

INDEX

Page numbers in italics refer to figures and tables.

Symbols

- ... (ellipsis notation) in Go, 107
- := operators in Go, 99
- _ (underscore) in Go, 100

A

- Accept and Install modal on Google
 - Chrome phishing site, 271, 272–273, 273, 281, 281
- access_addr variable, 166–167
- access control lists (ACLs), 166–167
- account lockouts, 69, 70–71, 87, 91–92, 95
- accounts.google.com login page
 - cloning with ChatGPT, 42–43, 43
 - cloning with wget, 39, 39
- ACLs (access control lists), 166–167
- Active Directory target network, 182, 182–183, 183
- addEventListener() function, 13
- AES-GCM encryption
 - building decryption tool, 116–120
 - building encryption tool, 110–116
 - writing custom ransomware, 122
- aesgcm.Open() function, 118
- aesgcm.Seal() function, 113
- AI. *See* artificial intelligence
- Amazon Machine Image (AMI), Ubuntu, 165
- Amazon Web Services (AWS), 158.
 - See also* EC2.
 - access keys, 163
 - for attack infrastructure, 159, 159–160, 160
 - CloudFormation, 173
 - Command Line Interface, 163–164
 - creating service account, 161, 161–164, 162, 163
 - hardware options, 168
 - key ID, 163
 - Lambda
 - creating redirector in Python, 260–262
 - defined, 172
 - deploying redirector to, 262
 - routing, 261
 - permission policies, 161–162, 162, 163
 - profiles, 164
 - regions, 165
 - Serverless Framework and, 172–173, 175
 - user accounts, 161, 161–164, 162, 163
- anchor (<a>) tag in basic phishing page, 9
- append() function, 113
- argparse library, 93
- args variable, 143
- Armitage, 286–289
 - Event Log feature, 289, 289
- artificial intelligence (AI)
 - malware development with, 147–153, 150
 - as phishing tool, 41–44, 43
- assumed breaches, 254
- atomic file writing, 123–124
- attempts parameter, 94
- attempts variable, 70, 71, 80
- authenticate_smb() function, 74–78, 80, 81
- authentication attacks, 49–50, 95.
 - See also* brute-force attacks
 - creating target application for, 50
 - adding authentication with PHP, 50–51
 - best practices for credential storage, 54–55

- authentication attacks (*continued*)
 - creating target application
 - for (*continued*)
 - implementing login response, 52–54
 - preventing SQL injection, 51–52
 - dissecting requests
 - building proxy server, 60–63, 62, 63
 - handling proxied requests with Python, 57–59
 - identifying data with web developer tools, 55–56, 56
 - intercepting requests with web proxy, 56–57
 - sending transparent proxy response, 59–60
 - password-spraying
 - SMB, 80–83
 - web applications, 70–73
 - tools for, 83
 - Burp Suite, 83–88, 84, 85, 86, 87
 - CrackMapExec, 88–92, 95
 - Spray Wrapper, 92–95
- authentication with PHP, 50–51
- autoincrement setting, primary keys, 26
- automating infrastructure deployment, 157–158. *See also* Serverless Framework
 - AWS for attack infrastructure, 159, 159–160, 160
 - creating AWS service account, 161, 161–164, 162, 163
 - Terraform, 164
 - deployment, 169–172
 - EC2 instance creation, 167–169
 - network access control lists, 166–167
 - server instantiation, 164–166
 - tools and technologies for, 158
- AWS. *See* Amazon Web Services
- aws_security_group resource group, 167

B

- baseURL variable, 141–142
- bind connections (bind shells), 185, 185, 186–188
- bind_param() function, 29, 51–52
- bind_result() function, 52
- blue teams, xxi
- Bootstrap framework, 5–6, 7
- Borges, Dan, 219
- Botocore library, 262
- browsers. *See also* Mozilla Firefox
 - configuring to use new proxy, 62
 - defining multiple proxy endpoints with extension, 62–63, 63
 - inspecting web requests with developer tools, 55–56, 56
- brute-force attacks
 - brute-force algorithm, 67
 - with Burp Suite, 83–88
 - with CME, 88–92
 - vs. password spraying, 69–70
 - targeting SMB, 73–74
 - automating attack, 77–79
 - implementing connection in Python, 74–75
 - running attack, 79–80
 - validating successful authentication to shares, 76–77
 - targeting web application, 64
 - automating attack, 67–69
 - building login request script, 64
 - running attack, 69
 - sending authentication data with Python, 65–67
- Burp Suite, 83–88, 84, 85, 86, 87
- button object, adding to basic phishing page, 9
- byte slice, 100, 102

C

- C2. *See* command and control callback
 - setting up exploit, 230–231
 - with Sliver C2, 293–295
- captive-portal attacks, 44–46, 45, 46

- Cascading Style Sheets (CSS), 5–6
- catch_all() function, 261
- cat command, 120
- CDNs (content delivery networks), 6–8, 8
- ChatGPT
 - AI malware development with, 147–153
 - cloning websites with, 41–44, 43
- cipher.NewGCM() function, 113
- ciphertext variable, 117
- cleartext credentials
 - dumping with CME, 90
 - in Salt Project, 213
 - storage of, 54
- CLI (Command Line Interface)
 - application, 163–164
- client/server architecture, 184, 184
- cloning websites
 - with ChatGPT, 41–44, 43
 - with goclone tool, 40–41, 41
 - with Gophish, 270, 270–273, 271, 272, 273
 - with wget tool, 37–40, 38, 39, 40
- CloudFormation, 173
- cloud providers, choosing, 159
- cmd.CombinedOutput() function, 108
- cmd.run argument, 212
- cmd variable, 108
- CME (CrackMapExec), 88–92, 95
- Cobalt Strike, teamserver for, 286–289
- CombinedOutput() function, 108
- command and control (C2), 190
 - applying network fundamentals to
 - Active Directory target
 - network, 182, 182–183, 183
 - bind connections, 185, 185
 - exploitation with Metasploit, 186–189
 - implications of connection
 - directionality, 189–190
 - network flows, 184, 184–185
 - OSI model, 180
 - reverse connections, 185–186, 186
 - simple target network, 180, 180–182, 181
 - defined, xx
 - multiplayer configuration, 285–286, 295
 - other tool kits for, 295
 - Sliver C2, 289–295
 - teamserver for Cobalt Strike and Armitage, 286–289
 - phishing attacks with redirectors, 255–256. *See also* Gophish
 - configuring custom server and implant, 263–264
 - implementing via Serverless Framework, 259–262
 - preparing custom implant and server, 256–259
 - in red team operations, xxii–xxiii
 - separating from phishing
 - infrastructure, 264
- servers
 - configuring custom, 263–264
 - configuring for red team operations, xxii–xxiii
 - configuring with Terraform, 226–232, 229
 - preparing custom, 256–259
 - Salt Project as, 219
 - using Empire on, 246–250, 247
 - writing in Python, 133–139, 139
 - writing implants, 139–147, 146, 147
- Command Line Interface (CLI)
 - application, 163–164
- commands
 - execution with Go os library, 106–110
 - running Salt Project, 212
- command variable, 107, 143
- configuration strings for web proxy, 57
- connection object, MySQL, 27–28
- connections, network
 - bind, 185, 185, 186–188
 - command-and-control network
 - flows, 184, 184–185
 - implications of directionality, 189–190
 - reverse, 185–186, 186, 188–189
- connect() method, 76
- console, xxii
- console.log() function, 13
- container class, 7

- content delivery networks (CDNs), 6–8, 8
- Content-Length header, 59
- content variable, 102
- continue-on-success flag, 89, 90, 95
- count variable, 70, 71, 80
- Covenant framework, 295
- CrackMapExec (CME), 88–92, 95
- credentials. *See also* authentication
 - attacks; web application exploits
 - AWS user, 163–164
 - brute-forcing, 64
 - automating attack, 67–69
 - building login request script, 64
 - running attack, 69
 - sending authentication data
 - with Python, 65–67
 - capturing server-side with PHP, 16–20, 20
 - capturing with JavaScript, 32–37
 - creating form for harvesting, 11–13
 - creating for MySQL database, 24–25
 - creating MySQL database for storing, 24–28
 - extracting with Mimikatz, 239–241
 - handling with JavaScript, 13–16, 15, 16
 - phishing website with database storage of, 28–32
 - saving without database, 20–24
 - storage of, 54–55
 - creds_all command, 240
 - cryptography, 110
 - CSS (Cascading Style Sheets), 5–6
 - curl command, 177, 264, 290
 - custom malware. *See* malware
 - development and distribution
 - custom web proxy. *See* web proxies
- MySQL, creating for credential storage, 24–28
 - saving credentials without, 20–24
- data type casting call in Go, 100
- data variable, 141–142
- \$dbPassword variable, 52
- \$dbUsername variable, 52
- DecryptFile() function, 115, 117
- Decrypt() function, 121–123, 127, 128
- decryption, building tool for with Go, 116–120
- defer keyword, 142
- delay variable, 70–71, 80
- desired state
 - Salt Project, 213
 - Terraform, 164
- DevTools (web developer tools), identifying data with, 55–56, 56
- disk-based implants, 234
- Document Object Model (DOM), 8, 11, 11
- doDecrypt() function, 124, 125–132
- doEncrypt() function, 124, 125–132
- Domain Name System (DNS), xxiii, 198–199, 199
- Download Chrome button, adding malicious link to, 271–272, 272
- Dropboxes, 243–244, 254
 - defined, 243
 - deploying
 - configuring reverse VPN, 245–246
 - post-exploitation via Python Impacket, 250–254
 - using Empire on C2 server, 246–250, 247
 - Vagrant, simulating with, 195–196

D

- databases
 - building phishing site with
 - credential storage in, 28–32
 - connecting users directly to, 180, 180–181

E

- EAPHammer captive-portal attacks, 44–46, 45, 46
- EC2 (Elastic Compute Cloud)
 - defined, 159
 - generating SSH key pair for, 159, 159–160, 160

- Terraform instance, 167–169, 193–196
- echo command, 18, 20
- ellipsis notation (...) in Go, 107
- Elliptic Curves, 199–200
- email, phishing
 - configuring SMTP sending profile, 276, 276–278, 277, 278
 - creating template with Gophish, 273–276, 274, 275, 276
 - executing campaign, 278–279, 279, 280
 - in mass phishing, 269
 - provider considerations, 264
 - spacing out, 269
 - in spear phishing, 269–270
- Empire. *See* PowerShell Empire C2 framework
- encodeURIComponent() function, 33
- EncryptFile() function, 111–113, 115
- Encrypt() function, 121–123, 126, 127
- encryption
 - building tool for with Go, 110–116
 - OpenVPN, setting strength of, 199–200
- end_headers() method, 59
- ephemeral ports, 190
- error handling
 - AWS Lambda redirector, 261
 - in Go, 99–100, 103, 104–105
 - for web proxy, 60
- err variable, 112
- EternalBlue vulnerability (MS17-010), 187, 225. *See also* Metasploit framework
- event listener, creating with JavaScript, 13
- events field, Serverless IaC
 - configuration, 175
- exec.Command() function, 108
- execute() function, 29
- execution loop of C2 implants, 139–140
- exploit -j command, 189

F

- Facebook
 - cloning with gocloner, 40–41, 41
 - cloning with wget, 39, 40

- \$fclose() function, 21
- fetch() function, 52
- fetch requests, 32–34
- file encryption with Go. *See under* Go language
- \$filename variable, 21
- fileName variable, 99, 104
- filepath.Walk() library function, 125
- file-reading functionality in Go, 98–101, 103–106
- \$file variable, 21
- file-writing functionality in Go, 102–106
- fingerprinting
 - configuring Salt Project, 210–211
 - defined, xx
- Firefox. *See* Mozilla Firefox
- firewalls, adding to network, 181, 181–182
- Flask API, Serverless, 173–174, 176
- fmt library, 98–99
- fmt.Println() function, 100, 104–105
- Font Awesome, 5–6, 7
- fopen() function, 21
- for loop
 - Go, 142
 - nested, in brute-force attacks, 67–68
 - in password-spraying tool, 71
- format string, 105
- \$formData string, 20–21
- forms, credential-harvesting, 11–13
- FoxyProxy extension, 62, 267, 267
- full-stack applications, building, 32
- function literals in Go, 125
- functions section, Serverless IaC
 - configuration, 175–176
- fuzzing, xx
- fwrite() function, 21

G

- gcm.NonceSize() function, 122
- gcm object, 122
- GenerateRandomBytes() function, 111, 112, 113
- generic/all payload listener, 230
- GetCmd() function, 258, 263

- GET requests
 - Go, 142–143
 - JavaScript, 32–35
 - Python, 134, 136, 148
 - get_user_input() function, 135–136
 - Gmail accounts, 264
 - GOARCH environment variable, 100
 - go build command, 100–101
 - gocloner tool, 40–41, 41
 - Go language, 97–98
 - C2 implant in, 139–147, 146, 147, 256–258
 - command execution with os
 - library, 106–110
 - configuring custom C2 server and implant, 263
 - file encryption with, 110
 - building decryption tool, 116–120
 - building encryption tool, 110–116
 - function literals, 125
 - ioutil library, 99
 - writing custom ransomware with, 120–121
 - defining doEncrypt() and doDecrypt() functions, 125–132
 - defining Encrypt() and Decrypt() functions, 121–123
 - running program, 132–133
 - scaffolding logic, 123–125
 - writing simple program in, 98
 - combining reading and writing programs, 103–106
 - setting up file-reading functionality, 98–101
 - setting up file-writing functionality, 102–103
 - Google Chrome download page
 - cloning, 270, 270–273, 271, 272, 273
 - phishing site based on, 280–281, 281
 - Google login page
 - cloning with ChatGPT, 42–43, 43
 - cloning with wget, 39, 39
 - Google Workspace SMTP services, 264
 - G00S environment variable, 100
 - Gophish
 - campaigns, 269–270
 - cloning landing page, 270, 270–273, 271, 272, 273
 - configuring SMTP sending profile, 276, 276–278, 277, 278
 - creating phishing email template, 273–276, 274, 275, 276
 - executing campaign, 278–279, 279, 280
 - phishing payload execution, 280, 280–282, 281, 282
 - receiving custom simple C2 callback, 282
 - chown utility, 266
 - configuring, 264–265
 - admin panel, 268, 268–269
 - permissions, 265–266
 - service account, 265
 - SSH proxy to admin panel, 266–267, 267
 - real-world scenarios, 283
 - in red team operations, xxiii
 - go run command, 101, 106
- ## H
- hacker_db MySQL database, 25–28, 55
 - hacking, xix–xx
 - handle_request() method, 58
 - handler field, 175
 - hardware virtual machine (HVM), 166
 - hashing passwords, 54–55
 - HCP Terraform, 172
 - help command, 291, 294–295
 - hexdump, 115
 - HTML (hypertext markup language)
 - building basic phishing page with, 5–7, 8–13
 - creating credential-harvesting form with, 11–13
 - handling credentials with JavaScript, 13–16
 - for remote credential harvesting via JavaScript, 35–37
 - tags, 6–9, 12

- HTTPConnection functions, 57
- http.Get() function, 142
- HTTP History tab, 83, 84, 84
- HTTP/HTTPS redirectors, 260
- HTTP proxy, custom. *See* web proxies
- HTTP server, Python, 35–37
- http.server.BaseHTTPRequestHandler class, 58
- HTTPServer object, 135
- HVM (hardware virtual machine), 166
- hypertext markup language. *See* HTML

I

- IaC (infrastructure as code), 157, 164, 181, 206. *See also* infrastructure automation; Salt Project
- id command, 108
- Identity and Access Management (IAM) privileges, 161
- id value in login form, 12
- if conditional in password-spraying tool, 71
- Impacket tool, 91, 250–254
- implants
 - AI malware development, 147–153, 150
 - C2
 - configuring custom server and, 263–264
 - preparing custom, 256–259
 - writing, 139–147, 146, 147
 - disk-based, 234
 - generating in Empire, 249–250
- index_route() function, 261
- info.IsDir() function, 125
- infrastructure as code (IaC), 157, 164, 181, 206. *See also* infrastructure automation; Salt Project
- infrastructure automation, 157–158, 178
 - AWS for attack infrastructure, 159, 159–160, 160
 - creating AWS service account, 161, 161–164, 162, 163
 - Serverless Framework, 172–173
 - boilerplate application, 173, 173–174, 174
 - deployment, 176–178
 - Flask API code, 176
 - Python deployment
 - dependencies, 174–176
 - Terraform, 164
 - deployment, 169–172
 - EC2 instance creation, 167–169
 - network access control lists, 166–167
 - server instantiation, 164–166
 - tools and technologies for, 158
- infrastructure engineering, 155–156. *See also* infrastructure automation; network fundamentals; reverse VPN tunneling
- infrastructure operations
 - evolution of, 206
 - red team, xxii–xxiii
- ingress rules, 167
- initialization vector (IV), 110, 111, 113, 115, 117
- inputFile variable, 111, 117
- input() function, 64
- instance_type hardware
 - configuration, 168
- Intruder feature, Burp Suite, 83, 84–87
- INVALID response, checking for in Burp Suite, 86, 86–87, 87
- ioutil.ReadFile() function, 99, 100, 104, 113, 117
- ioutil.WriteFile() function, 102–103
- IP address, configuring Salt Project, 210–211
- IPv6, blocking, 198
- IV (initialization vector), 110, 111, 113, 115, 117
- iv variable, 117

J

- JavaScript
 - credentials, 13–16, 15, 16, 32–37
 - importing libraries via CDNs, 6–8, 8

K

- key ID, 163
- key pairs, 159, 159–160, 160
- key variable, 111–112, 117

L

Lambda

- creating redirector in Python, 260–262
- defined, 172
- deploying redirector to, 262

landing page

- building basic, 5–10
- cloning with Gophish, 270, 270–273, 271, 272, 273

large language models (LLMs), 41–44, 43, 147–153

length parameter, 112

LHOST parameter, 189

links to multiple pages, adding, 9

listeners on Empire framework, 248

listShares() method, 76, 77

LLMs (large language models), 41–44, 43, 147–153

load kiwi command, 240

load mimikatz command, 240

Local Security Authority (LSA) Secrets registry, 90

lockouts, account, 69, 70–71, 91–92, 95

log.Fatal() function, 100, 102, 103, 104–105, 107

login form

- building for phishing page, 11–13
- creating for authentication attacks, 50–55
- handling credentials with JavaScript, 13–16

login request script, building, 64

log library, 99, 102

long listing format, 108–109

LSA (Local Security Authority) Secrets registry, 90

--lsa flag, 90

ls command, 108–109, 110, 146

Lupo C2 server, 139–140, 295

M

main() function, Go

- building decryption tool, 116–117
- building encryption tool, 111
- combining reading and writing programs, 104

command-execution program, 106–107, 108–109

file-reading utility, 99

writing C2 implant, 141

writing custom ransomware, 124, 132

main function, Python

brute-force attack for SMB targets, 77, 78

building login request script, 64

building proxy server, 60

implementing SMB connection, 75

SMB password-spraying tool, 80

writing C2 server, 134–135

main package, Go, 98

make() function, 112

malware development and distribution, 97–98, 153

with AI, 147–153, 150

command execution with os library, 106–110

file encryption with Go, 110

building decryption tool, 116–120

building encryption tool, 110–116

writing C2 implants, 139–147, 146, 147

writing C2 servers in Python, 133–139, 139

writing custom ransomware, 120–121

defining doEncrypt() and doDecrypt() functions, 125–132

defining Encrypt() and Decrypt() functions, 121–123

running program, 132–133

scaffolding logic, 123–125

writing simple program in Go, 98

combining reading and

writing programs, 103–106

setting up file-reading

functionality, 98–101

setting up file-writing

functionality, 102–103

man-in-the-middle attacks, 287

mark, defined, 269

- mass phishing, 269
- Metasploitable 3 virtual machine, 226
- Metasploit framework, 225–226, 241
 - acting on objective, 239–241
 - gaining access, 226–229, 229
 - exploiting target, 231–232
 - setting up exploit callback, 230–231
 - spinning up Metasploit on C2 server, 229, 229–230
- maintaining access, 233–234
 - cleanup automation resource, 238, 239
 - configuring persistence options, 236–238
 - disabling Windows Defender, 234–235
 - generating Meterpreter payload, 235–236
- network directionality and, 186–189
- payload options, 185, 186–187
- redirectors, 260
- Sliver C2 and, 289–290
- teamsrver and, 286–287, 289
- Mimikatz, extracting credentials with, 239–241
- most_recent attribute, 165
- Mozilla Firefox
 - configuring to use custom web proxy, 62, 62
 - defining multiple proxy endpoints with extension, 62–63, 63
 - inspecting web requests with web developer tools, 55–56, 56
- MS17-010 (EternalBlue vulnerability), 187, 225. *See also* Metasploit framework
- msfdb init command, 286–287
- mt-X class, Bootstrap, 7–8
- multi/handler module, 230
- multiplayer C2 configuration, 285–286, 295
 - Sliver C2, 289–290
 - configuring, 291–293
 - getting callback, 293–295
 - installing, 290–291

- teamsrver for Cobalt Strike and Armitage, 286–289
- tool kits for, 295
- MyRequestHandler() class, 136
- MySQL
 - building phishing site with database credential storage, 28–32
 - connection object, 27–28
 - database, creating for credential storage, 24–28
 - database table, 25–26

N

- name value in login form, 12
- navigation bar, adding to basic phishing page, 8–11, 11
- negative check in Burp Suite, 87, 87
- NetExec tool, 88
- netloc value, 58
- Net-NTLMv2 protocol, 75
- network access control lists, 166–167
- network fundamentals, 179, 190
 - Active Directory target network, 182, 182–183, 183
- bind connections, 185, 185, 186–188
- command-and-control network flows, 184, 184–185
- exploitation with Metasploit framework, 186–189
- implications of connection directionality, 189–190
- OSI model, 180
- reverse connections, 185–186, 186, 188–189
- simple target network, 180, 180–182, 181
- Networking for Beginners* (Scott), 179
- no-bruteforce flag, 88–89
- NoLogRequestHandler parameter, 136
- nonce, 110
- nonceSize variable, 122–123

O

- obfuscation, payload, 293
- offensive security, core concepts of, xix–xxi

- offensive security tools (OSTs). *See also*
 - brute-force attacks
 - for authentication attacks
 - Burp Suite, 83–88, 84, 85, 86, 87
 - CrackMapExec, 88–92
 - Spray Wrapper, 92–95
 - best practices, 4–5
 - development of in red teams, xxix–xxxi
 - in different offensive security disciplines, xxvii–xxix
 - ethical and professional use of, xxxi–xxxii
 - for phishing, 37
 - ChatGPT, 41–44, 43
 - EAPHammer, 44–46, 45, 46
 - goclone tool, 40–41, 41
 - wget tool, 37–40, 38, 39, 40
 - in red team operations, xxii–xxiii
 - OnSubmit event, 13
 - Open() 118
 - Open Systems Interconnection (OSI)
 - model, 180, 180–182, 181
 - OpenVPN
 - configuring with PiVPN, 197, 197–200, 198, 199
 - establishing reverse VPN tunnel, 202–203
 - installing client on VM, 196
 - options command, 248–249
 - org value, 175
 - OSI (Open Systems Interconnection)
 - model, 180, 180–182, 181
 - os library
 - command execution with, 106–110
 - importing, 103–104
 - os.Remove() function, 104
 - os.Rename() function, 125
 - OSTs. *See* offensive security tools
 - outputFile variable, 111, 117
 - owners attribute, 165
- P**
- parameter binding, 28–29
 - password input object in login form, 12
 - password parameter
 - editing in Burp Suite, 85, 85
 - Python SMBConnection object, 75
 - passwords. *See also*
 - authentication attacks
 - in brute-force algorithm, 67
 - predefined default, 252
 - typical patterns in, 49–50
 - password spraying
 - vs. brute-force attacks, 69–70
 - with Burp Suite, 88
 - with CME, 92
 - defined, xx
 - software custom built for, xxx–xxxi
 - with Spray Wrapper, 92–95
 - targeting SMB, 80–83
 - targeting web application, 70–73
 - password variable, creating with PHP, 17
 - payloads
 - Metasploit, 230–231
 - in Sliver C2, 293
 - penetration testing, xx, xxviii–xxix
 - permission bits, 102–103
 - persistence
 - establishing via Empire, 252–254
 - establishing with Metasploit, 233–234
 - cleanup automation
 - resource, 239
 - configuring options, 236–238
 - disabling Windows Defender, 234–235
 - generating Meterpreter payload, 235–236
 - phishing, 46–47
 - building basic page, 5
 - adding navigation bar, 8–11, 11
 - importing JavaScript libraries
 - via CDNs, 6–8, 8
 - using Bootstrap and Font Awesome, 5–6
 - building website with database
 - credential storage, 28–32
 - with C2 redirectors, 255–256. *See also* Gophish
 - configuring custom server and implant, 263–264

- implementing via Serverless Framework, 259–262
 - preparing custom implant and server, 256–259
 - real-world operations, 283
- capturing credentials server-side
 - with PHP, 16–20, 20
- capturing credentials with JavaScript, 32–37
- creating credential-harvesting form, 11–13
- creating MySQL database for credential storage, 24–28
- defined, xxi
- handling credentials with JavaScript, 13–16, 15, 16
- mass, 269
- saving credentials without database, 20–24
- separating form C2 infrastructure, 264
- spear, 269–270
- tools for
 - ChatGPT, 41–44, 43
 - EAPHammer captive-portal attacks, 44–46, 45, 46
 - goclone, 40–41, 41
 - wget, 37–40, 38, 39, 40
- PHP
 - authentication with, 50–51
 - building phishing website with database credential storage, 28–32
 - capturing credentials server-side with, 16–20, 20
 - reconfiguring application to connect to MySQL database, 27–28
 - saving credentials without database, 20–24
- pinging VPN servers, 202
- pivpn -c command, 202, 246
- pivpn command, 200–201
- PiVPN project
 - configuring OpenVPN with, 197, 197–200, 198, 199
 - generating VPN certificate in, 200–201
- plaintext value, 113
- platform security engineers, 206
- port forwarding, 184–185
- ports
 - bind connections, 185
 - ephemeral, 190
 - privilege level and use of, 188
 - Salt Project, 207
- post-exploitation via Python Impacket, 250–254
- POST requests
 - capturing credentials server-side with PHP, 17, 20
 - inspecting with browser developer tools, 56
 - JavaScript, 33, 35
 - PHP, 51
 - sending authentication data with Python, 65
 - viewing in Burp Suite, 84, 84, 85, 85
- PowerShell Empire C2 framework
 - deploying stager, 252
 - establishing persistence via, 252–254
 - Starkiller project, 295
 - using on C2 server, 246–250, 247
- practical red teaming scenarios, 223. *See also* real-world offensive security scenarios
- prepared statement, creating, 28–29
- prepare() function, 29
- prepend migrate option, 233
- preventDefault() method, 13
- primary keys, 25–26
- principle of least privilege, 161, 265
- programming languages, red teaming with different, xxiv
- provider value, 175
- provisioner statement, 228
- proxies, SSH, 266–267, 267. *See also* web proxies
- ProxyHandler class, 58
- Python
 - AI malware development with, 147–153
 - boilerplate Serverless application, 173–174

Python (*continued*)

- building custom web proxy with, 56–62
- building script to brute-force credentials with, 64–69
- creating AWS Lambda redirectors in, 260–262
- creating brute-forcing attack tool targeting SMB, 74–80
- creating password-spraying tools with, 70–73, 80–83
- Go vs., 97–98
- HTTP server, 35–37
- Impacket, post-exploitation via, 250–254
- libraries
 - argparse, 93
 - Botocore, 262
 - http.client, 57
 - http.server, 57, 133
 - logging, 133
 - re (regular expression), 64, 65–66
 - requests, 64, 65
 - SMBConnection, 74, 75, 76, 80
 - socketserver, 57
 - threading, 133
 - time, 69–70, 80
 - urllib.parse, 57, 133, 136
- sending authentication data with, 65–67
- Serverless deployment
 - dependencies, 174–176
- Serverless Flask API code, 176
- simple C2 server in, 258–259
- SimpleHTTPServer, 250
- Spray Wrapper tool, 92–95
- writing C2 servers in, 133–139, 139

R

- rand.Read() function, 112, 122
- ransomware, writing custom, 120–121
 - defining doEncrypt() and doDecrypt() functions, 125–132
 - defining Encrypt() and Decrypt() functions, 121–123

- running program, 132–133
- scaffolding logic, 123–125
- Rapid7 Metasploitable 3 VM, 226
- Raspbian operating system, 197
- raw request, inspecting with browser developer tools, 56
- RDP (Remote Desktop Protocol), 190
- read–evaluate–print loop (REPL), 135, 139
- real-world offensive security scenarios, 221–223, 283
- redirect() function, 261
- redirectors, phishing attacks with C2, 255–256. *See also* Gophish
 - configuring custom server and implant, 256–259, 263–264
 - implementing via Serverless Framework, 259–262
- red teaming
 - defined, xxi, xxviii
 - ethical implications of, xxi–xxii
 - vs. other offensive security disciplines, xxvii–xxix
 - practical scenarios for, 223
 - red team operations, xxii–xxiii
 - tool development in, xxix–xxxi
- Remote Desktop Protocol (RDP), 190
- remote-exec action, 228
- REPL (read–evaluate–print loop), 135, 139
- requests, dissecting
 - building proxy server, 60–63, 62, 63
 - handling proxied requests with Python, 57–59
 - identifying data with web developer tools, 55–56, 56
 - intercepting requests with web proxy, 56–57
 - sending transparent proxy response, 59–60
- reqURL variable, 142
- reverse connections (reverse shells), 185–186, 186, 188–189
- reverse TCP payload, 230–231
- reverse VPN tunneling, 191, 203
 - configuring EC2 instance with Terraform, 193–196

- configuring OpenVPN with
 - PiVPN, 197, 197–200, 198, 199
 - configuring reverse VPN dropbox, 245–246
 - establishing reverse VPN tunnel, 202–203
 - generating VPN certificate, 200–201
 - reverse VPN topology, 191–193, 192, 193
 - RHOST parameter, 187
 - Rivest-Shamir-Adleman (RSA)
 - algorithm, 199–200
 - routes in Serverless Framework, 176
 - row class, 7
 - RSA (Rivest-Shamir-Adleman)
 - algorithm, 199–200
 - runCommand() function, 106–108, 109, 141, 143–144, 152
- S**
- Salt Project, 205, 206–207
 - as C2 server, 219
 - configuring fingerprint and IP address, 210–211
 - configuring Terraform server, 207–209
 - desired state, 213
 - documentation, 209
 - filtering, 212
 - installing, 209
 - linking minions to master, 211–212
 - pillars, 213
 - running commands, 212
 - Salt master, 207, 209–212
 - Salt minions, 207, 209–212
 - Salt states
 - creating, 213–214
 - observing desired, 214–218
 - working with, 212
 - Scott, Russell, 179
 - scripts from internet,
 - running, 197
 - Seal() function, 113, 118
 - searchmodule command, 253
 - Secure Shell (SSH)
 - C2 server communications and, 190
 - key pairs for EC2 server, 159, 159–160, 160
 - setting up proxy to Gophish admin panel, 266–267, 267
 - security best practices, 4
 - SELECT query, 51, 52
 - sending profiles, configuring with
 - Gophish, 276, 276–278, 277, 278
 - send_login_request() function, 64
 - SendResponse() function, 258, 263
 - separation of duties, 264
 - Serverless Framework, 172–173
 - boilerplate application, 173, 173–174, 174
 - deployment, 176–178
 - Flask API code, 176
 - implementing C2 redirectors via, 259–262
 - permission policies for, 162
 - Python deployment dependencies, 174–176
 - Server Message Block (SMB) protocol
 - brute-forcing attack tool targeting, 73–74
 - automating attack, 77–79
 - implementing connection in Python, 74–75
 - running attack, 79–80
 - validating successful authentication to shares, 76–77
 - CME module for, 88–92
 - password-spraying tool targeting, 80–83
 - servers
 - accessing reverse VPN instance, 195
 - C2
 - configuring custom, 263–264
 - configuring for red team operations, xxii–xxiii
 - configuring with Terraform, 226–232, 229
 - Lupo, 139–140, 295
 - preparing custom, 256–259

- servers (*continued*)
 - C2 (*continued*)
 - Salt Project as, 219
 - using Empire on, 246–250, 247
 - writing in Python, 133–139, 139
 - compromising for reverse VPN
 - tunneling, 192, 192
 - DNS, 198
 - proxy, building, 60–63, 62, 63
 - SMTP, 264
 - Terraform
 - EC2 instance creation, 167–169
 - instantiation, 164–166
 - Salt Project, configuring, 207–209
 - web
 - capturing credentials with, 35–37
 - connecting users to, 180, 180–181
 - server-side PHP form, capturing credentials with, 16–23
 - service accounts
 - AWS, 161, 161–164, 162, 163
 - Gophish, creating, 265
 - setcap command, 266
 - shadow IT work, 270
 - shares, validating successful
 - authentication to SMB, 76–77
 - shares variable, 76, 77
 - shell, defined, xxi
 - shellwords library, 258
 - SimpleHTTPServer, 250
 - Simple Mail Transfer Protocol (SMTP)
 - sending profile, configuring with
 - Gophish, 276, 276–278, 277, 278
 - servers, access to, 264
 - simulation in red teaming, xxix–xxx
 - sleep duration, setting in Go, 144
 - sleep() method, 70–71
 - Sliver C2, 289–290
 - configuring multiplayer mode, 291–293
 - getting callback, 293–295
 - installing, 290–291
 - sls deploy command, 176–177
 - sls remove command, 178
 - SMB protocol. *See* Server Message Block protocol
 - SMTP. *See* Simple Mail Transfer Protocol
 - Sniper attack mode, 84
 - Socat, 260
 - social engineering, xxi
 - software development, 1–2
 - source command, 26
 - spear phishing, 269–270
 - spray() function, 94
 - Spraygen, 50
 - Spray Wrapper, 92–95
 - SQL. *See* MySQL; Structured Query Language
 - SSH. *See* Secure Shell
 - stagers, Empire, 249–250, 252
 - standard shell, defined, xxi
 - state.apply command, 214, 217–218
 - \$stmt variable, 29
 - strings.Fields() function, 143
 - strings library, 143
 - string() statement, 100
 - strip() method, 68, 78
 - Structured Query Language (SQL). *See also* MySQL
 - SQL injection, 28–29, 51–52
 - syntax in, 25
 - submit button in login form, 12–13
 - subprocess.checkoutput() function, Python, 150
 - sudo salt state.apply command, Salt Project, 214–216
 - system administrators, 206
- ## T
- tables, creating for MySQL database, 25–27
 - tactics, techniques, and procedures (TTPs), 222
 - tailgating, xxi
 - target_url object, 58
 - teamsrver, 286–289, 287, 288, 289
 - Terraform, 164
 - configuring C2 server with, 226–229
 - exploiting target, 231–232
 - setting up exploit callback, 230–231

- spinning up Metasploit on
 - C2 server, 229, 229–230
 - configuring EC2 instance with, 193–196
 - configuring Salt Project server, 207–209
 - data sources in, 165
 - defined, 164
 - deployment, 169–172
 - EC2 instance creation, 167–169
 - egress rules, 167
 - network access control lists, 166–167
 - vs. other IaC technologies, 206
 - server instantiation, 164–166
 - states, 164, 172
 - t2.micro hardware
 - configuration, 168
 - terraform apply command, 171, 195
 - terraform destroy command, 171–172
 - terraform init command, 169–170, 194
 - terraform plan command, 170–171
 - text file, saving credentials in, 21–24
 - threat emulation, 222. *See also*
 - real-world offensive security scenarios
 - security scenarios
 - time.Sleep() function, 144
 - TLS (Transport Layer Security), 57, 280
 - tmux, 229, 246, 266
 - tools, offensive security. *See* offensive security tools
 - topology
 - network, 181
 - reverse VPN, 191–193, 192, 193
 - top.sls file, 214
 - Transport Layer Security (TLS), 57, 280
 - TREVORspray, xxx–xxxi, 50, 92–95
 - try block, 68
 - try...catch block, 58–59, 65, 76, 78, 134–135
 - TTPs (tactics, techniques, and procedures), 222
 - type attributes in login form, 12–13

U

- Ubuntu Amazon Machine Image (AMI), 165
- unattended security patching, 200
- underscore (_) in Go, 100

- unit tests, 64
- URL parsing libraries for web proxy, 58
- url.QueryEscape() function, 142
- uselistener keyword, 248
- usemodule powershell_persistence
 - _elevated_schtasks
 - command, 253–254
- User-Agent string, 38
- user_input variable, 135, 137
- usernames. *See also*
 - authentication attacks
 - binding, 51
 - in brute-force algorithm, 67
 - login form input object for, 12
- username variable, creating with PHP, 17

V

- Vagrant, 193–194
 - configuring reverse VPN
 - dropbox, 245
 - establishing reverse VPN tunnel, 202–203
 - simulating dropbox with, 195–196
- vagrant ssh command, 196
- variadic function, 107
- VirtualBox configurations, 74
- virtual machines (VMs)
 - client certificate for, 201
 - configuring reverse VPN
 - dropbox, 245
 - establishing reverse VPN tunnel, 202–203
 - installing with Vagrant, 196
- virtual private networks (VPNs). *See also*
 - reverse VPN tunneling
 - Active Directory and, 182–183, 183
 - certificates, 200–201
 - configuring OpenVPN with
 - PiVPN, 197, 197–200, 198, 199
 - defined, xxi
 - Terraform, 164–165, 168–169
- vishing, xxi
- VMs. *See* virtual machines
- VPNs. *See* virtual private networks
- vulnerability management, tools used
 - in, xxviii, xxix

W

- web application exploits, 3–4, 46–47.
 - See also* authentication attacks
- building basic phishing page, 5
 - adding navigation bar, 8–11, 11
 - importing JavaScript libraries via CDNs, 6–8, 8
 - using Bootstrap and Font Awesome, 5–6
- building phishing site with database credential storage, 28–32
- capturing credentials server-side with PHP, 16–20, 20
- capturing credentials with JavaScript, 32–37
- creating credential-harvesting form, 11–13
- creating MySQL database for credential storage, 24–28
- handling credentials with JavaScript, 13–16, 15, 16
- OST best practices, 4–5
- saving credentials without database, 20–24
- tools for phishing, 37
 - ChatGPT, 41–44, 43
 - EAPHammer captive-portal attacks, 44–46, 45, 46
 - goclone, 40–41
 - wget, 37–40, 38, 39, 40
- web developer tools (DevTools), identifying data with, 55–56, 56
- web proxies
 - building proxy server, 56–63, 62, 63
 - configuring browser to use, 62, 62
 - defining multiple proxy endpoints, 62–63, 63
 - handling proxied requests with Python, 57–59
 - intercepting requests with, 56–57
 - sending transparent proxy response, 59–60
- web servers
 - capturing credentials with, 35–37
 - connecting users to, 180, 180–181
- web shell, xxi, 4
- website cloning
 - with ChatGPT, 41–44, 43
 - with goclone, 40–41, 41
 - with Gophish, 270, 270–273, 271, 272, 273
 - with wget, 37–40, 38, 39, 40
- wfile.write() function, 137
- wget, 37–40, 38, 39, 40, 265
- Windows
 - brute-forcing attack tool targeting SMB, 73–80
 - default lockout policy, 91–92
 - Defender, disabling, 234–235
 - password spraying with Spray Wrapper, 94–95
- windows/local/persistence module, 237–238
- with statement, 60
- WordPress login page, cloning with wget, 38, 38–39
- writeToFile() function, 123–125, 126, 128

X

- xxd command, 115