University of BRISTOL

Systems Security

COMSM1500

Office out of order
(sorry was only told the night before)

Coursework questions

- Open slack (one channel per lab)
  - https://bris-sys-sec.slack.com/
  - Did someone already asked the same question?
  - Can I understand the answer?
- Ask your question on slack
- Come for face to face support

- Groups are on blackboard (final I hope)

Next week two guest lectures

- Tuesday: Cloud Computing and Data Deletion
  - Marvin Kopo, Bristol Cyber Security Group
- Friday: Security and Machine Learning
  - Joe Gardiner, Bristol Cyber Security Group

Buffer overflow

Continued…

countermeasures

prevent

detect

recover

Detecting

Buffer overflow

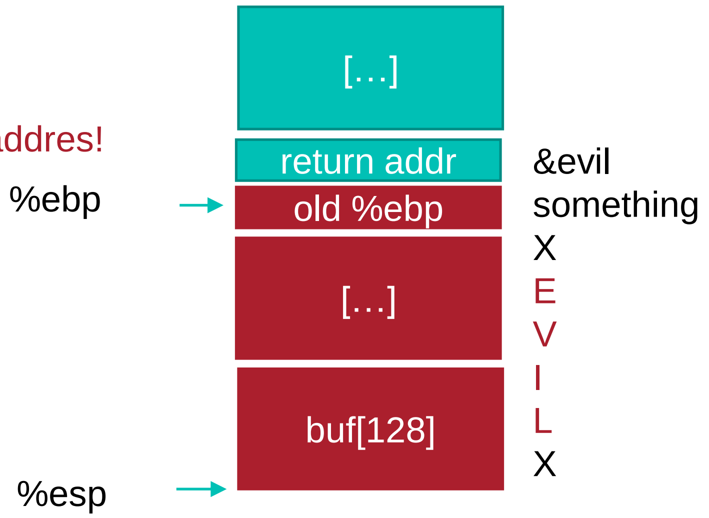
bristol.ac.uk

Example

```
▪ int read_get(void) {
▪   char buf[128];
▪   int i;
▪   gets(buf);
▪   i = atoi(buf);
▪   return I;
▪ }

▪ int main() {
▪   x = read_get();
▪   printf(“%s”, x);
▪ }
```

Changed returned addres! and old ebp.



Buffer overflow exploit

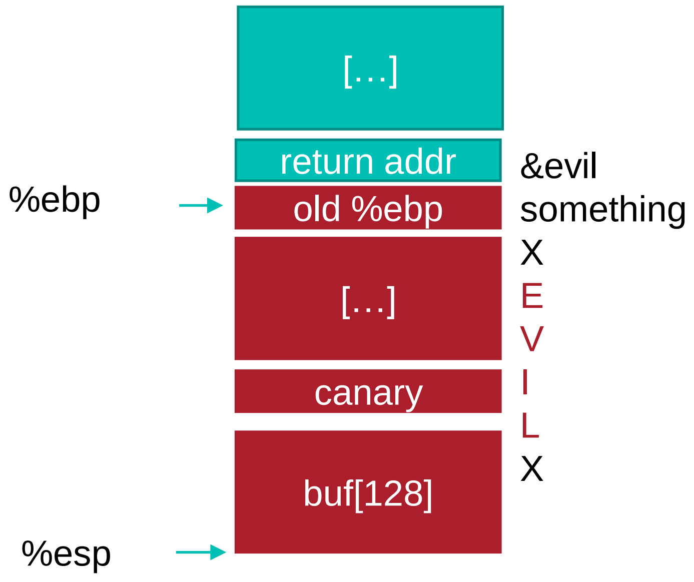
- Gaining control over the instruction pointer
  - i.e. changing return address
  - control what will be executed
- Make that pointer points to malicious code
  - embedding code (e.g. shell code last time)
  - jumping to unexpected part of code (i.e. open door)
- Gain control over stack pointer
  - i.e. control data

Stack canaries

- Let attacker overwrite stack

Stack canaries

- Let attacker overwrite stack
- Before return
  - Check the value of the canary
  - If it changed something bad happened
  - Compiler support

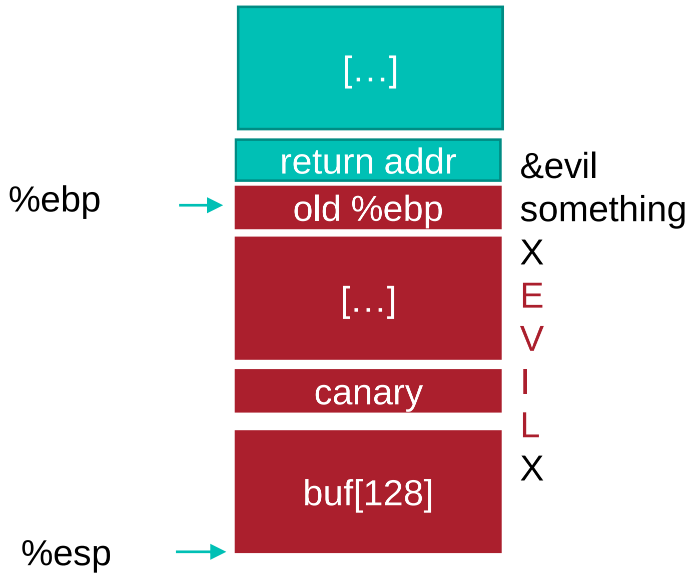


University of BRISTOL

Problem?

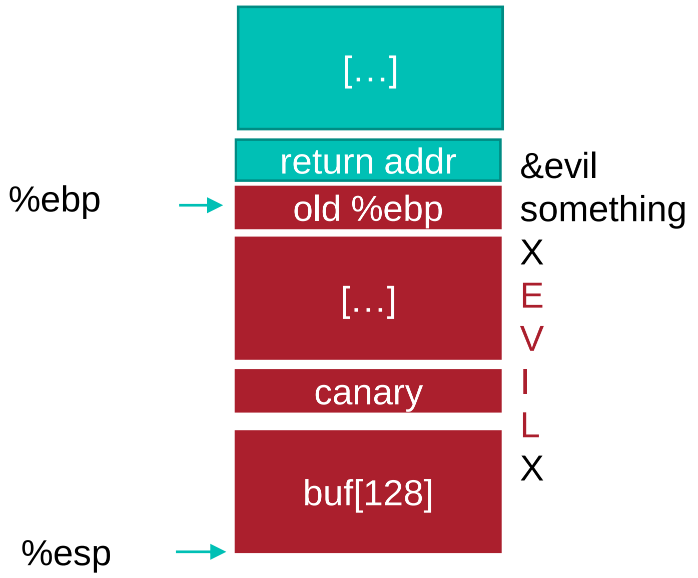
Stack canaries

- Let attacker overwrite stack
- Before return
- Careful about canary value
  - if deterministic can be guessed and avoided



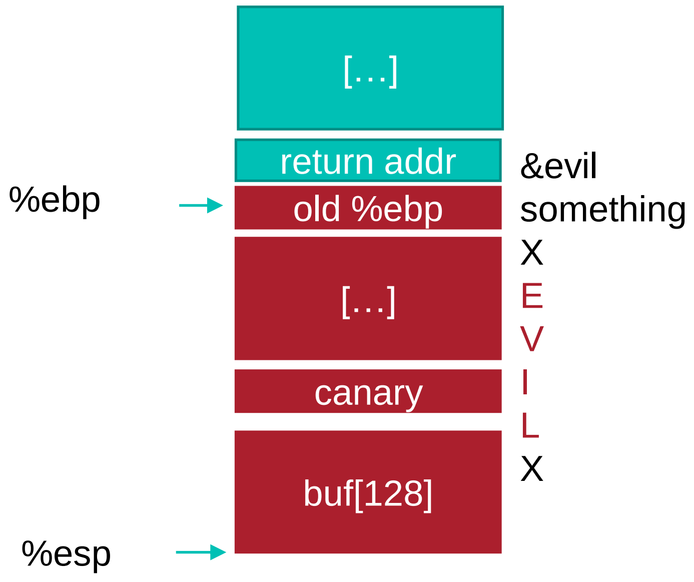
Stack canaries

- Let attacker overwrite stack
- Before return
- Careful about canary value
  - if deterministic can be guessed and avoided
- Use some special characters
  - e.g. \0, EOF etc…
  - remember last week
  - would only work for some input functions



Stack canaries

- Let attacker overwrite stack
- Before return
- Careful about canary value
  - if deterministic can be guessed and avoided
- Use some special characters
  - e.g. \0, EOF etc…
  - remember last week
  - would only work for some input functions
- Use some random value
  - careful with entropy



When do canaries fail?

- When attacker overwrite function pointers

When do canaries fail?

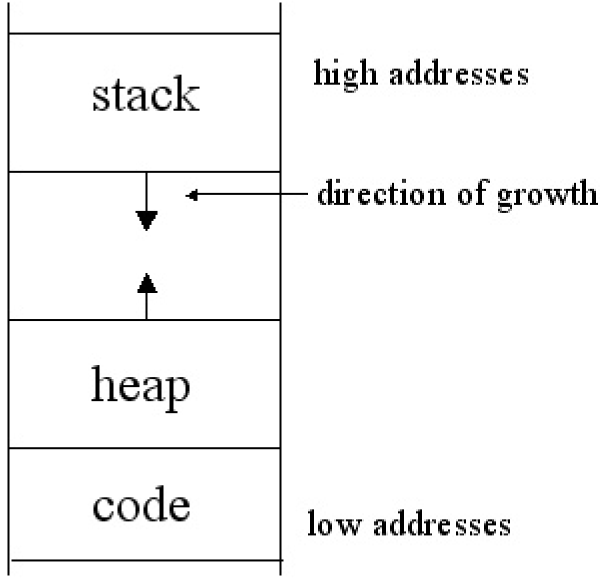
- When attacker overwrite function pointers
  - int *ptr = … ;
  - char buf[128];
  - gets(buff);
  - ptr(…);

When do canaries fail?

- When attacker overwrite function pointers
- Can attacker guess the randomness?
  - Source of randomness is a research topics on its own!

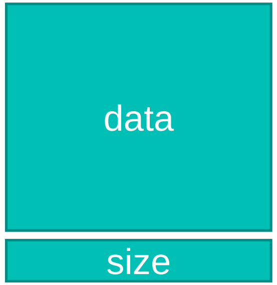
When do canaries fail?

- When attacker overwrite function pointers
- Can attacker guess the randomness?
- malloc and free (heap)
  - char *p, *q;
  - p = malloc(127);
  - q = malloc(127);
  - strcpy(p, buf);
  - free(p);
  - free(q);



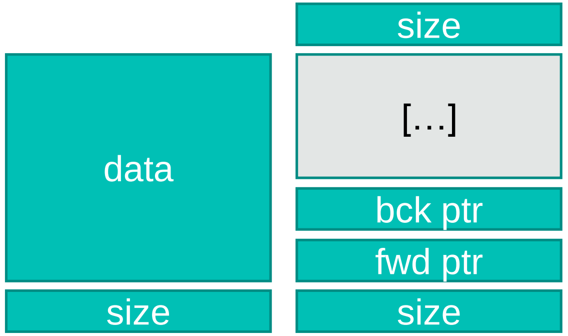
When do canaries fail?

- When attacker overwrite function pointers
- Can attacker guess the randomness?
- malloc and free (heap)
  - char *p, *q;
  - p = malloc(127);
  - q = malloc(127);
  - strcpy(p, buf);
  - free(p);
  - free(q);



When do canaries fail?

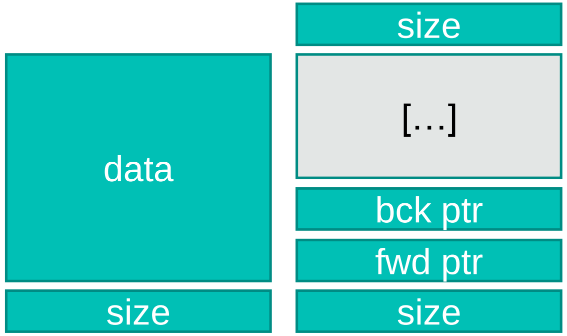
- When attacker overwrite function pointers
- Can attacker guess the randomness?
- malloc and free (heap)
  - char *p, *q;
  - p = malloc(127);
  - q = malloc(127);
  - strcpy(p, buf);
  - free(p);
  - free(q);



pointer and size for book keeping

When do canaries fail?

- When attacker overwrite function pointers
- Can attacker guess the randomness?
- malloc and free (heap)
  - p = get_free_block(size);
  - bck = p->bck;
  - fwd = p->fwd;
  - fwd->bck = bck;
  - fwd->fwd = fwd;

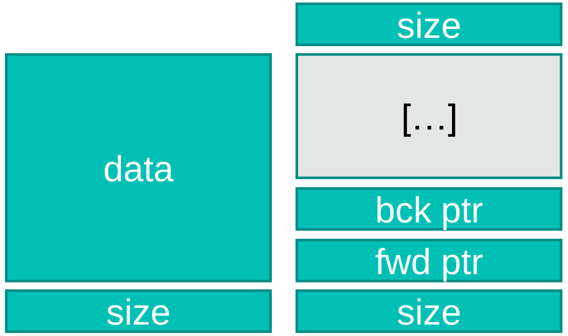


pointer and size for book keeping

When do canaries fail?

- When attacker overwrite function pointers
- Can attacker guess the randomness?
- malloc and free (heap)
  - p = get_free_block(size);
  - bck = p->bck;
  - fwd = p->fwd;
  - fwd->bck = bck;
  - fwd->fwd = fwd;

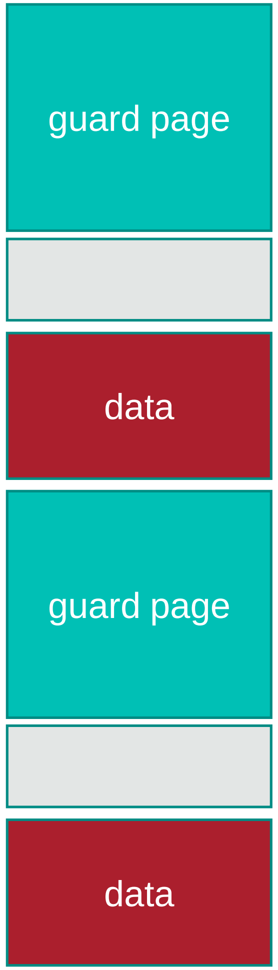
GAINED CONTROL OF
MEMORY ALLOCATION



pointer and size for book keeping

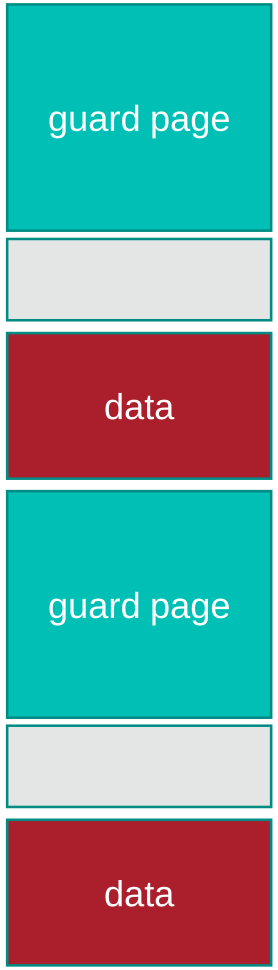
Electric Fence

- Use guard page
  - Page with memory protection so that if touched, create a fault



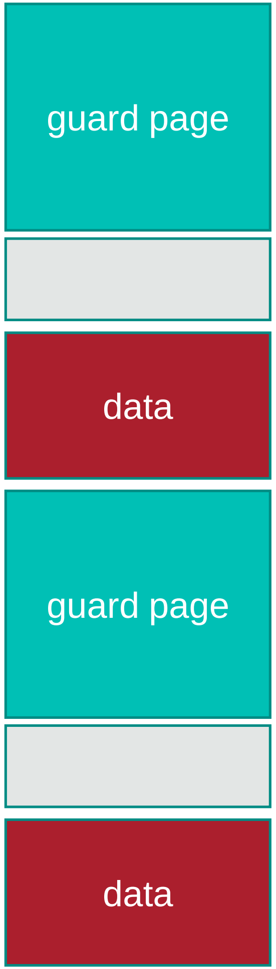
Electric Fence

- Use guard page
  - Page with memory protection so that if touched, create a fault
- Fault immediate



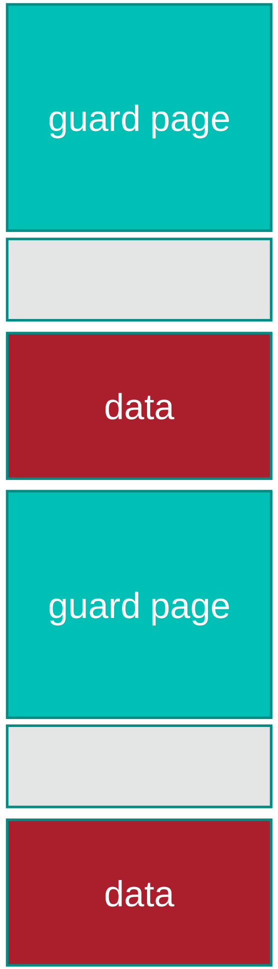
Electric Fence

- Use guard page
  - Page with memory protection so that if touched, create a fault
- Fault immediate
- No extra code check
- What may be the problem?



Electric Fence

- Use guard page
  - Page with memory protection so that if touched, create a fault
- Fault immediate
- No extra code check
- Very memory inefficient
- Work only across pages
- Generally used only for debugging/test



Bounds checking

- Make sure pointer refer to a specific memory object, and does not go out of that object

Bounds checking

- Make sure pointer refer to a specific memory object, and does not go out of that object
- Easy on paper…
- … a bit harder in C

Bounds checking

- Make sure pointer refer to a specific memory object, and does not go out of that object
- Easy on paper…
- … a bit harder in C
  - char x[1024];
  - char *y = x[107];

Bounds checking

- Make sure pointer refer to a specific memory object, and does not go out of that object
- Easy on paper…
- … a bit harder in C
  - char x[1024];
  - char *y = x[107];

  - union{
  -   int c;
  -   struct s{
  -     int j;
  -     int k;
  - }};

Bounds checking

- Make sure pointer refer to a specific memory object, and does not go out of that object
- Easy on paper…
- … a bit harder in C
  - char x[1024];
  - char *y = x[107];

  - union{
  -   int c;
  -   struct s{
  -     int j;
  -     int k;
  - }};

- `int *ptr = &(u.s.k);`

Bounds checking

- Make sure pointer refer to a specific memory object, and does not go out of that object
- Easy on paper…
- … a bit harder in C
  - char x[1024];
  - char *y = x[107];

  - union{
  -   int c;
  -   struct s{
  -     int j;
  -     int k;
  - }};

- `int *ptr = &(u.s.k);`

- Weaker guarantees
  - From a pointer p’ deriving from p. Then p’ should only be used to dereference memory that belongs to p.

Bounds checking

- Make sure pointer refer to a specific memory object, and does not go out of that object
- Easy on paper…
- … a bit harder in C
  - char x[1024];
  - char *y = x[107];

  - union{
  -   int c;
  -   struct s{
  -     int j;
  -     int k;
  - }};

- `int *ptr = &(u.s.k);`

- Weaker guarantees
  - From a pointer p’ deriving from p. Then p’ should only be used to dereference memory that belongs to p.
- You can still trample memory
- … but not arbitrary memory

Bounds checking

- Make sure pointer refer to a specific memory object, and does not go out of that object
- Easy on paper…
- … a bit harder in C
  - char x[1024];
  - char *y = x[107];

  - union{
  -   int c;
  -   struct s{
  -     int j;
  -     int k;
  - }};

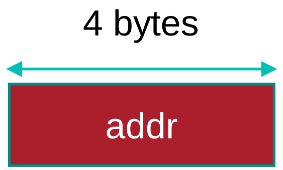
- `int *ptr = &(u.s.k);`

- Weaker guarantees
  - From a pointer p’ deriving from p. Then p’ should only be used to dereference memory that belongs to p.
- You can still trample memory
- … but not arbitrary memory

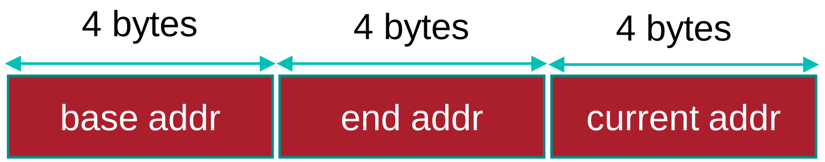
Requires compiler support: issue with legacy libraries

Fat address

- 32 bits address

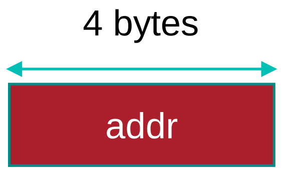


- fat address

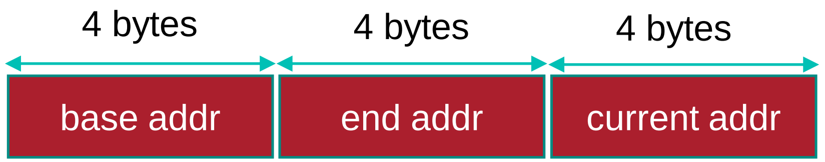


Fat address

- 32 bits address



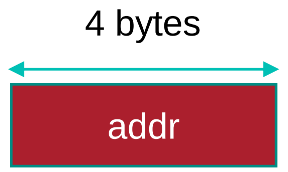
- fat address



```
▪ int *ptr = malloc(8);
▪ While(1) {
▪   *ptr = 42;
▪   ptr++;
▪ }
```

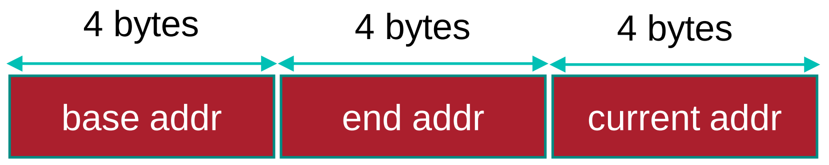
Fat address

- 32 bits address



```
▪ int *ptr = malloc(8);
▪ While(1) {
▪   *ptr = 42;
▪   ptr++;
▪ }
```

- fat address



Need to instrument code
i.e. compiler support

Problem with external library
Non-atomic

Worms

… and a bit of history

Morris Worm 1988

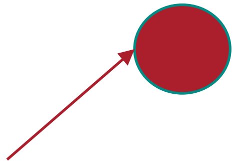
- a.k.a the Great Worm
- Designed by Robert Morris

Morris Worm 1988

- a.k.a the Great Worm
- Designed by Robert Morris in 1988

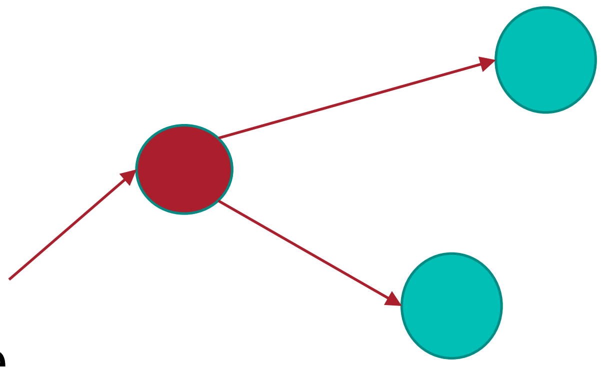
Morris Worm 1988

- a.k.a the Great Worm
- Designed by Robert Morris
- Exploit a vulnerability to execute a program on a machine



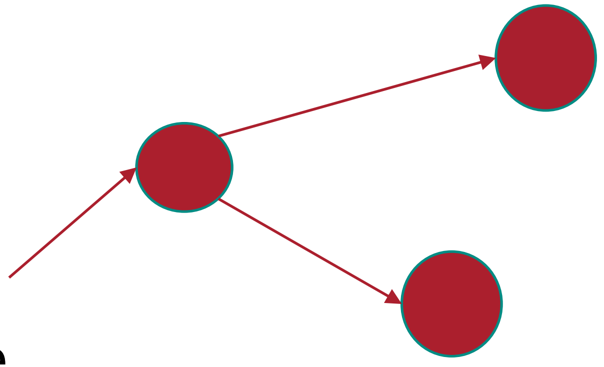
Morris Worm 1988

- a.k.a the Great Worm
- Designed by Robert Morris
- Exploit a vulnerability to execute a program on a machine
- Send pay load to compromise other machines



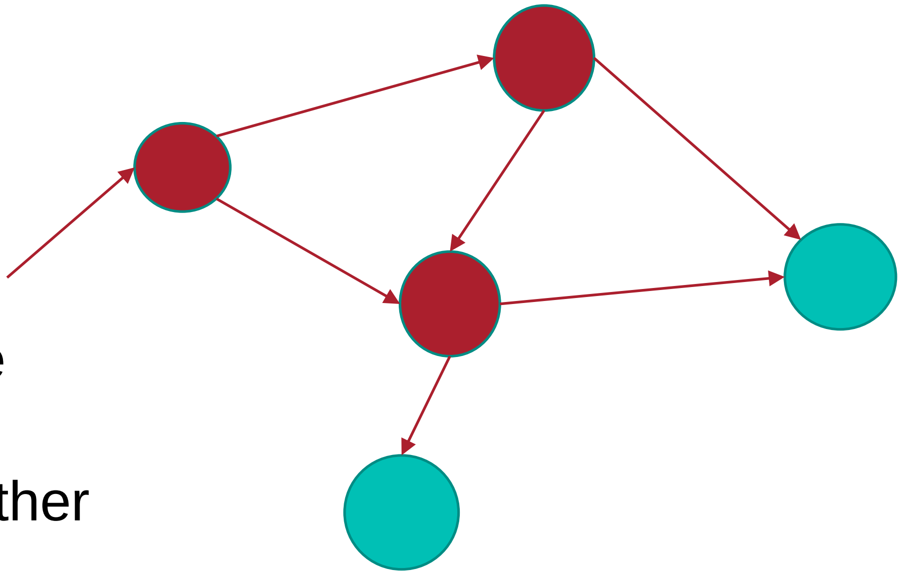
Morris Worm 1988

- a.k.a the Great Worm
- Designed by Robert Morris
- Exploit a vulnerability to execute a program on a machine
- Send pay load to compromise other machines on the same network



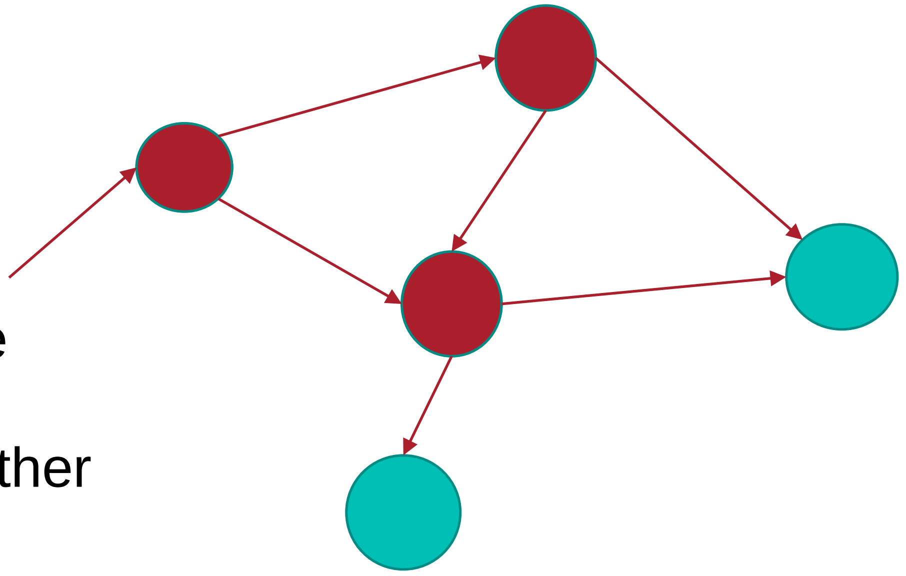
Morris Worm 1988

- a.k.a the Great Worm
- Designed by Robert Morris
- Exploit a vulnerability to execute a program on a machine
- Send pay load to compromise other machines on the same network
- Repeat



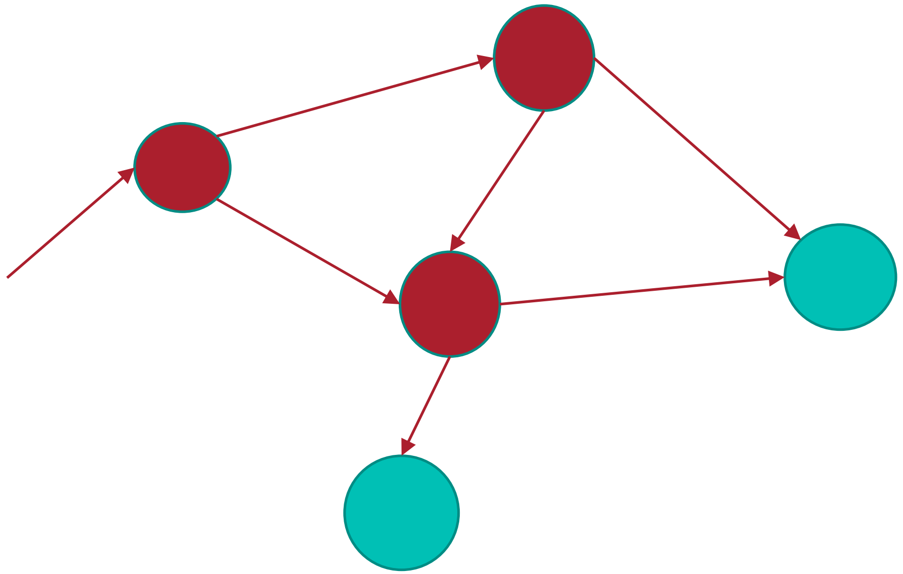
Morris Worm 1988

- a.k.a the Great Worm
- Designed by Robert Morris
- Exploit a vulnerability to execute a program on a machine
- Send pay load to compromise other machines on the same network
- Repeat
- Stated purpose “mapping” the internet



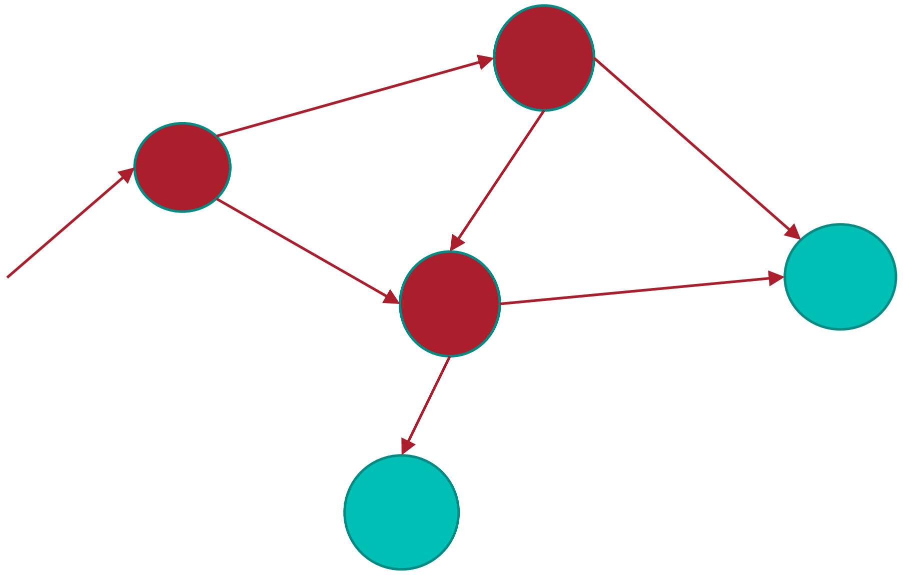
Morris Worm 1988

- Worm could test if a copy was already there by asking



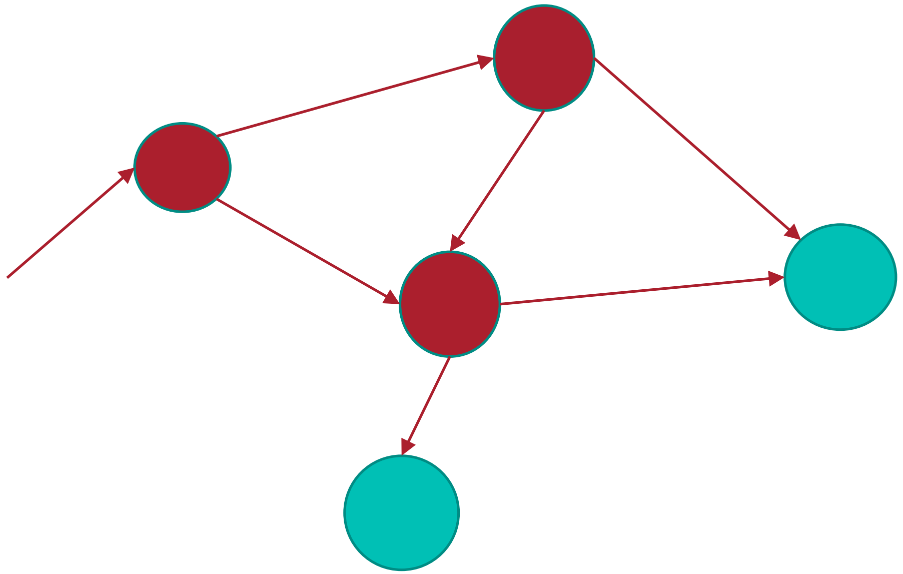
Morris Worm 1988

- Worm could test if a copy was already there by asking
  - Countermeasure?



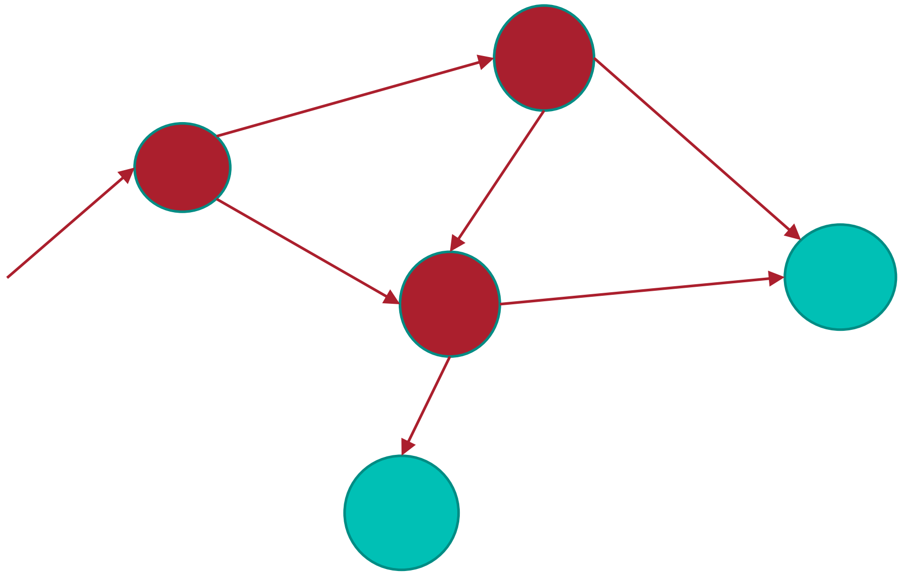
Morris Worm 1988

- Worm could test if a copy was already there by asking
  - Countermeasure?
  - Simply say yes



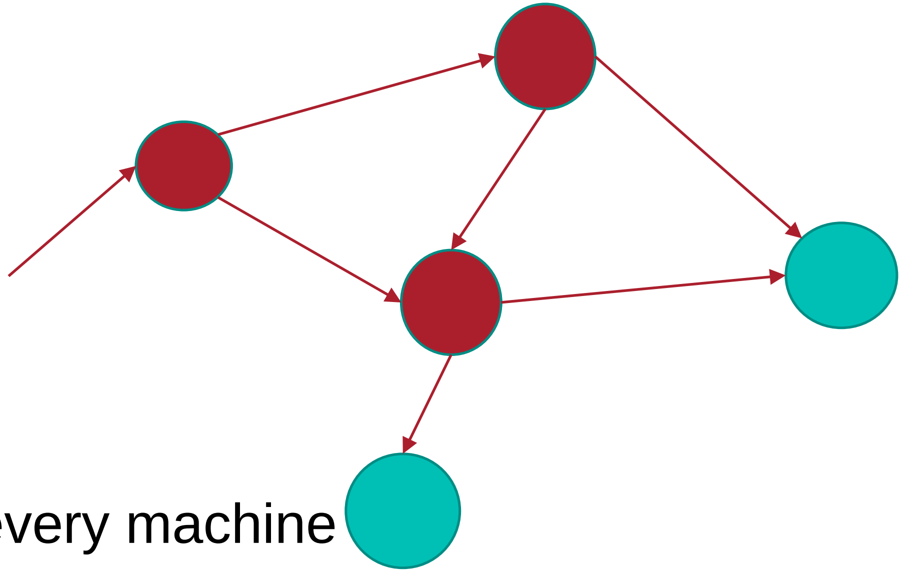
Morris Worm 1988

- Worm could test if a copy was already there by asking
  - Countermeasure?
  - Simply say yes
  - Copy anyway 1/7 time

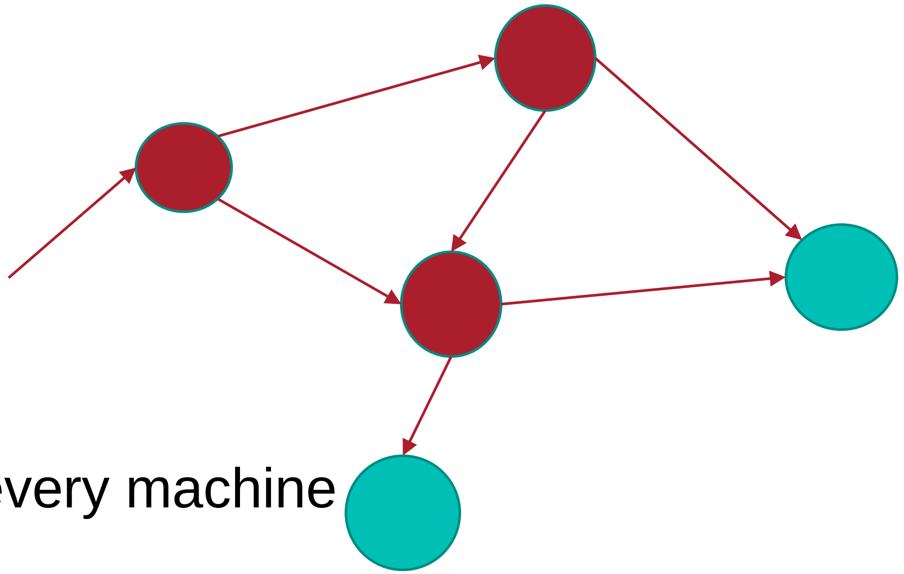


Morris Worm 1988

- Worm could test if a copy was already there by asking
  - Countermeasure?
  - Simply say yes
  - Copy anyway 1/7 time
- Results thousands process on every machine
- Machine running to a crawl



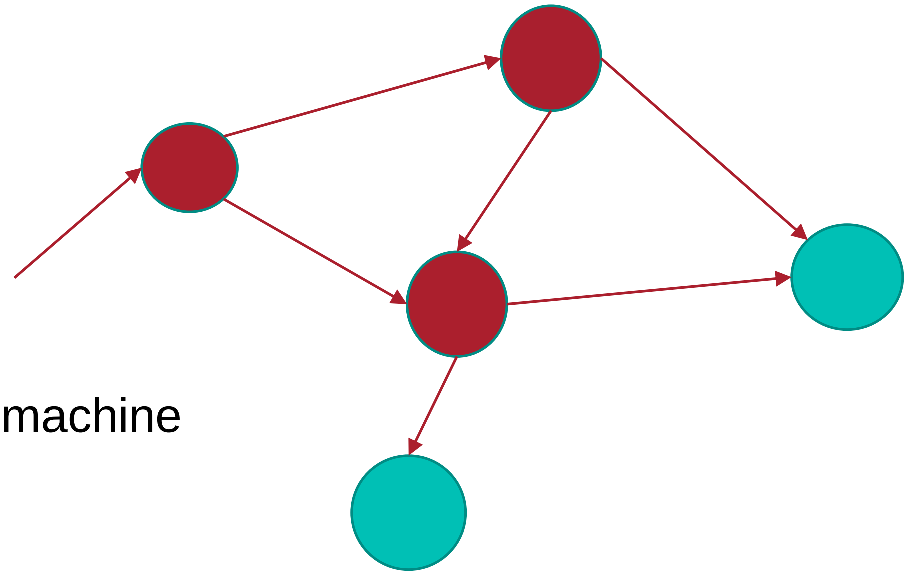
Morris Worm 1988



- Worm could test if a copy was already there by asking
  - Countermeasure?
  - Simply say yes
  - Copy anyway 1/7 time
- Results thousands process on every machine
- Machine running to a crawl
- Take down a machine to clean it
- Get infected again instantly

Morris Worm 1988

- Worm could test if a copy was already there by asking
  - Countermeasure?
  - Simply say yes
  - Copy anyway 1/7 time
- Results thousands process on every machine
- Machine running to a crawl
- Take down a machine to clean it
- Get reinfected instantly
- Required a coordinated effort to “clean” the internet
- Largest denial of service attack



bristol.ac.uk

Blaster

- 2003 – Affect Windows 2000/XP Machines
- Exploit buffer overflow vulnerability on Remote Procedure Call
  - Get a shell with “admin” privilege
  - To download payload via ftp
  - And install it

Blaster

- 2003 – Affect Windows 2000/XP Machines
- Exploit buffer overflow vulnerability on Remote Procedure Call
- Aim to remain undetected
  - No more thousands processes
  - Check existence of a mutex (“BILLY”)
- Infect other random machine on the network
- Variant A – start a thread to DDOS Microsoft update

Blaster

- 2003 – Affect Windows 2000/XP Machines
- Exploit buffer overflow vulnerability on Remote Procedure Call
- Aim to remain undetected
- Infect other random machine on the network
- Variant A – start a thread to DDOS Microsoft update
- Contains two messages
  - I just want to say LOVE YOU SAN!!
  - billy gates why do you make this possible ? Stop making money and fix your software!!

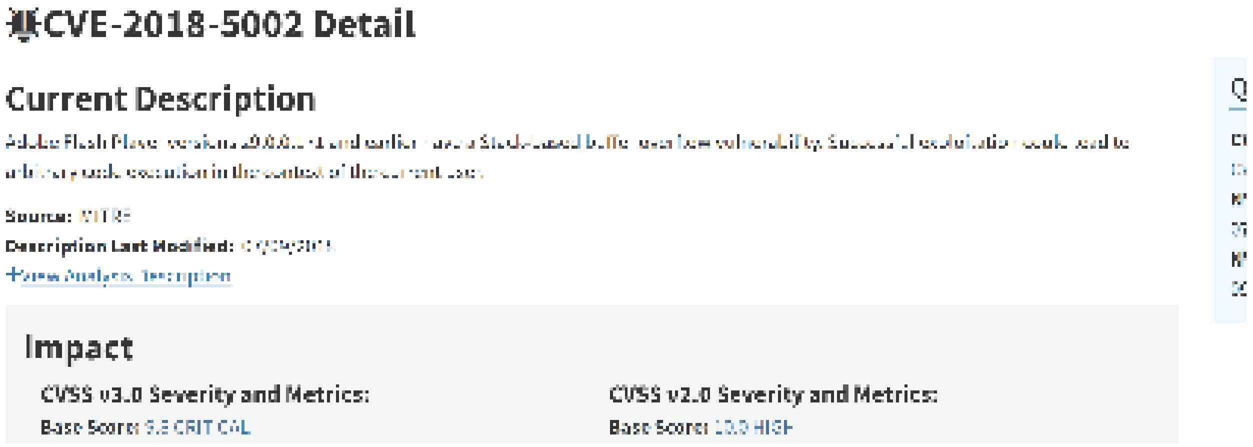
Blaster

- 2003 – Affect Windows 2000/XP Machines
- Exploit buffer overflow vulnerability on Remote Procedure Call
- Aim to remain undetected
- Infect other random machine on the network
- Variant A – start a thread to DDOS Microsoft update
- Later variant caused system to reboot every 60 seconds

Other buffer overflow example

- Twilight Hack (Wii)
  - Buffer Overflow on Legend of Zelda: Twilight Princess
  - When reading save files
  - Used to install pirated games

Buffer overflow in 2018? (just one of many)



CVE-2018-5002 Detail

Current Description

Adobe Flash Player versions 29.0.0.171 and earlier have a Stack-based buffer overflow vulnerability. Successful exploitation could lead to arbitrary code execution in the context of the current user.

Source: MITRE

Description Last Modified: [illegible]

+View Analysis Description

Impact

CVSS v3.0 Severity and Metrics:	CVSS v2.0 Severity and Metrics:
Base Score: 9.8 CRITICAL	Base Score: 10.0 HIGH

bristol.ac.uk

University of BRISTOL

Thank you

Office MVB 3.26 …

… out of order until?

bristol.ac.uk