Так ли проста стековая канарейка?

Мария Недяк

Разработчик группы харденингов KasperskyOS

Capture The Flag(CTF)

Outline

Outline

- **Что такое стековая канарейка?**

Outline

- **Что такое стековая канарейка?**

  - описание уязвимости

Outline

- **Что такое стековая канарейка?**

  - описание уязвимости
  - описание харденинга

Outline

- **Что такое стековая канарейка?**
  - описание уязвимости
  - описание харденинга
- **Ломаем стековую канарейку**

Outline

- **Что такое стековая канарейка?**
  - описание уязвимости
  - описание харденинга
- **Ломаем стековую канарейку (N раз)**

Outline

- **Что такое стековая канарейка?**
  - описание уязвимости
  - описание харденинга
- **Ломаем стековую канарейку (N раз)**
  - атака на канарейку

Outline

- **Что такое стековая канарейка?**
  - описание уязвимости
  - описание харденинга
- **Ломаем стековую канарейку (N раз)**
  - атака на канарейку
  - усиление канарейки

Outline

- **Что такое стековая канарейка?**
  - описание уязвимости
  - описание харденинга
- **Ломаем стековую канарейку (N раз)**
  - атака на канарейку
  - усиление канарейки
- **Выводы: какая она должна быть стековая канарейка?**

Стековая канарейка

```
int main(int argc, char **argv)
{
   check_password();
   return 0;
}
```

```
void check_password()
{
   char password[12];

   scanf(“%s”, password);

   // handle password
   // …
}

int main(int argc, char **argv)
{
   check_password();
   return 0;
}
```

```
void admin_panel()
{
   system(“/bin/sh”);
}

void check_password()
{
   char password[12];

   scanf(“%s”, password);

   // handle password
   // …
}

int main(int argc, char **argv)
{
   check_password();
   return 0;
}
```

```
void admin_panel()
{
   system(“/bin/sh”);
}

void check_password()
{
   char password[12];

   scanf(“%s”, password);

   // handle password
   // …
}

int main(int argc, char **argv)
{
   check_password();
   return 0;
}
```

Стек фрейм

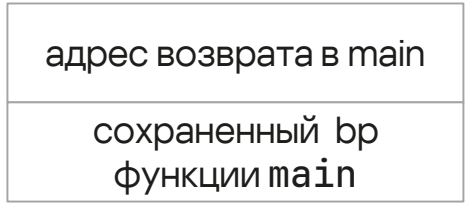
```
void admin_panel()
{
   system(“/bin/sh”);
}

void check_password()
{
   char password[12];

   scanf(“%s”, password);

   // handle password
   // …
}

int main(int argc, char **argv)
{
   check_password();
   return 0;
}
```

Стек фрейм

адрес возврата в main

```
void admin_panel()
{
   system(“/bin/sh”);
}

void check_password()
{
   char password[12];

   scanf(“%s”, password);

   // handle password
   // …
}

int main(int argc, char **argv)
{
   check_password();
   return 0;
}
```

Стек фрейм

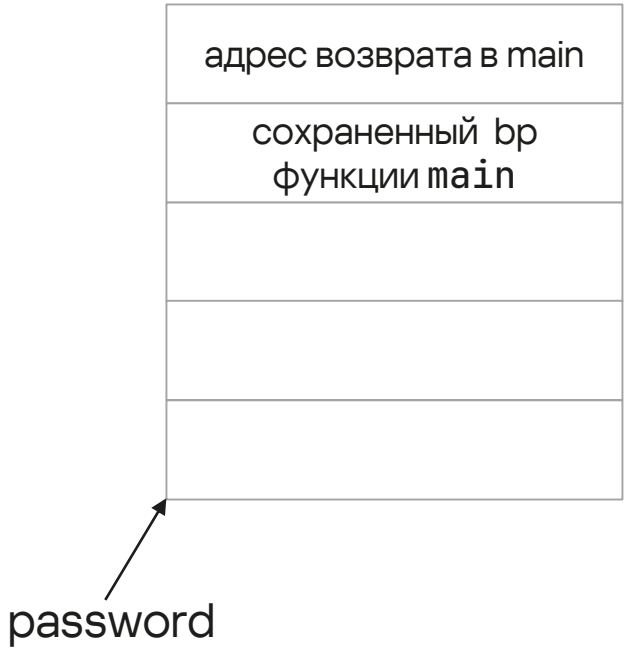


```
void admin_panel()
{
   system(“/bin/sh”);
}

void check_password()
{
   char password[12];

   scanf(“%s”, password);

   // handle password
   // …
}

int main(int argc, char **argv)
{
   check_password();
   return 0;
}
```

Стек фрейм

```
void admin_panel()
{
    system("/bin/sh");
}

void check_password()
{
    char password[12];

    scanf("%s", password);

    // handle password
    // …
}

int main(int argc, char **argv)
{
    check_password();
    return 0;
}
```

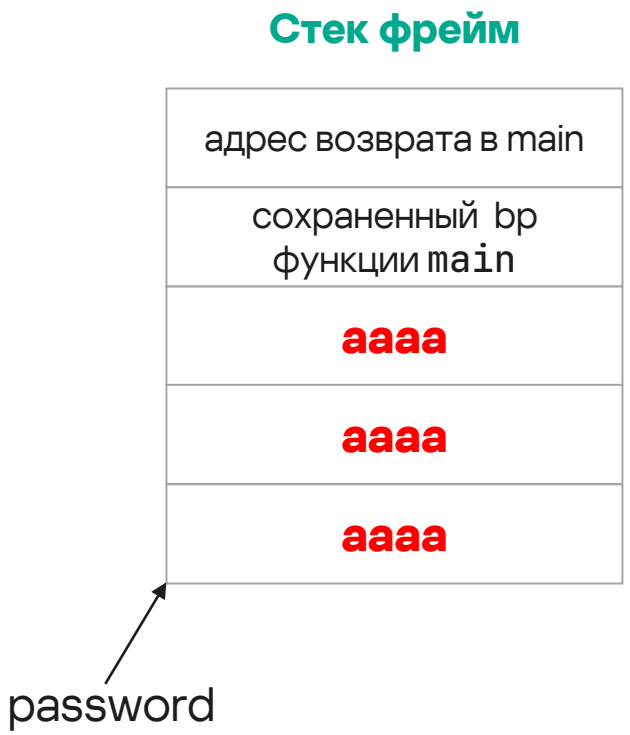
Стек фрейм



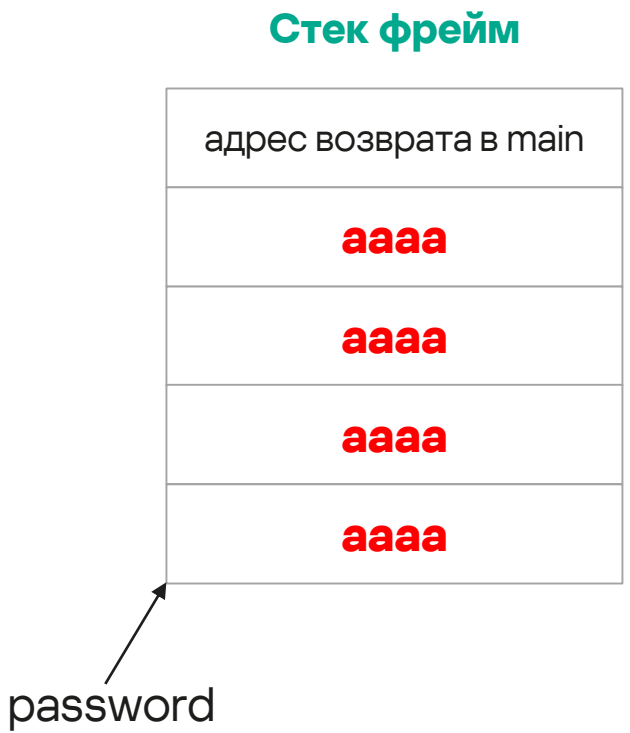
```
void admin_panel()
{
    system("/bin/sh");
}

void check_password()
{
    char password[12];

    scanf("%s", password);

    // handle password
    // …
}

int main(int argc, char **argv)
{
    check_password();
    return 0;
}
```

```
void admin_panel()
{
    system("/bin/sh");
}

void check_password()
{
    char password[12];

    scanf("%s", password);

    // handle password
    // …
}

int main(int argc, char **argv)
{
    check_password();
    return 0;
}
```

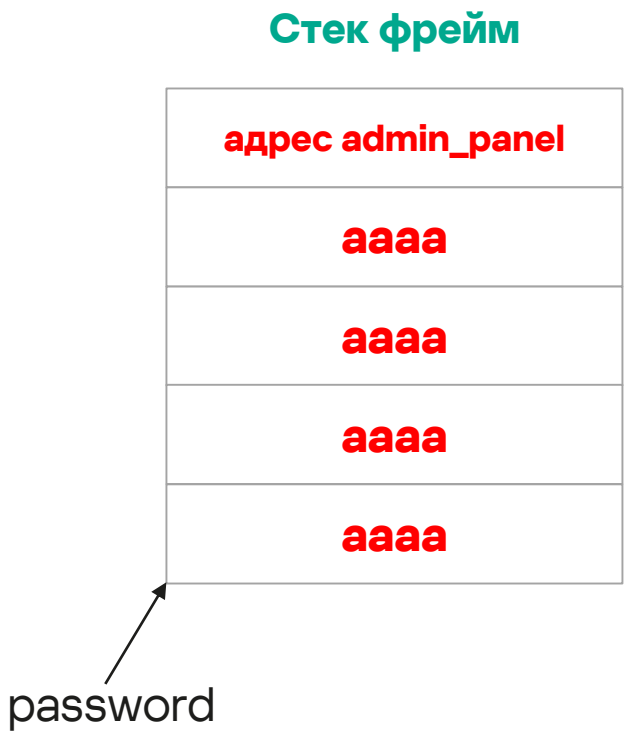
```
void admin_panel()
{
    system("/bin/sh");
}

void check_password()
{
    char password[12];

    scanf("%s", password);

    // handle password
    // …
}

int main(int argc, char **argv)
{
    check_password();
    return 0;
}
```



```
void admin_panel()
{
    system("/bin/sh");
}

void check_password()
{
    char password[12];

    scanf("%s", password);

    // handle password
    // …
}

int main(int argc, char **argv)
{
    check_password();
    return 0;
}
```



```
void admin_panel()
{
    system("/bin/sh");
}

void check_password()
{
    char password[12];

    scanf("%s", password);

    // handle password
    // …
}

int main(int argc, char **argv)
{
    check_password();
    return 0;
}
```



```
void admin_panel()
{
    system(“/bin/sh”);
}

void check_password()
{
    char password[12];

    scanf(“%s”, password);

    // handle password
    // …
}

int main(int argc, char
{
    check_password();
    return 0;
}
```

Стек фрейм

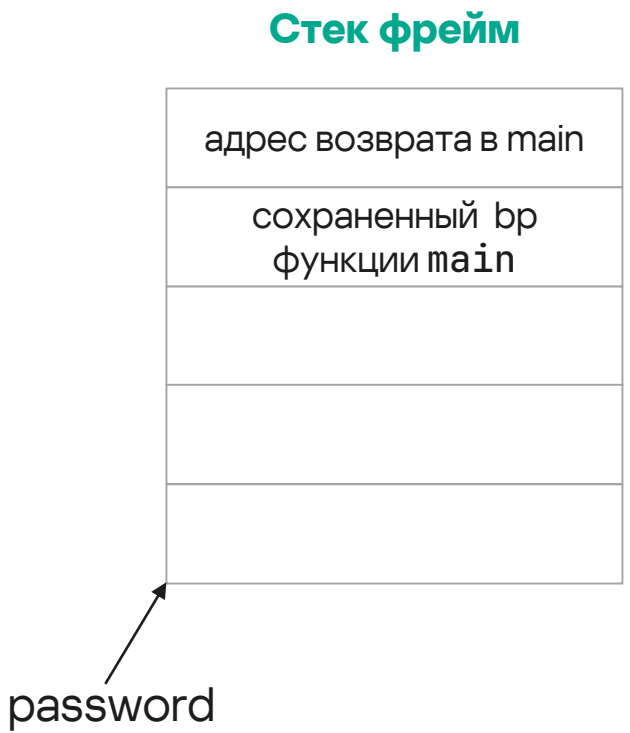
адрес admin_panel
aaaa
aaaa
aaaa
aaaa

password

Как защищаемся?

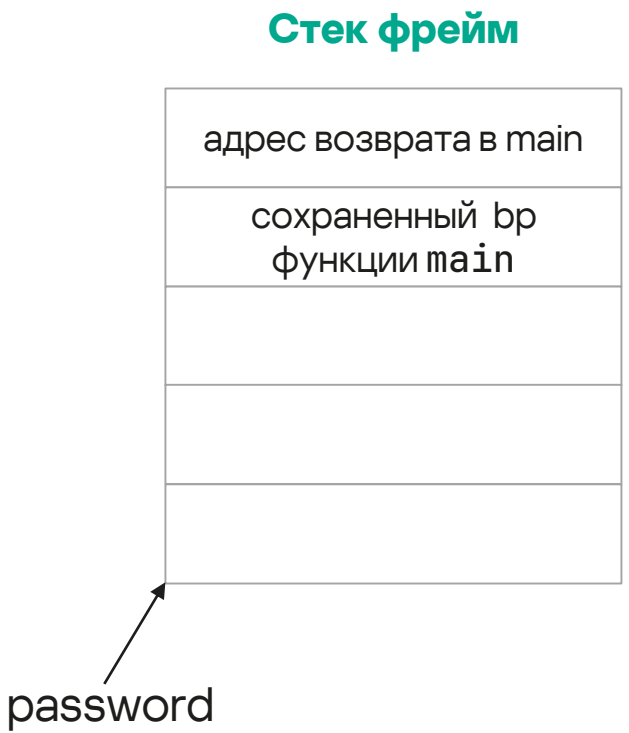
Переполнение стека. Как защищаемся?


```
…

void check_password()
{
   char password[12];

   scanf(“%s”, password);
   …
}

…
```



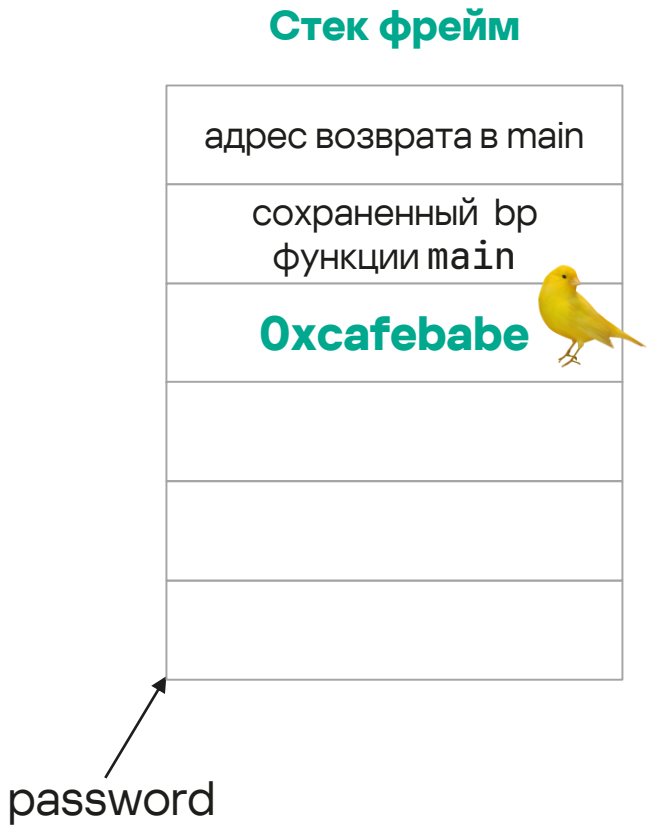
Переполнение стека. Как защищаемся?


```
int canary_value = 0xcafebabe;

…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf(“%s”, password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!!
   }
}

…
```



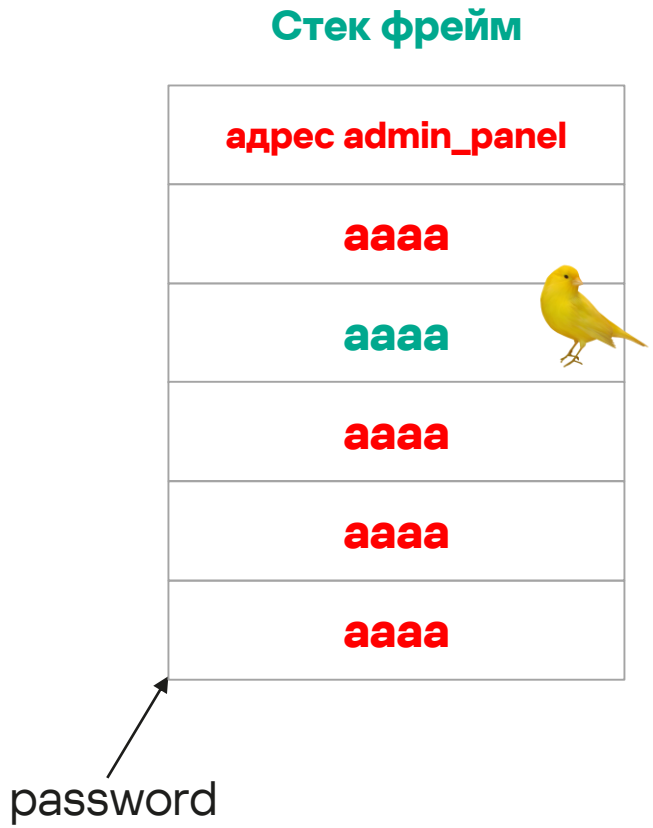
Переполнение стека. Как защищаемся?


```
int canary_value = 0xcafebabe;

…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf(“%s”, password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!!
   }
}

…
```



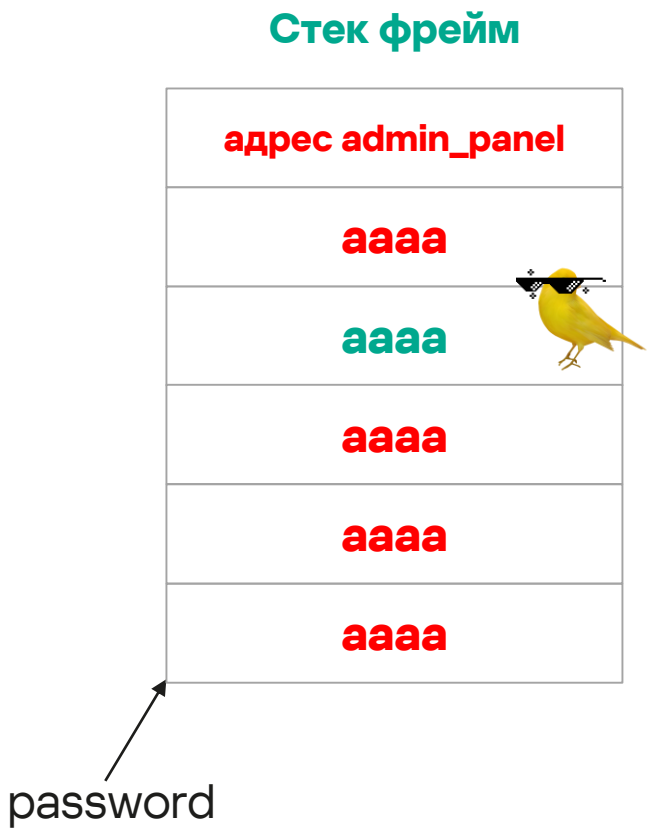
Переполнение стека. Как защищаемся?


```
int canary_value = 0xcafebabe;

…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf(“%s”, password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!!
   }
}

…
```



Переполнение стека. Как защищаемся?


```
int canary_value = 0xcafebabe;

…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf(“%s”, password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!
   }
}

…
```



Включаем стековую канарейку

Stack canaries are added by GCC and Clang through these flags:

- `-fstack-protector`
- `-fstack-protector-strong`
- `-fstack-protector-all`
- `-fstack-protector-explicit`

MSVC: /GS (Buffer Security Check)

Историческая справка

ист: Christal Pollock "The Canary in the Coal Mine," Journal of Avian Medicine and Surgery 30(4), 386-391, (1 December 2016). https://doi.org/10.1647/1082-6742-30.4.386

Ломаем канарейку

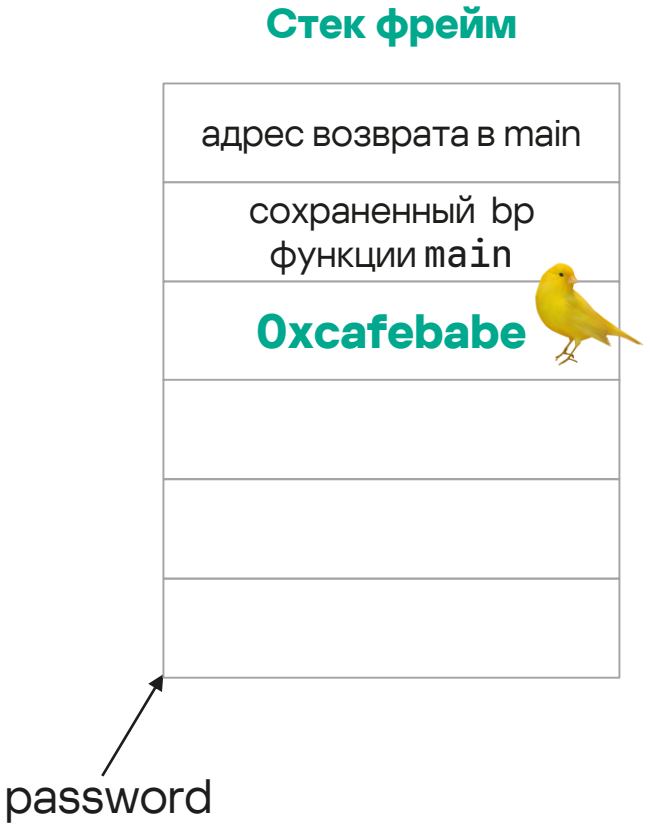
Обходим канарейку


```
int canary_value = 0xcafebabe;

…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf(“%s”, password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!!
   }
}

…
```

Обходим канарейку

Случайное значение!

```
int canary_value = 0xcafebabe;

…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf(“%s”, password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!!
   }
}

…
```



Обходим канарейку

Случайное значение!

```
int canary_value = 0xcafebabe;

…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf("%s", password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!!
   }
}

…
```

узнать

Обходим канарейку

Случайное значение!

```
int canary_value = 0xcafebabe;

…

void check_password()
{
                ry_value;
                ;

                ord);
                ary_value)

                T CONTINUE!!!!
```

узнать

переписать

password

Защита канарейки

	Перезапись	Чтение
Windows		
OpenBSD		
Glibc(Linux)		

Переписываем канарейку

Переписываем канарейку

Переписываем канарейку

- Знаем адрес глобальной канарейки

Переписываем канарейку

- Знаем адрес глобальной канарейки

- У нас есть возможность писать по этому адресу

Переписываем канарейку


```
int canary_value = 0xcafebabe;
…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf("%s", password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!!
   }
}

…
```

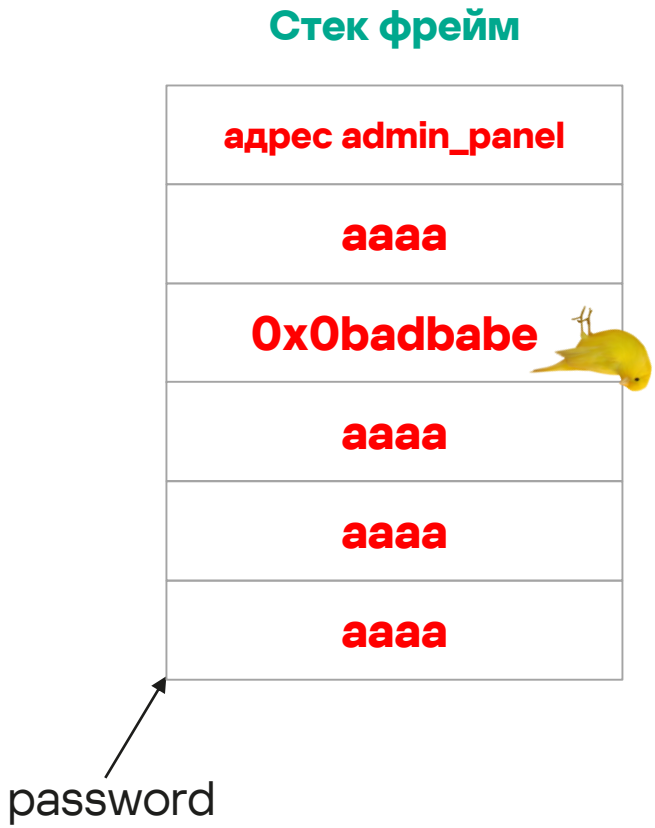
Переписываем канарейку


```
int canary_value = 0x0badbabe;
```

```
…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf("%s", password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!!
   }
}

…
```

Переписываем канарейку


```
int canary_value = 0x0badbabe;

…

void check_password()
{
   int canary = canary_value;
   char password[12];

   scanf(“%s”, password);
   …

   if (canary != canary_value)
   {
     // ALARM; DO NOT CONTINUE!!!!
   }
}

…
```



Как защититься от перезаписи?

Как защититься от перезаписи?

положить канарейку в read-only память!

Защита канарейки

	Перезапись	Чтение
Windows		
OpenBSD		
Glibc(Linux)		

Защита канарейки

	Перезапись	Чтение
Windows	data section (ntdll.dll) ✅	
OpenBSD		
Glibc(Linux)		

Защита канарейки

	Перезапись	Чтение
Windows	data section (ntdll.dll) ✅	
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы ✅	
Glibc(Linux)		

Защита канарейки

	Перезапись	**Чтение**
Windows	data section (ntdll.dll) ✅	
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы ✅	
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**, иначе в **data.rel.ro** секции	

Защита канарейки

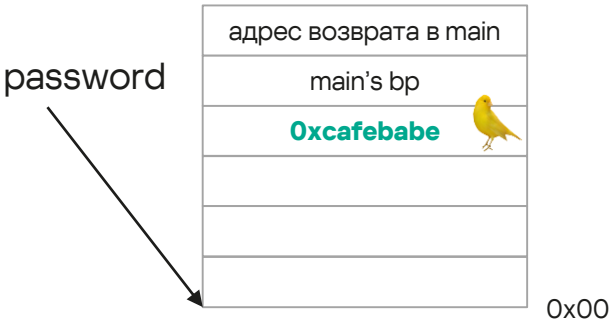
	Перезапись	**Чтение**
Windows	data section (ntdll.dll) ✅	
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы ✅	
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**, иначе в **data.rel.ro** секции ✅	

Защита канарейки

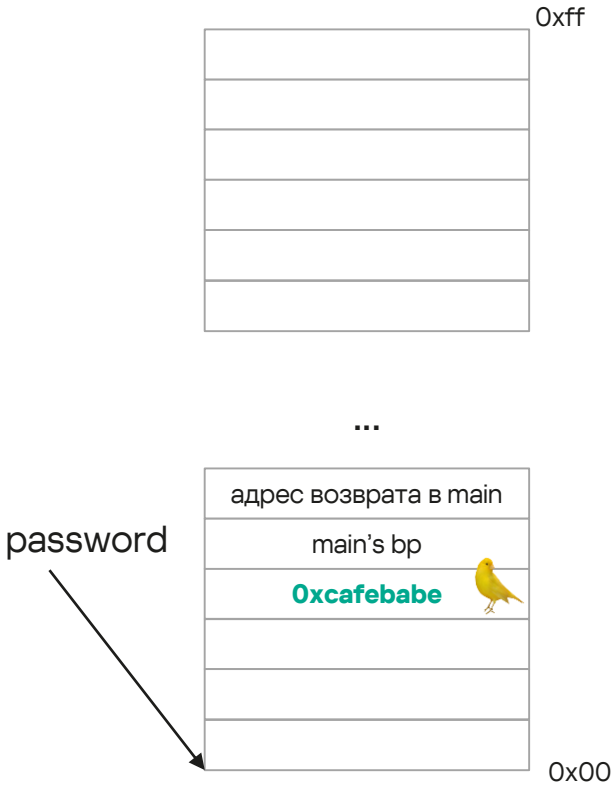
	Перезапись	Чтение
Windows	data section (ntdll.dll) ✅	
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы ✅	
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,??? иначе в **data.rel.ro** секции ✅	

Погружение в Thread Local Storage

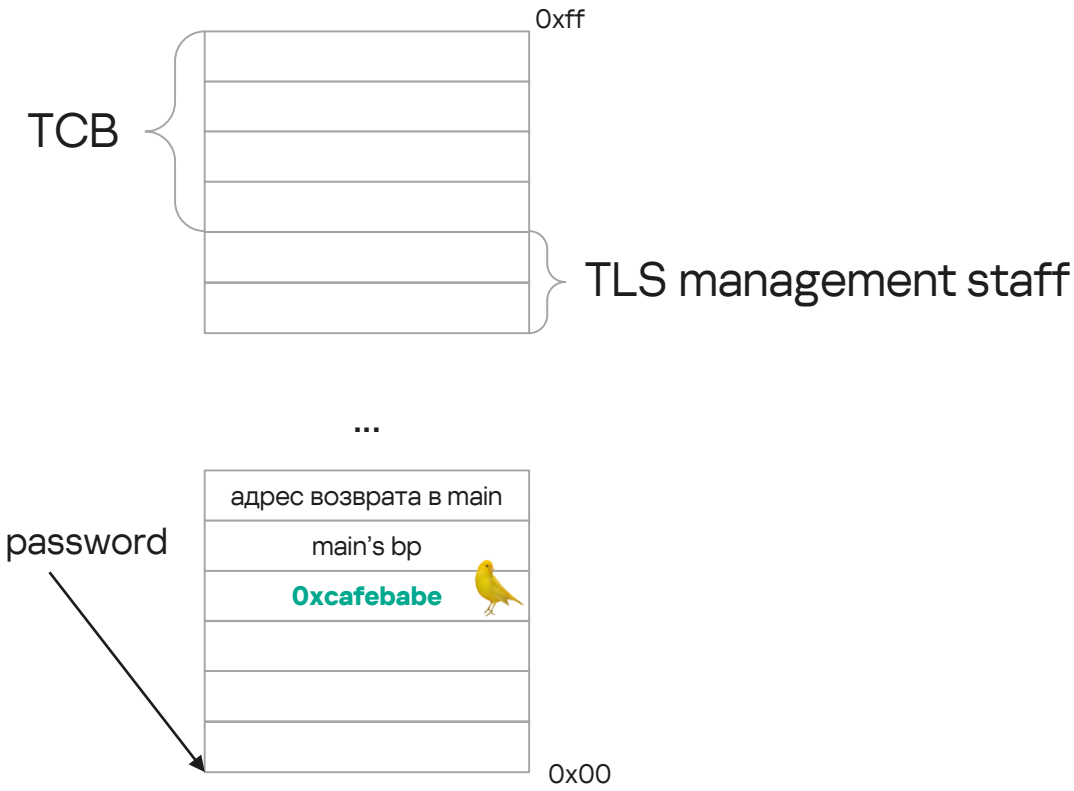
Погружение в Thread Local Storage



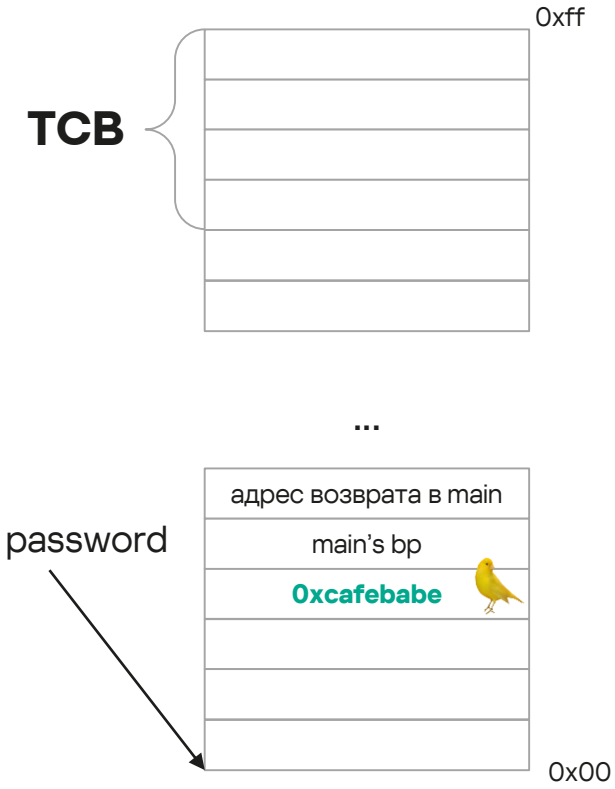
Погружение в Thread Local Storage



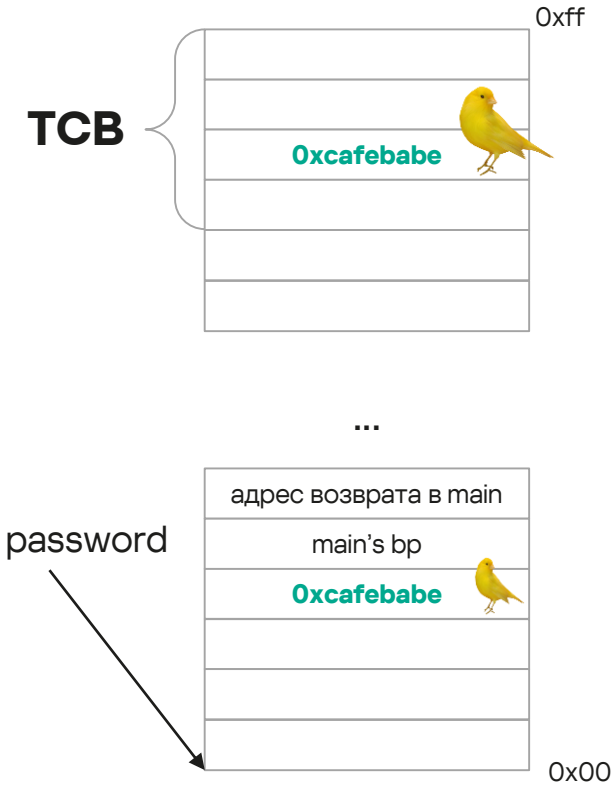
Погружение в Thread Local Storage



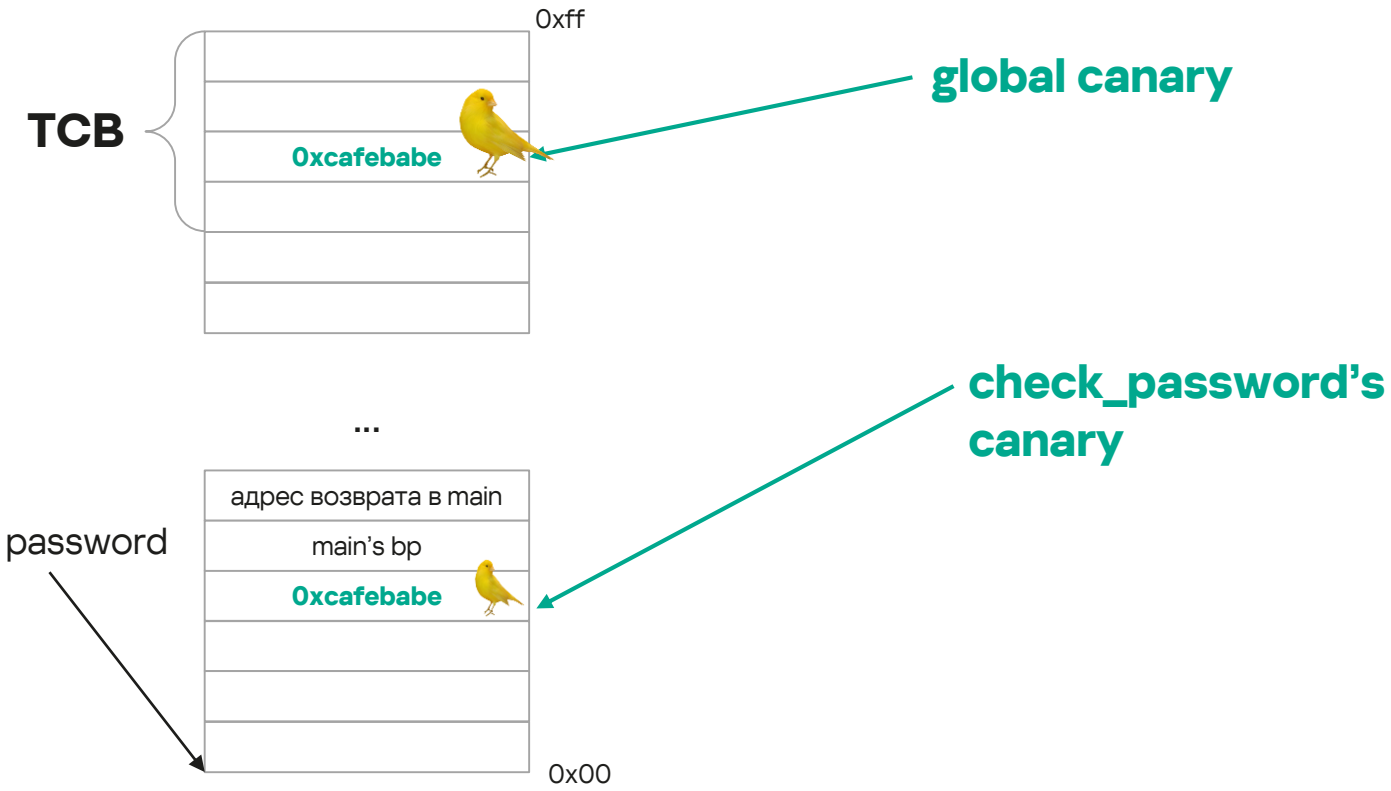
Погружение в Thread Local Storage



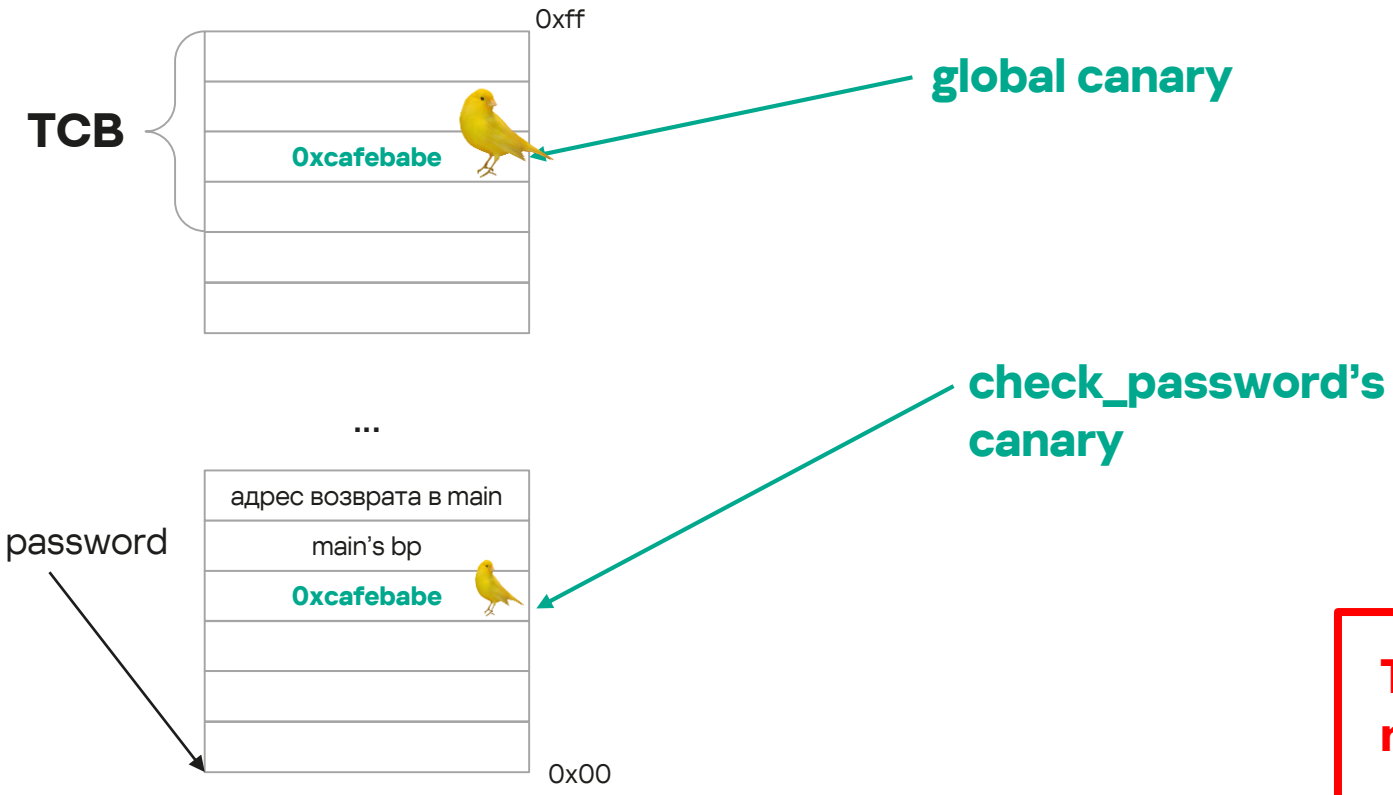
Погружение в Thread Local Storage



Погружение в Thread Local Storage

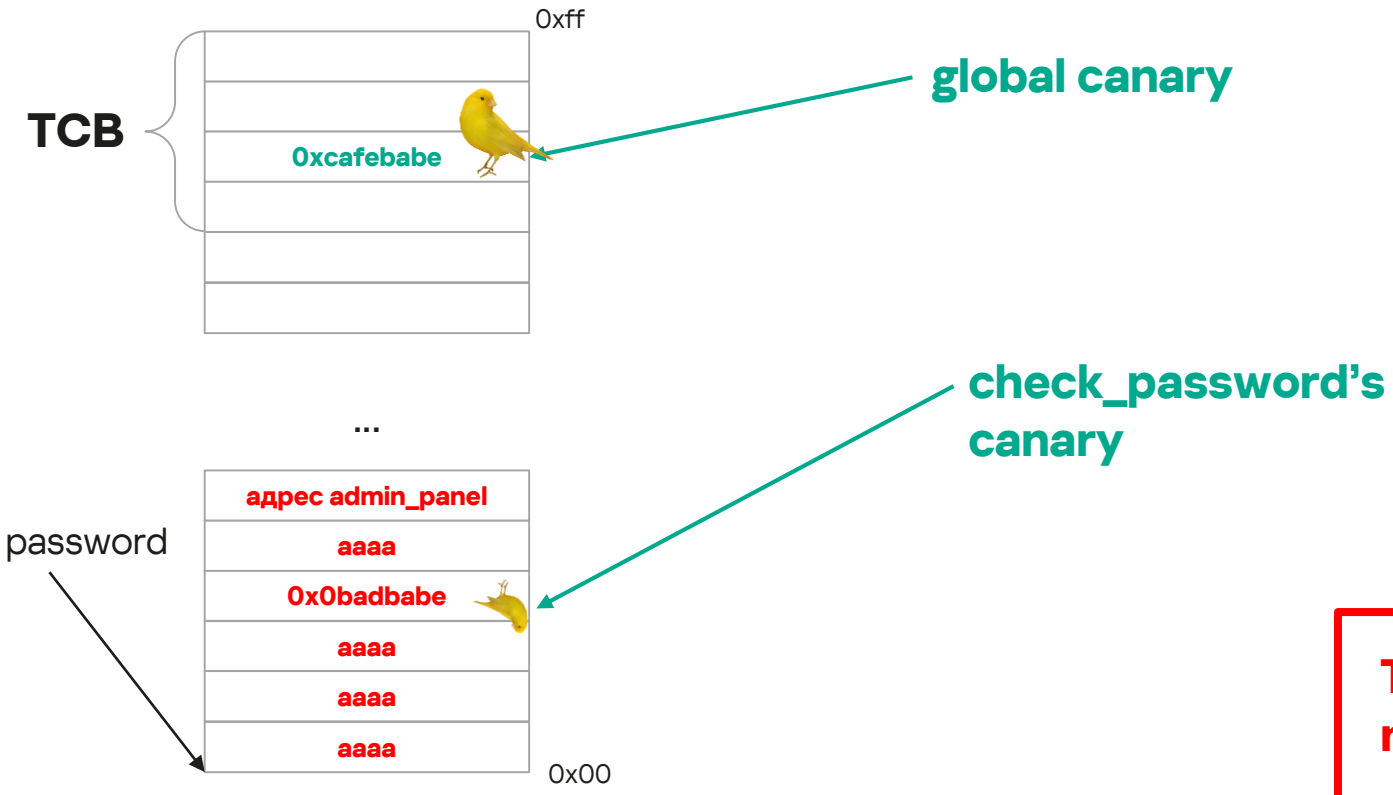


Погружение в Thread Local Storage



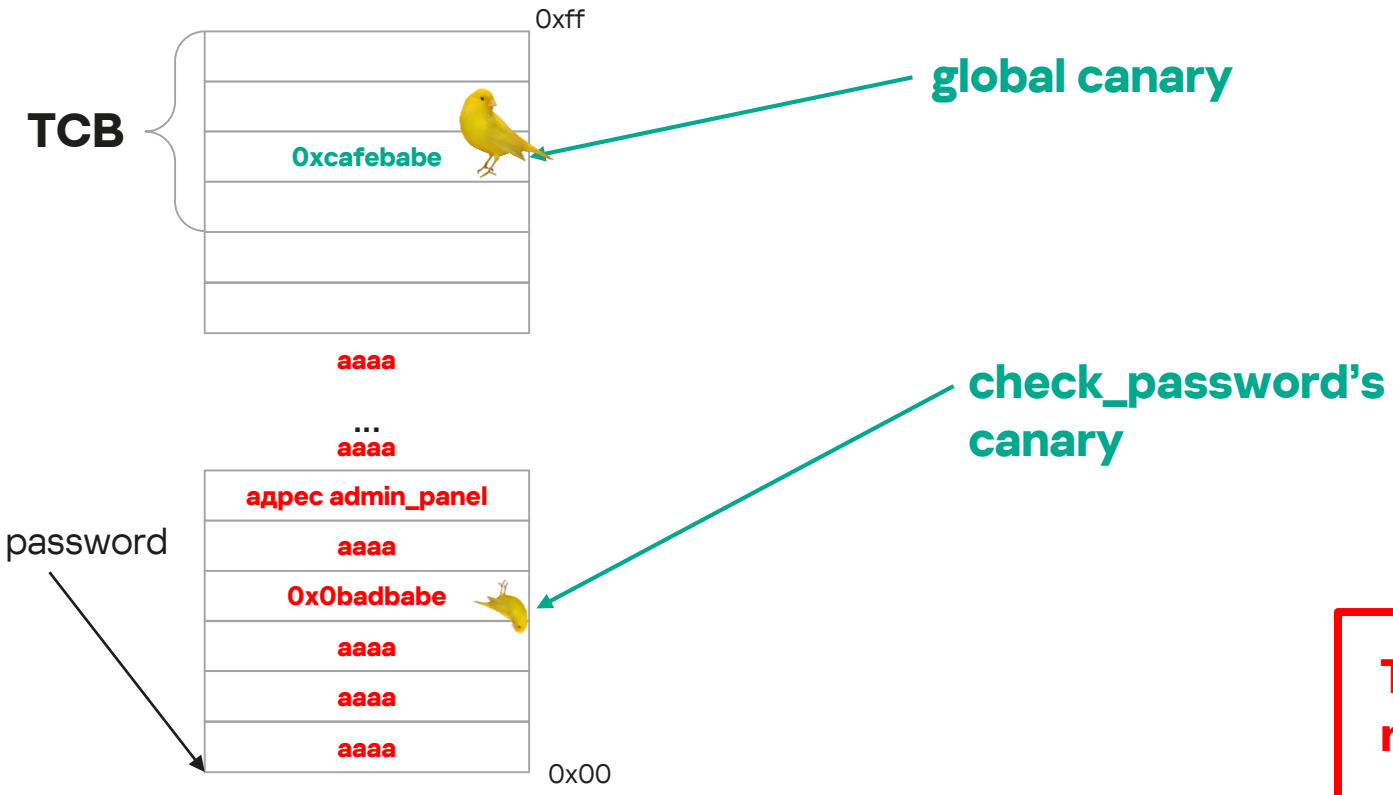
TCB is writable memory!

Погружение в Thread Local Storage



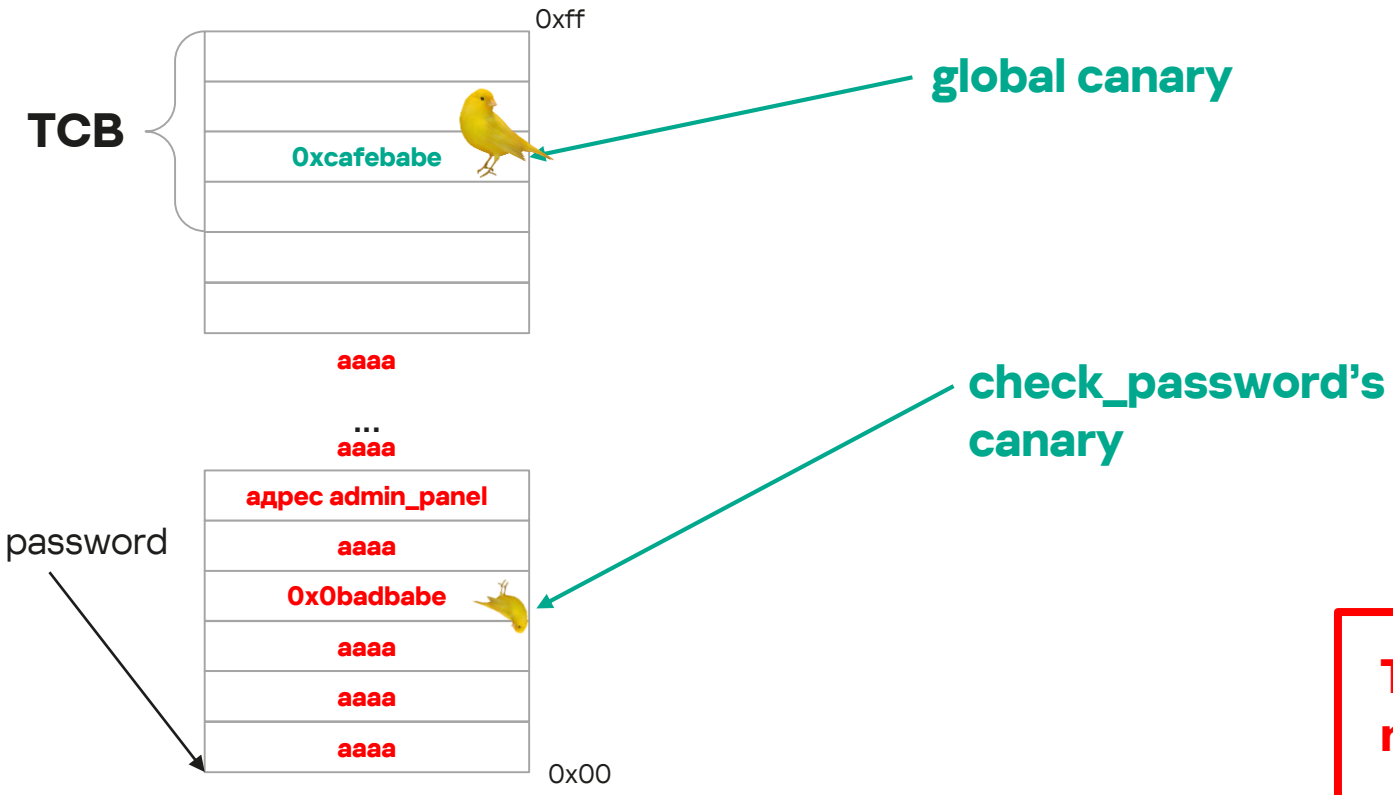
TCB is writable memory!

Погружение в Thread Local Storage



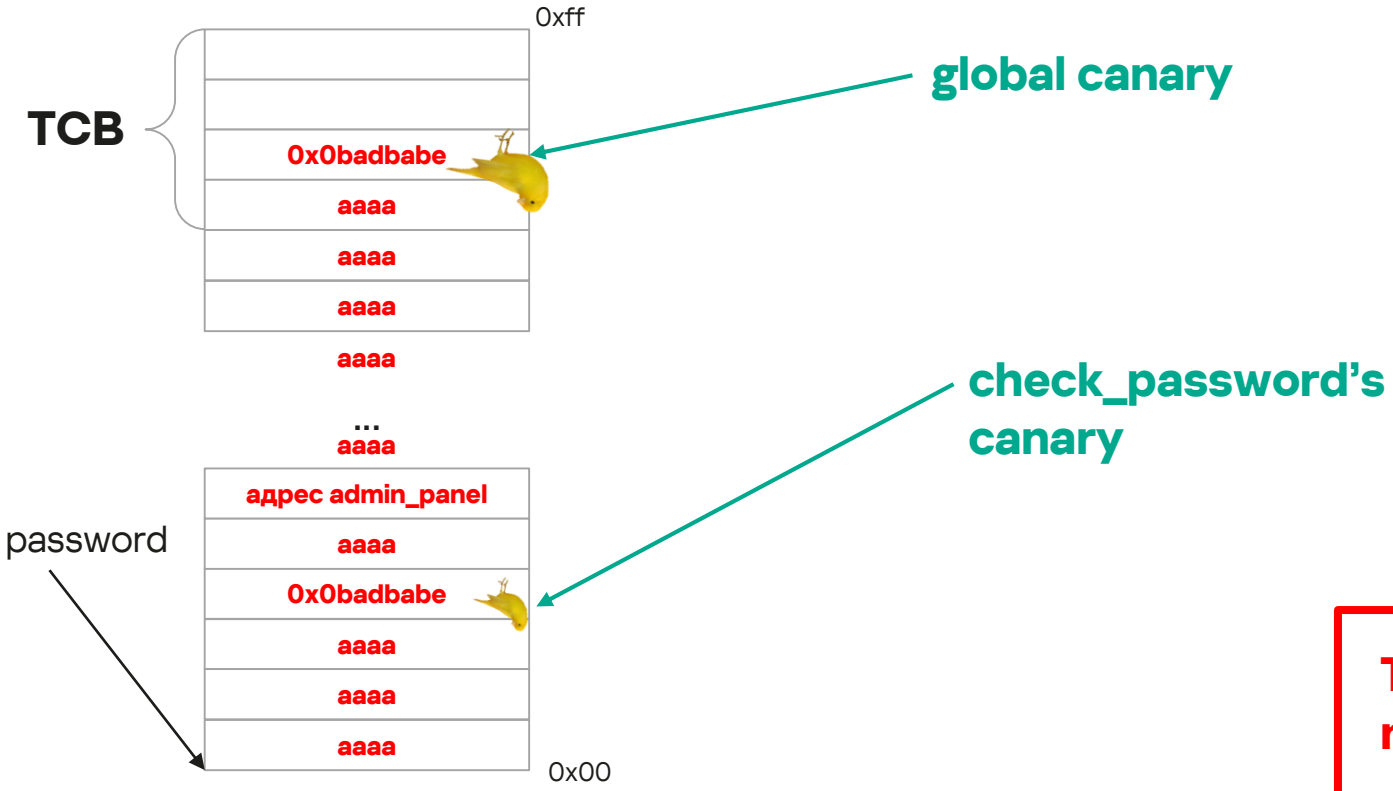
TCB is writable memory!

Погружение в Thread Local Storage



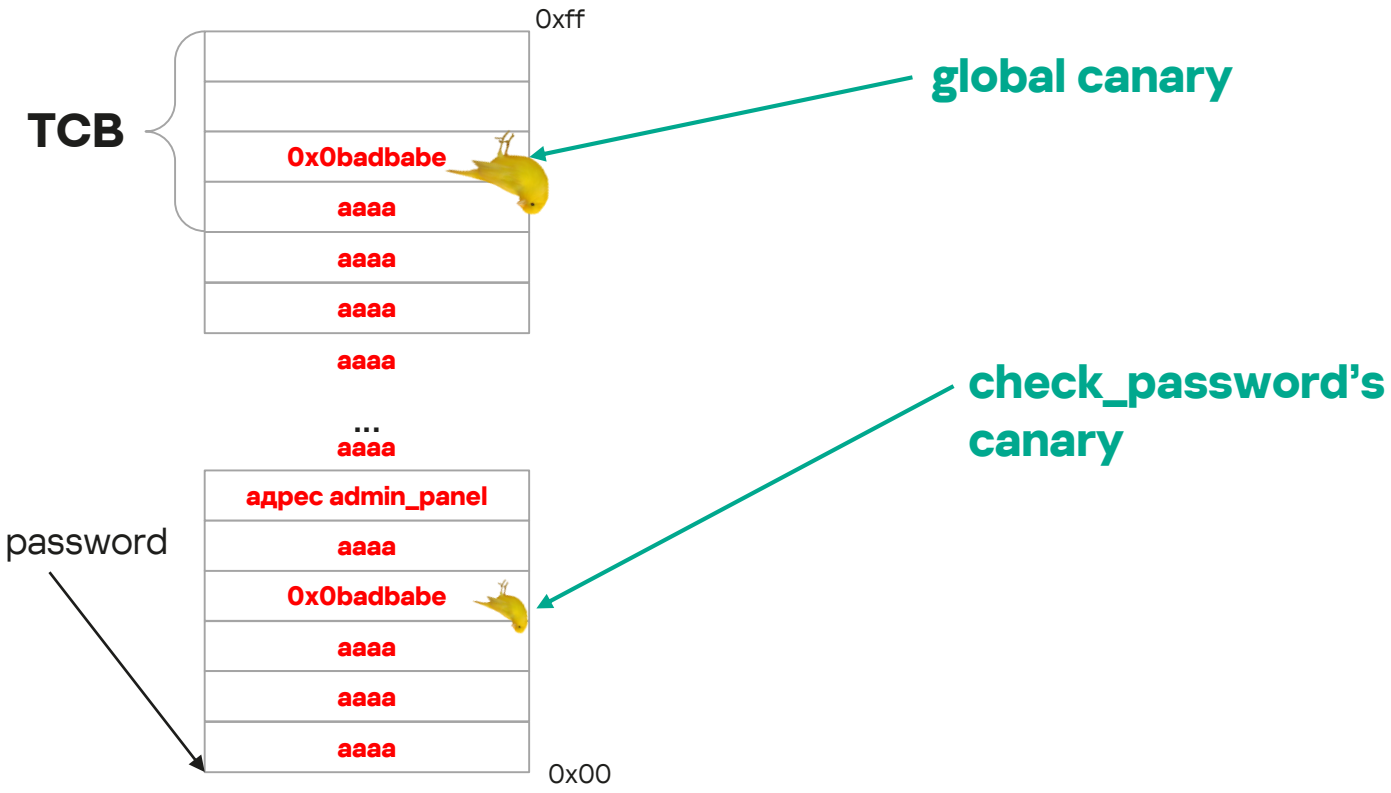
TCB is writable memory!

Погружение в Thread Local Storage



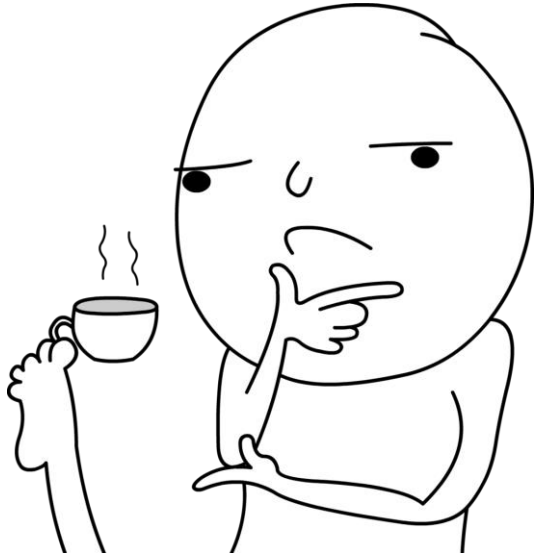
TCB is writable memory!

Погружение в Thread Local Storage



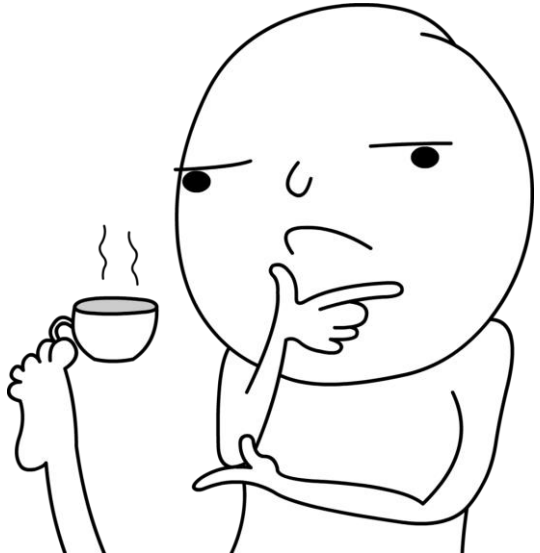
Пример

Почему TLS?



Почему TLS?

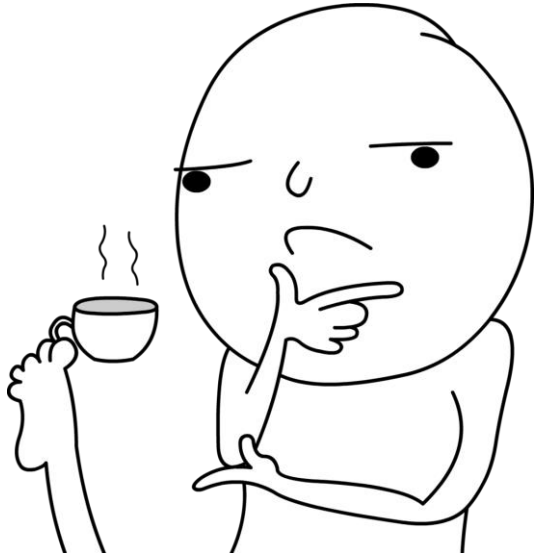
- Быстрый доступ к памяти => маленький overhead при включении канарейки



Почему TLS?

- Быстрый доступ к памяти => маленький overhead при включении канарейки

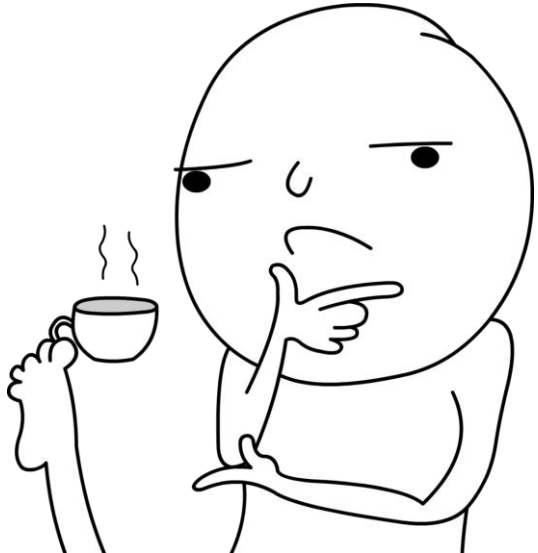
- Старая реализация



Почему TLS?

- Быстрый доступ к памяти => маленький overhead при включении канарейки

- Старая реализация **(сейчас бы использовалась rip адресация)**



Защита канарейки

	Перезапись	Чтение
Windows	data section (ntdll.dll) ✅	
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы ✅	
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**, ❌ иначе в **data.rel.ro** секции ✅	

Узнаём канарейку

Как узнать канарейку?

- Предсказать канарейку

Как узнать канарейку?

- Предсказать канарейку
- Угадать (fork)

Как узнать канарейку?

- Предсказать канарейку
- Угадать (fork)
- Прочитать

Как узнать канарейку?

- Предсказать канарейку
- Угадать (fork)
- Прочитать
  - со стека

Как узнать канарейку?

- Предсказать канарейку
- Угадать (fork)
- Прочитать
  - со стека
  - глобальную канарейку

Как узнать канарейку?

- **Предсказать канарейку**
- Угадать (fork)
- Прочитать
  - со стека
  - глобальную канарейку

Как узнать канарейку?

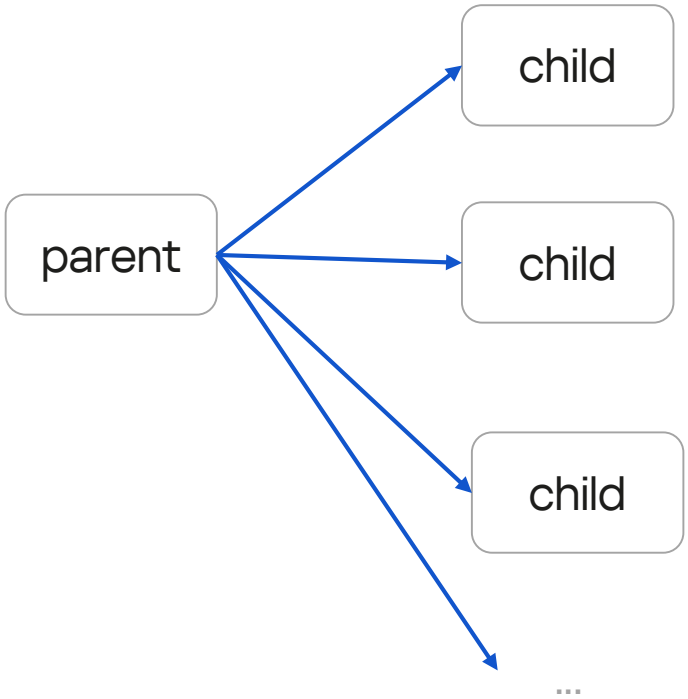
- **Предсказать канарейку**
- Угадать (fork)
- Прочитать
  - со стека
  - глобальную канарейку

Как узнать канарейку?

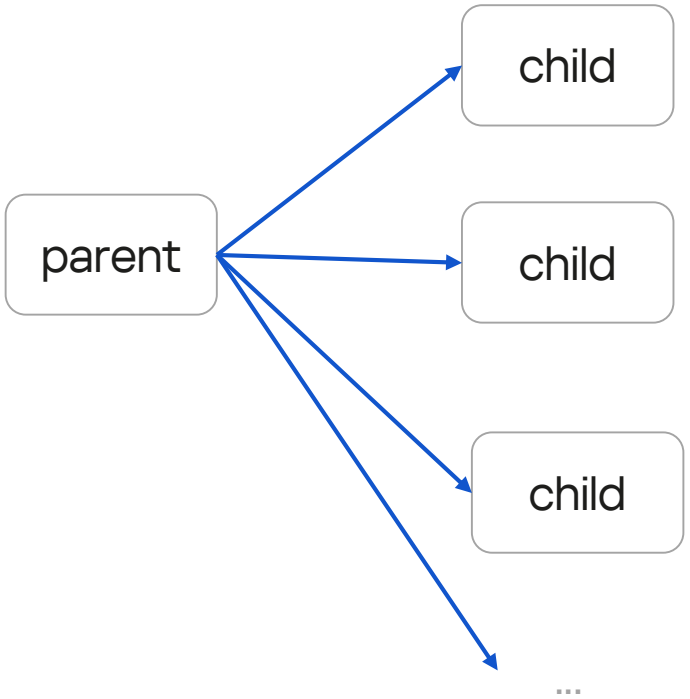
- Предсказать канарейку
- **Угадать (fork)**
- Прочитать
  - со стека
  - глобальную канарейку

Угадываем канарейку (fork)

Угадываем канарейку (fork)

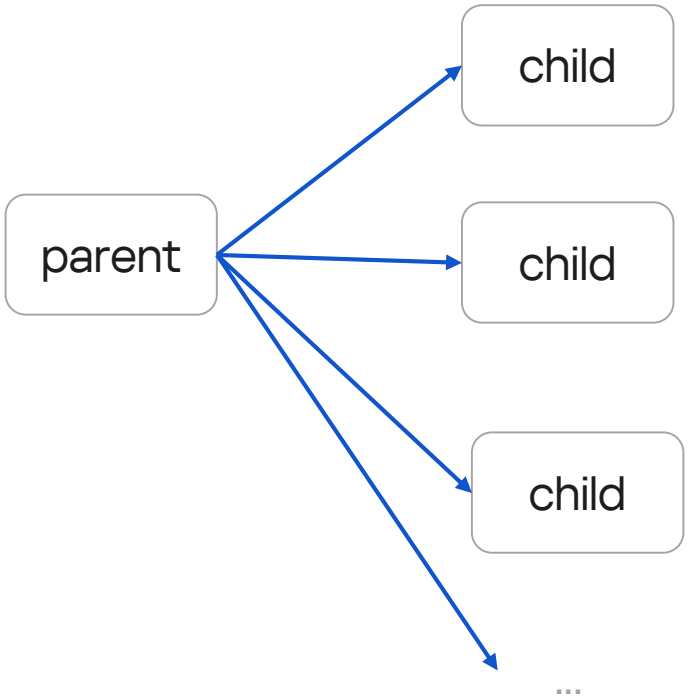


Угадываем канарейку (fork)



При копировании процесса (fork) - значение канарейки не меняется!

Угадываем канарейку (fork)



При копировании процесса (fork) - значение канарейки не меняется!

Как узнать канарейку?

- Предсказать канарейку
- Угадать (fork)
- **Прочитать**
  - **со стека**
  - **глобальную канарейку**

Читаем канарейку со стека

Читаем канарейку (со стека)

Читаем канарейку (со стека)

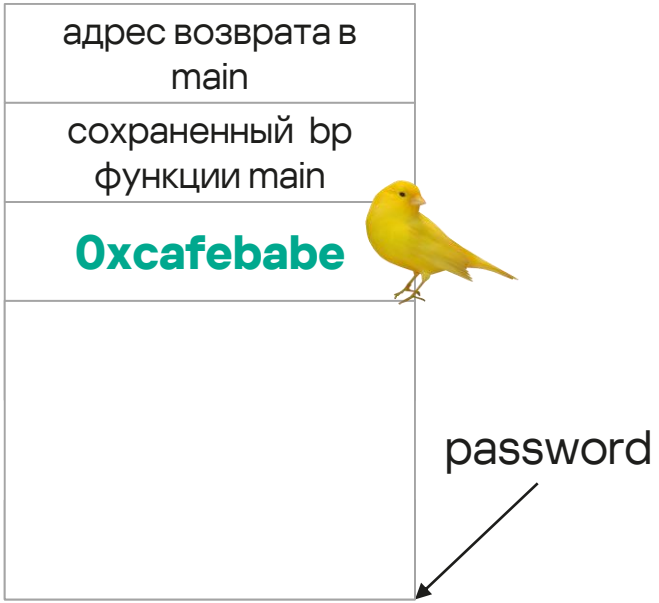
- недостаток программы, который позволяет читать стек

Читаем канарейку (со стека)

- недостаток программы, который позволяет читать стек
- **например:** использование функции strcpy

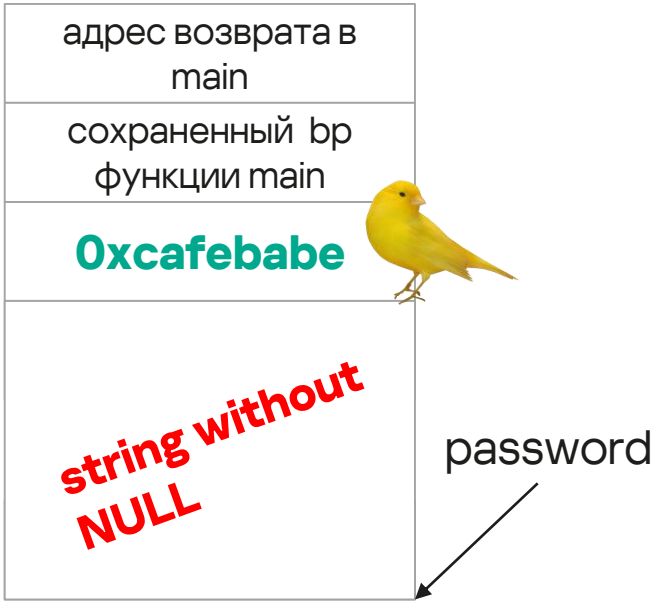
Читаем канарейку (со стека)

- недостаток программы, который позволяет читать стек
- **например:** использование функции strcpy



Читаем канарейку (со стека)

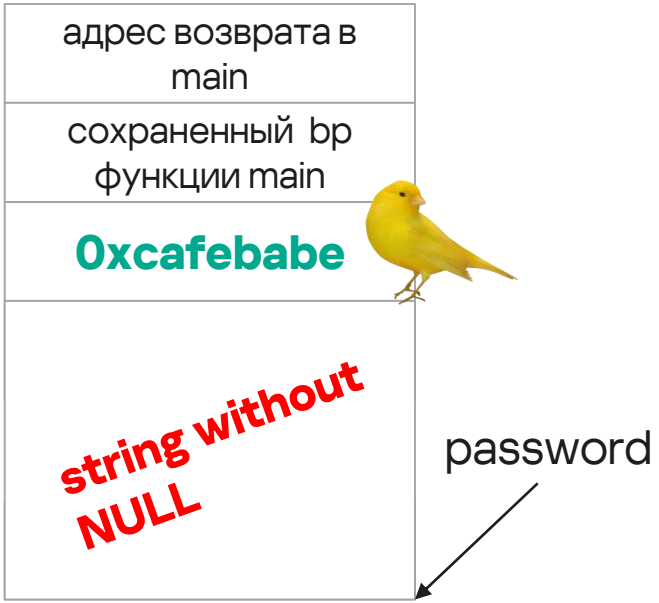
- недостаток программы, который позволяет читать стек
- **например:** использование функции strcpy



Читаем канарейку (со стека)

- недостаток программы, который позволяет читать стек
- **например:** использование функции strcpy

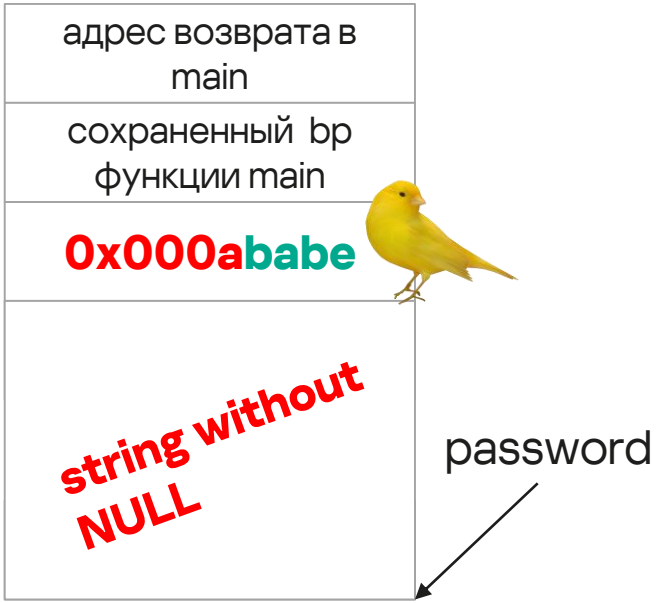
strcpy(password, your_buf);



Читаем канарейку (со стека)

- недостаток программы, который позволяет читать стек
- **например:** использование функции strcpy

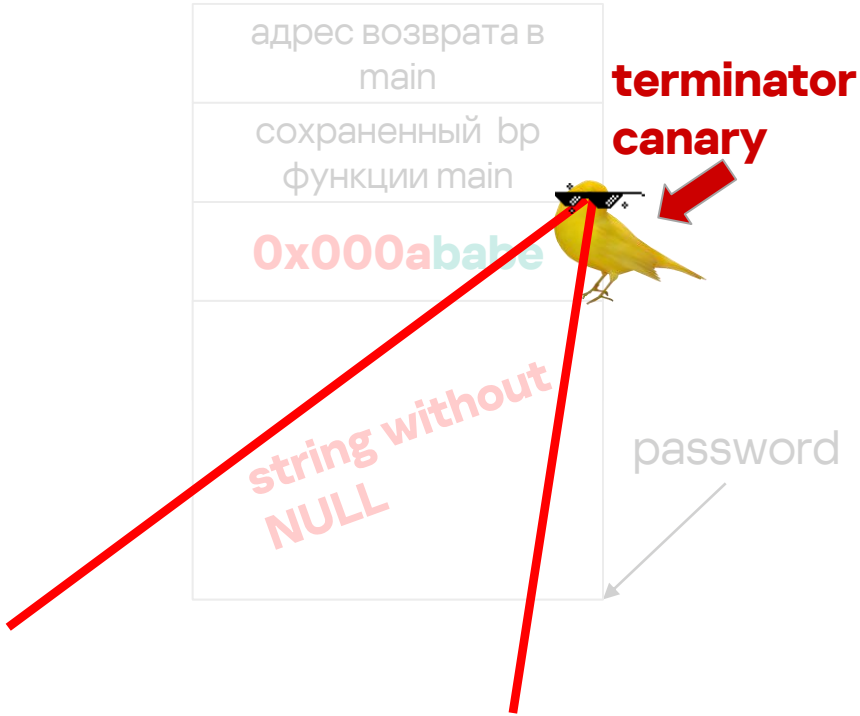
strcpy(password, your_buf);



Читаем канарейку (со стека)

- недостаток программы, который позволяет читать стек
- **например:** использование функции strcpy

```
strcpy(password, your_buf);
```



Читаем глобальную канарейку

Читаем канарейку (глобальную)

Читаем канарейку (глобальную)

- знаем адрес глобальной канарейки

Читаем канарейку (глобальную)

- знаем адрес глобальной канарейки
- есть возможность чтения этого адреса

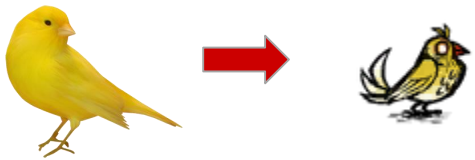
Читаем канарейку (глобальную)

- знаем адрес глобальной канарейки **<- затрудняет ASLR**
- есть возможность чтения этого адреса

Как защититься от чтения?

Как защититься от чтения?

Используем манглинг канарейки!



Xor-им канарейку с другим числом

Xor-им канарейку с другим числом

- xor с ebp (windows)

Xor-им канарейку с другим числом

- xor с ebp (windows)
- xor с return address (openbsd)

Xor-им канарейку с другим числом

- xor с ebp (windows)
- xor с return address (openbsd) **<- называют свой механизм RETGUARD**

Xor-им канарейку с другим числом

- xor с ebp (windows)
- xor с return address (openbsd)

Xor-им канарейку с другим числом

- xor с ebp (windows)
- xor с return address (openbsd)

- per-function уникальность

Xor-им канарейку с другим числом

- xor с ebp (windows)
- xor с return address (openbsd)

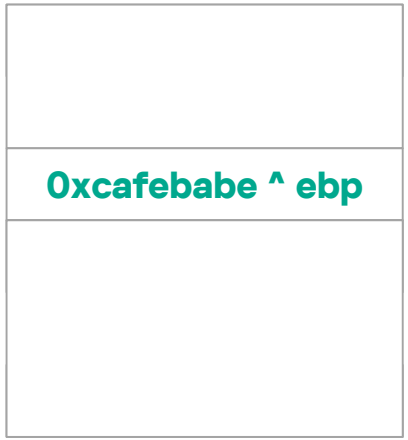
- per-function уникальность
- сложнее использовать полученную информацию

Почему сложнее использовать xor-канарейку?

Почему сложнее использовать xor-канарейку?

ebp = 0x1234

...



ebp = 0x4321

Почему сложнее использовать xor-канарейку?

смогли прочитать!

Почему сложнее использовать xor-канарейку?

смогли прочитать!

но на другом стеке использовать не можем, так как не знаем ebp!

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅		
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅		
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅		

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅	xor ebp ✅	
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅		
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅		

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅	xor ebp ✅	нет fork ✅
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅		
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅		

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅	xor ebp ✅	нет fork ✅
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅	xor return address ✅	
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅		

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅	xor ebp ✅	нет fork ✅
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅	xor return address ✅	❌
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅		

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅	xor ebp ✅	нет fork ✅
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅	xor return address ✅	❌
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅	❌	

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅	xor ebp ✅	нет fork ✅
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅	xor return address ✅	❌
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅	❌	❌

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅	xor ebp ✅	нет fork ✅
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅	xor return address ✅	❌
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅	❌	❌
KasperskyOS			

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅	xor ebp ✅	нет fork ✅
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅	xor return address ✅	❌
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅	❌	❌
KasperskyOS	data.rel.ro ✅		

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll✅	xor ebp ✅	нет fork ✅
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы✅	xor return address ✅	❌
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**,❌ иначе в **data.rel.ro** секции✅	❌	❌
KasperskyOS	data.rel.ro ✅	xor ebp ✅	

Защита канарейки

	Перезапись	Чтение	Угадывание
Windows	data section (ntdll.dll ✅	xor ebp ✅	нет fork ✅
OpenBSD	ввели свою секцию, которая инициализируется ядром ОС и всегда readonly при старте программы ✅	xor return address ✅	❌
Glibc(Linux)	если TLS(Thread Local Storage) реализован для архитектуры, то в **TLS**, ❌ иначе в **data.rel.ro** секции ✅	❌	❌
KasperskyOS	data.rel.ro ✅	xor ebp ✅	нет fork ✅

Что делать?

Что делать?

- **Проверять наличие канарейки(и других харденингов) в проекте:**

Что делать?

- **Проверять наличие канарейки(и других харденингов) в проекте:**
  - флаги компиляции

Что делать?

- **Проверять наличие канарейки(и других харденингов) в проекте:**
  - флаги компиляции
  - тулинг для проверки бинаря (hardening-check)

Что делать?

- **Проверять наличие канарейки(и других харденингов) в проекте:**
  - флаги компиляции
  - тулинг для проверки бинаря (hardening-check)

- **Обратить внимание на другие харденеги:**
  - shadow stack (with hw support)

Что делать?

- **Проверять наличие канарейки(и других харденингов) в проекте:**
  - флаги компиляции
  - тулинг для проверки бинаря (hardening-check)

- **Обратить внимание на другие харденеги:**
  - shadow stack (with hw support)

- **Не допускать ошибок работы с памятью:**

Что делать?

- **Проверять наличие канарейки(и других харденингов) в проекте:**
  - флаги компиляции
  - тулинг для проверки бинаря (hardening-check)

- **Обратить внимание на другие харденеги:**
  - shadow stack (with hw support)

- **Не допускать ошибок работы с памятью:**
  - статический анализ 

Что делать?

- **Проверять наличие канарейки(и других харденингов) в проекте:**
  - флаги компиляции
  - тулинг для проверки бинаря (hardening-check)

- **Обратить внимание на другие харденеги:**
  - shadow stack (with hw support)

- **Не допускать ошибок работы с памятью:**
  - статический анализ 

  - фаззиг с санитарами (asan, ...) 

А канарейка не проста!

Ссылки

1. Краткий экскурс в канарейки
2. Описание различных атак на канарейку
3. Предсказание случайности. Изучаем ASLR в Linux и GNU libc, обходим защиту адресного пространства и stack canary
4. Windows 8 ate my cookie
5. Добавление xor канарейки в llvm для windows

Так ли проста стековая канарейка?

Мария Недяк

Разработчик группы харденингов KasperskyOS

@MSH_SMLV