Preventing brute force attacks against stack canary protector on networking servers

Hector Marco-Gisbert, Ismael Ripoll
Universitat Politècnica de València (Spain)

2013 IEEE 12th International Symposium on Network Computing and Applications

August 22-24, 2013
Table of contents

1. Introduction

2. The problem: Network servers and their threats

3. How we solve it: RAF SSP

4. Conclusions
Buffer overflows are still a major software threat. [Top 25]

One of the most effective protection technique is the *stack canary protector (SSP)*. Currently employed in most servers: Apache, Lighthttp, etc.

Unfortunately, the SSP on network servers is prone to brute force attacks.

We have extended the SSP technique to prevent brute force attacks at zero cost: temporal, spacial and implementational!
How the Stack Canary Protector works

- The canary is a random value placed on the stack to detect buffer overflows.
- When a buffer overflows the canary is corrupted.
- If the verification of the canary fails → abort()
The canary is a random value placed on the stack to detect buffer overflows.

When a overflows the canary is corrupted.

If the verification of the canary fails → abort()
How the Stack Canary Protector works

- The canary is a random value placed on the stack to detect buffer overflows.

- When a buffer overflows, the canary is corrupted.

- If the verification of the canary fails → \texttt{abort()}

\texttt{gs:0x14}
Forking servers

- Processes created with `fork()` inherit most of its father state.
- Father and children have the same canary-reference value.
Forking servers

- Processes created with `fork()` inherit most of its father state.
- Father and children have the same canary-reference value.

When the attacker guesses an incorrect value, the child is killed by the SSP and a new child with the same canary is started.

The attack is modelled as sampling without replacement.
Brute force attacks

Sampling without replacement allows to build brute force attacks.

- **Full search attack:**
  - The frame-canary word is overwritten on each trial.
  - If the guessed word is not correct → abort().
  - 100% success on **93 hours** and 46 hours on average.

- **Byte for byte attack:**
  - Attackers control the number of overwritten bytes.
  - Overwrite only the first stack canary byte until child does not crash. (same for following bytes).
  - 100% success on **15 sec.** and 7 sec. on average.

Note: Some systems (i.e. x86) set to zero most significant byte.
Observations and facts

Facts:

- There is only one single reference-canary per process.
- The canary integrity check is done at the end of each function before returning.
- Upon return, only the current frame-canary is checked.
- Each child process of a network server is an error confinement region.
Observations and facts

Facts:

- There is only one single reference-canary per process.
- The canary integrity check is done at the end of each function before returning.
- Upon return, only the current frame-canary is checked.
- Each child process of a network server is an *error confinement region*.

Observation:

- After a `fork()` call, the child process terminates by calling `exit()`.
Core idea

“Renew the reference-canary of the child right after the \texttt{fork()}”
Preventing brute force attacks against stack canary protector on networking servers

Hector Marco

How we solve it: RAF SSP

Core idea

“Renew the reference-canary of the child right after the `fork()`”

- When the attacker guesses an incorrect value, the child is killed by the SSP and a new child with a new canary is started.
- As a result, brute force attacks can not be built.
void server()
{
    ...  
    while (1) {
        client = WaitClient();
        if (fork() == 0) {
            renewCanary();
            Child_Main();
        }
    }
} ...  
void Child_Main()
{
    Attend();
    _exit();
}
void Attend()
{
    bar();
}
void bar()
{
    qux();
}
void aux(){} 

void server()
{
    ... 
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
            
        }
        }
    }
    }

void Child_Main()
{
    Attend();
    _exit();
}
void Attend()
{
    bar();
}
void bar()
{
    qux();
}
void aux()
{ }
void server(){
    ...
    while(1) {
        client=WaitClient();
        if (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
    }
} ...
void Child_Main(){
    Attend();
    _exit();
}
void Attend(){
    bar();
}
void bar(){
    qux();
}
void aux(){} ;
void server()
{
...  
    while(1) {
        client=WaitClient();
        if (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }

} ...
}

void Child_MAIN()
{
    Attend();
    _exit();
}

void Attend()
{
    bar();
}

void bar()
{
    qux();
}

void aux(){}
void server()
{
    ...
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
} ...
void Child_Main()
{
    Attend();
    _exit();
}
void Attend()
{
    bar();
}
void bar()
{
    qux();
}
void aux(){} 

void server(){
    ...
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
} ...

void Child_Main(){
    Attend();
    _exit();
}

void Attend(){
    bar();
}

void bar(){
    qux();
}

void aux(){
}
Preventing brute force attacks against stack canary protector on networking servers

Hector Marco

How we solve it: RAF SSP

Stack evolution example: 7/21

```c
void server()
{
    ... 
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
}
} ... 

void Child_Main()
{
    Attend();
    _exit();
}
void Attend()
{
    bar();
}
void bar()
{
    qux();
}
void qux()
{   }
```
void server()
{
    ...
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
} ...

void Child_Main()
{
    Attend();
    _exit();
}

void Attend()
{
    bar();
}

void bar()
{
    qux();
}

void qux()
{
Stack evolution example: 9/21

```c
void server()
{
    ...
    while(1) {
        client=WaitClient();
        if (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
} ...

void Child_Main()
{
    Attend();
    _exit();
}
void Attend()
{
    bar();
}
void bar()
{
    qux();
}
void qux()
{ }
```
Stack evolution example: 10/21

```c
void server(){
    ...
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
}
}
void Child_Main(){
    Attend();
    _exit();
}
void Attend(){
    bar();
}
void bar(){
    qux();
}
void qux(){
}
```
void server()
{
    ...
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
} ...
void Child_Main()
{
    Attend();
    _exit();
}
void Attend()
{
    bar();
}
void bar()
{
    qux();
}
void qux()
{
}
Stack evolution example: 12/21

```c
void server(){
    ...
    while(1) {
        client=GetInt();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
} ...
void Child_Main(){
    Attend();
    _exit();
}
void Attend(){
    bar();
}
void bar(){
    qux();
}
void qux(){  };
```
void server()
{
  ...
  while(1) {
    client=WaitClient();
    If (fork()==0) {
      renewCanary();
      Child_Main();
    }
  }
} ...
void Child_Main()
{
  Attend();
  _exit();
}
void Attend()
{
  bar();
}
void bar()
{
  qux();
}
void qux(){ }:
Stack evolution example: 14/21

```c
void server(){
    ...
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
}

void Child_Main(){
    Attend();
    _exit();
}

void Attend(){
    bar();
}

void bar(){
    qux();
}

void aux(){ }
```
void server(){
    ...
    while(1) {
        client=\texttt{WaitClient}();
        If (\texttt{fork()==0}) {
            renewCanary();
            Child\_Main();
        }
    }
    }
}

void Child\_Main(){
    Attend();
    \_exit();
}

void Attend(){
    bar();
}

void bar(){
    \texttt{qux}();
}

void aux(){ }:
Preventing brute force attacks against stack canary protector on networking servers

Hector Marco

How we solve it: RAF SSP

Stack evolution example: 16/21

```c
void server(){
    ...
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
}
} ...

void Child_Main(){
    Attend();
    _exit();
}

void Attend(){
    bar();
}

void bar(){
    qux();
}

void aux(){ }
```
void server()
{
    ...
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
} ...

void Child_Main()
{
    Attend();
    _exit();
}

void Attend()
{
    bar();
}

void bar()
{
    qux();
}

void qux() ;
Stack evolution example: 18/21

```c
void server() {
  ... 
  while(1) {
    client=WaitClient();
    if (fork()==0) {
      renewCanary();
      Child>Main();
    }
  }
}
} ... 

void Child>Main() {
  Attend();
  _exit();
}
void Attend() {
  bar();
}
void bar() {
  qux();
}
void aux() { }
```
Stack evolution example: 19/21

```c
void server()
{
    ...
    while(1) {
        client=WaitClient();
        If (fork()==0) {
           renewCanary();
           Child_Main();
        }
    }
    ...
}

void Child_Main()
{
    Attend();
    _exit();
}

void Attend()
{
    bar();
}

void bar()
{
    qux();
}

void aux()
{
    ...
}
```

Father execution

Father loops to the while

Child's execution

Point of no return

Random
void server()
{
    ...  
            while(1) {
                client=\textbf{WaitClient}();
                    If (fork()==0) {
                        renewCanary();
                        Child\_Main();
                    }
            }
    ...}
void Child\_Main()
{
    Attend();
    \_\_exit();
}
void Attend()
{
    bar();
}
void bar()
{
    qux();
}
void aux()
{  }

Father execution

Father loops to the while

Child's execution

Point of no return

RANDOM
void server(){
    ... 
    while(1) {
        client=WaitClient();
        If (fork()==0) {
            renewCanary();
            Child_Main();
        }
    }
    ... 
    void Child_Main(){
        Attend();
        _exit();
    }
    void Attend(){
        bar();
    }
    void bar(){
        qux();
    }
    void qux(){  };

    void renewCanary(void) {
        uintptr_t stack_chk_guard = _dl_setup_stack chk guard (NULL);
        #ifdef THREAD_SET_STACK_GUARD
            THREAD_SET_STACK_GUARD (stack_chk_guard);
        #else
            __stack chk guard = stack chk guard;
        endif
    }
Preventing brute force attacks against stack canary protector on networking servers

Hector Marco

How we solve it: RAF SSP

Proof of concept code: Pre-load shared library

```c
pid_t (*native_fork) (void);

static void __raf_fork_init(void) {
    native_fork = dlsym(RTLD_NEXT, "fork");
    if (NULL == native_fork)
        error_exit();
}

pid_t fork(void) {
    pid_t pid;
    if (native_fork==NULL)
        __raf_fork_init();
    pid = native_fork();
    if (pid == 0)
        renew_reference_stack_guard();
    return pid;
}

#define THREAD_SET_STACK_GUARD(x) \
    asm("movl%0,%\%gs:0x14":"r"(x):"memory");

static void renew_reference_stack_guard(void) {
    union {
        uintptr_t num;
        unsigned char bytes[sizeof (uintptr_t)];
    } ret;
    ret.num = 0;
    const size_t ranb = sizeof(ret.bytes) - 1;
    int fd = __open("/dev/urandom", O_RDONLY);
    if (fd >= 0) {
        if (__read(fd, ret.bytes + 1, ranb) == ranb)
            THREAD_SET_STACK_GUARD(ret.num);
        __close (fd);
    }
}
```
Override fork

```c
pid_t (*native_fork)(void);

static void __raf_fork_init(void) {
    native_fork = dlsym(RTLD_NEXT, "fork");
    if (NULL == native_fork)
        error_exit();
}

static void renew_reference_stack_guard(void) {
    union {
        uintptr_t num;
        unsigned char bytes[sizeof(uintptr_t)];
    } ret;
    ret.num = 0;
    const size_t ranb = sizeof(ret.bytes) - 1;
    int fd = __open("/dev/urandom", O_RDONLY);
    if (fd >= 0) {
        if ((__read(fd, ret.bytes + 1, ranb) == ranb)
            THREAD_SET_STACK_GUARD(ret.num);
        __close (fd);
    }
}
```

```c
#define THREAD_SET_STACK_GUARD(x) \
    asm("mov\%0,\%gs:0x14":"r"(x):"memory");
```

Proof of concept code: Pre-load shared library
Proof of concept code: Pre-load shared library

### Override fork

```c
pid_t (*native_fork) (void);

static void __raf_fork_init(void) {
    native_fork = dlsym(RTLD_NEXT, "fork");
    if (NULL == native_fork)
        error_exit();
}

pid_t fork(void) {
    pid_t pid;
    if (native_fork==NULL)
        __raf_fork_init();
    pid = native_fork();
    if (pid == 0)
        renew_reference_stack_guard();
    return pid;
}
```

### Return random bytes

```c
#define THREAD_SET_STACK_GUARD(x) \  
asm("movu%0,%%gs:0x14":"r"(x):"memory");

static void renew_reference_stack_guard(void) {
    union {
        uintptr_t num;
        unsigned char bytes[sizeof (uintptr_t)];
    } ret;
    ret.num = 0;
    const size_t ranb = sizeof(ret.bytes) - 1;
    int fd = __open("/dev/urandom", O_RDONLY);
    if (fd >= 0) {
        if (__read(fd, ret.bytes + 1, ranb) == ranb)
            THREAD_SET_STACK_GUARD(ret.num);
        __close (fd);
    }
}```
Preventing brute force attacks against stack canary protector on networking servers

How we solve it: RAF SSP

Proof of concept code: Pre-load shared library

Override fork

```c
pid_t (*native_fork) (void);

static void __raf_fork_init(void) {
    native_fork = dlsym(RTLD_NEXT, "fork");
    if (NULL == native_fork)
        error_exit();
}

pid_t fork(void) {
    pid_t pid;
    if (native_fork==NULL)
        __raf_fork_init();
    pid = native_fork();
    if (pid == 0)
        renew_reference_stack_guard();
    return pid;
}
```

Renew referece-canary

```c
#define THREAD_SET_STACK_GUARD(x) \
    asm("movl%0,%gs:0x14":"r"(x):"memory");
```

Return random bytes

```c
static void renew_reference_stack_guard(void) {
    union {
        uintptr_t num;
        unsigned char bytes[sizeof (uintptr_t)];
    } ret;
    ret.num = 0;
    const size_t ranb = sizeof(ret.bytes) - 1;
    int fd = __open("/dev/urandom", O_RDONLY);
    if (fd >= 0) {
        if (__read(fd, ret.bytes + 1, ranb) == ranb)
            THREAD_SET_STACK_GUARD(ret.num);
        __close (fd);
    }
}
```
Standard SSP vs RAF SSP (1/2)

Standard SSP:
- Byte for byte: guessed in seconds.
- Full search: possible locally and even remotely.
- The values already guessed **can be discarded**.

RAF SSP:
- Trial and test: The attacker can guess the value of the canary as many times as they need.
- The values already guessed **can not be discarded**.


### Standard SSP vs RAF SSP (2/2)

A 32 bit system example:

- On most systems the canary have 24 bits of entropy.
- \(2^{24} = 16\) million of different canary values.
- 50 attempts per second attack.

<table>
<thead>
<tr>
<th></th>
<th>Standard SSP</th>
<th>RAF SSP</th>
<th>Mean increased by a factor of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to break in</td>
<td>100%</td>
<td>Mean</td>
<td>100%</td>
</tr>
<tr>
<td>Standard SSP</td>
<td>15 sec.</td>
<td>7 sec</td>
<td>∞</td>
</tr>
<tr>
<td>SSP_bfb</td>
<td>93 hours</td>
<td>46 hours</td>
<td>∞</td>
</tr>
<tr>
<td>SSP_full</td>
<td>93 hours</td>
<td>46 hours</td>
<td>∞</td>
</tr>
</tbody>
</table>

SSP_bfb: Byte for byte attack to the SSP  
SSP_full: Full search attack to the SSP.
Preventing brute force attacks against stack canary protector on networking servers

How we solve it: RAF SSP

RAF SSP combined with other protection techniques

Most systems implement two other protection techniques against memory errors, besides the SSP:

- **NX or DEP**: Data Execution Protection.
- **ASLR**: Address Space Layout Randomization.

### Standard SSP

- SSP
- NX
- ASLR

### RAF SSP

- RAF SSP
- NX
- ASLR

Trial and test
Preventing brute force attacks against stack canary protector on networking servers

How we solve it: RAF SSP

RAF SSP combined with NX + ASLR

A 32 bit system example:

- SSP with $2^{24} = 16$ million of different canary values.
- NX which force to re-use already mapped code.
- ASLR with $2^8 = 256$ different ASLR offsets.
- 50 attempts per second.

<table>
<thead>
<tr>
<th></th>
<th>Standard SSP</th>
<th>RAF SSP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time to break in</td>
<td>Mean</td>
</tr>
<tr>
<td>Time to break in</td>
<td>100%</td>
<td>Mean</td>
</tr>
<tr>
<td>SSP_bfb + ASLR</td>
<td>20 sec.</td>
<td>10 sec</td>
</tr>
<tr>
<td>SSP_full + ASLR</td>
<td>93 hours</td>
<td>46 hours</td>
</tr>
</tbody>
</table>

SSP_bfb: Byte for byte attack to the SSP
SSP_full: Full search attack to the SSP.
Conclusions

- RAF SSP prevents brute force attacks against SSP, specially the dangerous byte-for-byte attack.

- Also prevents brute force attacks against ASLR.

- It can be implemented easily.

- And has a negligible overhead.

- Validated on real network servers: apache2, lighttpd, samba, etc.

- We expect RAF SSP will be widely use soon.
Questions ?