Computer & Internet Security

A Hands-on Approach

Second Edition

Wenliang Du

Syracuse University

Chapter 1

Set-UID Privileged Programs and Attacks on Them

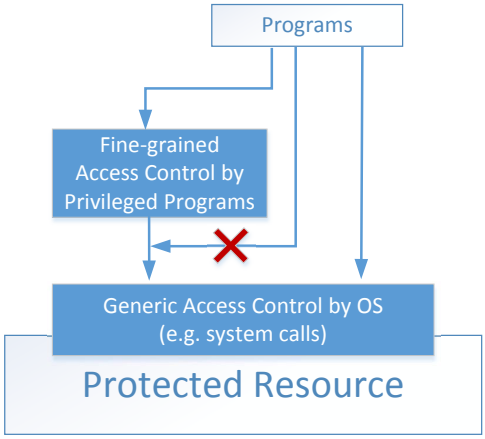


Figure 1.1: Two-Tier Approach for Access Control

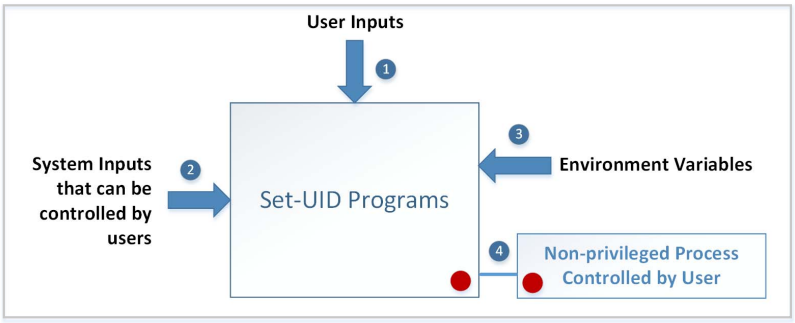


Figure 1.2: Attack Surface (inputs and behaviors that are controllable by users)

Chapter 2

Attacks Through Environment Variables

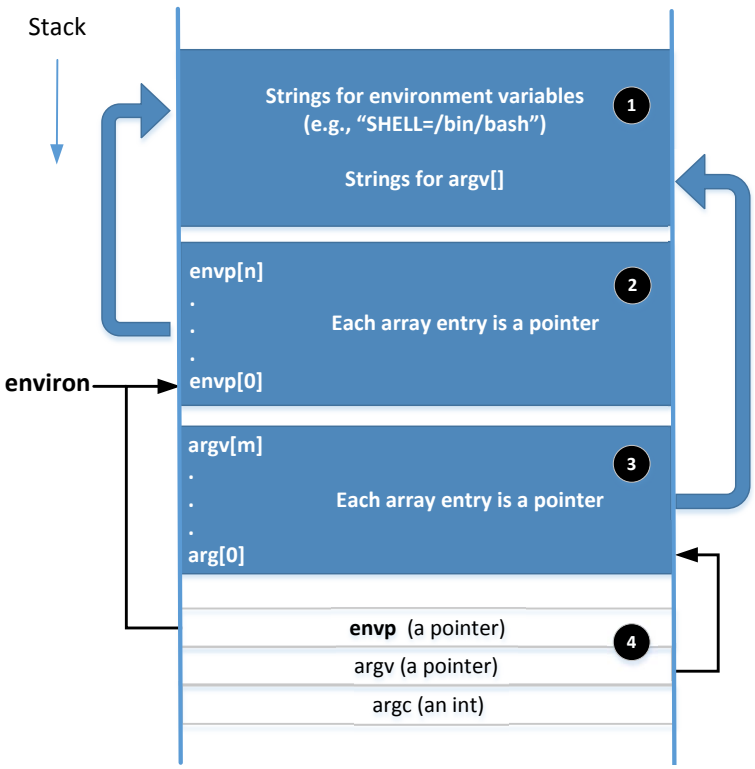


Figure 2.1: Memory location for environment variables

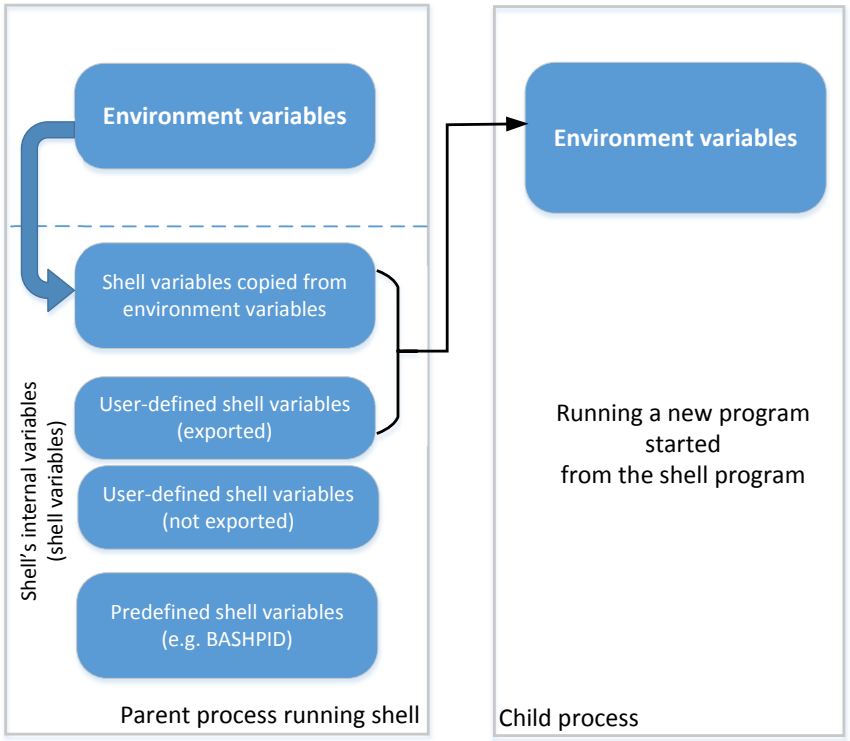


Figure 2.2: Shell variables and environment variables

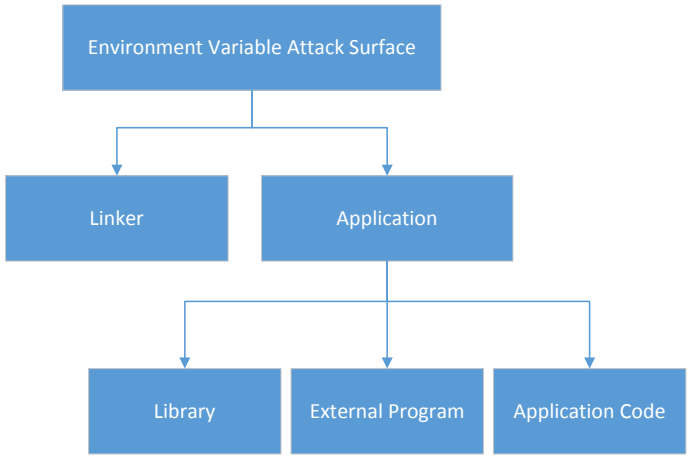


Figure 2.3: Attack surface created by environment variables

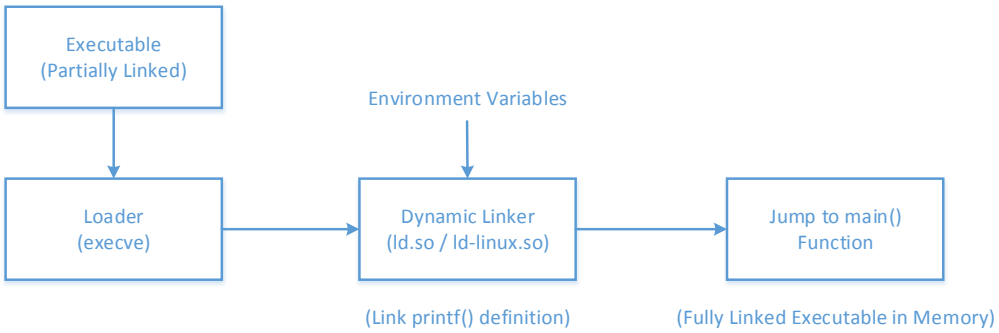


Figure 2.4: Dynamic Linking

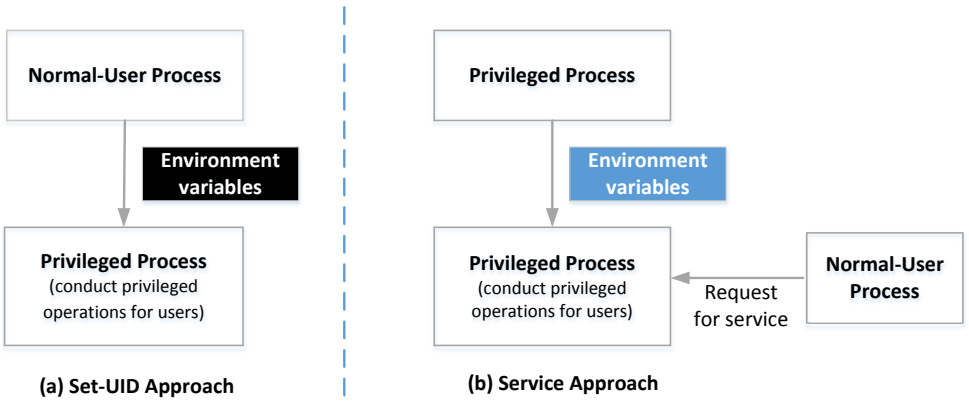


Figure 2.5: Attack surface comparison

Chapter 3

Shellshock Attack

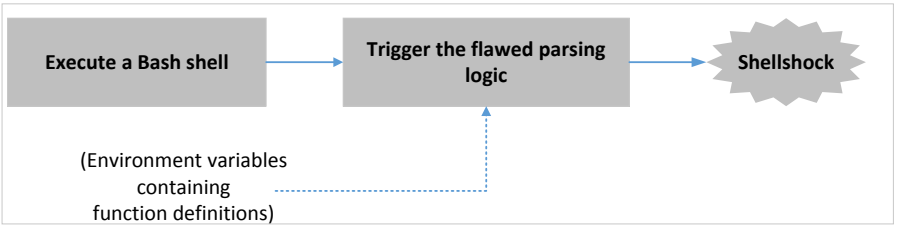


Figure 3.1: Conditions needed for exploiting the Shellshock Vulnerability

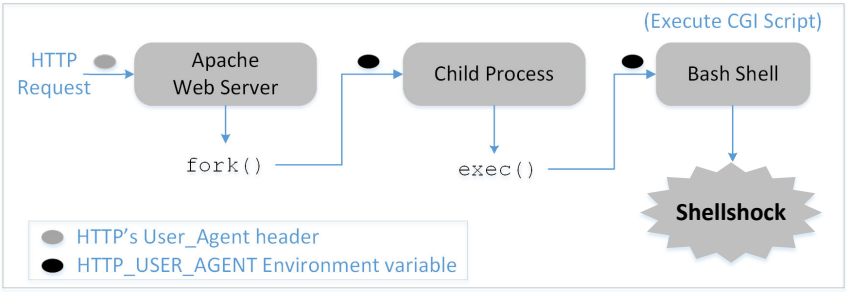


Figure 3.2: How CGI programs are invoked

Chapter 4

Buffer Overflow Attack

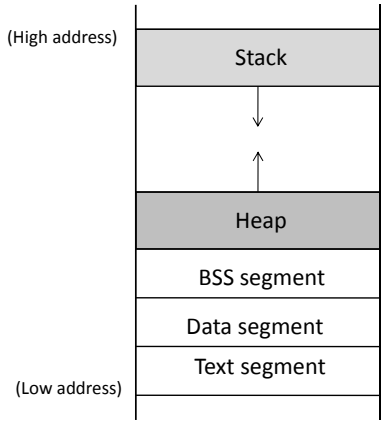


Figure 4.1: Program memory layout

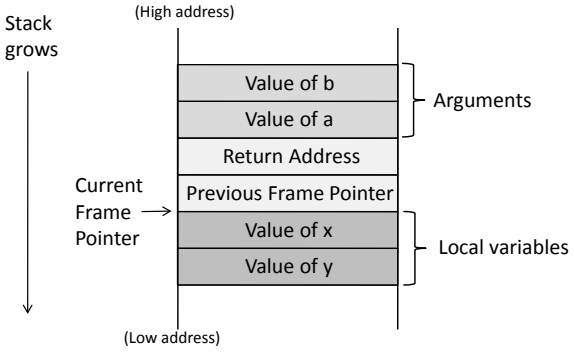


Figure 4.2: Layout for a function's stack frame

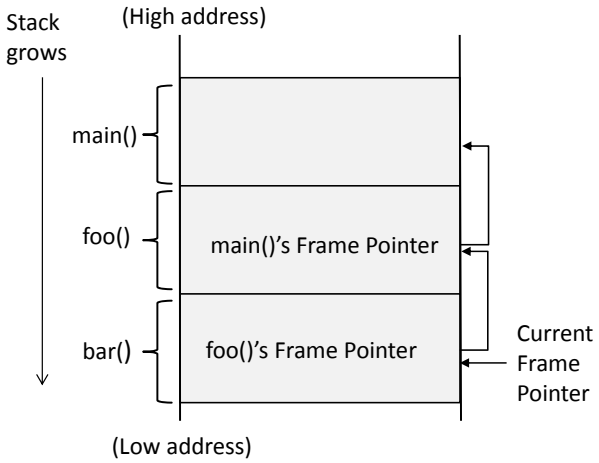


Figure 4.3: Stack layout for function call chain

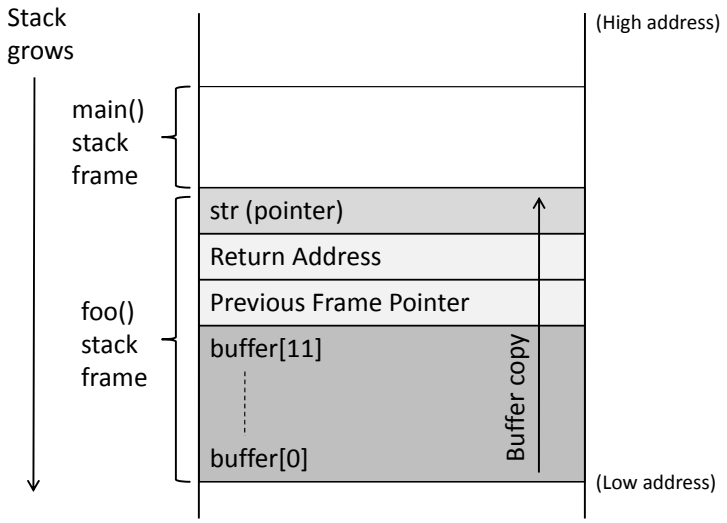


Figure 4.4: Buffer overflow

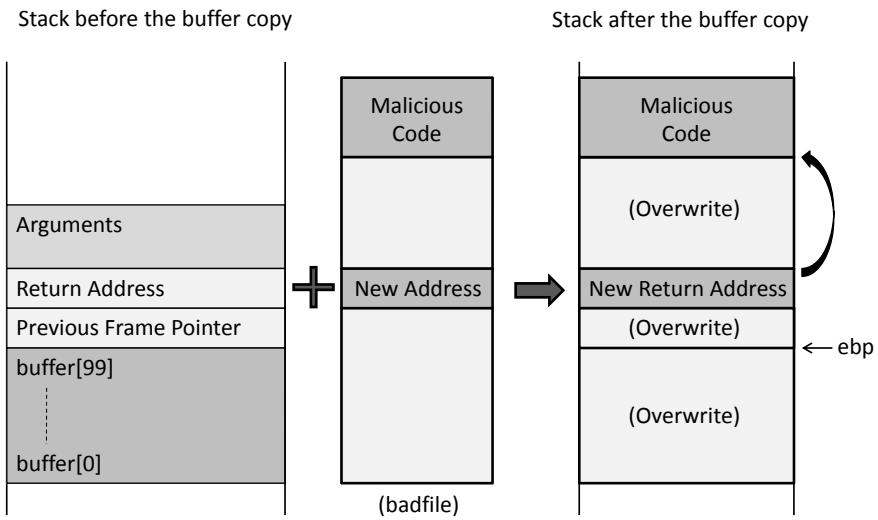


Figure 4.5: Insert and jump to malicious code

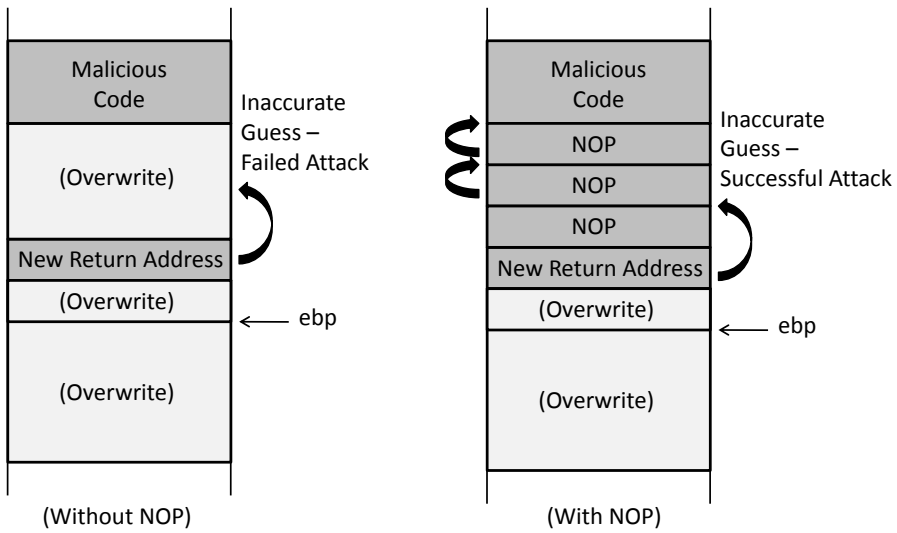


Figure 4.6: Using NOP to improve the success rate

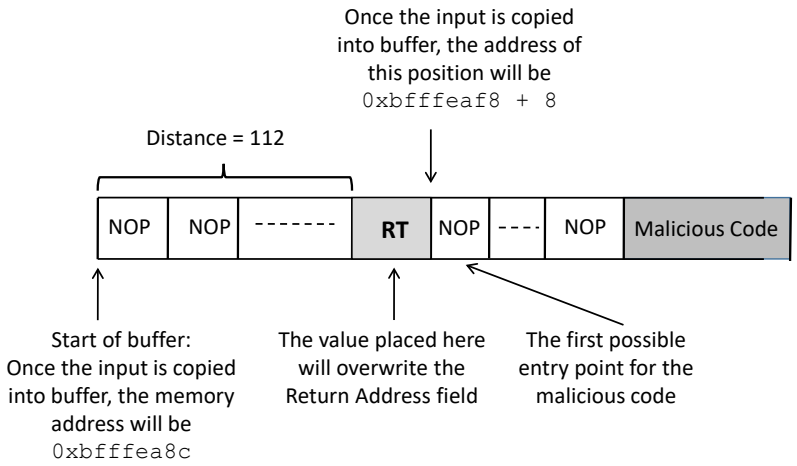


Figure 4.7: The structure of `badfile`

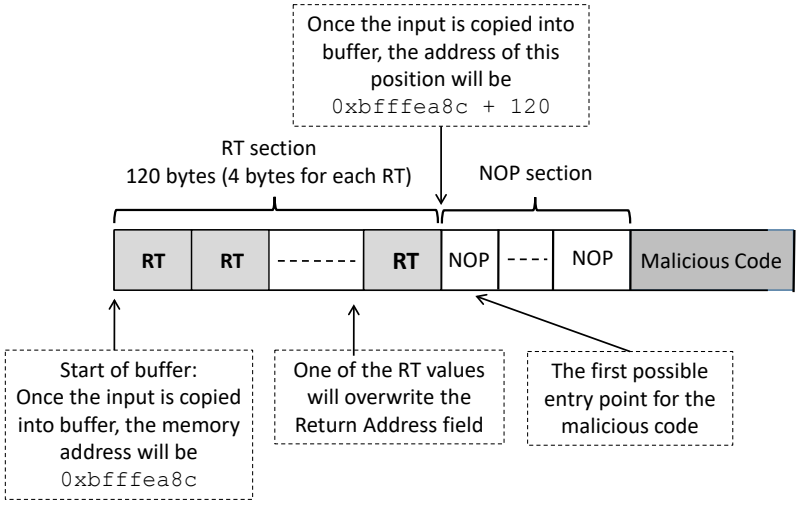


Figure 4.8: Spraying the buffer with return addresses.

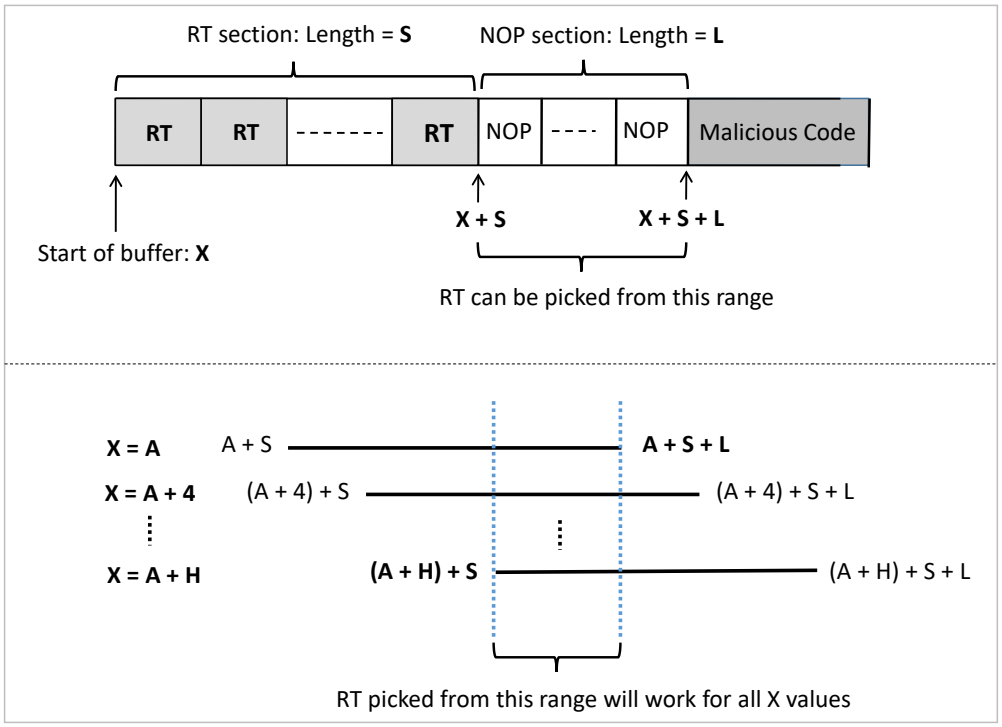


Figure 4.9: Find values for the return address `RT`

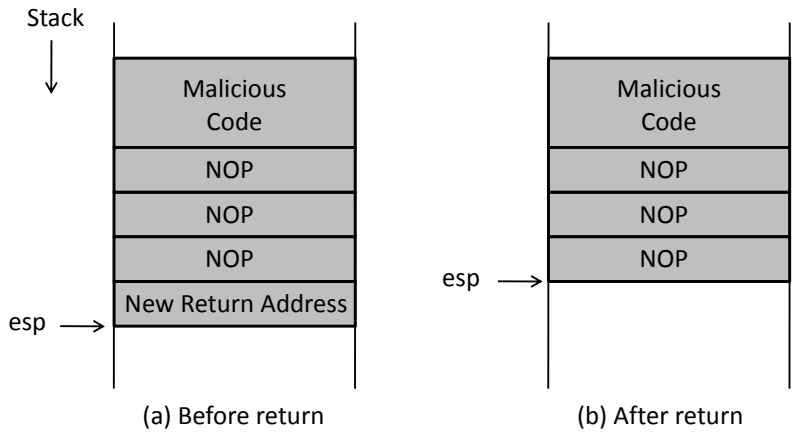


Figure 4.10: The positions of the stack pointer before and after function returns

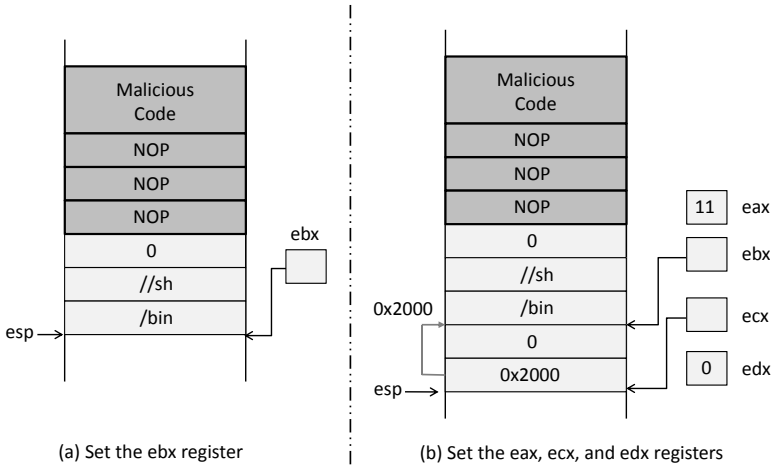


Figure 4.11: Shellcode Execution

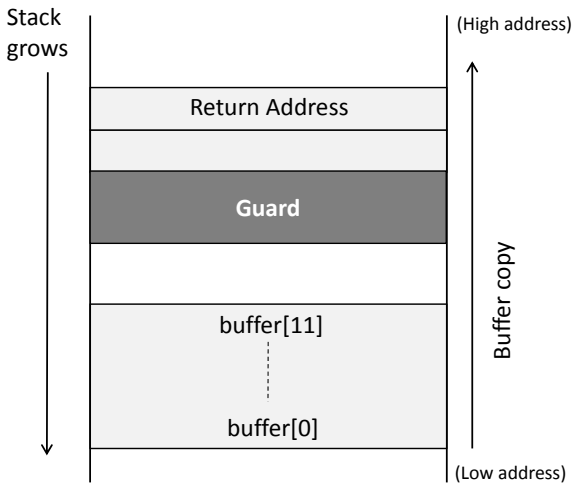


Figure 4.12: The idea of StackGuard

Chapter 5

Return-to-libc Attack and Return-Oriented Programming

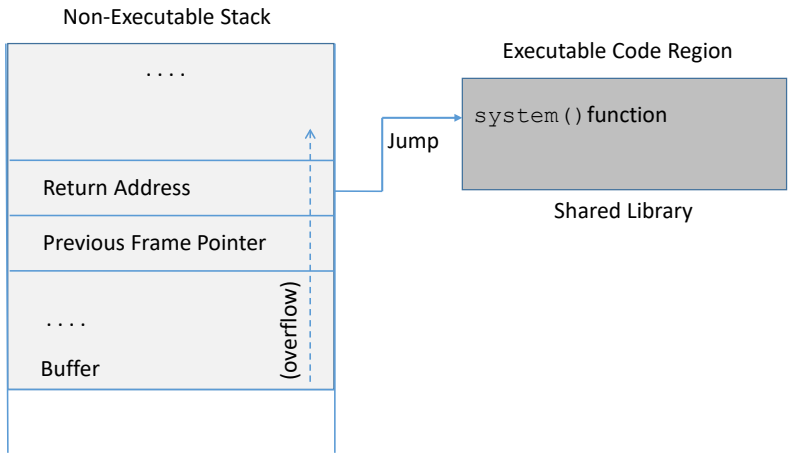


Figure 5.1: The idea of the return-to-libc attack

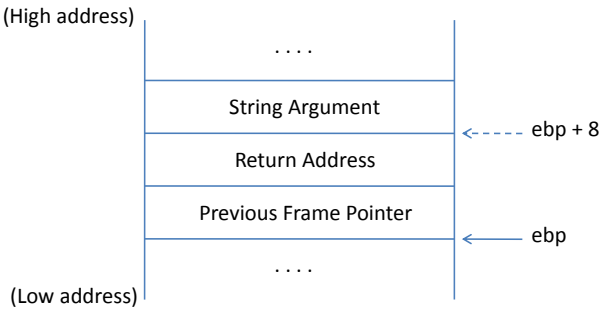


Figure 5.2: Frame for the `system()` function

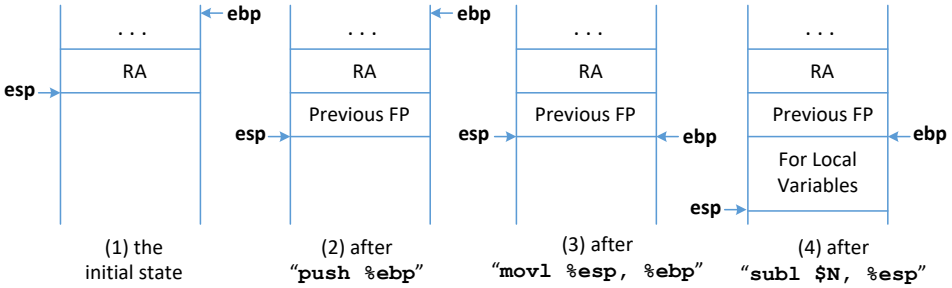


Figure 5.3: How the stack changes when executing the function prologue

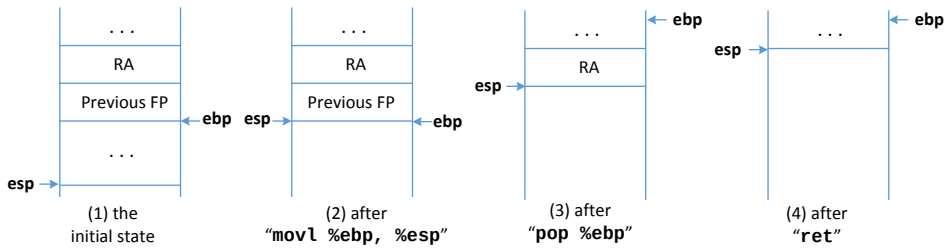


Figure 5.4: How the stack changes when executing the function epilogue

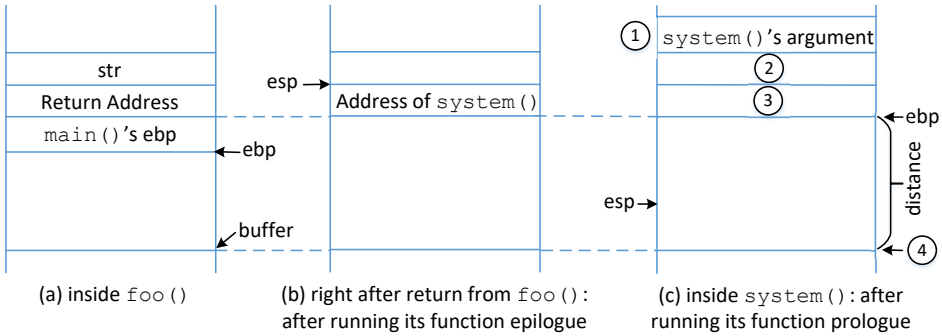


Figure 5.5: Construct the argument for `system()`

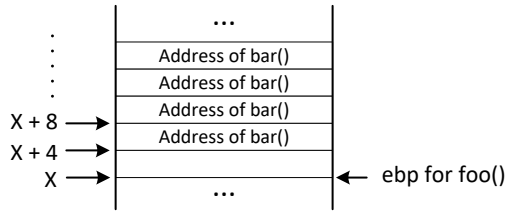


Figure 5.6: Chaining function calls (without arguments)

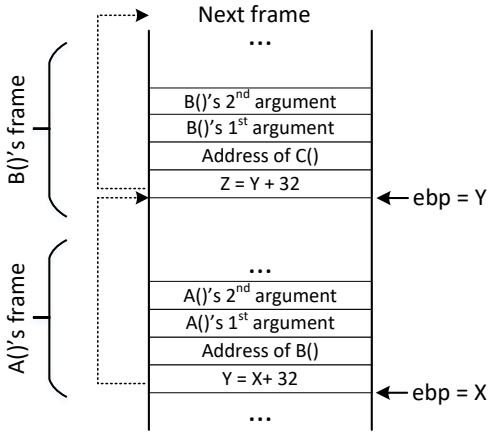


Figure 5.7: Chaining function calls with arguments (skipping function prologue)

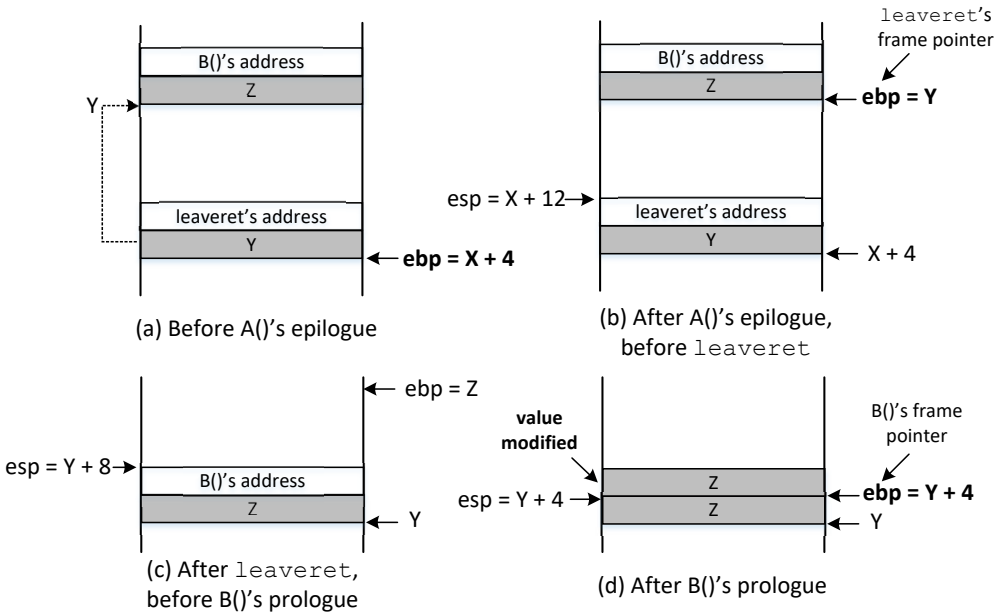


Figure 5.8: How

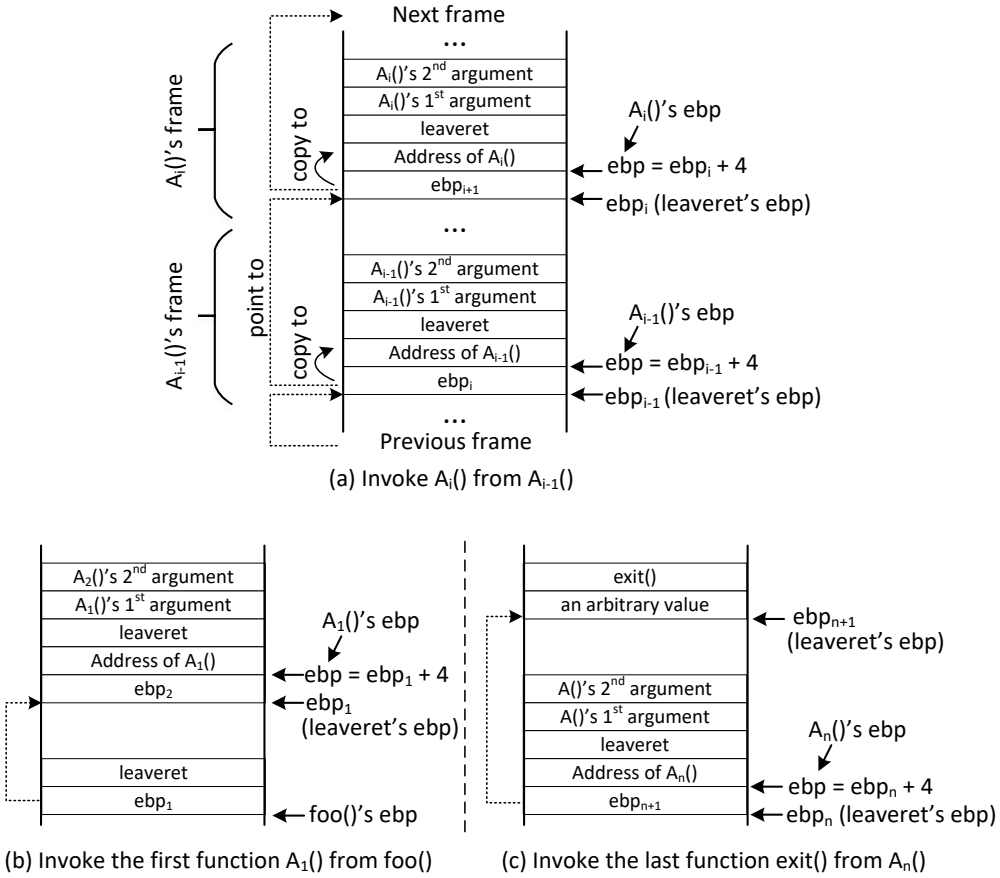


Figure 5.9: Chaining function calls via `leaveret`

Chapter 6

Format String Vulnerability

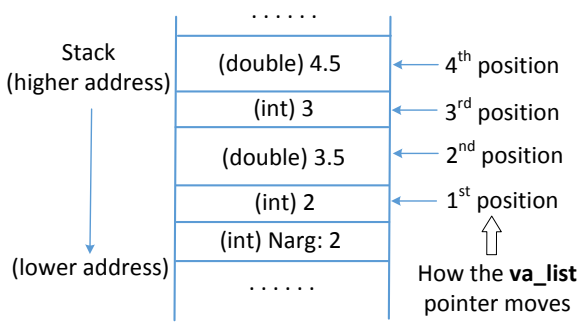


Figure 6.1: The stack layout for `myprint(2, 2, 3.5, 3, 4.5)`

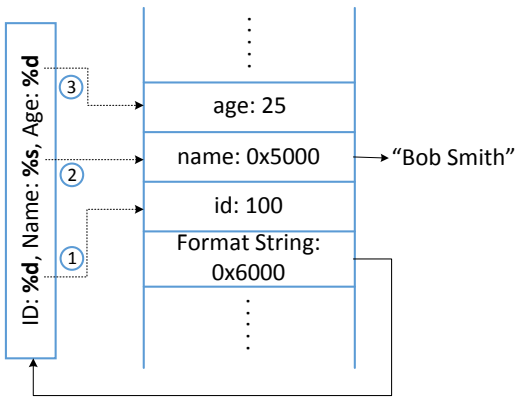


Figure 6.2: How `printf()` accesses the optional arguments

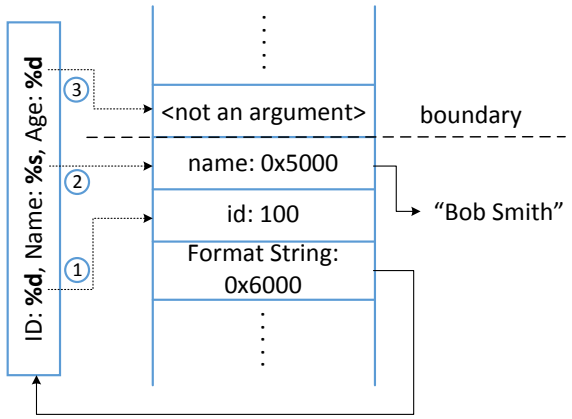


Figure 6.3: Missing Arguments

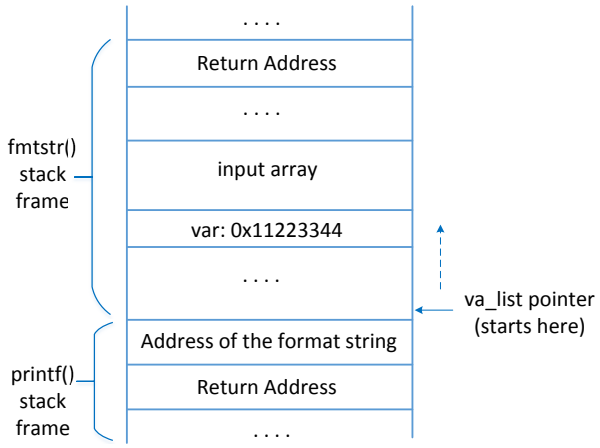


Figure 6.4: Vulnerable Program Stack Layout

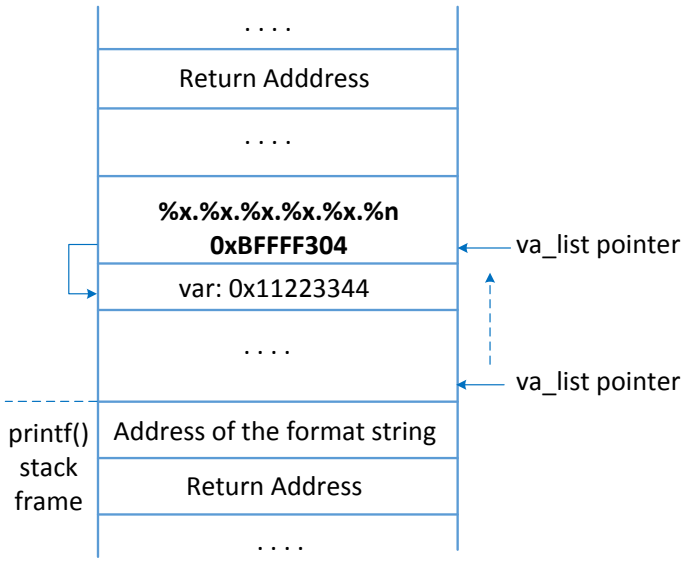


Figure 6.5: Using the format string vulnerability to change memory

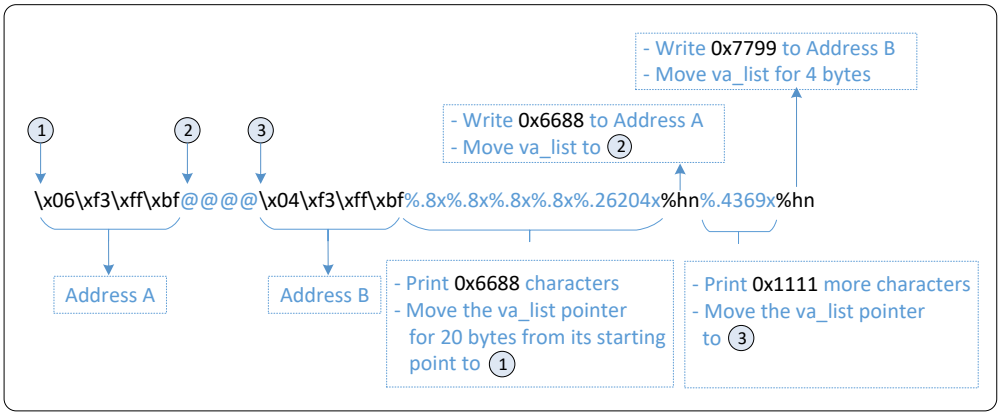


Figure 6.6: The break-down of the format string

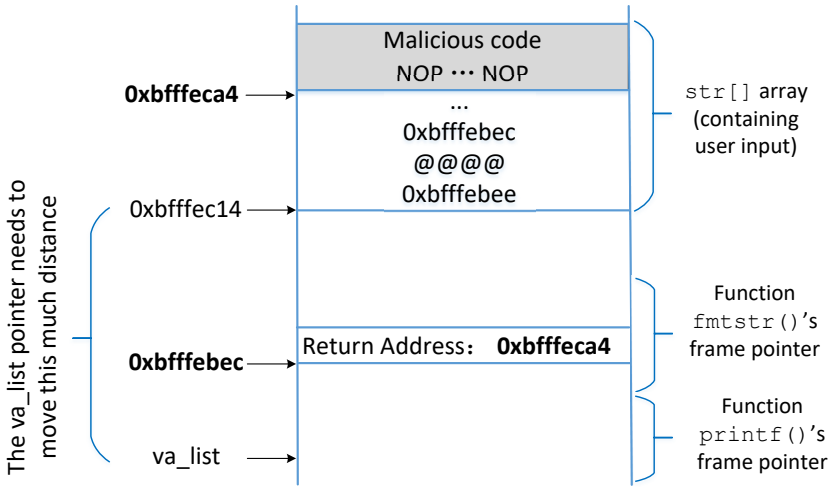


Figure 6.7: Modify the return address of `fmtstr()`, making it point to the injected shellcode.

```
Terminal
$ fmtvul
The address of the input array:  0xbfffec14
The value of the frame pointer:  0xbfffebe8
The value of the return address: 0x080485c4
îëÿ¿@@@@ìëÿ¿080485c4b7fba000b7ffd940bfffece8b7feff10bfffebe8bfffebe
cb7fba000b7fba000bfffece8080485c4bfffec1400000001000000c80804b008b7
ff37ec00000000b7fff000bfffed940000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
... Many zeros are omitted here ...
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000040404040������������������������������������
������������������������1Ø1·Ø1ØPh//shh/bin××PSØ°Ì

The value of the return address: 0xbfffeca4
# 
```

Figure 6.8: Running the vulnerable program and getting the root shell

Chapter 7

Race Condition Vulnerability

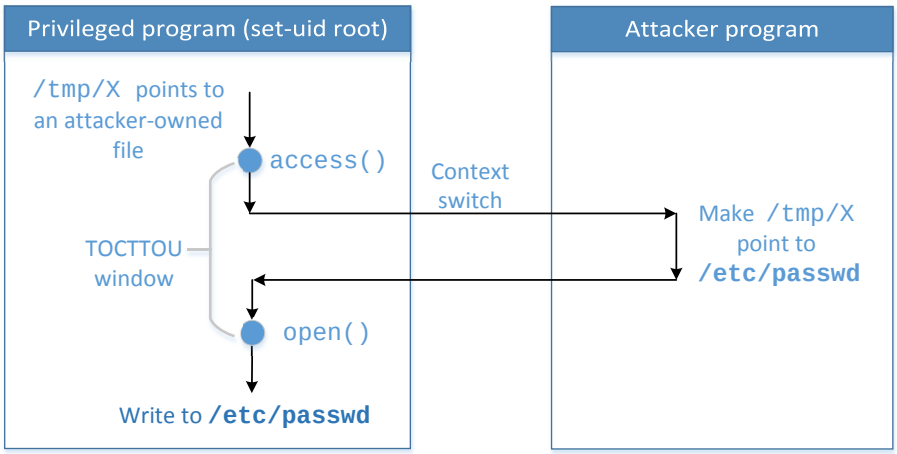
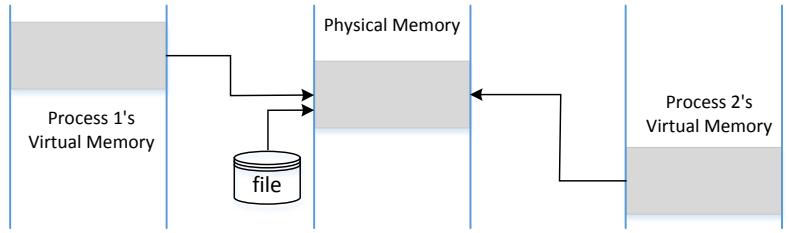


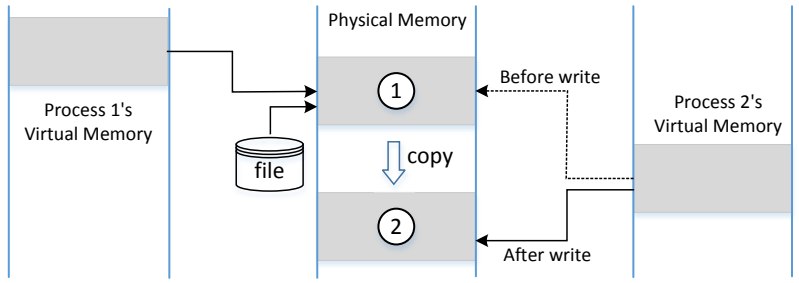
Figure 7.1: Exploiting the TOCTTOU race condition vulnerability

Chapter 8

The Dirty COW Race Condition Attack



(a) MAP_SHARED



(b) MAP_PRIVATE

Figure 8.1: MAP_SHARED and MAP_PRIVATE

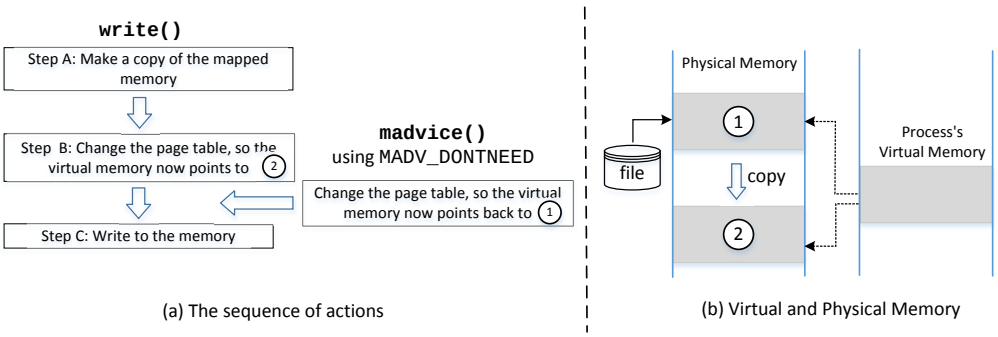


Figure 8.2: The Dirty COW Attack

Chapter 9

Reverse Shell

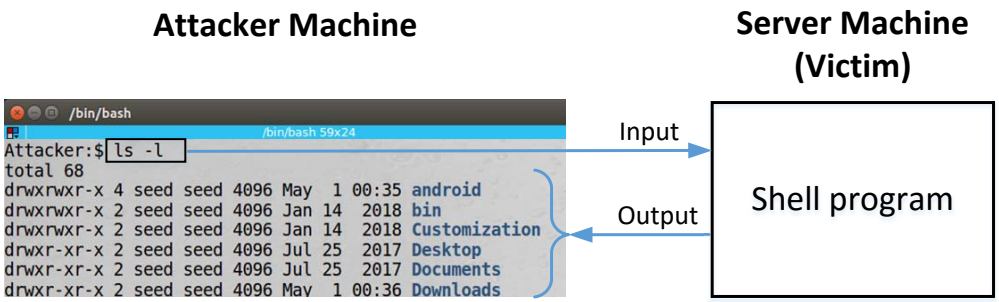


Figure 9.1: Reverse Shell

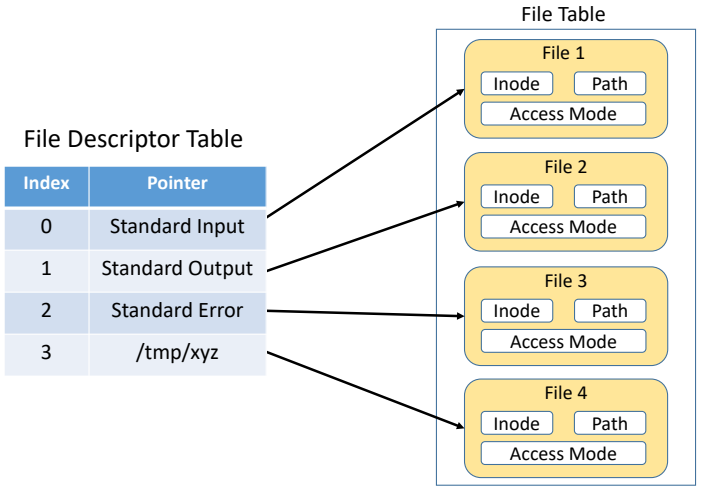


Figure 9.2: File descriptor table

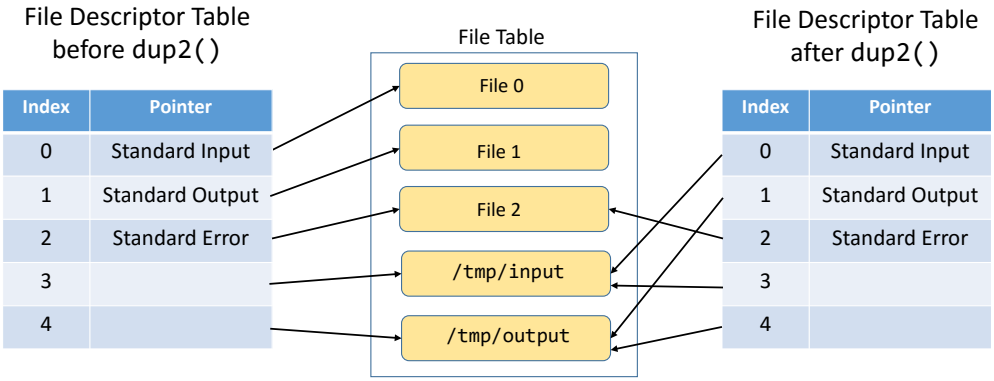


Figure 9.3: The changes of the file descriptor table caused by `dup2()`

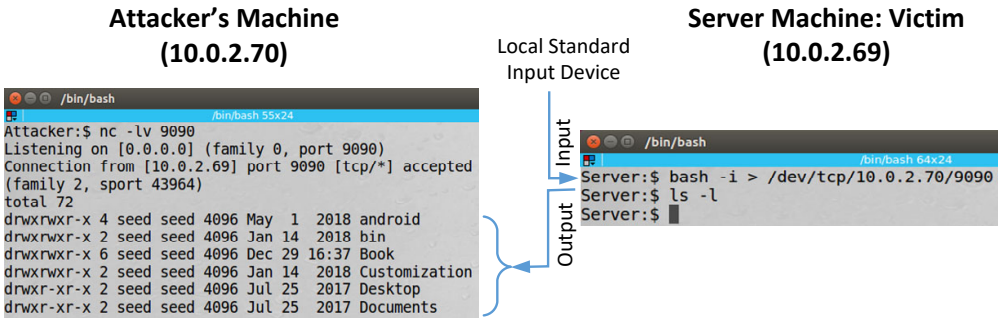


Figure 9.4: Redirect standard output

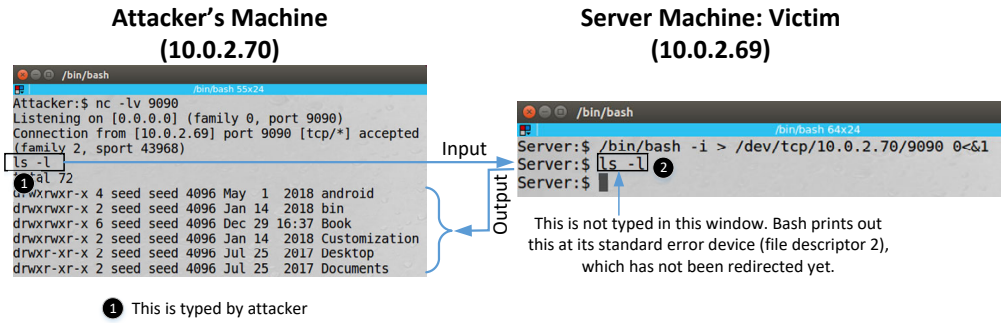


Figure 9.5: Redirect standard input and output

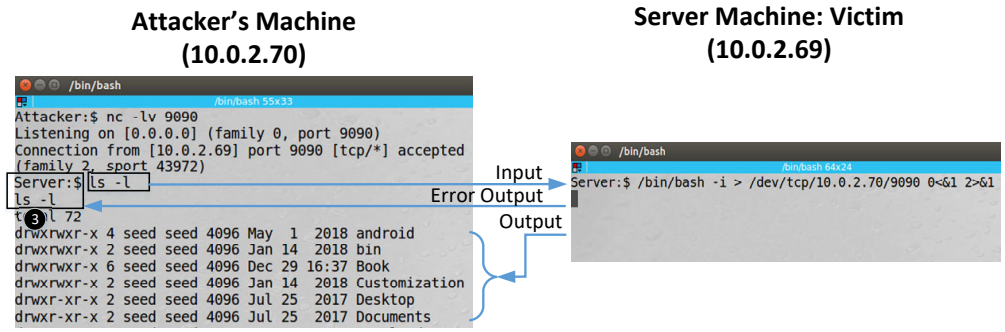


Figure 9.6: Redirect standard input, output, and error

Chapter 10

Cross Site Request Forgery

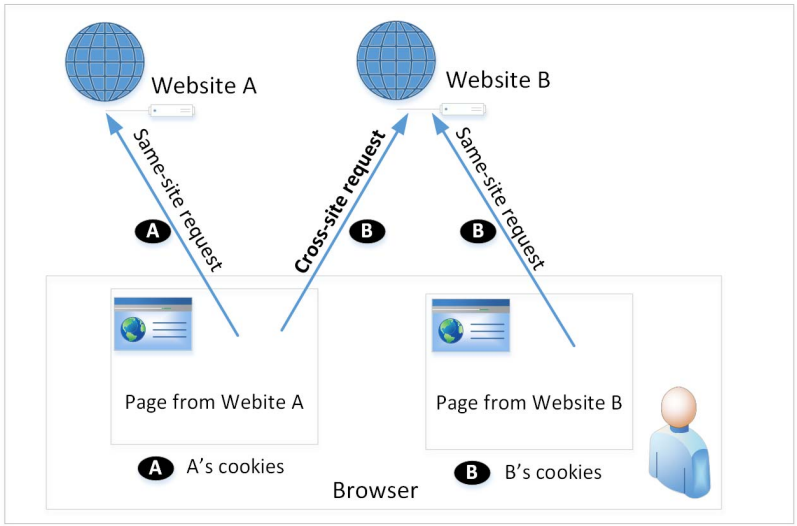


Figure 10.1: Cross-Site Requests

Figure 10.2: How a CSRF attack works

Chapter 11

Cross-Site Scripting Attack

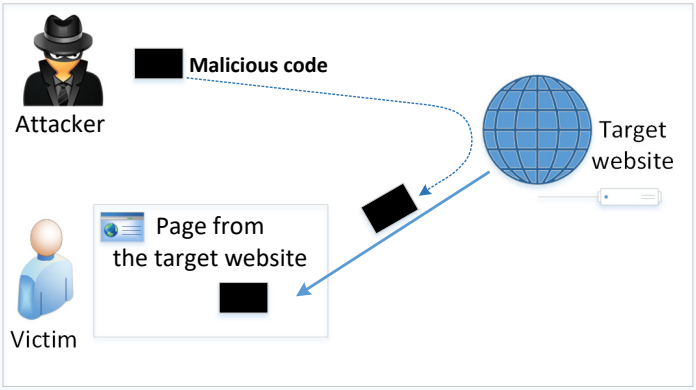
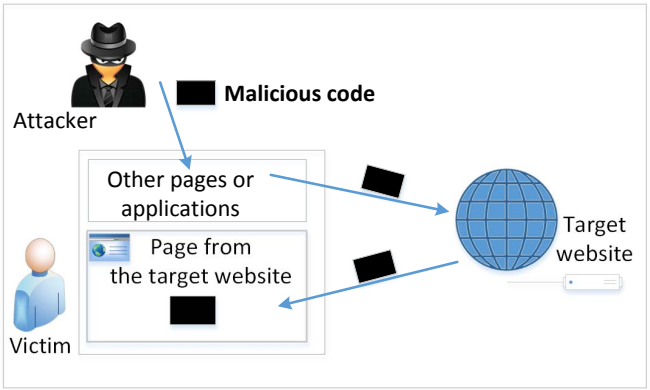
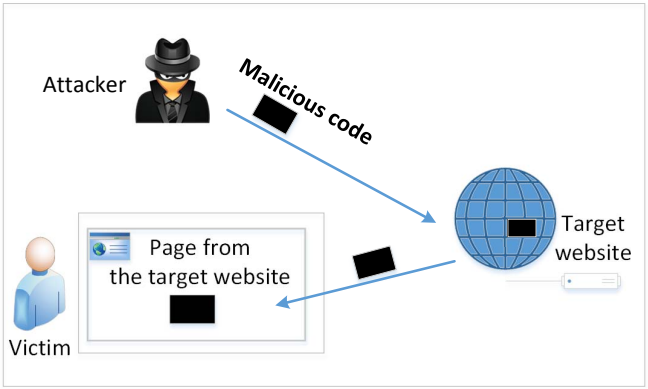


Figure 11.1: The general idea of the XSS attack



(a) Non-persistent (Reflected) XSS attack



(b) Persistent XSS attack

Figure 11.2: Two types of XSS attack

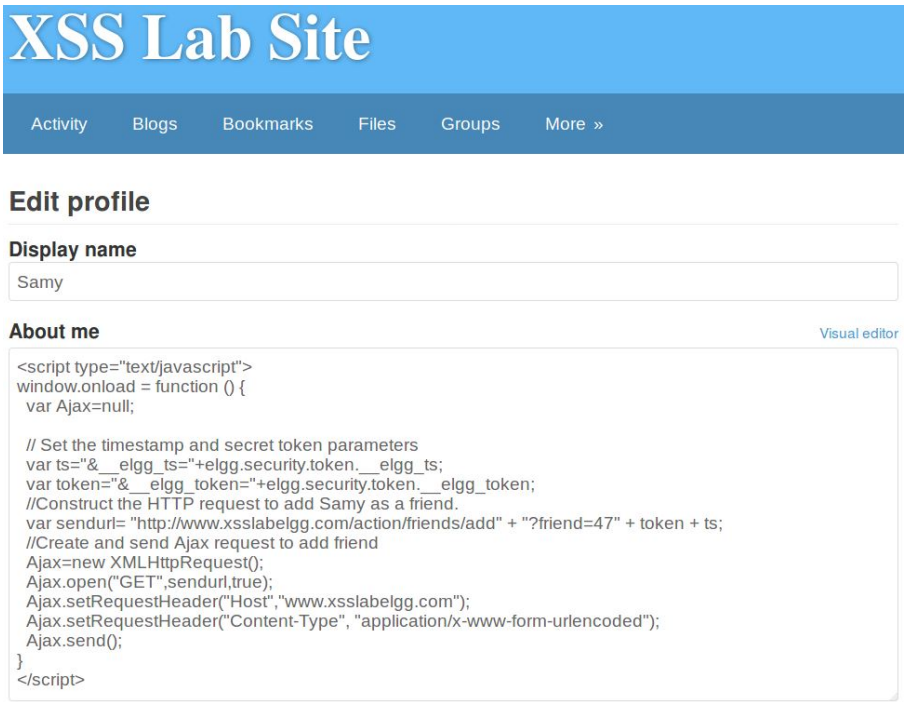


Figure 11.3: Inject JavaScript code to profile

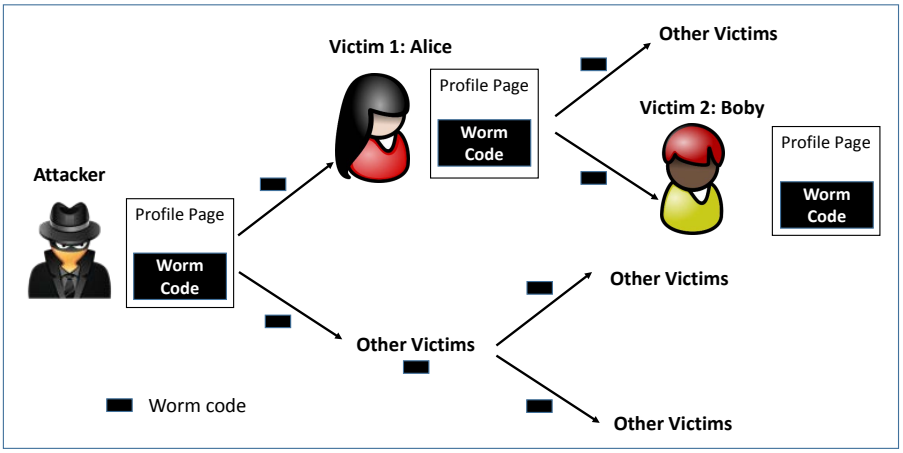


Figure 11.4: Self Propagating XSS Attack

Chapter 12

SQL Injection Attack

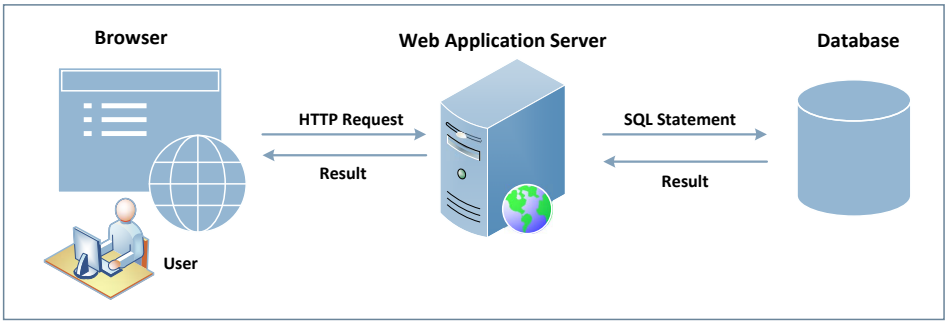


Figure 12.1: Web Architecture

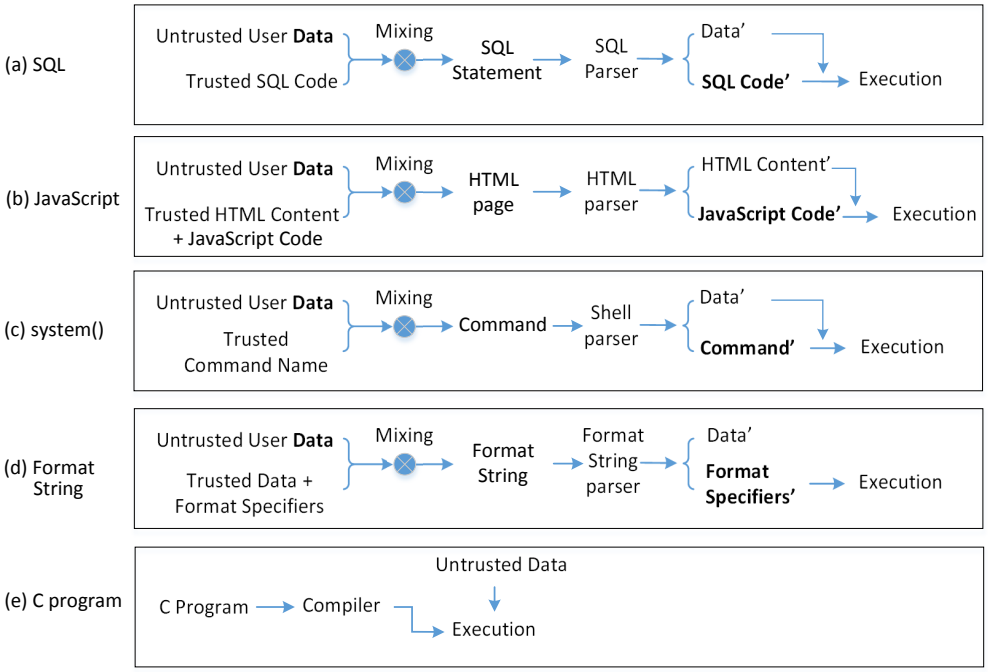


Figure 12.2: Mixing code with data

Chapter 13

Meltdown Attack

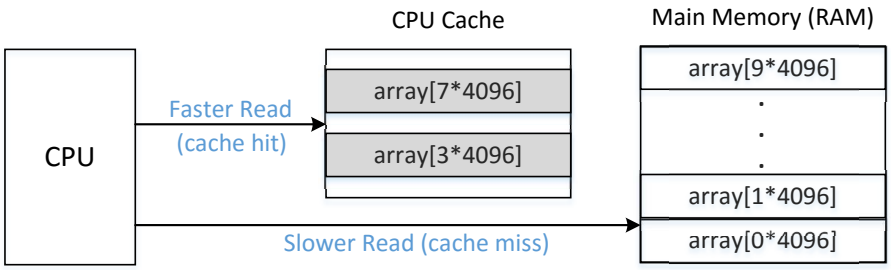


Figure 13.1: Cache hit and miss

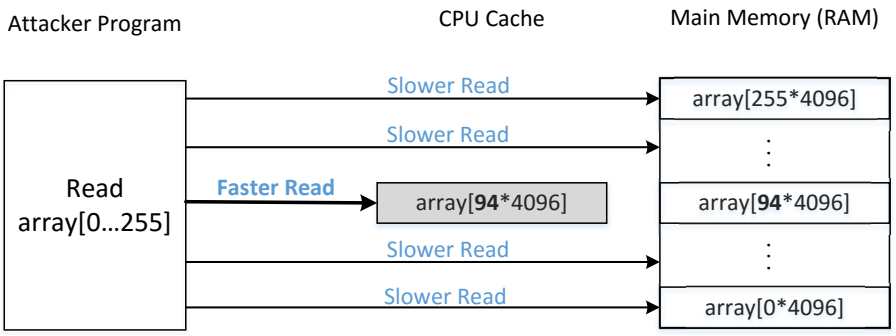


Figure 13.2: Diagram depicting the Side Channel Attack

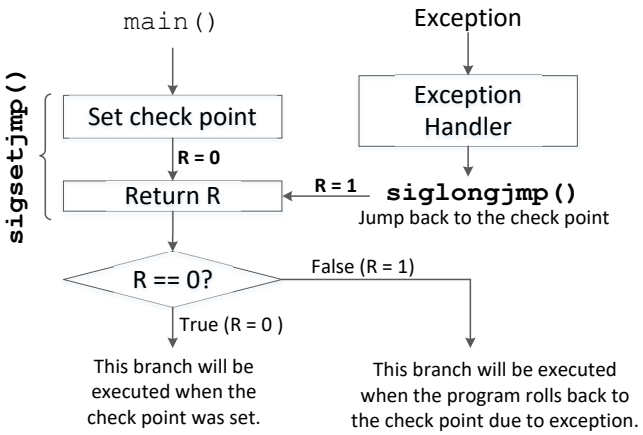


Figure 13.3: Illustration of how exception handling works in C

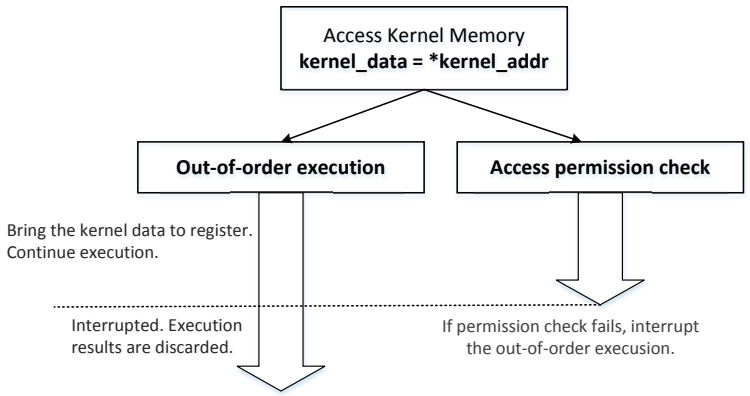


Figure 13.4: Out-of-order execution inside CPU

Chapter 14

Spectre Attack

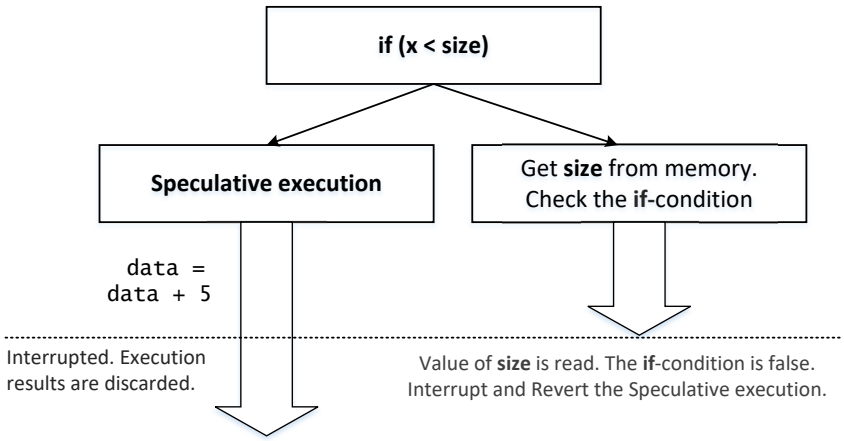


Figure 14.1: Speculative execution (out-of-order execution)

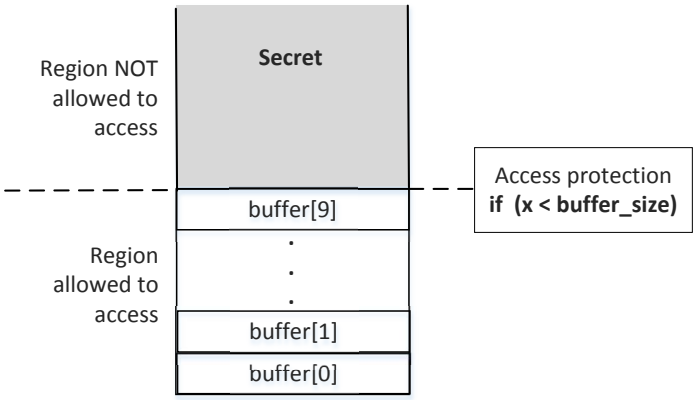


Figure 14.2: Experiment setup: the buffer and the protected secret

Chapter 15

Packet Sniffing and Spoofing

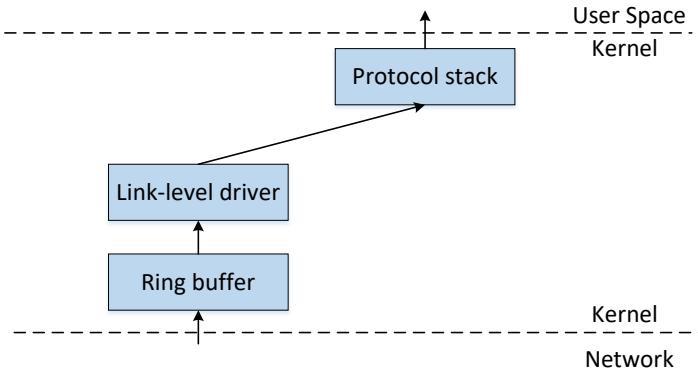


Figure 15.1: Packet flow

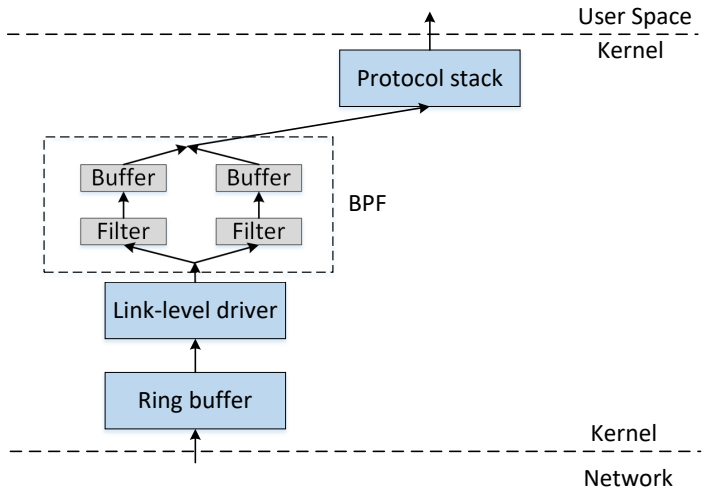


Figure 15.2: Packet flow with filter

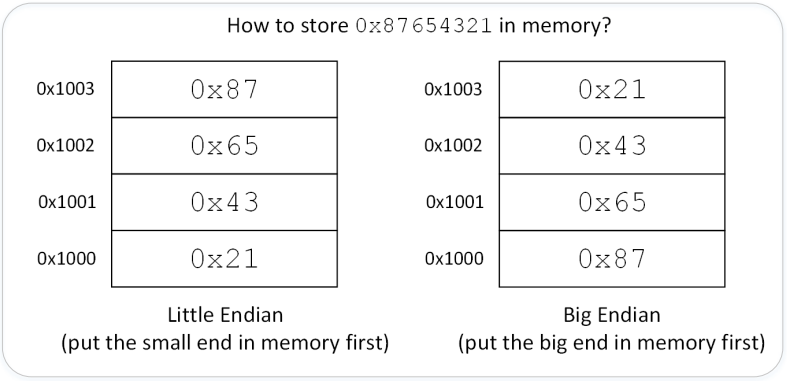


Figure 15.3: Big Endian and Little Endian byte order

Chapter 16

Attacks on the TCP Protocol

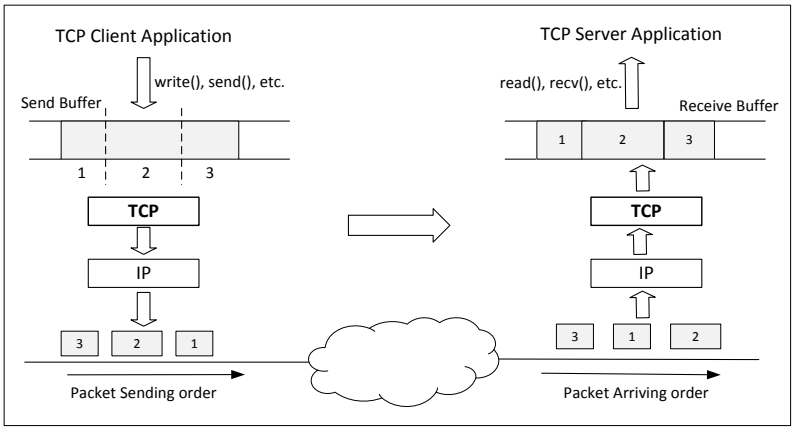


Figure 16.1: How TCP data are transmitted

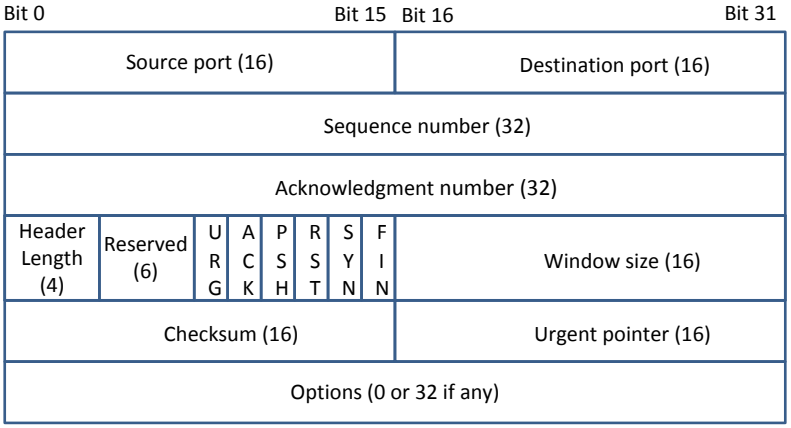


Figure 16.2: TCP Header

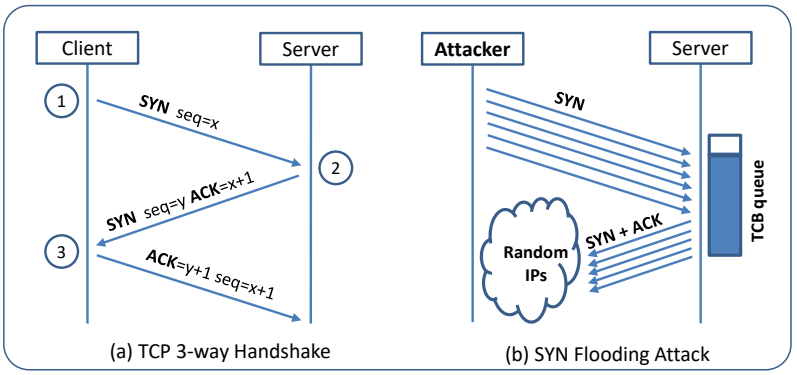


Figure 16.3: TCP Three-way Handshake Protocol and SYN Flooding

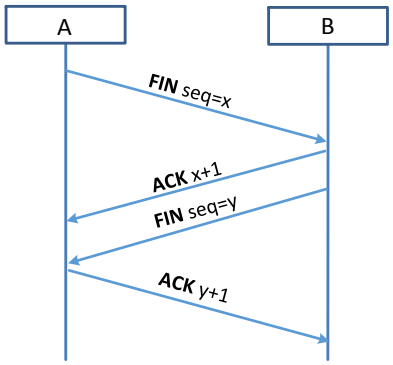
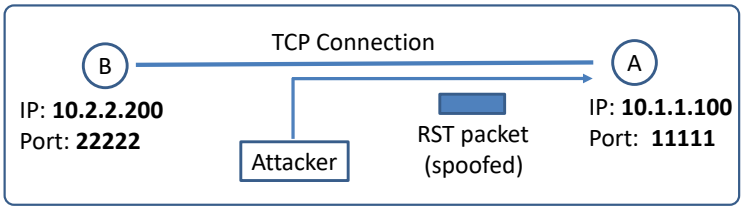
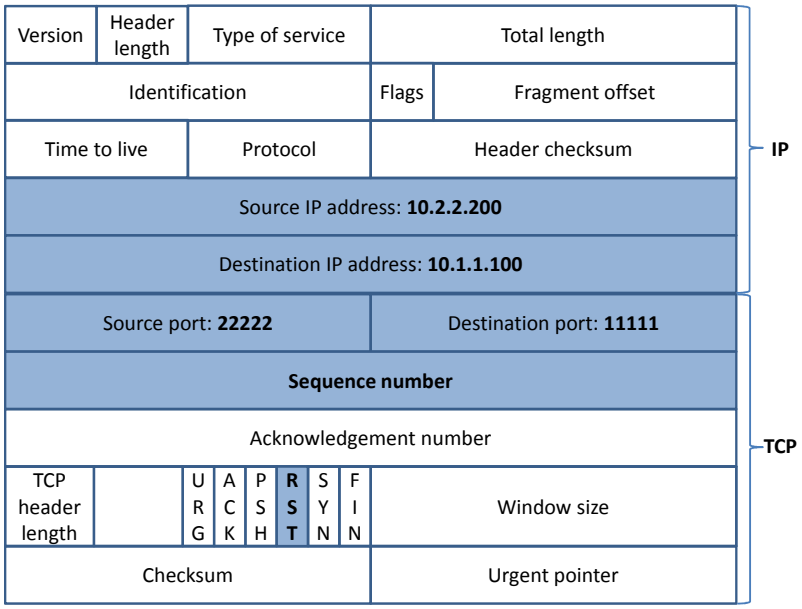


Figure 16.4: TCP FIN Protocol



(a) Attack diagram



(b) Attack packet

Figure 16.5: TCP Reset Attack

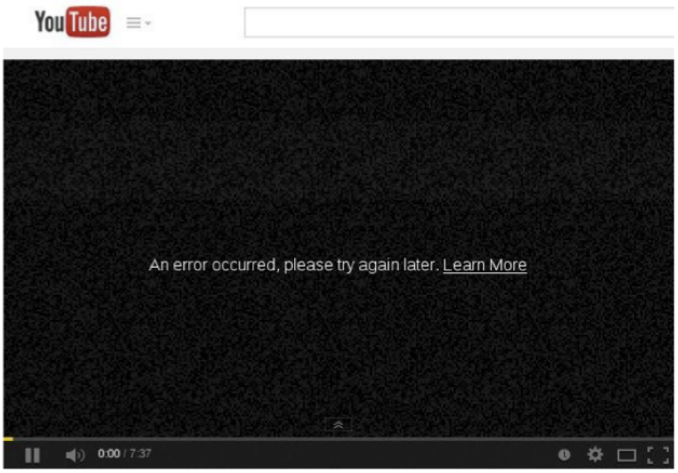
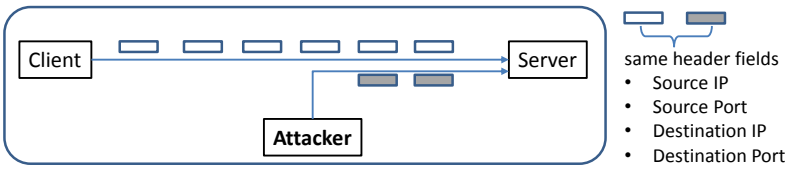
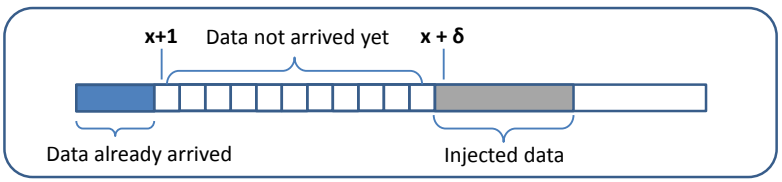


Figure 16.6: TCP Reset attack on video streaming



(a) Injecting data into a TCP connection



(b) Receiver's TCP buffer and sequence numbers

Figure 16.7: TCP Session Hijacking Attack

No.	Source	Destination	Protocol	Length	Info
19	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#2] [TCP ACKed unseen segment]
20	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
21	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#3] [TCP ACKed unseen segment]
22	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
23	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#4] [TCP ACKed unseen segment]
33	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
34	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#5] [TCP ACKed unseen segment]
40	10.0.2.68	10.0.2.69	TELNET	70	[TCP Spurious Retransmission] Telnet Data ...
41	10.0.2.69	10.0.2.68	TCP	78	[TCP Dup ACK 15#6] [TCP ACKed unseen segment]

Figure 16.8: TCP retransmissions caused by the session hijacking attack

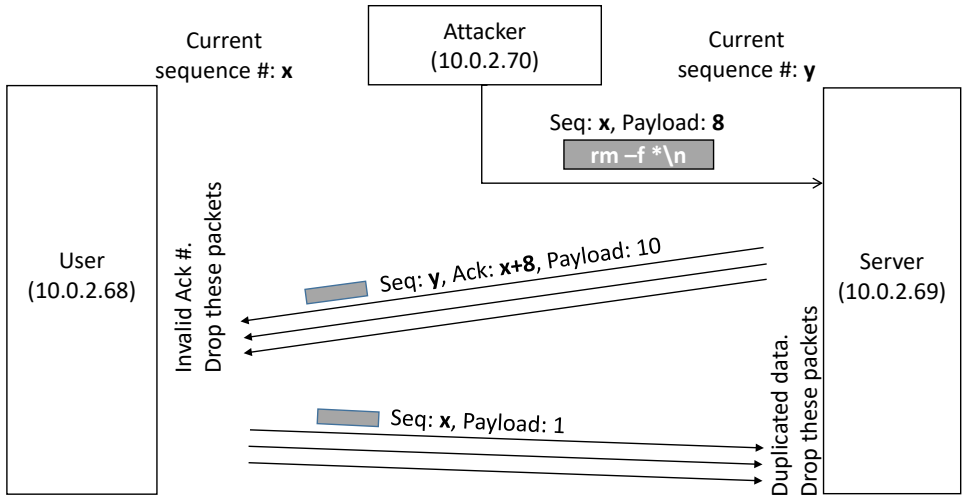


Figure 16.9: Why the connection freezes

Chapter 17

Firewall

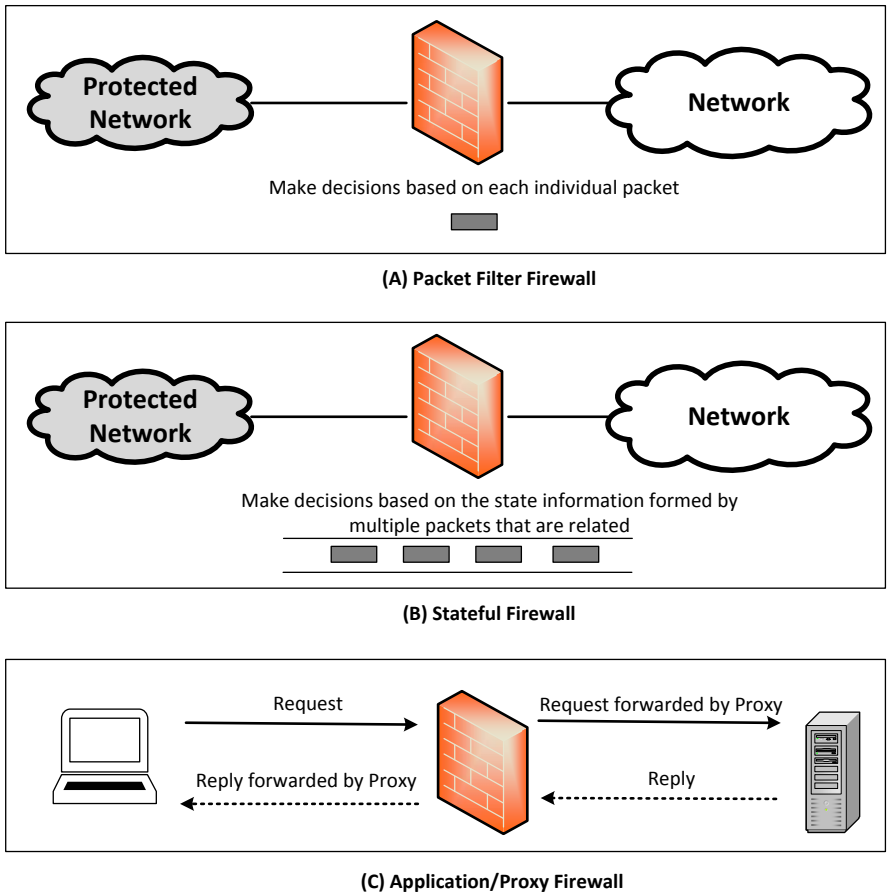


Figure 17.1: Three types of firewall

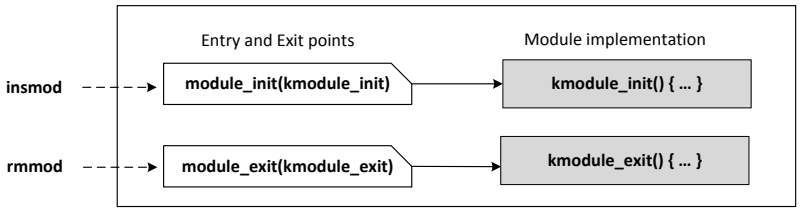


Figure 17.2: Loadable Kernel Modules

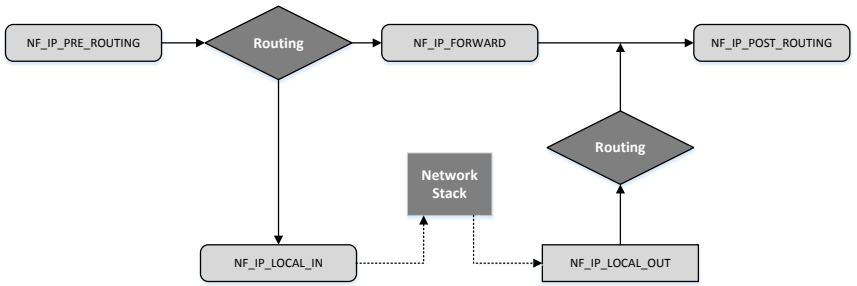


Figure 17.3: `netfilter` hooks in IPv4 stack

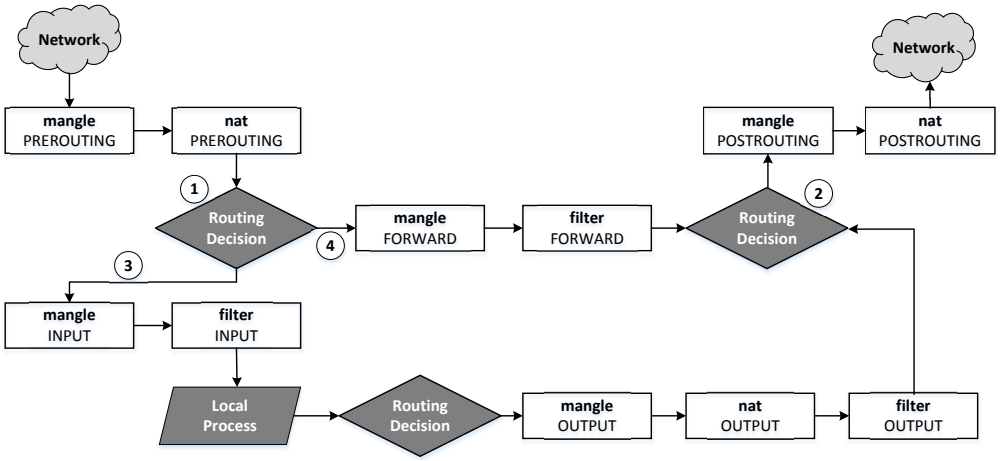


Figure 17.4: Network packet traversal through `iptables`.

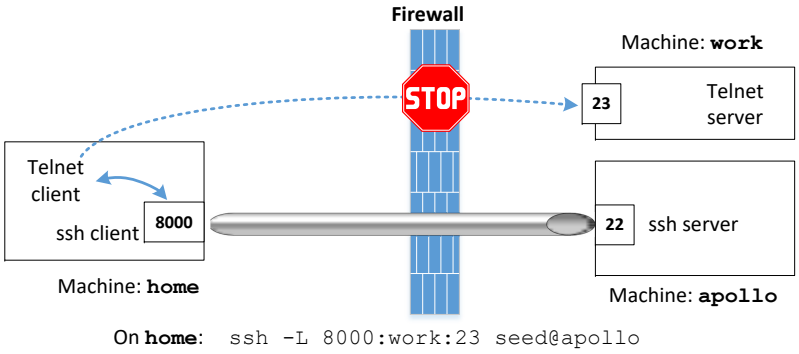


Figure 17.5: Evade firewall using `ssh` tunnel

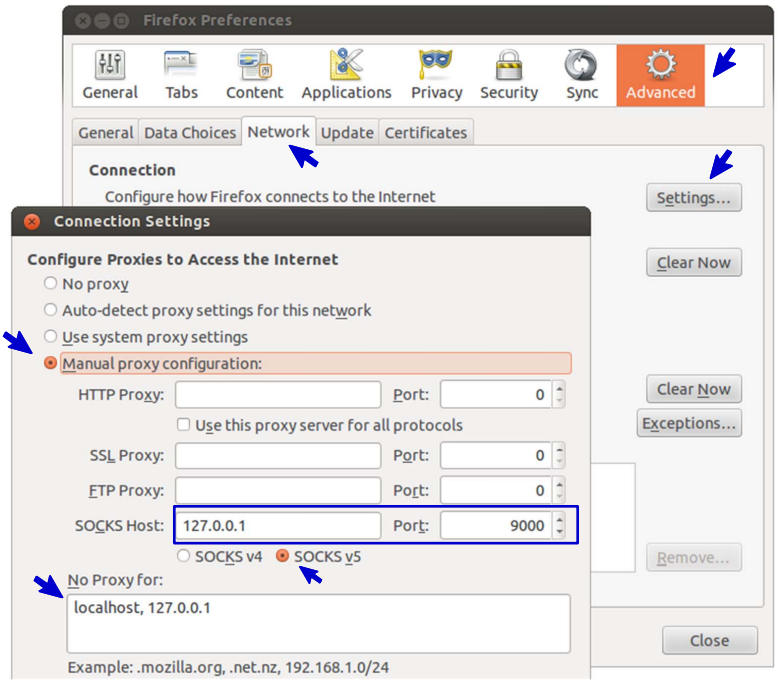


Figure 17.6: Configure the SOCKS Proxy

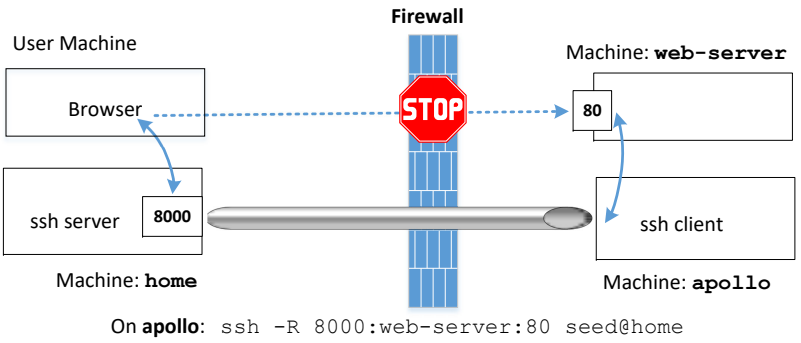


Figure 17.7: Use reverse SSH tunneling to access an internal web server

Chapter 18

Domain Name System (DNS) and Attacks

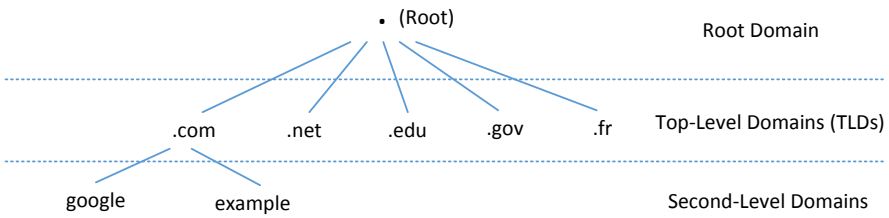


Figure 18.1: Domain Hierarchy

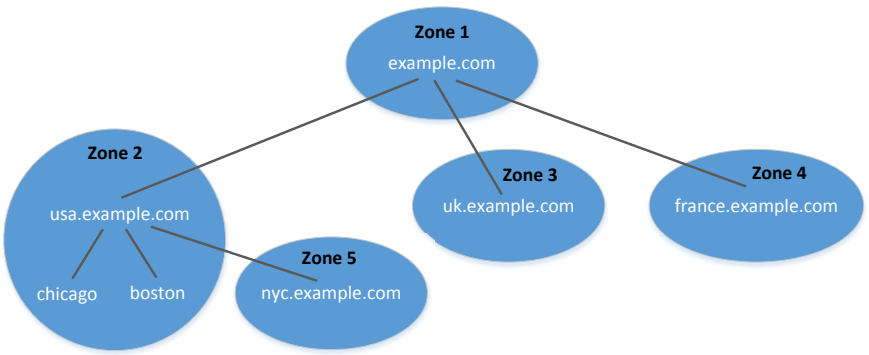


Figure 18.2: Zones for the `example.com` domain (fictitious)

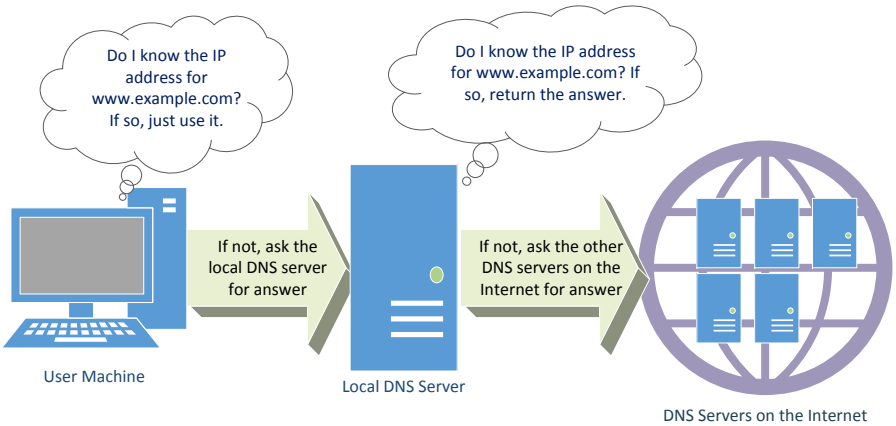


Figure 18.3: A high-level picture of how DNS works

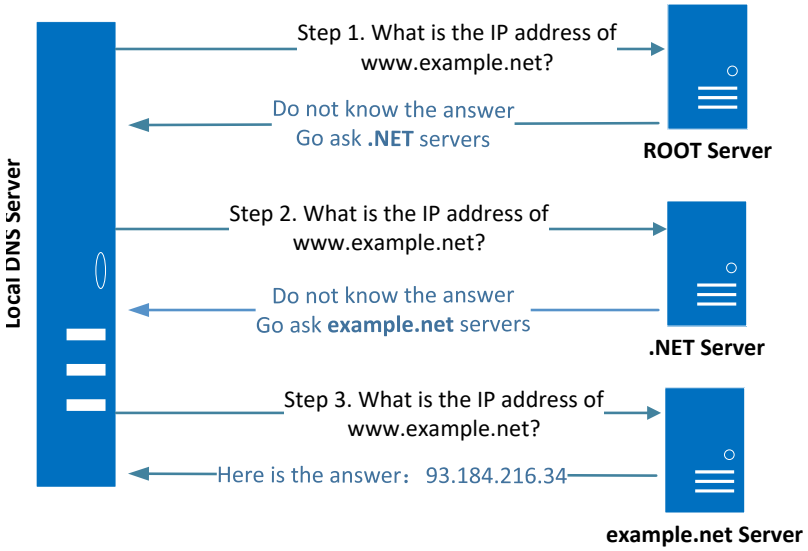


Figure 18.4: The iterative query process (finding the IP address of `www.example.net`)

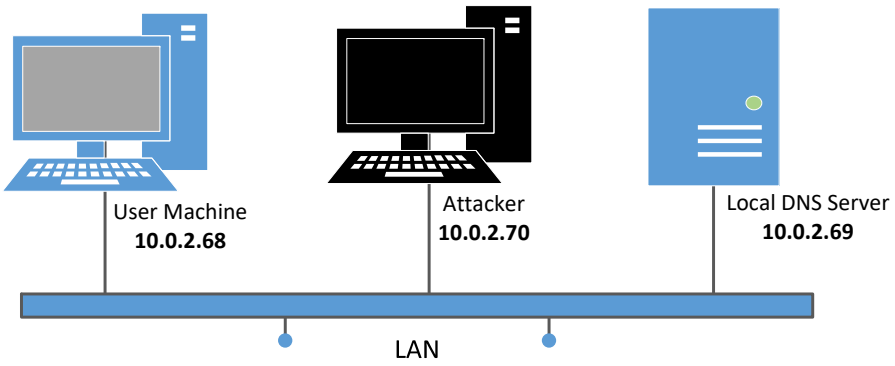


Figure 18.5: Environment setup for the experiment

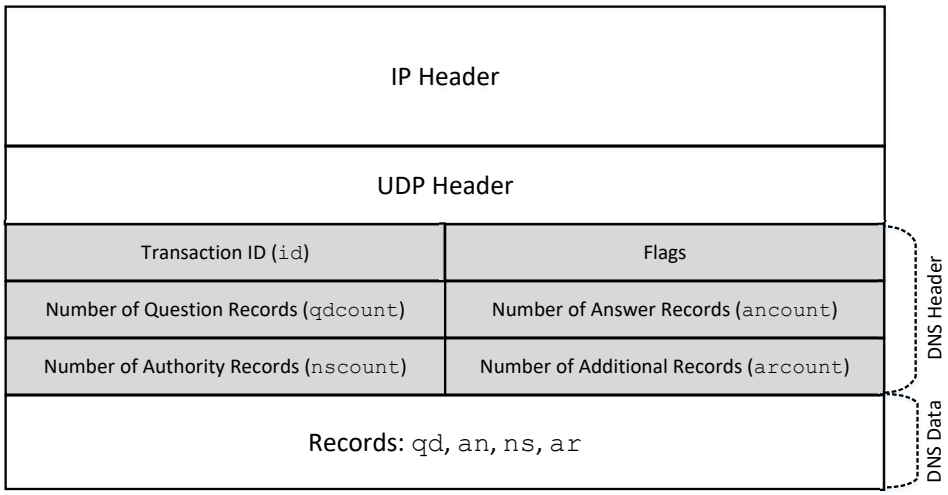


Figure 18.6: DNS packet

Question Record

Name	Record Type	Class
www.example.com	"A" Record 0x0001	Internet 0x0001

Answer Record

Name	Record Type	Class	Time to Live	Data Length	Data: IP Address
www.example.com	"A" Record 0x0001	Internet 0x0001	0x00002000 (seconds)	0x0004	1.2.3.4

Authority Record

Name	Record Type	Class	Time to Live	Data Length	Data: Name Server
example.com	"NS" Record 0x0002	Internet 0x0001	0x00002000 (seconds)	0x0013	ns.example.com

Figure 18.7: DNS records

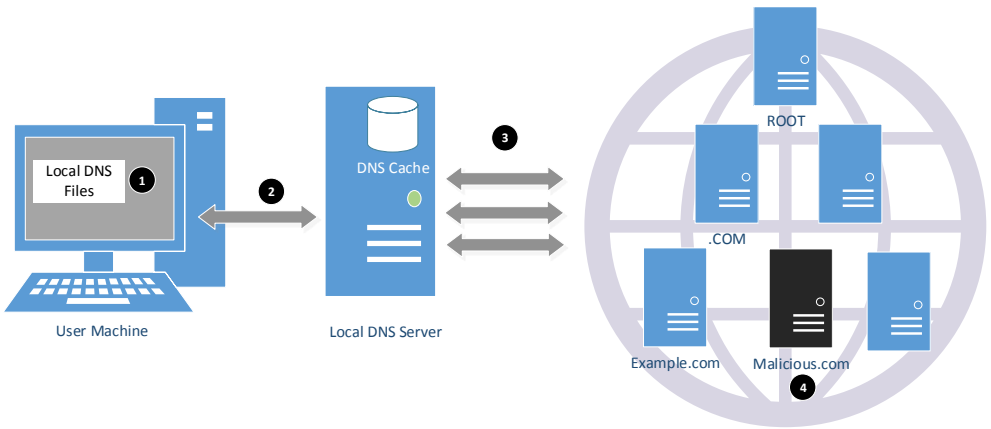


Figure 18.8: DNS Attack Surfaces

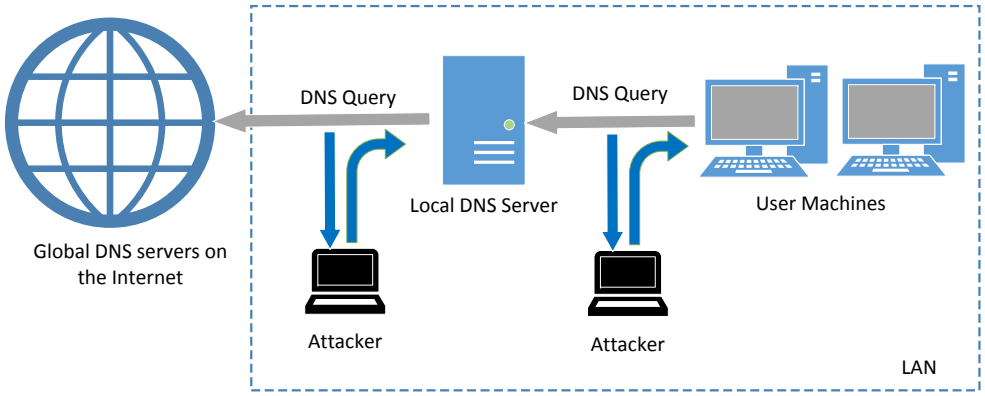


Figure 18.9: Local DNS Poisoning Attack

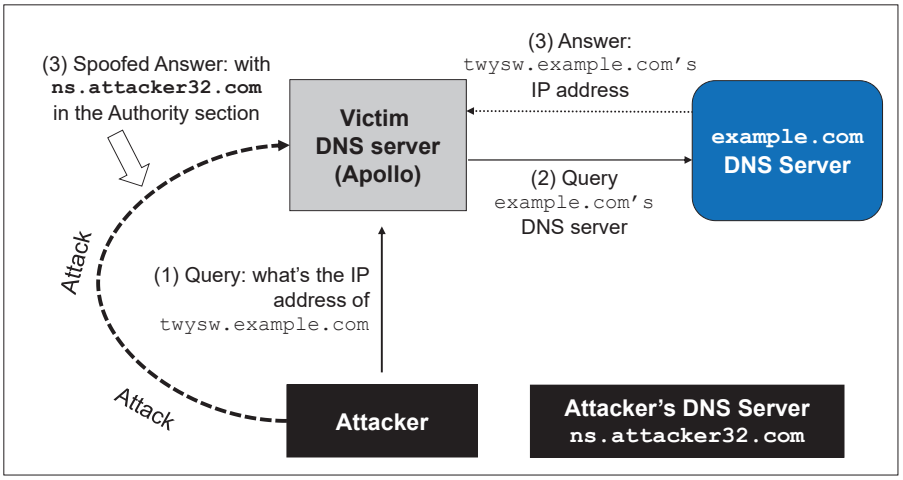


Figure 18.10: The Kaminsky attack (assuming that `Apollo` already knows the authoritative nameserver of `example.com`

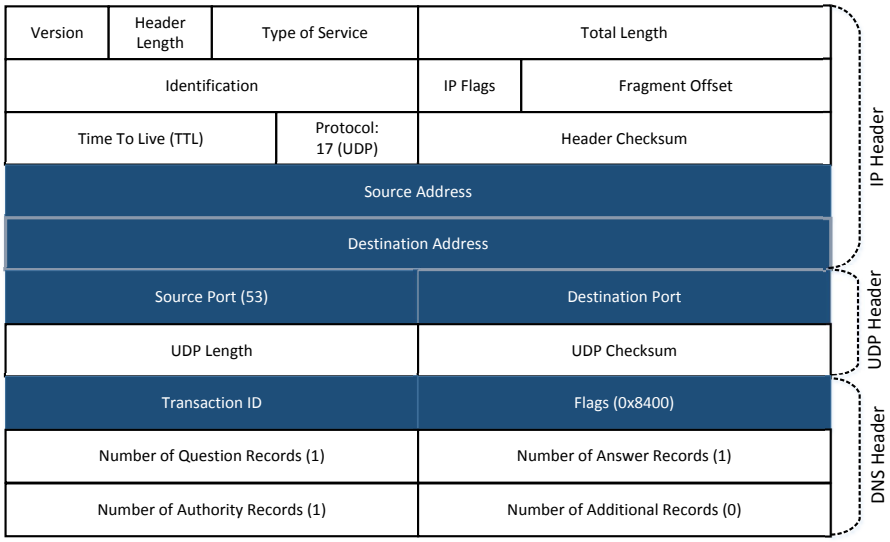


Figure 18.11: The IP, UDP, and DNS headers of the spoofed DNS reply

Question Record

Name	Record Type	Class
twysw.example.com	"A" Record 0x0001	Internet 0x0001

Answer Record

Name	Record Type	Class	Time to Live	Data Length	Data: IP Address
twysw.example.com	"A" Record 0x0001	Internet 0x0001	0x00002000 (seconds)	0x0004	1.2.3.4

Authority Record

Name	Record Type	Class	Time to Live	Data Length	Data: Name Server
example.com	"NS" Record 0x0002	Internet 0x0001	0x00002000 (seconds)	0x0013	ns.attacker32.net

Representation in the packet
(Total: 0x13 bytes)

02	n	s	0a	a	t	t	a	c	k	e	r	3	2	03	c	o	m	00

Figure 18.12: The DNS payload of the forged response packet

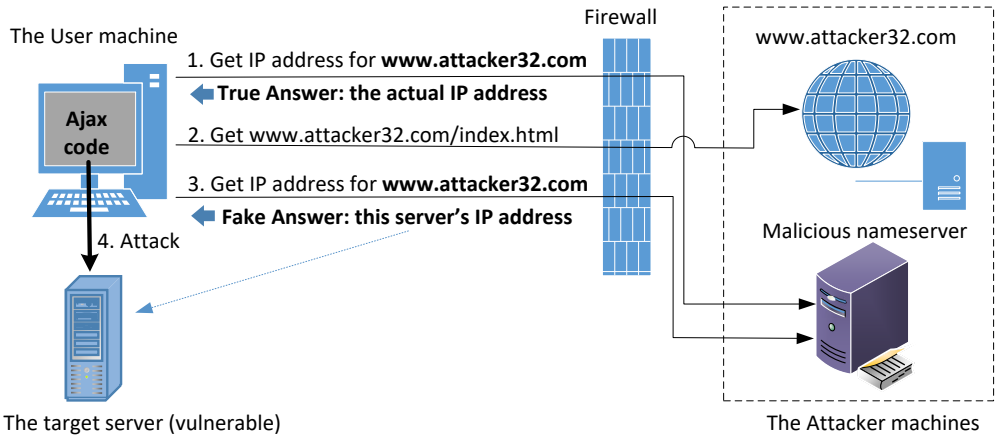


Figure 18.13: DNS Rebinding Attack

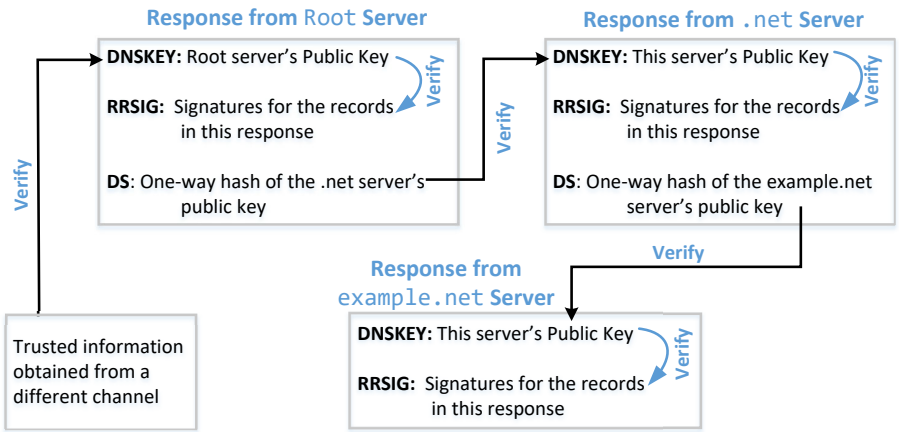


Figure 18.14: Chain of Trust in DNSSEC

Chapter 19

Virtual Private Network

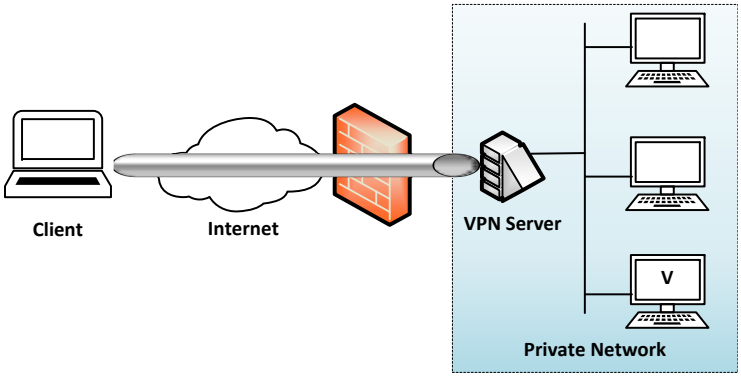
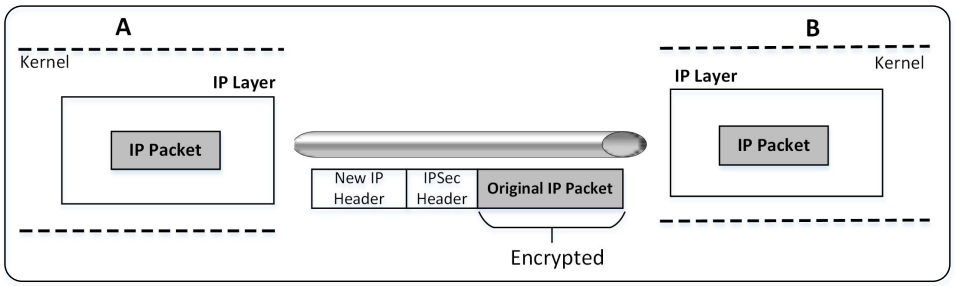
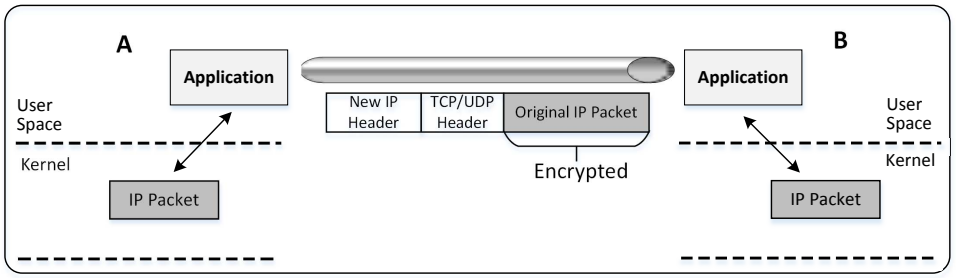


Figure 19.1: Overview of a Virtual Private Network.



(a) IPSec Tunneling



(b) TLS/SSL Tunneling

Figure 19.2: IPSec and TLS/SSL Tunneling

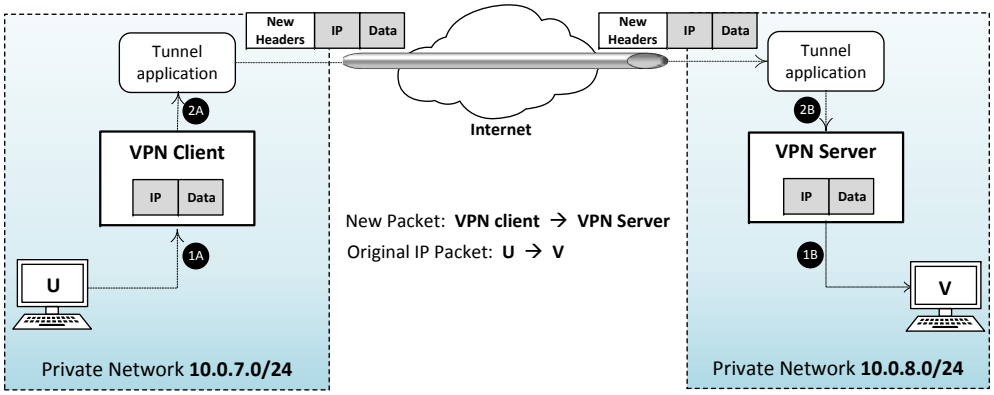


Figure 19.3: Overview of the operation of a tunneling application.

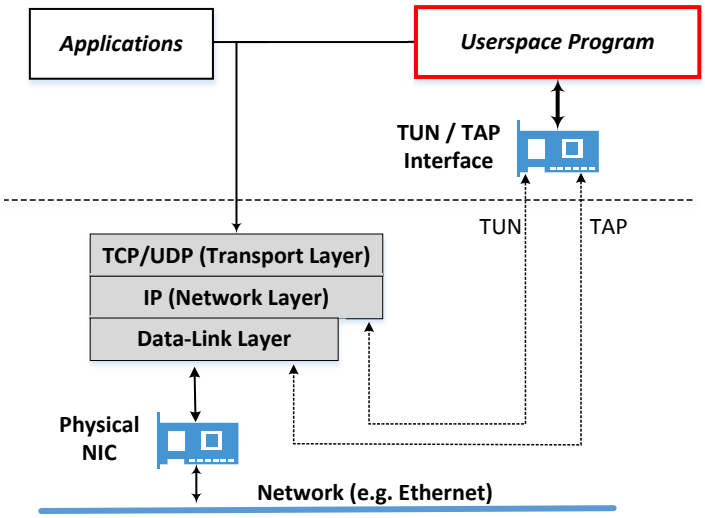


Figure 19.4: Virtual Network Interfaces

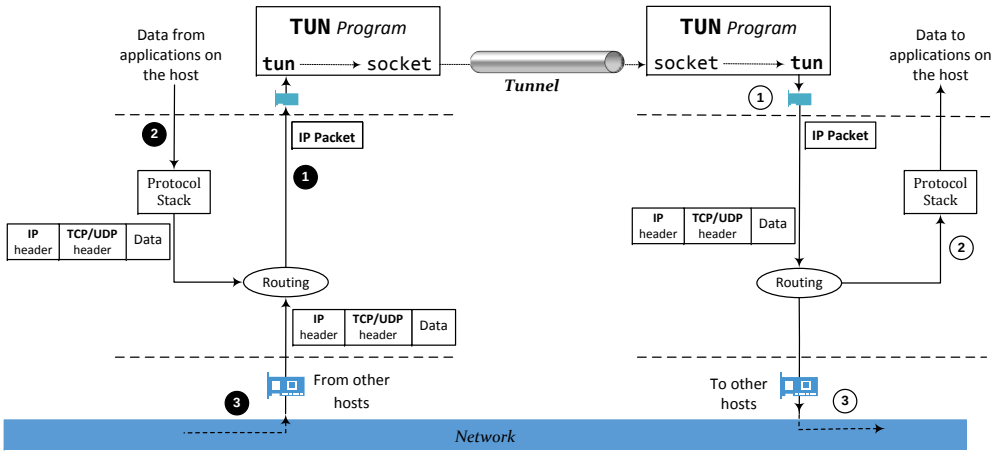
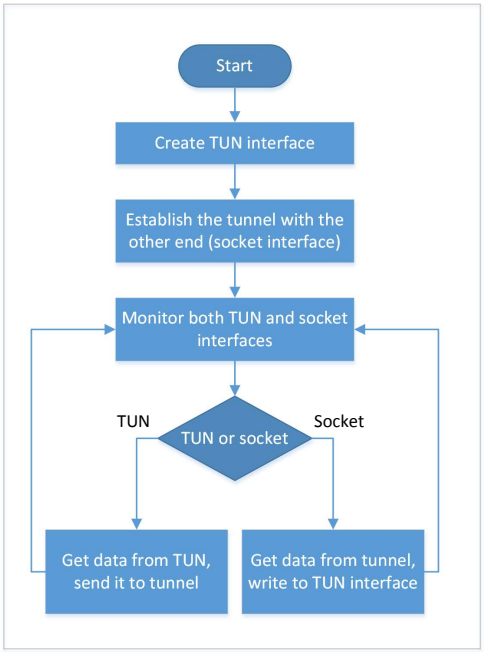
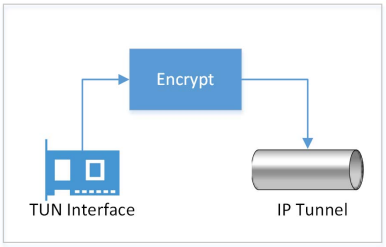


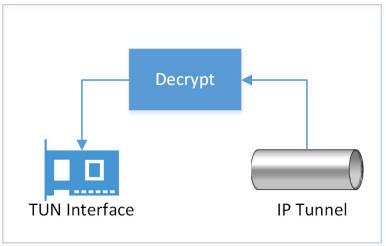
Figure 19.5: Detailed tunnel view



(a) The Flow of the Program



(b) From TUN Interface to Tunnel



(c) From Tunnel to TUN Interface

Figure 19.6: How a sample VPN program is implemented

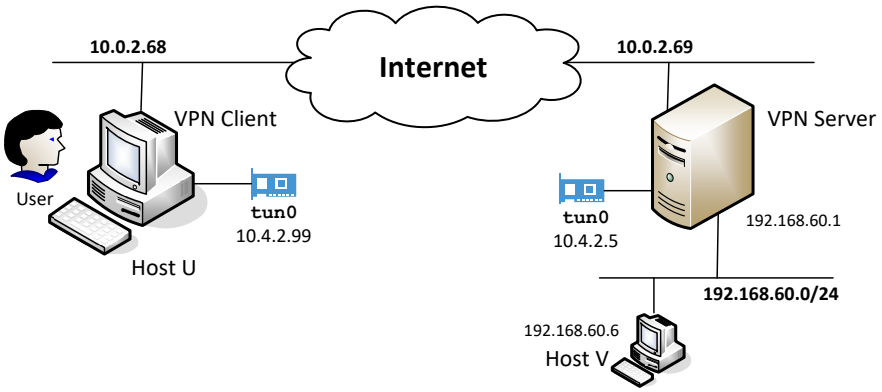


Figure 19.7: Network setup for a VPN.

No.	Source	Destination	Protocol	Info
1	10.4.2.99	192.168.60.6	ICMP	Echo (ping) request id=0x0286,
2	10.0.2.68	10.0.2.69	UDP	54915 → 55555 Len=84
3	10.0.2.69	10.0.2.68	UDP	55555 → 54915 Len=84
4	192.168.60.6	10.4.2.99	ICMP	Echo (ping) reply id=0x0286,
5	10.4.2.99	192.168.60.6	ICMP	Echo (ping) request id=0x0286,
6	10.0.2.68	10.0.2.69	UDP	54915 → 55555 Len=84
7	10.0.2.69	10.0.2.68	UDP	55555 → 54915 Len=84
8	192.168.60.6	10.4.2.99	ICMP	Echo (ping) reply id=0x0286,

Figure 19.8: Packets generated when pinging Host V from Host U

No.	Source	Destination	Protocol	Info
32	10.4.2.99	192.168.60.6	TELNET	Telnet Data ...
33	10.0.2.68	10.0.2.69	UDP	37674 → 55555 Len=54
34	10.0.2.69	10.0.2.68	ICMP	Destination unreachable (Port unreachable)
35	10.4.2.99	192.168.60.6	TCP	[TCP Retransmission] 45654 → 23 [PSH, ACK] Seq=340884658
36	10.0.2.68	10.0.2.69	UDP	37674 → 55555 Len=54
37	10.0.2.69	10.0.2.68	ICMP	Destination unreachable (Port unreachable)
38	10.4.2.99	192.168.60.6	TCP	[TCP Retransmission] 45654 → 23 [PSH, ACK] Seq=340884658
39	10.0.2.68	10.0.2.69	UDP	37674 → 55555 Len=54
40	10.0.2.69	10.0.2.68	ICMP	Destination unreachable (Port unreachable)
41	10.4.2.99	192.168.60.6	TCP	[TCP Retransmission] 45654 → 23 [PSH, ACK] Seq=340884658
42	10.0.2.68	10.0.2.69	UDP	37674 → 55555 Len=54
43	10.0.2.69	10.0.2.68	ICMP	Destination unreachable (Port unreachable)
44	10.4.2.99	192.168.60.6	TCP	[TCP Retransmission] 45654 → 23 [PSH, ACK] Seq=340884658

Figure 19.9: Network traffic after we break up a VPN connection

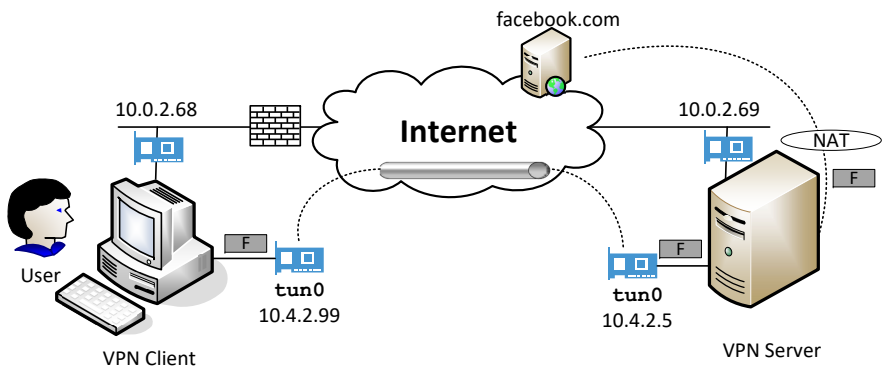


Figure 19.10: Bypassing firewall using VPN

Chapter 20

The Heartbleed Bug and Attack

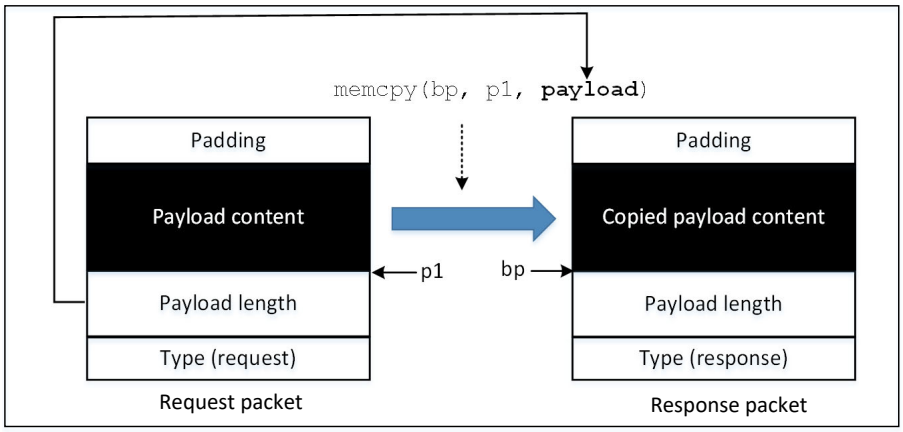


Figure 20.1: How the Heartbeat protocol copies the payload

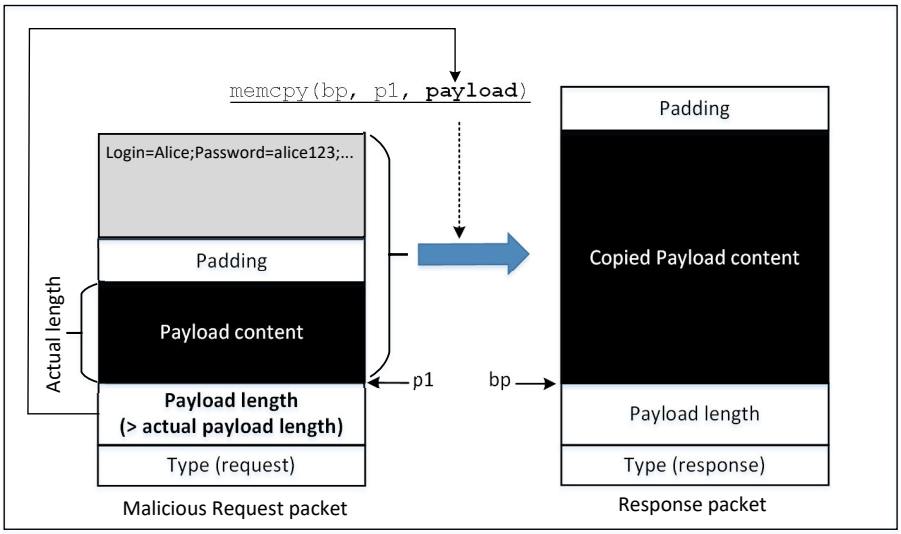


Figure 20.2: How the HeartBleed attack works

Chapter 21

Secret-Key Encryption

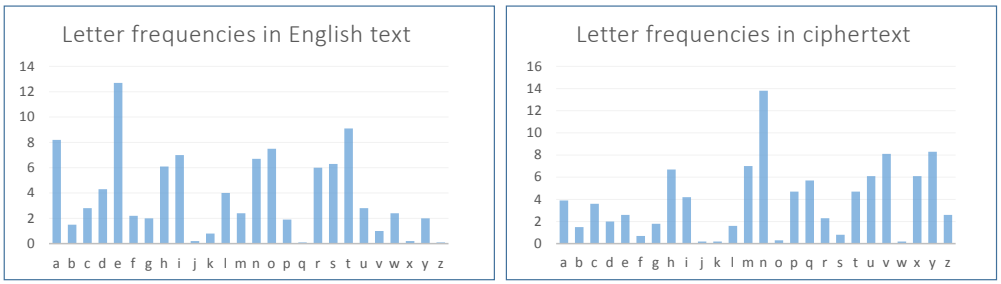


Figure 21.1: Frequencies of letters (The Y-axis is the percentage)

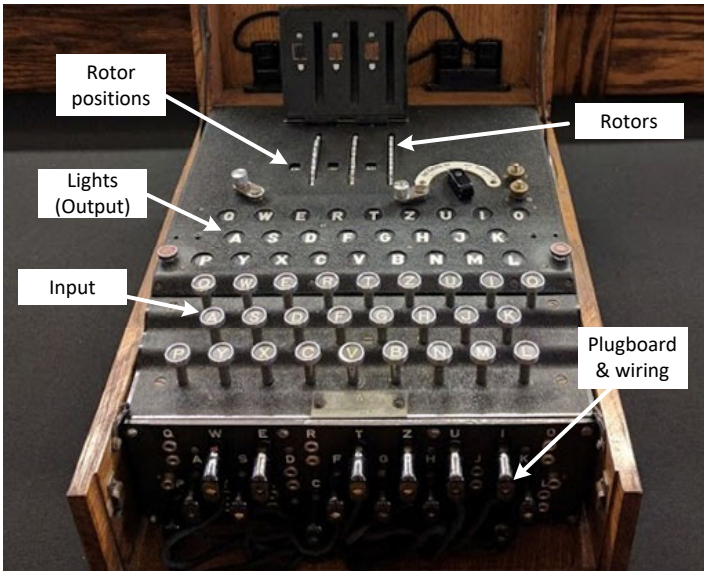
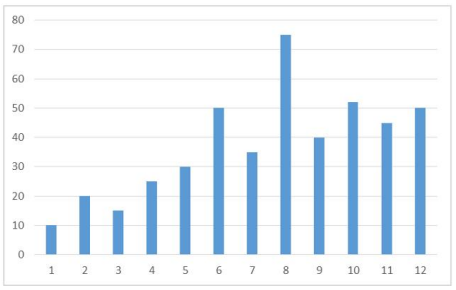
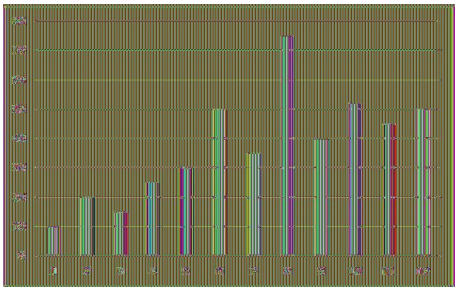


Figure 21.2: Enigma machine

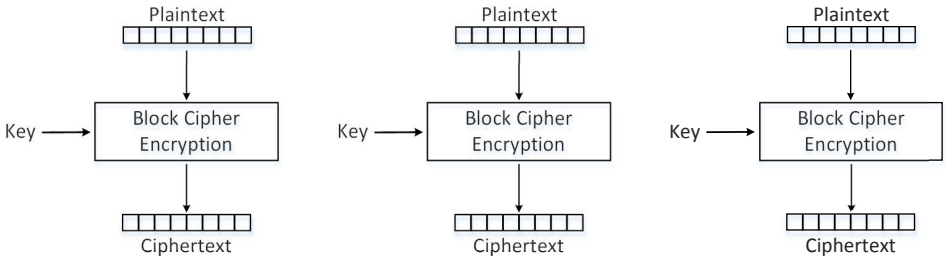


(a) The original image (`pic_original.bmp`)

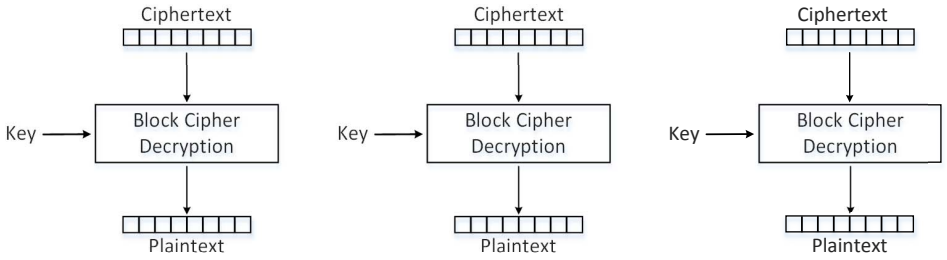


(b) The encrypted image (`pic_encrypted.bmp`)

Figure 21.3: The result of the naive encryption approach.

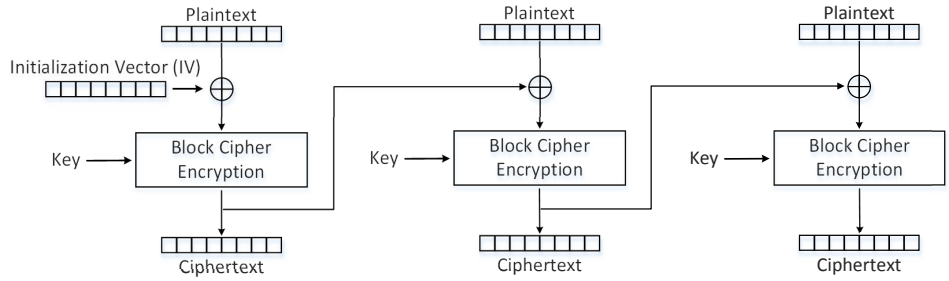


(a) Electronic Codebook (ECB) mode encryption

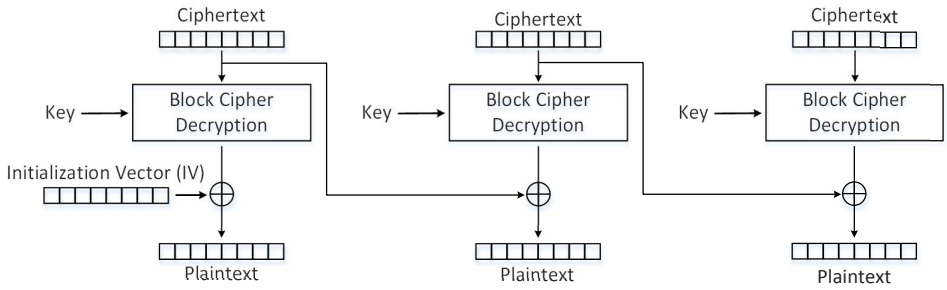


(b) Electronic Codebook (ECB) mode decryption

Figure 21.4: Electronic codebook (ECB) mode

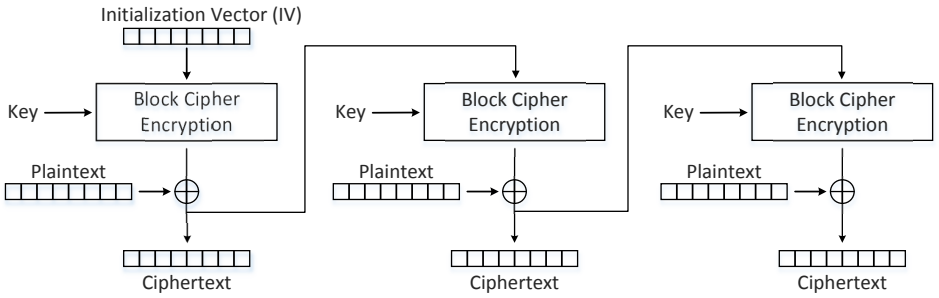


(a) Cipher Block Chaining (CBC) mode encryption

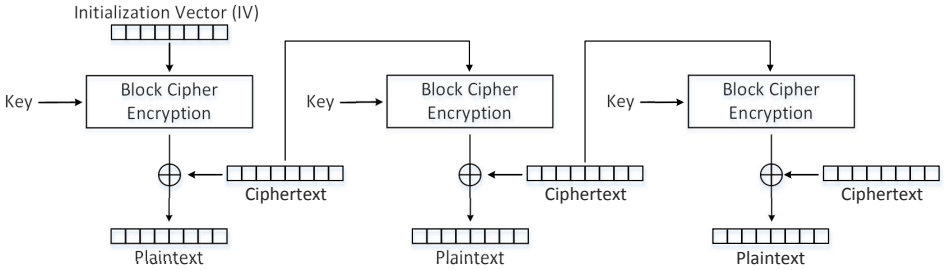


(b) Cipher Block Chaining (CBC) mode decryption

Figure 21.5: Cipher Block Chaining (CBC) mode

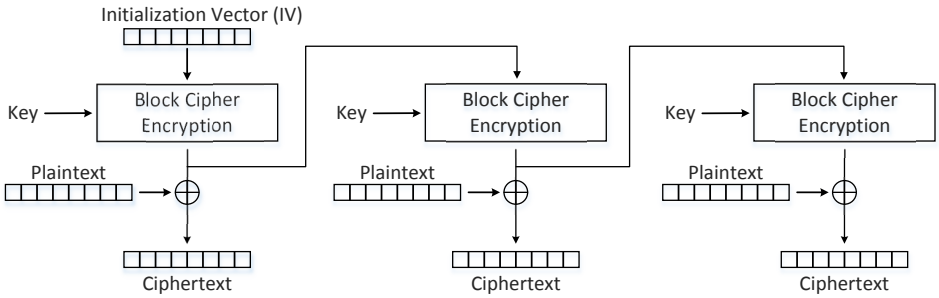


(a) Cipher Feedback (CFB) mode encryption

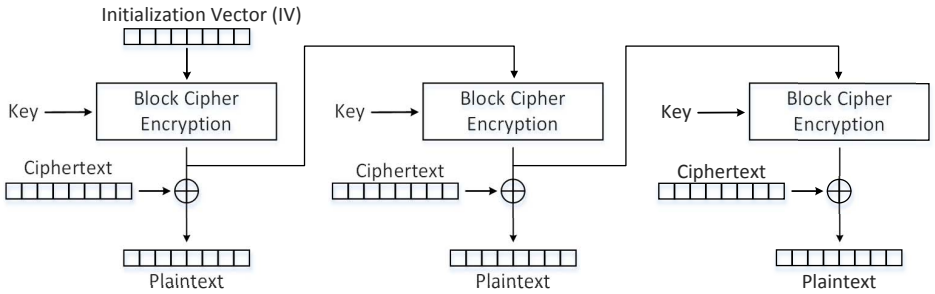


(b) Cipher Feedback (CFB) mode decryption

Figure 21.6: Cipher Feedback (CFB) mode

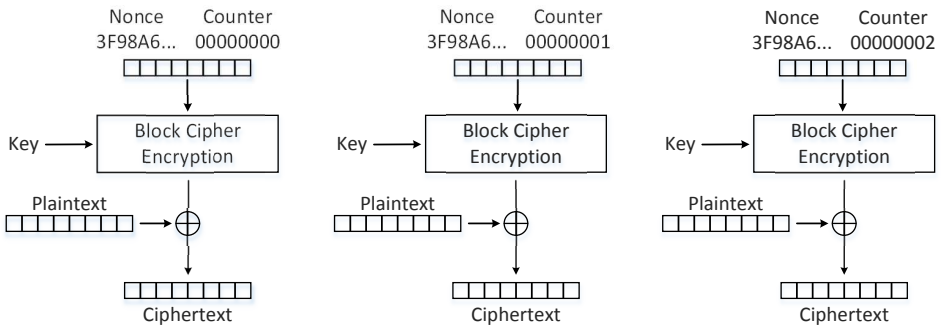


(a) Output Feedback (OFB) mode encryption

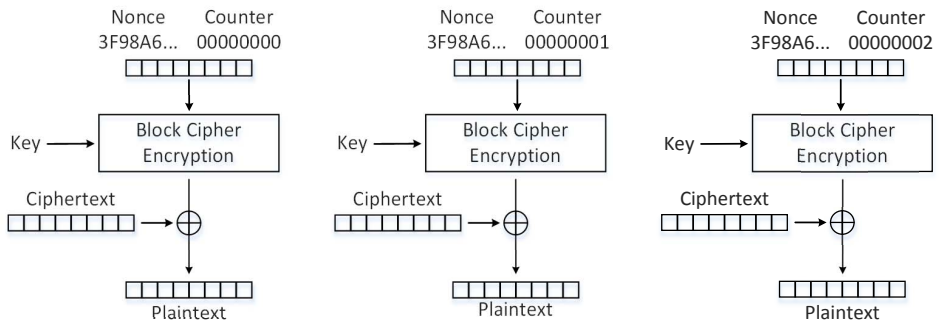


(b) Output Feedback (OFB) mode decryption

Figure 21.7: Output Feedback (OFB) mode

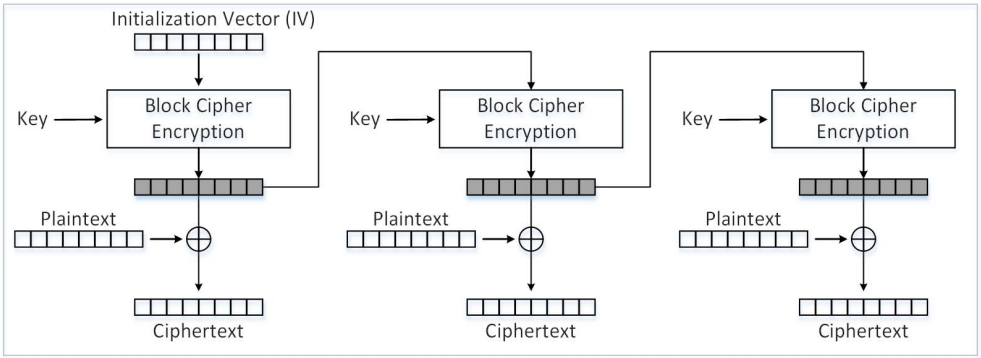


(a) Counter (CTR) mode encryption

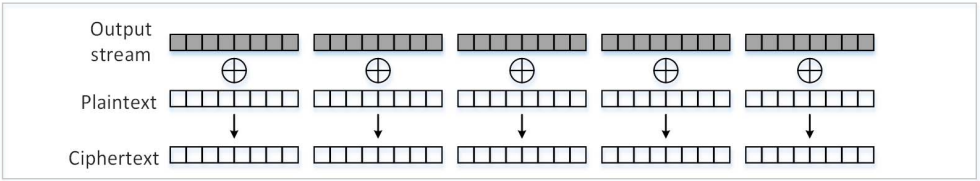


(b) Counter (CTR) mode decryption

Figure 21.8: Counter (CTR) mode



(a) Output Feedback (OFB) mode encryption



(b) XOR the plaintext with the output stream

Figure 21.9: Reusing IVs in the OFB mode

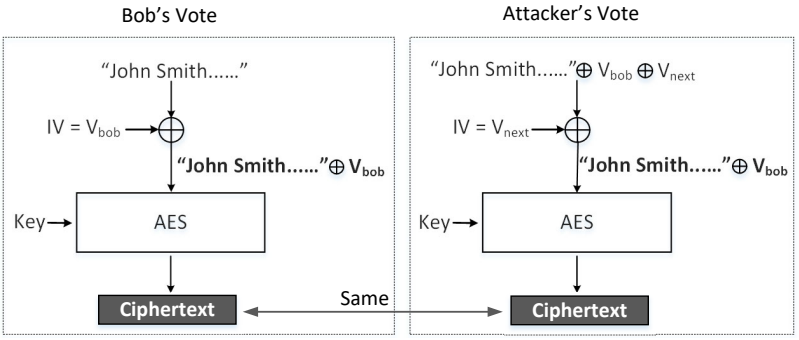


Figure 21.10: Attack on CBC when IV is predictable

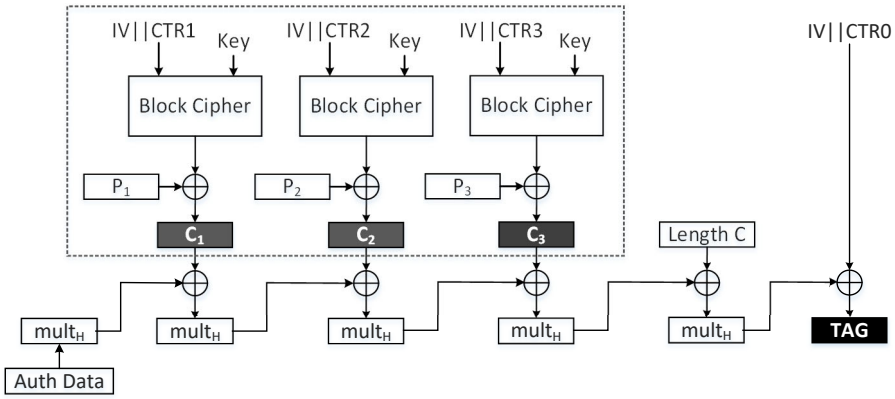


Figure 21.11: The GCM Encryption Mode

Chapter 22

One-Way Hash Function

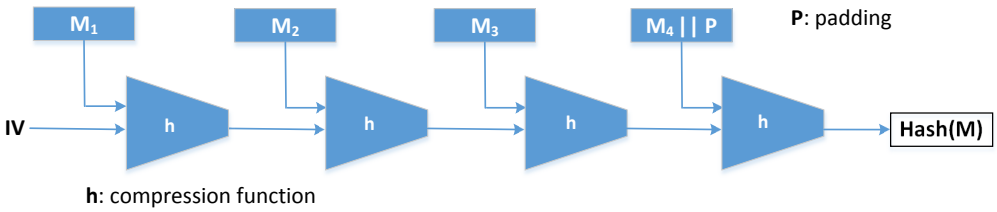


Figure 22.1: Merkle-Damgard Construction

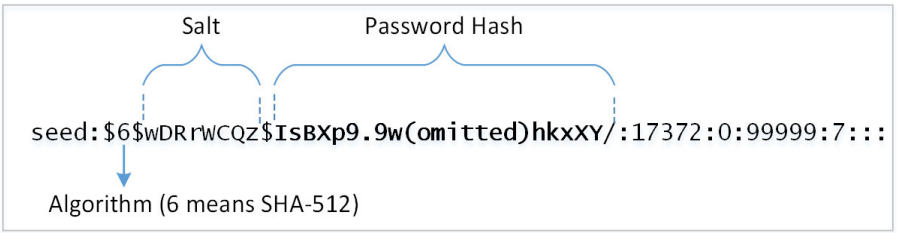


Figure 22.2: Password entry in `/etc/shadow`

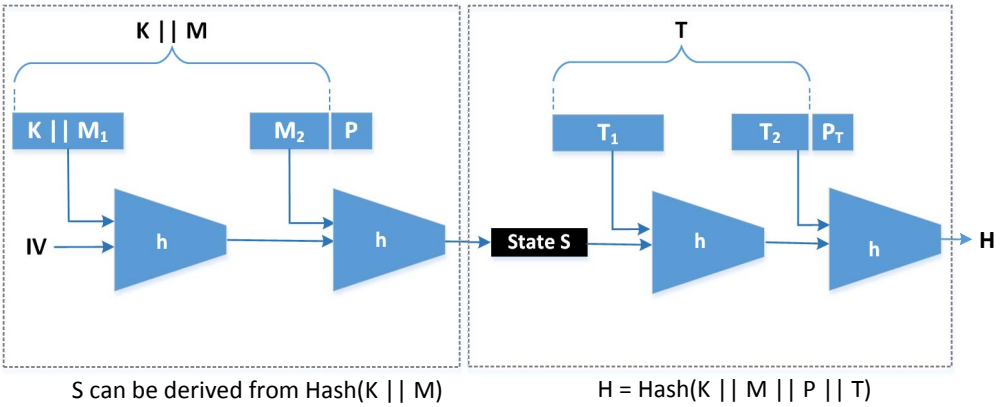


Figure 22.3: Length extension attack

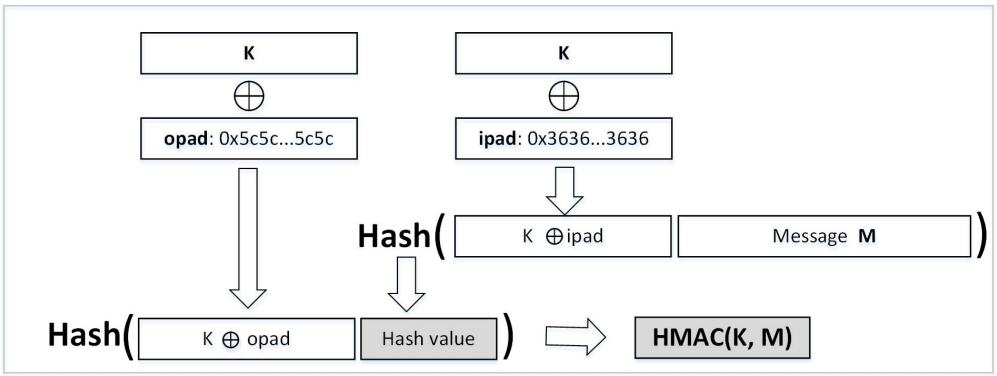


Figure 22.4: The HMAC Algorithm

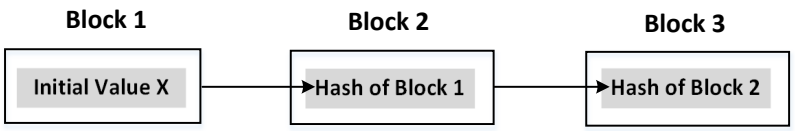


Figure 22.5: Hash chain

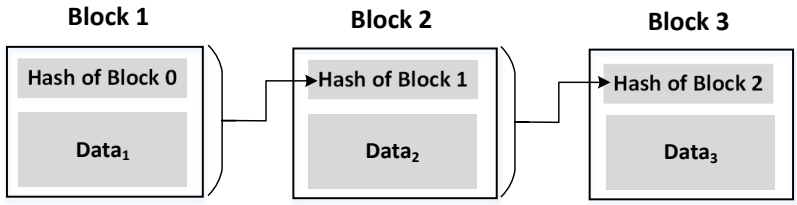


Figure 22.6: Blockchain

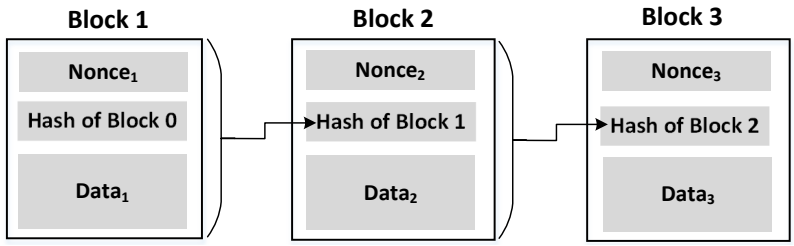


Figure 22.7: Blockchain: a nonce is added to each block

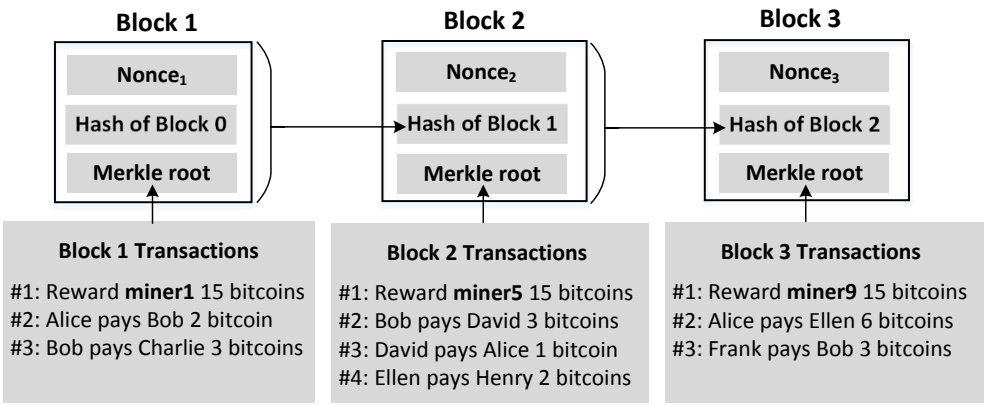


Figure 22.8: Bitcoin blockchain

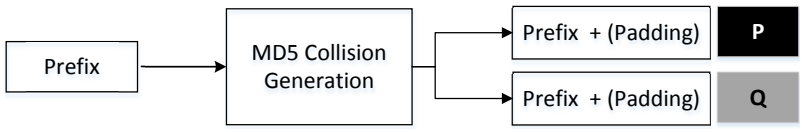


Figure 22.9: MD5 collision generation from a prefix

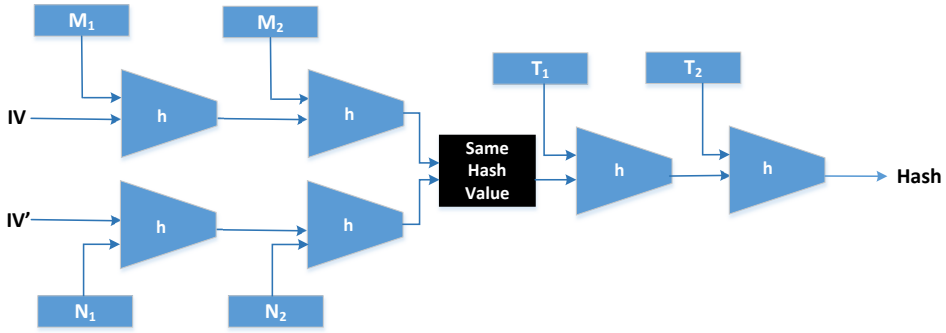


Figure 22.10: Generate more collision via length extension

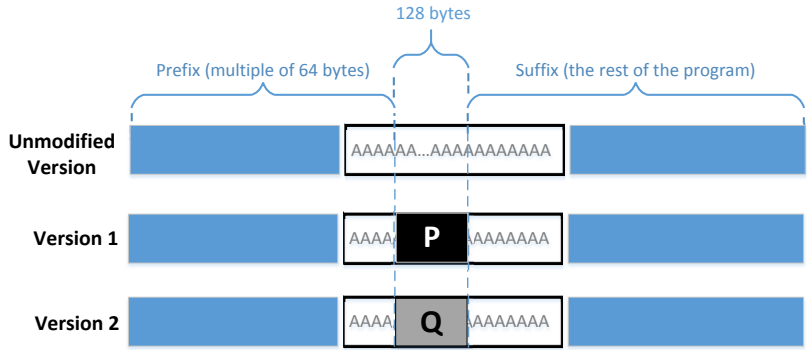


Figure 22.11: Break the executable file into three pieces, and create two versions of programs that have the same hash value.

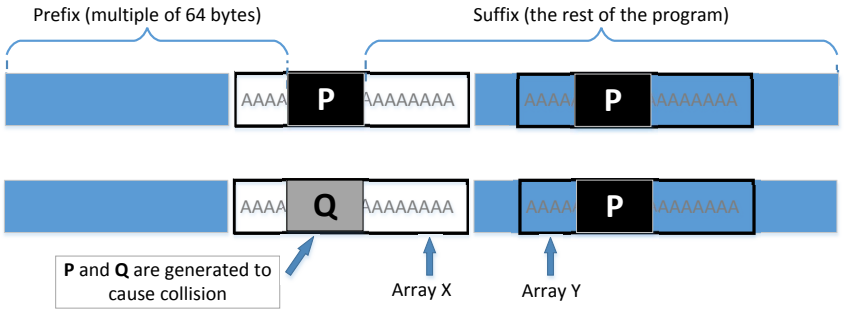


Figure 22.12: An approach to generate two hash-colliding programs with different behaviors.

Chapter 23

Public Key Cryptography

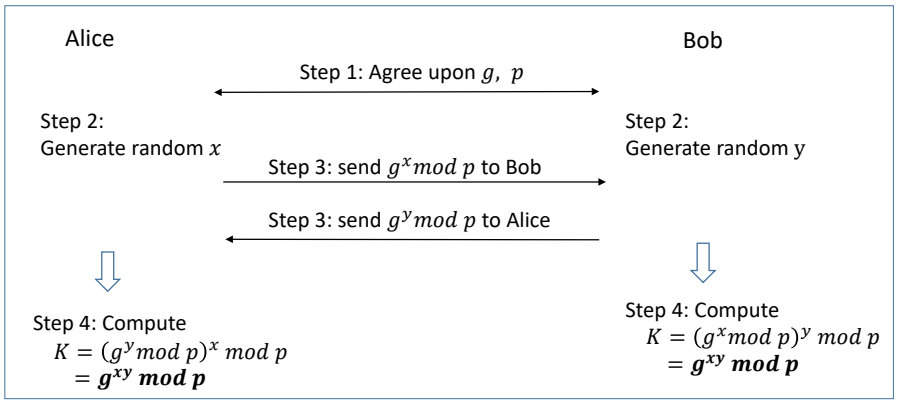


Figure 23.1: Diffie-Hellman key exchange protocol

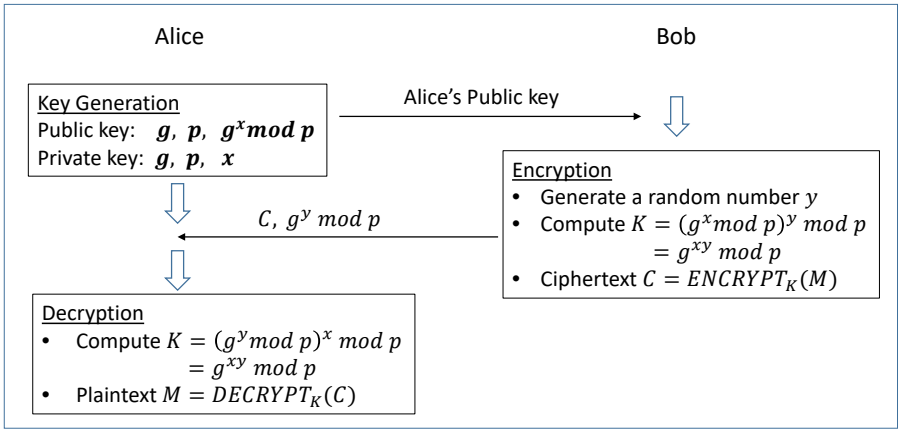


Figure 23.2: Turning DH key exchange protocol into public-key encryption scheme

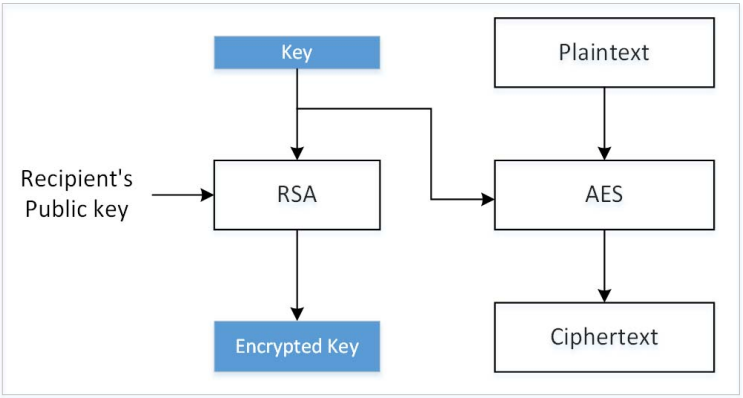


Figure 23.3: Hybrid encryption

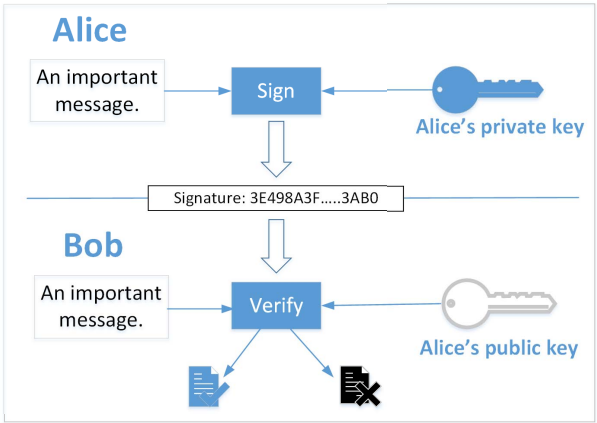


Figure 23.4: Digital signature

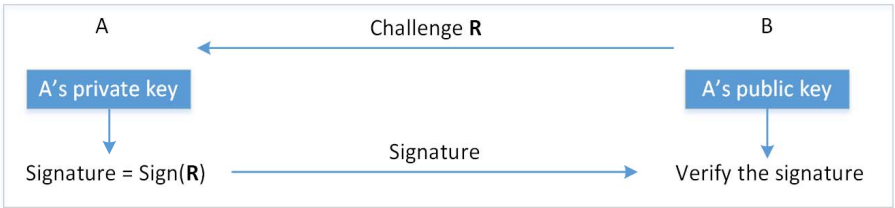


Figure 23.5: Public-key based authentication

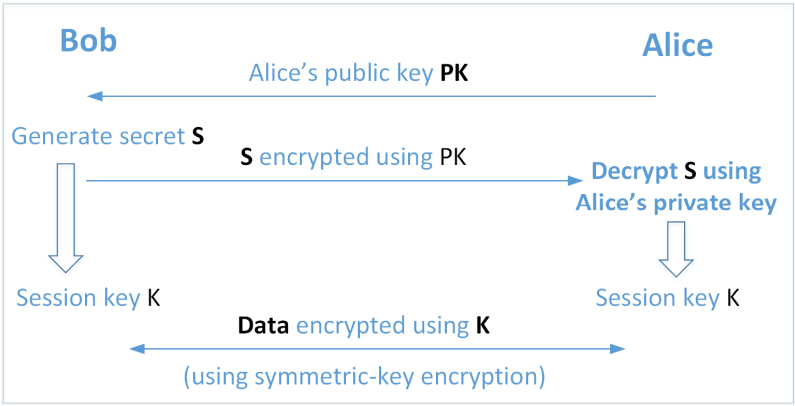


Figure 23.6: TLS/SSL Protocol

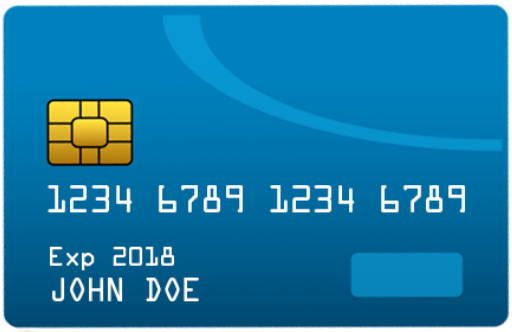


Figure 23.7: Chip card (add some illustration, point out the chip)

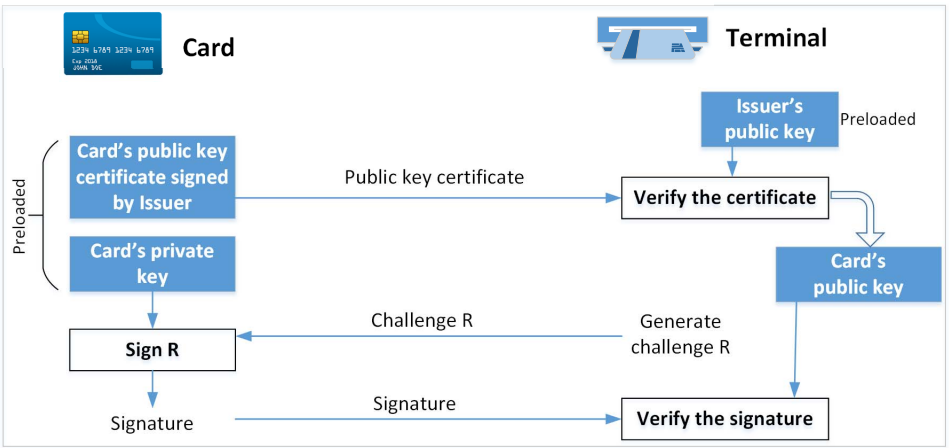


Figure 23.8: How a terminal (reader) authenticates cards

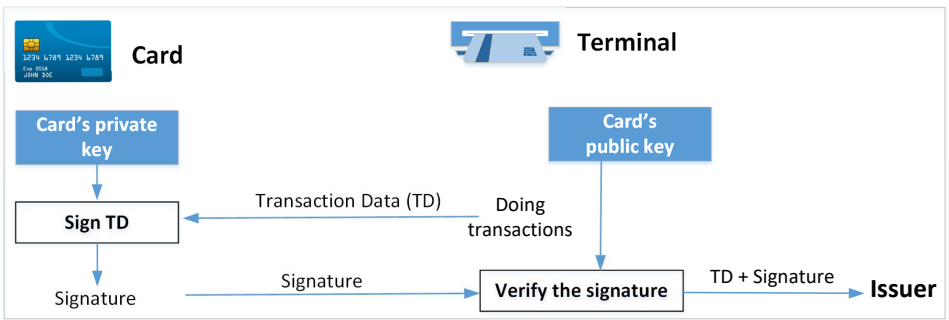


Figure 23.9: How a transaction is authenticated

Chapter 24

Public Key Infrastructure

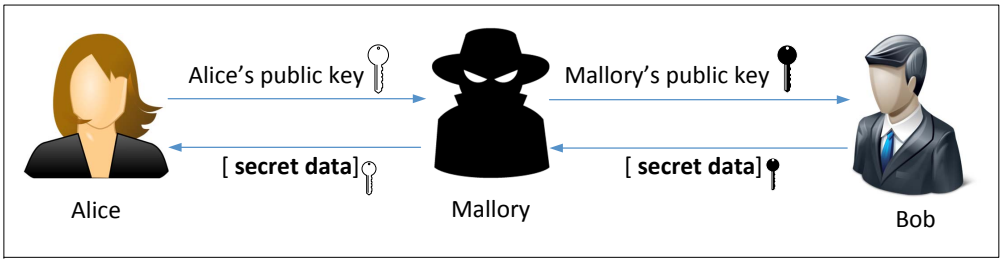


Figure 24.1: The Man-in-the-Middle Attack (MITM)

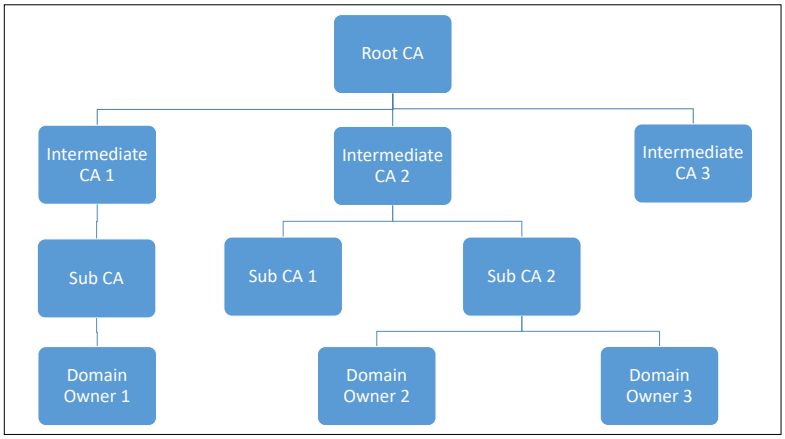


Figure 24.2: Hierarchy Of Certificate Authorities

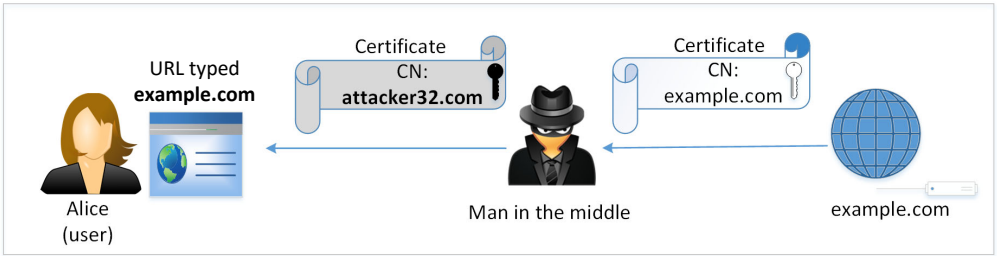


Figure 24.3: Defeating the MITM attack with PKI

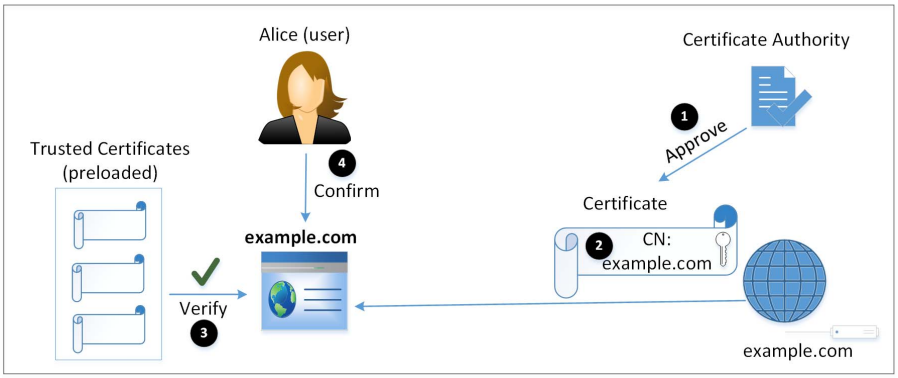


Figure 24.4: Security analysis of PKI

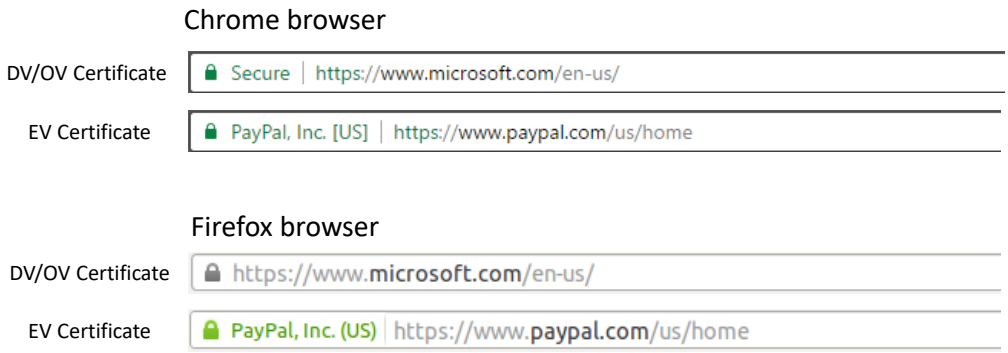


Figure 24.5: How browsers display the certificate type

Chapter 25

Transport Layer Security

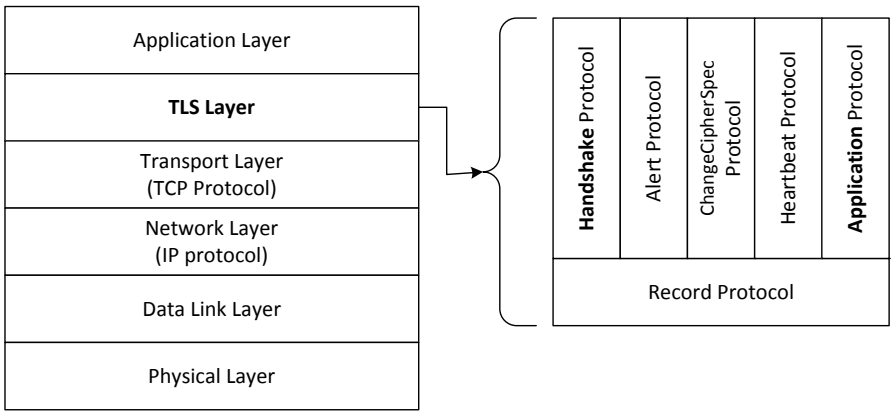


Figure 25.1: TCP/IP network stack with the TLS layer

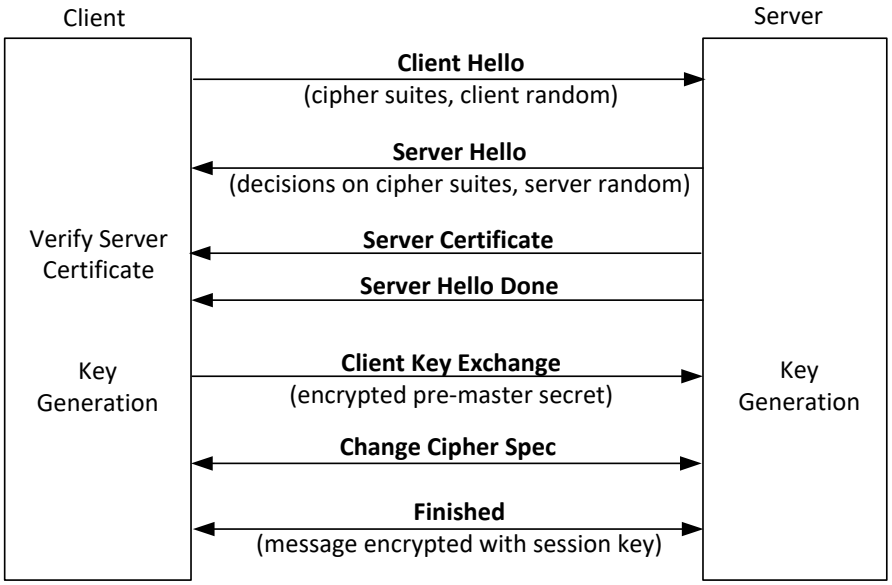


Figure 25.2: TLS handshake protocol

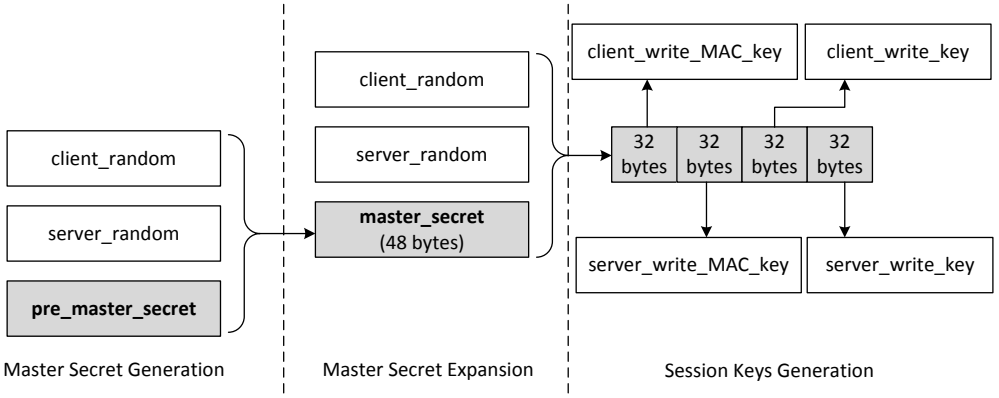


Figure 25.3: TLS key generation (master Secret and session keys)

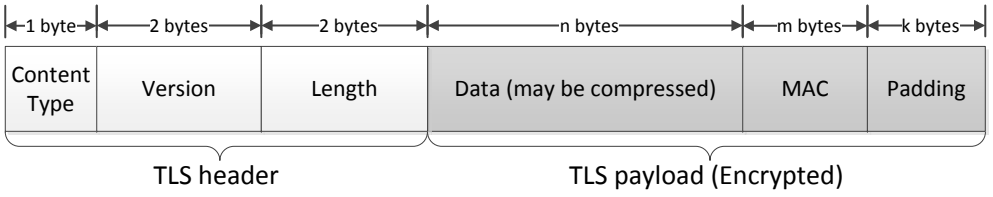


Figure 25.4: Record format of the TLS Record Protocol

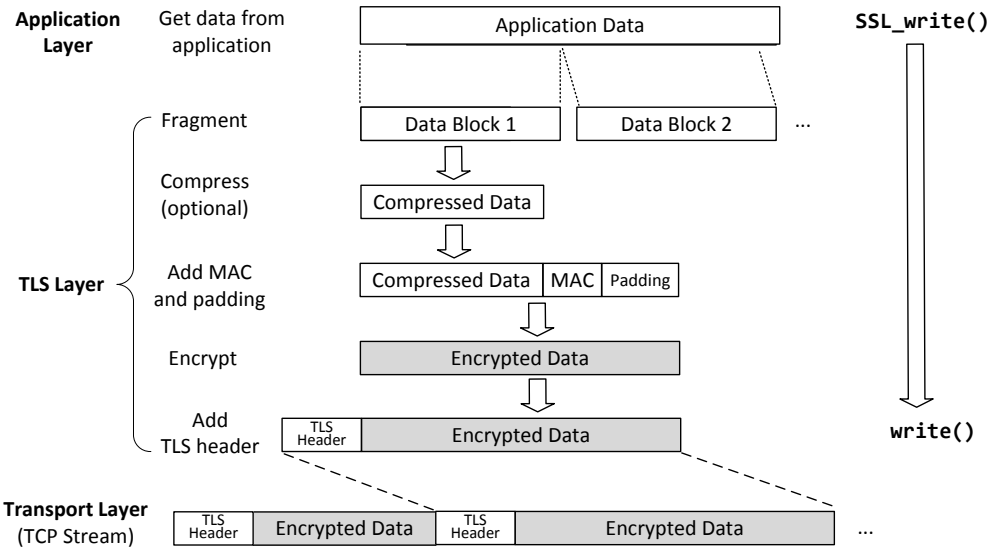


Figure 25.5: Sending data with TLS record protocol

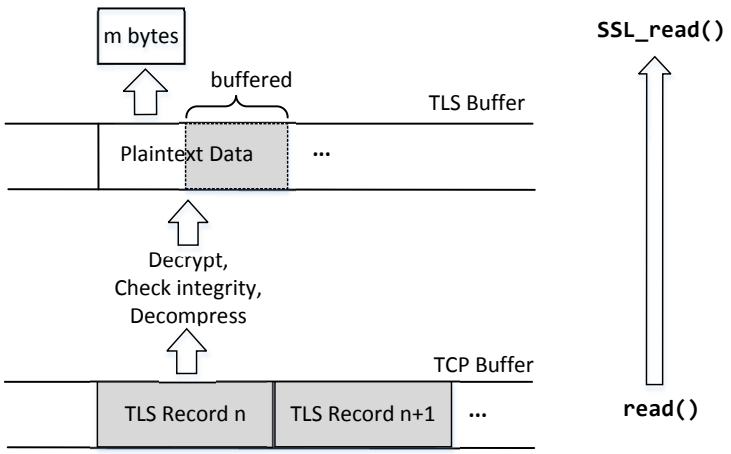


Figure 25.6: Reading data with TLS record protocol

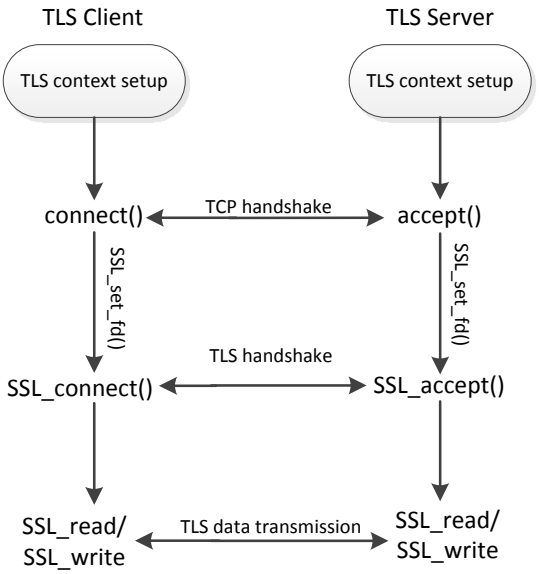


Figure 25.7: TLS programming overview

Chapter 26

Bitcoin and Blockchain

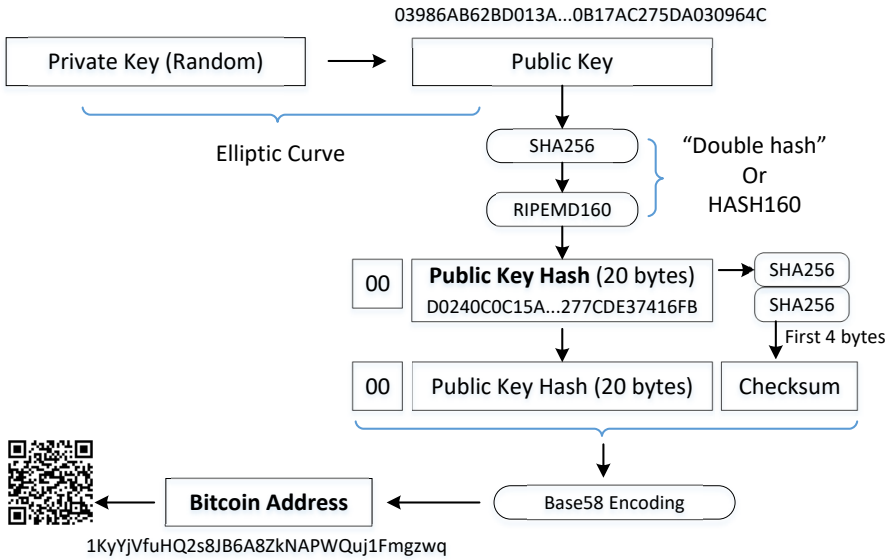


Figure 26.1: How Bitcoin address is generated.

Figure 26.2: QR code for bitcoin address: `1ETFfxDNaF8rWsuorMKhdHruxSuT9BDUGE`

Figure 26.3: Analogy

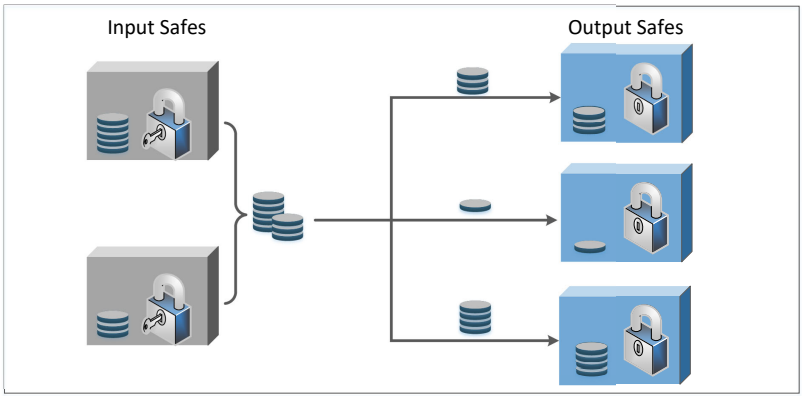


Figure 26.4: Transaction with multiple inputs and outputs

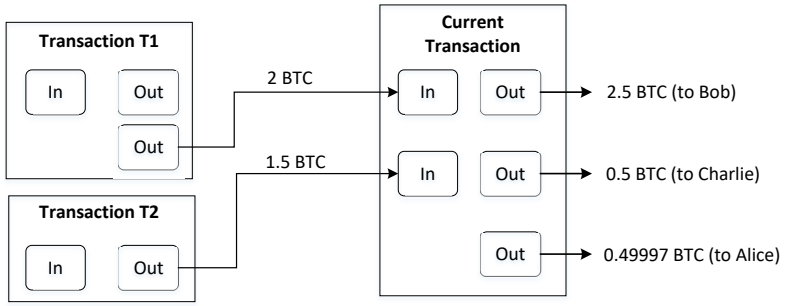


Figure 26.5: An example of transaction

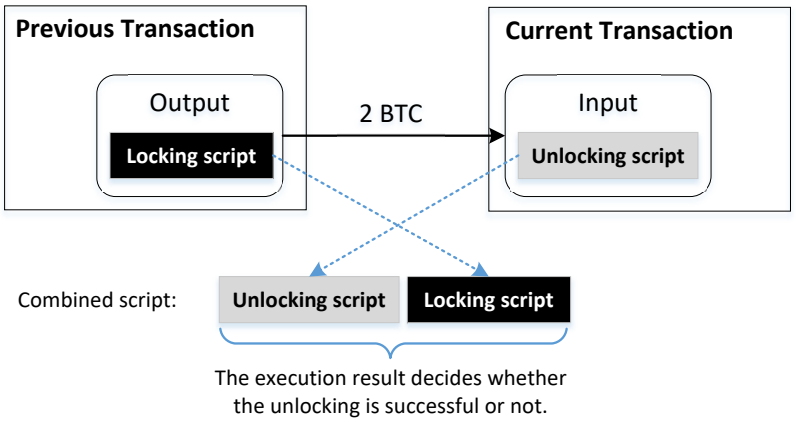


Figure 26.6: Unlocking and locking scripts

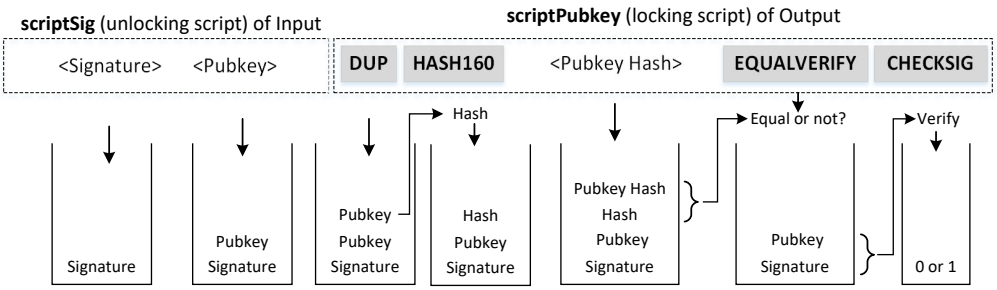


Figure 26.7: Pay-to-Pubkey-Hash script

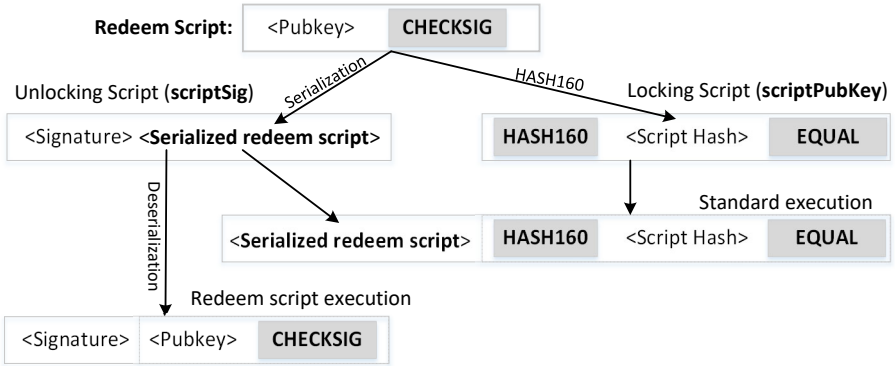


Figure 26.8: Pay-to-ScriptHash (P2SH)

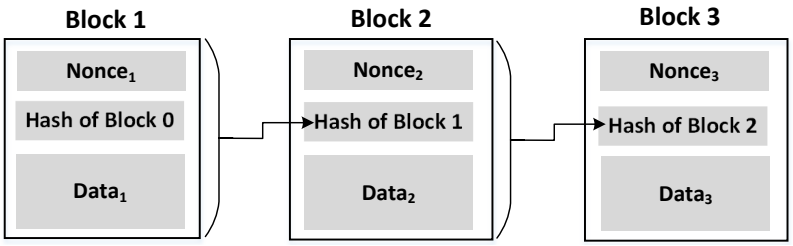


Figure 26.9: Blockchain: how blocks are chained together

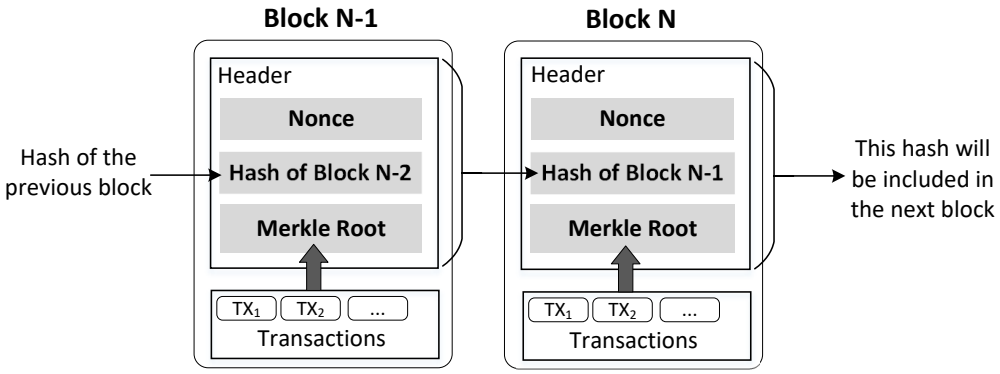


Figure 26.10: Blocks

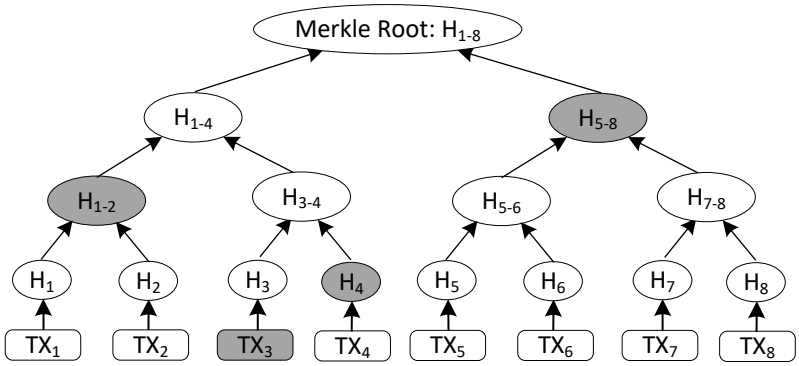


Figure 26.11: An example of Merkle tree

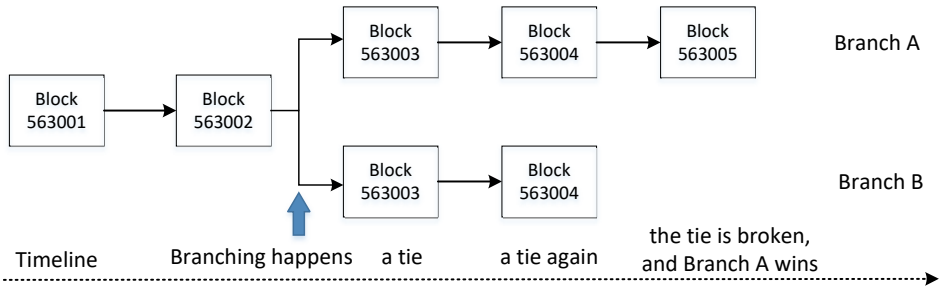


Figure 26.12: Branching

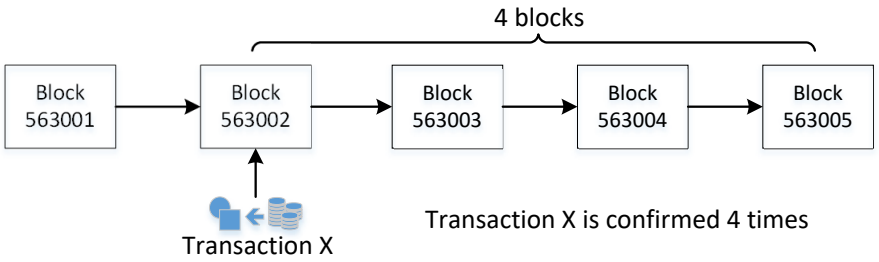


Figure 26.13: Confirmation number of transactions

Confirmation	2%	8%	10%	20%	30%	40%	50%
1	4%	16%	20%	40%	60%	80%	100%
2	0.237%	3.635%	5.600%	20.800%	43.200%	70.400%	100%
3	0.016%	0.905%	1.712%	11.584%	32.616%	63.488%	100%
4	0.001%	0.235%	0.546%	6.669%	25.207%	57.958%	100%
5	≈ 0	0.063%	0.178%	3.916%	19.762%	53.314%	100%
6	≈ 0	0.017%	0.059%	2.331%	15.645%	49.300%	100%
7	≈ 0	0.005%	0.020%	1.401%	12.475%	45.769%	100%
8	≈ 0	0.001%	0.007%	0.848%	10.003%	42.621%	100%

Figure 26.14: Probability of successful double spending

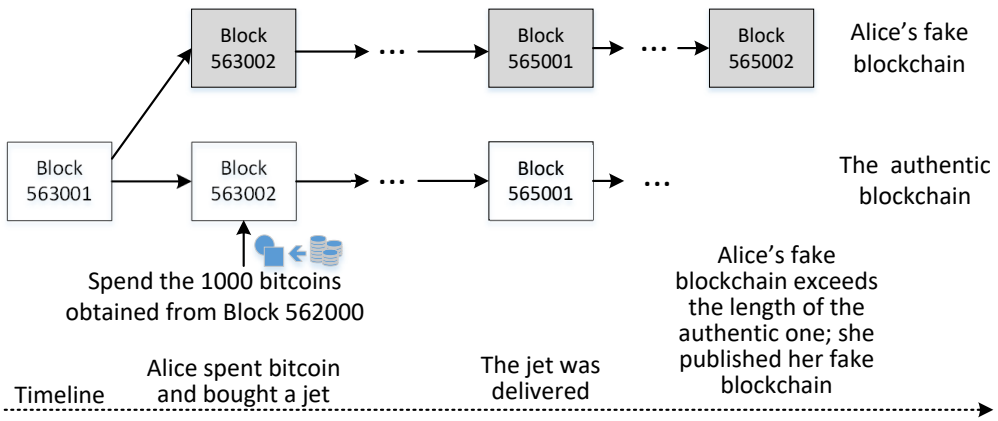


Figure 26.15: Double spending with a majority of hash power