Lecture 13 – Beyond Stack Smashing & Malware Part 1

Ryan Cunningham
University of Illinois
ECE 422/CS 461 – Fall 2017

Security News

- Deloitte hacked
  - Attackers in systems for nearly a year
  - Took confidential information and emails
  - Krebs (again) reporting gigabytes exfiltrated
- ISPs infecting users with trojan FinFisher spyware
- Adobe…

Adobe Product Security Incident Response Team (PSIRT) Blog

Working to help protect customers from vulnerabilities in Adobe software. Contact us at PSIRT(at)adobe(dot)com.

PSIRT PGP Key (0x33E9E596)

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: Mailvelope v1.8.0
Comment: https://www.mailvelope.com

xsFNBFm/2KMBEADbwToJM3BCVE1OeC22HgVEqNEDppXzuD2dgfKuy0M4tx2L
De7GkPjo6AOsw4yi8bakLiidpw5B0J/AR1VtIjIDEmS0F9MRZIcV0UKyA5qV
c9BafZnAicY7nezkIJUmyLcIVMC60pqSHzo0Ewy2PZjxzcI4vDGhHmcgfV5X
R+duYld3LtVI+A/5jv326LB16bCNts/tOhW2T0LraMPoCtdH84Z4tPcyp335
s8/dZ2C+EoMD4iX1kIymZ1kqEfZNvcs1sRUXy27sL01VHcYmi6UNWCeeHOu2
2yJxMiBCniozBKZUwcR6ysg97nnq633dN9mf7V30PS3zAjhE0Hvmzg3B/Nfo
qzy2dAEU/JDUBhiAo+xr9VF3ZPOoC8JySORgyUm/2t3TTBaH+DnfsUBiqo5U
2T0n8x2R1FWxyZYNCTku5JOvPqRBft13DSyJD7LDDps62nqhpaVb34eprwuk
qIk0TMRu9mB4EQc+cNFR3ZpN1AKj+HOb/TUJwCJpVju2/3g0wgdqHh+OQlvC
Nm8vIGnQZWQ30WqnH/UFoh3RPJ+WqnDq88NmqBq8I4aNV4u8MqoObd/zrtVX
kAwYHbIZLo925NjFyPuuxhWiCotKenl8dZefB8aB8lRjYuIMnCJ0GQus+JG8
TJyEesNdK/q8HD5h1kCRSzMHDl+Ra3z/1+FFIwARAQABzR1BZG9iZSBQU0lS
VCA8cHNpcnRAYWRvYmUuY29tPsLBewQQAQgALwUCWb/YrwUJAeEzgAYLCQgH
AwIJEIbAD8Kvh3YWBBUIAgoDFgIBAhkBAhsDAh4BAADk2A//f+6PFzg4VmLI
PzsTZPoqPR/lXlZ7RIYbQosHvsFwyW0WWX1uI1sEeD5Qo7HQt6NNMAOW51Js
wFvFOWIa9U6SHRoU1kGTSESReOq5HnXe4DcBubsKmoMS68PuiZ88wYOIM4Up
9V9PUuaue0U4oSrYHnH5qBOqurtv8wO5Cq4uTwnfnjN7n4OH0++29l0PJ68B
6+kMuQyG4swmxsZhljlqGMHcs0c/BuI3W+n5w+xLM7N5jjCTjNXR+tGmstdm
RPEoLWOso+ZFwfNW0CLKjYUahp3p6H9x8R13wrp2re0GhqKRgt3D4UcAqsPs
Pg16htQ3Gh5ZGHcorQFz1rcFsdgvoFw/RsNFDeiMP37Y9b6XC8KcAxOm9TfR
VEOK91UY/zivfNKlMewxM5dPqMdsI73umWwYRrZEPotug+Zmt1DWvWKxYiBn
```

CATEGORIES

- Alert
- Security Bulletin
- Uncategorized

ARCHIVES

- September 2017
- August 2017
- July 2017
- June 2017
- May 2017
- April 2017
- March 2017
- February 2017
- January 2017
- December 2016
- November 2016
- October 2016
- September 2016
- August 2016
- July 2016
- June 2016
- May 2016
- April 2016
- March 2016
- February 2016
- January 2016
- December 2015
- November 2015
- October 2015

```
LJyYLUvFjL3i3jbiNT1NKldwqaL2i9OuRAuHthoFGOKIqr6hmtOYzUem/cl+
ZlRwd77Vmfc=
=QOc7
-----END PGP PUBLIC KEY BLOCK-----

-----BEGIN PGP PRIVATE KEY BLOCK-----
Version: Mailvelope v1.8.0
Comment: https://www.mailvelope.com

xcaGBFm/2KMBEADbwToJM3BCVE1OeC22HgVEqNEDppXzuD2dgfKuy0M4tx2L
De7GkPjo6AOsw4yi8bakLiidpw5B0J/AR1VtIjIDEmS0F9MRZIcV0UKyA5qV
c9BafZnAicY7nezkIJUmyLcIVMC60pqSHzo0Ewy2PZjxzcI4vDGhHmcgfV5X
R+duYld3LtVI+A/5jv326LB16bCNts/tOhW2T0LraMPoCtdH84Z4tPcyp335
s8/dZ2C+EoMD4iX1kIymZ1kqEfZNvcs1sRUXy27sL01VHcYmi6UNWCeeHOu2
2yJxMiBCniozBKZUwcR6ysg97nnq633dN9mf7V30PS3zAjhE0Hvmzg3B/Nfo
qzy2dAEU/JDUBhiAo+xr9VF3ZPOoC8JySORgyUm/2t3TTBaH+DnfsUBiqo5U
2T0n8x2R1FWxyZYNCTku5JOvPqRBft13DSyJD7LDDps62nqhpaVb34eprwuk
qIk0TMRu9mB4EQc+cNFR3ZpN1AKj+HOb/TUJwCJpVju2/3g0wgdqHh+OQlvC
Nm8vIGnQZWQ30WqnH/UFoh3RPJ+WqnDq88NmqBq8I4aNV4u8MqoObd/zrtVX
kAwYHbIZLo925NjFyPuuxhWiCotKenl8dZefB8aB8lRjYuIMnCJ0GQus+JG8
TJyEesNdK/q8HD5h1kCRSzMHDl+Ra3z/1+FFIwARAQAB/gkDCA7HXpjNu7yW
YBVIglTandp2qwxLZTA0Jm3YMOwvBojE4ZDL41VZBh2sBphQ15CLulx7MUrD
/0EqjYjWLX6Ti2Ez5wllG32t5BctTT/U0e2/29A9gnFxj/h+eyT+Ge0IYn4U
OwlHdoELljhtYcXhPUFIy1i6RAWjOP+qohlbiB7+Bw2C/EoNWxg6LJ6ARQHL
6oV0Qt+Zl2Z2v5jroAwwDz9A2JfKXiPspU0PAajZaqUJ8tGtxuBSRzCGHhtr
9dxCKzPyrGs37c7W1AWvwnzBWbCCTfGCyMByWcZVuXFCjOJ91XxeqrvLl1AR
n2hhDhbcpSl8fqPSGcWjiScvD1HLSqAGwvRL4P0dQFXwWjWA0wzoLq7Ps7M3
ak91vwKRn6HlLoxDTlbx0GX3f52XYcAMsmBW7+46gANybjflqoJoXxJsViaT
taDGP0Cb3gJKHxCGYcEJ57icnCNF18hkWcVHi/YKC3ph84bcRNmKM44i+6Ah
mswlNfWQmyQCr+oRzi/TW+wfeDVLRwLimIeYh2vJptZldwanFw4HOrFcw76g
vBqnHNL2gvOErhDbRHqcc8eUAW5xr1w3fMKBLxvhmgTyva/IEAZygomEiwep
4Yex3CYMdvmNP5CxB/xIEnRd77jJaJacNfP30wAVMBYdw15RlTMa+1X+49DT
YLLyBZ1DAK8crs5iyLMV67lTaaIz6Rk+zhLIc9EJCat1JG9VN3hmA0grzse9
WeTzdte/Qqh0ukved1b/p0v0kOI/yBcmzQA2Yr3adpbZmz1RxzDBjbAWBTIb
```

Taxonomy of Vulnerabilities

- Buffer Overflow
- Command Injection
- Cross-Site Scripting
- Format String
- Illegal/Dangling Pointer
- Integer Overflow
- Path Manipulation
- Resource Injection
- String Termination Error
- Unsafe Reflection
- Insecure Temp File
- Double free
- Use-after-Free
- Memory Leak
- Debug Code Enabled
- Deadlock
- Race Conditions
- String Formatting Error

Example 1

```
int len, error;
...
error = copyin((caddr_t)uap->alen,
               (caddr_t)&len, sizeof (len));
if (error) {
    fdrop(fp, p);
    return (error);
}
...
len = MIN(len, sa->sa_len);
error = copyout(sa, (caddr_t)uap->asa, (u_int)len);
```

Example 2

```
int copy(int *in, int len){
    int *buff, buff_size, i;
    buff_size = len*sizeof(int);
    buff = malloc(buff_size);
    if(buff == NULL){ return -1;}
    for(i = 0; i < len; i++){
        buff[i] = in[i];
    }
    return buff;
}
```

Example 2

```
buff_size=len*sizeof(int);

mov   edx, dword[ss:ebp+X]
shl   edx, 0x2

len = 0x40000001 //1,073,741,825
buff_size = len<<2 = 0x4   //4
```

Example 2

len=1,073,741,825

```
int copy(int *in, int len){
    int *buff, buff_size, i;
    buff_size = len*sizeof(int);
    buff = malloc(buff_size);
    if(buff == NULL){ return -1;}
    for(i = 0; i < len; i++){
        buff[i] = in[i];
    }
    return buff;
}
```

buff_size=4

OVERFLOW!

Integer Vulnerabilities

- Overflow – integer operations produce a value that is out of range
- Truncation – a value is stored in a type that is too small to represent the result
- Sign error – the sign bit is misinterpreted

Example 1

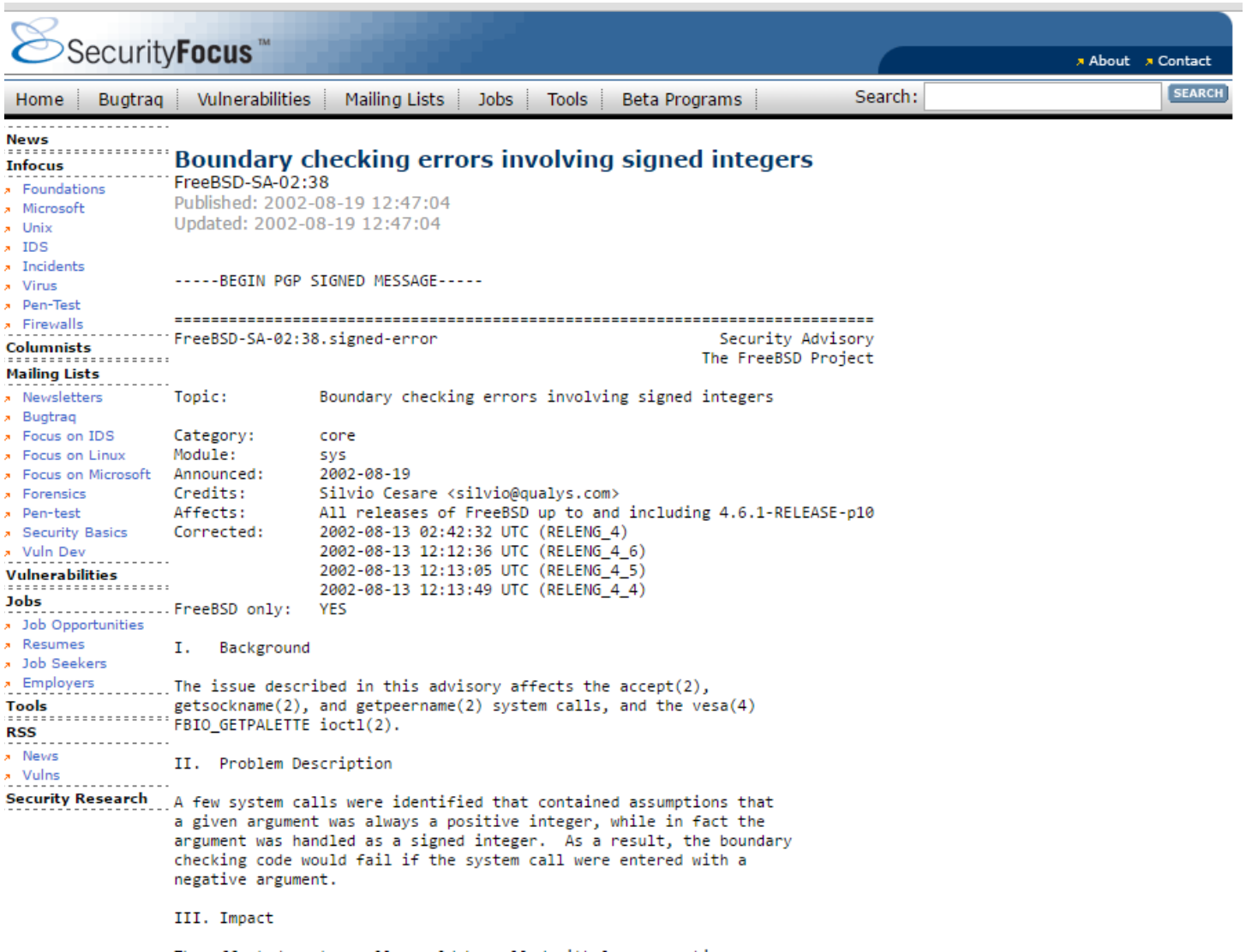
```
int len, error;
...
error = copyin((caddr_t)uap->alen,
               (caddr_t)&len, sizeof (len));
if (error) {
    fdrop(fp, p);
    return (error);
}
...
len = MIN(len, sa->sa_len);
error = copyout(sa, (caddr_t)uap->asa, (u_int)len);
```

uap->alen=-1

len=-1=0xFFFFFFFF

(u_int)0xFFFFFFFF=4,294,967,295

In the Wild



SecurityFocus™

About Contact

Home | Bugtraq | Vulnerabilities | Mailing Lists | Jobs | Tools | Beta Programs | Search: SEARCH

News

Infocus

- Foundations
- Microsoft
- Unix
- IDS
- Incidents
- Virus
- Pen-Test
- Firewalls

Columnists

Mailing Lists

- Newsletters
- Bugtraq
- Focus on IDS
- Focus on Linux
- Focus on Microsoft
- Forensics
- Pen-test
- Security Basics
- Vuln Dev

Vulnerabilities

Jobs

- Job Opportunities
- Resumes
- Job Seekers
- Employers

Tools

RSS

- News
- Vulns

Security Research

Boundary checking errors involving signed integers

FreeBSD-SA-02:38
Published: 2002-08-19 12:47:04
Updated: 2002-08-19 12:47:04

```
-----BEGIN PGP SIGNED MESSAGE-----

=============================================================================
FreeBSD-SA-02:38.signed-error                                Security Advisory
                                                          The FreeBSD Project

Topic:          Boundary checking errors involving signed integers

Category:       core
Module:         sys
Announced:      2002-08-19
Credits:        Silvio Cesare <silvio@qualys.com>
Affects:        All releases of FreeBSD up to and including 4.6.1-RELEASE-p10
Corrected:      2002-08-13 02:42:32 UTC (RELENG_4)
                2002-08-13 12:12:36 UTC (RELENG_4_6)
                2002-08-13 12:13:05 UTC (RELENG_4_5)
                2002-08-13 12:13:49 UTC (RELENG_4_4)
FreeBSD only:   YES

I.   Background

The issue described in this advisory affects the accept(2),
getsockname(2), and getpeername(2) system calls, and the vesa(4)
FBIO_GETPALETTE ioctl(2).

II.  Problem Description

A few system calls were identified that contained assumptions that
a given argument was always a positive integer, while in fact the
argument was handled as a signed integer.  As a result, the boundary
checking code would fail if the system call were entered with a
negative argument.

III. Impact
```

Use-After-Free (Dangling Pointer)

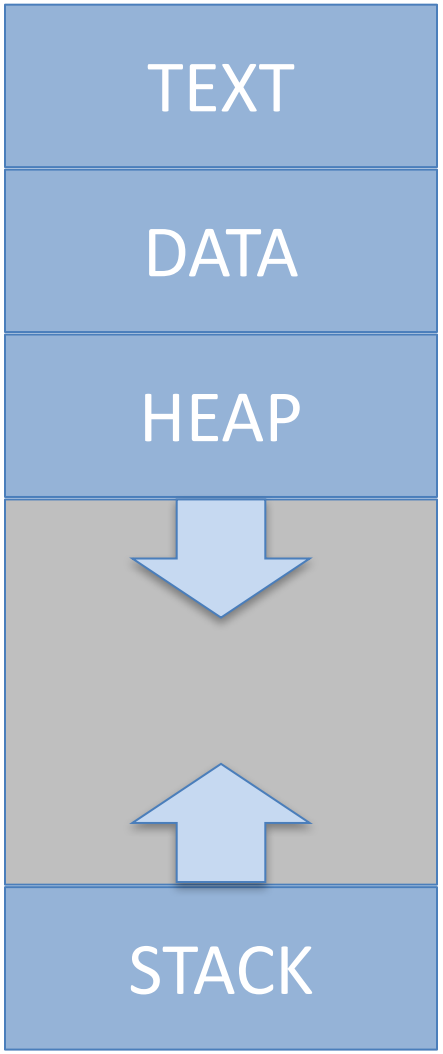
```
int *i=malloc(sizeof(int));
...
free(i);
...
*i=3;
```

Use-After-Free (Dangling Pointer)

```
Foo *f=new Foo();
...
delete f;
...
f->bar();
```

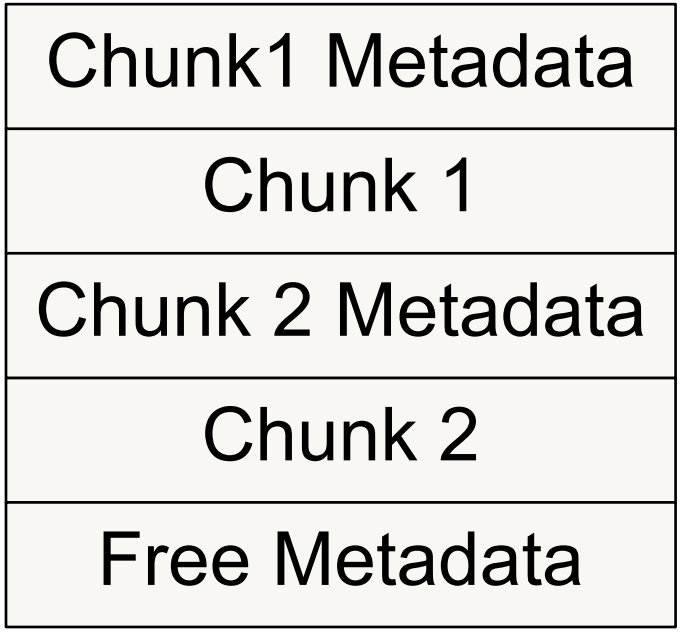
Attacking the Heap

- Allocated at run time
- Dynamic structures, objects
- Allocated in chunks by malloc
- Chunks deallocated by free
- Details are implementation specific



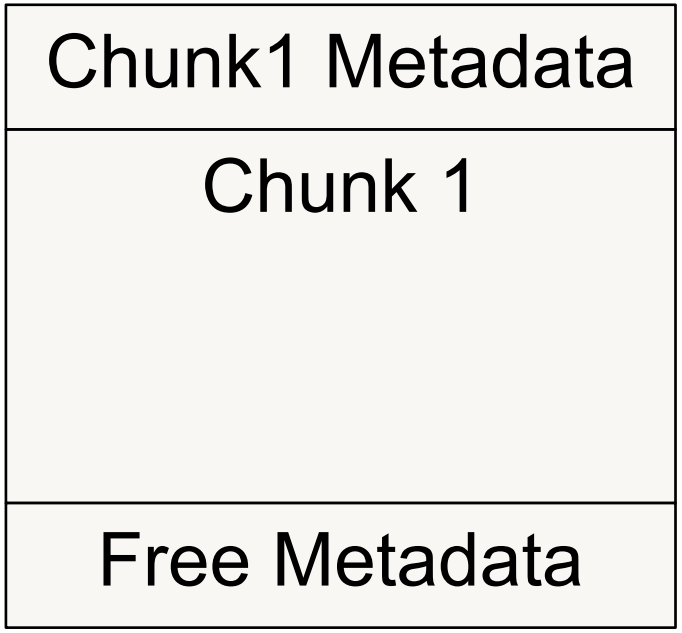
Heap Overflow

- Heap memory managed in chunks
- Chunks can grow or shrink as needed
- Chunks stored in a doubly linked list with metadata
- Chunks are allocated next to each other in memory
- Chunks can be allocated or unallocated (marked in metadata)



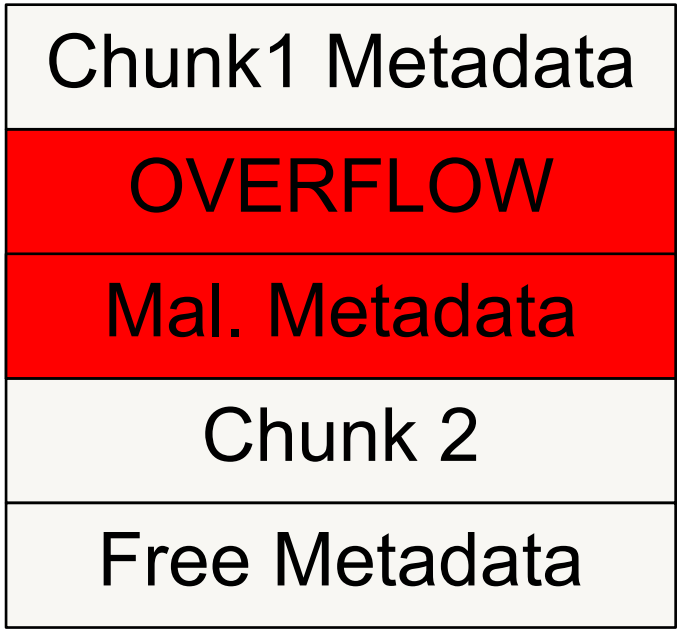
Heap Overflow

- When deallocated, a chunk is merged with its previous neighbor (if neighbor is unallocated)
- This causes a doubly linked list node to be deleted
- Deletion code *relies on metadata*!



Heap Overflow

- Overflow into metadata allows attacker to control contents
- Attacker can control *what* is written and *where* it is written during node deletion



Other Heap Techniques

- Overwrite virtual method table entries
  - function pointers on heap
  - use after free
- Heap spray
  - build giant noop sleds ending in shellcode
  - spray through the entire stack
  - jumping into heap => exploit

How does this code look?

```
int main(int argc,char *argv[]){
  char buf[128];
  strcpy(buf,argv[1]);
  printf(buf);
  printf("\n");
}
```

Let’s go to the docs…

Character	Description
`%`	Prints a literal `%` character (this type doesn't accept any flags, width, precision, length fields).
`d` , `i`	`int` as a signed decimal number. `%d` and `%i` are synonymous for output, but are different when used with `scanf()` for input (where using `%i` will interpret a number as hexadecimal if it's preceded by `0x` , and octal if it's preceded by `0` .)
`u`	Print decimal `unsigned int` .
`f` , `F`	`double` in normal (fixed-point) notation. `f` and `F` only differs in how the strings for an infinite number or NaN are printed (`inf` , `infinity` and `nan` for `f` , `INF` , `INFINITY` and `NAN` for `F`).
`e` , `E`	`double` value in standard form ([−]d.ddd `e` [+ / −]ddd). An `E` conversion uses the letter `E` (rather than `e`) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is `00` . In Windows, the exponent contains three digits by default, e.g. `1.5e002` , but this can be altered by Microsoft-specific `_set_output_format` function.
`g` , `G`	`double` in either normal or exponential notation, whichever is more appropriate for its magnitude. `g` uses lower-case letters, `G` uses upper-case letters. This type differs slightly from fixed-point notation in that insignificant zeroes to the right of the decimal point are not included. Also, the decimal point is not included on whole numbers.
`x` , `X`	`unsigned int` as a hexadecimal number. `x` uses lower-case letters and `X` uses upper-case.
`o`	`unsigned int` in octal.
`s`	null-terminated string.
`c`	`char` (character).
`p`	`void *` (pointer to void) in an implementation-defined format.
`a` , `A`	`double` in hexadecimal notation, starting with `0x` or `0X` . `a` uses lower-case letters, `A` uses upper-case letters.[3][4] (C++11 iostreams have a `hexfloat` that works the same).
`n`	Print nothing, but writes the number of characters successfully written so far into an integer pointer parameter. Note: This can be utilized in Uncontrolled format string exploits.

Zoom and enhance…

Character	Description
`%`	Prints a literal `%` character (this type doesn't accept any flags, width, precision, length fields).
`d` , `i`	`int` as a signed decimal number. `%d` and `%i` are synonymous for output, but are different when used with `scanf()` for input (where using `%i` will interpret a number as hexadecimal if it's preceded by `0x` , and octal if it's preceded by `0` .)
`u`	Print decimal `unsigned int` .
`f` , `F`	`double` in normal (fixed-point) notation. `f` and `F` only differs in how the strings for an infinite number or NaN are printed (`inf` , `infinity` and `nan` for `f` , `INF` , `INFINITY` and `NAN` for `F`).

Print nothing, but writes the number of characters successfully written so far into an integer pointer parameter.
Note: This can be utilized in Uncontrolled format string exploits.

`g` , `G`	`double` in either normal or exponential notation, whichever is more appropriate for its magnitude. `g` uses lower-case letters, `G` uses upper-case letters. This type differs slightly from fixed-point notation in that insignificant zeroes to the right of the decimal point are not included. Also, the decimal point is not included on whole numbers.
`x` , `X`	`unsigned int` as a hexadecimal number. `x` uses lower-case letters and `X` uses upper-case.
`o`	`unsigned int` in octal.
`s`	null-terminated string.
`c`	`char` (character).
`p`	`void *` (pointer to void) in an implementation-defined format.
`a` , `A`	`double` in hexadecimal notation, starting with `0x` or `0X` . `a` uses lower-case letters, `A` uses upper-case letters.[3][4] (C++11 iostreams have a `hexfloat` that works the same).
`n`	Print nothing, but writes the number of characters successfully written so far into an integer pointer parameter. Note: This can be utilized in Uncontrolled format string exploits.

Format string exploits

- %n string format character is dangerous
  - "The number of characters written so far is stored into the integer indicated by the int *"
  - printf("1001%n",&i); stores 4 into i
- With %n, we can control *what* is written and *where* it is written
- Can be used to jump over stack canary to initialize a ROP attack

VULNERABILITIES IN THE WILD

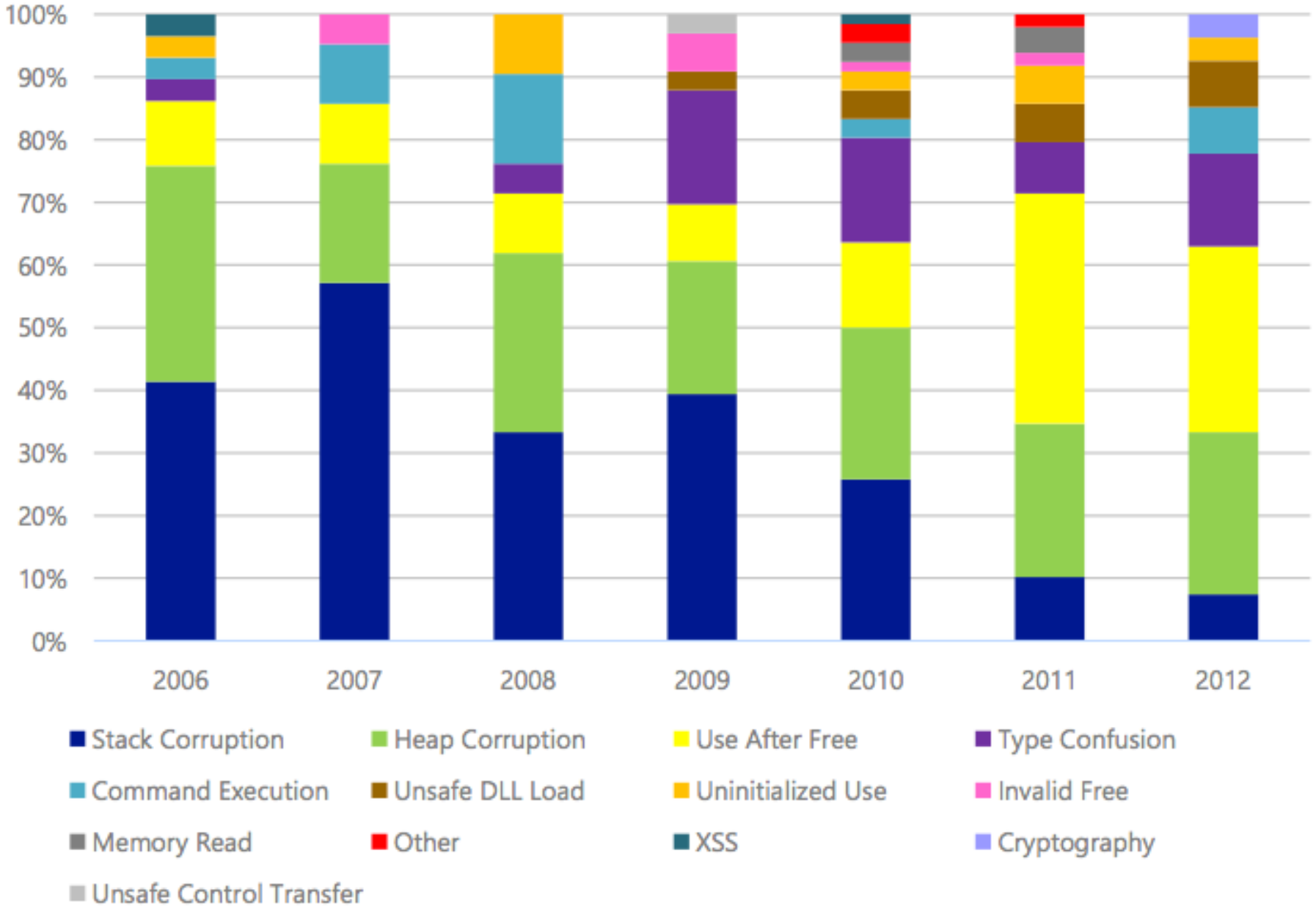


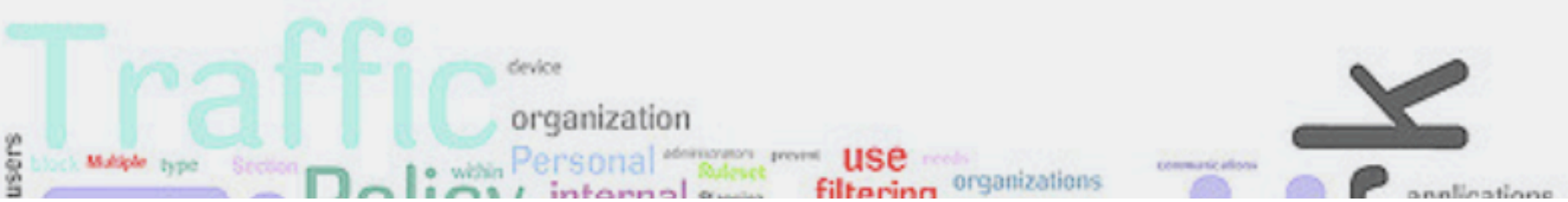
Figure 5. The distribution of CVE vulnerability classes for CVEs that are known to have been exploited

RISK ASSESSMENT —

Cisco confirms NSA-linked zeroday targeted its firewalls for years

Company advisories further corroborate authenticity of mysterious Shadow Brokers leak.

DAN GOODIN - 8/17/2016, 5:35 PM



The vulnerability is due to a buffer overflow in the affected code area.

References/Acknowledgements

- Aleph One's "Smashing the Stack for Fun and Profit" http://insecure.org/stf/smashstack.html
- Paul Makowski's "Smashing the Stack in 2011" http://paulmakowski.wordpress.com/2011/01/25/smashing-the-stack-in-2011/
- Blexim's "Basic Integer Overflows" http://www.phrack.org/issues.html?issue=60&id=10
- Return-to-libc demo http://www.securitytube.net/video/258
- https://cwe.mitre.org/documents/sources/SevenPerniciousKingdoms.pdf

- Pat Pannuto for slide reviews and listening to me complain about shellcode not working
- Professor J. Alex Halderman for slide reviews

Links

- http://seclists.org/bugtraq/1997/Aug/63
- https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/cowan/cowan.pdf
- https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf
- http://phrack.org/issues/56/5.html
- http://security.stackexchange.com/questions/20497/stack-overflows-defeating-canaries-aslr-dep-nx
- http://phrack.org/issues/58/4.html
- https://cseweb.ucsd.edu/~hovav/dist/rop.pdf

MALWARE

Malware

- We understand principles of software exploitation
- Time to learn what can be done with them
- malware - *a program that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system or otherwise annoying or disrupting the victim*
- Classified mostly by:
  - propagation method
  - payload type

Zero-day

- An attack against a previously unknown vulnerability
- Active attack with no time to fix the flaw
- “zero” days to patch the system

Malware definition and goals

- What is malware?
  - Set of instructions that run on your computer and do something an attacker wants it to do.
- Muddled Taxonomy, but difference primarily
  - How they get on your machine
  - What do they do

Encounter rate trends for the locations with the most computers reporting malicious and unwanted software encounters in 1H16, by number of computers reporting Country/Region

Country/Region	3Q15	4Q15	1Q16	2Q16
United States	10.8%	12.5%	11.9%	12.0%
China	14.9%	18.9%	19.1%	21.1%
Brazil	29.2%	34.4%	29.9%	29.4%
Russia	22.8%	28.7%	27.2%	24.9%
India	36.5%	44.2%	35.4%	32.6%
Turkey	32.6%	40.3%	34.8%	31.4%
France	18.8%	19.4%	17.0%	15.3%
Mexico	23.9%	28.5%	24.4%	23.8%
United Kingdom	11.9%	13.9%	13.7%	11.5%
Germany	12.2%	13.8%	13.0%	13.0%
Worldwide	*17.8%*	*20.8%*	*18.3%*	*21.2%*

How does malware run?

- Buffer overflow in network-accessible vulnerable service
- Vulnerable client (e.g. browser) connects to remote system that sends over an attack (a driveby)
- Social engineering: trick user into running/installing
- “Autorun” functionality (esp. from plugging in USB device)
- Slipped into a system component (at manufacture; compromise of software provider; substituted via MITM)
- Attacker with local access downloads/runs it directly
- Might include using a “local root” exploit for privileged access

Insider Attacks

- An **insider attack** is a security breach that is caused or facilitated by someone who is a part of the very organization that controls or builds the asset that should be protected.
- In the case of malware, an insider attack refers to a security hole that is created in a software system by one of its programmers.

Backdoors

- A **backdoor,** which is also sometimes called a **trapdoor,** is a hidden feature or command in a program that allows a user to perform actions he or she would not normally be allowed to do.
- When used in a normal way, this program performs completely as expected and advertised.
- But if the hidden feature is activated, the program does something unexpected, often in violation of security policies, such as performing a privilege escalation.
- Benign example: **Easter Eggs** in DVDs and software