đã lưu (EIP) mà lệnh gọi đã đẩy vào ngăn xếp khi lệnh gọi được gọi. Sau đó, nó đặt giá trị bộ đếm chương trình đã bật trở lại vào EIP và tiếp tục thực thi.

Lệnh jmp là một cấu trúc luồng điều khiển quan trọng khác, hoạt động đơn giản hơn lệnh gọi. Thay vì lo lắng về việc lưu EIP, jmp chỉ cần yêu cầu CPU di chuyển đến địa chỉ bộ nhớ được chỉ định làm tham số của nó và bắt đầu thực hiện ở đó. Ví dụ: jmp 0x12345678 yêu cầu CPU bắt đầu thực thi mã chương trình được lưu trữ tại vị trí bộ nhớ 0x12345678 trong lệnh tiếp theo.

Bạn có thể tự hỏi làm thế nào bạn có thể làm cho các lệnh gọi và jmp thực thi theo cách có điều kiện, chẳng hạn như “nếu chương trình đã nhận được gói công việc mạng, hãy thực hiện chức năng sau.” Câu trả lời là hợp ngữ x86 không có các cấu trúc cấp cao như if, then, other, other if, v.v. Thay vào đó, việc phân nhánh tới một địa chỉ trong mã của chương trình thường yêu cầu hai lệnh: lệnh cmp , kiểm tra giá trị trong một số thanh ghi so với một số giá trị kiểm tra và lưu trữ kết quả của phép kiểm tra đó trong thanh ghi EFLAGS và lệnh rẽ nhánh có điều kiện.

Hầu hết các hướng dẫn rẽ nhánh có điều kiện đều bắt đầu bằng chữ j, cho phép chương trình chuyển đến địa chỉ bộ nhớ và được cố định sau bằng các chữ cái đại diện cho điều kiện đang được kiểm tra. Ví dụ jge báo cho chương trình hãy nếu lớn hơn hoặc bằng. Điều này có nghĩa là giá trị trong thanh ghi đang được kiểm tra phải lớn hơn hoặc bằng giá trị kiểm tra.

Hướng dẫn cmp sử dụng cú pháp sau:

thanh ghi cmp , vị trí bộ nhớ hoặc chữ, thanh ghi, vị trí bộ nhớ hoặc chữ

Như đã nêu trước đó, cmp so sánh giá trị trong mục đích chung được chỉ định đăng ký với giá trị và sau đó lưu trữ kết quả so sánh đó trong thanh ghi EFLAGS.

Các hướng dẫn jmp có điều kiện khác nhau sau đó được gọi như sau:

địa chỉ j*

Như bạn có thể thấy, chúng ta có thể thêm tiền tố j vào bất kỳ số lượng hướng dẫn kiểm tra có điều kiện nào. Ví dụ: để chỉ rằng nếu giá trị được kiểm tra lớn hơn hoặc bằng giá trị trong thanh ghi, hãy sử dụng lệnh sau:

địa chỉ jge

Lưu ý rằng không giống như trường hợp lệnh call và ret , họ lệnh jmp không bao giờ chạm vào ngăn xếp chương trình. Trên thực tế, trong trường hợp họ lệnh jmp , chương trình x86 chịu trách nhiệm theo dõi luồng thực thi của chính nó và có khả năng lưu hoặc xóa thông tin về những địa chỉ mà nó đã truy cập và nơi nó sẽ quay lại sau khi thực hiện một chuỗi lệnh cụ thể .

Các khối cơ bản và đồ thị luồng điều khiển

Mặc dù các chương trình x86 trông có vẻ tuần tự khi chúng ta cuộn qua mã của chúng trong trình soạn thảo văn bản, nhưng chúng thực sự có các vòng lặp, nhánh có điều kiện và nhánh không có điều kiện (luồng điều khiển). Tất cả những thứ này cung cấp cho mỗi chương trình x86 một cấu trúc làm việc mạng. Hãy sử dụng chương trình lấp ráp đồ chơi đơn giản trong *Liệt kê 2-1* để xem chương trình này hoạt động như thế nào.

thiết lập: ký hiệu # thay thế cho địa chỉ hướng dẫn trên dòng tiếp theo

`mov eax, 10`

loopstart: ký hiệu # đại diện cho địa chỉ của lệnh trên dòng tiếp theo

`phu eax, 1`

`cmp 0, eax`

`jne $loopstart`

loopend: ký hiệu # đại diện cho địa chỉ của lệnh trên dòng tiếp theo

`di chuyển eax, 1`

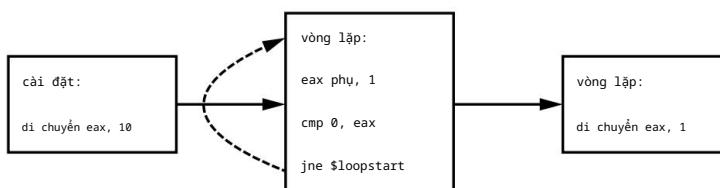
`# mã khác sẽ đến đây`

Liệt kê 2-1: Chương trình hợp ngữ để hiểu biểu đồ luồng điều khiển

Như bạn có thể thấy, chương trình này khởi tạo một bộ đếm với giá trị 10, được lưu trữ trong thanh ghi EAX. Tiếp theo, nó thực hiện một vòng lặp trong đó giá trị trong EAX giảm đi 1 trên mỗi lần lặp. Cuối cùng, khi EAX đạt đến giá trị 0, chương trình sẽ thoát khỏi vòng lặp.

Trong ngôn ngữ phân tích biểu đồ luồng điều khiển, chúng ta có thể coi các lệnh này bao gồm ba khối cơ bản. Một khối cơ bản là một chuỗi các hướng dẫn mà chúng tôi biết sẽ luôn thực thi liên tục. Nói cách khác, một khối cơ bản luôn kết thúc bằng lệnh `re` nhánh hoặc lệnh là đích của nhánh và nó luôn bắt đầu bằng lệnh đầu tiên của chương trình, được gọi là điểm vào của chương trình hoặc đích của nhánh.

Trong *Liệt kê 2-1*, bạn có thể thấy các khối cơ bản của pro đơn giản của chúng ta ở đâu. gram bắt đầu và kết thúc. Khối cơ bản đầu tiên bao gồm lệnh `mov eax, 10` theo thiết lập:. Khối cơ bản thứ hai bao gồm các dòng bắt đầu bằng `sub eax, 1` đến `jne $loopstart` dưới `loopstart:`, và khối thứ ba bắt đầu tại `mov eax, 1` dưới `loopend:`. Chúng ta có thể hình dung mối quan hệ giữa các khối cơ bản bằng biểu đồ trong Hình 2-2. (Chúng tôi sử dụng thuật ngữ đồ thị đồng nghĩa với thuật ngữ mạng; trong khoa học máy tính, các thuật ngữ này có thể hoán đổi cho nhau.)



Hình 2-2: Trực quan hóa đồ thị luồng điều khiển của chương trình hợp ngữ đơn giản của chúng tôi

Nếu một khôi cơ bản có thể chảy vào một khôi cơ bản khác, chúng ta sẽ kết nối nó, như trong Hình 2-2. Hình này cho thấy khôi cơ bản thiết lập dẫn đến khôi cơ bản khởi động vòng lặp, khôi này lặp lại 10 lần trước khi chuyển sang khôi cơ bản kết thúc vòng lặp. Các chương trình trong thế giới thực có các đồ thị luồng điều khiển như thế này, nhưng chúng phức tạp hơn nhiều, với hàng nghìn khôi cơ bản và hàng nghìn kết nối.

Tháo gỡ ircbot.exe bằng pefile và capstone

Bây giờ bạn đã hiểu rõ về kiến thức cơ bản của hợp ngữ, hãy phân tách 100 byte đầu tiên của mã hợp ngữ ircbot.exe bằng cách sử dụng phân tách tuyến tính. Để làm điều này, chúng tôi sẽ sử dụng thư viện mã nguồn mở Python pefile (được giới thiệu trong Chương 1) và capstone, là một thư viện mã nguồn mở có thể phân tách mã nhị phân x86 32-bit. Bạn có thể cài đặt cả hai thư viện này bằng pip bằng các lệnh sau:

```
pip cài đặt pefile
pip cài đặt capstone
```

Sau khi hai thư viện này được cài đặt, chúng ta có thể tận dụng chúng để loại bỏ ircbot.exe semble bằng cách sử dụng mã trong Liệt kê 2-2.

```
#!/usr/bin/trần
nhập khẩu pefile
từ capstone nhập khẩu *

# tải tệp PE mục tiêu
pe = pefile.PE("ircbot.exe")

# lấy địa chỉ của điểm vào chương trình từ tiêu đề chương trình
điểm vào = pe.OPTIONAL_HEADER.AddressOfEntryPoint

# tính toán địa chỉ bộ nhớ nơi mã nhập sẽ được tải vào bộ nhớ
entrypoint_address = entrypoint+pe.OPTIONAL_HEADER.ImageBase

# lấy mã nhị phân từ đối tượng tệp PE
binary_code = pe.get_memory_mapped_image()[entrypoint:entrypoint+100]

# khởi tạo trình phân tách để phân tách mã nhị phân 32 bit x86
trình dịch ngược = Cs(CS_ARCH_X86, CS_MODE_32)

# tháo mã
để được hướng dẫn trong trình dịch ngược mã.disasm(binary_code, entrypoint_address):
    in "%s\t%s" %(instruction.mnemonic, direction.op_str)
```

Liệt kê 2-2: Tháo rời ircbot.exe

Điều này sẽ tạo ra đầu ra sau:

dày	ebp
di chuyển	ebp, đặc biệt

```

dãy      -1
dãy      0x437588
dãy      0x41982c
mov      eax, dword ptr fs:[0]
dãy      trục
mov      dword ptr fs:[0], đặc biệt
thêm    đặc biệt, -0x5c
dãy      ebx
dãy      esi
dãy      chính sửa
di chuyển dword ptr [ebp - 0x18], đặc biệt
gọi    dword ptr [0x496308]
--snip--

```

Đừng lo lắng về việc hiểu tất cả các hướng dẫn trong đầu ra của dis assembly: điều đó sẽ liên quan đến sự hiểu biết về assembly vượt ra ngoài phạm vi của cuốn sách này. Tuy nhiên, bạn sẽ cảm thấy thoải mái với nhiều hướng dẫn trong đầu ra và hiểu phần nào về những gì chúng làm. Ví dụ: phần mềm độc hại đẩy giá trị trong thanh ghi EBP lên ngăn xếp , tiết kiệm giá trị của nó. Sau đó, nó tiến hành di chuyển giá trị trong ESP vào EBP và đẩy một số giá trị số vào ngăn xếp. Chương trình di chuyển một số dữ liệu trong bộ nhớ vào thanh ghi EAX , và nó thêm giá trị -0x5c vào giá trị trong thanh ghi ESP . Cuối cùng, chương trình sử dụng lời gọi

hướng dẫn gọi một chức năng được lưu trữ tại địa chỉ bộ nhớ 0x496308 .

Bởi vì đây không phải là một cuốn sách về kỹ thuật đảo ngược, tôi sẽ không đi sâu hơn ở đây về ý nghĩa của mã. Những gì tôi vừa trình bày là bước khởi đầu để hiểu cách hoạt động của hợp ngữ. Để biết thêm thông tin về ngôn ngữ lắp ráp, tôi giới thiệu hướng dẫn lập trình viên của Intel tại <http://www.intel.com/nội dung/www/us/en/processors/architectures-software-developer-manuals.html>.

Các yếu tố hạn chế phân tích tĩnh

Trong chương này và Chương 1, bạn đã học về nhiều cách khác nhau trong đó các kỹ thuật phân tích tĩnh có thể được sử dụng để làm sáng tỏ mục đích và phương pháp của một tệp nhị phân độc hại mới được phát hiện. Thật không may, phân tích tĩnh có những hạn chế khiến nó kém hữu ích hơn trong một số trường hợp. Ví dụ: tác giả phần mềm độc hại có thể sử dụng một số chiến thuật tấn công dễ thực hiện hơn nhiều so với việc chống lại. Chúng ta hãy xem xét một số chiến thuật tấn công này và xem cách phòng thủ chống lại chúng.

đóng gói

Đóng gói phần mềm độc hại là quá trình mà tác giả phần mềm độc hại nén, mã hóa hoặc nói cách khác là xử lý phần lớn chương trình độc hại của họ để các nhà phân tích phần mềm độc hại có vẻ khó hiểu. Khi phần mềm độc hại được chạy, nó sẽ tự giải nén và sau đó bắt đầu thực thi. Cách rõ ràng xung quanh việc đóng gói phần mềm độc hại là thực sự chạy phần mềm độc hại trong một môi trường an toàn, một kỹ thuật phân tích động mà tôi sẽ trình bày trong Chương 3.

Lưu ý Đón gói phần mềm cũng được sử dụng bởi những người cài đặt phần mềm lành tính vì những lý do chính đáng.

Các tác giả phần mềm lành tính sử dụng tính năng đóng gói để phân phối mã của họ vì nó cho phép họ nén tài nguyên chương trình để giảm kích thước tải xuống của trình cài đặt phần mềm. Nó cũng giúp họ ngăn chặn các nỗ lực kỹ thuật đảo ngược của các đối thủ cạnh tranh kinh doanh và nó cung cấp một cách thuận tiện để gói nhiều tài nguyên chương trình trong một tệp trình cài đặt.

Làm xáo trộn tài nguyên

Một kỹ thuật chống phân tích, chống phát hiện khác mà các tác giả phần mềm độc hại sử dụng là làm xáo trộn tài nguyên. Chúng làm xáo trộn cách các tài nguyên chương trình, chẳng hạn như chuỗi và hình ảnh đồ họa, được lưu trữ trên đĩa, sau đó giải mã chúng trong thời gian chạy để chương trình độc hại có thể sử dụng chúng. Ví dụ: một thao tác che giấu đơn giản sẽ là thêm giá trị 1 vào tất cả các byte trong hình ảnh và chuỗi được lưu trữ trong phần tài nguyên PE, sau đó trừ 1 khỏi tất cả dữ liệu này khi chạy. Tuy nhiên, bất kỳ số lượng che giấu nào cũng có thể xảy ra ở đây, tất cả đều có thể gây khó khăn cho các nhà phân tích phần mềm độc hại đang cố gắng hiểu ý nghĩa của nhị phân phần mềm độc hại bằng cách sử dụng phân tích tĩnh.

Đối với việc đóng gói, một cách để giải quyết vấn đề che giấu tài nguyên là chỉ chạy phần mềm độc hại trong một môi trường an toàn. Khi đây không phải là một tùy chọn, biện pháp giảm thiểu duy nhất đối với việc làm xáo trộn tài nguyên là thực sự tìm ra cách mà phần mềm độc hại đã làm xáo trộn tài nguyên của nó và giải mã chúng theo cách thủ công, đây là điều mà các nhà phân tích phần mềm độc hại chuyên nghiệp thường làm.

Kỹ thuật chống tháo gỡ

Nhóm kỹ thuật chống phân tích, chống phát hiện thứ ba được tác giả phần mềm độc hại sử dụng là các kỹ thuật chống tháo gỡ. Các kỹ thuật này được thiết kế để khai thác những hạn chế vốn có của các kỹ thuật tháo gỡ hiện đại nhằm che giấu mã khỏi các nhà phân tích phần mềm độc hại hoặc khiến các nhà phân tích phần mềm độc hại nghĩ rằng một khối mã được lưu trữ trên đĩa chứa các hướng dẫn khác với thực tế.

Một ví dụ về kỹ thuật chống tháo gỡ liên quan đến việc phân nhánh đến một vị trí bộ nhớ mà trình phân tách của tác giả phần mềm độc hại sẽ hiểu là một hướng dẫn khác, về cơ bản là che giấu hướng dẫn thực sự của phần mềm độc hại khỏi các kỹ sư đảo ngược. Các kỹ thuật chống tháo gỡ có tiềm năng rất lớn và không có cách nào hoàn hảo để chống lại chúng. Trên thực tế, hai biện pháp bảo vệ chính chống lại các kỹ thuật này là chạy các mẫu phần mềm độc hại trong môi trường động và tìm ra vị trí các chiến lược chống tháo gỡ hiển thị trong một mẫu phần mềm độc hại theo cách thủ công và cách vượt qua chúng.

Dữ liệu được tải xuống động

Lớp kỹ thuật chống phân tích cuối cùng mà tác giả phần mềm độc hại sử dụng liên quan đến việc tìm nguồn dữ liệu và mã nguồn bên ngoài. Ví dụ: mẫu phần mềm độc hại có thể tải mã động từ máy chủ bên ngoài vào thời điểm khởi động phần mềm độc hại. Nếu đây là trường hợp, phân tích tĩnh sẽ vô dụng đối với mã như vậy. Tương tự, phần mềm độc hại có thể lấy các khóa giải mã từ các máy chủ bên ngoài khi khởi động và sau đó sử dụng các khóa này để giải mã dữ liệu hoặc mã sẽ được sử dụng trong quá trình thực thi của phần mềm độc hại.

Rõ ràng, nếu phần mềm độc hại đang sử dụng thuật toán mã hóa cường độ công nghiệp, phân tích tĩnh sẽ không đủ để khôi phục dữ liệu và mã được mã hóa. Các kỹ thuật chống phân tích và chống phát hiện như vậy khá mạnh mẽ và cách duy nhất xung quanh chúng là lấy mã, dữ liệu hoặc khóa riêng trên các máy chủ bên ngoài bằng một số phương tiện và sau đó sử dụng chúng trong phân tích phần mềm độc hại được đề cập. .

Bản tóm tắt

Chương này đã giới thiệu phân tích mã hợp ngữ x86 và trình bày cách chúng ta có thể thực hiện phân tích tĩnh dựa trên tháo gỡ trên ircbot.exe bằng các công cụ Python mã nguồn mở. Mặc dù đây không phải là tài liệu hướng dẫn hoàn chỉnh về tổ hợp x86, nhưng bây giờ bạn sẽ cảm thấy đủ thoải mái để bắt đầu tìm hiểu điều gì đang xảy ra trong một bản kết xuất tập hợp phần mềm độc hại nhất định. Cuối cùng, bạn đã học được những cách mà tác giả phần mềm độc hại có thể bảo vệ chống lại các kỹ thuật phân tích tĩnh và tháo gỡ cũng như cách bạn có thể giảm thiểu các chiến lược chống phân tích và chống phát hiện này. Trong Chương 3, bạn sẽ học cách tiến hành phân tích phần mềm độc hại động để khắc phục nhiều điểm yếu của phân tích phần mềm độc hại tĩnh.

3

Tóm tắt tôi giới thiệu

Dyna m ic A na ly giải



Trong Chương 2, bạn đã học các kỹ thuật phân tích tinh nâng cao để tháo rời mã lắp ráp được khôi phục từ phần mềm độc hại.

Mặc dù phân tích tinh có thể là một cách hiệu quả để thu được thông tin hữu ích về phần mềm độc hại bằng cách nghiên cứu các thành phần khác nhau của nó trên đĩa, nhưng nó không cho phép chúng tôi quan sát hành vi của phần mềm độc hại.

Trong chương này, bạn sẽ tìm hiểu về những kiến thức cơ bản về phân tích phần mềm độc hại động. Không giống như phân tích tĩnh, vốn tập trung vào phần mềm độc hại trong một môi trường an toàn, có chứa để xem phần mềm đó hoạt động như thế nào. Điều này giống như việc đưa một chúng vi khuẩn nguy hiểm vào một môi trường kín để xem ảnh hưởng của nó đối với các tế bào khác.

Bằng cách sử dụng phân tích động, chúng tôi có thể vượt qua các rào cản phân tích tĩnh phổ biến, chẳng hạn như đóng gói và che giấu, cũng như hiểu rõ hơn về mục đích của một mẫu phần mềm độc hại nhất định. Chúng tôi bắt đầu bằng cách khám phá các kỹ thuật phân tích động cơ bản, mức độ liên quan của chúng với khoa học dữ liệu phần mềm độc hại và các ứng dụng của chúng. Chúng tôi sử dụng các công cụ nguồn mở như malwr.com để nghiên cứu các ví dụ về phân tích động trong thực tế. Lưu ý rằng đây là một cõi động

khảo sát về chủ đề và không nhằm mục đích toàn diện. Để có phần giới thiệu đầy đủ hơn, hãy xem Phân tích phần mềm độc hại thực tế (No Starch Press, 2012).

Tại sao nên sử dụng Phân tích động?

Để hiểu tại sao phân tích động lại quan trọng, hãy xem xét vấn đề về phần mềm độc hại được đóng gói. Hãy nhớ rằng việc đóng gói phần mềm độc hại để cài đặt việc nén hoặc làm xáo trộn mã hợp ngữ x86 của phần mềm độc hại để che giấu bản chất độc hại của chương trình. Một mẫu phần mềm độc hại được đóng gói sẽ tự giải nén khi lây nhiễm vào máy tar get để mã có thể thực thi.

Chúng ta có thể cố gắng phân tách một mẫu phần mềm độc hại được đóng gói hoặc làm rối bằng cách sử dụng các công cụ phân tích tĩnh được thảo luận trong Chương 2, nhưng đây là một quá trình tốn nhiều công sức. Ví dụ: với phân tích tĩnh, trước tiên chúng tôi phải tìm vị trí của mã bị xáo trộn trong tệp phần mềm độc hại. Sau đó, chúng ta phải tìm vị trí của các chương trình con gỡ rối mã nguồn để giải mã mã nguồn này để nó có thể chạy. Sau khi định vị các chương trình con, chúng ta phải tìm ra cách thức hoạt động của quy trình giải mã mã nguồn này để thực hiện nó trên mã. Chỉ khi đó, chúng tôi mới có thể bắt đầu quá trình thực sự của kỹ thuật đảo ngược mã độc hại.

Một giải pháp thay thế đơn giản nhưng thông minh cho quy trình này là thực thi phần mềm độc hại trong một môi trường an toàn, khép kín được gọi là sandbox. Chạy phần mềm độc hại trong hộp cáp cho phép phần mềm này tự giải nén giống như khi lây nhiễm một mục tiêu thực sự. Chỉ cần chạy phần mềm độc hại, chúng tôi có thể tìm ra máy chủ nào mà phần mềm độc hại nhị phân cụ thể kết nối với, thông số cấu hình hệ thống mà nó thay đổi và thiết bị I/O (đầu vào/đầu ra) mà nó có gắng thực hiện.

Phân tích động cho khoa học dữ liệu phần mềm độc hại

Phân tích động không chỉ hữu ích cho kỹ thuật đảo ngược phần mềm độc hại mà còn cho khoa học dữ liệu phần mềm độc hại. Do phân tích động cho biết hoạt động của một mẫu phần mềm độc hại nên chúng tôi có thể so sánh hành động của nó với hành động của các mẫu phần mềm độc hại khác. Ví dụ: vì phân tích động hiển thị những tệp mẫu phần mềm độc hại ghi vào đĩa, chúng tôi có thể sử dụng dữ liệu này để kết nối những mẫu phần mềm độc hại ghi tên tệp tương tự vào đĩa. Những loại mạnh mẽ này giúp chúng tôi phân loại các mẫu phần mềm độc hại dựa trên các đặc điểm chung. Chúng thậm chí có thể giúp chúng tôi xác định các mẫu phần mềm độc hại được tạo bởi cùng một nhóm hoặc là một phần của cùng một chiến dịch.

Quan trọng nhất, phân tích động rất hữu ích để xây dựng trình phát hiện phần mềm độc hại dựa trên máy học. Chúng ta có thể đào tạo một máy dò để phân biệt giữa các nhì phần độc hại và lành tính bằng cách quan sát hành vi của chúng trong quá trình phân tích động. Ví dụ: sau khi quan sát hàng nghìn nhật ký phân tích động từ cả tệp phần mềm độc hại và tệp lành tính, hệ thống máy học có thể biết rằng khi msword.exe khởi chạy quy trình có tên powershell.exe, hành động này là độc hại, nhưng khi msword.exe khởi chạy Internet Explorer, điều này có lẽ là vô hại. Chương 8 sẽ đi vào chi tiết hơn về cách chúng ta

có thể xây dựng trình phát hiện phần mềm độc hại bằng cách sử dụng dữ liệu dựa trên cả phân tích tĩnh và động. Nhưng trước khi tạo các công cụ phát hiện phần mềm độc hại tĩnh vi, hãy xem xét một số công cụ cơ bản để phân tích động.

Công cụ cơ bản để phân tích động

Bạn có thể tìm thấy một số công cụ mã nguồn mở, miễn phí để phân tích động trực tuyến. Phần này tập trung vào malwr.com và CuckooBox. Trang web malwr.com có giao diện web cho phép bạn gửi các tệp nhị phân để phân tích động miễn phí.

CuckooBox là một nền tảng phần mềm cho phép bạn thiết lập môi trường ysis hậu môn năng động của riêng mình để bạn có thể phân tích cục bộ các tệp nhị phân. Những người tạo ra nền tảng CuckooBox cũng điều hành malwr.com và malwr.com điều hành CuckooBox ở hậu trường. Do đó, học cách phân tích kết quả trên malwr.com sẽ cho phép bạn hiểu kết quả của CuckooBox.

LƯU Ý thời điểm in, giao diện CuckooBox của malwr.com đã ngừng hoạt động để bảo trì. Hy vọng rằng khi bạn đọc phần này, trang web sẽ hoạt động trở lại. Nếu không, thông tin được cung cấp trong chương này có thể được áp dụng cho đầu ra từ phiên bản CuckooBox của riêng bạn, bạn có thể thiết lập phiên bản này bằng cách làm theo hướng dẫn tại <https://cuckoosandbox.org/>.

Hành vi phần mềm độc hại điển hình

Sau đây là các loại hành động chính mà một mẫu phần mềm độc hại có thể thực hiện khi thực thi:

Sửa đổi hệ thống tệp Ví dụ: ghi trình điều khiển thiết bị vào đĩa, thay đổi tệp cấu hình hệ thống, thêm chương trình mới vào hệ thống tệp và sửa đổi khóa đăng ký để đảm bảo chương trình tự khởi động

Sửa đổi sổ đăng ký Windows để thay đổi cấu hình hệ thống Ví dụ: thay đổi cài đặt tường lửa

Đang tải trình điều khiển thiết bị Ví dụ: tải trình điều khiển thiết bị ghi lại thao tác gõ phím của người dùng

Các hành động mạng Ví dụ: phân giải tên miền và thực hiện yêu cầu HTTP

Chúng tôi sẽ kiểm tra các hành vi này chi tiết hơn bằng cách sử dụng một mẫu phần mềm độc hại và phân tích báo cáo của nó trên malwr.com.

Đang tải tệp trên malwr.com

Để chạy mẫu phần mềm độc hại thông qua malwr.com, hãy điều hướng đến <https://malwr.com/> và sau đó nhấp vào nút Gửi để tải lên và gửi tệp nhị phân để phân tích. Chúng ta sẽ sử dụng một mã nhị phân có hàm băm SHA256 bắt đầu bằng các ký tự d676d95, mà bạn có thể tìm thấy trong thư mục dữ liệu đi kèm với chương này. Tôi khuyến khích bạn gửi tệp nhị phân này tới malwr.com và tự mình kiểm tra kết quả khi chúng tôi thực hiện. Trang gửi được hiển thị trong Hình 3-1.

Hình 3-1: Trang gửi mẫu phần mềm độc hại

Sau khi bạn gửi mẫu của mình qua biểu mẫu này, trang web sẽ nhắc bạn đợi quá trình phân tích hoàn tất, thường mất khoảng năm phút. Khi tải kết quả, bạn có thể kiểm tra chúng để hiểu tệp thực thi đã làm gì khi chạy trong môi trường phân tích động.

Phân tích kết quả trên malwr.com

Trang kết quả cho mẫu của chúng ta sẽ giống như Hình 3-2.

CATEGORY	STARTED	COMPLETED
FILE	2016-12-30 11:56:05	2016-12-30 11:58:22

FILE	NAME
FILE	wordplugin.exe

FILE	SIZE
FILE	410112 bytes

FILE	TYPE
FILE	PE32 executable (GUI) intel 80386, for MS Windows, UPX compressed

MD5
9559709bd252e80dc957de683ce1743

SHA1
c8618b0abe866f3c018a950910f006a1bea2a920

SHA256
d676d9dfab6a4242258362b8ff579cf6e5e6db3f0cd3e0069ace50f80af1c5

SHA512
bdaca27f3a5b76baa5928940f4dfc81d641540050b26f02cf1ea7d94bdbfcf3630c54f2d0a54

Hình 3-2: Đầu trang kết quả cho mẫu phần mềm độc hại trên malwr.com

Các kết quả cho tệp này minh họa một số khía cạnh chính của phân tích động, mà chúng ta sẽ khám phá tiếp theo.

Bảng chữ ký

Hai bảng đầu tiên bạn sẽ thấy trên trang kết quả là Phân tích và Chi tiết tệp. Chúng chứa thời gian tệp được chạy và các chi tiết tĩnh khác về tệp. Bảng mà tôi sẽ tập trung vào ở đây là bảng Chữ ký, được hiển thị trong Hình 3-3. Bảng điều khiển này chứa thông tin cấp cao bắt nguồn từ chính tệp và hành vi của tệp khi tệp được chạy trong môi trường phân tích động. Hãy thảo luận ý nghĩa của từng chữ ký này.

Signatures

- File has been identified by at least one AntiVirus on VirusTotal as malicious
- The binary likely contains encrypted or compressed data.
- The executable is compressed using UPX
- Collects information to fingerprint the system (MachineGuid, DigitalProductId, SystemBiosDate)
- Creates an Alternate Data Stream (ADS)
- Installs itself for autorun at Windows startup

Hình 3-3: Chữ ký malwr.com phù hợp với hành vi của mẫu phần mềm độc hại của chúng tôi

Ba chữ ký đầu tiên hiển thị trong hình là kết quả của phân tích tĩnh (nghĩa là đây là kết quả từ các thuộc tính của chính tệp phần mềm độc hại, không phải hành động của nó). Chữ ký đầu tiên chỉ đơn giản cho chúng tôi biết rằng một số công cụ chống vi-rút trên trình tổng hợp chống vi-rút phổ biến VirusTotal.com đã đánh dấu tệp này là phần mềm độc hại. Thứ hai chỉ ra rằng tệp nhị phân chứa dữ liệu được nén hoặc mã hóa, một dấu hiệu phổ biến của sự xáo trộn. Phần thứ ba cho chúng ta biết rằng tệp nhị phân này đã được nén bằng trình đóng gói UPX phổ biến. Mặc dù bản thân các chỉ báo tĩnh này không cho chúng tôi biết tệp này làm gì, nhưng chúng cho chúng tôi biết rằng tệp có khả năng độc hại. (Lưu ý rằng màu sắc không tương ứng với các danh mục tĩnh và động; thay vào đó, nó thể hiện mức độ nghiêm trọng của từng quy tắc, với màu đỏ-màu xám đậm hơn ở đây-đáng ngờ hơn màu vàng.)

Ba chữ ký tiếp theo là kết quả phân tích động của tệp. Chữ ký đầu tiên chỉ ra rằng chương trình có gắng xác định phần cứng và hệ điều hành của hệ thống. Thứ hai chỉ ra rằng chương trình sử dụng một tính năng nguy hiểm của Windows được gọi là Luồng dữ liệu thay thế (ADS), cho phép phần mềm độc hại ẩn dữ liệu trên đĩa để nó không nhìn thấy được khi sử dụng các công cụ duyệt hệ thống tệp tiêu chuẩn. Chữ ký thứ ba chỉ ra rằng tệp thay đổi số đăng ký Windows để khi hệ thống khởi động lại, chương trình mà nó chỉ định sẽ tự động thực thi. Điều này sẽ khởi động lại phần mềm độc hại bất cứ khi nào người dùng khởi động lại hệ thống của họ.

Như bạn có thể thấy, ngay cả ở cấp độ của các bản chất sig được kích hoạt tự động này, phân tích động bổ sung đáng kể vào kiến thức của chúng ta về hành vi dự định của tệp.

Bảng điều khiển ánh chụp màn hình

Bên dưới bảng Chữ ký là bảng Ánh chụp màn hình. Bảng này hiển thị ánh chụp màn hình của màn hình môi trường phân tích động khi phần mềm độc hại đang chạy. Hình 3-4 cho thấy một ví dụ về điều này trông như thế nào.



Hình 3-4: Ánh chụp màn hình hành vi động của mẫu phần mềm độc hại của chúng tôi

Bạn có thể thấy rằng phần mềm độc hại mà chúng tôi đang xử lý là ransomware, đây là loại phần mềm độc hại mã hóa các tệp của mục tiêu và buộc họ phải trả tiền nếu muốn lấy lại dữ liệu của mình. Chỉ cần chạy phần mềm độc hại của chúng tôi, chúng tôi đã có thể phát hiện ra mục đích của nó mà không cần dùng đến kỹ thuật đảo ngược.

Bảng đối tượng hệ thống đã sửa đổi

Một hàng tiêu đề bên dưới Ánh chụp màn hình hiển thị hoạt động mạng của mẫu phần mềm độc hại. Hệ nhị phân của chúng tôi không tham gia vào bất kỳ giao tiếp mạng nào, nhưng nếu có, chúng tôi sẽ thấy các máy chủ mà nó đã liên hệ tại đây. Hình 3-5 hiển thị bảng Tóm tắt.

Summary

Files Registry Keys Mutexes

```
C:\DOCUME~1\User\LOCALS~1\Temp\wordplugin.exe
C:\DOCUME~1
C:\DOCUME~1\User
C:\DOCUME~1\User\LOCALS~1
C:\DOCUME~1\User\LOCALS~1\Temp
C:\Documents and Settings\User\Local Settings\Temp\wordplugin.exe
C:\WINDOWS\system32\msctftime.ime
```

Hình 3-5: Tab Tệp của ngăn Tóm tắt, hiển thị những tệp mà mẫu phần mềm độc hại của chúng tôi đã sửa đổi

Điều này cho biết các đối tượng hệ thống nào, như tệp, khóa đăng ký và bộ chuyển đổi, phần mềm độc hại này thực sự đã mã hóa các tệp người dùng trên đĩa.

Nhìn vào tab Tệp trong Hình 3-6, rõ ràng là phần mềm độc hại ransomware này thực sự đã mã hóa các tệp người dùng trên đĩa.

```
C:\Perl\win32\cpan.ico
C:\Perl\win32\03BAFC2EA6A713B05444C65E75689DA2.locked
C:\Perl\win32\onion.ico
C:\Perl\win32\03CDFF5FA6D613B12F44BD5F75199DD1FAB3.locked
C:\Perl\win32\perldoc.ico
C:\Perl\win32\00B9FF54A1DD13B22F31C65C756899A5FACECDED14AE.locked
C:\Perl\win32\perlhelp.ico
C:\Perl\win32\0D0B9FF54A1DD13B22F46C62D751E9ED2FEB2CD9C14DB4F25.locked
```

Hình 3-6: Đường dẫn tệp trong tab Tệp của ngăn Tóm tắt, gợi ý rằng mẫu của chúng tôi là mã độc tống tiền

Sau mỗi đường dẫn tệp là một tệp có phần mở rộng .locked , chúng ta có thể suy ra là phiên bản được mã hóa của tệp mà nó đã thay thế.

Tiếp theo, chúng ta sẽ xem tab Registry Keys, thể hiện trong Hình 3-7.

Summary

Files Registry Keys Mutexes

```
HKEY_CURRENT_USER\Control Panel\Mouse
HKEY_CURRENT_USER\Software\AutoIt v3\AutoIt
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\IMM
HKEY_USERS\S-1-5-21-1547161642-507921405-839522115-1004\Software\Microsoft\Windows
NT\CurrentVersion\AppCompatFlags\Layers
HKEY_CURRENT_USER\SOFTWARE\Microsoft\CTF
HKEY_LOCAL_MACHINE\Software\Microsoft\CTF\SystemShared
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\ComputerName
ActiveComputerName
```

Hình 3-7: Tab Khóa đăng ký của ngăn Tóm tắt, hiển thị các khóa đăng ký mà mẫu phần mềm độc hại của chúng tôi đã sửa đổi

Số đăng ký là cơ sở dữ liệu mà Windows sử dụng để lưu trữ thông tin cấu hình. Các tham số cấu hình được lưu dưới dạng khóa đăng ký và các khóa này có các giá trị liên quan. Tương tự như đường dẫn tệp trên hệ thống tệp Windows, các khóa đăng ký được phân cách bằng dấu gạch chéo ngược. Malwr.com cho chúng tôi thấy khóa đăng ký nào mà phần mềm độc hại của chúng tôi đã sửa đổi. Mặc dù điều này không được hiển thị trong Hình 3-7, nhưng nếu bạn xem báo cáo đầy đủ trên malwr.com, bạn sẽ thấy một khóa đăng ký đáng chú ý mà phần mềm độc hại của chúng tôi đã thay đổi là HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run, là khoá đăng ký yêu cầu Windows chạy các chương trình mỗi khi người dùng đăng nhập. Rất có thể phần mềm độc hại của chúng tôi đã sửa đổi số đăng ký này để yêu cầu Windows khởi động lại phần mềm độc hại mỗi khi hệ thống khởi động, điều này đảm bảo rằng phần mềm độc hại vẫn tồn tại từ lần khởi động lại này đến lần khởi động lại khác.

Tab Mutexes trong báo cáo malwr.com chứa tên của các mutex mà phần mềm độc hại đã tạo, như trong Hình 3-8.



Hình 3-8: Tab Mutexes của ngăn Tóm tắt, hiển thị các mutex nào mà mẫu phần mềm độc hại của chúng tôi đã tạo

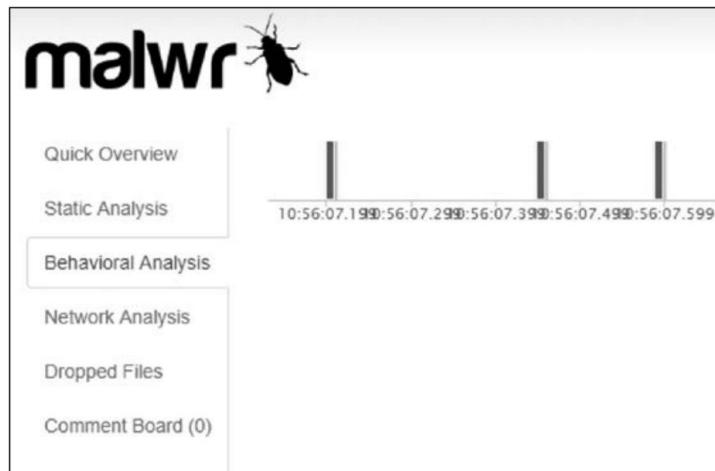
Mutexes là các tệp khóa báo hiệu rằng một chương trình đã chiếm quyền sở hữu một số tài nguyên. Phần mềm độc hại thường sử dụng mutexes để ngăn bản thân lây nhiễm hai lần vào hệ thống. Hóa ra ít nhất một mutex đã được tạo (CTF.TimListCache.FMPDefaultS-1-5-21-1547161642-507921405-839522115-1004MUTEX.DefaultS-1-5-21-1547161642-507921405-839522115-1004 ShimCacheMutex) được cộng đồng bảo mật biết là có liên quan đến phần mềm độc hại và có thể phục vụ mục đích này tại đây.

Phân tích cuộc gọi API

Nhấp vào tab Phân tích hành vi trên bảng điều khiển bên trái của giao diện người dùng malwr.com, như trong Hình 3-9, sẽ hiển thị thông tin chi tiết về hành vi của nhì phần phần mềm độc hại của chúng tôi.

Phần này cho biết lệnh gọi API nào được thực hiện bởi mỗi quy trình do phần mềm độc hại khởi chạy, cùng với các đối số và giá trị trả về của chúng. Việc đọc thông tin này tốn nhiều thời gian và đòi hỏi kiến thức chuyên môn về Windows API.

Mặc dù phần thảo luận chi tiết về phân tích lệnh gọi API của phần mềm độc hại nằm ngoài phạm vi của cuốn sách này, nhưng nếu bạn muốn tìm hiểu thêm, bạn có thể tra cứu các lệnh gọi API riêng lẻ để khám phá tác động của chúng.



Hình 3-9: Ngan Phân tích hành vi của báo cáo malwr.com cho mẫu phần mềm độc hại của chúng tôi, hiển thị khi các lệnh gọi API được thực hiện trong quá trình thực thi động

Mặc dù malwr.com là một tài nguyên tuyệt vời để phân tích động các mẫu phần mềm độc hại riêng lẻ, nhưng nó không phải là tài nguyên tuyệt vời để thực hiện phân tích động trên một số lượng lớn mẫu. Việc thực thi số lượng lớn mẫu trong môi trường động rất quan trọng đối với máy học và phân tích dữ liệu vì nó xác định mối quan hệ giữa các mẫu thực thi động của mẫu phần mềm độc hại. Việc tạo các hệ thống máy học có thể phát hiện các phiên bản phần mềm độc hại dựa trên các mẫu thực thi động của chúng yêu cầu chạy hàng nghìn mẫu phần mềm độc hại.

Ngoài hạn chế này, malwr.com không cung cấp phân tích phần mềm độc hại sis dẫn đến các định dạng có thể phân tích cú pháp bằng máy như XML hoặc JSON. Để giải quyết những vấn đề này, bạn phải thiết lập và chạy CuckooBox của riêng mình. May mắn thay, CuckooBox là mã nguồn mở và miễn phí. Nó cũng đi kèm với hướng dẫn từng bước để thiết lập môi trường phân tích động của riêng bạn. Tôi khuyến khích bạn làm như vậy bằng cách truy cập <http://cuckoosandbox.org/>. Nay giờ bạn đã hiểu cách diễn giải kết quả phân mềm độc hại động từ malwr.com, sử dụng CuckooBox đãng sau hậu trường, bạn cũng sẽ biết cách phân tích kết quả CuckooBox sau khi bạn thiết lập và chạy CuckooBox.

Hạn chế của Phân tích động cơ bản

Phân tích động là một công cụ mạnh mẽ, nhưng nó không phải là thuốc chữa bách bệnh phân tích phần mềm độc hại. Trong thực tế, nó có những hạn chế nghiêm trọng. Một hạn chế là các tác giả phần mềm độc hại biết về CuckooBox và các khung phân tích động khác và cố gắng phá vỡ chúng bằng cách làm cho phần mềm độc hại của họ không thực thi được khi phát hiện ra rằng nó đang chạy trong CuckooBox. Những người bảo trì CuckooBox biết rằng các tác giả phần mềm độc hại cố gắng làm điều này, vì vậy họ cố gắng vượt qua các nỗ lực phá vỡ CuckooBox của phần mềm độc hại. Trò chơi mèo và chuột này

phát liên tục sao cho một số mẫu phần mềm độc hại chắc chắn sẽ phát hiện ra rằng chúng đang chạy trong môi trường phân tích động và không thực thi được khi chúng tôi cố gắng chạy chúng.

Một hạn chế khác là ngay cả khi không có bất kỳ nỗ lực vượt thoát nào, phân tích động có thể không tiết lộ các hành vi quan trọng của phần mềm độc hại. Hãy xem xét trường hợp nhị phân phần mềm độc hại kết nối trở lại máy chủ từ xa khi thực thi và chờ lệnh được đưa ra. Ví dụ: các lệnh này có thể yêu cầu mẫu phần mềm độc hại tìm kiếm một số loại tệp nhất định trên máy chủ của nạn nhân, ghi lại các lần nhấn phím hoặc bật webcam.

Trong trường hợp này, nếu máy chủ từ xa không gửi lệnh hoặc không hoạt động nữa, thì sẽ không có hành vi nguy hiểm nào bị lộ. Do những hạn chế này, phân tích động không phải là cách khắc phục tất cả để phân tích phần mềm độc hại. Trên thực tế, các nhà phân tích phần mềm độc hại chuyên nghiệp kết hợp phân tích động và tĩnh để đạt được kết quả tốt nhất có thể.

Bản tóm tắt

Trong chương này, bạn đã chạy phân tích động trên mẫu phần mềm độc hại ransomware với malwr.com để phân tích kết quả. Bạn cũng đã tìm hiểu về những ưu điểm và nhược điểm của phân tích động. Bây giờ bạn đã học được cách tiến hành phân tích động cơ bản, bạn đã sẵn sàng đi sâu vào khoa học dữ liệu phần mềm độc hại.

Phần còn lại của cuốn sách này tập trung vào việc thực hiện khoa học dữ liệu phần mềm độc hại trên dữ liệu phần mềm độc hại dựa trên phân tích tĩnh. Tôi sẽ tập trung vào phân tích tĩnh sis vì nó đơn giản hơn và dễ dàng hơn để có được kết quả tốt so với phân tích động, khiến nó trở thành điểm khởi đầu tốt để bạn bắt tay vào nghiên cứu khoa học dữ liệu về phần mềm độc hại. Tuy nhiên, trong mỗi chương tiếp theo, tôi cũng sẽ giải thích cách bạn có thể áp dụng các phương pháp khoa học dữ liệu vào phân tích động dữ liệu dựa trên.

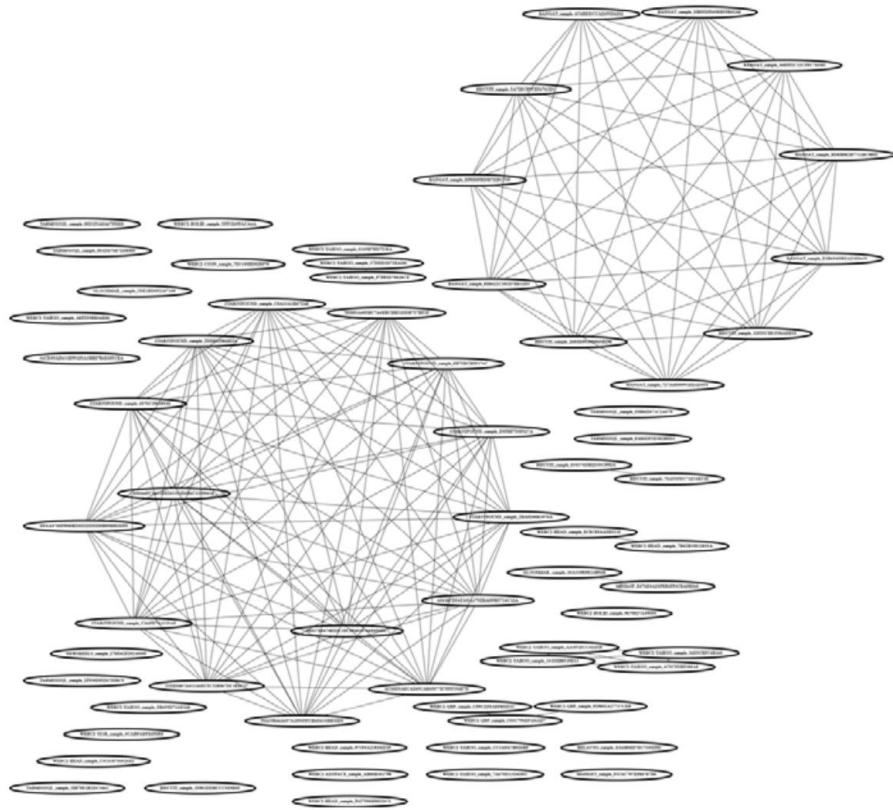
4

Xác định Tấn công Cam paign s Sử dụng Malware Networks



Phân tích mạng phần mềm độc hại có thể biến bộ dữ liệu phần mềm độc hại thành thông tin tình báo về mối đe dọa có giá trị, tiết lộ các chiến dịch tấn công đối thủ, chiến thuật phần mềm độc hại phổ biến và nguồn mẫu phần mềm độc hại. Cách tiếp cận này bao gồm phân tích cách thức mà các nhóm mẫu phần mềm độc hại được kết nối bởi các thuộc tính được chia sẻ của chúng, cho dù đó là địa chỉ IP nhúng, tên máy chủ, chuỗi ký tự có thể in, đồ họa hoặc tương tự.

Ví dụ: Hình 4-1 cho thấy một ví dụ về sức mạnh của phân tích mạng phần mềm độc hại trong một biểu đồ chỉ mất vài giây để tạo bằng các kỹ thuật bạn sẽ học trong chương này.



Hình 4-1: Các kết nối mạng xã hội của phần mềm độc hại cấp quốc gia được tiết lộ thông qua phân tích thuộc tính được chia sẻ

Hình này hiển thị một nhóm các mẫu phần mềm độc hại cấp quốc gia (được biểu thị bằng các nút hình bầu dục) và các kết nối "xã hội" của chúng (các đường kết nối các nút). Các kết nối dựa trên thực tế là các mẫu này "gọi lại" đến cùng tên máy chủ và địa chỉ IP, cho thấy chúng được triển khai bởi cùng một kẻ tấn công. Như bạn sẽ tìm hiểu trong chương này, bạn có thể sử dụng các kết nối này để giúp phân biệt giữa một cuộc tấn công phối hợp nhằm vào tổ chức của bạn và một loạt những kẻ tấn công có động cơ tội phạm khác nhau.

Đến cuối chương, bạn sẽ học được:

- Các nguyên tắc cơ bản của lý thuyết phân tích mạng liên quan đến việc trích xuất thông tin tình báo về mối đe dọa từ phần mềm độc hại
- Cách sử dụng trực quan hóa để xác định mối quan hệ giữa phần mềm độc hại mẫu
- Cách tạo, trực quan hóa và trích xuất thông tin tình báo từ mạng phần mềm độc hại hoạt động bằng Python và các bộ công cụ nguồn mở khác nhau để phân tích và trực quan hóa dữ liệu
- Làm thế nào để liên kết tất cả kiến thức này lại với nhau để phát hiện và phân tích cam tấn công paigns trong bộ dữ liệu phần mềm độc hại trong thế giới thực

Nút và Cạnh

Trước khi bạn có thể thực hiện phân tích thuộc tính dùng chung trên phần mềm độc hại, bạn cần hiểu một số điều cơ bản về mạng. Mạng là tập hợp các đối tượng được kết nối (được gọi là các nút). Các kết nối giữa các nút này được gọi là các cạnh. Là các đối tượng toán học trừu tượng, các nút trong một mạng lưới có thể đại diện cho hầu hết mọi thứ, cũng như các cạnh của chúng. Điều chúng tôi quan tâm cho mục đích của mình là cấu trúc của các kết nối giữa các nút và cạnh này, vì điều này có thể tiết lộ thông tin chi tiết về phần mềm độc hại.

Khi sử dụng mạng để phân tích phần mềm độc hại, chúng tôi có thể xử lý từng cá nhân tệp phần mềm độc hại dưới dạng định nghĩa của một nút và chúng tôi có thể coi các mối quan hệ quan tâm (chẳng hạn như mã được chia sẻ hoặc hành vi mạng) là định nghĩa của một cạnh. Các tệp phần mềm độc hại tương tự chia sẻ các cạnh và do đó nhóm lại với nhau khi chúng tôi áp dụng mạng hướng lực (bạn sẽ thấy chính xác cách thức hoạt động của điều này sau).

Ngoài ra, chúng ta có thể coi cả các mẫu và thuộc tính của phần mềm độc hại là các nút đối với chính chúng. Ví dụ: các địa chỉ IP gọi lại có các nút và các mẫu phần mềm độc hại cũng vậy. Bất cứ khi nào các mẫu phần mềm độc hại gọi lại một địa chỉ IP cụ thể, chúng sẽ được kết nối với nút địa chỉ IP đó.

Mạng của phần mềm độc hại có thể phức tạp hơn chữ không chỉ đơn giản là một tập hợp các nút và cạnh. Cụ thể, chúng có thể có các thuộc tính được gắn vào một trong hai nút hoặc cạnh, chẳng hạn như tỷ lệ phần trăm mà hai mẫu được kết nối chia sẻ.

Một thuộc tính cạnh phổ biến là trọng số, với trọng số lớn hơn cho thấy các kết nối mạnh hơn giữa các mẫu. Các nút có thể có thuộc tính riêng của chúng, chẳng hạn như kích thước tệp của mẫu phần mềm độc hại mà chúng đại diện, nhưng chúng thường chỉ được gọi là thuộc tính.

Mạng lưỡng cực

Mạng lưỡng cực là mạng có các nút có thể được chia thành hai phân vùng (nhóm), trong đó không phân vùng nào chứa kết nối bên trong. Các mạng loại này có thể được sử dụng để hiển thị các thuộc tính được chia sẻ giữa các mẫu phần mềm độc hại.

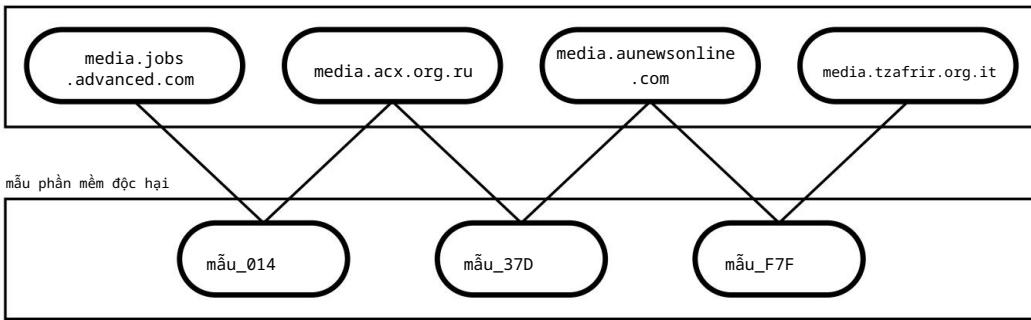
Hình 4-2 cho thấy một ví dụ về mạng hai bên trong đó các nút mẫu phần mềm độc hại đi vào phân vùng dưới cùng và tên miền mà các mẫu "gọi lại" đến (để liên lạc với kẻ tấn công) đi vào phân vùng khác. Lưu ý rằng các lệnh gọi lại không bao giờ kết nối trực tiếp với các lệnh gọi lại khác và các mẫu phần mềm độc hại không bao giờ kết nối trực tiếp với các mẫu phần mềm độc hại khác, như đặc điểm của mạng hai bên.

Như bạn có thể thấy, ngay cả một hình ảnh trực quan đơn giản như vậy cũng cho thấy một thông tin tình báo quan trọng: dựa trên các máy chủ gọi lại được chia sẻ của các mẫu phần mềm độc hại, chúng tôi có thể đoán rằng sample_014 có thể đã được triển khai bởi cùng một kẻ tấn công như sample_37D. Chúng ta cũng có thể đoán rằng sample_37D và sample_F7F có thể có chung kẻ tấn công, và sample_014 và sample_F7F có thể có cùng kẻ tấn công, bởi vì chúng được kết nối bởi mẫu sample_37D (và thực tế, các mẫu trong Hình 4-2 đều đến từ cùng một "APT1"

nhóm tấn công người Trung Quốc).

LƯU Ý Tôi muốn cảm ơn Mandiant và Mila Parkour vì đã quản lý các mẫu APT1 và cung cấp chúng cho cộng đồng nghiên cứu.

Gọi lại tên miền

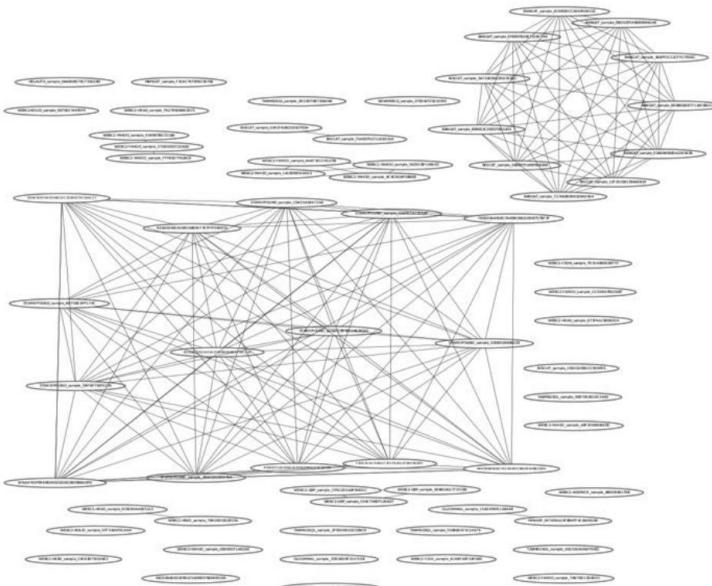


Hình 4-2: Mạng luồng cyclic. Các nút trên cùng (phân vùng được gán) là các tên miền gọi lại.

Các nút ở dưới cùng (phân vùng phần mềm độc hại) là các mẫu phần mềm độc hại.

Khi số lượng nút và kết nối trong mạng của chúng tôi tăng lên rất lớn, chúng tôi có thể muốn xem các mẫu phần mềm độc hại có liên quan với nhau như thế nào mà không cần phải kiểm tra chật chẽ tất cả các kết nối thuộc tính. Chúng tôi có thể kiểm tra sự giống nhau của mẫu phần mềm độc hại bằng cách tạo phép chiếu mạng hai bên, đây là phiên bản đơn giản hơn của mạng hai bên trong đó chúng tôi liên kết các nút trong một phân vùng của mạng nếu chúng có các nút trong phân vùng khác (thuộc tính phân vùng) chung. Ví dụ: trong trường hợp mẫu phần mềm độc hại được hiển thị trong Hình 4-1, chúng tôi sẽ tạo một mạng trong đó các mẫu phần mềm độc hại được liên kết nếu chúng chia sẻ tên miền gọi lại.

Hình 4-3 cho thấy mạng dự kiến của các máy chủ gọi lại được chia sẻ của toàn bộ bộ dữ liệu APT1 của Trung Quốc đã đề cập trước đó.



Hình 4-3: Hình chiếu các mẫu phần mềm độc hại từ bộ dữ liệu APT1, chỉ hiển thị các kết nối giữa các mẫu phần mềm độc hại nếu chúng chia sẻ ít nhất một máy chủ.

Hai cụm lớn được sử dụng trong hai chiến dịch tấn công khác nhau.

Các nút ở đây là các mẫu phần mềm độc hại và chúng được liên kết nếu chúng chia sẻ ít nhất một máy chủ gọi lại. Bằng cách chỉ hiển thị các kết nối giữa các mẫu phần mềm độc hại nếu chúng dùng chung máy chủ gọi lại, chúng ta có thể bắt đầu thấy “mạng xã hội” tổng thể của các mẫu phần mềm độc hại này. Như bạn có thể thấy trong Hình 4-3, tồn tại hai nhóm lớn (cụm hình vuông lớn ở khu vực trung tâm bên trái và cụm hình tròn ở khu vực trên cùng bên phải), khi kiểm tra kỹ hơn, chúng tương ứng với hai chiến dịch khác nhau. được thực hiện trong lịch sử 10 năm của nhóm APT1.

Trực quan hóa mạng phần mềm độc hại

Khi bạn thực hiện phân tích thuộc tính dùng chung của phần mềm độc hại bằng cách sử dụng mạng, bạn sẽ thấy rằng bạn phụ thuộc rất nhiều vào phần mềm trực quan hóa mạng để tạo các mạng giống như các mạng được hiển thị cho đến nay. Phần này giới thiệu cách tạo các trực quan hóa mạng này từ góc độ thuật toán.

Điều quan trọng, thách thức lớn trong việc thực hiện trực quan hóa mạng là bố cục mạng, đây là quá trình quyết định vị trí hiển thị từng nút trong mạng hoạt động trong không gian tọa độ hai hoặc ba chiều, tùy thuộc vào việc bạn muốn trực quan hóa của mình là hai- hoặc ba chiều. Khi bạn đặt các nút trên mạng, cách lý tưởng là đặt chúng trong không gian tọa độ sao cho khoảng cách trực quan của chúng với nhau tỷ lệ thuận với khoảng cách đường đi ngắn nhất giữa chúng trong mạng. Nói cách khác, các nút cách xa nhau hai bước nhảy có thể cách nhau khoảng hai inch và các nút cách xa ba bước nhảy có thể cách nhau khoảng ba inch. Làm điều này cho phép chúng tôi hình dung chính xác các cụm nút tương tự với mối quan hệ thực tế của chúng. Tuy nhiên, như bạn sẽ thấy trong phần tiếp theo, điều này thường khó đạt được, đặc biệt khi bạn đang làm việc với nhiều hơn ba nút.

Vấn đề biến dạng

Hóa ra, thường không thể giải quyết vấn đề bố cục mạng này một cách hoàn hảo. Hình 4-4 minh họa khó khăn này.

Như bạn có thể thấy trong các mạng đơn giản này, tất cả các nút được kết nối với tất cả các nút khác bằng các cạnh có trọng số bằng 1. Bố cục lý tưởng cho các kết nối này sẽ đặt tất cả các nút cách đều nhau trên trang.

Nhưng như bạn có thể thấy, khi chúng ta tạo các mạng gồm bốn và sau đó là năm nút, như trong (c) và (d), chúng ta bắt đầu tạo ra nhiều biến dạng hơn do các cạnh có độ dài không bằng nhau. Thật không may, chúng ta chỉ có thể giảm thiểu chứ không loại bỏ được sự biến dạng này và việc giảm thiểu đó trở thành một trong những mục tiêu chính của thuật toán trực quan hóa công việc mạng.

a) Hai nút thông nhau, không méo,
tất cả các nút có chiều dài bằng nhau

b) Ba nút được kết nối, không biến dạng,
tất cả các nút có chiều dài bằng nhau

c) Bốn nút được kết nối, một số biến dạng,
một số nút gần hơn những nút khác

d) Năm nút được kết nối, biến dạng nhiều hơn,
khoảng cách nút không đồng nhất

Hình 4-4: Bố cục mạng hoàn hảo thường không thể thực hiện được trên các mạng phần mềm độc hại trong thế giới thực. Các trường hợp đơn giản như (a) và (b) cho phép chúng ta bố trí tất cả các nút cách đều nhau. Tuy nhiên, (c) thêm biến dạng (các cạnh không còn có chiều dài bằng nhau) và (d) cho thấy biến dạng nhiều hơn.

Thuật toán hướng lực

Để giảm thiểu biến dạng bố cục một cách tốt nhất, các nhà khoa học máy tính thường sử dụng hướng lực thuật toán bối cục. Các thuật toán hướng lực dựa trên các mô phỏng vật lý của các lực giống như lò xo cũng như từ tính. Mô phỏng các cạnh của mạng dưới dạng các lò xo vật lý thường dẫn đến việc định vị nút tốt, bởi vì các lò xo mô phỏng đẩy và kéo để cố gắng đạt được độ dài đồng nhất giữa các nút và các cạnh. Để hình dung rõ hơn về khái niệm này, hãy xem xét cách lò xo hoạt động: khi bạn nén hoặc kéo dài lò xo, nó sẽ “cố gắng” quay trở lại chiều dài của nó ở trạng thái cân bằng. Các thuộc tính này tương quan tốt với mong muốn của chúng tôi là cố tắt cả các cạnh trong mạng của chúng tôi có độ dài bằng nhau. Các thuật toán hướng lực là những gì chúng ta tập trung vào trong chương này.

Xây dựng mạng với NetworkX

Bây giờ bạn đã có hiểu biết cơ bản về mạng phần mềm độc hại, bạn đã sẵn sàng tìm hiểu cách tạo mạng mối quan hệ phần mềm độc hại bằng thư viện phân tích mạng NetworkX Python mã nguồn mở và GraphViz mở

40 Chương 4

bộ công cụ trực quan hóa mạng nguồn. Tôi chỉ cho bạn cách trích xuất dữ liệu liên quan đến phần mềm độc hại theo chương trình, sau đó sử dụng dữ liệu này để xây dựng, trực quan hóa và phân tích mạng để thể hiện bộ dữ liệu phần mềm độc hại.

Hãy bắt đầu với NetworkX, một dự án mã nguồn mở được duy trì bởi một nhóm có trung tâm là Phòng thí nghiệm Quốc gia Los Alamos và thư viện xử lý mạng trên thực tế của Python (hãy nhớ rằng bạn có thể cài đặt các phần phụ thuộc của thư viện trong chương này, bao gồm cả NetworkX, bằng cách nhập tên của chương này mã và thư mục dữ liệu và lệnh pip install -r tests.txt). Nếu bạn biết Python, bạn sẽ thấy NetworkX dễ sử dụng một cách đáng ngạc nhiên. Sử dụng mã trong [Liệt kê 4-1](#) để nhập NetworkX và khởi tạo mạng.

```
#!/usr/bin/trǎn
nhập khẩu mạngx

# khởi tạo một mạng không có nút và không có cạnh.
mạng = networkx.Graph()
```

Liệt kê 4-1: Khởi tạo mạng

Mã này chỉ sử dụng một lệnh gọi hàm tới hàm tạo NetworkX Graph để tạo một mạng trong NetworkX.

Lưu ý / viễn NetworkX đổi khi sử dụng thuật ngữ đồ thị thay cho mạng , vì hai thuật ngữ này đồng nghĩa với nhau trong khoa học máy tính-cả hai đều chỉ một tập hợp các nút được kết nối bởi các cạnh.

Thêm nút và cạnh

Bây giờ chúng ta đã khởi tạo một mạng, hãy thêm một số nút. Một nút trong mạng NetworkX có thể là bất kỳ đối tượng Python nào. Ở đây tôi chỉ cho bạn cách thêm các nút thuộc nhiều loại khác nhau vào mạng của chúng tôi:

```
các nút = ["xin chào", "thé giới", 1, 2, 3]
cho nút trong nút:
mạng.add_node(nút)
```

Như được hiển thị, chúng tôi đã thêm năm nút vào mạng của mình: "xin chào", "thé giới", 1, 2 và 3.

Sau đó, để thêm các cạnh, chúng ta gọi add_edge(), như hình sau:

```
network.add_edge("xin chào", "thé giới")
mạng.add_edge(1, 2)
mạng.add_edge(1, 3)
```

Ở đây, chúng tôi đang kết nối một số trong năm nút này thông qua các cạnh. Ví dụ, dòng mã đầu tiên kết nối các nút "hello" và "world" với nhau bằng cách tạo một cạnh giữa chúng.

Thêm thuộc tính

NetworkX cho phép chúng tôi dễ dàng định kèm các thuộc tính cho cả nút và cạnh. Để gắn một thuộc tính vào một nút (và để truy cập vào thuộc tính đó sau này), bạn có thể thêm thuộc tính đó làm đối số từ khóa khi bạn thêm nút vào mạng, như sau:

```
network.add_node(1,myattribute="foo")
```

Để thêm một thuộc tính sau này, hãy truy cập từ điển nút của mạng bằng cách sau:

```
network.node[1]["myattribute"] = "foo"
```

Sau đó, để truy cập nút, hãy truy cập từ điển nút :

```
in network.node[1]["myattribute"] # in "foo"
```

Như với các nút, bạn có thể thêm các thuộc tính vào các cạnh bằng cách sử dụng các đối số từ khóa khi bạn thêm các cạnh ban đầu, như minh họa ở đây:

```
network.add_edge("node1","node2",myattribute="thuộc tính của một cạnh")
```

Tương tự, bạn có thể thêm các thuộc tính cho các cạnh sau khi chúng được thêm vào mạng bằng cách sử dụng từ điển cạnh , như minh họa ở đây:

```
network.edge["node1"]["node2"]["myattribute"] = "thuộc tính của một cạnh"
```

Từ điển cạnh kỳ diệu ở chỗ nó cho phép bạn truy cập các thuộc tính nút theo cách khác, mà không phải lo lắng về việc bạn tham chiếu đến nút nào trước, như thể hiện trong Liệt kê 4-2.

```
network.edge["node1"]["node2"]["myattribute"] = 321
in mạng.edge["node2"]["node1"]["myattribute"] # in 321
```

Liệt kê 4-2: Sử dụng từ điển cạnh để truy cập các thuộc tính nút bất kể thứ tự

Như bạn có thể thấy, dòng đầu tiên đặt myattribute trên một cạnh kết nối node1 và node2 , và dòng thứ hai truy cập myattribute mặc dù node1 và các tham chiếu node2 bị lật .

Lưu mạng vào đĩa

Để trực quan hóa mạng của chúng tôi, chúng tôi cần lưu chúng vào đĩa từ NetworkX ở định dạng .dot –một định dạng thường được sử dụng trong thế giới phân tích mạng có thể được nhập vào nhiều bộ công cụ trực quan hóa mạng. Để lưu một mạng ở định dạng .dot , chỉ cần gọi hàm NetworkX write_dot() , như trong Liệt kê 4-3.

```
#!/usr/bin/trần
nhập khẩu mạngx
từ networkx.drawing.nx_agraph nhập write_dot

# khởi tạo mạng, thêm một số nút và kết nối chúng
các nút = ["xin chào","thế giới",1,2,3]
mạng = networkx.Graph()
cho nút trong nút:

    mạng.add_node(nút)
network.add_edge("xin chào","thế giới")
write_dot(unetwork, "network.dot")
```

Liệt kê 4-3: Sử dụng write_dot() để lưu mạng vào file

Như bạn có thể thấy, ở cuối mã, chúng ta sử dụng hàm write_dot() để chỉ định mạng mà chúng ta muốn lưu cũng như đường dẫn hoặc tên tệp mà chúng ta muốn lưu vào .

Trực quan hóa mạng với GraphViz

Khi chúng ta đã ghi một mạng vào file bằng hàm write_dot() NetworkX, chúng ta có thể trực quan hóa tệp kết quả bằng GraphViz. GraphViz là gói dòng lệnh tốt nhất hiện có để trực quan hóa mạng của bạn. Nó được hỗ trợ bởi các nhà nghiên cứu tại AT&T và đã trở thành một phần tiêu chuẩn của hộp công cụ phân tích mạng được các nhà phân tích dữ liệu sử dụng. Nó chứa một loạt các công cụ bố trí mạng dòng lệnh có thể được sử dụng để bố trí và kết xuất mạng. GraphViz được cài đặt sẵn trên máy ảo được cung cấp cùng với cuốn sách này và cũng có thể tải xuống tại <https://graphviz.gitlab.io/>

Tải xuống/. Mỗi công cụ dòng lệnh của GraphViz đều nhận các mạng được biểu thị ở định dạng .dot và có thể được gọi bằng cú pháp sau để hiển thị mạng dưới dạng tệp .png :

```
$ <tên công cụ> <dotfile> -T png -o <outputfile.png>
```

Trình kết xuất đồ thị hướng lực fdp là một công cụ trực quan hóa mạng GraphViz. Nó sử dụng giao diện dòng lệnh cơ bản giống như mọi công cụ GraphViz khác, như được hiển thị ở đây:

```
$ fdp apt1callback.dot -T png -o apt1callback.png
```

Ở đây, chúng tôi xác định rằng chúng tôi muốn sử dụng công cụ fdp và đặt tên cho tệp .dot mà chúng tôi muốn bố trí, đó là apt1callback.dot, được tìm thấy trong thư mục ~/ch3/ của dữ liệu đi kèm cuốn sách này. Chúng tôi chỉ định -T png để chỉ ra định dạng (PNG) mà chúng tôi muốn sử dụng. Cuối cùng, chúng tôi chỉ định nơi chúng tôi muốn lưu tệp đầu ra bằng cách sử dụng -o apt1callback.png.

Sử dụng Tham số để Điều chỉnh Mạng

Các công cụ GraphViz bao gồm nhiều tham số mà bạn có thể sử dụng để điều chỉnh cách vẽ mạng của mình. Nhiều tham số trong số này được thiết lập bằng cách sử dụng -G cờ dòng lệnh ở định dạng sau:

```
G<ten tham so>=<gia tri tham so>
```

Hai tham số đặc biệt hữu ích là `chồng chéo` và `splines`. Đặt `chồng chéo` thành `false` để yêu cầu GraphViz không cho phép bất kỳ nút nào chồng lên nhau. Sử dụng tham số `splines` để yêu cầu GraphViz vẽ các đường cong thay vì đường thẳng để dễ dàng theo dõi các cạnh trên mạng của bạn. Sau đây là một số cách để đặt tham số chồng lấp và `splines` trong GraphViz.

Sử dụng cách sau để ngăn các nút chồng lên nhau:

```
$ <tên công cụ> <dotfile> -Goverlap=false -T png -o <outputfile.png>
```

Vẽ các cạnh dưới dạng các đường cong (`splines`) để cải thiện khả năng đọc mạng:

```
$ <tên công cụ> <dotfile> -Gsplines=true -T png -o <outputfile.png>
```

Vẽ các cạnh dưới dạng các đường cong (`splines`) để cải thiện khả năng đọc mạng và không cho phép các nút chồng chéo trực quan:

```
$ <tên công cụ> <dotfile> -Gsplines=true -Goverlap=false -T png -o <outputfile.png>
```

Lưu ý rằng chúng tôi chỉ liệt kê một tham số sau tham số kia: `-Gsplines=true -Goverlap=false` (thứ tự không quan trọng), tiếp theo là `-T png -o <outputfile.png>`.

Trong phần tiếp theo, tôi sẽ giới thiệu một số công cụ GraphViz hữu ích nhất (ngoài `fdp`).

Công cụ dòng lệnh GraphViz

Dưới đây là một số công cụ GraphViz có sẵn mà tôi thấy hữu ích nhất, cũng như một số ý nghĩa về thời điểm thích hợp để sử dụng từng công cụ.

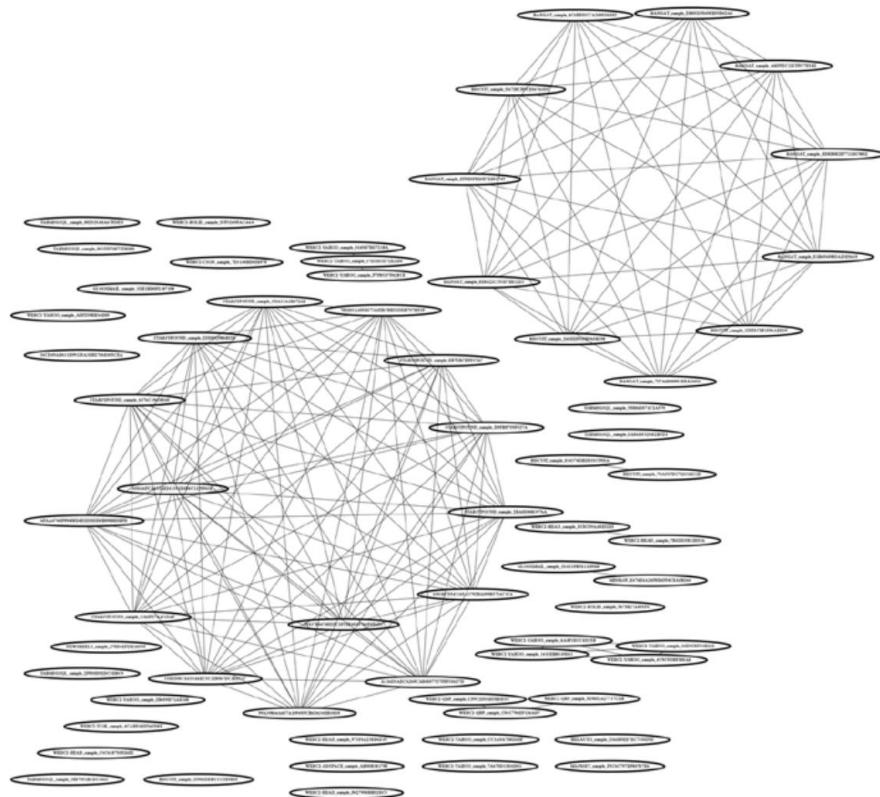
`fdp`

Chúng tôi đã sử dụng công cụ `fdp` trong ví dụ trước, công cụ này chúng tôi đã sử dụng để tạo bố cục hướng lực, như được mô tả trong "Thuật toán hướng lực" trên trang 40. Khi bạn đang tạo mạng phần mềm độc hại có ít hơn 500 nút, `fdp` sẽ thực hiện một công việc tốt để tiết lộ cấu trúc mạng trong một khoảng thời gian hợp lý. Nhưng khi bạn đang làm việc với hơn 500 nút và đặc biệt là khi kết nối giữa các nút phức tạp, bạn sẽ thấy rằng `fdp` chậm lại khá nhanh.

Để dùng thử fdp trên mạng máy chủ gọi lại dùng chung APT1 được hiển thị trong Hình 4-3, nhập thông tin sau từ thư mục ch4 của dữ liệu đi kèm cuốn sách này (bạn phải cài đặt GraphViz):

```
$ fdp callback_servers_malware_projection.dot -T png -o fdp_servers.png -  
Chinh phu = sai
```

Lệnh này sẽ tạo một tệp .png (fdp_servers.png) hiển thị mạng hoạt động giống như hiển thị trong Hình 4-5.



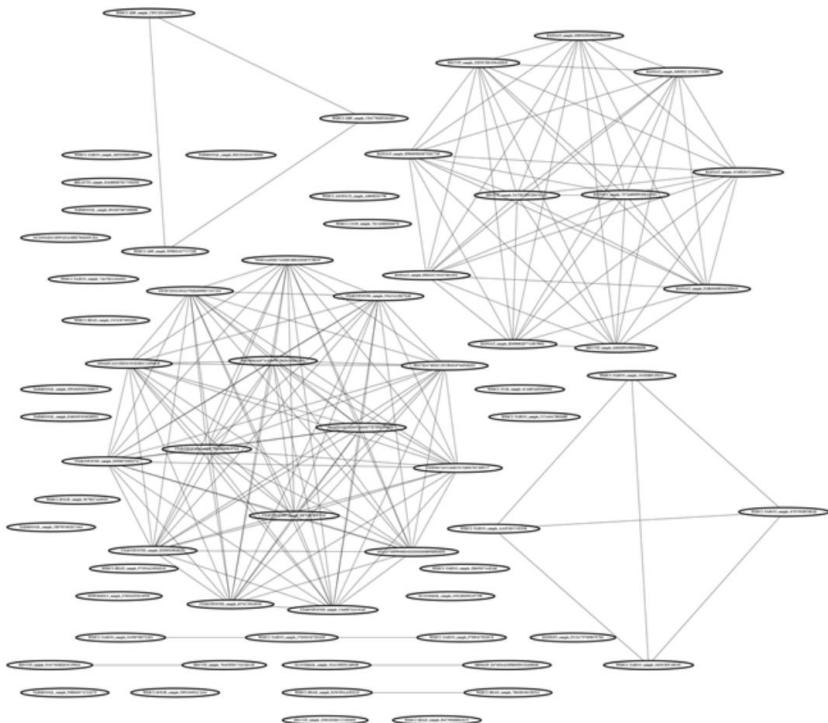
Hình 4-5: Bố cục mẫu APT1 sử dụng công cụ fdp

Bố cục fdp làm cho một số chủ đề rõ ràng trong hình. Đầu tiên, hai cụm mảng lớn có mối quan hệ mật thiết với nhau, như có thể thấy rõ ở khu vực phía trên bên phải và phía dưới bên trái của hình. Thứ hai, một số cặp mảng có liên quan với nhau, có thể nhìn thấy ở phía dưới bên phải. Cuối cùng, nhiều mảng cho thấy không có mối quan hệ rõ ràng với nhau và không được kết nối với bất kỳ nút nào khác. Điều quan trọng cần nhớ là hình ảnh trực quan này dựa trên mối quan hệ máy chủ gọi lại được chia sẻ giữa các nút. Có thể các mảng không được kết nối có liên quan đến các mảng khác trong hình bằng các loại quan hệ khác, chẳng hạn như quan hệ mã được chia sẻ—các mối quan hệ mà chúng ta sẽ khám phá trong Chương 5.

sfdfp

Công cụ sfdfp sử dụng cách tiếp cận bô cục gần giống như fdp, nhưng nó chia tỷ lệ tốt hơn vì nó tạo ra một hệ thống phân cấp đơn giản hóa, được gọi là thô, trong đó các nút được hợp nhất thành các siêu nút dựa trên mức độ gần nhau của chúng. Sau khi hoàn thành việc chỉnh sửa thô, công cụ sfdfp đưa ra các phiên bản được hợp nhất của biểu đồ có ít nút và cạnh hơn, giúp tăng tốc đáng kể quá trình bô cục. Bằng cách này, sfdfp có thể thực hiện ít tính toán hơn để tìm ra vị trí tốt nhất trong mạng. Do đó, sfdfp có thể bô trí hàng chục nghìn nút trên một máy tính xách tay thông thường, khiến nó trở thành thuật toán tốt nhất để bô trí các mạng phần mềm độc hại rất lớn.

Tuy nhiên, khả năng mở rộng này phải trả giá: sfdfp tạo ra các bô cục đôi khi ít rõ ràng hơn bô cục của các mạng có cùng kích thước trong fdp. Ví dụ, so sánh Hình 4-6 mà tôi đã tạo bằng sfdfp với mạng được tạo bằng fdp, được minh họa trong Hình 4-5.



Hình 4-6: Bô cục mạng máy chủ gọi lại dùng chung của các mẫu APT1 bằng lệnh sfdfp

Như bạn có thể thấy, có nhiều hơn một chút trên mỗi cụm trong Hình 4-6, khiến bạn khó nhìn thấy điều gì đang diễn ra hơn một chút.

Để tạo mạng này, hãy nhập thư mục ch4 của dữ liệu đi kèm cuốn sách này rồi nhập đoạn mã sau để tạo sfdfp_servers.png

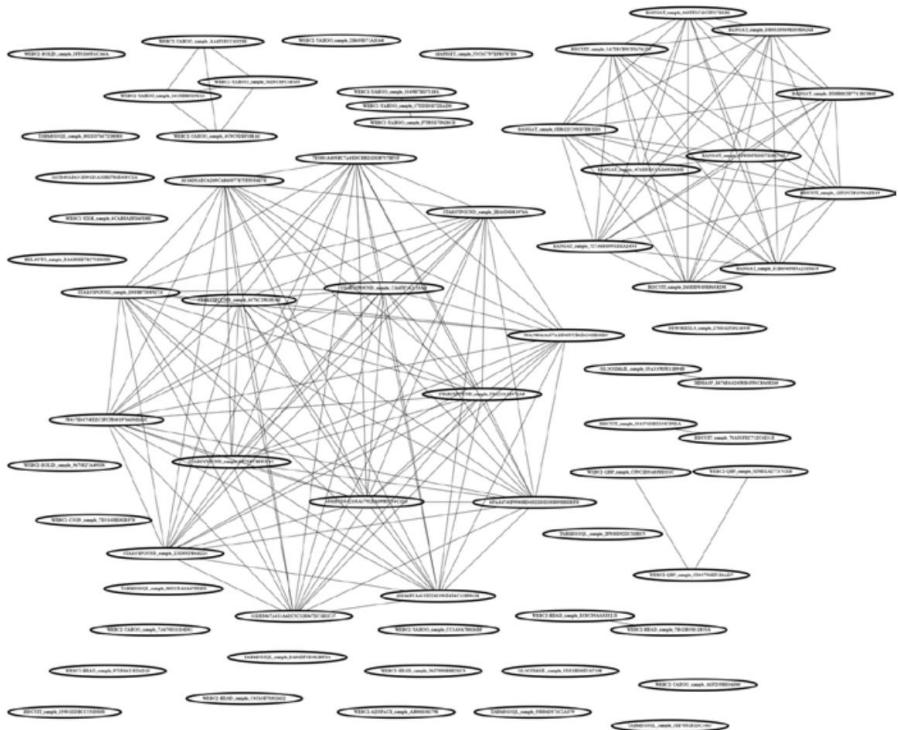
tệp hình ảnh được hiển thị trong Hình 4-6:

```
$ sfdfp callback_servers_malware_projection.dot -T png -o sfdfp_servers.png -Goverlap=false
```

Lưu ý cách mục đầu tiên trong mã này xác định rằng chúng tôi đang sử dụng công cụ sfdp, trái ngược với fdp trước đây. Mọi thứ khác đều giống nhau, hãy lưu tên tệp đầu ra.

gọn gàng

Công cụ gọn gàng là triển khai GraphViz của một thuật toán bồ trí mạng hướng lực khác nhau tạo ra các lò xo mô phỏng giữa tất cả các nút (bao gồm cả các nút không được kết nối) để giúp đẩy mọi thứ đến vị trí lý tưởng, nhưng với chi phí tính toán bồ sung. Thật khó để biết khi nào thì gọn gàng sẽ tạo ra bồ cục tốt nhất cho một mạng cụ thể: lời khuyên của tôi là bạn nên dùng thử, kết hợp với fdp và xem bạn thích bồ cục nào hơn. Hình 4-7 cho thấy bồ cục gọn gàng trông như thế nào trên mạng máy chủ gọi lại dùng chung APT1.



Hình 4-7: Bồ cục của mạng máy chủ gọi lại chia sẻ APT1 sử dụng bồ cục gọn gàng

Như bạn có thể thấy, trong trường hợp này, gọn gàng tạo ra một bồ cục mạng tương tự như bồ cục mạng do fdp và sfdp tạo ra. Tuy nhiên, đối với một số tập dữ liệu, bạn sẽ thấy rằng cách sắp xếp gọn gàng tạo ra bồ cục tốt hơn hoặc xấu hơn - bạn chỉ cần thử với tập dữ liệu của mình và xem. Để dùng thử cách gọn gàng, hãy nhập thông tin sau từ thư mục ch4 của dữ liệu đi kèm cuốn sách này; điều này sẽ tạo ra tệp hình ảnh công việc mạng gọn gàng_servers.png như trong Hình 4-7:

```
$ gong_gang callback_servers_malware_projection.dot -T png -o gọn_gàng_servers.png -Goverlap=false
```

Để tạo mạng này, chúng ta chỉ cần sửa lại mã mà chúng ta đã sử dụng để tạo Hình 4-6 để xác định rằng chúng ta muốn sử dụng công cụ gọn gàng và sau đó lưu .png vào gọn gàng.png. Bây giờ bạn đã biết cách tạo các trực quan hóa mạng này, hãy xem xét các cách để cải thiện chúng.

Thêm các thuộc tính trực quan vào các nút và cạnh

Ngoài việc quyết định bố cục mạng chung của bạn, có thể hữu ích nếu có thể chỉ định cách hiển thị các nút và cạnh riêng lẻ. Ví dụ: bạn có thể muốn đặt độ dày của cạnh dựa trên cường độ kết nối giữa hai nút hoặc đặt màu nút dựa trên mức độ xâm phạm mà mỗi nút mẫu phần mềm độc hại được liên kết, điều này sẽ cho phép bạn hình dung rõ hơn các cụm phần mềm độc hại. NetworkX và GraphViz giúp bạn dễ dàng thực hiện điều này bằng cách cho phép bạn chỉ định các thuộc tính trực quan của các nút và cạnh chỉ bằng cách gán giá trị cho một tập hợp các thuộc tính. Tôi chỉ thảo luận một vài thuộc tính như vậy trong các phần tiếp theo, nhưng chủ đề này đủ sâu để lập đầy toàn bộ cuốn sách.

Chiều rộng cạnh

Để đặt độ rộng của đường viền mà GraphViz vẽ xung quanh các nút hoặc đường mà nó vẽ cho các cạnh, bạn có thể đặt thuộc tính penwidth của các nút và cạnh thành một số tùy ý, như trong Liệt kê 4-4.

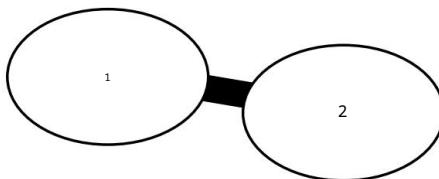
```
#!/usr/bin/trần
nhập khẩu mạngx
từ networkx.drawing.nx_agraph nhập writedot

g = networkx.Graph()
g.add_node(1)
g.add_node(2)
g.add_edge(1,2,vpenwidth=10) # làm cho cạnh thêm rộng
write_dot(g,'network.dot')
```

Liệt kê 4-4: Thiết lập thuộc tính penwidth

Ở đây, tôi tạo một mạng đơn giản với hai nút được kết nối bởi một cạnh, và tôi đặt thuộc tính penwidth của cạnh là 10 (giá trị mặc định là 1).

Chạy mã này, và bạn sẽ thấy một hình ảnh giống như Hình 4-8.



Hình 4-8: Một mạng đơn giản với một cạnh có penwidth là 10

Như bạn có thể thấy trong Hình 4-8, một chiều rộng bằng 10 dẫn đến một cạnh rất dày. Chiều rộng của cạnh (hoặc, độ dày của đường viền của nút nếu bạn đặt chiều rộng của nút) tỷ lệ thuận với giá trị của

thuộc tính băng thông , vì vậy hãy chọn cho phù hợp. Ví dụ: nếu giá trị cường độ cạnh của bạn thay đổi từ 1 đến 1000, nhưng bạn muốn có thể xem tất cả các cạnh, bạn có thể muốn xem xét việc gán các thuộc tính băng thông dựa trên tỷ lệ nhật ký của các giá trị cường độ cạnh của bạn.

Màu nút và cạnh

Để đặt màu cho đường viền của nút hoặc cạnh, hãy sử dụng thuộc tính color .

Liệt kê 4-5 cho thấy cách thực hiện việc này.

```
#!/usr/bin/trần
```

```
nhập khẩu mạngx
từ networkx.drawing.nx_agraph nhập write_dot
```

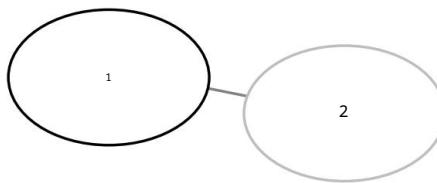
```
g = mạngx.Graph()
g.add_node(1,ucolor="blue") # làm cho đường viền của nút có màu xanh lam
g.add_node(2,vcolor="pink") # tạo đường viền nút màu hồng
g.add_edge(1,2,wcolor="red") # làm cho cạnh có màu đỏ
write_dot(g,'network.dot')
```

Liệt kê 4-5: Đặt màu cho nút và cạnh

Ở đây, tôi tạo cùng một mạng đơn giản mà tôi đã tạo trong Liệt kê 4-4, với hai nút và một cạnh kết nối chúng. Đối với mỗi nút mà tôi tạo, tôi đặt giá trị màu của nó (và). Tôi cũng đặt giá trị màu cho cạnh khi tôi tạo nó.

Hình 4-9 cho thấy kết quả của Liệt kê 4-5. Như mong đợi, bạn sẽ thấy rằng mỗi nút đều tiên (cạnh) và nút thứ hai có một màu duy nhất.

Để biết danh sách đầy đủ các màu bạn có thể sử dụng, hãy tham khảo <http://www.graphviz.org/doc/html/colors.html>.



Hình 4-9: Một mạng đơn giản minh họa cách đặt màu cho nút và cạnh

Màu sắc có thể được sử dụng để hiển thị các lớp nút và cạnh khác nhau.

Hình dạng nút

Để đặt hình dạng của nút, hãy sử dụng thuộc tính hình dạng với một chuỗi chỉ định hình dạng, như được định nghĩa tại <http://www.GraphViz.org/doc/info/shapes.html>. Các giá trị thường được sử dụng là hình hộp, hình elip, hình tròn, hình quả trứng , hình thoi, hình tam giác , hình ngũ giác và hình lục giác. Liệt kê 4-6 cho thấy cách thiết lập thuộc tính hình dạng của một nút.

```
#!/usr/bin/trần

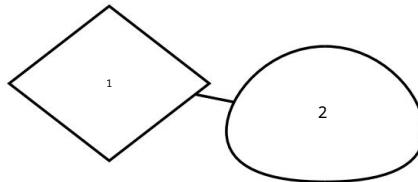
nhập khẩu mạngx
từ networkx.drawing.nx_agraph nhập write_dot

g = mạngx.Graph()
g.add_node(1, ushape='kim cương')
g.add_node(2, vshape='quả trứng')
g.add_edge(1,2)

write_dot(g, 'network.dot')
```

Liệt kê 4-6: Đặt hình dạng nút

Tương tự như cách chúng ta đặt màu cho nút, chúng ta chỉ cần sử dụng đối số từ khóa `shape` trong hàm `add_node()` để chỉ định hình dạng mà chúng ta muốn mỗi nút có. Ở đây, chúng tôi đặt nút đầu tiên thành hình thoi và nút thứ hai thành hình quả trứng. Kết quả của mã này được hiển thị trong Hình 4-10.



Hình 4-10: Một mạng đơn giản cho thấy cách chúng ta có thể thiết lập hình dạng nút

Kết quả, cho thấy một nút hình kim cương và một nút hình quả trứng, phản ánh các hình dạng mà chúng tôi đã chỉ định trong Liệt kê 4-6.

Nhãn văn bản

Cuối cùng, GraphViz cũng cho phép bạn thêm nhãn cho các nút và cạnh bằng thuộc tính `label`. Mặc dù các nút được tự động gắn nhãn dựa trên ID được chỉ định của chúng (ví dụ: nhãn cho nút được thêm là 123 sẽ là 123), bạn có thể chỉ định nhãn bằng cách sử dụng `label=<thuộc tính nhãn của tôi>`. Không giống như các nút, các cạnh không được gắn nhãn theo mặc định, nhưng bạn có thể gán nhãn cho chúng bằng cách sử dụng thuộc tính `label`. Liệt kê 4-7 cho thấy cách tạo mạng hai nút quen thuộc của chúng ta nhưng với các thuộc tính nhãn được gắn cho cả hai nút và cạnh kết nối.

```
#!/usr/bin/trần

nhập khẩu mạngx
từ networkx.drawing.nx_agraph nhập write_dot

g = mạngx.Graph()
g.add_node(1, ulabel="nút đầu tiên")
g.add_node(2, vlabel="nút thứ hai")
```

```

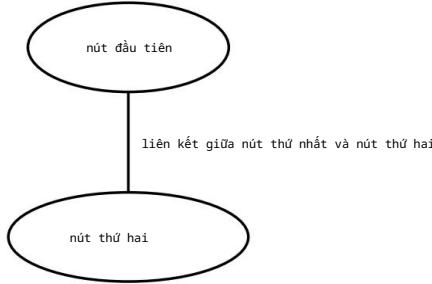
g.add_edge(1,2,wlabel="liên kết giữa nút thứ nhất và nút thứ hai")

write_dot(g,'network.dot')

```

Liệt kê 4-7: Ghi nhãn các nút và cạnh

Chúng tôi đặt tên tương ứng cho các nút là nút đầu tiên và nút thứ hai. Chúng tôi cũng đặt tên cho cạnh kết nối chúng là liên kết giữa nút thứ nhất và nút thứ hai. Hình 4-11 cho thấy đầu ra đồ họa mà chúng ta mong đợi.



Hình 4-11: Một mạng đơn giản cho thấy cách chúng ta có thể gắn nhãn các nút và cạnh

Bây giờ bạn đã biết cách thao tác các thuộc tính cơ bản của các nút và cạnh, bạn đã sẵn sàng bắt đầu xây dựng mạng từ đầu.

Xây dựng mạng phần mềm độc hại

Chúng ta sẽ bắt đầu thảo luận về việc xây dựng mạng phần mềm độc hại bằng cách sao chép và mở rộng trên ví dụ máy chủ gọi lại được chia sẻ như trong Hình 4-1, sau đó kiểm tra phân tích hình ảnh phần mềm độc hại được chia sẻ.

Chương trình sau trích xuất tên miền gọi lại từ phần mềm độc hại các tệp và sau đó xây dựng một mạng gồm các mẫu phần mềm độc hại gồm hai bên. Tiếp theo, nó thực hiện một phép chiếu mạng để hiển thị các mẫu phần mềm độc hại nào chia sẻ các máy chủ gọi lại phổ biến và nó thực hiện một phép chiếu khác để hiển thị các máy chủ gọi lại nào được gọi bởi các mẫu phần mềm độc hại phổ biến. Cuối cùng, chương trình lưu ba mạng-mạng lưỡng cực ban đầu, phép chiếu mẫu phần mềm độc hại và phép chiếu máy chủ gọi lại-dưới dạng các tệp để chúng có thể được hiển thị trực quan bằng GraphViz.

Tôi hướng dẫn bạn qua chương trình, từng phần một. mã hoàn chỉnh có thể được tìm thấy trong dữ liệu đi kèm cuốn sách này tại đường dẫn tệp ch4/gọi lại_server_network.py.

Liệt kê 4-8 cho thấy cách bắt đầu bằng cách nhập các mô-đun cần thiết.

```
#!/usr/bin/trần
```

```

nhập pefileu
nhập khẩu hệ thống

```

```

nhập argparse
nhập hệ điều hành
nhập pprint
nhập mạngxv
nhập lại
từ networkx.drawing.nx_agraph nhập write_dot
bộ sưu tập nhập khẩu
từ networkx.algorithms nhập lưỡng cực

```

Liệt kê 4-8: Nhập mô-đun

Trong số các mô-đun cần thiết mà chúng tôi đã nhập, đáng chú ý nhất là tệp pefile. Mô-đun phân tích cú pháp PE mà chúng tôi sử dụng để phân tích cú pháp nhị phân PE đích và thư viện networkx mà chúng tôi sử dụng để tạo mạng thuộc tính phần mềm độc hại.

Tiếp theo, chúng ta phân tích các đối số dòng lệnh bằng cách thêm mã vào Liệt kê 4-9.

```

args = argparse.ArgumentParser("Trực quan hóa mỗi quan hệ nhập DLL được chia sẻ giữa
một thư mục mẫu phần mềm độc hại")
args.add_argument( "target_path",help="thư mục chứa các mẫu phần mềm độc hại")
args.add_argument( "output_file",help="file để ghi tệp DOT vào")
args.add_argument( "malware_projection",help="file để ghi tệp DOT vào")
args.add_argument( "resource_projection",help="file để ghi tệp DOT vào")
args = args.parse_args()

```

Liệt kê 4-9: Phân tích đối số dòng lệnh

Các đối số này bao gồm target_path (đường dẫn đến thư mục chứa phần mềm độc hại chúng tôi đang phân tích), output_file (đường dẫn chúng tôi viết toàn bộ mạng), malware_projection (đường dẫn chúng tôi viết phiên bản rút gọn của biểu đồ và hiển thị mẫu phần mềm độc hại nào chia sẻ thuộc tính) và resource_projection (đường dẫn nơi chúng tôi viết phiên bản rút gọn của biểu đồ và hiển thị những thuộc tính nào được nhìn thấy cùng nhau trong các mẫu phần mềm độc hại).

Bây giờ chúng ta đã sẵn sàng để đi vào cốt lõi của chương trình. Liệt kê 4-10 chương trình mã để tạo mạng cho chương trình.

```

#!/usr/bin/trần

nhập khẩu pefile
nhập sys
nhập argparse
nhập hệ điều hành
nhập pprint
nhập khẩu mạngx
nhập lại
từ networkx.drawing.nx_agraph nhập write_dot
bộ sưu tập nhập khẩu
từ networkx.algorithms nhập lưỡng cực

args = argparse.ArgumentParser(
"Trực quan hóa tên máy chủ được chia sẻ giữa một thư mục mẫu phần mềm độc hại"
)

```

```

args.add_argument("target_path",help="thư mục chứa các mẫu phần mềm độc hại")
args.add_argument("output_file",help="file để ghi tệp DOT vào")
args.add_argument("malware_projection",help="file cần ghi DOT vào")
args.add_argument("hostname_projection",help="file để ghi tệp DOT vào") args =
args.parse_args() network =
networkx.Graph()

valid_hostname_suffixes = map( lambda
string: string.strip(), open("domain_suffixes.txt") )

valid_hostname_suffixes = set(valid_hostname_suffixes)    def
find_hostnames(string): có
    thè_hostnames = re.findall( r'(?:[a-zA-Z0-9](?:[a-zA-Z0-9\-.]{,61}[a-zA-Z0-9])?\.)+[a-zA-Z]{2,6} ', chuỗi ) hợp lệ_hostnames =
bộ lọc ( 

    tên máy chủ lambda: hostname.split(".")[-1].lower() \ trong
    valid_hostname_suffixes,
    possible_hostnames

) trả lại tên máy chủ hợp lệ

# tìm kiếm thư mục đích để tìm các tệp thực thi Windows PE hợp lệ cho thư mục gốc,
thư mục, tệp trong os.walk(args.target_path): để tìm đường
dẫn trong tệp: # thủ
    mở tệp bằng pefile để xem nó có thực sự là tệp PE không, hãy thử: pe =
        pefile.PE(os.path.join(root,path)) ngoại trừ
        pefile.PEFormatError:
            Tiếp tục
        fullpath = os.path.join(root,path) #
        trích xuất các chuỗi có thể in được từ mẫu đích    strings =
os.popen("strings '{0}'".format(fullpath)).read()

        # sử dụng chức năng search_doc trong môđun reg để tìm tên
        máy chủ
        tên máy chủ = find_hostnames(chuỗi)
        if len(hostnames): #
            thêm các nút và cạnh cho mạng hai bên
            network.add_node(path,label=path[:32],color='black',penwidth=5, bipartite=0)
            cho tên máy
            chủ trong các tên máy chủ:
            network.add_node(hostname,label=hostname,color='blue',
                penwidth=10,bipartite=1)
            network.add_edge(hostname,path,penwidth=2) nếu tên
            máy chủ:
            in "Tên máy chủ được trích xuất từ:",đường dẫn
            pprint.pprint(tên máy chủ)

```

Liệt kê 4-10: Tạo mạng

Đầu tiên chúng ta tạo một mạng mới bằng cách gọi hàm tạo
`networkx.Graph()` . Sau đó, chúng tôi xác định hàm `find_hostnames()`, sẽ trích xuất

tên máy chủ từ chuỗi . Đừng lo lắng quá nhiều về cơ chế của chức năng này: về cơ bản, đó là một biểu thức chính quy và một số mã lọc chuỗi cố gắng hết sức để xác định các miền.

Tiếp theo, chúng tôi duyệt qua tất cả các tệp trong thư mục đích, kiểm tra xem chúng có phải là tệp PE hay không bằng cách xem lớp pefile.PE có thể tải chúng hay không (nếu không, chúng tôi sẽ không phân tích tệp). Cuối cùng, chúng tôi trích xuất các thuộc tính tên máy chủ từ tệp hiện tại bằng cách trước tiên trích xuất tất cả các chuỗi có thể in được từ tệp và sau đó tìm kiếm các chuỗi cho tài nguyên tên máy chủ được nhúng . Nếu chúng tôi tìm thấy bất kỳ nút nào, chúng tôi sẽ thêm chúng dưới dạng các nút trong mạng của mình và sau đó thêm các cạnh từ nút dành cho phần mềm độc hại hiện tại vào các nút tài nguyên tên máy chủ .

Bây giờ chúng ta đã sẵn sàng để kết thúc chương trình, như trong [Liệt kê 4-11](#).

```
# ghi tệp chấm vào đĩa
write_dot(mạng, args.output_file)
phân mềm độc hại = set(n for n,d in network.nodes(data=True) if d['bipartite']==0)
tên máy chủ = set(network)-malware

# sử dụng chức năng trình chiếu mạng lưỡng cực của NetworkX để tạo phần mềm độc hại
# và phép chiếu tên máy chủ
malware_network = bipartite.projected_graph(mạng, phân mềm độc hại)
hostname_network = bipartite.projected_graph(mạng, tên máy chủ)

# ghi các mạng dự kiến vào đĩa theo chỉ định của người dùng
write_dot(malware_network,args.malware_projection)
write_dot(hostname_network,args.hostname_projection)
```

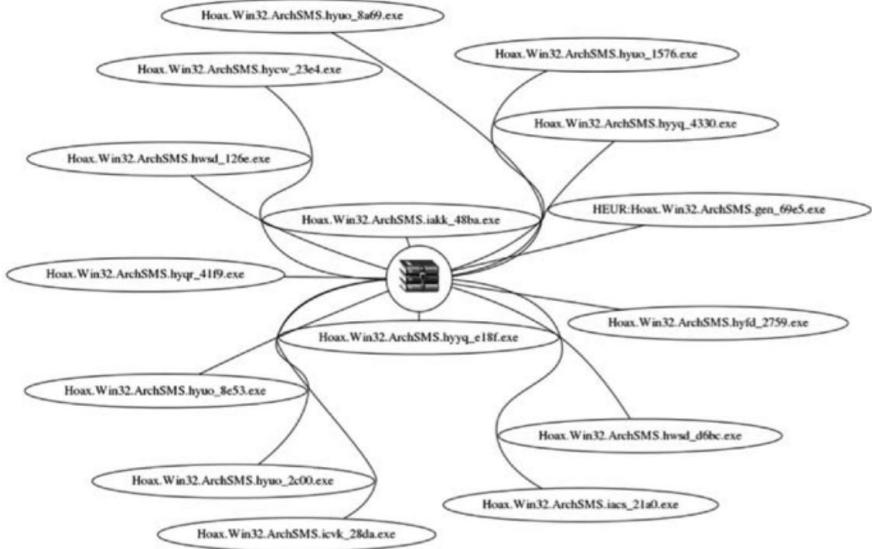
Liệt kê 4-11: Ghi mạng vào tệp

Chúng tôi bắt đầu bằng cách ghi mạng của chúng tôi vào đĩa tại vị trí được chỉ định trong các đối số dòng lệnh . Sau đó, chúng tôi tạo hai mạng rút gọn ("dự đoán" được giới thiệu trước đó trong chương này) hiển thị mối quan hệ phần mềm độc hại và mối quan hệ tài nguyên tên máy chủ. Chúng tôi thực hiện việc này bằng cách trước tiên tạo một bộ Python chứa ID của các nút phần mềm độc hại và một bộ Python khác chứa ID của các nút tài nguyên . Sau đó, chúng tôi sử dụng hàm `projected_graph()` cụ thể của NetworkX để nhận các phép chiếu cho bộ phần mềm độc hại và tài nguyên, đồng thời chúng tôi ghi các mạng này vào đĩa tại các vị trí đã chỉ định .

Và thế là xong! Bạn có thể sử dụng chương trình này trên bất kỳ bộ dữ liệu phần mềm độc hại nào trong cuốn sách này để xem mối quan hệ giữa phần mềm độc hại giữa các tài nguyên tên máy chủ dùng chung được nhúng trong các tệp. Bạn thậm chí có thể sử dụng nó trên bộ dữ liệu của riêng mình để xem bạn có thể thu thập thông tin về mối đe dọa nào thông qua chế độ phân tích này.

Xây dựng mạng lưới quan hệ hình ảnh được chia sẻ

Ngoài việc phân tích phần mềm độc hại dựa trên các máy chủ gọi lại được chia sẻ của họ, chúng tôi cũng có thể phân tích chúng dựa trên việc họ sử dụng các biểu tượng được chia sẻ và các nội dung đồ họa khác. Ví dụ, Hình 4-12 cho thấy một phần kết quả phân tích hình ảnh được chia sẻ cho các Trojan được tìm thấy trong ch4/data/Trojan.



Hình 4-12: Trực quan hóa mạng nội dung hình ảnh được chia sẻ cho một số Trojan

Bạn có thể thấy rằng tất cả những con ngựa thành Troy này đóng vai trò là các tệp lưu trữ và sử dụng cùng một biểu tượng tệp lưu trữ (hiển thị ở giữa hình), mặc dù chúng là các tệp thực thi. Việc họ sử dụng chính xác cùng một hình ảnh như một phần trong nỗ lực đánh lừa người dùng cho thấy rằng họ có thể đến từ cùng một kẻ tấn công. Tôi đã xác nhận điều này bằng cách chạy các mẫu phần mềm độc hại thông qua công cụ chống vi-rút của Kaspersky, công cụ này gán cho tất cả chúng cùng một họ (ArchSMS).

Tiếp theo, tôi sẽ chỉ cho bạn cách tạo kiểu trực quan hóa như trong Hình 4-12, để xem mối quan hệ hình ảnh được chia sẻ giữa các mẫu phần mềm độc hại. Để trích xuất hình ảnh từ phần mềm độc hại, chúng tôi sử dụng thư viện hình ảnh của trình trợ giúp, sau đó dựa vào wrestool (được thảo luận trong Chương 1) để tạo chương trình image_network.py . Nhớ lại rằng wrestool trích xuất hình ảnh từ các tệp thực thi của Windows.

Hãy xem qua quy trình tạo mạng hình ảnh được chia sẻ, bắt đầu với phần đầu tiên của mã, được hiển thị trong Liệt kê 4-12.

```

#!/usr/bin/trần

nhập khẩu pefile
nhập khẩu hệ thống
nhập argparse
nhập hệ điều hành
nhập pprint
nhập ký nhập khẩu
nhập khẩu mạngx
bộ sưu tập nhập khẩu
nhập tệp tạm thời

```

```

từ networkx.drawing.nx_agraph nhập write_dot từ
networkx.algorithms nhập luồng cực

# Sử dụng argparse để phân tích bất kỳ đối số dòng lệnh nào

args = argparse.ArgumentParser( "Trực
quan hóa mỗi quan hệ hình ảnh được chia sẻ giữa thư mục chứa mẫu phần mềm độc hại" )

args.add_argument("target_path",help="thư mục chứa mẫu phần mềm độc hại")
args.add_argument("output_file",help="file to write DOT vào")
args.add_argument("malware_projection",help="file ghi tệp DOT vào")
args.add_argument("resource_projection",help="file ghi tệp DOT vào") args = args.parse_args()
mạng = networkx.Graph()

lớp ExtractImages(): def
    __init__(self,target_binary):
        self.target_binary = target_binary
        self.image_basedir = Không có
        self.images = []

    def work(self):
        self.image_basedir = tempfile.mkdtemp() icondir
        = os.path.join(self.image_basedir,"icons") bitmapdir =
        os.path.join(self.image_basedir,"bitmaps") raw_resources =
        os .path.join(self.image_basedir,"raw") cho thư mục trong
        [icondir,bitmapdir,raw_resources]: os.mkdir(directory) rawcmd =
            "wrestool -x {0} -o {1}
2> \ /dev/

        null".format( self.target_binary,raw_resources )

        bmpcmd = "mv {0}/*.bmp {1} 2> /dev/


        null".format( raw_resources,bitmapdir ) icocmd =
            "icotool -x {0}/*.ico -o
            {1} \ 2> / dev/
        null".format( raw_resources,icondir )
đối với cmd trong [rawcmd,bmpcmd,icocmd]:
    hãy
        thử: os.system(cmd)
    except Exception,msg: vượt
        qua
    cho dirname trong [icondir,bitmapdir]: cho
        đường dẫn trong os.listdir(dirname):
            logging.info(path)
            path = os.path.join(dirname,path)
            imagehash = hash(open(path).read()) if
            path.endswith(".png"):
                self.images.append((path,imagehash)) if
            path.endswith(".bmp"):
                self.images.append((path,imagehash))

```

```

dọn dẹp def (tự):
    os.system("rm -rf {0}".format(self.image_basingir))

# tìm kiếm thư mục đích cho các tệp PE để trích xuất hình ảnh từ
image_objects = []
cho root,dirs,files trong os.walk(args.target_path):v
    cho đường dẫn trong tệp:
        # thử phân tích đường dẫn để xem đó có phải là tệp PE hợp lệ không
        thử:
            pe = pefile.PE(os.path.join(root,path))
        ngoại trừ pefile.PEFormatError:
            Tiếp tục

```

Liệt kê 4-12: Phân tích đối số ban đầu và mã tải tệp trong chương trình mạng hình ảnh được chia sẻ của chúng tôi

Chương trình bắt đầu giống như chương trình đồ thị tên máy chủ (bắt đầu từ Liệt kê 4-8) mà chúng ta vừa thảo luận. Đầu tiên, nó nhập một số mô-đun, bao gồm pefile và networkx. Tuy nhiên, ở đây, chúng tôi cũng định nghĩa lớp trình trợ giúp ExtractImages , mà chúng tôi sử dụng để trích xuất nội dung đồ họa từ các mẫu phần mềm độc hại mục tiêu. Sau đó, chương trình đi vào một vòng lặp trong đó chúng tôi lặp lại tất cả các tệp nhị phân của phần mềm độc hại mục tiêu .

Bây giờ chúng ta đang ở trong vòng lặp của mình, đã đến lúc trích xuất nội dung đồ họa từ các nhị phân phần mềm độc hại mục tiêu bằng cách sử dụng lớp ExtractImages (lớp này là lớp bao quanh các chương trình icoutils được thảo luận trong Chương 1). Liệt kê 4-13 cho thấy phần mã thực hiện điều này.

```

fullpath = os.path.join(root,path)
hình ảnh = ExtractImages(fullpath)
hình ảnh.work()
image_objects.append(hình ảnh)

# tạo mạng bằng cách liên kết các mẫu phần mềm độc hại với hình ảnh của chúng
cho đường dẫn, image_hash trong images.images:
    # đặt thuộc tính hình ảnh trên các nút hình ảnh để yêu cầu GraphViz # hiển thị hình
    # ảnh trong các nút này
    nếu không image_hash trong mạng:
        network.add_node(image_hash,image=path,label=' ',type='image')
    node_name = path.split("/")[-1]
    network.add_node(node_name,type="malware")
    network.add_edge(node_name,image_hash)

```

Liệt kê 4-13: Trích xuất nội dung đồ họa từ phần mềm độc hại mục tiêu

Đầu tiên, chúng tôi chuyển một đường dẫn đến tệp nhị phân phần mềm độc hại mục tiêu tới ExtractImages lớp , và sau đó chúng ta gọi phương thức work() của thẻ hiện kết quả là . Điều này dẫn đến lớp ExtractImages tạo một thư mục tạm thời để lưu trữ các hình ảnh của phần mềm độc hại, sau đó lưu trữ một tệp điển chứa dữ liệu về từng hình ảnh trong thuộc tính lớp hình ảnh .

Bây giờ chúng tôi có danh sách các hình ảnh được trích xuất từ ExtractImages, chúng tôi lặp lại nó , tạo một nút mạng mới cho một hình ảnh nếu chúng tôi chưa thấy hàm băm của nó trước đó và liên kết mẫu phần mềm độc hại hiện đang được xử lý với hình ảnh trong mạng .

Bây giờ chúng tôi đã tạo mạng các mẫu phần mềm độc hại được liên kết với hình ảnh mà chúng chưa, chúng ta sẵn sàng ghi biểu đồ vào đĩa, như trong Liệt kê 4-14.

```
# viết mạng lưỡng cực, sau đó thực hiện hai phép chiếu và viết chúng
write_dot(mạng, args.output_file)
phần mềm độc hại = set(n for n,d in network.nodes(data=True) if d['type']=='malware')
tài nguyên = bộ (mạng) - phần mềm độc hại
malware_network = bipartite.projected_graph(mạng, phần mềm độc hại)
resource_network = bipartite.projected_graph(mạng, tài nguyên)

write_dot(malware_network,args.malware_projection)
write_dot(resource_network,args.resource_projection)
```

Liệt kê 4-14: Ghi biểu đồ vào đĩa

Chúng ta làm điều này giống hệt như cách chúng ta đã làm trong Liệt kê 4-11. Đầu tiên, chúng tôi ghi mạng hoàn chỉnh vào đĩa , sau đó chúng tôi ghi hai hình chiếu (hình chiếu cho phần mềm độc hại và hình chiếu cho hình ảnh, mà chúng tôi gọi là tài nguyên ở đây) vào đĩa .

Bạn có thể sử dụng `image_network.py` để phân tích nội dung đồ họa trong bất kỳ bộ dữ liệu phần mềm độc hại nào trong cuốn sách này hoặc để trích xuất thông tin tình báo từ bộ dữ liệu phần mềm độc hại mà bạn chọn.

Bản tóm tắt

Trong chương này, bạn đã tìm hiểu về các công cụ và phương pháp cần thiết để thực hiện phân tích thuộc tính dùng chung biểu mẫu trên bộ dữ liệu phần mềm độc hại của riêng bạn. Cụ thể, bạn đã tìm hiểu cách các mạng, mạng hai bên và dự đoán mạng hai bên có thể giúp xác định các kết nối xã hội giữa các mẫu phần mềm độc hại, tại sao bộ lọc mạng lại là trung tâm của trực quan hóa mạng và cách thức hoạt động của các mạng định hướng cưỡng bức. Bạn cũng đã học cách tạo và trực quan hóa mạng phần mềm độc hại hoạt động bằng Python và các công cụ nguồn mở như NetworkX. Trong Chương 5, bạn sẽ tìm hiểu cách xây dựng mạng phần mềm độc hại dựa trên mối quan hệ mà được chia sẻ giữa các mẫu.

5

Chia sẻ d Code A na ly



Giả sử bạn đã phát hiện ra một mẫu phần mềm độc hại mới trên mạng của mình. Làm thế nào bạn sẽ bắt đầu phân tích nó? Bạn có thể gửi nó đến một trình quét chống vi-rút đa công cụ, chẳng hạn như VirusTotal để tìm hiểu xem nó thuộc họ phần mềm độc hại nào. Tuy nhiên, những kết quả như vậy thường không rõ ràng và mơ hồ, bởi vì các công cụ thường gắn nhãn phần mềm độc hại theo thuật ngữ chung chung như "tác nhân" không có ý nghĩa gì. Bạn cũng có thể chạy mẫu thông qua CuckooBox hoặc một số sandbox phần mềm độc hại khác để nhận báo cáo hạn chế về hành vi và máy chủ gọi lại của mẫu phần mềm độc hại.

Khi các phương pháp này không cung cấp đủ thông tin, bạn có thể cần thiết kế ngược mẫu. Ở giai đoạn này, phân tích mã được chia sẻ có thể cải thiện đáng kể quy trình làm việc của bạn. Bằng cách tiết lộ các mẫu đã phân tích trước đây tương tự với mẫu phần mềm độc hại mới và do đó tiết lộ mã mà chúng chia sẻ, phân tích mã được chia sẻ cho phép bạn sử dụng lại các phân tích trước đây của mình về phần mềm độc hại mới để bạn không phải bắt đầu lại từ đầu. Hiểu được nguồn gốc của phần mềm độc hại đã thấy trước đây này cũng có thể giúp bạn tìm ra ai có thể đã triển khai phần mềm độc hại đó.

Phân tích mã được chia sẻ, còn được gọi là phân tích tương tự, là quá trình chúng tôi so sánh hai mẫu phần mềm độc hại bằng cách ước tính tỷ lệ phần trăm mã nguồn biến dịch trước mà chúng chia sẻ. Nó khác với phân tích thuộc tính được chia sẻ, so sánh các mẫu phần mềm độc hại dựa trên các thuộc tính bên ngoài của chúng (ví dụ: biểu tượng trên màn hình mà chúng sử dụng hoặc máy chủ mà chúng gọi ra).

Trong kỹ thuật đảo ngược, phân tích mã được chia sẻ giúp xác định các mẫu có thể được phân tích cùng nhau (vì chúng được tạo từ cùng một bộ công cụ phân mềm độc hại hoặc là các phiên bản khác nhau của cùng một dòng phần mềm độc hại), có thể xác định xem liệu cùng một nhà phát triển có thể chịu trách nhiệm cho một nhóm mẫu phần mềm độc hại hay không.

Xem xét kết quả hiển thị trong Liệt kê 5-1, xuất phát từ một chương trình bạn sẽ xây dựng sau trong chương này để minh họa giá trị của phân tích mã chia sẻ phần mềm độc hại. Nó hiển thị các mẫu đã xem trước đó có thể chia sẻ mã với mẫu mới cũng như nhận xét về các mẫu cũ hơn đó.

Hiển thị các mẫu tương tự với WEBC2-GREENCAT_sample_E54CE5F0112C9FDDE86DB17E85A5E2C5 tên mẫu	mã chia sẻ
[*] WEBC2-GREENCAT_sample_55FB1409170C91740359D1D96364F17B	0,9921875
[*] GREENCAT_sample_55FB1409170C91740359D1D96364F17B	0,9921875
[*] WEBC2-GREENCAT_sample_E83F60FB0E0396EA309FAF0AED64E53F	0,984375

[binh luận] Mẫu này được xác định là chắc chắn đến từ vùng liên tục nâng cao nhón môi đe dọa được quan sát vào tháng 7 năm ngoái trên mạng Bờ Tây của chúng tôi
[*]GREENCAT_sample_E83F60FB0E0396EA309FAF0AED64E53F 0,984375

Liệt kê 5-1: Kết quả phân tích mã chia sẻ cơ bản

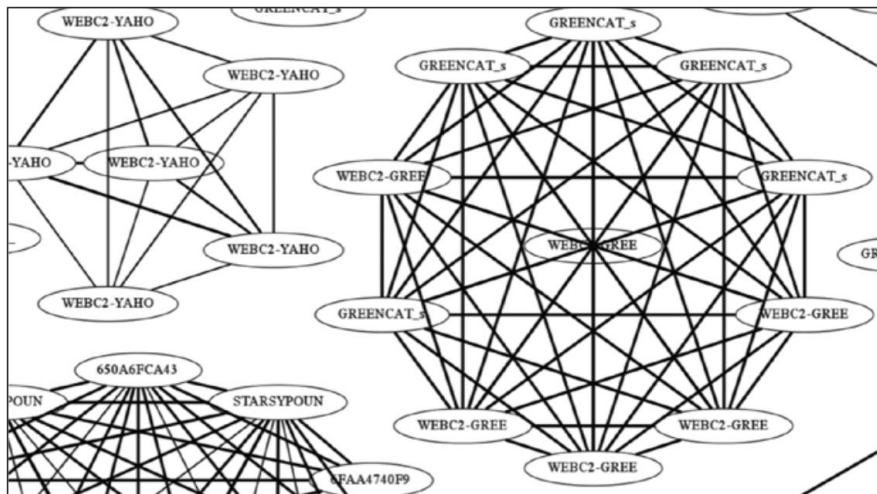
Đưa ra một mẫu mới, ước tính mã được chia sẻ cho phép chúng tôi xem, trong vòng vài giây, mẫu nào có khả năng chia sẻ mã và những gì chúng tôi biết về các mẫu đó.

Tương ví dụ này, nó tiết lộ rằng một mẫu rất giống nhau là từ một APT đã biết hoặc môi đe dọa liên tục nâng cao, do đó cung cấp bối cảnh ngay lập tức cho phần mềm độc hại mới này.

Chúng ta cũng có thể trực quan hóa các mối quan hệ mã được chia sẻ mẫu bằng cách sử dụng tính năng trực quan hóa mạng mà bạn đã học trong Chương 4. Ví dụ: Hình 5-1 cho thấy một mạng lưới các mối quan hệ mã được chia sẻ giữa các mẫu trong bộ dữ liệu môi đe dọa liên tục nâng cao.

Như bạn có thể thấy từ hình ảnh trực quan, các kỹ thuật phân tích mã được chia sẻ tự động có thể nhanh chóng phát hiện ra sự tồn tại của các họ phần mềm độc hại mà lẽ ra phải mất vài ngày hoặc vài tuần để phát hiện thông qua phân tích thủ công. Trong chương này, bạn sẽ học cách sử dụng các kỹ thuật này để thực hiện những việc sau:

- Xác định các họ phần mềm độc hại mới đến từ cùng một bộ công cụ phần mềm độc hại hoặc được viết bởi cùng những kẻ tấn công.
- Xác định sự giống nhau về mã giữa mẫu mới và mẫu đã thấy trước đó mẫu.
- Trực quan hóa các mối quan hệ của phần mềm độc hại để hiểu rõ hơn về các kiểu chia sẻ mã giữa các mẫu phần mềm độc hại và để truyền đạt kết quả của bạn cho người khác.
- Sử dụng hai công cụ chứng minh khái niệm mà tôi đã tạo cho cuốn sách này để triển khai những ý tưởng này và cho phép bạn xem các mối quan hệ mã được chia sẻ của phần mềm độc hại.



Hình 5-1: Một ví dụ về kiểu trực quan hóa mà bạn sẽ học cách tạo trong chương này, hiển thị các mối quan hệ mã được chia sẻ giữa một số mẫu APT1

Đầu tiên, tôi giới thiệu các mẫu phần mềm độc hại thử nghiệm mà bạn sẽ sử dụng trong chương này, đó là các mẫu PLA APT1 từ Chương 4 và một loạt các mẫu phần mềm tội phạm. Sau đó, bạn tìm hiểu về phép so sánh độ tương tự toán học và khái niệm về chỉ mục Jaccard, một phương pháp lý thuyết tập hợp để so sánh các mẫu phần mềm độc hại về các tính năng được chia sẻ của chúng. Tiếp theo, tôi giới thiệu khái niệm về các tính năng để chỉ ra cách bạn có thể sử dụng chúng trong mối liên kết với chỉ mục Jaccard để tính gần đúng lượng mã mà hai mẫu phần mềm độc hại chia sẻ. Bạn cũng tìm hiểu cách đánh giá các tính năng của phần mềm độc hại về mức độ hữu dụng của chúng. Cuối cùng, chúng tôi tạo trực quan hóa việc chia sẻ mã phần mềm độc hại ở nhiều quy mô, như trong Hình 5-1, bằng cách tận dụng kiến thức của bạn về trực quan hóa mạng từ Chương 4.

Các mẫu phần mềm độc hại được sử dụng trong chương này

Trong chương này, chúng tôi sử dụng các dòng phần mềm độc hại trong thế giới thực chia sẻ số lượng mã đáng kể với nhau để thực hiện các thử nghiệm của chúng tôi. Các bộ dữ liệu này có sẵn nhờ Mandiant và Mila Parkour, những người đã tuyển chọn các mẫu này và cung cấp chúng cho cộng đồng nghiên cứu. Tuy nhiên, trên thực tế, bạn có thể không biết mẫu phần mềm độc hại thuộc họ nào hoặc mẫu phần mềm độc hại mới của bạn giống với các mẫu đã thấy trước đó ở mức độ nào. Nhưng xem qua các ví dụ mà chúng tôi biết sẽ là một phương pháp hay, bởi vì nó cho phép chúng tôi xác minh rằng các suy luận tự động của chúng tôi về độ tương tự của mẫu phù hợp với kiến thức của chúng tôi về mẫu nào thực sự thuộc cùng một nhóm.

Các mẫu đầu tiên đến từ bộ dữ liệu APT1 mà chúng tôi đã sử dụng trong Chương 4 để minh họa phân tích tài nguyên dùng chung. Các mẫu khác bao gồm hàng ngàn

(còn tiếp)

về các mẫu phần mềm độc hại phần mềm tội phạm do bọn tội phạm phát triển để đánh cắp thẻ tín dụng của mọi người, biến máy tính của chúng thành máy chủ xác sống được kết nối với mạng botnet, v.v. Đây là các mẫu trong thế giới thực được lấy từ nguồn cấp dữ liệu phần mềm độc hại thương mại được cung cấp dưới dạng dịch vụ trả phí cho các nhà nghiên cứu tình báo mối đe dọa.

Để xác định họ của họ, tôi đã nhập từng mẫu vào Kaspersky công cụ chống vi-rút. Kaspersky có thẻ phân loại 30.104 mẫu trong số này bằng phân loại phân cấp mạnh mẽ (chẳng hạn như trojan.win32.jorik.skor.akr, biểu thị họ jorik.skor), gán một loại "không xác định" cho 41.830 mẫu và gán nhãn chung (chẳng hạn như, nói chung, "win32.Trojan") cho 28.481 mẫu còn lại.

Do nhãn Kaspersky không nhất quán (một số nhóm nhãn Kaspersky, chẳng hạn như họ jorik, đại diện cho một loạt phần mềm độc hại rất phổ biến, trong khi những nhóm khác, chẳng hạn như webprefix, đại diện cho một tập hợp các biến thể rất cụ thể) và thực tế là Kaspersky thường bỏ sót hoặc gán nhãn sai phần mềm độc hại, tôi đã chọn bảy loại phần mềm độc hại mà Kaspersky phát hiện với độ tin cậy cao. Cụ thể, chúng bao gồm các họ dapato, pasta, skor, vbna, webprefix, xtoober và zango.

Chuẩn bị mẫu để so sánh bằng cách trích xuất các tính năng

Làm thế nào để chúng ta thậm chí bắt đầu nghĩ về việc ước tính số lượng mã mà hai tệp nhị phân độc hại có thể đã chia sẻ trước khi chúng được những kẻ tấn công biên dịch? Có nhiều cách người ta có thể cân nhắc tiếp cận vấn đề này, nhưng trong hàng trăm tài liệu nghiên cứu khoa học máy tính đã được xuất bản về chủ đề này, một chủ đề chung đã xuất hiện: để ước tính lượng mã được chia sẻ giữa các tệp nhị phân, chúng tôi nhóm các mẫu phần mềm độc hại thành "túi tính năng" trước khi so sánh.

Theo tính năng, ý tôi là bất kỳ thuộc tính phần mềm độc hại nào mà chúng tôi có thể muốn xem xét khi ước tính mức độ giống nhau của mã giữa các mẫu. Ví dụ: các tính năng chúng tôi sử dụng có thể là các chuỗi có thể in được mà chúng tôi có thể trích xuất từ các tệp nhị phân. Thay vì coi các mẫu là một hệ thống chức năng được kết nối với nhau, nhập thư viện động, v.v., chúng tôi coi phần mềm độc hại là một nhóm các tính năng độc lập để thuận tiện về mặt toán học (ví dụ: một tập hợp các chuỗi đã được trích xuất từ phần mềm độc hại).

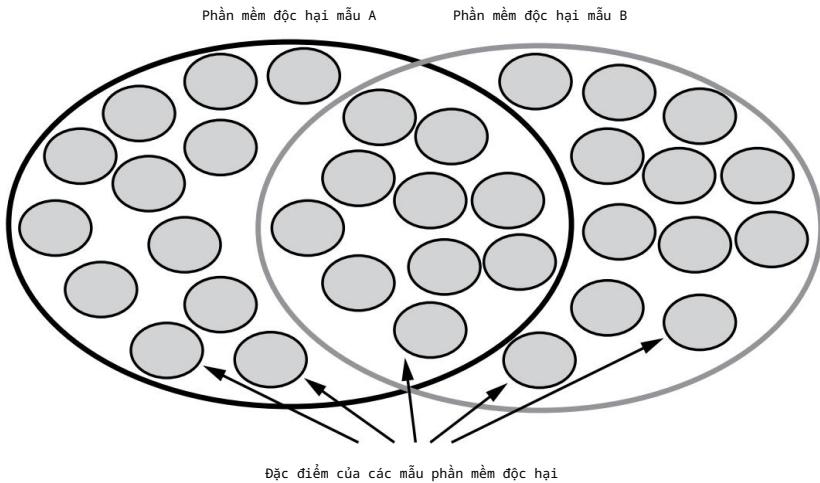
Mô hình túi tính năng hoạt động như thế nào

Để hiểu cách thức hoạt động của một nhóm tính năng, hãy xem xét biểu đồ Venn giữa hai mẫu phần mềm độc hại, như trong Hình 5-2.

Ở đây, mẫu A và mẫu B được hiển thị dưới dạng túi các tính năng (các tính năng được biểu diễn dưới dạng hình elip bên trong sơ đồ Venn). Chúng ta có thể so sánh chúng bằng cách kiểm tra những tính năng được chia sẻ giữa hai mẫu.

Việc tính toán sự trùng lặp giữa hai nhóm tính năng diễn ra nhanh chóng và có thể được sử dụng để so sánh sự giống nhau của các mẫu phần mềm độc hại dựa trên các tính năng tùy ý mà chúng tôi đưa ra.

Ví dụ: khi xử lý phần mềm độc hại được đóng gói, chúng tôi có thể muốn sử dụng các tính năng dựa trên nhật ký chạy động của phần mềm độc hại vì chạy phần mềm độc hại trong hộp cát là một cách để phần mềm độc hại tự giải nén. Trong các trường hợp khác, chúng tôi có thể sử dụng các chuỗi được trích xuất từ tệp nhị phân phần mềm độc hại tĩnh để thực hiện so sánh.



Hình 5-2: Minh họa mô hình “túi tính năng” để phân tích chia sẻ mã phần mềm độc hại

Trong trường hợp phân tích phần mềm độc hại động, chúng tôi có thể muốn so sánh các mẫu không chỉ dựa trên những hành vi chúng chia sẻ mà còn dựa trên thứ tự chúng thể hiện hành vi hoặc cái mà chúng tôi gọi là chuỗi hành vi của chúng. Một cách phổ biến để kết hợp thông tin trình tự vào so sánh mẫu phần mềm độc hại là mở rộng mô hình túi tính năng để chứa dữ liệu tuần tự bằng cách sử dụng N-gram.

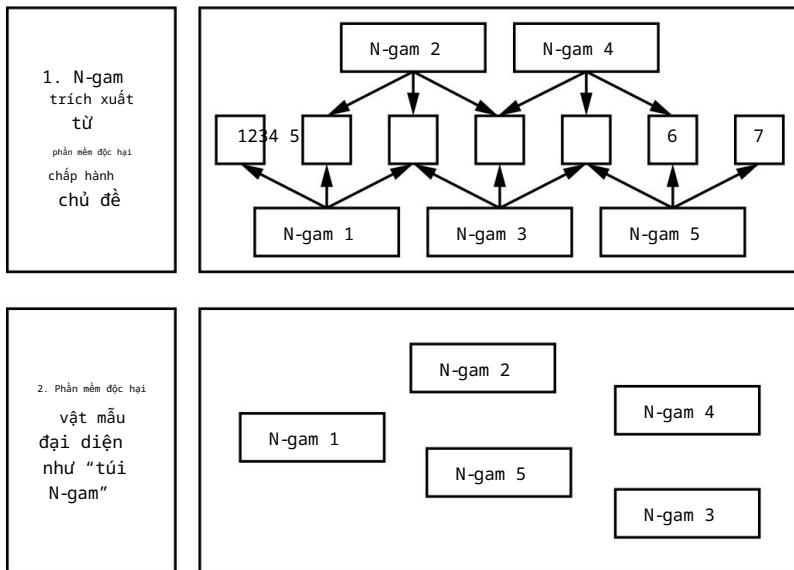
N-Gram là gì?

N -gram là một dãy con các sự kiện có độ dài nhất định, N, của một số chuỗi sự kiện lớn hơn. Chúng tôi trích xuất chuỗi con này từ một chuỗi lớn hơn bằng cách trượt một cửa sổ trên dữ liệu tuần tự. Nói cách khác, chúng ta nhận được N-gram bằng cách lặp qua một chuỗi và, ở mỗi bước, ghi lại chuỗi con từ sự kiện tại chỉ mục i đến sự kiện tại chỉ mục i + N - 1, như trong Hình 5-3.

Trong Hình 5-3, dãy số nguyên (1,2,3,4,5,6,7) được dịch thành năm dãy con khác nhau có độ dài 3: (1,2,3), (2,3,4), (3,4,5), (4,5,6), (5,6,7).

Tất nhiên, chúng ta có thể làm điều này với bất kỳ dữ liệu tuần tự nào. Ví dụ, sử dụng một Độ dài từ N-gram là 2, câu “how now brown cow” tạo ra các dãy con sau: “how now”, “now brown” và “brown cow”. Trong phân tích phần mềm độc hại, chúng tôi sẽ trích xuất N-gram lệnh gọi API tuần tự mà một mẫu phần mềm độc hại đã thực hiện. Sau đó, chúng tôi sẽ trình bày phần mềm độc hại dưới dạng một túi các tính năng và

sử dụng các tính năng N-gram để so sánh mẫu phần mềm độc hại với một số mẫu N-gram của phần mềm độc hại khác, từ đó kết hợp thông tin trình tự vào mô hình so sánh túi tính năng.



Hình 5-3: Giải thích trực quan về cách chúng tôi có thể trích xuất N-gram từ hướng dẫn lắp ráp của phần mềm độc hại và chuỗi lệnh gọi API động, trong đó $N = 3$

Việc đưa thông tin trình tự vào so sánh các mẫu phần mềm độc hại của chúng tôi có những ưu điểm và nhược điểm. Ưu điểm là khi thứ tự quan trọng trong so sánh (ví dụ: khi chúng tôi quan tâm rằng lệnh gọi API A đã được quan sát trước lệnh gọi API B, được quan sát trước lệnh gọi API C), nó cho phép chúng tôi nắm bắt thứ tự, nhưng khi thứ tự là thừa (ví dụ: phần mềm độc hại sắp xếp ngẫu nhiên thứ tự các lệnh gọi API A, B và C trên mỗi lần chạy), nó thực sự có thể làm cho ước tính mã được chia sẻ của chúng tôi trở nên tồi tệ hơn nhiều. Việc quyết định có đưa thông tin đơn đặt hàng vào công việc ước tính mã chia sẻ phần mềm độc hại hay không tùy thuộc vào loại phần mềm độc hại mà chúng tôi đang xử lý và yêu cầu chúng tôi thử nghiệm.

Sử dụng chỉ số Jaccard để định lượng tính tương đồng

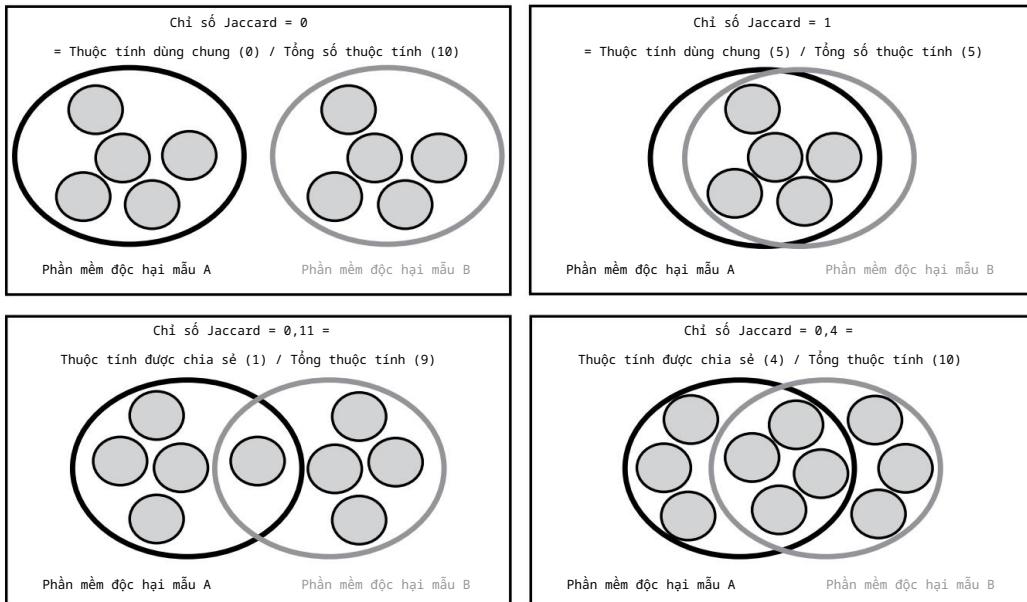
Khi bạn đã trình bày mẫu phần mềm độc hại dưới dạng một nhóm tính năng, bạn sẽ cần đo mức độ tương tự giữa nhóm tính năng của mẫu đó và một số nhóm tính năng của mẫu khác. Để ước tính mức độ chia sẻ mã giữa hai mẫu phần mềm độc hại, chúng tôi sử dụng hàm tương tự, hàm này phải có các thuộc tính sau:

- Nó mang lại một giá trị chuẩn hóa sao cho tất cả các so sánh tương tự giữa các cặp mẫu phần mềm độc hại có thể được đặt trên một thang điểm chung. Theo quy ước, hàm sẽ mang lại một giá trị nằm trong khoảng từ 0 (không chia sẻ mã) đến 1 (các mẫu chia sẻ 100 phần trăm mã của chúng).

- Hàm này sẽ giúp chúng ta ước lượng chính xác mã har giữa hai mẫu (chúng ta có thể xác định điều này theo kinh nghiệm thông qua thử nghiệm).
- Chúng ta có thể dễ dàng hiểu được lý do tại sao các mô hình chức năng mã tương tự tốt (nó không phải là một hộp đen toán học phức tạp cần nhiều nỗ lực để hiểu hoặc giải thích).

Chỉ mục Jaccard là một hàm đơn giản có các thuộc tính này. Trên thực tế, mặc dù các phương pháp toán học khác để ước tính độ tương tự của mã đã được thử nghiệm trong cộng đồng nghiên cứu bảo mật (ví dụ: khoảng cách cosine, khoảng cách L1, khoảng cách Euclidean [L2], v.v.), chỉ số Jaccard đã nổi lên như là chỉ số phổ biến nhất. thông qua và vì lý do chính đáng. Nó thể hiện một cách đơn giản và trực quan mức độ trùng lặp giữa hai bộ tính năng của phần mềm độc hại, cho chúng ta tỷ lệ phần trăm các tính năng duy nhất chung cho cả hai bộ được chuẩn hóa bằng tỷ lệ phần trăm các tính năng duy nhất tồn tại trong cả hai bộ.

Hình 5-4 minh họa các ví dụ về giá trị chỉ số Jaccard.



Hình 5-4: Minh họa trực quan về ý tưởng đằng sau chỉ số Jaccard

Hình này minh họa bốn cặp tính năng phần mềm độc hại được trích xuất từ bốn cặp mẫu phần mềm độc hại. Mỗi hình ảnh hiển thị các tính năng được chia sẻ giữa hai bộ, các tính năng không được chia sẻ giữa hai bộ và chỉ mục Jaccard kết quả cho cặp mẫu phần mềm độc hại nhất định và các tính năng liên quan. Bạn có thể thấy rằng chỉ số Jaccard giữa các mẫu đơn giản là số đặc điểm được chia sẻ giữa các mẫu chia cho tổng số đặc điểm được vẽ trong biểu đồ Venn.

Sử dụng ma trận tương tự để đánh giá mă chia sẻ phần mềm độc hại phương pháp ước tính

Hãy thảo luận về bốn phương pháp để xác định xem hai mẫu phần mềm độc hại có thuộc cùng một họ hay không: tính tương tự dựa trên trình tự hướng dẫn, tính tương tự dựa trên chuỗi, tính tương tự dựa trên Bảng địa chỉ nhập và tính tương tự dựa trên lệnh gọi API động. Để so sánh bốn phương pháp này, chúng tôi sẽ sử dụng kỹ thuật trực quan hóa ma trận tương tự. Mục tiêu của chúng ta ở đây sẽ là so sánh các điểm mạnh và điểm yếu tương đối của từng phương pháp về khả năng làm sáng tỏ các mối quan hệ mà được chia sẻ giữa các mẫu.

Để bắt đầu, chúng ta hãy xem qua khái niệm về ma trận tương tự. Hình 5-5 so sánh một bộ tưởng tượng gồm bốn mẫu phần mềm độc hại bằng cách sử dụng ma trận tương tự.

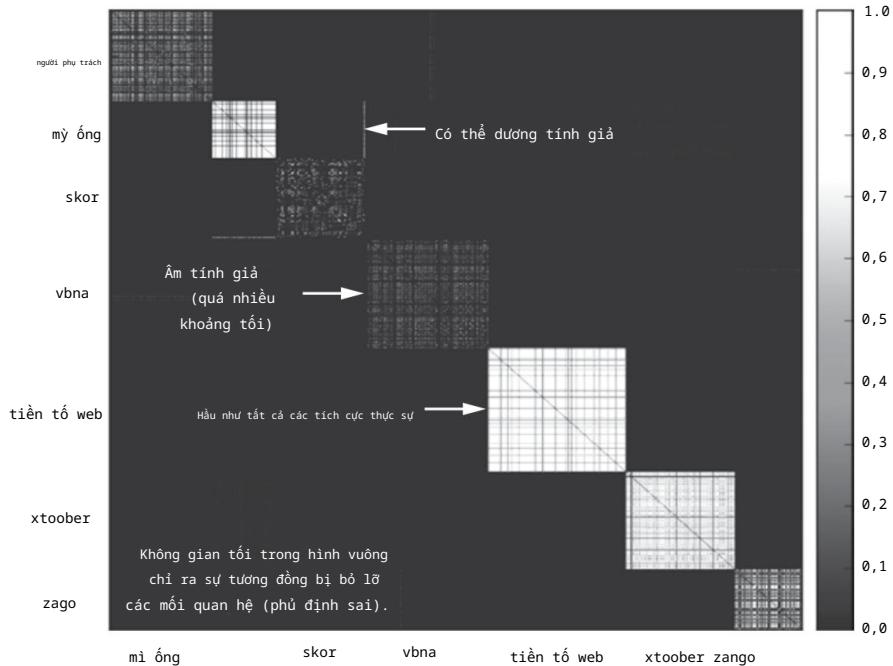
	mẫu 1	mẫu 2	mẫu 3	mẫu 4
mẫu 1	tương tự giữa 1 và 1	tương tự giữa 1 và 2	tương tự giữa 1 và 3	tương tự giữa 1 và 4
mẫu 2	tương tự giữa 2 và 1	tương tự giữa 2 và 2	tương tự giữa 2 và 3	tương tự giữa 2 và 4
mẫu 3	tương tự giữa 3 và 1	tương tự giữa 3 và 2	tương tự giữa 3 và 3	tương tự giữa 3 và 4
mẫu 4	tương tự giữa 4 và 1	tương tự giữa 4 và 2	tương tự giữa 4 và 3	tương tự giữa 4 và 4

Hình 5-5: Minh họa về ma trận tương tự danh nghĩa

Ma trận này cho phép bạn xem mối quan hệ tương tự giữa tất cả các mẫu. Bạn có thể thấy rằng số không gian bị lãng phí trong ma trận này. Ví dụ: chúng tôi không quan tâm đến những điểm tương đồng được thể hiện trong các hộp được tô bóng, vì các mục nhập này chỉ chứa các so sánh giữa một mẫu nhất định và chính nó. Bạn cũng có thể thấy rằng thông tin ở hai bên của các hộp được tô bóng được lặp lại, vì vậy bạn chỉ cần nhìn vào cái này hay cái kia.

Hình 5-6 đưa ra một ví dụ thực tế về ma trận tương tự phần mềm độc hại. Lưu ý rằng do số lượng lớn các mẫu phần mềm độc hại được hiển thị trong hình, mỗi giá trị tương tự được biểu thị bằng một pixel được tô bóng. Thay vì hiển thị tên của từng mẫu, chúng tôi hiển thị họ cho từng mẫu đọc theo trục ngang và trục dọc. Một ma trận tương tự hoàn hảo sẽ trông giống như một chuỗi các ô vuông màu trắng chạy theo đường chéo từ trên cùng bên trái sang

ở dưới cùng bên phải, vì các hàng và cột đại diện cho mỗi họ được nhóm lại với nhau và chúng tôi mong muốn tất cả các thành viên của một họ nhất định giống nhau, nhưng không phải mẫu từ các họ khác.



Hình 5-6: Ma trận tương tự phân mềm độc hại trong thế giới thực được tính toán trên bảy họ phần mềm độc hại

Trong các kết quả được đưa ra trong Hình 5-6, bạn có thể thấy rằng một số ô vuông họ có màu trắng hoàn toàn đây là những kết quả tốt, bởi vì các điểm ảnh màu trắng trong ô vuông họ biểu thị mối quan hệ tương tự được suy ra giữa các mẫu của cùng một họ. Một số tối hơn nhiều, điều đó có nghĩa là chúng tôi không phát hiện ra mối quan hệ tương đồng mạnh mẽ. Cuối cùng, đôi khi có các dòng pixel bên ngoài các ô vuông họ, đó là bằng chứng về các họ phần mềm độc hại có liên quan hoặc dương tính giả, nghĩa là chúng tôi đã phát hiện thấy việc chia sẻ mã giữa các họ mặc dù chúng vốn đã khác nhau.

Tiếp theo, chúng ta sẽ sử dụng trực quan hóa ma trận tương tự như Hình 5-6 để so sánh kết quả của bốn phương pháp ước tính chia sẻ mã khác nhau, bắt đầu bằng mô tả về phân tích tương tự dựa trên trình tự lệnh.

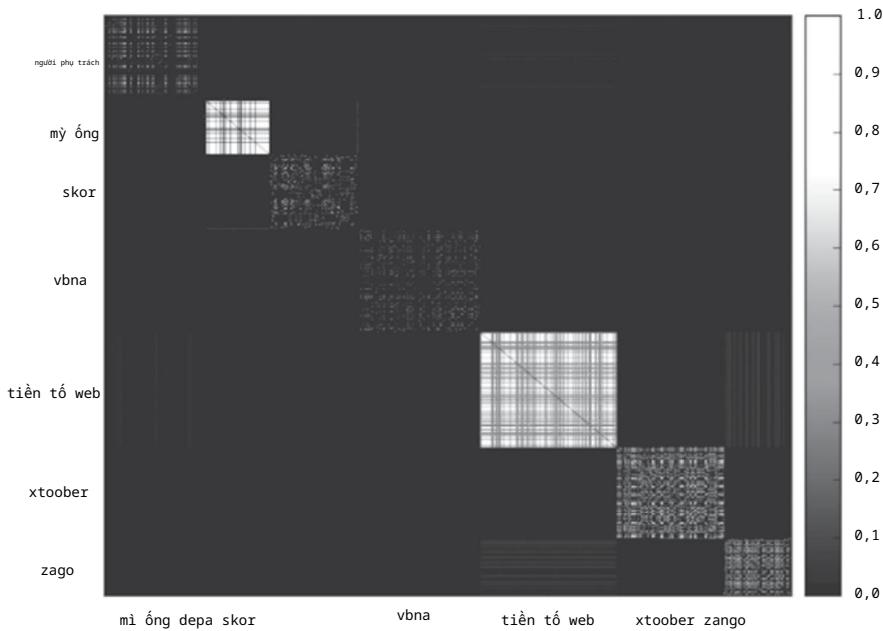
Tương tự dựa trên trình tự hướng dẫn

Cách trực quan nhất để so sánh hai mã nhị phân phần mềm độc hại về số lượng mã mà chúng chia sẻ là so sánh trình tự hướng dẫn lắp ráp x86 của chúng, vì các mẫu chia sẻ trình tự hướng dẫn có khả năng đã chia sẻ mã nguồn thực trước khi biên dịch. Điều này yêu cầu loại bỏ các mẫu phần mềm độc hại tương tự bằng cách sử dụng, chẳng hạn như kỹ thuật phân tách tuyến tính được giới thiệu trong Chương 2. Sau đó, chúng ta có thể sử dụng phương pháp trích xuất N-gram

phương pháp mà tôi đã thảo luận trước đây để trích xuất các chuỗi hướng dẫn theo thứ tự chúng xuất hiện trong phần .text của tệp phần mềm độc hại. Cuối cùng, chúng ta có thể sử dụng N-gram hướng dẫn để tính toán các chỉ số Jaccard giữa các mẫu để ước tính lượng mã mà chúng ta nghĩ rằng chúng chia sẻ.

Giá trị chúng tôi sử dụng cho N trong quá trình trích xuất N-gram phụ thuộc vào mục tiêu phân tích của chúng tôi. N càng lớn, các chuỗi hướng dẫn được trích xuất của chúng tôi sẽ càng lớn và do đó, các chuỗi của mẫu phần mềm độc hại sẽ càng khó khớp. Đặt N thành một số lớn giúp xác định chỉ các mẫu có khả năng chia sẻ mã với nhau cao. Một khác, bạn có thể làm cho N nhỏ hơn để tìm kiếm những điểm tương đồng tinh tế giữa các mẫu hoặc nếu bạn nghi ngờ rằng các mẫu sử dụng sắp xếp lại hướng dẫn để phân tích sự tương đồng che khuất.

Trong Hình 5-7, N được đặt thành 5, đây là cài đặt tích cực khiến các mẫu khó khớp hơn.



Hình 5-7: Ma trận tương tự được tạo bằng cách sử dụng các tính năng N-gram của lệnh. Sử dụng $N = 5$, chúng tôi hoàn toàn bỏ lỡ mối quan hệ tương tự của nhiều gia đình nhưng hoạt động tốt trên tienn_tu_web và mi_ong.

Các kết quả trong Hình 5-7 không hấp dẫn lắm. Mặc dù phân tích độ tương tự dựa trên hướng dẫn xác định chính xác các điểm tương đồng giữa một số lời nói dối fami, nhưng nó không nằm trong các họ khác (ví dụ: nó phát hiện một số mối quan hệ tương tự trong dapato, skor và vbna). Tuy nhiên, điều quan trọng cần lưu ý là có rất ít kết quả dương tính giả trong phân tích này (suy luận sai về sự giống nhau giữa các mẫu từ các họ khác nhau, so với suy luận thực sự về mối quan hệ tương tự trong các mẫu của cùng một họ).

Như bạn có thể thấy, một hạn chế của phân tích mã chia sẻ chuỗi hướng dẫn là nó có thể bỏ sót nhiều mối quan hệ chia sẻ mã giữa các mẫu.

Điều này là do các mẫu phần mềm độc hại có thể được đóng gói sao cho hầu hết các hướng dẫn của chúng chỉ hiển thị sau khi chúng tôi thực thi các mẫu phần mềm độc hại và để chúng tự giải nén. Nếu không giải nén các mẫu phần mềm độc hại của chúng tôi, phương pháp ước tính mã chia sẻ trình tự hướng dẫn có thể sẽ không hoạt động tốt.

Ngay cả khi chúng tôi giải nén các mẫu phần mềm độc hại, cách tiếp cận này có thể theo nguyên tắc, vì tiếng ồn được tạo ra bởi quá trình biên dịch mã nguồn.Ật vậy, các trình biên dịch có thể biên dịch cùng một mã nguồn thành các chuỗi hướng dẫn hợp ngữ hoàn toàn khác nhau. Lấy ví dụ, hàm đơn giản sau được viết bằng C:

```
int f(void) {
    int a = 1;
    int b = 2;
    bạn trả về (a*b)+3;
}
```

Bạn có thể nghĩ rằng bất kể trình biên dịch nào, chức năng sẽ giảm với cùng một trình tự hướng dẫn lắp ráp. Nhưng trên thực tế, quá trình biên dịch phụ thuộc rất nhiều không chỉ vào trình biên dịch bạn sử dụng mà còn vào cài đặt trình biên dịch. Ví dụ: biên dịch hàm này bằng trình biên dịch clang trong cài đặt mặc định của nó sẽ tạo ra các hướng dẫn sau tương ứng với dòng tại u trong mã nguồn:

```
movl    $1, -4(%rbp)
movl    $2, -8(%rbp)
movl    -4(%rbp), %eax
imull -8(%rbp), %eax
thêm $3, %eax
```

Ngược lại, biên dịch chức năng tương tự với bộ cờ -O3 , mà yêu cầu trình biên dịch tối ưu hóa mã cho tốc độ, tạo ra tập hợp sau cho cùng một dòng mã nguồn:

chuyển thể	\$5, %eax
------------	-----------

Sự khác biệt xuất phát từ thực tế là trong ví dụ thứ hai, trình biên dịch đã tính toán trước kết quả của hàm thay vì com đặt nó một cách rõ ràng, như trong ví dụ biên dịch đầu tiên. Điều này có nghĩa là nếu chúng ta so sánh các hàm này dựa trên các chuỗi hướng dẫn, thì chúng sẽ không giống nhau chút nào, mặc dù trên thực tế chúng được biên dịch từ cùng một mã nguồn.

Ngoài vấn đề mã C và C++ giống hệt nhau có vẻ rất khác khi chúng ta xem hướng dẫn lắp ráp của nó, còn có một vấn đề khác phát sinh khi chúng ta so sánh các tệp nhị phân dựa trên mã lắp ráp của chúng: nhiều tệp nhị phân phần mềm độc hại hiện được tạo ở mức cao ngôn ngữ ngôn ngữ như C#. Các tệp nhị phân này chứa mã hợp ngữ soạn sẵn tiêu chuẩn diễn giải đơn giản mã byte của các ngôn ngữ cấp cao hơn này. Vì vậy mặc dù

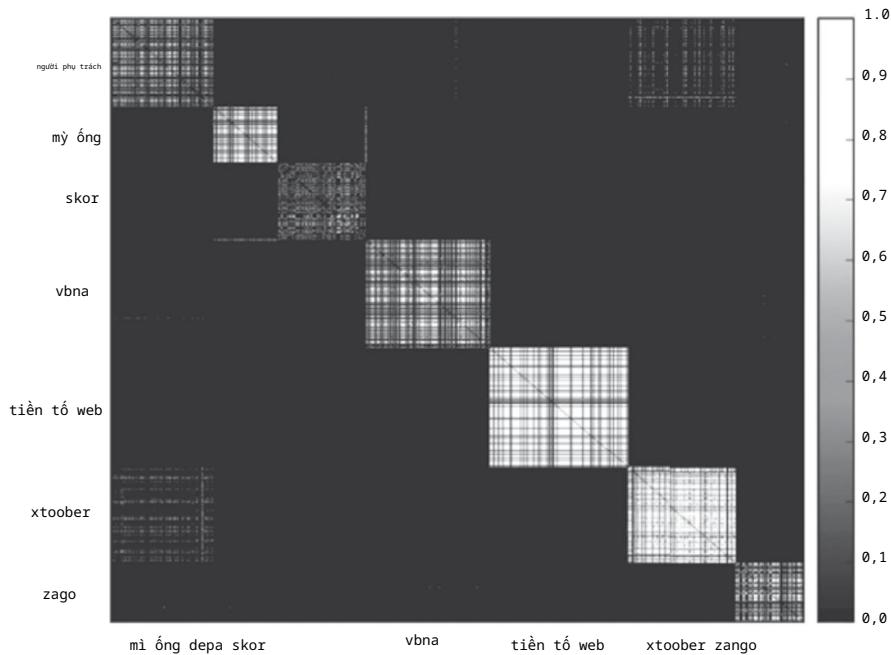
các tệp nhị phân được viết bằng cùng một ngôn ngữ cấp cao có thể chia sẻ các hướng dẫn x86 rất giống nhau, mã byte thực tế của chúng có thể phản ánh thực tế là chúng đến từ mã nguồn rất khác nhau.

Tương tự dựa trên chuỗi

Chúng tôi có thể tính toán độ tương tự của phần mềm độc hại dựa trên chuỗi bằng cách trích xuất tất cả các chuỗi ký tự có thể in liền kề trong các mẫu và sau đó tính toán chỉ mục Jaccard giữa tất cả các cặp mẫu phần mềm độc hại dựa trên mối quan hệ chuỗi được chia sẻ của chúng.

Cách tiếp cận này giải quyết vấn đề về trình biên dịch vì các chuỗi được trích xuất từ tệp nhị phân có xu hướng là các chuỗi định dạng do chương trình mer xác định, mà các trình biên dịch theo quy tắc chung không biến đổi, bất kể tác giả phần mềm độc hại đang sử dụng trình biên dịch nào hoặc chúng đưa ra các tham số nào. trình biên dịch. Ví dụ: một chuỗi điển hình được trích xuất từ tệp nhị phân phần mềm độc hại có thể đọc, "Trình ghi khóa đã bắt đầu lúc %s vào %s và thời gian %. " Bất kể cài đặt trình biên dịch, chuỗi này sẽ có xu hướng trông giống hệt nhau giữa nhiều tệp nhị phân và có liên quan đến việc chúng có dựa trên cùng một cơ sở mã nguồn hay không.

Hình 5-8 cho thấy chỉ số chia sẻ mã dựa trên chuỗi tốt như thế nào xác định đúng mối quan hệ chia sẻ mã trong tập dữ liệu phần mềm tội phạm.



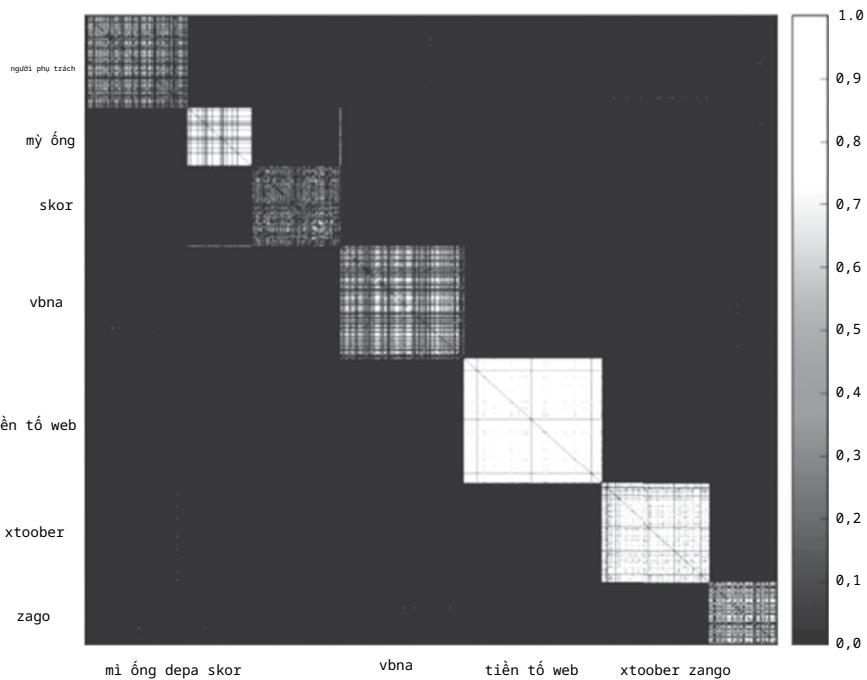
Hình 5-8: Ma trận độ tương tự được tạo bằng các tính năng chuỗi

Thoạt nhìn, phương pháp này xác định các họ phần mềm độc hại tốt hơn nhiều so với phương pháp dựa trên hướng dẫn, khôi phục chính xác phần lớn các mối quan hệ tương tự cho cả bảy họ. Tuy nhiên, không giống như các

phương pháp tương tự hướng dẫn, có một vài kết quả dương tính giả, vì nó dự đoán không chính xác rằng xtoober và dapato chia sẻ một số mức mã. Cũng cần lưu ý rằng phương pháp này không phát hiện ra sự tương đồng giữa các mẫu trong một số họ, đặc biệt kém hiệu quả đối với các họ zango, skor và dapato.

Nhập bảng địa chỉ dựa trên tính tương đồng

Chúng tôi có thể tính toán cái mà tôi gọi là "Độ tương tự dựa trên bảng địa chỉ nhập" bằng cách so sánh các lần nhập DLL được thực hiện bởi các nhị phân phần mềm độc hại. Ý tưởng đằng sau cách tiếp cận này là ngay cả khi tác giả phần mềm độc hại đã sắp xếp lại các hướng dẫn, làm xáo trộn phần dữ liệu khởi tạo của tệp nhị phân phần mềm độc hại và triển khai các kỹ thuật chống phân tích trình gõ lỗi và chống máy ảo, họ có thể đã để lại các khai báo nhập giống hệt nhau tại chỗ. Các kết quả cho phương pháp Nhập bảng địa chỉ được hiển thị trong Hình 5-9.



Hình 5-9: Ma trận tương tự được tạo bằng các tính năng của Bảng địa chỉ nhập

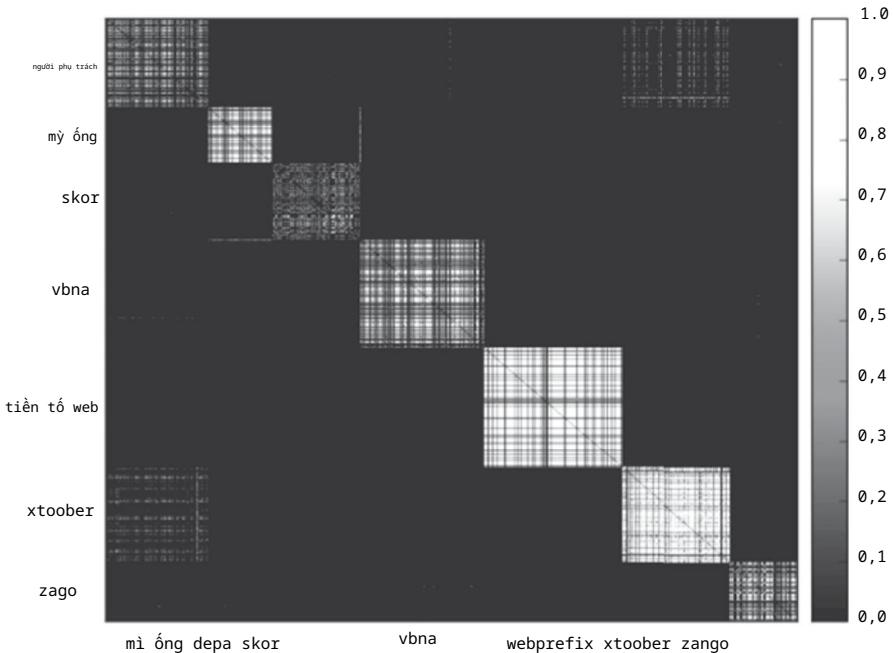
Hình này cho thấy phương pháp Bảng địa chỉ nhập khẩu hoạt động tốt hơn bất kỳ phương pháp nào trước đó trong việc ước tính mối quan hệ tương tự giữa các mẫu webprefix và xtoober và hoạt động rất tốt trên tất cả, mặc dù nó bỏ sót nhiều mối quan hệ skor, dapato và vbna .

Điều đáng chú ý là phương pháp này đưa ra một số kết quả dương tính giả trên tập dữ liệu thử nghiệm của chúng tôi.

Tương tự dựa trên lệnh gọi API động

Phương pháp so sánh cuối cùng mà tôi giới thiệu trong chương này là tính tương tự của phần mềm độc hại động. Ưu điểm của việc so sánh các chuỗi động là ngay cả khi các mẫu phần mềm độc hại bị xáo trộn hoặc đóng gói cực kỳ phức tạp, chúng sẽ có xu hướng thực hiện các chuỗi hành động tương tự trong một máy ảo có hộp cát miễn là chúng bắt nguồn từ cùng một mã hoặc mượn mã của nhau. Để triển khai phương pháp này, bạn sẽ cần chạy các mẫu phần mềm độc hại trong hộp cát và ghi lại lệnh gọi API mà chúng thực hiện, trích xuất N-gram lệnh gọi API từ nhật ký động và cuối cùng so sánh các mẫu bằng cách lấy chỉ số Jaccard giữa các túi của chúng của N-gram.

Hình 5-10 cho thấy phương pháp tương tự N-gram động không cũng như các phương thức nhập và chuỗi trong hầu hết các trường hợp.



Hình 5-10: Ma trận tương tự được tạo bằng cách sử dụng các tính năng N-gram của lệnh gọi API động

Các kết quả không hoàn hảo ở đây cho thấy rằng phương pháp này không phải là thuốc chữa bách bệnh. Chỉ chạy phần mềm độc hại trong sandbox là không đủ để kích hoạt nhiều hành vi của nó. Ví dụ: các biến thể của công cụ phần mềm độc hại động lệnh có thể hoặc không thể kích hoạt môđun mã quan trọng và do đó thực thi các chuỗi hành vi khác nhau, mặc dù chúng có thể chia sẻ hầu hết mã của mình.

Một vấn đề khác là một số mẫu phát hiện ra rằng chúng đang chạy trong một hộp cát và sau đó nhanh chóng thoát khỏi quá trình thực thi, để lại cho chúng tôi rất ít thông tin để so sánh. Tóm lại, giống như các cách tiếp cận tương tự khác mà tôi đã phác thảo, tương tự trình tự lệnh gọi API động không hoàn hảo, nhưng nó có thể cung cấp cái nhìn sâu sắc ấn tượng về sự tương đồng giữa các mẫu.

Xây dựng biểu đồ tương tự

Bây giờ bạn đã hiểu các khái niệm đằng sau các phương pháp xác định việc chia sẻ mã phần mềm độc hại, hãy xây dựng một hệ thống đơn giản thực hiện phân tích này trên tập dữ liệu phần mềm độc hại.

Trước tiên, chúng tôi cần ước tính lượng mã mà các mẫu chia sẻ bằng cách trích xuất các tính năng mà chúng tôi muốn sử dụng. Đây có thể là bất kỳ tính năng nào được mô tả trước đây, chẳng hạn như các hàm, chuỗi, N-gram dựa trên Bảng Địa chỉ Nhập, hoặc N-gram của hành vi động. Ở đây, chúng tôi sẽ sử dụng các tính năng chuỗi có thể in được vì chúng hoạt động tốt và dễ trích xuất và hiểu.

Khi chúng tôi đã trích xuất các tính năng chuỗi, chúng tôi cần lặp lại từng cặp mẫu phần mềm độc hại, so sánh các tính năng của chúng bằng cách sử dụng chỉ mục Jaccard. Sau đó, chúng ta cần xây dựng biểu đồ chia sẻ mã. Để làm điều này, trước tiên chúng ta cần quyết định ngưỡng xác định lượng mã mà hai mẫu chia sẻ – một giá trị tiêu chuẩn tôi sử dụng trong nghiên cứu của mình là 0,8. Nếu chỉ mục Jaccard cho một cặp mẫu phần mềm độc hại nhất định cao hơn giá trị đó, chúng tôi sẽ tạo một liên kết giữa chúng để trực quan hóa. Bước cuối cùng là nghiên cứu biểu đồ để xem mẫu nào được kết nối bởi các mối quan hệ mã được chia sẻ.

Liệt kê 5-2 đến 5-6 chứa chương trình mẫu của chúng tôi. Bởi vì danh sách dài, tôi chia nó thành nhiều phần và giải thích từng phần khi tôi tiếp tục. Liệt kê 5-2 nhập các thư viện mà chúng ta sẽ sử dụng và khai báo hàm jaccard() , hàm này tính toán chỉ mục Jaccard giữa hai bộ tính năng của mẫu.

```
#!/usr/bin/trần

nhập argparse
nhập hệ điều hành
nhập khâu mạngx
từ networkx.drawing.nx_pydot nhập write_dot
nhập itertools

def jaccard (set1, set2):

    Tính khoảng cách Jaccard giữa hai bộ bằng cách lấy
    giao điểm của chúng, hợp nhất và sau đó chia số
    của các phần tử trong giao điểm bằng số phần tử
    trong liên minh của họ.

    """
    giao lộ = set1.intersection(set2)
    ngã tư_length = float(len(ngã tư))
    công đoàn = set1.union(set2)
    union_length = float(len(union))
    trả lại giao lộ_length/union_length
```

Liệt kê 5-2: Hàm nhập và hàm trợ giúp để tính chỉ số Jaccard giữa hai mẫu

Tiếp theo, trong Liệt kê 5-3, chúng tôi khai báo hai hàm tiện ích bổ sung: getstrings(), tìm tập hợp các chuỗi chuỗi có thể in được trong các tập phần mềm độc hại mà chúng tôi sẽ phân tích và pecheck(), đảm bảo mục tiêu đó

các tệp thực sự là các tệp Windows PE. Chúng tôi sẽ sử dụng các chức năng này sau khi chúng tôi thực hiện trích xuất tính năng trên các nhị phân phần mềm độc hại mục tiêu.

```
def getstrings (đường dẫn đầy đủ):
```

Trích xuất các chuỗi từ tệp nhị phân được chỉ định bởi tham số 'đường dẫn đầy đủ', sau đó trả về tập hợp các chuỗi duy nhất trong tệp nhị phân.

```
chuỗi = os.popen("strings '{0}'".format(fullpath)).read() strings =
set(strings.split("\n")) trả về chuỗi
```

```
def pecheck (đường dẫn đầy đủ):
```

Thực hiện kiểm tra sơ bộ để đảm bảo rằng 'fullpath' là một tệp thực thi của Windows PE (các tệp thực thi PE bắt đầu bằng haj, byte 'MZ')

```
trả về open(fullpath).read(2) == "MZ"
```

Liệt kê 5-3: Khai báo các hàm chúng ta sẽ sử dụng trong trích xuất tính năng

Tiếp theo, trong Liệt kê 5-4, chúng ta phân tích các đối số dòng lệnh của người dùng. Các đối số này bao gồm thư mục đích chứa phần mềm độc hại mà chúng tôi sẽ phân tích, tệp .dot đầu ra mà chúng tôi sẽ viết mạng mã chia sẻ mà chúng tôi xây dựng và ngưỡng chỉ mục Jaccard, xác định chỉ số Jaccard phải cao như thế nào giữa hai mẫu để chương trình quyết định rằng chúng chia sẻ cơ sở mã chung với nhau.

```
Nếu __name__ == "__main__":
    parser =
        argparse.ArgumentParser( description="Xác định điểm tương đồng giữa các mẫu phần mềm độc hại và xây dựng biểu đồ"
    )

    parser.add_argument( "thư
        mục đích", help="Thư
        mục chứa phần mềm độc hại"
    )

    parser.add_argument( "output_dot_file", help="Nơi lưu tệp DOT biểu đồ đầu ra"
    )

    parser.add_argument( "--jaccard_index_threshold", "-j", dest="threshold", type=float, default=0.8,
        help="Ngưỡng trên để tạo 'cạnh' giữa các mẫu"
    )

    args = trình phân tích cú pháp.parse_args()
```

Liệt kê 5-4: Phân tích cú pháp đối số dòng lệnh của người dùng

Tiếp theo, trong Liệt kê 5-5, chúng ta sử dụng các hàm trợ giúp mà chúng ta đã khai báo trước đó để thực hiện công việc chính của chương trình: tìm các nhị phân PE trong thư mục đích, trích xuất các tính năng từ chúng và khởi tạo một mạng mà chúng ta sẽ sử dụng để biểu diễn quan hệ tương tự giữa các nhị phân.

```

malware_paths = [] # nơi chúng tôi sẽ lưu trữ đường dẫn tệp phần mềm độc hại
malware_features = dict() # nơi chúng tôi sẽ lưu trữ các chuỗi phần mềm độc hại
graph = networkx.Graph() # biểu đồ tương tự

cho thư mục gốc, thư mục, đường dẫn trong os.walk(args.target_directory):
    # duyệt cây thư mục đích và lưu trữ tất cả các đường dẫn
    # cho đường dẫn trong đường dẫn:
        full_path = os.path.join(root, đường dẫn)
        phần mềm độc hại_paths.append(full_path)

    # lọc ra bất kỳ đường dẫn nào không phải là tệp PE
    malware_paths = bộ lọc(pecheck, malware_paths)

    # lấy và lưu trữ các chuỗi cho tất cả các tệp PE của phần mềm độc hại
    # cho đường dẫn trong malware_paths:
        tính năng = getstrings(đường dẫn)
        in "Trích xuất {0} tính năng từ {1} ...".format(len(features), đường dẫn)
        malware_features[path] = tính năng

    # thêm từng tệp phần mềm độc hại vào biểu đồ
    graph.add_node(đường dẫn, nhãn=os.path.split(đường dẫn)[-1][:10])

```

Liệt kê 5-5: Trích xuất các tính năng từ tệp PE trong thư mục đích và khởi tạo mạng mã chia sẻ

Sau khi trích xuất các tính năng từ các mẫu mục tiêu của mình, chúng tôi cần lặp lại từng cặp mẫu phần mềm độc hại, so sánh các tính năng của chúng bằng cách sử dụng chỉ mục Jaccard. Chúng tôi làm điều này trong Liệt kê 5-6. Chúng tôi cũng xây dựng biểu đồ chia sẻ mã trong đó các mẫu được liên kết với nhau nếu chỉ số Jaccard của chúng vượt quá một số ngưỡng do người dùng xác định. Ngưỡng mà tôi thấy hoạt động tốt nhất trong nghiên cứu của mình là 0,8.

```

# lặp qua tất cả các cặp phần mềm độc hại
đối với phần mềm độc hại1, phần mềm độc hại2 trong itertools.combinations(malware_paths, 2):

    # tính toán khoảng cách jaccard cho cặp hiện tại
    jaccard_index = jaccard(malware_features[malware1], malware_features[malware2])

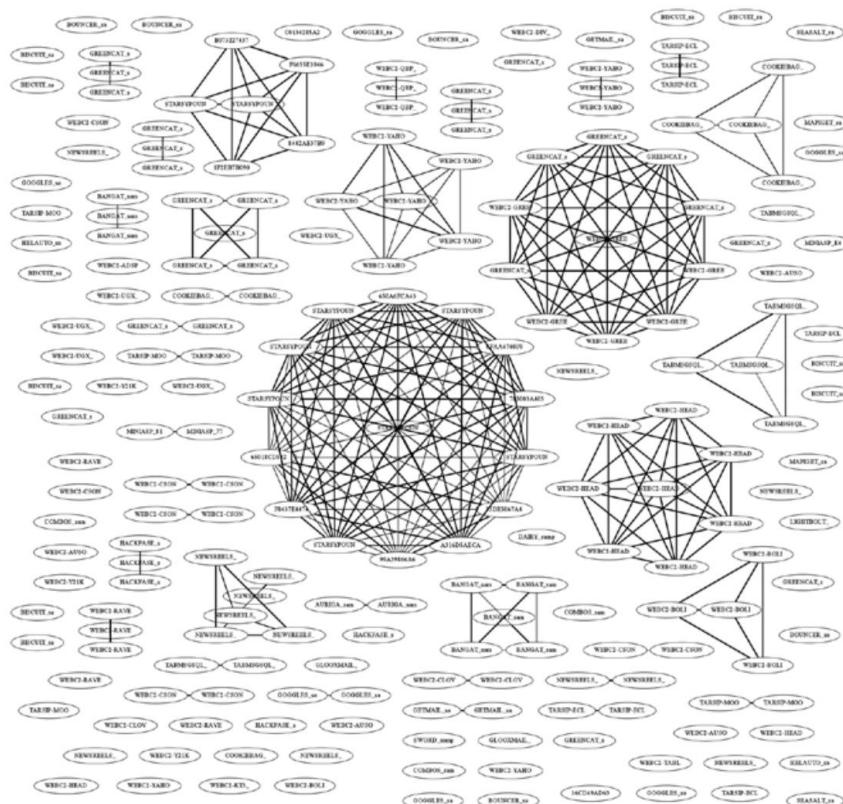
    # nếu khoảng cách jaccard vượt quá ngưỡng, hãy thêm một cạnh
    # nếu jaccard_index > args.threshold:
        in phần mềm độc hại1, phần mềm độc hại2, jaccard_index
        graph.add_edge(malware1, malware2, penwidth=1+(jaccard_index-args.threshold)*10)

    # ghi biểu đồ vào file để chúng ta có thể hình dung nó
    write_dot(đồ thị, args.output_dot_file)

```

Liệt kê 5-6: Tạo biểu đồ chia sẻ mã trong Python

Mã trong Liệt kê 5-2 đến 5-6 tạo ra biểu đồ như trong Hình 5-11 khi được áp dụng cho các mẫu phần mềm độc hại APT1. Để trực quan hóa biểu đồ, bạn cần sử dụng công cụ fdp Graphviz (đã thảo luận trong Chương 4) để nhập lệnh fdp -Tpng network.dot -o network.png.



Hình 5-11: Biểu đồ tương tự hoàn chỉnh dựa trên chuỗi cho các mẫu APT1

Điều đáng kinh ngạc về kết quả này là trong vòng vài phút, chúng tôi đã sao chép phần lớn công việc thủ công, tỉ mỉ mà các phân tích ban đầu về APT1 đã tạo ra trong báo cáo của họ, xác định nhiều họ phần mềm độc hại được sử dụng bởi những kẻ tấn công cấp quốc gia này.

Chúng tôi biết rằng phương pháp của chúng tôi đã thực hiện chính xác so với công việc kỹ thuật đảo ngược thủ công mà các nhà phân tích này đã thực hiện, bởi vì tên trên các nút là tên do các nhà phân tích phân tích Mandiant đặt cho chúng. Bạn có thể thấy điều này trong cách các mẫu có tên tương tự nhau trong trực quan hóa mạng trong Hình 5-11, chẳng hạn như các mẫu "STARSPOUN" trong vòng tròn trung tâm. Vì phần mềm độc hại trong trực quan hóa mạng của chúng tôi tự động nhóm lại với nhau theo cách phù hợp với các họ này nên phương pháp của chúng tôi có vẻ "đồng ý" với các nhà phân tích phần mềm độc hại của Mandiant.

Bạn có thể mở rộng mã trong Liệt kê 5-2 đến 5-6 và áp dụng nó cho phần mềm độc hại của riêng bạn để có được thông tin tình báo tương tự.

Mở rộng so sánh tương đồng

Mặc dù mã trong Liệt kê 5-2 đến 5-6 hoạt động tốt đối với các bộ dữ liệu phần mềm độc hại nhỏ, nhưng nó không hoạt động tốt đối với một số lượng lớn các mẫu phần mềm độc hại. Điều này là do so sánh tất cả các cặp mẫu phần mềm độc hại trong tập dữ liệu tăng theo phương trình bậc hai với số lượng mẫu. Cụ thể, phương trình sau đây đưa ra số lần tính toán chỉ số Jaccard cần thiết để tính toán một ma trậnity tương tự trên tập dữ liệu có kích thước n:

$$\frac{n^2 -}{2}$$

Ví dụ: hãy quay lại ma trận tương tự trong Hình 5-5 để xem chúng ta cần bao nhiêu chỉ số Jaccard để tính bốn mẫu.

Thoạt nhìn, bạn có thể nói 16 (42), vì đó là số ô trong ma trận. Tuy nhiên, vì tam giác dưới cùng của ma trận chứa các phần trùng lặp của tam giác trên cùng của ma trận, nên chúng ta không cần tính hai lần này. Điều này có nghĩa là chúng ta có thể trừ 6 khỏi tổng số phép tính của mình. Hơn nữa, chúng tôi không cần so sánh các mẫu phần mềm độc hại với chính chúng, vì vậy chúng tôi có thể loại bỏ đường chéo khỏi ma trận, cho phép chúng tôi trừ thêm bốn lần tính toán.

Số phép tính cần thiết như sau:

$$\frac{4^2 - 4}{2} = \frac{16 - 4}{2} = 6$$

Điều này dường như có thể quản lý được, cho đến khi tập dữ liệu của chúng tôi tăng lên 10.000 mẫu phần mềm độc hại, chẳng hạn, sẽ yêu cầu 49.995.000 phép tính. Một bộ dữ liệu có 50.000 mẫu sẽ yêu cầu 1.249.975.000 phép tính chỉ số Jaccard!

Để mở rộng quy mô so sánh độ tương tự của phần mềm độc hại, chúng ta cần sử dụng các thuật toán xấp xỉ so sánh ngẫu nhiên. Ý tưởng cơ bản là cho phép một số lỗi trong tính toán so sánh của chúng tôi để đổi lấy việc giảm thời gian tính toán. Đối với mục đích của chúng tôi, một phương pháp so sánh gần đúng được gọi là minhash phục vụ mục đích này một cách tuyệt vời. Phương pháp minhash cho phép chúng tôi tính toán chỉ số Jaccard bằng phép tính gần đúng để tránh tính toán sự tương đồng giữa các mẫu phần mềm độc hại không giống nhau dưới một số ngữ cảnh tương tự được xác định trước để chúng tôi có thể phân tích các mối quan hệ mã được chia sẻ giữa hàng triệu mẫu.

Tước khi bạn đọc về lý do tại sao minhash hoạt động, hãy lưu ý rằng đây là một thuật toán phức tạp có thể mất một chút thời gian để hiểu. Nếu bạn quyết định bỏ qua phần "Minhash in Depth", chỉ cần đọc phần "Minhash in a Nutshell" và sử dụng mã được cung cấp, bạn sẽ không gặp vấn đề gì khi mở rộng phân tích chia sẻ mã của mình.

Tóm tắt Minhash

Minhash lấy các tính năng của mẫu phần mềm độc hại và băm chúng bằng hàm băm. Đối với mỗi hàm băm, chúng tôi chỉ giữ lại giá trị tối thiểu của

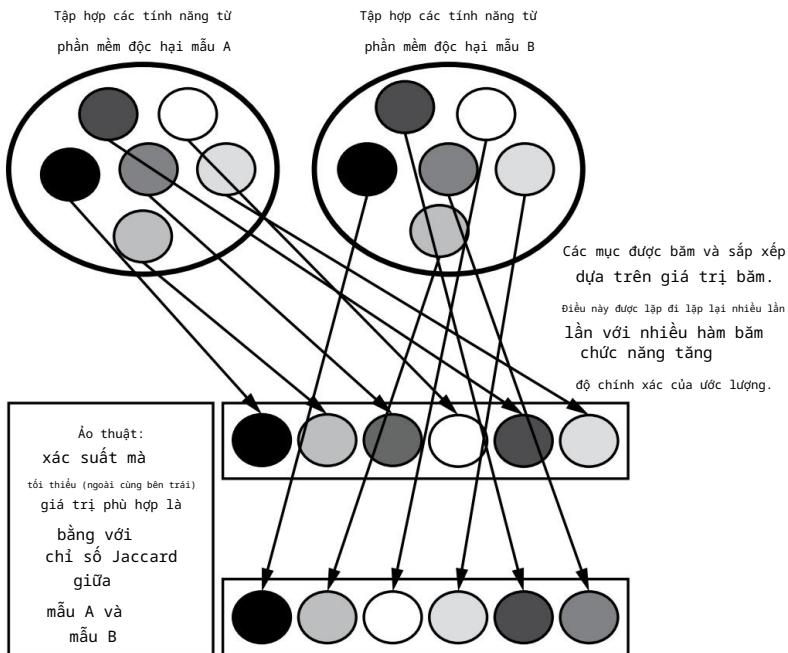
các hàm băm được tính toán trên tất cả các tính năng, do đó tập hợp các tính năng của phần mềm độc hại được giảm xuống thành một mảng có kích thước cố định gồm k số nguyên mà chúng tôi gọi là minhash. Để tính toán chỉ số Jaccard gần đúng giữa hai mẫu dựa trên mảng minhash của chúng, bây giờ bạn chỉ cần kiểm tra xem có bao nhiêu k minhash khớp và chia cho k.

Thật kỳ diệu, con số nằm ngoài các tính toán này là một giá trị gần đúng của chỉ số Jaccard thực giữa hai mẫu bất kỳ. Lợi ích của việc sử dụng minhash thay vì tính toán chỉ số Jaccard theo nghĩa đen là tính toán nhanh hơn nhiều.

Trên thực tế, chúng tôi thậm chí có thể sử dụng minhash để lập chỉ mục phần mềm độc hại một cách khéo léo trong cơ sở dữ liệu sao cho chúng tôi chỉ cần tính toán so sánh giữa các mẫu phần mềm độc hại có khả năng giống nhau vì ít nhất một trong số các giá trị băm của chúng khớp với nhau, do đó tăng tốc đáng kể việc tính toán các điểm tương đồng trong bộ dữ liệu phần mềm độc hại.

Minhash chuyên sâu

Bây giờ chúng ta hãy thảo luận sâu về toán học đằng sau minhash. Hình 5-12 cho thấy các bộ tính năng (được biểu thị bằng các vòng tròn tông bóng) cho hai mẫu phần mềm độc hại, cách chúng được băm và sau đó được sắp xếp dựa trên giá trị băm của chúng và cách chúng được so sánh cuối cùng dựa trên giá trị của phần tử đầu tiên của mỗi mẫu danh sách.



Hình 5-12: Minh họa ý tưởng đằng sau minhash

Xác suất mà các phần tử đầu tiên sẽ khớp bằng với chỉ số Jaccard giữa các mẫu. Cách thức hoạt động của nó nằm ngoài phạm vi của cuốn sách này, nhưng thực tế tình cờ này là thứ cho phép chúng ta tính gần đúng chỉ số Jaccard bằng cách sử dụng hàm bam .

Tất nhiên, việc chỉ thực hiện thao tác bam , sắp xếp và kiểm tra phần tử đầu tiên này sẽ không cho chúng ta biết nhiều nếu chúng ta chỉ thực hiện một lần—các giá trị bam khớp hoặc không khớp và chúng ta không thể đoán chỉ mục Jaccard cơ bản rất chính xác dựa trên một trận đấu đó. Để có ước tính tốt hơn về giá trị cơ bản này, chúng ta phải sử dụng k hàm bam và lặp lại thao tác này k lần, sau đó ước tính chỉ số Jaccard bằng cách chia số lần các phần tử đầu tiên này khớp với k. Lỗi dự kiến của chúng tôi trong việc ước tính chỉ số Jaccard được xác định như sau:

$$\frac{1}{\sqrt{k}}$$

Vì vậy, chúng tôi thực hiện quy trình này càng nhiều lần, chúng tôi càng chắc chắn hơn (Tôi có xu hướng đặt k thành 256 để ước tính trung bình giảm 6 phần trăm).

Giả sử chúng ta tính toán một mảng minhash cho mọi mẫu phần mềm độc hại trong tập dữ liệu phần mềm độc hại chứa một triệu mẫu. Làm cách nào để chúng tôi sử dụng minhashes để tăng tốc độ tìm kiếm các họ phần mềm độc hại trong bộ dữ liệu? Chúng tôi có thể lặp lại từng cặp mẫu phần mềm độc hại trong bộ dữ liệu và so sánh các mảng minhash của chúng, điều này sẽ dẫn đến 499.999.500.000 con so sánh. Mặc dù việc so sánh các mảng minhash nhanh hơn so với tính toán chỉ số Jaccard, nhưng đây vẫn là quá nhiều so sánh để thực hiện trên phần cứng hiện đại. Chúng tôi cần một số cách khai thác minhash để tối ưu hóa quá trình so sánh hơn nữa.

Cách tiếp cận tiêu chuẩn cho vấn đề này là sử dụng phác thảo kết hợp với lập chỉ mục cơ sở dữ liệu, tạo ra một hệ thống trong đó chúng tôi chỉ so sánh các mẫu mà chúng tôi đã biết là có khả năng tương tự cao. Chúng tôi tạo một bản phác thảo bằng cách bam nhiều minhash lại với nhau.

Khi chúng tôi nhận được một mẫu mới, chúng tôi sẽ kiểm tra xem cơ sở dữ liệu có chứa bất kỳ bản phác thảo nào khớp với các bản phác thảo của mẫu mới hay không. Nếu vậy, mẫu mới được so sánh với các mẫu phù hợp bằng cách sử dụng mảng minhash của chúng để tính gần đúng chỉ số Jaccard giữa mẫu mới và các mẫu tương tự cũ hơn.

Điều này giúp tránh phải so sánh mẫu mới với tất cả các mẫu trong cơ sở dữ liệu và thay vào đó chỉ so sánh mẫu đó với những mẫu có nhiều khả năng có chỉ số Jaccard cao với mẫu mới này.

Xây dựng hệ thống tìm kiếm tương tự phần mềm độc hại liên tục

Bây giờ bạn đã biết được những ưu và nhược điểm của việc sử dụng nhiều loại tính năng phần mềm độc hại khác nhau để ước tính mối quan hệ mà được chia sẻ giữa các mẫu phần mềm độc hại.

Bạn cũng đã tìm hiểu về chỉ mục Jaccard, mà trận đấu tương tự và cách minhash có thể tạo ra sự tương đồng về điện toán giữa các mẫu phần mềm độc hại trong các bộ dữ liệu thậm chí rất lớn có thể kiểm tra được. Với tất cả kiến thức này trong tay, bạn hiểu tất cả các khái niệm cơ bản cần thiết để xây dựng một hệ thống tìm kiếm mã chia sẻ phần mềm độc hại có thể mở rộng.

Liệt kê 5-7 đến 5-12 cho thấy một ví dụ về một hệ thống đơn giản trong đó tôi lập chỉ mục các mẫu phần mềm độc hại dựa trên các tính năng chuỗi của chúng. Trong công việc của mình, bạn nên cảm thấy tự tin khi sửa đổi hệ thống này để sử dụng các tính năng phần mềm độc hại khác hoặc mở rộng nó để hỗ trợ nhiều tính năng trực quan hơn. Vì danh sách dài nên tôi đã chia nhỏ nó ra và chúng ta sẽ lần lượt đề cập đến từng tiêu mục.

Để bắt đầu, Liệt kê 5-7 nhập các gói Python cần thiết cho chương trình.

```
#!/usr/bin/trần
```

```
nhập argparse
nhập hệ điều hành
tiếng thi thảm nhập khẩu
ké nhập khẩu
nhập numpy dưới dạng np
từ danh sách_5_2_dến_5_6 nhập *
```

```
NUM_MINHASHES = 256
SKETCH_RATIO = 8
```

Liệt kê 5-7: Nhập các mô-đun Python và khai báo các hằng số liên quan đến minhash

Ở đây, tôi nhập các gói như thi thảm, ké và sim_graph. Ví dụ: thi thảm là một thư viện băm mà chúng tôi sử dụng để tính toán thuật toán minhash mà tôi vừa thảo luận. Chúng tôi sử dụng giá đỡ, một mô-đun cơ sở dữ liệu đơn giản có trong thư viện chuẩn Python, để lưu trữ thông tin về các mẫu và minhash của chúng, mà chúng tôi sử dụng để tính toán các điểm tương đồng. Chúng tôi sử dụng danh sách_5_2_to_5_6.py để nhận các hàm tính toán độ tương tự của mẫu.

Chúng ta cũng khai báo hai hằng số trong Liệt kê 5-7: NUM_MINHASHES và SKETCH_TỶ LỆ. Chúng tương ứng với số minhash và tỷ lệ minhash với các bản phác thảo mà chúng tôi tính toán cho từng mẫu. Hãy nhớ rằng chúng ta càng sử dụng nhiều minhash và bản phác thảo thì các tính toán về độ tương tự của chúng ta càng chính xác. Ví dụ: 256 minhash và tỷ lệ 8:1 (32 bản phác thảo) là đủ để mang lại độ chính xác chấp nhận được với chi phí tính toán thấp.

Liệt kê 5-8 thực hiện chức năng cơ sở dữ liệu mà chúng tôi sử dụng để khởi tạo kích thước, truy cập và xóa cơ sở dữ liệu giá đỡ mà chúng tôi sử dụng để lưu trữ thông tin mẫu phần mềm độc hại.

```
def wipe_database():
```

Sự cố này sử dụng cơ sở dữ liệu 'giá đỡ' của thư viện tiêu chuẩn python để tồn tại thông tin, lưu trữ cơ sở dữ liệu trong tệp 'samples.db' trong cùng một thư mục dưới dạng tập lệnh Python thực tế. 'wipe_database' xóa tệp này khi lập lại hệ thống một cách hiệu quả.

```
dbpath = "/".join(__file__.split('/')[-1:-1] + ['samples.db'])
os.system("rm -f {}".format(dbpath))
```

```
def get_database():
```

Hàm trợ giúp để truy xuất cơ sở dữ liệu 'giá đỡ', đây là một cách đơn giản lưu trữ giá trị khóa.

```
"""
dbpath = "/".join(__file__.split('/')[-1:-1] + ['samples.db'])
return shelve.open(dbpath, protocol=2, writeback=True)
```

Liệt kê 5-8: Các hàm trợ giúp cơ sở dữ liệu

Chúng tôi xác định `wipe_database()` để xóa cơ sở dữ liệu của chương trình trong trường hợp chúng tôi muốn xóa sạch thông tin mẫu mà chúng tôi đã lưu trữ và bắt đầu lại. Sau đó, chúng tôi xác định `get_database()` để mở cơ sở dữ liệu của mình, tạo cơ sở dữ liệu nếu nó chưa tồn tại, sau đó trả về một đối tượng cơ sở dữ liệu, cho phép chúng tôi lưu trữ và truy xuất dữ liệu về các mẫu phần mềm độc hại của mình.

Liệt kê 5-9 triển khai một đoạn mã cót lõi để phân tích mã dùng chung của chúng ta: `minhash`.

```
def minhash (tính năng):
```

Đây là nơi xảy ra phép thuật `minhash`, tính toán cả `minhash` của các tính năng của mẫu và bản phác thảo của các `minhash` đó. Số lượng `minhashes` và bản phác thảo được tính toán được kiểm soát bởi `NUM_MINHASHES` và `NUM_SKETCHES` biến toàn cục được khai báo ở đầu tập lệnh.

```
minhashes = []
phác thảo = []
cho tôi trong phạm vi(NUM_MINHASHES):
    minhashes.append(
        min([murmur.string_hash(`feature`, i) cho tính năng trong các tính năng]))
)
cho tôi trong xrange(0,NUM_MINHASHES,SKETCH_RATIO):
    sketch = thi thảm.string_hash(`minashes[i:i+SKETCH_RATIO]`)
    bản phác thảo.append( bản phác thảo)
trả lại np.array(minashes), bản phác thảo
```

Liệt kê 5-9: Lấy `minhash` và bản phác thảo cho một mẫu

Chúng tôi lặp `NUM_MINHASHES` lần và nối thêm một giá trị `minhash`. Mỗi giá trị `minhash` được tính bằng cách băm tất cả các tính năng và sau đó lấy giá trị băm tối thiểu. Để thực hiện tính toán này, chúng tôi sử dụng hàm `string_hash()` của gói `ri` rầm tuổi để băm các tính năng, sau đó chúng tôi lấy giá trị mệ nhỏ của danh sách băm bằng cách gọi hàm `min()` của Python .

Đối số thứ hai của `string_hash` là một giá trị hạt giống, làm cho hàm băm ánh xạ tới các giá trị băm khác nhau tùy thuộc vào giá trị của hạt giống.

Bởi vì mỗi giá trị `minhash` yêu cầu một hàm băm duy nhất sao cho tất cả các giá trị băm 256 phút của chúng tôi không giống nhau, trên mỗi lần lặp, chúng tôi thêm hàm `string_hash` với giá trị bộ đếm `i`, khiến các tính năng ánh xạ tới các giá trị băm khác nhau trên mỗi lần lặp.

Sau đó, chúng tôi lặp lại các `minhash` mà chúng tôi đã tính toán và sử dụng các `minhash` để tính toán các bản phác thảo . Hãy nhớ rằng các bản phác thảo là hàm băm của nhiều `minhash` mà chúng tôi sử dụng để lập chỉ mục cơ sở dữ liệu cho các mẫu phần mềm độc hại để chúng tôi có thể nhanh chóng truy xuất các mẫu có khả năng giống nhau bằng cách truy vấn cơ sở dữ liệu. Trong danh sách mã tiếp theo, chúng tôi

lặp qua tất cả các minhash mẫu của chúng tôi với kích thước bước SKETCH_RATIO, băm từng đoạn băm khi chúng tôi di lấp bản phác thảo của mình. Cuối cùng, chúng tôi sử dụng hàm string_hash của gói riêng để băm các minhash lại với nhau .

Liệt kê 5-10 sử dụng get_database() từ Liệt kê 5-8, hàm getstrings () từ mô-đun sim_graph mà chúng ta đã nhập, và hàm minhash() từ Liệt kê 5-9 để tạo một hàm đánh chỉ mục các mẫu vào cơ sở dữ liệu của hệ thống.

```
def store_sample(path):
    """Chức năng lưu trữ một mẫu và minhashes và phác thảo của nó trong
    'kết' cơ sở dữ liệu

    db = get_database()
    tính năng = getstrings(path)
    minhashes,sketches = minhash(features)

    cho sketch trong sketch:
        phác thảo = str(phác thảo)
        nếu không sketch trong db:
            db[sketch] = thiết lập([path])
        khác:
            obj = db[bản phác thảo]
            obj.add(path)
            db[sketch] = obj
    db[path] = {'minhashes':minhashes,'comments':[]}
    db.sync()

    in "Trích xuất {0} tính năng từ {1} ...".format(len(features),path)
```

Liệt kê 5-10: Lưu trữ minhash của một mẫu trong cơ sở dữ liệu giá bằng cách sử dụng các bản phác thảo của nó làm khóa

Chúng tôi gọi get_database() , getstrings() , và minhash() và sau đó lặp lại các bản phác thảo mẫu của chúng tôi bắt đầu từ . Tiếp theo, để lập chỉ mục các mẫu của chúng tôi trong cơ sở dữ liệu, chúng tôi sử dụng một kỹ thuật được gọi là lặp chỉ mục đảo ngược, cho phép chúng tôi lưu trữ các mẫu dựa trên giá trị phác thảo của chúng thay vì ID. Cụ thể hơn, đổi với mỗi giá trị trong số 32 giá trị bản phác thảo của một mẫu, chúng tôi tra cứu bản ghi của bản phác thảo đó trong cơ sở dữ liệu và nối ID mẫu của chúng tôi vào danh sách các mẫu được liên kết với bản phác thảo đó. Ở đây, chúng tôi sử dụng đường dẫn hệ thống tệp của mẫu làm ID của nó.

Bạn có thể thấy điều này được triển khai như thế nào trong mã: chúng tôi lặp lại các phác thảo mà chúng tôi đã tính toán cho một mẫu , chúng tôi tạo một bản ghi cho phác thảo nếu nó chưa tồn tại (liên kết mẫu của chúng tôi với phác thảo trong khi chúng tôi đang tại nó) , và cuối cùng, chúng tôi thêm đường dẫn mẫu vào tập hợp các đường dẫn mẫu được liên kết của bản phác thảo nếu bản ghi của bản phác thảo tồn tại .

Liệt kê 5-11 cho thấy phân khai báo của hai hàm quan trọng: comment_sample() và search_sample().

```
def comment_sample(path):
```

Chức năng cho phép người dùng nhận xét về một mẫu. Nhận xét mà người dùng cung cấp sẽ hiển thị bất cứ khi nào mẫu này được nhìn thấy trong danh sách các mẫu tương tự với một số mẫu mới, cho phép người dùng sử dụng lại kiến thức về cơ sở dữ liệu phần mềm độc hại của họ.

```

db = get_database()
comment = raw_input("Nhập nhận xét của bạn:") nếu
không phải là đường
dẫn trong db:
store_sample(path) comments = db[path]
['comments']
comments.append(comment) db[path]
['comments']
'] = nhận xét db.sync() in "Nhận xét đã lưu:", nhận xét

def search_sample(đường dẫn):
    Hàm tìm kiếm các mẫu tương tự như mẫu được cung cấp bởi đối số 'đường dẫn',
    liệt kê các nhận xét, tên tệp và giá trị tương tự của chúng

    db = get_database()
    features = getstrings(path)
    minhashes, sketches = minhash(features)
    neighbours = []

    cho sketch trong sketch:
        sketch = str(sketch)

        nếu không phác thảo trong db:
            Tiếp tục

            cho Neighbor_path trong db[sketch]:
                neighbor_minhashes = db[neighbor_path]['minhashes'] tương tự
                = (neighbor_minhashes == minhashes).sum() / float(NUM_MINHASHES)

                neighbour.append((neighbor_path, tương tự))

    hàng xóm = danh sách (bộ (hàng xóm))
    neighbours.sort(key=lambda entry:entry[1], reverse=True)
    print
    print "Sample name".ljust(64), "Chia sẻ mã ước tính" cho hàng
    xóm, sự giống nhau giữa các hàng xóm:
        short_neighbor = neighbor.split("/)[-1]
        comments = db[neighbor]['comments'] print
        str(["*"] +short_neighbor).ljust(64), sự giống nhau của comment
        trong comment: print "
            \t[bình luận]", bình luận

```

Liệt kê 5-11: Khai báo các hàm cho phép người dùng nhận xét về mẫu và tìm kiếm mẫu tương tự như mẫu truy vấn

Như mong đợi, `comment_sample()` thêm bản ghi nhận xét do người dùng xác định vào bản ghi cơ sở dữ liệu của mẫu. Chức năng này rất hữu ích vì nó cho phép người dùng chương trình đưa thông tin chi tiết thu được từ kỹ thuật đảo ngược một mẫu vào cơ sở dữ liệu để khi họ nhìn thấy một mẫu mới giống với mẫu mà họ có nhận xét, họ có thể tận dụng những nhận xét đó để hiểu nhanh hơn nguồn gốc và mục đích của mẫu mới.

Tiếp theo, `search_sample()` tận dụng minhash để tìm các mẫu tương tự như một mẫu truy vấn. Để thực hiện việc này, trước tiên, chúng tôi trích xuất các tính năng chuỗi, minhash và bản phác thảo từ mẫu truy vấn. Sau đó, chúng tôi lặp lại các bản phác thảo của mẫu, tra cứu các mẫu được lưu trữ trong cơ sở dữ liệu cũng có bản phác thảo đó. Đối với mỗi mẫu chia sẻ một bản phác thảo với mẫu truy vấn, chúng tôi tính toán chỉ số Jaccard gần đúng của nó bằng cách sử dụng minhash của nó. Cuối cùng, chúng tôi báo cáo các mẫu tương tự nhất với mẫu truy vấn cho người dùng, cùng với mọi nhận xét liên quan đến các mẫu này đã được lưu trữ trong cơ sở dữ liệu.

Liệt kê 5-12 kết thúc mã chương trình của chúng ta bằng cách triển khai phần phân tích cú pháp đối số trong chương trình của chúng ta.

```

nếu __name__ == '__main__':
    trình phân tích cú pháp = argparse.ArgumentParser(
        mô tả="""
Hệ thống tìm kiếm chia sẻ mã đơn giản cho phép bạn xây dựng cơ sở dữ liệu về
các mẫu phần mềm độc hại (được lập chỉ mục theo đường dẫn tệp) và
sau đó tìm kiếm các mẫu tương tự với một số mẫu mới
    )

    trình phân tích cú pháp.add_argument(
        "-l", "--load", dest="load", default=None,
        help="Đường dẫn đến thư mục phần mềm độc hại hoặc tập tin để lưu trữ trong cơ sở dữ liệu"
    )

    trình phân tích cú pháp.add_argument(
        "-s", "-search", dest="search", default=None,
        help="Tệp phần mềm độc hại riêng lẻ để thực hiện tìm kiếm tương tự trên"
    )

    trình phân tích cú pháp.add_argument(
        "-c", "--comment", dest="comment", default=None,
        help="Nhận xét về đường dẫn mẫu phần mềm độc hại"
    )

    trình phân tích cú pháp.add_argument(
        "-w", "--wipe", action="store_true", default=False,
        help="Xóa cơ sở dữ liệu mẫu"
    )

args = trình phân tích cú pháp.parse_args()
nếu args.load:
    malware_paths = [] # nơi chúng tôi sẽ lưu trữ đường dẫn tệp phần mềm độc hại
    malware_features = dict() # nơi chúng tôi sẽ lưu trữ các chuỗi phần mềm độc hại

```

```

    cho root, thư mục, đường dẫn trong os.walk(args.load):
        # duyệt cây thư mục đích và lưu trữ tất cả các đường dẫn tệp
        cho đường dẫn trong đường dẫn:
            full_path = os.path.join(root, path)
            phần mềm độc hại_paths.append(full_path)

        # lọc ra bất kỳ đường dẫn nào không phải là tệp PE
        malware_paths = bộ lọc(pecheck, malware_paths)

        # lấy và lưu trữ các chuỗi cho tất cả các tệp PE của phần mềm độc hại
        cho đường dẫn trong malware_paths:
            store_sample(đường dẫn)

nếu args.search:
    search_sample(args.search)

nếu args.comment:
    comment_sample(args.comment)

nếu args.wipe:
    wipe_database()

```

Liệt kê 5-12: Thực hiện các truy vấn và cập nhật cơ sở dữ liệu tương tự dựa trên các đối số dòng lệnh của người dùng

Tại đây, chúng tôi cho phép người dùng tải các mẫu phần mềm độc hại vào cơ sở dữ liệu để các mẫu này sẽ được so sánh với các mẫu phần mềm độc hại mới khi người dùng tìm kiếm các mẫu tương tự trong cơ sở dữ liệu . Tiếp theo, chúng tôi cho phép người dùng tìm kiếm các mẫu tương tự như mẫu mà người dùng đã chuyển vào , in kết quả ra thiết bị đầu cuối. Chúng tôi cũng cho phép người dùng nhận xét về các mẫu đã có trong cơ sở dữ liệu . Cuối cùng, chúng tôi cho phép người dùng xóa cơ sở dữ liệu hiện có .

Chạy hệ thống tìm kiếm tương đồng

Khi bạn đã triển khai mã này, bạn có thể chạy hệ thống tìm kiếm tương tự, bao gồm bốn thao tác đơn giản:

Tải Tải các mẫu vào hệ thống sẽ lưu trữ chúng trong cơ sở dữ liệu hệ thống để tìm kiếm chia sẻ mã trong tương lai. Bạn có thể tải các mẫu riêng lẻ hoặc chỉ định một thư mục mà hệ thống sẽ tìm kiếm để quy các tệp PE, tải chúng vào cơ sở dữ liệu. Bạn có thể tải các mẫu vào cơ sở dữ liệu bằng lệnh sau chạy trong thư mục mã của chương này:

danh sách python_5_7_to_5_12.py -l <đường dẫn đến thư mục hoặc mẫu phần mềm độc hại riêng lẻ>

Nhận xét Nhận xét về một mẫu rất hữu ích vì nó cho phép bạn lưu trữ kiến thức về mẫu đó. Ngoài ra, khi bạn nhìn thấy các mẫu mới tương tự với mẫu này, tìm kiếm tương tự trên các mẫu đó sẽ hiển thị các nhận xét bạn đã thực hiện trên mẫu tương tự cũ hơn, do đó tăng tốc quy trình làm việc của bạn. Bạn có thể nhận xét về mẫu phần mềm độc hại bằng lệnh sau:

danh sách python_5_7_to_5_12.py -c <đường dẫn đến mẫu phần mềm độc hại>

Tìm kiếm Với một mẫu phần mềm độc hại duy nhất, việc tìm kiếm sẽ xác định tất cả các mẫu tương tự trong cơ sở dữ liệu và in chúng theo thứ tự tương tự giảm dần. Ngoài ra, bất kỳ nhận xét nào bạn có thể đã thực hiện trên các mẫu đó đều được in. Bạn có thể tìm kiếm các mẫu phần mềm độc hại tương tự như mẫu đã cho bằng cách sử dụng lệnh sau:

danh sách python_5_7_to_5_12.py -s <đường dẫn đến mẫu phần mềm độc hại>

Wipe Việc xóa cơ sở dữ liệu chỉ đơn giản là xóa tất cả các bản ghi khỏi cơ sở dữ liệu hệ thống, bạn có thể thực hiện thao tác này bằng lệnh sau:

danh sách python_5_7_to_5_12.py -w

Liệt kê 5-13 cho thấy nó trông như thế nào khi chúng ta tải các mẫu APT1 vào hệ thống.

```
mds@mds:~/malware_data_science/ch5/code$ python lists_5_7_to_5_12.py -l ..
dữ liệu
Đã trích xuất 240 thuộc tính từ .. /data/APT1_MALWARE_FAMILIES /WEBC2-YAHOO /WEBC2-YAHOO_sample /WEBC2-YAHOO_sample_A8F259BB36E00D124963CFA9B86F502E ...
Đã trích xuất 272 thuộc tính từ .. /data/APT1_MALWARE_FAMILIES /WEBC2-YAHOO /WEBC2-YAHOO_sample /WEBC2-YAHOO_sample_0149B7BD7218AAB4E257D28469FDD80D ...
Đã trích xuất 236 thuộc tính từ .. /data/APT1_MALWARE_FAMILIES /WEBC2-YAHOO /WEBC2-YAHOO_sample /WEBC2-YAHOO_sample_CC3AA9A7B026BFE0E55FF219FD6AA7D94 ...
Đã trích xuất 272 thuộc tính từ .. /data/APT1_MALWARE_FAMILIES /WEBC2-YAHOO /WEBC2-YAHOO_sample /WEBC2-YAHOO_sample_1415EB8519D13328091CC5C76A624E3D ...
Đã trích xuất 236 thuộc tính từ .. /data/APT1_MALWARE_FAMILIES /WEBC2-YAHOO /WEBC2-YAHOO_sample /WEBC2-YAHOO_sample_7A670D13D4D014169C4080328B8FEB86 ...
Đã trích xuất 243 thuộc tính từ .. /data/APT1_MALWARE_FAMILIES /WEBC2-YAHOO /WEBC2-YAHOO_sample /WEBC2-YAHOO_sample_37DDDD3D72EAD03C7518F5D47650C8572 ...
--snip--
```

Liệt kê 5-13: Đầu ra mẫu từ dữ liệu tải vào hệ thống tìm kiếm tương đồng được triển khai trong chương này

Và Liệt kê 5-14 cho thấy nó trông như thế nào khi chúng ta thực hiện tìm kiếm tương đồng.

```
mds@mds:~/malware_data_science/ch5/code$ python lists_5_7_to_5_12.py -s \ .. /data/
APT1_MALWARE_FAMILIES /GREENCAT /GREENCAT_sample /GREENCAT_sample_AB20 \
8F0B517BA9850F1551C9555B5313
```

Tên mẫu ước tính	mã chia sẻ
[*]GREENCAT_sample_5AEAA53340A281074FCB539967438E3F	1.0
[*]GREENCAT_sample_1F92FF8711716CA795FBD81C477E45F5	1.0
[*]GREENCAT_sample_3E69945E5865CCC861F69B248C1166B6	1.0
[*]GREENCAT_sample_AB208F0B517BA9850F1551C9555B5313	1.0
[*]GREENCAT_sample_3E6ED3EE47BCE9946E2541332CB34C69	0,99609375
[*]GREENCAT_sample_C044715C2626AB515F6C85A21C47C7DD	0,6796875
[*]GREENCAT_sample_871CC547FEB9DBEC0285321068E392B8	0,62109375
[*]GREENCAT_sample_57E79F7DF13C0CB01910D0C688FCDF296	0,62109375

Liệt kê 5-14: Đầu ra mẫu từ hệ thống tìm kiếm tương đồng được triển khai trong chương này

Lưu ý rằng hệ thống của chúng tôi xác định chính xác rằng mẫu truy vấn (mẫu "greencat") chia sẻ mã với các mẫu greencat khác. Nếu chúng tôi không may mắn biết được những mẫu này là thành viên của gia đình mèo xanh, thì hệ thống của chúng tôi sẽ giúp chúng tôi tiết kiệm được rất nhiều công việc kỹ thuật đảo ngược.

Hệ thống tìm kiếm sự tương đồng này chỉ là một ví dụ nhỏ về những gì sẽ được triển khai trong hệ thống tìm kiếm sự tương đồng trong sản xuất. Nhưng bạn sẽ không gặp vấn đề gì khi sử dụng những gì bạn đã học cho đến nay để thêm khả năng trực quan hóa vào hệ thống và mở rộng nó để hỗ trợ nhiều phương pháp tìm kiếm tương đồng.

Bản tóm tắt

Trong chương này, bạn đã học cách xác định mối quan hệ mã được chia sẻ giữa các mẫu phần mềm độc hại, tính toán độ tương tự chia sẻ mã trên hàng nghìn mẫu phần mềm độc hại để xác định họ phần mềm độc hại mới, xác định độ tương tự mã của mẫu phần mềm độc hại mới với hàng nghìn mẫu phần mềm độc hại đã thấy trước đó và trực quan hóa mối quan hệ phần mềm độc hại để hiểu các mẫu chia sẻ mã.

Giờ đây, bạn sẽ cảm thấy thoải mái khi thêm phân tích mã được chia sẻ vào hộp công cụ phân tích phần mềm độc hại của mình, điều này sẽ cho phép bạn thu được thông tin nhanh chóng về khối lượng phần mềm độc hại lớn và tăng tốc quy trình phân tích phần mềm độc hại của bạn.

Trong các Chương 6, 7 và 8, bạn sẽ học cách xây dựng các hệ thống máy học để phát hiện phần mềm độc hại. Kết hợp các kỹ thuật phát hiện này với những gì bạn đã học sẽ giúp bạn phát hiện phần mềm độc hại nâng cao mà các công cụ khác bù sót, cũng như phân tích mối quan hệ của nó với phần mềm độc hại đã biết khác để có manh mối về người đã triển khai phần mềm độc hại và mục tiêu của chúng là gì.

6

U nders tan di ng M achine L thu nhập-Ba se d M alware máy dò s



Với các công cụ máy học mã nguồn mở hiện có, bạn có thể xây dựng các công cụ phát hiện phần mềm độc hại dựa trên máy học, tùy chỉnh, dù là công cụ phát hiện chính của bạn hay để bổ sung cho các giải pháp thương mại, với tương đối ít nỗ lực.

Nhưng tại sao phải xây dựng các công cụ máy học của riêng bạn khi các giải pháp chống vi-rút thương mại đã có sẵn? Khi bạn có quyền truy cập vào các ví dụ về các mối đe dọa cụ thể, chẳng hạn như phần mềm độc hại do một nhóm kẻ tấn công nhất định sử dụng để lấy mạng của bạn, việc xây dựng các công nghệ phát hiện dựa trên máy học của riêng bạn có thể cho phép bạn nắm bắt các ví dụ mới về các mối đe dọa này.

Ngược lại, các công cụ chống vi-rút thương mại có thể bỏ sót các mối đe dọa này trừ khi họ đã bao gồm chữ ký cho họ. Các công cụ thương mại cũng là "sách đã đóng"—nghĩa là chúng ta không nhất thiết phải biết chúng hoạt động như thế nào và chúng ta có khả năng hạn chế để điều chỉnh chúng. Khi chúng tôi xây dựng các phương pháp phát hiện của riêng mình, chúng tôi biết cách chúng hoạt động và có thể điều chỉnh chúng theo ý thích của chúng tôi để giảm các kết quả dương tính giả hoặc âm tính giả. Điều này hữu ích vì trong một số ứng dụng, bạn có thể sẵn sàng chấp nhận nhiều thông báo sai hơn để đổi lấy ít thông báo sai hơn.

(ví dụ: khi bạn đang tìm kiếm các tệp đáng ngờ trên mạng của mình để bạn có thể kiểm tra thử công chúng nhằm xác định xem chúng có độc hại hay không) và trong các ứng dụng khác, bạn có thể sẵn sàng chấp nhận nhiều thông báo phủ định sai hơn để đổi lấy ít thông báo xác thực sai hơn (ví dụ: nếu ứng dụng của bạn chặn các chương trình thực thi nếu ứng dụng xác định chúng là độc hại, nghĩa là các kết quả dương tính giả gây khó chịu cho người dùng).

Trong chương này, bạn tìm hiểu quy trình phát triển các công cụ phát hiện của riêng mình ở cấp độ cao. Tôi bắt đầu bằng cách giải thích những ý tưởng lớn đằng sau quá trình học máy, bao gồm không gian đặc trưng, ranh giới quyết định, dữ liệu huấn luyện, trang bị thiếu và trang bị thừa. Sau đó, tôi tập trung vào bốn cách tiếp cận cơ bản-hồi quy logistic, k-láng giềng gần nhất, cây quyết định và rừng ngẫu nhiên—và cách áp dụng những cách này để thực hiện phát hiện.

Sau đó, bạn sẽ sử dụng những gì đã học trong chương này để tìm hiểu cách đánh giá độ chính xác của các hệ thống máy học trong Chương 7 và triển khai các hệ thống máy học bằng Python trong Chương 8. Hãy bắt đầu nào.

Các bước xây dựng bộ phát hiện dựa trên máy học

Có một sự khác biệt cơ bản giữa học máy và các loại thuật toán máy tính khác. Trong khi các thuật toán truyền thống cho máy tính biết phải làm gì, thì các hệ thống máy học sẽ học cách giải quyết vấn đề bằng ví dụ. Chẳng hạn, thay vì chỉ lấy từ một bộ quy tắc được cấu hình sẵn, các hệ thống phát hiện bảo mật máy học có thể được đào tạo để xác định xem một tệp là xấu hay tốt bằng cách học hỏi từ các ví dụ về các tệp tốt và xấu.

Lời hứa của các hệ thống máy học đối với bảo mật máy tính là chúng tự động hóa công việc tạo chữ ký và chúng có khả năng thực hiện chính xác hơn các phương pháp phát hiện phần mềm độc hại dựa trên chữ ký, đặc biệt là trên phần mềm độc hại mới, chưa từng thấy trước đây.

Về cơ bản, quy trình làm việc mà chúng tôi tuân theo để xây dựng bất kỳ máy học nào-máy dò dựa trên, bao gồm cả cây quyết định, thực hiện theo các bước sau:

1. Thu thập các ví dụ về phần mềm độc hại và phần mềm lành tính. Chúng tôi sẽ sử dụng các ví dụ này (được gọi là ví dụ đào tạo) để đào tạo hệ thống máy học nhận dạng phần mềm độc hại.
2. Trích xuất các đặc trưng từ mỗi ví dụ huấn luyện để biểu diễn ví dụ đó dưới dạng một mảng số. Bước này cũng bao gồm nghiên cứu để thiết kế các tính năng tốt sẽ giúp hệ thống máy học của bạn đưa ra những suy luận chính xác.
3. Huấn luyện hệ thống máy học nhận dạng phần mềm độc hại bằng cách sử dụng features chúng tôi đã trích xuất.
4. Kiểm tra cách tiếp cận trên một số dữ liệu không có trong các ví dụ đào tạo của chúng tôi để xem hệ thống phát hiện của chúng tôi hoạt động tốt như thế nào.

Hãy thảo luận chi tiết hơn về từng bước này trong các phần sau.

Tập hợp các ví dụ đào tạo

Máy dò học máy sống hay chết bởi dữ liệu huấn luyện được cung cấp cho chúng.

Khả năng nhận dạng các tệp nhị phân đáng ngờ của trình phát hiện phần mềm độc hại của bạn phụ thuộc rất nhiều vào số lượng và chất lượng của các ví dụ đào tạo mà bạn cung cấp. Hãy sẵn sàng dành phần lớn thời gian của bạn để thu thập các ví dụ đào tạo khi xây dựng bộ phát hiện dựa trên máy học, bởi vì bạn càng cung cấp nhiều ví dụ cho hệ thống của mình thì hệ thống càng có khả năng chính xác hơn.

Chất lượng của các ví dụ đào tạo của bạn cũng rất quan trọng. Phần mềm độc hại và phần mềm lành tính mà bạn thu thập phải phản ánh loại phần mềm độc hại và phần mềm lành tính mà bạn mong muốn trình phát hiện của mình nhìn thấy khi bạn yêu cầu nó quyết định xem các tệp mới là độc hại hay lành tính.

Ví dụ: nếu bạn muốn phát hiện phần mềm độc hại từ một tác nhân đe dọa cụ thể nhóm, bạn phải thu thập càng nhiều phần mềm độc hại càng tốt từ nhóm đó để sử dụng trong việc đào tạo hệ thống của mình. Nếu mục tiêu của bạn là phát hiện nhiều loại phần mềm độc hại (chẳng hạn như phần mềm tống tiền), thì điều cần thiết là phải thu thập càng nhiều mẫu đại diện của loại này càng tốt.

Tương tự như vậy, các ví dụ đào tạo lành tính mà bạn cung cấp cho hệ thống của mình sẽ phản ánh các loại tệp lành tính mà bạn sẽ yêu cầu trình phát hiện của mình phân tích khi bạn triển khai nó. Ví dụ: nếu bạn đang phát hiện phần mềm độc hại trên mạng của trường đại học, thì bạn nên đào tạo hệ thống của mình bằng một mẫu rộng rãi phản hồi phần mềm độc hại mà sinh viên và nhân viên trường đại học sử dụng để tránh thông báo sai. Những ví dụ lành tính này sẽ bao gồm trò chơi máy tính, trình chỉnh sửa tài liệu, phần mềm tùy chỉnh do khoa CNTT của trường đại học viết và các loại chương trình không độc hại khác.

Để đưa ra một ví dụ trong thế giới thực, tại công việc hiện tại của tôi, chúng tôi đã xây dựng một trình phát hiện có thể phát hiện các tài liệu Office độc hại. Chúng tôi đã dành khoảng một nửa thời gian cho dự án này để thu thập dữ liệu đào tạo, bao gồm cả việc thu thập các tài liệu lành tính do hơn một nghìn nhân viên của công ty tôi tạo ra.

Sử dụng các ví dụ này để đào tạo hệ thống của chúng tôi đã giảm đáng kể tỷ lệ dương tính giả của chúng tôi.

Trích xuất các tính năng

Để phân loại tệp là tốt hay xấu, chúng tôi đào tạo các hệ thống máy học bằng cách hiển thị cho chúng các tính năng của tệp nhị phân phần mềm; đây là những thuộc tính tệp sẽ giúp hệ thống phân biệt tệp tốt và tệp xấu. Ví dụ: đây là một số tính năng chúng tôi có thể sử dụng để xác định xem một tệp là tốt hay xấu:

- Cho dù đó là chữ ký điện tử
- Sự hiện diện của các tiêu đề không đúng định dạng
- Sự hiện diện của dữ liệu được mã hóa
- Cho dù nó đã được nhìn thấy trên hơn 100 máy trạm mạng

Để có được các tính năng này, chúng tôi cần trích xuất chúng từ các tệp. Ví dụ: chúng tôi có thể viết mã để xác định xem tệp có được ký điện tử hay không, có tiêu đề không đúng định dạng, chứa dữ liệu được mã hóa, v.v.

Thông thường, trong khoa học dữ liệu bảo mật, chúng tôi sử dụng một số lượng lớn các tính năng trong trình phát hiện máy học của mình. Ví dụ: chúng tôi có thể tạo một tính năng cho mọi lệnh gọi thư viện trong API Win32, sao cho tệp nhị phân sẽ có tính năng đó nếu nó có lệnh gọi API tương ứng. Chúng ta sẽ xem lại trích xuất tính năng trong Chương 8, nơi chúng ta thảo luận về các khái niệm trích xuất tính năng nâng cao hơn cũng như cách sử dụng chúng để triển khai các hệ thống máy học trong Python.

Thiết kế các tính năng tốt

Mục tiêu của chúng tôi là chọn các tính năng mang lại kết quả chính xác nhất.

Phần này cung cấp một số quy tắc chung để làm theo.

Đầu tiên, khi chọn các tính năng, hãy chọn những tính năng thể hiện dự đoán tốt nhất của bạn về những gì có thể giúp hệ thống máy học phân biệt tệp xấu với tệp tốt. Ví dụ: tính năng "chứa dữ liệu được mã hóa" có thể là dấu hiệu tốt cho phần mềm độc hại vì chúng tôi biết rằng phần mềm độc hại thường chứa dữ liệu được mã hóa và chúng tôi đoán rằng phần mềm lành tính sẽ hiếm khi chứa dữ liệu được mã hóa hơn. Cái hay của học máy là nếu giả thuyết này sai và phần mềm lành tính chứa dữ liệu được mã hóa thường xuyên như phần mềm độc hại, hệ thống sẽ ít nhiều bỏ qua tính năng này. Nếu giả thuyết của chúng tôi là đúng, hệ thống sẽ học cách sử dụng tính năng "chứa dữ liệu được mã hóa" để phát hiện phần mềm độc hại.

Thứ hai, không sử dụng quá nhiều tính năng khiến bộ tính năng của bạn trở nên quá lớn so với số lượng ví dụ đào tạo cho hệ thống phát hiện của bạn. Đây là điều mà các chuyên gia học máy gọi là "lời nguyền của chiều không gian". Ví dụ: nếu bạn có một nghìn tính năng và chỉ một nghìn ví dụ đào tạo, thì xác có thể bạn không có đủ ví dụ đào tạo để dạy cho hệ thống máy học của mình biết mỗi tính năng thực sự nói gì về một nhị phân nhất định. Số liệu thống kê cho chúng tôi biết rằng tốt hơn là cung cấp cho hệ thống của bạn một số tính năng tương ứng với số ví dụ đào tạo mà bạn có sẵn và để hệ thống hình thành niềm tin có cơ sở về những tính năng thực sự chỉ ra phần mềm độc hại.

Cuối cùng, hãy đảm bảo các tính năng của bạn đại diện cho một loạt giả thuyết về yếu tố cấu thành phần mềm độc hại hoặc phần mềm lành tính. Ví dụ: bạn có thể chọn xây dựng các tính năng liên quan đến mã hóa, chẳng hạn như liệu một tệp có sử dụng lệnh gọi API liên quan đến mã hóa hay cơ sở hạ tầng khóa công khai (PKI) hay không, nhưng hãy đảm bảo cũng sử dụng các tính năng không liên quan đến mã hóa để phòng ngừa rủi ro cho các khoản đặt cược của bạn. Theo cách đó, nếu hệ thống của bạn không phát hiện được phần mềm độc hại dựa trên một loại tính năng, thì hệ thống vẫn có thể phát hiện ra phần mềm đó bằng các tính năng khác.

Hệ thống máy học đào tạo

Sau khi bạn đã trích xuất các tính năng từ các tệp nhị phân đào tạo của mình, đã đến lúc đào tạo hệ thống máy học của bạn. Điều này trông như thế nào về mặt thuật toán phụ thuộc hoàn toàn vào phương pháp học máy mà bạn đang sử dụng. Ví dụ, đào tạo một cách tiếp cận cây quyết định (mà chúng ta sẽ thảo luận ngay sau đây) liên quan đến một thuật toán học tập khác với việc đào tạo một phương pháp hồi quy logistic (mà chúng ta cũng sẽ thảo luận).

May mắn thay, tất cả các máy đào tạo máy đều cung cấp giao diện cơ bản giống nhau. Bạn cung cấp cho họ dữ liệu đào tạo có chứa các tính năng từ các tệp nhị phân mẫu, cũng như các nhãn tương ứng cho biết thuật toán

nhi phần nào là phần mềm độc hại và phần mềm nào là lành tính. Sau đó, các thuật toán thuật toán học cách xác định xem các tệp nhị phân mới, chưa từng thấy trước đây là độc hại hay lành tính. Chúng tôi đề cập đến việc đào tạo chi tiết hơn ở phần sau của chương này.

LƯU Ý Trong cuốn sách này, chúng tôi tập trung vào một loại thuật toán học máy được gọi là thuật toán học máy siêu trực quan. Để đào tạo các mô hình sử dụng các thuật toán này, chúng tôi cho chúng biết ví dụ nào độc hại và ví dụ nào lành tính. Một loại thuật toán học máy khác, thuật toán không giám sát, không yêu cầu chúng tôi biết ví dụ nào là độc hại hoặc lành tính trong tập huấn luyện của chúng tôi. Các thuật toán này kém hiệu quả hơn nhiều trong việc phát hiện phần mềm độc hại và hành vi độc hại, và chúng tôi sẽ không đề cập đến chúng trong cuốn sách này.

Kiểm tra hệ thống máy học

Khi bạn đã đào tạo hệ thống máy học của mình, bạn cần kiểm tra độ chính xác của nó. Bạn làm điều này bằng cách chạy hệ thống được đào tạo trên dữ liệu mà bạn không đào tạo nó và xem nó xác định các tệp nhị phân là độc hại hay lành tính tốt như thế nào. Về bảo mật, chúng tôi thường huấn luyện hệ thống của mình trên các tệp nhị phân mà chúng tôi đã thu thập tại một thời điểm nào đó, sau đó chúng tôi kiểm tra các tệp nhị phân mà chúng tôi đã thấy sau thời điểm đó, để đo lường mức độ hệ thống của chúng tôi sẽ phát hiện phần mềm độc hại mới và để đo lường mức độ hệ thống của chúng tôi sẽ tránh tạo ra các thông tin xác thực sai trên phần mềm lành tính mới. Hầu hết các nghiên cứu về máy học liên quan đến hàng nghìn lần lặp đi lặp lại như sau: chúng tôi tạo ra một hệ thống máy học, kiểm tra và sau đó điều chỉnh nó, đào tạo lại và kiểm tra lại cho đến khi chúng tôi hài lòng với kết quả. Tôi sẽ đề cập chi tiết đến việc thử nghiệm các hệ thống máy học trong Chương 8.

Bây giờ chúng ta hãy thảo luận về cách hoạt động của nhiều thuật toán máy học. Đây là phần khó nhất của chương, nhưng cũng là phần bổ ích nhất nếu bạn dành thời gian để hiểu nó. Trong cuộc thảo luận này, tôi nói về các ý tưởng thống nhất làm nền tảng cho các thuật toán này và sau đó chuyển sang chi tiết từng thuật toán.

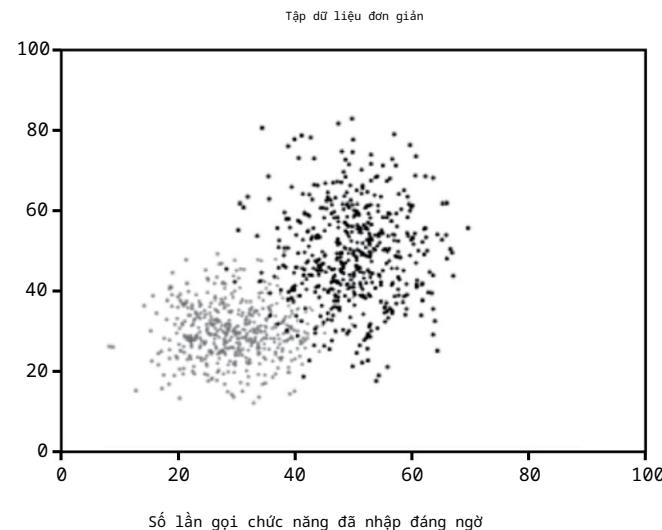
Hiểu không gian tính năng và ranh giới quyết định

Hai ý tưởng hình học đơn giản có thể giúp bạn hiểu toàn bộ quá trình học máy – các thuật toán phát hiện dựa trên: ý tưởng về không gian đặc trưng hình học và ý tưởng về ranh giới quyết định. Không gian đặc trưng là không gian hình học được xác định bởi các tính năng bạn đã chọn và ranh giới quyết định là cấu trúc hình học chạy qua không gian này sao cho các tệp nhị phân ở một bên của ranh giới này được xác định là phần mềm độc hại và các tệp nhị phân ở phía bên kia của ranh giới này. ranh giới được định nghĩa là phần mềm lành tính. Khi chúng tôi sử dụng thuật toán máy học để phân loại tệp là độc hại hay lành tính, chúng tôi trích xuất các tính năng để có thể đặt các mẫu vào không gian tính năng, sau đó chúng tôi kiểm tra xem các mẫu nằm ở phía nào của ranh giới quyết định để xác định xem các tệp là phần mềm độc hại hoặc phần mềm lành tính.

Cách hiểu hình học này về các không gian đặc trưng và các ranh giới quyết định là chính xác đối với các hệ thống hoạt động trên các không gian đặc trưng một, hai hoặc ba chiều (các đặc trưng), nhưng nó cũng đúng cho các không gian đặc trưng với

hàng triệu chiều, mặc dù không thể hình dung hay quan niệm về không gian hàng triệu chiều. Chúng ta sẽ sử dụng các ví dụ có hai chiều trong chương này để dễ hình dung, nhưng chỉ cần nhớ rằng các hệ thống máy học bảo mật trong thế giới thực hầu như luôn sử dụng hàng trăm, hàng nghìn hoặc hàng triệu chiều và các khái niệm cơ bản mà chúng ta thảo luận trong phần bối cảnh hai chiều giữ cho các hệ thống trong thế giới thực có nhiều hơn hai chiều.

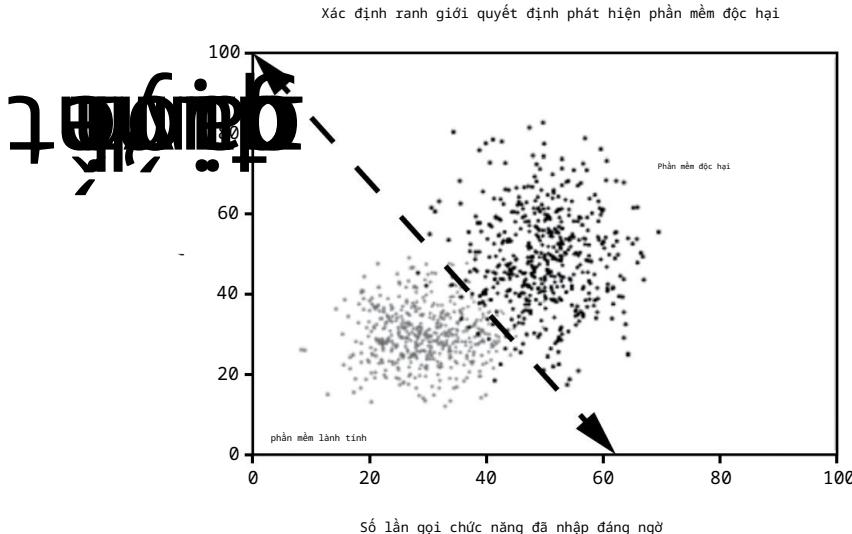
Hãy tạo một vấn đề phát hiện phần mềm độc hại đồ chơi để làm rõ ý tưởng về ranh giới quyết định trong một không gian tính năng. Giả sử chúng ta có một tập dữ liệu đào tạo bao gồm các mẫu phần mềm độc hại và phần mềm lành tính. Bởi giờ, giả sử chúng tôi trích xuất hai tính năng sau từ mỗi tệp nhị phân: tỷ lệ phần trăm của tệp dường như được nén và số lượng chức năng đáng ngờ mà mỗi tệp nhị phân nhập. Chúng ta có thể hình dung tập dữ liệu đào tạo của mình như trong Hình 6-1 (hãy nhớ rằng tôi đã tạo dữ liệu trong biểu đồ một cách giả tạo, cho các mục đích ví dụ).



Hình 6-1: Biểu đồ của tập dữ liệu mẫu mà chúng ta sẽ sử dụng trong chương này, trong đó các chấm màu xám là phần mềm lành tính và các chấm đen là phần mềm độc hại

Không gian hai chiều được hiển thị trong Hình 6-1, được xác định bởi hai tính năng của chúng tôi, là không gian tính năng cho tập dữ liệu mẫu của chúng tôi. Bạn có thể thấy một mẫu rõ ràng trong đó các chấm đen (phần mềm độc hại) thường nằm ở phần trên bên phải của không gian. Nói chung, những phần mềm này có nhiều lệnh gọi chức năng được nhập đáng ngờ hơn và nhiều dữ liệu nén hơn so với phần mềm lành tính, phần lớn nằm ở phần dưới bên trái của cốt truyện. Giả sử, sau khi xem cốt truyện này, bạn được yêu cầu tạo một hệ thống phát hiện phần mềm độc hại chỉ dựa trên hai tính năng chúng tôi đang sử dụng ở đây. Có vẻ như rõ ràng rằng, dựa trên dữ liệu, bạn có thể xây dựng quy tắc sau: nếu một tệp nhị phân có nhiều dữ liệu nén và nhiều lệnh gọi hàm được nhập đáng ngờ, thì đó là phần mềm độc hại và nếu nó không có nhiều lệnh gọi được nhập đáng ngờ cũng không có nhiều dữ liệu nén, đó là phần mềm lành tính.

Về mặt hình học, chúng ta có thể hình dung quy tắc này dưới dạng một đường chéo phân tách các mẫu phần mềm độc hại khỏi các mẫu phần mềm độc hại trong không gian tính năng, sao cho các tệp nhị phân có đủ dữ liệu nén và lệnh gọi chức năng đã nhập (được xác định là phần mềm độc hại) nằm phía trên đường này và phần còn lại của các tệp nhị phân (được định nghĩa là phần mềm lành tính) nằm bên dưới dòng. Hình 6-2 cho thấy một đường như vậy, mà chúng ta gọi là ranh giới quyết định.

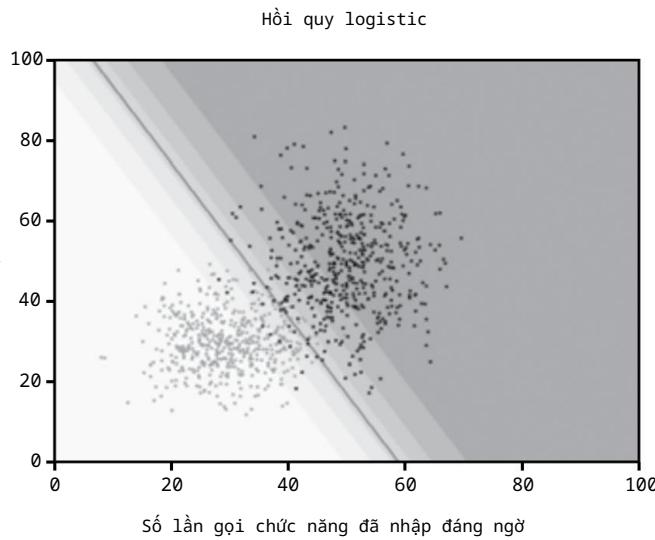


Hình 6-2: Ranh giới quyết định được vẽ thông qua bộ dữ liệu mẫu của chúng tôi, xác định quy tắc phát hiện phần mềm độc hại

Như bạn có thể thấy từ đường kẻ, hầu hết các chấm đen (phần mềm độc hại) nằm ở một bên của ranh giới và hầu hết các mẫu màu xám (phần mềm lành tính) nằm ở phía bên kia của ranh giới quyết định. Lưu ý rằng không thể vẽ một đường phân cách tất cả các mẫu với nhau vì các đám mây đen và xám trong tập dữ liệu này chồng lên nhau. Nhưng khi xem xét ví dụ này, có vẻ như chúng tôi đã vạch ra một đường phân loại chính xác các mẫu phần mềm độc hại mới và mẫu phần mềm lành tính trong hầu hết các trường hợp, giả sử chúng tuân theo mẫu được thấy trong dữ liệu trong hình ảnh này.

Trong Hình 6-2, chúng tôi đã vẽ ranh giới quyết định theo cách thủ công thông qua dữ liệu của mình. Nhưng nếu chúng ta muốn có một ranh giới quyết định chính xác hơn và muốn thực hiện nó theo cách tự động thì sao? Đây chính xác là những gì máy học làm. Nói cách khác, tất cả các thuật toán phát hiện máy học đều xem xét dữ liệu và sử dụng quy trình tự động đổi chiều để xác định ranh giới quyết định lý tưởng, sao cho có cơ hội lớn nhất thực hiện phát hiện chính xác dữ liệu mới, chưa từng thấy trước đây.

Hãy xem cách một thuật toán học máy thường được sử dụng trong thế giới thực xác định ranh giới quyết định trong dữ liệu mẫu được hiển thị trong Hình 6-3. Ví dụ này sử dụng một thuật toán gọi là hồi quy logistic.



Hình 6-3: Ranh giới quyết định được tạo tự động bằng cách huấn luyện mô hình hồi quy logistic

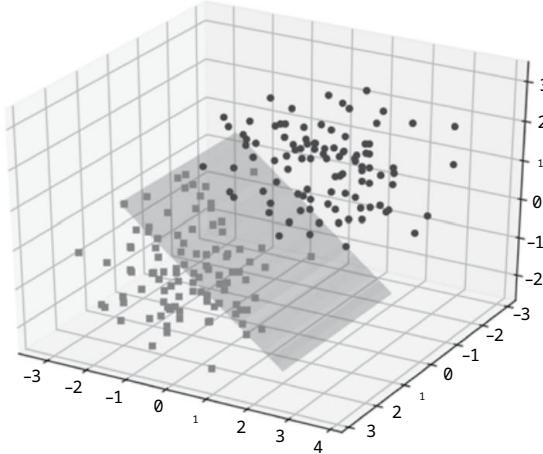
Lưu ý rằng chúng tôi đang sử dụng cùng một dữ liệu mẫu mà chúng tôi đã sử dụng trong các ô trước đó, trong đó các chấm màu xám là phần mềm lành tính và các chấm đen là phần mềm độc hại. Đường chạy qua trung tâm của biểu đồ là ranh giới quyết định mà thuật toán hồi quy logistic học được bằng cách xem dữ liệu. Ở phía bên phải của dòng, thuật toán hồi quy logistic chỉ định xác suất lớn hơn 50 phần trăm rằng các tệp nhị phân là phần mềm độc hại và ở phía bên trái của dòng, nó chỉ định xác suất dưới 50 phần trăm rằng tệp nhị phân là phần mềm độc hại.

Bây giờ lưu ý các khu vực bóng mờ của cốt truyện. Vùng bóng mờ màu xám đậm là khu vực mà mô hình hồi quy logistic tin chắc rằng các tệp là phần mềm độc hại. Bất kỳ tệp mới nào mà mô hình hồi quy logistic thấy có các tính năng nằm trong khu vực này đều có khả năng cao là phần mềm độc hại. Khi chúng ta tiến gần hơn đến ranh giới quyết định, mô hình ngày càng ít tin cậy hơn về việc liệu các tệp nhị phân có phải là phần mềm độc hại hay phần mềm lành tính hay không.

Hồi quy logistic cho phép chúng tôi dễ dàng di chuyển dòng lên vùng tối hơn hoặc xuống vùng sáng hơn, tùy thuộc vào mức độ tích cực mà chúng tôi muốn phát hiện phần mềm độc hại. Ví dụ: nếu chúng tôi di chuyển nó xuống, chúng tôi sẽ phát hiện được nhiều phần mềm độc hại hơn nhưng nhận được nhiều thông báo sai hơn. Nếu chúng tôi nâng nó lên, chúng tôi sẽ phát hiện được ít phần mềm độc hại hơn nhưng nhận được ít thông tin xác thực giả hơn.

Tôi muốn nhấn mạnh rằng hồi quy logistic và tất cả các thuật toán học máy khác, có thể hoạt động trong các không gian đặc trưng có chiều cao tùy ý. Hình 6-4 minh họa cách hoạt động của hồi quy logistic trong không gian đặc trưng có số chiều cao hơn một chút.

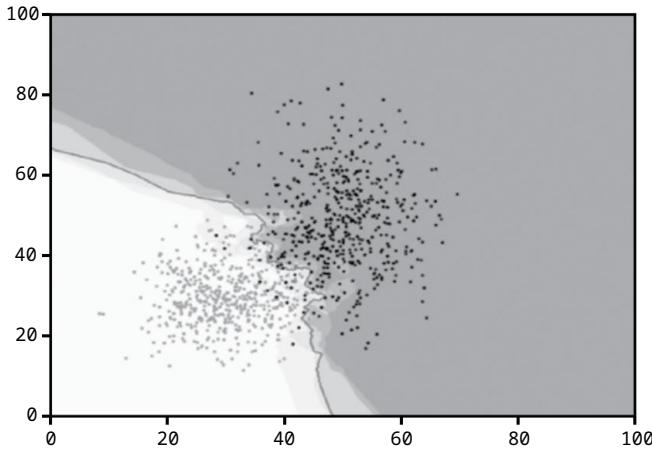
Trong không gian nhiều chiều hơn này, ranh giới quyết định không phải là một đường thẳng mà là một mặt phẳng phân tách các điểm trong khối 3D. Nếu chúng ta chuyển sang bốn chiều trở lên, thì hồi quy logistic sẽ tạo ra một siêu phẳng, là một cấu trúc giống như mặt phẳng n chiều, phân tách phần mềm độc hại khỏi các điểm phần mềm lành tính trong không gian nhiều chiều này.



Hình 6-4: Ranh giới quyết định phẳng thông qua không gian đặc trưng ba chiều giả định được tạo bởi hồi quy logistic

Bởi vì hồi quy logistic là một thuật toán học máy tương đối đơn giản algorithm, nó chỉ có thể tạo ra các ranh giới quyết định hình học đơn giản như đường thẳng, mặt phẳng và các mặt phẳng có chiều cao hơn. Các thuật toán học máy khác có thể tạo ra các ranh giới quyết định phức tạp hơn. Ví dụ, hãy xem xét ranh giới quyết định được hiển thị trong Hình 6-5, được đưa ra bởi thuật toán k-láng giềng gần nhất (mà tôi sẽ thảo luận chi tiết ngay sau đây).

K-Hàng xóm gần nhất



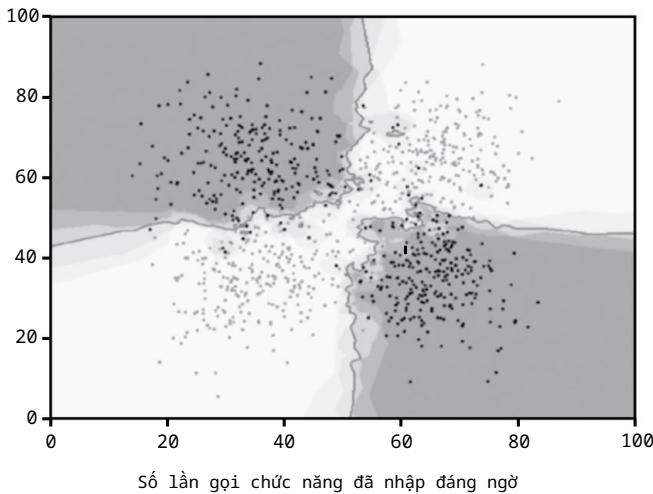
Số lần gọi chức năng đã nhập đáng ngờ

Hình 6-5: Một ranh giới quyết định được tạo bởi thuật toán k-láng giềng gần nhất

Như bạn có thể thấy, ranh giới quyết định này không phải là một mặt phẳng: nó là một cấu trúc rất bất quy tắc. Cũng lưu ý rằng một số thuật toán học máy có thể tạo

ranh giới quyết định rời rạc, xác định một số vùng của không gian đặc trưng là độc hại và một số vùng là lành tính, ngay cả khi những vùng đó không liền kề nhau. Hình 6-6 cho thấy ranh giới quyết định với cấu trúc bất thường này, sử dụng tập dữ liệu mẫu khác với mẫu phần mềm độc hại và phần mềm độc hại phức tạp hơn trong không gian tính năng mẫu của chúng tôi.

K-Hàng xóm gần nhất



Hình 6-6: Ranh giới quyết định rời rạc được tạo bởi thuật toán k-láng giềng gần nhất

Mặc dù ranh giới quyết định không liền kề, nhưng theo cách nói phổ biến của máy học vẫn gọi các ranh giới quyết định rời rạc này đơn giản là “ranh giới quyết định”. Bạn có thể sử dụng các thuật toán máy học khác nhau để thể hiện các loại ranh giới quyết định khác nhau và sự khác biệt về tính biểu cảm này là lý do tại sao chúng tôi có thể chọn một thuật toán máy học này thay vì một thuật toán máy học khác cho một dự án nhất định.

Bây giờ chúng ta đã khám phá các khái niệm máy học cốt lõi như không gian đặc trưng và ranh giới quyết định, tiếp theo hãy thảo luận về những gì các nhà thực hành máy học gọi là trang bị quá mức và trang bị thiếu.

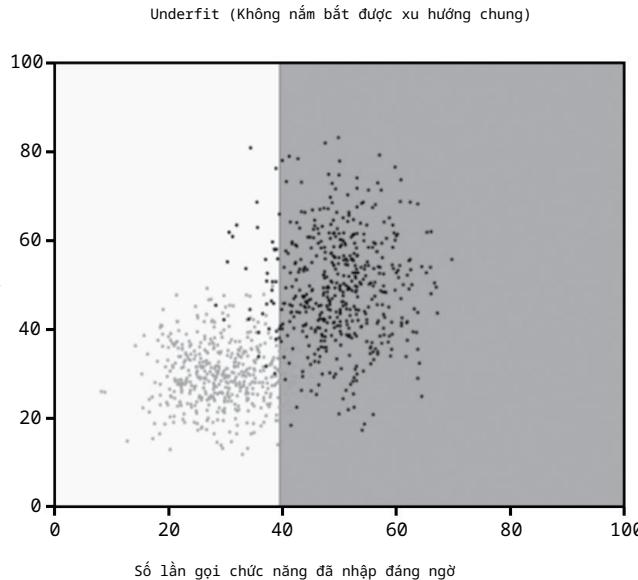
Điều gì tạo nên các mô hình tốt hay xấu: Trang bị quá mức và đồ lót

Tôi không thể nhấn mạnh quá mức tầm quan trọng của việc trang bị thừa và thiếu trong học máy. Tránh cả hai trường hợp là điều xác định một thuật toán học máy tốt. Các mô hình phát hiện tốt, chính xác trong máy học nắm bắt được xu hướng chung trong dữ liệu đào tạo cho biết điều gì phân biệt phần mềm độc hại với phần mềm lành tính, mà không bị phân tâm bởi các giá trị ngoại lệ hoặc ngoại lệ chứng minh quy tắc.

Người mẫu underfit bỏ qua ngoại lệ nhưng không nắm bắt được xu hướng chung, dẫn đến độ chính xác kém trên các tập nhị phân mới, chưa từng thấy trước đây. Các mô hình overfit bị phân tâm bởi các ngoại lệ theo những cách không phản ánh xu hướng chung và chúng mang lại độ chính xác kém trên các nhị phân chưa từng thấy trước đây.

Xây dựng các mô hình phát hiện phần mềm độc hại bằng máy học là nhằm nắm bắt xu hướng chung để phân biệt phần mềm độc hại với phần mềm lành tính.

Hãy sử dụng các ví dụ về mô hình vừa vặn, vừa vặn và vừa vặn trong Hình 6-7, 6-8 và 6-9 để minh họa các thuật ngữ này. Hình 6-7 cho thấy một người mẫu mặc đồ lót.



Hình 6-7: Một mô hình học máy không phù hợp

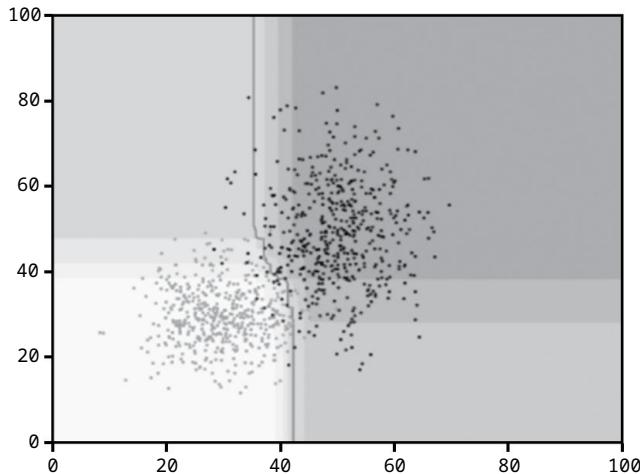
Tại đây, bạn có thể thấy cụm chấm đen (phần mềm độc hại) ở phía trên bên phải của ô và cụm chấm xám (phần mềm lành tính) ở phía dưới bên trái. Tuy nhiên, mô hình học máy của chúng tôi chỉ đơn giản là cắt các dấu chấm ở giữa, phân tách dữ liệu một cách sơ sài mà không nắm bắt được xu hướng chéo. Bởi vì mô hình không nắm bắt được xu hướng chung, chúng tôi nói rằng nó không phù hợp.

Cũng lưu ý rằng chỉ có hai sắc thái charkin mà mô hình mang lại trong tất cả các vùng của biểu đồ: hoặc là bóng có màu xám đậm hoặc là màu trắng. Nói cách khác, mô hình hoặc hoàn toàn charkin rằng các điểm trong không gian đặc trưng là độc hại hoặc hoàn toàn charkin rằng chúng lành tính.

Việc không thể diễn đạt sự charkin một cách chính xác cũng là một lý do khiến mô hình này không phù hợp.

Hãy đổi chiều mô hình vừa vặn trong Hình 6-7 với mô hình vừa vặn trong Hình 6-8.

Vừa vặn (Nắm bắt xu hướng chung)



Số lần gọi chức năng đã nhập đáng ngờ

Hình 6-8: Một mô hình học máy phù hợp

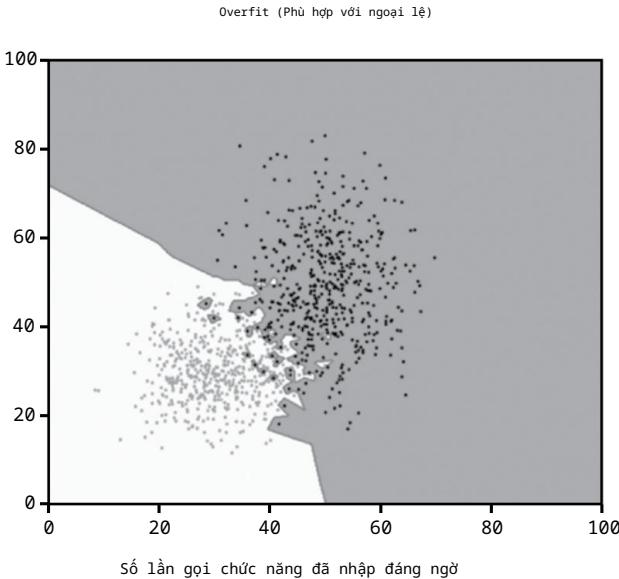
Trong trường hợp này, mô hình không chỉ nắm bắt xu hướng chung trong dữ liệu mà còn tạo ra một mô hình chắc chắn hợp lý đối với ước tính của nó về vùng nào của không gian đặc trưng chắc chắn là độc hại, chắc chắn lành tính hoặc nằm trong vùng xám.

Lưu ý đường quyết định chạy từ trên xuống dưới của biểu đồ này. Mô hình có một lý thuyết đơn giản về những gì phân chia phần mềm độc hại khỏi phần mềm lành tính: một đường thẳng đứng với một đường chéo ở giữa biểu đồ. Cũng lưu ý các vùng được tô bóng trong biểu đồ, cho chúng ta biết rằng mô hình chỉ chắc chắn rằng dữ liệu ở phần trên bên phải của biểu đồ là phần mềm độc hại và chỉ chắc chắn rằng các tệp nhị phân ở góc dưới bên trái của biểu đồ là phần mềm lành tính.

Cuối cùng, hãy so sánh mô hình overfit được hiển thị tiếp theo trong Hình 6-9 với mô hình underfit mà bạn đã thấy trong Hình 6-7 cũng như mô hình vừa vặn trong Hình 6-8.

Mô hình overfit trong Hình 6-9 không nắm bắt được xu hướng chung trong dữ liệu. Thay vào đó, nó ám ảnh về các ngoại lệ trong dữ liệu, bao gồm một số chấm đen (ví dụ đào tạo phần mềm độc hại) xuất hiện trong cụm chấm xám (ví dụ đào tạo lành tính) và vẽ ra các ranh giới quyết định xung quanh chúng. Tương tự như vậy, nó tập trung vào một số ví dụ về phần mềm lành tính xuất hiện trong cụm phần mềm độc hại, đồng thời vạch ra các ranh giới xung quanh chúng.

Điều này có nghĩa là khi chúng ta thấy các nhị phân mới, chưa từng thấy trước đó xảy ra pen có các tính năng đặt chúng gần với các ngoại lệ này, mô hình máy học sẽ nghĩ chúng là phần mềm độc hại trong khi chúng gần như chắc chắn là phần mềm lành tính và ngược lại. Trên thực tế, điều này có nghĩa là mô hình này sẽ không chính xác như mong đợi.



Hình 6-9: Mô hình học máy overfit

Các loại thuật toán học máy chính

Cho đến nay, tôi đã thảo luận về học máy theo các thuật ngữ rất chung chung, đề cập đến hai phương pháp học máy: hồi quy logistic và k-láng giềng gần nhất.

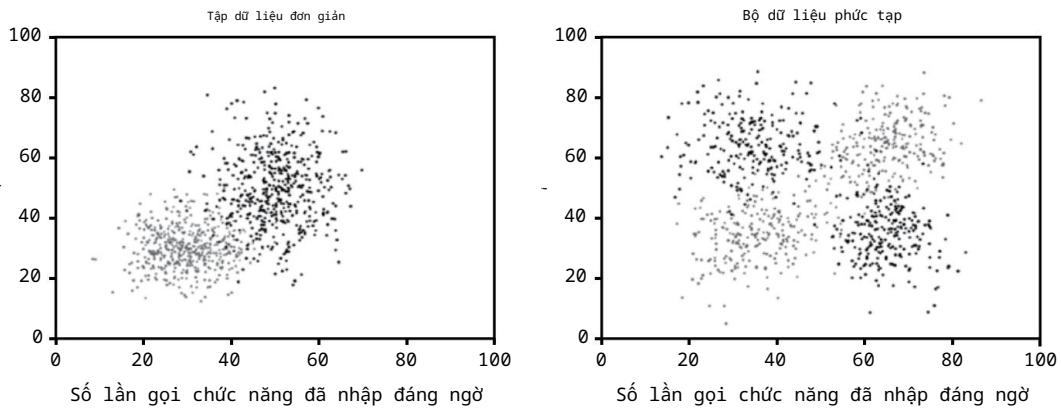
Trong phần còn lại của chương này, chúng ta sẽ nghiên cứu sâu hơn và thảo luận chi tiết hơn về hồi quy logistic, k-láng giềng gần nhất, cây quyết định và thuật toán rừng ngẫu nhiên.

Chúng tôi sử dụng các thuật toán này khá thường xuyên trong cộng đồng khoa học dữ liệu bao mật. Các thuật toán này rất phức tạp, nhưng những ý tưởng đằng sau chúng rất trực quan và dễ hiểu.

Trước tiên, hãy xem các bộ dữ liệu mẫu mà chúng tôi sử dụng để khám phá điểm mạnh và điểm yếu của từng thuật toán, được minh họa trong Hình 6-10.

Tôi đã tạo các bộ dữ liệu này cho các mục đích ví dụ. Ở bên trái, chúng ta có tập dữ liệu đơn giản mà tôi đã sử dụng trong Hình 6-7, 6-8 và 6-9. Trong trường hợp này, chúng ta có thể tách các ví dụ huấn luyện màu đen (phản mềm độc hại) khỏi các ví dụ huấn luyện màu xám (phản mềm lành tính) bằng cách sử dụng cấu trúc hình học đơn giản, chẳng hạn như một đường thẳng.

Tập dữ liệu bên phải, mà tôi đã chỉ ra trong Hình 6-6, là complex vì chúng tôi không thể tách phần mềm độc hại khỏi phần mềm lành tính bằng một dòng đơn giản. Nhưng vẫn có một khuôn mẫu rõ ràng đối với dữ liệu: chúng ta chỉ cần sử dụng các phương pháp phức tạp hơn để tạo ranh giới quyết định. Hãy xem các thuật toán khác nhau hoạt động như thế nào với hai bộ dữ liệu mẫu này.



Hình 6-10: Hai bộ dữ liệu mẫu mà chúng tôi sử dụng trong chương này, với các chấm đen đại diện cho phần mềm độc hại và các chấm màu xám đại diện cho phần mềm đáng ngờ

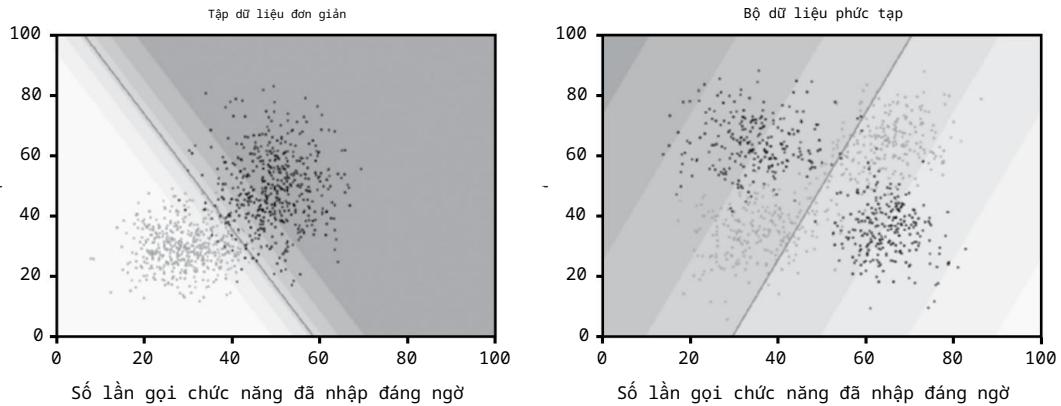
Hồi quy logistic

Như bạn đã biết trước đây, hồi quy logistic là một thuật toán học máy tạo ra một đường thẳng, mặt phẳng hoặc siêu phẳng (tùy thuộc vào số lượng tính năng bạn cung cấp) phân tách hình học phần mềm độc hại đào tạo của bạn khỏi phần mềm lành tính đào tạo của bạn. Khi bạn sử dụng mô hình được đào tạo để phát hiện phần mềm độc hại mới, hồi quy logistic sẽ kiểm tra xem tệp nhị phân chưa từng thấy trước đó nằm ở phía phần mềm độc hại hay phía phần mềm độc hại của ranh giới để xác định xem phần mềm đó độc hại hay lành tính.

Một hạn chế của hồi quy logistic là nếu dữ liệu của bạn không thể được xếp hạng riêng biệt chỉ bằng cách sử dụng một đường hoặc siêu phẳng, thì hồi quy logistic không phải là giải pháp phù hợp. Việc bạn có thể sử dụng hồi quy logistic cho vấn đề của mình hay không tùy thuộc vào dữ liệu và các tính năng của bạn. Ví dụ: nếu vấn đề của bạn có nhiều đặc điểm riêng lẻ mà bản thân chúng là những chỉ báo mạnh mẽ về tính độc hại (hoặc "tính lành tính"), thì hồi quy logistic có thể là một cách tiếp cận thành công. Nếu dữ liệu của bạn đến mức bạn cần sử dụng các mối quan hệ phức tạp giữa các tính năng để quyết định rằng một tệp là phần mềm độc hại, thì một cách tiếp cận khác, chẳng hạn như k-hàng xóm gần nhất, cây quyết định hoặc rừng ngẫu nhiên, có thể tạo ra nhiều kết quả hơn gián quan.

Để minh họa điểm mạnh và điểm yếu của hồi quy logistic, hãy xem hiệu suất của hồi quy logistic trên hai bộ dữ liệu mẫu của chúng tôi, như trong Hình 6-11. Chúng tôi thấy rằng hồi quy logistic mang lại sự phân tách rất hiệu quả giữa phần mềm độc hại và phần mềm lành tính trong tập dữ liệu đơn giản của chúng tôi (ở bên trái). Ngược lại, hiệu suất của hồi quy logistic trên tập dữ liệu phức tạp của chúng tôi (ở bên phải) không hiệu quả. Trong trường hợp này, thuật toán hồi quy logistic bị nhầm lẫn, bởi vì nó chỉ có thể biểu thị một ranh giới quyết định tuyến tính. Bạn có thể thấy cả hai loại nhị phân ở cả hai bên của đường và các dải tin cậy màu xám được tô bóng không thực sự có ý nghĩa gì so với dữ liệu. Đôi với tập dữ liệu phức tạp hơn này, chúng tôi cần sử dụng một thuật toán có khả năng thể hiện nhiều cấu trúc hình học hơn.

Hồi quy logistic



Hình 6-11: Ranh giới quyết định được vẽ thông qua bộ dữ liệu mẫu của chúng tôi bằng hồi quy logistic

Toán học đằng sau hồi quy logistic

Bây giờ chúng ta hãy xem phép toán đằng sau cách hồi quy logistic phát hiện các mẫu phần mềm độc hại. Liệt kê 6-1 cho thấy mã giả Pythonic để tính toán khả năng phát hiện một tệp nhị phân là phần mềm độc hại bằng cách sử dụng hồi quy logistic.

```
def logistic_regression (dữ_liệu_nén_cuộc_gọi_nghi_nghi,_thông_số_dâ_học): u
nén_dữ_liệu = nén_data * learn_parameters["compressed_data_weight"]
    nghi_ngờ_cuộc_gọi = nghi_ngờ_cuộc_gọi * thông_số_dâ_học["suspicious_calls_weight"]
diểm = dữ_liệu_nén + cuộc_gọi_đáng_ngờ + thiêん_vị_w
    trả_về logistic_function(diểm)

def logistic_function(diểm): x
    trả_về 1/(1.0+math.e**(-score))
```

Liệt kê 6-1: Mã giả sử dụng hồi quy logistic để tính xác suất

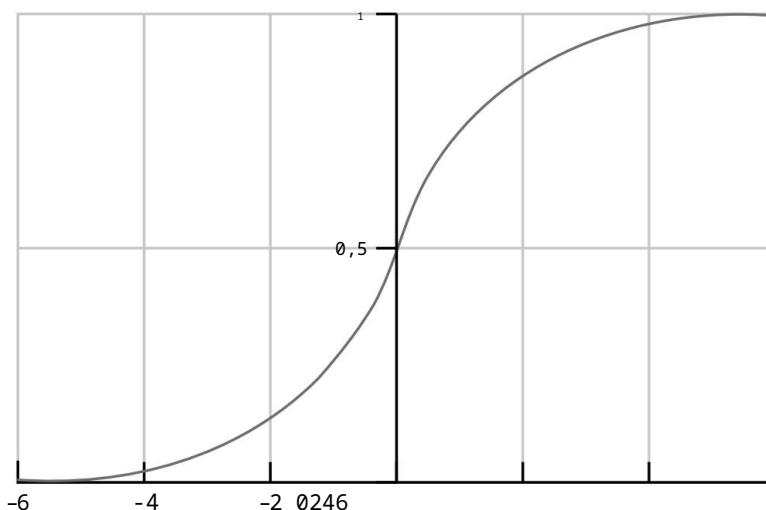
Hãy xem qua mã để hiểu điều này có nghĩa là gì. Đầu tiên chúng ta định nghĩa hàm `logistic_regression` và các tham số của nó. Các tham số của nó là các tính năng của tệp nhị phân (`compressed_data` và `dirty_calls`) đại diện cho lượng dữ liệu nén và số lệnh gọi đáng ngờ mà nó thực hiện, và tham số `learn_parameters` là viết tắt của các phần tử của hàm hồi quy logistic đã học được bằng cách đào tạo mô hình hồi quy logistic trên dữ liệu huấn luyện. Tôi sẽ thảo luận về cách học các tham số ở phần sau của chương này; hiện tại, chỉ cần chấp nhận rằng chúng được lấy từ dữ liệu huấn luyện.

Sau đó, chúng ta lấy đặc tính của dữ liệu nén và nhân nó với tham số trọng lượng_dữ_liệu_nén. Trọng số này tăng hoặc giảm tỷ lệ tính năng, tùy thuộc vào mức độ biểu thị của phần mềm độc hại mà chức năng hồi quy logistic cho rằng tính năng này. Lưu ý rằng trọng số cũng có thể âm, điều này cho biết mô hình hồi quy logistic cho rằng đối tượng địa lý là một chỉ báo cho thấy tệp là lành tính.

Ở dòng bên dưới, chúng tôi thực hiện bước tương tự cho các cuộc gọi đáng ngờ tham số. Sau đó, chúng tôi cộng hai tinh năng có trọng số này lại với nhau , đồng thời thêm vào một tham số gọi là tham số sai lệch (cũng được học từ dữ liệu huấn luyện).

Tóm lại, chúng tôi sử dụng tinh năng `nén_data` , được chia tỷ lệ theo mức độ biểu thị của hành vi độc hại mà chúng tôi tin rằng nó là như thế nào, thêm tinh năng `nghi ngờ_cuộc gọi` , cũng được chia tỷ lệ theo mức độ biểu thị của hành vi độc hại mà chúng tôi tin rằng nó là như thế nào và thêm tham số thiên vị , biểu thị mức độ nghi ngờ mô hình hồi quy logistic nghĩ rằng chúng ta nên có các tệp nói chung. Kết quả của những phép cộng và phép nhân này là điểm số cho biết khả năng một tệp nhất định là độc hại.

Cuối cùng, chúng tôi sử dụng `logistic_function` để chuyển đổi điểm đáng ngờ của chúng tôi thành một xác suất. Hình 6-12 trực quan hóa cách chức năng này hoạt động.



Hình 6-12: Biểu đồ hàm logistic được sử dụng trong hồi quy logistic

Ở đây, hàm logistic lấy điểm (hiển thị trên trục x) và chuyển đổi điểm đó thành giá trị nằm trong khoảng từ 0 đến 1 (xác suất).

Toán học hoạt động như thế nào

Hãy quay trở lại xanh giới quyết định mà bạn đã thấy trong Hình 6-11 để xem phép toán này hoạt động như thế nào trong thực tế. Nhớ lại cách chúng tôi tính toán xác suất của chúng tôi:

```
logistic_function(feature1_weight * Feature1 + Feature2_weight*feature2 + bias)
```

Ví dụ: nếu chúng ta vẽ đồ thị xác suất kết quả tại mọi điểm trong các không gian đặc trưng được hiển thị trong Hình 6-11 bằng cách sử dụng cùng một trọng số đặc trưng và tham số sai lệch , chúng ta sẽ kết thúc với các vùng được tô bóng được hiển thị trong cùng một hình, cho biết vị trí mà mô hình “nghi” là các mẫu độc hại và lành tính, và với bao nhiêu tự tin.

Nếu sau đó chúng ta đặt ngưỡng là 0,5 (nhớ lại rằng có xác suất lớn hơn 50 phần trăm, các tệp được xác định là độc hại), dòng trong

Hình 6-11 sẽ xuất hiện dưới dạng ranh giới quyết định của chúng tôi. Tôi khuyến khích bạn thử nghiệm mã mẫu của tôi, bổ sung một số trọng số tính năng và thuật ngữ thiên vị rồi tự mình thử.

LƯU Ý

Quy logistic không ràng buộc chúng tôi chỉ sử dụng hai tính năng. Trong thực tế, chúng tôi thường sử dụng điểm số hoặc hàng trăm hoặc thậm chí hàng nghìn tính năng với hồi quy logistic. Nhưng toán học không thay đổi: chúng tôi chỉ tính xác suất của mình như sau cho bất kỳ số lượng tính năng nào:

```
logistic_function(feature1 * Feature1_weight + Feature2 * Feature2_weight + bias)
tính năng3 * tính năng3_trọng số ...
```

Vậy chính xác làm thế nào để hồi quy logistic học cách đặt ranh giới quyết định vào đúng vị trí dựa trên dữ liệu đào tạo? Nó sử dụng một cách tiếp cận lặp đi lặp lại, dựa trên phép tính được gọi là giảm dần độ dốc. Chúng ta sẽ không đi sâu vào chi tiết của cách tiếp cận này trong cuốn sách này, nhưng ý tưởng cơ bản là đường thẳng, mặt phẳng hoặc siêu phẳng (tùy thuộc vào số lượng tính năng bạn đang sử dụng) được điều chỉnh lặp lại sao cho nó tối đa hóa xác suất mà mô hình hồi quy logistic sẽ nhận được câu trả lời ngay khi được hỏi liệu một điểm dữ liệu trong tập huấn luyện là mẫu phần mềm độc hại hay mẫu phần mềm lành tính.

Bạn có thể đào tạo các mô hình hồi quy logistic để thuật toán học hồi quy logistic hướng tới việc đưa ra các lý thuyết đơn giản hơn hoặc phức tạp hơn về những gì cấu thành phần mềm độc hại và phần mềm lành tính. Những phương pháp đào tạo này nằm ngoài phạm vi của cuốn sách này, nhưng nếu bạn quan tâm đến việc tìm hiểu về những phương pháp hữu ích này, tôi khuyến khích bạn truy cập Google "hồi quy logistic và chính quy hóa" và đọc giải thích về chúng trực tuyến.

Khi nào nên sử dụng hồi quy logistic

Hồi quy logistic có những ưu điểm và nhược điểm khác biệt so với các thuật toán học máy khác. Một lợi thế của hồi quy logistic là người ta có thể dễ dàng giải thích mô hình hồi quy logistic cho rằng cấu thành phần mềm lành tính và phần mềm độc hại. Ví dụ: chúng ta có thể hiểu một mô hình hồi quy logistic nhất định bằng cách xem xét các trọng số đặc trưng của nó. Các tính năng có trọng số cao là những tính năng mà mô hình diễn giải là độc hại. Các tính năng có trọng số âm là những tính năng mà mô hình tin là phần mềm lành tính. Hồi quy logistic là một cách tiếp cận khá đơn giản và khi dữ liệu bạn đang làm việc chứa các chỉ báo rõ ràng về sự độc hại, nó có thể hoạt động tốt. Nhưng khi dữ liệu phức tạp hơn, hồi quy logistic thường thất bại.

Bây giờ, hãy khám phá một phương pháp học máy đơn giản khác có thể thể hiện ranh giới quyết định phức tạp hơn nhiều: k-lắng giềng gần nhất.

K-hàng xóm gần nhất

K-hàng xóm gần nhất là một thuật toán học máy dựa trên ý tưởng rằng nếu một tệp nhị phân trong không gian tính năng gần với các tệp nhị phân độc hại khác, thì nó độc hại và nếu các tính năng của nó đặt nó gần các tệp nhị phân lành tính, thì nó phải là nhẹ. Chính xác hơn, nếu phần lớn

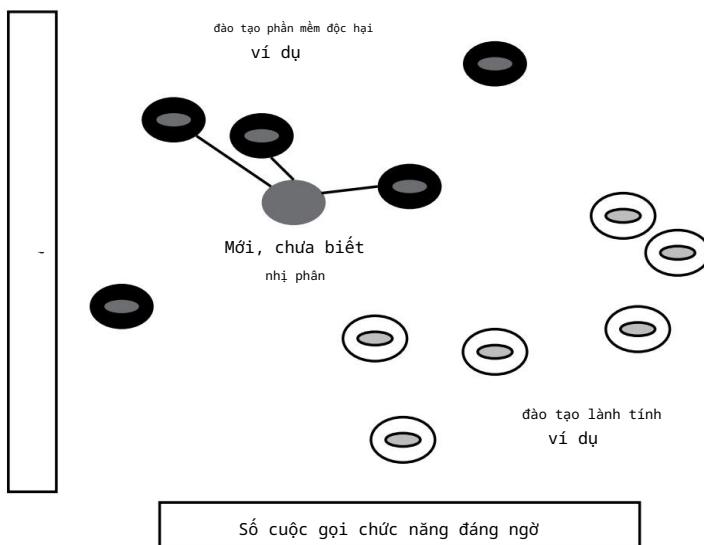
k tệp nhị phân gần nhất với tệp nhị phân không xác định là độc hại, thì tệp đó độc hại. Lưu ý rằng k đại diện cho số lượng hàng xóm lân cận mà chúng tôi chọn và tự xác định, tùy thuộc vào số lượng hàng xóm mà chúng tôi nghĩ nên tham gia vào việc xác định xem một mẫu là lành tính hay độc hại.

Trong thế giới thực, điều này có ý nghĩa trực quan. Ví dụ: nếu bạn có tập dữ liệu về cân nặng và chiều cao của cả vận động viên bóng rổ và vận động viên bóng bàn, thì rất có thể cân nặng và chiều cao của các vận động viên bóng rổ có thể gần nhau hơn so với số đo của vận động viên bóng bàn. Tương tự, trong cài đặt bảo mật, phần mềm độc hại thường sẽ có các tính năng tương tự như phần mềm độc hại khác và phần mềm lành tính thường sẽ có các tính năng tương tự như phần mềm lành tính khác.

Chúng ta có thể dịch ý tưởng này thành thuật toán k-hàng xóm gần nhất để com xác định xem tệp nhị phân là độc hại hay lành tính bằng cách sử dụng các bước sau:

1. Trích xuất các đặc trưng của nhị phân và tìm k mẫu gần nhất với nó trong không gian đặc trưng.
2. Chia số mẫu phần mềm độc hại gần với mẫu cho k để có được tỷ lệ phần trăm của những người hàng xóm gần nhất là độc hại.
3. Nếu có đủ mẫu là độc hại, hãy xác định mẫu đó là độc hại.

Hình 6-13 cho thấy thuật toán k-hàng xóm gần nhất hoạt động như thế nào ở mức cao.



Hình 6-13: Minh họa cách sử dụng k-hàng xóm gần nhất để phát hiện phần mềm độc hại chưa từng thấy trước đó

Chúng tôi thấy một tập hợp các ví dụ đào tạo về phần mềm độc hại ở phía trên bên trái và một tập hợp các ví dụ về phần mềm lành tính ở phía dưới bên phải. Chúng tôi cũng thấy một nhị phân mới, chưa biết được kết nối với ba hàng xóm gần nhất của nó. Trong trường hợp này, chúng tôi đã đặt k thành 3,

nghĩa là chúng tôi đang xem xét ba hàng xóm gần nhất với các nhị phân chưa biết. Bởi vì cả ba hàng xóm gần nhất đều độc hại, nên chúng tôi sẽ phân loại tệp nhị phân mới này là độc hại.

Toán học đằng sau hàng xóm gần nhất của K

Bây giờ chúng ta hãy thảo luận về phép toán cho phép chúng ta tính toán khoảng cách giữa các đặc trưng của các nhị phân mới, chưa biết và các mẫu trong tập huấn luyện. Chúng tôi sử dụng hàm khoảng cách để thực hiện việc này, hàm này cho chúng tôi biết khoảng cách giữa ví dụ mới của chúng tôi và các ví dụ trong tập huấn luyện. Hàm khoảng cách phổ biến nhất là khoảng cách Euclidean, là độ dài của đường đi ngắn nhất giữa hai điểm trong không gian đặc trưng của chúng ta. Liệt kê 6-2 hiển thị mã giả cho khoảng cách Euclidean trong không gian đặc trưng hai chiều mẫu của chúng ta.

nhập toán

```
def euclidean_distance(compression1,suspicious_calls1,compression2,nghi_ngor_calls2): u
    comp_ distance = (compression1-compression2)**2 v
    call_ distance = (suspicious_calls1-suspicious_calls2)**2 w
    trả về math.sqrt(comp_ distance + call_ distance) x
```

Liệt kê 6-2: Mã giả để viết hàm euclidean_distance

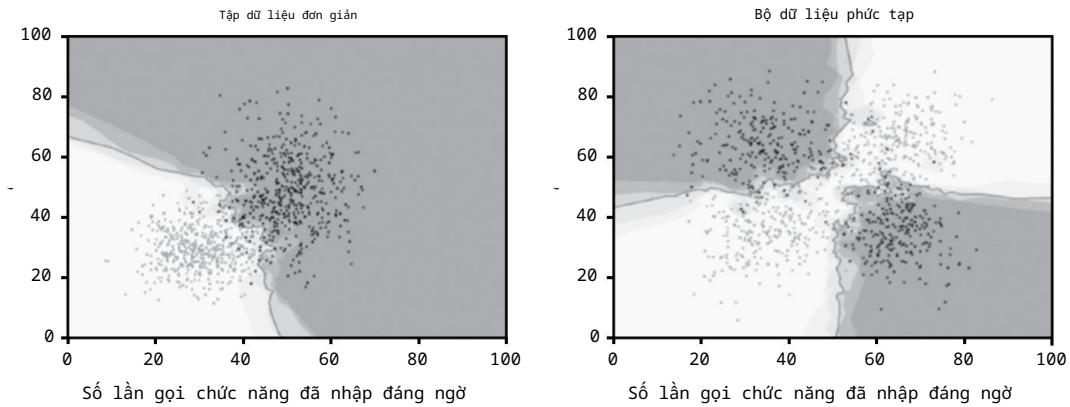
Hãy cùng tìm hiểu cách toán học trong mã này hoạt động. Liệt kê 6-2 lấy một cặp mẫu và tính toán khoảng cách giữa chúng dựa trên sự khác biệt giữa các đặc điểm của chúng. Đầu tiên, người gọi chuyển vào các tính năng của nhị phân , trong đó nén1 là tính năng nén của ví dụ đầu tiên, nghi ngor_calls1 là tính năng đáng ngờ_gọi của ví dụ đầu tiên, nén2 là tính năng nén của ví dụ thứ hai và nghi ngor_calls2 là các cuộc gọi đáng ngờ đặc điểm của ví dụ thứ hai.

Sau đó, chúng tôi tính toán sự khác biệt bình phương giữa các tính năng nén của từng mẫu và chúng tôi tính toán sự khác biệt bình phương giữa các tính năng cuộc gọi đáng ngờ của từng mẫu . Chúng tôi sẽ không đề cập đến lý do chúng tôi sử dụng khoảng cách bình phương, nhưng lưu ý rằng sự khác biệt kết quả luôn luôn dương. Cuối cùng, chúng ta tính căn bậc hai của hai hiệu số, là khoảng cách Euclidean giữa hai vectơ đặc trưng, và trả về hàm gọi . Mặc dù có nhiều cách khác để tính khoảng cách giữa các ví dụ, nhưng khoảng cách Euclidean được sử dụng phổ biến nhất với thuật toán k-láng giềng gần nhất và nó hoạt động tốt cho các vấn đề về khoa học dữ liệu bảo mật.

Chọn số lượng hàng xóm bầu chọn

Bây giờ chúng ta hãy xem xét các loại ranh giới quyết định và xác suất mà thuật toán k-láng giềng gần nhất tạo ra cho các bộ dữ liệu mẫu mà chúng ta đang sử dụng trong chương này. Trong Hình 6-14, tôi đặt k thành 5, do đó cho phép năm người hàng xóm gần nhất “bỏ phiếu”.

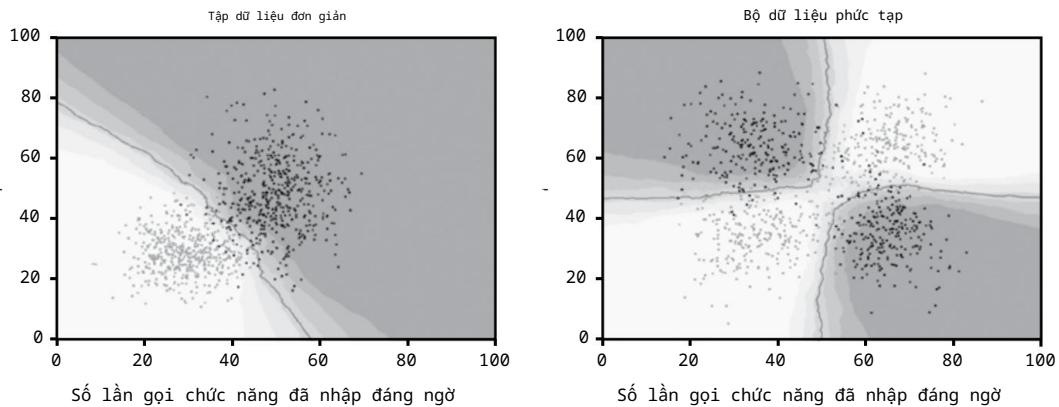
K-Hàng Xóm Gần Nhất, 5 Hàng Xóm



Hình 6-14: Ranh giới quyết định được tạo bởi k-hàng xóm gần nhất khi k được đặt thành 5

Nhưng trong Hình 6-15, tôi đặt k là 50, cho phép 50 người hàng xóm gần nhất "bỏ phiếu".

K-Hàng Xóm Gần Nhất, 50 Hàng Xóm



Hình 6-15: Ranh giới quyết định được tạo bởi k-hàng xóm gần nhất khi k được đặt thành 50

Lưu ý sự khác biệt đáng kể giữa các mô hình tùy thuộc vào số lượng hàng xóm bỏ phiếu. Mô hình trong Hình 6-14 cho thấy một ranh giới quyết định phức tạp và cục bộ cho cả hai tập dữ liệu, quá khớp theo nghĩa là nó vẽ ra các ranh giới quyết định cục bộ xung quanh các ngoại lệ, nhưng lại không phù hợp vì nó không nắm bắt được các xu hướng chung, đơn giản. Ngược lại, mô hình trong Hình 6-15 rất phù hợp với cả hai bộ dữ liệu, bởi vì nó không bị phân tâm bởi các yếu tố ngoại lai và xác định rõ ràng các xu hướng chung.

Như bạn có thể thấy, k-hàng xóm gần nhất có thể tạo ra ranh giới quyết định phức tạp hơn nhiều so với hồi quy logistic. Chúng ta có thể kiểm soát độ phức tạp của ranh giới này để bảo vệ chống lại việc trang bị thừa và trang bị thiếu bằng cách thay đổi k , số lượng láng giềng có quyền biểu quyết xem một mẫu có phù hợp hay không.

độc hại hay lành tính. Trong khi mô hình hồi quy logistic trong Hình 6-11 hoàn toàn sai, k-hàng xóm gần nhất làm tốt việc tách phần mềm độc hại khỏi phần mềm lành tính, đặc biệt là khi chúng tôi để 50 người hàng xóm bỏ phiếu.

Bởi vì k-hàng xóm gần nhất không bị hạn chế bởi cấu trúc tuyến tính và chỉ đơn giản là nhìn vào những hàng xóm gần nhất của mỗi điểm để đưa ra quyết định, nó có thể tạo ra các ranh giới quyết định với các hình dạng tùy ý, do đó mô hình hóa các bộ dữ liệu phức tạp hiệu quả hơn nhiều.

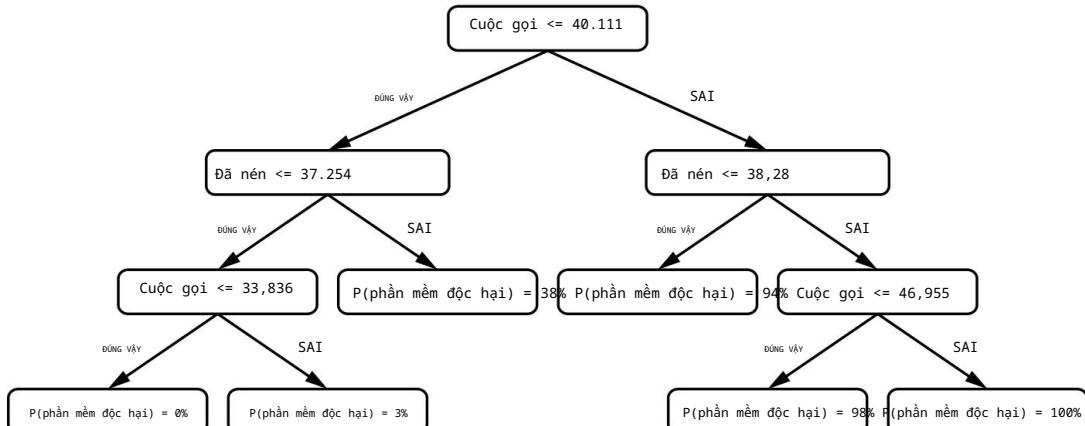
Khi nào nên sử dụng K-Hàng xóm gần nhất

K-hàng xóm gần nhất là một thuật toán tốt để xem xét khi bạn có dữ liệu trong đó các tính năng không ánh xạ rõ ràng vào khái niệm về sự đáng ngờ, nhưng sự gần gũi với các mẫu độc hại là một chỉ báo mạnh mẽ về mức độ độc hại. Ví dụ: nếu bạn đang cố gắng phân loại phần mềm độc hại thành các họ chia sẻ mã, k-hàng xóm gần nhất có thể là một thuật toán tốt để thử, vì bạn muốn phân loại một mẫu phần mềm độc hại thành một họ nếu các tính năng của phần mềm đó giống với các thành viên đã biết của một nhóm. cho gia đình.

Một lý do khác để sử dụng k-hàng xóm gần nhất là nó cung cấp các giải thích rõ ràng về lý do tại sao nó đưa ra quyết định phân loại nhất định. Nói cách khác, thật dễ dàng để xác định và so sánh sự giống nhau giữa các mẫu và một mẫu chưa biết để tìm ra lý do tại sao thuật toán lại phân loại nó là phần mềm độc hại hoặc phần mềm lành tính.

Cây quyết định

Cây quyết định là một phương pháp học máy thường được sử dụng khác để giải các bài toán phát hiện. Cây quyết định tự động tạo ra một loạt câu hỏi thông qua quy trình đào tạo để quyết định xem một tệp nhị phân nhất định có phải là phần mềm độc hại hay không, tương tự như trò chơi Twenty Questions. Hình 6-16 cho thấy một cây quyết định mà tôi đã tự động tạo ra bằng cách huấn luyện nó trên tập dữ liệu đơn giản mà chúng ta đã sử dụng trong chương này. Hãy theo dòng logic trong cây.

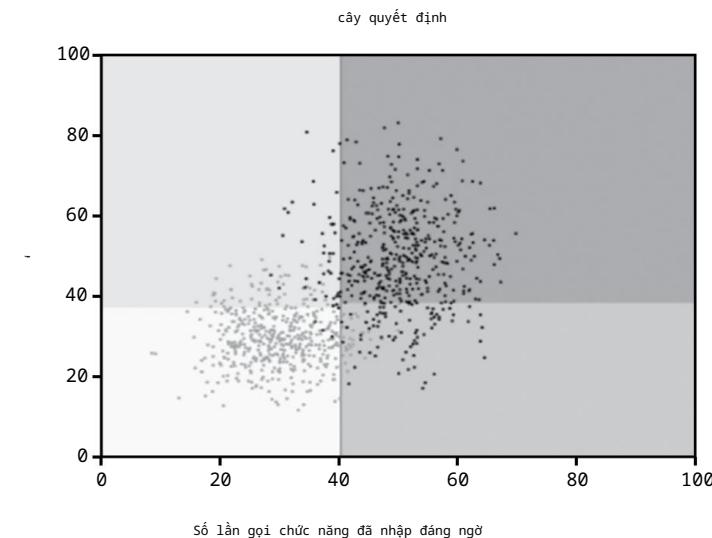


Hình 6-16: Một cây quyết định đã học cho ví dụ tập dữ liệu đơn giản của chúng tôi

Luồng cây quyết định bắt đầu khi chúng tôi nhập các tính năng mà chúng tôi đã trích xuất từ một nhị phân mới, chưa từng thấy trước đây vào cây. Sau đó, cây xác định một loạt các câu hỏi để hỏi các tính năng của nhị phân này. Hộp ở trên cùng của cây, mà chúng ta gọi là nút, đặt câu hỏi đầu tiên: số lần gọi đúng ngở trong cây có nhỏ hơn hoặc bằng 40.111 không? Lưu ý rằng cây quyết định sử dụng số dấu phẩy động ở đây vì chúng tôi đã chuẩn hóa số lượng lệnh gọi đúng ngở trong mỗi nhị phân thành phạm vi từ 0 đến 100.

Nếu câu trả lời là "có", chúng tôi hỏi một câu hỏi khác: phần trăm dữ liệu được nén trong tệp có nhỏ hơn hoặc bằng 37,254 không? Nếu câu trả lời là "có", thì chúng ta chuyển sang câu hỏi tiếp theo: số lượng cuộc gọi đúng ngở trong hệ nhị phân có nhỏ hơn hoặc bằng 33,836 không? Nếu câu trả lời là "có", chúng ta sẽ đi đến cuối cây quyết định. Tại thời điểm này, xác suất nhị phân là phần mềm độc hại là 0 phần trăm.

Hình 6-17 cho thấy một giải thích hình học của cây quyết định này.



Hình 6-17: Ranh giới quyết định được tạo bởi cây quyết định cho ví dụ tập dữ liệu đơn giản của chúng tôi

Ở đây, các vùng được tô bóng cho biết nơi cây quyết định cho rằng các mẫu là độc hại. Các vùng sáng hơn cho biết nơi cây quyết định cho rằng các mẫu lành tính. Xác suất được xác định bởi chuỗi câu hỏi và câu trả lời trong Hình 6-16 phải tương ứng với xác suất trong các vùng tô đậm trong Hình 6-17.

Chọn một nút gốc tốt

Vậy làm cách nào để chúng ta sử dụng thuật toán học máy để tạo cây quyết định như thế này từ dữ liệu huấn luyện? Ý tưởng cơ bản là cây quyết định bắt đầu với một câu hỏi ban đầu được gọi là nút gốc. Nút gốc tốt nhất là nút mà chúng tôi nhận được câu trả lời "có" cho hầu hết nếu không phải tất cả các mẫu của một loại và câu trả lời "không" cho hầu hết nếu không phải tất cả các mẫu của loại kia. Ví dụ,

trong Hình 6-16, câu hỏi về nút gốc hỏi liệu một nhị phân chưa từng thấy trước đây có 40.111 lệnh gọi trả xuống hay không. (Lưu ý rằng số lượng cuộc gọi trên mỗi nhị phân ở đây được chuẩn hóa theo tỷ lệ từ 0 đến 100, làm cho các giá trị dấu phẩy động hợp lệ.) Như bạn có thể thấy từ đường thẳng đứng trong Hình 6-17, hầu hết dữ liệu lành tính có ít hơn con số này, trong khi hầu hết dữ liệu phần mềm độc hại có nhiều hơn số cuộc gọi đáng ngờ này, khiến đây là một câu hỏi ban đầu hay.

Chọn câu hỏi tiếp theo

Sau khi chọn nút gốc, hãy chọn các câu hỏi tiếp theo bằng cách sử dụng phương pháp tương tự như phương pháp chúng ta đã sử dụng để chọn nút gốc. Ví dụ: nút gốc cho phép chúng tôi chia các mẫu thành hai nhóm: một nhóm có ít hơn hoặc bằng 40.111 cuộc gọi đáng ngờ (không gian tính năng âm) và một nhóm khác có hơn 40.111 cuộc gọi đáng ngờ (không gian tính năng dương). Để chọn câu hỏi tiếp theo, chúng ta chỉ cần các câu hỏi sẽ phân biệt rõ hơn các mẫu trong từng khu vực của không gian đặc trưng thành các ví dụ huấn luyện độc hại và lành tính.

Chúng ta có thể thấy điều này trong cách cấu trúc cây quyết định trong Hình 6-16 và 6-17. Ví dụ, Hình 6-16 cho thấy rằng sau khi chúng tôi hỏi một câu hỏi "gốc" ban đầu về số lượng lệnh gọi đáng ngờ mà các tệp nhị phân thực hiện, chúng tôi đặt câu hỏi về lượng dữ liệu nén mà các tệp nhị phân có. Hình 6-17 cho thấy lý do tại sao chúng tôi làm điều này dựa trên dữ liệu: sau khi chúng tôi đặt câu hỏi đầu tiên về các lệnh gọi hàm đáng ngờ, chúng tôi có một ranh giới quyết định sơ bộ phân tách hầu hết phần mềm độc hại khỏi hầu hết phần mềm lành tính trong sơ đồ. Làm thế nào chúng ta có thể tinh chỉnh ranh giới quyết định hơn nữa bằng cách đặt câu hỏi tiếp theo? Rõ ràng là câu hỏi hay nhất tiếp theo sẽ tinh chỉnh ranh giới quyết định của chúng ta, sẽ là về lượng dữ liệu nén trong các tệp nhị phân.

Khi nào nên ngừng đặt câu hỏi

Tại một số thời điểm trong quá trình tạo cây quyết định, chúng tôi cần quyết định khi nào cây quyết định nên ngừng đặt câu hỏi và chỉ cần xác định xem tệp nhị phân là lành tính hay độc hại dựa trên sự chắc chắn về câu trả lời của chúng tôi. Một cách đơn giản là giới hạn số lượng câu hỏi mà cây quyết định của chúng ta có thể hỏi hoặc giới hạn độ sâu của nó (số lượng câu hỏi tối đa mà chúng ta có thể đặt ra cho bất kỳ hệ nhị phân nào). Một cách khác là cho phép cây quyết định tiếp tục phát triển cho đến khi chúng tôi hoàn toàn chắc chắn về việc liệu mọi ví dụ trong tập huấn luyện của chúng tôi có phải là phần mềm độc hại hay phần mềm lành tính hay không dựa trên cấu trúc của cây.

Ưu điểm của việc hạn chế kích thước của cây là nếu cây càng đơn giản, chúng ta càng có cơ hội trả lời đúng (hay nghĩ đến chiếc dao cạo của Occam - lý thuyết càng đơn giản càng tốt). Nói cách khác, sẽ ít có khả năng cây quyết định khớp dữ liệu huấn luyện hơn nếu chúng ta giữ nó nhỏ.

Ngược lại, việc cho phép cây phát triển đến kích thước tối đa có thể hữu ích nếu chúng ta không trang bị đầy đủ dữ liệu huấn luyện. Ví dụ: cho phép cây phát triển hơn nữa sẽ làm tăng độ phức tạp của ranh giới quyết định, điều mà chúng ta sẽ muốn thực hiện nếu chúng ta không phù hợp. Nhìn chung, những người thực hành máy học

thường thử nhiều độ sâu hoặc cho phép độ sâu tối đa trên các tệp nhị phân chưa từng thấy trước đó, lặp lại quá trình này cho đến khi chúng nhận được kết quả chính xác nhất.

Sử dụng mã giả để khám phá các thuật toán tạo cây quyết định

Bây giờ hãy kiểm tra thuật toán tạo cây quyết định tự động. Bạn đã học được rằng ý tưởng cơ bản dồn sau thuật toán này là tạo nút gốc trong cây bằng cách tìm câu hỏi làm tăng mức độ chắc chắn của chúng ta về việc liệu các ví dụ huấn luyện là độc hại hay lành tính, sau đó tìm các câu hỏi tiếp theo sẽ làm tăng mức độ chắc chắn của chúng ta hơn nữa. Thuật toán sẽ ngừng đặt câu hỏi và đưa ra quyết định khi sự chắc chắn của nó về các ví dụ đào tạo đã vượt qua một số ngưỡng mà chúng tôi đặt trước.

Theo lập trình, chúng ta có thể làm điều này một cách đệ quy. Mã giả giống Python trong Liệt kê 6-3 cho thấy toàn bộ quy trình để xây dựng một cây quyết định ở dạng đơn giản hóa.

```
cây = Cây()
def add_question(đào tạo ví dụ):
    câu hỏi = pick_best_question(training_examples)
    không chắc chắn_yes, yes_samples = ask_question(câu hỏi, huấn luyện ví dụ, "có")
    không chắc chắn_no, no_samples = ask_question(câu hỏi, huấn luyện ví dụ, "không")
    nếu không phải là không chắc chắn_có < MIN_UNCERTAINTY:
        add_question(yes_samples)
    nếu không phải là không chắc chắn_không < MIN_UNCERTAINTY:
        add_question(no_samples)
    add_question(huấn luyện ví dụ)
```

Liệt kê 6-3: Mã giả để xây dựng thuật toán cây quyết định

Mã giả thêm đệ quy các câu hỏi vào cây quyết định, bắt đầu với nút gốc và tiếp tục đi xuống cho đến khi thuật toán cảm thấy tin rằng cây quyết định có thể đưa ra câu trả lời rất chắc chắn về việc một tệp mới là lành tính hay độc hại.

Khi chúng tôi bắt đầu tạo cây, chúng tôi sử dụng `pick_best_question()` để chọn nút gốc (hiện tại, đừng lo lắng về cách thức hoạt động của chức năng này). Sau đó, chúng tôi xem xét mức độ không chắc chắn hiện tại của chúng tôi về các mẫu đào tạo mà câu trả lời là “có” cho câu hỏi ban đầu này . Điều này sẽ giúp chúng tôi quyết định xem chúng tôi có cần tiếp tục đặt câu hỏi về các mẫu này hay không hoặc liệu chúng tôi có thể dừng lại và dự đoán xem các mẫu đó là độc hại hay lành tính. Chúng tôi làm tương tự với những mẫu mà chúng tôi đã trả lời “không” cho câu hỏi ban đầu .

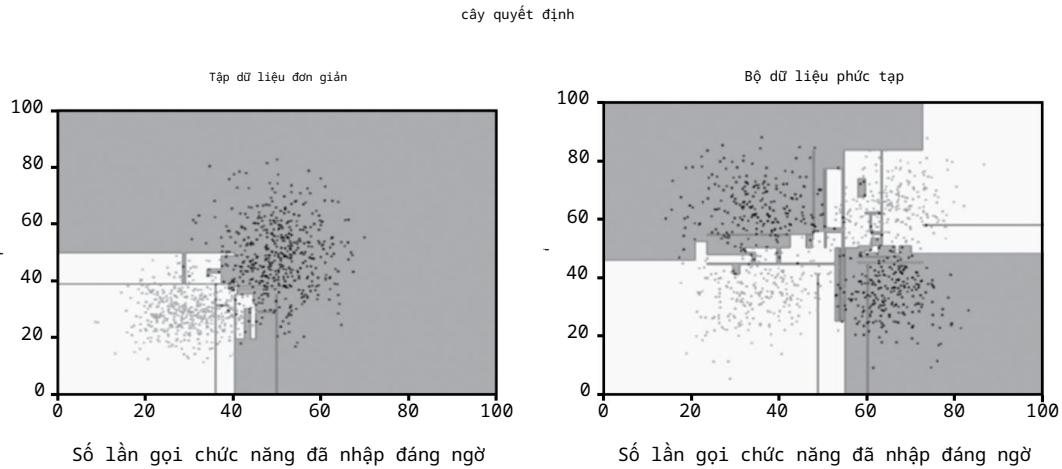
Tiếp theo, chúng tôi kiểm tra xem chúng tôi có sự không chắc chắn về các mẫu mà chúng tôi đã trả lời “có” (độ không chắc chắn_có) đủ thấp để quyết định liệu chúng có hại hay lành tính . Nếu chúng tôi có thể xác định liệu chúng là độc hại hay lành tính vào thời điểm này, thì chúng tôi sẽ không hỏi thêm bất kỳ câu hỏi nào. Nhưng nếu không thể, chúng tôi gọi lại `add_question()`, sử dụng `yes_samples` hoặc số lượng mẫu mà chúng tôi đã trả lời “có” làm đầu vào. Đây là một ví dụ cổ điển về đệ quy, là một hàm gọi chính nó. Chúng tôi đang sử dụng đệ quy để

lặp lại quy trình tương tự mà chúng tôi đã thực hiện cho nút gốc với một tập hợp con các ví dụ đào tạo. Câu lệnh if tiếp theo thực hiện tương tự cho các ví dụ "không" của chúng ta. Cuối cùng, chúng ta gọi hàm xây dựng cây quyết định trên các ví dụ huấn luyện của mình là

Cách hoạt động chính xác của pick_best_question() liên quan đến toán học nằm ngoài phạm vi của cuốn sách này, nhưng ý tưởng thì đơn giản. Để chọn câu hỏi hay nhất tại bất kỳ thời điểm nào trong quá trình xây dựng cây quyết định, chúng tôi xem xét các ví dụ đào tạo mà chúng tôi vẫn chưa chắc chắn, liệt kê tất cả các câu hỏi chúng tôi có thể đặt ra về chúng, sau đó chọn câu hỏi làm giảm sự không chắc chắn của chúng tôi tốt nhất về việc liệu các ví dụ là phần mềm độc hại hay phần mềm lành tính. Chúng tôi đo lường mức giảm độ không chắc chắn này bằng cách sử dụng phép đo thống kê được gọi là độ lợi thông tin. Phương pháp chọn câu hỏi hay nhất đơn giản này hoạt động hiệu quả một cách đáng ngạc nhiên.

LƯU Ý là một ví dụ đơn giản về cách hoạt động của các thuật toán máy học, tạo cây quyết định, trong thế giới thực. Tôi đã bỏ qua phép toán cần thiết để tính toán mức độ mà một câu hỏi đưa ra làm tăng mức độ chắc chắn của chúng ta về việc liệu một tệp có bị lỗi hay không.

Bây giờ chúng ta hãy xem hành vi của cây quyết định trên hai tập dữ liệu mẫu mà chúng ta đã sử dụng trong chương này. Hình 6-18 cho thấy giới hạn quyết định được học bởi bộ phát hiện cây quyết định.



Hình 6-18: Ranh giới quyết định cho bộ dữ liệu mẫu của chúng tôi được tạo ra bằng cách tiếp cận cây quyết định

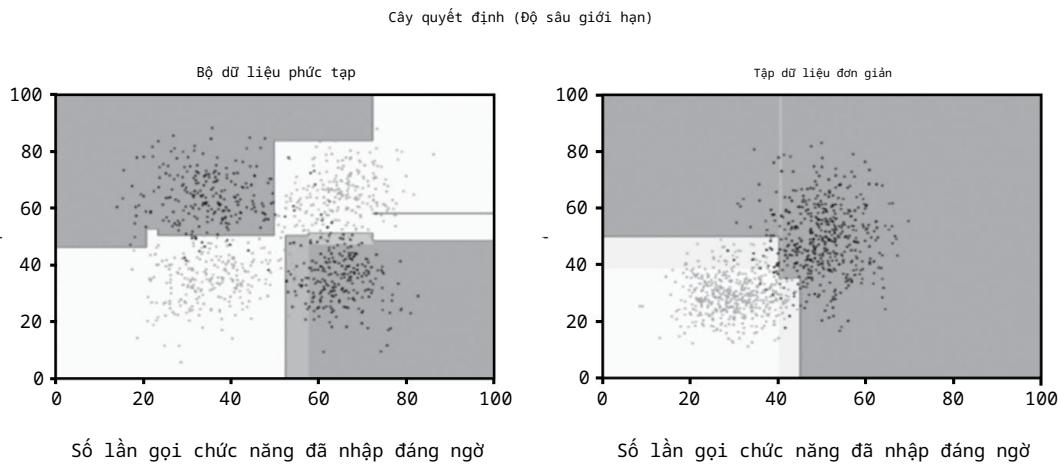
Trong trường hợp này, thay vì đặt độ sâu tối đa cho cây, chúng tôi cho phép chúng phát triển đến mức không có kết quả dương tính giả hoặc âm tính giả so với dữ liệu huấn luyện để mọi mẫu huấn luyện đều được phân loại chính xác.

Lưu ý rằng các cây quyết định chỉ có thể vẽ các đường ngang và dọc trong không gian đặc trưng, ngay cả khi có vẻ rõ ràng và hiển nhiên rằng đường cong hoặc đường chéo có thể phù hợp hơn. Điều này là do cây quyết định

chỉ cho phép chúng tôi thể hiện các điều kiện đơn giản trên các tính năng riêng lẻ (chẳng hạn như lớn hơn hoặc bằng và nhỏ hơn hoặc bằng), điều này luôn dẫn đến các đường ngang hoặc dọc.

Bạn cũng có thể thấy rằng mặc dù các cây quyết định trong các ví dụ này thành công trong việc tách phần mềm lành tính khỏi phần mềm độc hại, nhưng các cây quyết định có vẻ rất bất thường và có các tạo tác kỳ lạ. Ví dụ: vùng phần mềm độc hại mở rộng sang vùng phần mềm lành tính theo những cách kỳ lạ và ngược lại. Về mặt tích cực, cây quyết định làm tốt hơn nhiều so với hồi quy logistic trong việc tạo ranh giới quyết định cho tập dữ liệu phức tạp.

Bây giờ chúng ta hãy so sánh các cây quyết định trong Hình 6-18 với các mô hình cây quyết định trong Hình 6-19.



Hình 6-19: Ranh giới quyết định cho bộ dữ liệu mẫu của chúng tôi được tạo bởi cây quyết định có độ sâu giới hạn

Các cây quyết định trong Hình 6-19 sử dụng cùng một cách tạo cây quyết định thuật toán được sử dụng cho Hình 6-18, ngoại trừ việc tối ưu hóa độ sâu của cây ở năm nút. Điều này có nghĩa là đối với bất kỳ nhị phân cụ thể nào, cây có thể hỏi tối đa năm câu hỏi về các tính năng của nó.

Kết quả là kịch tính. Trong khi các mô hình cây quyết định trong Hình 6-18 rõ ràng là quá phù hợp, tập trung vào các giá trị ngoại lệ và vẽ ra các ranh giới quá phức tạp không nắm bắt được xu hướng chung, thì các cây quyết định trong Hình 6-19 phù hợp với dữ liệu một cách tinh tế hơn nhiều, xác định một xu hướng chung. mẫu trong cả hai bộ dữ liệu mà không tập trung vào các giá trị ngoại lệ (với một ngoại lệ, vùng quyết định da nhỏ hơn ở khu vực phía trên bên phải của bộ dữ liệu đơn giản). Như bạn có thể thấy, việc chọn độ sâu cây quyết định tối đa tốt có thể có ảnh hưởng lớn đến trình phát hiện học máy dựa trên cây quyết định của bạn.

Khi nào nên sử dụng cây quyết định

Bởi vì cây quyết định mang tính biểu cảm và đơn giản, nên chúng có thể tìm hiểu cả ranh giới đơn giản và ranh giới bất thường cao dựa trên các câu hỏi có hoặc không đơn giản. Chúng tôi cũng có thể đặt độ sâu tối đa để kiểm soát mức độ đơn giản hoặc phức tạp của lý thuyết của họ về yêu tố câu thành phần mềm độc hại so với phần mềm lành tính.

Thật không may, nhược điểm của cây quyết định là chúng thường không dẫn đến các mô hình rất chính xác. Lý do cho điều này rất phức tạp, nhưng nó liên quan đến thực tế là các cây quyết định thể hiện các ranh giới quyết định lỏng chõm, không phù hợp với dữ liệu huấn luyện theo cách tổng quát hóa rất tốt cho các ví dụ chưa từng thấy trước đây.

Tương tự như vậy, cây quyết định thường không tìm hiểu xác suất chính xác xung quanh ranh giới quyết định của chúng. Chúng ta có thể thấy điều này bằng cách kiểm tra các vùng bóng mờ xung quanh ranh giới quyết định trong Hình 6-19. Quá trình phân rã không diễn ra tự nhiên hoặc từ từ và không xảy ra ở những khu vực mà lẽ ra nó phải xảy ra -những khu vực nơi các ví dụ về phần mềm độc hại và phần mềm độc hại chồng lên nhau.

Tiếp theo, tôi thảo luận về cách tiếp cận rừng ngẫu nhiên, kết hợp nhiều cây quyết định để mang lại kết quả tốt hơn nhiều.

rừng ngẫu nhiên

Mặc dù công đồng bảo mật chủ yếu dựa vào cây quyết định để phát hiện phần mềm độc hại, nhưng hầu như họ không bao giờ sử dụng chúng một cách riêng lẻ. Thay vào đó, hàng trăm hoặc hàng nghìn cây quyết định được sử dụng phối hợp để phát hiện thông qua một phương pháp gọi là rừng ngẫu nhiên. Thay vì đào tạo một cây quyết định, chúng tôi đào tạo nhiều cây, thường là hàng trăm hoặc nhiều hơn, nhưng chúng tôi đào tạo mỗi cây quyết định theo cách khác nhau để nó có một góc nhìn khác về dữ liệu. Cuối cùng, để quyết định xem một nhị phân mới là độc hại hay lành tính, chúng tôi cho phép các cây quyết định bỏ phiếu. Xác suất một tệp nhị phân là phần mềm độc hại là số phiếu tích cực chia cho tổng số cây.

Tất nhiên, nếu tất cả các cây quyết định giống hệt nhau, thì tất cả chúng sẽ bỏ phiếu theo cùng một cách và rừng ngẫu nhiên sẽ chỉ sao chép kết quả của các cây quyết định riêng lẻ. Để giải quyết vấn đề này, chúng tôi muốn các cây quyết định có các quan điểm khác nhau về những gì cấu thành phần mềm độc hại và phần mềm lành tính, đồng thời chúng tôi sử dụng hai phương pháp mà tôi sẽ thảo luận tiếp theo, để đưa sự đa dạng này vào bộ sưu tập cây quyết định của chúng tôi. Bằng cách tạo ra sự đa dạng, chúng tôi tạo ra động lực “trí tuệ của đám đông” trong mô hình của mình, điều này thường dẫn đến một mô hình chính xác hơn.

Chúng tôi sử dụng các bước sau để tạo thuật toán rừng ngẫu nhiên:

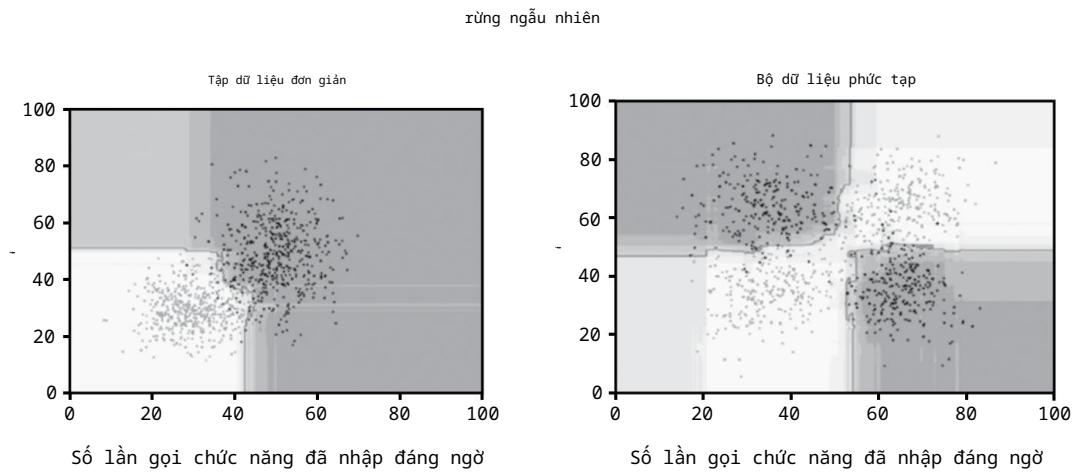
1. Đào tạo: cho mỗi cây trong số chúng tôi dự định tạo (thường là 100 cây trở lên)

- Lấy mẫu ngẫu nhiên một số ví dụ huấn luyện từ tập huấn luyện của chúng ta.
- Xây dựng cây quyết định từ mẫu ngẫu nhiên.
- Đối với mỗi cây mà chúng tôi xây dựng, mỗi lần chúng tôi cần nhắc việc “đặt câu hỏi”, hãy xem xét chỉ đặt câu hỏi về một số đặc điểm và bỏ qua các đặc điểm khác.

2. Phát hiện trên nhị phân chưa từng thấy trước đây

- Chạy phát hiện cho từng cây riêng lẻ trên hệ nhị phân.
- Quyết định xem tệp nhị phân có phải là phần mềm độc hại hay không dựa trên số lượng cây đã bình chọn là “có”.

Để hiểu điều này chi tiết hơn, hãy xem xét các kết quả được tạo bằng cách tiếp cận rừng ngẫu nhiên trên hai bộ dữ liệu mẫu của chúng tôi, như trong Hình 6-20. Những kết quả này được tạo ra bằng cách sử dụng 100 cây quyết định.



Hình 6-20: Các ranh giới quyết định được tạo bằng phương pháp rừng ngẫu nhiên

Trái ngược với kết quả cây quyết định riêng lẻ được hiển thị trong Hình 6-18 và 6-19, rừng ngẫu nhiên có thể biểu thị các ranh giới quyết định trực quan và mượt mà hơn cho cả tập dữ liệu đơn giản và phức tạp so với các cây quyết định riêng lẻ. Thật vậy, mô hình rừng ngẫu nhiên phù hợp với tập dữ liệu huấn luyện rất rõ ràng, không có cạnh lởm chởm; mô hình dường như đã học được những lý thuyết tốt về những gì cấu thành “độc hại so với lành tính” cho cả hai bộ dữ liệu.

Ngoài ra, các khu vực bóng mờ là trực quan. Ví dụ: bạn càng thu thập được nhiều thông tin từ các ví dụ lành tính hoặc độc hại, thì ran dom forest càng ít chắc chắn hơn về việc liệu các ví dụ đó là độc hại hay lành tính. Điều này báo hiệu tốt cho hiệu suất của rừng ngẫu nhiên trên các nhị phân chưa từng thấy trước đây. Trên thực tế, như bạn sẽ thấy trong chương tiếp theo, khu rừng ngẫu nhiên là mô hình hoạt động tốt nhất trên các nhị phân chưa từng thấy trước đây trong tất cả các phương pháp được thảo luận trong chương này.

Để hiểu tại sao rừng ngẫu nhiên vẽ ranh giới quyết định rõ ràng như vậy so với các cây quyết định riêng lẻ, hãy nghĩ xem 100 cây quyết định đang làm gì. Mỗi cây chỉ nhìn thấy khoảng hai phần ba dữ liệu huấn luyện và chỉ được xem xét một tính năng được chọn ngẫu nhiên bất cứ khi nào nó đưa ra quyết định về câu hỏi sẽ hỏi. Điều này có nghĩa là đằng sau hậu trường, chúng ta có 100 ranh giới quyết định khác nhau được tính trung bình để tạo ranh giới quyết định cuối cùng trong các ví dụ (và các vùng được tách biệt).

Động lực “trí tuệ của đám đông” này tạo ra một ý kiến tổng hợp có thể xác định các xu hướng trong dữ liệu theo cách tinh vi hơn nhiều so với các cây quyết định riêng lẻ có thể làm được.

Bản tóm tắt

Trong chương này, bạn đã có phần giới thiệu cấp cao về máy học-phát hiện phần mềm độc hại dựa trên cũng như bốn cách tiếp cận chính đối với máy học: hồi quy logistic, k-láng giềng gần nhất, cây quyết định và rồng dom chạy. Các hệ thống phát hiện dựa trên máy học có thể tự động hóa công việc viết chữ ký phát hiện và chúng thường hoạt động tốt hơn trong thực tế so với chữ ký viết tay chính.

Trong các chương tiếp theo, tôi sẽ chỉ cho bạn thấy các cách tiếp cận này hoạt động như thế nào về các vấn đề phát hiện phần mềm độc hại trong thế giới thực. Cụ thể, bạn sẽ học cách sử dụng phần mềm mã nguồn mở, máy học để xây dựng trình phát hiện máy học nhằm phân loại chính xác các tệp là độc hại hoặc lành tính và cách sử dụng số liệu thống kê cơ bản để đánh giá hiệu suất của trình phát hiện của bạn trên các tệp nhị phân chưa từng thấy trước đây.

7

E định giá phần mềm M

Hệ thống phát hiện



Trong chương trước, bạn đã biết cách máy học có thể giúp bạn xây dựng bộ phát hiện phần mềm độc hại. Trong chương này, bạn tìm hiểu các khái niệm cơ bản cần thiết để dự đoán cách

hệ thống phát hiện phần mềm độc hại sẽ thực hiện. Những ý tưởng mà bạn học được ở đây sẽ chứng minh tính quan trọng trong việc cài thiện bất kỳ hệ thống phát hiện phần mềm độc hại nào mà bạn xây dựng, bởi vì nếu không có cách đo lường hiệu suất của hệ thống, bạn sẽ không biết cách cài thiện nó. Xin lưu ý rằng mặc dù chương này được dành riêng để giới thiệu các khái niệm đánh giá cơ bản, nhưng Chương 8 vẫn tiếp tục chủ đề này, giới thiệu các khái niệm đánh giá thiết yếu như xác thực chéo.

Đầu tiên, tôi giới thiệu các ý tưởng cơ bản đằng sau việc đánh giá độ chính xác của phát hiện, sau đó tôi giới thiệu các ý tưởng nâng cao hơn liên quan đến môi trường mà bạn triển khai hệ thống của mình khi đánh giá hiệu suất của nó. Để làm điều này, tôi sẽ hướng dẫn bạn đánh giá một hệ thống phát hiện phần mềm độc hại giả định.

Bốn kết quả phát hiện có thể xảy ra

Giả sử bạn chạy hệ thống phát hiện phần mềm độc hại trên tệp nhị phân phần mềm và nhận được "ý kiến" của hệ thống về việc liệu tệp nhị phân đó là độc hại hay lành tính. Như minh họa trong Hình 7-1, bốn kết quả có thể xảy ra.

	Ví dụ là độc hại	Ví dụ không phải độc hại
↔	Đúng tích cực	dương tính giả
-	Âm tính giả	Âm tính thật

Hình 7-1: Bốn kết quả phát hiện có thể xảy ra

Những kết quả này có thể được định nghĩa như sau:

Đúng tích cực Nhị phân là phần mềm độc hại và hệ thống cho biết đó là phần mềm độc hại.

Âm tính giả Nhị phân là phần mềm độc hại và hệ thống cho biết đó không phải là phần mềm độc hại.

Dương tính giả Tệp nhị phân không phải là phần mềm độc hại và hệ thống cho biết đó là phần mềm độc hại.

Đúng âm Bản nhị phân không phải là phần mềm độc hại và hệ thống cho biết đó không phải là phần mềm độc hại.

Như bạn có thể thấy, có hai tình huống trong đó hệ thống phát hiện phần mềm độc hại của bạn có thể tạo ra kết quả không chính xác: âm tính giả và dương tính giả. Trong thực tế, kết quả tích cực thực sự và tiêu cực thực sự là những gì chúng ta mong muốn, nhưng chúng thường khó đạt được.

Bạn sẽ thấy các thuật ngữ này được sử dụng xuyên suốt chương này. Trên thực tế, hầu hết lý thuyết đánh giá phát hiện được xây dựng trên từ vựng đơn giản này.

Tỷ lệ tích cực đúng và sai

Bây giờ, giả sử bạn muốn kiểm tra độ chính xác của hệ thống phát hiện bằng cách sử dụng một bộ phần mềm lành tính và phần mềm độc hại. Bạn có thể chạy trình phát hiện trên mỗi nhị phân và đếm xem kết quả nào trong số bốn kết quả có thể xảy ra mà trình phát hiện mang lại cho bạn trên toàn bộ tập kiểm tra. Tại thời điểm này, bạn cần một số thống kê tổng tắt để cung cấp cho bạn cảm nhận tổng thể về độ chính xác của hệ thống (nghĩa là khả năng hệ thống của bạn tạo ra kết quả dương tính giả hoặc âm tính giả là bao nhiêu).

Một thống kê tóm tắt như vậy là tỷ lệ dương tính thực sự của hệ thống phát hiện mà bạn có thể tính toán bằng cách chia số lượng dương tính thực sự trong bộ thử nghiệm của mình cho tổng số mẫu phần mềm độc hại trong bộ thử nghiệm của bạn.

Vì chỉ số này tính toán tỷ lệ phần trăm mẫu phần mềm độc hại mà hệ thống của bạn có thể phát hiện nên nó đo khả năng nhận dạng phần mềm độc hại của hệ thống khi hệ thống “thấy” phần mềm độc hại.

Tuy nhiên, chỉ cần biết rằng hệ thống phát hiện của bạn sẽ đưa ra cảnh báo khi thấy phần mềm độc hại là không đủ để đánh giá độ chính xác của nó. Ví dụ: nếu bạn chỉ sử dụng tỷ lệ tích cực thực sự làm tiêu chí đánh giá, thì một chức năng đơn giản cho biết “có, đây là phần mềm độc hại” trên tất cả các tệp sẽ mang lại tỷ lệ tích cực thực sự hoàn hảo. Thử nghiệm thực sự của một hệ thống phát hiện là liệu nó có nói “có, đây là phần mềm độc hại” khi nhìn thấy phần mềm độc hại và “không, đây không phải là phần mềm độc hại” khi nhìn thấy phần mềm độc hại.

Để đo khả năng phân biệt thứ gì đó không phải là phần mềm độc hại của hệ thống, bạn cũng cần đo tỷ lệ dương tính giả của hệ thống, đây là tốc độ hệ thống của bạn đưa ra cảnh báo phần mềm độc hại khi nhìn thấy phần mềm độc hại.

Bạn có thể tính toán tỷ lệ dương tính giả của hệ thống bằng cách chia số mẫu lành tính mà hệ thống gán cờ là phần mềm độc hại cho tổng số mẫu lành tính được kiểm tra.

Mối quan hệ giữa tỷ lệ tích cực đúng và sai

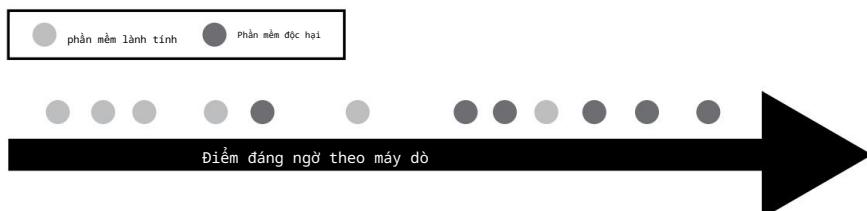
Khi thiết kế một hệ thống phát hiện, bạn muốn giữ tỷ lệ dương tính giả càng thấp càng tốt trong khi vẫn giữ tỷ lệ dương tính thật càng cao càng tốt.

Trừ khi bạn xây dựng một hệ thống phát hiện phần mềm độc hại thực sự hoàn hảo và luôn đúng (điều này thực sự là không thể do bản chất phát triển của phần mềm độc hại), sẽ luôn có sự căng thẳng giữa mong muốn có tỷ lệ dương tính thật cao và mong muốn có tỷ lệ dương tính giả thấp.

Để biết lý do tại sao lại như vậy, hãy tưởng tượng một hệ thống phát hiện, trước khi quyết định xem tệp nhị phân có phải là phần mềm độc hại hay không, sẽ bổ sung tất cả bằng chứng cho thấy tệp nhị phân là phần mềm độc hại để tạo điểm đáng ngờ cho tệp nhị phân. Hãy gọi hệ thống tạo điểm đáng ngờ giả định này là MalDetect.

Hình 7-2 cho thấy một ví dụ về các giá trị mà MalDetect có thể xuất ra cho 12 tệp nhị phân mẫu, trong đó các vòng tròn biểu thị các tệp nhị phân phần mềm riêng lẻ.

Nhị phân càng xa về bên phải, điểm đáng ngờ do MalDetect đưa ra càng cao.



Hình 7-2: Kết quả điểm số đáng ngờ do hệ thống MalDetect giả định cho các nhị phân phần mềm riêng lẻ

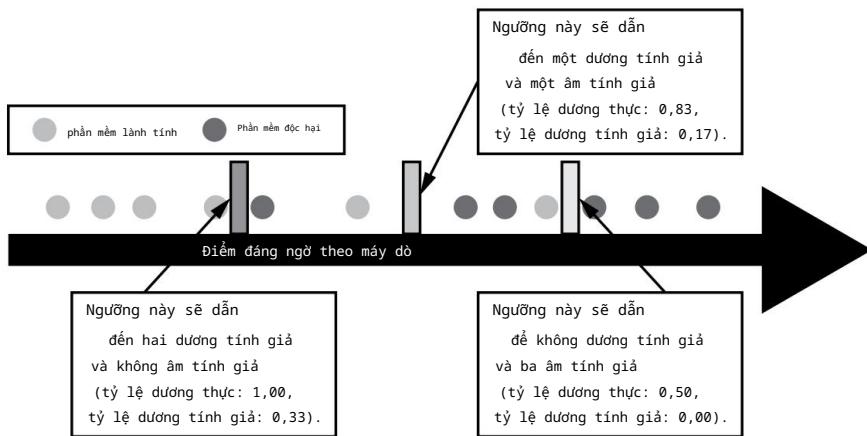
Điểm đáng ngờ là thông tin, nhưng để tính toán

Tỷ lệ dương tính thực sự và tỷ lệ dương tính giả của MalDetect trên các tệp của chúng tôi, chúng tôi

cần chuyển đổi điểm đáng ngờ của MalDetect thành câu trả lời “có” hoặc “không” liên quan đến việc một phần mềm nhị phân nhất định có độc hại hay không. Để làm điều này, chúng tôi sử dụng một quy tắc ngưỡng. Ví dụ: chúng tôi quyết định rằng nếu điểm đáng ngờ lớn hơn hoặc bằng một số nào đó, thì mã nhị phân được đề cập sẽ đưa ra cảnh báo về phần mềm độc hại. Nếu điểm thấp hơn ngưỡng, thì không.

Quy tắc ngưỡng như vậy là cách tiêu chuẩn để chuyển đổi điểm đáng ngờ thành lựa chọn phát hiện nhị phân, nhưng chúng ta nên đặt ngưỡng ở đâu?

Vấn đề là không có câu trả lời đúng. Hình 7-3 cho thấy trống conun: chúng ta đặt ngưỡng càng cao, chúng ta càng ít có khả năng nhận được kết quả dương tính giả, nhưng chúng ta càng có nhiều khả năng nhận được kết quả âm tính giả.



Hình 7-3: Minh họa mối quan hệ giữa tỷ lệ dương tính giả và tỷ lệ dương tính thật khi quyết định giá trị ngưỡng

Ví dụ: hãy xem xét ngưỡng ngoài cùng bên trái được hiển thị trong Hình 7-3, trong đó các nhị phân ở bên trái của ngưỡng được phân loại là lành tính và các nhị phân ở bên phải của nó được phân loại là phần mềm độc hại. Vì ngưỡng này thấp nên chúng tôi nhận được tỷ lệ dương tính thực sự cao (phân loại chính xác 100% mẫu phần mềm độc hại) nhưng lại có tỷ lệ dương tính giả khủng khiếp (phân loại sai 33% mẫu lành tính là độc hại).

Trực giác của chúng tôi có thể là tăng ngưỡng để chỉ những mẫu có điểm khả nghi cao hơn mới được coi là phần mềm độc hại. Một giải pháp như vậy được đưa ra bởi ngưỡng giữa trong Hình 7-3. Ở đây, tỷ lệ dương tính giả giảm xuống 0,17, nhưng thật không may, tỷ lệ dương tính thật cũng giảm xuống 0,83. Nếu chúng tôi tiếp tục di chuyển ngưỡng sang bên phải, như được hiển thị bằng ngưỡng ngoài cùng bên phải, thì chúng tôi sẽ loại bỏ mọi thông báo sai nhưng chỉ phát hiện được 50 phần trăm phần mềm độc hại.

Như bạn có thể thấy, không có cái gọi là ngưỡng hoàn hảo. Ngưỡng phát hiện mang lại tỷ lệ dương tính giả thấp (tốt) sẽ có xu hướng bỏ sót nhiều phần mềm độc hại hơn, mang lại tỷ lệ dương tính giả thấp (xấu). Ngược lại, sử dụng ngưỡng phát hiện có tỷ lệ dương tính thực cao (tốt) cũng sẽ làm tăng tỷ lệ dương tính giả (xấu).

Đường cong ROC

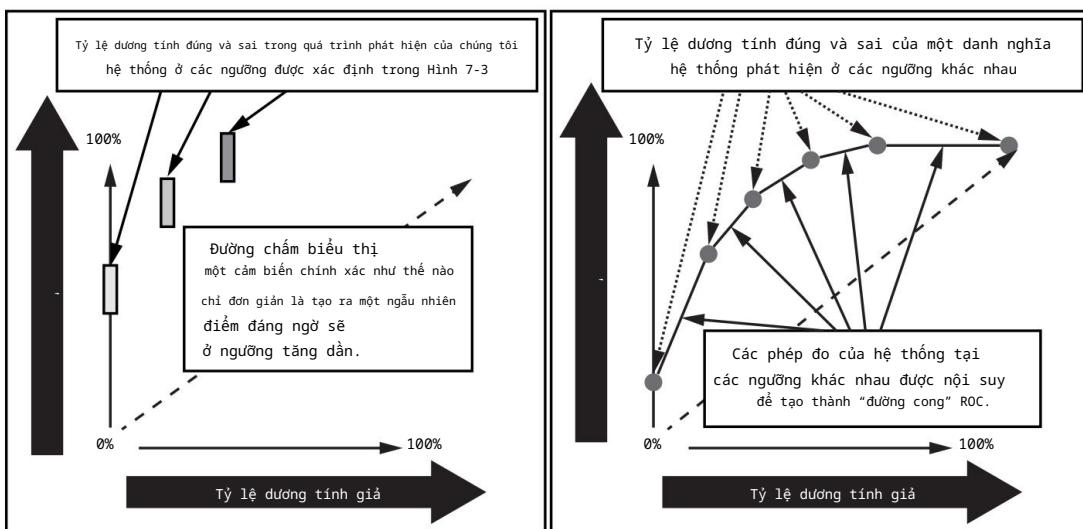
Sự đánh đổi giữa tỷ lệ dương tính thực và tỷ lệ dương tính giả của các hệ thống phát hiện là một vấn đề phổ biến đối với tất cả các máy dò, không chỉ máy dò phần mềm độc hại. Các kỹ sư và nhà thông kê đã suy nghĩ rất lâu về hiện tượng này và đưa ra Đặc tính Hoạt động của Máy thu (ROC) đường cong để mô tả và phân tích nó.

Lưu ý Bạn cảm thấy bối rối với cụm từ Đặc điểm hoạt động của máy thu, đừng lo lắng về nó—cụm từ này dễ gây nhầm lẫn và liên quan đến bối cảnh ban đầu các đường cong ROC được phát triển, đó là khả năng phát hiện các đối tượng vật lý dựa trên radar.

Các đường cong ROC mô tả một hệ thống phát hiện bằng cách vẽ biểu đồ tỷ lệ dương tính giả so với tỷ lệ dương tính thực sự liên quan của chúng ở các cài đặt nguồn khác nhau. Điều này giúp chúng tôi đánh giá sự cân bằng giữa tỷ lệ dương tính giả thấp hơn và tỷ lệ dương tính thật cao hơn, đồng thời xác định nguồn “tốt nhất” cho tình huống của chúng tôi.

Ví dụ, đối với hệ thống MalDetect giả định của chúng ta từ Hình 7-3, tỷ lệ dương tính thực của hệ thống là 0,5 khi tỷ lệ dương tính giả của nó là 0 (nguồn thấp) và tỷ lệ dương tính thực sự của hệ thống là 1,00 khi tỷ lệ dương tính giả là 0,33 (nguồn cao).

Hình 7-4 cho thấy cách thức hoạt động chi tiết hơn.



Hình 7-4: Minh họa ý nghĩa của các đường cong ROC và cách chúng được xây dựng

Để xây dựng đường cong ROC, chúng tôi bắt đầu với ba nguồn được sử dụng trong Hình 7-3 và vẽ biểu đồ tỷ lệ dương sai và thực của chúng, được thể hiện ở nửa bên trái của Hình 7-3. Biểu đồ bên phải của Hình 7-4 cho thấy điều tương tự, nhưng đối với tất cả các nguồn có thể. Như bạn có thể thấy, tỷ lệ dương tính giả càng cao thì tỷ lệ dương tính thật càng cao. Tương tự, tỷ lệ dương tính giả càng thấp thì tỷ lệ dương tính thật càng thấp.

"Đường cong" của đường cong ROC là một đường nằm trong không gian hai chiều

Biểu đồ ROC thể hiện cách chúng tôi nghĩ rằng hệ thống phát hiện của mình sẽ hoạt động với tỷ lệ dương thực của nó trên tất cả các giá trị dương tính giả có thể có và cách chúng tôi nghĩ rằng hệ thống phát hiện của mình sẽ hoạt động với tỷ lệ dương tính giả trên tất cả các giá trị dương thực có thể có. Có nhiều cách để tạo ra một đường cong như vậy, nhưng cách đó nằm ngoài phạm vi của cuốn sách này.

Tuy nhiên, một phương pháp đơn giản là thử nhiều giá trị ngưỡng, quan sát tỷ lệ dương sai và đúng tương ứng, vẽ chúng và nối các dấu chấm bằng một đường thẳng. Đường được kết nối này, được hiển thị trong biểu đồ bên phải của Hình 7-4, trở thành đường cong ROC của chúng ta.

Xem xét tỷ lệ cơ bản trong đánh giá của bạn

Như bạn đã thấy, các đường cong ROC có thể cho bạn biết hệ thống của bạn sẽ hoạt động như thế nào xét theo tốc độ mà hệ thống gọi là các nhị phân độc hại là độc hại (tỷ lệ dương thực sự) và tốc độ mà hệ thống gọi là các nhị phân lành tính là độc hại (tỷ lệ dương tính giả). Tuy nhiên, các đường cong ROC sẽ không cho bạn biết tỷ lệ cảnh báo của hệ thống là tích cực thực sự, mà chúng tôi gọi là độ chính xác của hệ thống. Độ chính xác của một hệ thống có liên quan đến tỷ lệ phần trăm các tệp nhị phân mà hệ thống gặp phải thực sự là phần mềm độc hại, mà chúng tôi gọi là tỷ lệ cơ sở. Dưới đây là bảng phân tích của từng thuật ngữ:

Độ chính xác Phần trăm cảnh báo phát hiện hệ thống là tích cực thực sự (có nghĩa là chúng là phát hiện phần mềm độc hại thực tế). Nói cách khác, độ chính xác là số lần dương tính thật / (dương tính thật + dương tính giả) của hệ thống phát hiện khi được kiểm tra đối với một số tập hợp nhị phân.

Tỷ lệ cơ bản Tỷ lệ phần dữ liệu được cung cấp cho hệ thống có chất lượng mà chúng tôi đang tìm kiếm. Trong trường hợp của chúng tôi, tỷ lệ cơ bản đề cập đến tỷ lệ phần trăm của các tệp nhị phân thực sự là phần mềm độc hại.

Chúng tôi thảo luận về cách hai số liệu này có liên quan trong phần tiếp theo.

Tỷ lệ cơ bản ảnh hưởng đến độ chính xác như thế nào

Mặc dù tỷ lệ xác thực đúng và sai của hệ thống phát hiện không thay đổi khi tỷ lệ cơ sở thay đổi, nhưng độ chính xác của hệ thống bị ảnh hưởng bởi những thay đổi trong tỷ lệ cơ bản của phần mềm độc hại-thường là đáng kể. Để biết tại sao điều này lại đúng, chúng ta hãy xem xét hai trường hợp sau đây.

Giả sử tỷ lệ dương tính giả của MalDetect là 1% và tỷ lệ dương tính thật là 100%. Bây giờ, giả sử chúng ta thả lỏng MalDetect trên mạng hoạt động mà chúng ta biết trước là không có phần mềm độc hại trên đó (có lẽ mạng vừa được tạo từ đầu trong phòng thí nghiệm). Bởi vì chúng tôi biết trước rằng không có phần mềm độc hại nào trên mạng, mọi cảnh báo mà MalDetect đưa ra theo định nghĩa sẽ là thông báo sai, bởi vì các tệp nhị phân duy nhất mà MalDetect gặp phải sẽ là phần mềm lành tính. Nói cách khác, độ chính xác sẽ là 0 phần trăm.

Ngược lại, nếu chúng tôi chạy MalDetect trên tập dữ liệu bao gồm toàn bộ phần mềm độc hại, thì sẽ không có cảnh báo nào của nó là dương tính giả: đơn giản là sẽ có

không bao giờ là cơ hội để MalDetect tạo ra kết quả dương tính giả vì không có phần mềm lành tính trong bộ dữ liệu phần mềm. Do đó, độ chính xác sẽ là 100 phần trăm.

Trong cả hai trường hợp cực đoan này, tỷ lệ cơ sở có tác động rất lớn đến độ chính xác của MalDetect hoặc khả năng cảnh báo của nó là dương tính giả.

Ước tính độ chính xác trong môi trường triển khai

Bây giờ bạn biết rằng tùy thuộc vào tỷ lệ phần mềm độc hại trong tập dữ liệu thử nghiệm (tỷ lệ cơ sở), hệ thống của bạn sẽ mang lại các giá trị chính xác khác nhau.

Điều gì sẽ xảy ra nếu bạn muốn ước tính độ chính xác mà hệ thống của bạn sẽ có dựa trên ước tính tốc độ cơ bản của môi trường mà bạn triển khai nó? Tất cả những gì bạn phải làm là sử dụng tỷ lệ cơ sở ước tính của môi trường triển khai để ước tính các biến trong công thức chính xác: giá trị dương thực / (dương tính thật + giá trị dương tính giả). Bạn sẽ cần ba số:

- Tỷ lệ tích cực thực sự (TPR) của hệ thống hoặc tỷ lệ phần trăm phần mềm độc hại mẫu hệ thống sẽ phát hiện chính xác
- Tỷ lệ dương tính giả (FPR) của hệ thống hoặc tỷ lệ mẫu lành tính mà hệ thống sẽ báo động sai
- Tỷ lệ cơ bản (BR) của các tệp nhị phân mà bạn sẽ sử dụng hệ thống (ví dụ: tỷ lệ phần trăm tệp nhị phân được tải xuống từ các trang web vi phạm bản quyền mà bạn mong đợi sẽ là phần mềm độc hại, nếu đây là thứ bạn sẽ sử dụng hệ thống của mình)

Tử số của phương trình chính xác-số lượng xác nhận thực sự-có thể được ước tính bằng tỷ lệ xác thực xác thực × tỷ lệ cơ bản, cung cấp cho bạn tỷ lệ phần trăm tuổi của phần mềm độc hại mà hệ thống của bạn sẽ phát hiện chính xác. Tương tự, mẫu số của phương trình-nghĩa là (dương tính thật + dương tính giả)-có thể được ước tính bằng tỷ lệ dương thực × tỷ lệ cơ bản + tỷ lệ dương tính giả × (1 - tỷ lệ cơ bản), cho bạn tỷ lệ phần trăm của tất cả các nhị phân hệ thống sẽ báo động bằng cách tính toán số lượng nhị phân phần mềm độc hại sẽ được phát hiện chính xác cộng với tỷ lệ phần mềm nhị phân phần mềm lành tính sẽ đưa ra kết quả dương tính giả.

Tóm lại, bạn tính toán độ chính xác dự kiến của hệ thống như sau:

$$\text{độ chính xác} = \frac{\text{lãi suất cơ bản} \times \text{lãi suất dương thực sự}}{\text{tỷ lệ dương thực} + \text{tỷ lệ} \times \text{tỷ lệ dương tính giả} + (1 - \text{tỷ lệ cơ bản})}$$

Hãy xem xét một ví dụ khác để xem tỷ lệ cơ sở có thể có tác động như thế nào đối với hiệu suất của một hệ thống phát hiện. Ví dụ: giả sử chúng ta có một hệ thống phát hiện có tỷ lệ dương tính thật là 80% và tỷ lệ dương tính giả là 10%, đồng thời 50% tệp nhị phân phần mềm mà chúng ta chạy hệ thống được cho là phần mềm độc hại. Điều này sẽ dẫn đến độ chính xác dự kiến là 89 phần trăm. Nhưng khi tỷ lệ cơ sở là 10 phần trăm, độ chính xác của chúng tôi giảm xuống 47 phần trăm.

Điều gì xảy ra nếu lãi suất cơ bản của chúng tôi rất thấp? Ví dụ, trong một hiện đại mạng doanh nghiệp, rất ít phần mềm nhị phân thực sự là phần mềm độc hại.

Sử dụng phương trình chính xác của chúng tôi, nếu chúng tôi giả sử tỷ lệ cơ bản là 1 phần trăm (1 trong 100 tệp nhị phân là phần mềm độc hại), chúng tôi sẽ nhận được độ chính xác khoảng 7,5 phần trăm, tức là

có nghĩa là 92,5 phần trăm cảnh báo trong hệ thống của chúng tôi sẽ là cảnh báo sai! Và nếu chúng tôi giả định tỷ lệ cơ bản là 0,1 phần trăm (1 trong 1000 tệp nhị phân có khả năng là phần mềm độc hại), thì chúng tôi nhận được độ chính xác là 1 phần trăm, nghĩa là 99 phần trăm cảnh báo của hệ thống của chúng tôi sẽ là dương tính giả! Cuối cùng, với tỷ lệ cơ bản là 0,01 phần trăm (1 trong 10.000 tệp nhị phân có khả năng là phần mềm độc hại-có thể là giả định thực tế nhất trên mạng doanh nghiệp), độ chính xác dự kiến của chúng tôi giảm xuống 0,1 phần trăm, nghĩa là phần lớn các cảnh báo trong hệ thống của chúng tôi sẽ là dương tính giả.

Một điểm rút ra từ phân tích này là các hệ thống phát hiện có tỷ lệ dương tính giả cao hầu như sẽ không bao giờ hữu ích trong môi trường doanh nghiệp, vì độ chính xác của chúng sẽ quá thấp. Do đó, mục tiêu chính trong việc xây dựng các hệ thống phát hiện phần mềm độc hại là giảm thiểu tỷ lệ dương tính giả sao cho độ chính xác của hệ thống là hợp lý.

Một điểm đáng chú ý khác có liên quan là khi bạn thực hiện phân tích đường cong ROC được giới thiệu trước đó trong chương này, bạn nên bỏ qua tỷ lệ dương tính giả trên một cách hiệu quả, chẳng hạn như 1%, nếu bạn đang phát triển hệ thống của mình để triển khai trong môi trường doanh nghiệp, bởi vì bất kỳ tỷ lệ sai nào cao hơn tỷ lệ dương có thể sẽ dẫn đến một hệ thống có độ chính xác thấp đến mức trở nên vô dụng.

Bản tóm tắt

Trong chương này, bạn đã học các khái niệm đánh giá phát hiện cơ bản, bao gồm tỷ lệ dương tính thực, tỷ lệ dương tính giả, đường cong ROC, tỷ lệ cơ sở và độ chính xác.

Bạn đã thấy việc tối đa hóa tỷ lệ dương tính thực và giảm thiểu tỷ lệ dương tính giả đều quan trọng như thế nào trong việc xây dựng hệ thống phát hiện phần mềm độc hại. Do tỷ lệ cơ sở ảnh hưởng đến độ chính xác như thế nào nên việc giảm tỷ lệ dương tính giả đặc biệt quan trọng nếu bạn muốn triển khai hệ thống phát hiện của mình trong một doanh nghiệp.

Nếu bạn không cảm thấy hoàn toàn thông thạo các khái niệm này, đừng lo lắng. Bạn sẽ thực hành nhiều hơn với chúng trong chương tiếp theo, nơi bạn sẽ xây dựng và đánh giá hệ thống phát hiện phần mềm độc hại ngay từ đầu. Trong quá trình này, bạn sẽ tìm hiểu các khái niệm đánh giá cụ thể về máy học bổ sung sẽ giúp bạn cải thiện các công cụ phát hiện dựa trên máy học của mình.

SỐ 8

Xây dựng máy L thu nhập máy dò s



Ngày nay, nhờ có phần mềm mã nguồn mở chất lượng cao xử lý khói lượng toán học nặng nề của máy thực hiện

hệ thống máy học, bất kỳ ai biết Python cơ bản và hiểu các khái niệm chính đều có thể sử dụng máy học.

Trong chương này, tôi sẽ chỉ cho bạn cách xây dựng các hệ thống phát hiện phần mềm độc hại bằng máy học bằng cách sử dụng scikit-learning, gói phổ biến nhất và tốt nhất, theo ý kiến của tôi—gói học máy mã nguồn mở hiện có. Chương này chứa rất nhiều mã mẫu. Các khói mã chính có thể truy cập được trong thư mục `malware_data_science/ch8/code` và dữ liệu mẫu tương ứng có thể truy cập được trong thư mục `malware_data_science/ch8/data` trong mã và dữ liệu (và trên máy ảo) đi kèm cuốn sách này.

Bằng cách làm theo văn bản, kiểm tra mã mẫu và thử các ví dụ được cung cấp, bạn sẽ cảm thấy thoải mái khi xây dựng và đánh giá các hệ thống máy học của riêng mình vào cuối chương này.

Bạn cũng học cách xây dựng trình phát hiện phần mềm độc hại nói chung và sử dụng các công cụ cần thiết để xây dựng trình phát hiện phần mềm độc hại cho các dòng phần mềm độc hại cụ thể. Các kỹ năng

bạn đạt được ở đây sẽ có một ứng dụng rộng rãi, cho phép bạn áp dụng máy học cho các vấn đề bảo mật khác, chẳng hạn như phát hiện email độc hại hoặc luồng mạng đáng ngờ.

Trước tiên, bạn tìm hiểu các thuật ngữ và khái niệm bạn cần biết trước khi sử dụng scikit-learning. Sau đó, bạn sử dụng scikit-learning để triển khai trình phát hiện cây quyết định cơ bản dựa trên các khái niệm về cây quyết định mà bạn đã học trong Chương 6. Tiếp theo, bạn tìm hiểu cách tích hợp mã trích xuất tính năng với scikit learn để xây dựng trình phát hiện phần mềm độc hại trong thế giới thực sử dụng thực tế.

khai thác tính năng -world và cách tiếp cận rừng ngẫu nhiên để phát hiện phần mềm độc hại. Cuối cùng, bạn tìm hiểu cách sử dụng scikit-learning để đánh giá các hệ thống máy học bằng công cụ phát hiện rừng ngẫu nhiên mẫu.

Thuật ngữ và khái niệm

Tiên hãy xem lại một số thuật ngữ. Thư viện mã nguồn mở scikit-learning (viết tắt là sklearn) đã trở nên phổ biến trong cộng đồng máy học vì nó vừa mạnh mẽ vừa dễ sử dụng. Nhiều nhà khoa học dữ liệu sử dụng thư viện trong cộng đồng bảo mật máy tính và trong các lĩnh vực khác, đồng thời nhiều người dựa vào nó làm hộp công cụ chính để thực hiện các tác vụ học máy. Mặc dù sklearn liên tục được cập nhật với các phương pháp học máy mới, nhưng nó cung cấp một giao diện lập trình nhất quán giúp việc sử dụng các phương pháp học máy này trở nên đơn giản.

Giống như nhiều khung học máy, sklearn yêu cầu dữ liệu đào tạo ở dạng vectơ . Các vectơ là các mảng số trong đó mỗi chỉ mục trong mảng tương ứng với một tính năng duy nhất của mảng phân phần ví dụ đào tạo.

Ví dụ: nếu hai tính năng của tệp nhị phân phần mềm mà trình phát hiện máy học của chúng tôi sử dụng được nén và chứa dữ liệu được mã hóa, thì vectơ đặc trưng của chúng tôi cho tệp nhị phân ví dụ đào tạo có thể là $[0,1]$. Ở đây, chỉ mục đầu tiên trong vectơ sẽ biểu thị liệu tệp nhị phân có được nén hay không, với số 0 biểu thị "không" và chỉ mục thứ hai sẽ biểu thị liệu tệp nhị phân có chứa dữ liệu được mã hóa hay không, với chỉ mục biểu thị "có".

Các vectơ có thể gây khó khăn khi làm việc vì bạn phải nhớ tính năng mà mỗi chỉ mục ánh xạ tới. May mắn thay, sklearn cung cấp mã trợ giúp dịch các biểu diễn dữ liệu khác sang dạng véc-tơ. Ví dụ: bạn có thể sử dụng lớp DictVectorizer của sklearn để chuyển đổi các biểu diễn từ điển của dữ liệu đào tạo của bạn (ví dụ: {"is_nén":1,"chứa_dữ_liệu_được_mã_hóa":0}) thành biểu diễn véc-tơ mà sklearn hoạt động trên đó, chẳng hạn như $[0,1]$. Sau này, bạn có thể sử dụng DictVectorizer để khôi phục ánh xạ giữa các chỉ số của vectơ và tên đối tượng ban đầu.

Để đào tạo trình phát hiện dựa trên sklearn , bạn cần chuyển hai đối tượng riêng biệt cho sklearn: vectơ đặc trưng (như đã mô tả trước đây) và vectơ nhãn. Một vectơ nhãn chứa một số cho mỗi ví dụ đào tạo, trong trường hợp của chúng tôi, tương ứng với việc ví dụ đó có phải là phần mềm độc hại hay phần mềm lành tính hay không. Ví dụ: nếu chúng tôi chuyển ba ví dụ đào tạo tới sklearn, sau đó chuyển vectơ nhãn $[0,1,0]$, thì chúng tôi đang nói với sklearn rằng mẫu đầu tiên là phần mềm lành tính, mẫu thứ hai là phần mềm độc hại và mẫu thứ ba là phần mềm lành tính. Theo quy ước, các kỹ sư máy học sử dụng chữ X viết hoa

bíen để biểu thị dữ liệu đào tạo và bíen y chữ thường để thay thế các nhãn đã gửi. Sự khác biệt trong trường hợp phản ánh quy ước trong toán học viết hoa các biến đại diện cho ma trận (mà chúng ta có thể coi là mảng vectơ) và biến chữ thường đại diện cho các vectơ riêng lẻ. Bạn sẽ thấy quy ước này được sử dụng trong mã mẫu học máy trực tuyến và tôi sử dụng quy ước này cho phần còn lại của cuốn sách này để giúp bạn cảm thấy thoải mái với nó.

Khung sklearn sử dụng các thuật ngữ khác mà bạn có thể thấy mới cung. Thay vì gọi các máy dò dựa trên máy học là "máy dò", sklearn gọi chúng là "máy phân loại". Trong bối cảnh này, thuật ngữ bộ phân loại chỉ đơn giản có nghĩa là một hệ thống máy học phân loại mọi thứ thành hai hoặc nhiều loại. Do đó, máy dò (thuật ngữ tôi sử dụng xuyên suốt cuốn sách này) là một kiểu phân loại đặc biệt, xếp mọi thứ thành hai loại, như phần mềm độc hại và phần mềm lành tính. Ngoài ra, thay vì sử dụng thuật ngữ đào tạo, tài liệu và API của sklearn thường sử dụng thuật ngữ phù hợp. Ví dụ: bạn sẽ thấy một câu như "phù hợp với trình phân loại máy học bằng ví dụ đào tạo", tương đương với câu nói "đào tạo trình phát hiện máy học bằng cách sử dụng ví dụ đào tạo".

Cuối cùng, thay vì sử dụng thuật ngữ phát hiện trong ngữ cảnh của bộ phân loại, sklearn sử dụng thuật ngữ dự đoán. Thuật ngữ này được sử dụng trong khuôn khổ của sklearn và trong cộng đồng máy học nói chung, bất cứ khi nào hệ thống máy học được sử dụng để thực hiện một tác vụ, cho dù dự đoán giá trị của cổ phiếu một tuần kể từ bây giờ hay phát hiện xem một tệp nhị phân không xác định có phải là phần mềm độc hại hay không.

Xây dựng máy dò dựa trên cây quyết định đồ chơi

Bây giờ bạn đã quen thuộc với thuật ngữ kỹ thuật của sklearn , hãy tạo một cây quyết định đơn giản theo những gì chúng ta đã thảo luận trong Chương 6, sử dụng khung sklearn . Hãy nhớ lại rằng cây quyết định chơi một loại trò chơi "20 câu hỏi", trong đó chúng đặt một loạt câu hỏi về các vectơ đầu vào để đưa ra quyết định liên quan đến việc liệu các vectơ này là độc hại hay lành tính. Chúng ta từng bước xây dựng một bộ phân loại cây quyết định, sau đó khám phá một ví dụ về một chương trình hoàn chỉnh. Liệt kê 8-1 cho thấy cách nhập các mô-đun cần thiết từ sklearn.

từ cây nhập khẩu sklearn
từ sklearn.feature_extract nhập DictVectorizer

Liệt kê 8-1: Nhập mô-đun sklearn

Mô-đun đầu tiên chúng tôi nhập, cây, là mô-đun cây quyết định của sklearn . Mô-đun thứ hai, feature_extract, là mô-đun trợ giúp của sklearn mà từ đó chúng tôi nhập lớp DictVectorizer . Lớp DictVectorizer chuyển đổi thuận tiện dữ liệu đào tạo được cung cấp ở dạng từ điển, có thể đọc được sang biểu diễn véc-tơ mà sklearn yêu cầu để thực sự huấn luyện các máy dò học máy.

Sau khi chúng tôi nhập các mô-đun chúng tôi cần từ sklearn, chúng tôi khởi tạo mô-đun các lớp sklearn cần thiết , như trong Liệt kê 8-2.

```
trình phân loại = utree.DecisionTreeClassifier()
vectorizer = DictVectorizer(sparse= False)
```

Liệt kê 8-2: Khởi tạo bộ phân loại cây quyết định và bộ tạo vectơ

Lớp đầu tiên mà chúng tôi khởi tạo, `DecisionTreeClassifier`, đại diện cho trình phát hiện của chúng tôi. Mặc dù `sklearn` cung cấp một số tham số kiểm soát chính xác cách cây quyết định của chúng tôi sẽ hoạt động, ở đây chúng tôi không chọn bất kỳ tham số nào để chúng tôi sử dụng cài đặt cây quyết định mặc định của `sklearn`.

Lớp tiếp theo chúng ta khởi tạo là `DictVectorizer`. Chúng tôi đặt thưa thành Sai trong hàm tạo, nói với `sklearn` rằng chúng tôi không muốn nó sử dụng các vectơ thưa thớt, giúp tiết kiệm bộ nhớ nhưng thao tác phức tạp. Vì mô-đun cây quyết định của `sklearn` không thể sử dụng các vectơ thưa thớt nên chúng tôi tắt tính năng này.

Bây giờ chúng ta đã khởi tạo các lớp của mình, chúng ta có thể khởi tạo một số mẫu dữ liệu huấn luyện, như trong [Liệt kê 8-3](#).

```
# khai báo dữ liệu huấn luyện đồ chơi
training_examples = [
    {'packed':1,'contains_encrypted':0},
    {'packed':0,'contains_encrypted':0},
    {'packed':1,'contains_encrypted':1},
    {'packed':1,'contains_encrypted':0},
    {'packed':0,'contains_encrypted':1},
    {'packed':1,'contains_encrypted':0},
    {'packed':0,'contains_encrypted':0},
    {'packed':0,'contains_encrypted':0},
]
ground_truth = [1,1,1,1,0,0,0,0]
```

Liệt kê 8-3: Khai báo các vectơ huấn luyện và nhãn

Trong ví dụ này, chúng tôi khởi tạo hai cấu trúc – vectơ đặc trưng và vectơ nhãn – cùng nhau tạo thành dữ liệu đào tạo của chúng tôi. Các vectơ đặc trưng, được gán cho biến `training_examples`, được đưa ra ở dạng từ điển. Như bạn có thể thấy, chúng tôi đang sử dụng hai tính năng đơn giản. Tệp đầu tiên được đóng gói, biểu thị liệu một tệp nhất định có được đóng gói hay không và tệp thứ hai có chứa `_encrypted`, đại diện cho biết tệp có chứa dữ liệu được mã hóa hay không. Vectơ nhãn, được gán cho biến `ground_truth`, biểu thị xem mỗi ví dụ đào tạo là độc hại hay lành tính. Trong cuốn sách này và nói chung trong số các nhà khoa học dữ liệu bảo mật, 0 luôn tượng trưng cho lành tính và 1 luôn tượng trưng cho độc hại. Trong trường hợp này, vectơ nhãn tuyên bố rằng bốn vectơ đặc trưng đầu tiên là độc hại và bốn vectơ thứ hai là lành tính.

Đào tạo Trình phân loại cây quyết định của bạn

Bây giờ chúng ta đã khai báo các vectơ huấn luyện và vectơ nhãn, hãy huấn luyện mô hình cây quyết định của chúng ta bằng cách gọi phương thức phù hợp của thể hiện của lớp cây quyết định, như được hiển thị trong [Liệt kê 8-4](#).

```
# khởi tạo vectorizer với dữ liệu huấn luyện
vectorizer.fit(training_examples)
```

```
# chuyển đổi các ví dụ đào tạo sang dạng vector
X = vectorizer.transform(training_examples)
y = ground_truth # gọi sự thật nền là 'y', theo quy ước
```

Liệt kê 8-4: Khởi tạo lớp vectorizer với dữ liệu huấn luyện

Đoạn mã trong Liệt kê 8-4 trước hết khởi tạo lớp vectorizer mà chúng ta được khởi tạo trong Liệt kê 8-2 bằng cách gọi phương thức phù hợp . Ở đây, sự phù hợp phương thức yêu cầu sklearn tạo ánh xạ giữa tính năng được đóng gói và tính năng chứa_encrypted và các chỉ số mảng vectơ. Sau đó, chúng tôi chuyển đổi các vectơ đặc trưng dựa trên từ điển của mình thành dạng vectơ số bằng cách gọi phương thức biến đổi của lớp vectorizer . Hãy nhớ lại rằng chúng ta gán các vectơ đặc trưng của mình cho một biến có tên là X và vectơ nhãn của chúng ta cho một biến có tên là y, đây là quy ước đặt tên trong công đồng máy học.

Bây giờ chúng ta đã thiết lập tất cả dữ liệu đào tạo của mình, chúng ta có thể đào tạo trình phát hiện cây quyết định của mình bằng cách gọi phương thức phù hợp trên các thẻ hiện của bộ phân loại cây quyết định, như sau:

```
# huấn luyện bộ phân loại (còn gọi là 'phù hợp' với bộ phân loại)
phân loại.fit(X,y)
```

Như bạn có thể thấy, đào tạo trình phát hiện sklearn đơn giản như vậy. Tuy nhiên, đáng sau hậu trường, sklearn đang trải qua quá trình thuật toán xác định một cây quyết định tốt để phát hiện chính xác liệu phần mềm mới là độc hại hay lành tính, dọc theo dòng thuật toán mà chúng ta đã thảo luận trong chương trước.

Bây giờ chúng ta đã huấn luyện bộ phát hiện, hãy sử dụng mã trong Liệt kê 8-5 để phát hiện xem tệp nhị phân là độc hại hay lành tính.

```
test_example = u({'packed':1,'contains_encrypted':0}
test_vector = vvectorizer.transform(test_example)
print classifier.predict(test_vector) # print [1]
```

Liệt kê 8-5: Xác định xem một tệp nhị phân có độc hại hay không

Ở đây, chúng tôi khởi tạo một vectơ đặc trưng dựa trên từ điển cho một nhị phân phần mềm giả định , dịch nó sang dạng vectơ số bằng cách sử dụng trình tạo vectơ , mà chúng tôi đã khai báo trước đó trong mã của mình, sau đó chạy trình phát hiện cây quyết định mà chúng tôi đã tạo để xác định xem không phải nhị phân là độc hại. Bạn sẽ thấy khi chúng tôi chạy mã mà trình phân loại "nghĩ" rằng tệp nhị phân mới là độc hại (vì nó cho kết quả là "1" làm đầu ra) và bạn sẽ thấy tại sao lại xảy ra trường hợp này khi chúng tôi hình dung cây quyết định của mình .

Trực quan hóa cây quyết định

Chúng ta có thể hình dung cây quyết định mà sklearn đã tự động tạo dựa trên dữ liệu huấn luyện của mình, như trong Liệt kê 8-6.

```
# hình dung cây quyết định
với open( "classifier.dot","w") là output_file:
```

```

tree.export_graphviz(
    phân loại,
    Feature_names=vectorizer.get_feature_names(),
    out_file=out_file
)

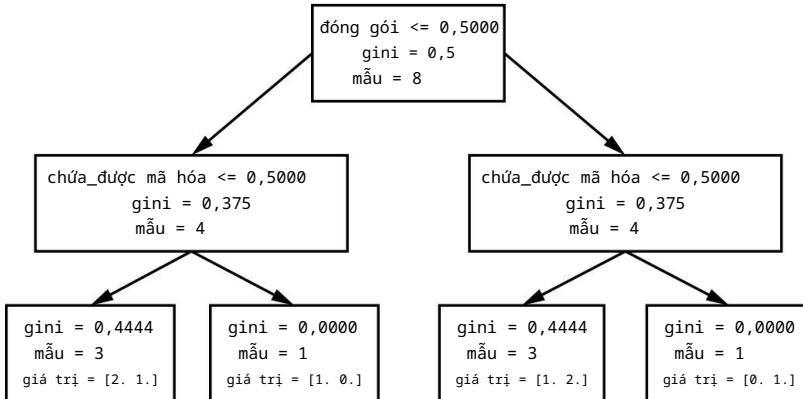
nhập hệ điều hành
os.system("dot classifier.dot -Tpng -o classifier.png")

```

Liệt kê 8-6: Tạo tệp hình ảnh của cây quyết định bằng GraphViz

Ở đây, chúng tôi mở một tệp có tên là classifier.dot mà chúng tôi viết một biểu diễn mạng của cây quyết định bằng cách sử dụng hàm `export_graphviz()` mà mô-đun cây của sklearn cung cấp. Sau đó, chúng tôi gọi `tree.export_graphviz` để ghi tệp GraphViz .dot vào `classifier.dot`, tệp này ghi biểu diễn mạng của cây quyết định vào đĩa. Cuối cùng, chúng ta sử dụng chương trình dòng lệnh GraphViz dot để tạo một tệp hình ảnh trực quan hóa cây quyết định, ở dạng tương ứng với những gì bạn đã học về cây quyết định trong Chương 6.

Khi bạn chạy chương trình này, bạn sẽ nhận được một tệp hình ảnh đầu ra có tên là `classifier.png` giống như Hình 8-1.



Hình 8-1: Trực quan hóa cây quyết định

Mặc dù hình ảnh cây quyết định này đã quen thuộc từ Chương 6, nhưng nó chứa một số từ vựng mới. Dòng đầu tiên trong mỗi hộp chứa tên của tính năng mà nút đặt câu hỏi về nó (theo cách nói của máy học, chúng tôi nói rằng nút “tách ra” tính năng này).

Ví dụ: nút đầu tiên phân tách theo tính năng “được đóng gói”: nếu tệp nhị phân không được đóng gói, chúng tôi di chuyển dọc theo mũi tên bên trái; mặt khác, chúng tôi di chuyển dọc theo bên phải mũi tên.

Dòng văn bản thứ hai trong mỗi hộp đề cập đến chỉ số gini của nút đó, chỉ số này đo lường mức độ bất bình đẳng giữa các ví dụ đào tạo phần mềm độc hại và phần mềm độc hại phù hợp với nút đó. Chỉ số gini càng cao, các mẫu phù hợp với nút đó càng nghiêng về phần mềm lành tính hoặc phần mềm độc hại. Điều này có nghĩa là chỉ số gini cao trong mỗi nút là tốt, bởi vì càng nhiều ví dụ đào tạo càng nghiêng về phần mềm độc hại.

hoặc phần mềm lành tính, chúng tôi càng chắc chắn hơn về việc liệu các ví dụ thử nghiệm mới là phần mềm độc hại hay phần mềm lành tính. Dòng thứ ba trong mỗi hộp chỉ cung cấp số lượng ví dụ đào tạo phù hợp với nút đó.

Bạn sẽ nhận thấy rằng trong các nút lá của cây, văn bản trong hộp sẽ khác. Các nút này không “đặt câu hỏi;” thay vào đó, chúng đưa ra câu trả lời cho câu hỏi “nhị phân này độc hại hay lành tính?” Ví dụ, ở nút lá ngoài cùng bên trái, chúng ta có “value = [2, 1.],” có nghĩa là hai ví dụ đào tạo lành tính khớp với nút này (không được đóng gói và không được mã hóa) và một ví dụ đào tạo phần mềm độc hại khớp với nút. Nghĩa là, nếu chúng tôi tiếp cận nút này, chúng tôi sẽ chỉ định xác suất 33 phần trăm cho phần mềm độc hại nhị phân (1 mẫu phần mềm độc hại / tổng số 3 mẫu = 33 phần trăm). Giá trị gini trong các hộp này cho biết lượng thông tin thu được về việc liệu tập nhị phân là phần mềm độc hại hay phần mềm lành tính khi chúng tôi phân chia câu hỏi trực tiếp dẫn đến các nút này. Như bạn có thể thấy, có thể hữu ích khi kiểm tra trực quan hóa cây quyết định do sklearn tạo ra để hiểu cách cây quyết định của chúng tôi thực hiện các phát hiện.

Hoàn thành mã mẫu

Liệt kê 8-7 hiển thị mã hoàn chỉnh cho quy trình cây quyết định mà tôi đã mô tả cho đến nay. Mã này sẽ dễ đọc đối với bạn vì chúng tôi đã xử lý mã này từng phần một.

```
#!/usr/bin/tienganh

# nhập mô-đun sklearn
từ cây nhập khẩu sklearn
từ sklearn.feature_extract nhập DictVectorizer

# khởi tạo trình phân loại và vector hóa cây quyết định
trình phân loại = tree.DecisionTreeClassifier()
vectorizer = DictVectorizer(sparse=False)

# khai báo dữ liệu huấn luyện đồ chơi
training_examples = [
{'packed':1,'contains_encrypted':0},
{'packed':0,'contains_encrypted':0},
{'packed':1,'contains_encrypted':1},
{'packed':1,'contains_encrypted':0},
{'packed':0,'contains_encrypted':1},
{'packed':1,'contains_encrypted':0},
{'packed':0,'contains_encrypted':0},
{'packed':0,'contains_encrypted':0},
]
ground_truth = [1,1,1,1,0,0,0,0]

# khởi tạo vectorizer với dữ liệu huấn luyện
vectorizer.fit(training_examples)

# chuyển đổi các ví dụ đào tạo sang dạng vector
X = vectorizer.transform(training_examples)
y = ground_truth # gọi sự thật nền là 'y', theo quy ước
```

```

# huấn luyện bộ phân loại (còn gọi là 'phù hợp' với bộ phân loại)
phân loại.fit(X,y)

test_example = {'packed':1,'contains_encrypted':0}
test_vector = vectorizer.transform(test_example)
in `classifier.predict(test_vector)` # in [1]

# trực quan hóa cây quyết định
với open("classifier.dot","w") là output_file:
    tree.export_graphviz(
        phân loại,
        Feature_names=vectorizer.get_feature_names(),
        out_file=output_file
    )

nhập hệ điều hành
os.system("dot classifier.dot -Tpng -o classifier.png")

```

Liệt kê 8-7: Mã mẫu quy trình công việc cây quyết định hoàn chỉnh

Trình phát hiện phần mềm độc hại máy học mẫu mà chúng tôi vừa khám phá trình bày cách bắt đầu với chức năng của sklearn , nhưng nó thiếu một số tính năng thiết yếu cần thiết cho trình phát hiện phần mềm độc hại trong thế giới thực. Bây giờ chúng ta hãy khám phá những yêu cầu của một trình phát hiện phần mềm độc hại trong thế giới thực.

Xây dựng công cụ phát hiện học máy trong thế giới thực với sklearn

Để xây dựng một trình phát hiện trong thế giới thực, bạn cần sử dụng các tính năng sức mạnh công nghiệp của các tệp nhị phân phần mềm cũng như viết mã để trích xuất các tính năng này từ các tệp nhị phân phần mềm. Các tính năng sức mạnh công nghiệp là những tính năng phần ánh nội dung của các tệp nhị phân ở tất cả mức độ phức tạp của chúng, điều đó có nghĩa là chúng tôi cần sử dụng hàng trăm hoặc hàng nghìn tính năng. Bằng cách "trích xuất" các tính năng, ý tôi là bạn phải viết mã xác định sự hiện diện của các tính năng này trong các tệp nhị phân. Bạn cũng cần sử dụng hàng nghìn ví dụ đào tạo và đào tạo một mô hình máy học trên quy mô lớn. Cuối cùng, bạn cần sử dụng các phương pháp phát hiện nâng cao hơn của sklearn vì các phương pháp cây quyết định đơn giản mà chúng tôi vừa khám phá không cung cấp đủ độ chính xác cho việc phát hiện.

Trích xuất tính năng trong thế giới thực

Các tính năng mẫu mà tôi đã sử dụng trước đây, chẳng hạn như được đóng gói và chứa dữ liệu được mã hóa, là các ví dụ về đồ chơi đơn giản và chỉ riêng hai tính năng này sẽ không bao giờ khiến trình phát hiện phần mềm độc hại hoạt động. Như tôi đã đề cập trước đây, các hệ thống phát hiện phần mềm độc hại trong thế giới thực sử dụng hàng trăm, hàng nghìn hoặc thậm chí hàng triệu tính năng. Ví dụ: trình phát hiện dựa trên máy học có thể sử dụng hàng triệu chuỗi ký tự xuất hiện trong tệp nhị phân phần mềm làm tính năng. Hoặc nó có thể sử dụng các giá trị của tiêu đề Portable Executable (PE) nhị phân phần mềm, các chức năng được nhập bởi một nhị phân nhất định hoặc một số kết hợp của tất cả những thứ này. Mặc dù chúng ta sẽ chỉ làm việc với các tính năng chuỗi trong chương này, nhưng hãy dành chút thời gian để khám phá

các loại tính năng phổ biến được sử dụng trong quá trình phát hiện phần mềm độc hại dựa trên máy học, bắt đầu bằng cách tính năng chuỗi.

Tính năng chuỗi

Các tính năng chuỗi của tệp nhị phân phần mềm là tất cả các chuỗi liền kề của các ký tự có thể in được trong tệp có ít nhất một số độ dài tối thiểu (trong cuốn sách này, mức tối thiểu này được đặt thành năm ký tự). Ví dụ: giả sử tệp nhị phân chứa các chuỗi ký tự có thể in được sau:

["A", "The", "PE thực thi", "Tài trọng độc hại"]

Trong trường hợp này, các chuỗi chúng ta có thể sử dụng làm tính năng sẽ là "PE thực thi" và "Tài trọng độc hại" vì hai chuỗi này có hơn năm ký tự trong đó.

Để chuyển đổi các tính năng chuỗi thành định dạng mà sklearn có thể hiểu được, chúng ta cần đưa chúng vào từ điển Python. Chúng tôi thực hiện việc này bằng cách sử dụng các chuỗi thực tế làm khóa từ điển, sau đó đặt giá trị của chúng thành 1 để cho biết rằng tệp nhị phân được đề cập có chứa chuỗi đó. Ví dụ: tệp nhị phân mẫu trước đó sẽ nhận được một vectơ đặc trưng của ("PE có thể thực thi": 1, "Tài trọng độc hại": 1). Tất nhiên, hầu hết các tệp nhị phân phần mềm đều có hàng trâm chuỗi có thể in được trong đó, không chỉ hai chuỗi và các chuỗi này có thể chứa nhiều thông tin về chức năng của chương trình.

Trên thực tế, các tính năng chuỗi hoạt động tốt với tính năng phát hiện dựa trên máy học bởi vì họ nắm bắt rất nhiều thông tin về phần mềm nhị phân. Nếu tệp nhị phân là một mẫu phần mềm độc hại được đóng gói, thì nó có thể có ít chuỗi thông tin, bản thân chuỗi này có thể là dấu hiệu cho thấy tệp độc hại. Mặt khác, nếu các phần trong phần tài nguyên của tệp không được đóng gói hoặc làm rách, thì các chuỗi đó sẽ tiết lộ nhiều điều về hành vi của tệp. Ví dụ: nếu chương trình nhị phân được đề cập thực hiện các yêu cầu HTTP, thông thường bạn sẽ thấy các chuỗi như "GET %s" trong tập hợp chuỗi của tệp đó.

Tuy nhiên, các tính năng chuỗi có một số hạn chế. Ví dụ, chúng không nắm bắt được bất cứ điều gì về logic thực sự của một chương trình nhị phân, bởi vì chúng không bao gồm mã chương trình thực tế. Vì vậy, mặc dù các chuỗi có thể là các tính năng hữu ích ngay cả trên các tệp nhị phân được đóng gói, nhưng chúng không tiết lộ các tệp nhị phân được đóng gói thực sự làm gì. Do đó, các trình phát hiện dựa trên các tính năng chuỗi không lý tưởng để phát hiện phần mềm độc hại được đóng gói.

Tính năng tiêu đề Portable Executable (PE)

Các tính năng tiêu đề PE được trích xuất từ siêu dữ liệu tiêu đề PE nằm trong mọi tệp .exe và .dll của Windows. Để biết thêm thông tin về định dạng của các tiêu đề này, hãy tham khảo Chương 1. Để trích xuất các tính năng PE từ các tệp nhị phân portable executable, bạn có thể sử dụng mã được cung cấp trong chương đó, sau đó mã hóa các tính năng tệp ở dạng từ điển Python, trong đó tên trường tiêu đề là khóa từ điển và giá trị trường là giá trị tương ứng với mỗi khóa.

Các tính năng tiêu đề PE bổ sung tốt cho các tính năng chuỗi. Ví dụ: trong khi các tính năng chuỗi thường thực hiện tốt công việc nắm bắt các lệnh gọi hàm và truyền mạng được thực hiện bởi một chương trình, như ví dụ "NHẬN %s" ,

Các tính năng tiêu đề PE nắm bắt thông tin như dấu thời gian biên dịch của chương trình nhị phân, bộ cục của các phần PE của nó và phần nào trong số đó được đánh dấu là có thể thực thi được và dung lượng của chúng trên đĩa. Chúng cũng nắm bắt dung lượng bộ nhớ mà chương trình phân bổ khi khởi động và nhiều đặc điểm thời gian chạy khác của chương trình nhị phân mà các tính năng chuỗi không nắm bắt được.

Ngay cả khi bạn đang xử lý các tệp nhị phân được đóng gói, các tính năng tiêu đề PE vẫn có thể thực hiện tốt công việc phân biệt phần mềm độc hại được đóng gói với phần mềm lành tính được đóng gói. Điều này là do mặc dù chúng tôi không thấy mã của các tệp nhị phân được đóng gói do bị xáo trộn, nhưng chúng tôi vẫn có thể thấy mã chiếm bao nhiêu dung lượng trên đĩa và cách tệp nhị phân được bố trí trên đĩa hoặc được nén trên một loạt các phần tệp. Đây là những thông tin chi tiết có thể giúp hệ thống máy học phân biệt phần mềm độc hại với phần mềm lành tính. Mặt khác, các tính năng tiêu đề PE không nắm bắt được các hướng dẫn thực tế mà chương trình thực thi khi chạy hoặc các chức năng mà chương trình gọi.

Các tính năng của Bảng địa chỉ nhập (IAT)

Bảng địa chỉ nhập (IAT) mà bạn đã học trong Chương 1, cũng là một nguồn quan trọng của các tính năng máy học. IAT chứa danh sách các chức năng và thư viện mà phần mềm nhị phân nhập từ các tệp DLL bên ngoài. Như vậy, IAT chứa thông tin quan trọng về hành vi của chương trình mà bạn có thể sử dụng để bổ sung cho các tính năng tiêu đề PE được mô tả trong phần trước.

Để sử dụng IAT làm nguồn tính năng máy học, bạn cần hiểu thị trường tệp dưới dạng từ điển các tính năng, trong đó tên của thư viện và chức năng đã nhập là khóa và khóa ánh xạ tới 1, cho biết tệp đó trong câu hỏi có chứa nội dung nhập cụ thể đó (ví dụ: khóa "KERNEL32.DLL:LoadLibraryA", trong đó KERNEL32.DLL là DLL và LoadLibraryA là lệnh gọi hàm). Từ điển tính năng là kết quả của việc tính toán các tính năng IAT theo cách này cho một mẫu sẽ giống như { KERNEL32.DLL:LoadLibraryA: 1, ... }, trong đó chúng tôi sẽ gán 1 cho bất kỳ khóa nào được quan sát trong tệp nhị phân.

Theo kinh nghiệm của tôi khi xây dựng trình phát hiện phần mềm độc hại, tôi nhận thấy rằng các tính năng IAT hiếm khi tự hoạt động tốt-mặc dù các tính năng này nắm bắt thông tin cấp cao hữu ích về hành vi của chương trình, nhưng phần mềm độc hại thường làm xáo trộn IAT để khiến chính nó trông giống như phần mềm lành tính. Ngay cả khi phần mềm độc hại không chứa obfuscation, nó thường nhập các lệnh gọi DLL giống như phần mềm lành tính cũng nhập, khiến khó phân biệt giữa phần mềm độc hại và phần mềm lành tính chỉ dựa trên thông tin IAT. Cuối cùng, khi phần mềm độc hại được đóng gói (được nén hoặc mã hóa, sao cho mã phần mềm độc hại thực sự chỉ hiển thị sau khi phần mềm độc hại thực thi và tự giải nén hoặc giải mã), IAT chỉ chứa các mục nhập mà trình đóng gói sử dụng, không phải các mục nhập mà phần mềm độc hại sử dụng.

Điều đó nói rằng, khi bạn sử dụng các tính năng IAT kết hợp với các tính năng khác như tính năng tiêu đề PE và tính năng chuỗi, chúng có thể tăng độ chính xác của hệ thống.

N-gam

Cho đến nay, bạn đã tìm hiểu về các tính năng máy học không liên quan đến bất kỳ khái niệm nào về thứ tự. Ví dụ, chúng tôi đã thảo luận về các tính năng chuỗi để

kiểm tra xem một tệp nhị phân có một chuỗi cụ thể hay không, nhưng không phải liệu một chuỗi cụ thể có xuất hiện trước hay sau một chuỗi khác trong bộ cục của tệp nhị phân trên đĩa hay không.

Nhưng đôi khi thử tự quan trọng. Ví dụ: chúng tôi có thể thấy rằng một dòng phần mềm độc hại quan trọng chỉ nhập các chức năng được sử dụng phổ biến, nhưng nhập chúng theo một thứ tự rất cụ thể, sao cho khi quan sát các chức năng theo thứ tự đó, chúng tôi biết mình đang nhìn thấy dòng phần mềm độc hại đó chứ không phải phần mềm lành tính. Để nắm bắt loại thông tin đặt hàng này, bạn có thể sử dụng khái niệm học máy có tên là N-gram.

N-gram nghe có vẻ kỳ lạ hơn bản chất của chúng: chúng chỉ liên quan đến việc sắp xếp các đặc điểm của bạn theo trình tự mà chúng xuất hiện và sau đó trượt một cửa sổ có độ dài n trên trình tự, coi trình tự các đặc điểm bên trong cửa sổ ở mỗi bước là một tính năng đơn lẻ, tổng hợp. Ví dụ: nếu chúng tôi có chuỗi ["how", "now", "brown", "cow"] và chúng tôi muốn trích xuất các tính năng N-gram có độ dài 2 ($n = 2$) từ chuỗi này, chúng tôi sẽ có [("làm thế nào", "bây giờ"), ("bây giờ", "nâú"), ("nâú", "bò")] làm các tính năng của chúng tôi.

Trong bối cảnh phát hiện phần mềm độc hại, một số loại dữ liệu thường được biểu diễn dưới dạng các tính năng N-gram. Ví dụ: khi bạn phân tách một tệp nhị phân thành các hướng dẫn cấu thành của nó, như ["inc", "dec", "sub", "mov"], sau đó sử dụng phương pháp N-gram để nắm bắt các chuỗi hướng dẫn này là hợp lý bởi vì đại diện cho một chuỗi các hướng dẫn có thể hữu ích trong việc phát hiện việc triển khai phần mềm độc hại cụ thể. Ngoài ra, khi bạn đang thực thi các tệp nhị phân để kiểm tra hành vi động của chúng, bạn có thể sử dụng phương pháp N-gram để biểu thị các chuỗi lệnh gọi API hoặc hành vi cấp cao của các tệp nhị phân.

Tôi khuyên bạn nên thử nghiệm các tính năng N-gram trong hệ thống phát hiện phần mềm độc hại dựa trên máy học của mình bất cứ khi nào bạn làm việc với dữ liệu xảy ra theo một số loại trình tự. Thường mất một số lần thử và sai để xác định bạn nên đặt n thành số nào, số này xác định độ dài của N-gram của bạn. Quá trình thử và sai này liên quan đến việc thay đổi n

value để xem giá trị nào mang lại độ chính xác tốt nhất trên dữ liệu thử nghiệm của bạn. Khi bạn tìm đúng số, N-gram có thể là các tính năng mạnh mẽ để nắm bắt các hành vi tuần tự thực tế của các nhị phân chương trình, do đó nâng cao độ chính xác của hệ thống.

Tại sao bạn không thể sử dụng tất cả các tính năng có thể

Bây giờ bạn đã biết điểm mạnh và điểm yếu của các loại tính năng khác nhau, bạn có thể tự hỏi tại sao bạn không thể sử dụng tất cả các tính năng này cùng một lúc để xây dựng máy dò tốt nhất có thể. Có một số lý do khiến việc sử dụng tất cả các tính năng có thể không phải là một ý kiến hay.

Đầu tiên, trích xuất tất cả các tính năng mà chúng tôi vừa khám phá mất nhiều thời gian, điều này ảnh hưởng đến tốc độ hệ thống của bạn có thể quá tốn kém. Quan trọng hơn, nếu bạn sử dụng quá nhiều tính năng trên các thuật toán máy học, bạn có thể gặp phải các vấn đề về bộ nhớ và hệ thống của bạn có thể mất quá nhiều thời gian để đào tạo. Đây là lý do tại sao khi xây dựng hệ thống của bạn, tôi khuyên bạn nên thử các tính năng khác nhau và tập trung vào những tính năng hoạt động tốt trên các loại phần mềm độc hại mà bạn đang cố gắng phát hiện (và các loại phần mềm độc hại mà bạn đang cố gắng tránh tạo ra kết quả dương tính giả).

Thật không may, ngay cả khi bạn tập trung vào một danh mục tính năng, chẳng hạn như tính năng chuỗi, bạn sẽ thường có nhiều tính năng hơn hầu hết các thuật toán máy học có thể xử lý. Khi sử dụng các tính năng chuỗi, bạn phải có một tính năng cho mỗi chuỗi duy nhất xuất hiện trong dữ liệu đào tạo của bạn. Ví dụ: nếu mẫu đào tạo A chứa chuỗi "hello world" và mẫu đào tạo B chứa chuỗi "hello world!", thì bạn sẽ cần xử lý "hello world" và "hello world!" như hai tính năng riêng biệt. Điều này có nghĩa là khi bạn đang xử lý hàng nghìn mẫu đào tạo, bạn sẽ nhanh chóng gặp phải hàng nghìn chuỗi duy nhất và hệ thống của bạn sẽ sử dụng nhiều tính năng đó.

Sử dụng thủ thuật băm để nén các tính năng

Để giải quyết vấn đề có quá nhiều tính năng, bạn có thể sử dụng một giải pháp phổ biến và đơn giản được gọi là thủ thuật băm, còn được gọi là băm tính năng. Ý tưởng như sau: giả sử bạn có một triệu tính năng chuỗi duy nhất trong tập huấn luyện của mình, nhưng phần cứng và thuật toán máy học mà bạn đang sử dụng chỉ có thể xử lý 4.000 tính năng duy nhất trên toàn bộ tập huấn luyện. Bạn cần một số cách để nén một triệu tính năng xuống một vectơ đặc trưng dài 4.000 mục.

Thủ thuật băm làm cho một triệu tính năng này nambi gọn trong không gian 4.000 tính năng bằng cách băm từng tính năng vào một trong 4.000 chỉ mục. Sau đó, bạn thêm giá trị của đối tượng địa lý ban đầu vào số tại chỉ mục đó trong vectơ đối tượng 4.000 chiều của bạn. Tất nhiên, các tính năng thường xung đột với cách tiếp cận này vì các giá trị của chúng được cộng lại với nhau theo cùng một thứ nguyên. Điều này có thể ảnh hưởng đến độ chính xác của hệ thống vì thuật toán máy học bạn đang sử dụng không thể "thấy" giá trị của các tính năng riêng lẻ nữa. Nhưng trong thực tế, sự suy giảm độ chính xác này thường rất nhỏ và lợi ích mà bạn nhận được từ việc nén biểu diễn các tính năng của mình vượt xa sự suy giảm nhẹ này xảy ra do thao tác nén.

Thực hiện thủ thuật băm

Để làm cho những ý tưởng này rõ ràng hơn, tôi sẽ hướng dẫn bạn mã mẫu triển khai thủ thuật băm. Ở đây tôi trình bày đoạn mã này để minh họa cách thức hoạt động của thuật toán `rithm`; sau này, chúng ta sẽ sử dụng triển khai chức năng này của `sklearn`. Mã mẫu của chúng tôi bắt đầu bằng một khai báo hàm:

```
def apply_hashing_trick(feature_dict, vector_size=2000):
```

Hàm `apply_hashing_trick()` nhận hai tham số: từ điển đối tượng ban đầu và kích thước của vectơ mà chúng tôi lưu trữ vectơ đối tượng nhỏ hơn sau khi áp dụng thủ thuật băm.

Tiếp theo, chúng tôi tạo mảng tính năng mới bằng mã sau:

```
new_features = [0] * len(feature_dict)
```

Mảng `new_features` lưu trữ thông tin tính năng sau khi áp dụng thủ thuật băm. Sau đó, chúng ta thực hiện các thao tác chính của thủ thuật băm bên trong vòng lặp `for`, như trong Liệt kê 8-8.

```
    cho khóa trong ufeature_dict:
        array_index = vhash(key) % vector_size
        new_features[array_index] += wfeature_dict[key]
```

Liệt kê 8-8: Sử dụng vòng lặp for để thực hiện thao tác băm

Ở đây, chúng tôi sử dụng vòng lặp for để lặp qua mọi đối tượng trong từ điển đối tượng . Để làm điều này, trước tiên chúng ta băm các khóa của từ điển (trong trường hợp các tính năng chuỗi, chúng sẽ tương ứng với các chuỗi riêng lẻ của nhị phân phần mềm) modulo vector_size sao cho các giá trị băm được giới hạn giữa 0 và vector_size - 1 . Chúng tôi lưu trữ kết quả của thao tác này trong biến array_index .

Vẫn trong vòng lặp for , chúng ta tăng giá trị của mảng new_feature mục nhập tại chỉ mục array_index bằng giá trị trong mảng đối tượng ban đầu của chúng ta . Trong trường hợp các tính năng chuỗi, trong đó các giá trị tính năng của chúng tôi được đặt thành 1 để cho biết rằng nhị phân phần mềm có chuỗi cụ thể đó, chúng tôi sẽ tăng mục nhập này lên một. Trong trường hợp các tính năng tiêu đề PE, trong đó các tính năng có nhiều giá trị (ví dụ: tương ứng với dung lượng bộ nhớ mà một phần PE sẽ chiếm), chúng tôi sẽ thêm giá trị của tính năng vào mục nhập.

Cuối cùng, bên ngoài vòng lặp for , chúng ta chỉ cần trả về từ điển new_features , như sau:

trả về new_features

Tại thời điểm này, sklearn có thể hoạt động trên new_features chỉ bằng cách sử dụng hàng nghìn thay vì hàng triệu tính năng độc đáo.

Mã hoàn chỉnh cho thủ thuật băm

Liệt kê 8-9 hiển thị mã hoàn chỉnh cho thủ thuật băm, mà bây giờ bạn đã quen thuộc.

```
def apply_hashing_trick(feature_dict,vector_size=2000):
    # tạo một mảng các số 0 có độ dài 'vector_size'
    new_features = [0 for _ in range(vector_size)]

    # lặp lại mọi tính năng trong từ điển tính năng
    # cho khóa trong feature_dict:

        # lấy chỉ mục vào mảng tính năng mới
        array_index = hash(key) % vector_size

        # thêm giá trị của tính năng vào mảng tính năng mới
        # tại chỉ mục chúng tôi đã sử dụng thủ thuật băm
        new_features[array_index] += feature_dict[key]
```

trả về new_features

Liệt kê 8-9: Mã hoàn chỉnh để triển khai thủ thuật băm

Như bạn đã thấy, thủ thuật băm tính năng rất dễ thực hiện theo cách riêng của bạn và làm như vậy đảm bảo rằng bạn hiểu cách thức hoạt động của nó. Tuy nhiên, bạn cũng có thể chỉ sử dụng triển khai của sklearn , dễ sử dụng và tối ưu hóa hơn.

Sử dụng FeatureHasher của sklearn

Để sử dụng triển khai tích hợp sẵn của sklearn thay vì triển khai giải pháp băm của riêng bạn, trước tiên bạn cần nhập lớp FeatureHasher của sklearn , như sau:

từ sklearn.feature_extract nhập FeatureHasher

Tiếp theo, khởi tạo lớp FeatureHasher :

máy băm = FeatureHasher(n_features=2000)

Để làm điều này, bạn khai báo n_features là kích thước của mảng mới do áp dụng thủ thuật băm.

Sau đó, để áp dụng thủ thuật băm cho một số vector đặc trưng, bạn chỉ cần chạy chúng thông qua phương thức biến đổi của lớp FeatureHasher :

```
features = [{"how": 1, "now": 2, "brown": 4}, {"cow": 2, ".": 5}]
hashed_features = hasher.transform(tính năng)
```

Kết quả thực sự giống như việc triển khai tùy chỉnh thủ thuật băm tính năng của chúng tôi được hiển thị trong Liệt kê 8-9. Sự khác biệt là ở đây chúng tôi chỉ đơn giản sử dụng triển khai của sklearn , vì việc sử dụng thư viện máy học được duy trì tốt sẽ dễ dàng hơn so với mã của chính chúng tôi. Mã mẫu hoàn chỉnh được hiển thị trong Liệt kê 8-10.

từ sklearn.feature_extract nhập FeatureHasher

máy băm = FeatureHasher(n_features=10)
features = [{"how": 1, "now": 2, "brown": 4}, {"cow": 2, ".": 5}]
hashed_features = hasher.transform(tính năng)

Liệt kê 8-10: Triển khai FeatureHasher

Có một số điều cần lưu ý về tính năng băm trước khi chúng ta tiếp tục. Đầu tiên, như bạn có thể đoán, tính năng băm tính năng làm xáo trộn thông tin tính năng mà bạn chuyển vào thuật toán máy học vì nó cộng các giá trị tính năng lại với nhau chỉ đơn giản dựa trên thực tế là chúng được băm vào cùng một thùng.

Nói chung, điều này có nghĩa là bạn sử dụng càng ít ngăn (hoặc càng nhiều tính năng bạn băm vào một số lượng ngăn cố định), thuật toán của bạn sẽ hoạt động càng kém. Đáng ngạc nhiên là các thuật toán học máy vẫn có thể hoạt động tốt ngay cả khi sử dụng thủ thuật băm, và vì đơn giản là chúng ta không thể xử lý hàng triệu hoặc hàng tỷ tính năng trên phần cứng hiện đại, chúng ta thường phải sử dụng thủ thuật băm tính năng trong khoa học dữ liệu bảo mật.

Một hạn chế khác của thủ thuật băm tính năng là nó làm cho nó khác khó hoặc không thể khôi phục các tính năng ban đầu mà bạn đã băm khi

phân tích nội bộ của mô hình của bạn. Hãy ví dụ về cây quyết định: bởi vì chúng tôi đang băm các tính năng tùy ý vào từng mục trong vectơ đặc trưng của mình, chúng tôi không biết tính năng nào chúng tôi đã thêm vào một mục nhất định đang khiến thuật toán cây quyết định bị phân tách trên mục này, vì bất kỳ số lượng tính năng nào cũng có thể khiến cây quyết định nghĩ rằng việc tách mục nhập này là một ý tưởng hay. Mặc dù đây là một hạn chế đáng kể, nhưng các nhà khoa học dữ liệu bảo mật vẫn chấp nhận nó vì những lợi ích to lớn của thủ thuật băm tính năng trong việc nén hàng triệu tính năng xuống một con số có thể quản lý được.

Bây giờ, chúng ta đã xem qua các khía cạnh xây dựng cần thiết để xây dựng trình phát hiện phần mềm độc hại trong thế giới thực, hãy khám phá cách xây dựng trình phát hiện phần mềm độc hại máy học từ đầu đến cuối.

Xây dựng máy dò sức mạnh công nghiệp

Từ góc độ yêu cầu phần mềm, trình phát hiện trong thế giới thực của chúng tôi sẽ cần thực hiện ba việc: trích xuất các tính năng từ tệp nhị phân phần mềm để sử dụng trong đào tạo và phát hiện, tự đào tạo để phát hiện phần mềm độc hại bằng cách sử dụng dữ liệu đào tạo và thực sự thực hiện phát hiện trên tệp nhị phân phần mềm mới. Hãy xem qua đoạn mã thực hiện từng việc này, đoạn mã này sẽ cho bạn thấy tất cả chúng ăn khớp với nhau như thế nào.

Bạn có thể truy cập mã tôi sử dụng trong phần này tại [malware_data_science/ch8/code/complete_detector.py](#) trong mã đi kèm với cuốn sách này hoặc tại cùng một vị trí trong máy ảo được cung cấp cùng với cuốn sách này. Tập lệnh trình bao một dòng, [malware_data_science/ch8/code/run_complete_detector.sh](#), cho biết cách chạy trình phát hiện từ trình bao.

Trích xuất các tính năng

Để tạo bộ phát hiện của chúng tôi, điều đầu tiên chúng tôi triển khai là mã để trích xuất các tính năng từ các tệp nhị phân đào tạo (Tôi bỏ qua mã soạn sẵn ở đây và tập trung vào các chức năng cốt lõi của chương trình). Trích xuất các tính năng liên quan đến việc trích xuất dữ liệu có liên quan từ các tệp nhị phân đào tạo, lưu trữ các tính năng này trong từ điển Python và sau đó, nếu chúng tôi nghĩ rằng số lượng tính năng duy nhất của mình sẽ trở nên quá lớn, hãy chuyển đổi chúng bằng cách sử dụng thủ thuật băm triễn khai của sklearn .

Để đơn giản, chúng tôi chỉ sử dụng các tính năng chuỗi và chọn sử dụng thủ thuật băm. Liệt kê 8-11 cho thấy cách thực hiện cả hai.

```
def get_string_features(upath,vhasher):
    # trích xuất chuỗi từ tệp nhị phân bằng biểu thức chính quy
    ký tự = r" ~"
    min_length = 5
    string_regexp = '[%s]{%d,}' % (ký tự, độ dài tối thiểu)
    file_object = mở (đường dẫn)
    dữ liệu = file_object.read()
    mẫu = re.compile(string_regexp)
    chuỗi = pattern.findall(dữ liệu)

    # lưu trữ các tính năng chuỗi ở dạng từ điển
    string_features = {}
```

```

    cho chuỗi trong chuỗi:
    string_features[string] = 1

    # băm các tính năng bằng thủ thuật băm
    hashed_features = hasher.transform([string_features])

    # thực hiện trộn dữ liệu để lấy mảng đặc trưng
    hashed_features = hashed_features.todense()
    hashed_features = numpy.asarray(hashed_features)
    hashed_features = hashed_features[0]

    # trả về các tính năng chuỗi băm
    in "Đã trích xuất {0} chuỗi từ {1}".format(len(string_features),path)
    trả về hashed_features

```

Liệt kê 8-11: Định nghĩa hàm get_string_features

Ở đây, chúng tôi khai báo một hàm gọi là `get_string_features` nhận đường dẫn đến nhị phân đích và một thể hiện của lớp băm tính năng của `sklearn` làm đối số của nó. Sau đó, chúng tôi trích xuất các chuỗi của tệp đích bằng biểu thức chính quy, biểu thức này phân tích tất cả các chuỗi có thể in được có độ dài tối thiểu 5. Sau đó, chúng tôi lưu trữ các tính năng trong từ điển Python để xử lý thêm bằng cách đặt giá trị của mỗi chuỗi thành 1 trong từ điển, chỉ cầnindi xác nhận rằng tính năng đó có trong tệp nhị phân.

Tiếp theo, chúng tôi băm các tính năng bằng cách triển khai thủ thuật băm của `sklearn` bằng cách gọi `hasher`. Lưu ý rằng chúng tôi bọc từ điển `string_features` trong một danh sách Python khi chúng tôi chuyển nó vào đối tượng hàm băm vì `sklearn` yêu cầu chúng tôi chuyển vào một danh sách các từ điển sẽ được chuyển đổi, thay vì một từ điển duy nhất.

Bởi vì chúng tôi đã chuyển vào từ điển tính năng của mình dưới dạng danh sách từ điển, nên các tính năng được trả về dưới dạng danh sách các mảng. Ngoài ra, chúng được trả về ở định dạng thừa thớt, một biểu diễn nén có thể hữu ích để xử lý các ma trận lớn, điều mà chúng ta sẽ không thảo luận trong cuốn sách này. Chúng ta cần đưa dữ liệu của mình trở lại một vectơ có nhiều mảng thông thường.

Để đưa dữ liệu trở lại định dạng bình thường, chúng tôi gọi `.todense()` và `.asarray()`, sau đó chọn mảng đầu tiên trong danh sách kết quả của trình băm để khôi phục vectơ đặc trưng cuối cùng của chúng tôi. Bước cuối cùng trong hàm chỉ đơn giản là trả về vectơ đặc trưng `hashed_features` cho người gọi.

Huấn luyện máy dò

Bởi vì `sklearn` thực hiện hầu hết các công việc khó khăn trong việc đào tạo hệ thống máy học, nên việc đào tạo một trình phát hiện chỉ yêu cầu một lượng mã nhỏ sau khi chúng tôi trích xuất các tính năng máy học từ các tệp nhị phân mục tiêu của mình.

Để đào tạo trình phát hiện, trước tiên chúng tôi cần trích xuất các tính năng từ các ví dụ đào tạo của mình, sau đó khởi tạo trình băm tính năng và trình phát hiện máy học `sklearn` mà chúng tôi muốn sử dụng (trong trường hợp này, chúng tôi sử dụng trình phân loại nhóm rứng ngẫu nhiên). Sau đó, chúng ta cần gọi phương thức phù hợp của `sklearn` trên trình phát hiện để huấn luyện nó trên các tệp nhị phân của ví dụ. Cuối cùng, chúng tôi lưu trình phát hiện và trình băm tính năng vào đĩa để có thể sử dụng chúng khi muốn quét tệp trong tương lai.

Liệt kê 8-12 cho thấy mã để đào tạo máy dò.

```

def uget_training_data(benign_path,malicious_path,hasher):
    def vget_training_paths(thư mục):
        mục tiêu = []
        cho đường dẫn trong os.listdir(thư mục):
            target.append(os.path.join(thư mục,đường dẫn))
        trả lại mục tiêu
    malware_paths = get_training_paths(malicious_path)
    ben_paths = get_training_paths(benign_path)
    X = [get_string_features(path,hasher)
        cho đường dẫn trong malware_paths + ben_paths]
    y = [1 for i in range(len(malicious_paths))] + [0
        for i in range(len(benign_paths))]
    trả về X, y
def ztrain_detector(X,y,hasher):
    bộ phân loại = tree.RandomForestClassifier()
    classifier.fit(X,y)
    pickle.dump((classifier,hasher),open("saved_detector.pkl","w+"))

```

Liệt kê 8-12: Lập trình sklearn để huấn luyện máy dò

Hãy bắt đầu bằng cách khai báo hàm `get_training_data()` , hàm này trích xuất các tính năng từ các ví dụ đào tạo mà chúng tôi cung cấp. Hàm này có ba đối số: đường dẫn đến thư mục chứa các ví dụ về chương trình nhị phân lành tính (`benign_path`) , đường dẫn đến thư mục chứa ví dụ về chương trình nhị phân độc hại (`malicious_path`) và một thê hiện của lớp `sklearn FeatureHasher` được sử dụng để thực hiện băm tính năng (máy băm).

Tiếp theo, chúng tôi khai báo `get_training_paths()` , một hàm trợ giúp cục bộ cung cấp cho chúng tôi danh sách các đường dẫn tệp tuyệt đối cho các tệp xuất hiện trong một thư mục nhất định. Trong hai dòng tiếp theo, chúng ta sử dụng `get_training_paths` để lấy danh sách các đường dẫn xuất hiện trong các thư mục mẫu đào tạo độc hại và lành tính.

Cuối cùng, chúng tôi trích xuất các tính năng của mình và tạo vectơ nhän. Chúng ta làm điều này bằng cách gọi hàm `get_string_features` được mô tả trong Liệt kê 8-11 trên mọi đường dẫn tệp ví dụ đào tạo . Lưu ý rằng vectơ nhän có 1 cho mọi đường dẫn độc hại và 0 cho mọi đường dẫn lành tính, sao cho các số tại các chỉ số trong vectơ nhän tương ứng với nhän của các vectơ đặc trưng tại cùng các chỉ số đó trong mảng X. Đây là biểu mẫu mà sklearn mong đợi dữ liệu nhän và tính năng, đồng thời nó cho phép chúng tôi bảo cho thư viện biết nhän cho mỗi vectơ đặc trưng.

Bây giờ chúng tôi đã hoàn thành việc trích xuất các tính năng và tạo vectơ đặc trưng X và vectơ nhän y của chúng tôi, chúng tôi đã sẵn sàng yêu cầu sklearn huấn luyện bộ phát hiện của chúng tôi bằng cách sử dụng vectơ đặc trưng và vectơ nhän.

Chúng tôi thực hiện việc này bằng cách sử dụng hàm `train_detector()` , hàm này nhận ba đối số: vectơ đặc trưng ví dụ đào tạo (X) , vectơ nhän (y) và ví dụ về hàm băm tính năng của sklearn (hàm băm). Trong phần thân hàm, chúng ta khởi tạo cây.`RandomForestClassifier`, trình phát hiện sklearn . Sau đó, chúng tôi chuyển X và y vào phương thức phù hợp của trình phát hiện để huấn luyện nó , sau đó sử dụng mô-đun `pickle` của Python để lưu trình phát hiện và trình băm để sử dụng trong tương lai.

Chạy Trình dò trên các tệp nhị phân mới

Bây giờ, hãy xem cách sử dụng trình phát hiện đã lưu mà chúng tôi vừa đào tạo để phát hiện phần mềm độc hại trong các tệp nhị phân của chương trình mới. Liệt kê 8-13 cho thấy cách viết hàm `scan_file()` để thực hiện việc này.

```
def scan_file (đường dẫn):
    nếu không phải os.path.exists("saved_detector.pkl"):
        print "Đào tạo máy dò trước khi quét tệp."
        sys.exit(1)
    với open("saved_detector.pkl") là save_detector:
        phân loại, hasher = pickle.load(save_detector)
        các tính năng = vget_string_features(path,hasher)
        result_proba = wclassifier.predict_proba( tính năng)[1]
        # nếu người dùng chỉ định Malware_paths và #
        Beneware_paths, hãy đào tạo trình phát hiện
        nếu result_proba > 0,5:
            print "Có vẻ như tệp tin này độc hại!", `result_proba`
        khác:
            print "Có vẻ như tệp tin này lành tính.", `result_proba`
```

Liệt kê 8-13: Chạy máy dò trên các tệp nhị phân mới

Ở đây, chúng tôi khai báo hàm `scan_file()` để quét một tệp để xác định xem tệp đó độc hại hay lành tính. Đối số duy nhất của nó là đường dẫn đến tệp nhị phân mà chúng ta sẽ quét. Công việc đầu tiên của hàm là tải trình phát hiện và trình băm đã lưu từ tệp dựa chua mà chúng đã được lưu vào .

Tiếp theo, chúng tôi trích xuất các tính năng từ tệp đích bằng cách sử dụng hàm `get_string_features` mà chúng tôi đã xác định trong Liệt kê 8-11.

Cuối cùng, chúng tôi gọi phương pháp dự đoán của máy dò để quyết định có hay không tệp được đề cập là độc hại, do các tính năng được trích xuất. Chúng tôi thực hiện việc này bằng cách sử dụng phương thức `predict_proba` của phiên bản trình phân loại và chọning phần tử thứ hai của mảng mà nó trả về, tương ứng với xác suất tệp độc hại. Nếu xác suất này lớn hơn 0,5 hoặc 50 phần trăm , thì chúng tôi nói rằng tệp đó độc hại; mặt khác, chúng tôi nói với người dùng rằng nó lành tính. Chúng ta có thể thay đổi ngưỡng quyết định này thành ngưỡng cao hơn nhiều để giảm thiểu kết quả dương tính giả.

Những gì chúng tôi đã thực hiện cho đến nay

Liệt kê 8-14 hiển thị toàn bộ mã cho trình phát hiện phần mềm độc hại có quy mô nhỏ nhưng thực tế này. Tôi hy vọng rằng mã này sẽ đọc trôi chảy cho bạn khi bạn đã thấy cách thức hoạt động của từng phần riêng lẻ.

```
#!/usr/bin/trần
```

```
nhập hệ điều hành
nhập khẩu hệ thống
dựa chua nhập khẩu
nhập argparse
nhập lại
nhập numpy
```

```

từ sklearn.ensemble nhập RandomForestClassifier từ
sklearn.feature_extraction nhập FeatureHasher

def get_string_features(path,hasher):
    # trích xuất các chuỗi từ tệp nhị phân bằng biểu thức chính quy
    chars = r" -~"
    min_length = 5
    string_regex = '[%s]{%d,}' % (chars, min_length)
    file_object = open( đường
    dẫn) data =
    file_object.read() pattern =
    re.compile(string_regex) strings = pattern.findall(data)

    # lưu trữ các tính năng chuỗi ở dạng từ điển
    string_features = {}
    cho chuỗi trong chuỗi:
        string_features[string] = 1

    # băm các tính năng bằng thủ thuật băm
    hased_features = hasher.transform([string_features])

    # trộn dữ liệu để lấy mảng đối tượng hasshed_features
    = hashed_features.todense() hashed_features =
    numpy.asarray(hashed_features) hased_features =
    hash_features[0]

    # return hash string features
    print "Đã trích xuất {} chuỗi từ {}".format(len(string_features),path)
    return hased_features

def scan_file(path):
    # quét một tập tin để xác định xem nó độc hại hay lành tính
    nếu không os.path.exists("saved_detector.pkl"):
        print "Đào tạo máy dò trước khi quét tệp."
        sys.exit(1)
    với open("saved_detector.pkl") dưới dạng save_detector:
        phân loại, hasher = pickle.load(save_detector)
    features = get_string_features(path,hasher)
    result_proba = classifier.predict_proba([features])[:,1] #
    nếu người dùng chỉ định malware_paths và #
    benzenware_paths, đào tạo trình phát
    hiện nếu result_proba
        > 0.5: print "Có vẻ như tệp này độc hại! ",`result_proba`

    other: print "Có vẻ như tệp này lành tính.",`result_proba`

def train_detector(benign_path,malicious_path,hasher): #
    đào tạo trình phát hiện trên dữ liệu đào tạo đã chỉ
    định def get_training_paths(directory):
        target = []
        cho đường dẫn trong os.listdir(thư
        mục): target.append(os.path.join(thư mục ,path))
        trả về mục
    tiêu malware_paths =
    get_training_paths(malicious_path)benign_paths = get_training_paths(benign_path)

```

```

X = [get_string_features(path,hasher) cho đường dẫn trong malware_paths + ben_paths]
y = [1 for i in range(len(malicious_paths))] + [0 for i in range(len(benign_paths))]
bộ phân loại = tree.RandomForestClassifier(64)
phân loại.fit(X,y)
pickle.dump((classifier,hasher),open("saved_detector.pkl","w+"))

def get_training_data(benign_path,malicious_path,hasher):
    def get_training_paths (thư mục):
        mục tiêu = []
        cho đường dẫn trong os.listdir(thư mục):
            target.append(os.path.join(thư mục,đường dẫn))
        trở lại mục tiêu
    malware_paths = get_training_paths(malicious_path)
    ben_paths = get_training_paths(benign_path)
    X = [get_string_features(path,hasher) cho đường dẫn trong malware_paths + ben_paths]
    y = [1 for i in range(len(malicious_paths))] + [0 for i in range(len(benign_paths))]
    trả về X, y

trình phân tích cú pháp = argparse.ArgumentParser("lấy vectơ đối tượng của số cho tệp")
parser.add_argument("--malware_paths",default=None,help="Đường dẫn đến tệp đào tạo phần mềm độc hại")
parser.add_argument("--benignware_paths",default=None,help="Đường dẫn đến tệp đào tạo phần mềm lành tính")
parser.add_argument("--scan_file_path",default=None,help="Tệp cần quét")
args = trình phân tích cú pháp.parse_args()

máy băm = FeatureHasher(20000)
nếu args.malware_paths và args.benignware_paths:
    train_detector(args.benignware_paths,args.malware_paths,hasher)
Elif args.scan_file_path:
    scan_file(args.scan_file_path)
khác:
    print "[*] Bạn không chỉ định đường dẫn để quét, " \
          " bạn cũng không chỉ định đường dẫn đến các tệp đào tạo lành tính và độc hại" \
          " vui lòng chỉ định một trong số này để sử dụng trình phát hiện.\n"
    trình phân tích cú pháp.print_help()

```

Liệt kê 8-14: Mã phát hiện phần mềm độc hại máy học cơ bản

Viết một trình phát hiện phần mềm độc hại dựa trên máy học là điều tuyệt vời, nhưng việc đánh giá và cải thiện hiệu suất của nó là cần thiết nếu bạn định triển khai trình phát hiện với bất kỳ sự tin nào về hiệu quả của nó. Tiếp theo, bạn tìm hiểu các cách khác nhau để đánh giá hiệu suất của máy dò.

Đánh giá hiệu suất của máy dò của bạn

Thuận tiện là sklearn chứa mã giúp dễ dàng đánh giá các hệ thống phát hiện bằng cách sử dụng các phép đo như đường cong ROC mà bạn đã tìm hiểu trong Chương 7. Thư viện sklearn cũng cung cấp chức năng đánh giá bổ sung dành riêng cho việc đánh giá các hệ thống máy học. Ví dụ: bạn có thể sử dụng các chức năng của sklearn để thực hiện xác thực chéo, đây là một phương pháp hiệu quả để dự đoán trình phát hiện của bạn sẽ hoạt động tốt như thế nào khi bạn triển khai nó.

Trong phần này, bạn tìm hiểu cách sử dụng sklearn để vẽ các đường cong ROC thể hiện độ chính xác của máy dò. Bạn cũng tìm hiểu về xác thực chéo và cách triển khai nó với sklearn.

Sử dụng đường cong ROC để đánh giá hiệu quả của máy dò

Hãy nhớ rằng các đường cong Đặc tính hoạt động của máy thu (ROC) đo lường các thay đổi về tỷ lệ dương tính thực sự của trình phát hiện (tỷ lệ phần trăm phần mềm độc hại mà nó phát hiện thành công) và tỷ lệ dương tính giả (tỷ lệ phần mềm lành tính mà nó đánh dấu nhầm là phần mềm độc hại) khi bạn điều chỉnh độ nhạy của nó.

Độ nhạy càng cao, bạn càng nhận được nhiều kết quả dương tính giả nhưng tỷ lệ phát hiện của bạn càng cao. Độ nhạy càng thấp, càng ít kết quả dương tính giả nhưng bạn càng nhận được ít phát hiện hơn. Để tính toán đường cong ROC, bạn cần một công cụ phát hiện có thể đưa ra điểm số về mỗi đe dọa sao cho giá trị của nó càng cao thì càng có nhiều khả năng tệp nhị phân là độc hại. Thuận tiện là, việc triển khai cây quyết định, hồi quy logistic, k-lảng giềng gần nhất, rồng ngẫu nhiên và các phương pháp học máy khác của sklearn được đề cập trong cuốn sách này đều cung cấp tùy chọn xuất ra điểm số mỗi đe dọa phần ánh liêu một tệp là phần mềm độc hại hay phần mềm lành tính. Hãy khám phá cách chúng ta có thể sử dụng các đường cong ROC để xác định độ chính xác của máy dò.

Tính toán đường cong ROC

Để tính toán đường cong ROC cho trình phát hiện máy học mà chúng ta đã tạo trong Liệt kê 8-14, chúng ta cần thực hiện hai việc: thứ nhất, xác định thiết lập thử nghiệm và thứ hai, triển khai thử nghiệm bằng cách sử dụng mô-dun số liệu của sklearn. Đối với thiết lập thử nghiệm cơ bản của chúng tôi, chúng tôi sẽ chia các ví dụ đào tạo của mình thành hai nửa sao cho chúng tôi sử dụng nửa đầu để đào tạo và nửa sau để tính toán đường cong ROC. Sự phân chia này mô phỏng vẫn đề phát hiện phần mềm độc hại zero-day.

Về cơ bản, bằng cách chia nhỏ dữ liệu, chúng tôi đang nói với chương trình, "hãy cho tôi xem một bộ phần mềm độc hại và phần mềm độc hại mà tôi sẽ sử dụng để tìm hiểu cách xác định phần mềm độc hại và phần mềm độc hại, sau đó cho tôi xem bộ khác để kiểm tra xem tôi hoạt động tốt như thế nào."

Tôi đã học được khái niệm về phần mềm độc hại và phần mềm độc hại." Do trình phát hiện chưa bao giờ nhìn thấy phần mềm độc hại (hoặc phần mềm lành tính) trong tập hợp thử nghiệm nên thiết lập đánh giá này là một cách đơn giản để dự đoán trình phát hiện sẽ hoạt động tốt như thế nào đối với phần mềm độc hại thực sự mới.

Việc thực hiện phân tách này với sklearn rất đơn giản. Trước tiên, chúng tôi thêm một tùy chọn vào lớp trình phân tích cú pháp đối số của chương trình trình phát hiện để báo hiệu rằng chúng tôi muốn đánh giá độ chính xác của trình phát hiện, như sau:

```
trình phân tích cú pháp.add_argument("--evaluate", default=False,
action="store_true", help="Thực hiện xác thực chéo")
```

Sau đó, trong một phần của chương trình nơi chúng tôi xử lý dòng lệnh argu như trong Liệt kê 8-15, chúng ta thêm một mệnh đề elif khác để xử lý trường hợp mà người dùng đã thêm -evaluate vào các đối số dòng lệnh.

```

elif args.malware_paths và args.benignware_paths và args.evaluate:
    hasher = FeatureHasher()
    X, y = vget_training_data(
        args.benignware_paths, args.malware_paths, hasher)
    đánh giá(X, y, hasher)
def wevaluate(X, y, hasher):
    nhập ngẫu nhiên
    từ só liệu nhập sklearn
    từ matplotlib nhập pyplot

```

Liệt kê 8-15: Chạy máy dò trên các tệp nhị phân mới

Hãy xem qua mã này một cách chi tiết. Đầu tiên, chúng tôi khởi tạo một sklearn trình băm tính năng , lấy dữ liệu đào tạo mà chúng tôi yêu cầu cho thử nghiệm đánh giá , sau đó gọi hàm có tên đánh giá , hàm này lấy dữ liệu đào tạo (X, y) và phiên bản trình băm tính năng (hàm băm) làm tham số của nó rồi nhập ba mô-đun chúng ta cần thực hiện đánh giá. Chúng tôi sử dụng mô-đun ngẫu nhiên để chọn ngẫu nhiên ví dụ huấn luyện nào sẽ sử dụng để huấn luyện máy dò và ví dụ nào sẽ sử dụng để kiểm tra nó. Chúng tôi sử dụng mô-đun số liệu từ sklearn để tính toán đường cong ROC và chúng tôi sử dụng mô-đun pyplot từ matplotlib (thư viện Python chuẩn trên thực tế để trực quan hóa dữ liệu) để trực quan hóa đường cong ROC.

Tách dữ liệu thành các tập huấn luyện và kiểm tra

Bây giờ chúng ta đã sắp xếp ngẫu nhiên các mảng X và y tương ứng với dữ liệu huấn luyện của mình, chúng ta có thể chia các mảng này thành các tập huấn luyện và kiểm tra có kích thước bằng nhau, như trong Liệt kê 8-16, tiếp tục xác định hàm đánh giá () đã bắt đầu trong Liệt kê 8-15.

```

X, y = numpy.array(X), numpy.array(y)
chi số = phạm vi (len (y))
random.shuffle(chi số)
X, y = X[chi số], y[chi số]
diểm chia = len(X) *           0,5
diểm chia = int(diểm chia)
training_X, test_X = X[:splitpoint], X[splitpoint:]
training_y, test_y = y[:splitpoint], y[splitpoint:]

```

Liệt kê 8-16: Tách dữ liệu thành tập huấn luyện và kiểm tra

Đầu tiên, chúng tôi chuyển đổi X và y thành các mảng numpy , sau đó chúng tôi tạo một danh sách các chỉ số tương ứng với số phần tử trong X và y . Tiếp theo, chúng tôi xáo trộn ngẫu nhiên các chỉ số này và sắp xếp lại X và y dựa trên thứ tự mới này . Điều này cho phép chúng tôi chỉ định ngẫu nhiên các mẫu cho tập huấn luyện hoặc tập kiểm tra của chúng tôi, đảm bảo rằng chúng tôi không phân chia các mẫu đơn giản theo thứ tự xuất hiện trong thư mục dữ liệu thử nghiệm của chúng tôi. Để hoàn thành quá trình phân tách ngẫu nhiên, chúng tôi chia các mảng thành hai nửa bằng cách tìm chỉ số mảng chia đều của tập dữ liệu thành hai nửa, làm tròn điểm này thành số nguyên gần nhất bằng cách sử dụng hàm int() , sau đó thực sự tách mảng X và y thành tập huấn luyện và kiểm tra .

Bây giờ chúng ta đã có tập huấn luyện và kiểm tra, chúng ta có thể khởi tạo và huấn luyện trình phát hiện cây quyết định của chúng tôi bằng cách sử dụng dữ liệu đào tạo bằng cách sử dụng như sau:

```
bộ phân loại = RandomForestClassifier()
classifier.fit(đào tạo_X, đào tạo_y)
```

Sau đó, chúng tôi sử dụng trình phân loại được đào tạo để lấy điểm cho các ví dụ thử nghiệm tương ứng với khả năng các ví dụ thử nghiệm này là độc hại:

```
điểm = classifier.predict_proba(test_X)[:, -1]
```

Ở đây, chúng tôi gọi phương thức predict_proba() trên bộ phân loại của chúng tôi, phương thức này dự đoán khả năng các mẫu thử nghiệm của chúng tôi là phần mềm lành tính hoặc phần mềm độc hại. Sau đó, bằng cách sử dụng phép thuật lập chỉ mục numpy , chúng tôi chỉ rút ra xác suất mà các mẫu là độc hại, trái ngược với xác suất lành tính. Hãy nhớ rằng những xác suất này là dư thừa (ví dụ: nếu xác suất một ví dụ là độc hại là 0,99, thì xác suất nó lành tính là 0,01, vì các xác suất cộng lại bằng 1), vì vậy tất cả những gì chúng ta cần là xác suất phần mềm độc hại ở đây.

Tính toán đường cong ROC

Bây giờ chúng tôi đã tính toán xác suất phần mềm độc hại (mà chúng tôi cũng có thể gọi là "điểm số") bằng trình phát hiện của mình, đến lúc tính toán đường cong ROC của chúng tôi. Chúng tôi thực hiện việc này bằng cách gọi hàm roc_curve đầu tiên trong mô-đun số liệu của sklearn , như sau:

```
fpr, tpr, ngưỡng = metrics.roc_curve(test_y, điểm số)
```

Hàm roc_curve kiểm tra nhiều ngưỡng quyết định khác nhau hoặc các ngưỡng điểm mà trên đó chúng tôi cho rằng một phần mềm nhị phân là độc hại và do lưỡng tỷ lệ dương tính giả và tỷ lệ dương tính thật của bộ phát hiện sẽ là bao nhiêu nếu chúng ta sử dụng bộ phát hiện đó.

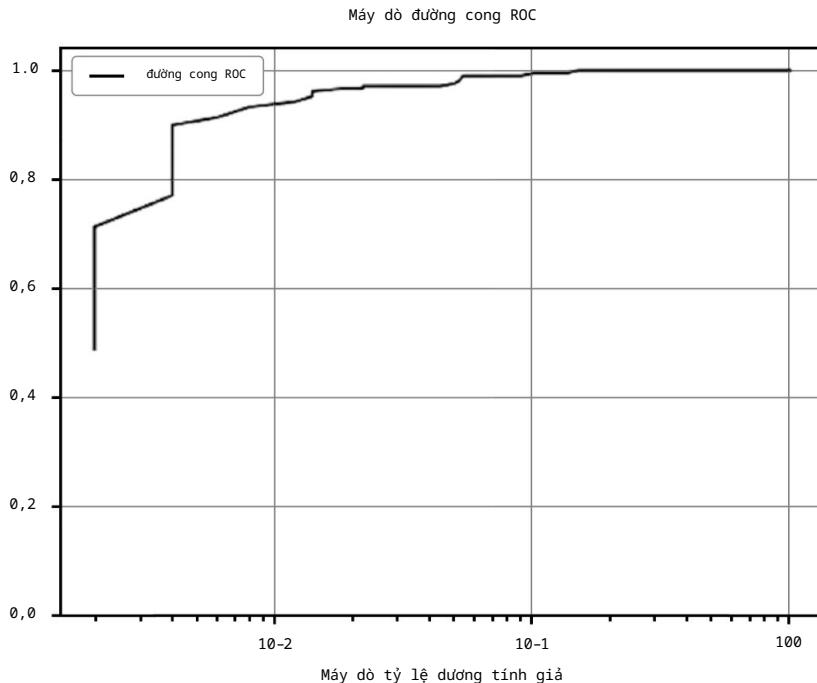
Bạn có thể thấy rằng hàm roc_curve nhận hai đối số: vectơ nhãn cho các ví dụ thử nghiệm của chúng tôi (test_y) và mảng điểm số , chứa phán đoán của trình phát hiện của chúng tôi về mức độ độc hại của mỗi ví dụ đào tạo. Hàm trả về ba mảng liên quan: fpr, tpr và ngưỡng . Các mảng này đều có độ dài bằng nhau, sao cho tỷ lệ dương tính giả, tỷ lệ dương tính thật và ngưỡng quyết định tại mỗi chỉ mục tương ứng với nhau.

Bây giờ chúng ta có thể sử dụng matplotlib để trực quan hóa đường cong ROC mà chúng ta vừa tính toán. Chúng tôi làm điều này bằng cách gọi phương thức cốt truyện trên mô-đun pyplot của matplotlib , như được hiển thị ở đây:

```
pyplot.plot(fpr, tpr, 'r-')
pyplot.xlabel("Tỷ lệ dương tính giả của máy dò")
pyplot.ylabel("Tỷ lệ xác thực của máy dò")
pyplot.title("Đường cong ROC của máy dò")
pyplot.show()
```

Chúng tôi gọi các phương thức xlabel, ylabel và title để gắn nhãn cho các trục của biểu đồ và tiêu đề, sau đó là phương thức hiển thị để làm cho cửa sổ biểu đồ bật lên.

Đường cong ROC kết quả được hiển thị trong Hình 8-2.



Hình 8-2: Trực quan hóa đường cong ROC của máy dò

Bạn có thể thấy từ biểu đồ trong Hình 8-2 rằng máy dò của chúng tôi hoạt động tốt đối với một ví dụ cơ bản như vậy. Với tỷ lệ dương tính giả khoảng 1 phần trăm (10^{-2}), nó có thể phát hiện khoảng 94 phần trăm mẫu phần mềm độc hại trong bộ thử nghiệm. Chúng tôi chỉ đào tạo nó trên vài trăm ví dụ đào tạo ở đây; để có được độ chính xác cao hơn, chúng ta cần đào tạo nó trên hàng chục nghìn, hàng trăm nghìn hoặc thậm chí hàng triệu ví dụ (than ôi, việc mở rộng quy mô học máy đến mức độ này nằm ngoài phạm vi của cuốn sách này).

Xác thực chéo

Mặc dù trực quan hóa đường cong ROC là hữu ích, nhưng chúng ta thực sự có thể làm tốt hơn trong việc dự đoán độ chính xác trong thế giới thực của máy dò bằng cách thực hiện nhiều thử nghiệm trên dữ liệu đào tạo của mình, không chỉ một. Hãy nhớ lại rằng để thực hiện thử nghiệm của mình, chúng tôi chia các ví dụ huấn luyện của mình thành hai nửa, huấn luyện máy dò ở nửa đầu và kiểm tra nó ở nửa sau. Đây thực sự là một bài kiểm tra không đầy đủ đối với máy dò của chúng tôi. Trong thế giới thực, chúng tôi sẽ không được đánh giá dựa trên độ chính xác của chúng tôi đối với tập hợp ví dụ thử nghiệm cụ thể này mà dựa trên độ chính xác của chúng tôi đối với phần mềm độc hại mới, chưa từng thấy trước đây. Để hiểu rõ hơn về cách chúng tôi sẽ thực hiện khi chúng tôi

triển khai, chúng tôi cần chạy nhiều hơn một thử nghiệm trên một bộ dữ liệu thử nghiệm; chúng ta cần thực hiện nhiều thử nghiệm trên nhiều bộ thử nghiệm và hiểu được xu hướng chung về độ chính xác.

Chúng ta có thể sử dụng xác thực chéo để làm điều này. Ý tưởng cơ bản dồn sau xác thực chéo là chia các ví dụ đào tạo của chúng tôi thành một số nếp gấp (ở đây tôi sử dụng ba nếp gấp, nhưng bạn có thể sử dụng nhiều hơn). Ví dụ: nếu bạn có 300 ví dụ và quyết định chia chúng thành ba phần, 100 mẫu đầu tiên sẽ được đưa vào phần đầu tiên, 100 mẫu thứ hai sẽ được đưa vào phần thứ hai và 100 mẫu thứ ba sẽ được đưa vào phần thứ ba.

Sau đó, chúng tôi chạy ba bài kiểm tra. Trong lần kiểm tra đầu tiên, chúng tôi huấn luyện hệ thống ở lần 2 và 3 và kiểm tra hệ thống ở lần 1. Ở lần kiểm tra thứ hai, chúng tôi lặp lại quy trình này nhưng huấn luyện hệ thống ở lần 1 và 3 và kiểm tra hệ thống ở lần 2. Trên thử nghiệm thứ ba, như bây giờ bạn có thể dự đoán, chúng tôi đào tạo hệ thống ở lần 1 và 2 và kiểm tra hệ thống ở lần 3. Hình 8-3 minh họa quá trình xác thực chéo này.



Hình 8-3: Trục quan hóa quy trình xác thực chéo mẫu

Thư viện sklearn giúp việc triển khai xác thực chéo trở nên dễ dàng. Làm này, hãy viết lại hàm đánh giá của chúng ta từ Liệt kê 8-15 dưới dạng cv_evaluate.

```
def cv_evaluate(X,y,hasher):
    nhập ngẫu nhiên
    từ số liệu nhập sklearn
    từ matplotlib nhập pyplot
    từ sklearn.cross_validation nhập KFold
```

Chúng ta bắt đầu hàm cv_evaluate() giống như cách chúng ta bắt đầu hàm đánh giá ban đầu, ngoại trừ ở đây chúng ta cũng nhập lớp KFold từ mô-đun cross_validation của sklearn . Xác thực chéo K-fold, hay gọi tắt là KFold , đồng nghĩa với loại xác thực chéo mà tôi vừa thảo luận và là cách phổ biến nhất để thực hiện xác thực chéo.

Tiếp theo, chúng tôi chuyển đổi dữ liệu đào tạo của mình thành các mảng có nhiều mảng để chúng tôi có thể sử dụng lập chỉ mục mảng nâng cao của numpy trên đó:

```
X, y = numpy.array(X), numpy.array(y)
```

Đoạn mã sau thực sự bắt đầu quá trình xác thực chéo:

```
fold_count = 0
đối với đào tạo, kiểm tra trong KFold(len(X),3,ushuffle=True):
```

```
training_X, training_y = X[train], y[train]
test_X, test_y = X[test], y[test]
```

Trước tiên, chúng tôi khởi tạo lớp KFold , chuyển số lượng ví dụ đào tạo mà chúng tôi có làm tham số đầu tiên và số nếp gấp mà chúng tôi muốn sử dụng làm đối số thứ hai. Đối số thứ ba, shuffle=True , yêu cầu sklearn sắp xếp ngẫu nhiên dữ liệu đào tạo của chúng tôi trước khi chia thành ba phần.

Phiên bản KFold thực sự là một trình vòng lặp cung cấp một ví dụ thử nghiệm hoặc đào tạo khác nhau được phân tách trên mỗi lần lặp. Trong vòng lặp for , chúng ta chỉ định các phiên bản đào tạo và phiên bản thử nghiệm cho các mảng training_X và training_y chứa các phần tử tương ứng.

Sau khi chuẩn bị dữ liệu huấn luyện và kiểm tra, chúng ta đã sẵn sàng khởi tạo và huấn luyện RandomForestClassifier , như bạn đã học cách làm trước đây trong chương này:

```
bộ phân loại = RandomForestClassifier()
classifier.fit(đào tạo_X,đào tạo_y)
```

Cuối cùng, chúng tôi tính toán đường cong ROC cho nếp gấp cụ thể này và sau đó vẽ đồ thị đường biểu thị đường cong ROC này:

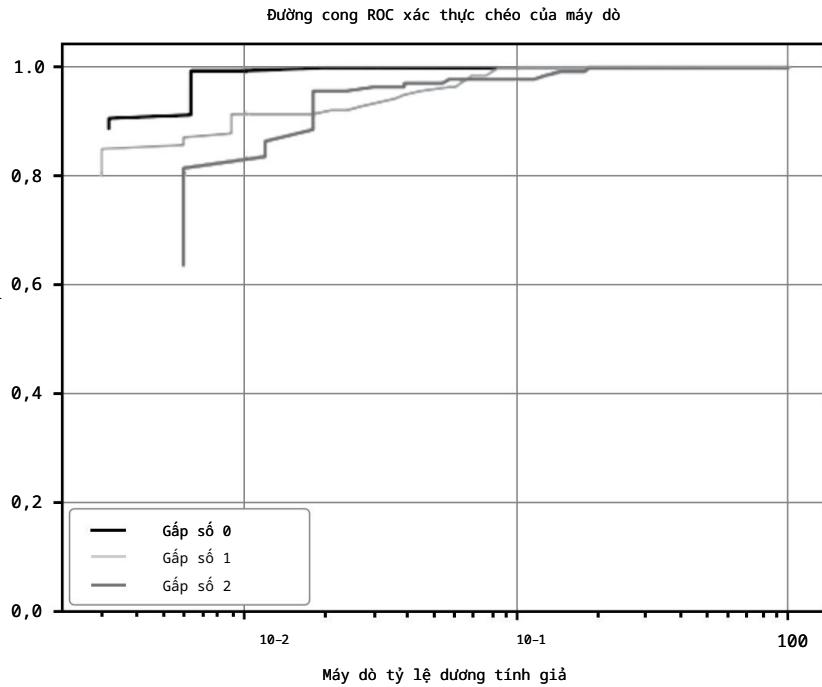
```
diểm = classifier.predict_proba(test_X)[:, -1]
fpr, tpr, ngưỡng = metrics.roc_curve(test_y, điểm số)
pyplot.semilogx(fpr,tpr,label="Fold number {0}".format(fold_counter))
fold_count += 1
```

Lưu ý rằng chúng tôi chưa gọi phương thức hiển thị matplotlib để hiển thị biểu đồ. Chúng tôi làm điều này sau khi tất cả các nếp gấp đã được đánh giá và chúng tôi sẵn sàng hiển thị cả ba dòng cùng một lúc. Như chúng ta đã làm trong phần trước, chúng ta gắn nhãn cho các trục của mình và đặt tiêu đề cho biểu đồ, như sau:

```
pyplot.xlabel("Tỷ lệ dương tính giả của máy dò")
pyplot.ylabel("Tỷ lệ xác thực của máy dò")
pyplot.title("Đường cong ROC xác thực chéo của trình phát hiện")
pyplot.legend()
pyplot.grid()
pyplot.show()
```

Đường cong ROC kết quả được hiển thị trong Hình 8-4.

Như bạn có thể thấy, kết quả của chúng tôi tương tự nhau trên mọi nếp gấp, nhưng chắc chắn có một số khác biệt. Tỷ lệ phát hiện của chúng tôi (tỷ lệ dương tính thực) trong ba lần chạy trung bình khoảng 90 phần trăm với tỷ lệ dương tính giả là 1 phần trăm. Ước tính này, có tính đến cả ba thử nghiệm xác thực chéo, là Ước tính chính xác hơn về hiệu suất của máy dò của chúng tôi so với những gì chúng tôi nhận được nếu chỉ chạy một thử nghiệm trên dữ liệu của mình; trong trường hợp đó, mẫu nào chúng tôi tình cờ sử dụng để đào tạo và thử nghiệm sẽ dẫn đến một kết quả hơi ngẫu nhiên. Bằng cách chạy nhiều thử nghiệm hơn, chúng tôi có thể hiểu rõ hơn về hiệu quả của giải pháp.



Hình 8-4: Vẽ đồ thị đường cong ROC của máy dò bằng xác thực chéo

Lưu ý rằng những kết quả này không tuyệt vời vì chúng tôi đang đào tạo trên một lượng dữ liệu rất nhỏ: vài trăm mẫu phần mềm độc hại và phần mềm độc hại. Tại nơi làm việc hàng ngày của tôi, nơi chúng tôi đào tạo các hệ thống phát hiện phần mềm độc hại máy học quy mô lớn, chúng tôi thường đào tạo trên hàng trăm triệu mẫu. Bạn không cần hàng trăm triệu mẫu để đào tạo trình phát hiện phần mềm độc hại của riêng mình, nhưng bạn sẽ muốn tập hợp các bộ dữ liệu gồm ít nhất hàng chục nghìn mẫu để bắt đầu đạt được hiệu suất thực sự tốt (ví dụ: tỷ lệ phát hiện 90 phần trăm tại tỷ lệ dương tính giả 0,1 phần trăm).

Bước tiếp theo

Cho đến nay, tôi đã trình bày cách sử dụng Python và sklearn để trích xuất các tính năng từ tập dữ liệu đào tạo gồm các tệp nhị phân phần mềm, sau đó đào tạo và đánh giá phương pháp học máy dựa trên cây quyết định. Để cải thiện hệ thống, bạn có thể sử dụng các tính năng khác ngoài hoặc bổ sung cho các tính năng chuỗi có thể in được (ví dụ: tiêu đề PE, hướng dẫn N-gram hoặc các tính năng của Bảng địa chỉ nhập đã thảo luận trước đó) hoặc bạn có thể sử dụng một thuật toán học máy khác.

Để làm cho trình phát hiện chính xác hơn, tôi khuyên bạn nên vượt qua RandomForestClassifier của sklearn (`sklearn.ensemble.RandomForestClassifier`) để thử các bộ phân loại khác. Nhớ lại từ chương trước rằng các máy dò rừng ngẫu nhiên cũng dựa trên cây quyết định, nhưng thay vì chỉ một cây quyết định, chúng xây dựng nhiều cây quyết định, ngẫu nhiên hóa cách chúng được xây dựng. ĐẾN

xác định xem một tệp mới là phần mềm độc hại hay phần mềm độc hại, mỗi cây quyết định này đưa ra các quyết định riêng lẻ, chúng tôi kết hợp chúng bằng cách tổng hợp chúng lại và chia chúng cho tổng số cây để có kết quả trung bình.

Bạn cũng có thể sử dụng các thuật toán khác mà sklearn cung cấp, chẳng hạn như hồi quy logistic. Sử dụng bất kỳ thuật toán nào trong số này có thể đơn giản như thực hiện tìm kiếm và thay thế trong mã mẫu được thảo luận trong chương này. Ví dụ, trong chương này, chúng ta khởi tạo và huấn luyện cây quyết định như sau:

```
bộ phân loại = RandomForestClassifier()
classifier.fit(đào tạo_X, đào tạo_y)
```

Nhưng bạn chỉ có thể thay thế mã đó bằng mã này:

```
từ sklearn.linear_model nhập LogisticRegression
phân loại = LogisticRegression()
classifier.fit(đào tạo_X, đào tạo_y)
```

Sự thay thế này tạo ra một bộ phát hiện hồi quy logistic thay vì một bộ phát hiện dựa trên cây quyết định. Bằng cách tính toán đánh giá dựa trên xác thực chéo mới của trình phát hiện Hồi quy Logistic này và so sánh nó với kết quả từ Hình 8-4, bạn có thể xác định cái nào hoạt động tốt hơn.

Bản tóm tắt

Trong chương này, bạn đã tìm hiểu thông tin chi tiết về việc xây dựng máy học-phát hiện phần mềm độc hại dựa trên. Cụ thể, bạn đã học cách trích xuất các tính năng từ tệp nhị phân phần mềm cho máy học, cách nén các tính năng này bằng thủ thuật băm và cách đào tạo trình phát hiện phần mềm độc hại dựa trên máy học sử dụng các tính năng được trích xuất này. Bạn cũng đã học cách vẽ các đường cong ROC để kiểm tra mối quan hệ giữa ngưỡng phát hiện của máy dò và tỷ lệ dương tính thật và giả của nó. Cuối cùng, bạn đã học về xác thực chéo, khái niệm đánh giá nâng cao hơn và các phần mở rộng khả thi khác để nâng cao trình phát hiện được sử dụng trong chương này.

Điều này kết thúc cuộc thảo luận của cuốn sách này về phát hiện phần mềm độc hại dựa trên máy học bằng cách sử dụng sklearn. Chúng ta sẽ đề cập đến một bộ phương pháp học máy khác, được gọi là phương pháp học sâu hoặc mạng thần kinh nhân tạo trong Chương 10 và 11. Giờ đây, bạn đã có kiến thức cơ bản cần thiết để sử dụng hiệu quả học máy trong bối cảnh nhận dạng phần mềm độc hại.

Tôi khuyến khích bạn đọc thêm về học máy. Bởi vì bảo mật máy tính theo nhiều cách là một vấn đề phân tích dữ liệu, máy học ở đây để tồn tại trong ngành bảo mật và sẽ tiếp tục hữu ích không chỉ trong việc phát hiện các tệp nhị phân độc hại mà còn phát hiện hành vi nguy hiểm trong lưu lượng truy cập mạng, nhật ký hệ thống và các ngữ cảnh khác .

Trong chương tiếp theo, chúng ta sẽ đi sâu vào việc hình dung các mối quan hệ của phần mềm độc hại, điều này có thể giúp chúng ta nhanh chóng hiểu được những điểm tương đồng và khác biệt giữa một số lượng lớn các mẫu phần mềm độc hại.

9

V isualizing Malware T ren ds



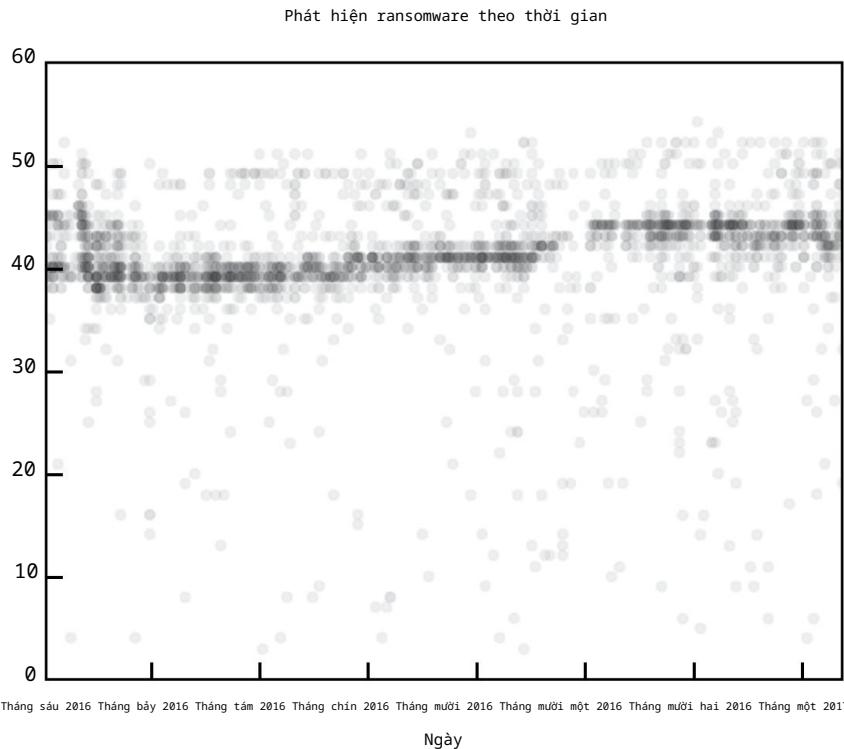
Đôi khi cách tốt nhất để phân tích các bộ sưu tập phần mềm độc hại là trực quan hóa chúng. Trực quan hóa dữ liệu bảo mật cho phép chúng tôi nhanh chóng nhận ra các xu hướng trong phần mềm độc hại và trong phạm vi rộng của mối đe dọa. Những hình ảnh trực quan này thường trực quan hơn nhiều so với số liệu thống kê không trực quan và chúng có thể giúp truyền đạt những hiểu biết sâu sắc cho các đối tượng khác nhau. Ví dụ: trong chương này, bạn sẽ thấy cách trực quan hóa có thể giúp chúng tôi xác định các loại phần mềm độc hại phổ biến trong tập dữ liệu, xu hướng trong tập dữ liệu phần mềm độc hại (ví dụ: sự xuất hiện của ransomware như một xu hướng trong năm 2016) và hiệu quả tương đối của hệ thống chống vi-rút thương mại trong việc phát hiện phần mềm độc hại.

Làm việc qua các ví dụ này, bạn sẽ hiểu cách tạo trực quan hóa của riêng mình để có thể dẫn đến những hiểu biết có giá trị bằng cách sử dụng gói phân tích dữ liệu Python pandas, cũng như gói trực quan hóa dữ liệu Python seaborn và matplotlib. Gói pandas được sử dụng chủ yếu để tải và thao tác dữ liệu và không liên quan nhiều đến bản thân việc trực quan hóa dữ liệu, nhưng nó rất hữu ích để chuẩn bị dữ liệu cho việc trực quan hóa.

Tại sao trực quan hóa dữ liệu phần mềm độc hại lại quan trọng

Để xem việc trực quan hóa dữ liệu phần mềm độc hại có thể hữu ích như thế nào, hãy xem qua hai ví dụ.

Hình dung đầu tiên giải quyết câu hỏi sau: khả năng phát hiện ransomware của ngành công nghiệp chống virus có được cải thiện không? Hình dung thứ hai hỏi loại phần mềm độc hại nào có xu hướng trong khoảng thời gian một năm. Hãy xem ví dụ đầu tiên trong Hình 9-1.



Hình 9-1: Trực quan hóa quá trình phát hiện ransomware theo thời gian

Tôi đã tạo hình ảnh trực quan về mã độc tổng tiền này bằng cách sử dụng dữ liệu được thu thập từ hàng nghìn mẫu phần mềm độc hại mã độc tổng tiền. Dữ liệu này chưa kết quả của việc chạy 57 công cụ chống vi-rút riêng biệt đối với từng tệp. Mỗi đại diện vòng kết nối gửi lại một mẫu phần mềm độc hại. Trực y biểu thị số lần phát hiện hoặc số lần khẳng định, mỗi mẫu phần mềm độc hại nhận được từ công cụ chống vi-rút khi nó được quét. Hãy nhớ rằng mặc dù trực y này dừng ở 60, nhưng số lượng tối đa cho một lần quét nhất định là 57, tổng số máy quét. Phản hồi trực x được gửi khi từng mẫu phần mềm độc hại lần đầu tiên được nhìn thấy trên trang web phân tích phần mềm độc hại VirusTotal.com và được quét.

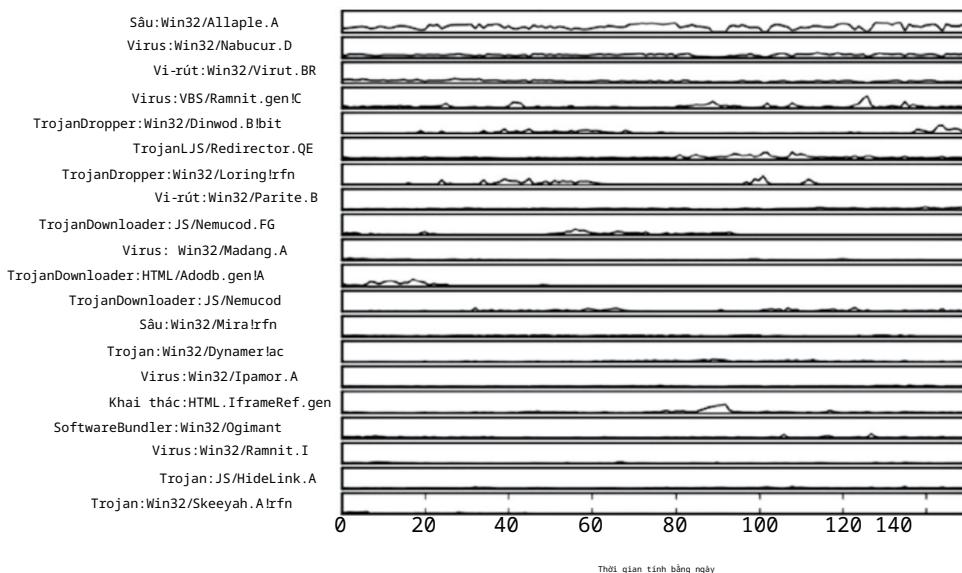
Trong biểu đồ này, chúng ta có thể thấy khả năng của cộng đồng chống vi-rút trong việc phát hiện những tệp độc hại bắt đầu tương đối mạnh vào tháng 6 năm 2016, giảm xuống vào khoảng tháng 7 năm 2016 và sau đó tăng đều trong thời gian còn lại của năm. Đến cuối năm 2016,

trung bình khoảng 25 phần trăm các công cụ chống vi-rút vẫn bỏ sót các tệp ransomware, vì vậy chúng tôi có thể kết luận rằng cộng đồng bảo mật vẫn hơi yếu trong việc phát hiện các tệp này trong thời gian này.

Để mở rộng cuộc điều tra này, bạn có thể tạo một hình ảnh trực quan cho biết công cụ chống vi-rút nào đang phát hiện phần mềm tống tiền và ở tốc độ nào cũng như cách chúng đang cải thiện theo thời gian. Hoặc bạn có thể xem xét một số danh mục phần mềm độc hại khác (ví dụ: ngựa thành Troy). Những sơ đồ như vậy rất hữu ích trong việc quyết định mua công cụ chống vi-rút nào hoặc quyết định loại phần mềm độc hại nào bạn có thể muốn thiết kế các giải pháp phát hiện tùy chỉnh cho-có thể là để bổ sung cho hệ thống phát hiện chống vi-rút thương mại (để biết thêm về cách xây dựng hệ thống phát hiện tùy chỉnh, xem Chương 8).

Bây giờ hãy xem Hình 9-2, đây là một hình ảnh mẫu khác, được tạo bằng cách sử dụng cùng bộ dữ liệu được sử dụng cho Hình 9-1.

Gia đình phần mềm độc hại phổ biến hơn 150 ngày



Hình 9-2: Trục quan hóa quá trình phát hiện phần mềm độc hại theo nhóm theo thời gian

Hình 9-2 cho thấy 20 họ phần mềm độc hại phổ biến nhất và tần suất chúng xuất hiện so với nhau trong khoảng thời gian 150 ngày. Cốt truyện tiết lộ một số thông tin chi tiết quan trọng: trong khi họ phần mềm độc hại phổ biến nhất, Allapple.A, xuất hiện liên tục trong khoảng thời gian 150 ngày, thì các phần mềm độc hại khác thuộc họ đối trá, như Nemucod.FG, phổ biến trong khoảng thời gian ngắn hơn và sau đó im lặng. Một ám mưu như thế này, được tạo bằng cách sử dụng phần mềm độc hại được phát hiện trên mạng tại nơi làm việc của chính bạn, có thể tiết lộ các xu hướng hữu ích cho biết loại phần mềm độc hại nào có liên quan đến các cuộc tấn công chống lại tổ chức của bạn theo thời gian. Nếu không tạo ra một số liệu so sánh như số liệu này, việc hiểu và so sánh các định và khối lượng tương đối của các loại phần mềm độc hại này theo thời gian sẽ rất khó khăn và tốn thời gian.

Hai ví dụ này cho thấy khả năng trực quan hóa phần mềm độc hại hữu ích như thế nào. Các phần còn lại của chương này cho thấy cách tạo trực quan hóa của riêng bạn. Chúng tôi bắt đầu bằng cách thảo luận về bộ dữ liệu mẫu được sử dụng trong chương này và sau đó chúng tôi sử dụng gói gấu trúc để phân tích dữ liệu. Cuối cùng, chúng tôi sử dụng matplotlib và seaborn gói để trực quan hóa dữ liệu.

Hiểu tập dữ liệu phần mềm độc hại của chúng tôi

Bộ dữ liệu chúng tôi sử dụng chứa dữ liệu mô tả 37.000 mã nhí phần mềm độc hại duy nhất được thu thập bởi VirusTotal, một dịch vụ tổng hợp phát hiện phần mềm độc hại. Mỗi tệp nhị phân được gắn nhãn với bốn trường: số lượng công cụ chống vi-rút (trong số 57) đã gắn cờ tệp nhị phân là độc hại (tôi gọi đây là số lượng kết quả tích cực được liên kết với mỗi mẫu), kích thước của mỗi tệp nhị phân, loại tệp nhị phân (công cụ khai thác bitcoin, keylogger, ransomware, trojan hoặc worm) và ngày mà tệp nhị phân được nhìn thấy lần đầu tiên. Chúng ta sẽ thấy rằng ngay cả với số lượng siêu dữ liệu khá hạn chế này cho mỗi nhị phân, chúng ta vẫn có thể phân tích và trực quan hóa dữ liệu theo cách tiết lộ những hiểu biết quan trọng về tập dữ liệu.

Đang tải dữ liệu vào gấu trúc

Thư viện phân tích dữ liệu phổ biến của Python pandas giúp dễ dàng tải dữ liệu vào các đối tượng phân tích được gọi là DataFrames, sau đó cung cấp các phương thức để cắt, chuyển đổi và phân tích dữ liệu được đóng gói lại đó. Chúng tôi sử dụng gấu trúc để tải và phân tích dữ liệu của mình cũng như chuẩn bị dữ liệu để dễ hình dung. Hãy sử dụng Liệt kê 9-1 để xác định và tải một số dữ liệu mẫu vào trình thông dịch Python.

Trong [135]: nhập gấu trúc

```
Trong [135]: example_data = [{u'column1': 1, 'column2': 2},
...: {'cột1': 10, 'cột2': 32},
...: {'cột1': 3, 'cột2': 58}]
```

Trong [137]: vpandas.DataFrame(example_data)

```
Ra[137]:
    cột1   cột2
0        1       2
1      10      32
2        3      58
```

Liệt kê 9-1: Tải trực tiếp dữ liệu vào gấu trúc

Ở đây chúng tôi định nghĩa một số dữ liệu mà chúng tôi gọi là example_data, dưới dạng danh sách các từ điển Python u. Khi chúng tôi đã tạo danh sách các ký tự này, chúng tôi chuyển nó tới hàm tạo DataFrame v để lấy DataFrame gấu trúc tương ứng. Mỗi ký tự này trở thành một hàng trong DataFrame kết quả . Các khóa trong ký tự (cột1 và cột2) trở thành cột. Đây là một cách để tải dữ liệu trực tiếp vào gấu trúc .

Bạn cũng có thể tải dữ liệu từ tệp CSV bên ngoài. Hãy sử dụng mã trong Liệt kê 9-2 để tải bộ dữ liệu của chương này (có sẵn trên máy ảo hoặc trong kho lưu trữ dữ liệu và mã đi kèm với cuốn sách này).

```
gầu trúc nhập khẩu
phần mềm độc hại = pandas.read_csv("malware_data.csv")
```

Liệt kê 9-2: Đang tải dữ liệu vào gấu trúc từ tệp CSV bên ngoài

Khi bạn nhập malware_data.csv, đối tượng phần mềm độc hại thu được sẽ nhìn một cái gì đó như thế này:

	tích cực 45	kích cỡ	loại	fs_bucket
0		32	trojan 2017-01-05 00:00:00	
1		227048	trojan 2016-06-30 00:00:00	
2		53	sâu 2016-07-30 00:00:00	
3		39	trojan 2016-06-29 00:00:00	
4		29 571904 31	trojan 2016-12-24 00:00:00	
		582352 50 2031661	trojan 2016-09-23 00:00:00	
5 6			sâu 2017-01-04 00:00:00	

Bây giờ chúng tôi có một DataFrame gấu trúc bao gồm bộ dữ liệu phần mềm độc hại của chúng tôi. Nó có bốn cột: tích cực (số lần phát hiện phần mềm chống vi-rút trong số 57 công cụ chống vi-rút cho mẫu đó), kích thước (số byte mà nhị phân phần mềm độc hại chiếm trên đĩa), loại (loại phần mềm độc hại, chẳng hạn như ngựa thành Troy, sâu, v.v.) và fs_bucket (ngày mà phần mềm độc hại này được nhìn thấy lần đầu tiên).

Làm việc với DataFrame gấu trúc

Bây giờ chúng ta đã có dữ liệu của mình trong DataFrame của gấu trúc, hãy xem cách truy cập và thao tác với nó bằng cách gọi phương thức `description()`, như trong Liệt kê 9-3.

Trong [51]: `phần mềm độc hại.describe()`

Hết[51]:

	số lượng	kích cỡ
tích cực	37511.000000	3.751100e+04
có nghĩa là	39.446536	1.300639e+06
tiêu	15.039759	3.0006031e+06
chuẩn xác chuẩn	3.000000	3.370000e+02
25%	32.000000	1.653960e+05
50%	45.000000	4.828160e+05
75%	51.000000	1.290056e+06
tối đa	57.000000	1.294244e+08

Liệt kê 9-3: Gọi phương thức `description()`

Như được hiển thị trong Liệt kê 9-3, gọi phương thức `description()` sẽ hiển thị một số thống kê hữu ích về DataFrame của chúng ta. Dòng đầu tiên, dém, dém tổng số hàng dương không null và tổng số hàng không null.

Dòng thứ hai đưa ra giá trị trung bình hoặc số lượng dương tính trung bình trên mỗi mẫu và kích thước trung bình của các mẫu phần mềm độc hại. Tiếp theo là độ lệch chuẩn cho cả giá trị dương và kích thước cũng như giá trị tối thiểu của mỗi cột trong tất cả các mẫu trong tập dữ liệu. Cuối cùng, chúng ta thấy các giá trị phần trăm cho từng cột và giá trị tối đa cho các cột.

Giả sử chúng tôi muốn truy xuất dữ liệu cho một trong các cột trong DataFrame của phần mềm độc hại, chẳng hạn như cột khẳng định (ví dụ: để xem số lần phát hiện trung bình mà mỗi tệp có hoặc vẽ biểu đồ biểu đồ hiển thị phân phối của các tích cực trên tập dữ liệu). Để làm điều này, chúng ta chỉ cần viết malware['positives'], trả về cột positives dưới dạng một danh sách các số, như trong Liệt kê 9-4.

Trong [3]: phần mềm độc hại['tích cực']

Hết[3]:

0	45
1	32
2	53
	39
3 4	29
	31
5 6	50
	40
7 8	20
	40
9 --snip--	

Liệt kê 9-4: Trả về cột dương

Sau khi truy xuất một cột, chúng ta có thể tính toán trực tiếp số liệu thống kê về cột đó. Ví dụ: malware['positives'].mean() tính giá trị trung bình của cột, malware['positives'].max() tính giá trị tối đa, malware['positives'].min() tính toán giá trị tối thiểu và phần mềm độc hại['positives'].std() tính toán độ lệch chuẩn. Liệt kê 9-5 cho thấy các ví dụ của từng loại.

Trong [7]: phần mềm độc hại['positives'].mean()

Ra[7]: 39.446535682866362

Trong [8]: phần mềm độc hại['positives'].max()

Hết[8]: 57

Trong [9]: phần mềm độc hại['positives'].min()

Hết[9]: 3

Trong [10]: phần mềm độc hại['positives'].std()

Ra[10]: 15.039759380778822

Liệt kê 9-5: Tính giá trị trung bình, giá trị lớn nhất và giá trị nhỏ nhất và độ lệch chuẩn

Chúng tôi cũng có thể chia nhỏ dữ liệu để phân tích chi tiết hơn. Ví dụ: Liệt kê 9-6 tính toán giá trị dương trung bình cho các loại phần mềm độc hại trojan, bitcoin và worm.

Trong [67]: malware[malware['type'] == 'trojan']['positives'].mean()

Ra[67]: 33.43822473365119

Trong [68]: phần mềm độc hại[phần mềm độc hại['type'] == 'bitcoin']['positives'].mean()

Hết[68]: 35.857142857142854

Trong [69]: phần mềm độc hại[phần mềm độc hại['type'] == 'sâu']['tích cực'].mean()
Ra[69]: 49.90857904874796

Liệt kê 9-6: Tính toán tỷ lệ phát hiện trung bình của các phần mềm độc hại khác nhau

Trước tiên, chúng tôi chọn các hàng của DataFrame nơi loại được đặt thành trojan bằng cách sử dụng ký hiệu sau: phần mềm độc hại [phần mềm độc hại ['type'] == 'trojan']. Để chọn cột khẳng định của dữ liệu kết quả và tính giá trị trung bình, chúng tôi mở rộng biểu thức này như sau: malware[malware['type'] == 'trojan']['positives'] .nghĩa là(). Liệt kê 9-6 mang lại một kết quả thú vị, đó là sâu máy tính được phát hiện thường xuyên hơn khai thác bitcoin và phần mềm độc hại ngựa thành Troy. Bởi vì trung bình $49,9 > 35,8$ và $33,4$, các mẫu sâu độc hại ($49,9$) được nhiều nhà cung cấp phát hiện hơn các mẫu bitcoin và trojan độc hại ($35,8, 33,4$).

Lọc dữ liệu bằng điều kiện

Chúng tôi cũng có thể chọn một tập hợp con của dữ liệu bằng các điều kiện khác. Ví dụ: chúng ta có thể sử dụng các điều kiện kiểu "lớn hơn" và "nhỏ hơn" đối với dữ liệu số như kích thước tệp phần mềm độc hại để lọc dữ liệu, sau đó tính toán số liệu thống kê trên các tập con kết quả. Điều này có thể hữu ích nếu chúng tôi quan tâm đến việc tìm hiểu xem hiệu quả của các công cụ chống vi-rút có liên quan đến kích thước tệp hay không. Chúng ta có thể kiểm tra điều này bằng cách sử dụng mã trong

Liệt kê 9-7.

Trong [84]: phần mềm độc hại[phần mềm độc hại['size'] > 1000000]['positives'].mean()
Ra[84]: 33.507073192162373

Trong [85]: phần mềm độc hại[phần mềm độc hại['size'] > 2000000]['positives'].mean()
Ra[85]: 32.76142050415432

Trong [86]: phần mềm độc hại[phần mềm độc hại['size'] > 3000000]['positives'].mean()
Hết[86]: 27.20672682526661

Trong [87]: phần mềm độc hại[phần mềm độc hại['size'] > 4000000]['positives'].mean()
Ra[87]: 25.652548725637182

Trong [88]: phần mềm độc hại[phần mềm độc hại['size'] > 5000000]['positives'].mean()
Ra[88]: 24.411069317571197

Liệt kê 9-7: Lọc kết quả theo kích thước tệp phần mềm độc hại

Lấy dòng đầu tiên trong mã trước: đầu tiên, chúng tôi tạo hợp DataFrame của chúng tôi chỉ bởi các mẫu có kích thước trên một triệu (phần mềm độc hại [phần mềm độc hại ['size'] > 1000000]). Sau đó, chúng tôi lấy cột dương và tính giá trị trung bình ([positives].mean()), là khoảng 33,5. Khi chúng tôi thực hiện điều này đối với các kích thước tệp ngày càng cao, chúng tôi thấy rằng số lần phát hiện trung bình cho mỗi nhóm giảm xuống. Điều này có nghĩa là chúng tôi đã phát hiện ra rằng thực sự có mối quan hệ giữa kích thước tệp phần mềm độc hại và số lượng trung bình các công cụ chống vi-rút phát hiện các mẫu phần mềm độc hại đó, điều này rất thú vị và đáng để điều tra thêm. Chúng tôi khám phá điều này một cách trực quan tiếp theo bằng cách sử dụng matplotlib và

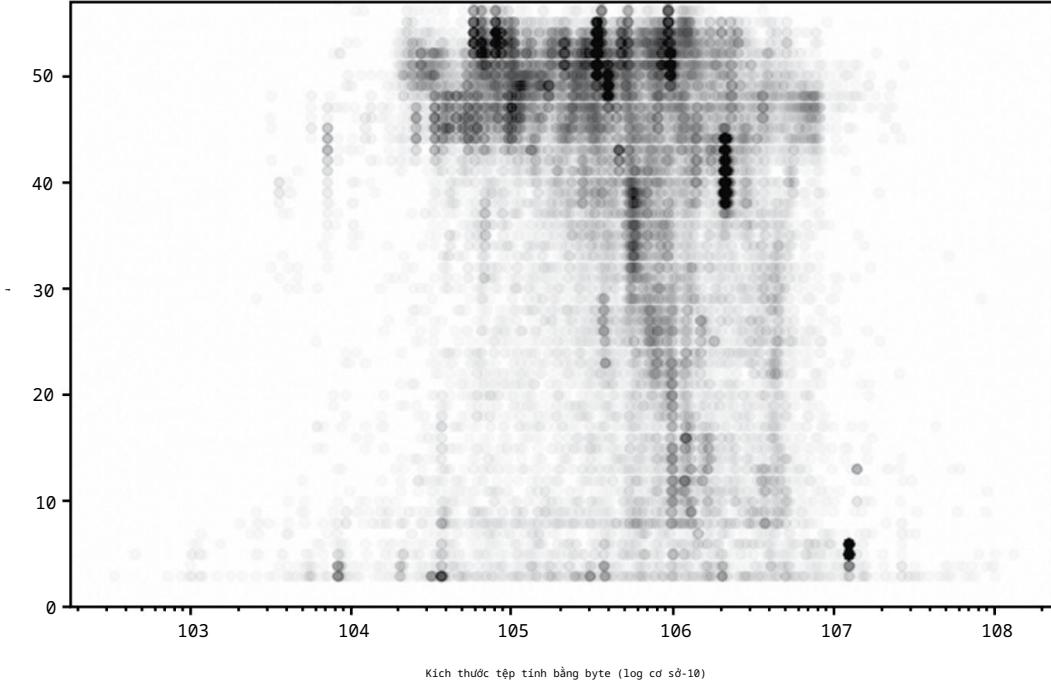
sinh ra biến.

Sử dụng matplotlib để trực quan hóa dữ liệu

Thư viện truy cập để trực quan hóa dữ liệu Python là matplotlib; trên thực tế, hầu hết các thư viện trực quan hóa Python khác về cơ bản là các trình bao bọc tiện lợi xung quanh matplotlib. Thật dễ dàng để sử dụng matplotlib với giao trúc: chúng tôi sử dụng giao trúc để lấy, cắt và chia dữ liệu mà chúng tôi muốn vẽ và chúng tôi sử dụng matplotlib để vẽ đồ thị đó. Chức năng matplotlib hữu ích nhất cho mục đích của chúng tôi là chức năng cốt truyện.

Hình 9-3 cho thấy chức năng vẽ đồ thị có thể làm gì.

Số lần phát hiện phần mềm chống vi-rút so với kích thước tệp



Hình 9-3: Biểu đồ kích thước mẫu phần mềm độc hại và số lần phát hiện phần mềm chống vi-rút

Ở đây, tôi vẽ các thuộc tính tích cực và kích thước của bộ dữ liệu phần mềm độc hại của chúng tôi. MỘT kết quả thú vị xuất hiện, như được báo trước bởi cuộc thảo luận của chúng ta về giao trúc trong phần trước. Nó cho thấy rằng các tệp nhỏ và tệp rất lớn hiếm khi được phát hiện bởi hầu hết 57 công cụ chống vi-rút đã quét các tệp này. Tuy nhiên, các tệp có kích thước trung bình (khoảng 104,5-107) được hầu hết các công cụ phát hiện.

Điều này có thể là do các tệp nhỏ không chứa đủ thông tin để cho phép các công cụ xác định chúng là độc hại và các tệp lớn quá chậm để quét, khiến nhiều hệ thống chống vi-rút hoàn toàn không thể quét chúng.

Vẽ sơ đồ mối quan hệ giữa kích thước phần mềm độc hại và khả năng phát hiện của nhà cung cấp. Hãy xem qua cách tạo biểu đồ như trong Hình 9-3 bằng cách sử dụng mã trong Liệt kê 9-8.

```
bạn nhập gấu trúc
    từ matplotlib nhập pyplot
    phần mềm đọc hại = vpandas.read_csv("malware_data.csv")
    pyplot.plot(wmalware['size'], xmalware['positives'],
                y'bo', zalpha=0.01)
    pyplot.xscale({"log"})
    | pyplot.ylim([0,57])
    pyplot.xlabel("Kích thước tệp tính bằng byte (log base-10)")
    pyplot.ylabel("Số lần phát hiện")
    pyplot.title("Số lần phát hiện phần mềm chống vi-rút so với kích thước tệp")
} pyplot.show()
```

Liệt kê 9-8: Trực quan hóa dữ liệu bằng hàm plot()

Như bạn có thể thấy, không cần nhiều mã để hiển thị biểu đồ này. chúng ta hãy đi bộ thông qua những gì mỗi dòng làm. Đầu tiên, chúng tôi nhập vào u các thư viện cần thiết, bao gồm gấu trúc và mô-đun pyplot của thư viện matplotlib . Sau đó, chúng tôi gọi hàm read_csv v, như bạn đã học trước đó, tải tập dữ liệu phần mềm độc hại của chúng tôi vào DataFrame của gấu trúc.

Tiếp theo, chúng ta gọi hàm plot() . Đôi số đầu tiên của hàm là dữ liệu kích thước phần mềm độc hại w và đối số tiếp theo là phần mềm độc hại tích cực dữ liệu x hoặc số lần phát hiện tích cực cho từng mẫu phần mềm độc hại. Các đối số này xác định dữ liệu mà matplotlib sẽ vẽ, với đối số đầu tiên biểu thị dữ liệu sẽ được hiển thị trên trục x và đối số thứ hai biểu thị dữ liệu sẽ được hiển thị trên trục y. Đôi số tiếp theo, 'bo' y, cho matplotlib biết màu sắc và hình dạng sẽ sử dụng để thể hiện dữ liệu.

Cuối cùng, chúng tôi đặt alpha hoặc độ trong suốt của các vòng tròn thành 0,1 z, vì vậy chúng tôi có thể thấy mức độ dày đặc của dữ liệu trong các vùng khác nhau của biểu đồ, ngay cả khi các vòng tròn hoàn toàn chồng lênh nhau.

LƯU Ý Chữ b trong bo là viết tắt của màu xanh lam và chữ o là viết tắt của hình tròn, nghĩa là chúng ta đang nói ing matplotlib để vẽ các vòng tròn màu xanh để thể hiện dữ liệu của chúng tôi. Các màu khác bạn có thể thử là xanh lục (g), đỏ (r), lục lam (c), đỏ tươi (m), vàng (y), đen (k) và trắng (w). Các hình dạng khác mà bạn có thể thử là một điểm (.), một pixel trên mỗi điểm dữ liệu (,), (các) hình vuông và một hình ngũ giác (p). Để biết chi tiết đầy đủ, hãy xem tài liệu matplotlib tại <http://matplotlib.org>.

Sau khi chúng ta gọi hàm plot() , chúng ta đặt tỷ lệ của trục x là logarit {. Điều này có nghĩa là chúng ta sẽ xem dữ liệu kích thước phần mềm độc hại theo lũy thừa 10, giúp dễ dàng xem mối quan hệ giữa các tệp rất nhỏ và rất lớn.

Bây giờ chúng tôi đã vẽ biểu đồ dữ liệu của mình, chúng tôi gắn nhãn các trục và đặt tên cho biểu đồ của mình. Trục x biểu thị kích thước của tệp phần mềm độc hại ("Kích thước tệp tính bằng byte (cơ sở nhật ký-10)") và trục y biểu thị số lần phát hiện ("Số lần phát hiện"). Vì có 57 công cụ chống vi-rút mà chúng tôi đang phân tích nên chúng tôi đặt tỷ lệ trục y thành phạm vi từ 0 đến 57 tại |. Cuối cùng, chúng ta gọi hàm show() để hiển thị cốt truyện. Chúng tôi có thể thay thế cuộc gọi này bằng pyplot.savefig("myplot.png") thay vào đó, nếu chúng tôi muốn lưu cốt truyện dưới dạng hình ảnh.

Bây giờ chúng ta đã trải qua một ví dụ ban đầu, hãy làm một ví dụ khác.

Biểu đồ tỷ lệ phát hiện ransomware

Lần này, hãy thử tái tạo Hình 9-1, biểu đồ phát hiện mã độc tống tiền mà tôi đã trình bày ở đầu chương này. Liệt kê 9-9 trình bày toàn bộ mã biểu thị quá trình phát hiện mã độc tống tiền của chúng ta theo thời gian.

```

nhập dateutil
gõa trúc nhập khẩu
từ matplotlib nhập pyplot

phần mềm độc hại = pandas.read_csv("malware_data.csv")
phần mềm độc hại['fs_date'] = [dateutil.parser.parse(d) for d trong phần mềm độc hại['fs_bucket']]
ransomware = malware[malware['type'] == 'ransomware']
pyplot.plot(ransomware['fs_date'], ransomware['positives'], 'ro', alpha=0,05)
pyplot.title("Phát hiện Ransomware theo thời gian")
pyplot.xlabel("Ngày")
pyplot.ylabel("Số lần phát hiện công cụ chống vi-rút")
pyplot.show()

```

Liệt kê 9-9: Biểu đồ tỷ lệ phát hiện ransomware theo thời gian

Một số mã trong Liệt kê 9-9 sẽ quen thuộc với những gì tôi đã giải thích cho đến nay, và một số thì không. Hãy xem qua mã, từng dòng một:

```

nhập dateutil

```

Gói dateutil hữu ích của Python cho phép bạn dễ dàng phân tích cú pháp ngày từ nhiều định dạng khác nhau. Chúng tôi nhập dateutil vì chúng tôi sẽ phân tích ngày tháng để có thể hình dung chúng.

```

gõa trúc nhập khẩu
từ matplotlib nhập pyplot

```

Chúng tôi cũng nhập mô-đun pyplot của thư viện matplotlib cũng như gõa trúc.

```

phần mềm độc hại = pandas.read_csv("malware_data.csv")
phần mềm độc hại['fs_date'] = [dateutil.parser.parse(d) for d trong phần mềm độc hại['fs_bucket']]
ransomware = malware[malware['type'] == 'ransomware']

```

Những dòng này đọc trong tập dữ liệu của chúng tôi và tạo tập dữ liệu được lọc có tên ransomware chỉ chứa các mẫu ransomware, bởi vì đó là loại dữ liệu mà chúng tôi quan tâm trong việc vẽ sơ đồ ở đây.

```

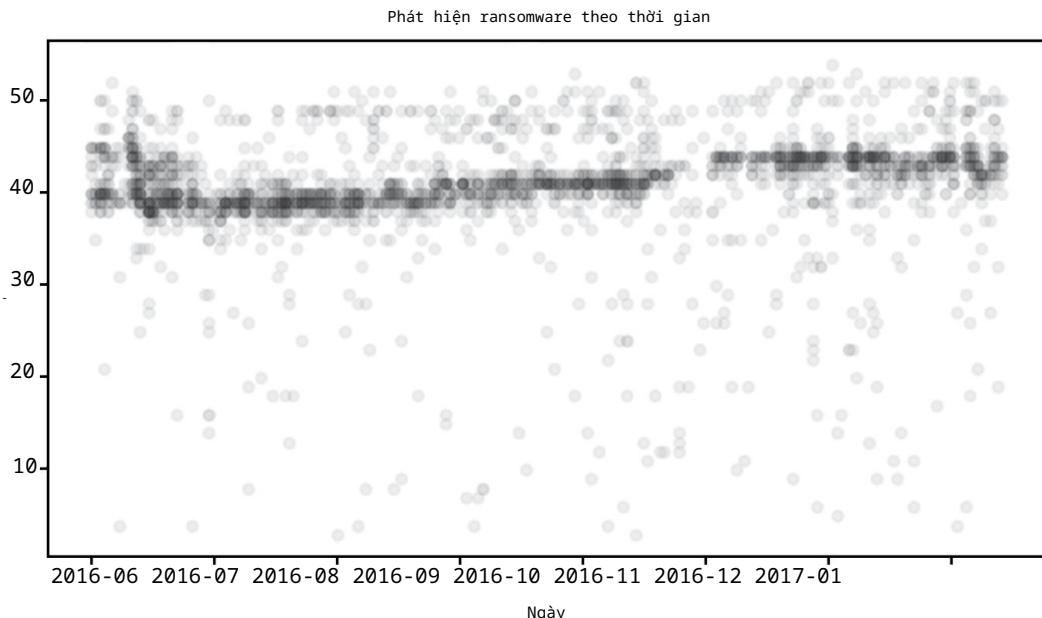
pyplot.plot(ransomware['fs_date'], ransomware['positives'], 'ro', alpha=0,05)
pyplot.title("Phát hiện Ransomware theo thời gian")
pyplot.xlabel("Ngày")

```

```
pyplot.ylabel("Số lần phát hiện công cụ chống vi-rút")
pyplot.show()
```

Năm dòng mã này phản ánh mã trong Liệt kê 9-8: chúng vẽ biểu đồ dữ liệu, đặt tên cho biểu đồ, gắn nhãn cho các trục x và y của nó, sau đó hiển thị mọi thứ lên màn hình (xem Hình 9-4).

Một lần nữa, nếu chúng ta muốn lưu biểu đồ này vào đĩa, chúng ta có thể thay thế lệnh gọi `pyplot.show()` bằng `pyplot.savefig("myplot.png")`.



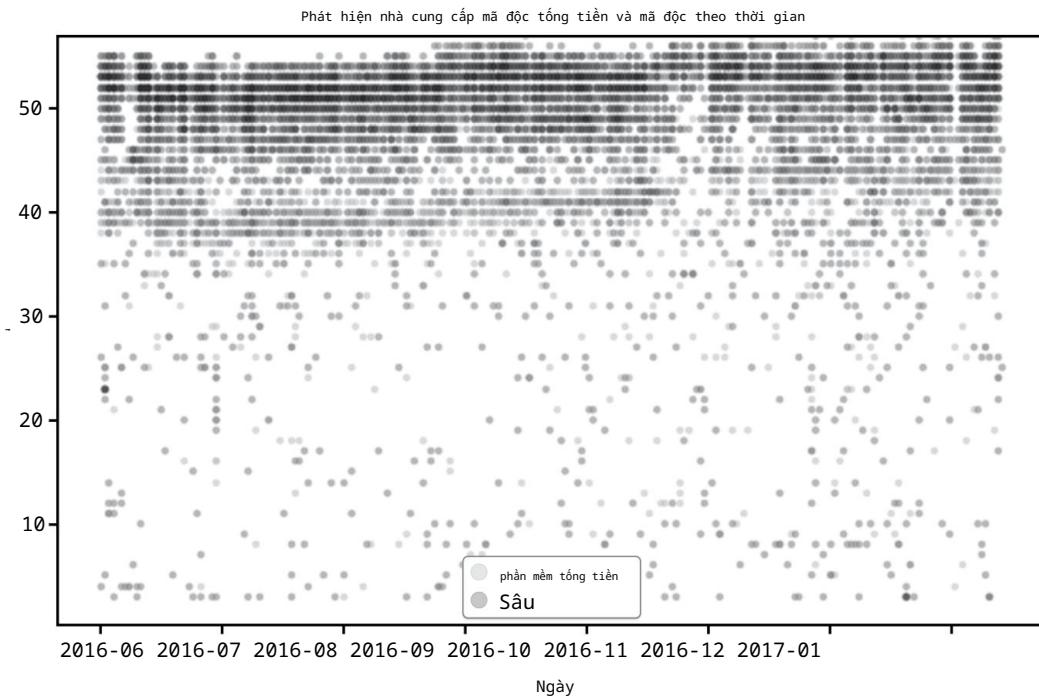
Hình 9-4: Trực quan hóa quá trình phát hiện ransomware theo thời gian

Hãy thử thêm một biểu đồ bằng cách sử dụng hàm `plot()` .

Vẽ biểu đồ Ransomware và tỷ lệ phát hiện sâu

Lần này, thay vì chỉ vẽ đồ thị phát hiện ransomware theo thời gian, chúng ta hãy vẽ đồ thị phát hiện sâu trong cùng một biểu đồ. Điều trở nên rõ ràng trong Hình 9-5 là ngành công nghiệp chống vi-rút phát hiện sâu tốt hơn (một xu hướng phần mềm độc hại cũ hơn) so với ransomware (một xu hướng phần mềm độc hại mới hơn).

Trong biểu đồ này, chúng tôi thấy có bao nhiêu công cụ chống vi-rút đã phát hiện ra các mẫu phần mềm độc hại (trục y) theo thời gian (trục x). Mỗi chấm đỏ đại diện cho một mẫu phần mềm độc hại `type="ransomware"` , trong khi mỗi chấm xanh đại diện cho một mẫu `type="sâu"` . Chúng ta có thể thấy rằng trung bình, nhiều công cụ phát hiện mẫu sâu hơn mẫu ransomware. Tuy nhiên, số lượng động cơ phát hiện cả hai mẫu đang có xu hướng tăng dần theo thời gian.



Hình 9-5: Trục quan hóa quá trình phát hiện phản mềm độc hại ransomware và worm theo thời gian

Liệt kê 9-10 cho thấy mã để tạo biểu đồ này.

```

nhập dateutil nhập
gõi trúc từ
matplotlib nhập pyplot

phản mềm độc hại = pandas.read_csv("malware_data.csv") phản
mềm độc hại['fs_date'] = [dateutil.parser.parse(d) for d trong phản mềm độc hại['fs_bucket']]

ransomware = malware[malware['type'] == 'ransomware'] worm =
malware[malware['type'] == 'sâu']

pyplot.plot(ransomware['fs_date'], ransomware['positives'],
            'ro', label="Ransomware", markersize=3, alpha=0.05)
pyplot.plot(worms['fs_date'], worm['positives'], 'bo',
            label="Worm", markersize=3, alpha=0.05)
pyplot.legend(framealpha=1, markerscale=3.0)
pyplot.xlabel("Ngày")
pyplot.ylabel("Số lần phát hiện") pyplot.ylim([0,
57]) pyplot.title("Ransomware
và phát hiện nhà cung cấp sâu theo thời gian") pyplot.show()

```

Liệt kê 9-10: Vẽ biểu đồ tốc độ phát hiện sâu và mã độc tổng tiền theo thời gian

Hãy xem qua đoạn mã này bằng cách xem phần đầu tiên của Liệt kê 9-10:

```

nhập dateutil
gửi trúc nhập khẩu
từ matplotlib nhập pyplot

phần mềm độc hại = pandas.read_csv("malware_data.csv")
phần mềm độc hại['fs_date'] = [dateutil.parser.parse(d) for d trong phần mềm độc hại['fs_bucket']]

ransomware = malware[malware['type'] == 'ransomware']
sâu u = phần mềm độc hại [phần mềm độc hại ['loại'] == "sâu"]
--snip--

```

Mã này tương tự như ví dụ trước. Sự khác biệt cho đến nay là chúng tôi tạo phiên bản đã lọc sâu của dữ liệu u bằng cách sử dụng cùng một phương pháp mà chúng tôi tạo dữ liệu đã lọc ransomware . Bây giờ chúng ta hãy xem phần còn lại của mã:

```

--snip--
u pyplot.plot(ransomware['fs_date'], ransomware['positives'],
              'ro', nhánh="Ransomware", markersize=3, alpha=0.05)
v pyplot.plot(sâu['fs_bucket'], worm['positives'],
              'bo', nhánh="Sâu", kích thước đánh dấu=3, alpha=0.05)
w pyplot.legend(framealpha=1, markerscale=3.0)
pyplot.xlabel("Ngày")
pyplot.ylabel("Số lần phát hiện")
pyplot.ylim([0,57])
pyplot.title("Phát hiện nhà cung cấp Ransomware và Worm theo thời gian")
pyplot.show()
pyplot(gcf()).clf()

```

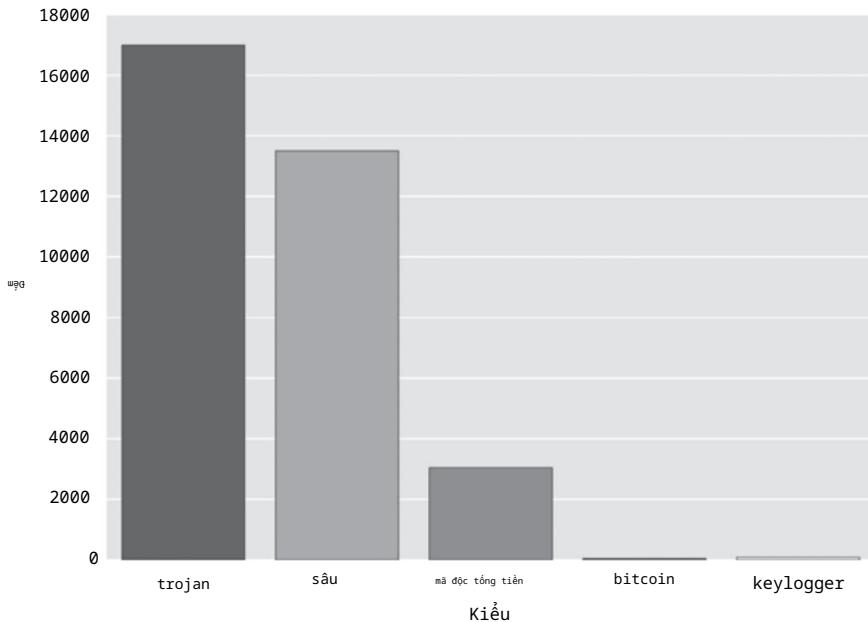
Sự khác biệt chính giữa mã này và Liệt kê 9-9 là chúng ta gọi hàm plot() hai lần: một lần cho dữ liệu ransomware bằng cách sử dụng bộ chọn ro u để tạo các vòng tròn màu đỏ và một lần nữa cho dữ liệu sâu máy tính bằng cách sử dụng bo bộ chọn v để tạo các vòng tròn màu xanh cho dữ liệu sâu. Lưu ý rằng nếu chúng tôi muốn vẽ tập dữ liệu thứ ba, chúng tôi cũng có thể làm điều này. Ngoài ra, không giống như Liệt kê 9-9, ở đây, tại w, chúng ta tạo chú thích cho hình của mình cho thấy rằng các dấu màu xanh đại diện cho phần mềm độc hại sâu và các dấu màu đỏ đại diện cho phần mềm tổng tiền. Tham số framealpha xác định độ trong mờ của nền của chú giải (bằng cách đặt nó thành 1, chúng tôi làm cho nó mờ hoàn toàn) và tham số markerscale chia tỷ lệ kích thước của các điểm đánh dấu trong chú giải (trong trường hợp này, theo hệ số ba).

Trong phần này, bạn đã học cách tạo một số biểu đồ đơn giản trong matplotlib. Tuy nhiên, hãy thành thật mà nói - chúng không đẹp. Trong phần tiếp theo, chúng ta sẽ sử dụng một thư viện vẽ biểu đồ khác sẽ cho phép chúng ta tạo cho các biểu đồ của mình một giao diện chuyên nghiệp hơn và giúp chúng ta thực hiện các hình ảnh hóa phức tạp hơn một cách nhanh chóng.

Sử dụng seaborn để trực quan hóa dữ liệu

Bây giờ chúng ta đã thảo luận về pandas và matplotlib, hãy chuyển sang seaborn, đây là một thư viện trực quan thực sự được xây dựng trên matplotlib nhưng được gói gọn trong một thùng chứa bóng bẩy. Nó bao gồm các chủ đề tích hợp sẵn để tạo kiểu cho đồ họa của chúng tôi cũng như các chức năng cấp cao hơn được tạo sẵn giúp tiết kiệm thời gian thực hiện các phân tích phức tạp hơn. Những tính năng này làm cho nó trở nên đơn giản và dễ dàng để tạo ra những ô đẹp, phức tạp.

Để khám phá seaborn, hãy bắt đầu bằng cách tạo biểu đồ thanh cho biết có bao nhiêu ví dụ về từng loại phần mềm độc hại mà chúng tôi có trong tập dữ liệu của mình (xem Hình 9-6).



Hình 9-6: Sơ đồ thanh của các loại phần mềm độc hại khác nhau trong tập dữ liệu của chương này

Liệt kê 9-11 cho thấy đoạn mã để tạo biểu đồ này.

```
gõ vào trống nhập khẩu
từ matplotlib nhập pyplot
hải sản nhập khẩu
```

```
u phần mềm độc hại = pandas.read_csv("malware_data.csv")
v seaborn.countplot(x='type', data=malware)
w pyplot.show()
```

Liệt kê 9-11: Tạo biểu đồ thanh số lượng phần mềm độc hại theo loại

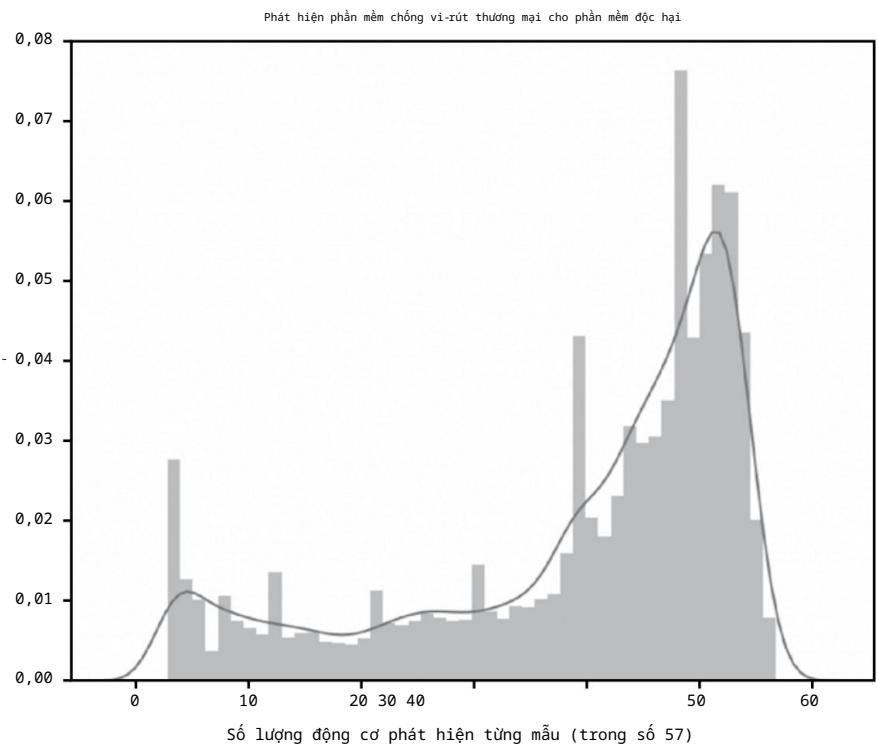
Trong mã này, trước tiên chúng tôi đọc dữ liệu của mình qua pandas.read_csv u và sau đó sử dụng hàm đếm của seaborn để tạo một biểu đồ thanh của cột loại trong DataFrame v. Cuối cùng, chúng tôi làm cho biểu đồ xuất hiện bằng cách gọi show() của pyplot

phương pháp tại w. Nhớ lại rằng matplotlib bao bọc seaborn , có nghĩa là chúng ta cần yêu cầu matplotlib hiển thị các số liệu seaborn của chúng tôi . Bây giờ hãy chuyển sang một biểu đồ mẫu phức tạp hơn.

Sơ đồ phân phối các phát hiện chống vi-rút

Tiền đề cho cốt truyện sau đây như sau: giả sử chúng tôi muốn hiểu mức độ phân phối (tần suất) phát hiện phần mềm chống vi-rút trên các mẫu phần mềm độc hại trong bộ dữ liệu của chúng tôi để hiểu phần trăm phần mềm độc hại bị bỏ sót bởi hầu hết các công cụ chống vi-rút và phần trăm được phát hiện bởi hầu hết động cơ.

Thông tin này cung cấp cho chúng tôi cái nhìn về hiệu quả của ngành công nghiệp chống vi-rút thương mại. Chúng ta có thể làm điều này bằng cách vẽ một biểu đồ thanh (một biểu đồ) hiển thị, đối với mỗi số lần phát hiện, tỷ lệ các mẫu phần mềm độc hại có số lần phát hiện đó, như thể hiện trong Hình 9-7.



Hình 9-7: Trục quan hóa phân phối phát hiện chống vi-rút (tích cực)

Trục x của hình này biểu thị các danh mục mẫu phần mềm độc hại, được sắp xếp theo số lượng trong tổng số 57 công cụ chống vi-rút đã phát hiện ra chúng. Nếu một mẫu bị 50 trong số 57 công cụ phát hiện là độc hại, mẫu đó sẽ được xếp vào hạng 50, nếu mẫu đó chỉ được phát hiện bởi 10 công cụ trong số 57 công cụ, thì mẫu đó sẽ được xếp vào hạng 10. Chiều cao của mỗi thanh tỷ lệ thuận với tổng số mẫu kết thúc trong danh mục đó.

Cốt truyện cho thấy rõ ràng nhiều mẫu phần mềm độc hại được phát hiện bởi hầu hết trong số 57 công cụ chống vi-rút của chúng tôi (được hiển thị bằng vết sưng lớn về tần suất ở khu vực phía bên phải của biểu đồ) nhưng cũng có một số ít đáng kể các mẫu được phát hiện bởi một số ít công cụ (được hiển thị ở khu vực ngoài cùng bên trái của biểu đồ). Chúng tôi không hiển thị các mẫu do ít hơn 5 công cụ phát hiện do phương pháp mà tôi đã sử dụng để xây dựng bộ dữ liệu này: Tôi xác định phần mềm độc hại là các mẫu mà 5 công cụ chống vi-rút trả lên phát hiện được. Kết quả đó thi này, với số lượng mẫu đáng kể chỉ nhận được 5-30 lần phát hiện, cho thấy rằng vẫn còn sự bất đồng đáng kể giữa các công cụ trong việc phát hiện phần mềm độc hại. Một mẫu bị 10 trong số 57 công cụ phát hiện là phần mềm độc hại cho thấy 47 công cụ không phát hiện được phần mềm đó hoặc 10 công cụ đã mắc lỗi và đưa ra kết quả dương tính giả trên một tập lệnh tính. Khả năng thứ hai là rất khó xảy ra, bởi vì các sản phẩm của nhà cung cấp phần mềm chống vi-rút có tỷ lệ dương tính giả rất thấp: nhiều khả năng là hầu hết các công cụ đã bỏ sót các mẫu này.

Tạo biểu đồ này chỉ cần một vài dòng mã biểu đồ, như trong [Liệt kê 9-12](#).

```

gửi trúc nhập khẩu
hải sản nhập khẩu
từ matplotlib nhập pyplot
phần mềm độc hại = pandas.read_csv("malware_data.csv")
u axis = seaborn.distplot(malware['positives'])
v axis.set(xlabel="Số lượng động cơ phát hiện từng mẫu (trong số 57)",
            ylabel="Số lượng mẫu trong tập dữ liệu",
            title="Phát hiện phần mềm độc hại chống vi-rút thương mại")
pyplot.show()

```

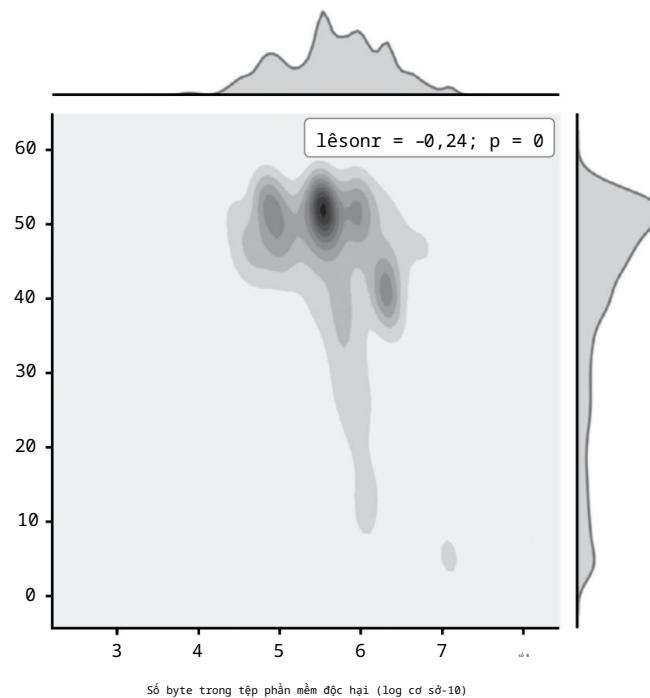
[Liệt kê 9-12: Vẽ biểu đồ phân phối các giá trị dương](#)

Gói seaborn có chức năng tích hợp để tạo các lô phân phối (biểu đồ), và do đó, tất cả những gì chúng ta đã làm là chuyển cho hàm distplot dữ liệu mà chúng ta muốn hiển thị, đó là phần mềm độc hại['positives']. Sau đó, chúng tôi sử dụng đối tượng trực do seaborn trả về để định cấu hình tiêu đề lô, nhãn trục x và nhãn trục y để mô tả lô v của chúng tôi.

Bây giờ, hãy thử một biểu đồ seaborn với hai biến số: số lần phát hiện tích cực đối với phần mềm độc hại (tệp có năm lần phát hiện trả lên) và kích thước tệp của chúng. Trước đây chúng ta đã tạo biểu đồ này bằng matplotlib trong [Hình 9-3](#), nhưng chúng ta có thể đạt được kết quả hấp dẫn và nhiều thông tin hơn bằng cách sử dụng biểu đồ chung của seaborn chức năng. Biểu đồ kết quả, được hiển thị trong [Hình 9-8](#), có nhiều thông tin nhưng cần một chút nỗ lực để hiểu lúc đầu, vì vậy chúng ta hãy xem qua biểu đồ này.

Biểu đồ này tương tự như biểu đồ mà chúng tôi đã tạo trong [Hình 9-7](#), nhưng thay vì hiển thị phân phối của một biến duy nhất thông qua độ cao của thanh, biểu đồ này hiển thị phân phối của hai biến (kích thước của tệp phần mềm độc hại, trên trục x và số lần phát hiện, trên trục y) thông qua cường độ màu. Vùng càng tối thì càng có nhiều dữ liệu trong vùng đó. Ví dụ: chúng ta có thể thấy rằng các tệp thường có kích thước khoảng 105,5 và giá trị dương khoảng 53.

Các ô con ở phía trên và bên phải của các ô chính hiển thị một phiên bản mượt mà của các tần số của dữ liệu kích thước và phát hiện, cho thấy sự phân bố của các phát hiện (như chúng ta đã thấy trong biểu đồ trước) và kích thước tệp.



Hình 9-8: Trực quan hóa việc phân phối kích thước tệp phần mềm độc hại so với các phát hiện tích cực

Cốt truyện trung tâm là thú vị nhất, bởi vì nó cho thấy mối quan hệ giữa kích thước và tích cực. Thay vì hiển thị các điểm dữ liệu riêng lẻ, như trong Hình 9-3 với matplotlib, nó hiển thị xu hướng chung theo cách rõ ràng hơn nhiều. Điều này cho thấy rằng các tệp phần mềm độc hại rất lớn (kích thước 106 trở lên) thường ít bị các công cụ chống vi-rút phát hiện hơn, điều này cho chúng tôi biết rằng chúng tôi có thể muốn tùy chỉnh xây dựng một giải pháp chuyên phát hiện phần mềm độc hại đó.

Việc tạo biểu đồ này chỉ yêu cầu một lệnh gọi biểu đồ đến seaborn, như trong [Liệt kê 9-13](#).

giao trúc nhập khẩu
hai sản nhập khẩu
nhập numpy
từ matplotlib nhập pyplot

```

phần mềm độc hại = pandas.read_csv("malware_data.csv")
u axis=seaborn.jointplot(x=numpy.log10(phần mềm độc hại['size']),
y=phần mềm độc hại['tích cực'],
loại="kde")
v axis.set_axis_labels("Số byte trong tệp phần mềm độc hại (log cơ số-10)",
"Số lượng công cụ phát hiện phần mềm độc hại (trong số 57)")
pyplot.show()

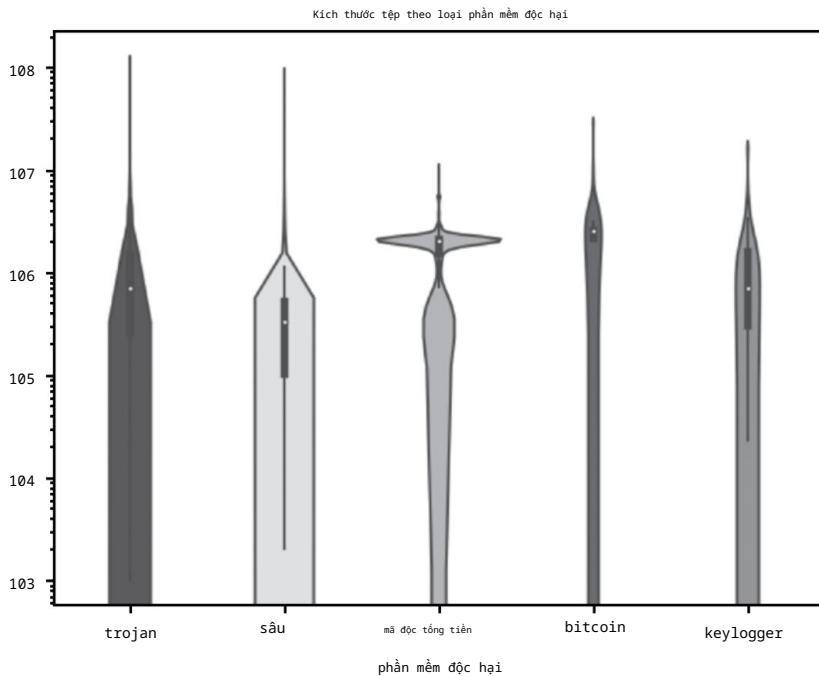
```

[Liệt kê 9-13: Sơ đồ phân phối kích thước tệp phần mềm độc hại so với phát hiện tích cực](#)

Ở đây, chúng ta sử dụng hàm `jointplot` của `seaborn` để tạo biểu đồ phân phối chung của các cột tích cực và kích thước trong DataFrame `u` của chúng tôi. Ngoài ra, hơi khó hiểu, đối với hàm `nối` của `seaborn`, chúng ta phải gọi một hàm khác với trong [Liệt kê 9-11](#) để gắn nhãn cho các trục của chúng ta: hàm `set_axis_labels()` v, đổi số đầu tiên của nó là nhãn trục x và đổi số thứ hai của nó là nhãn trục y.

Tạo một âm mưu vĩ cầm

Loại cốt truyện cuối cùng mà chúng ta khám phá trong chương này là cốt truyện về cây vĩ cầm trên biển. Biểu đồ này cho phép chúng tôi khám phá một cách tinh tế sự phân phối của một biến nhất định trên một số loại phần mềm độc hại. Ví dụ: giả sử chúng tôi muốn xem phân bố kích thước tệp cho mỗi loại phần mềm độc hại trong tập dữ liệu của mình. Trong trường hợp này, chúng ta có thể tạo một biểu đồ như [Hình 9-9](#).



Hình 9-9: Trực quan hóa kích thước tệp theo loại phần mềm độc hại

Trên trục y của biểu đồ này là kích thước tệp, được biểu thị bằng lũy thừa của 10. Trên trục x, chúng tôi liệt kê từng loại phần mềm độc hại. Như bạn có thể thấy, độ dày của các thanh biểu thị từng loại tệp khác nhau ở các mức kích thước khác nhau, cho biết lượng dữ liệu cho loại phần mềm độc hại đó có kích thước đó. Ví dụ: bạn có thể thấy rằng có một số lượng đáng kể các tệp phần mềm tổng tiền rất lớn và các loại sâu đó có xu hướng có kích thước tệp nhỏ hơn—có thể là do các loại sâu nhằm mục đích lây lan nhanh chóng trên mạng và do đó, các tác giả sâu có xu hướng giảm thiểu kích thước tệp của chúng. Việc biết các mẫu này có khả năng giúp chúng tôi phân loại các tệp không xác định tốt hơn (tệp lớn hơn có nhiều khả năng là phần mềm tổng tiền hơn và ít có khả năng là sâu hơn) hoặc dạy chúng tôi nên tập trung vào kích thước tệp nào trong một công cụ phòng thủ nhằm mục tiêu vào một loại cụ thể của phần mềm độc hại.

Tạo biểu đồ violin để cầm đầu một cuộc gọi biểu đồ, như trong Liệt kê 9-14.

```

gửi trúc nhập khẩu
hải sản nhập khẩu
từ matplotlib nhập pyplot

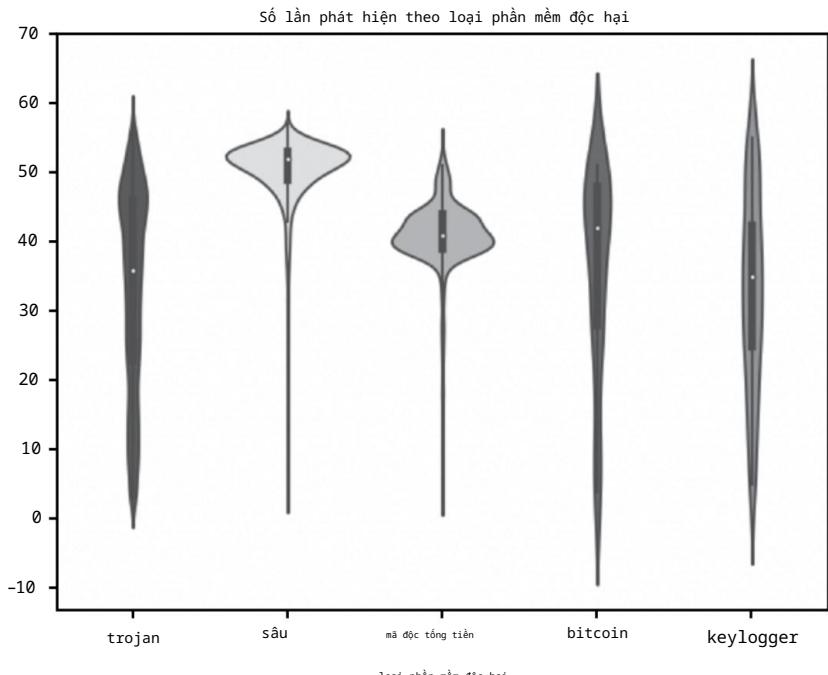
phần mềm độc hại = pandas.read_csv("malware_data.csv")

u axis = seaborn.violinplot(x=malware['type'], y=malware['size'])
v axis.set(xlabel="Loại phần mềm độc hại", ylabel="Kích thước tệp tính bằng byte (log base-10)",
            title="Kích thước tệp theo loại phần mềm độc hại", yscale="log")
w pyplot.show()

```

Liệt kê 9-14: Tạo một cột truyện vĩ cầm

Trong Liệt kê 9-14, đầu tiên chúng ta tạo biểu đồ violin. Tiếp theo, chúng tôi yêu cầu seaborn đặt nhãn và tiêu đề trực và đặt trực y thành tỷ lệ log v. Cuối cùng, chúng tôi làm cho biểu đồ xuất hiện w. Chúng ta cũng có thể tạo một biểu đồ tương tự hiển thị số lượng xác nhận cho từng loại phần mềm độc hại, như trong Hình 9-10.



Hình 9-10: Trực quan hóa số lượng phần mềm chống vi rút dương tính (phát hiện) trên mỗi loại phần mềm độc hại

Sự khác biệt duy nhất giữa Hình 9-9 và Hình 9-10 là thay vì xem xét kích thước tệp trên trục y, chúng tôi đang xem xét số lượng kết quả xác thực mà mỗi tệp nhận được. Kết quả cho thấy một số xu hướng thú vị. Ví dụ: ransomware hầu như luôn được phát hiện bởi hơn 30 máy quét. Ngược lại, các loại phần mềm độc hại bitcoin, trojan và keylogger được phát hiện bởi ít hơn 30 máy quét trong một phần đáng kể thời gian, có nghĩa là nhiều loại này hơn

đang vượt qua khả năng phòng thủ của ngành bảo mật (những người không có máy quét phát hiện các tệp này được cài đặt có khả năng bị nhiễm các mẫu này). Liệt kê 9-15 cho thấy cách tạo biểu đồ như trong Hình 9-10.

giao trúc nhập khẩu
hải sản nhập khẩu
từ matplotlib nhập pyplot

```
phần mềm độc hại = pandas.read_csv("malware_data.csv")

axis = seaborn.violinplot(x=malware['type'], y=malware['positives'])
axis.set(xlabel="Loại phần mềm độc hại", ylabel="Số lần phát hiện nhà cung cấp",
         title="Số lần phát hiện theo loại phần mềm độc hại")
pyplot.show()
```

Liệt kê 9-15: Trực quan hóa các phát hiện chống vi-rút cho mỗi loại phần mềm độc hại

Sự khác biệt duy nhất trong mã này và trước đó là chúng tôi vượt qua chức năng violinplot dữ liệu khác nhau (phần mềm độc hại ['dương tính'] thay vì phần mềm độc hại ['size']), chúng tôi gắn nhãn các trục khác nhau, chúng tôi đặt tiêu đề khác và chúng tôi bỏ qua việc đặt tỷ lệ trực y thành log-10.

Bản tóm tắt

Trong chương này, bạn đã học cách trực quan hóa dữ liệu phần mềm độc hại cho phép bạn có được những hiểu biết rõ ràng về các mối đe dọa có xu hướng và hiệu quả của các công cụ bảo mật. Bạn đã sử dụng pandas, matplotlib và seaborn để tạo các hình ảnh trực quan của riêng mình và rõ ràng hơn về các bộ dữ liệu mẫu.

Bạn cũng đã học cách sử dụng các phương pháp như `description()` trong pandas để hiển thị số liệu thống kê hữu ích và cách trích xuất các tập hợp con của tập dữ liệu của bạn. Sau đó, bạn đã sử dụng các tập hợp con dữ liệu này để tạo trực quan hóa của riêng mình nhằm đánh giá các cải tiến trong khả năng phát hiện phần mềm chống vi-rút, phân tích các loại phần mềm độc hại có xu hướng và trả lời các câu hỏi rộng hơn khác.

Đây là những công cụ mạnh mẽ giúp chuyển đổi dữ liệu bảo mật mà bạn có thành trí thông minh có thể hành động có thể cung cấp thông tin cho sự phát triển của các công cụ và kỹ thuật mới. Tôi hy vọng bạn sẽ tìm hiểu thêm về trực quan hóa dữ liệu và kết hợp xếp hạng chúng vào quy trình phân tích bảo mật và phần mềm độc hại của mình.

10

Sâu L kiêm Ba si cs



Học sâu là một loại máy học đã phát triển nhanh chóng trong vài năm qua, do những cải tiến trong quy trình sức mạnh và kỹ thuật học sâu.

Thông thường, học sâu đề cập đến các mạng thần kinh sâu hoặc nhiều lớp, vượt trội trong việc thực hiện các nhiệm vụ rất phức tạp, thường lấy con người làm trung tâm trong lịch sử, như nhận dạng hình ảnh và dịch ngôn ngữ.

Ví dụ: việc phát hiện xem một tệp có chứa mã độc chính xác của một số mã độc mà bạn đã thấy trước đây hay không rất đơn giản đối với chương trình máy tính và không yêu cầu máy học nâng cao. Nhưng việc phát hiện xem một tệp có chứa mã độc hại tương tự như mã độc mà bạn đã thấy trước đây hay không là một nhiệm vụ phức tạp hơn nhiều. Các sơ đồ phát hiện dựa trên chữ ký truyền thống thường cứng nhắc và hoạt động kém đối với phần mềm độc hại chưa từng thấy hoặc bị che giấu, trong khi các mô hình học sâu có thể nhìn thấu những thay đổi bề ngoài và xác định các tính năng cốt lõi tạo ra một mã độc hại. Giống nhau

dành cho hoạt động mạng, phân tích hành vi và các lĩnh vực liên quan khác. Khả năng chọn ra các đặc điểm hữu ích trong một loạt tiếng ồn làm cho học sâu trở thành một công cụ cực kỳ mạnh mẽ cho các ứng dụng an ninh mạng.

Học sâu chỉ là một loại học máy (chúng tôi đã đề cập đến học máy nói chung trong Chương 6 và 7). Nhưng nó thường dẫn đến các mô hình đạt được độ chính xác cao hơn so với các phương pháp mà chúng ta đã thảo luận trong các chương trước, đó là lý do tại sao toàn bộ lĩnh vực máy học đã nhấn mạnh vào học sâu trong khoảng 5 năm trở lại đây. Nếu bạn quan tâm đến việc làm việc trong lĩnh vực khoa học dữ liệu bảo mật tiên tiến nhất, bạn cần học cách sử dụng học sâu. Tuy nhiên, cần thận trọng: học sâu khó hiểu hơn so với các phương pháp học máy mà chúng ta đã thảo luận ở phần đầu của cuốn sách này và nó đòi hỏi một số cam kết cũng như phép tính ở cấp trung học để hiểu đầy đủ. Bạn sẽ thấy rằng thời gian bạn đầu tư để hiểu nó sẽ mang lại lợi ích cho công việc khoa học dữ liệu bảo mật của bạn về khả năng xây dựng các hệ thống máy học chính xác hơn. Vì vậy, chúng tôi khuyên bạn nên đọc chương này một cách cẩn thận và cố gắng hiểu nó cho đến khi bạn hiểu được nó! Bắt đầu nào.

Học sâu là gì?

Các mô hình học sâu học cách xem dữ liệu đào tạo của chúng dưới dạng một hệ thống phân cấp khái niệm lồng nhau, cho phép chúng thể hiện các mẫu cực kỳ phức tạp. Nói cách khác, các mô hình này không chỉ xem xét các tính năng ban đầu mà bạn cung cấp cho chúng mà còn tự động kết hợp các tính năng này để tạo thành các siêu tính năng mới, được tối ưu hóa, sau đó chúng kết hợp để tạo thành nhiều tính năng hơn, v.v.

"Deep" cũng đề cập đến kiến trúc được sử dụng để thực hiện điều này, thường bao gồm nhiều lớp đơn vị xử lý, mỗi lớp sử dụng đầu ra của lớp trước làm đầu vào. Mỗi đơn vị xử lý này được gọi là nơ-ron và toàn bộ kiến trúc mô hình được gọi là mạng nơ-ron hoặc mạng nơ-ron sâu khi có nhiều lớp.

Để xem kiến trúc này có thể hữu ích như thế nào, chúng ta hãy nghĩ về một chương trình có gắng phân loại hình ảnh là xe đẹp hoặc xe đẹp một bánh. Đối với con người, đây là một nhiệm vụ dễ dàng, nhưng việc lập trình một máy tính để xem xét một lưới pixel và cho biết đối tượng mà nó đại diện là khá khó. Một số pixel nhất định cho biết xe đẹp một bánh tồn tại trong một hình ảnh sẽ có ý nghĩa hoàn toàn khác trong hình ảnh tiếp theo nếu xe đẹp một bánh di chuyển một chút, được đặt ở một góc khác hoặc có màu khác.

Các mô hình học sâu vượt qua điều này bằng cách chia nhỏ vấn đề thành các phần dễ quản lý hơn. Ví dụ: lớp nơ-ron đầu tiên của mạng nơ-ron sâu trước tiên có thể chia hình ảnh thành các phần và chỉ xác định các đặc điểm hình ảnh ở mức độ thấp, như các cạnh và đường viền của hình dạng trong hình ảnh. Các tính năng đã tạo này được đưa vào lớp tiếp theo của mạng để tìm các mẫu trong số các tính năng. Các mẫu này sau đó được đưa vào các lớp tiếp theo, cho đến khi mạng xác định được các hình dạng chung và cuối cùng là các đối tượng hoàn chỉnh. Trong ví dụ về xe đẹp một bánh của chúng ta, lớp đầu tiên có thể tìm thấy các đường, lớp thứ hai có thể thấy các đường tạo thành vòng tròn và lớp thứ ba có thể xác định rằng một số vòng tròn nhất định thực sự là bánh xe. Theo cách này, thay vì nhìn vào một khối pixe-

mô hình có thể thấy rằng mỗi hình ảnh có một số tính năng meta "bánh xe" nhất định. Ví dụ, sau đó, nó có thể biết rằng hai bánh xe có thể biểu thị một chiếc xe đạp, trong khi một bánh xe có nghĩa là một chiếc xe đạp một bánh.

Trong chương này, chúng ta tập trung vào cách mạng nơ-ron thực sự hoạt động, cả về mặt toán học và cấu trúc. Đầu tiên, tôi sử dụng một mạng nơ-ron rất cơ bản làm ví dụ để giải thích chính xác nơ-ron là gì và cách nó kết nối với các nơ-ron khác để tạo ra một mạng nơ-ron. Thứ hai, tôi mô tả các quy trình toán học được sử dụng để huấn luyện các mạng này. Cuối cùng, tôi mô tả một số loại mạng thần kinh phổ biến, chúng đặc biệt như thế nào và chúng giỏi ở điểm nào.

Điều này sẽ giúp bạn chuẩn bị tốt cho Chương 11, nơi bạn sẽ thực sự tạo các mô hình học sâu bằng Python.

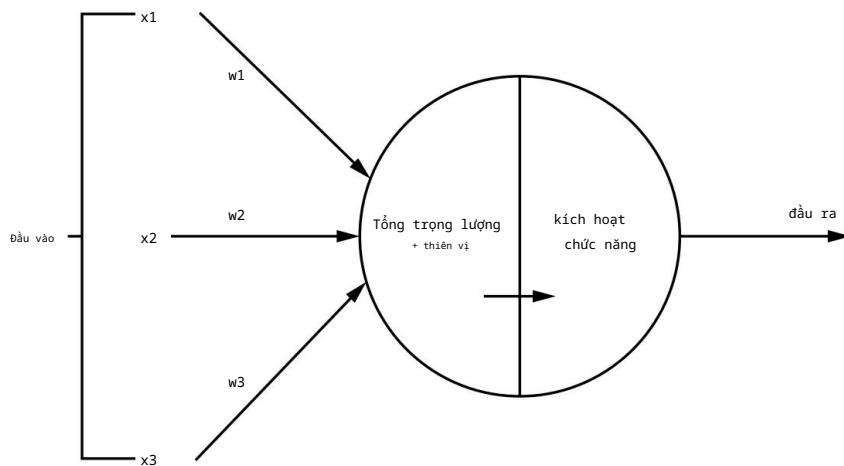
Mạng lưới thần kinh hoạt động như thế nào

Các mô hình học máy chỉ đơn giản là các hàm toán học lớn. Ví dụ: chúng tôi lấy dữ liệu đầu vào (chẳng hạn như tệp HTML được biểu thị dưới dạng một chuỗi số), áp dụng chức năng máy học (chẳng hạn như mạng thần kinh) và chúng tôi nhận được kết quả đầu ra cho chúng tôi biết tệp HTML trông độc hại như thế nào. Mỗi mô hình học máy chỉ là một chức năng chứa các tham số có thể điều chỉnh được tối ưu hóa trong quá trình đào tạo.

Nhưng chức năng học sâu thực sự hoạt động như thế nào và nó trông như thế nào? Mạng nơ-ron, như tên của nó, chỉ là mạng của nhiều nơ-ron. Vì vậy, trước khi hiểu cách thức hoạt động của mạng nơ-ron, trước tiên chúng ta cần biết nơ-ron là gì.

Giải phẫu của một tế bào thần kinh

Bản thân các tế bào thần kinh chỉ là một loại chức năng nhỏ, đơn giản. Hình 10-1 cho thấy một nơ-ron đơn trông như thế nào.



Hình 10-1: Trực quan hóa một nơ-ron đơn lẻ

Bạn có thể thấy rằng dữ liệu đầu vào đến từ bên trái và một đầu ra duy nhất số xuất hiện ở bên phải (mặc dù một số loại tế bào thần kinh tạo ra nhiều đầu ra). Giá trị của đầu ra là một hàm của dữ liệu đầu vào của nơ-ron và một số tham số (được tối ưu hóa trong quá trình huấn luyện).

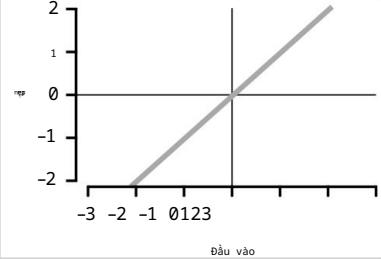
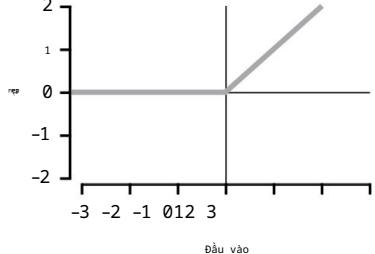
Hai bước xảy ra bên trong mỗi nơ-ron để biến đổi dữ liệu đầu vào thành đầu ra.

Đầu tiên, tổng trọng số của các đầu vào của nơ-ron được tính toán. Trong Hình 10-1, mỗi số đầu vào, x_i , đi vào nơ-ron được nhân với một giá trị trọng số liên quan, w_i . Các giá trị kết quả được cộng lại với nhau (tạo ra một tổng trọng số) mà một thuật ngữ sai lệch được thêm vào. Độ lệch và trọng số là các tham số của nơ-ron được sửa đổi trong quá trình đào tạo để tối ưu hóa mô hình.

Thứ hai, một hàm kích hoạt được áp dụng cho tổng trọng số cộng với giá trị sai lệch. Mục đích của hàm kích hoạt là áp dụng phép biến đổi tuyến tính cho tổng trọng số, đây là phép biến đổi tuyến tính của dữ liệu đầu vào của nơ-ron. Có nhiều loại hàm kích hoạt phổ biến và chúng có xu hướng khá đơn giản. Yêu cầu duy nhất của hàm kích hoạt là nó có thể khả vi, cho phép chúng ta sử dụng lan truyền ngược để tối ưu hóa các tham số (chúng ta sẽ thảo luận ngắn gọn về quy trình này trong "Đào tạo Mạng nơ-ron" ở trang 189).

Bảng 10-1 cho thấy nhiều chức năng kích hoạt phổ biến khác và giải thích những chức năng nào có xu hướng tốt cho những mục đích nào.

Bảng 10-1: Các chức năng kích hoạt phổ biến

Tên Lô đất	phương trình	Sự miêu tả
Danh tính		f_{xx} phương trình Về cơ bản: không có chức năng kích hoạt!
ReLU		f_x $\begin{aligned} &0 \text{ cho } x < 0 \\ &x \text{ cho } x \geq 0 \end{aligned}$ Chỉ tối đa ($0, x$). ReLU cho phép học nhanh và linh hoạt hơn đối với vấn đề biến đổi độ dốc (được giải thích ở phần sau của chương này) so với các chức năng khác, chẳng hạn như sigmoid.

Tên Lô đất	phương trình	Sự miêu tả
rò rỉ ReLU	$fx = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$	Giống như ReLU bình thường, nhưng thay vì α , một phần không đổi nhỏ của x được trả lại. Nói chung, bạn chọn α rất nhỏ, chẳng hạn như $0,01$. Ngoài ra, α vẫn cố định trong quá trình huấn luyện.
TRƯỚC	$fx = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases}$	Điều này giống như ReLU bị rò rỉ, nhưng trong PReLU, α là tham số có giá trị được tối ưu hóa trong quá trình huấn luyện, cùng với tham số trọng số và độ lệch chuẩn.
ELU	$fx = \begin{cases} ex - 1 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	Thích PReLU ở chỗ α là một tham số, nhưng thay vì đi xuống vô tận với hệ số góc α khi $x < 0$, đường cong được giới hạn bởi α , vì ex sẽ luôn nằm trong khoảng từ 0 đến 1 khi $x < 0$.
Bứt c chân	$fx = \begin{cases} 0 & \text{cho } x < 0 \\ 1 & \text{cho } x \geq 0 \end{cases}$	Chỉ là hàm bước: hàm trả về 0 trừ khi $x \geq 0$, trong trường hợp đó hàm trả về 1 .
Gaussian	$ngôai hối = e^{-x^2}$	Một đường cong hình chuông có giá trị lớn nhất đạt định bằng 1 khi $x = 0$.

Bảng 10-1: Các chức năng kích hoạt phổ biến, tiếp theo

Tên Lô đất	phương trình	Sự miêu tả
sigmoid	<p>fx $\frac{e^x}{e^x + 1}$</p>	<p>Do vấn đề về độ dốc biến mất (được giải thích ở phần sau của chương này), các hàm kích hoạt sigmoid thường chỉ được sử dụng ở lớp cuối cùng của mạng nơ-ron. Bởi vì đầu ra là liên tục và bị giới hạn giữa 0 và 1, nơ-ron sigmoid là một đại diện tốt cho xác suất đầu ra.</p>
Softmax (đa đầu ra)	<p>fx $\frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}}$ cho $j = 1, 2, \dots, K$</p>	<p>Xuất ra nhiều giá trị có tổng bằng 1. Các hàm kích hoạt Softmax thường được sử dụng trong lớp cuối cùng của mạng để biểu thị xác suất phân loại, bởi vì Softmax buộc tất cả các đầu ra từ một nơ-ron phải có tổng bằng 1.</p>

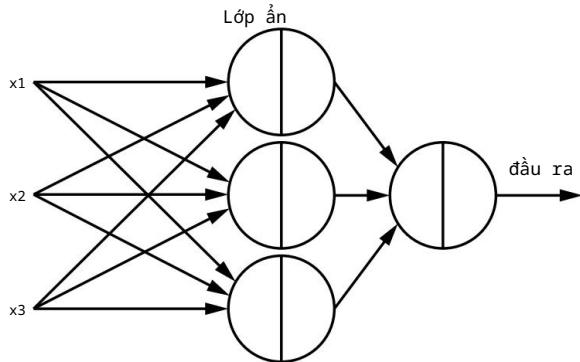
Đơn vị tuyến tính được chỉnh lưu (ReLU) cho đến nay là chức năng kích hoạt phổ biến nhất được sử dụng ngày nay và nó đơn giản là $\max(0, s)$. Ví dụ: giả sử tổng trọng số cộng với giá trị sai lệch của bạn được gọi là s . Nếu s lớn hơn 0, thì đầu ra nơ-ron của bạn là s và nếu s bằng hoặc nhỏ hơn 0, thì đầu ra nơ-ron của bạn bằng 0. Bạn có thể biểu diễn toàn bộ chức năng của nơ-ron ReLU dưới dạng $\max(0, \text{tổng trọng số của -inputs + bias})$, hoặc cụ thể hơn, như sau đối với n đầu vào:

$$\text{tối đa } 0, \quad \sum_{i=1}^n w_i x_i + b$$

Các hàm kích hoạt phi tuyến thực sự là lý do chính giải thích tại sao các mạng nơ-ron như vậy có thể xấp xỉ bất kỳ hàm liên tục nào, đó là lý do chính giải thích tại sao chúng rất mạnh. Trong các phần sau, bạn sẽ tìm hiểu cách các nơ-ron được kết nối với nhau để tạo thành một mạng và sau đó bạn sẽ hiểu được tại sao các hàm kích hoạt phi tuyến tính lại quan trọng đến vậy.

Một mạng nơ-ron

Để tạo một mạng nơ-ron, bạn sắp xếp các nơ-ron trong một đồ thị có hướng (một mạng lưới) với một số lớp, kết nối để tạo thành một hàm lớn hơn nhiều. Hình 10-2 cho thấy một ví dụ về mạng thần kinh nhỏ.



Hình 10-2: Ví dụ về mạng nơ-ron bốn nơ-ron rất nhỏ, nơi dữ liệu được truyền từ nơ-ron này sang nơ-ron khác thông qua các kết nối.

Trong Hình 10-2, chúng ta có các đầu vào ban đầu: x_1 , x_2 và x_3 ở phía bên trái. Các bản sao của các giá trị xi này được gửi dọc theo các kết nối tới từng nơ-ron trong lớp ẩn (một lớp nơ-ron có đầu ra không phải là đầu ra cuối cùng của mô hình), dẫn đến ba giá trị đầu ra, một giá trị từ mỗi nơ-ron. Cuối cùng, mỗi đầu ra của ba nơ-ron này được gửi đến một nơ-ron cuối cùng, nơ-ron này đưa ra kết quả cuối cùng của mạng nơ-ron.

Mọi kết nối trong mạng nơ-ron đều được liên kết với một trọng số tham số, w và mọi nơ-ron cũng chứa tham số sai lệch, b (được thêm vào tổng trọng số), do đó, tổng số tham số có thể tối ưu hóa trong mạng nơ-ron cơ bản là số cạnh kết nối đầu vào với nơ-ron, cộng với số lượng tế bào thần kinh. Ví dụ, trong mạng được hiển thị trong Hình 10-2, có tổng số 4 nơ-ron, cộng với 9 + 3 cạnh, mang lại tổng cộng 16 tham số có thể tối ưu hóa. Vì đây chỉ là một ví dụ nên chúng tôi đang sử dụng một mạng nơ-ron rất nhỏ—mạng nơ-ron thường có hàng nghìn nơ-ron và hàng triệu kết nối.

Định lý xấp xỉ phô quát

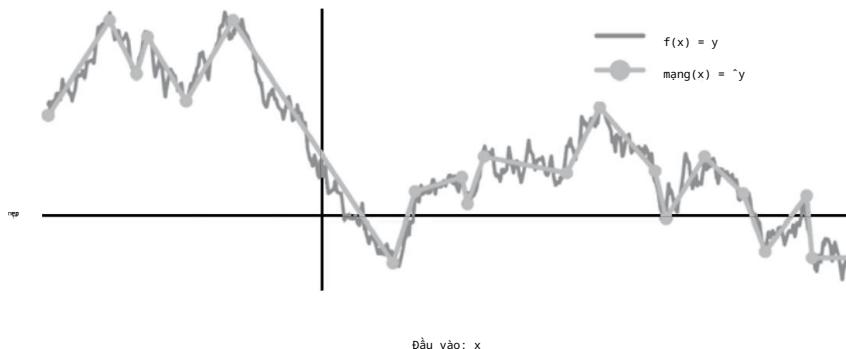
Một khía cạnh nổi bật của mạng nơ-ron là chúng là các giá trị xấp xỉ phô quát: được cung cấp đủ nơ-ron, cũng như các giá trị độ lệch và trọng số phù hợp, về cơ bản, mạng nơ-ron có thể mô phỏng bất kỳ loại hành vi nào. Mạng thần kinh được hiển thị trong Hình 10-2 là mạng truyền tiếp, có nghĩa là dữ liệu luôn truyền về phía trước (từ trái sang phải trong hình ảnh).

Định lý xấp xỉ phô quát mô tả khái niệm về tính phô quát một cách hình thức hơn. Nó tuyên bố rằng một mạng chuyển tiếp với một lớp tế bào thần kinh ẩn duy nhất có các hàm kích hoạt phi tuyến tính có thể xấp xỉ (với một lỗi nhỏ tùy ý) bất kỳ hàm liên tục nào trên một tập hợp con nhỏ gọn của R^n

¹ Điều đó hơi khó nghe, nhưng điều đó chỉ có nghĩa là với đủ nơ-ron, mạng nơ-ron có thể gần đúng với bất kỳ hàm liên tục, có giới hạn nào với số lượng đầu vào và đầu ra hữu hạn.

1. Có thể coi R^n là một không gian Euclidian n chiều, trong đó mọi số đều là số thực. Ví dụ: R^2 đại diện cho tất cả các bộ giá trị thực có thể có của độ dài 2, như $(3.5, -5)$.

Nói cách khác, định lý phát biểu rằng bất kể hàm chúng ta muốn tính gần đúng là gì, về mặt lý thuyết, vẫn có một số mạng thần kinh với các tham số phù hợp có thể thực hiện công việc đó. Ví dụ, nếu bạn vẽ một hàm liên tục, nguệch ngoạc, $f(x)$, như trong Hình 10-3, sẽ tồn tại một số công việc của mạng nơ-ron sao cho với mọi đầu vào có thể có của x , $f(x) \approx \text{network}(x)$, không hàm $f(x)$ phức tạp đến đâu. Đây là một lý do khiến mạng lưới thần kinh có thể trờ nên mạnh mẽ như vậy.

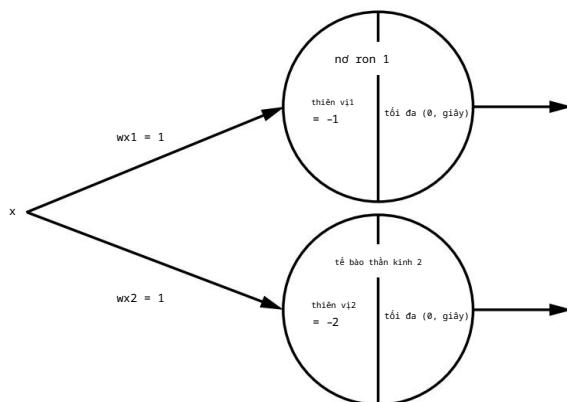


Hình 10-3: Ví dụ về cách một mạng lưới thần kinh nhỏ có thể xấp xỉ một hàm thú vị. Khi số lượng tế bào thần kinh tăng lên, sự khác biệt giữa y và \hat{y} sẽ tiến tới 0.

Trong các phần tiếp theo, chúng tôi sẽ xây dựng một mạng nơ-ron đơn giản bằng tay để giúp bạn hiểu cách thức và lý do chúng tôi có thể lập mô hình các loại hành vi khác nhau như vậy, với các tham số phù hợp. Mặc dù chúng tôi làm điều này ở quy mô rất nhỏ chỉ sử dụng một đầu vào và đầu ra duy nhất, nhưng nguyên tắc tương tự vẫn đúng khi bạn đang xử lý nhiều đầu vào và đầu ra cũng như các hành vi cực kỳ phức tạp.

Xây dựng mạng lưới thần kinh của riêng bạn

Để xem tính phỏng biến này đang hoạt động, hãy thử xây dựng mạng lưới thần kinh của riêng chúng ta. Chúng tôi bắt đầu với hai tế bào thần kinh ReLU, sử dụng một đầu vào duy nhất x , như trong Hình 10-4. Sau đó, chúng ta sẽ thấy các giá trị (tham số) trọng số và sai lệch khác nhau có thể được sử dụng như thế nào để mô hình hóa các chức năng và kết quả khác nhau.



Hình 10-4: Trực quan hóa hai nơ-ron được cung cấp dữ liệu đầu vào x

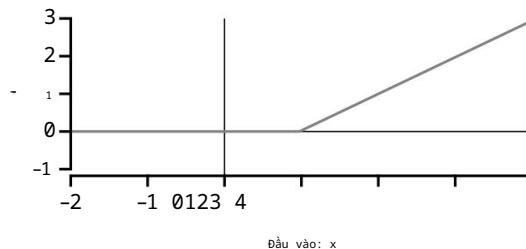
Ở đây, cả hai nơ-ron đều có trọng số là 1 và cả hai đều sử dụng chức năng kích hoạt ReLU. Sự khác biệt duy nhất giữa hai loại này là nơ-ron1 áp dụng giá trị sai lệch là -1, trong khi nơ-ron2 áp dụng giá trị sai lệch là -2. Hãy xem điều gì sẽ xảy ra khi chúng ta cung cấp cho nơ-ron1 một vài giá trị khác nhau của x. Bảng 10-2 tóm tắt các kết quả.

Bảng 10-2: Nơ-ron1

Đầu vào Tổng trọng số Tổng trọng số + sai số Đầu ra			
x	$x * wx + 1$	$x * wx + 1 + \text{sai số}$	tối đa ($x * wx + 1 + \text{sai số}$)
0	$0 * 1 = 0$	$0 + -1 = -1$	$\text{tối da}(0, -1) = 0$
1	$1 * 1 = 1$	$1 + -1 = 0$	$\text{tối da}(0, 0) = 0$
2	$2 * 1 = 2$	$2 + -1 = 1$	$\text{tối da}(0, 1) = 1$
3	$3 * 1 = 3$	$3 + -1 = 2$	$1 \text{ tối da}(0, 2) = 2$
4	$4 * 1 = 4$	$4 + -1 = 3$	$\text{tối da}(0, 3) = 3$
5	$5 * 1 = 5$	$5 + -1 = 4$	$\text{tối da}(0, 4) = 4$

Cột đầu tiên hiển thị một số đầu vào mẫu cho x và cột thứ hai hiển thị tổng trọng số kết quả. Cột thứ ba thêm tham số sai lệch và cột thứ tư áp dụng chức năng kích hoạt ReLU để tạo ra đầu ra của nơ-ron cho một đầu vào nhất định của x. Hình 10-5 cho thấy đồ thị của hàm nơron1.

nơ ron 1



Hình 10-5: Trực quan hóa nơ-ron1 dưới dạng một chức năng. Trục x biểu thị giá trị đầu vào duy nhất của nơ-ron và trục y biểu thị đầu ra của nơ-ron.

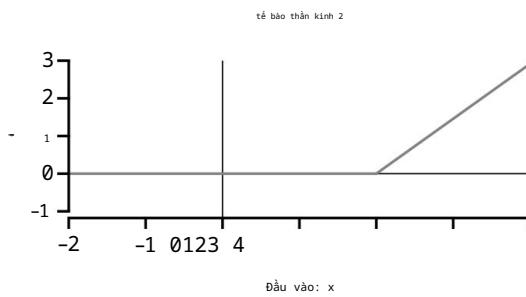
Bởi vì nơ-ron1 có độ lệch là -1, nên đầu ra của nơ-ron1 giữ ở mức 0 cho đến khi tổng trọng số vượt quá 1, sau đó nó đi lên với một độ dốc nhất định, như bạn có thể thấy trong Hình 10-5. Hệ số góc 1 đó được liên kết với giá trị trọng số $wx + 1$ là 1. Hãy nghĩ xem điều gì sẽ xảy ra với trọng số là 2: vì giá trị tổng trọng số sẽ tăng gấp đôi, nên góc trong Hình 10-5 sẽ xảy ra tại $x = 0,5$ thay vì $x = 1$ và đường thẳng sẽ di lên với hệ số góc là 2 thay vì 1.

Bây giờ hãy xem nơ-ron2, có giá trị sai lệch là -2 (xem Bảng 10-3).

Bảng 10-3: Nơ-ron2

Đầu vào TỔNG trọng số TỔNG trọng số + sai số Đầu ra			
x	$x * wx_2$	$(x * wx_2) + bias2$	$\max(0, (x * wx_2) + bias2)$
0	$0 * 1 = 0$	$0 + -2 = -2$	$\text{tối đa}(0, -2) = 0$
1	$1 * 1 = 1$	$1 + -2 = -1$	$\text{tối đa}(0, -1) = 0$
2	$2 * 1 = 2$	$2 + -2 = 0$	$\text{tối đa}(0, 0) = 0$
3	$3 * 1 = 3$	$3 + -2 = 1$	$\text{tối đa}(0, 1) = 1$
4	$4 * 1 = 4$	$4 + -2 = 2$	$\text{tối đa}(0, 2) = 2$
5	$5 * 1 = 5$	$5 + -2 = 3$	$\text{tối đa}(0, 3) = 3$

Bởi vì độ chêch của nơ-ron2 là -2, nên góc trong Hình 10-6 xảy ra tại $x = 2$ thay vì $x = 1$.



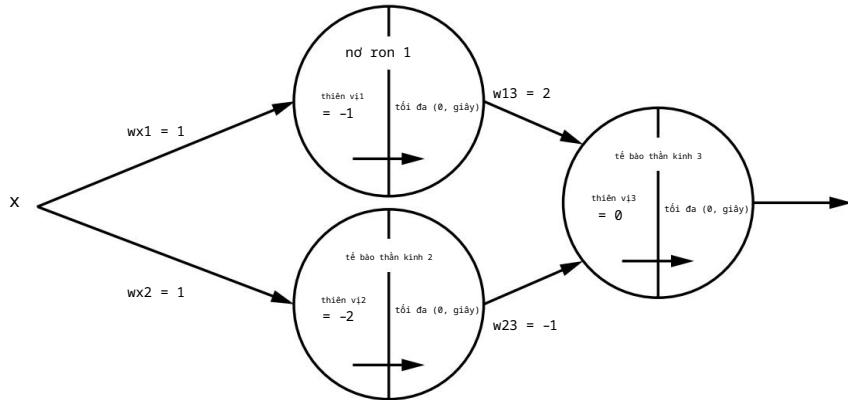
Hình 10-6: Trục quan hóa nơ-ron2 dưới dạng một chức năng

Vì vậy, bây giờ chúng tôi đã xây dựng hai chức năng rất đơn giản (nơ-ron), cả hai đều không làm gì trong một khoảng thời gian nhất định, sau đó tăng lên vô tận với độ dốc bằng 1. Bởi vì chúng tôi đang sử dụng nơ-ron ReLU, độ dốc của chức năng của từng nơ-ron bị ảnh hưởng bởi trọng số của nó, trong khi các số hạng độ lệch và trọng lượng của nó đều ảnh hưởng đến nơi bắt đầu của độ dốc. Khi bạn sử dụng các chức năng kích hoạt khác, các quy tắc tương tự sẽ được áp dụng. Bằng cách điều chỉnh các tham số, chúng tôi có thể thay đổi góc và độ dốc của chức năng của từng nơ-ron theo cách chúng tôi muốn.

Tuy nhiên, để đạt được tính phổ quát, chúng ta cần kết hợp các nơ-ron với nhau, điều này sẽ cho phép chúng ta tính gần đúng các chức năng phức tạp hơn. Hãy kết nối hai nơ-ron của chúng ta với một nơ-ron thứ ba, như trong Hình 10-7.

Điều này sẽ tạo ra một mạng ba nơ-ron nhỏ với một lớp ẩn duy nhất, bao gồm nơ-ron1 và nơ-ron2.

Trong Hình 10-7, dữ liệu đầu vào x được gửi đến cả nơ-ron1 và nơ-ron2. Sau đó, đầu ra của nơ-ron1 và nơ-ron2 được gửi làm đầu vào cho nơ-ron3, tạo ra đầu ra cuối cùng của mạng.



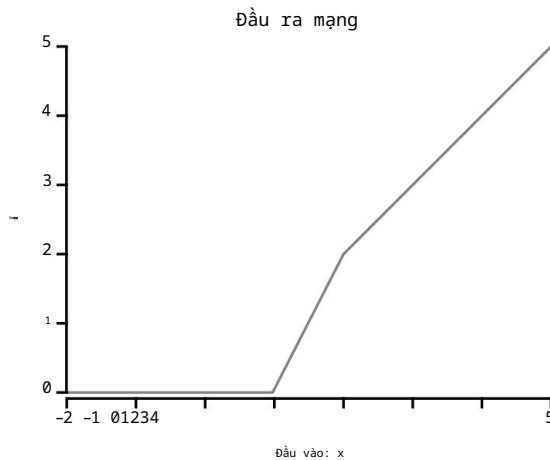
Hình 10-7: Trực quan hóa mạng ba n̄-ron nhỏ

Nếu bạn kiểm tra các trọng số trong Hình 10-7, bạn sẽ nhận thấy rằng trọng số w_{13} là 2, tăng gấp đôi đóng góp của $n̄\text{-ron1}$ cho $n̄\text{-ron3}$. Trong khi đó, w_{23} là -1, đảo ngược đóng góp của $n̄\text{-ron2}$. Về bản chất, $n̄\text{-ron3}$ chỉ đơn giản là áp dụng chức năng kích hoạt của nó cho $n̄\text{-ron1} * 2 - n̄\text{-ron2}$. Bảng 10-4 tóm tắt đầu vào và đầu ra tương ứng cho mạng kết quả.

Bảng 10-4: Mạng ba n̄-ron

Đầu vào	Đầu vào cho $n̄\text{-ron3}$	Tổng trọng số	Tổng trọng số + thiên vị	Đầu ra mạng cuối cùng
mạng ban đầu	$n̄\text{-ron1 } n̄\text{-ron2 } (n̄\text{-ron1} * w_{13}) + (n̄\text{-ron2} * w_{23})$	$(n̄\text{-ron1} * w_{13}) + (n̄\text{-ron2} * w_{23})$	$\max(0, (n̄\text{-ron1} * w_{13}) + (n̄\text{-ron2} * w_{23}) + \text{sai lệch3})$	thiên vị
0	0	0	$(0 * 2) + (0 * -1) = 0 \ 0 + 0 + 0 = 0$	tối đa (0, 0) = 0
1	0	0	$(0 * 2) + (0 * -1) = 0 \ 0 + 0 + 0 = 0$	tối đa (0, 0) = 0
2	1	0	$(1 * 2) + (0 * -1) = 2 \ 2 + 0 + 0 =$	tối đa (0, 2) =
3	2	1	$2(2 * 2) + (1 * -1) = 3 \ 4 + -1 + 0 =$	2 tối đa (0, 3) = 3
4	3	2	$3(3 * 2) + (2 * -1) = 4 \ 6 + -2 + 0 =$	tối đa (0, 4) = 4
5	4	3	$4(4 * 2) + (3 * -1) = 5 \ 8 + -3 + 0 = 5$	tối đa (0, 5) = 5

Cột đầu tiên hiển thị đầu vào mạng ban đầu, x , sau là kết quả đầu ra của $n̄\text{-ron1}$ và $n̄\text{-ron2}$. Các cột còn lại cho biết cách $n̄\text{-ron3}$ xử lý các kết quả đầu ra: tổng trọng số được tính toán, độ lệch được thêm vào và cuối cùng ở cột cuối cùng, hàm kích hoạt ReLU được áp dụng để đạt được kết quả đầu ra của $n̄\text{-ron}$ và mạng cho mỗi giá trị đầu vào ban đầu cho x . Hình 10-8 cho thấy đồ thị chức năng của mạng.



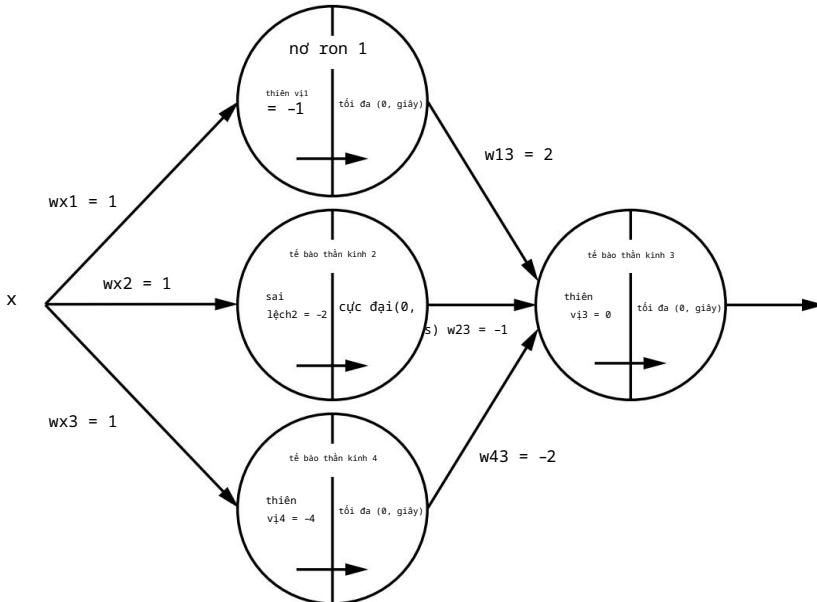
Hình 10-8: Trục quan hóa đầu vào mạng của chúng tôi và đầu ra liên quan

Chúng ta có thể thấy rằng thông qua sự kết hợp của các hàm đơn giản này, chúng ta có thể tạo ra một đồ thị đi lên bất kỳ khoảng thời gian hoặc độ dốc nào mong muốn trên các điểm khác nhau, như chúng ta đã làm trong Hình 10-8. Nói cách khác, chúng ta tiến gần hơn đến khả năng biểu diễn bất kỳ hàm hữu hạn nào cho đầu vào x của chúng ta!

Thêm một nơ-ron khác vào Mạng Chúng ta đã thấy

cách làm cho đồ thị chức năng của mạng đi lên (với bất kỳ độ dốc nào) bằng cách thêm các nơ-ron, nhưng chúng ta sẽ làm cho đồ thị đi xuống như thế nào?

Hãy thêm một nơ-ron khác (neuron4) vào hỗn hợp, như trong Hình 10-9.



Hình 10-9: Trục quan hóa mạng bốn nơ-ron nhỏ với một lớp ẩn duy nhất

Trong Hình 10-9, dữ liệu đầu vào x được gửi đến nơ-ron1, nơ-ron2 và nơ-ron4. Đầu ra của chúng sau đó được cung cấp làm đầu vào cho nơ-ron3, tạo ra đầu ra cuối cùng của mạng. Neuron4 giống như nơ-ron1 và nơ-ron2, nhưng với độ lệch được đặt thành -4. Bảng 10-5 tóm tắt đầu ra của nơ-ron4.

Bảng 10-5: Nơ-ron4

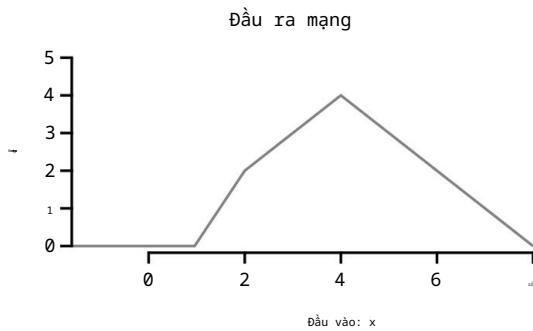
Đầu vào Tổng trọng số Tổng trọng số + sai số Đầu ra			
x	x * wx 4	(x * wx 4) + độ lệch4 max(0, (x * wx 4) + độ lệch4)	
0	0 * 1 = 0	0 + -4 = -4 max(0, -4) = 0	
1	1 * 1 = 1	1 + -4 = -3	tối đa(0, -3) = 0
2	2 * 1 = 2	2 + -4 = -2	tối đa(0, -2) = 0
3	3 * 1 = 3	3 + -4 = -1	tối đa(0, -1) = 0
4	4 * 1 = 4	4 + -4 = 0	tối đa(0, 0) = 0
5	5 * 1 = 5	5 + -4 = 1	tối đa(0, 1) = 1

Để làm cho biểu đồ mạng của chúng ta giảm dần, chúng ta trừ hàm của nơ-ron4 khỏi hàm của nơ-ron1 và nơ-ron2 trong tổng trọng số của nơ-ron3 bằng cách đặt trọng số kết nối nơ-ron4 với nơ-ron3 thành -2. Bảng 10-6 cho thấy đầu ra mới của toàn bộ mạng.

Bảng 10-6: Mạng bốn nơ-ron

Đầu vào mạng ban đầu	Đầu vào cho nơ-ron3	Tổng trọng lượng	Tổng trọng số + sai lệch	Đầu ra mạng cuối cùng
x	nơ-ron1 nơ-ron2 nơ-ron4 (nơ-ron1 * w1 3) + (nơ- ron2 * w2 3) + (nơ- ron4 * w4 3)	(nơ-ron1 * w1 3) + (nơ-ron2 * w2 3) + (nơ-ron4 * w4 3) + thiên vị3	(nơ-ron1 * w1 3) + (nơ-ron2 * w2 3) + (nơ-ron4 * w4 3) + sai lệch3	max(0, (nơ-ron1 * w1 3) + (nơ-ron2 * w2 3) + (nơ-ron4 * w4 3) + sai lệch3)
0	0 0 0	(0 * 2) + (0 * -1) + (0 * -2) = 0	0 + 0 + 0 + 0 = 0	tối đa(0, 0) = 0
1	0 0 0	(0 * 2) + (0 * -1) + (0 * -2) = 0	0 + 0 + 0 + 0 = 0	tối đa(0, 0) = 1
2	1 0 0	(1 * 2) + (0 * -1) + (0 * -2) = 2	2 + 0 + 0 + 0 = 2	cực đại(0, 2) = 2
3	2 1 0	(2 * 2) + (1 * -1) + (0 * -2) = 3	4 + -1 + 0 + 0 = 3	cực đại(0, 3) = 3
4	3 2 0	(3 * 2) + (2 * -1) + (0 * -2) = 4	6 + -2 + 0 + 0 = 4	tối đa(0, 4) = 4
5	4 3 1	(4 * 2) + (3 * -1) + (1 * -2) = 5	8 + -3 + -2 + 0 = 3	cực đại(0, 3) = 3

Hình 10-10 cho thấy điều này trông như thế nào.



Hình 10-10: Trục quan hóa mạng bốn nơ-ron của chúng tôi

Hy vọng, bây giờ bạn có thể thấy kiến trúc mạng thần kinh cho phép chúng ta di chuyển lên và xuống với bất kỳ tốc độ nào trên bất kỳ điểm nào trên biểu đồ, chỉ bằng cách kết hợp một số nơ-ron đơn giản (phổ quát!). Chúng ta có thể tiếp tục thêm nhiều nơ-ron hơn để tạo ra các chức năng phức tạp hơn nhiều.

Tạo tính năng tự động

Bạn đã học được rằng một mạng nơ-ron với một lớp ẩn duy nhất có thể xấp xỉ bất kỳ chức năng hữu hạn nào với đủ nơ-ron. Đó là một ý tưởng khá mạnh mẽ. Nhưng điều gì xảy ra khi chúng ta có nhiều lớp tế bào thần kinh ẩn? Nói tóm lại, quá trình tạo tính năng tự động xảy ra, đây có lẽ là một khía cạnh thậm chí còn mạnh mẽ hơn của mạng nơ-ron.

Trước đây, một phần quan trọng của quá trình xây dựng các mô hình học máy là trích xuất tính năng. Đối với tệp HTML, sẽ dành nhiều thời gian để quyết định khía cạnh số nào của tệp HTML (số tiêu đề phần, số từ duy nhất, v.v.) có thể hỗ trợ mô hình.

Mạng nơ-ron với nhiều lớp và khả năng tạo tính năng tự động cho phép chúng tôi giảm tải rất nhiều công việc đó. Nói chung, nếu bạn cung cấp các tính năng khái thô (chẳng hạn như ký tự hoặc từ trong tệp HTML) cho mạng nơ-ron, thì mỗi lớp nơ-ron có thể học cách biểu diễn các tính năng thô đó theo cách hoạt động tốt như đầu vào cho các lớp sau. Nói cách khác, một mạng lưới thần kinh sẽ học cách đếm số lần chữ cái a xuất hiện trong tài liệu HTML, nếu điều đó đặc biệt liên quan đến việc phát hiện phần mềm độc hại, mà không có đầu vào thực sự nào từ con người nói rằng nó có hay không.

Trong ví dụ về xe đạp xử lý hình ảnh của chúng tôi, không ai nói cụ thể với mạng rằng các tính năng meta của các cạnh hoặc bánh xe là hữu ích. Mô hình đã học được rằng những tính năng đó hữu ích khi làm đầu vào cho lớp nơ-ron tiếp theo trong quá trình đào tạo. Điều đặc biệt hữu ích là các tính năng đã học ở cấp độ thấp hơn này có thể được các lớp sau sử dụng theo những cách khác nhau, điều đó có nghĩa là mạng nơ-ron sâu có thể ước tính nhiều mẫu cực kỳ phức tạp bằng cách sử dụng ít nơ-ron và tham số hơn nhiều so với mạng một lớp.

Các mạng thần kinh không chỉ thực hiện nhiều công việc trích xuất tính năng mà trước đây rất nhiều thời gian và công sức, họ làm điều đó theo cách tối ưu hóa và tiết kiệm không gian, được hướng dẫn bởi quy trình đào tạo.

Đào tạo mạng lưới thần kinh

Cho đến nay, chúng ta đã khám phá làm thế nào, với một số lượng lớn các nơ-ron và các trọng số và số hạng sai lệch phù hợp, một mạng nơ-ron có thể xấp xỉ các chức năng phức tạp. Trong tất cả các ví dụ của chúng tôi cho đến nay, chúng tôi đặt các tham số trọng số và độ lệch đó theo cách thủ công. Tuy nhiên, vì các mạng nơ-ron thực thường chứa hàng nghìn nơ-ron và hàng triệu tham số, nên chúng ta cần một cách hiệu quả để tối ưu hóa các giá trị này.

Thông thường, khi huấn luyện một mô hình, chúng tôi bắt đầu với một tập dữ liệu huấn luyện và một mạng có nhiều tham số không được tối ưu hóa (khởi tạo ngẫu nhiên).

Đào tạo yêu cầu tối ưu hóa các tham số để giảm thiểu hàm mục tiêu.

Trong học có giám sát, nơi chúng tôi đang cố gắng đào tạo mô hình của mình để có thể dự đoán nhãn, chẳng hạn như 0 cho "lành tính" và 1 cho "phản mềm độc hại", hàm mục tiêu đó sẽ liên quan đến lỗi dự đoán của mạng trong quá trình đào tạo. Đối với một số đầu vào x nhất định (ví dụ: một tệp HTML cụ thể), đây là sự khác biệt giữa nhãn y mà chúng tôi biết là chính xác (ví dụ: 1.0 cho "là phản mềm độc hại") và đầu ra ^y chúng tôi nhận được từ mạng hiện tại (ví dụ: ví dụ, 0,7). Bạn có thể coi lỗi là sự khác biệt giữa nhãn dự đoán ^y và nhãn đúng y đã biết, trong đó mạng x ^y và mạng đang có ước lượng một hàm f chưa biết nào đó, sao cho $f(x) = y$. Nói cách khác, mạng ^f .

() =

Ý tưởng cơ bản dàn sau các mạng đào tạo là cung cấp cho mạng một người quan sát vation, x, từ tập dữ liệu huấn luyện của bạn, nhận một số đầu ra, ^y, và sau đó tìm hiểu xem việc thay đổi các tham số của bạn sẽ dịch chuyển ^y đến gần mục tiêu của bạn như thế nào, y. Hãy tưởng tượng bạn đang ở trong một con tàu vũ trụ với nhiều nút bấm khác nhau. Bạn không biết chức năng của từng nút, nhưng bạn biết hướng bạn muốn đi (y). Để giải quyết vấn đề, bạn nhấn ga và ghi lại hướng bạn đã đi (^y). Sau đó, bạn xoay một nút nhỏ và đạp ga một lần nữa. Sự khác biệt giữa hướng thứ nhất và hướng thứ hai cho bạn biết mức độ ảnh hưởng của nút đó đến hướng của bạn. Bằng cách này, cuối cùng bạn có thể tìm ra cách lái tàu vũ trụ khá tốt.

Đào tạo một mạng lưới thần kinh là tương tự. Đầu tiên, bạn cung cấp cho mạng một giá trị quan sát, x, từ tập dữ liệu huấn luyện của mình và bạn nhận được một số đầu ra, ^y. Bước này được gọi là lan truyền về phía trước bởi vì bạn đưa đầu vào x của mình về phía trước qua mạng để có được đầu ra cuối cùng ^y. Tiếp theo, bạn xác định mỗi tham số ảnh hưởng như thế nào đến đầu ra ^y của bạn. Ví dụ: nếu đầu ra mạng của bạn là 0,7, nhưng bạn biết đầu ra chính xác phải gần với 1 hơn, bạn có thể thử tăng một tham số, w, chỉ một chút, xem liệu ^y tiến gần hơn hay xa y hơn và bằng cách bao nhiêu.2 Đây được gọi là đạo hàm riêng của ^y theo w, hoặc ^y w.

/

Sau đó, tất cả các tham số trên toàn mạng được dịch chuyển một chút theo hướng làm cho ^y dịch chuyển gần y hơn một chút (và do đó mạng gần f hơn). Nếu ^y w dương, thì bạn biết bạn nên tăng w thêm một /

2. Trong thực tế, việc tăng tham số lên một chút và sau đó đánh giá lại kết quả đầu ra của mạng là không cần thiết. Điều này là do toàn bộ mạng là một hàm khả vi, có nghĩa là chúng ta chỉ có thể tính toán ($\hat{y} w$) một cách chính xác và nhanh chóng hơn bằng cách sử dụng phép tính. Tuy nhiên, tôi nhận thấy rằng suy nghĩ theo hướng thúc đẩy và đánh giá lại có xu hướng trực quan hơn là sử dụng phép tính đạo hàm.

một lượng nhỏ (cụ thể, tỷ lệ với y^w), sao cho new^y của bạn sẽ di chuyển một chút khỏi 0,7 và hướng tới 1 (y). Nói cách khác, bạn dạy mạng của mình tính gần đúng hàm f chưa biết bằng cách sửa sai của nó đối với dữ liệu huấn luyện với các nhãn đã biết.

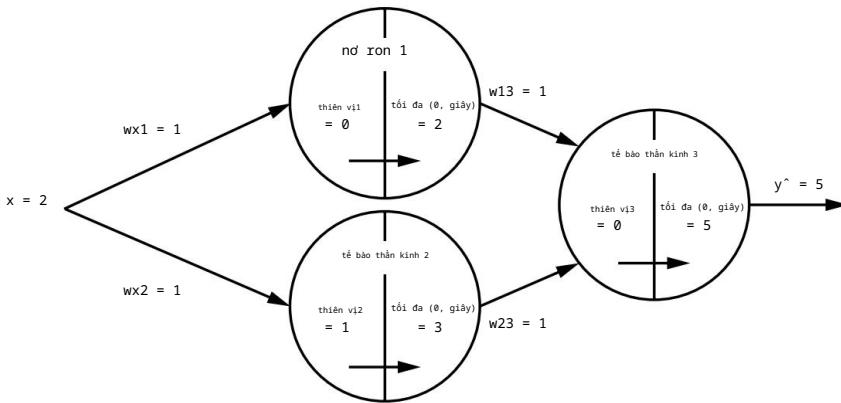
Quá trình tính lặp lại các đạo hàm riêng này, cập nhật tham số, và sau đó lặp lại được gọi là độ dốc gốc. Tuy nhiên, với một mạng gồm hàng nghìn nơ-ron, hàng triệu tham số và thường là hàng triệu quan sát huấn luyện, tất cả phép tính đó đều đòi hỏi rất nhiều tính toán.

Để giải quyết vấn đề này, chúng tôi sử dụng một thuật toán gọn gàng có tên là lan truyền ngược để làm cho các tính toán này khả thi về mặt tính toán. Về cốt lõi, lan truyền ngược cho phép chúng ta tính toán đạo hàm riêng một cách hiệu quả dọc theo các đồ thị tính toán giống như một mạng lưới thần kinh!

Sử dụng Backpropagation để tối ưu hóa mạng lưới thần kinh

Trong phần này, chúng tôi xây dựng một mạng lưới thần kinh đơn giản để giới thiệu cách thức hoạt động của lan truyền ngược. Giả sử rằng chúng ta có một ví dụ đào tạo có giá trị là $x = 2$ và nhãn thực được liên kết là $y = 10$. Thông thường, x sẽ là một mảng gồm nhiều giá trị, nhưng hãy sử dụng một giá trị duy nhất để đơn giản hóa mọi thứ.

Cắm các giá trị này vào, chúng ta có thể thấy trong Hình 10-11 rằng đầu ra mạng của chúng ta đặt giá trị y là 5 với giá trị x đầu vào là 2.



Hình 10-11: Trực quan hóa mạng ba nơ-ron của chúng tôi, với đầu vào là $x = 2$

Để dịch chuyển các tham số sao cho đầu ra của mạng \hat{y} , với $x = 2$, tiến gần hơn đến giá trị y đã biết là 10, chúng ta cần tính toán cách thức $w1 \cdot 3$ ành hưởng đến đầu ra cuối cùng của chúng tôi \hat{y} . Hãy xem điều gì xảy ra khi chúng ta tăng $w1 \cdot 3$ chỉ một chút (giả sử, $0,01$). Tổng trọng số trong nơ-ron3 trở thành $1,01 \cdot 2 + (1 \cdot 3)$, làm cho đầu ra cuối cùng \hat{y} thay đổi từ 5 thành 5,02, dẫn đến tăng $0,02$. Nói cách khác, đạo hàm riêng của \hat{y} đối với $w1 \cdot 3$ là 2, bởi vì thay đổi $w1 \cdot 3$ mang lại gấp đôi thay đổi \hat{y} .

Bởi vì y là 10 và đầu ra hiện tại của chúng ta \hat{y} (với các giá trị tham số hiện tại của chúng ta và $x = 2$) là 5, bây giờ chúng ta biết rằng chúng ta nên tăng $w1 \cdot 3$ một lượng nhỏ để di chuyển y đến gần 10 hơn.

Điều đó khá đơn giản. Nhưng chúng ta cần có khả năng biết hướng nào để đầy tất cả các tham số trong mạng của mình, không chỉ các tham số trong nơ-ron ở lớp cuối cùng. Ví dụ, còn wx_1 thì sao? Tính toán \hat{y} wx_1 phức tạp hơn vì nó chỉ ảnh hưởng gián tiếp đến \hat{y} . Đầu tiên, chúng tôi hỏi chức năng của nơ-ron3 bị ảnh hưởng như thế nào bởi đầu ra của nơ-ron1. Nếu chúng ta thay đổi đầu ra của nơ-ron1 từ 2 đến 2,01, đầu ra cuối cùng của nơ-ron3 thay đổi từ 5 thành 5,01, vì vậy \hat{y} nơ-ron1 với mức độ wx_1 ảnh hưởng đến đầu ra của nơ-ron1. Nếu chúng ta thay đổi wx_1 từ 1 thành 1,01, đầu ra của nơ-ron1 thay đổi từ 2 thành 2,02, 1 wx bằng nơ-ron1. Do đó:

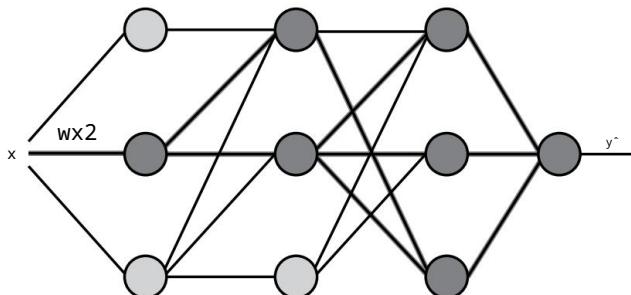
$$\frac{\hat{y}}{wx_1} \quad \frac{\hat{y}}{t\hat{e} b\hat{o} t\hat{h}\hat{a}n k\hat{inh}_1} * \frac{t\hat{e} b\hat{o} t\hat{h}\hat{a}n k\hat{inh}_1}{wx_1}$$

Hoặc:

$$\frac{\hat{y}}{wx_1} \quad 1^* 2$$

Bạn có thể nhận thấy rằng chúng tôi vừa sử dụng quy tắc dây chuyền.3

Nói cách khác, để tìm hiểu xem một tham số như wx_1 sâu bên trong mạng ảnh hưởng như thế nào đến đầu ra cuối cùng \hat{y} của chúng ta, chúng ta nhân các đạo hàm riêng tại mỗi điểm dọc theo đường dẫn giữa tham số wx_1 và \hat{y} . Điều này có nghĩa là nếu wx_1 được đưa vào một nơ-ron mà đầu ra của nó được đưa vào mồi nơ-ron khác, thì việc tính toán tác động của wx_1 lên \hat{y} sẽ liên quan đến việc tính tổng tất cả các đường dẫn từ wx_1 to \hat{y} , thay vì chỉ một. Hình 10-12 trực quan hóa các đường dẫn bị ảnh hưởng bởi tham số trọng lượng wx_2 .



Hình 10-12: Trực quan hóa các đường dẫn bị ảnh hưởng bởi wx_2 (hiển thị bằng màu xám đậm): trọng số liên quan đến kết nối giữa dữ liệu đầu vào x và nơ-ron ở giữa trong lớp đầu tiên (ngoài cùng bên trái)

3. Quy tắc dây chuyền là công thức tính đạo hàm của hàm hợp. Ví dụ: nếu f và g đều là các hàm và h là hàm hợp $h(x) = f(g(x))$, thì quy tắc dây chuyền phát biểu rằng $h'(x) = f'(g(x)) * g'(x)$, trong đó $f'(x)$ biểu thị đạo hàm riêng của một hàm, f , đối với x .

Lưu ý rằng các lớp ẩn trong mạng này không phải là các lớp được kết nối đầy đủ, điều này giúp giải thích tại sao nơ-ron dưới cùng của lớp ẩn thứ hai không được tô sáng.

Vụ nổ đường dẫn

Nhưng điều gì sẽ xảy ra khi mạng của chúng ta thậm chí còn lớn hơn? Số lượng đường dẫn chúng ta cần thêm để tính đạo hàm riêng của tham số cấp thấp tăng theo cấp số nhân. Hãy xem xét một nơ-ron có đầu ra được đưa vào một lớp gồm 1.000 nơ-ron, đầu ra của lớp này được đưa vào 1.000 nơ-ron khác, đầu ra của chúng sau đó được đưa vào nơ-ron đầu ra cuối cùng.

Điều đó dẫn đến một triệu con đường! May mắn thay, đi qua từng đường dẫn đơn lẻ và sau đó tính tổng chúng để lấy tham số \hat{y} là không cần thiết. Đây là nơi backpropagation có ích. Thay vì đi dọc theo từng con đường đơn lẻ dẫn đến (các) đầu ra cuối cùng của chúng ta, \hat{y} , các đạo hàm riêng được tính toán theo từng lớp, bắt đầu từ trên xuống hoặc ngược lại.

Sử dụng logic quy tắc dây chuyền từ phần trước, chúng ta có thể tính toán bất kỳ đạo hàm riêng \hat{y} w, trong đó w là tham số kết nối đầu ra từ layeri-1 đến một nơ-ron trong layeri, bằng cách tính tổng các giá trị sau cho tất cả nơ-ron+1, trong đó mỗi nơ-ron+1 là một nơ-ron trong layeri+1 nơ-ron đó (nơ-ron của w) được kết nối:

$$\hat{y} = \frac{\text{tổng}_1}{\text{tổng}_2} * \frac{\text{tổng}_2}{\text{tổng}_3} * \dots * \frac{\text{tổng}_n}{w}$$

Bằng cách thực hiện từng lớp một từ trên xuống, chúng tôi hạn chế sự bùng nổ đường dẫn bằng cách hợp nhất các đạo hàm ở mỗi lớp. Nói cách khác, các đạo hàm được tính toán trong layeri+1 cấp cao nhất (như \hat{y} /nơ-ron 1) được ghi lại để giúp tính toán các đạo hàm trong layeri. Sau đó, để tính các đạo hàm trong layeri-1, chúng ta sử dụng các đạo hàm đã lưu từ layeri (như \hat{y} neuron). Sau đó, layeri-2 sử dụng các dẫn xuất từ layeri-1, v.v. Thủ thuật này giúp giảm đáng kể số lượng phép tính mà chúng tôi phải lặp lại và giúp chúng tôi huấn luyện mạng lưới thần kinh một cách nhanh chóng.

Độ dốc biến mất

Một vấn đề mà các mạng thần kinh rất sâu phải đối mặt là vấn đề độ dốc biến mất. Hãy xem xét một tham số trọng số trong lớp đầu tiên của mạng thần kinh có mười lớp. Tín hiệu mà nó nhận được từ lan truyền ngược là tổng của tất cả các tín hiệu của đường dẫn từ nơ-ron của trọng số này đến đầu ra cuối cùng.

Vấn đề là tín hiệu của mỗi đường dẫn có thể cực kỳ nhỏ, bởi vì chúng tôi tính toán tín hiệu đó bằng cách nhân các đạo hàm riêng tại mỗi điểm dọc theo đường dẫn có độ sâu 10 nơ-ron, tất cả đều có xu hướng là các số nhỏ hơn 1. Điều này có nghĩa là một Các tham số của nơ-ron cấp thấp được cập nhật dựa trên tổng của một số lượng lớn các số rất nhỏ, nhiều số trong số đó sẽ triệt tiêu lẫn nhau. Do đó, mạng có thể khó phổi hợp gửi tín hiệu mạnh xuống

đến các tham số ở các lớp thấp hơn. Vấn đề này trở nên tồi tệ hơn theo cấp số nhân khi bạn thêm nhiều lớp hơn. Khi bạn tìm hiểu trong phần sau, một số thiết kế mạng có gắng giải quyết vấn đề phổ biến này.

Các loại mạng thần kinh

Để đơn giản, mọi ví dụ mà tôi đã chỉ cho bạn cho đến nay đều sử dụng một loại công việc mạng được gọi là mạng thần kinh chuyển tiếp nguồn cấp dữ liệu. Trong thực tế, có nhiều cấu trúc mạng hữu ích khác mà bạn có thể sử dụng cho các loại bài toán khác nhau. Hãy thảo luận về một số loại mạng thần kinh phổ biến nhất và cách chúng có thể được áp dụng trong bối cảnh an ninh mạng.

Mạng thần kinh chuyển tiếp nguồn cấp dữ liệu

Loại mạng nơ-ron đơn giản nhất (và đầu tiên), mạng nơ-ron chuyển tiếp nguồn cấp dữ liệu, giống như một con búp bê Barbie không có phụ kiện: các loại mạng nơ-ron khác thường chỉ là các biến thể của cấu trúc “mặc định” này. Kiến trúc chuyển tiếp nguồn cấp dữ liệu nghe có vẻ quen thuộc: nó bao gồm các chồng lớp nơ-ron. Mỗi lớp té bào thần kinh được kết nối với một số hoặc tất cả các té bào thần kinh trong lớp tiếp theo, nhưng các kết nối không bao giờ đi lùi hoặc hình thành các chu kỳ, do đó có tên là “feed forward”.

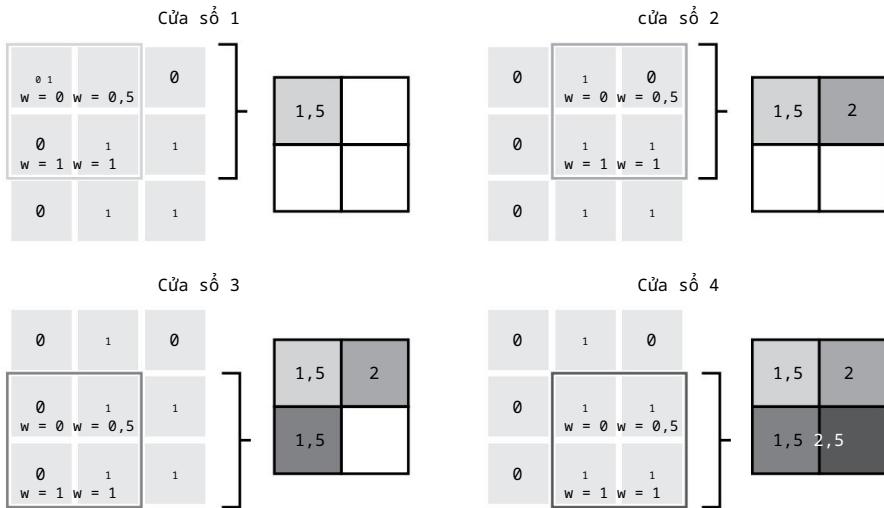
Trong các mạng nơ-ron chuyển tiếp nguồn cấp dữ liệu, mọi kết nối tồn tại là kết nối một nơ-ron (hoặc đầu vào ban đầu) trong lớp i với một nơ-ron trong lớp $j > i$. Mỗi nơ-ron trong lớp i không nhất thiết phải kết nối với mọi nơ-ron trong lớp $i + 1$, nhưng tất cả các kết nối phải được truyền về phía trước, kết nối các lớp trước với các lớp sau.

Các mạng chuyển tiếp nguồn cấp dữ liệu nói chung là loại mạng mà bạn giải quyết vấn đề trước tiên, trừ khi bạn đã biết về một kiến trúc khác hoạt động đặc biệt tốt cho vấn đề hiện tại (chẳng hạn như mạng nơ-ron tích chập hoạt động để nhận dạng hình ảnh).

Mạng thần kinh tích chập

Mạng thần kinh tích chập (CNN) chứa các lớp tích chập, trong đó đầu vào cung cấp cho mỗi nơ-ron được xác định bởi một cửa sổ trượt trên không gian đầu vào. Hãy tưởng tượng một cửa sổ hình vuông nhỏ trượt trên một bức tranh lớn hơn, nơi chỉ các pixel nhìn thấy được qua cửa sổ sẽ được kết nối với một nơ-ron cụ thể trong lớp tiếp theo. Sau đó, cửa sổ trượt và tập hợp các pixel mới được kết nối với một nơ-ron mới. Hình 10-13 minh họa điều này.

Cấu trúc của các mạng này khuyến khích việc học các tính năng được bản địa hóa. Ví dụ: các lớp thấp hơn của mạng tập trung vào mối quan hệ giữa các pixel lân cận trong một hình ảnh (tạo thành các cạnh, hình dạng, v.v.) sẽ hữu ích hơn là tập trung vào mối quan hệ giữa các pixel phân tán ngẫu nhiên trên một hình ảnh (là đường như không có nhiều ý nghĩa). Các cửa sổ trượt buộc rõ ràng phải tập trung vào điều này, giúp cải thiện và tăng tốc độ học tập ở những khu vực mà việc trích xuất tính năng cục bộ là đặc biệt quan trọng.



Hình 10-13: Trực quan hóa cửa sổ tích chập 2×2 trượt trên không gian đầu vào 3×3 với bước (kích thước bước) là 1, để tạo ra đầu ra 2×2

Do khả năng tập trung vào các phần được bắn địa hóa của dữ liệu đầu vào, các mạng thần kinh tích chập cực kỳ hiệu quả trong việc nhận dạng và phân loại hình ảnh. Chúng cũng đã được chứng minh là có hiệu quả đối với một số loại xử lý ngôn ngữ tự nhiên, điều này có ý nghĩa đối với an ninh mạng.

Sau khi các giá trị của mỗi cửa sổ tích chập được cung cấp cho các nơ-ron cụ thể trong một lớp tích chập, một cửa sổ trượt lại được trượt trên các đầu ra của các nơ-ron này, nhưng thay vì chúng được đưa đến các nơ-ron tiêu chuẩn (ví dụ: ReLU) với các trọng số được liên kết với mỗi đầu vào, chúng được cung cấp cho các nơ-ron không có trọng số (nghĩa là cố định ở mức 1) và chức năng kích hoạt tối đa (hoặc tương tự). Nói cách khác, một cửa sổ nhỏ được trượt trên đầu ra của lớp tích chập và giá trị tối đa của mỗi cửa sổ được lấy và chuyển sang lớp tiếp theo. Đây được gọi là một lớp tổng hợp. Mục đích của việc gộp các lớp là để "thu nhỏ" dữ liệu (thường là hình ảnh), do đó giảm kích thước của các đối tượng địa lý để tính toán nhanh hơn trong khi vẫn giữ được thông tin quan trọng nhất.

Các mạng thần kinh tích chập có thể có một hoặc nhiều tập hợp các lớp tổng hợp và phức tạp. Một kiến trúc tiêu chuẩn có thể bao gồm một lớp chập, một lớp tổng hợp, theo sau là một tập hợp các lớp tích chập và gộp khác, và cuối cùng là một vài lớp được kết nối đầy đủ, như trong các mạng chuyển tiếp nguồn cấp dữ liệu. Mục tiêu của kiến trúc này là các lớp được kết nối đầy đủ cuối cùng này nhận các tính năng cấp cao làm đầu vào (nghĩ rằng bánh xe trên xe đạp một bánh) và kết quả là có thể phân loại chính xác dữ liệu phức tạp (chẳng hạn như hình ảnh).

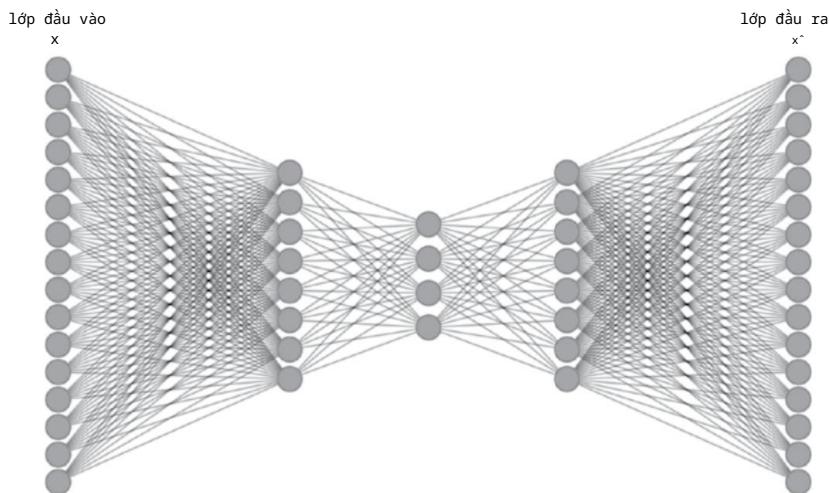
Mạng thần kinh mã hóa tự động

Bộ mã hóa tự động là một loại mạng nơ-ron có găng nén và sau đó giải nén đầu vào với sự khác biệt tối thiểu giữa đầu vào huấn luyện ban đầu và đầu ra được giải nén. Mục tiêu của bộ mã hóa tự động là học

một đại diện hiệu quả cho một tập hợp dữ liệu. Nói cách khác, bộ mã hóa tự động hoạt động giống như các chương trình nén mất dữ liệu được tối ưu hóa, trong đó chúng nén dữ liệu đầu vào thành một biểu diễn nhỏ hơn, sau đó giải nén dữ liệu đó trở lại kích thước đầu vào ban đầu.

Thay vì mạng thần kinh tối ưu hóa các tham số bằng cách giảm thiểu sự khác biệt giữa nhãn đã biết (y) và nhãn dự đoán (\hat{y}) đối với đầu vào x đã cho, mạng sẽ cố gắng giảm thiểu sự khác biệt giữa đầu vào ban đầu x và đầu ra được xây dựng lại \hat{x} .

Về mặt cấu trúc, bộ mã hóa tự động thường rất giống với mạng nơ-ron chuyển tiếp nguồn cấp dữ liệu tiêu chuẩn, ngoại trừ các lớp giữa chứa ít nơ-ron hơn so với các lớp ở giai đoạn đầu và sau, như thể hiện trong Hình 10-14.



Hình 10-14: Trục quan hóa mạng bộ mã hóa tự động

Như bạn có thể thấy, lớp ở giữa nhỏ hơn nhiều so với lớp ngoài cùng bên trái (đầu vào) và ngoài cùng bên phải (đầu ra), mỗi lớp có cùng kích thước. Lớp cuối cùng phải luôn chứa cùng số lượng đầu ra như đầu vào ban đầu, vì vậy mỗi đầu vào đào tạo xi có thể được so sánh với người anh em họ \hat{x} được nén và tái tạo của nó.

Sau khi mạng mã hóa tự động được đào tạo, nó có thể được sử dụng cho các mục đích khác nhau. Các mạng bộ mã hóa tự động có thể được sử dụng đơn giản như các chương trình nén/giải nén hiệu quả. Ví dụ: bộ mã hóa tự động được đào tạo để nén các tệp hình ảnh có thể tạo ra hình ảnh trông rõ ràng hơn nhiều so với cùng một hình ảnh được nén qua JPEG với cùng kích thước.

Mạng đối thủ sáng tạo

Mạng đối thủ chung (GAN) là một hệ thống gồm hai mạng thần kinh cạnh tranh với nhau để cải thiện bản thân trong các nhiệm vụ tương ứng.

Thông thường, mạng tổng quát cố gắng tạo các mẫu giả (ví dụ: một số loại hình ảnh) từ nhiều ngẫu nhiên. Sau đó, một mạng phân biệt đối xử thứ hai

có gắng phân biệt giữa mẫu thật và mẫu giả, được tạo ra (ví dụ: phân biệt giữa hình ảnh thực của phòng ngủ và hình ảnh được tạo ra).

Cả hai mạng thần kinh trong GAN đều được tối ưu hóa với lan truyền ngược. Mạng trình tạo tối ưu hóa các tham số của nó dựa trên mức độ nó đánh lừa mạng phân biệt đối xử trong một vòng nhất định, trong khi mạng phân biệt đối xử tối ưu hóa các tham số của nó dựa trên mức độ chính xác mà nó có thể phân biệt giữa các mẫu được tạo và mẫu thực. Nói cách khác, các hàm mất mát của chúng đối lập trực tiếp với nhau.

GAN có thể được sử dụng để tạo dữ liệu giống như thật hoặc nâng cao mức thấp chất lượng hoặc dữ liệu bị hỏng.

Mạng thần kinh tái phát

Mạng hồi quy (RNN) là một loại mạng nơ-ron tương đối rộng, trong đó các kết nối giữa các nơ-ron tạo thành các chu kỳ có hướng có chức năng kích hoạt phụ thuộc vào các bước thời gian. Điều này cho phép mạng phát triển một bộ nhớ, giúp nó học các mẫu trong chuỗi dữ liệu. Trong RNN, đầu vào, đầu ra hoặc cả đầu vào và đầu ra là một số loại chuỗi thời gian.

RNN rất phù hợp cho các tác vụ có thứ tự dữ liệu quan trọng, như nhận dạng chữ viết tay được kết nối, nhận dạng giọng nói, dịch ngôn ngữ và phân tích chuỗi thời gian. Trong bối cảnh an ninh mạng, chúng có liên quan đến các vấn đề như phân tích lưu lượng mạng, phát hiện hành vi và phân tích tệp tinh. Vì mã chương trình tương tự như ngôn ngữ tự nhiên theo thứ tự quan trọng nên nó có thể được coi là một chuỗi thời gian.

Một vấn đề với RNN là do vấn đề độ dốc biến mất, mỗi bước thời gian được giới thiệu trong RNN tương tự như toàn bộ lớp bổ sung trong mạng thần kinh chuyển tiếp nguồn cấp dữ liệu. Trong quá trình lan truyền ngược, vấn đề độ dốc biến mất khiến tín hiệu ở các lớp cấp thấp hơn (hoặc trong trường hợp này là các bước thời gian trước đó) trở nên vô cùng mờ nhạt.

Mạng bộ nhớ ngắn hạn dài (LSTM) là một loại RNN đặc biệt được thiết kế để giải quyết vấn đề này. LSTM chứa các tế bào bộ nhớ và tế bào thần kinh đặc biệt có gắng quyết định thông tin nào cần nhớ và thông tin nào cần quên.

Việc loại bỏ hầu hết thông tin sẽ hạn chế rất nhiều vấn đề độ dốc biến mất vì nó làm giảm sự bùng nổ đường dẫn.

ResNet

ResNet (viết tắt của mạng còn lại) là một loại mạng thần kinh tạo ra các kết nối bỏ qua giữa các nơ-ron ở các lớp đầu/nhông của mạng đến các lớp sâu hơn bằng cách bỏ qua một hoặc nhiều lớp trung gian. Ở đây, từ thặng dư đề cập đến thực tế là các mạng này học cách truyền thông tin dạng số trực tiếp giữa các lớp mà thông tin dạng số đó không cần phải truyền qua các loại hàm kích hoạt mà chúng tôi đã minh họa trong Bảng 10-1.

Cấu trúc này giúp giảm đáng kể vấn đề độ dốc biến mất, cho phép ResNets cực kỳ sâu-dài khi hơn 100 lớp.

Các mạng thần kinh rất sâu vượt trội trong việc mô hình hóa các mối quan hệ cực kỳ phức tạp, kỳ quặc trong dữ liệu đầu vào. Vì ResNets có thể có rất nhiều lớp nên chúng đặc biệt phù hợp với các vấn đề phức tạp. Giống như các mạng thần kinh chuyển tiếp nguồn cấp dữ liệu, ResNets quan trọng hơn vì tính hiệu quả chung của chúng trong việc giải quyết các vấn đề phức tạp hơn là chuyên môn của chúng trong các lĩnh vực vấn đề rất cụ thể.

Bản tóm tắt

Trong chương này, bạn đã học về cấu trúc của các nơ-ron và cách chúng được kết nối với nhau để tạo thành mạng nơ-ron. Bạn cũng đã khám phá cách các mạng này được đào tạo thông qua lan truyền ngược và bạn đã phát hiện ra một số lợi ích và vấn đề mà mạng thần kinh gặp phải, chẳng hạn như tính phổ quát, tạo tính năng tự động và vấn đề độ dốc biến mất. Cuối cùng, bạn đã học được cấu trúc và lợi ích của một số loại mạng lưới thần kinh phổ biến.

Trong chương tiếp theo, bạn sẽ thực sự xây dựng mạng lưới thần kinh để phát hiện lỗi ware, sử dụng gói Keras của Python.

11

Xây dựng mạng lưới N eural N
Trình phát hiện phần mềm độc hại với K era s



Một thập kỷ trước, việc xây dựng một mạng lưới thần kinh hoạt động, có khả năng mở rộng và nhanh chóng đã đến lúc tổng hợp và yêu cầu khá nhiều mã. Tuy nhiên, trong vài năm qua, quá trình này đã trở nên ít khó khăn hơn, khi ngày càng có nhiều giao diện cấp cao cho thiết kế mạng thần kinh được phát triển.

Gói Python Keras là một trong những giao diện này.

Trong chương này, tôi sẽ hướng dẫn bạn cách xây dựng một mạng thần kinh mẫu bằng gói Keras. Đầu tiên, tôi giải thích cách xác định kiến trúc của mô hình trong Keras. Thứ hai, chúng tôi đào tạo mô hình này để phân biệt giữa các tệp HTML lành tính và độc hại, đồng thời bạn tìm hiểu cách lưu và tải các mô hình đó. Thứ ba, bằng cách sử dụng gói Python sklearn, bạn sẽ học cách đánh giá độ chính xác của mô hình đối với dữ liệu xác thực. Cuối cùng, chúng tôi sử dụng những gì bạn đã học để tích hợp báo cáo độ chính xác của xác thực vào quy trình đào tạo mô hình.

Tôi khuyến khích bạn đọc chương này trong khi đọc và chỉnh sửa mã liên quan trong dữ liệu đi kèm cuốn sách này. Bạn có thể tìm thấy tất cả các mã

thảo luận trong chương này ở đó (được tổ chức thành các chức năng được tham số hóa để giúp mọi thứ dễ chạy và điều chỉnh hơn), cũng như một vài ví dụ bổ sung. Khi kết thúc chương này, bạn sẽ cảm thấy sẵn sàng bắt đầu xây dựng một số mạng lưới của riêng mình!

Để chạy danh sách mã trong chương này, bạn không chỉ cần cài đặt các gói được liệt kê trong tệp ch11/requirements.txt của chương này (`pip install -r tests.txt`), mà còn làm theo hướng dẫn để cài đặt một trong các công cụ phụ trợ của Keras trên máy tính của bạn. hệ thống (TensorFlow, Theano hoặc CNTK). Cài đặt TensorFlow bằng cách làm theo hướng dẫn tại đây: <https://www.tensorflow.org/> cài đặt/.

Xác định kiến trúc của một mô hình

Để xây dựng một mạng thần kinh, bạn cần xác định kiến trúc của nó: nơ-ron nào đi đến đâu, cách chúng kết nối với các nơ-ron tiếp theo và cách dữ liệu truyền qua toàn bộ. May mắn thay, Keras cung cấp một giao diện đơn giản, linh hoạt để xác định tất cả những điều này. Máy ảnh thực sự hỗ trợ hai cú pháp tương tự để định nghĩa mô hình, nhưng chúng tôi sẽ sử dụng cú pháp API chức năng, vì nó linh hoạt và mạnh mẽ hơn cú pháp ("tuần tự") kia.

Khi thiết kế một mô hình, bạn cần ba thứ: đầu vào, thứ ở giữa xử lý đầu vào và đầu ra. Đôi khi các mô hình của bạn sẽ có nhiều đầu vào, nhiều đầu ra và những thứ rất phức tạp ở giữa, nhưng ý tưởng cơ bản là khi xác định kiến trúc của một mô hình, bạn chỉ đang xác định cách đầu vào-dữ liệu của bạn, chẳng hạn như các tính năng liên quan đến HTML file –chảy qua nhiều nơ-ron khác nhau (thứ ở giữa), cho đến khi cuối cùng nơ-ron cuối cùng tạo ra một số đầu ra.

Để xác định kiến trúc này, Keras sử dụng các lớp. Một lớp là một nhóm các nơ-ron thần kinh đều sử dụng cùng một loại chức năng kích hoạt, tất cả đều nhận dữ liệu từ một lớp trước đó và tất cả đều gửi kết quả đầu ra của chúng đến một lớp nơ-ron tiếp theo. Trong một mạng nơ-ron, dữ liệu đầu vào thường được cung cấp cho một lớp nơ-ron ban đầu, lớp này gửi kết quả đầu ra của nó tới lớp tiếp theo, lớp này gửi kết quả đầu ra của nó tới một lớp khác, v.v., cứ tiếp tục như vậy cho đến khi lớp nơ-ron cuối cùng tạo ra đầu ra cuối cùng của mạng.

Liệt kê 11-1 là một ví dụ về một mô hình đơn giản được xác định bằng cách sử dụng cú pháp API chức năng của Keras . Tôi khuyến khích bạn mở một tệp Python mới để tự viết và chạy mã khi chúng ta xem qua mã, từng dòng một. Ngoài ra, bạn có thể thử chạy mã được liên kết trong dữ liệu đi kèm cuốn sách này, bằng cách sao chép và dán các phần của tệp ch11/model_architecture.py vào phiên ipython hoặc bằng cách chạy python ch11/model_architecture.py trong cửa sổ cuối.

từ các lớp nhập máy ảnh
từ keras.models nhập Mô hình

```
đầu vào = lớp.Input(shape=(1024,), dtype='float32')
giữa = lớp.Dense(units=512, activation='relu')(input)
đầu ra = layer.Dense(units=1, activation='sigmoid')(giữa)
```

```
mô hình = Mô hình(đầu vào=đầu vào, đầu ra=đầu ra)
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Liệt kê 11-1: Định nghĩa một mô hình đơn giản sử dụng cú pháp hàm API

Đầu tiên, chúng ta nhập mô hình con lớp của gói Keras cũng như lớp Model từ mô hình con mô hình của Keras .

Tiếp theo, chúng tôi chỉ định loại dữ liệu mà mô hình này sẽ chấp nhận cho một lần quan sát bằng cách chuyển một giá trị hình dạng (một bộ số nguyên) và một loại dữ liệu (chuỗi) cho hàm layers.Input() . Ở đây, chúng tôi đã tuyên bố rằng dữ liệu đầu vào cho mô hình của chúng tôi sẽ là một mảng gồm 1.024 số float. Ví dụ, nếu đầu vào của chúng ta là một ma trận các số nguyên, thì dòng đầu tiên sẽ giống như input = Input(shape=(100, 100,) dtype='int32') .

Lưu ý Mô hình nhận đầu vào có kích thước thay đổi trên một chiều, bạn có thể sử dụng Không có thay vì một số ví dụ: (100, Không,) .

Tiếp theo, chúng tôi chỉ định lớp tế bào thần kinh mà dữ liệu đầu vào này sẽ được gửi đến. Để thực hiện việc này, chúng ta lại sử dụng mô-đun con của các lớp mà chúng ta đã nhập, cụ thể là hàm Dense , để xác định rằng lớp này sẽ là một lớp được kết nối dày đặc (còn gọi là được kết nối dày dặn), có nghĩa là mọi đầu ra từ lớp trước được gửi đến mọi nơ-ron trong lớp này. Dày đặc là loại lớp phổ biến nhất mà bạn có thể sẽ sử dụng khi phát triển các mô hình Keras . Những thứ khác cho phép bạn thực hiện những việc như thay đổi hình dạng của dữ liệu (Reshape) và triển khai lớp tùy chỉnh của riêng bạn (Lambda) .

Chúng tôi chuyển cho hàm Dense hai đối số: units=512, để xác định rằng chúng tôi muốn có 512 nơ-ron trong lớp này và activation='relu' , để xác định rằng chúng tôi muốn các nơ-ron này là nơ-ron đơn vị tuyến tính được chỉnh lưu (ReLU) . (Nhớ lại ở Chương 10 rằng các nơ-ron ReLU sử dụng một loại chức năng kích hoạt đơn giản cho kết quả đầu ra bất kỳ giá trị nào lớn hơn: 0 hoặc tổng trọng số của các đầu vào của nơ-ron.) Chúng tôi sử dụng các lớp.Dense(units=512, activation='relu') để xác định lớp, và sau đó là phần cuối cùng của dòng-(đầu vào)-khai báo đầu vào cho lớp này (cụ thể là đổi tượng đầu vào của chúng ta) . Điều quan trọng là phải hiểu rằng việc chuyển đầu vào này đến lớp của chúng ta là cách xác định luồng dữ liệu trong mô hình, trái ngược với thứ tự của các dòng mã.

Trong dòng tiếp theo, chúng tôi xác định lớp đầu ra của mô hình, lớp này lại sử dụng hàm Dense . Nhưng lần này, chúng tôi chỉ chỉ định một nơ-ron duy nhất cho lớp và sử dụng hàm kích hoạt 'sigmoid' , rất phù hợp để kết hợp nhiều dữ liệu thành một điểm duy nhất trong khoảng từ 0 đến 1. Lớp đầu ra lấy đối tượng (ở giữa) làm đầu vào, tuyên bố rằng tất cả các đầu ra từ 512 nơ-ron ở lớp giữa của chúng ta sẽ được gửi đến nơ-ron này.

Bây giờ chúng ta đã xác định các lớp của mình, chúng ta sử dụng lớp Model từ các mô hình mô hình con để bao bọc tất cả các lớp này lại với nhau như một mô hình . Lưu ý rằng bạn chỉ phải chỉ định (các) lớp đầu vào và (các) lớp đầu ra của mình. Bởi vì mỗi lớp sau lớp đầu tiên được cung cấp cho lớp trước làm đầu vào, đầu ra cuối cùng

lớp chứa tất cả thông tin mà mô hình cần về các lớp trước đó.

Chúng tôi có thể có thêm 10 lớp trung gian được khai báo giữa đầu vào và đầu ra của chúng tôi lớp, nhưng dòng mã tại sẽ không thay đổi.

Biên soạn mô hình

Cuối cùng, chúng ta cần biên dịch mô hình của mình. Chúng tôi đã xác định kiến trúc của mô hình và luồng dữ liệu, nhưng chúng tôi vẫn chưa chỉ định cách chúng tôi muốn mô hình thực hiện đào tạo của nó. Để làm điều này, chúng tôi sử dụng phương thức biên dịch riêng của mô hình và truyền cho nó ba tham số:

- Tham số đầu tiên, trình tối ưu hóa , chỉ định loại thuật toán lan truyền ngược sẽ sử dụng. Bạn có thể chỉ định tên của thuật toán mà bạn muốn sử dụng thông qua một chuỗi ký tự như chúng tôi đã làm ở đây hoặc bạn có thể nhập một thuật toán thuật toán trực tiếp từ keras.optimizers để chuyển các tham số cụ thể cho thuật toán hoặc thậm chí thiết kế thuật toán của riêng bạn.
- Tham số mắt mèo chỉ định thử được giảm thiểu trong quá trình quá trình huấn luyện (lan truyền ngược). Cụ thể, điều này chỉ định công thức bạn muốn sử dụng để thể hiện sự khác biệt giữa nhãn đào tạo thực sự của bạn và nhãn dự đoán của mô hình (đầu ra). Một lần nữa, bạn có thể chỉ định tên của hàm mắt mèo hoặc chuyển vào một hàm thực tế, chẳng hạn như keras.losses.mean_squared_error.
- Cuối cùng, đối với tham số chỉ số , bạn có thể chuyển danh sách các chỉ số bạn muốn Keras báo cáo khi phân tích hiệu suất của mô hình trong và sau khi đào tạo. Một lần nữa, bạn có thể chuyển chuỗi hoặc hàm số liệu thực tế, chẳng hạn như ['categorical_accuracy', keras.metrics.top_k_categorical_accuracy].

Sau khi chạy mã trong Liệt kê 11-1, hãy chạy model.summary() để xem cấu trúc mô hình được in ra màn hình của bạn. Đầu ra của bạn sẽ giống như Hình 11-1.

In [2]: model.summary()		
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 1024)	0
dense_1 (Dense)	(None, 512)	524800
dense_2 (Dense)	(None, 1)	513
<hr/>		
Total params: 525,313		
Trainable params: 525,313		
Non-trainable params: 0		

Hình 11-1: Đầu ra của model.summary()

Hình 11-1 cho thấy đầu ra của `model.summary()`. Mô tả của mỗi lớp được in ra màn hình, cùng với số tham số được liên kết với lớp đó. Ví dụ: lớp `thick_1` có 524.800 tham số vì mỗi trong số 512 nơ-ron của nó nhận được một bản sao của mỗi trong số 1.024 giá trị đầu vào từ lớp đầu vào, nghĩa là có 1.024×512 trọng số. Thêm 512 tham số sai lệch và bạn nhận được $1.024 \times 512 + 512 = 524.800$.

Mặc dù chúng tôi chưa đào tạo mô hình của mình hoặc kiểm tra nó khi xác thực dữ liệu, đây là mô hình Keras đã biên dịch sẵn sàng đào tạo!

Lưu ý `lỗi` tra mã mẫu trong `ch11/model_architecture.py` để biết ví dụ về mô hình phức tạp hơn một chút!

Đào tạo người mẫu

Để đào tạo mô hình của chúng tôi, chúng tôi cần dữ liệu đào tạo. Máy ảo đi kèm với cuốn sách này bao gồm một tập hợp khoảng nửa triệu tệp HTML lành tính và độc hại. Điều này bao gồm hai thư mục lành tính (`ch11/data/html/benign_files/`) và các tệp HTML độc hại (`ch11/data/html/malicious_files/`). (Hãy nhớ không mở những tệp này trong trình duyệt!) Trong phần này, chúng tôi sử dụng những tệp này để đào tạo mạng lưới thần kinh của mình để dự đoán xem tệp HTML là lành tính (0) hay độc hại (1).

Trích xuất các tính năng

Để làm điều này, trước tiên chúng ta cần quyết định cách biểu diễn dữ liệu của mình. Nói cách khác, chúng ta muốn trích xuất những tính năng nào từ mỗi tệp HTML để sử dụng làm đầu vào cho mô hình của mình? Ví dụ: chúng tôi có thể chỉ cần chuyển 1.000 ký tự đầu tiên trong mỗi tệp HTML cho mô hình, chúng tôi có thể chuyển số lượng tần suất của tất cả các chữ cái trong bảng chữ cái hoặc chúng tôi có thể sử dụng trình phân tích cú pháp HTML để phát triển một số tính năng phức tạp hơn. Để làm cho mọi thứ dễ dàng hơn, chúng tôi sẽ chuyển đổi từng tệp HTML có độ dài thay đổi, có khả năng rất lớn thành một biểu diễn nén, có kích thước thống nhất, cho phép mô hình của chúng tôi nhanh chóng xử lý và tìm hiểu các mẫu quan trọng.

Trong ví dụ này, chúng tôi chuyển đổi từng tệp HTML thành một vectơ có độ dài 1.024 để đếm danh mục, trong đó mỗi số danh mục biểu thị số lượng mã thông báo trong tệp HTML có hàm băm được phân giải theo danh mục nhất định.

Liệt kê 11-2 hiển thị mã trích xuất tính năng.

nhập numpy dưới dạng np

tiếng thi thảm nhập khẩu

nhập lại

nhập hệ điều hành

```
def read_file(sha, dir):
    với open(os.path.join(dir, sha), 'r') dưới dạng fp:
        tệp = fp.read()
    trả lại tệp tin
```

```

def extract_features(sha, path_to_files_dir,
                    hash_dim=1024, usplit_regex=r"\s+"):
    tệp = read_file(sha=sha, dir=path_to_files_dir)
    tokens = re.split(pattern=split_regex, string=file) # bây giờ lấy modulo(hàm băm của mỗi token) để mỗi token được thay thế
    # theo nhóm (danh mục) từ 1:hash_dim.
    token_hash_buckets = [
        (murmur.string_hash(w) % (hash_dim - 1) + 1) cho w bằng mã thông báo
    ]
    # Cuối cùng, chúng tôi sẽ đếm xem mỗi nhóm có bao nhiêu lượt truy cập, để các tính năng của chúng tôi
    # luôn có độ dài hash_dim, bất kể kích thước của tệp HTML:
    token_bucket_counts = np.zeros(hash_dim)
    # điều này trả về tần số đếm cho từng giá trị duy nhất trong
    # token_hash_buckets:
    nhóm, số lượng = np.unique(token_hash_buckets, return_counts=True)
    # và bây giờ chúng tôi chèn các số đếm này vào đối tượng token_bucket_counts của chúng tôi:
    # đối với xô, số lượng trong zip(xô, số lượng):
    token_bucket_counts[xô] = đếm
    trả lại np.array (token_bucket_counts)

```

Liệt kê 11-2: Mã trích xuất tính năng

Bạn không cần phải hiểu tất cả các chi tiết của mã này để hiểu cách thức hoạt động của Keras , nhưng tôi khuyến khích bạn đọc qua các nhận xét trong mã để hiểu rõ hơn những gì đang diễn ra.

Hàm extract_features bắt đầu bằng cách đọc trong tệp HTML dưới dạng tệp lớn chuỗi và sau đó tách chuỗi này thành một tập hợp các mã thông báo dựa trên biểu thức chính quy . Tiếp theo, hàm băm số của mỗi mã thông báo được lấy và các hàm băm này được chia thành các loại bằng cách lấy modulo của mỗi hàm băm .

Tập hợp các tính năng cuối cùng là số lượng giá trị băm trong mỗi danh mục , giống như số lượng thùng biểu đồ. Nếu muốn, bạn có thể thử thay đổi biểu thức chính quy split_regex chia tệp HTML thành nhiều phần để xem nó ảnh hưởng như thế nào đến các mã thông báo và tính năng thu được.

Nếu bạn bỏ qua hoặc không hiểu tất cả những điều đó, không sao cả: chỉ cần biết rằng chức năng extract_features của chúng tôi lấy đường dẫn đến tệp HTML làm đầu vào và sau đó chuyển đổi nó thành một mảng tính năng có độ dài 1.024 hoặc bất kỳ hàm hash_dim nào.

Tạo Trình tạo dữ liệu

Bây giờ chúng ta cần làm cho mô hình Keras của mình thực sự huấn luyện các tính năng này. Khi làm việc với một lượng nhỏ dữ liệu đã được tải vào bộ nhớ, bạn có thể sử dụng một dòng mã đơn giản như Liệt kê 11-3 để huấn luyện mô hình của mình trong Keras.

```
# trước tiên, bạn sẽ tải my_data và my_labels thông qua một số phương tiện, sau đó:
model.fit(my_data, my_labels, epochs=10, batch_size=32)
```

Liệt kê 11-3: Huấn luyện mô hình của bạn khi dữ liệu đã được tải vào bộ nhớ

Tuy nhiên, điều này không thực sự hữu ích khi bạn bắt đầu làm việc với một lượng lớn dữ liệu, bởi vì bạn không thể đưa tất cả dữ liệu đào tạo của mình vào bộ nhớ máy tính cùng một lúc. Để giải quyết vấn đề này, chúng tôi sử dụng nhiều hơn một chút

chức năng `model.fit_generator` phức tạp nhưng có thể mở rộng hơn . Thay vì chuyển tất cả dữ liệu đào tạo cùng một lúc cho chức năng này, bạn chuyển một trình tạo tạo ra dữ liệu đào tạo theo lô để RAM máy tính của bạn không bị nghẹt.

Trình tạo Python hoạt động giống như các hàm Python, ngoại trừ chúng có câu lệnh `năng suất` . Thay vì trả về một kết quả duy nhất, các trình tạo trả về một đối tượng có thể được gọi đi gọi lại để tạo ra nhiều hoặc vô số tập hợp kết quả. Liệt kê 11-4 cho thấy cách chúng ta có thể tạo trình tạo dữ liệu của riêng mình bằng cách sử dụng hàm trích xuất đặc trưng.

```
def my_generator(tệp_lành_tính, tệp_độc_hại,
                 đường_dẫn_đến_tệp_lành_tính, đường_dẫn_đến_tệp_độc_hại,
                 batch_size, features_length=1024):
    n_samples_per_class = batch_size / 2
    khẳng định len(benign_files) >= n_samples_per_class
    khẳng định len(malicious_files) >= n_samples_per_class
    trong_khi True:
        ben_features = [
            extract_features(sha, path_to_files_dir=path_to_benign_files,
                             hash_dim=features_length)
            cho sha trong np.random.choice(benign_files, n_samples_per_class,
                                             thay_thé=Sai)
        ]
        mal_features = [
            extract_features(sha, path_to_files_dir=path_to_malicious_files,
                             hash_dim=features_length)
            cho sha trong np.random.choice(malicious_files, n_samples_per_class,
                                             thay_thé=Sai)
        ]
        all_features = ben_features + mal_features
        nhãn = [0 cho tôi trong phạm vi (n_samples_per_class)] + [1 cho tôi trong phạm vi (
            n_samples_per_class)]

        idx = np.random.choice(range(batch_size), batch_size)
        all_features = np.array([np.array(all_features[i]) for i in idx]) label =
            np.array([labels[i] for i in idx]) sinh ra
        all_features, label
```

Liệt kê 11-4: Viết trình tạo dữ liệu

Đầu tiên, đoạn mã tạo hai câu lệnh khẳng định để kiểm tra xem có đủ dữ liệu không . Sau đó, bên trong một vòng lặp (vì vậy nó sẽ lặp lại mãi mãi), cả các tính năng lành tính và độc hại đều được lấy bằng cách chọn một mẫu ngẫu nhiên của các khóa tệp và sau đó trích xuất các tính năng cho các tệp đó bằng cách sử dụng hàm `extract_features` của chúng tôi . Tiếp theo, các tính năng lành tính và độc hại và các nhãn liên quan (0 và 1) được nối và xáo trộn . Cuối cùng, các tính năng và nhãn này được trả về .

Sau khi được khởi tạo, trình tạo này sẽ tạo ra các tính năng và nhãn `batch_size` để mô hình huấn luyện (50% độc hại, 50% lành tính) mỗi khi phương thức `next()` của trình tạo được gọi.

Liệt kê 11-5 cho thấy cách tạo một trình tạo dữ liệu đào tạo bằng cách sử dụng dữ liệu đi kèm với cuốn sách này và cách huấn luyện mô hình của chúng ta bằng cách chuyển trình tạo này sang phương thức `fit_generator` của mô hình .

```

nhập hệ điều hành

lô_size = 128
features_length = 1024
path_to_training_benign_files = 'data/html/benign_files/training/'
path_to_training_malicious_files = 'data/html/malicious_files/training/'
bước_per_epoch = 1000 # nhỏ một cách giả tạo đối với tốc độ mã ví dụ!

train_benign_files = os.listdir(path_to_training_benign_files)
train_malicious_files = os.listdir(path_to_training_malicious_files)

# tao trình tạo dữ liệu đào tạo của chúng tôi!
training_generator = my_generator(
    ben_files=train_benign_files,
    malware_files=train_malicious_files,
    path_to_benign_files=path_to_training_benign_files,
    path_to_malicious_files=path_to_training_malicious_files,
    batch_size=batch_size,
    features_length=features_length
)

model.fit_generator(
    máy phát điện = đào tạo_máy phát điện,
    step_per_epoch=steps_per_epoch,
    ký nguyên=10
)

```

Lịt kê 11-5: Tạo trình tạo đào tạo và sử dụng nó để đào tạo mô hình

Hãy thử đọc qua mã này để hiểu điều gì đang xảy ra. Sau đó nhập một gói cần thiết và tạo một số biến tham số, chúng tôi đọc tên tệp cho dữ liệu huấn luyện là tên và đặc hại vào bộ nhớ (chứ không phải bản thân các tệp). Chúng tôi chuyển các giá trị này cho hàm `my_generator` mới của chúng tôi để nhận trình tạo dữ liệu đào tạo của chúng tôi. Cuối cùng, bằng cách sử dụng mô hình của chúng ta từ Lịt kê 11-1, chúng ta sử dụng hàm `fit_generator` tích hợp sẵn của mô hình phương pháp để bắt đầu huấn luyện.

Phương thức `fit_generator` nhận ba tham số. Thông số máy phát điện `generator` chỉ định trình tạo dữ liệu tạo dữ liệu huấn luyện cho mỗi đợt. Trong quá trình đào tạo, các tham số được cập nhật một lần cho mỗi đợt bằng cách lấy trung bình tất cả các tín hiệu của các quan sát đào tạo cho đợt đó. Các `bước_per_epoch` tham số đặt số lô mà chúng tôi muốn mô hình xử lý mỗi ký nguyên. Do đó, tổng số quan sát mà mô hình thấy trên mỗi ký nguyên là `batch_size*steps_per_epoch`. Theo quy ước, số lượng quan sát mà một mô hình nhìn thấy trên mỗi ký nguyên phải bằng với kích thước tập dữ liệu, nhưng trong chương này và trong mã mẫu máy áo, tôi giảm `bước_per_epoch` để giúp mã của chúng tôi chạy nhanh hơn. Tham số ký nguyên đặt số lượng ký nguyên mà chúng ta muốn chạy.

Hãy thử chạy mã này trong thư mục ch11/ đi kèm với cuốn sách này. Tùy thuộc vào sức mạnh của máy tính của bạn, mỗi ký nguyên đào tạo sẽ mất một khoảng thời gian nhất định để chạy. Nếu bạn đang sử dụng phiên tương tác, vui lòng hủy quy trình (`ctrl-C`) sau một vài ký nguyên nếu quá trình này mất một lúc. Cái này

sẽ dừng đào tạo mà không làm mất tiến độ. Sau khi bạn hủy quy trình (hoặc mã hoàn tất), bạn sẽ có một mô hình được đào tạo! Kết quả đọc trên màn hình máy ảo của bạn sẽ giống như Hình 11-2.

```
Using TensorFlow backend.
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcublas.so.7.5 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcudnn.so.5 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcufft.so.7.5 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcuda.so.1 locally
I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcurand.so.7.5 locally
Epoch 1/10
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE3 instructions
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions
NVIDIA: no NVIDIA devices found
E tensorflow/stream_executor/cuda/cuda_driver.cc:509] failed call to cuInit: CUDA_ERROR_UNKNOWN
I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:145] kernel driver does not appear to be running on
39/39 [=====] - 7s 171ms/step - loss: 0.3463 - acc: 0.8476
Epoch 2/10
39/39 [=====] - 7s 168ms/step - loss: 0.2181 - acc: 0.9139
Epoch 3/10
39/39 [=====] - 7s 168ms/step - loss: 0.1864 - acc: 0.9253
Epoch 4/10
18/39 [=====>.....] - ETA: 3s - loss: 0.1871 - acc: 0.9262
```

Hình 11-2: Đầu ra của bảng điều khiển từ việc huấn luyện mô hình Keras

Một vài dòng trên cùng lưu ý rằng TensorFlow, phần phụ trợ mặc định cho Keras, đã được tải. Bạn cũng sẽ thấy một số cảnh báo như trong Hình 11-2; điều này chỉ có nghĩa là quá trình đào tạo sẽ được thực hiện trên CPU thay vì GPU (GPU thường nhanh hơn khoảng 2-20 lần để đào tạo mạng thần kinh, nhưng với mục đích của cuốn sách này, việc đào tạo dựa trên CPU vẫn ổn). Cuối cùng, bạn sẽ thấy một thanh tiến trình cho mỗi ký nguyên cho biết thời gian cụ thể sẽ mất bao lâu, cũng như các số liệu về độ chính xác và tổn thất của ký nguyên.

Kết hợp dữ liệu xác thực

Trong phần trước, bạn đã học cách đào tạo mô hình Keras trên các tệp HTML bằng phương pháp `fit_generator` có thể mở rộng. Như bạn đã thấy, mô hình in các câu lệnh trong quá trình đào tạo, cho biết số liệu thống kê về tổn thất hiện tại và độ chính xác của mỗi ký nguyên. Tuy nhiên, điều bạn thực sự quan tâm là mô hình được đào tạo của bạn thực hiện như thế nào trên dữ liệu xác thực hoặc dữ liệu mà nó chưa từng thấy trước đây.

Điều này thể hiện tốt hơn loại dữ liệu mà mô hình của bạn sẽ gặp phải trong môi trường sản xuất thực tế.

Khi cố gắng thiết kế các mô hình tốt hơn và tìm hiểu thời gian đào tạo mô hình của bạn, bạn nên cố gắng tối đa hóa độ chính xác xác nhận thay vì độ chính xác đào tạo, điều sau được thể hiện trong Hình 11-2. Tốt hơn nữa là sử dụng các tệp xác thực có nguồn gốc từ các ngày sau dữ liệu đào tạo để mô phỏng môi trường sản xuất tốt hơn.

Liệt kê 11-6 cho thấy cách tải các tính năng xác thực của chúng ta vào bộ nhớ bằng cách sử dụng hàm `my_generator` của chúng ta từ Liệt kê 11-4.

```
nhập hệ điều hành
path_to_validation_benign_files = 'dữ liệu/html/benign_files/xác thực/'
```

```

path_to_validation_malicious_files = 'data/html/malicious_files/validation/'
# lấy khóa xác thực:
val_benign_file_keys = os.listdir(path_to_validation_benign_files)
val_malicious_file_keys = os.listdir(path_to_validation_malicious_files)
# lấy dữ liệu xác thực và trích xuất các tính năng:
validation_data = my_generator(
    ben_files=val_benign_files,
    malware_files=val_malicious_files,
    path_to_benign_files=path_to_validation_benign_files,
    path_to_malicious_files=path_to_validation_malicious_files,
    batch_size=10000,
    features_length=features_length
).tiếp theo()

```

Liệt kê 11-6: Đọc các tính năng xác thực và nhãn vào bộ nhớ bằng cách sử dụng hàm `my_generator`

Mã này rất giống với cách chúng tôi tạo trình tạo dữ liệu đào tạo của mình, ngoại trừ việc đường dẫn tệp đã thay đổi và bây giờ chúng tôi muốn tải tất cả dữ liệu hợp lệ vào bộ nhớ. Vì vậy, thay vì chỉ tạo trình tạo, chúng tôi tạo trình tạo dữ liệu xác thực với `batch_size` lớn bằng số lượng tệp chúng tôi muốn xác thực và chúng tôi gọi ngay phương thức `.next()` của nó chỉ một lần.

Bây giờ chúng ta đã tải một số dữ liệu xác thực vào bộ nhớ, Keras cho phép chúng ta chỉ cần chuyển dữ liệu xác thực `fit_generator()` của mình trong quá trình đào tạo, như trong Liệt kê 11-7.

```

model.fit_generator(
    validation_data=validation_data,
    máy phát điện = đào tạo_máy phát điện,
    step_per_epoch=steps_per_epoch,
    kỳ nguyên = 10
)

```

Liệt kê 11-7: Sử dụng dữ liệu xác thực để theo dõi tự động trong quá trình đào tạo

Liệt kê 11-7 gần như giống với phần cuối của Liệt kê 11-5, ngoại trừ việc dữ liệu xác thực hiện được chuyển đến `fit_generator`. Điều này giúp tăng cường giám sát mô hình bằng cách đảm bảo rằng độ chính xác và độ mất xác thực được tính toán cùng với độ chính xác và độ mất mát khi đào tạo.

Bây giờ, các câu lệnh huấn luyện sẽ trông giống như Hình 11-3.

```

Epoch 1/10
39/39 [=====] - 8s 192ms/step - loss: 0.1146 - acc: 0.9571 - val_loss: 0.5067 - val_acc: 0.7690
Epoch 2/10
39/39 [=====] - 7s 184ms/step - loss: 0.1392 - acc: 0.9463 - val_loss: 0.2621 - val_acc: 0.8970
Epoch 3/10
39/39 [=====] - 7s 189ms/step - loss: 0.1234 - acc: 0.9527 - val_loss: 0.3382 - val_acc: 0.8790
Epoch 4/10
39/39 [=====] - 7s 189ms/step - loss: 0.0981 - acc: 0.9611 - val_loss: 0.2770 - val_acc: 0.8970
Epoch 5/10
39/39 [=====] - 7s 189ms/step - loss: 0.1232 - acc: 0.9541 - val_loss: 0.3053 - val_acc: 0.8790
Epoch 6/10
37/39 [=====]>..] - ETA: 0s - loss: 0.1068 - acc: 0.9552

```

Hình 11-3: Đầu ra của bảng điều khiển từ việc đào tạo mô hình Keras với dữ liệu xác thực

Hình 11-3 tương tự như Hình 11-2, ngoại trừ việc thay vì chỉ hiển thị số liệu acc và mất đào tạo cho mỗi ký nguyên, giờ đây Keras cũng tính toán và hiển thị val_loss (mất xác thực) và val_acc (độ chính xác của xác thực) cho mỗi ký nguyên. Nói chung, nếu độ chính xác của quá trình xác thực giảm xuống thay vì tăng lên, thì đó là dấu hiệu cho thấy mô hình của bạn khớp quá mức với dữ liệu đào tạo của bạn và tốt nhất bạn nên tạm dừng đào tạo. Nếu độ chính xác của xác thực tăng lên, như trường hợp ở đây, điều đó có nghĩa là mô hình của bạn vẫn đang tốt hơn và bạn nên tiếp tục đào tạo.

Lưu và tải mô hình

Bây giờ bạn đã biết cách xây dựng và huấn luyện mạng thần kinh, hãy xem cách lưu mạng để bạn có thể chia sẻ với người khác.

Liệt kê 11-8 cho thấy cách lưu mô hình được đào tạo của chúng ta vào một tệp .h5 và tải lại nó (có thể là sau này).

```
từ keras.models nhập load_model
# lưu mô hình
model.save('my_model.h5')
# tải mô hình trở lại bộ nhớ từ tệp:
same_model = load_model('my_model.h5')
```

Liệt kê 11-8: Lưu và tải các mô hình Keras

Đánh giá mô hình

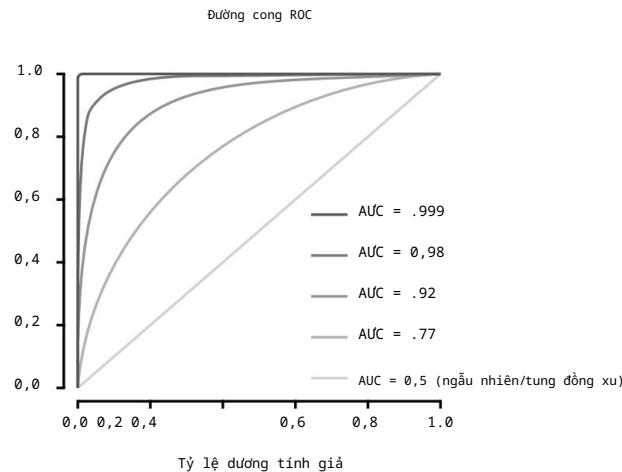
Trong phần đào tạo mô hình, chúng tôi đã quan sát thấy một số chỉ số đánh giá mô hình mặc định như độ chính xác và độ mất mát trong quá trình đào tạo cũng như độ chính xác và độ chính xác của độ chính xác. Bây giờ, hãy xem xét một số chỉ số phức tạp hơn để đánh giá tốt hơn các mô hình của chúng ta.

Một số liệu hữu ích để đánh giá độ chính xác của bộ dự đoán nhị phân được gọi là diện tích dưới đường cong (AUC). Đường cong đề cập đến đường cong Đặc tính hoạt động của máy thu (ROC) (xem Chương 8), biểu thị tỷ lệ dương tính giả (trục x) so với tỷ lệ dương tính thật (trục y) cho tất cả các ngưỡng điểm có thể.

Ví dụ: mô hình của chúng tôi có gắng dự đoán xem một tệp có độc hại hay không bằng cách sử dụng điểm từ 0 (lành tính) đến 1 (độc hại). Nếu chúng tôi chọn ngưỡng điểm tương đối cao để phân loại tệp là độc hại, chúng tôi sẽ nhận được ít kết quả xác thực sai hơn (tốt) nhưng cũng ít kết quả xác thực đúng hơn (xấu). Mặt khác, nếu chúng tôi chọn ngưỡng điểm thấp, chúng tôi có thể có tỷ lệ dương tính giả cao (xấu) nhưng tỷ lệ phát hiện xác cao (tốt).

Hai khả năng mẫu này sẽ được biểu diễn dưới dạng hai điểm trên đường cong ROC của mô hình của chúng tôi, trong đó điểm đầu tiên sẽ nằm về phía bên trái của đường cong và điểm thứ hai nằm gần bên phải. AUC thể hiện tất cả các khả năng này bằng cách lấy diện tích dưới đường cong ROC này, như trong Hình 11-4.

Nói một cách đơn giản, AUC là 0,5 thể hiện khả năng dự đoán của một lật đồng xu, trong khi AUC là 1 là hoàn hảo.



Hình 11-4: Các đường cong ROC mẫu khác nhau. Mỗi đường cong (đường) ROC tương ứng với một giá trị AUC khác nhau.

Hãy sử dụng dữ liệu xác thực của chúng ta để tính toán AUC xác thực bằng cách sử dụng mã trong Liệt kê 11-9.

từ số liệu nhập sklearn

```
xác_thực_nhân = xác_thực_dữ_liệu[1]
validation_scores = [el[0] for el in model.predict(validation_data[0])]
fpr, tpr, thres = metric.roc_curve(y_true=validation_labels,
                                      y_score=validation_scores)
auc = metrics.auc(fpr, tpr)
print('Xác thực AUC = {}'.format(auc))
```

Liệt kê 11-9: Tính toán xác thực AUC bằng cách sử dụng mô đun con số liệu của sklearn

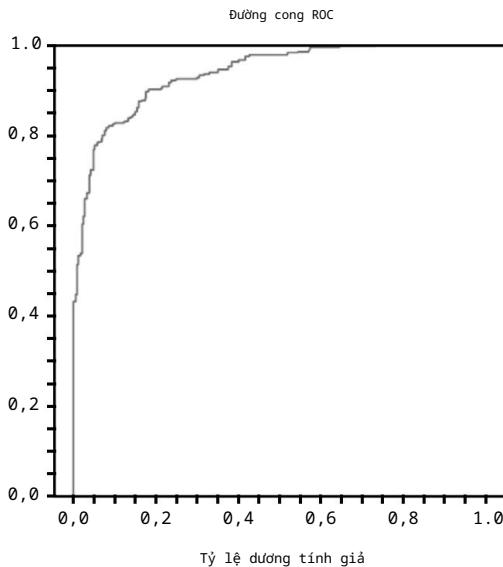
Ở đây, chúng tôi chia bộ dữ liệu_validation_data của mình thành hai đối tượng: các nhãn xác thực được biểu thị bằng các_nhân_xác_thực và các dự đoán mô hình xác thực phẳng được biểu thị bằng các_diểm_xác_thực. Sau đó, chúng tôi sử dụng hàm metrics.roc_curve từ sklearn để tính tỷ lệ dương tính giả, tỷ lệ dương tính thật và các giá trị ngưỡng liên quan cho dự đoán mô hình. Sử dụng những thứ này, chúng tôi tính toán số liệu AUC của mình, một lần nữa sử dụng sklearn hàm .

Mặc dù tôi sẽ không xem qua mã hàm ở đây, nhưng bạn cũng có thể sử dụng hàm roc_plot() có trong tệp ch11/model_evaluation.py trong dữ liệu đi kèm cuốn sách này để vẽ đồ thị đường cong ROC thực tế, như trong Liệt kê 11-10 .

từ ch11.model_evaluation nhập roc_plot
roc_plot(fpr=fpr, tpr=tpr, path_to_file='roc_curve.png')

Liệt kê 11-10: Tạo biểu đồ đường cong ROC bằng cách sử dụng hàm roc_plot từ dữ liệu đi kèm của cuốn sách này, trong ch11/model_evaluation.py

Việc chạy mã trong Liệt kê 11-10 sẽ tạo ra một biểu đồ (được lưu vào roc_curve.png) giống như Hình 11-5.



Hình 11-5: Một đường cong ROC!

Mỗi điểm trong đường cong ROC trong Hình 11-5 biểu thị một tỷ lệ dương tính giả cụ thể (trục x) và tỷ lệ dương tính thật (trục y) được liên kết với các ngưỡng dự đoán mô hình khác nhau nằm trong khoảng từ 0 đến 1. Khi tỷ lệ dương tính giả tăng lên, tỷ lệ dương tính thực tăng và ngược lại. Trong môi trường sản xuất, bạn thường phải chọn một ngưỡng duy nhất (một điểm duy nhất trên đường cong này, giả sử dữ liệu xác thực bắt chước dữ liệu sản xuất) để đưa ra quyết định của bạn, dựa trên mức độ săn sàng chấp nhận sai tích cực của bạn, so với mức độ săn sàng chấp nhận rủi ro của bạn một tệp độc hại để lọt qua vết nứt.

Tăng cường quy trình đào tạo mô hình với các cuộc gọi lại

Cho đến giờ, bạn đã học cách thiết kế, đào tạo, lưu, tải và đánh giá Keras người mẫu. Mặc dù đây thực sự là tất cả những gì bạn cần để có một khởi đầu khá tốt, tôi cũng muốn giới thiệu các cuộc gọi lại Keras, có thể làm cho quá trình đào tạo mô hình của chúng ta trở nên tốt hơn.

Một cuộc gọi lại Keras đại diện cho một tập hợp các chức năng mà Keras áp dụng trong các giai đoạn nhất định của quy trình đào tạo. Ví dụ: bạn có thể sử dụng lệnh gọi lại Keras để đảm bảo rằng tệp .h5 được lưu ở cuối mỗi kỳ nguyên hoặc AUC xác thực đó được in ra màn hình ở cuối mỗi kỳ nguyên. Điều này có thể giúp ghi lại và thông báo cho bạn chính xác hơn về cách thức hoạt động của mô hình trong quá trình đào tạo.

Chúng tôi bắt đầu bằng cách sử dụng lệnh gọi lại tích hợp sẵn, sau đó chúng tôi thử viết lệnh gọi lại tùy chỉnh của riêng mình.

Sử dụng Gọi lại tích hợp

Để sử dụng chức năng gọi lại tích hợp, chỉ cần chuyển một phiên bản gọi lại cho phương thức `fit_generator()` của mô hình của bạn trong quá trình đào tạo. Chúng tôi sẽ sử dụng các cuộc gọi lại `ModelCheckpoint` gọi lại, đánh giá tồn thât xác thực sau mỗi kỳ nguyên đào tạo và lưu mô hình hiện tại vào một tệp nếu tồn thât xác thực nhỏ hơn bất kỳ tồn thât xác thực nào của kỳ nguyên trước đó. Để làm điều này, lệnh gọi lại cần quyền truy cập vào dữ liệu xác thực của chúng ta, vì vậy chúng ta sẽ chuyển dữ liệu đó vào phương thức `fit_generator()`, như trong [Liệt kê 11-11](#).

từ máy ảnh nhập gọi lại

```
model.fit_generator(
    máy phát điện = đào tạo_máy phát điện,
    # giảm bước_per_epoch để mã ví dụ chạy nhanh:
    bước_per_epoch=50,
    kỳ nguyên = 5,
    verify_data=validation_data,
    gọi lại = [
        gọi lại.ModelCheckpoint(save_best_only=True,
                               filepath='results/best_model.h5',
                               màn hình='val_loss')
    ],
)
```

[Liệt kê 11-11](#): Thêm cuộc gọi lại `ModelCheckpoint` vào quy trình đào tạo

Mã này đảm bảo rằng mô hình được ghi đè vào một tệp duy nhất, 'results/best_model.h5', bắt cứ khi nào 'val_loss' (tồn thât xác thực) đạt mức thấp mới. Điều này đảm bảo rằng mô hình đã lưu hiện tại ('results/best_model.h5') luôn đại diện cho mô hình tốt nhất trong tất cả các kỳ nguyên đã hoàn thành liên quan đến việc mất xác thực.

Ngoài ra, chúng ta có thể sử dụng mã trong [Liệt kê 11-12](#) để lưu mô hình sau mỗi kỳ nguyên thành một tệp riêng biệt bắt kè mất xác thực.

```
goi lại.ModelCheckpoint(save_best_only=False,
                       filepath='results/model_epoch_{epoch}.h5',
                       màn hình='val_loss')
```

[Liệt kê 11-12](#): Thêm một cuộc gọi lại `ModelCheckpoint` vào quy trình đào tạo để lưu mô hình vào một tệp khác sau mỗi kỳ nguyên

Để làm điều này, chúng ta sử dụng cùng mã trong [Liệt kê 11-11](#) và cùng chức năng `ModelCheckpoint`, nhưng với `save_best_only=False` và một đường dẫn tệp yêu cầu Keras điền vào số kỳ nguyên . Thay vì chỉ lưu phiên bản "tốt nhất" duy nhất của mô hình của chúng ta, lệnh gọi lại của [Liệt kê 11-12](#) lưu phiên bản của từng kỳ nguyên của mô hình của chúng ta, trong `results/model_epoch_0.h5`, `results/model_epoch_1.h5`, `results/model_epoch_2.h5`, v.v..

Sử dụng gọi lại tùy chỉnh

Mặc dù Keras không hỗ trợ AUC, nhưng chúng tôi có thể thiết kế lệnh gọi lại tùy chỉnh của riêng mình, chẳng hạn như cho phép chúng tôi in AUC ra màn hình sau mỗi kỳ nguyên.

Để tạo một cuộc gọi lại Keras tùy chỉnh, chúng ta cần tạo một lớp kế thừa nó từ `keras.callbacks.Callback`, lớp cơ sở trừu tượng được sử dụng để xây dựng các cuộc gọi lại mới. Chúng ta có thể thêm một hoặc nhiều lựa chọn phương thức, phương thức này sẽ được chạy tự động trong quá trình đào tạo, vào những thời điểm mà tên của chúng chỉ định: `on_epoch_begin`, `on_epoch_end`, `on_batch_begin`, `on_batch_end`, `on_train_begin` và `on_train_end`.

Liệt kê 11-13 cho thấy cách tạo một cuộc gọi lại để tính toán và in AUC xác thực ra màn hình ở cuối mỗi kỳ nguyên.

```
nhập numpy dưới dạng np
từ máy ảnh nhập gọi lại
từ số liệu nhập sklearn
```

lớp `MyCallback(callbacks.Callback)`:

```
def on_epoch_end(self, epoch, logs={}):
    xác thực_nhân = self.validation_data[1]
    verify_scores = self.model.predict(self.validation_data[0])
    # làm phẳng điểm số:
    verify_scores = [el[0] cho el trong verify_scores]
    fpr, tpr, thres = metric.roc_curve(y_true=validation_labels,
                                         y_score=validation_scores)
    auc = metrics.auc(fpr, tpr)
    print('\n\tEpoch {}, AUC xác thực = {}'.format(Epoch,
                                                    np.round(auc, 6)))

model.fit_generator(
    máy phát điện = đào tạo_máy phát điện,
    # giảm bước_per_epoch để mã ví dụ chạy nhanh:
    bước_per_epoch=50,
    ký nguyên = 5,
    validation_data=validation_data,
    callbacks=[
        callbacks.ModelCheckpoint('results/model_epoch_{epoch}.h5',
                                 màn hình='val_loss',
                                 save_best_only=Sai,
                                 save_weights_only=Sai)
    ]
)
```

Liệt kê 11-13: Tạo và sử dụng lệnh gọi lại tùy chỉnh để in AUC ra màn hình sau mỗi giai đoạn đào tạo

Trong ví dụ này, trước tiên chúng ta tạo lớp `MyCallback`, lớp kế thừa từ các cuộc gọi lại `.Callbacks`. Để mọi thứ đơn giản, chúng tôi ghi đè lên một phương thức duy nhất, `on_epoch_end`, và cung cấp cho nó hai đối số mà Keras mong đợi: `epoch` và `log` (từ điển thông tin nhật ký), cả hai đối số này Keras sẽ cung cấp khi nó gọi hàm trong quá trình đào tạo.

Sau đó, chúng tôi lấy dữ liệu xác thực , đã được lưu trữ trong tự đổi tương nhở vào tính năng gọi lại. Ké thừa gọi lại, đồng thời chúng ta tính toán và in ra AUC giống như chúng ta đã làm trong phần “Đánh giá mô hình” ở trang 209. Lưu ý rằng để mã này hoạt động, dữ liệu xác thực cần được chuyển đến `fit_generator()` để cuộc gọi lại có quyền truy cập vào `self.validation_data` trong quá trình đào tạo . Cuối cùng, chúng tôi yêu cầu mô hình đào tạo và chỉ định hàm gọi lại mới của chúng tôi. Kết quả sẽ giống như Hình 11-6.

```
Epoch 1/5
39/39 [=====] - 7s 186ms/step - loss: 0.1148 - acc: 0.9515 - val_loss: 0.3693 - val_acc: 0.8630

    Epoch 0, Validation AUC = 0.922248
Epoch 2/5
39/39 [=====] - 7s 175ms/step - loss: 0.1308 - acc: 0.9507 - val_loss: 0.2938 - val_acc: 0.8640

    Epoch 1, Validation AUC = 0.947984
Epoch 3/5
39/39 [=====] - 7s 179ms/step - loss: 0.1120 - acc: 0.9599 - val_loss: 0.3064 - val_acc: 0.8730

    Epoch 2, Validation AUC = 0.949036
Epoch 4/5
39/39 [=====] - 7s 179ms/step - loss: 0.1134 - acc: 0.9625 - val_loss: 0.3167 - val_acc: 0.8520

    Epoch 3, Validation AUC = 0.958548
Epoch 5/5
22/39 [=====>.....] - ETA: 2s - loss: 0.1336 - acc: 0.9474
```

Hình 11-6: Đầu ra của bảng điều khiển từ việc đào tạo mô hình Keras với lệnh gọi lại AUC tùy chỉnh

Nếu điều bạn thực sự quan tâm là giảm thiểu AUC xác thực, cuộc gọi lại này giúp dễ dàng xem mô hình của bạn đang hoạt động như thế nào trong quá trình đào tạo, do đó giúp bạn đánh giá xem bạn có nên dừng quá trình đào tạo hay không (ví dụ: nếu độ chính xác của dữ liệu hợp lệ liên tục giảm theo thời gian).

Bản tóm tắt

Trong chương này, bạn đã học cách xây dựng mạng thần kinh của riêng mình bằng Keras. Bạn cũng đã học cách đào tạo, đánh giá, lưu và tải nó. Sau đó, bạn đã học cách nâng cao quy trình đào tạo mô hình bằng cách thêm các lệnh gọi lại tùy chỉnh và tích hợp sẵn. Tôi khuyến khích bạn chơi xung quanh mã đi kèm với cuốn sách này để xem những thay đổi mà kiến trúc mô hình và trích xuất tính năng có thể có đối với độ chính xác của mô hình.

Chương này nhằm mục đích làm ướt chân bạn, nhưng không phải là một hướng dẫn tham khảo. Truy cập <https://keras.io> để biết tài liệu chính thức cập nhật nhất. Tôi đặc biệt khuyên khích bạn dành thời gian nghiên cứu các khía cạnh của Keras mà bạn quan tâm. Hy vọng rằng chương này đã phục vụ như một điểm khởi đầu tốt cho tất cả các cuộc phiêu lưu tìm hiểu sâu về bảo mật của bạn!

12

Trở thành một nhà khoa học dữ liệu



Để kết thúc cuốn sách này, chúng ta hãy lùi lại một bước và thảo luận về cách bạn có thể tạo dựng cuộc sống và sự nghiệp với tư cách là nhà khoa học dữ liệu phần mềm độc hại hoặc

một nhà khoa học dữ liệu bảo mật nói chung. Mặc dù đây là một chương phi kỹ thuật, nhưng nó cũng quan trọng như các chương kỹ thuật trong cuốn sách này, nếu không muốn nói là quan trọng hơn. Điều này là do trở thành một nhà khoa học dữ liệu bảo mật thành công liên quan đến nhiều thứ hơn là chỉ đơn giản là hiểu vấn đề.

Trong chương này, chúng tôi, các tác giả, chia sẻ con đường sự nghiệp của chính mình để trở thành nhà khoa học dữ liệu bảo mật chuyên nghiệp. Bạn sẽ có cái nhìn thoáng qua về cuộc sống hàng ngày của một nhà khoa học dữ liệu bảo mật và những gì cần thiết để trở thành một nhà khoa học dữ liệu hiệu quả. Chúng tôi cũng chia sẻ các mẹo về cách tiếp cận các vấn đề khoa học dữ liệu và cách duy trì sự kiên cường khi đối mặt với những thách thức không thể tránh khỏi.

Con đường trở thành nhà khoa học dữ liệu bảo mật

Vì khoa học dữ liệu bảo mật là một lĩnh vực mới nên có nhiều con đường để trở thành nhà khoa học dữ liệu bảo mật. Trong khi nhiều nhà khoa học dữ liệu được đào tạo chính quy thông qua trường cao học, thì nhiều người khác lại tự học. Ví dụ, tôi lớn lên trong bối cảnh hack máy tính những năm 1990, nơi tôi học lập trình bằng C và hợp ngữ cũng như viết các công cụ hack mũ đen. Sau đó, tôi lấy bằng cử nhân và sau đó là bằng thạc sĩ về nhân văn trước khi quay trở lại thế giới công nghệ với tư cách là nhà phát triển phần mềm bảo mật. Trên đường đi, tôi đã tự học cách trực quan hóa dữ liệu và học máy vào ban đêm, cuối cùng chuyển sang vai trò khoa học dữ liệu bảo mật chính thức tại Sophos, một công ty nghiên cứu và phát triển bảo mật. Hillary Sanders, đồng tác giả của tôi trong cuốn sách này, đã nghiên cứu về thống kê và kinh tế ở trường đại học, đã làm việc với tư cách là nhà khoa học dữ liệu trong một thời gian và sau đó tìm được việc làm tại một công ty bảo mật với tư cách là nhà khoa học dữ liệu, thu thập kiến thức bảo mật của cô ấy trong công việc.

Đội ngũ của chúng tôi tại Sophos cũng rất đa dạng. Các đồng nghiệp của chúng tôi có nhiều bằng cấp trong nhiều lĩnh vực: tâm lý học, khoa học dữ liệu, toán học, hóa sinh, thống kê và khoa học máy tính. Mặc dù khoa học dữ liệu bảo mật thiên về những người được đào tạo chính quy về các phương pháp định lượng trong khoa học, nhưng nó bao gồm những người có nền tảng kiến thức khác nhau trong các lĩnh vực này. Và mặc dù đào tạo khoa học và định lượng rất hữu ích cho việc học khoa học dữ liệu bảo mật, kinh nghiệm của riêng tôi cho thấy rằng bạn cũng có thể tham gia và xuất sắc trong lĩnh vực của chúng tôi với nền tảng phi truyền thống, miễn là bạn sẵn sàng tự học.

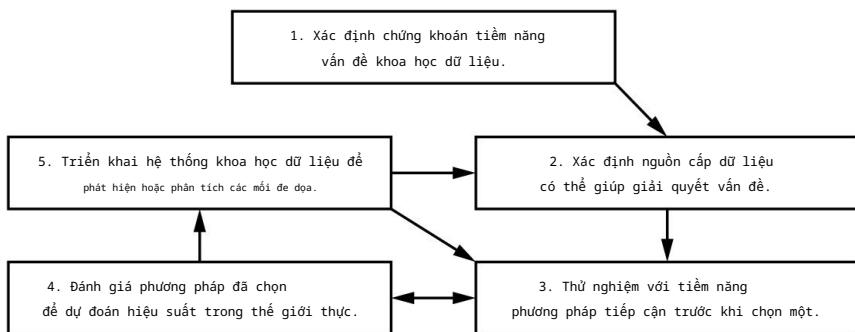
Xuất sắc trong khoa học dữ liệu bảo mật phụ thuộc vào sự sẵn sàng không ngừng học hỏi những điều mới của một người. Điều này là do kiến thức thực tế cũng quan trọng như kiến thức lý thuyết trong lĩnh vực của chúng tôi và bạn tiếp thu kiến thức thực tế thông qua thực hành chứ không phải thông qua bài tập ở trường.

Sẵn sàng học những điều mới cũng rất quan trọng vì học máy, phân tích mạng và trực quan hóa dữ liệu liên tục thay đổi, vì vậy những gì bạn học ở trường sẽ nhanh chóng lỗi thời. Ví dụ: học sâu chỉ mới bắt đầu trở thành một xu hướng trong những năm kể từ khoảng năm 2012 và đã phát triển nhanh chóng kể từ đó, vì vậy hầu hết mọi người trong ngành khoa học dữ liệu đã tốt nghiệp trước đó đều phải tự học những ý tưởng mạnh mẽ này. Đây là một tin tốt cho những người muốn tham gia khoa học dữ liệu bảo mật một cách chuyên nghiệp.

Vì những người đã có kinh nghiệm trong lĩnh vực này phải liên tục dạy cho mình những kỹ năng mới, nên bạn có thể đặt một chân vào cửa bằng cách đã biết những kỹ năng đó.

Một ngày trong cuộc đời của một nhà khoa học dữ liệu bảo mật

Công việc của một nhà khoa học dữ liệu bảo mật là áp dụng loại kỹ năng được dạy trong cuốn sách này vào các vấn đề bảo mật khó. Nhưng việc áp dụng những kỹ năng này có xu hướng được đưa vào trong một quy trình làm việc lớn hơn bao gồm cả những kỹ năng khác. Hình 12-1 minh họa quy trình công việc điển hình của một nhà khoa học dữ liệu bảo mật, dựa trên kinh nghiệm của chúng tôi và của các đồng nghiệp tại các công ty và tổ chức khác.



Hình 12-1: Mô hình quy trình làm việc khoa học dữ liệu bảo mật

Như Hình 12-1 cho thấy, quy trình làm việc của khoa học dữ liệu bảo mật liên quan đến sự tương tác giữa năm lĩnh vực công việc. Lĩnh vực đầu tiên, xác định vấn đề, liên quan đến việc xác định các vấn đề bảo mật mà khoa học dữ liệu có thể trợ giúp. Ví dụ: chúng tôi có thể đưa ra giả thuyết rằng việc xác định email lừa đảo trực tuyến có thể được giải quyết bằng các phương pháp khoa học dữ liệu hoặc việc xác định phương pháp cụ thể được sử dụng để che giấu phần mềm độc hại đã biết là một vấn đề đáng để điều tra.

Ở giai đoạn này, bất kỳ giả định nào cho rằng một vấn đề nhất định có thể được giải quyết bằng khoa học dữ liệu chỉ là một giả thuyết. Khi bạn có một cái búa (khoa học dữ liệu), mọi vấn đề có thể giống như một cái đinh (vấn đề học máy, trực quan hóa dữ liệu hoặc phân tích mạng). Chúng ta phải suy nghĩ xem liệu những vấn đề này có thực sự được giải quyết tốt nhất bằng cách sử dụng các phương pháp khoa học dữ liệu hay không, lưu ý rằng sẽ cần xây dựng một giải pháp khoa học dữ liệu nguyên mẫu và sau đó thử nghiệm giải pháp này để hiểu rõ hơn liệu khoa học dữ liệu có thực sự cung cấp giải pháp tốt nhất hay không .

Khi bạn đang làm việc trong một tổ chức, việc xác định một vấn đề tốt hầu như luôn liên quan đến việc tương tác với các bên liên quan, những người không phải là nhà khoa học dữ liệu. Ví dụ: trong công ty của mình, chúng tôi thường giao tiếp với các nhà quản lý sản phẩm, giám đốc điều hành, nhà phát triển phần mềm và nhân viên bán hàng, những người cho rằng khoa học dữ liệu giống như cây đua thắn có thể giải quyết mọi vấn đề hoặc khoa học dữ liệu giống như "tri tuệ nhân tạo" và do đó có một số khả năng kỳ diệu để đạt được kết quả phi thực tế.

Điều quan trọng cần nhớ khi giao dịch với các bên liên quan như vậy là trung thực về khả năng và hạn chế của các phương pháp tiếp cận dựa trên khoa học dữ liệu, đồng thời duy trì thái độ khôn ngoan, cân nhắc để bạn không theo đuổi các vấn đề sai lầm. Bạn nên loại bỏ các vấn đề không có dữ liệu để thúc đẩy các thuật toán khoa học dữ liệu hoặc không có cách nào để đánh giá liệu các phương pháp khoa học dữ liệu của bạn có thực sự hiệu quả hay không, cũng như các vấn đề mà bạn có thể giải quyết rõ ràng hơn thông qua các phương pháp thủ công hơn.

Ví dụ: đây là một số vấn đề chúng tôi đã từ chối sau khi những người khác yêu cầu chúng tôi giải quyết chúng:

- Tự động xác định những nhân viên bắt hổ có thể làm rõ rỉ rả dữ liệu cho đối thủ cạnh tranh.
- Không có đủ dữ liệu để thúc đẩy thuật toán máy học, nhưng điều này có thể được thực hiện bằng cách sử dụng trực quan hóa dữ liệu hoặc phân tích công việc mạng.

- Giải mã lưu lượng mạng. Toán học của máy học cho thấy rằng máy học đơn giản là không có khả năng giải mã dữ liệu được mã hóa ở cấp độ vũ khí!
- Tự động xác định các email lừa đảo được tạo thủ công để nhắm mục tiêu đến các nhân viên cụ thể dựa trên kiến thức cơ bản chi tiết về lối sống của họ. Một lần nữa, không có đủ dữ liệu để thúc đẩy thuật toán máy học, nhưng điều này có thể thực hiện được thông qua trực quan hóa chuỗi thời gian hoặc dữ liệu email.

Sau khi bạn xác định thành công một vấn đề tiềm ẩn về khoa học dữ liệu bảo mật, nhiệm vụ tiếp theo của bạn là xác định các nguồn cấp dữ liệu mà bạn có thể sử dụng để giúp giải quyết vấn đề bằng cách sử dụng các kỹ thuật khoa học dữ liệu được giải thích trong cuốn sách này. Điều này được thể hiện trong bước 2 của Hình 12-1. Vào cuối ngày, nếu bạn không có nguồn cấp dữ liệu mà bạn có thể sử dụng để đào tạo các mô hình học máy, cung cấp trực quan hóa hoặc thúc đẩy phân tích công việc mạng để giải quyết vấn đề bảo mật đã chọn của bạn, khoa học dữ liệu có thể sẽ không giúp được gì cho bạn.

Sau khi bạn đã chọn một vấn đề và xác định nguồn cấp dữ liệu sẽ cho phép bạn xây dựng giải pháp dựa trên khoa học dữ liệu cho vấn đề, đã đến lúc bắt đầu xây dựng giải pháp của bạn. Điều này thực sự xảy ra trong một vòng lặp lặp đi lặp lại giữa bước 3 và 4 của Hình 12-1: bạn xây dựng một thứ gì đó, đánh giá nó, cải thiện nó, đánh giá lại nó, v.v.

Cuối cùng, khi hệ thống của bạn đã sẵn sàng, bạn triển khai nó, như thể hiện trong bước 5 của Hình 12-1. Miễn là hệ thống của bạn vẫn được triển khai, bạn sẽ cần quay lại và tích hợp các nguồn cấp dữ liệu mới khi chúng khả dụng, thử các phương pháp khoa học dữ liệu mới và triển khai lại các phiên bản mới của hệ thống.

Đặc điểm của một nhà khoa học dữ liệu bảo mật hiệu quả

Thành công trong khoa học dữ liệu bảo mật phụ thuộc rất nhiều vào thái độ của bạn. Trong phần này, chúng tôi liệt kê một số thuộc tính tinh thần mà chúng tôi thấy là quan trọng để thành công trong công việc khoa học dữ liệu bảo mật.

Cởi mở

Dữ liệu chứa đầy những điều bất ngờ và điều này phá vỡ những gì chúng tôi nghĩ rằng chúng tôi đã biết về một vấn đề. Điều quan trọng là bạn phải luôn cởi mở với dữ liệu chứng minh định kiến của bạn là sai. Nếu không, bạn sẽ bỏ lỡ những bài học quan trọng từ dữ liệu của mình và thậm chí đọc quá nhiều thành tiếng ôn ngẫu nhiên để thuyết phục bản thân về một lý thuyết sai lầm. May mắn thay, bạn càng nghiên cứu nhiều về khoa học dữ liệu bảo mật, bạn sẽ càng cởi mở hơn về việc "học hỏi" từ dữ liệu của mình và bạn sẽ càng hài lòng với lượng kiến thức ít ỏi mà mình biết và lượng kiến thức bạn phải học từ mỗi dữ liệu mới. Vấn đề. Theo thời gian, bạn sẽ vừa thích thú vừa mong đợi những điều bất ngờ từ dữ liệu của mình.

Trí tuệ vô biên

Các dự án khoa học dữ liệu xác khác với công nghệ phần mềm và

Các dự án CNTT yêu cầu khám phá dữ liệu để tìm ra các mẫu, ngoại lệ,

và xu hướng, sau đó chúng tôi tận dụng để xây dựng hệ thống của mình. Việc xác định những động lực này không dễ dàng: nó thường yêu cầu chạy hàng trăm thử nghiệm hoặc phân tích để hiểu được hình dạng tổng thể của dữ liệu của bạn và những câu chuyện ẩn bên trong. Một số người có động lực tự nhiên để chạy các thử nghiệm được thiết kế thông minh và tìm hiểu sâu hơn về dữ liệu của họ, gần như gây nghiện, trong khi những người khác thì không. Người đầu tiên là kiểu người có xu hướng thành công trong lĩnh vực khoa học dữ liệu. Do đó, sự tò mò là một yếu cầu trong lĩnh vực này bởi vì nó là điều làm nên sự khác biệt giữa khả năng hiểu sâu về dữ liệu của chúng ta so với hiểu nông can. Bạn càng có thể nuôi dưỡng thái độ tò mò khi xây dựng các mô hình và trực quan hóa dữ liệu của mình, hệ thống của bạn sẽ càng hữu ích.

Nỗi ám ảnh về kết quả

Khi bạn đã xác định được một vấn đề khoa học dữ liệu bảo mật tốt và đã bắt đầu lặp đi lặp lại thử các giải pháp và đánh giá chúng, bạn có thể bị ám ảnh về kết quả, đặc biệt là trong các dự án máy học. Đây là một dấu hiệu tốt. Ví dụ: khi tôi tham gia nhiều vào một dự án máy học, tôi có nhiều thử nghiệm chạy 24 giờ một ngày, 7 ngày một tuần.

Điều này có nghĩa là tôi có thể thức dậy nhiều lần trong đêm để kiểm tra trạng thái của các thử nghiệm và thường cần sửa lỗi cũng như khởi động lại các thử nghiệm lúc 3 giờ sáng. Tôi có xu hướng kiểm tra các thí nghiệm của mình trước khi đi ngủ mỗi tối và nhiều lần trong suốt cuối tuần.

Loại quy trình làm việc suốt ngày đêm này thường cần thiết để xây dựng hệ thống khoa học dữ liệu bảo mật hàng đầu. Không có nó, thật dễ dàng để giải quyết các kết quả tầm thường, không thoát ra khỏi lối mòn hoặc vượt qua các tắc nghẽn được tạo ra từ các giả định không đúng chổ về dữ liệu.

Chủ nghĩa hoài nghi về kết quả

Thật dễ dàng để đánh lừa bản thân khi nghĩ rằng bạn đang thành công trong một dự án khoa học dữ liệu bảo mật. Ví dụ: có lẽ bạn đã thiết lập đánh giá của mình không chính xác, vì vậy có vẻ như độ chính xác của hệ thống của bạn tốt hơn nhiều so với thực tế. Đánh giá hệ thống của bạn dựa trên dữ liệu quá giống với dữ liệu đào tạo của bạn hoặc quá khác với dữ liệu trong thế giới thực là một cạm bẫy phổ biến. Bạn cũng có thể vô tình chọn được các ví dụ từ trực quan hóa mạng mà bạn nghĩ là hữu ích nhưng hầu hết người dùng không tìm thấy nhiều giá trị trong đó. Hoặc có lẽ bạn đã làm việc chăm chỉ với phương pháp của mình đến mức tự thuyết phục bản thân rằng số liệu thống kê đánh giá là tốt, khi trên thực tế, chúng không đủ tốt để làm cho hệ thống của bạn trở nên hữu ích trong thế giới thực của bạn. Điều quan trọng là duy trì mức độ hoài nghi lành mạnh đối với kết quả của bạn, kèo một ngày nào đó bạn sẽ rơi vào tình huống xấu hổ.

Đi đâu từ đây

Chúng tôi đã trình bày rất nhiều trong cuốn sách này, nhưng chúng tôi cũng hầu như không làm trầy xước bề mặt. Nếu cuốn sách này đã thuyết phục bạn theo đuổi khoa học dữ liệu bảo mật một cách nghiêm túc, thì chúng tôi có hai đề xuất dành cho bạn: trước tiên, hãy bắt đầu áp dụng

những công cụ bạn đã học được trong cuốn sách này cho những vấn đề bạn quan tâm ngay lập tức. Thứ hai, đọc thêm sách về khoa học dữ liệu và khoa học dữ liệu bảo mật. Dưới đây là một số ví dụ về các vấn đề bạn có thể cân nhắc áp dụng các kỹ năng mới tìm thấy của mình để:

- Phát hiện tên miền độc hại
- Phát hiện URL độc hại
- Phát hiện tệp đính kèm email độc hại
- Trực quan hóa lưu lượng mạng để phát hiện sự bất thường
- Trực quan hóa các mẫu người gửi/người nhận email để phát hiện email lừa đảo

Để mở rộng kiến thức của bạn về các phương pháp khoa học dữ liệu, chúng tôi khuyên bạn nên bắt đầu từ những bài viết đơn giản trên Wikipedia về các thuật toán khoa học dữ liệu mà bạn muốn tìm hiểu thêm. Wikipedia là một nguồn tài nguyên lặp đi lặp lại của tác giả và có thể truy cập một cách đáng ngạc nhiên khi nói đến khoa học dữ liệu và nó miễn phí. Đối với những người muốn tìm hiểu sâu hơn, đặc biệt là về học máy, chúng tôi khuyên bạn nên chọn sách về đại số tuyến tính, lý thuyết xác suất, thống kê, phân tích đồ thị và phép tính đa biến hoặc tham gia các khóa học trực tuyến miễn phí. Học những nguyên tắc cơ bản này sẽ mang lại lợi ích cho phần còn lại của sự nghiệp khoa học dữ liệu của bạn, bởi vì chúng là nền tảng mà lĩnh vực của chúng tôi dựa vào. Ngoài việc tập trung vào những nguyên tắc cơ bản này, chúng tôi cũng khuyên bạn nên tham gia các khóa học hoặc đọc thêm sách “Ứng dụng” về Python, numpy, sklearn, matplotlib, seaborn, Keras và bất kỳ công cụ nào khác được đề cập trong cuốn sách này được sử dụng nhiều trong cộng đồng khoa học dữ liệu.

n T ống quan v ề b ộ d ū li ệu v à T ool s



Tất cả dữ liệu và mã cho cuốn sách này
có sẵn để tải về tại <http://www.malwaredatascience.com/>. Được cảnh báo: có phần
mềm độc hại Windows trong dữ liệu. Nếu bạn giải nén dữ liệu trên
một máy có công cụ chống vi-rút đang chạy trên đó, nhiều ví dụ về
phần mềm độc hại có thể sẽ bị xóa hoặc cách ly.

LƯU Ý Ông tôi đã sửa đổi một vài byte trong mỗi tệp thực thi của phần mềm độc hại để vô hiệu hóa nó khỏi
quá trình thực thi. Điều đó đang được nói, bạn không thể quá cẩn thận về nơi bạn lưu trữ nó.
Chúng tôi khuyên bạn nên lưu trữ nó trên một máy không chạy Windows được cách ly với mạng gia
đình hoặc doanh nghiệp của bạn.

Lý tưởng nhất là bạn chỉ nên thử nghiệm mã và dữ liệu trong một máy ảo bị cô lập. Để thuận tiện, chúng tôi đã cung cấp một phiên bản VirtualBox Ubuntu tại <http://www.malwaredatascience.com/> có dữ liệu và mã được tải sẵn vào đó, cùng với tất cả các thư viện nguồn mở cần thiết.

Tổng quan về Bộ dữ liệu

Bây giờ, hãy xem qua các bộ dữ liệu đi kèm với mỗi chương của cuốn sách này.

Chương 1: Phân tích phần mềm độc hại tĩnh cơ bản

Nhớ lại rằng trong Chương 1, chúng ta đã đi qua phân tích tĩnh cơ bản của một nhị phân phần mềm độc hại được gọi là ircbot.exe. Phần mềm độc hại này là một phần mềm cầy ghép, nghĩa là nó ẩn trên hệ thống của người dùng và chờ lệnh từ kẻ tấn công, cho phép kẻ tấn công thu thập dữ liệu riêng tư từ máy tính của nạn nhân hoặc đạt được mục đích xấu như xóa ổ cứng của nạn nhân. Mã nhị phân này có sẵn trong dữ liệu kèm theo cuốn sách này tại ch1/ircbot.exe.

Chúng tôi cũng sử dụng một ví dụ về fakepdfmalware.exe trong chương này (có tại ch1/fakepdfmalware.exe). Đây là chương trình phần mềm độc hại có biểu tượng Adobe Acrobat/PDF trên màn hình để lừa người dùng nghĩ rằng họ đang mở tài liệu PDF trong khi thực tế họ đang chạy chương trình độc hại và lây nhiễm vào hệ thống của họ.

Chương 2: Ngoài phân tích tĩnh cơ bản: Tháo gỡ x86

Trong chương này, chúng ta khám phá một chủ đề sâu hơn trong kỹ thuật đảo ngược phần mềm độc hại: phân tích quá trình tháo gỡ x86. Chúng tôi sử dụng lại ví dụ ircbot.exe từ Chương 1 trong chương này.

Chương 3: Giới thiệu tóm tắt về Phân tích động

Để thảo luận về phân tích phần mềm độc hại động trong Chương 3, chúng tôi thử nghiệm một ví dụ về ransomware được lưu trữ trong đường dẫn ch3/d676d9dfab6a4242258362b8ff579cfe6e5e6db3f0cdd3e0069ace50f80af1c5 trong dữ liệu đi kèm với cuốn sách này. Tên tệp tương ứng với hàm băm mật mã SHA256 của tệp. Không có gì đặc biệt về phần mềm tổng tiền này, mà chúng tôi có được bằng cách tìm kiếm cơ sở dữ liệu phần mềm độc hại của VirusTotal.com để biết các ví dụ về mã độc tổng tiền.

Chương 4: Xác định các chiến dịch tấn công sử dụng mạng phần mềm độc hại

Chương 4 giới thiệu ứng dụng phân tích mạng và trực quan hóa phần mềm độc hại. Để chứng minh những kỹ thuật này, chúng tôi sử dụng một tập hợp các mẫu phần mềm độc hại chất lượng cao được sử dụng trong các cuộc tấn công nổi tiếng, tập trung phân tích của chúng tôi vào một tập hợp các mẫu phần mềm độc hại có khả năng được tạo ra bởi một nhóm trong quân đội Trung Quốc được cộng đồng bảo mật gọi là Mối đe dọa dai dẳng nâng cao 1 (hoặc APT1 gọi tắt).

Các mẫu này và nhóm APT1 tạo ra chúng đã được phát hiện và được công bố bởi công ty an ninh mạng Mandiant. Trong báo cáo của mình (được trích dẫn tại đây) có tiêu đề "APT1: Vạch trần một trong những đơn vị gián điệp mạng của Trung Quốc" (<https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf>), Mandiant đã tìm thấy như sau:

- Từ năm 2006, Mandiant đã quan sát thấy APT1 xâm phạm 141 công ty trải rộng trên 20 ngành công nghiệp chính.
- APT1 có phương pháp tấn công được xác định rõ ràng, được mài giũa qua nhiều năm và được thiết kế để đánh cắp khối lượng lớn tài sản trí tuệ có giá trị.
- Sau khi APT1 thiết lập quyền truy cập, chúng định kỳ truy cập lại mạng của nạn nhân trong vài tháng hoặc vài năm và đánh cắp nhiều loại tài sản trí tuệ, bao gồm bản thiết kế công nghệ, quy trình sản xuất độc quyền, kết quả kiểm tra, kế hoạch kinh doanh, tài liệu định giá, thỏa thuận hợp tác và email và danh sách liên lạc từ lãnh đạo của các tổ chức nạn nhân.
- APT1 sử dụng một số công cụ và kỹ thuật mà chúng tôi chưa thấy các nhóm khác sử dụng, bao gồm hai tiện ích được thiết kế để đánh cắp email: GETMAIL và MAPIGET.
- APT1 duy trì quyền truy cập vào mạng nạn nhân trong trung bình 356 ngày.
- Khoảng thời gian dài nhất mà APT1 duy trì quyền truy cập vào mạng của nạn nhân là 1.764 ngày, hay 4 năm 10 tháng.
- Trong số các hành vi trộm cắp tài sản trí tuệ quy mô lớn khác, chúng tôi đã quan sát thấy APT1 đánh cắp 6,5 TB dữ liệu nén từ một tổ chức trong khoảng thời gian mười tháng.
- Tháng đầu tiên của năm 2011, APT1 đã xâm nhập thành công ít nhất 17 nạn nhân mới hoạt động trong 10 ngành công nghiệp khác nhau.

Như đoạn trích này của báo cáo cho thấy, các mẫu APT1 đã được sử dụng cho hoạt động gián điệp cấp quốc gia, có nguy cơ cao. Các mẫu này có sẵn trong dữ liệu đi kèm cuốn sách này tại ch4/data/APT1_MALWARE_FAMILYIES.

Chương 5: Phân tích mã chia sẻ

Chương 5 sử dụng lại các mẫu APT1 được sử dụng trong Chương 4. Để thuận tiện, các mẫu này cũng nằm trong thư mục Chương 5, tại ch5/data/APT1_MALWARE_FAMILYIES.

Chương 6: Tìm hiểu về công cụ phát hiện phần mềm độc hại dựa trên máy học và
Chương 7: Đánh giá hệ thống phát hiện phần mềm độc hại

Các chương khái niệm này không yêu cầu bất kỳ dữ liệu mẫu nào.

Chương 8: Xây dựng máy dò học máy

Chương 8 khám phá việc xây dựng các trình phát hiện phần mềm độc hại dựa trên máy học và sử dụng 1.419 tệp nhị phân mẫu làm bộ dữ liệu mẫu để đào tạo hệ thống phát hiện máy học của riêng bạn. Các tệp nhị phân này được đặt tại ch8/data/partitioned malware cho các mẫu phần mềm độc hại.

Bộ dữ liệu chứa 991 mẫu phần mềm lành tính và 428 mẫu phần mềm độc hại, và chúng tôi lấy dữ liệu này từ VirusTotal.com. Các mẫu này là đại diện, trong trường hợp phần mềm độc hại, cho loại phần mềm độc hại được phát hiện trên internet vào năm 2017 và, trong trường hợp phần mềm lành tính, cho loại tệp nhị phân mà người dùng đã tải lên VirusTotal.com vào năm 2017.

Chương 9: Trực quan hóa xu hướng phần mềm độc hại

Chương 9 khám phá trực quan hóa dữ liệu và sử dụng dữ liệu mẫu trong tệp ch9/code/malware_data.csv. Trong số 37.511 hàng dữ liệu trong tệp, mỗi hàng hiển thị một bản ghi của một tệp phần mềm độc hại riêng lẻ, khi nó được nhìn thấy lần đầu tiên, có bao nhiêu sản phẩm chống vi-rút đã phát hiện ra nó và loại phần mềm độc hại đó (ví dụ: Trojan horse, ransomware và sör). Dữ liệu này được thu thập từ VirusTotal.com.

Chương 10: Kiến thức cơ bản về Deep Learning

Chương này giới thiệu các mạng nơ-ron sâu và không sử dụng bất kỳ dữ liệu mẫu nào.

Chương 11: Xây dựng bộ phát hiện phần mềm độc hại mạng nơ-ron với Keras

Chương này hướng dẫn cách xây dựng bộ phát hiện phần mềm độc hại mạng thần kinh để phát hiện các tệp HTML độc hại và lành tính. Các tệp HTML lành tính là từ các trang web hợp pháp và các trang web độc hại là từ các trang web có gắng lây nhiễm cho nạn nhân thông qua trình duyệt web của họ. Chúng tôi đã lấy cả hai bộ dữ liệu này từ VirusTotal.com bằng cách sử dụng đăng ký trả phí cho phép truy cập vào hàng triệu mẫu trang HTML lành tính và độc hại.

Tất cả dữ liệu được lưu trữ tại thư mục gốc ch11/data/html. Phần mềm lành tính được lưu trữ tại ch11/data/html/benign_files và phần mềm độc hại được lưu trữ tại ch11/data/html/malicious_files. Ngoài ra, trong mỗi thư mục này là các thư mục con đào tạo và xác nhận. Các thư mục đào tạo chứa các tệp mà chúng tôi huấn luyện mạng nơ-ron trong chương này và các thư mục xác thực chứa các tệp mà chúng tôi kiểm tra mạng nơ-ron để đánh giá nó.

sự chính xác.

Chương 12: Tạo thành nhà khoa học dữ liệu

Chương 12 thảo luận về cách tạo thành nhà khoa học dữ liệu và không sử dụng bất kỳ dữ liệu mẫu nào.

Hướng dẫn triển khai công cụ

Mặc dù tất cả mã trong cuốn sách này là mã mẫu, nhằm mục đích thể hiện các ý tưởng trong sách và không được lấy toàn bộ và sử dụng trong thế giới thực, nhưng một số mã chúng tôi cung cấp có thể được sử dụng như một công cụ trong phân tích phần mềm độc hại của riêng bạn hoạt động, đặc biệt nếu bạn sẵn sàng mở rộng nó cho mục đích riêng của mình.

LƯU Ý Các dùng làm ví dụ và nơi khởi đầu cho các công cụ khoa học dữ liệu phần mềm độc hại chính thức, các công cụ này không được triển khai mạnh mẽ. Chúng đã được thử nghiệm trên Ubuntu 17 và dự kiến sẽ hoạt động trên nền tảng này, nhưng với một chút công việc xung quanh việc cài đặt đúng yêu cầu, bạn sẽ có thể có được các công cụ để hoạt động trên các nền tảng khác như macOS và các phiên bản khác của Linux khá dễ dàng .

Trong phần này, chúng ta sẽ đi qua các công cụ sơ khai được cung cấp trong cuốn sách này theo thứ tự mà chúng xuất hiện.

Trực quan hóa mạng tên máy chủ được chia sẻ

Công cụ trực quan hóa mạng tên máy chủ dùng chung được đưa ra trong Chương 4 và được đặt tại ch4/code/listing-4-8.py. Công cụ này trích xuất tên máy chủ từ các tệp phần mềm độc hại mục tiêu và sau đó hiển thị các kết nối giữa các tệp dựa trên tên máy chủ phổ biến có trong chúng.

Công cụ này lấy một thư mục phần mềm độc hại làm đầu vào và sau đó xuất ra ba tệp GraphViz mà sau đó bạn có thể hình dung. Để cài đặt các yêu cầu cho công cụ này, hãy chạy lệnh run bash install_requirements.sh trong thư mục ch4/code . Liệt kê A-1 hiển thị đầu ra “trợ giúp” từ công cụ, sau đó chúng ta thảo luận về ý nghĩa của các tham số.

cách sử dụng: Trực quan hóa tên máy chủ được chia sẻ giữa một thư mục mẫu phần mềm độc hại
[-h] target_path output_file malware_projection hostname_projection

đối số vị trí:

thư mục target_path chứa các mẫu phần mềm độc hại
tệp output_file để ghi tệp DOT vào
tệp malware_projection để ghi tệp DOT vào
tệp hostname_projection để ghi tệp DOT vào

đối số tùy chọn:

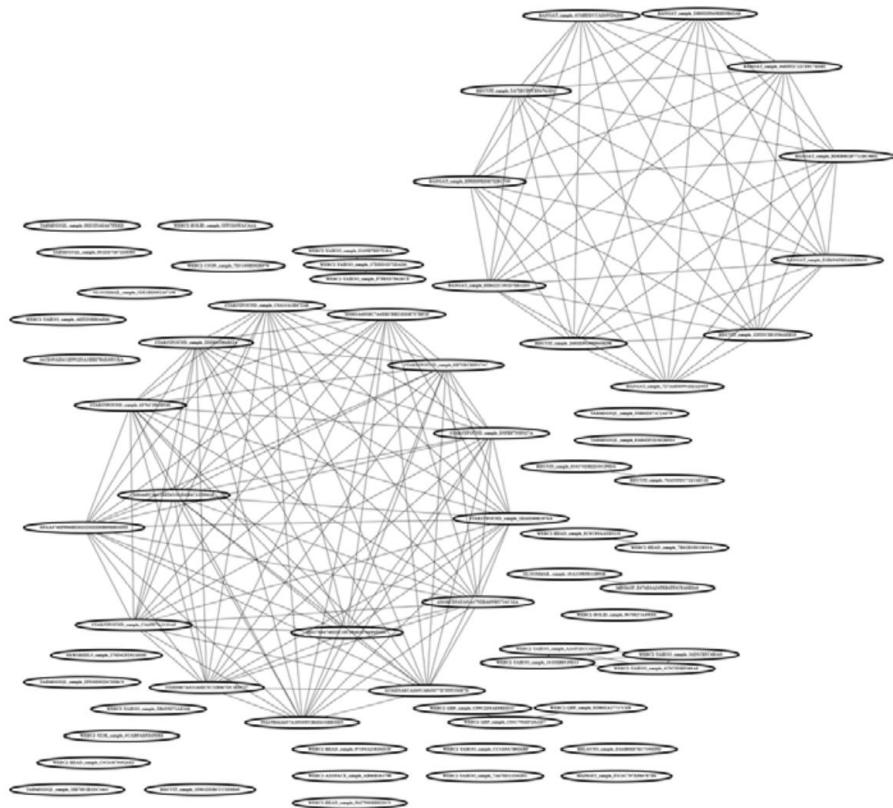
-h, --trợ giúp hiển thị thông báo trợ giúp này và thoát

Liệt kê A-1: Đầu ra trợ giúp từ công cụ trực quan hóa mạng tên máy chủ được chia sẻ được cung cấp trong Chương 4

Như được hiển thị trong Liệt kê A-1, công cụ trực quan hóa tên máy chủ được chia sẻ yêu cầu bốn đối số dòng lệnh: target_path , output_file , malware_projection và hostname_projection . Tham số target_path là đường dẫn đến thư mục mẫu phần mềm độc hại mà bạn muốn phân tích. Tham số output_file là đường dẫn đến tệp nơi chương trình sẽ ghi tệp GraphViz .dot đại diện cho mạng liên kết các mẫu phần mềm độc hại với tên máy chủ mà chúng chứa.

Các tham số malware_projection và hostname_projection cũng là các đường dẫn tệp và chỉ định vị trí nơi chương trình sẽ ghi các tệp .dot đại diện cho các mạng dẫn xuất này (để biết thêm về các phép chiếu mạng, xem Chương 4). Khi bạn đã chạy chương trình, bạn có thể sử dụng bộ GraphViz đã thảo luận trong Chương 4 và 5 để trực quan hóa mạng. Ví dụ: bạn có thể sử dụng lệnh fdp malware_projection.dot -Tpng -o malware

_projection.png để tạo một tệp giống như tệp .png được hiển thị trong Hình A-1 trên bộ dữ liệu phần mềm độc hại của riêng bạn.



Hình A-1: Đầu ra mẫu từ công cụ trực quan hóa tên máy chủ được chia sẻ trong Chương 4

Trực quan hóa mạng hình ảnh được chia sẻ

Chúng tôi trình bày một công cụ trực quan hóa mạng hình ảnh được chia sẻ trong Chương 4, có tại ch4/code/listing-4-12.py. Chương trình này hiển thị các mối quan hệ mạng giữa các mẫu phần mềm độc hại dựa trên hình ảnh nhưng mà chúng chia sẻ.

Công cụ lấy một thư mục phần mềm độc hại làm đầu vào và sau đó xuất ra ba GraphViz mà sau đó bạn có thể hình dung. Để cài đặt các yêu cầu cho công cụ này, hãy chạy lệnh run bash install_requirements.sh trong thư mục ch4/code . Hãy thảo luận về các tham số trong đầu ra "trợ giúp" từ công cụ (xem Liệt kê A-2).

cách sử dụng: Trực quan hóa các mối quan hệ hình ảnh được chia sẻ giữa một thư mục mẫu phần mềm độc hại

```
[ -h ] target_path output_file malware_projection resource_projection
```

đối số vị trí:

target_path	thư mục với các mẫu phần mềm độc hại
output_file	tệp để ghi tệp DOT vào
tệp malware_projection	dể ghi tệp DOT vào
tệp resource_projection	dể ghi tệp DOT vào

đối số tùy chọn:

-h, --trợ giúp	hiển thị thông báo trợ giúp này và thoát
----------------	--

Liệt kê A-2: Đầu ra trợ giúp từ công cụ trực quan hóa mạng tài nguyên được chia sẻ được đưa ra trong Chương 4

Như được hiển thị trong Liệt kê A-2, công cụ trực quan hóa mối quan hệ hình ảnh được chia sẻ yêu cầu bốn đối số dòng lệnh: target_path , output_file , malware_projection và resource_projection . Giống như trong chương trình tên máy chủ được chia sẻ, ở đây target_path là đường dẫn đến thư mục mẫu phần mềm độc hại mà bạn muốn phân tích và output_file là đường dẫn đến tệp nơi chương trình sẽ ghi tệp GraphViz .dot biểu thị biểu đồ hai bên liên kết mẫu phần mềm độc hại vào hình ảnh mà chúng chứa (đồ thị hai bên được thảo luận trong Chương 4). malware_projection và resource_projection _

tham số cũng là đường dẫn tệp và chỉ định vị trí mà chương trình sẽ ghi tệp .dot đại diện cho các mạng này.

Như với chương trình tên máy chủ được chia sẻ, khi bạn đã chạy chương trình, bạn có thể sử dụng bộ GraphViz để trực quan hóa mạng. Ví dụ: bạn có thể sử dụng lệnh fdp resource_projection.dot -Tpng -o resource_projection.png trên bộ dữ liệu phần mềm độc hại của riêng bạn để tạo một tệp giống như tệp .png được hiển thị trong Hình 4-12 trên trang 55.

Trực quan hóa tương tự phần mềm độc hại

Trong Chương 5, chúng tôi thảo luận về sự giống nhau của phần mềm độc hại và phân tích và trực quan hóa mã được chia sẻ. Công cụ mẫu đầu tiên mà chúng tôi cung cấp có trong ch5/code/listing_5_1.py. Công cụ này lấy một thư mục chứa phần mềm độc hại làm đầu vào và sau đó trực quan hóa các mối quan hệ mã được chia sẻ giữa các mẫu phần mềm độc hại trong thư mục. Để cài đặt các yêu cầu cho công cụ này, hãy chạy lệnh run bash install_requirements.sh trong thư mục ch5/code . Liệt kê A-3 hiển thị đầu ra trợ giúp cho công cụ.

cách sử dụng: list_5_1.py [-h] [--jaccard_index_threshold THRESHOLD]
target_directory output_dot_file

Xác định sự giống nhau giữa các mẫu phần mềm độc hại và xây dựng biểu đồ tương đồng

đối số vị trí:

target_directory	Thư mục chứa phần mềm độc hại
output_dot_file	Nơi lưu tệp DOT biểu đồ đầu ra

đối số tùy chọn:

-h, --help hiển thị thông báo trợ giúp này và thoát

--jaccard_index_threshold NGƯỜNG, -j NGƯỜNG

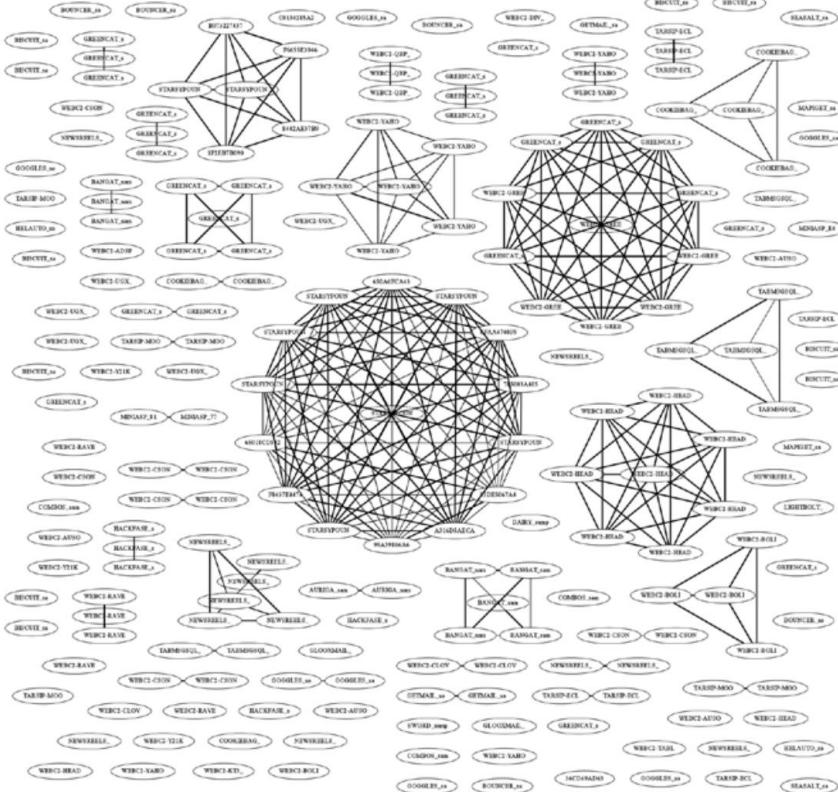
Ngořng trên đó để tạo một 'cạnh' giữa
mẫu

Liệt kê A-3: Đầu ra trợ giúp từ công cụ trực quan hóa tính tương tự của phần mềm độc hại được cung cấp trong Chương 5

Khi bạn chạy công cụ phân tích mã được chia sẻ này từ dòng lệnh, bạn cần chuyển vào hai đối số dòng lệnh: `target_directory` và `output_dot_file`. Bạn có thể sử dụng đối số tùy chọn, `jaccard_index_threshold`, để đặt ngưỡng mà chương trình sử dụng với chỉ số Jaccard giống nhau giữa hai mẫu để xác định có tạo cạnh giữa các mẫu đó hay không. Chỉ số Jaccard được thảo luận chi tiết trong Chương 5.

Hình A-2 hiển thị đầu ra mẫu từ công cụ này sau khi bạn kết xuất đầu ra_dot_file bằng lệnh fdp output_dot_file.dot -Tpng -o

`similar_network.png`. Đây là mạng mã chia sẻ được suy luận bởi công cụ dành cho các mẫu phần mềm độc hại APT1 mà chúng tôi vừa mô tả.



Hình A-2: Đầu ra mẫu từ công cụ phân tích tính tương tự của phần mềm độc hại được đưa ra trong Chương 5

Hệ thống tìm kiếm tương tự phần mềm độc hại

Công cụ ước tính chia sẻ mã thứ hai mà chúng tôi cung cấp trong Chương 5 được cung cấp trong ch5/code/listing_5_2.py. Công cụ này cho phép bạn lập chỉ mục hàng nghìn mẫu trong cơ sở dữ liệu, sau đó thực hiện tìm kiếm sự tương đồng trên chúng với một mẫu phần mềm độc hại truy vấn, cho phép bạn tìm các mẫu phần mềm độc hại có khả năng chia sẻ mã với mẫu đó. Để cài đặt các yêu cầu cho công cụ này, hãy chạy lệnh run bash install_requirements.sh trong thư mục ch5/code . Liệt kê A-4 hiển thị đầu ra trợ giúp cho công cụ.

cách sử dụng: list_5_2.py [-h] [-l TÌM KIẾM] [-s TÌM KIẾM] [-c BÌNH LUẬN] [-w]

Hệ thống tìm kiếm chia sẻ mã đơn giản cho phép bạn xây dựng cơ sở dữ liệu về mẫu phần mềm độc hại (được lập chỉ mục theo đường dẫn tệp) và sau đó tìm kiếm các mẫu tương tự đưa ra một số mẫu mới

đối số tùy chọn:

-h, --trợ giúp	hiển thị thông báo trợ giúp này và thoát
-l LOAD, --load LOAD	Đường dẫn đến thư mục chứa phần mềm độc hại hoặc cá nhân
	tệp phần mềm độc hại, để lưu trữ trong cơ sở dữ liệu
-s TÌM KIẾM, --search TÌM KIẾM	Tệp phần mềm độc hại riêng lẻ để thực hiện tìm kiếm tương tự
	TRÊN
-c BÌNH LUẬN, --comment BÌNH LUẬN	Nhận xét về đường dẫn mẫu phần mềm độc hại
-w, --wipe	Xóa cơ sở dữ liệu mẫu

Liệt kê A-4: Đầu ra trợ giúp từ hệ thống tìm kiếm tương tự phần mềm độc hại được đưa ra trong Chương 5

Công cụ này có bốn chế độ mà nó có thể chạy. Chế độ đầu tiên, LOAD , tải phần mềm độc hại vào cơ sở dữ liệu tìm kiếm tương tự và lấy một đường dẫn làm tham số của nó, đường dẫn này sẽ trả đến một thư mục chứa phần mềm độc hại trong đó. Bạn có thể chạy LOAD nhiều lần và thêm phần mềm độc hại mới vào cơ sở dữ liệu mỗi lần.

Chế độ thứ hai, TÌM KIẾM , lấy đường dẫn đến một tệp phần mềm độc hại riêng lẻ làm tham số của nó và sau đó tìm kiếm các mẫu tương tự trong cơ sở dữ liệu. Chế độ thứ ba, BÌNH LUẬN , lấy đường dẫn mẫu phần mềm độc hại làm đối số và sau đó nhắc bạn nhập nhận xét bằng văn bản ngắn về mẫu đó. Ưu điểm của việc sử dụng tính năng BÌNH LUẬN là khi bạn tìm kiếm các mẫu tương tự như mẫu phần mềm độc hại truy vấn, bạn sẽ thấy các nhận xét tương ứng với mẫu tương tự, do đó làm phong phú thêm kiến thức của bạn về mẫu truy vấn.

Chế độ thứ tư, xóa , xóa tất cả dữ liệu trong cơ sở dữ liệu tìm kiếm tương tự, trong trường hợp bạn muốn bắt đầu lại và lập chỉ mục cho một bộ dữ liệu phần mềm độc hại khác. Liệt kê A-5 hiển thị một số kết quả đầu ra mẫu từ truy vấn TÌM KIẾM , cho bạn biết kết quả đầu ra từ công cụ này trông như thế nào. Ở đây, chúng tôi đã lập chỉ mục các mẫu APT1 được mô tả trước đây bằng cách sử dụng lệnh LOAD và sau đó đã tìm kiếm cơ sở dữ liệu để tìm các mẫu tương tự như một trong các mẫu APT1.

Hiển thị các mẫu tương tự với WEBC2-GREENCAT_sample_E54CE5F0112C9FD86DB17E85A5E2C5 tên mẫu	mã chia sẻ
[*] WEBC2-GREENCAT_sample_55FB1409170C91740359D1D96364F17B	0,9921875
[*] GREENCAT_sample_55FB1409170C91740359D1D96364F17B	0,9921875
[*] WEBC2-GREENCAT_sample_E83F60FB0E0396EA309FAF0AED64E53F	0,984375
[bình luận] Mẫu này được xác định là chắc chắn đến từ vùng liên tục nâng cao nhóm mối đe dọa được quan sát vào tháng 7 năm ngoái trên mạng Bờ Tây của chúng tôi	
[*] GREENCAT_sample_E83F60FB0E0396EA309FAF0AED64E53F 0,984375	

Liệt kê A-5: Đầu ra mẫu cho hệ thống tìm kiếm tương tự phần mềm độc hại được đưa ra trong Chương 5

Hệ thống phát hiện phần mềm độc hại Machine Learning

Công cụ cuối cùng bạn có thể sử dụng trong công việc phân tích phần mềm độc hại của riêng mình là trình phát hiện phần mềm độc hại máy học được sử dụng trong Chương 8, có thể tìm thấy tại ch8/malcomplete_detector.py. Công cụ này cho phép bạn đào tạo hệ thống phát hiện phần mềm độc hại trên phần mềm độc hại và phần mềm lành tính, sau đó sử dụng hệ thống này để phát hiện xem một mẫu mới là độc hại hay lành tính. Bạn có thể cài đặt các yêu cầu cho công cụ này bằng cách chạy lệnh bash install.sh trong thư mục ch8/code .

Liệt kê A-6 hiển thị đầu ra trợ giúp cho công cụ này.

cách sử dụng: Hệ thống phát hiện phần mềm độc hại học máy [-h]

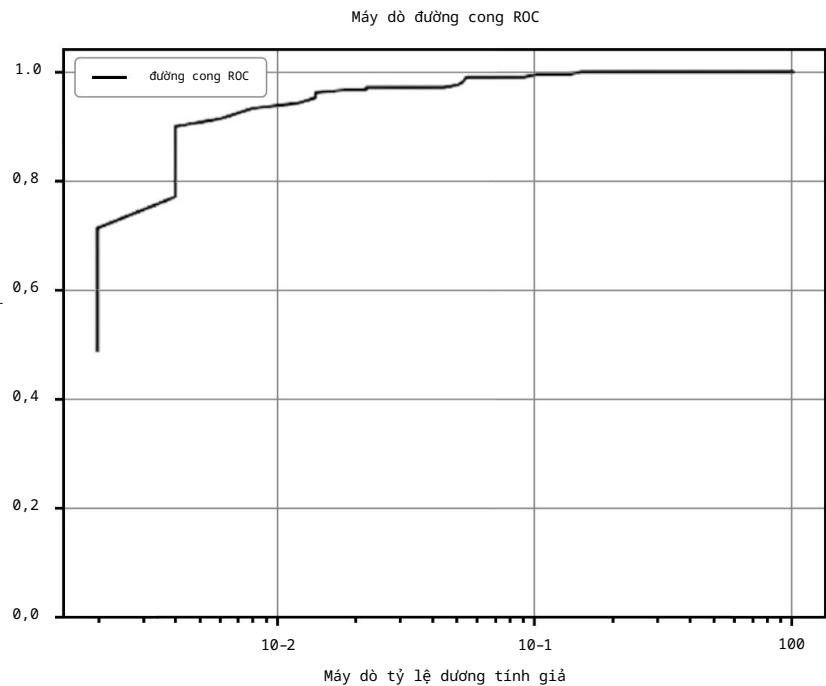
```
[--malware_paths MALWARE_PATHS]
[--benignware_paths BENIGNWARE_PATHS]
[--scan_file_path SCAN_FILE_PATH]
[--đánh giá]
```

đối số tùy chọn:

-h, --help hiển thị thông báo trợ giúp này và thoát	
--malware_paths MALWARE_PATHS	Đường dẫn đến tập đào tạo phần mềm độc hại
--benignware_paths BENIGNWARE_PATHS	Đường dẫn đến tập đào tạo phần mềm lành tính
--scan_file_path SCAN_FILE_PATH	Tệp để quét
--đánh giá	Thực hiện xác thực chéo

Liệt kê A-6: Đầu ra trợ giúp cho công cụ phát hiện phần mềm độc hại máy học được cung cấp trong Chương 8

Công cụ này có ba chế độ mà nó có thể được chạy. Chế độ đánh giá , kiểm tra độ chính xác của hệ thống trên dữ liệu bạn chọn để huấn luyện và đánh giá hệ thống. Bạn có thể gọi chế độ này bằng cách chạy python Complete_detect.py --malware_paths <đường dẫn đến thư mục chứa phần mềm độc hại> --benignware_paths <đường dẫn đến thư mục chứa phần mềm lành tính> --evaluate. Lệnh này sẽ gọi một số matplotlib hiển thị đường cong ROC của máy dò (đường cong ROC được thảo luận trong Chương 7). Hình A-3 cho thấy một số đầu ra mẫu từ chế độ đánh giá .



Hình A-3: Đầu ra mẫu từ công cụ phát hiện phần mềm độc hại được cung cấp trong Chương 8, chạy trong chế độ đánh giá

Chế độ đào tạo đào tạo một mô hình phát hiện phần mềm độc hại và lưu nó vào đĩa. Bạn có thể gọi chế độ này bằng cách chạy python Complete_detector.py -malware_paths <đường dẫn đến thư mục chứa phần mềm độc hại trong đó> --benignware_paths <đường dẫn đến thư mục chứa phần mềm độc hại trong đó>. Lưu ý rằng sự khác biệt duy nhất giữa lời gọi lệnh này và lời gọi chế độ đánh giá là chúng ta đã bỏ cờ --evaluate . Kết quả của lệnh này là nó tạo ra một mô hình mà nó lưu vào một tệp có tên là save_detector.pkl, được lưu trong thư mục làm việc hiện tại của bạn.

Chế độ thứ ba, quét w, tải tệp save_detector.pkl rồi quét tệp tar get, dự đoán xem tệp đó có độc hại hay không. Đảm bảo rằng bạn đã chạy chế độ đào tạo trước khi chạy quét. Bạn có thể quét bằng cách chạy python Complete_detector.py -scan_file_path <PE EXE file> trong thư mục mà bạn đã đào tạo hệ thống. Đầu ra sẽ là xác suất đích là tệp độc hại.

Mục lục

- Lưu ý: Số trang đề cập đến hình và bảng được
theo sau bởi một f hoặc t in nghiêng tương ứng.
- một
- chức năng kích hoạt
 - phổ biến, 178t-180t
 - được xác định,
178 hàm `add_edge`, 41
hàm `add_node`, 49-50 hàm
`add_question`, 112 thêm lệnh
số học, 15
 - ADS (Luồng dữ liệu thay thế), 29
kẻ tấn công Mối đe dọa dai dẳng nâng cao 1
 - nhóm. Xem kẻ tấn công APT1
nhóm
 - các mối đe dọa dai dẳng nâng cao (APT), 60
dòng phần mềm độc hại Allapple.A, 157,
157f Luồng dữ liệu thay thế (ADS), 29 kỹ
thuật chống tháo gỡ, 22 lệnh gọi API,
32-33, 33f hàm
`apply_hashing_trick`, 138 APT1 (Mối đe
dọa dai dẳng nâng cao 1) nhóm kẻ tấn công,
37-39, 38f, 45-47, 45f-47f, 61,
61f, 76, 76f, 86, 222-223 APT
 - (các mối đe dọa
dai dẳng nâng cao), 60 dòng Trojan ArchSMS,
55 khu vực dưới đường cong (AUC) ,
209-210, 210f, 213 lệnh số học, 15,
 - 15t `.asarray` method, 142 hợp ngữ,
định nghĩa, 12. Xem thêm
hợp ngữ x86
 - AT&T, 43 tuổi
 - Cú pháp AT&T, 13
 - thuộc tính, 37
 - thêm vào nút và cạnh, 42 và cạnh,
48-51
 - AUC (diện tích dưới đường cong), 209-210,
210f, 213
 - mạng thần kinh mã hóa tự động, 194-
195, 195f tạo
tính năng tự động, 188

thông số nén_data_weight , 103
 thông số nén_data , 103-104
 các nhánh có điều kiện, được xác định, 15
 luồng điều khiển, 17
 đồ thị, 19-20, 19f
 hướng dẫn, 17-18
 thanh ghi, 14-15
 mạng thần kinh tích chập (CNN), 193-194, 194f

 Các thanh ghi CPU, 13-15, 14f
 thanh ghi mục đích chung, 13-14
 thanh ghi luồng ngăn xếp và điều khiển, 14-15
 mô-đun cross-validation , 151
 xác thực chéo, 150-153, 151f, 153f
 Nền tảng phần mềm CuckooBox, 27, 33-34,
 59
 “lời nguyền của kích thước,” 92
 hàm cv_evaluate , 151

 Đ.

 họ phần mềm độc hại dapato, 62, 67f-68f,
 70f-72f
 Các đối tượng DataFrame , 158-161
 hướng dẫn di chuyển dữ liệu, 15-20, 16t
 khối cơ bản, 19-20, 19f
 đồ thị luồng điều khiển, 19-20, 19f
 hướng dẫn luồng điều khiển, 17-18
 hướng dẫn ngăn xếp, 16-17
 khoa học dữ liệu, iii, iv
 áp dụng cho phần mềm độc hại, v
 tầm quan trọng của, iv-v
 phần .data (ở định dạng tệp PE), 4
 gói dateutil , 164
 lệnh số học tháng mười hai, 15
 ranh giới quyết định, 93-98, 95f-98f
 xác định với k-hàng xóm gần nhất, 97-98, 97f-98f
 xác định bằng hồi quy logistic, 96-97, 96f-97f
 mô hình máy học overfit, 100, 101f

 mô hình máy học phù hợp, 99, 99f

 mô hình học máy phù hợp, 100, 100f

 nguồn quyết định, 149
 Lớp DecisionTreeClassifier , 130

 cây quyết định, 109-115, 109f-110f, 113f-114f
 máy dò dựa trên cây quyết định, 129
 nhập mô-đun, 129
 khởi tạo dữ liệu đào tạo mẫu, 130

 các lớp khởi tạo, 130
 mã mẫu, 133-134
 huấn luyện, 130-131
 hình dung, 131-133, 132f
 câu hỏi tiếp theo, 111
 giới hạn độ sâu hoặc số lượng câu hỏi, 111-112
 mã giả cho, 112-113
 nút gốc, 110-111
 khi nào nên sử dụng, 114-115
 học sâu, 175-197, 216. Xem thêm
 mạng lưới thần kinh
 tạo tính năng tự động, 188
 xây dựng mạng lưới thần kinh, 182-188
 té bào thần kinh, 176
 giải phẫu của, 177-180
 mạng của, 180-181
 tổng quan, 176-177
 huấn luyện mạng lưới thần kinh, 189-193
 các loại mạng thần kinh, 193-197
 định lý xấp xỉ phổ quát, 181-182

 mạng lưới thần kinh sâu. Xem mạng lưới thần kinh

 Chức năng dày đặc , 200-201
 mô tả phương pháp, 159
 đánh giá độ chính xác của phát hiện, 119-126,
 146-153
 lãi suất cơ bản và độ chính xác, 124-126
 ảnh hưởng của lãi suất cơ bản đến độ chính xác, 124-125
 ước tính độ chính xác trong môi trường triển khai, 125-126

 với xác thực chéo, 150-153, 151f, 153f

 mạng lưới thần kinh, 209-211, 210f-211f
 kết quả phát hiện có thể, 120, 120f

 với các đường cong ROC, 123-124, 123f,
 147-150, 150f
 tỷ lệ dương tính đúng và sai, 120-124
 mối quan hệ giữa, 121-122, 121f-122f

 Đường cong ROC, 123-124, 123f

Lớp DictVectorizer , 128-130 đồ
thì có hướng, 180 hàm
khoảng cách, 107 DLL (thư
viện liên kết động), 13 tiêu đề DOS
(định dạng tệp PE), 3 định dạng .dot,
42 dữ liệu được
tải xuống động, 22-23 phân tích động, 25-
34 túi mô hình tính năng,
63 bộ dữ liệu cho, 222 cho
việc tháo gỡ, 26
hạn chế của, 33-34
cho khoa học dữ liệu về
phản mềm độc hại, 26 hành vi
diễn hình của phản mềm độc hại,
27 sử dụng malwr.com, 26-
33 kết quả phân tích, 28-
33 hạn chế, 33
tải tệp, 27-28 tính
tương tự dựa trên lệnh gọi API động,
72, 72f
thư viện liên kết động (DLL), 13

e

Thanh ghi EAX, 14
thanh ghi EBP, 14
thanh ghi EBX, 14
thanh ghi ECX, 14
cạnh, 37
thuộc tính thêm, 42
thêm vào mạng quan hệ chia sẻ, 41
thêm thuộc
tính trực quan vào, màu 48-51, nhãn
văn bản 49,
49f, chiều rộng 50-
51 , 48-49, 48f
thanh ghi EDX, 14
thanh ghi EFLAGS, 15
thanh ghi EIP, 14-15
chức năng kích hoạt ELU, 179t
diễn vào, 3, 19
tham số epoch , 206
thanh ghi ESP, 14
hàm khoảng cách euclidean , 107 khoảng
cách Euclidean, 107 hàm đánh
giá , 148 đánh giá ché
độ, 231-232 đánh giá hệ
thống phát hiện phản mềm độc hại.
Xem đánh giá độ chính
xác của
phát hiện hàm export_graphviz ,
132 hàm extract_features , 204-205 Lớp
trình trợ giúp ExtractImages , 56-57

F

fakepdfmalware.exe, 7
âm tính giả, đã xác định, 120, 120f
dương tính giả, 120, 120f
tỷ lệ cơ bản và độ chính xác, 124-126
tỷ lệ dương tính giả,
121 mối quan hệ giữa tỷ lệ dương tính
thật và sai, 121-122, 121f-
122f đường
cong ROC, 123-124, 123f công
cụ fdp , 43-45, 45f, 76
mô-đun trích xuất tính năng , 129
trích xuất tính năng, 134-138
Nhập tính năng Bảng địa chỉ, 136 trình
phát hiện phản mềm độc hại dựa trên
máy học, 90-92, 141-142 N-
gram, 136 -137 Tiêu
đè Portable Executable
các tính năng, 135-
136 phân tích mã được chia sẻ,
73, 75 tính năng
chuỗi, 135 đào tạo mạng thần kinh với
gói Keras , 203-204
tại sao không thể sử dụng tất cả các
tính năng có thể cùng
một lúc, 137-138 Lớp
FeatureHasher , 140-141 băm tính năng.
Xem không gian tính năng thủ thuật
băm, 93-98, 94f-98f mạng thần kinh
chuyển
tiếp nguồn cấp dữ liệu, 181, 181f, 193 hàm
fit_generator , 204-206,
208, 212, 214
phương thức khớp ,
130-131, 142 cờ, đã xác định, định dạng 15 chuỗi, 70 lan tr
g

Chức năng kích hoạt Gaussian, mạng đối
thủ chung 179t
(GAN) , 195-196 tham
số trình tạo , 206 hàm
get_database , 80-82 hàm
get_string_features , 141-142,
144 hàm
get_strings , 82 hàm
get_training_data , 143 hàm
get_training_paths , 143
Tiện ích GETMAIL, hàm
getstrings 223 , 73-74
-cờ G , 44

chỉ số gini, 132, giám
 độ dốc 132f, 105, 190
 Trình tạo đồ thị, 41, 52-53
 phân tích hình ảnh đồ họa, 7-8
 chuyển đổi các tệp .ico đã trích
 xuất thành đồ
 họa .png, 8 tạo thư mục để chứa các hình
 ảnh đã trích
 xuất, 7-8 trích xuất tài nguyên hình
 ảnh bằng
 wrestool, 8
 GraphViz, 76 trình phát hiện dựa
 trên cây quyết
 định, 131-133, 132f phân tích mạng
 phản mềm độc hại, 43-51 thêm
 thuộc tính trực quan cho
 các nút và cạnh, 48-51
 công cụ fdp, 44-45, 45f
 công cụ gọn
 gàng, 47-48, 47 tham
 số f, 44 công cụ sfdp,
 46-47, đồ thị tương tự 46f, 76 biến ground_truth, 130 trên lệnh gọi API động 73-
 75, tương tự
h
 thủ thuật băm (băm tính năng), 138-
 141
 mã hoàn chỉnh cho, 139-140
 Lớp FeatureHasher, 140-141
 triển khai, 138-139 lớp
 ẩn, 181 biểu đồ
 (biểu đồ thanh), 168-170, 168f-169f
 đôi số
 hostname_projection, 225 siêu phẳng,
 96, 97f
 ~
 IAT. Xem Bộ công cụ icoutils của
 Bảng Địa chỉ Nhập,
 5 IDA Pro,
 12 phần .idata (nhập) (ở định dạng tệp
 PE), 4 Chức
 năng kích hoạt danh tính, 178t Bảng
 Địa chỉ Nhập (IAT), 4 kết xuất bằng
 pefile, 6-7 tính năng trích
 xuất, 136 phân tích tương
 tự dựa trên, 71, phân tích nhập 71f,
 lệnh số học inc 6-7,
 15 thu được thông tin, 113 Chức
 năng nhập, 200-201

tính tương tự dựa trên trình tự
 lệnh, giới hạn
 68f của, tổng quan 68-
 70, cú pháp 67-68
 của Intel, 13 Trò
 chuyện chuyển tiếp Internet
 (IRC), 2 chức năng
 int, 148 lập chỉ mục
 đảo ngược, 82
 bot ircbot.exe, 2 phân
 tách, phân tích
 20-21, 5-7 kết xuất
 IAT, 6-7 chuỗi phân tích, 9-10

J

đổi số jaccard_index_threshold, hàm
 jaccard
 227-228, 73
 Chỉ số Jaccard, đồ thị tương
 tự xây dựng 61, 65, 65f, tương tự dựa
 trên lệnh gọi API động 73-
 75, tương tự
 dựa trên trình tự lệnh 72,
 phương pháp
 minhash 68, so sánh tương
 tự theo tỷ lệ 77-79, tương tự dựa trên
 chuỗi 77, hướng dẫn 70 jge,
 18 hướng dẫn jmp,
 18 chức năng ghép
 nối, 171-172

K

kaspersky, 62
 Gói Keras, xây dựng mạng thần kinh
 với, mô hình biên dịch 199-
 214, 202-203, 202f xác định kiến trúc
 của mô hình, mô hình đánh giá 200-
 202, các
 lớp 209-211, 210f-211f, mô
 hình lưu
 và tải 200,
 cú pháp 209, đào tạo 200 mô hình,
 203-209, 211-
 214 lệnh gọi lại tích hợp, 212 lệnh
 gọi lại tùy chỉnh, 213-
 214, trình tạo dữ liệu 214f, 204-
 207, trích xuất tính năng 207f, dữ
 liệu xác thực 203-204, 207-209,
 208f keylogger, 158, 168f, 172f-
 173f, 173

- Lớp KFold , 151-152
 Xác thực chéo K-fold , 151
 k-hàng xóm gần nhất , 105-109, 106f, 108f
 xác định ranh giới quyết định với , 97-
 98, 97f-98f
 hồi quy logistic so với , 108-109
 toán phía sau , 107
 mã giả cho , 107
 khi nào dùng , 109
- l**
- thuộc tính nhãn , 50-51
 mô đun con lớp , 200-201
 lea hướng dẫn , 16
 Chức năng kích hoạt ReLU bị rò rỉ , 179t
 tham số learn_parameters , 103
 tháo gỡ tuyến tính , 12
 giới hạn của , 12
 phân tích mã được chia sẻ , 67-68
 Chế độ TÀI , 229
 hàm logistic_function , 103-
 104, 104f
 hàm logistic_regression , 103
 hồi quy logistic , 102-105, 103f-
 104f, 154
 giảm độ dốc , 105
 xác định ranh giới quyết định với , 96-
 97, 96f-97f
 k-hàng xóm gần nhất so với , 108-109
 giới hạn của , 102
 toán phía sau , 103-104
 biểu đồ hàm logistic , 104f
 mã giả cho , 103
 khi nào dùng , 105
 mạng bộ nhớ ngắn hạn dài (LSTM) , 196
- Phòng thí nghiệm quốc gia Los Alamos , 41
 thông số tồn thắt , 201-202
- m**
- trình phát hiện phần mềm độc hại dựa
 trên máy học , 89-117, 127-154
 xây dựng máy dò cơ bản , 129
 mã mẫu , 133-134
 huấn luyện , 130-131
 hình dung , 131-133, 132f
 tổng quan tòa nhà , 90-93
 thu thập các ví dụ đào tạo , 90-91
 thiết kế các tính năng tốt , 92
- trích xuất các tính năng , 90-92
 lý do cho , 89-90
 hệ thống kiểm tra , 90, 93
 hệ thống đào tạo , 90, 92-93
 xây dựng máy dò trong thế giới
 thực , 141-146
- mã hoàn chỉnh cho , 144-146
 trích xuất tính năng , 141-142
 chạy máy dò trên các nhị phân
 mới , 144
 huấn luyện , 142-143
 tập dữ liệu cho , 224
 ranh giới quyết định , 93-98, 95f-
 98f
- dánh giá hiệu suất của
 máy dò , 146
 xác thực chéo , 150-153, 151f,
 153f
- Đường cong ROC , 147-150, 150f
 tách dữ liệu thành tập huấn luyện và
 kiểm tra , 148-149
- trích xuất đặc trưng , 134-138
- Nhập các tính năng của
 Bảng Địa chỉ , 136
- N-gam , 136-137
 Các tính năng tiêu đề có thể
 thực thi được , 135-136
 tính năng chuỗi , 135
 tại sao không thể sử dụng tất cả các
 tính năng có thể cùng một lúc , 137-138
- dấu cách , 93-98, 94f-98f
- thủ thuật băm , 138-141
 mã hoàn chỉnh cho , 139-140
 Lớp FeatureHasher , 140-141
 thực hiện , 138-139
- trang bị thừa và trang bị thiếu , 98-99,
 99f-101f
- các thuật toán được giám sát và
 không được giám sát , 93
- thuật ngữ và khái niệm , 128-129
- công cụ cho , 230-232, 231f
- thuật toán truyền thống so với , 90
- các loại thuật toán , 101, 102f
 cây quyết định , 109-115,
 109f-110f, 113f-114f
 k-hàng xóm gần nhất , 97-98,
 97f-98f, 105-109, 106f, 108f
 hồi quy logistic , 96-97, 96f-
 97f, 102-105, 103f-104f
 rừng ngẫu nhiên , 115-116, 116f
- đối số malware_projection , 52, 225-
 227

đánh giá phát hiện phần mềm độc hại.

Xem đánh giá độ chính xác
của phát hiện

phân tích mạng phần mềm độc hại, 35-58, thuộc
tính 36f, đã xác định, 37

mạng luồng cyclic, 37-39, 38f tạo mạng

mối quan hệ gọi lại được chia
sẻ, 51-54, 225-226, mã 226f cho,
52-54 nhập mô-
đun, phân tích cú
pháp 51-52 đổi số dòng lệnh, 52

lưu mạng vào đĩa, 54 tạo
mạng quan hệ hình
ảnh được chia sẻ, 54-58, 55f,
226-227 trích xuất nội dung đồ họa, 57

phân tích đổi số ban đầu
và mã tái

tệp, 55-57 lưu mạng vào đĩa, 58 bộ
dữ liệu cho, 222-223 cạnh, đã xác
định, 37

GraphViz, tạo trực quan hóa với, 43-51

công cụ fdp ,
44-45, 45f công cụ gọn
gàng , 47-48, 47 tham số
f, 44 công cụ
sfepy , 46-47, 46f thuộc
tính trực quan, 48-51

Thư viện NetworkX, tạo mạng với,
40-43 thêm thuộc tính, 42

thêm nút và cạnh, 41 lưu
mạng vào đĩa, 42-43 nút, đã xác
định, 37 phép chiếu, 38 phân tích
mã được chia sẻ và,
60-61 thử thách

trực quan hóa, 39-40

vẫn đè biến dạng, 39-40, 40f thuật
toán hướng lực, 40 bô cục mạng, 39-
40 mẫu phần mềm độc hại,
61-62, 222-224 malwr.com, 26-33, 28f

phân tích kết quả trên,
28-33 lệnh gọi API, 32 -33, 33f

đổi tượng hệ thống đã
sửa đổi, 30-32 Bảng ảnh chụp màn
hình, 30, 30f Bảng chữ ký, 29-
30, 29f Bảng tóm tắt, 30-32, 31f-
32f giới hạn của, 33 tài liệu trên, 27-

Bắt buộc, 61, 76, 223

Tiện ích MAPIGET, 223

Mastercard, iii

thư viện matplotlib , 148-150,
162-167, 162f

biểu đồ tỷ lệ phát hiện phần
mềm độc hại và sâu, 165-
167, 166f biểu

đồ tỷ lệ phát hiện phần mềm tổng
tiền, 164-165, 165f biểu

đồ mối quan hệ giữa kích thước và
khả năng phát hiện phần mềm độc
hại, 162-
163 chức năng tối

đa , 160 hàm trung bình ,
160-161 ô nhớ, 196

mô-đun số liệu, 147-148 tham
số số liệu, 201-202 hàm tối
thiểu, 81, 160 phương

pháp minhash kết hợp
với phác thảo, 79 phép toán đẳng
sau, 78-79, tổng quan 78f, 77-
78 hàm minhash , 82

gọi lại ModelCheckpoint , 212

Lớp mô hình , 201

mô hình con mô hình, 201-202

lệnh mov , 15-16 mô-đun rì
rầm , 80, 82 mutexes, đã
xác định, 32 chức năng

my_generator , 205, 207-208

lớp MyCallback , 213-214

N

công cụ gọn gàng , 47-48,
47f Dòng phần mềm độc hại Nemucod.FG, 157, 157f

Thư viện NetworkX, 40-43 tạo

mạng quan hệ chia sẻ, 41-42 tổng
quan, 41 lưu mạng
vào đĩa, 42-43

mạng nd-ron, 176, 177-188 tạo tính
năng tự động, tòa nhà 188 với bốn nd-
ron, 186-
188, 186f-187f, 187t với ba nd-
ron, 184-186,
185f-186f, 185t với hai nd-ron, 182-
184, 182f- 184f,
183t-184t

xây dựng với gói Keras , 199-214

mô hình biên dịch, 202-203, 202f
xác định kiến trúc của mô hình, 200-
202
đánh giá mô hình, 209-211, 210f-
211f
lưu và tải mô hình, 209
mô hình đào tạo, 203-209, 211-
214

tập dữ liệu cho, 224

tế bào thần kinh, 176

giải phẫu của, 177-180, 177f,
178t-180t

mạng của, 180-181, 181f

huấn luyện, 189-193

sử dụng lan truyền ngược,
190-192, 190f-191f

sử dụng lan truyền thuận, 189-190

vấn đề độ dốc biến mất, 192-193

các loại, 193-197

bộ mã hóa tự động, 194-195, 195f

tích chập, 193-194, 194f

chuyển tiếp, 193

đối thủ chung, 195-196

tái phát, 196

ResNet, 196-197

định lý xấp xỉ phổ quát, 181-182, 182f

tế bào thần kinh, 176

giải phẫu của, 177-180, 177f, 178t-180t

mạng của, 180-181, 181f

phương pháp tiếp theo, 205, 208

N-gam, 63-64, 64f

tính tương tự dựa trên lệnh

gọi API động, 72

trích xuất các tính năng, 136-137

tính tương tự dựa trên trình tự

lệnh, 67-68

nút, 37

thêm thuộc tính, 42

thêm vào mạng lưới quan hệ chia sẻ,

41

thêm các thuộc tính trực quan vào, 48-51

màu, 49, 49f

hình dạng, 49-50, 50f

nhấn văn bản, 50-51

chiều rộng, 48-49

trong cây quyết định, 110-111

NUM_MINHASHES hàng số, 80-81

Ô

hàm mục tiêu, 189

tham số trình tối ưu hóa , 201-202

tiêu đề tùy chọn (ở định dạng tệp PE) , 3-4

đối số output_dot_file , 227-228

đối số output_file , 52, 225, 227

mô hình máy học overfit, 98-99, 101f

tham số chồng chéo , 44

P

đóng gói, 21

khó phân tách phần mềm độc hại được đóng
gói, 26

sử dụng hợp pháp, 22

gói gấu trúc , 158-161

lọc dữ liệu sử dụng điều kiện, 161

đang tải dữ liệu, 158-159

thao tác với DataFrame, 159-161

Parkour, Mila, 61

họ phần mềm độc hại pasta, 62, 67f-68f,
70f-72f

THẺ DỤC. Xem Định dạng tệp thực thi di động

Tiêu đề PE (Portable Executable), 3, 135-
136

hàm pecheck , 73-74

mô-đun pefile , 5-7

tháo gỡ bằng cách sử dụng, 20

bán phá giá IAT, 6-7

cài đặt, 5, 20

mở và phân tích tệp, 5-6

lấy thông tin từ các trường PE, 6

mô-đun phân tích PE pefile , 51-52

thuộc tính băng thông , 48-49

hệ thống tìm kiếm tương tự phần mềm độc

hại dai dẳng, 79-87

xây dựng

cho phép người dùng tìm kiếm và nhận

xét về các mẫu, 82-84

triển khai chức năng cơ sở

dữ liệu, 80-81

gói nhập khẩu, 80

lập chỉ mục các mẫu vào cơ sở dữ liệu

của hệ thống, 82

tải mẫu, 85

lấy minhashes và bắn phác

thảo, 81-82

phân tích đối số dòng lệnh của

người dùng, 84-85

tìm kiếm tương tự phần mềm độc hại dai dẳng
hệ thống, tiếp tục
nhận xét mẫu, 86
đưa ra mẫu, 86-87
tìm kiếm các mẫu tương tự, 86
xóa cơ sở dữ liệu, 86
hàm pick_best_question , 112-113
môđun đưa chua , 143-144
chức năng cốt truyện , 162-163, 167
định dạng .png , 43
lớp tổng hợp, 194
hướng dẫn pop , 16-17
Định dạng tệp thực thi di động
(PE), 2-5
môxé các tệp bằng pefile, 5-7
diễn vào, 3
cấu trúc tệp, 2-5, 3f
Tiêu đề DOS, 3
tiêu đề tùy chọn, 3-4
Tiêu đề PE, 3
tiêu đề phần, 4-5
phần, xác định, 4
Tiêu đề Portable Executable (PE), 3, 135-
136
vị trí độc lập, 5
độ chính xác, 124-126
ánh hưởng của lãi suất cơ bản trên, 124-125
ước tính trong môi trường
triển khai, 125-126
phương pháp dự đoán_proba , 144, 149
Chức năng kích hoạt PRELU, 179t
ngăn xếp chương trình, đã xác định, 14
project_đò thị hàm, 54
dự đoán, 38
hướng dẫn đầy , 16-17
môđun pyplot , 148-149, 163

I

rừng ngẫu nhiên
tổng quan, 115-116, 116f
máy dò dựa trên rừng ngẫu nhiên, 141-
146
mã hoàn chỉnh cho, 144-146
chạy máy dò trên các nhị phân
mới, 144
huấn luyện, 142-143
Lớp RandomForestClassifier , 143, 152
phần mềm tổng tiền, 30-31, 31f, 155-158, 156f,
158, 164-168, 165f-166f,
168f, 172-173, 172f-173f
phần .rdata (ở định dạng tệp PE), 4

Đường cong đặc tính hoạt động của máy
thu. Xem đường cong ROC
chức năng kích hoạt đơn vị tuyến tính được
chỉnh lưu (ReLU), 177f, 178t,
180, 182f, 183-185, 201
mạng thần kinh tái phát (RNNs), 196
khóa đăng ký, 32
phần .reloc (ở định dạng tệp PE), 5
Chức năng kích hoạt ReLU (đơn vị tuyến tính
chỉnh lưu), 177f, 178t, 180,
182f, 183-185, 201
ResNets (mạng còn lại), 196-197
đối số resource_projection , 52, 227
xáo trộn tài nguyên, 22
ret hướng dẫn, 17-18
kỹ thuật đảo ngược, 12
kỹ thuật chống tháo gỡ, 22
phân tích động cho, 26
phương pháp cho, 12
phân tích mã dùng chung, 60
sử dụng pefile và capstone, 20-21
RNNs (mạng lưới thần kinh tái phát), 196
Đường cong ROC (Đặc điểm hoạt
động của máy thu), 123-124,
123f, 126, 147-150, 230-231, 231f
máy tính, 147-150
xác thực chéo, 151-152, 153f
mạng lưới thần kinh, 209-210,
210f-211f
hình dung, 149, 150f
hàm roc_curve , 149, 210
Phần .rsrc (tài nguyên) (ở định dạng tệp
PE), 4-5

S

hộp cát, 26
Sanders, Hillary, 216
chức năng savefig , 165
hàm scan_file , 144
chè độ quét , 230-231
gói máy học scikit-learn
(sklearn) , 127-128
xây dựng bộ phát hiện dựa trên cây quyết
định cơ bản, 129-134
xây dựng máy dò dựa trên rừng ngẫu
nhiên, 141-146
đánh giá hiệu suất của máy dò, 146-153
trích xuất tính năng, 134-135
thủ thuật băm, 140-141

thuật ngữ và khái niệm, 128-129
 phân loại, 129
 phù hợp, 129
 vector nhẫn, 128-129
 dự đoán, 129
 vectơ, 128
 gói seaborn , 168-174, 168f
 tạo ô vĩ cầm, 172-174, 172f-173f
 sơ đồ phân phối phát hiện chông vi-rút, 169-172, 169f, 171f
 hàm search_sample , 82-84
 Chế độ TÌM KIẾM , 229
 tiêu đề phần (ở định dạng tệp PE), 4-5
 phần .data , 4
 phần .idata (nhập), 4
 phần .rdata , 4
 phần .reloc , 5
 phần .rsrc (tài nguyên), 4-5
 phần .text , 4
 nhà khoa học dữ liệu bảo mật, 215-220
 mở rộng kiến thức về các phương pháp, 219-220
 con đường trở thành, 216
 đặc điểm hiệu quả, 218-219
 tờ mở, 218-219
 ám ảnh với kết quả, 219
 cởi mở, 218
 hoài nghi kết quả, 219
 sẵn sàng học hỏi, 216
 quy trình làm việc của, 216-218, 217f
 nhận dạng nguồn cấp dữ liệu, 218
 giao dịch với các bên liên quan, 217
 triển khai, 218
 xác định vấn đề, 217-218
 xây dựng và đánh giá giải pháp, 218
 mã tự sửa đổi, 12
 hàm set_axis_labels , 172
 công cụ sfdp , 46-47, 46f
 thuộc tính hình dạng , 49-50
 phân tích thuộc tính dùng chung. Xem phân tích
 mạng phân mềm độc hại
 phân tích mã dùng chung (phân tích
 độ tương đồng), 59-87, 60, 61f
 mô hình túi tính năng, 62-64, 63f
 tính năng, định nghĩa, 62
 N-gam, 63-64, 64f
 thông tin đặt hàng và, 63-64
 tổng quan, 62-63
 tập dữ liệu cho, 223
 Chỉ số Jaccard, 64-65, 65f
 hệ thống tìm kiếm tương tự phân mềm độc hại dai dẳng, 79-87
 cho phép người dùng tìm kiếm và nhận xét về các mẫu, 82-84
 nhận xét mẫu, 86
 triển khai chức năng cơ sở dữ liệu, 80-81
 gói nhập khẩu, 80
 lập chỉ mục các mẫu vào cơ sở dữ liệu hệ thống, 82
 tài mẫu, 85
 lấy minhashes và bản phác thảo, 81-82
 phân tích đối số dòng lệnh của người dùng, 84-85
 đầu ra mẫu, 86-87
 tìm kiếm các mẫu tương tự, 86
 xóa cơ sở dữ liệu, 86
 mở rộng so sánh tương tự, 77-79
 khó khăn với, 77
 phương pháp minhash, 77-79, 78f
 đồ thị tương tự, 73-76, 76f
 khai báo các hàm tiện ích, 73-74
 trích xuất các tính năng, 73, 75
 nhập thư viện, 73
 lặp qua các cặp, 75
 Ngưỡng chỉ số Jaccard, 73
 phân tích đối số dòng lệnh của người dùng, 74
 trực quan hóa đồ thị, 76
 ma trận tương tự, 66-72, 66f-67f
 khái niệm về, 66
 tương tự dựa trên lệnh gọi API động, 72, 72f
 Độ tương tự dựa trên bảng địa chỉ
 nhập khẩu, 71, 71f
 tính tương tự dựa trên trình tự lệnh, 67-70, 68f
 độ tương tự dựa trên chuỗi, 70-71, 70f
 công cụ cho, 227-230, 228f
 mạng quan hệ hình ảnh được chia sẻ, 54-58,
 55f, 226-227
 trích xuất nội dung đồ họa, 57
 phân tích đối số ban đầu và mã tài
 tệp, 55-57
 lưu mạng vào đĩa, 58
 mô -đun kẽ , 80
 chức năng hiển thị , 152, 163, 165, 168

Chức năng kích hoạt sigmoid, 180t, mô-dun sim_graph 201, phân tích tương tự 80, 82. Xem các hàm tương tự phân tích mã được chia sẻ, 64-65 biểu đồ tương tự, 73-76, 76f khai báo các hàm tiện ích, 73-74 trích xuất các tính năng, 73, 75 nhập thư viện, 73 lặp qua các cặp, 75 Nguồn chỉ mục Jaccard, 73 phân tích đối số dòng lệnh của người dùng, 74 đồ thị trực quan, 76 ma trận tương tự, 66-72, 66f-67f

tương tự dựa trên lệnh gọi API động, 72, 72f Tương tự dựa trên bảng địa chỉ nhập, 71, 71f tương tự dựa trên trình tự hướng dẫn, 67-70, 68f tương tự dựa trên chuỗi, 70-71, 70f hằng số SKETCH_RATIO, 80, 82 sklearn.

Xem máy học scikit

gói tìm hiểu họ phần mềm độc hại skor, 62, 67f-68f, 70f-72f

Chức năng kích hoạt Softmax, 180t Sophos, tham số 216 splines, 44 biểu thức split_regex, ngăn xếp 203-204, đã xác định, 16 hướng dẫn ngăn xếp, thanh ghi quản lý ngăn xếp 16-17, phân tích phần mềm độc hại tinh 14-15, bộ dữ liệu 1-23 cho, kỹ thuật đảo ngược và tháo gỡ 222, 12 phương pháp cho, 12 sử dụng pefile và capstone, 20-21 phân tích hình ảnh đồ họa, 7-8

phân tích nhập khẩu, 6-7 giới hạn của, 21-23 kỹ thuật chống tháo gỡ, 22 dữ liệu được tải xuống động, 22-23 đóng gói, 21-22 xáo trộn tài nguyên, 22 mô-dun pefile, 5-7

Định dạng tệp thực thi di động, phân tích chuỗi 2-5, 8-10

chức năng tiêu chuẩn, 160

Hàm Kích hoạt theo bước, tham số 179steps_per_epoch, 206 hàm string_hash, xác định 81-82 chuỗi, 8 trích xuất tính năng, 135, 141-142 phân tích chuỗi, 8-10 phân tích chuỗi có thể in, 8-10 thông tin được tiết lộ thông qua, 8 in tất cả các chuỗi trong một tệp vào thiết bị đầu cuối, độ tương tự dựa trên chuỗi 8-9, 70-71, công cụ chuỗi 70f, Hướng dẫn số học phụ 8-10, 15 hàm tóm tắt, 202-203, siêu nút 202f, 46 tham số nghi ngờ_calls, 103-104 điểm đáng ngờ, 121-122, 121f-122f

t

Mục tiêu, iii đối số target_directory, 227-228 đổi số target_path, 52, 225, 227 TensorFlow, 200, 207 phần .text (ở định dạng tệp PE), 4 điểm số mỗi đe dọa, 147 phương pháp .todense, 142 hàm train_detector, 143 biến training_examples, 130 phương pháp biến đổi, mô-dun 131, 140 cây, 129 Trojan, 54-55, 55f, 158-161, 168f, 172f-173f, 173 phủ định thực, được xác định, 120, 120f khẳng định thực, 120, 120f tốc độ cơ sở và độ chính xác, 124-126 mối quan hệ giữa tỷ lệ dương tính thật và sai, 121-122, 121f-122f đường cong ROC, 123-124, 123f tỷ lệ dương tính thật, 121

bạn

các mô hình học máy không phù hợp, 98-99, định lý xấp xỉ phỏng quát 99f, 181-182, 182f

Máy đóng gói UPX, 29

V

đổi tương validation_labels , 210-211
 đổi tương verify_scores , 210
 vấn đề độ dốc biến mất, 192-193
 họ phần mềm độc hại vbna, 62, 67f-68f,
 70f-72f
 vectơ, 128
 âm mưu vĩ cầm, 172-174, 172f-173f
 VirtualBox, vii-viii, 222
 kích thước ảo, 6
 VirusTotal.com, 29, 59
 hình dung, 155-174
 trình phát hiện phần mềm độc hại dựa
 trên máy học cơ bản,
 131-133, 132f
 tập dữ liệu cho, 224
 tầm quan trọng của, 156-158, 157f
 phân tích mạng phần mềm độc hại
 thách thức, 40f
 tạo bằng GraphViz, 45f-47f
 Phân tích mạng
 thử thách đối với, 39-40
 tạo bằng GraphViz, 43-51
 Đường cong ROC, 149, 150f, 152-153, 153f
 phân tích mã dùng chung, 76
 sử dụng matplotlib, 162-167, 162f
 biểu đồ tỷ lệ phát hiện
 ransomware và sâu, 165-
 167, 166f
 biểu đồ tỷ lệ phát hiện ransomware,
 164-165, 165f
 vẽ mối quan hệ giữa
 kích thước và phát hiện phần mềm độc
 hại, 162-163
 sử dụng giao trúc, 158-161
 lọc dữ liệu sử dụng
 điều kiện, 161
 đang tải dữ liệu, 158-159
 thao tác với DataFrame, 159-
 161
 sử dụng seaborn, 168-174, 168f
 tạo ô vĩ cầm, 172-174, 172f-173f

sơ đồ phân phối phát hiện
 chống vi-rút, 169-172,
 169f, 171f

W

dòng phần mềm độc hại webprefix, 62, 67f-68f,
 70f-72f
 thuộc tính trọng lượng, 37
 tham số trọng lượng, 178, 181
 Wells Fargo, III
 Wikipedia, 220
 hàm wipe_database , 80-81
 chế độ xóa , 229
 phương pháp làm việc , 57
 sâu, 158-161, 165-167, 166f, 168f, 172, 172f-
 173f
 công cụ đầu vật , 55
 đang tải xuống, 8
 trích xuất tài nguyên hình ảnh, 7-8
 hàm write_dot , 42-43

X

hợp ngữ x86, 12-20
 hướng dẫn số học, 15, 15t
 Các thanh ghi CPU, 13-15, 14f
 thanh ghi mục đích chung, 13-14
 thanh ghi luồng ngăn xếp và điều khiển,
 14-15
 hướng dẫn di chuyển dữ liệu, 15-
 20, 16t
 khối cơ bản và luồng điều khiển
 đồ thị, 19-20, 19f
 hướng dẫn luồng điều khiển, 17-18
 hướng dẫn ngăn xếp, 16-17
 phương ngữ của, 13
 phân tích mã dùng chung, 67
 dòng phần mềm độc hại xtoober, 62, 67f-68f,
 70f-72f

Y

tuyên bố năng suất , 205

Z

họ phần mềm độc hại zango, 62, 67f-68f,
 70f-72f

Khoa học dữ liệu phản mềm đặc hại lấy bối cảnh ở New Baskerville, Futura, Dogma và TheSans MonoCondensed.

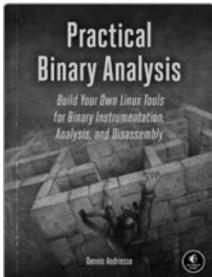
cập nhật

Truy cập <https://www.nostarch.com/malwaredatascience/> để biết các bản cập nhật, lõi in và các thông tin khác.

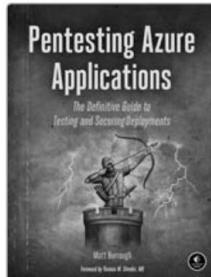
Nhiều cuốn sách vô nghĩa từ



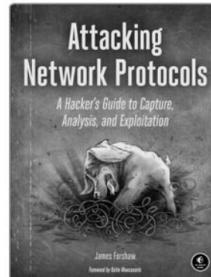
Không ép tinh bột



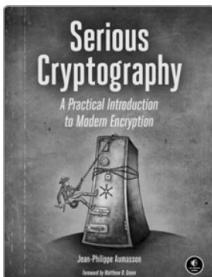
Phân tích nhị phân thực tế Xây dựng các công cụ Linux của riêng bạn cho thiết bị nhị phân, phân tích và phân tách bởi dennis andriesse mùa thu 2018, 440 trang, \$49,95 isbn 978-1-59327-912-7



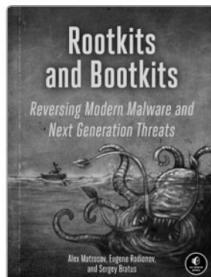
Pentesting Azure Applications Hướng dẫn đứt khoát để kiểm tra và bảo mật triền khai của matt burrough tháng 7 năm 2018, 216 trang, \$39,95 isbn 978-1-59327-863-2



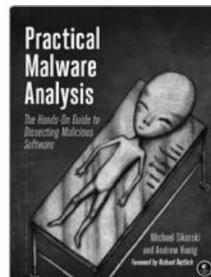
Tấn công các giao thức mạng Hướng dẫn nắm bắt, phân tích và khai thác của hacker bởi james forshaw tháng 12 năm 2017, 336 trang, \$49,95 ty đồng 978-1-59327-750-5



Mã hóa nghiêm túc Giới thiệu thực tế về mã hóa hiện đại của jean-philippe aumasson tháng 11 năm 2017, 312 trang, \$49,95 isbn 978-1-59327-826-7



Rootkit và Bootkit đảo ngược phần mềm độc hại hiện đại và các mối đe dọa thế hệ tiếp theo của alex matrosov, eugene rodionov và sergey bratus mùa xuân 2019, 504 trang, \$49,95 isbn 978-1-59327-716-1



Phân tích phần mềm độc hại thực tế Hướng dẫn thực hành để Phân tích phần mềm độc hại của michael sikorski và andrew honig tháng 2 năm 2012, 800 trang, \$59,95 isbn 978-1-59327-290-6

điện

thoại: 1.800.420.7240
hoặc 1.415.863.9900

email: sales@nostarch.com

web:
www.nostarch.com

“Hãy đón đầu những thay đổi trong công nghệ và những kẻ thù mà bạn có nhiệm vụ phải đánh bại.”

—Anup Ghosh, Tiến sĩ, người sáng lập Invincea, Inc

Với hàng triệu tệp phần mềm độc hại được tạo ra mỗi năm và vô số dữ liệu liên quan đến bảo mật được tạo ra mỗi ngày, bảo mật đã trở thành một vấn đề “dữ liệu lớn”. Vì vậy, khi bảo vệ chống lại phần mềm độc hại, tại sao không suy nghĩ như một nhà khoa học dữ liệu?

Trong Khoa học dữ liệu về phần mềm độc hại, các nhà khoa học dữ liệu bảo mật Joshua Saxe và Hillary Sanders chỉ cho bạn cách áp dụng máy học, thống kê và trực quan hóa dữ liệu khi bạn xây dựng hệ thống phát hiện và tình báo của riêng mình. Sau phần tổng quan về các khái niệm kỹ thuật đảo ngược cơ bản như phân tích tĩnh và động, bạn sẽ học cách đo lường sự tương đồng về mã trong các mẫu phần mềm độc hại và sử dụng khung học máy hoạt động như scikit-learning và Keras để xây dựng và huấn luyện các công cụ phát hiện của riêng bạn.

Học cách:

- ⌚ Xác định phần mềm độc hại mới được viết bởi cùng một nhóm đối thủ thông qua phân tích mã được chia sẻ
- ⌚ Bắt phần mềm độc hại zero-day bằng cách xây dựng hệ thống phát hiện máy học của riêng bạn
- ⌚ Sử dụng các đường cong ROC để đo độ chính xác của trình phát hiện phần mềm độc hại nhằm giúp bạn chọn phương pháp tốt nhất cho sự cố bảo mật



TUYẾT VỐI NHẤT TRONG GIẢI TRÍ GEEK™
www.nostarch.com

Một phần số tiền thu được từ cuốn sách này
sẽ được quyên góp cho Quỹ Bảo vệ Môi trường.

⌚ Sử dụng trực quan hóa dữ liệu để xác định và khám phá các chiến dịch, xu hướng và mối quan hệ của phần mềm độc hại

⌚ Sử dụng Python để triển khai các hệ thống phát hiện dựa trên mạng thần kinh sâu

Cho dù bạn là nhà phân tích phần mềm độc hại đang tìm cách bổ sung các kỹ năng vào kho vũ khí hiện có của mình hay nhà khoa học dữ liệu quan tâm đến việc phát hiện cuộc tấn công và thông tin tình báo về mối đe dọa, thì Khoa học dữ liệu phần mềm độc hại sẽ giúp bạn luôn dẫn đầu.

Giới thiệu về tác giả

Joshua Saxe là nhà khoa học dữ liệu trưởng tại Sophos, một nhà cung cấp phần mềm bảo mật lớn, nơi ông giúp phát minh ra các công nghệ khoa học dữ liệu để phát hiện các chương trình độc hại trên Android, Windows và web. Trước khi gia nhập Sophos, Saxe đã có 5 năm lãnh đạo các dự án nghiên cứu dữ liệu bảo mật do DARPA tài trợ cho chính phủ Hoa Kỳ.

Hillary Sanders là kỹ sư phần mềm cao cấp và nhà khoa học dữ liệu tại Sophos, nơi cô đã đóng vai trò quan trọng trong việc phát minh và sản xuất các công nghệ bảo mật phân tích độ tương tự của mạng nơ-ron, máy học và phần mềm độc hại. Cô là diễn giả thường xuyên tại các hội nghị an ninh như Black Hat USA và BSides Las Vegas.

Giá: 49,95 đô la (65,95 đô la CDN)

Shelve In: Máy tính/An ninh

ISBN: 978-1-59327-859-5

54995
9 781593 278595