

Penetration Testing Student

System Attacks

Section 03 | Module 05

© Caendra Inc. 2019
All Rights Reserved

Table of Contents

MODULE 05 | System Attacks

5.1 Malware

5.2 Password Attacks

5.3 Buffer Overflows Attacks



Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ What malware is and the common types
- ✓ Methods of attacking passwords
- ✓ What a buffer overflow is and why it is a serious security risk



5.1

Malware



5.1 Malware

How does this support my pentesting career?

- Ability to use the right malware incarnation during an engagement
- Knowledgeable of how to use malware while keeping your test under the rules of engagement
- Ability to maintain access to a compromised machine

5.1 Malware

Malware, short for “**Malicious software**”, is any software used to misuse computer systems with the intent to:

- Cause denial of service
- Spy on users activity
- Get unauthorized control over one or more computer systems
- Cause other malicious activities

```
21 def _init__(self, context):
22     """Initialize the experiment with context"""
23     # Store context - the experiment will be run in this context
24     self.observations = [] # an array of Observations, in this case
25     # control - the control observation
26     self.control = None
27
28 def initialize(experiment, observations = [], control = None):
29     @experiment
30     @observations = observations
31     @control = control
32     @candidates = observations + [control]
33     evaluate_candidates
34
35 freeze
36
37 def context
38     experiment.context
39
40
41 # Initialize the name of the experiment
42
43 def experiment_name
44     experiment.name
45
46
47
48 # A function that will return a match between all
49 def match
50     """
51     Returns a result of 1 if
52     """
53     return result[0] == 1
```



5.1 Malware

In this chapter, you will see a complete representation of the various types of malware used by cybercriminals.

Moreover, you will learn how to use some malware incarnations during a **penetration test engagement**.



5.1 Malware

Malware classification is based on the behavior of the software, rather than the malicious features it provides.

- Virus
- Trojan Horses
- Rootkit
- Bootkit
- Backdoors
- Adware
- Spyware
- Greyware
- Dialer
- Key-logger
- Botnet
- Ransomware
- Data-Stealing Malware
- Worm



5.1.1 Virus

A **computer virus** is a small piece of code that spreads from computer to computer, without any direct action or authorization by the owners of the infected machines.

Viruses usually copy themselves to special sections of the hard disk, inside legitimate programs or documents. They then run every time an infected program or file is opened.



5.1.2 Trojan Horse

A **Trojan horse**, as the name suggests, is a malware that comes embedded in a seemingly harmless file such as an executable, an MS Office document, a screen saver or a PDF file.

When a user opens the malicious file, they fall victim to the malware.

5.1.2 Trojan Horse

The most common Trojan horses used by penetration testers are **backdoors**; this kind of Trojan horse lets an attacker get a shell on the infected system.



5.1.3 Backdoor

Backdoors are software made by two components: a server and a backdoor client.

The Backdoor server runs on the victim machine listening on the network and accepting connections. The client usually runs on the attacker machine, and it is used to connect to the backdoor to control it.

5.1.3 Backdoor

NetBus and **SubSeven** are very famous, old school backdoors; they allow the attacker to browse the victim's hard drive, upload and download files, execute programs and perform a number of other activities.

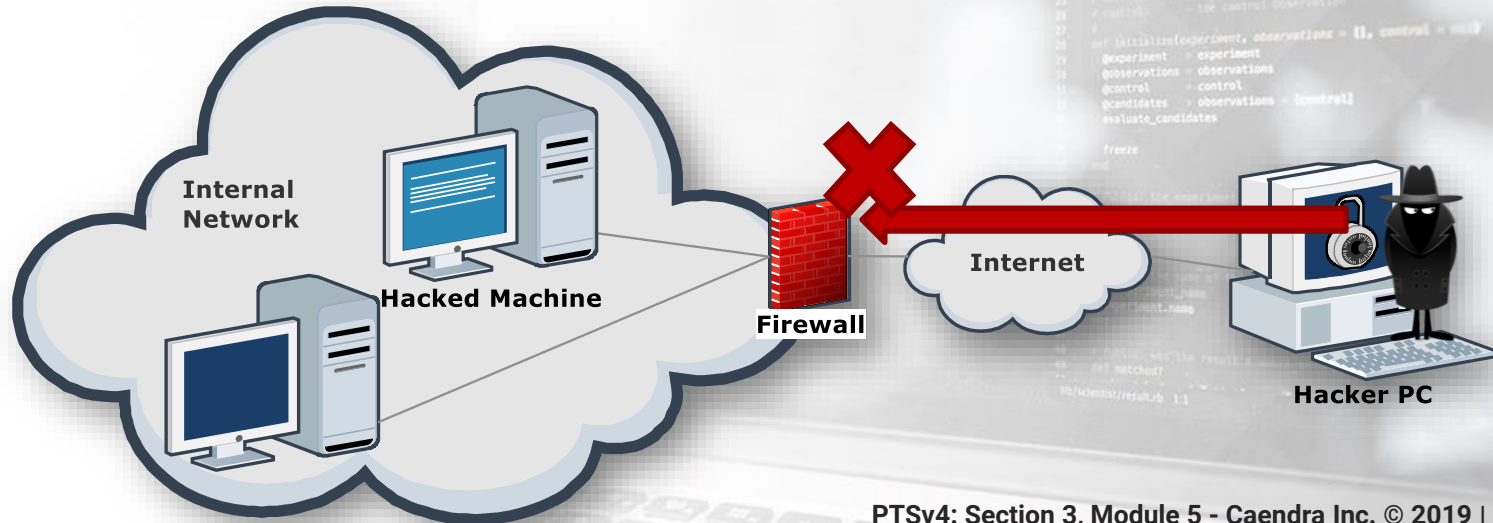
After installing and connecting to a backdoor, a penetration tester gets full control over the remote host.



5.1.3.1 Firewalls vs. Backdoors

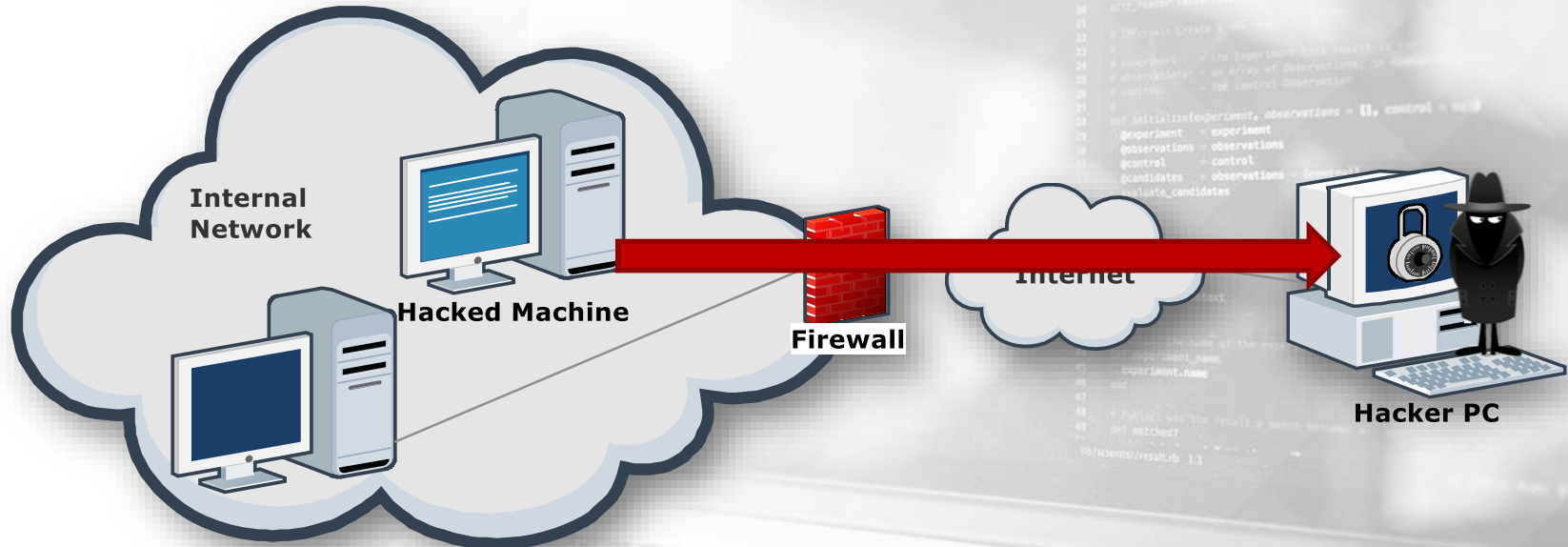
But what if a wise system administrator configures the network firewall to **block connections** from the internet to internal machines?

The attacker will not be able to connect to the backdoor!!!



5.1.3.2 Firewalls vs. Connect-Back Backdoors

A connect-back backdoor, or reverse backdoor, is a common mechanism to bypass firewalls.



5.1.3.2 Firewalls vs. Connect-Back Backdoors

Instead of having the victim machine act as a server and listening to the client's command, it acts as a client and connects back to the penetration tester's machine.

The attacker machine would listen on a port that is known to be commonly allowed on most of the firewalls, such as port 80 (the web server port).

```
1 def initialize_experiment, observations = [], control = nil
2
3   @experiment = experiment
4   @observations = observations
5   @control = control
6   @candidates = observations - {control}
7   evaluate_candidates
8
9   freeze
10
11   experiment.context
12
13   experiment.name
14
15   def match?
16     # A match was the result a match between an observation and a control
17     !! @candidates[result.sub 1]
```



5.1.3.2 Firewalls vs. Connect-Back Backdoors

A firewall cannot tell the difference between a user surfing the web and a backdoor connecting back to the attacker's machine!



5.1.4 Rootkit

A **rootkit** is a malware which is designed to hide itself from users and the antivirus program in order to completely subvert the OS functioning.

It lets an attacker maintain privileged access to the victim machine without being noticed.

```
def initialize(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - @control
  evaluate_candidates

  freeze
end

# TODO: the experiment's level
# TODO: the name of the experiment
def experiment_name
  @experiment_name
end

# TODO: what the result a match between an experiment
def match?
  @experiment.result == 1
end
```

5.1.5 Bootkit

Bootkits are rootkits which circumvent OS protection mechanisms by executing during the bootstrap phase.

They start before the operating system, so they get complete control over the machine and the OS.



5.1.6 Adware

Adware is annoying software that shows advertisements to computer users.



5.1.7 Spyware

Spyware is software used to collect information about users' activity. Spyware collects information such as:

- The OS installed on a machine
- Visited websites
- Passwords

The information is sent back to a log collection server controlled by the attacker.

5.1.8 Greyware

Greyware is a general term used to indicate Malware which does not fall under a specific category.

For example, it can be either spyware, adware or both.



5.1.9 Dialer

A **Dialer** is a software that tries to dial numbers on dial-up connections in order to collect money from the victim's phone bill.

Nowadays, dialers target **smartphones**.

```
38 def initialize(experiment, observations = [], candidates = [])
39   @experiment = experiment
40   @observations = observations
41   @control = control
42   @candidates = observations - @control
43   evaluate_candidates
44 end
45
46 def freeze
47   end
48
49 # PARS: the experiment's context
50 def context
51   experiment.context
52 end
53
54 # PARS: the name of the experiment
55 def experiment_name
56   experiment.name
57 end
58
59 # PARS: was the result a match between an agent
60 def matched?
61   ..
62 end
63
64 # PARS: result up
65 def result_up
66   1.1
```

5.1.10 Keylogger

A keylogger is a special software which records every keystroke on the remote victim machine.

Operations performed by keyloggers are:

- Recording keystrokes
- Recording the window name where the victim user was typing
- Saving the keystrokes in a log file on the victim machine
- Sending the logs to a server controlled by the penetration tester

```
21 def __init__(self, host, port):
22     self.host = host
23     self.port = port
24     self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
25     self.socket.connect((self.host, self.port))
26     self.send_data = {}
27     self.receive_data = {}
28     self.send_data['host'] = self.host
29     self.send_data['port'] = self.port
30     self.send_data['keylogger'] = True
31     self.send_data['keylogger'] = True
32     self.send_data['keylogger'] = True
33     self.send_data['keylogger'] = True
34     self.send_data['keylogger'] = True
35     self.send_data['keylogger'] = True
36     self.send_data['keylogger'] = True
37     self.send_data['keylogger'] = True
38     self.send_data['keylogger'] = True
39     self.send_data['keylogger'] = True
40     self.send_data['keylogger'] = True
41     self.send_data['keylogger'] = True
42     self.send_data['keylogger'] = True
43     self.send_data['keylogger'] = True
44     self.send_data['keylogger'] = True
45     self.send_data['keylogger'] = True
46     self.send_data['keylogger'] = True
47     self.send_data['keylogger'] = True
48     self.send_data['keylogger'] = True
49     self.send_data['keylogger'] = True
50     self.send_data['keylogger'] = True
51     self.send_data['keylogger'] = True
52     self.send_data['keylogger'] = True
53     self.send_data['keylogger'] = True
54     self.send_data['keylogger'] = True
55     self.send_data['keylogger'] = True
56     self.send_data['keylogger'] = True
57     self.send_data['keylogger'] = True
58     self.send_data['keylogger'] = True
59     self.send_data['keylogger'] = True
60     self.send_data['keylogger'] = True
61     self.send_data['keylogger'] = True
62     self.send_data['keylogger'] = True
63     self.send_data['keylogger'] = True
64     self.send_data['keylogger'] = True
65     self.send_data['keylogger'] = True
66     self.send_data['keylogger'] = True
67     self.send_data['keylogger'] = True
68     self.send_data['keylogger'] = True
69     self.send_data['keylogger'] = True
70     self.send_data['keylogger'] = True
71     self.send_data['keylogger'] = True
72     self.send_data['keylogger'] = True
73     self.send_data['keylogger'] = True
74     self.send_data['keylogger'] = True
75     self.send_data['keylogger'] = True
76     self.send_data['keylogger'] = True
77     self.send_data['keylogger'] = True
78     self.send_data['keylogger'] = True
79     self.send_data['keylogger'] = True
80     self.send_data['keylogger'] = True
81     self.send_data['keylogger'] = True
82     self.send_data['keylogger'] = True
83     self.send_data['keylogger'] = True
84     self.send_data['keylogger'] = True
85     self.send_data['keylogger'] = True
86     self.send_data['keylogger'] = True
87     self.send_data['keylogger'] = True
88     self.send_data['keylogger'] = True
89     self.send_data['keylogger'] = True
90     self.send_data['keylogger'] = True
91     self.send_data['keylogger'] = True
92     self.send_data['keylogger'] = True
93     self.send_data['keylogger'] = True
94     self.send_data['keylogger'] = True
95     self.send_data['keylogger'] = True
96     self.send_data['keylogger'] = True
97     self.send_data['keylogger'] = True
98     self.send_data['keylogger'] = True
99     self.send_data['keylogger'] = True
100    self.send_data['keylogger'] = True
```

5.1.10 Keylogger

Keyloggers are subject to the same restrictions that firewalls pose to backdoors.

If configured wisely, the traffic they generate should not be stopped by a firewall.

```
23 # @param @experiment - the experiment this result is for
24 # @param @observations - an array of Observations, in ascending
25 # @param @control - the control Observation
26
27
28 def initialize(experiment, observations = [], control = nil)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations - [control]
33   evaluate_candidates
34
35   freeze
36
37   @context =
38     experiment.context
39   end
40
41   # TODO: the name of the experiment
42
43   def experiment_name
44     @experiment.name
45   end
46
47   # TODO: the result a match between an
48
49   def matched?
50     # TODO:
51   end
52
53   # TODO: result up 1.1
```


5.1.10 Keylogger

You have just seen how software keyloggers work. There are also:

- **Hardware keyloggers**
- **Rootkit keyloggers**, which are stealthy and more invisible to the victim user than software keyloggers



5.1.10.1 Hardware Keylogger

Hardware keyloggers are small devices you can install between a keyboard and a computer.

They log keystrokes into an internal memory. An attacker needs two trips to the victim machine to exploit a hardware keylogger: one to install it and one to retrieve it.



5.1.10.1 Hardware Keylogger

Hardware keyloggers are less common than software ones, but they can be used by a penetration tester while performing physical security tests.

Unauthorized access to laboratories or offices may allow a malicious user to use these devices and record proprietary information.

```
24 # The experiment will result in 2  
25 # observations - an array of Observations, an Experiment  
26 # control - the control observation  
27  
28 def initialize(experiment, observations = [], control = null)  
29   @experiment = experiment  
30   @observations = observations  
31   @control = control  
32   @candidates = observations - [control]  
33   evaluate_candidates  
34  
35   freeze  
36  
37   experiment.context  
38  
39   @experiment_name =  
40     experiment.name  
41  
42   nil  
43  
44 # A class that will result a match between an  
45 def match?  
46   nil  
47  
48   nil  
49   nil  
50   nil  
51   nil  
52   nil  
53   nil  
54   nil  
55   nil  
56   nil  
57   nil  
58   nil  
59   nil  
60   nil  
61   nil  
62   nil  
63   nil  
64   nil  
65   nil  
66   nil  
67   nil  
68   nil  
69   nil  
70   nil  
71   nil  
72   nil  
73   nil  
74   nil  
75   nil  
76   nil  
77   nil  
78   nil  
79   nil  
80   nil  
81   nil  
82   nil  
83   nil  
84   nil  
85   nil  
86   nil  
87   nil  
88   nil  
89   nil  
90   nil  
91   nil  
92   nil  
93   nil  
94   nil  
95   nil  
96   nil  
97   nil  
98   nil  
99   nil  
100  nil  
101  nil  
102  nil  
103  nil  
104  nil  
105  nil  
106  nil  
107  nil  
108  nil  
109  nil  
110  nil  
111  nil  
112  nil  
113  nil  
114  nil  
115  nil  
116  nil  
117  nil  
118  nil  
119  nil  
120  nil  
121  nil  
122  nil  
123  nil  
124  nil  
125  nil  
126  nil  
127  nil  
128  nil  
129  nil  
130  nil  
131  nil  
132  nil  
133  nil  
134  nil  
135  nil  
136  nil  
137  nil  
138  nil  
139  nil  
140  nil  
141  nil  
142  nil  
143  nil  
144  nil  
145  nil  
146  nil  
147  nil  
148  nil  
149  nil  
150  nil  
151  nil  
152  nil  
153  nil  
154  nil  
155  nil  
156  nil  
157  nil  
158  nil  
159  nil  
160  nil  
161  nil  
162  nil  
163  nil  
164  nil  
165  nil  
166  nil  
167  nil  
168  nil  
169  nil  
170  nil  
171  nil  
172  nil  
173  nil  
174  nil  
175  nil  
176  nil  
177  nil  
178  nil  
179  nil  
180  nil  
181  nil  
182  nil  
183  nil  
184  nil  
185  nil  
186  nil  
187  nil  
188  nil  
189  nil  
190  nil  
191  nil  
192  nil  
193  nil  
194  nil  
195  nil  
196  nil  
197  nil  
198  nil  
199  nil  
200  nil  
201  nil  
202  nil  
203  nil  
204  nil  
205  nil  
206  nil  
207  nil  
208  nil  
209  nil  
210  nil  
211  nil  
212  nil  
213  nil  
214  nil  
215  nil  
216  nil  
217  nil  
218  nil  
219  nil  
220  nil  
221  nil  
222  nil  
223  nil  
224  nil  
225  nil  
226  nil  
227  nil  
228  nil  
229  nil  
230  nil  
231  nil  
232  nil  
233  nil  
234  nil  
235  nil  
236  nil  
237  nil  
238  nil  
239  nil  
240  nil  
241  nil  
242  nil  
243  nil  
244  nil  
245  nil  
246  nil  
247  nil  
248  nil  
249  nil  
250  nil  
251  nil  
252  nil  
253  nil  
254  nil  
255  nil  
256  nil  
257  nil  
258  nil  
259  nil  
260  nil  
261  nil  
262  nil  
263  nil  
264  nil  
265  nil  
266  nil  
267  nil  
268  nil  
269  nil  
270  nil  
271  nil  
272  nil  
273  nil  
274  nil  
275  nil  
276  nil  
277  nil  
278  nil  
279  nil  
280  nil  
281  nil  
282  nil  
283  nil  
284  nil  
285  nil  
286  nil  
287  nil  
288  nil  
289  nil  
290  nil  
291  nil  
292  nil  
293  nil  
294  nil  
295  nil  
296  nil  
297  nil  
298  nil  
299  nil  
300  nil  
301  nil  
302  nil  
303  nil  
304  nil  
305  nil  
306  nil  
307  nil  
308  nil  
309  nil  
310  nil  
311  nil  
312  nil  
313  nil  
314  nil  
315  nil  
316  nil  
317  nil  
318  nil  
319  nil  
320  nil  
321  nil  
322  nil  
323  nil  
324  nil  
325  nil  
326  nil  
327  nil  
328  nil  
329  nil  
330  nil  
331  nil  
332  nil  
333  nil  
334  nil  
335  nil  
336  nil  
337  nil  
338  nil  
339  nil  
340  nil  
341  nil  
342  nil  
343  nil  
344  nil  
345  nil  
346  nil  
347  nil  
348  nil  
349  nil  
350  nil  
351  nil  
352  nil  
353  nil  
354  nil  
355  nil  
356  nil  
357  nil  
358  nil  
359  nil  
360  nil  
361  nil  
362  nil  
363  nil  
364  nil  
365  nil  
366  nil  
367  nil  
368  nil  
369  nil  
370  nil  
371  nil  
372  nil  
373  nil  
374  nil  
375  nil  
376  nil  
377  nil  
378  nil  
379  nil  
380  nil  
381  nil  
382  nil  
383  nil  
384  nil  
385  nil  
386  nil  
387  nil  
388  nil  
389  nil  
390  nil  
391  nil  
392  nil  
393  nil  
394  nil  
395  nil  
396  nil  
397  nil  
398  nil  
399  nil  
400  nil  
401  nil  
402  nil  
403  nil  
404  nil  
405  nil  
406  nil  
407  nil  
408  nil  
409  nil  
410  nil  
411  nil  
412  nil  
413  nil  
414  nil  
415  nil  
416  nil  
417  nil  
418  nil  
419  nil  
420  nil  
421  nil  
422  nil  
423  nil  
424  nil  
425  nil  
426  nil  
427  nil  
428  nil  
429  nil  
430  nil  
431  nil  
432  nil  
433  nil  
434  nil  
435  nil  
436  nil  
437  nil  
438  nil  
439  nil  
440  nil  
441  nil  
442  nil  
443  nil  
444  nil  
445  nil  
446  nil  
447  nil  
448  nil  
449  nil  
450  nil  
451  nil  
452  nil  
453  nil  
454  nil  
455  nil  
456  nil  
457  nil  
458  nil  
459  nil  
460  nil  
461  nil  
462  nil  
463  nil  
464  nil  
465  nil  
466  nil  
467  nil  
468  nil  
469  nil  
470  nil  
471  nil  
472  nil  
473  nil  
474  nil  
475  nil  
476  nil  
477  nil  
478  nil  
479  nil  
480  nil  
481  nil  
482  nil  
483  nil  
484  nil  
485  nil  
486  nil  
487  nil  
488  nil  
489  nil  
490  nil  
491  nil  
492  nil  
493  nil  
494  nil  
495  nil  
496  nil  
497  nil  
498  nil  
499  nil  
500  nil  
501  nil  
502  nil  
503  nil  
504  nil  
505  nil  
506  nil  
507  nil  
508  nil  
509  nil  
510  nil  
511  nil  
512  nil  
513  nil  
514  nil  
515  nil  
516  nil  
517  nil  
518  nil  
519  nil  
520  nil  
521  nil  
522  nil  
523  nil  
524  nil  
525  nil  
526  nil  
527  nil  
528  nil  
529  nil  
530  nil  
531  nil  
532  nil  
533  nil  
534  nil  
535  nil  
536  nil  
537  nil  
538  nil  
539  nil  
540  nil  
541  nil  
542  nil  
543  nil  
544  nil  
545  nil  
546  nil  
547  nil  
548  nil  
549  nil  
550  nil  
551  nil  
552  nil  
553  nil  
554  nil  
555  nil  
556  nil  
557  nil  
558  nil  
559  nil  
560  nil  
561  nil  
562  nil  
563  nil  
564  nil  
565  nil  
566  nil  
567  nil  
568  nil  
569  nil  
570  nil  
571  nil  
572  nil  
573  nil  
574  nil  
575  nil  
576  nil  
577  nil  
578  nil  
579  nil  
580  nil  
581  nil  
582  nil  
583  nil  
584  nil  
585  nil  
586  nil  
587  nil  
588  nil  
589  nil  
590  nil  
591  nil  
592  nil  
593  nil  
594  nil  
595  nil  
596  nil  
597  nil  
598  nil  
599  nil  
600  nil  
601  nil  
602  nil  
603  nil  
604  nil  
605  nil  
606  nil  
607  nil  
608  nil  
609  nil  
610  nil  
611  nil  
612  nil  
613  nil  
614  nil  
615  nil  
616  nil  
617  nil  
618  nil  
619  nil  
620  nil  
621  nil  
622  nil  
623  nil  
624  nil  
625  nil  
626  nil  
627  nil  
628  nil  
629  nil  
630  nil  
631  nil  
632  nil  
633  nil  
634  nil  
635  nil  
636  nil  
637  nil  
638  nil  
639  nil  
640  nil  
641  nil  
642  nil  
643  nil  
644  nil  
645  nil  
646  nil  
647  nil  
648  nil  
649  nil  
650  nil  
651  nil  
652  nil  
653  nil  
654  nil  
655  nil  
656  nil  
657  nil  
658  nil  
659  nil  
660  nil  
661  nil  
662  nil  
663  nil  
664  nil  
665  nil  
666  nil  
667  nil  
668  nil  
669  nil  
670  nil  
671  nil  
672  nil  
673  nil  
674  nil  
675  nil  
676  nil  
677  nil  
678  nil  
679  nil  
680  nil  
681  nil  
682  nil  
683  nil  
684  nil  
685  nil  
686  nil  
687  nil  
688  nil  
689  nil  
690  nil  
691  nil  
692  nil  
693  nil  
694  nil  
695  nil  
696  nil  
697  nil  
698  nil  
699  nil  
700  nil  
701  nil  
702  nil  
703  nil  
704  nil  
705  nil  
706  nil  
707  nil  
708  nil  
709  nil  
710  nil  
711  nil  
712  nil  
713  nil  
714  nil  
715  nil  
716  nil  
717  nil  
718  nil  
719  nil  
720  nil  
721  nil  
722  nil  
723  nil  
724  nil  
725  nil  
726  nil  
727  nil  
728  nil  
729  nil  
730  nil  
731  nil  
732  nil  
733  nil  
734  nil  
735  nil  
736  nil  
737  nil  
738  nil  
739  nil  
740  nil  
741  nil  
742  nil  
743  nil  
744  nil  
745  nil  
746  nil  
747  nil  
748  nil  
749  nil  
750  nil  
751  nil  
752  nil  
753  nil  
754  nil  
755  nil  
756  nil  
757  nil  
758  nil  
759  nil  
760  nil  
761  nil  
762  nil  
763  nil  
764  nil  
765  nil  
766  nil  
767  nil  
768  nil  
769  nil  
770  nil  
771  nil  
772  nil  
773  nil  
774  nil  
775  nil  
776  nil  
777  nil  
778  nil  
779  nil  
780  nil  
781  nil  
782  nil  
783  nil  
784  nil  
785  nil  
786  nil  
787  nil  
788  nil  
789  nil  
790  nil  
791  nil  
792  nil  
793  nil  
794  nil  
795  nil  
796  nil  
797  nil  
798  nil  
799  nil  
800  nil  
801  nil  
802  nil  
803  nil  
804  nil  
805  nil  
806  nil  
807  nil  
808  nil  
809  nil  
810  nil  
811  nil  
812  nil  
813  nil  
814  nil  
815  nil  
816  nil  
817  nil  
818  nil  
819  nil  
820  nil  
821  nil  
822  nil  
823  nil  
824  nil  
825  nil  
826  nil  
827  nil  
828  nil  
829  nil  
830  nil  
831  nil  
832  nil  
833  nil  
834  nil  
835  nil  
836  nil  
837  nil  
838  nil  
839  nil  
840  nil  
841  nil  
842  nil  
843  nil  
844  nil  
845  nil  
846  nil  
847  nil  
848  nil  
849  nil  
850  nil  
851  nil  
852  nil  
853  nil  
854  nil  
855  nil  
856  nil  
857  nil  
858  nil  
859  nil  
860  nil  
861  nil  
862  nil  
863  nil  
864  nil  
865  nil  
866  nil  
867  nil  
868  nil  
869  nil  
870  nil  
871  nil  
872  nil  
873  nil  
874  nil  
875  nil  
876  nil  
877  nil  
878  nil  
879  nil  
880  nil  
881  nil  
882  nil  
883  nil  
884  nil  
885  nil  
886  nil  
887  nil  
888  nil  
889  nil  
890  nil  
891  nil  
892  nil  
893  nil  
894  nil  
895  nil  
896  nil  
897  nil  
898  nil  
899  nil  
900  nil  
901  nil  
902  nil  
903  nil  
904  nil  
905  nil  
906  nil  
907  nil  
908  nil  
909  nil  
910  nil  
911  nil  
912  nil  
913  nil  
914  nil  
915  nil  
916  nil  
917  nil  
918  nil  
919  nil  
920  nil  
921  nil  
922  nil  
923  nil  
924  nil  
925  nil  
926  nil  
927  nil  
928  nil  
929  nil  
930  nil  
931  nil  
932  nil  
933  nil  
934  nil  
935  nil  
936  nil  
937  nil  
938  nil  
939  nil  
940  nil  
941  nil  
942  nil  
943  nil  
944  nil  
945  nil  
946  nil  
947  nil  
948  nil  
949  nil  
950  nil  
951  nil  
952  nil  
953  nil  
954  nil  
955  nil  
956  nil  
957  nil  
958  nil  
959  nil  
960  nil  
961  nil  
962  nil  
963  nil  
964  nil  
965  nil  
966  nil  
967  nil  
968  nil  
969  nil  
970  nil  
971  nil  
972  nil  
973  nil  
974  nil  
975  nil  
976  nil  
977  nil  
978  nil  
979  nil  
980  nil  
981  nil  
982  nil  
983  nil  
984  nil  
985  nil  
986  nil  
987  nil  
988  nil  
989  nil  
990  nil  
991  nil  
992  nil  
993  nil  
994  nil  
995  nil  
996  nil  
997  nil  
998  nil  
999  nil  
1000 nil
```

5.1.10.2 Rootkit Keylogger

Rootkit keyloggers are software keyloggers working at the Kernel level by **hijacking the operating system APIs** to record keystrokes.

Every time a key is pressed on a keyboard, a particular function of the OS Kernel is called through a mechanism called an interrupt.

5.1.10.2 Rootkit Keylogger

There are many different interrupts in a system, each handling a specific function in the system: reading/writing to disk, calling device drivers, and so on.

Every time someone presses a key on a keyboard, the keyboard interrupt is called.

5.1.10.2 Rootkit Keylogger

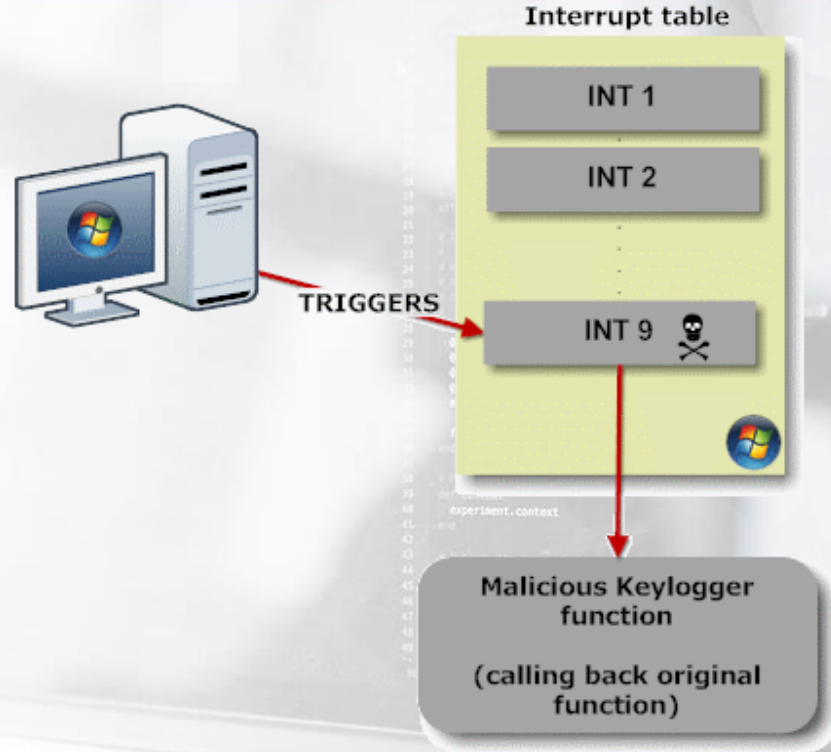
The interrupt calls a particular function of the operating system that actually performs the operation intended for the key.

By taking control of this function, the rootkit manages to know which key has been pressed and records it for later use.

```
24 * @param result - the experiment result as a list
25 * @param observations - an array of Observations, an Observation
26 * @param control - the control Observation
27
28 def skillRanking(experiment, observations = [], control = null)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations - [control]
33   evaluate_candidates
34
35   freeze
36
37   return
38   experiment.context
39
40   experiment.name
41   experiment.name
42
43   return
44
45 * @param result - the result as a match between an experiment
46 * @param match - the match
47
48 def match
49   return result
50
51 def match
52   return result
53
54 def match
55   return result
56
57 def match
58   return result
59
60 def match
61   return result
62
63 def match
64   return result
65
66 def match
67   return result
68
69 def match
70   return result
71
72 def match
73   return result
74
75 def match
76   return result
77
78 def match
79   return result
80
81 def match
82   return result
83
84 def match
85   return result
86
87 def match
88   return result
89
90 def match
91   return result
92
93 def match
94   return result
95
96 def match
97   return result
98
99 def match
100   return result
```

5.1.10.2 Rootkit Keylogger

First, the keylogger logs the key pressed and then it calls the original function of the operating system.



5.1.11 Bots

Bots are small pieces of software that get installed on millions of Internet-connected machines to perform Distributed Denial of Service or serving as spamming sources.

These Bots are commanded remotely by a so-called **Command and Control** server. The C&C server can instruct thousands or even millions of bots to perform a given operation simultaneously.

```
24 # The experiment data
25 # observations - an array of Observations, in this case
26 # control - the control observation
27
28 def initialize(experiment, observations = [], control = null)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = initialize_candidates
33   evaluate_candidates
34
35   freeze
36
37   experiment.context
38
39   experiment.name
40
41   experiment.name
42
43   experiment.name
44
45   experiment.name
46
47   experiment.name
48
49   experiment.name
50
51   experiment.name
52
53   experiment.name
54
55   experiment.name
56
57   experiment.name
58
59   experiment.name
60
61   experiment.name
62
63   experiment.name
64
65   experiment.name
66
67   experiment.name
68
69   experiment.name
70
71   experiment.name
72
73   experiment.name
74
75   experiment.name
76
77   experiment.name
78
79   experiment.name
80
81   experiment.name
82
83   experiment.name
84
85   experiment.name
86
87   experiment.name
88
89   experiment.name
90
91   experiment.name
92
93   experiment.name
94
95   experiment.name
96
97   experiment.name
98
99   experiment.name
100  experiment.name
```

5.1.12 Ransomware

Ransomware is software that encrypts a computer or smartphone content with a secret key.

It then asks its victims for a ransom to give them the content back.

```
def initialize_experiment(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations + [control]
  evaluate_candidates

  freeze

  experiment.context
end

# Initialize the name of the experiment
def experiment_name
  experiment.name
end

# Check if the result is a match between the
def matched?
  # ...
end

# ...
end
```


5.1.13 Data Stealing Malware

Data stealing malware has one precise goal: stealing the most important data on the victim's hard disk and sending it back to the attacker. Most of the time, this specific malware incarnation is **targeted** to a specific company and tailored to work on the target environment.

As an alternative, an attacker could use a backdoor to perform data stealing.



5.1.14 Worms

Worms spread over the network by exploiting operating systems and software vulnerabilities. Worms can also exploit default credentials or misconfigurations to attack a service or a machine.

Usually, worms are part of other malware, and they offer an entry point into the target system.



5.1.15 Video – Backdoor

Backdoor

In this video, you will see how to use different tools to create a backdoor on an exploited machine. Every backdoor has its own unique features, so you will learn how to choose the right tool for the job!



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

5.1.16 References

If you want to study more on this subject, a book you can reference and some malware examples to test!

- [Hacking Exposed Malware & Rootkits: Security Secrets and Solutions, Second Edition](#) – Structure, detection and attack methods
- [Malware samples](#) – Do not run them on your computer!
Use a virtual machine as a testing environment!

http://www.amazon.com/Hacking-Exposed-Malware-amp-Rootkits/dp/0071823077/ref=sr_1_3?s=books&ie=UTF8&qid=1423220013&sr=1-3
<http://www.offensivecomputing.net/?q=taxonomy/term/1>

Password Attacks



5.2 Password Attacks

How does this support my pentesting career?

Ability to:

- Gain persistent access to an exploited machine
- Choose the right method to attack passwords
- Exploit re-used credentials

5.2 Password Attacks

Passwords are the first and, most of the time, the only line of defense of systems, services and accounts against unauthorized access. To protect their users and applications, operating systems have to store passwords securely.

Passwords are usually stored inside files or databases. If they were **stored in clear-text**, attackers would just have to open the password file, or database, to steal them.

5.2 Password Attacks

Getting control over the credentials of an account means getting full control over the account and its privileges on a system.

Because of that, computer systems passwords must be stored in an **encrypted form**; this prevents a malicious local user from getting to know other users' passwords.

```
24 # the experiment context is the experiment
25 # observations - an array of Observations, in this case
26 # control - the control observation
27
28 def initialize(experiment, observations = [], control = nil)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations + [control]
33   evaluate_candidates
34
35   freeze
36
37   @experiment = experiment
38   @experiment.context
39
40   @experiment_name
41   @experiment.name
42
43   @experiment
44   @experiment
45
46   @experiment
47   @experiment
48
49   @experiment
50   @experiment
51
52   @experiment
53   @experiment
54
55   @experiment
56   @experiment
57
58   @experiment
59   @experiment
60
61   @experiment
62   @experiment
63
64   @experiment
65   @experiment
66
67   @experiment
68   @experiment
69
70   @experiment
71   @experiment
72
73   @experiment
74   @experiment
75
76   @experiment
77   @experiment
78
79   @experiment
80   @experiment
81
82   @experiment
83   @experiment
84
85   @experiment
86   @experiment
87
88   @experiment
89   @experiment
90
91   @experiment
92   @experiment
93
94   @experiment
95   @experiment
96
97   @experiment
98   @experiment
99
100  @experiment
101  @experiment
102
103  @experiment
104  @experiment
105
106  @experiment
107  @experiment
108
109  @experiment
110  @experiment
111
112  @experiment
113  @experiment
114
115  @experiment
116  @experiment
117
118  @experiment
119  @experiment
120
121  @experiment
122  @experiment
123
124  @experiment
125  @experiment
126
127  @experiment
128  @experiment
129
130  @experiment
131  @experiment
132
133  @experiment
134  @experiment
135
136  @experiment
137  @experiment
138
139  @experiment
140  @experiment
141
142  @experiment
143  @experiment
144
145  @experiment
146  @experiment
147
148  @experiment
149  @experiment
150
151  @experiment
152  @experiment
153
154  @experiment
155  @experiment
156
157  @experiment
158  @experiment
159
160  @experiment
161  @experiment
162
163  @experiment
164  @experiment
165
166  @experiment
167  @experiment
168
169  @experiment
170  @experiment
171
172  @experiment
173  @experiment
174
175  @experiment
176  @experiment
177
178  @experiment
179  @experiment
180
181  @experiment
182  @experiment
183
184  @experiment
185  @experiment
186
187  @experiment
188  @experiment
189
190  @experiment
191  @experiment
192
193  @experiment
194  @experiment
195
196  @experiment
197  @experiment
198
199  @experiment
200  @experiment
201
202  @experiment
203  @experiment
204
205  @experiment
206  @experiment
207
208  @experiment
209  @experiment
210
211  @experiment
212  @experiment
213
214  @experiment
215  @experiment
216
217  @experiment
218  @experiment
219
220  @experiment
221  @experiment
222
223  @experiment
224  @experiment
225
226  @experiment
227  @experiment
228
229  @experiment
230  @experiment
231
232  @experiment
233  @experiment
234
235  @experiment
236  @experiment
237
238  @experiment
239  @experiment
240
241  @experiment
242  @experiment
243
244  @experiment
245  @experiment
246
247  @experiment
248  @experiment
249
250  @experiment
251  @experiment
252
253  @experiment
254  @experiment
255
256  @experiment
257  @experiment
258
259  @experiment
260  @experiment
261
262  @experiment
263  @experiment
264
265  @experiment
266  @experiment
267
268  @experiment
269  @experiment
270
271  @experiment
272  @experiment
273
274  @experiment
275  @experiment
276
277  @experiment
278  @experiment
279
280  @experiment
281  @experiment
282
283  @experiment
284  @experiment
285
286  @experiment
287  @experiment
288
289  @experiment
290  @experiment
291
292  @experiment
293  @experiment
294
295  @experiment
296  @experiment
297
298  @experiment
299  @experiment
300
301  @experiment
302  @experiment
303
304  @experiment
305  @experiment
306
307  @experiment
308  @experiment
309
310  @experiment
311  @experiment
312
313  @experiment
314  @experiment
315
316  @experiment
317  @experiment
318
319  @experiment
320  @experiment
321
322  @experiment
323  @experiment
324
325  @experiment
326  @experiment
327
328  @experiment
329  @experiment
330
331  @experiment
332  @experiment
333
334  @experiment
335  @experiment
336
337  @experiment
338  @experiment
339
340  @experiment
341  @experiment
342
343  @experiment
344  @experiment
345
346  @experiment
347  @experiment
348
349  @experiment
350  @experiment
351
352  @experiment
353  @experiment
354
355  @experiment
356  @experiment
357
358  @experiment
359  @experiment
360
361  @experiment
362  @experiment
363
364  @experiment
365  @experiment
366
367  @experiment
368  @experiment
369
370  @experiment
371  @experiment
372
373  @experiment
374  @experiment
375
376  @experiment
377  @experiment
378
379  @experiment
380  @experiment
381
382  @experiment
383  @experiment
384
385  @experiment
386  @experiment
387
388  @experiment
389  @experiment
390
391  @experiment
392  @experiment
393
394  @experiment
395  @experiment
396
397  @experiment
398  @experiment
399
400  @experiment
401  @experiment
402
403  @experiment
404  @experiment
405
406  @experiment
407  @experiment
408
409  @experiment
410  @experiment
411
412  @experiment
413  @experiment
414
415  @experiment
416  @experiment
417
418  @experiment
419  @experiment
420
421  @experiment
422  @experiment
423
424  @experiment
425  @experiment
426
427  @experiment
428  @experiment
429
430  @experiment
431  @experiment
432
433  @experiment
434  @experiment
435
436  @experiment
437  @experiment
438
439  @experiment
440  @experiment
441
442  @experiment
443  @experiment
444
445  @experiment
446  @experiment
447
448  @experiment
449  @experiment
450
451  @experiment
452  @experiment
453
454  @experiment
455  @experiment
456
457  @experiment
458  @experiment
459
460  @experiment
461  @experiment
462
463  @experiment
464  @experiment
465
466  @experiment
467  @experiment
468
469  @experiment
470  @experiment
471
472  @experiment
473  @experiment
474
475  @experiment
476  @experiment
477
478  @experiment
479  @experiment
480
481  @experiment
482  @experiment
483
484  @experiment
485  @experiment
486
487  @experiment
488  @experiment
489
490  @experiment
491  @experiment
492
493  @experiment
494  @experiment
495
496  @experiment
497  @experiment
498
499  @experiment
500  @experiment
501
502  @experiment
503  @experiment
504
505  @experiment
506  @experiment
507
508  @experiment
509  @experiment
510
511  @experiment
512  @experiment
513
514  @experiment
515  @experiment
516
517  @experiment
518  @experiment
519
520  @experiment
521  @experiment
522
523  @experiment
524  @experiment
525
526  @experiment
527  @experiment
528
529  @experiment
530  @experiment
531
532  @experiment
533  @experiment
534
535  @experiment
536  @experiment
537
538  @experiment
539  @experiment
540
541  @experiment
542  @experiment
543
544  @experiment
545  @experiment
546
547  @experiment
548  @experiment
549
550  @experiment
551  @experiment
552
553  @experiment
554  @experiment
555
556  @experiment
557  @experiment
558
559  @experiment
560  @experiment
561
562  @experiment
563  @experiment
564
565  @experiment
566  @experiment
567
568  @experiment
569  @experiment
570
571  @experiment
572  @experiment
573
574  @experiment
575  @experiment
576
577  @experiment
578  @experiment
579
580  @experiment
581  @experiment
582
583  @experiment
584  @experiment
585
586  @experiment
587  @experiment
588
589  @experiment
590  @experiment
591
592  @experiment
593  @experiment
594
595  @experiment
596  @experiment
597
598  @experiment
599  @experiment
600
601  @experiment
602  @experiment
603
604  @experiment
605  @experiment
606
607  @experiment
608  @experiment
609
610  @experiment
611  @experiment
612
613  @experiment
614  @experiment
615
616  @experiment
617  @experiment
618
619  @experiment
620  @experiment
621
622  @experiment
623  @experiment
624
625  @experiment
626  @experiment
627
628  @experiment
629  @experiment
630
631  @experiment
632  @experiment
633
634  @experiment
635  @experiment
636
637  @experiment
638  @experiment
639
640  @experiment
641  @experiment
642
643  @experiment
644  @experiment
645
646  @experiment
647  @experiment
648
649  @experiment
650  @experiment
651
652  @experiment
653  @experiment
654
655  @experiment
656  @experiment
657
658  @experiment
659  @experiment
660
661  @experiment
662  @experiment
663
664  @experiment
665  @experiment
666
667  @experiment
668  @experiment
669
670  @experiment
671  @experiment
672
673  @experiment
674  @experiment
675
676  @experiment
677  @experiment
678
679  @experiment
680  @experiment
681
682  @experiment
683  @experiment
684
685  @experiment
686  @experiment
687
688  @experiment
689  @experiment
690
691  @experiment
692  @experiment
693
694  @experiment
695  @experiment
696
697  @experiment
698  @experiment
699
700  @experiment
701  @experiment
702
703  @experiment
704  @experiment
705
706  @experiment
707  @experiment
708
709  @experiment
710  @experiment
711
712  @experiment
713  @experiment
714
715  @experiment
716  @experiment
717
718  @experiment
719  @experiment
720
721  @experiment
722  @experiment
723
724  @experiment
725  @experiment
726
727  @experiment
728  @experiment
729
730  @experiment
731  @experiment
732
733  @experiment
734  @experiment
735
736  @experiment
737  @experiment
738
739  @experiment
740  @experiment
741
742  @experiment
743  @experiment
744
745  @experiment
746  @experiment
747
748  @experiment
749  @experiment
750
751  @experiment
752  @experiment
753
754  @experiment
755  @experiment
756
757  @experiment
758  @experiment
759
760  @experiment
761  @experiment
762
763  @experiment
764  @experiment
765
766  @experiment
767  @experiment
768
769  @experiment
770  @experiment
771
772  @experiment
773  @experiment
774
775  @experiment
776  @experiment
777
778  @experiment
779  @experiment
780
781  @experiment
782  @experiment
783
784  @experiment
785  @experiment
786
787  @experiment
788  @experiment
789
790  @experiment
791  @experiment
792
793  @experiment
794  @experiment
795
796  @experiment
797  @experiment
798
799  @experiment
800  @experiment
801
802  @experiment
803  @experiment
804
805  @experiment
806  @experiment
807
808  @experiment
809  @experiment
810
811  @experiment
812  @experiment
813
814  @experiment
815  @experiment
816
817  @experiment
818  @experiment
819
820  @experiment
821  @experiment
822
823  @experiment
824  @experiment
825
826  @experiment
827  @experiment
828
829  @experiment
830  @experiment
831
832  @experiment
833  @experiment
834
835  @experiment
836  @experiment
837
838  @experiment
839  @experiment
840
841  @experiment
842  @experiment
843
844  @experiment
845  @experiment
846
847  @experiment
848  @experiment
849
850  @experiment
851  @experiment
852
853  @experiment
854  @experiment
855
856  @experiment
857  @experiment
858
859  @experiment
860  @experiment
861
862  @experiment
863  @experiment
864
865  @experiment
866  @experiment
867
868  @experiment
869  @experiment
870
871  @experiment
872  @experiment
873
874  @experiment
875  @experiment
876
877  @experiment
878  @experiment
879
880  @experiment
881  @experiment
882
883  @experiment
884  @experiment
885
886  @experiment
887  @experiment
888
889  @experiment
890  @experiment
891
892  @experiment
893  @experiment
894
895  @experiment
896  @experiment
897
898  @experiment
899  @experiment
900
901  @experiment
902  @experiment
903
904  @experiment
905  @experiment
906
907  @experiment
908  @experiment
909
910  @experiment
911  @experiment
912
913  @experiment
914  @experiment
915
916  @experiment
917  @experiment
918
919  @experiment
920  @experiment
921
922  @experiment
923  @experiment
924
925  @experiment
926  @experiment
927
928  @experiment
929  @experiment
929
```

5.2 Password Attacks

To make things even harder for attackers, passwords are stored by using a **one-way encryption algorithm**; there is no way to know the password starting from its encrypted form.

Cryptographic hashing functions are used to transform a password from its clear-text form to an encrypted and safe to store form.



5.2 Password Attacks

When you log into your computer, you type your username and password. The operating system takes the password, hashes it and then tries to match the result against the saved hash in the password database. If the two values match, you log in successfully.

The operating system does not need to know the clear-text password!



5.2 Password Attacks

As you can imagine, trying to manually crack a password is more than impractical. To automate such processes, there are two main strategies:

- **Brute force attacks**
- **Dictionary attacks**

In the following slides, we will discuss the differences between the two, pros and cons, and the tools you can use to crack a password.

```
24 # def __init__(self, experiment, observations, control):
25     self.experiment = experiment
26     self.observations = observations
27     self.control = control
28
29 # def __str__(self):
30     return f'Experiment: {self.experiment}, Observations: {self.observations}, Control: {self.control}'
31
32 # def __repr__(self):
33     return f'Experiment: {self.experiment}, Observations: {self.observations}, Control: {self.control}'
34
35 # def __eq__(self, other):
36     return self.experiment == other.experiment and self.observations == other.observations and self.control == other.control
37
38 # def __neq__(self, other):
39     return not self.__eq__(other)
40
41 # def __lt__(self, other):
42     return self.experiment < other.experiment
43
44 # def __gt__(self, other):
45     return self.experiment > other.experiment
46
47 # def __le__(self, other):
48     return self.experiment <= other.experiment
49
50 # def __ge__(self, other):
51     return self.experiment >= other.experiment
52
53 # def __hash__(self):
54     return hash(self.experiment + self.observations + self.control)
55
56 # def __getitem__(self, key):
57     if key == 'experiment':
58         return self.experiment
59     elif key == 'observations':
60         return self.observations
61     elif key == 'control':
62         return self.control
63     else:
64         raise KeyError(f'Key {key} not found')
65
66 # def __setitem__(self, key, value):
67     if key == 'experiment':
68         self.experiment = value
69     elif key == 'observations':
70         self.observations = value
71     elif key == 'control':
72         self.control = value
73     else:
74         raise KeyError(f'Key {key} not found')
75
76 # def __delitem__(self, key):
77     if key == 'experiment':
78         del self.experiment
79     elif key == 'observations':
80         del self.observations
81     elif key == 'control':
82         del self.control
83     else:
84         raise KeyError(f'Key {key} not found')
85
86 # def __contains__(self, key):
87     return key in self.__dict__
88
89 # def __iter__(self):
90     return iter(self.__dict__.items())
91
92 # def __len__(self):
93     return len(self.__dict__)
94
95 # def __bool__(self):
96     return True
97
98 # def __call__(self):
99     return self
100
101 # def __getattr__(self, name):
102     if name in self.__dict__:
103         return self.__dict__[name]
104     else:
105         raise AttributeError(f'Attribute {name} not found')
106
107 # def __setattr__(self, name, value):
108     self.__dict__[name] = value
109
110 # def __delattr__(self, name):
111     del self.__dict__[name]
112
113 # def __copy__(self):
114     return self.__class__(self.experiment, self.observations, self.control)
115
116 # def __deepcopy__(self, memo):
117     return self.__copy__()
```

5.2.1 Brute Force Attacks

Q

*How can you be **certain** about finding someone's password?*

A

*You simply have to **try them all**!*

This is the method that brute force attacks use; they generate and test all the possible valid passwords, as this is the only method that gives you the certainty of finding a user's password.

```
1 # Brute Force Attack
2 # This script will attempt to guess the password of a user by trying
3 # every possible combination of characters.
4 # The user's password is stored in a file named 'passwords.txt'.
5 # The script will attempt to guess the password by trying every
6 # possible combination of characters.
7 # The script will attempt to guess the password by trying every
8 # possible combination of characters.
9 # The script will attempt to guess the password by trying every
10 # possible combination of characters.
11 # The script will attempt to guess the password by trying every
12 # possible combination of characters.
13 # The script will attempt to guess the password by trying every
14 # possible combination of characters.
15 # The script will attempt to guess the password by trying every
16 # possible combination of characters.
17 # The script will attempt to guess the password by trying every
18 # possible combination of characters.
19 # The script will attempt to guess the password by trying every
20 # possible combination of characters.
21 # The script will attempt to guess the password by trying every
22 # possible combination of characters.
23 # The script will attempt to guess the password by trying every
24 # possible combination of characters.
25 # The script will attempt to guess the password by trying every
26 # possible combination of characters.
27 # The script will attempt to guess the password by trying every
28 # possible combination of characters.
29 # The script will attempt to guess the password by trying every
30 # possible combination of characters.
31 # The script will attempt to guess the password by trying every
32 # possible combination of characters.
33 # The script will attempt to guess the password by trying every
34 # possible combination of characters.
35 # The script will attempt to guess the password by trying every
36 # possible combination of characters.
37 # The script will attempt to guess the password by trying every
38 # possible combination of characters.
39 # The script will attempt to guess the password by trying every
40 # possible combination of characters.
41 # The script will attempt to guess the password by trying every
42 # possible combination of characters.
43 # The script will attempt to guess the password by trying every
44 # possible combination of characters.
45 # The script will attempt to guess the password by trying every
46 # possible combination of characters.
47 # The script will attempt to guess the password by trying every
48 # possible combination of characters.
49 # The script will attempt to guess the password by trying every
50 # possible combination of characters.
51 # The script will attempt to guess the password by trying every
52 # possible combination of characters.
53 # The script will attempt to guess the password by trying every
54 # possible combination of characters.
55 # The script will attempt to guess the password by trying every
56 # possible combination of characters.
57 # The script will attempt to guess the password by trying every
58 # possible combination of characters.
59 # The script will attempt to guess the password by trying every
60 # possible combination of characters.
61 # The script will attempt to guess the password by trying every
62 # possible combination of characters.
63 # The script will attempt to guess the password by trying every
64 # possible combination of characters.
65 # The script will attempt to guess the password by trying every
66 # possible combination of characters.
67 # The script will attempt to guess the password by trying every
68 # possible combination of characters.
69 # The script will attempt to guess the password by trying every
70 # possible combination of characters.
71 # The script will attempt to guess the password by trying every
72 # possible combination of characters.
73 # The script will attempt to guess the password by trying every
74 # possible combination of characters.
75 # The script will attempt to guess the password by trying every
76 # possible combination of characters.
77 # The script will attempt to guess the password by trying every
78 # possible combination of characters.
79 # The script will attempt to guess the password by trying every
80 # possible combination of characters.
81 # The script will attempt to guess the password by trying every
82 # possible combination of characters.
83 # The script will attempt to guess the password by trying every
84 # possible combination of characters.
85 # The script will attempt to guess the password by trying every
86 # possible combination of characters.
87 # The script will attempt to guess the password by trying every
88 # possible combination of characters.
89 # The script will attempt to guess the password by trying every
90 # possible combination of characters.
91 # The script will attempt to guess the password by trying every
92 # possible combination of characters.
93 # The script will attempt to guess the password by trying every
94 # possible combination of characters.
95 # The script will attempt to guess the password by trying every
96 # possible combination of characters.
97 # The script will attempt to guess the password by trying every
98 # possible combination of characters.
99 # The script will attempt to guess the password by trying every
100 # possible combination of characters.
```

5.2.1 Brute Force Attacks

In the following slides, we will discuss how *Brute Force* attacks work, their strengths, their weaknesses, and some tools that can be used to carry out a brute force attack.



5.2.1 Brute Force Attacks

Brute force attacks are the only way to be certain of finding someone's password. To automate a brute force attack, you have to write a program which generates every possible password.

Let's see how to implement a brute force algorithm!



5.2.1.1 Brute Force Algorithm

In the following slides, you will see how a brute force attack can be used to crack a password.

The algorithm proposed here works with a password of any length. It starts with one-character long passwords and increments the password size until a valid password is found.

5.2.1.1 Brute Force Algorithm

Here we see a pseudo-code version of a brute force password attack:



```
password_found = false
password_length = 1

while password_found == false
do
  while can_create_password_of_length(password_length)
  do
    password = create_password_of_length(password_length)
    if (hash(password) match attacked_hash)
    then
      password_found = true
    done
    password_length = password_length + 1
  done
```

5.2.1.1 Brute Force Algorithm

To find an unknown password of unknown length, the algorithm must cycle over every possible lowercase character.

a → b → c → d → e → ... → z

5.2.1.1 Brute Force Algorithm

It will then test uppercase characters.

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow \dots \rightarrow z$



$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow \dots \rightarrow Z$

5.2.1.1 Brute Force Algorithm

After cycling through uppercase characters, it will test numbers.

a → b → c → d → e → ... → z



A → B → C → D → E → ... → Z



0 → 1 → 2 → 3 → 4 → 5 → ... → 9

5.2.1.1 Brute Force Algorithm

Finally, it will cycle through special symbols.

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow \dots \rightarrow z$



$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow \dots \rightarrow Z$



$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \dots \rightarrow 9$



$! \rightarrow " \rightarrow \# \rightarrow \$ \rightarrow \% \rightarrow \& \rightarrow \dots \rightarrow @$

5.2.1.1 Brute Force Algorithm

If the password is not found, the password length increases to two characters. The algorithm then chooses a value for the first character, performing the same operations we have just seen on the second one.

aa → ab → ... → a1 → ... → a! → ... → a@

5.2.1.1 Brute Force Algorithm

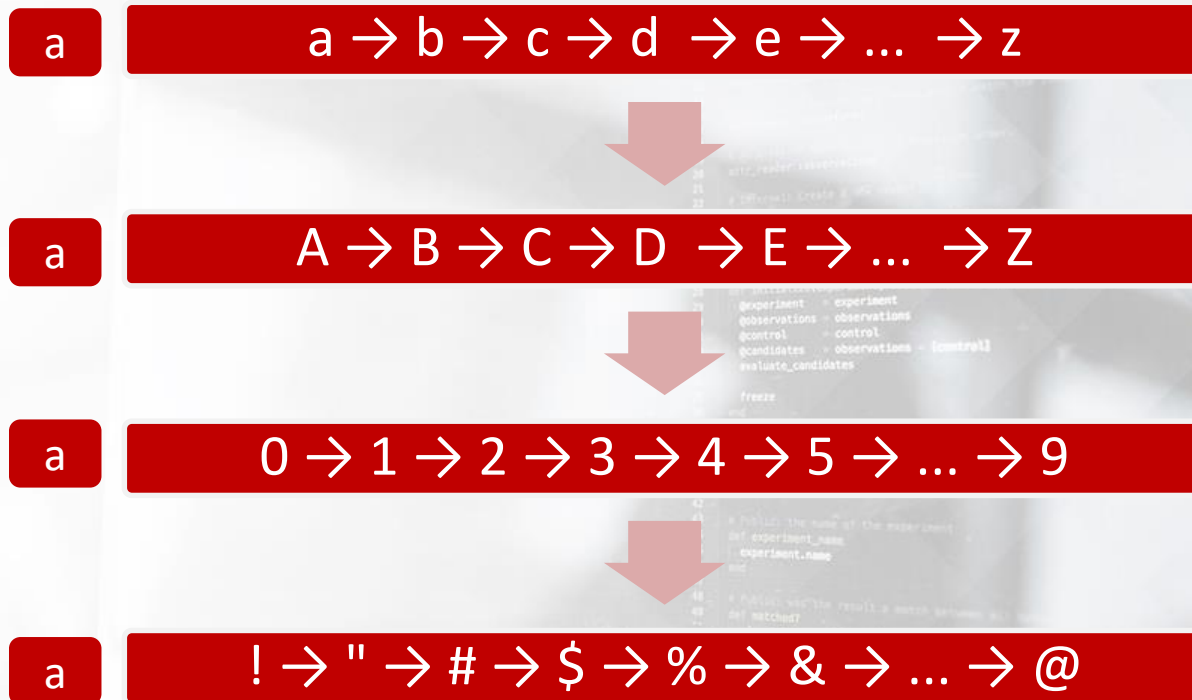
If no password is found, it changes the value of the first character and cycles again over the second one. This operation is performed for every possible value of the first character.

aa → ab → ... → ba → ... → z! → ... → @@

5.2.1.1 Brute Force Algorithm

The algorithm repeats.

Over lowercase characters...



5.2.1.1 Brute Force Algorithm

Uppercase
characters...

J

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow \dots \rightarrow z$



J

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow \dots \rightarrow Z$



J

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \dots \rightarrow 9$



J

$! \rightarrow " \rightarrow \# \rightarrow \$ \rightarrow \% \rightarrow \& \rightarrow \dots \rightarrow @$

5.2.1.1 Brute Force Algorithm

Numbers...

9

$a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow \dots \rightarrow z$



9

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow \dots \rightarrow Z$



9

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \dots \rightarrow 9$



9

$! \rightarrow " \rightarrow \# \rightarrow \$ \rightarrow \% \rightarrow \& \rightarrow \dots \rightarrow @$

5.2.1.1 Brute Force Algorithm

and symbols.

= $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow \dots \rightarrow z$



= $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow \dots \rightarrow Z$



= $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \dots \rightarrow 9$



= $! \rightarrow " \rightarrow \# \rightarrow \$ \rightarrow \% \rightarrow \& \rightarrow \dots \rightarrow @$

5.2.1.1 Brute Force Algorithm

If the password is longer than two characters, then the algorithm moves to passwords that are three characters long, then four and so on until a valid password is found! Eventually, it will try every possible combination of letters, numbers, and symbols.

Given **enough time**, a brute force attack is **always successful**!

5.2.1.2 Brute Forcing Weaknesses

Brute force attacks appear to be the definitive weapon for password cracking because they always crack a password. In real-world attack scenarios, brute force attacks are used only when other attack vectors fail.

- Why does this happen?
- What are the intrinsic weaknesses of brute forcing?
- Why doesn't every attacker just use the brute force method when cracking a password?

5.2.1.2 Brute Forcing Weaknesses

Brute force attacks cycle through every possible input combination. They start from a single character and escalate to n-characters long, complex passwords. The resulting password could be made up of combinations of many letters, numbers, and symbols.

Generating and testing all those passwords takes time, even for a computer! In fact, one of the major caveats of a brute force attack is the time it takes to find the password.

5.2.1.2 Brute Forcing Weaknesses

If the password is poorly chosen by the user (short and made just of letters), then cracking it could take a couple of minutes or a couple of hours. If a password is long (fifteen or more characters) and made by upper and lower case letters, numbers and symbols, cracking it could take days, or even years.

Choosing a proper password, means making a brute force attack unfeasible because of timing constraints.

5.2.1.3 John the Ripper

Although writing a script to implement a brute force attack is not a difficult task per se, some tools include helpful functions such as session saving and multi-threading support, as well as the ability to brute force against different password file formats.



5.2.1.3 John the Ripper

John the Ripper is an extremely popular password cracking tool written for Unix-based operating systems. Nowadays, you can find sources and binaries for Linux, Mac OSX and Windows on [Openwall website](http://www.openwall.com/john/).

John the Ripper can mount both brute force and dictionary-based attacks against a password database.

5.2.1.3 John the Ripper

You can use it on nearly one hundred encryption formats. You can check them out by using the `--list=formats` command line option:



```
# john --list=formats
des, bsdi, md5, bf, afs, lm, dynamic_n, bfegg, dmd5, dominosec, epi, hdaa,
ipb2, krb4, krb5, mschapy2, netlm, netlmv2, netntlm, netntlmv2, nethalflm,
md5ns, nt, phps, po, xsha, crc32, gost, keychain, lotus5, md4-gen, mediawiki,
mscash, mscash2, mskrb5, mssql, mssql05, mysql-sha1, mysql, nsldap, nt2, odf,
office, oracle11, oracle, osc, phpass, pix-md5, pkzip, racf, raw-md4,
raw-md5, raw-sha1, raw-sha1-linkedin, raw-md5u, salted-sha1, sapb, sapg,
sha1-gen, sip, vnc, wbb3, hmac-md5, hmac-sha1, raw-sha, raw-sha224,
raw-sha256, raw-sha384, raw-sha512, hmac-sha224, hmac-sha256, hmac-sha384,
hmac-sha512, xsha512, hmailserver, sybasease, dragonfly3-64, dragonfly4-64,
dragonfly3-32, dragonfly4-32, drupal7, sha256crypt, sha512crypt, episerver,
keepass, pwsafe, django, raw-sha1-ng, crypt, trip, ssh, pdf, wpapsk, rar,
zip, dummy
```



5.2.1.3 John the Ripper

The tool is extremely fast because of the high use of parallelization. It can also use many different cracking strategies during a brute force attack, and you can specify different password character sets, such as letters only or numbers only.

In the following slides, we will see how to use *John the Ripper* during a cracking session.



5.2.1.3 John the Ripper

In this scenario, after getting unauthorized access to a Linux machine, we made a copy of the password files:

- `/etc/passwd` that contains information about user accounts
- `/etc/shadow` that contains the actual password hashes

We want to use John the Ripper to crack some passwords.

5.2.1.3.1 Unshadow

John needs the username and the password hashes to be in the same file, so we need to use the *unshadow* utility. *Unshadow* comes with *John the Ripper*.



```
# unshadow passwd shadow > crackme
```

The actual cracking process can now start on the *crackme* file.

5.2.1.3.2 Brute force with John the Ripper

To brute force the password of the *victim* user, you have to type:

Example:



```
# john -incremental -users:victim crackme
```

5.2.1.3.2 Brute force with John the Ripper

Here you can see the John the Ripper output after finishing a simple cracking session:



```
# john -incremental -users:victim crackme
Created directory: /root/.john
Loaded 1 password hash (sha512crypt [64/64])
test                (victim)
guesses: 1  time: 0:00:00:28 DONE (Sun Feb 15 15:36:43 2015)  c/s: 856
trying: tomr - tina
Use the "--show" option to display all of the cracked passwords
reliably
```



5.2.1.3.2 Brute force with John the Ripper

To display the passwords recovered by *John*, you can use the `--show` option:



```
# john --show crackme  
victim:test:1001:1003:,,,:/home/victim:/bin/bash
```

5.2.1.3.2 Brute force with John the Ripper

Most cracking sessions take hours or days to complete. During a long cracking session, you can press any key to show the status of *john*:

```
</>
# john -incremental -users:victim crackme
Loaded 1 password hash (sha512crypt [64/64])
guesses: 0   time: 0:00:00:02 0.00%   c/s: 718   trying: 0101975 - 0100091
guesses: 0   time: 0:00:00:08 0.00%   c/s: 822   trying: andy21 - arc130
guesses: 0   time: 0:00:00:17 0.00%   c/s: 848   trying: chattest -
cheres96
```


5.2.1.3.2 Brute force with John the Ripper

Cracking a very long and complex password by brute force could take years, even on a very powerful machine. The time needed to crack a password is the only way to keep it safe from brute force attacks.

Keep this in mind when choosing a password or setting up a brute force attack!

```
18 # Initialize the experiment's observations
19 # Initialize the experiment's candidates
20 # Initialize the experiment's context
21 # Initialize the experiment's result
22 # Initialize the experiment's result
23 # Initialize the experiment's result
24 # Initialize the experiment's result
25 # Initialize the experiment's result
26 # Initialize the experiment's result
27 # Initialize the experiment's result
28 # Initialize the experiment's result
29 # Initialize the experiment's result
30 # Initialize the experiment's result
31 # Initialize the experiment's result
32 # Initialize the experiment's result
33 # Initialize the experiment's result
34 # Initialize the experiment's result
35 # Initialize the experiment's result
36 # Initialize the experiment's result
37 # Initialize the experiment's result
38 # Initialize the experiment's result
39 # Initialize the experiment's result
40 # Initialize the experiment's result
41 # Initialize the experiment's result
42 # Initialize the experiment's result
43 # Initialize the experiment's result
44 # Initialize the experiment's result
45 # Initialize the experiment's result
46 # Initialize the experiment's result
47 # Initialize the experiment's result
48 # Initialize the experiment's result
49 # Initialize the experiment's result
50 # Initialize the experiment's result
51 # Initialize the experiment's result
52 # Initialize the experiment's result
53 # Initialize the experiment's result
54 # Initialize the experiment's result
55 # Initialize the experiment's result
56 # Initialize the experiment's result
57 # Initialize the experiment's result
58 # Initialize the experiment's result
59 # Initialize the experiment's result
60 # Initialize the experiment's result
61 # Initialize the experiment's result
62 # Initialize the experiment's result
63 # Initialize the experiment's result
64 # Initialize the experiment's result
65 # Initialize the experiment's result
66 # Initialize the experiment's result
67 # Initialize the experiment's result
68 # Initialize the experiment's result
69 # Initialize the experiment's result
70 # Initialize the experiment's result
71 # Initialize the experiment's result
72 # Initialize the experiment's result
73 # Initialize the experiment's result
74 # Initialize the experiment's result
75 # Initialize the experiment's result
76 # Initialize the experiment's result
77 # Initialize the experiment's result
78 # Initialize the experiment's result
79 # Initialize the experiment's result
80 # Initialize the experiment's result
81 # Initialize the experiment's result
82 # Initialize the experiment's result
83 # Initialize the experiment's result
84 # Initialize the experiment's result
85 # Initialize the experiment's result
86 # Initialize the experiment's result
87 # Initialize the experiment's result
88 # Initialize the experiment's result
89 # Initialize the experiment's result
90 # Initialize the experiment's result
91 # Initialize the experiment's result
92 # Initialize the experiment's result
93 # Initialize the experiment's result
94 # Initialize the experiment's result
95 # Initialize the experiment's result
96 # Initialize the experiment's result
97 # Initialize the experiment's result
98 # Initialize the experiment's result
99 # Initialize the experiment's result
100 # Initialize the experiment's result
```

5.2.2 Dictionary Attacks

Now that we have seen how brute force attacks work, we can study a generally faster attack: **dictionary attacks**.

Dictionary attacks do not try to generate every possible password, but they use a **dictionary of common passwords**, testing every single entry it contains.

```
def generate_candidates(passwords):
    candidates = []
    for password in passwords:
        candidates.append(password)
    return candidates

def generate_candidates_from_file(filename):
    passwords = []
    with open(filename, 'r') as f:
        for line in f:
            passwords.append(line.strip())
    return generate_candidates(passwords)

def generate_candidates_from_web(url):
    passwords = []
    response = requests.get(url)
    for line in response.text.splitlines():
        passwords.append(line.strip())
    return generate_candidates(passwords)

def generate_candidates_from_all_sources():
    candidates = []
    candidates.extend(generate_candidates_from_file('passwords.txt'))
    candidates.extend(generate_candidates_from_web('https://www.example.com/passwords.txt'))
    return candidates

def generate_candidates_from_all_sources():
    candidates = []
    candidates.extend(generate_candidates_from_file('passwords.txt'))
    candidates.extend(generate_candidates_from_web('https://www.example.com/passwords.txt'))
    return candidates
```

5.2.2 Dictionary Attacks

Dictionary attacks are faster than pure brute force attacks because even a large dictionary, in order of magnitude, is smaller than the number of possible valid passwords for an account.



5.2.2 Dictionary Attacks

The number of commonly used 8-character long passwords is **3,051,366**, while the number of valid 8-character long lowercase passwords is **208,827,064,576**.

208 billion vs. 3 million passwords to test during an attack!!!



5.2.2.1 Performing a Dictionary Attack

To carry out a dictionary attack, you need:

- A password file containing the hashed passwords to crack.
- A dictionary, or wordlist, of passwords.
- A tool to test every password in the wordlist against the password file.

5.2.2.1 Performing a Dictionary Attack

During a dictionary attack, the password recovery tool used simply tests every entry in the wordlist.

Wordlists usually contain commonly used passwords such as "admin", "password1234" or "trustNO1".

```
23 # The password to be tested
24 # The password to be tested
25 # The password to be tested
26 # The password to be tested
27 # The password to be tested
28 # The password to be tested
29 # The password to be tested
30 # The password to be tested
31 # The password to be tested
32 # The password to be tested
33 # The password to be tested
34 # The password to be tested
35 # The password to be tested
36 # The password to be tested
37 # The password to be tested
38 # The password to be tested
39 # The password to be tested
40 # The password to be tested
41 # The password to be tested
42 # The password to be tested
43 # The password to be tested
44 # The password to be tested
45 # The password to be tested
46 # The password to be tested
47 # The password to be tested
48 # The password to be tested
49 # The password to be tested
50 # The password to be tested
51 # The password to be tested
52 # The password to be tested
53 # The password to be tested
54 # The password to be tested
55 # The password to be tested
56 # The password to be tested
57 # The password to be tested
58 # The password to be tested
59 # The password to be tested
60 # The password to be tested
61 # The password to be tested
62 # The password to be tested
63 # The password to be tested
64 # The password to be tested
65 # The password to be tested
66 # The password to be tested
67 # The password to be tested
68 # The password to be tested
69 # The password to be tested
70 # The password to be tested
71 # The password to be tested
72 # The password to be tested
73 # The password to be tested
74 # The password to be tested
75 # The password to be tested
76 # The password to be tested
77 # The password to be tested
78 # The password to be tested
79 # The password to be tested
80 # The password to be tested
81 # The password to be tested
82 # The password to be tested
83 # The password to be tested
84 # The password to be tested
85 # The password to be tested
86 # The password to be tested
87 # The password to be tested
88 # The password to be tested
89 # The password to be tested
90 # The password to be tested
91 # The password to be tested
92 # The password to be tested
93 # The password to be tested
94 # The password to be tested
95 # The password to be tested
96 # The password to be tested
97 # The password to be tested
98 # The password to be tested
99 # The password to be tested
100 # The password to be tested
```

5.2.2.2 Weaknesses of Dictionary Attacks

Poorly chosen or default passwords are more exposed to dictionary cracking.

Uncommon passwords could be safe, but only a truly random and long password can be considered safe.

```
24 # @param candidates - the list of candidates to be evaluated
25 # @param observations - an array of Observations, in which
26 # @param control - the control observation
27
28 def skillRanking(experiment, observations = [], control = null)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = candidates
33   evaluate_candidates
34
35   freeze
36   @control =
37     experiment.context
38     experiment.name
39     experiment.name
40   end
41
42   # @param result - the result of the match between the candidates
43   # @param match - the result of the match between the candidates
44   # @param result - the result of the match between the candidates
45   # @param result - the result of the match between the candidates
46   # @param result - the result of the match between the candidates
47   # @param result - the result of the match between the candidates
48   # @param result - the result of the match between the candidates
49   # @param result - the result of the match between the candidates
50   # @param result - the result of the match between the candidates
51   # @param result - the result of the match between the candidates
52   # @param result - the result of the match between the candidates
53   # @param result - the result of the match between the candidates
54   # @param result - the result of the match between the candidates
55   # @param result - the result of the match between the candidates
56   # @param result - the result of the match between the candidates
57   # @param result - the result of the match between the candidates
58   # @param result - the result of the match between the candidates
59   # @param result - the result of the match between the candidates
60   # @param result - the result of the match between the candidates
61   # @param result - the result of the match between the candidates
62   # @param result - the result of the match between the candidates
63   # @param result - the result of the match between the candidates
64   # @param result - the result of the match between the candidates
65   # @param result - the result of the match between the candidates
66   # @param result - the result of the match between the candidates
67   # @param result - the result of the match between the candidates
68   # @param result - the result of the match between the candidates
69   # @param result - the result of the match between the candidates
70   # @param result - the result of the match between the candidates
71   # @param result - the result of the match between the candidates
72   # @param result - the result of the match between the candidates
73   # @param result - the result of the match between the candidates
74   # @param result - the result of the match between the candidates
75   # @param result - the result of the match between the candidates
76   # @param result - the result of the match between the candidates
77   # @param result - the result of the match between the candidates
78   # @param result - the result of the match between the candidates
79   # @param result - the result of the match between the candidates
80   # @param result - the result of the match between the candidates
81   # @param result - the result of the match between the candidates
82   # @param result - the result of the match between the candidates
83   # @param result - the result of the match between the candidates
84   # @param result - the result of the match between the candidates
85   # @param result - the result of the match between the candidates
86   # @param result - the result of the match between the candidates
87   # @param result - the result of the match between the candidates
88   # @param result - the result of the match between the candidates
89   # @param result - the result of the match between the candidates
90   # @param result - the result of the match between the candidates
91   # @param result - the result of the match between the candidates
92   # @param result - the result of the match between the candidates
93   # @param result - the result of the match between the candidates
94   # @param result - the result of the match between the candidates
95   # @param result - the result of the match between the candidates
96   # @param result - the result of the match between the candidates
97   # @param result - the result of the match between the candidates
98   # @param result - the result of the match between the candidates
99   # @param result - the result of the match between the candidates
100  # @param result - the result of the match between the candidates
```


5.2.2.2 Weaknesses of Dictionary Attacks

"rXQqCI1\9_DhwX9RH\zF" or
"rN9QmGAUC4WnMdUgUSsdVlghX" are safe
passwords, but they are not easy to remember, so
you need a tool, such as Keepass, to securely
store them. **Most users are lazy and do not do
that!**



5.2.2.2 Weaknesses of Dictionary Attacks

According to <https://howsecureismypassword.net/>, the previous passwords are also secure from pure brute force attacks.

A brute force attack against each of those passwords would take respectively:

- 35 sextillion ($35 * 10^{21}$ years) for the first one
- 5 octillion ($5 * 10^{27}$ years) for the second one

5.2.2.2 Weaknesses of Dictionary Attacks

However, this does not mean that only a strong password can resist a dictionary attack. It simply could not be included in the dictionary!

If the password is not in the wordlist, the tool performing a dictionary attack will not be able to crack the password.

5.2.2.3 Mangling words

Being unable to crack a password just because of case differences or some digits at the end of a word would be a shame. Because of that, cracking tools provide some options to **mangle** the words in a dictionary.

Example:

Some variations on "cat" could be: cat12, caT, CAT, Cat, CA@, c@t and so on...

```
def initialize(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations -> uniq
  evaluate_candidates

  freeze
end

# Returns the experiment's context
def context
  @experiment.context
end

# Returns the results of the experiment
def results
  @experiment.results
end

# Returns whether the results match the control
def matches?
  @control == @results
end

# Returns the result of the experiment
def result
  @results[0]
end
```

5.2.2.4 Dictionary Attacks with John the Ripper

You can run dictionary attacks with *John* by passing it the `-wordlist` argument. You can also specify a custom wordlist.



```
$ john -wordlist<=custom wordlist file> <file to crack>
```

5.2.2.4 Dictionary Attacks with John the Ripper

You can also apply some mangling by using the `-rules` parameter:



```
$ john -wordlist<=custom wordlist file> -rules <file to crack>
```

5.2.2.4 Dictionary Attacks with John the Ripper

In this example, we want to crack the *crackme* file by using the *John* default wordlist:



```
# john -wordlist -users=victim,victim2 crackme
Loaded 2 password hashes with 2 different salts
(sha512crypt [64/64])
guesses: 0  time: 0:00:00:05 DONE (Mon Feb 16
17:28:00 2015)  c/s: 1415  trying: paagal - sss
```



5.2.2.4 Dictionary Attacks with John the Ripper

If the default wordlist does not work, you can use a custom one. We first check the content of the custom wordlist and then use it with *John*:



```
# cat mywordlist
mysteryguy
MEGAPASSWORD

# john -wordlist=mywordlist -users=victim,victim2 crackme
Loaded 2 password hashes with 2 different salts (sha512crypt [64/64])
mysteryguy          (victim)
guesses: 1  time: 0:00:00:00 DONE (Mon Feb 16 17:29:43 2015)  c/s: 25.00  trying:
mysteryguy - MEGAPASSWORD

Use the "--show" option to display all of the cracked passwords reliably
```



5.2.2.4 Dictionary Attacks with John the Ripper

As a last resort, you can enable dictionary mangling:

```
</>
# cat mywordlist
mysteryguy
MEGAPASSWORD
# john -wordlist=mywordlist -rules -users=victim,victim2 crackme
Loaded 2 password hashes with 2 different salts (sha512crypt [64/64])
megapassword (victim2)
mysteryguy (victim)
guesses: 2 time: 0:00:00:00 DONE (Mon Feb 16 17:36:07 2015) c/s: 512 trying:
mysteryguy - Megapassword7
Use the "--show" option to display all of the cracked passwords reliably
```

5.2.2.5 Installing Password dictionaries

You can find some useful password dictionaries as part of the [OWASP Seclists Project](https://github.com/danielmiessler/SecLists) on [GITHub](https://github.com). If you use Kali Linux, you can install the `seclists` package.



```
# apt-get install seclists
```

After installing it, you will find the dictionaries in:
`/usr/share/seclists/Passwords/`

5.2.3 Rainbow Tables

There is another very clever way to crack passwords: **rainbow tables**.

Rainbow tables offer a tradeoff between the processing time needed to calculate the hash of a password and the storage space needed to mount an attack.

5.2.3 Rainbow Tables

Discussing in detail the mathematical principles behind rainbow tables is out of the scope of this course.

To understand how the attack works you only need to know that a rainbow table is a table containing links between the results of a run of one hashing function and another.



5.2.3 Rainbow Tables

The space needed to store a rainbow table depends on how many characters are allowed in a password (upper and lower case characters, digits, symbols) and its length. Moreover, the specific hash function used to store the password plays a role.

Rainbow tables' sizes vary from hundreds of megabytes to hundreds of gigabytes.



5.2.3 Rainbow Tables

By performing a lookup in the rainbow tables cracking tools will save the computational time needed to hash every candidate password.

This approach reduces a cracking session time from days to seconds!

```
def rainbow_tables(experiment, observations = [], control = null)
  @experiment = experiment
  @observations = observations
  @control = control
  @candidates = observations - {control}
  evaluate_candidates

  freeze
end

# Build the experiment's rainbow
def rainbow(experiment, observations = [], control = null)
  # Build the name of the experiment
  def experiment_name
    experiment.name
  end

  # Build the result a match between an experiment and a password
  def match?
    # ...
  end

  # Build the rainbow
  rainbow = {}
  rainbow[experiment_name] = rainbow_tables(experiment, observations, control)
end
```


5.2.3.1 Rainbow Tables Limitations

The limit of rainbow tables is the storage space needed to guarantee successful cracking sessions.

Example:

A rainbow table that guarantees a 96.8% success rate when cracking a password that's length ranges from 1 to 9 characters and hashed with *MD5* weights 864GB!

Furthermore, they can be a great resource to crack simple and complex short password (5 - 8 character long passwords).

5.2.3.2 Ophcrack

A great tool to perform rainbow cracking is ***ophcrack***. It is a tool aimed at Windows password recovery, so you can use it only to crack Windows authentication passwords.

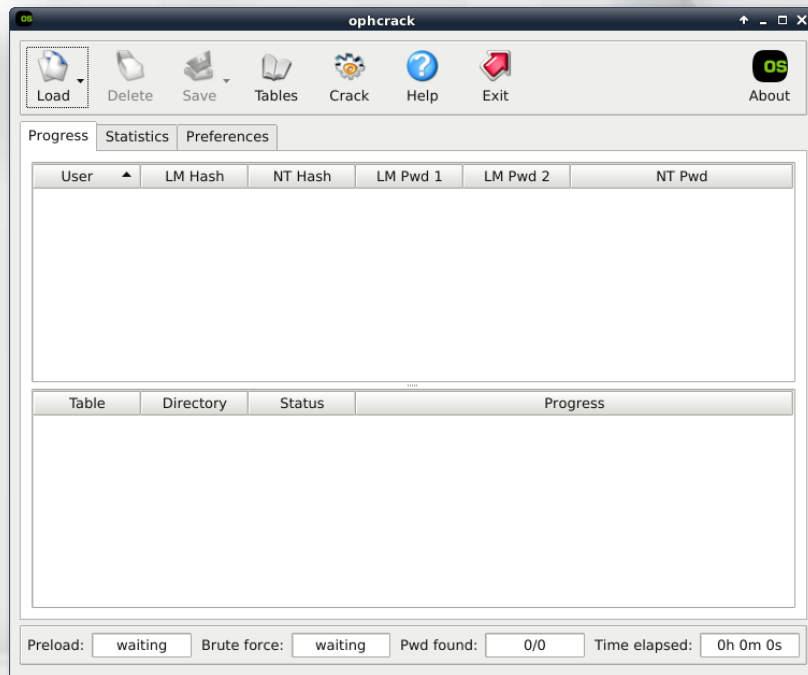
On its site, you will also find some **free rainbow tables**, where its size ranges can be from 300MB to 2TB.



5.2.3.2 Ophcrack

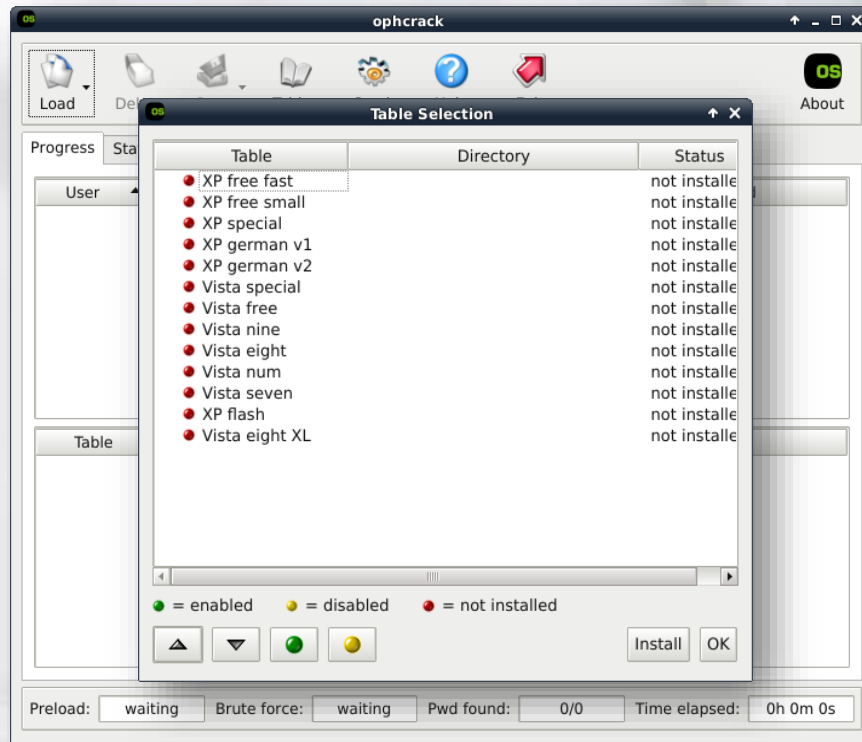
Ophcrack can run on Windows, Linux, Unix, and OSX.

Here we see the main window.



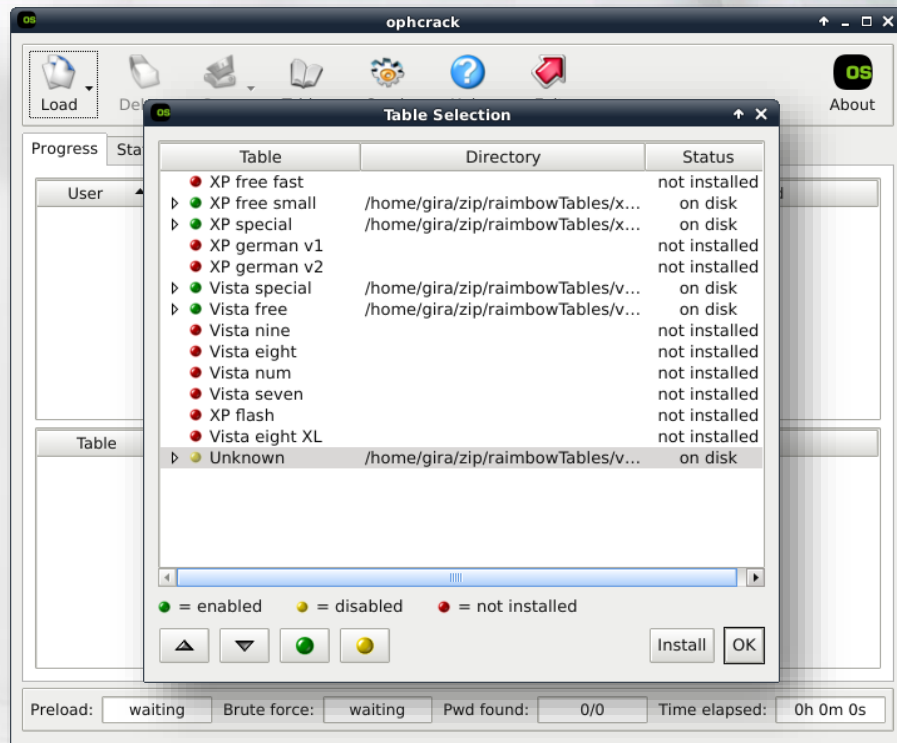
5.2.3.2 Ophcrack

To start cracking passwords, it is just a matter of clicking *Tables* in the main window and then selecting install.



5.2.3.2 Ophcrack

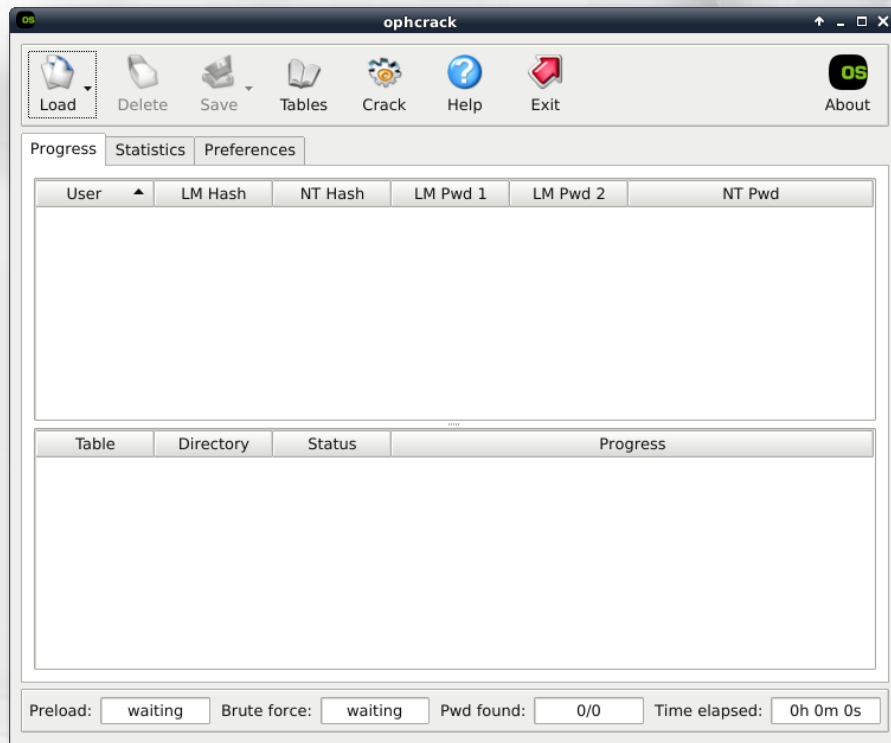
You can install free or purchased tables and do not need to install them all.



5.2.3.2 Ophcrack

You can then load a password file by using the *Load* button in the main window.

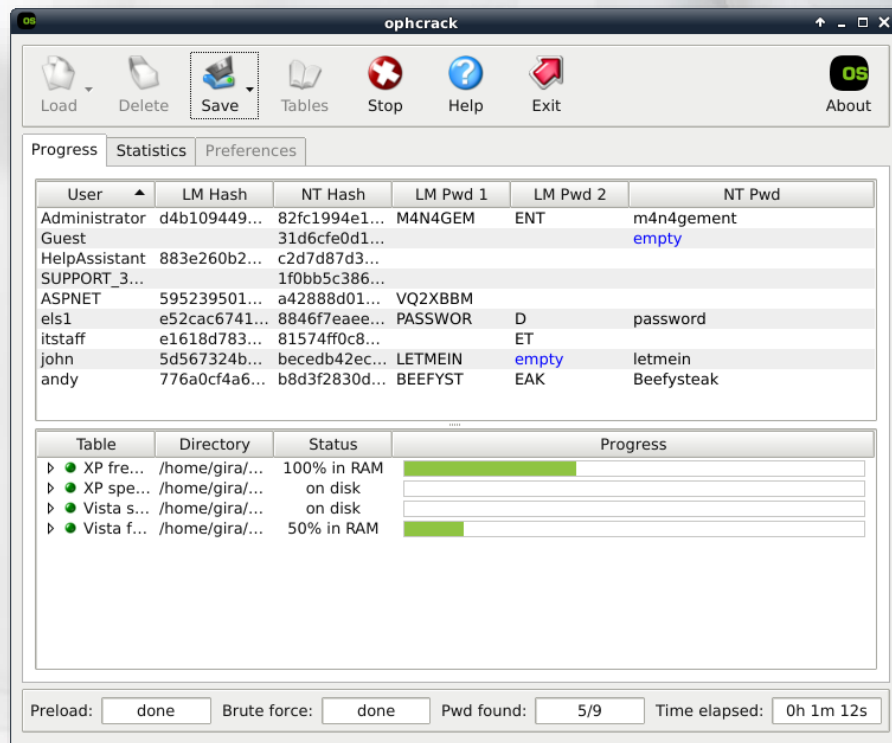
You will see how to get a password file from an exploited machine in the *Meterpreter* video at the end of this module.



5.2.3.2 Ophcrack

Finally, you click on the *Crack* button to start the process.

According to the tables you installed and the encryption format, you will be able to recover some or all the passwords!



5.2.4 Video – John the Ripper

John the Ripper

In this video, you will see how to use *unshadow* and *John the Ripper* to perform password cracking attacks against an exploited machine.



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

5.2.5 Hashcat Video

Hashcat

In this video, you will see how to use hashcat for password cracking using GPU on windows.



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

5.2.6 Conclusion

Password cracking is one of the ways to maintain access to a compromised machine. After getting access to a machine, a penetration tester can dump the password database and then crack it. This activity lets them access the machine by using valid credentials, thus being able to make their control over the machine persistent.



5.2.6 Conclusion

You can also apply the techniques you saw in this chapter to other password resources, like some credentials you dumped via SQL injection or a password file backup you somehow stole from a system.



Buffer Overflow Attacks



5.3 Buffer Overflow Attacks

How does this support my pentesting career?

- Remote code execution
- Privilege escalation attacks
- Understanding basics of memory corruption attacks.

5.3 Buffer Overflow Attacks

Many different attacks widely exploit **buffer overflow vulnerabilities**. They work by taking control of the execution flow of a piece of software or a routine of the operating system.

Taking control of the execution of a program means being able to force it to behave differently compared to what the application author designed.

5.3 Buffer Overflow Attacks

A buffer overflow attack can lead to:

- An application or operating system crash, thus causing a denial of service
- Privilege escalation
- Remote code execution
- Security features bypass

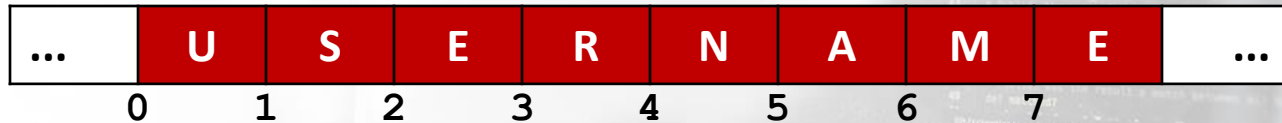
In the following slides, you will see how a buffer overflow vulnerability can arise and how a skilled attacker can exploit it.

5.3.1 Buffers

Buffers have a **finite size**; this means that they can only contain a certain amount of data.

Example:

If a client-server application is designed to accept only 8-characters long usernames, the username buffer will be 8 bytes long.



Other data in RAM

Other data in RAM

5.3.1 Buffers

If the developer of an application does not enforce buffers' limits, an attacker could find a way to write data beyond those limits, thus actually writing arbitrary code in the computer RAM; this can be exploited to get control over the program execution flow!



5.3.1.1 Buffer Overflow Example

A developer creates a text editor. By design the maximum length of a single line is 256 characters, so the editor does not accept user input that creates lines longer than 256 characters.

A penetration tester discovered that the application does not enforce this restriction when opening a file created with another editor. Moreover, when the editor opens a file, it inserts the first line in the line buffer.



5.3.1.1 Buffer Overflow Example

So if the penetration tester creates a file with just a single line made of, let's say, 512 random characters and opens it with the editor, the application crashes! This means that the data in the file somewhat overwrote some of the editor code that is loaded in RAM.

The penetration tester then writes a script that first generates files with very long lines and then opens them with the application.

5.3.1.1 Buffer Overflow Example

After some trial and error, the penetration tester is able to generate files that, when opened with the editor, overwrite the program execution flow with valid code, giving the pentester control over the application!

This example should have given you a basic idea of how buffer overflow attacks work. Let's delve into them more!

5.3.2 The Stack

Buffers are stored in a special data structure in the computer memory called a stack.

A **stack** is a data structure used to store data.



5.3.2 The Stack

You can imagine a stack as a pile of plates where you can add or remove just one plate at a time; this means that you can only add a plate on the top of the pile or remove a plate from the top of the pile.



5.3.2 The Stack

This approach is called **Last in First Out (LIFO)** and uses two methods:

- **Push**, which adds an element to the stack
- **Pop** removes the last inserted element



5.3.2.1 Push Operation

Push adds elements on the top of the stack.

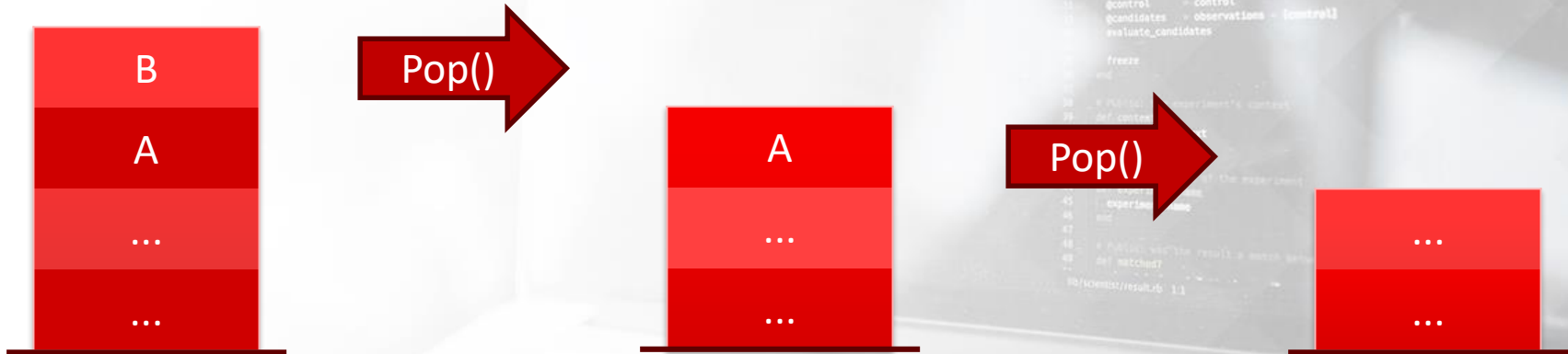
If we add A and then B to the stack, it will change according to the following picture:



5.3.2.2 Pop Operation

Pop removes an element from the top of the stack. The element removed from the stack can be used by the software routine which popped it.

Example:



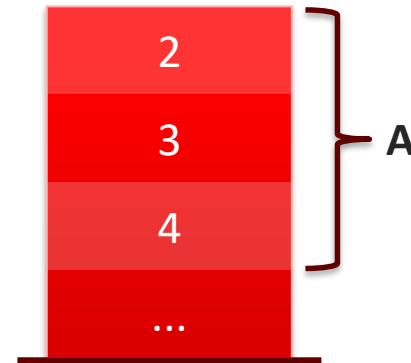
EXAMPLE

5.3.2.3 Allocating Space on the Stack

If an application needs to store an array of three integers, it can just allocate three positions on the stack.

The application code, in C, could be:

```
int A[3] = {2, 3, 4};
```



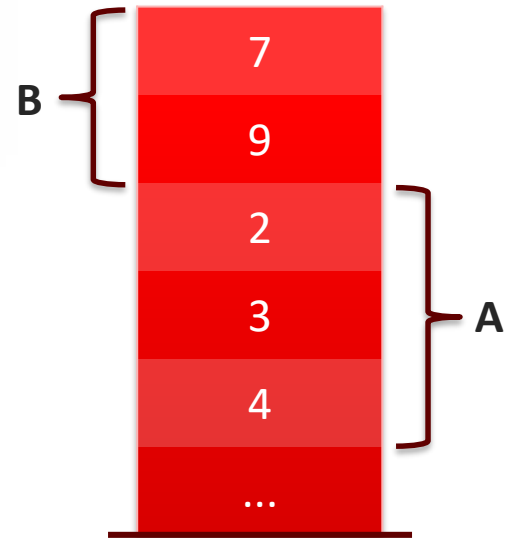
EXAMPLE

5.3.2.3 Allocating Space on the Stack

Allocating two arrays:

```
int A[3] = {2, 3, 4};
```

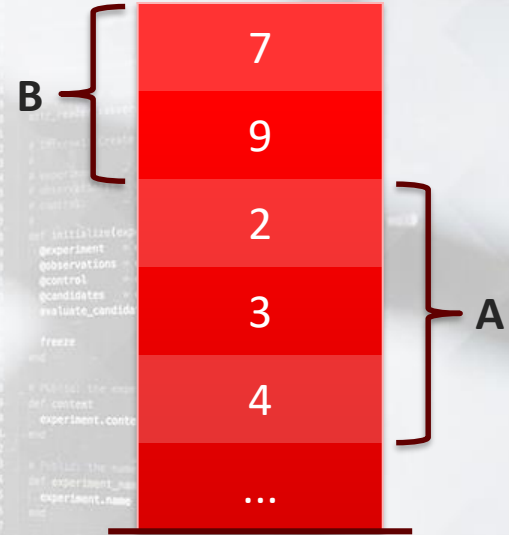
```
int B[2] = {7, 9};
```



5.3.2.4 Overflows in the Stack

The previous examples show how the stack grows bottom-up, while variables are written top-down. So, referring to the previous example, what happens if an attacker writes in *B* more than two integers?

They will be able to overwrite *A*!!!

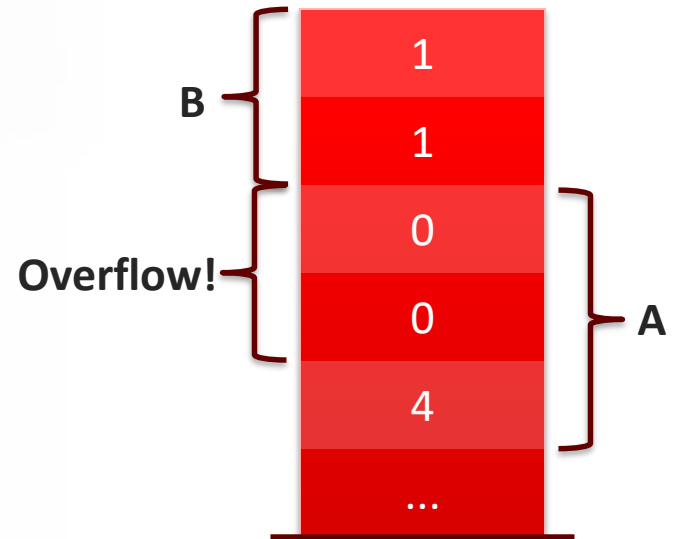


5.3.2.4 Overflows in the Stack

If an attacker finds a way to copy over *B* a **four-elements array**, they will overwrite a part of *A*.

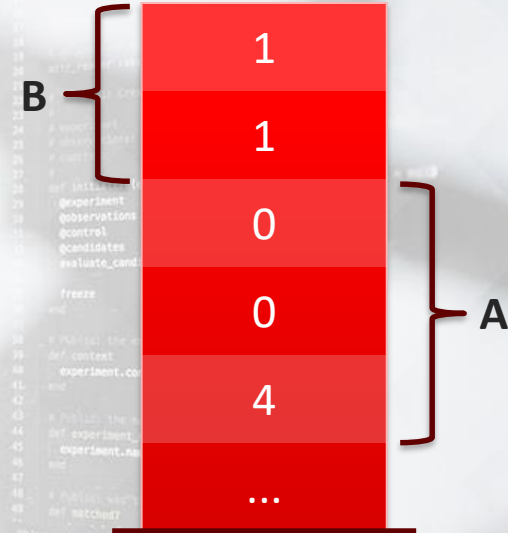
The picture shows an overflow performed by copying over *B* the following array:

```
int overflow[4] = {1, 1, 0, 0};
```



5.3.2.4 Overflows in the Stack

After the attacker exploited the buffer overflow vulnerability, the application will see A content as {0, 0, 4}!



5.3.3 The Stack in an Application

So overwriting a function return address means getting control over the application. Moreover, if an attacker manages to write some valid code in RAM, they can force the victim function to run their code.

A raw overflow that just overwrites some memory locations will crash the application, while a well-engineered attack is able to execute code on the victim machine.

5.3.4 How Buffer Overflow Attacks Work

Deep analysis about how the stack works in an operating system is out of the scope of this course.



A vertical stack of red rectangular boxes representing memory layout. From top to bottom, the boxes are labeled: 'Local Variable 2', 'Local Variable 1', 'Base Pointer', 'Return Address', and 'Function Parameters'. Below the 'Function Parameters' box is a box containing three dots '...'. The entire stack is positioned on the right side of the slide, with a faint background image of a laptop and code.

Local Variable 2

Local Variable 1

Base Pointer

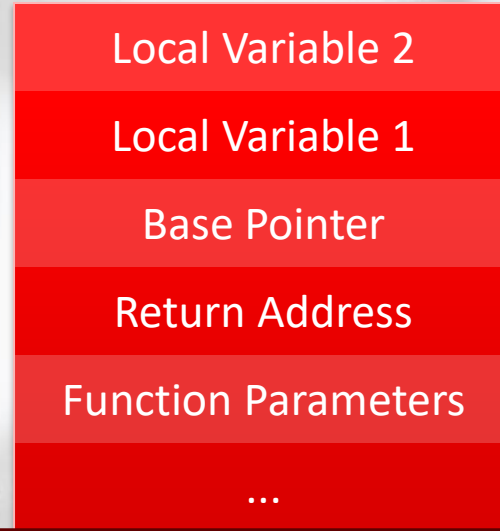
Return Address

Function Parameters

...

5.3.4 How Buffer Overflow Attacks Work

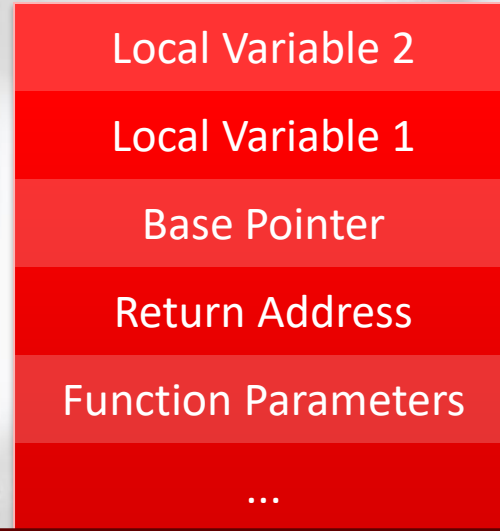
Looking at this picture you should understand that if an attacker manages to overflow *Local Variable 1*, they are able to overwrite *Base Pointer* and then ***Return address!***



5.3.4 How Buffer Overflow Attacks Work

If they overwrite *Return Address* with the right value, they are able to control the execution flow of the program!

This technique can be exploited by writing custom tools and applications or by using hacking tools such as *Metasploit*.



5.3.4 How Buffer Overflow Attacks Work

In the next module about network attacks, you will see a real-world buffer overflow exploitation with *Metasploit*!

Local Variable 2

Local Variable 1

Base Pointer

Return Address

Function Parameters

...



5.3.4 How Buffer Overflow Attacks Work

Being able to write a buffer overflow exploit requires a deep understanding of assembly programming, how applications and operating systems work and some exotic programming skills.

If you are interested in that, you can check out the eLearnSecurity Penetration Testing Professional Course!



References



References

[Hacking Exposed Malware & Rootkits, Second Edition](#)

http://www.amazon.com/Hacking-Exposed-Malware-amp-Rootkits/dp/0071823077/ref=sr_1_3?s=books&ie=UTF8&qid=1423220013&sr=1-3

[Keepass](#)

<http://keepass.info/>

[John the Ripper](#)

<http://www.openwall.com/john/>

[OWASP Seclists Passwords](#)

<https://github.com/danielmiessler/SecLists/tree/master/Passwords>



References

Malware samples

<http://www.offensivecomputing.net/?q=taxonomy/term/1>

KeepassX

<http://www.keepassx.org/>

How Secure is my Password?

<https://howsecureismypassword.net/>

Ophcrack

<http://ophcrack.sourceforge.net/>



Videos

Backdoors

In this video, you will see how to use different tools to create a backdoor on an exploited machine. Every backdoor has its own unique features. So, you will learn how to choose the right tool for the job!

John the Ripper

In this video, you will see how to use *unshadow* and *John the Ripper* to perform password cracking attacks against an exploited machine.

**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*

Videos

Hashcat

In this video, you will see how to use hashcat for password cracking using GPU on windows.



**Videos are only available in Full or Elite Editions of the course. To upgrade, click [HERE](#). To access, go to the course in your members area and click the resources drop-down in the appropriate module line.*