Other Vulnerabilities & Fixes

Review

Attackers would like to plant shellcode then change a return address to point to their shellcode so it can execute. The most convenient place to put shellcode is on the stack

Other Vulnerabilities & Fixes

Review

Attackers would like to plant shellcode then change a return address to point to their shellcode so it can execute. The most convenient place to put shellcode is on the stack

So, processor manufacturers support **DEP** (Data Execution Protection) which allows assigning a r/w bit to each page in memory – execution of instructions is prohibited from pages that are read only, including the stack

Other Vulnerabilities & Fixes

Review

Attackers would like to plant shellcode then change a return address to point to their shellcode so it can execute. The most convenient place to put shellcode is on the stack

So, processor manufacturers support **DEP** (Data Execution Protection) which allows assigning a r/w bit to each page in memory – execution of instructions is prohibited from pages that are read only, including the stack

So, attackers ran code in system libraries, which must be in executable pages. The stack is modified so the return address is, for example, to `system` with the address of `/bin/sh` in the parameter section of the stack

Other Vulnerabilities & Fixes

Review

Attackers would like to plant shellcode then change a return address to point to their shellcode so it can execute. The most convenient place to put shellcode is on the stack

So, processor manufacturers support **DEP** (Data Execution Protection) which allows assigning a r/w bit to each page in memory – execution of instructions is prohibited from pages that are read only, including the stack

So, attackers ran code in system libraries, which must be in executable pages. The stack is modified so the return address is, for example, to `system` with the address of `/bin/sh` in the parameter section of the stack

So, kernel developers use **ASLR** (Address Space Layout Randomization) to make it harder to find those addresses

Other Vulnerabilities & Fixes

Review

Attackers would like to plant shellcode then change a return address to point to their shellcode so it can execute. The most convenient place to put shellcode is on the stack

So, processor manufacturers support **DEP** (Data Execution Protection) which allows assigning a r/w bit to each page in memory – execution of instructions is prohibited from pages that are read only, including the stack

So, attackers ran code in system libraries, which must be in executable pages. The stack is modified so the return address is, for example, to `system` with the address of `/bin/sh` in the parameter section of the stack

So, kernel developers use **ASLR** (Address Space Layout Randomization) to make it harder to find those addresses

Can attackers use tricks to find those addresses or maybe use addresses that are not randomized?

Other Vulnerabilities & Fixes

IPv6 privacy

IPv6 address has 128 bits – example, my laptop:

`fe80::a6db:30ff:fe29:61c5/64`

means MAC address is there

But my laptop MAC address is

`a4:db:30:29:61:c5`

2nd LSB is reversed

So, my IPv6 address exposes bits of my MAC address

Other Vulnerabilities & Fixes

IPv6 privacy

IPv6 address has 128 bits – example, my laptop:

`fe80::a6db:30ff:fe29:61c5/64`

means MAC address is there

But my laptop MAC address is

`a4:db:30:29:61:c5`

2^{nd} LSB is reversed

So, my IPv6 address exposes bits of my MAC address

Is this something to worry about?

Part of the IPv6 address is fixed regardless of location

ipv6-test.com shows that the MAC address is public via IPv6

might be possible to track laptop/mobile phone using data collected from WiFi networks (data mining logs)

can be used to falsify a device's MAC address to get access to some service where MAC address is white-listed

Other Vulnerabilities & Fixes

IPv6 privacy – why did they do it?

generation of the address is stateless

thus devices on a network can configure their IP addresses without the need of a server

How can privacy be restored?

Cryptographically generated addresses

Hashed info about a host's public key is in the identifier part of the address – binding of addresses to public keys

Temporary address

Temporary addresses that change every hour or so are generated. An example:

`e4:ce:8f:00:93:3e`	MAC address
`fe80::e6ce:8fff:fe00:953e%en0`	link-local address
`2001:db8:1f15:d79:e6ce:8fff:fe00:933e`	IPv6 addr
`2001:db8:1f15:d79:1511:ed4a:b5bc:4420`	temp addr

Other Vulnerabilities & Fixes

IPv6 privacy – enable?

Windows

temporary addresses enabled by default

```
netsh interface ipv6 set privacy state=enabled
```

Linux

temporary addresses enabled by default

```
sysctl net.ipv6.conf.all.use_tempaddr=2
sysctl net.ipv6.conf.default.use_tempaddr=2
```

MAC OS X

temporary addresses enabled by default

```
sysctl -w net.inet6.ip6.use_tempaddr=1
```

Other Vulnerabilities & Fixes

Restrict visibility of kernel symbol & module locations

Malicious agent may wish to exploit knowledge of location of modules, data structures, kernel symbols to attack kernel

Some of that information is available in files. For example:

```
/proc/kallsyms
/proc/modules
```

Linux

these addresses can be hidden by the following

```
sysctl kernel.kptr_restrict=1
```

Note:

```
KernSymb/kernsym.c
```

is an example of how a system variable address may be printed

Other Vulnerabilities & Fixes

time-of-check-time-of-use cross-privilege attack

From `https://cwe.mitre.org/data/definitions/367.html`

Software checks the state of a resource before using it
Resource's state changes after the check, before the use
so that the result of the check is invalid.

Software performs invalid operations when the resource is
in an unexpected state

This happens with shared resources in multi-threaded progs

Consequences:

- Attacker can access unauthorized resources
- Race condition may allow r/w access, otherwise denied
- Resource may be changed in unwanted way
- There may be no log of this event
- Files may be deleted by an attacker

Other Vulnerabilities & Fixes

time-of-check-time-of-use cross-privilege attack

Example:

```
if(!access(file, W_OK)) {
    ...
    f = fopen(file, "w+");
    operate(f);
    ...
} else {
    fprintf(stderr,"Unable to open file %s.\n",file);
}
```

Root execs to perform operation on '`file`' on behalf of user

Performs access check on '`file`' to make sure user has privileges

Between `access` and `fopen` attacker relinks 'file'

Operation proceeds on new file with root privileges

Feasbility: see `access.cc`

Other Vulnerabilities & Fixes

time-of-check-time-of-use cross-privilege attack

Protection:

Ensure that locking occurs before the check, as opposed to afterwards, so that the resource, as checked, is the same as it is when in use.

Recheck the resource after the use call to verify that the action was taken appropriately.

Linux

```
sysctl fs.protected_hardlinks = 1
sysctl fs.protected_symlinks = 1
```

Permit symlinks to only be followed when outside a sticky world-writable directory, or when the uid of the symlink and follower match, or when the directory owner matches the symlink's owner

Permit hardlinks to only be created when the user is already the existing file's owner, or if they already have read/write access to the existing file.

Other Vulnerabilities & Fixes

time-of-check-time-of-use cross-privilege attack

Actual Example:

Server script wrote private and public keys into temp files then read those keys and put them into a database

Because the temp files were in a publicly writable directory, an attacker was able to create a race condition by substituting the attacker's own files before the keys were reread causing the script to insert the attacker's private and public keys instead.

After that, anything encrypted or authenticated using those keys was under the attacker's control.

Alternatively, the attacker can read the private keys, which can be used to decrypt encrypted data. [CVE-2005-2519]

Other Vulnerabilities & Fixes

time-of-check-time-of-use cross-privilege attack

Actual Example:

Several threads attempt to fill a buffer

All threads check that the contents do not overrun it

All threads increment an index variable pointing to the next character position in the buffer

The action of all the threads together causes a buffer overrun

See `multiple.c`

Other Vulnerabilities & Fixes

DoS via syn flood attack

What is going on?

Client

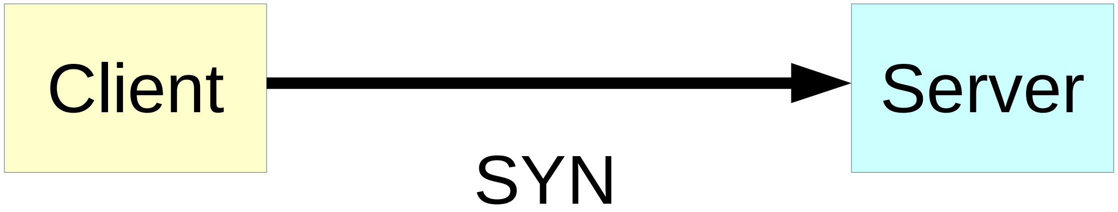
Server

Client wants to open a connection to the server

Other Vulnerabilities & Fixes

DoS via syn flood attack

What is going on?

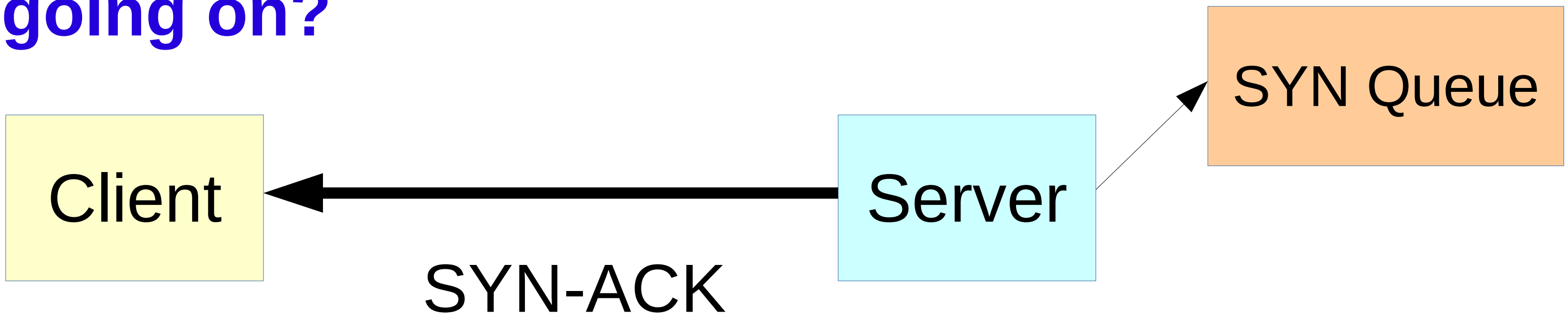


Client wants to open a connection to the server
Client send a 'SYN' packet to the server – an initial sequence number is sent to the Server – Server must synchronize

Other Vulnerabilities & Fixes

DoS via syn flood attack

What is going on?

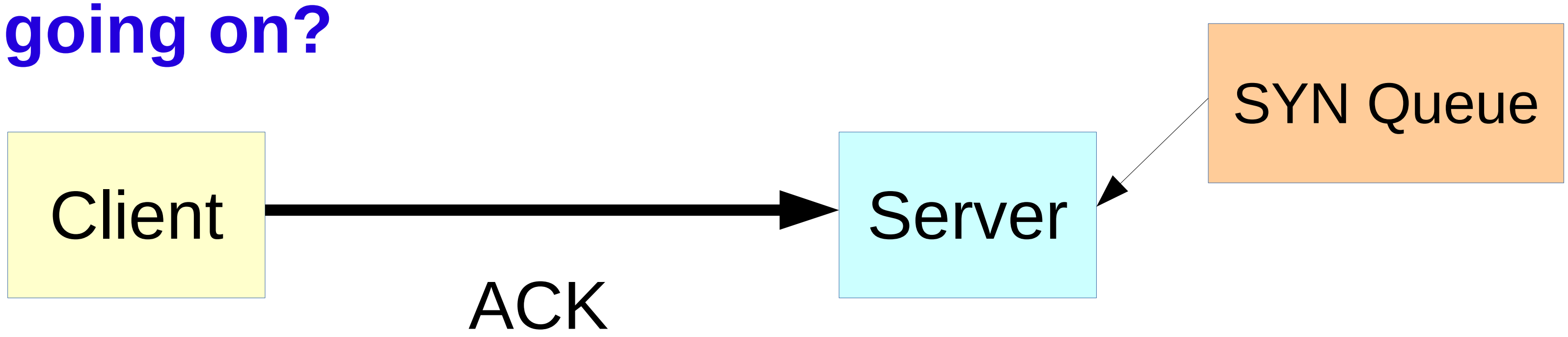


Client wants to open a connection to the server
Client send a 'SYN' packet to the server – an initial sequence number is sent to the Server – Server must synchronize
Server sends 'SYN' packet requesting synchronization of Sequence number and 'ACK' to acknowledge receipt of Client's request. Server records Client sequence number, waits up to 75 seconds for a response

Other Vulnerabilities & Fixes

DoS via syn flood attack

What is going on?



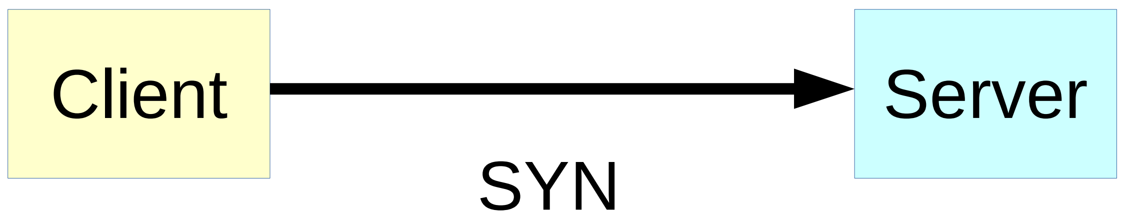
Client wants to open a connection to the server
Client send a 'SYN' packet to the server – an initial sequence number is sent to the Server – Server must synchronize
Server sends 'SYN' packet requesting synchronization of Sequence number and 'ACK' to acknowledge receipt of Client's request. Server records Client sequence number, waits up to 75 seconds for a response
Client sends 'ACK' to Server. The 'ACK' packets are used to confirm accurate receipt of sequence numbers

Note: state is saved in the Server for each connection. If enough clients fail to send an 'ACK' the server runs out of Space and cannot complete any more handshakes

Other Vulnerabilities & Fixes

DoS via syn flood attack

What is a SYN cookie?



Server's initial sequence number (32 bits):

t mod 32	MSS	Client & Server IP addresses & ports + t

t: a 32-bit time counter that increases every 64 seconds

MSS: Maximum Segment Size. The MSS field in the cookie is a Server selected encoding of Client's MSS.

Rightmost field (24 bits): IP addresses, port numbers, and t are encrypted by a function selected by the Server.

Note: It is unnecessary to put a pending connection into a SYN Queue until after the handshake.

Other Vulnerabilities & Fixes

Null pointer dereference

`https://blogs.oracle.com/ksplice/entry/much_ado_about_null_exploiting1`

Basic attack:

`mmap` a process to the zero page.

The 0 page contains kernel functions like exception handlers

Create a pointer to a function at the 0 address, have it point to a malicious function

Do something to raise an exception.

Then the malicious function will be called instead

Mitigation:

Set a minimum address for mapping memory

`sysctl vm.mmap_min_addr = 65536`

Other Vulnerabilities & Fixes

Kernel Memory Leaks

Description:

Unintentional form of memory consumption

Developer fails to free an allocated block of memory

Leaks in kernel code may result in instability (DoS) or in attacker gaining knowledge of addresses

Causes:

Handling of error cases

Confusion over which code segments are responsible for freeing memory

Use by Attacker:

Attacker intentionally triggers a leak, finds out addresses

Reason: ASLR only randomizes the location of pages - populating a region within a page with shellcode allows accurate execution of the shellcode using a leaked address to find the page base address

Other Vulnerabilities & Fixes

Kernel Memory Leaks

Example:

Use bug in keyring facility

Keyring used by drivers to manage security data

Function `keyctl` provided in user space

Kernel has `process_key` module

Every process can create keyring from user space with

`keyctl(KEYCTL_JOIN_SESSION_KEYRIGN, name)`

If a process already has a session keyring, this system call will replace its keyring with a new one.

Replacing the current session keyring with the same one bypasses the `key_put` function (the bug) which otherwise would release the keyring.

Hence, do `cat /proc/keys` after invoking the bug from `leak.c` to see that the added keyring is still around

Other Vulnerabilities & Fixes

Covert Channels

Description:

Any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy.

Example:

Hacker puts data to transfer in the IP headers
Bypasses filrewall checks from the inside
Bypasses sniffers looking for nefarious things

See `covert_tcp.c`

Why is this Interesting to a Malicious Agent?

A covert channel can be used for command & control, bypassing monitor alerts

Agents can communicate with each other covertly
Steganography

Other Vulnerabilities & Fixes

Covert Channels

Example:

AES uses table lookups – timing of the algorithm depends on what was or was not looked up, hence timings are dependent on key bits

Example:

Reworking generation of numbers for RSA keys can result in covert information as part of the public key