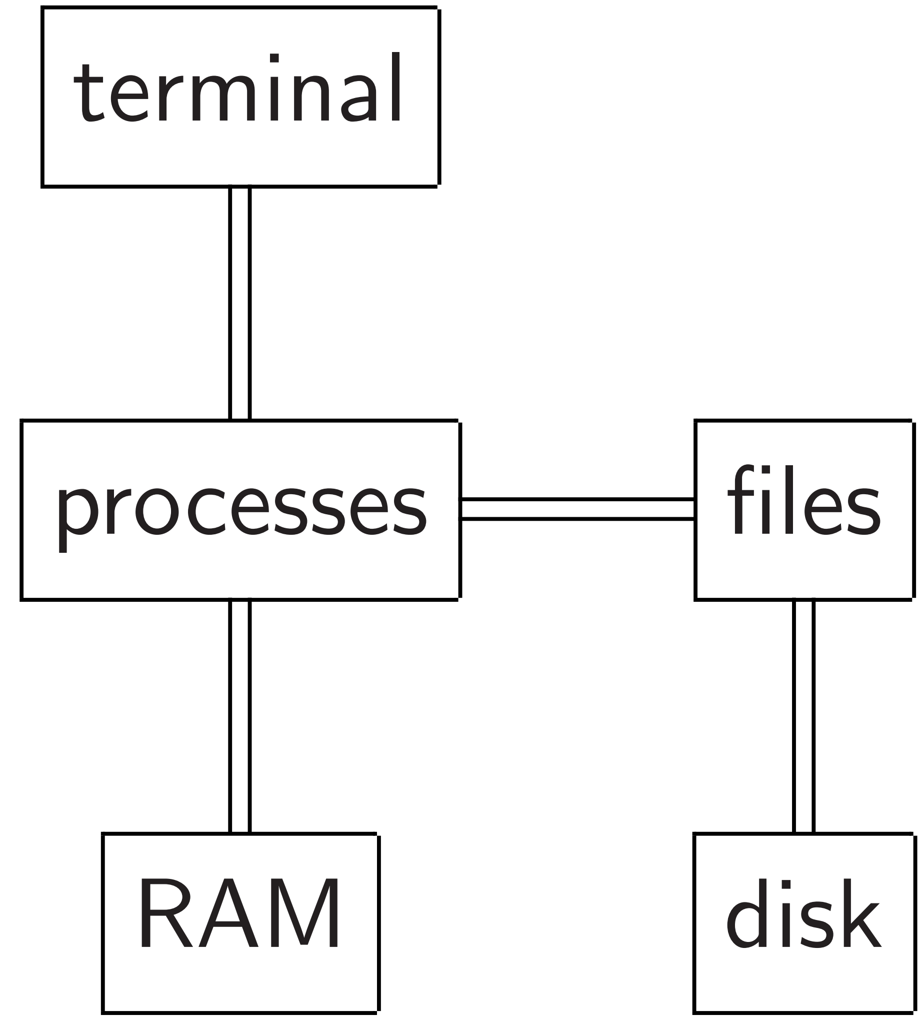
Usable verification of fast cryptographic software

Daniel J. Bernstein

University of Illinois at Chicago &
Technische Universiteit Eindhoven

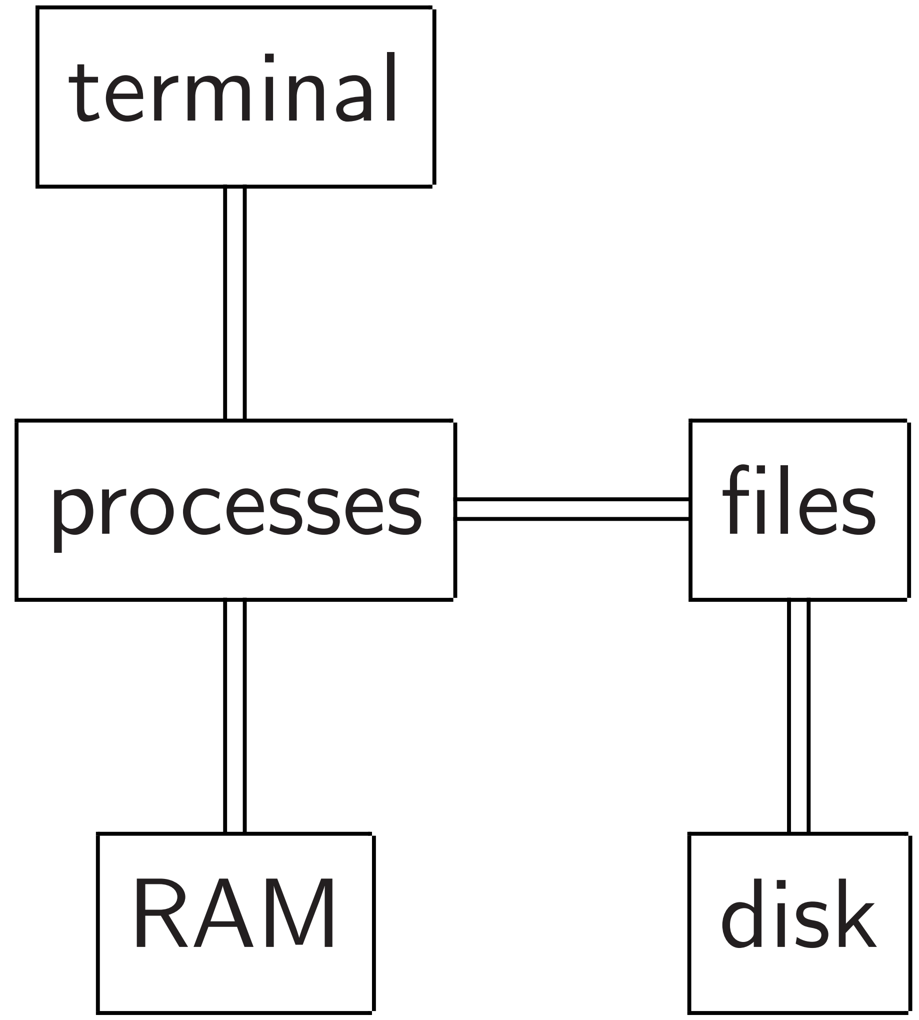


Operating-system kernel
divides RAM among processes,
divides disk among files.
Provides convenient functions
for processes to access files,
start new processes, etc.

erification of
tographic software

. Bernstein

ty of Illinois at Chicago &
che Universiteit Eindhoven

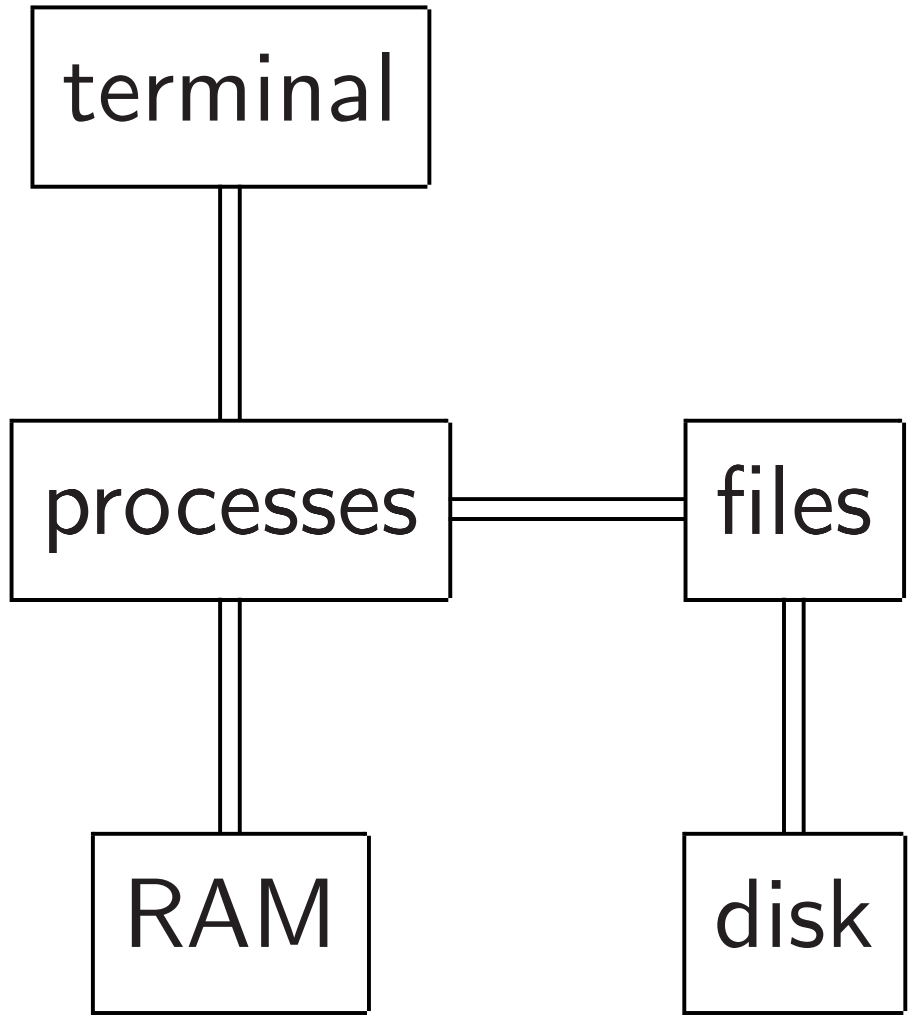


Operating-system kernel
divides RAM among processes,
divides disk among files.
Provides convenient functions
for processes to access files,
start new processes, etc.

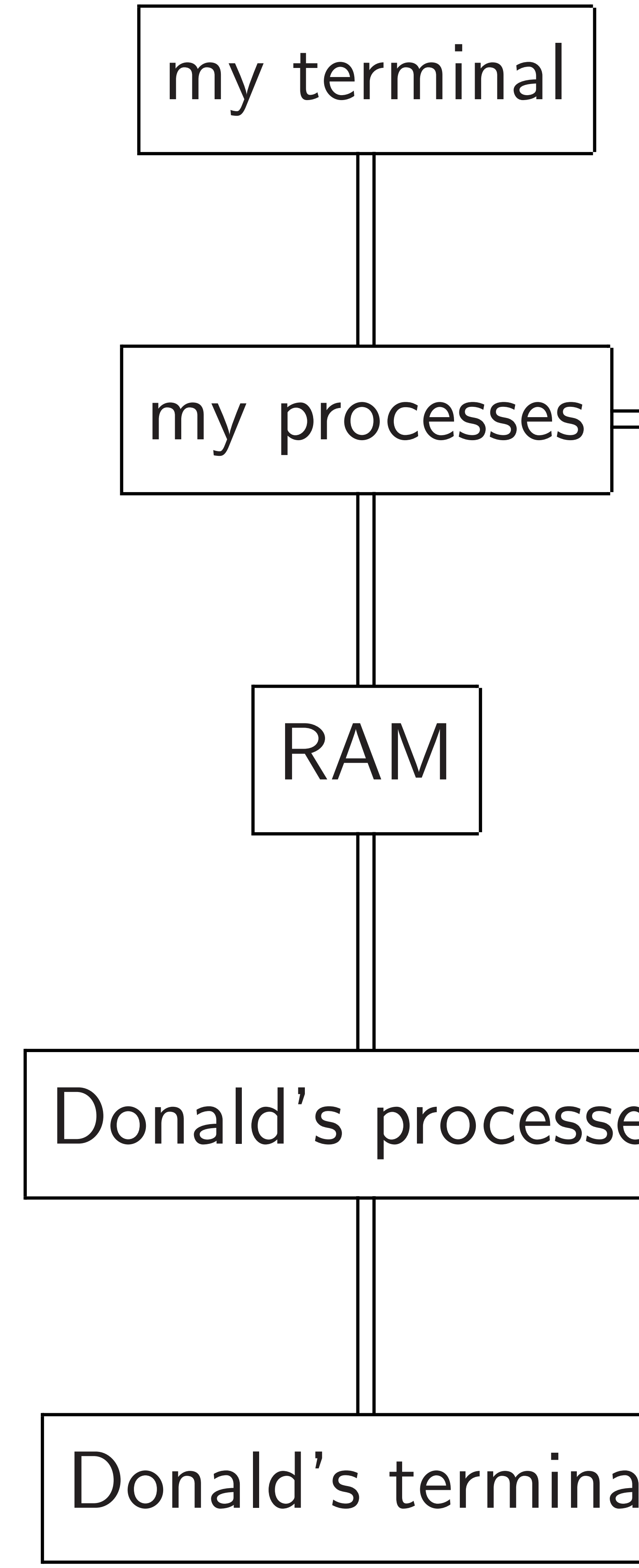
of
software

n

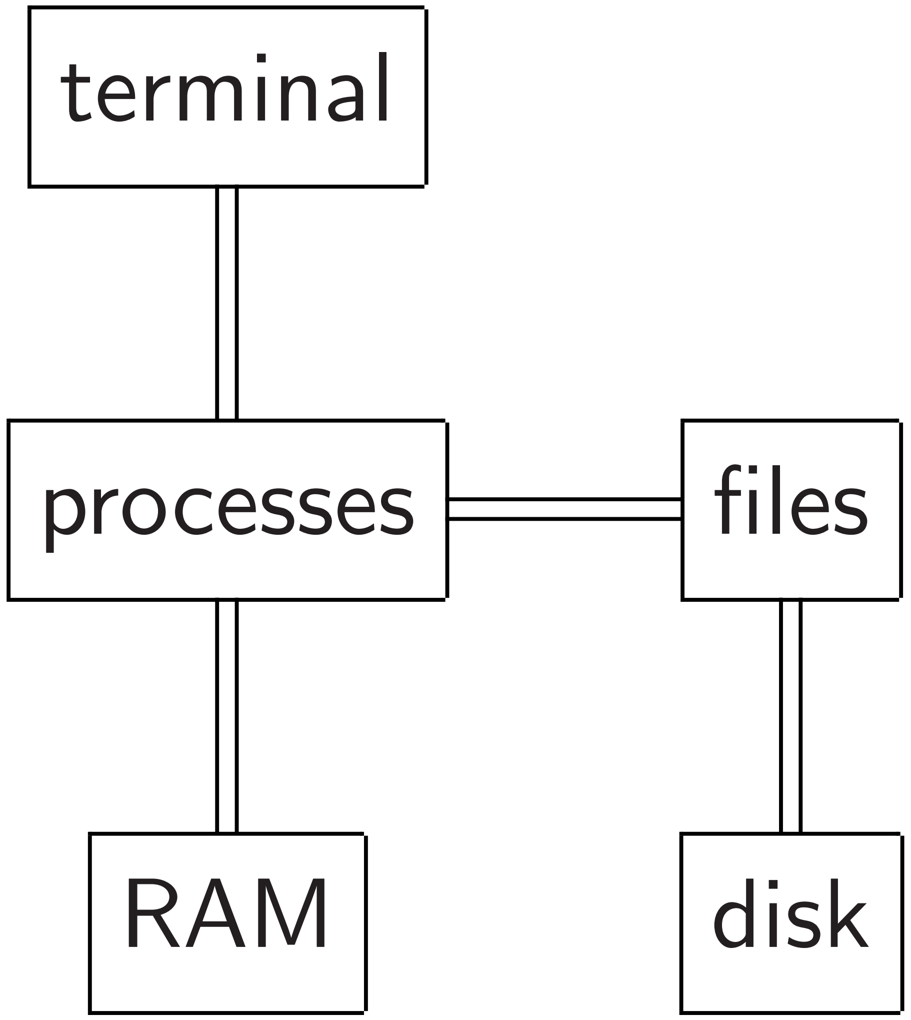
is at Chicago &
siteit Eindhoven



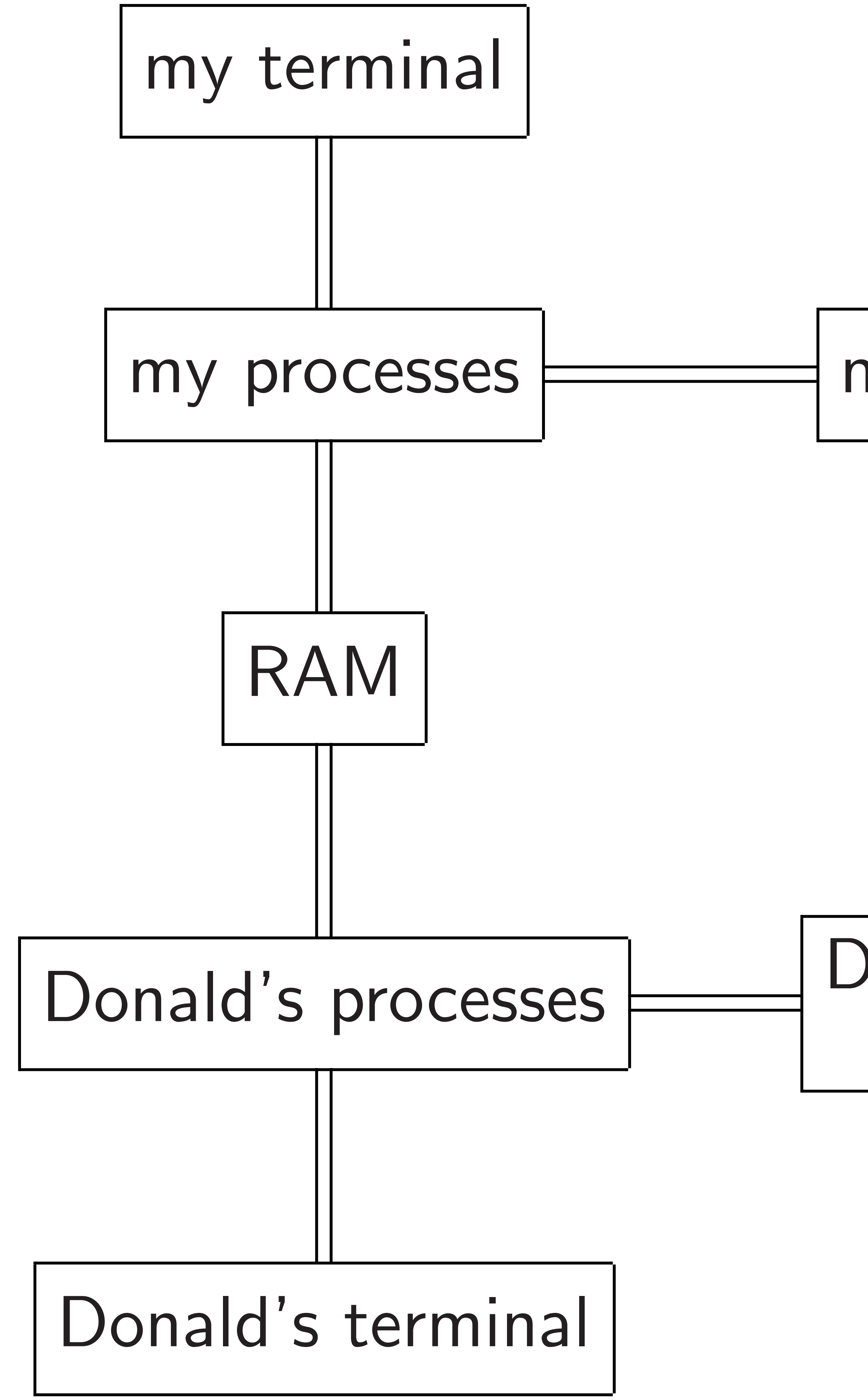
Operating-system kernel
divides RAM among processes,
divides disk among files.
Provides convenient functions
for processes to access files,
start new processes, etc.

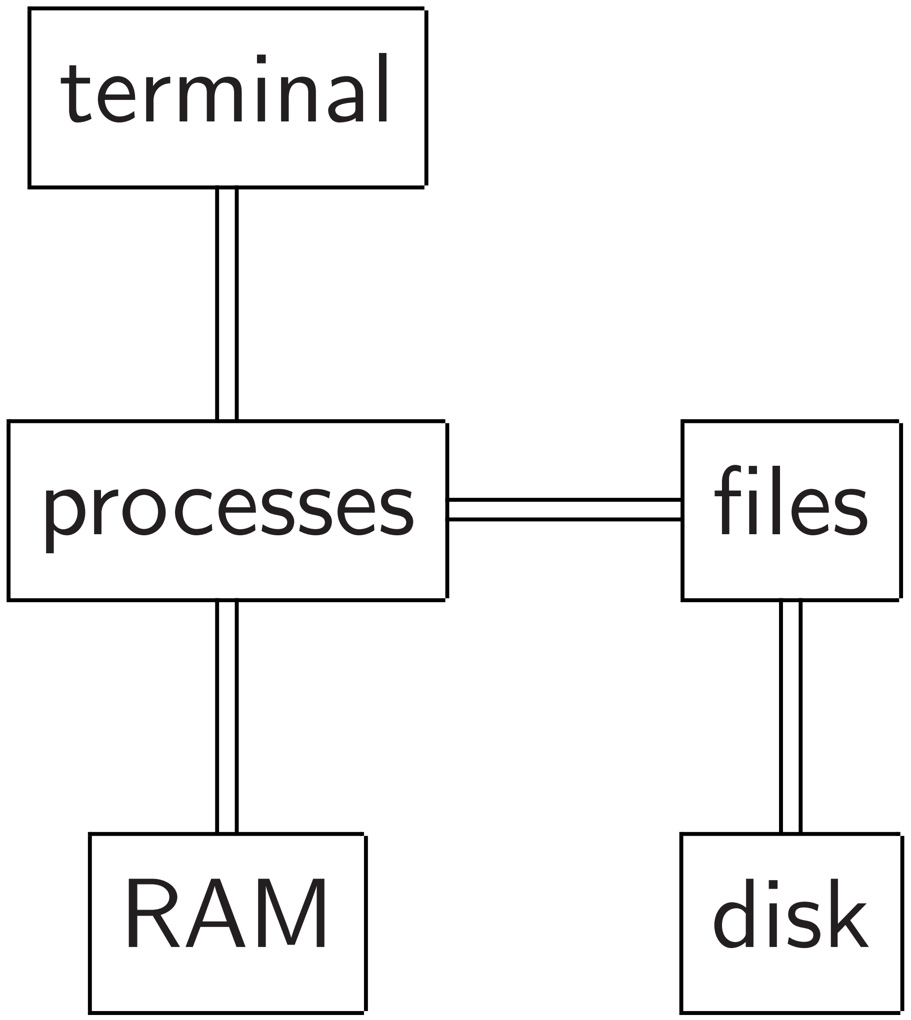


ago &
hoven

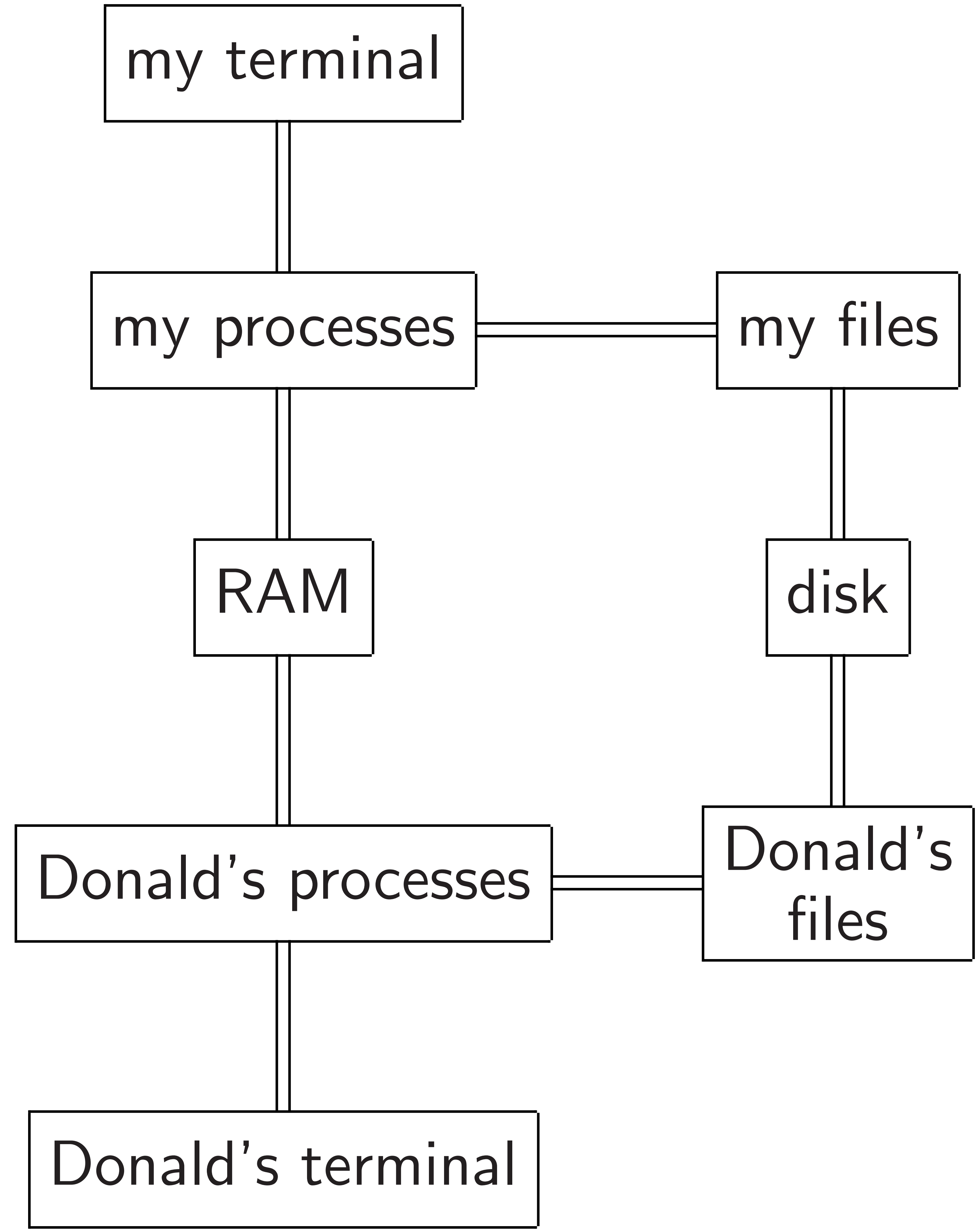


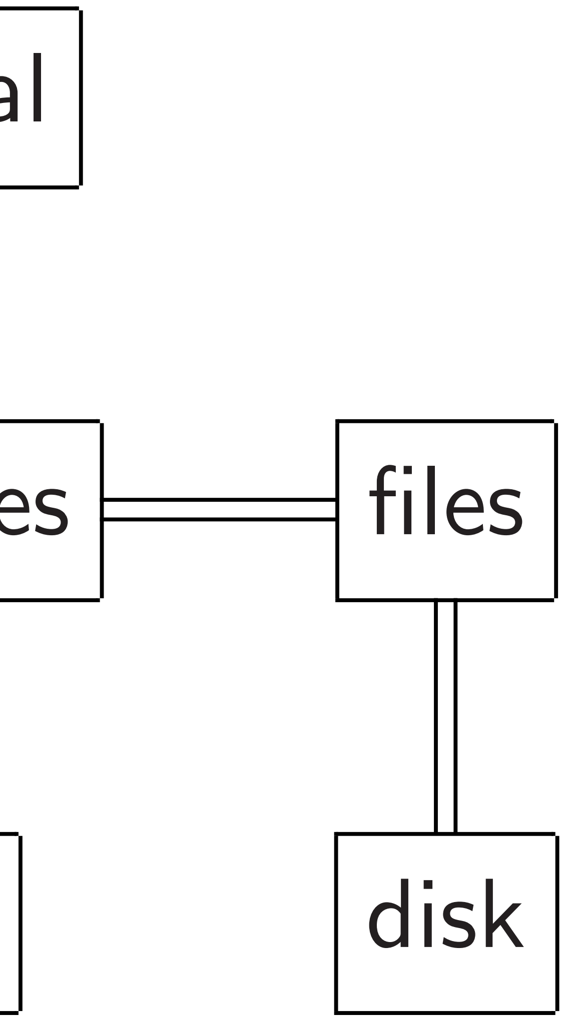
Operating-system kernel
divides RAM among processes,
divides disk among files.
Provides convenient functions
for processes to access files,
start new processes, etc.



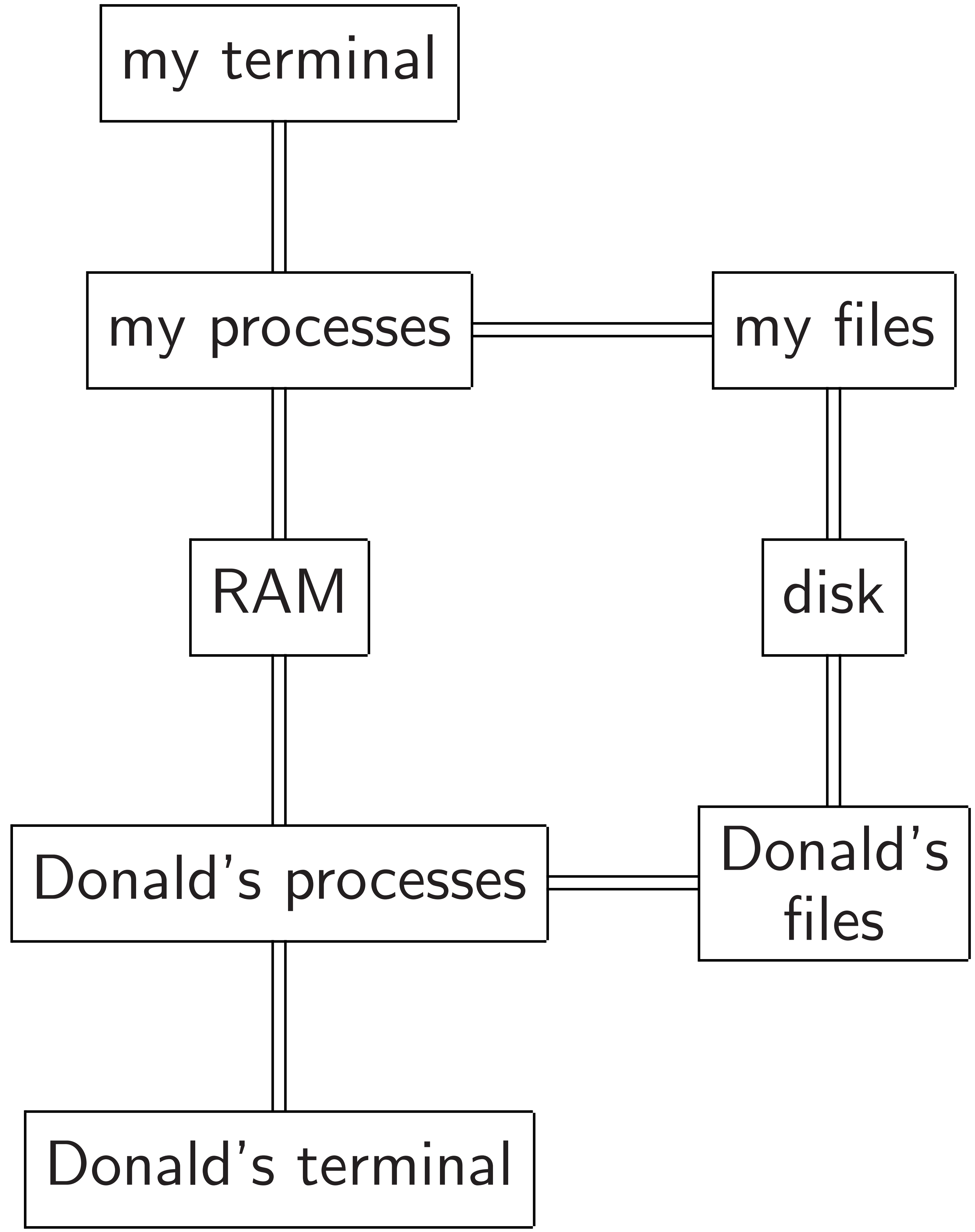


Operating-system kernel divides RAM among processes, divides disk among files. Provides convenient functions for processes to access files, start new processes, etc.

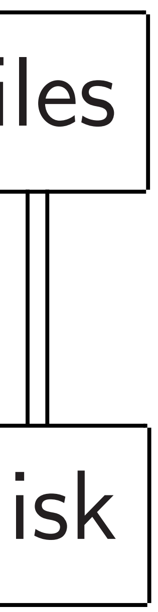




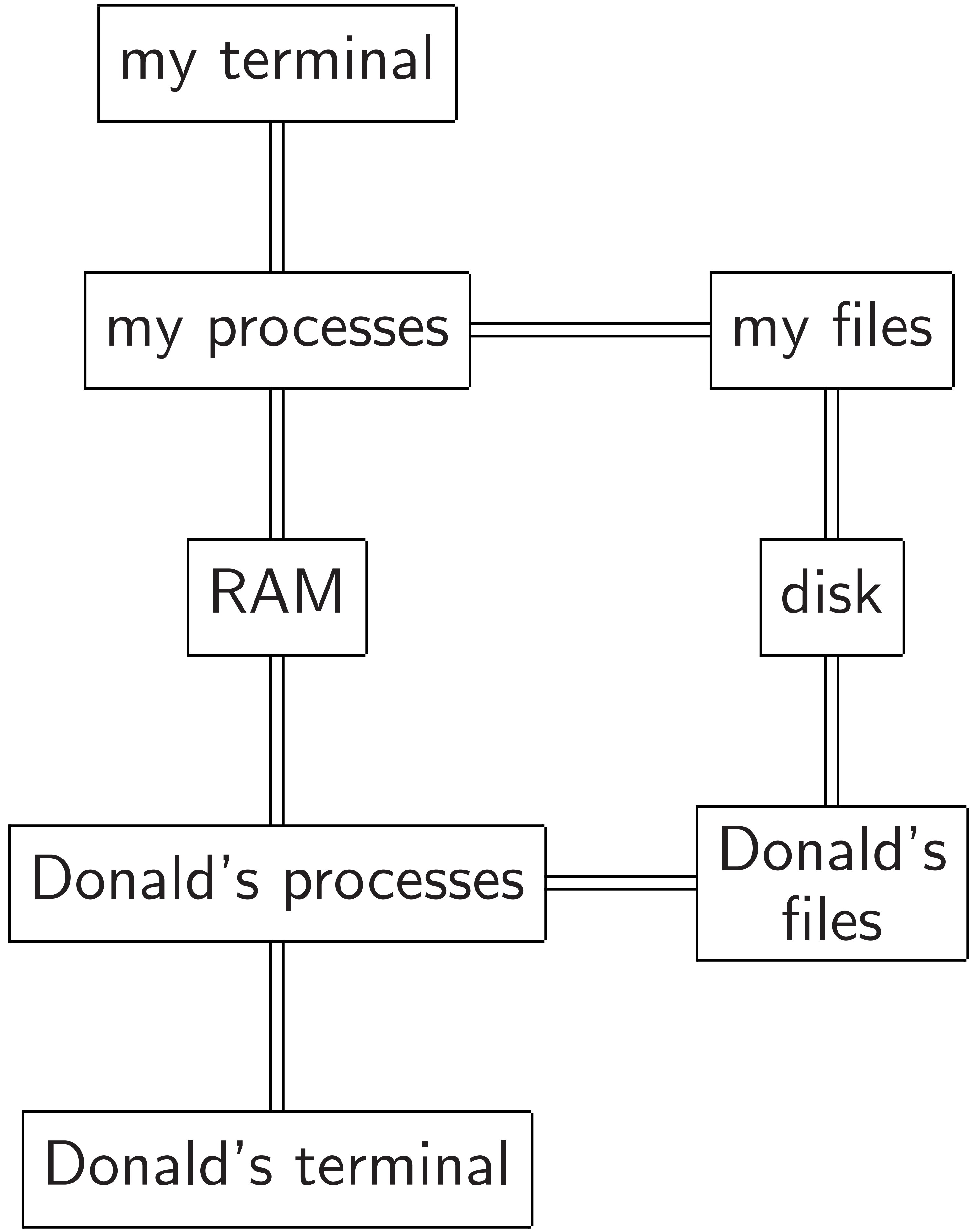
ng-system kernel
RAM among processes,
disk among files.
s convenient functions
esses to access files,
w processes, etc.



Can Dor
appearin



kernel
ng processes,
g files.
nt functions
ccess files,
es, etc.



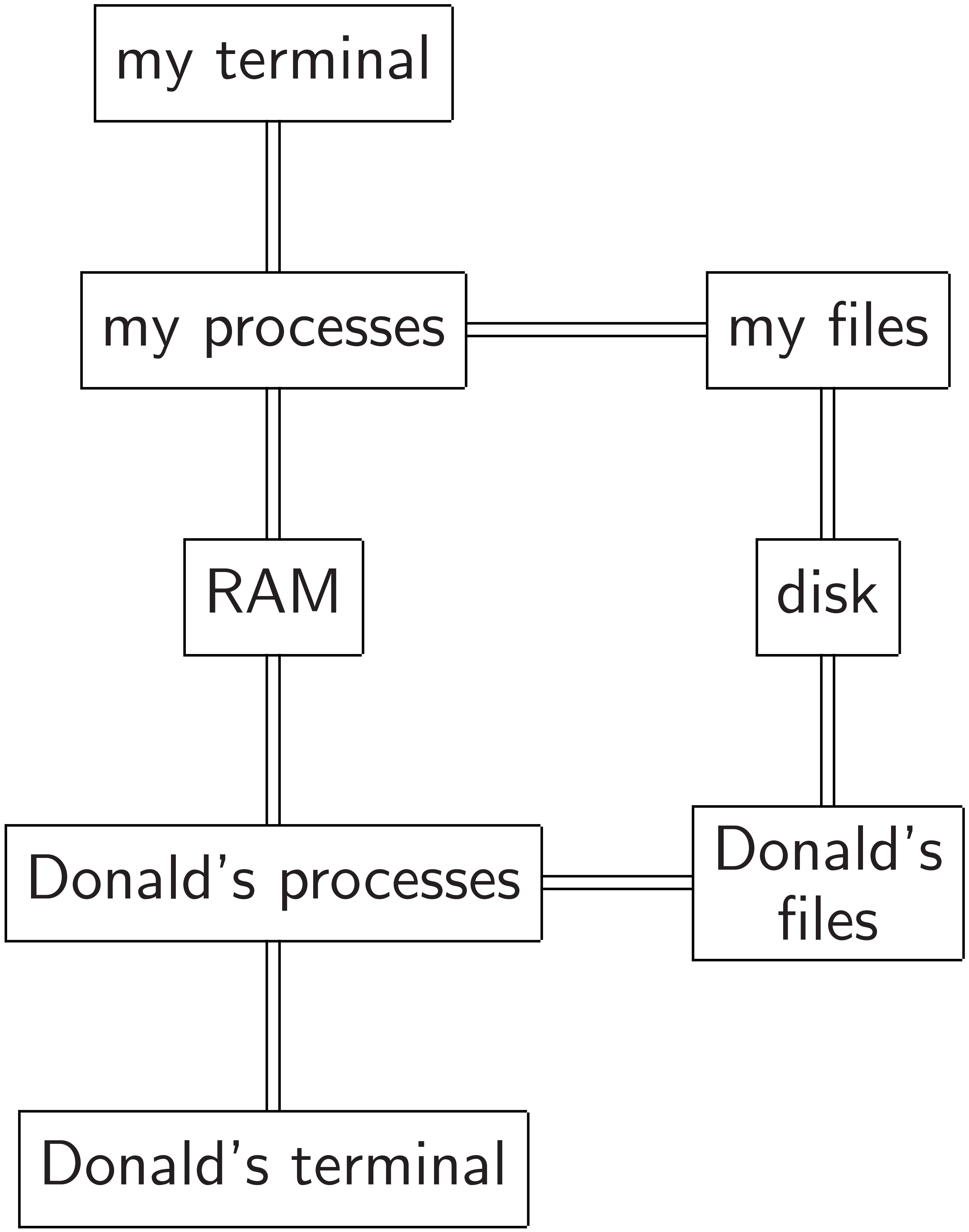
Can Donald corru
appearing on my t

2

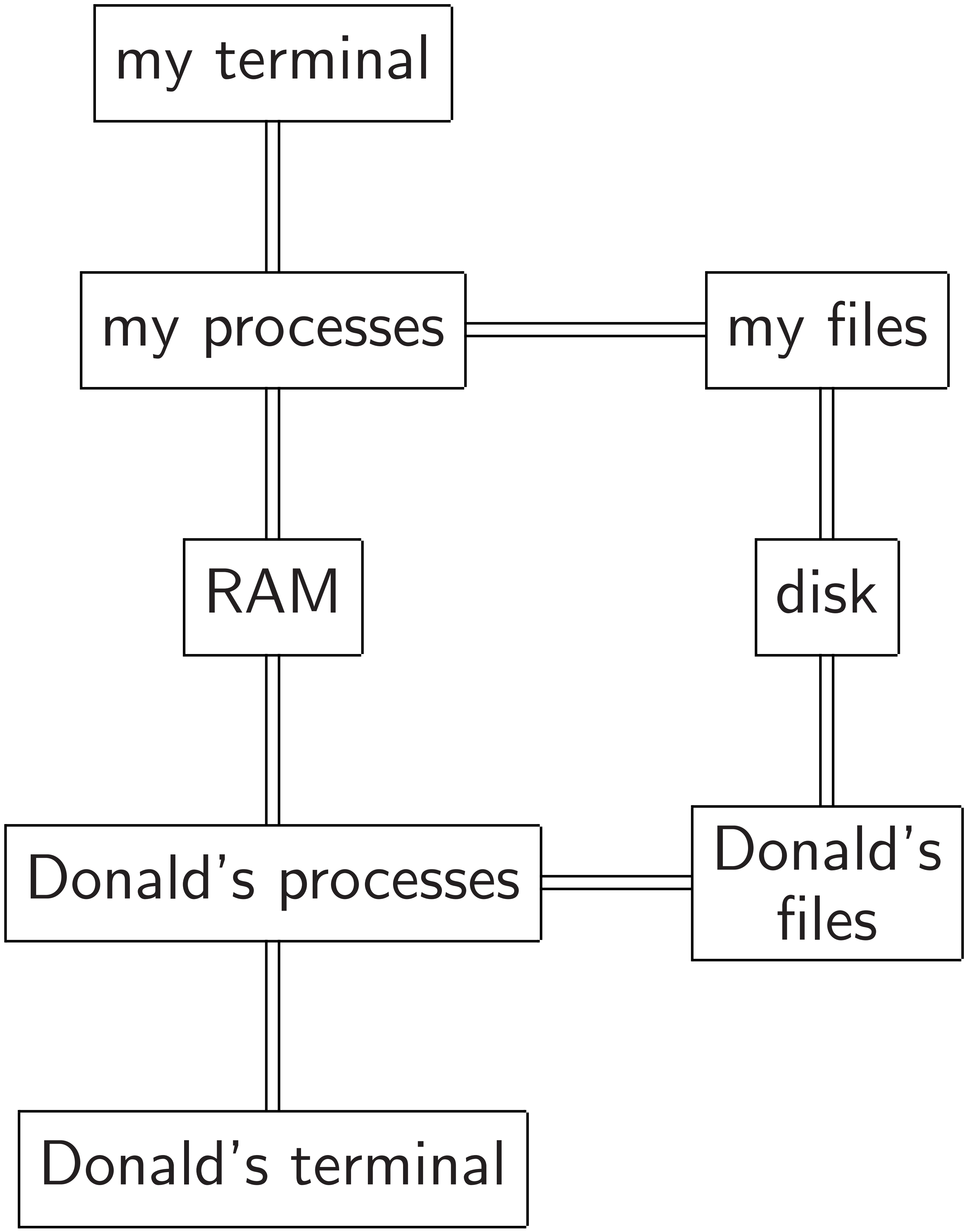
es,

ns

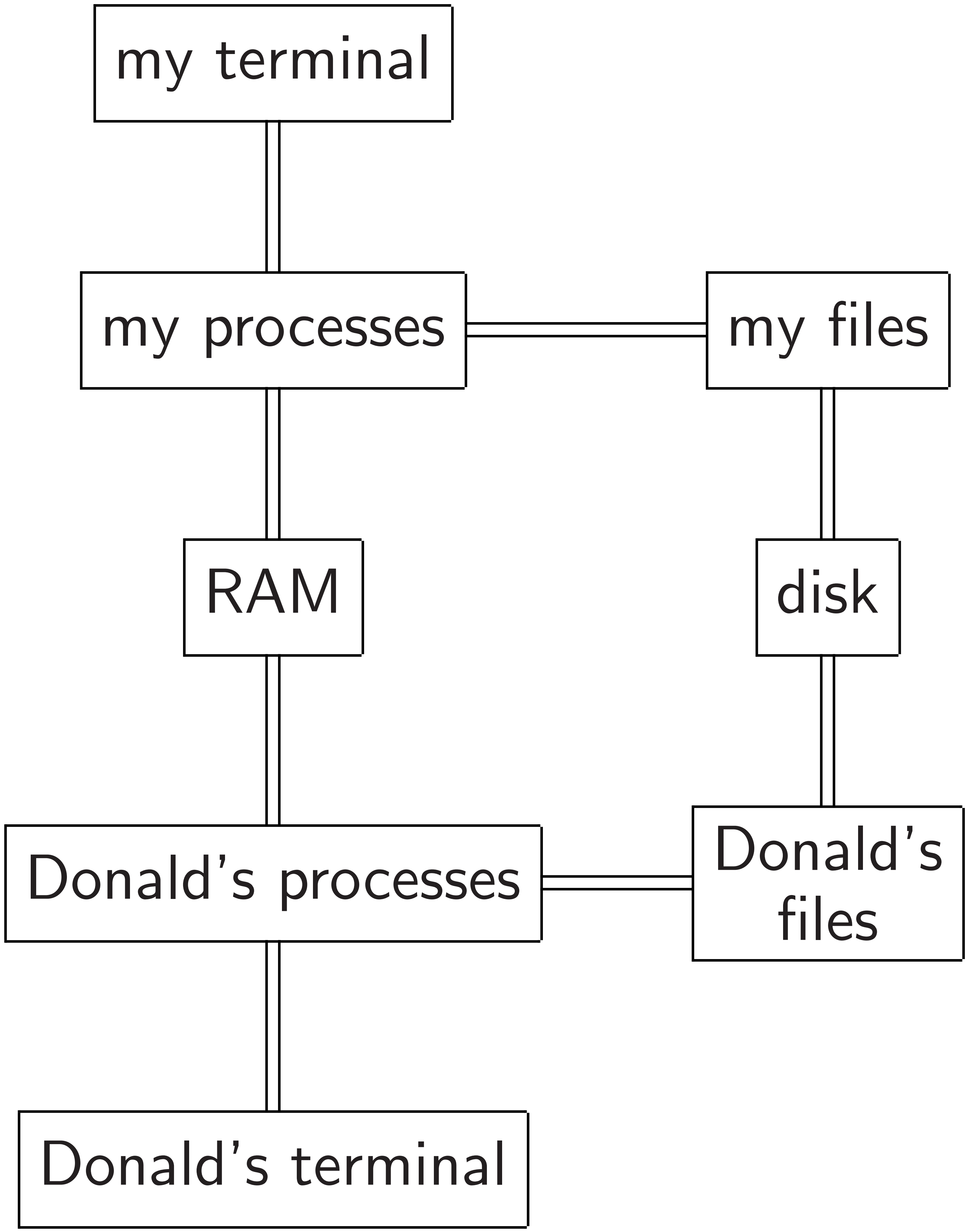
3



Can Donald corrupt the dat
appearing on my terminal?

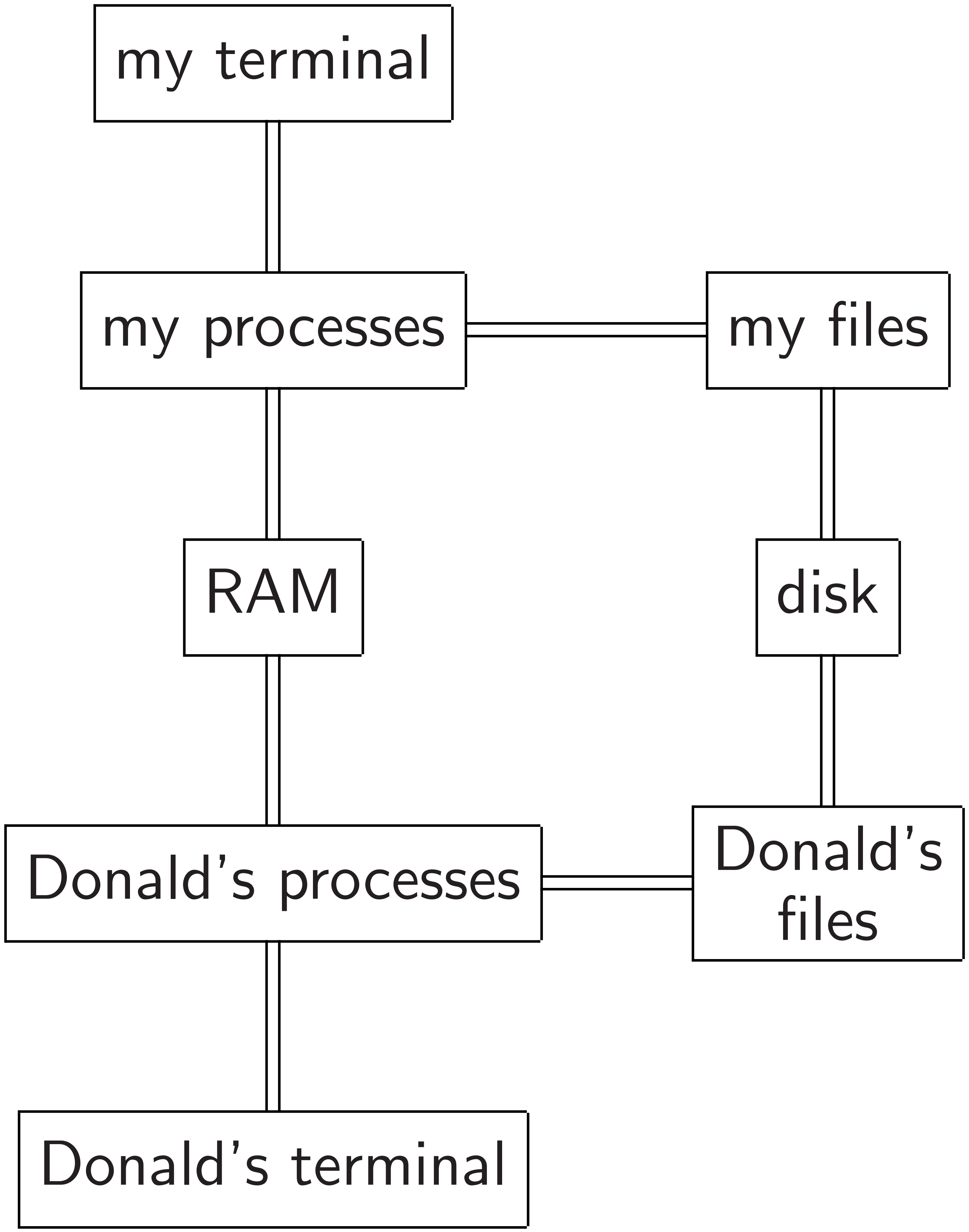


Can Donald corrupt the data appearing on my terminal?



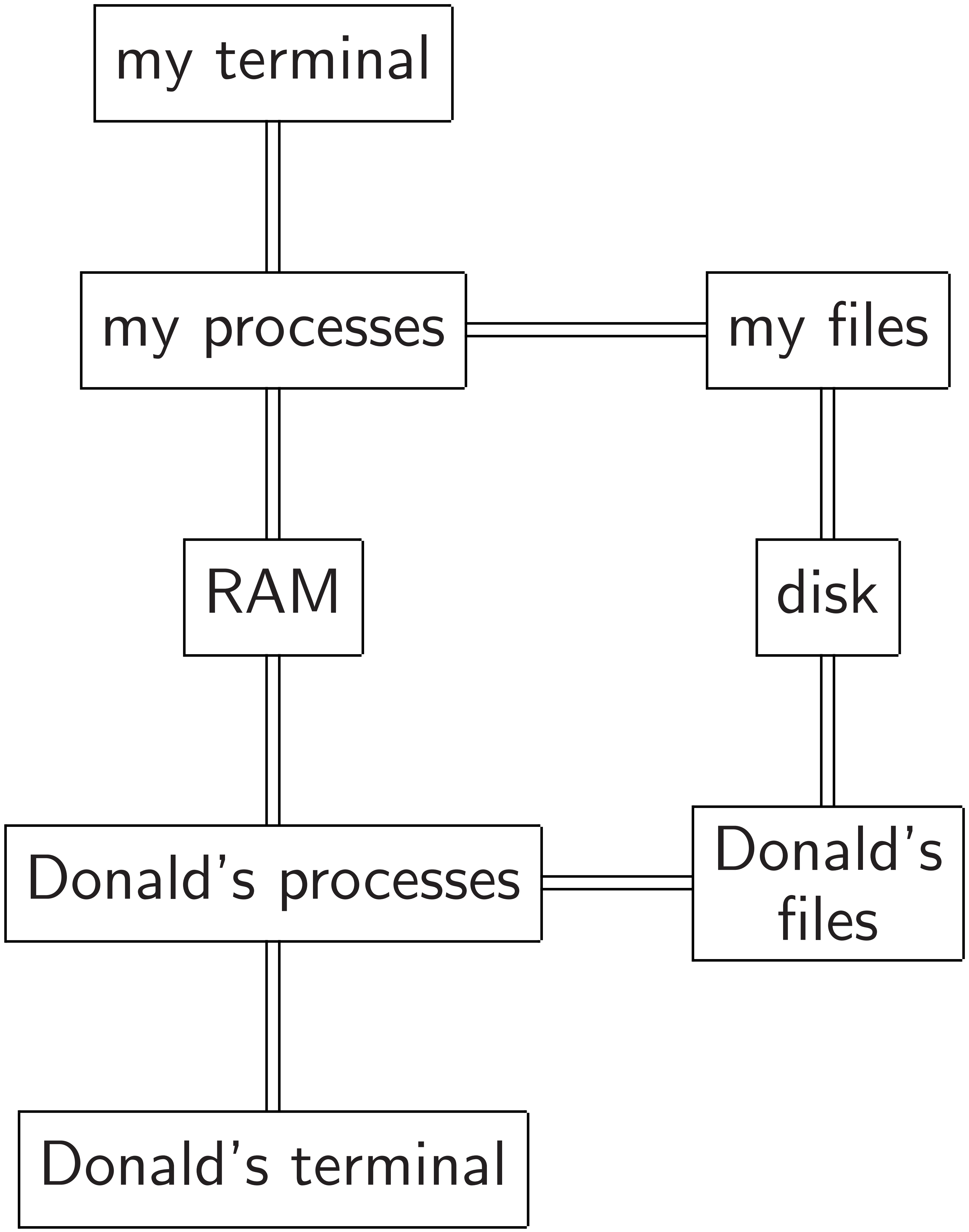
Can Donald corrupt the data appearing on my terminal?

Attack: guess my password.



Can Donald corrupt the data appearing on my terminal?

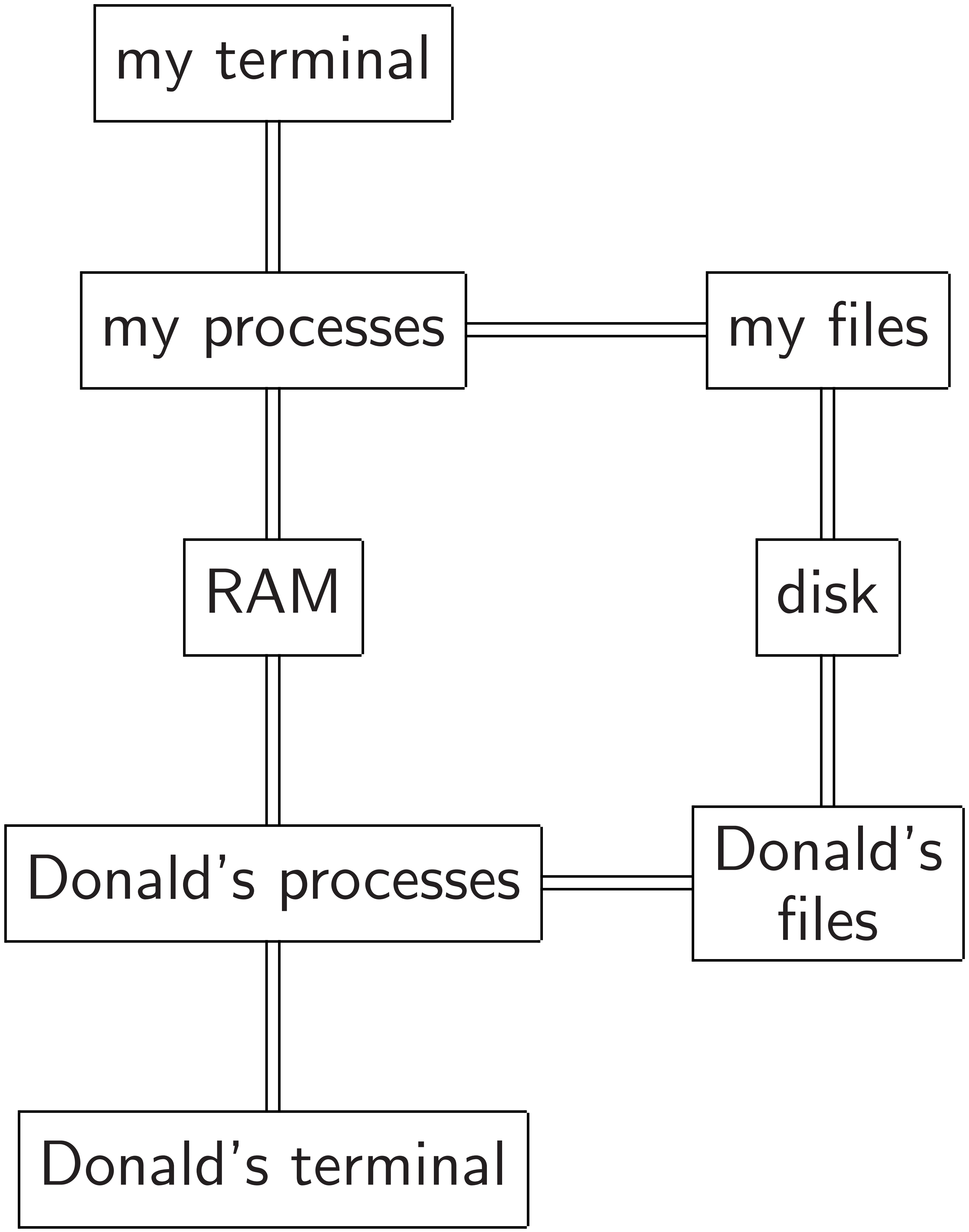
Attack: guess my password.
Defense: I have a high-entropy randomly generated password.



Can Donald corrupt the data appearing on my terminal?

Attack: guess my password.
Defense: I have a high-entropy randomly generated password.

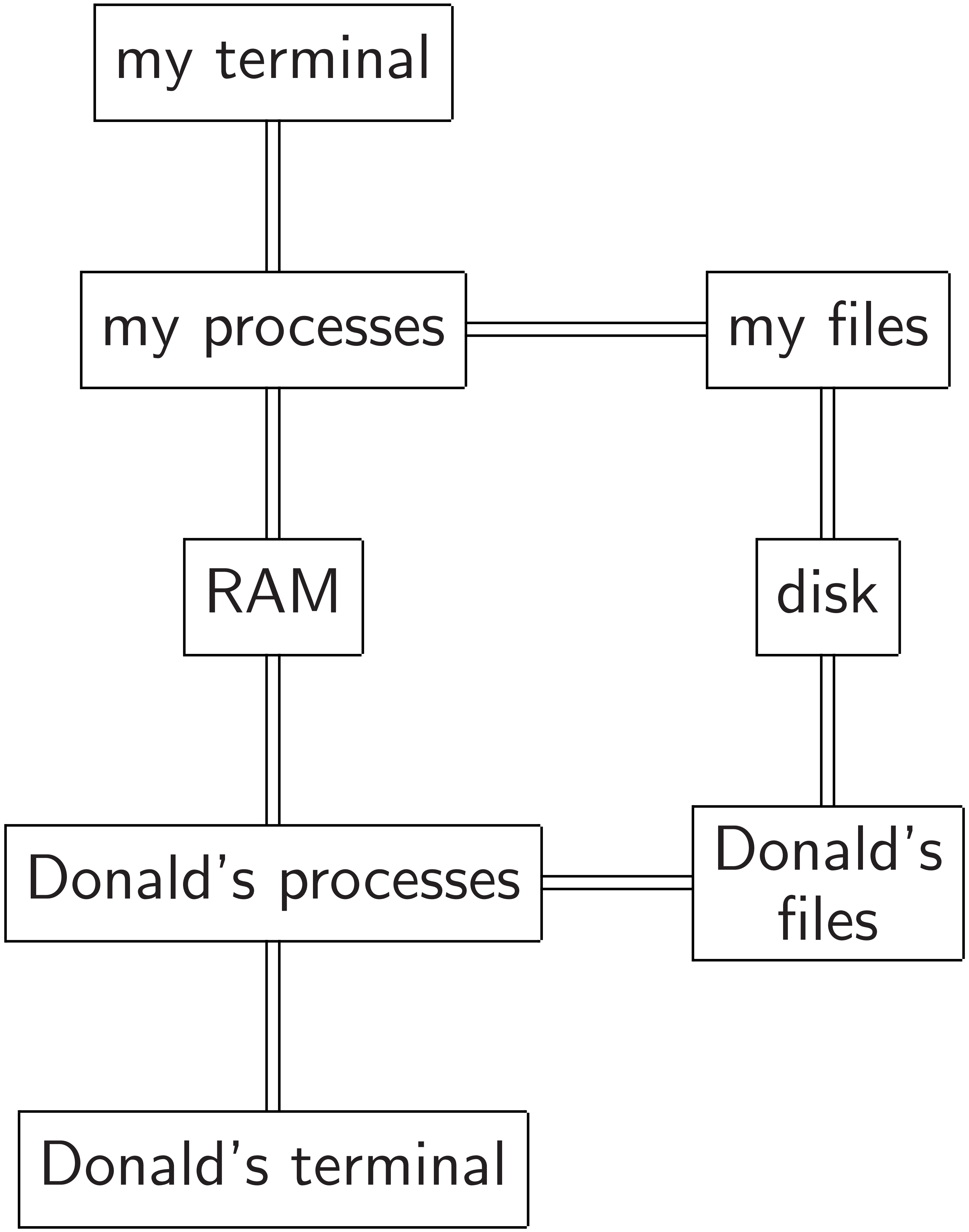
Attack: replace the terminal with a rigged terminal that intercepts my password.



Can Donald corrupt the data appearing on my terminal?

Attack: guess my password.
Defense: I have a high-entropy randomly generated password.

Attack: replace the terminal with a rigged terminal that intercepts my password.
Defense: physical security.

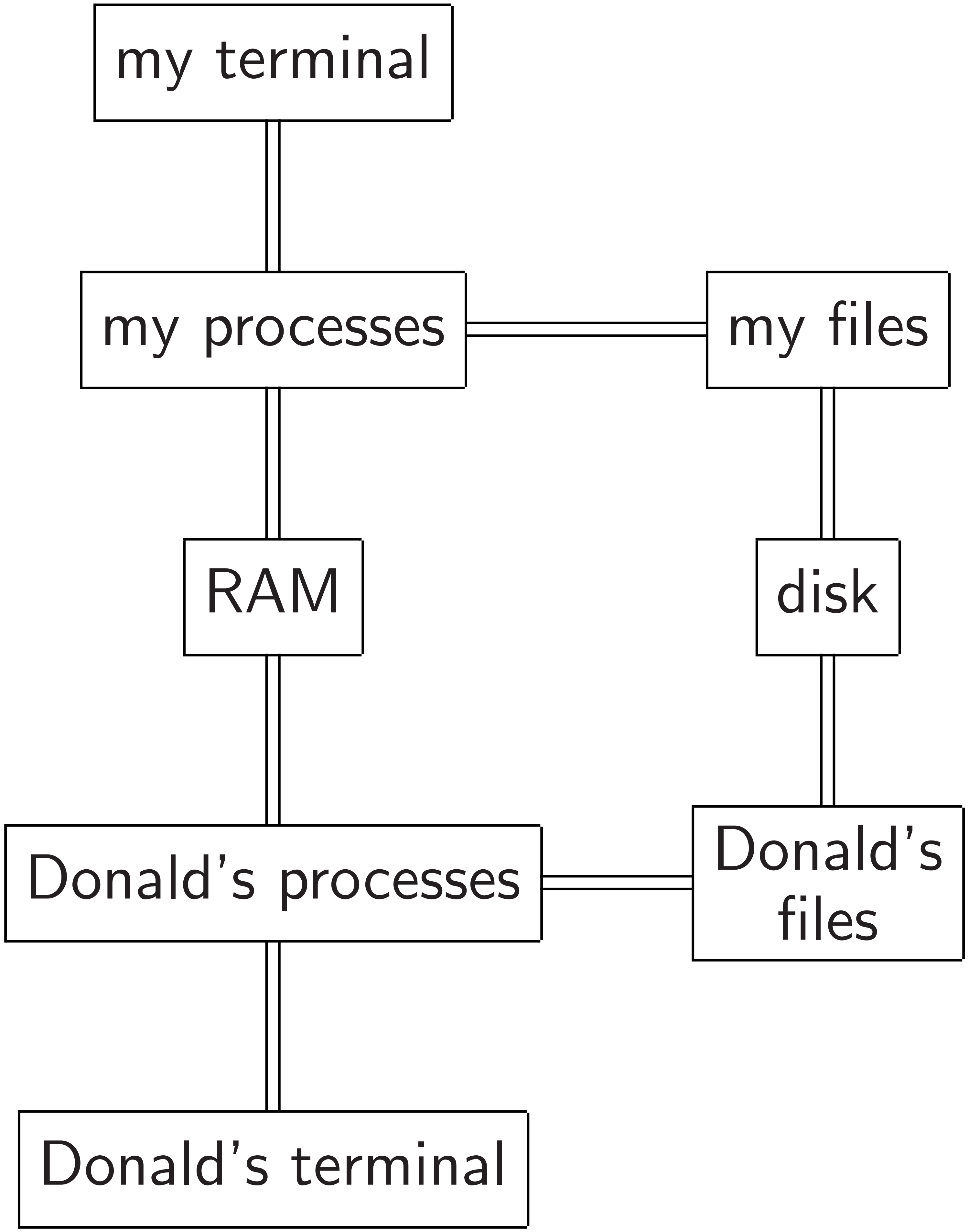


Can Donald corrupt the data appearing on my terminal?

Attack: guess my password.
Defense: I have a high-entropy randomly generated password.

Attack: replace the terminal with a rigged terminal that intercepts my password.
Defense: physical security.

Attack: use my terminal earlier and leave a program running that looks like the usual login screen but intercepts my password.

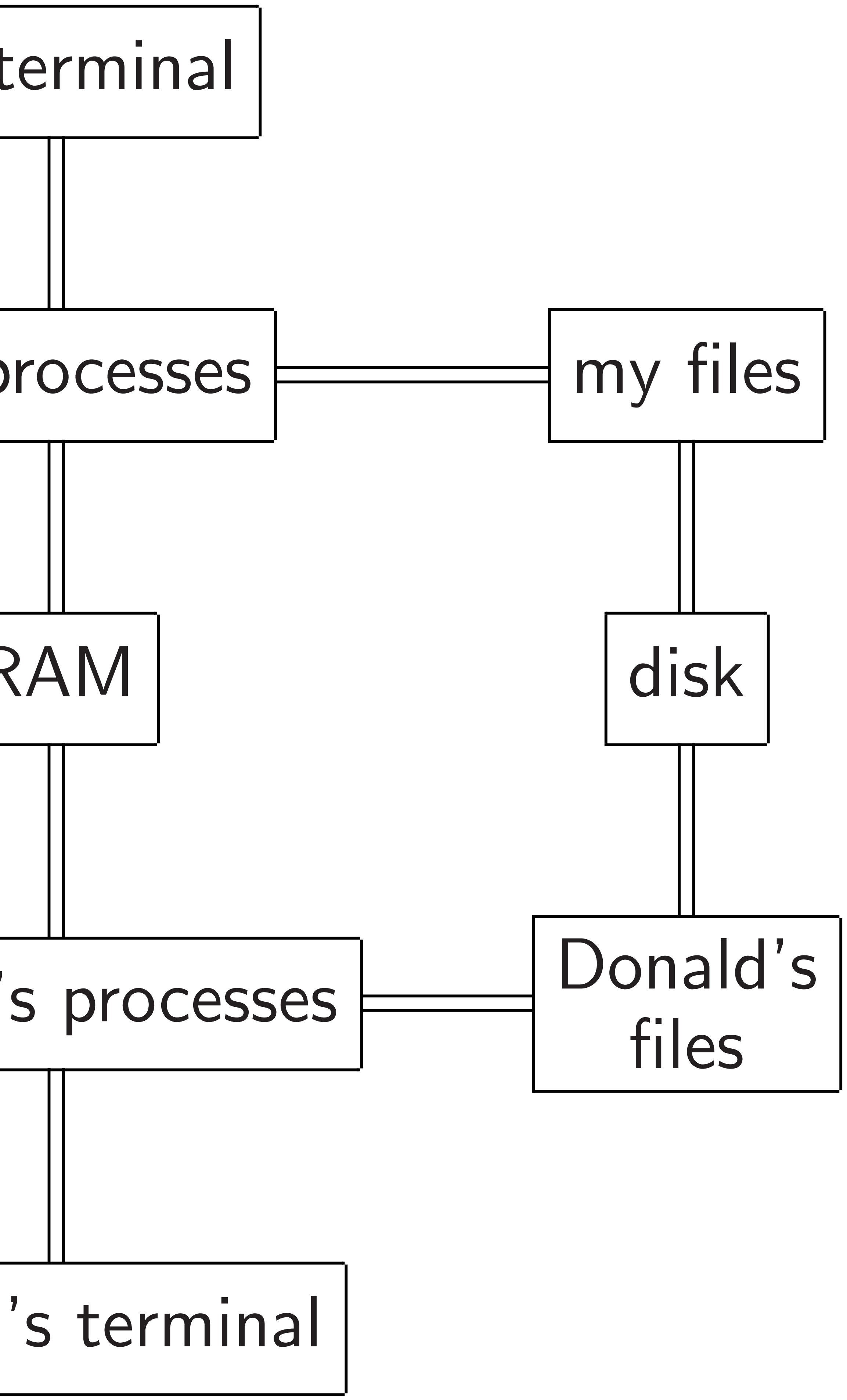


Can Donald corrupt the data appearing on my terminal?

Attack: guess my password.
Defense: I have a high-entropy randomly generated password.

Attack: replace the terminal with a rigged terminal that intercepts my password.
Defense: physical security.

Attack: use my terminal earlier and leave a program running that looks like the usual login screen but intercepts my password.
Defense: secure attention key.



Can Donald corrupt the data appearing on my terminal?

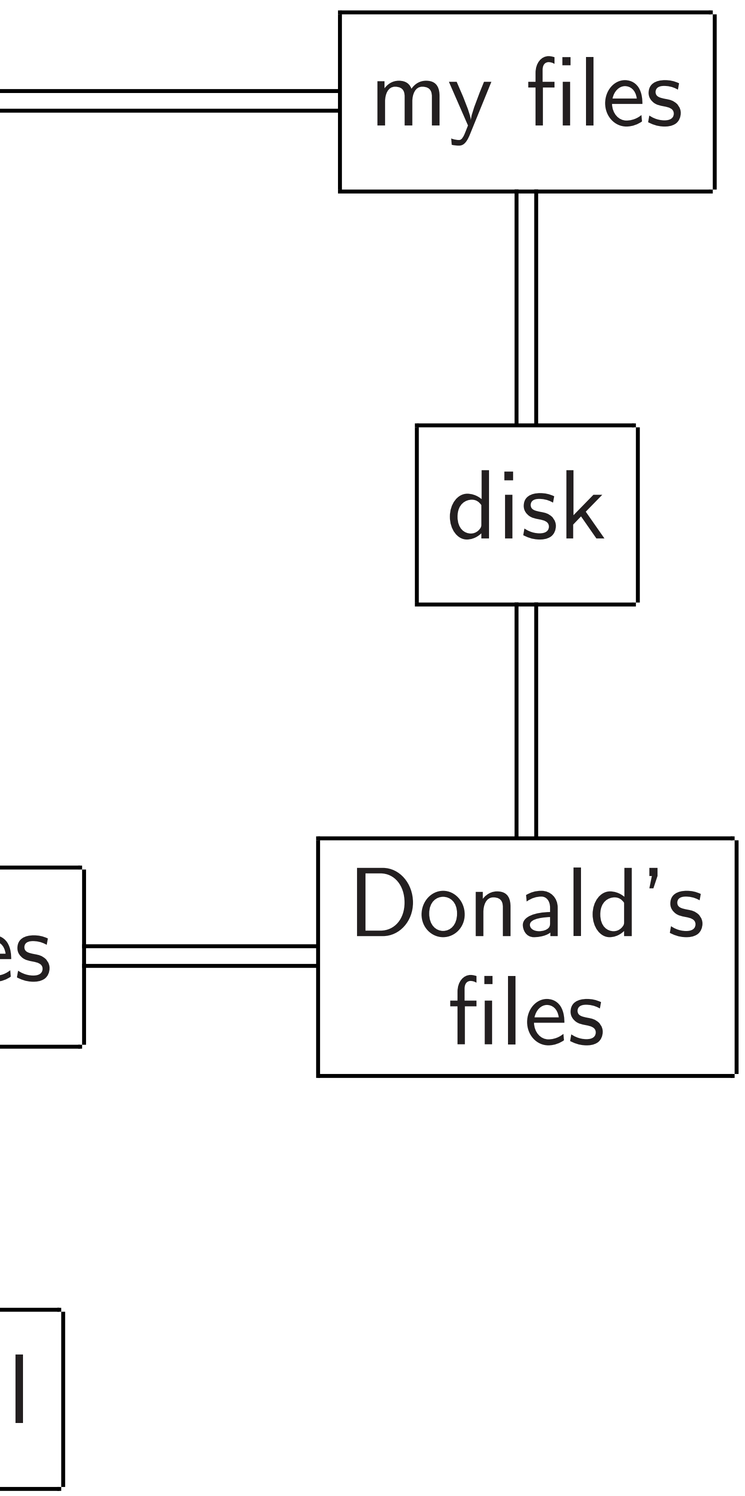
Attack: guess my password.
Defense: I have a high-entropy randomly generated password.

Attack: replace the terminal with a rigged terminal that intercepts my password.
Defense: physical security.

Attack: use my terminal earlier and leave a program running that looks like the usual login screen but intercepts my password.
Defense: secure attention key.

Donald i
data on

Attack:
part of F



Can Donald corrupt the data appearing on my terminal?

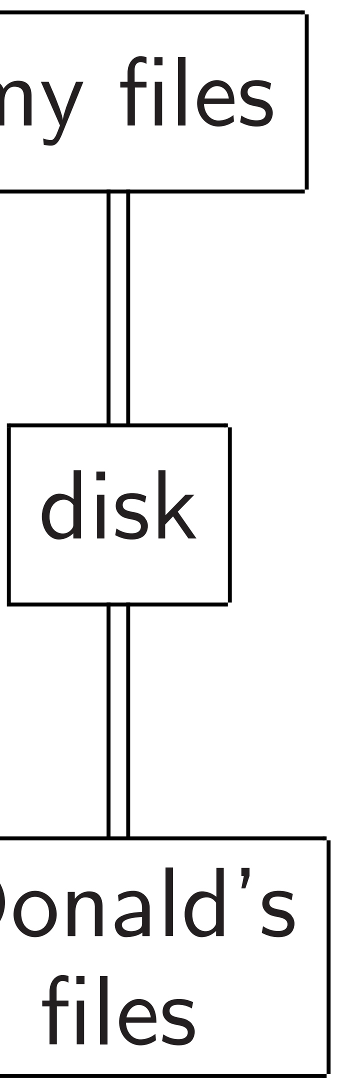
Attack: guess my password.
Defense: I have a high-entropy randomly generated password.

Attack: replace the terminal with a rigged terminal that intercepts my password.
Defense: physical security.

Attack: use my terminal earlier and leave a program running that looks like the usual login screen but intercepts my password.
Defense: secure attention key.

Donald is authoriz
data on the same

Attack: Donald st
part of RAM, or m



Can Donald corrupt the data appearing on my terminal?

Attack: guess my password.
Defense: I have a high-entropy randomly generated password.

Attack: replace the terminal with a rigged terminal that intercepts my password.
Defense: physical security.

Attack: use my terminal earlier and leave a program running that looks like the usual login screen but intercepts my password.
Defense: secure attention key.

Donald is authorized to stor
data on the same computer.

Attack: Donald stores data
part of RAM, or my part of

Can Donald corrupt the data appearing on my terminal?

Attack: guess my password.
Defense: I have a high-entropy randomly generated password.

Attack: replace the terminal with a rigged terminal that intercepts my password.
Defense: physical security.

Attack: use my terminal earlier and leave a program running that looks like the usual login screen but intercepts my password.
Defense: secure attention key.

Donald is authorized to store data on the same computer.

Attack: Donald stores data in my part of RAM, or my part of disk.

Can Donald corrupt the data appearing on my terminal?

Attack: guess my password.
Defense: I have a high-entropy randomly generated password.

Attack: replace the terminal with a rigged terminal that intercepts my password.
Defense: physical security.

Attack: use my terminal earlier and leave a program running that looks like the usual login screen but intercepts my password.
Defense: secure attention key.

Donald is authorized to store data on the same computer.

Attack: Donald stores data in my part of RAM, or my part of disk.

Two-part defense:

1. "Memory protection". Hardware does not allow processes to access data outside areas marked by kernel.

2. Kernel keeps track of which parts of RAM and disk are mine, and which parts are Donald's.

nald corrupt the data
ng on my terminal?

guess my password.
: I have a high-entropy
y generated password.

replace the terminal
igged terminal that
ts my password.
: physical security.

use my terminal earlier
e a program running that
e the usual login screen
rcepts my password.
: secure attention key.

Donald is authorized to store data on the same computer.

Attack: Donald stores data in my part of RAM, or my part of disk.

Two-part defense:

1. “Memory protection”. Hardware does not allow processes to access data outside areas marked by kernel.

2. Kernel keeps track of which parts of RAM and disk are mine, and which parts are Donald’s.

Bugs in
can com
allowing
to my pa

pt the data
terminal?

password.
high-entropy
ed password.

e terminal
inal that
sword.
security.

rminal earlier
m running that
l login screen
password.
ttention key.

Donald is authorized to store data on the same computer.

Attack: Donald stores data in my part of RAM, or my part of disk.

Two-part defense:

1. "Memory protection". Hardware does not allow processes to access data outside areas marked by kernel.

2. Kernel keeps track of which parts of RAM and disk are mine, and which parts are Donald's.

Bugs in this kerne
can compromise se
allowing Donald to
to my part of RAM

a

opy
rd.

l

rlier
g that
reen

ey.

Donald is authorized to store data on the same computer.

Attack: Donald stores data in my part of RAM, or my part of disk.

Two-part defense:

1. "Memory protection". Hardware does not allow processes to access data outside areas marked by kernel.

2. Kernel keeps track of which parts of RAM and disk are mine, and which parts are Donald's.

Bugs in this kernel code can compromise security, allowing Donald to write to my part of RAM or disk.

Donald is authorized to store data on the same computer.

Attack: Donald stores data in my part of RAM, or my part of disk.

Two-part defense:

1. “Memory protection”. Hardware does not allow processes to access data outside areas marked by kernel.

2. Kernel keeps track of which parts of RAM and disk are mine, and which parts are Donald’s.

Bugs in this kernel code can compromise security, allowing Donald to write to my part of RAM or disk.

Donald is authorized to store data on the same computer.

Attack: Donald stores data in my part of RAM, or my part of disk.

Two-part defense:

1. "Memory protection". Hardware does not allow processes to access data outside areas marked by kernel.

2. Kernel keeps track of which parts of RAM and disk are mine, and which parts are Donald's.

Bugs in this kernel code can compromise security, allowing Donald to write to my part of RAM or disk.

Fix: Eliminate the bugs!

Bug-free code is expensive but not impossible when code volume is small enough.

Successful example: computer-verified proof of seL4 microkernel correctness, including RAM partitioning etc.

is authorized to store
the same computer.

Donald stores data in my
RAM, or my part of disk.

rt defense:

mory protection".
re does not allow
s to access data
areas marked by kernel.

el keeps track of which
RAM and disk are mine,
ch parts are Donald's.

Bugs in this kernel code
can compromise security,
allowing Donald to write
to my part of RAM or disk.

Fix: Eliminate the bugs!

Bug-free code is expensive
but not impossible when
code volume is small enough.

Successful example:
computer-verified proof of
seL4 microkernel correctness,
including RAM partitioning etc.

If a sma
has cut
commun

I can rur
program
and still
Donald i
the outp

ed to store
computer.

ores data in my
ny part of disk.

ction".
t allow
s data
ked by kernel.

ack of which
disk are mine,
re Donald's.

Bugs in this kernel code
can compromise security,
allowing Donald to write
to my part of RAM or disk.

Fix: Eliminate the bugs!

Bug-free code is expensive
but not impossible when
code volume is small enough.

Successful example:
computer-verified proof of
seL4 microkernel correctness,
including RAM partitioning etc.

If a small bug-free
has cut off Donald
communication wi

I can run a 100000
program filled with
and still be confide
Donald is unable t
the output of the

e

in my
disk.

nel.

ich
mine,
s.

Bugs in this kernel code
can compromise security,
allowing Donald to write
to my part of RAM or disk.

Fix: Eliminate the bugs!

Bug-free code is expensive
but not impossible when
code volume is small enough.

Successful example:
computer-verified proof of
seL4 microkernel correctness,
including RAM partitioning etc.

If a small bug-free kernel
has cut off Donald's
communication with me:

I can run a 10000000-line
program filled with bugs,
and still be confident that
Donald is unable to corrupt
the output of the program.

Bugs in this kernel code
can compromise security,
allowing Donald to write
to my part of RAM or disk.

Fix: Eliminate the bugs!

Bug-free code is expensive
but not impossible when
code volume is small enough.

Successful example:
computer-verified proof of
seL4 microkernel correctness,
including RAM partitioning etc.

If a small bug-free kernel
has cut off Donald's
communication with me:

I can run a 10000000-line
program filled with bugs,
and still be confident that
Donald is unable to corrupt
the output of the program.

Bugs in this kernel code
can compromise security,
allowing Donald to write
to my part of RAM or disk.

Fix: Eliminate the bugs!

Bug-free code is expensive
but not impossible when
code volume is small enough.

Successful example:
computer-verified proof of
seL4 microkernel correctness,
including RAM partitioning etc.

If a small bug-free kernel
has cut off Donald's
communication with me:

I can run a 10000000-line
program filled with bugs,
and still be confident that
Donald is unable to corrupt
the output of the program.

The **trusted computing base**
(TCB) is the part of the system
that enforces security policy.
The 10000000-line program
is not part of the TCB.

this kernel code
promise security,
Donald to write
art of RAM or disk.

minate the bugs!

e code is expensive
impossible when
ume is small enough.

ful example:
er-verified proof of
crokernel correctness,
g RAM partitioning etc.

If a small bug-free kernel has cut off Donald's communication with me:

I can run a 10000000-line program filled with bugs, and still be confident that Donald is unable to corrupt the output of the program.

The **trusted computing base** (TCB) is the part of the system that enforces security policy. The 10000000-line program is not part of the TCB.

But we

Today:
I downlo
These us
to put d

l code
ecurity,
o write
M or disk.

bugs!

xpensive
e when
nall enough.

e:
proof of
correctness,
rtitioning etc.

If a small bug-free kernel has cut off Donald's communication with me:

I can run a 10000000-line program filled with bugs, and still be confident that Donald is unable to corrupt the output of the program.

The **trusted computing base** (TCB) is the part of the system that enforces security policy. The 10000000-line program is not part of the TCB.

But we *want* comm

Today: Alice send
I download Bob's
These users are au
to put data on my

n.

s,
etc.

If a small bug-free kernel has cut off Donald's communication with me:

I can run a 10000000-line program filled with bugs, and still be confident that Donald is unable to corrupt the output of the program.

The **trusted computing base** (TCB) is the part of the system that enforces security policy. The 10000000-line program is not part of the TCB.

But we *want* communicatio

Today: Alice sends me emai
I download Bob's web page.
These users are authorized to put data on my screen.

If a small bug-free kernel has cut off Donald's communication with me:

I can run a 10000000-line program filled with bugs, and still be confident that Donald is unable to corrupt the output of the program.

The **trusted computing base** (TCB) is the part of the system that enforces security policy. The 10000000-line program is not part of the TCB.

But we *want* communication!

Today: Alice sends me email. I download Bob's web page. These users are authorized to put data on my screen.

If a small bug-free kernel has cut off Donald's communication with me:

I can run a 10000000-line program filled with bugs, and still be confident that Donald is unable to corrupt the output of the program.

The **trusted computing base** (TCB) is the part of the system that enforces security policy. The 10000000-line program is not part of the TCB.

But we *want* communication!

Today: Alice sends me email. I download Bob's web page. These users are authorized to put data on my screen.

Security policy: Whenever the computer shows me a file, it also tells me the source of the file.

If a small bug-free kernel has cut off Donald's communication with me:

I can run a 10000000-line program filled with bugs, and still be confident that Donald is unable to corrupt the output of the program.

The **trusted computing base** (TCB) is the part of the system that enforces security policy. The 10000000-line program is not part of the TCB.

But we *want* communication!

Today: Alice sends me email. I download Bob's web page. These users are authorized to put data on my screen.

Security policy: Whenever the computer shows me a file, it also tells me the source of the file.

If Donald creates a file and convinces the computer to show me the file as having source "Alice" then this policy is violated.

ll bug-free kernel
off Donald's
ication with me:

n a 10000000-line
filled with bugs,
be confident that
is unable to corrupt
ut of the program.

sted computing base
s the part of the system
orces security policy.
00000-line program
art of the TCB.

But we *want* communication!

Today: Alice sends me email.
I download Bob's web page.
These users are authorized
to put data on my screen.

Security policy: Whenever the
computer shows me a file, it also
tells me the source of the file.

If Donald creates a file
and convinces the computer
to show me the file
as having source "Alice"
then this policy is violated.

kernel
l's
th me:

000-line
n bugs,
ent that
to corrupt
program.

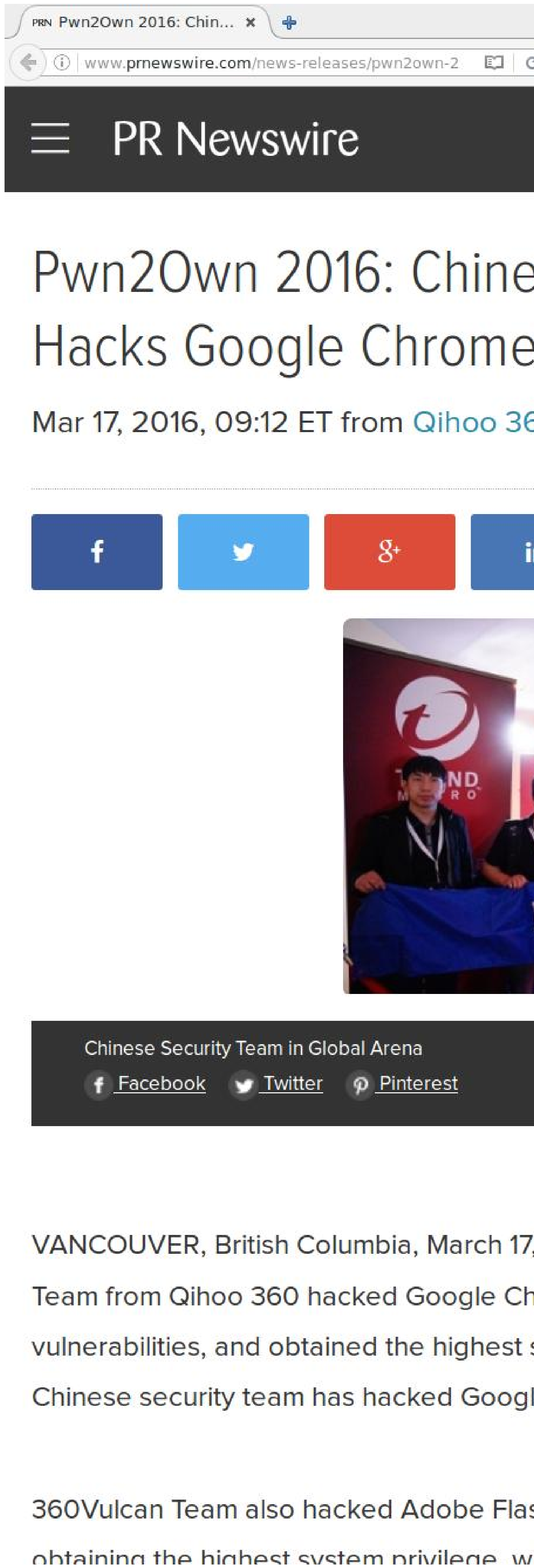
puting base
of the system
rity policy.
e program
TCB.

But we *want* communication!

Today: Alice sends me email.
I download Bob's web page.
These users are authorized
to put data on my screen.

Security policy: Whenever the
computer shows me a file, it also
tells me the source of the file.

If Donald creates a file
and convinces the computer
to show me the file
as having source "Alice"
then this policy is violated.



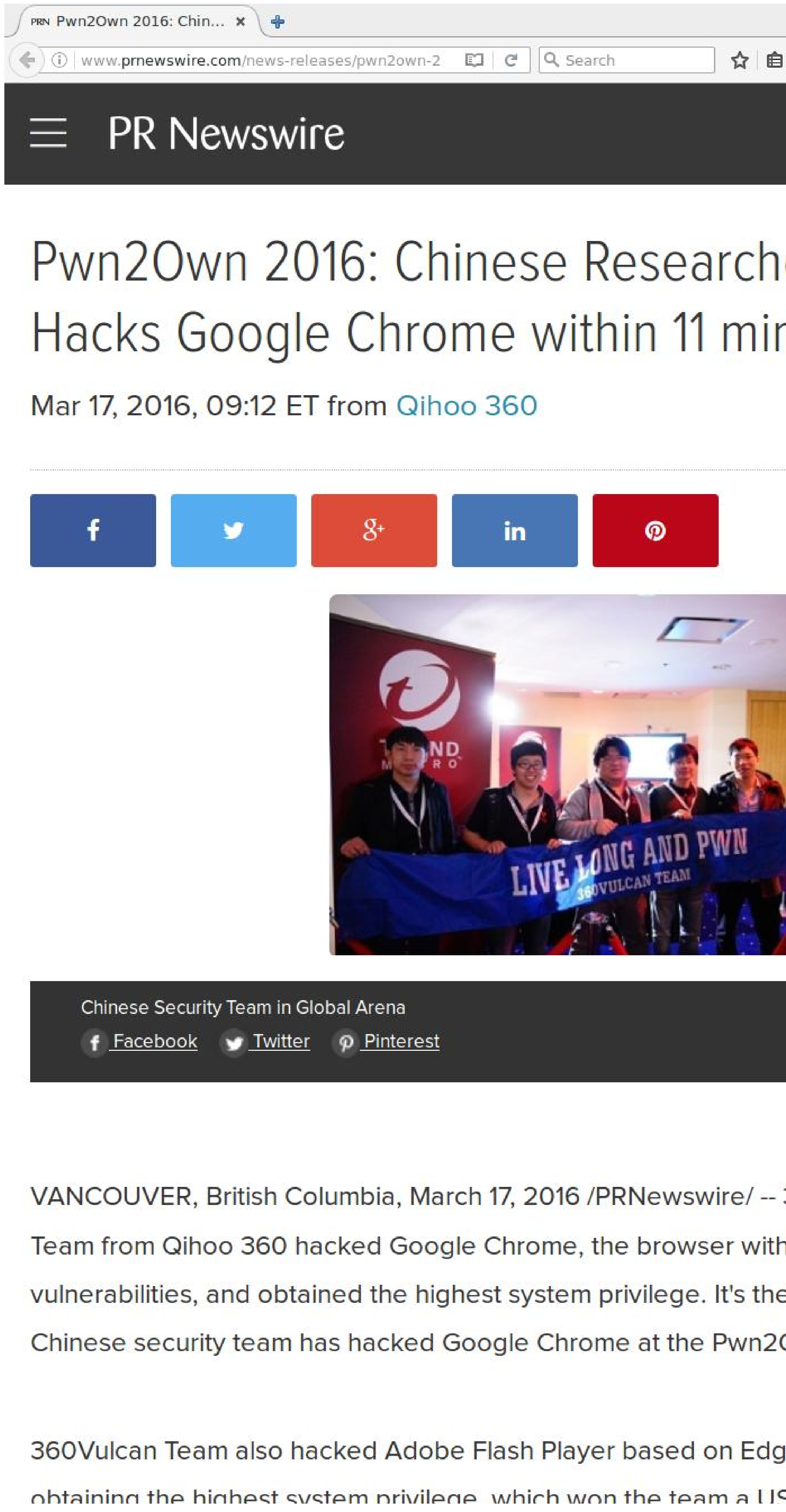
se
tem

But we *want* communication!

Today: Alice sends me email.
I download Bob's web page.
These users are authorized
to put data on my screen.

Security policy: Whenever the computer shows me a file, it also tells me the source of the file.

If Donald creates a file
and convinces the computer
to show me the file
as having source "Alice"
then this policy is violated.


Pwn2Own 2016: Chin...
www.prnewswire.com/news-releases/pwn2own-2
Search
PR Newswire

Pwn2Own 2016: Chinese Research
Hacks Google Chrome within 11 mir

Mar 17, 2016, 09:12 ET from Qihoo 360

Chinese Security Team in Global Arena
Facebook Twitter Pinterest

VANCOUVER, British Columbia, March 17, 2016 /PRNewswire/ --
Team from Qihoo 360 hacked Google Chrome, the browser with
vulnerabilities, and obtained the highest system privilege. It's the
Chinese security team has hacked Google Chrome at the Pwn2

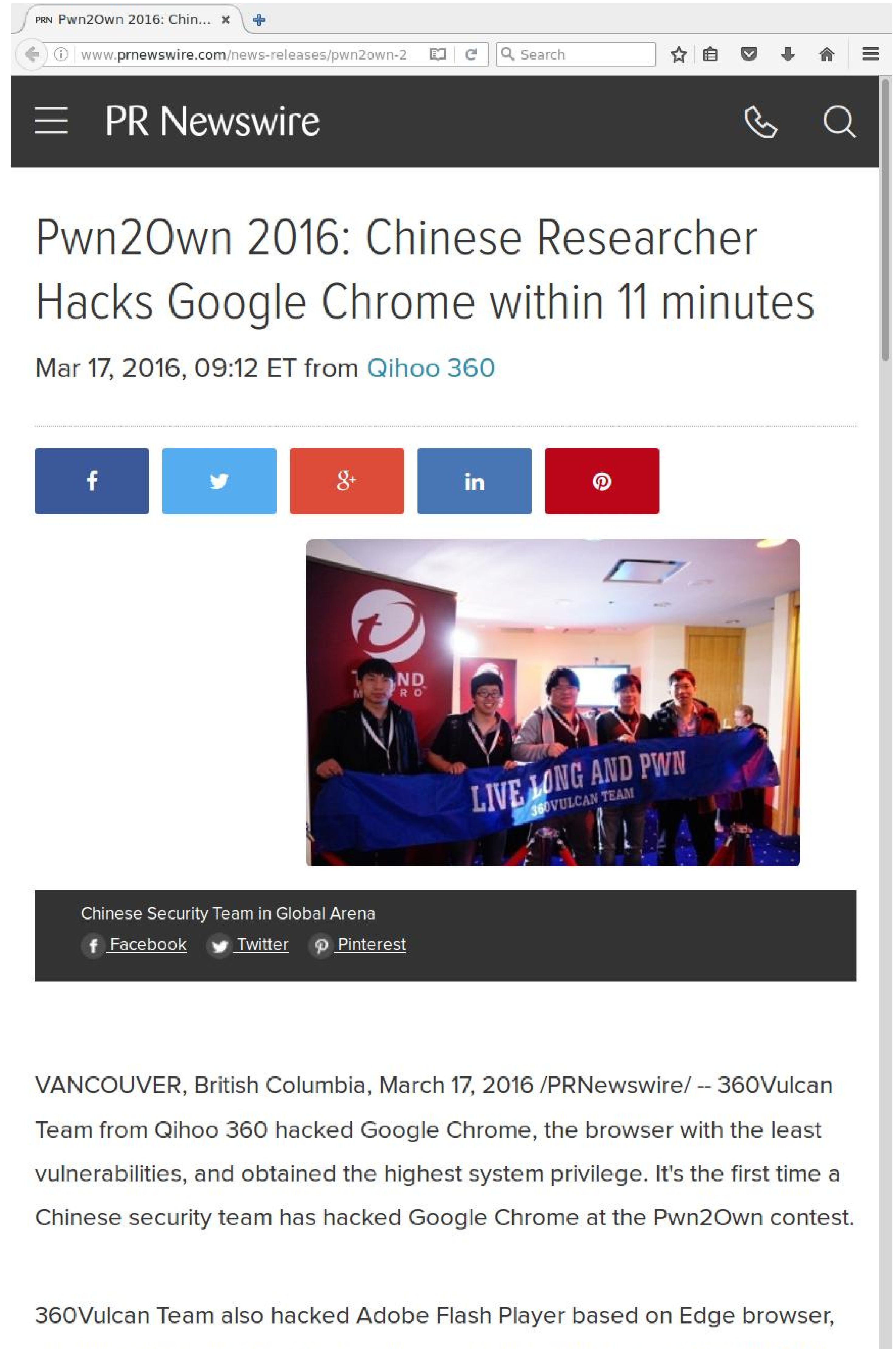
360Vulcan Team also hacked Adobe Flash Player based on Edg
obtaining the highest system privilege, which won the team a US

But we *want* communication!

Today: Alice sends me email.
I download Bob's web page.
These users are authorized
to put data on my screen.

Security policy: Whenever the computer shows me a file, it also tells me the source of the file.

If Donald creates a file
and convinces the computer
to show me the file
as having source "Alice"
then this policy is violated.



Pwn2Own 2016: Chin...

www.prnewswire.com/news-releases/pwn2own-2

PR Newswire

Pwn2Own 2016: Chinese Researcher Hacks Google Chrome within 11 minutes

Mar 17, 2016, 09:12 ET from Qihoo 360

Chinese Security Team in Global Arena

Facebook Twitter Pinterest

VANCOUVER, British Columbia, March 17, 2016 /PRNewswire/ -- 360Vulcan Team from Qihoo 360 hacked Google Chrome, the browser with the least vulnerabilities, and obtained the highest system privilege. It's the first time a Chinese security team has hacked Google Chrome at the Pwn2Own contest.

360Vulcan Team also hacked Adobe Flash Player based on Edge browser, obtaining the highest system privilege, which won the team a USD 80,000 *want* communication!

Alice sends me email.
oad Bob's web page.
sers are authorized
ata on my screen.

policy: Whenever the
er shows me a file, it also
the source of the file.

d creates a file
vinces the computer
me the file
g source "Alice"
s policy is violated.

Pwn2Own 2016: Chin... ×
www.prnewswire.com/news-releases/pwn2own-2

PR Newswire

Pwn2Own 2016: Chinese Researcher Hacks Google Chrome within 11 minutes

Mar 17, 2016, 09:12 ET from Qihoo 360

Chinese Security Team in Global Arena
Facebook Twitter Pinterest

VANCOUVER, British Columbia, March 17, 2016 /PRNewswire/ -- 360Vulcan Team from Qihoo 360 hacked Google Chrome, the browser with the least vulnerabilities, and obtained the highest system privilege. It's the first time a Chinese security team has hacked Google Chrome at the Pwn2Own contest.

360Vulcan Team also hacked Adobe Flash Player based on Edge browser, obtaining the highest system privilege, which won the team a USD 80,000

Which p
enforces

munication!

s me email.

web page.

uthorized

screen.

/henever the

ne a file, it also

e of the file.

a file

computer

e

'Alice"

violated.

Pwn2Own 2016: Chin... ×

www.prnewswire.com/news-releases/pwn2own-2

Search

PR Newswire

Pwn2Own 2016: Chinese Researcher Hacks Google Chrome within 11 minutes

Mar 17, 2016, 09:12 ET from Qihoo 360

Chinese Security Team in Global Arena
Facebook Twitter Pinterest

VANCOUVER, British Columbia, March 17, 2016 /PRNewswire/ -- 360Vulcan Team from Qihoo 360 hacked Google Chrome, the browser with the least vulnerabilities, and obtained the highest system privilege. It's the first time a Chinese security team has hacked Google Chrome at the Pwn2Own contest.

360Vulcan Team also hacked Adobe Flash Player based on Edge browser, obtaining the highest system privilege, which won the team a USD 80,000

Which part of the

enforces the securi

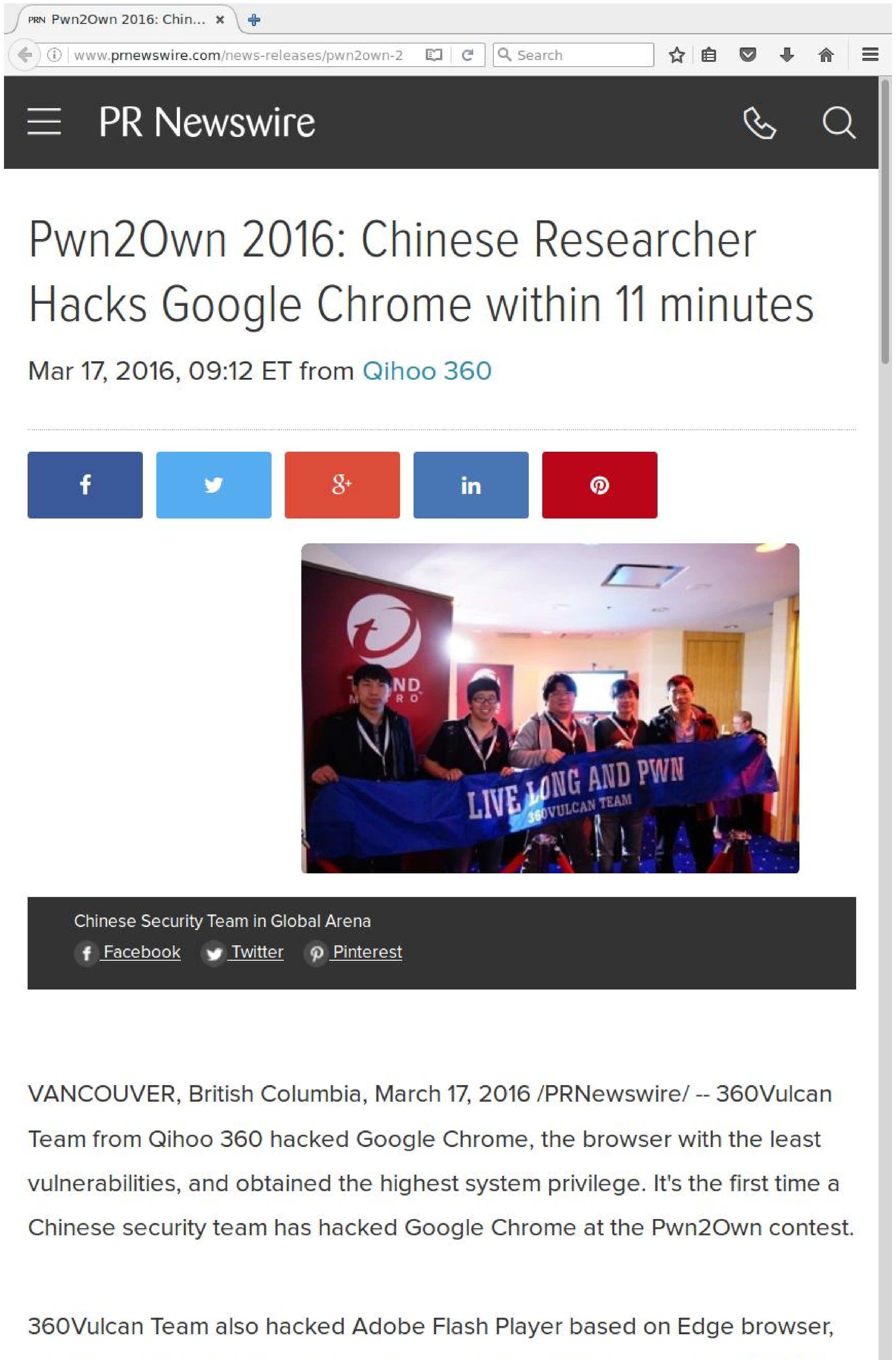
n!

l.

he

also

e.



PRN Pwn2Own 2016: Chin... ×

www.prnewswire.com/news-releases/pwn2own-2

PR Newswire

Pwn2Own 2016: Chinese Researcher Hacks Google Chrome within 11 minutes

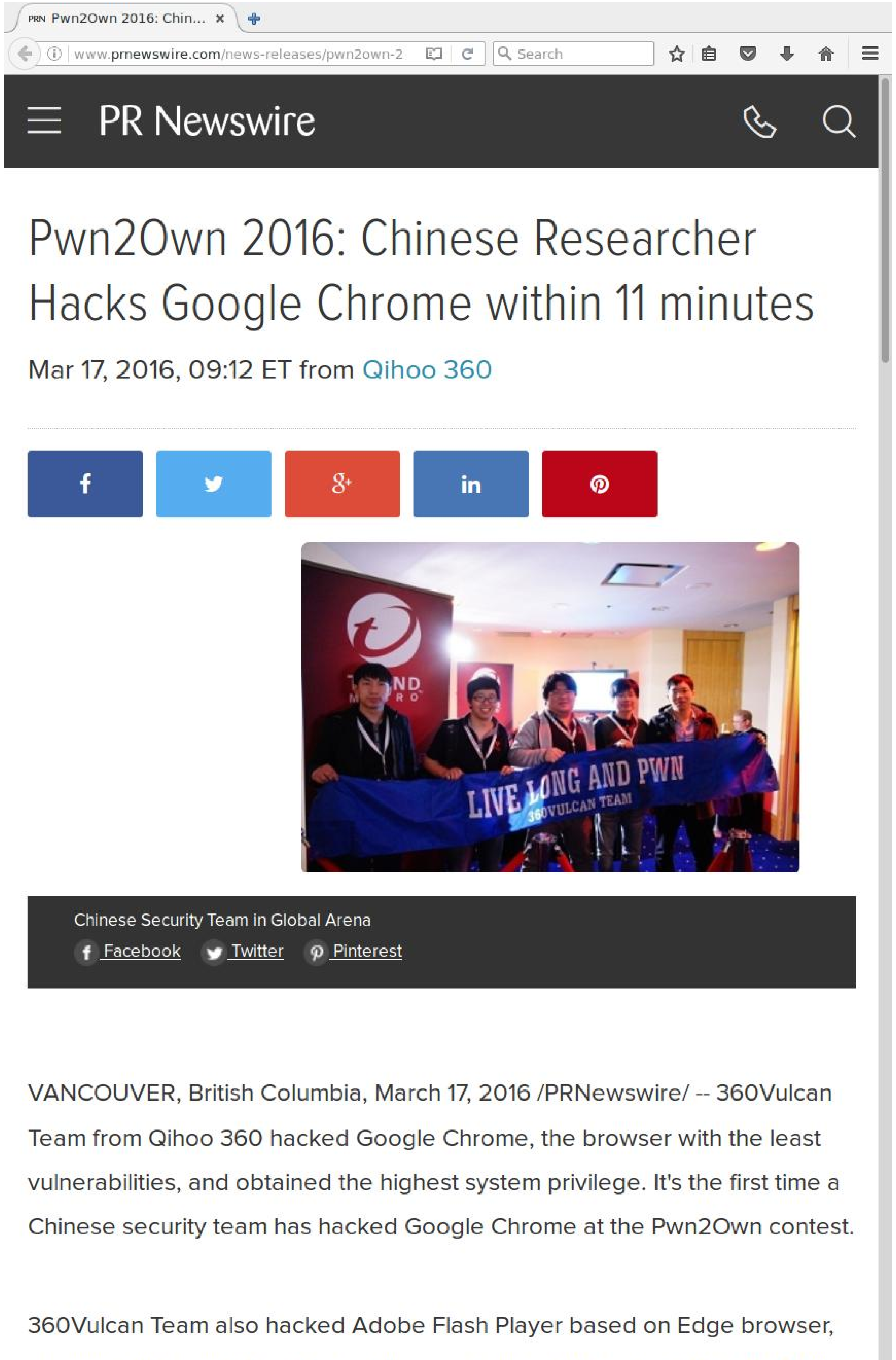
Mar 17, 2016, 09:12 ET from Qihoo 360

Chinese Security Team in Global Arena
Facebook Twitter Pinterest

VANCOUVER, British Columbia, March 17, 2016 /PRNewswire/ -- 360Vulcan Team from Qihoo 360 hacked Google Chrome, the browser with the least vulnerabilities, and obtained the highest system privilege. It's the first time a Chinese security team has hacked Google Chrome at the Pwn2Own contest.

360Vulcan Team also hacked Adobe Flash Player based on Edge browser, obtaining the highest system privilege, which won the team a USD 80,000

Which part of the system enforces the security policy?


Pwn2Own 2016: Chin... ×

www.prnewswire.com/news-releases/pwn2own-2

PR Newswire

Pwn2Own 2016: Chinese Researcher Hacks Google Chrome within 11 minutes

Mar 17, 2016, 09:12 ET from Qihoo 360

Chinese Security Team in Global Arena
Facebook Twitter Pinterest

VANCOUVER, British Columbia, March 17, 2016 /PRNewswire/ -- 360Vulcan Team from Qihoo 360 hacked Google Chrome, the browser with the least vulnerabilities, and obtained the highest system privilege. It's the first time a Chinese security team has hacked Google Chrome at the Pwn2Own contest.

360Vulcan Team also hacked Adobe Flash Player based on Edge browser, obtaining the highest system privilege, which won the team a USD 80,000

Which part of the system enforces the security policy?

PRN Pwn2Own 2016: Chin... ×

www.prnewswire.com/news-releases/pwn2own-2 Search

PR Newswire

Pwn2Own 2016: Chinese Researcher Hacks Google Chrome within 11 minutes

Mar 17, 2016, 09:12 ET from Qihoo 360

Chinese Security Team in Global Arena
Facebook Twitter Pinterest

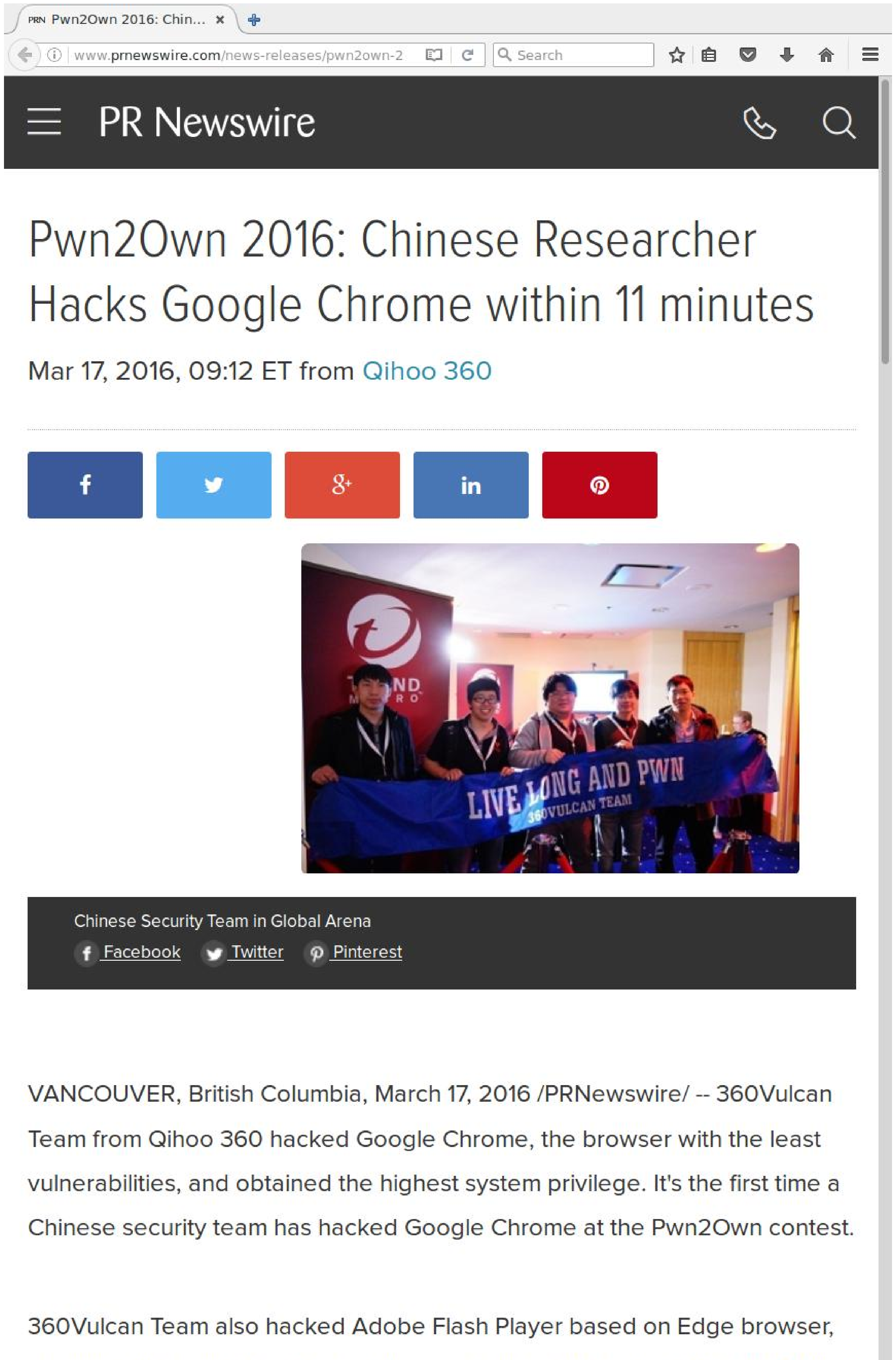
VANCOUVER, British Columbia, March 17, 2016 /PRNewswire/ -- 360Vulcan Team from Qihoo 360 hacked Google Chrome, the browser with the least vulnerabilities, and obtained the highest system privilege. It's the first time a Chinese security team has hacked Google Chrome at the Pwn2Own contest.

360Vulcan Team also hacked Adobe Flash Player based on Edge browser, obtaining the highest system privilege, which won the team a USD 80,000

Which part of the system enforces the security policy?

Widely deployed software systems make no real efforts to limit this.

There is some “security” code inside kernel and browser.
But bugs in other code
can and do compromise security.
TCB has >30000000 lines.

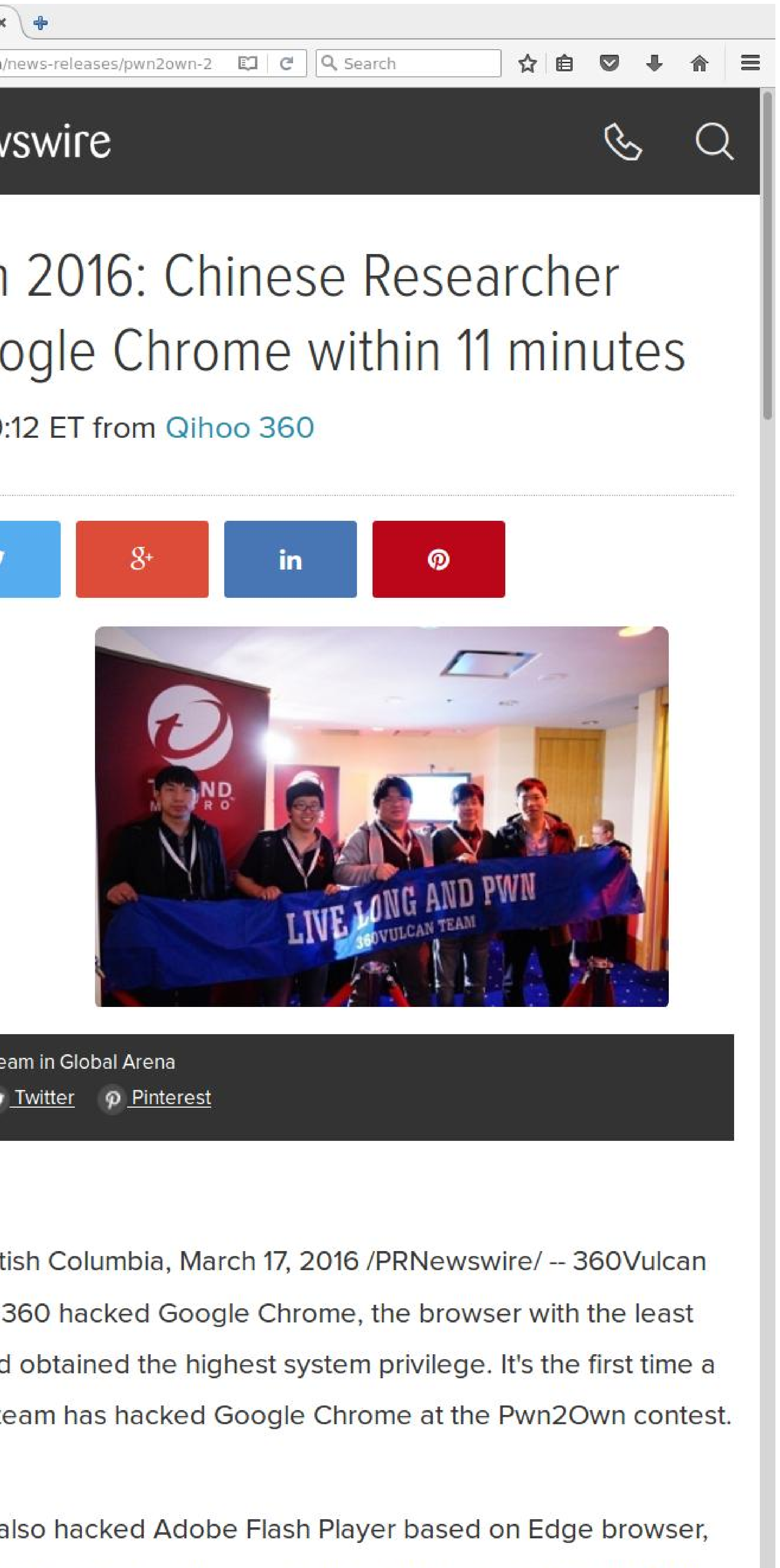


Which part of the system enforces the security policy?

Widely deployed software systems make no real efforts to limit this.

There is some "security" code inside kernel and browser.
But bugs in other code can and do compromise security.
TCB has >30000000 lines.

Fix: rearchitect entire system so that a **small** TCB tracks sources of all data.
Eliminate all bugs in TCB.


/news-releases/pwn2own-2
Search
vswire
n 2016: Chinese Researcher
ogle Chrome within 11 minutes
:12 ET from Qihoo 360
LIVE LONG AND PWN
360VULCAN TEAM
eam in Global Arena
Twitter Pinterest
tish Columbia, March 17, 2016 /PRNewswire/ -- 360Vulcan
360 hacked Google Chrome, the browser with the least
d obtained the highest system privilege. It's the first time a
eam has hacked Google Chrome at the Pwn2Own contest.
also hacked Adobe Flash Player based on Edge browser,
est system privilege, which won the team a USD 80,000

Which part of the system
enforces the security policy?

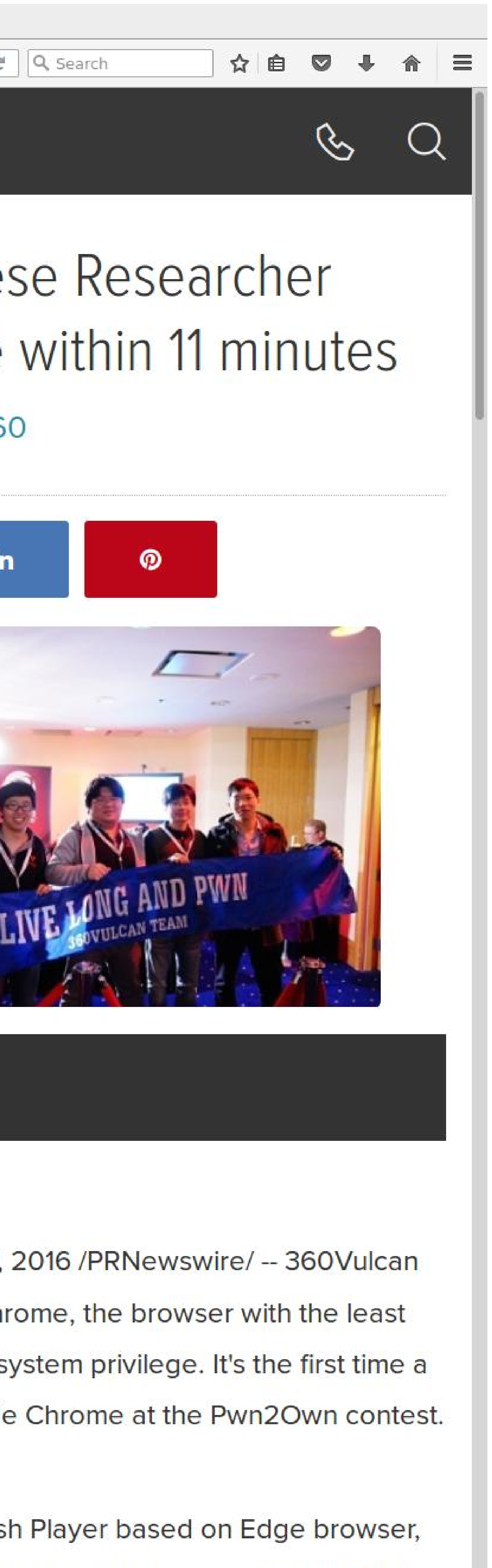
Widely deployed software systems
make no real efforts to limit this.

There is some "security" code
inside kernel and browser.
But bugs in other code
can and do compromise security.
TCB has >30000000 lines.

Fix: rearchitect entire system
so that a **small** TCB
tracks sources of all data.
Eliminate all bugs in TCB.

Cryptogr

What ha
through



Which part of the system enforces the security policy?

Widely deployed software systems make no real efforts to limit this.

There is some "security" code inside kernel and browser.
But bugs in other code can and do compromise security.
TCB has >30000000 lines.

Fix: rearchitect entire system so that a **small** TCB tracks sources of all data.
Eliminate all bugs in TCB.

Cryptography in th

What happens if c
through Donald's

er
nutes

360Vulcan
the least
first time a
Own contest.

e browser,
SD 80,000

Which part of the system enforces the security policy?

Widely deployed software systems make no real efforts to limit this.

There is some “security” code inside kernel and browser.
But bugs in other code can and do compromise security.
TCB has >30000000 lines.

Fix: rearchitect entire system so that a **small** TCB tracks sources of all data.
Eliminate all bugs in TCB.

Cryptography in the TCB

What happens if data is sen
through Donald’s network?

Which part of the system enforces the security policy?

Widely deployed software systems make no real efforts to limit this.

There is some "security" code inside kernel and browser.
But bugs in other code can and do compromise security.
TCB has >30000000 lines.

Fix: rearchitect entire system so that a **small** TCB tracks sources of all data.
Eliminate all bugs in TCB.

Cryptography in the TCB

What happens if data is sent through Donald's network?

Which part of the system enforces the security policy?

Widely deployed software systems make no real efforts to limit this.

There is some "security" code inside kernel and browser.
But bugs in other code can and do compromise security.
TCB has >30000000 lines.

Fix: rearchitect entire system so that a **small** TCB tracks sources of all data.
Eliminate all bugs in TCB.

Cryptography in the TCB

What happens if data is sent through Donald's network?

Solution: Sender and receiver scramble communication in a way that Donald cannot understand and cannot silently corrupt.

art of the system
the security policy?

deployed software systems
real efforts to limit this.

some "security" code
ernel and browser.
s in other code
do compromise security.
s >30000000 lines.

rchitect entire system
a **small** TCB
ources of all data.
e all bugs in TCB.

Cryptography in the TCB

What happens if data is sent through Donald's network?

Solution: Sender and receiver scramble communication in a way that Donald cannot understand and cannot silently corrupt.

OpenSS
500000
are man

All of th
Many de

Why is

system
ity policy?

oftware systems
ts to limit this.

curity" code
browser.
code
omise security.
000 lines.

ntire system
CB
all data.
in TCB.

Cryptography in the TCB

What happens if data is sent through Donald's network?

Solution: Sender and receiver scramble communication in a way that Donald cannot understand and cannot silently corrupt.

OpenSSL crypto li
500000 lines of co
are many other cry

All of this is in the
Many devastating

Why is crypto so

stems
this.

de

urity.

m

Cryptography in the TCB

What happens if data is sent through Donald's network?

Solution: Sender and receiver scramble communication in a way that Donald cannot understand and cannot silently corrupt.

OpenSSL crypto library has
500000 lines of code, and th
are many other crypto librar

All of this is in the TCB.
Many devastating security b

Why is crypto so big?

Cryptography in the TCB

What happens if data is sent through Donald's network?

Solution: Sender and receiver scramble communication in a way that Donald cannot understand and cannot silently corrupt.

OpenSSL crypto library has 500000 lines of code, and there are many other crypto libraries.

All of this is in the TCB.
Many devastating security bugs.

Why is crypto so big?

Cryptography in the TCB

What happens if data is sent through Donald's network?

Solution: Sender and receiver scramble communication in a way that Donald cannot understand and cannot silently corrupt.

OpenSSL crypto library has 500000 lines of code, and there are many other crypto libraries.

All of this is in the TCB.
Many devastating security bugs.

Why is crypto so big?

Most important answer:
the pursuit of performance.

(Same issue elsewhere in TCB,
but most blatant for crypto.
The rest of this talk
will focus on crypto.)

raphy in the TCB

appens if data is sent
Donald's network?

: Sender and receiver
e communication in a way
nald cannot understand
not silently corrupt.

OpenSSL crypto library has
500000 lines of code, and there
are many other crypto libraries.

All of this is in the TCB.
Many devastating security bugs.

Why is crypto so big?

Most important answer:
the pursuit of performance.

(Same issue elsewhere in TCB,
but most blatant for crypto.
The rest of this talk
will focus on crypto.)

e.g. Vari
arithmet
consume
Includes
optimize

ne TCB

data is sent
network?

and receiver
ication in a way
ot understand
y corrupt.

OpenSSL crypto library has
500000 lines of code, and there
are many other crypto libraries.

All of this is in the TCB.
Many devastating security bugs.

Why is crypto so big?

Most important answer:
the pursuit of performance.

(Same issue elsewhere in TCB,
but most blatant for crypto.
The rest of this talk
will focus on crypto.)

e.g. Variable-lengt
arithmetic library i
consumes 50000 li
Includes 38 asm in
optimized for vario

t

er
a way
and

OpenSSL crypto library has 500000 lines of code, and there are many other crypto libraries.

All of this is in the TCB.
Many devastating security bugs.

Why is crypto so big?

Most important answer:
the pursuit of performance.

(Same issue elsewhere in TCB,
but most blatant for crypto.
The rest of this talk
will focus on crypto.)

e.g. Variable-length-big-inte
arithmetic library inside Ope
consumes 50000 lines of coc
Includes 38 asm implementa
optimized for various CPUs.

OpenSSL crypto library has 500000 lines of code, and there are many other crypto libraries.

All of this is in the TCB.
Many devastating security bugs.

Why is crypto so big?

Most important answer:
the pursuit of performance.

(Same issue elsewhere in TCB, but most blatant for crypto.
The rest of this talk
will focus on crypto.)

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code.
Includes 38 asm implementations optimized for various CPUs.

OpenSSL crypto library has 500000 lines of code, and there are many other crypto libraries.

All of this is in the TCB.
Many devastating security bugs.

Why is crypto so big?

Most important answer:
the pursuit of performance.

(Same issue elsewhere in TCB, but most blatant for crypto.
The rest of this talk will focus on crypto.)

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code.
Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA signature verification:
$(H(M)/S)B + (x(R)/S)A = R$,
with S checked to be nonzero.

OpenSSL has complicated code for fast computation of $1/S$.

Checking $H(M)B + x(R)A = SR$ would be somewhat slower.

L crypto library has
lines of code, and there
y other crypto libraries.

is is in the TCB.
evastating security bugs.

crypto so big?

portant answer:
uit of performance.

ssue elsewhere in TCB,
t blatant for crypto.
of this talk
s on crypto.)

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA signature verification: $(H(M)/S)B + (x(R)/S)A = R$, with S checked to be nonzero.

OpenSSL has complicated code for fast computation of $1/S$.

Checking $H(M)B + x(R)A = SR$ would be somewhat slower.

e.g. NIS
$2^{256} - 2$

ECDSA
reductio
an integ

Write A
$(A_{15}, A_1$
$A_8, A_7,$
meaning

Define
$T; S_1; S_2$
as

brary has
de, and there
ypto libraries.

e TCB.
security bugs.

big?

nswer:
formance.

here in TCB,
for crypto.
lk
to.)

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA signature verification: $(H(M)/S)B + (x(R)/S)A = R$, with S checked to be nonzero.

OpenSSL has complicated code for fast computation of $1/S$.

Checking $H(M)B + x(R)A = SR$ would be somewhat slower.

e.g. NIST P-256 p
$2^{256} - 2^{224} + 2^{192}$

ECDSA standard s
reduction procedu
an integer “A less

Write A as
$(A_{15}, A_{14}, A_{13}, A_{1}$
A_8, A_7, A_6, A_5, A
meaning $\sum_i A_i 2^{32}$

Define
$T; S_1; S_2; S_3; S_4; D$
as

here
ies.
ugs.
CB,

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA signature verification: $(H(M)/S)B + (x(R)/S)A = R$, with S checked to be nonzero.

OpenSSL has complicated code for fast computation of $1/S$.

Checking $H(M)B + x(R)A = SR$ would be somewhat slower.

e.g. NIST P-256 prime p is
$2^{256} - 2^{224} + 2^{192} + 2^{96} -$ 1

ECDSA standard specifies reduction procedure given an integer "A less than p^2":

Write A as
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}$
$A_8, A_7, A_6, A_5, A_4, A_3, A_2,$
meaning $\sum_i A_i 2^{32i}$.

Define
$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3$
as

e.g. Variable-length-big-integer arithmetic library inside OpenSSL consumes 50000 lines of code. Includes 38 asm implementations optimized for various CPUs.

e.g. ECDSA signature verification: $(H(M)/S)B + (x(R)/S)A = R$, with S checked to be nonzero.

OpenSSL has complicated code for fast computation of $1/S$.

Checking $H(M)B + x(R)A = SR$ would be somewhat slower.

e.g. NIST P-256 prime p is $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

ECDSA standard specifies reduction procedure given an integer "A less than p^2":

Write A as $(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9,$
$A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$,
meaning $\sum_i A_i 2^{32i}$.

Define
$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$
as

able-length-big-integer
tic library inside OpenSSL
es 50000 lines of code.
38 asm implementations
ed for various CPUs.

DSA signature verification:
$S)B + (x(R)/S)A = R$,
checked to be nonzero.

L has complicated code
computation of $1/S$.

g $H(M)B + x(R)A = SR$
e somewhat slower.

e.g. NIST P-256 prime p is
$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

ECDSA standard specifies reduction procedure given an integer "A less than p^2":

Write A as
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9, A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$,
meaning $\sum_i A_i 2^{32i}$.

Define
$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$
as

$(A_7, A_6,$
$(A_{15}, A_1$
$(0, A_{15},$
$(A_{15}, A_1$
$(A_8, A_{13}$
$(A_{10}, A_8$
$(A_{11}, A_9$
$(A_{12}, 0,$
$(A_{13}, 0,$

Comput
$S_4 - D_1$

Reduce
subtract

h-big-integer
nside OpenSSL
nes of code.
mplementations
ous CPUs.

ture verification:
$(R)/S)A = R$,
be nonzero.

plicated code
on of $1/S$.

$+ x(R)A = SR$
at slower.

e.g. NIST P-256 prime p is
$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

ECDSA standard specifies
reduction procedure given
an integer "A less than p^2":

Write A as
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9,$
$A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$,
meaning $\sum_i A_i 2^{32i}$.

Define
$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$
as

$(A_7, A_6, A_5, A_4, A_3$
$(A_{15}, A_{14}, A_{13}, A_{12}$
$(0, A_{15}, A_{14}, A_{13},$
$(A_{15}, A_{14}, 0, 0, 0,$
$(A_8, A_{13}, A_{15}, A_{14},$
$(A_{10}, A_8, 0, 0, 0, A_1$
$(A_{11}, A_9, 0, 0, A_{15},$
$(A_{12}, 0, A_{10}, A_9, A_8$
$(A_{13}, 0, A_{11}, A_{10},$

Compute $T + 2S_1$
$S_4 - D_1 - D_2 - D$

Reduce modulo p
subtracting a few

ger
enSSL
de.
tions

cation:
$= R$,
ro.

ode
.

$= SR$

e.g. NIST P-256 prime p is
$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

ECDSA standard specifies reduction procedure given an integer "A less than p^2":

Write A as
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9,$
$A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$,
meaning $\sum_i A_i 2^{32i}$.

Define
$T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$
as

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1,$
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0$
$(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0)$
$(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8$
$(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11},$
$(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_1$
$(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13},$
$(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}$
$(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15},$

Compute $T + 2S_1 + 2S_2 +$
$S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p "by addin
subtracting a few copies" of

e.g. NIST P-256 prime p is $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

ECDSA standard specifies reduction procedure given an integer "A less than p^2":

Write A as $(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, A_{10}, A_9,$ $A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$, meaning $\sum_i A_i 2^{32i}$.

Define $T; S_1; S_2; S_3; S_4; D_1; D_2; D_3; D_4$ as

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$;
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0)$;
$(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0)$;
$(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8)$;
$(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9)$;
$(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11})$;
$(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12})$;
$(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13})$;
$(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14})$.

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p "by adding or subtracting a few copies" of p.

T P-256 prime p is
$^{224} + 2^{192} + 2^{96} - 1$.

standard specifies
n procedure given
er “A less than p^2”:

as
$_4, A_{13}, A_{12}, A_{11}, A_{10}, A_9,$
$A_6, A_5, A_4, A_3, A_2, A_1, A_0),$
; $\sum_i A_i 2^{32i}$.

$_2; S_3; S_4; D_1; D_2; D_3; D_4$

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$;
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0)$;
$(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0)$;
$(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8)$;
$(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9)$;
$(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11})$;
$(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12})$;
$(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13})$;
$(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14})$.

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p “by adding or subtracting a few copies” of p.

Next-ger

One of r
removing
security,

In partic
simple h
setting r

e.g. 200(
is twice
and muc

>10000(
today: i(
Tor, QU

rime p is

$+ 2^{96} - 1$.

specifies

re given

than p^2":

$, A_{11}, A_{10}, A_9,$

$, A_3, A_2, A_1, A_0),$

i.

$_1; D_2; D_3; D_4$

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$;
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0)$;
$(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0)$;
$(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8)$;
$(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9)$;
$(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11})$;
$(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12})$;
$(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13})$;
$(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14})$.

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p "by adding or subtracting a few copies" of p.

Next-generation cr

One of my favorite
removing tensions
security, simplicity

In particular, desig
simple high-securit
setting new speed

e.g. 2006 Bernstei
is twice as fast as
and much simpler

>1000000000 Cur
today: iOS, Signa
Tor, QUIC, Whats

1.

$, A_9,$

$A_1, A_0),$

$; D_4$

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0);$
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0);$
$(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0);$
$(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8);$
$(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9);$
$(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11});$
$(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12});$
$(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13});$
$(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14}).$

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p "by adding or subtracting a few copies" of p.

<u>Next-generation crypto</u>

One of my favorite topics: removing tensions between security, simplicity, speed.

In particular, designing simple high-security crypto setting new speed records.

e.g. 2006 Bernstein "Curve2
is twice as fast as standard
and much simpler to implem

$>$1000000000 Curve25519 u
today: iOS, Signal, OpenSS
Tor, QUIC, WhatsApp, more

$(A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0)$;
$(A_{15}, A_{14}, A_{13}, A_{12}, A_{11}, 0, 0, 0)$;
$(0, A_{15}, A_{14}, A_{13}, A_{12}, 0, 0, 0)$;
$(A_{15}, A_{14}, 0, 0, 0, A_{10}, A_9, A_8)$;
$(A_8, A_{13}, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9)$;
$(A_{10}, A_8, 0, 0, 0, A_{13}, A_{12}, A_{11})$;
$(A_{11}, A_9, 0, 0, A_{15}, A_{14}, A_{13}, A_{12})$;
$(A_{12}, 0, A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13})$;
$(A_{13}, 0, A_{11}, A_{10}, A_9, 0, A_{15}, A_{14})$.

Compute $T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4$.

Reduce modulo p "by adding or subtracting a few copies" of p.

Next-generation crypto

One of my favorite topics: removing tensions between security, simplicity, speed.

In particular, designing simple high-security crypto setting new speed records.

e.g. 2006 Bernstein "Curve25519" is twice as fast as standard ECC and much simpler to implement.

$>$1000000000 Curve25519 users today: iOS, Signal, OpenSSH, Tor, QUIC, WhatsApp, more.

$A_5, A_4, A_3, A_2, A_1, A_0)$;

$_4, A_{13}, A_{12}, A_{11}, 0, 0, 0)$;

$A_{14}, A_{13}, A_{12}, 0, 0, 0)$;

$_4, 0, 0, 0, A_{10}, A_9, A_8)$;

$, A_{15}, A_{14}, A_{13}, A_{11}, A_{10}, A_9)$;

$, 0, 0, 0, A_{13}, A_{12}, A_{11})$;

$, 0, 0, A_{15}, A_{14}, A_{13}, A_{12})$;

$A_{10}, A_9, A_8, A_{15}, A_{14}, A_{13})$;

$A_{11}, A_{10}, A_9, 0, A_{15}, A_{14})$.

e $T + 2S_1 + 2S_2 + S_3 +$
$- D_2 - D_3 - D_4$.

modulo p “by adding or
ing a few copies” of p.

Next-generation crypto

One of my favorite topics: removing tensions between security, simplicity, speed.

In particular, designing simple high-security crypto setting new speed records.

e.g. 2006 Bernstein “Curve25519” is twice as fast as standard ECC and much simpler to implement.

$>$1000000000 Curve25519 users today: iOS, Signal, OpenSSH, Tor, QUIC, WhatsApp, more.

NaCl: fa
high-sec
work wit
nacl.cr

$_3, A_2, A_1, A_0)$;

$_2, A_{11}, 0, 0, 0)$;

$A_{12}, 0, 0, 0)$;

$A_{10}, A_9, A_8)$;

$A_{13}, A_{11}, A_{10}, A_9)$;

$_{13}, A_{12}, A_{11})$;

$A_{14}, A_{13}, A_{12})$;

$_8, A_{15}, A_{14}, A_{13})$;

$A_9, 0, A_{15}, A_{14})$.

$+ 2S_2 + S_3 +$

$D_3 - D_4$.

"by adding or

copies" of p.

Next-generation crypto

One of my favorite topics: removing tensions between security, simplicity, speed.

In particular, designing simple high-security crypto setting new speed records.

e.g. 2006 Bernstein "Curve25519" is twice as fast as standard ECC and much simpler to implement.

>1000000000 Curve25519 users today: iOS, Signal, OpenSSH, Tor, QUIC, WhatsApp, more.

NaCl: fast easy-to

high-security crypt

work with Lange a

nacl.cr.yp.to

A_0);

, 0);

);

);

A_{10}, A_9);

$_1$);

A_{12});

., A_{13});

A_{14}).

$S_3 +$

g or

p.

Next-generation crypto

One of my favorite topics: removing tensions between security, simplicity, speed.

In particular, designing simple high-security crypto setting new speed records.

e.g. 2006 Bernstein "Curve25519" is twice as fast as standard ECC and much simpler to implement.

$>$1000000000 Curve25519 users today: iOS, Signal, OpenSSH, Tor, QUIC, WhatsApp, more.

NaCl: fast easy-to-use high-security crypto library. work with Lange and Schwa

nacl.cr.yp.to

Next-generation crypto

One of my favorite topics: removing tensions between security, simplicity, speed.

In particular, designing simple high-security crypto setting new speed records.

e.g. 2006 Bernstein "Curve25519" is twice as fast as standard ECC and much simpler to implement.

$>$1000000000 Curve25519 users today: iOS, Signal, OpenSSH, Tor, QUIC, WhatsApp, more.

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe.

`nacl.cr.yp.to`

Next-generation crypto

One of my favorite topics: removing tensions between security, simplicity, speed.

In particular, designing simple high-security crypto setting new speed records.

e.g. 2006 Bernstein "Curve25519" is twice as fast as standard ECC and much simpler to implement.

>1000000000 Curve25519 users today: iOS, Signal, OpenSSH, Tor, QUIC, WhatsApp, more.

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe. `nacl.cr.yp.to`

TweetNaCl: self-contained 100-tweet C library providing the same easy-to-use high-security functions. Joint work with van Gastel, Janssen, Lange, Schwabe, Smetsers. `twitter.com/tweetnacl`

Next-generation crypto

One of my favorite topics: removing tensions between security, simplicity, speed.

In particular, designing simple high-security crypto setting new speed records.

e.g. 2006 Bernstein "Curve25519" is twice as fast as standard ECC and much simpler to implement.

$>$1000000000 Curve25519 users today: iOS, Signal, OpenSSH, Tor, QUIC, WhatsApp, more.

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe. nacl.cr.yp.to

TweetNaCl: self-contained 100-tweet C library providing the same easy-to-use high-security functions. Joint work with van Gastel, Janssen, Lange, Schwabe, Smetsers. twitter.com/tweetnacl

Can we guarantee zero bugs in TweetNaCl? And in NaCl?

neration crypto

my favorite topics:
g tensions between
simplicity, speed.

cular, designing
igh-security crypto
new speed records.

6 Bernstein "Curve25519"
as fast as standard ECC
ch simpler to implement.

00000 Curve25519 users
OS, Signal, OpenSSH,
IC, WhatsApp, more.

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe.
nacl.cr.yp.to

TweetNaCl: self-contained 100-tweet C library providing the same easy-to-use high-security functions. Joint work with van Gastel, Janssen, Lange, Schwabe, Smetsers.
twitter.com/tweetnacl

Can we guarantee zero bugs in TweetNaCl? And in NaCl?

Biggest
between
such as
and (e.g

rypto

e topics:
between
, speed.

gning
ty crypto
records.

n "Curve25519"
standard ECC
to implement.

ve25519 users
l, OpenSSH,
App, more.

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe. nacl.cr.yp.to

TweetNaCl: self-contained 100-tweet C library providing the same easy-to-use high-security functions. Joint work with van Gastel, Janssen, Lange, Schwabe, Smetsers. twitter.com/tweetnacl

Can we guarantee zero bugs in TweetNaCl? And in NaCl?

Biggest challenge:
between big-intege
such as $a, b \mapsto ab$
and (e.g.) 32-bit c

25519”
ECC
ent.

sers
H,
e.

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe.
nacl.cr.yp.to

TweetNaCl: self-contained 100-tweet C library providing the same easy-to-use high-security functions. Joint work with van Gastel, Janssen, Lange, Schwabe, Smetsers.
twitter.com/tweetnacl

Can we guarantee zero bugs in TweetNaCl? And in NaCl?

Biggest challenge: the gap
between big-integer operatio
such as $a, b \mapsto ab \bmod 2^{255}$
and (e.g.) 32-bit operations.

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe.

nacl.cr.yp.to

TweetNaCl: self-contained 100-tweet C library providing the same easy-to-use high-security functions. Joint work with van Gastel, Janssen, Lange, Schwabe, Smetsers.

twitter.com/tweetnacl

Can we guarantee zero bugs in TweetNaCl? And in NaCl?

Biggest challenge: the gap between big-integer operations such as $a, b \mapsto ab \bmod 2^{255} - 19$ and (e.g.) 32-bit operations.

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe.
nacl.cr.yp.to

TweetNaCl: self-contained 100-tweet C library providing the same easy-to-use high-security functions. Joint work with van Gastel, Janssen, Lange, Schwabe, Smetsers.
twitter.com/tweetnacl

Can we guarantee zero bugs in TweetNaCl? And in NaCl?

Biggest challenge: the gap between big-integer operations such as $a, b \mapsto ab \bmod 2^{255} - 19$ and (e.g.) 32-bit operations.

Some big-integer software has been formally verified. Could NaCl switch to this?

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe.
nacl.cr.yp.to

TweetNaCl: self-contained 100-tweet C library providing the same easy-to-use high-security functions. Joint work with van Gastel, Janssen, Lange, Schwabe, Smetsers.
twitter.com/tweetnacl

Can we guarantee zero bugs in TweetNaCl? And in NaCl?

Biggest challenge: the gap between big-integer operations such as $a, b \mapsto ab \bmod 2^{255} - 19$ and (e.g.) 32-bit operations.

Some big-integer software has been formally verified. Could NaCl switch to this?

1. Not state-of-the-art speed. Okay for TweetNaCl; not NaCl.

NaCl: fast easy-to-use high-security crypto library. Joint work with Lange and Schwabe.
nacl.cr.yp.to

TweetNaCl: self-contained 100-tweet C library providing the same easy-to-use high-security functions. Joint work with van Gastel, Janssen, Lange, Schwabe, Smetsers.
twitter.com/tweetnacl

Can we guarantee zero bugs in TweetNaCl? And in NaCl?

Biggest challenge: the gap between big-integer operations such as $a, b \mapsto ab \bmod 2^{255} - 19$ and (e.g.) 32-bit operations.

Some big-integer software has been formally verified. Could NaCl switch to this?

1. Not state-of-the-art speed. Okay for TweetNaCl; not NaCl.

2. Input-dependent timing. Timing can leak secret keys. Not okay even for TweetNaCl.

ast easy-to-use
urity crypto library. Joint
th Lange and Schwabe.
r.yp.to

aCl: self-contained
et C library providing
e easy-to-use
urity functions. Joint
th van Gastel, Janssen,
Schwabe, Smetsers.
r.com/tweetnacl

**guarantee zero bugs in
laCl? And in NaCl?**

Biggest challenge: the gap
between big-integer operations
such as $a, b \mapsto ab \bmod 2^{255} - 19$
and (e.g.) 32-bit operations.

Some big-integer software
has been formally verified.
Could NaCl switch to this?

1. Not state-of-the-art speed.
Okay for TweetNaCl; not NaCl.

2. Input-dependent timing.
Timing can leak secret keys.
Not okay even for TweetNaCl.

ACM C
Schwabe
"Verifyi
compute
correctn
in two h
Curve25

-use
o library. Joint
nd Schwabe.

ontained
y providing
use
ions. Joint
stel, Janssen,
Smetsers.
eetnacl

e zero bugs in
d in NaCl?

Biggest challenge: the gap between big-integer operations such as $a, b \mapsto ab \bmod 2^{255} - 19$ and (e.g.) 32-bit operations.

Some big-integer software has been formally verified. Could NaCl switch to this?

1. Not state-of-the-art speed. Okay for TweetNaCl; not NaCl.

2. Input-dependent timing. Timing can leak secret keys. Not okay even for TweetNaCl.

ACM CCS 2014 C
Schwabe–Tsai–Wa
"Verifying Curve25
computer-aided pr
correctness of mai
in two high-speed
Curve25519 impler

Joint
be.

g

nt
en,

gs in
?

Biggest challenge: the gap between big-integer operations such as $a, b \mapsto ab \bmod 2^{255} - 19$ and (e.g.) 32-bit operations.

Some big-integer software has been formally verified. Could NaCl switch to this?

1. Not state-of-the-art speed. Okay for TweetNaCl; not NaCl.

2. Input-dependent timing. Timing can leak secret keys. Not okay even for TweetNaCl.

ACM CCS 2014 Chen–Hsu–
Schwabe–Tsai–Wang–Yang–
"Verifying Curve25519 softw
computer-aided proof of
correctness of main loops
in two high-speed asm
Curve25519 implementation

Biggest challenge: the gap between big-integer operations such as $a, b \mapsto ab \bmod 2^{255} - 19$ and (e.g.) 32-bit operations.

Some big-integer software has been formally verified. Could NaCl switch to this?

1. Not state-of-the-art speed. Okay for TweetNaCl; not NaCl.

2. Input-dependent timing. Timing can leak secret keys. Not okay even for TweetNaCl.

ACM CCS 2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": computer-aided proof of correctness of main loops in two high-speed asm Curve25519 implementations.

Biggest challenge: the gap between big-integer operations such as $a, b \mapsto ab \bmod 2^{255} - 19$ and (e.g.) 32-bit operations.

Some big-integer software has been formally verified. Could NaCl switch to this?

1. Not state-of-the-art speed. Okay for TweetNaCl; not NaCl.

2. Input-dependent timing. Timing can leak secret keys. Not okay even for TweetNaCl.

ACM CCS 2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": computer-aided proof of correctness of main loops in two high-speed asm Curve25519 implementations.

Proof required extensive human effort for each implementation: many detailed annotations, plus higher-level composition work.

Biggest challenge: the gap between big-integer operations such as $a, b \mapsto ab \bmod 2^{255} - 19$ and (e.g.) 32-bit operations.

Some big-integer software has been formally verified. Could NaCl switch to this?

1. Not state-of-the-art speed. Okay for TweetNaCl; not NaCl.

2. Input-dependent timing. Timing can leak secret keys. Not okay even for TweetNaCl.

ACM CCS 2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": computer-aided proof of correctness of main loops in two high-speed asm Curve25519 implementations.

Proof required extensive human effort for each implementation: many detailed annotations, plus higher-level composition work.

Each proof also required many hours of computer time.

challenge: the gap
big-integer operations
$a, b \mapsto ab \bmod 2^{255} - 19$
.) 32-bit operations.

g-integer software
n formally verified.
aCl switch to this?

state-of-the-art speed.
r TweetNaCl; not NaCl.

-dependent timing.
can leak secret keys.
y even for TweetNaCl.

ACM CCS 2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": computer-aided proof of correctness of main loops in two high-speed asm Curve25519 implementations.

Proof required extensive human effort for each implementation: many detailed annotations, plus higher-level composition work.

Each proof also required many hours of computer time.

Joint wo
new veri
focusing
gfverif

Automa
graph fr

Automa
convert
New pee

Ask hum
annotati
computa

the gap
er operations
mod $2^{255} - 19$
operations.

software
verified.
n to this?

e-art speed.
Cl; not NaCl.

nt timing.
ecret keys.
TweetNaCl.

ACM CCS 2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": computer-aided proof of correctness of main loops in two high-speed asm Curve25519 implementations.

Proof required extensive human effort for each implementation: many detailed annotations, plus higher-level composition work.

Each proof also required many hours of computer time.

Joint work with Sc
new verifier `gfver`
focusing on arithm
`gfverif.cryptoj`

Automatically buil
graph from origina

Automatically ana
convert ops into p
New peephole rang

Ask human for occ
annotations expres
computations on i

ons
– 19

d.
aCl.

.
Cl.

ACM CCS 2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": computer-aided proof of correctness of main loops in two high-speed asm Curve25519 implementations.

Proof required extensive human effort for each implementation: many detailed annotations, plus higher-level composition work.

Each proof also required many hours of computer time.

Joint work with Schwabe: new verifier `gfverif` focusing on arithmetic mod
gfverif.cryptojedi.org

Automatically build computa
graph from original code.

Automatically analyze range
convert ops into polynomials
New peephole range optimiz

Ask human for occasional
annotations expressing high-
computations on integers m

ACM CCS 2014 Chen–Hsu–Lin–Schwabe–Tsai–Wang–Yang–Yang "Verifying Curve25519 software": computer-aided proof of correctness of main loops in two high-speed asm Curve25519 implementations.

Proof required extensive human effort for each implementation: many detailed annotations, plus higher-level composition work.

Each proof also required many hours of computer time.

Joint work with Schwabe: new verifier `gfverif` focusing on arithmetic mod p. `gfverif.cryptojedi.org`

Automatically build computation graph from original code.

Automatically analyze ranges, convert ops into polynomials. New peephole range optimizer.

Ask human for occasional annotations expressing high-level computations on integers mod p.

CS 2014 Chen–Hsu–Lin–
e–Tsai–Wang–Yang–Yang
ng Curve25519 software":
er-aided proof of
ess of main loops
igh-speed asm
519 implementations.

quired extensive human
r each implementation:
etailed annotations, plus
evel composition work.

oof also required
ours of computer time.

Joint work with Schwabe:
new verifier `gfverif`
focusing on arithmetic mod p.
`gfverif.cryptojedi.org`

Automatically build computation graph from original code.

Automatically analyze ranges, convert ops into polynomials.
New peephole range optimizer.

Ask human for occasional annotations expressing high-level computations on integers mod p.

Have ve
computa
for anot

Only 1 r

Under 3
annotati

Usable

Continui
annotati
be able
annotati

hen–Hsu–Lin–
ang–Yang–Yang
5519 software”:
oof of
n loops
asm
mentations.

ensive human
lementation:
otations, plus
osition work.

quired
mputer time.

Joint work with Schwabe:
new verifier `gfverif`
focusing on arithmetic mod p.
`gfverif.cryptojedi.org`

Automatically build computation graph from original code.

Automatically analyze ranges, convert ops into polynomials.
New peephole range optimizer.

Ask human for occasional annotations expressing high-level computations on integers mod p.

Have verified entir
computation, not j
for another implem

Only 1 minute of c

Under 300 lines of
annotations per im

Usable by crypto

Continuing to imp
annotation langua
be able to reduce
annotations per im

Lin–
-Yang
vare”:

s.

man
on:
plus
rk.

ne.

Joint work with Schwabe:
new verifier `gfverif`
focusing on arithmetic mod p.
gfverif.cryptojedi.org

Automatically build computation graph from original code.

Automatically analyze ranges, convert ops into polynomials.
New peephole range optimizer.

Ask human for occasional annotations expressing high-level computations on integers mod p.

Have verified entire Curve25
computation, not just main
for another implementation.

Only 1 minute of computer

Under 300 lines of easy
annotations per implementa

Usable by crypto develope

Continuing to improve `gfve`
annotation language. Shoul
be able to reduce below 100
annotations per implementa

Joint work with Schwabe:
new verifier `gfverif`
focusing on arithmetic mod p.
`gfverif.cryptojedi.org`

Automatically build computation graph from original code.

Automatically analyze ranges, convert ops into polynomials. New peephole range optimizer.

Ask human for occasional annotations expressing high-level computations on integers mod p.

Have verified entire Curve25519 computation, not just main loop, for another implementation.

Only 1 minute of computer time.

Under 300 lines of easy annotations per implementation.

Usable by crypto developers.

Continuing to improve `gfverif` annotation language. Should be able to reduce below 100 annotations per implementation.