2014 IEEE 13th International Symposium on Network Computing and Applications

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

Hector Marco-Gisbert, Ismael Ripoll
Departamento de Informática de Sistemas y Computadores. Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia, Spain
{hecmargi,iripoll}@upv.es

***Abstract*—Stack Smashing Protector (SSP), Address-Space Layout Randomization (ASLR) and Non-eXecutable (NX) are techniques for defending systems against buffer overflow attacks but not limited to them. These mechanism are available in modern operating systems like Android, GNU/Linux and Windows.**

Unfortunately, to keep up with the rapidly evolving landscape of cyber-security it is necessary to reassess the effectiveness of these protection techniques to avoid a false sense of security. This paper assess the effectiveness of these techniques against stack buffer overflow exploitation.

Our study indicates that the SSP technique is the most effective against stack buffer overflows. On forking servers, the ASLR technique is almost useless on 32-bit architectures due to the limited entropy provided by the size of the address space. The recently proposed technique RenewSSP, which is an improvement of the well known SSP, outperforms the original SSP in all the cases; it is highly effective against the dangerous byte-for-byte attack and on systems with low secret entropy as x86 and ARM.

I. Introduction

Over the years, a set of defensive techniques have been developed to protect malicious users. The security field is a very active always changing area, where the innovations and advances renders obsolete the technology. Therefore, it is mandatory to periodically reassess the effectiveness of those techniques. The requisites and constraints that were considered when a technique was initially developed, may be no longer valid when applied to current systems. Some protection techniques are outdated by changes on the execution framework or by newly developed counter attacks.

Stack Smashing Protection [1], Renew Stack Smashing Protector [2], Address-space Layout Randomization [3] and Non-eXecutable [4] are techniques that mitigate the execution of malicious code. They do not remove the underlying error which leads to a vulnerability but prevents or hinders the exploitation of the fault. The key idea of first three techniques (SSP, RenewSSP and ASLR) is to introduce a secret that must be known by the attacker in order to bypass it; and to restrict the execution capabilities of the processes in the case of the NX technique.

In this paper, the authors evaluate the effectiveness of each technique both when used individually and when combined, on different execution environments, considering different error manifestations and different exploitation techniques with respect to the stack buffer overflow vulnerability. According to SANS [5] it is the third most dangerous out of 20 vulnerability in current systems.

The main contributions of this paper are the following:

1) A statistical characterization of the remote attacks against the NX, SSP, RenewSSP and ASLR protection techniques and when used in combination.
2) A detailed analysis of the time needed to break-in and the probability of success is also presented.
3) We identify the scenarios (executing framework, operating system, etc.) which jeopardise the expected effectiveness of the classical techniques (SSP and ASLR), due to information leaks.
4) The results show that the RenewSSP is a promising modification of the SSP which makes the SSP robust against brute-force-attacks under all scenarios.

This paper is organised as follows. Section II presents the background and context needed to undertake the statistical analysis: *i*) the analysed vulnerability; *ii*) the execution environment where the programs are executed; *iii*) the protection techniques under study; *iv*) the threats to bypass the protection techniques; *v*) and finally the generic structure of an attack. Section III presents an statistical analysis of the attacks; special attention is given to the forking server since it is the most widely used. Section IV evaluates the practical effectiveness of the techniques on current systems. Finally, the concluding section summarizes the contributions of the paper and sketches the main findings.

II. Background and Terminology

The four techniques analysed are used with minor modifications in most modern operating systems. Each operating systems has its own particularities. In order to avoid excessive duplication, we have used only the UNIX style (`fork()`,`exec()`, etc.) to refer to the way processes are created. The conclusions are applicable to the other systems.

A. Stack buffer overflow vulnerability

The stack buffer overflow vulnerability (also known as stack smashing) occurs when a program writes to a memory address on the program's call stack outside of the intended data structure; usually a fixed length buffer. Stack buffer overflow bugs are caused when a program writes more data to a buffer located on the stack than there was actually allocated for that

978-1-4799-5393-6/14 $31.00 © 2014 IEEE
DOI 10.1109/NCA.2014.28

```
void vuln_function(char *srcbuff, int lsrcbuff) {
  char buff[48];
  ...
  memcpy(buff, srcbuff, lsrcbuff);
  ...
}
```

Listing 1. Example function which has a stack buffer overflow.

buffer, which most of the times results in corruption of adjacent data on the stack. When the overflow is done accidentally (i.e. it is not malicious), the program behalves improperly due to data corruption or executes illegal instructions which trigger a program crash.

But, if the attacker is able to control the way the overflow is produced (i.e. it is intentional), then it may take the control of the execution flow of the buggy program in such a way that they may execute arbitrary code. This is illustrated in the listing 1 which is an example with `memcpy()`. which shows a trivial example of a stack buffer overflow.

The canonical method for exploiting a stack based buffer overflow is to overwrite the return address stored in the stack with a pointer to an attacker's selected direction.

B. Types of server architectures

The execution environment and architecture of the server have an important impact on the effectiveness of each technique. Attending to the impact on the protection techniques we have identified three different server architectures.

Single process A single process server is a program that attends itself all client requests. Attending to the internal architecture of the server we can distinguish three different sub-types: *i*) sequential, *ii*) event based and *iii*) multi-thread. From the point of view of the security, all the three sub-types have the same behavior. We assume that the server crashes when an incorrect faked request is received, and then the service is stopped at once. There is little chances to break into the server, but it is easy to perform a DoS attack.

Inetd Every client request is attended by a different process launched from the server using the sequence `fork()`→`exec()`. A new process image is loaded in memory, and so, all the secrets used by the child process during each client request are renewed.
We decided to use the inetd name to honor the old network server daemon. This sequence is also called *self-reexecution*, which is used by the SSH suit.

Forking The operation of a forking server is very close to that of the inetd, but the children processes are in charge of directly attending the client requests, that is, no new executable image is loaded using the `exec()` call. Therefore, all the children have the same secrets as the father (except when the new RenewSSP technique is used). The behaviour of these kinds of servers can be used by the attackers to perform more effective attacks.
Android applications belong to this "category". All Android applications are child processes of the *Zygote* process. The difference with respect to a conventional forking server is that although each child executes the same Dalvik virtual machine, the application is different on each one.

C. Protection techniques

Following is an overview of the four techniques analysed in this paper.

NX or DEP Memory sections (pages) of the process which contain code are marked as executable and read-only. On the other hand, those areas containing data are marked as read/write and non-executable. Processors must provide hardware support to check for this policy when fetching instructions from main memory. Even if an attacker successfully injects code into a writeable (not executable) memory region, any attempt to execute this code would lead to a process crash. This technique is also known as "W^X" because a memory page can be marked as executable or writable, but not both at the same time.

SSP A random value, commonly known as *canary* or *guard*, is placed on the stack, just below the saved registers by the function prologue code. That value is checked at the end of the function, before returning, and the program aborts if the stored canary does not match its initial value. Any attempt to overwrite the saved return address on the stack will also overwrite the canary which leads to a process crash to prevent the intrusion.

ASLR Whenever a new process is loaded in main memory, the operating system loads the different areas of the process (code, data, heap, stack, etc.) at random positions in the virtual memory space of the process. Attacks relying on precisely knowing the absolute address of the injected code or a library function, like `ret2libc`, are very likely to crash the process (unless they know the memory map of the target process), thus preventing a successful intrusion

RenewSSP It is a modification of the stack-smashing protector (SSP) technique which renews the value of the reference canary of a process on any "non-returning" function [2]. It is specially effective when used on the child's code right after the new process is created with the `fork()` or `clone()` calls.

D. Threats to the protection techniques

Over the years, several strategies to bypass each protection technique have been developed, [6], [7], [8]. Due to space limitations, only the core of each attack strategy is presented. We do not considers attacks based on information leaks other than the one that can be obtained from the stack buffer overflow vulnerability; other forms of information leak require the existence of additional vulnerabilities and are out of the scope of this work.

Following is a brief description of the attacks considered in the paper.

NX/DEP The Non-eXecutable bit (NX)/Data Execution Prevention (DEP) mechanism can be bypassed using attacks that do not require to execute the injected code but reuse the already existing and mapped code on the target application. There is a family of techniques referred as *ret2** [9] and more generally the Return Oriented Programming (ROP) technique [10]. ROP is a very effective technique to bypass the NX. As a result, ASLR countermeasure was developed.

It is realistic to assume that modern attacks do not inject code, but use the ROP method. Therefore, from now on we will assume that the NX bit protection is bypassed directly, and then the security relies on the effectiveness of the remaining techniques.

It is important to point out that although the NX is defeated by the ROP, it must not be considered deprecated, and shall be maintained as far as the ASLR is not 100% secure and the NX does not introduce any execution overhead.

SSP-tat SSP trial-and-test. If the canary value is replaced or renewed after each trial, then the experiment is known as "sampling with replacement". The attacker can try at will, but it can not discard the already tested value.

SSP-bff SSP brute-force-full. In order to perform this attack the target service always has the same canary value and the service is restarted automatically after any server crash. The attacker can try as many times as needed the attack. On each trial, the attacker guesses a different value of canary until it matches. Since the canary value is always the same on the server, the attacker can discard incorrectly guessed values. Statistically, this is known as "sampling without replacement" experiment. Typically, the values are tested sequentially starting from zero up to the maximum value.

```
1  for (k=0; k<c; k++)
2    if( OK == end_request_up_to_canary(k) )
3      break;
4  printf("Canary value: %d\n", k);
```

Listing 2. Brute force to the canary.

SSP-bfb SSP byte-for-byte. If the manifestation of the error allows attackers to overflow up to the desired byte with any value, then it is possible to perform the byte-for-byte attack. It consists in trying all possible values of each byte sequentially. The code on listing 3 implements the code of the attack [11],[12]. All values from zero to 255 are tried sequentially until the correct value is found. The process continues with the next byte until all the bytes have been found.

This method allows to build very fast attacks. Unfortunately (for the attacker) this possibility is quite odd.

RenewSSP-tat RenewSSP trial-and-test. The brute force attack can not be employed against the RenewSSP, as pointed by the authors in [2]. In this scenario the only attack strategy is trial-and-test against the whole canary, independently of type of server (single, inetd or forking).

ASLR-bff ASLR brute force full. To bypass the ASLR, the attacker needs to know the absolute address where the ROP "program" starts [13]. If the memory map is the same in all the attacker trials and the attacker can perform as many trials as required, then it is possible to build a brute force attack [8], that is, an experiment "without replacement".

```
1  union {
2     unsigned char single_bytes[n];
3     unsigned int full_val;
4  } secret;
5  int idx, k=0;
6  for (idx=0; idx<n; idx++){
7     for (a=0; a < 256; a++){
8        if ( OK == send_request_up_to(idx, a) ){
9           secret.single_bytes[idx]=a;
10          k += (a + 1);
11          break;
12       }
13    }
14 }
15 printf("Secret value: %x\n", secret.full_val);
16 printf("Trials needed: %d\n", k);
```

Listing 3. Byte-for-byte attack.

ASLR-tat ASLR trial-and-test. When the memory map of the server is renewed after every trial (of after a fail trial), then the attacker can try different values of base address until it matches. But it can not discard already tested ones.

ASLR-one ASLR one shot. If there are one or more memory sections of the server that are not randomised then the attackers can use the statically (and so known) areas to build the ROP sequence. For example, when the code of the application (not the libraries) is not randomised, then it is possible to build a one-shot attack with a probability of 95.6% in x86, and 61.8% in x86_64, as shown by Roglia et al. [7]. This attack is effective on all server architectures.

Another strategy to bypass the ASLR is by directly observing the memory map of the target. Most operating systems (Windows, Android applications, and the other major player OS) libraries are randomised only per boot time and shared between all the applications. Any local user knows the ASLR secret. On those systems, the ASLR is completely useless from a local attack.

The GNU/Linux, which implements the position independent code (PIC) for libraries and when the executable is compiled with position independent executable (PIE), is not affected by the ASLR-one attacks on local attacks.

Another form of disclosure is through the information which some applications automatically report to the vendor provider (as debugging information) after a crash, which could contain valuable information to the attacker. We will consider that the ASLR is bypassed with a ASLR-one when there are enough gadgets to build the attack according to [7] or when it is a local attack on systems where the ASLR randomization only done at boot time.

E. Generic structure of an attack

In this work, we will consider that the attackers have access to the following information: *i*) access to the source code, *ii*) compiler and built options and *iii*) the execution environment of the target. The attacker can work off-line, using an in-house replicated target, testing and tuning the attack as long as necessary before the attack to the target is actually started.

The work that can be done off-line is considered to have no cost. That is, it takes zero time to achieve it. This is a realistic assumption (from the defendants point of view) because the attack starts only when the server is effectively attacked.

The attack consists on sending faked client requests, specially designed to overflow a buffer. The faked client request can be seen as a string long enough to overflow the buffer with the following elements (figure 1):

Fig. 1. Fields of a fake request.

Padding to canary Extra bytes inserted to increase the length of the request. The number of added bytes must be computed so that the next field exactly overwrites the stack canary. The length of this field can be accurately estimated off-line from the binary image of the server and a few trials against the target server. Since the cost of this part is relatively low, we will assume that the attacker know this value.

Canary This field will overwrite the canary. In order to succeed, it is necessary to know the actual value of the frame canary used in the target server. The canary is commonly a word (4 or 8 bytes). Let C be the entropy bits of the canary.

Padding to return Typically it is a few bytes (4 or 8 depending on the platform). We will assume that the attackers do not need to know these values (to build a ROP).

Return address Absolute entry point of the ROP code. The ROP code is located in a section with execution rights. We will suppose that the attacker must know the current memory layout of the server. Let R be the entropy bits of the ASLR.

ROP payload The injected ROP payload. This payload is basically a list of gadgets addresses. Gadgets are blocks of code located at relative positions with respect to the ROP entry point. Therefore, once the attacker know the entry point of the ROP, the rest of the ROP payload can be automatically adjusted with the appropriate offset. We will assume that the attackers are able to both build the ROP and adjust the resulting payload to the server memory layout once it is known the entry point (i.e. the jump address). Therefore, no extra information is required to build this field.

The grey fields of the request on Figure 1 can be filled by the attacker inspecting off-line the code of the server. But dark fields can only by obtained through direct "interaction" with the target. In most cases the attacker has a very limited and controlled interaction path.

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

TABLE I. SUMMARY OF SYMBOLS.

It can only submit a faked request and wait for the result. There are basically only two possible values for the result:

1) The server returns an answer, which is interpreted by the attacker as a correct guess. That is, the faked request did not crash the server.
2) The server does not respond, which is interpreted as an incorrect guess. The server should have crashed.

Depending on the architecture and execution environment of the server, they interpret the result differently (success or failure) and will tune or adjust the faked request.

III. ANALYSIS OF THE PROTECTION TECHNIQUES

This section analyzes the protections mechanisms for each server type. The probability of a successful attack is measured as the number of trials needed to break in the system.

A. Single process server

The attacker have only one single trial (assuming that the server service is not restarted by the administrator) to both the SSP and the ASLR. The probability to break into the system is given by the Bernoulli distribution:

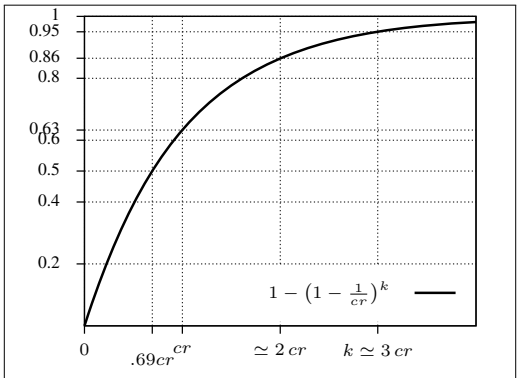
$$Pr(\mathcal{X} = n) = \begin{cases} 1 - \frac{1}{cr} & \text{if } n = 0, \text{"failure"} \\ \frac{1}{cr} & \text{if } n = 1, \text{"success"} \end{cases} \quad (1)$$

As far as the values of c and r are commonly large, there is little chances to break-in. There is little interest on trying to break-in unless other vulnerabilities or memory leaks are available (which are out of the scope of this paper). This type of server has been included for completeness.

B. Inetd based server

The attacker can do as many trials as needed. On each trial, they have the same probability of success: $\frac{1}{cr}$. There is no benefit on attacking first the canary and then the return address because there is no way to learn from previous failures. The attack is SSP-tat jointly with the ASLR-tat.

This strategy is modelled as a Bernoulli trial experiment: k trials are made with a probability of $\frac{1}{cr}$ of success in any trial. We are interested in counting the number of trials needed to get the first success, which follows a Geometric distribution defined for an infinite number of trials on the range $k \in [1, \infty[$.



Geometric	
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 100%	$= \infty$
95%	$\simeq 3\,cr$
50%	$\simeq 0.693\,cr$

TABLE II. INETD BASED SERVER SUMMARY.

The probability that the k^{th} trial is the first success is given by the PMF:

$$Pr(\mathcal{X} = k) = \frac{1}{cr}\left(1 - \frac{1}{cr}\right)^{k-1} \quad (2)$$

The cumulative distribution function (CDF) provides more valuable information; rather that the probability of succeeding at exactly the k^{th} trial, we are interested in the probability of succeed at any time up to the k^{th} trial. The CDF is defined as $Pr(\mathcal{X} \leq k)$ and is given by:

$$Pr(\mathcal{X} \leq k) = \sum_{i=1}^{k} \frac{1}{cr}\left(1 - \frac{1}{cr}\right)^{i-1} = 1 - \left(1 - \frac{1}{cr}\right)^{k} \quad (3)$$

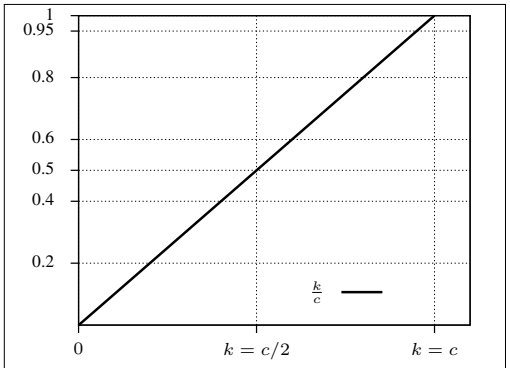
Since both secrets must be correctly guessed at once, the probability of success at each trial is one out of $c \times r$.

C. Forking server

The behaviour of these kinds of servers can be used by the attackers to perform more effective attacks. The rest of this section covers in detail how each protection technique can be bypassed, both individually and when used in combination.

1) SSP brute-force-full (SSP-bff): It is assumed that the behavior (success or fail) of the server can be detected, for example an incorrect guess closes the connection abruptly.

The probability that at the k^{th} trial the attacker try the correct value is given by the Uniform distribution with a PMF given by $Pr(\mathcal{X}_c = k) = \frac{1}{c}$. And the cumulative distribution function (CDF) is the sum of the PMF: $Pr(\mathcal{X}_c \leq k) = \sum_{i=1}^{k} \frac{1}{c} = \frac{k}{c}$. This distribution function is only "valid"[1] in the range $[0, k]$. Table III is a summary of the attack against the whole canary.



Uniform	
Mean	$\mu = c/2$
Variance	$\sigma^2 = (c-1)/12$
PMF	$1/c$
CDF	k/c
Trials for 100%	$= c$
95%	$= 0.95c$
50%	$= c/2$

TABLE III. SUMMARY OF THE SSP-BFF.

2) SSP byte-for-byte (SSP-bfb): Note that overflows caused by most string manipulation functions can not be used to implement this attack because a null byte is always copied at the end.

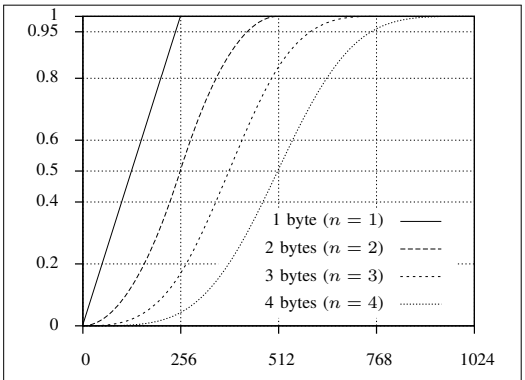
Each brute force attack against a single byte can be modelled as an Uniform distribution. The sum of several, n in our case, uniform distributions is known as the Irwin-Hall distribution, which quickly (for $n > 3$) approximates quite accurately for our purposes to a Normal distribution. The Figure on Table IV shows how the CDF changes with the length (number of bytes) of the canary. It is important to note that regardless the number of bytes, all CDF reach the value of one when $a = 256 \times$ `BYTES_PER_WORD`. That is, in the worst case, they have to do $256 \times$ `BYTES_PER_WORD` trials to break the canary.

A vulnerability of this type is very dangerous. The canary can be obtained in no more than one second regardless the word width of the architecture.

3) RenewSSP trial-and-test (RenewSSP-tat): This attack strategy is modelled as a Bernoulli trial experiment: k trials are made with a probability of $\frac{1}{c}$ of success in any trial; and the number of trials needed to break in the system is modeled as a Geometric distribution. The summary is in Table II, where the value of $r = 1$, since in this case we are considering only bypassing the canary.

4) ASLR brute-force-full (ASLR-bff): In forking servers the libraries mapping is inherited by all children. Therefore the ASLR-bff has the same behavior (distribution) than the attack

[1]To be mathematically correct it shall be said that its "*support is*".



Sum of n uniforms
$\simeq$ Normal when $n > 3$

Mean	$\mu = \frac{256n}{2} = \frac{256\log_2(c)}{2}$
Variance	$\sigma^2 = \frac{(256-1)n}{12}$
PMF	$\simeq \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-(x-\mu)^2/2\sigma^2\right)}$
CDF	$\simeq \frac{1}{2}\left(1 - erf\left(\frac{k-\mu}{\sqrt{2\sigma^2}}\right)\right)$
Trials for 100%	$= 2\mu$
95%	$= \mu + 1.645\sigma^2$
50%	$= \mu$

TABLE IV. SUMMARY OF THE SSP-BFB.

to whole canary, i.e. a Uniform distribution, sampling without replacement. The mean is $r/2$ and its range is $[0, r]$. Table III can be applied to the ASLR-bff attack only changing the variable c by r.

5) SSP brute-force-full + ASLR brute-force-full (SSP-bff + ASLR-bff): Since it is possible to split the attack in two phases, first attack the canary and then the ASLR. When the whole word of the canary has to be the attacked, the resulting distribution is the sum of two Uniforms. Where each Uniform have a different range of values: $[0, c]$ for the canary and $[0, r]$ for the ASLR. The sum of two different uniforms gives a trapezoidal distribution. If $c = r$ then it becomes a triangular.

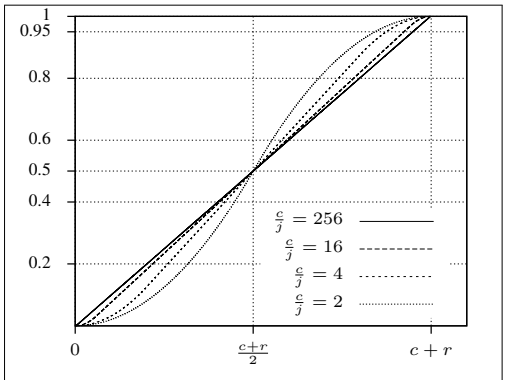
For simplicity, we will assume that $c > r$. The PMF es given by:

$$Pr(\mathcal{X} = k) = \begin{cases} \frac{k-2}{(c-1)(r-1)} & \text{for } k \in [2, j+1[\\ \frac{1}{c-1} & \text{for } k \in [r+1, c+1[\\ \frac{c+r-k}{(c-1)(r-1)} & \text{for } k \in [c+1, c+r[\end{cases} \quad (4)$$

When the value of r is much smaller than that of c, as it is the case on real systems, the expression 4 can be approximated to a Uniform distribution.

6) SSP byte-for-byte + ASLR brute-force-full (SSP-bfb + ASLR-bff): In this case it is also possible to split the attack to bypass first the SSP and then the ASLR. The statistical distribution of the attack to the canary plus the ASLR is given by the sum of the distributions of both random variables. In the case that the canary can be attacked with the SSP-bfb method, then it can be computed as the sum $n + 1$ Uniforms variables (n of the SSP plus the uniform of the ASLR-bff). Where n is the number of unknown bytes of the canary and $R/8$ the entropy bytes introduced by the ASLR. And the result is even closer to a normal distribution. The parameters of the resulting Normal are:

$$\text{Mean} \quad \mu = \frac{256(n+R/8)}{2}$$
$$\text{Variance} \quad \sigma^2 = \frac{(256-1)(n+R/8)}{12}$$



Trapezoidal
$\simeq$ Uniform when $c/r > 256$

Mean	$\mu = \frac{c+r}{2}$
Variance	$\sigma^2 = \frac{c+r-2}{12}$
PMF	$\simeq \frac{1}{c+r}$
CDF	$\simeq \frac{k}{c+r}$
Trials for 100%	$= c + r$
95%	$= 0.95\,(c + r)$
50%	$= 0.50\,(c + r)$

TABLE V. SUMMARY OF THE SSP + ASLR-FULL ATTACK.

7) RenewSSP trial-and-test + ASLR trial-and-test (RenewSSP-tat + ASLR-tat): The attack strategy to bypass this combination of techniques is similar to that used for the inetd server. There is not benefit on attacking first the canary and then the ASLR because there is no way to learn/discard from previous trials. Each trial ha the same probability of success: $\frac{1}{cr}$. This strategy is modelled as a Bernoulli trial experiment: k trials are made with a probability of $\frac{1}{cr}$ of success in any trial. Equation 2 shows the PMF and the cumulative distribution function (CDF) is showed in equation 3.

D. Servers summary

This section is a summary of the most relevant statistical parameters for single, inetd and forking servers. The forking server is the most interested because it is widely used in real systems.

In real systems, NX, SSP or RenewSSP and ASLR are used simultaneously. Table VI shows the distribution, mean, variance, and trials required to break the system with a probability of 100%, 95% and 50%. The same Table VI with the value of $r = 1$ represents the cost of the attacks when the ASLR can be bypassed with the ASLR-one attack.

	Distrib.	μ	Trails for a prob. of: 100%	95%	50%
Single: Single-Shoot	Bernoulli	$\frac{1}{cr}$	–	–	–
Inetd: SSP-tat+ASLR-tat	Geom.	cr	∞	3μ	0.693μ
Forking: SSP-bff+ASLR-bff	Uniform	$\frac{c+r}{2}$	2μ	0.95μ	μ
Forking: SSP-bfb+ASLR-bff	Normal	$\frac{2^8 n+r}{2}$	2μ	$\mu + 1.645\sigma^2$	μ
Forking: RenewSSP-tat+ ASLR-tat	Geom.	cr	∞	3μ	0.693μ
Forking: RenewSSP-tat+ ASLR-one	Geom.	c	∞	3μ	0.693μ

TABLE VI. SUMMARY OF THE MOST COMMON SYSTEMS AND ATTACKS.

IV. DISCUSSION

To evaluate the effectiveness of the NX, SSP, RenewSSP and ASLR, we have selected the most common server architectures and configurations. Current systems are all protected with the three protection techniques: NX, SSP and ASLR. We have also included the RenewSSP technique which although not widely used, we expect that it will replace the original SSP in the near future.

The cost is measured as the number of attempts (trials) needed by the attackers to break-in the system, summarised in Table VII.

	Attack/Bypass	100%	Mean
32bits syst.	SSP-bff + ASLR-bff	4 Hours	2 Hours
	SSP-bff + ASLR-one	4 Hours	2 Hours
	SSP-bfb + ASLR-bff	1 sec	$<$ 1 sec
	SSP-bfb + ASLR-one	$<$ 1 sec	$<$ 1 sec
	RenewSSP-tat + ASLR-one	∞	3 Hours
	RenewSSP-tat + ASLR-tat	∞	34 Days
64bits syst.	SSP-bff + ASLR-bff	2.32 Myr	1.16 Myr
	SSP-bff + ASLR-one	2.32 Myr	1.16 Myr
	SSP-bfb + ASLR-bff	74 Hours	37 Hours
	SSP-bfb + ASLR-one	1 sec	$<$ 1 sec
	RenewSSP-tat + ASLR-one	∞	1605.79 Kyr
	RenewSSP-tat+ASLR-tat	∞	431.05 Tyr

TABLE VII. TIME COST FOR ATTACKS IN FORKING SERVERS AT 1000 TRIALS/SEC.

The system is broken when the secrets are correctly guessed. The values been calculated using the following parameters:

- the system configuration (processor, network, firewalls, etc..) allows the attacker to perform 1000 trials per second,
- the entropy of the SSP and RenewSSP is 24 and 56 bits for 32 bits and 64 bits systems respectively,
- the entropy of the ASLR is 8 and 28 bits for 32 bits and 64 bits systems respectively.

On systems which are regularly monitored by humans or other advanced event correlation tools, the techniques are effective if the time to bypass them is longer than the reaction time. A protection of a few hours can give to the defenders enough time to apply specific corrective measures. On stand-alone non-supervised systems, the system shall resist in the order of years to be considered effective.

Table VIII is a more complete list of systems and attacks, including combinations that are no longer released but may be still operative.

		32 bits		64 bits	
	Technique	**100%**	**Mean**	**100%**	**Mean**
Inetd based	SSP-tat	∞	1.2×10^7	∞	5.0×10^{16}
	ASLR-one	1×10^0	0.5×10^0	1×10^0	0.5×10^0
	ASLR-bff	∞	1.8×10^2	∞	1.9×10^8
	SSP-tat+ASLR-one	∞	1.2×10^7	∞	5.0×10^{16}
	SSP-tat+ASLR-tat	∞	3.0×10^9	∞	1.3×10^{25}
Forking based	SSP-bff	1.7×10^7	8.4×10^6	7.2×10^{16}	3.6×10^{16}
	SSP-bfb	7.7×10^2	3.8×10^2	1.8×10^3	9.0×10^2
	RenewSSP-tat	∞	1.2×10^7	∞	5.0×10^{16}
	ASLR-one	1×10^0	1×10^0	1×10^0	1×10^0
	ASLR-bff	2.6×10^2	1.3×10^2	2.7×10^8	1.3×10^8
	SSP-full+ASLR-one	1.7×10^7	8.4×10^6	7.2×10^{16}	3.6×10^{16}
	SSP-bff+ASLR-bff	1.7×10^7	8.4×10^6	7.2×10^{16}	3.6×10^{16}
	SSP-bfb+ASLR-one	7.7×10^2	3.8×10^2	1.8×10^3	9.0×10^2
	SSP-bfb+ASLR-bff	1.0×10^3	5.1×10^2	2.7×10^8	1.3×10^8
	RenewSSP-tat+ASLR-one	∞	1.2×10^7	∞	5.0×10^{16}
	RenewSSP-tat+ASLR-tat	∞	3.0×10^9	∞	1.3×10^{25}

TABLE VIII. ATTEMPTS TO BYPASS PROTECTION MECHANISMS.

The following list summarises the most important results of this evaluation.

- The NX was rendered mainly obsoleted first by the family of ret2* attacks and then by the ROP. Although it slightly increases the difficulty of building an exploit, since it is an un-expensive technique (the check is performed by hardware: the MMU), it still worthwhile using it. Basically, there is no benefit in removing it from a system where it is already implemented.
- In the inetd architecture, the combination of the three techniques (NX, SSP and ASLR) is a very effective, it has a multiplicative effect.
  This robust architecture is used by the SSH suite. Each connection request is handled with the following sequence of system calls: `fork()`→ `exec(sshd)`→ `do_work`→ `exit()`. This way, it receives all randomization the operating system can provide.
  In 64bit systems it is impossible to bypass, the mean time is at least 1.605.000 years with probability 95% the best case for the attacker. In 32bits, although it less effective, it is still provides an acceptable protection.
  This architecture is not affected by brute force attacks. The dangerous byte-for-byte attack can not be employed by the attackers even when the vulnerability allows to overwrite at byte granularity.
- Unfortunately, the forking server architecture greatly reduces the effectiveness of the protections, because it allows new exploitation strategies:
  - Split the attack of the SSP and the ASLR, which has an additive effect rather than a multiplicative one.
  - It is possible to implement brute force attacks to both the SSP and the ASLR secrets.

- If the manifestation of the vulnerability allows overwrite at byte level then it is possible to make byte-for-byte attacks.

  In forking servers, the protection techniques are only effective in 64 bits systems when the byte-for-byte can not be employed. The byte-for-byte attack renders useless the SSP and the ASLR is not strong enough on its own. The RenewSSP technique restores the effectiveness of both the SSP and the ASLR in forking servers. The attacker can no longer discard tested values (i.e. brute force attacks can not be make to the SSP). Also, it is not possible to slit the SSP-ASLR attack.
  Therefore the RenewSSP technique provides the same level of protection than that of the inetd.
- Both SSP and the ASLR techniques are basically useless against local attacks for Android Applications. All Android applications share the same canary and memory maps.
  This problem only affects the Android applications, that is, those the use the Dalvik virtual machine. On the other hand, Android native processes enjoy one of the best protections[2].
- The ASLR on Windows and the bitten fruit OSs is implemented on a per-boot basis, which greatly reduces the effectiveness against local attacks. The attacker know the layout of all the libraries.

V. Conclusions

In this paper the authors reassess the effectiveness of three mature techniques: NX, SSP and ASLR, as well as the new RenewSSP (properly speaking, it can not be considered a new technique but an improvement to the standard SSP). The study has been focused to the stack buffer overflow vulnerabilities, which is still one of the most serious security issues.

Besides the direct exploitation of the buffer overflow, this paper consider the presence of multiple attack vectors, as for example the possibility to obtain information from the target service using applications that collect public data from within the same system. Another attack vector considered is the weak implementation of the ASLR on most systems (Windows, Android, and the other) resulting in that all the applications share the same library maps, which renders useless the ASLR from local attacks.

Although the NX/DEP was a revolutionary technique when initially developed, currently it been rendered obsoleted by new attacking methods: ret* and ROP. Our evaluation indicates that the ASLR on 64 bits systems is an effective counter-measure against those new attacking methods but it is not for 32 bits systems. The SSP effectiveness is reasonably good for both 64 bits systems but rather weak for 32 bits architectures even when it is combined with the ASLR.

Both techniques, ASLR and SSP relay on keeping secret some internal keys (the guard and the memory map for SSP and ASLR respectively). There are a lot of information shared/inherited from the father to the children processes, and the fact that the normal policy applied when a server process crashes is to restart the process automatically, allows to make brute force attacks. The more often the protection secrets are renewed, the harder will be for the attackers to bypass them.

Some systems (Android applications, Windows OSs and the bitten fruit company OS) renew the secrets of the ASLR once per boot, while GNU/Linux renew them on a per process (`exec()`) basis. The once per boot is not robust against local attacks.

Our results show that the forking server architecture greatly reduces the effectiveness of the protection techniques, specially when the target system is vulnerable to the dangerous byte-for-byte attack. When this type of vulnerability is present in the application, the only solution is the recently proposed extension to SSP called RenewSSP.

[2]As far as the current published state of the art is concerned.

References

[1] C. Cowan, C. Pu, D. Maier, H. Hintongif, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang, "Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks," in *Proc. of the 7th USENIX Security Symposium*, Jan 1998, pp. 63–78.

[2] H. Marco-Gisbert and I. Ripoll, "Preventing brute force attacks against stack canary protection on networking servers," in *Network Computing and Applications (NCA), 2013 12th IEEE International Symposium on*, 2013, pp. 243–250.

[3] Pax Team. (2003) PaX address space layout randomization (ASLR). [Online]. Available: http://pax.grsecurity.net/docs/aslr.txt

[4] L. D. Paulson, "New chips stop buffer overflow attacks," *Computer*, vol. 37, no. 10, pp. 28–30, 2004.

[5] Mitre. (2011) CWE/SANS top 25 most dangerous software errors. [Online]. Available: http://cwe.mitre.org/top25

[6] A. 'pi3' Zabrocki, "Scraps of notes on remote stack overflow exploitation," November 2010. [Online]. Available: http://www.phrack.org/issues.html?issue=67&id=13#article

[7] G. F. Roglia, L. Martignoni, R. Paleari, and D. Bruschi, "Surgically returning to randomized lib(c)," in *Proceedings of the 2009 Annual Computer Security Applications Conference*, ser. ACSAC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 60–69. [Online]. Available: http://dx.doi.org/10.1109/ACSAC.2009.16

[8] J. Han, D. Gao, and R. H. Deng, "On the effectiveness of software diversity: A systematic study on real-world vulnerabilities," in *Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. DIMVA '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 127–146. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02918-9_8

[9] Nergal, "The advanced return-into-lib(c) exploits: PaX case study," *Phrack*, no. 58, 2001.

[10] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-oriented programming: Systems, languages, and applications," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, pp. 2:1–2:34, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2133375.2133377

[11] Bulba and Kil3r, "Bypassing stackguard and stackshield," *Phrack*, 56, 2002.

[12] NIST. (2011, September) Vulnerability Summary for CVE-2010-3867. [Online]. Available: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2010-3867

[13] E. Bhatkar, D. C. Duvarney, and R. Sekar, "Address obfuscation: an efficient approach to combat a broad range of memory error exploits," in *In Proceedings of the 12th USENIX Security Symposium*, 2003, pp. 105–120.