On the effectiveness of NX, SSP, RenewSSP and ASLR against stack buffer overflows

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

2014 IEEE 13th International Symposium on Network Computing and Applications

August 21-23, 2014
On the effectiveness of NX, SSP, RenewSSP and ASLR against stack buffer overflows
Hector Marco

Table of contents

1 Motivation

2 Introduction
   • Stack buffer overflow vulnerabilities
   • Type of severs
   • Protection techniques

3 Threats
   • Bypassing NX, SSP, RenewSSP and ASLR

4 Analysis of the protection techniques
   • Single process
   • Inetd server
   • Forking server

5 Results and conclusions
Motivation

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

- The NX, SSP, RenewSSP and ASLR protection techniques:
  - Try to defeat/mitigate stack buffer overflows.
  - Used on modern operating systems like Windows, Linux, Android etc.,

- **New attack vectors**, not considered when these techniques were developed, makes necessary to reassess their effectiveness to avoid a false sense of security.

- We reassess the NX, SSP, RenewSSP and ASLR exploiting a stack buffer overflow on: Single process, Inted and Forking servers.
Stack buffer overflow vulnerabilities

- The study has been focused on the stack buffer overflow vulnerabilities, considering multiple attack vectors.

```c
void func1(char *src, int lsrc) {
    char buff[48];
    int i = 0;
    ...
    memcpy(buff, src, lsrc);
    ...
}
```

Listing 1: memcpy example.

```c
void func2(char *str, int lstr) {
    char buff[48];
    int i = 0;
    ...
    for (i = 0; i < lstr; i++) {
        if (str[i] != '\n')
            buff[lbuff++] = str[i];
    }
    ...
}
```

Listing 2: loop example.

- Exploit successfully these vulnerabilities depends on the kind of server.
- It is more reliable to exploit these vulnerabilities on forking servers.
Example 1/3

Higher addresses

function1()
stack frame

function2()
stack frame

Lower addresses

%gs:0x14
reference canary

Stack growth
Example 2/3

%gs:0x14

reference canary

Higher addresses

function1()

stack frame

buffers

saved reg 1

saved reg 2

Lower addresses

Stack growth

%gs:0x14

reference canary

frame canary

arg 3

arg 2

arg 1

return address

saved frame ptr
Example 3/3

%gs:0x14

reference canary

Higher addresses

NO!

function1()

stack frame

function2()

stack frame

Lower addresses

Stack growth

arg 3
arg 2
arg 1
return address
saved frame ptr
buffers
frame canary
saved reg 1
saved reg 2
arg 1
return address
saved frame ptr
buffers
frame canary
saved reg 1
Type of servers

**Single server:**

- An incorrect attempt attack $\rightarrow$ crash $\rightarrow$ service stopped.
- Little chances to break into the server but easy to do a DoS.
- No real servers use this model.
Type of servers

**Single server:**
- An incorrect attempt attack $\rightarrow$ crash $\rightarrow$ service stopped.
- Little chances to break into the server but easy to do a DoS.
- No real servers use this model.

**Inted server:**
- An incorrect attempt attack $\rightarrow$ crash $\rightarrow$ relaunch the service.
- Every attempt $\rightarrow$ renew all secrets. ($\text{fork()} + \text{exec()} \rightarrow \text{attend()}$)
- Paranoid servers (**SSH suit**) or services through the Inted (**ftpd**).
Type of servers

**Single server:**
- An incorrect attempt attack → crash → service stopped.
- Little chances to break into the server but easy to do a DoS.
- No real servers use this model.

**Inted server:**
- An incorrect attempt attack → crash → relaunch the service.
- Every attempt → renew all secrets. \((\text{fork()} + \text{exec()} \rightarrow \text{attend()}\))
- Paranoid servers (SSH suit) or services through the Inted (ftpd).

**Forking server:**
- An incorrect attempt attack → crash → use a new child.
- Every attempt → **not** renew all secrets. \((\text{fork()} \rightarrow \text{attend()}\))
- Most servers use it. Examples: Apache, lighttpd, etc.
Protection techniques

**NX or DEP:**
- Executable pages are not writable.
- Prevent the execution of the injected code.
Protection techniques

**NX or DEP:**
- Executable pages are not writable.
- Prevent the execution of the injected code.

**SSP:**
- Random value placed on the stack initially to protect the return address.
- Detects stack buffer overflows and aborts the execution.
Protection techniques

NX or DEP:
- Executable pages are not writable.
- Prevent the execution of the injected code.

SSP:
- Random value placed on the stack initially to protect the return address.
- Detects stack buffer overflows and aborts the execution.

ASLR:
- New process are loaded randomly in the main memory.
- Prevents attacks relying on the knowing absolute addresses.
Protection techniques

**NX or DEP:**
- Executable pages are not writable.
- Prevent the execution of the injected code.

**SSP:**
- Random value placed on the stack initially to protect the return address.
- Detects stack buffer overflows and aborts the execution.

**ASLR:**
- New process are loaded randomly in the main memory.
- Prevents attacks relying on the knowing absolute addresses.

**RenewSSP:**
- A recent modification of the SSP.
- Prevents SSP brute force attacks on forking servers.
Bypassing NX, SSP, RenewSSP and ASLR 1/3

NX/DEP:
- Using attacks that do not require to execute the injected code.
- Modern attacks do not inject code but use ROP, JOP etc.
Bypassing NX, SSP, RenewSSP and ASLR 1/3

**NX/DEP:**
- Using attacks that do not require to execute the injected code.
- Modern attacks do not inject code but use ROP, JOP etc.

**SSP-tat (SSP trial-and-test):**
- The canary value is replaced after each trial. *(sampling with replacement)*
- The attacker can try at will but can not discard already tested values.
Bypassing NX, SSP, RenewSSP and ASLR 1/3

**NX/DEP:**
- Using attacks that do not require to execute the injected code.
- Modern attacks do not inject code but use ROP, JOP etc.

**SSP-tat (SSP trial-and-test):**
- The canary value is replaced after each trial. *(sampling with replacement)*
- The attacker can try at will but cannot discard already tested values.

**SSP-bff (SSP brute-force-full):**
- The canary value is the same in every trial. *(sampling without replacement)*
- The attacker can build a brute force attack to obtain the canary.
Bypassing NX, SSP, RenewSSP and ASLR 2/3

**SSP-bfb** (SSP byte-for-byte):

- The canary value is the same in every trial. *(sampling *without* replacement)*
- The attacker can build a brute force attack but trying all possible values of each byte sequentially.
Bypassing NX, SSP, RenewSSP and ASLR 2/3

**SSP-bfb** *(SSP byte-for-byte):*
- The canary value is the same in every trial. *(sampling *without* replacement)*
- The attacker can build a brute force attack but trying all possible values of each byte sequentially.

**RenewSSP-tat** *(RenewSSP trial-and-test):*
- The canary value is replaced after each trial. *(sampling *with* replacement)*
- Only trial-and-test is possible independently of type of server *(single, inted or forking)*
Threats

Bypassing NX, SSP, RenewSSP and ASLR 2/3

**SSP-bfb** (SSP byte-for-byte):
- The canary value is the same in every trial. *(sampling without replacement)*
- The attacker can build a brute force attack but trying all possible values of each byte sequentially.

**RenewSSP-tat** (RenewSSP trial-and-test):
- The canary value is replaced after each trial. *(sampling with replacement)*
- Only trial-and-test is possible independently of type of server (single, inted or forking)

**ASLR-bff** (ASLR brute force full):
- The memory map is the same in all trials. *(sampling without replacement)*
- The attacker can build a brute force attack trying all possible addresses.
Threats

Bypassing NX, SSP, RenewSSP and ASLR 3/3

**ASLR-tat (ASLR trial-and-test):**

- The memory map is the same in all trials. *(sampling with replacement)*
- The attacker can **not** build a brute force attack trying all possible addresses.
Bypassing NX, SSP, RenewSSP and ASLR 3/3

**ASLR-tat (ASLR trial-and-test):**
- The memory map is the same in all trials. (sampling with replacement)
- The attacker can **not** build a brute force attack trying all possible addresses.

**ASLR-one (ASLR one shot):**
- Applications under certain circumstances the ASLR can be bypassed using a single attempt.
- For example building a ROP sequence from non-randomised applications (Not PIE compiled)
Summary of symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>entropy bits of the canary.</td>
</tr>
<tr>
<td>$n$</td>
<td>number of entropy bytes of the canary ($n = C/8$).</td>
</tr>
<tr>
<td>$c$</td>
<td>number of values that can take the canary ($c = 2^C$).</td>
</tr>
<tr>
<td>$R$</td>
<td>entropy bits of the ASLR for libraries.</td>
</tr>
<tr>
<td>$r$</td>
<td>number of places where the library can be located ($r = 2^R$).</td>
</tr>
<tr>
<td>$k$</td>
<td>number of trials (attempts) done by an attacker to a service.</td>
</tr>
</tbody>
</table>

Table: Summary of symbols.

Example on some 32 bit architectures:

- $n = 3$ canary bytes (one byte is zeroed)
- $C = 24 \rightarrow c = \frac{2^{24}}{8} = 16777216$ possible canary values.
- $R = 8 \rightarrow r = 2^8 = 256$ places to load the library.
The attacker only has a single trial to bypass both the SSP and the ASLR.

\[
Pr(\mathcal{X} = n) = \begin{cases} 
1 - \frac{1}{cr} & \text{if } n = 0, "\text{failure}" \\
\frac{1}{cr} & \text{if } n = 1, "\text{success}" 
\end{cases}
\]  

- A crash $\rightarrow$ service stopped. (the service is not restarted)
- This type of server has been introduced for completeness.
On the effectiveness of NX, SSP, RenewSSP and ASLR against stack buffer overflows

Analysis of the protection techniques

Inetd server

- The attacker can do as many trials as needed but the success is not guaranteed.
- Each trial has a probability of success of $\frac{1}{cr}$.
- Approx. 3 times more effort than in forking servers. (95% of success in 3 $cr$ trials).

<table>
<thead>
<tr>
<th>Geometric</th>
<th>PMF</th>
<th>$\frac{1}{cr} \left(1 - \frac{1}{cr}\right)^{k-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDF</td>
<td>$1 - \left(1 - \frac{1}{cr}\right)^k$</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>$\mu = cr$</td>
<td></td>
</tr>
<tr>
<td>Variance</td>
<td>$\sigma^2 = \frac{1-cr}{cr}$</td>
<td></td>
</tr>
<tr>
<td>Trials for</td>
<td>$100% = \infty$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$95% \approx 3 cr$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$50% \approx 0.693 cr$</td>
<td></td>
</tr>
</tbody>
</table>
Forking server

Forking server attacks

- SSP
- NX
- ASLR

The attacker can do as many trials as needed:

- Success **is** guaranteed.
- Some times is not practical.

Different attack strategies are possible.

Realistic attacks bypasses the three protection mechanisms.

The attacker can attack first the SSP and later the ASLR.
Forking server: SSP-bff + ASLR-one

- Full search SSP → Uniform distribution.
- One shot ASLR attack → zero cost.
- Full search SSP + One shot ASLR = Full search SSP.

---

**Uniform**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mean</strong></td>
<td>(\mu = c/2)</td>
</tr>
<tr>
<td><strong>Variance</strong></td>
<td>(\sigma^2 = (c - 1)/12)</td>
</tr>
<tr>
<td><strong>PMF</strong></td>
<td>(1/c)</td>
</tr>
<tr>
<td><strong>CDF</strong></td>
<td>(k/c)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trials for</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>(c)</td>
</tr>
<tr>
<td>95%</td>
<td>(0.95c)</td>
</tr>
<tr>
<td>50%</td>
<td>(c/2)</td>
</tr>
</tbody>
</table>
On the effectiveness of NX, SSP, RenewSSP and ASLR against stack buffer overflows

Analysis of the protection techniques

Forking server: SSP-bff + ASLR-bff

- Full search SSP → Uniform distribution.
- Full search ASLR → Uniform distribution.
- Since $c/r > 256$ then:
  - SSP-full + ASLR-full $\approx$ Uniform. ($k = c + r$)

<table>
<thead>
<tr>
<th>Uniform</th>
<th>Mean</th>
<th>$\mu = \frac{c}{2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Variance</td>
<td>$\sigma^2 = \frac{(c - 1)}{12}$</td>
</tr>
<tr>
<td></td>
<td>PMF</td>
<td>$\frac{1}{c}$</td>
</tr>
<tr>
<td></td>
<td>CDF</td>
<td>$k/c$</td>
</tr>
<tr>
<td>Trials for $100%$</td>
<td>$= c$</td>
<td></td>
</tr>
<tr>
<td>$95%$</td>
<td>$= 0.95c$</td>
<td></td>
</tr>
<tr>
<td>$50%$</td>
<td>$= c/2$</td>
<td></td>
</tr>
</tbody>
</table>
On the effectiveness of NX, SSP, RenewSSP and ASLR against stack buffer overflows

Analysis of the protection techniques

**Forking server: SSP-bfb + ASLR-one**

- Each SSP brute-forced byte → Uniform distribution.
- One shot ASLR attack → zero cost.
- The sum of distributions > 3 can be approx. to a Normal distribution.

---

**TABLE IV. Summary of the attack strategies**

<table>
<thead>
<tr>
<th>Attack Strategy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSP-bfb</td>
<td>SSP byte-for-byte, NX Re-use code, ASLR Full search, One-shot, Full search</td>
</tr>
<tr>
<td>SSP-tat</td>
<td>SSP brute-force-full, NX Re-use code, ASLR trial-and-test, One-shot, Full search</td>
</tr>
<tr>
<td>RenewSSP-tat</td>
<td>RenewSSP trial-and-test, NX Re-use code, ASLR trial-and-test, One-shot, Full search</td>
</tr>
<tr>
<td>SSP-bff</td>
<td>SSP byte-for-byte, NX Re-use code, ASLR brute-force-full, One-shot, Full search</td>
</tr>
<tr>
<td>SSP-tat</td>
<td>SSP brute-force-full, NX Re-use code, ASLR brute-force-full, One-shot, Full search</td>
</tr>
<tr>
<td>RenewSSP-tat</td>
<td>RenewSSP trial-and-test, NX Re-use code, ASLR brute-force-full, One-shot, Full search</td>
</tr>
<tr>
<td>SSP-bfb</td>
<td>SSP byte-for-byte, NX Re-use code, ASLR brute-force-full, One-shot, Full search</td>
</tr>
<tr>
<td>SSP-tat</td>
<td>SSP brute-force-full, NX Re-use code, ASLR brute-force-full, One-shot, Full search</td>
</tr>
<tr>
<td>RenewSSP-tat</td>
<td>RenewSSP trial-and-test, NX Re-use code, ASLR brute-force-full, One-shot, Full search</td>
</tr>
</tbody>
</table>

---

**TABLE V. Summary of the mean and variance**

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Parameters</th>
<th>Mean</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>c</td>
<td>μ = 256n</td>
<td>σ^2 = (256-1)n/12</td>
</tr>
<tr>
<td>Normal</td>
<td>n</td>
<td>μ = 256μ</td>
<td>σ^2 = 256σ^2</td>
</tr>
<tr>
<td>Triangular</td>
<td>k</td>
<td>0.5μ</td>
<td>0.5σ^2</td>
</tr>
</tbody>
</table>

**Figure:**

- **Sum of n uniforms**
  - Normal when n > 3
  - Mean: μ = 256n
  - Variance: σ^2 = (256-1)n/12
  - PMF: \( \frac{1}{\sqrt{2\pi\sigma^2}} e^\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \)
  - CDF: \( \frac{1}{2} \left(1 - erf\left(\frac{k-\mu}{\sqrt{2}\sigma}\right)\right) \)

**Trials for:**
- 100%: 2μ
- 95%: μ + 1.645σ
- 50%: μ
On the effectiveness of NX, SSP, RenewSSP and ASLR against stack buffer overflows

Analysis of the protection techniques

Hector Marco

Forking server: SSP-bfb + ASLR-bff

- Each SSP brute-forced byte → Uniform distribution.
- Full search ASLR → Uniform distribution.
- The sum of distributions > 3 can be approx. to a Normal distribution.
- Example, in Ubuntu 13.10 (x86):
  The canary has 3 bytes \((2^{3 \times 8})\), and the ASLR \(2^8\) which can be seen as a canary value of 4 bytes ≈ Normal distribution.

\[ \text{Mean } \mu = \frac{256n}{2} = \frac{256 \log_2(c)}{2} \]
\[ \text{Variance } \sigma^2 = \frac{(256-1)n}{12} \]
\[ \text{PMF } \approx \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]
\[ \text{CDF } \approx \frac{1}{2} \left( 1 - erf \left( \frac{k-\mu}{\sqrt{2\sigma^2}} \right) \right) \]

<table>
<thead>
<tr>
<th>Trials for 100%</th>
<th>2\mu</th>
</tr>
</thead>
<tbody>
<tr>
<td>95%</td>
<td>\mu + 1.645\sigma^2</td>
</tr>
<tr>
<td>50%</td>
<td>\mu</td>
</tr>
</tbody>
</table>
On the effectiveness of NX, SSP, RenewSSP and ASLR against stack buffer overflows

Hector Marco

Analysis of the protection techniques

Forking server: RenewSSP-tat + ASLR-one

- Each child has a different canary value → prevents brute force attacks.
- ASLR one shot → $r = 1$
- Success not guarantee.
- Each trial has a probability of success of $\frac{1}{c}$.

### Geometric

<table>
<thead>
<tr>
<th>PMF</th>
<th>$\frac{1}{cr} \left(1 - \frac{1}{cr}\right)^{k-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDF</td>
<td>$1 - \left(1 - \frac{1}{cr}\right)^k$</td>
</tr>
<tr>
<td>Mean</td>
<td>$\mu = cr$</td>
</tr>
<tr>
<td>Variance</td>
<td>$\sigma^2 = \frac{1-cr}{cr}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trials for 100%</th>
<th>$= \infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>95%</td>
<td>$\approx 3cr$</td>
</tr>
<tr>
<td>50%</td>
<td>$\approx 0.693 cr$</td>
</tr>
</tbody>
</table>
On the effectiveness of NX, SSP, RenewSSP and ASLR against stack buffer overflows

Analysis of the protection techniques

Forking server: RenewSSP-tat + ASLR-tat

- Each child has a different canary value → **prevents** brute force attacks.
- **Success not guarantee.**
- Each trial has a probability of success of $\frac{1}{cr}$.
- Similar to Inted protection but on forking servers.

### Table II

<table>
<thead>
<tr>
<th>Protection</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSP</td>
<td>byte for byte</td>
</tr>
<tr>
<td>NX</td>
<td>Re-use code</td>
</tr>
<tr>
<td>ASLR</td>
<td>One-shot</td>
</tr>
</tbody>
</table>

- Full search
- Re-use code
- One-shot
- Trial and test

### Geometric Distribution

- **PMF**
  \[ \frac{1}{cr} \left(1 - \frac{1}{cr}\right)^{k-1} \]
- **CDF**
  \[ 1 - \left(1 - \frac{1}{cr}\right)^k \]
- **Mean**
  \[ \mu = cr \]
- **Variance**
  \[ \sigma^2 = \frac{1 - cr}{cr} \]

### Trials for

<table>
<thead>
<tr>
<th>Probability</th>
<th>Trials for</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>$= \infty$</td>
</tr>
<tr>
<td>95%</td>
<td>$\approx 3\ cr$</td>
</tr>
<tr>
<td>50%</td>
<td>$\approx 0.693\ cr$</td>
</tr>
</tbody>
</table>
Results

Putting all together ....

<table>
<thead>
<tr>
<th>Attack/Bypass</th>
<th>100%</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSP-bff + ASLR-bff</td>
<td>4 Hours</td>
<td>2 Hours</td>
</tr>
<tr>
<td>SSP-bff + ASLR-one</td>
<td>4 Hours</td>
<td>2 Hours</td>
</tr>
<tr>
<td>SSP-bfb + ASLR-bff</td>
<td>1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>SSP-bfb + ASLR-one</td>
<td>&lt; 1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>RenewSSP-tat + ASLR-one</td>
<td>∞</td>
<td>3 Hours</td>
</tr>
<tr>
<td>RenewSSP-tat + ASLR-tat</td>
<td>∞</td>
<td>34 Days</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attack/Bypass</th>
<th>100%</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSP-bff + ASLR-bff</td>
<td>2.32 Myr</td>
<td>1.16 Myr</td>
</tr>
<tr>
<td>SSP-bff + ASLR-one</td>
<td>2.32 Myr</td>
<td>1.16 Myr</td>
</tr>
<tr>
<td>SSP-bfb + ASLR-bff</td>
<td>74 Hours</td>
<td>37 Hours</td>
</tr>
<tr>
<td>SSP-bfb + ASLR-one</td>
<td>1 sec</td>
<td>&lt; 1 sec</td>
</tr>
<tr>
<td>RenewSSP-tat + ASLR-one</td>
<td>∞</td>
<td>1605.79 Kyr</td>
</tr>
<tr>
<td>RenewSSP-tat + ASLR-tat</td>
<td>∞</td>
<td>431.05 Tyr</td>
</tr>
</tbody>
</table>

Table: Time cost for attacks in forking servers at 1000 trials/sec.
Conclusions

- NX/DEP obsoleted by new attacks: ret*, ROP, JOP etc.
- Forking servers reduce the effectiveness of the protection techniques:
  - Allow attack first the SSP and later the ASLR.
  - Allow build brute force attacks.
- SSP is reasonably effective, but fails on forking servers, specially against byte-for-byte attacks.
- The effectiveness of SSP is much better than that of the ASLR (but the ASLR covers more types of attacks).
- RenewSSP removes the dangerous byte-for-byte attack.
- SSP and ASLR are useless on Android.
- The ASLR in Windows is useless against local attacks.
Thank you for your attention!