

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

EuroBSDCon 2022

Samuel AUBERTIN

18/09/2022

Samuel AUBERTIN - sk4nz

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



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

— HAL 9000

What if HAL 9000 got backdoored ?

- Physical security

- Physical security
- Hardware

- Physical security
- Hardware
- Firmware

- Physical security
- Hardware
- Firmware
- Kernel

- Physical security
- Hardware
- Firmware
- Kernel
- Userland

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

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

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

Trusting Trust

*"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.
2. If you compile login(1), backdoor it.

Thompson's backdoor : wrapping features altogether

Compiler Source $CS \rightarrow X \rightarrow$ Compiler C

Thompson's backdoor : wrapping features altogether

Compiler Source $CS \rightarrow X \rightarrow$ Compiler C

Backdoored Compiler Source $\rightarrow C \rightarrow$ Backdoored Compiler BC

Thompson's backdoor : wrapping features altogether

Compiler Source $CS \rightarrow X \rightarrow$ Compiler C

Backdoored Compiler Source $\rightarrow C \rightarrow$ Backdoored Compiler BC

$CS \rightarrow BC \rightarrow$ Self-Replicating Backdoored Compiler KBC'

Thompson's backdoor : wrapping features altogether

Compiler Source $CS \rightarrow X \rightarrow$ Compiler C

Backdoored Compiler Source $\rightarrow C \rightarrow$ Backdoored Compiler BC

$CS \rightarrow BC \rightarrow$ Self-Replicating Backdoored Compiler KBC'

$CS \rightarrow KBC' \rightarrow KBC''$

Thompson's backdoor : wrapping features altogether

Compiler Source $CS \rightarrow X \rightarrow$ Compiler C

Backdoored Compiler Source $\rightarrow C \rightarrow$ Backdoored Compiler BC

$CS \rightarrow BC \rightarrow$ Self-Replicating Backdoored Compiler KBC'

$CS \rightarrow KBC' \rightarrow KBC''$

Program Source $S \rightarrow KBC \rightarrow$ Backdoored Program

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 1984 paper cites an *Unknown Air Force Document*.
MULTICS SECURITY EVALUATION: VULNERABILITY ANALYSIS²
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."

- 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

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

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

\$(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
```

\$(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
```

0. Templated backdoor
template(backdoor)

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

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

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

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

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

Detection : runtime analysis

Detection : runtime analysis

- btrace
- ktrace
- gdb
- radare2

David A. Wheeler PhD dissertation³

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

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

David A. Wheeler PhD dissertation³

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

$CS \rightarrow Y \rightarrow$ Compiler $Y1$

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

David A. Wheeler PhD dissertation³

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

$CS \rightarrow Y \rightarrow$ Compiler $Y1$

$CS \rightarrow X1 \rightarrow$ Compiler $X2$

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

David A. Wheeler PhD dissertation³

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

$CS \rightarrow Y \rightarrow$ Compiler $Y1$

$CS \rightarrow X1 \rightarrow$ Compiler $X2$

$CS \rightarrow Y1 \rightarrow$ Compiler $Y2$

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

David A. Wheeler PhD dissertation³

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

$CS \rightarrow Y \rightarrow$ Compiler $Y1$

$CS \rightarrow X1 \rightarrow$ Compiler $X2$

$CS \rightarrow Y1 \rightarrow$ Compiler $Y2$

Are $X2$ and $Y2$ binary equivalent?

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

- *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>

Let's discuss !

<https://www.sk4.nz/eurobsdcon22.git>