CS356: Covert Channels and Bad Buffers

W. Michael Petullo

University of Wisconsin–La Crosse

As of September 22, 2021

Describe the reconnaissance phase.

1. Recon
2. Exploit
3. Persist
4. Move
5. Exfiltrate

Describe the reconnaissance phase.

1. Recon
2. Exploit
3. Persist
4. Move
5. Exfiltrate

passive

social engineering

`tcpdump` filters

active

`tcpdump`

`nmap -Pn`

public knowledge

`nmap`

`nmap -p0-65536`

Describe the exploitation phase.

1. Recon
2. Exploit
3. Persist
4. Move
5. Exfiltrate

Describe the exploitation phase.

1. Recon
2. Exploit
3. Persist
4. Move
5. Exfiltrate

programming errors	misconfigurations	missing protections
coming up!	`host4` webserver	unencrypted traffic

Describe the persistence phase.

1. Recon
2. Exploit
3. Persist
4. Move
5. Exfiltrate

Describe the persistence phase.

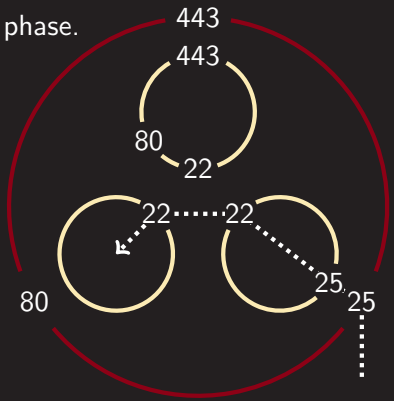
1. Recon
2. Exploit
3. Persist — convenient access — establish foothold — rootkits
4. Move
5. Exfiltrate

Describe the lateral movement phase.

1. Recon
2. Exploit
3. Persist
4. Move
5. Exfiltrate

Describe the lateral movement phase.

1. Recon
2. Exploit
3. Persist
4. Move
5. Exfiltrate



1. Recon
2. Exploit
3. Persist
4. Move
5. Exfiltrate

Today's lecture includes matters related to exfiltration: how to pull information out of a target network.

Covert Channels and Bad Buffers: Covert Channels

A covert channel is a communication channel that was not intended for the transfer of information. Some channels are not strictly covert but are still surprising.

Examples:

- Modulating information on CPU usage
- Modulating information on the presence or absence of a file
- File names that convey information: `cannot-read-but-the-secret-is-X.txt`
- Modulating information using the size of files
- `EADDRINUSE`-type channels: Port in use? That is a one.
- Abusing network header fields

The term Covert Channel comes from Butler Lampson "A Note on the Confinement Problem", 1973

Covert Channels and Bad Buffers: TCP over DNS

Sometimes network aim to prevent users from connecting to the Internet with the exception of DNS queries. DNS is so pervasive that it can be difficult to engineer a network that does not permit its use.

How can we circumvent this constraint to make broader use of the Internet?

TCP over DNS! Systems like iodine (`https://code.kryo.se/iodine/`) tunnel TCP through DNS. These systems usually require that you have control of a domain. The server transforms incoming DNS requests into outgoing TCP segments and vice versa.

n is one or more hostnames that represent TCP segment

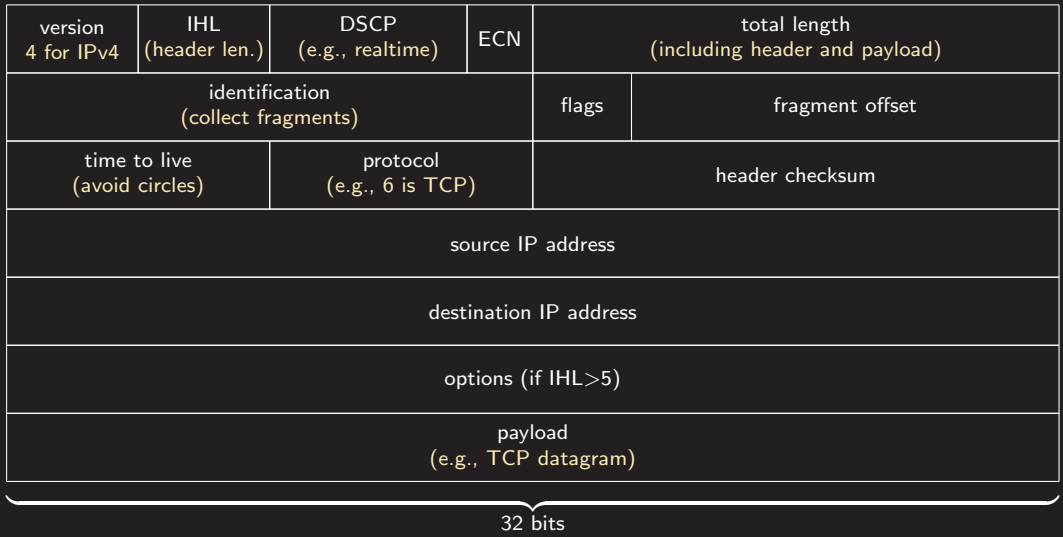
Covert Channels and Bad Buffers: Abusing Proto. Headers

Network protocol headers contain many fields that serve the purpose of describing how to transmit a message from one place to another.

These are not meant to contain application-layer data.

But, we can find cases where it is safe to place whatever we want in these fields. For example, IP's identification field is of little use when packets are not fragmented. `covert_tcp.c` stores two data bytes in this field.

Firewalls: Internet Protocol Packets



Covert Channels and Bad Buffers: Abusing Proto. Headers

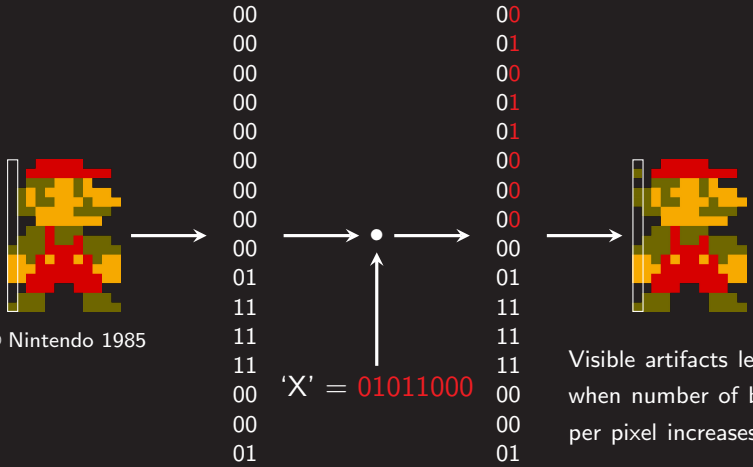
```
[user@host]$ gcc -o covert_tcp covert_tcp.c
[user@host]$ sudo ./covert_tcp -dest localhost
   -source localhost -dest_port=1025
   -source_port=1024 -server -file output
```

```
[user@host]$ sudo ./covert_tcp -dest localhost
   -source localhost -source_port 1024
   -dest_port 1025 -file /etc/hosts
```

- Watch progress of file with `watch cat output`.
- Watch traffic with Wireshark or `tcpdump`.

Covert Channels and Bad Buffers: Steganography

Steganography: concealing a message within another message.



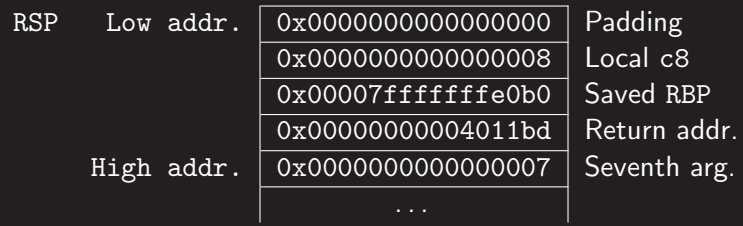
Example: encoding information in the least-significant bits of a bitmap.

Exploitation: AMD64 Call Stack

```
void foo(long c1, long c2, long c3, long c4,
    long c5, long c6, long c7) {
    long c8 = 8;
    bar(c7, c8);        <-- RIP
}

foo(1, 2, 3, 4, 5, 6, 7);
```

RDI: 1 RSI: 2 RDX: 3 RCX: 4 R8: 5 R9: 6

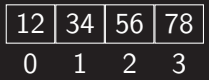


- Stack grows from high to low addresses (down)
- First six args. in registers RDI, RSI, RDX, RCX, R8, and R9
- Subsequent args. on stack
- Return value on stack
- Local variables on stack (i.e., array bytes go towards return address)

Exploitation: Endianness

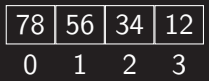
Different instruction set architectures store numbers differently. Big endian places more significant bytes in lower addresses:

`uint32_t n = 0x12345678`



Little endian places more significant bytes in higher addresses:

`uint32_t n = 0x12345678`



AMD64 is little endian, the network order is big endian, and your debugger will display numbers naturally.

Exploitation: Endianness: Convince Yourself

```c
#include <stdint.h>

int main(void) {
    uint32_t n = 0x12345678;
}
```

using gdb to confirm AMD64 endianness

```
gcc -g -o endian endian.c
gdb endian
(gdb) break endian.c:5
(gdb) run
(gdb) print/x n
$3 = 0x12345678
(gdb) printf "%x%x%x%x\n",
    ((char *)(&n))[0], ((char *)(&n))[1],
        ((char *)(&n))[2], ((char *)(&n))[3]
78563412
```

Debugger will present numbers naturally!

Debugger:
0x0000000000400363

AMD64 C:

```
"\x63\x03\x40\x00"
"\x00\x00\x00\x00"
```

Crash gets.c

- GDB's `break FN` or `break SOURCE:LINE`
- GDB's `run`
- GDB's `next`
- GDB's `backtrace`
- GDB's `x/16xg` (`$rsp`)
- GDB's `disassemble`
- Draw stack
- Crash with direct use of `echo` and pipe
- Capture input and watch with debugger (`run <input`)
- Talk through network attack

```
#include <stdio.h>

void bar(void)
{
    printf("Hack!\n");
}

void foo(void)
{
    char buf[16];
    gets(buf);
}

int main(void)
{
    foo();
}
```

Covert Channels and Bad Buffers: Assignments

Graded Homework Aquinas: network and overflow in C

Reading 0x321–0359

https://www.flyn.org/courses/cs356-2021-fall/schedule