Group Privacy Technology II

Information Privacy with Applications

David Sidi (dsidi@email.arizona.edu)


THE END OF TRUST
An investigation of surveillance in the digital age, featuring
EDWARD SNOWDEN
JENNA WORTHAM
CORY DOCTOROW
DOUGLAS RUSHKOFF
MALKIA CYRIL
GABRIELLA COLEMAN
JULIA ANGWIN
AND MANY MORE
With special advisor the Electronic Frontier Foundation
THE END OF TRUST
McSWEENEY'S 54

Administration

- Assignment I, part II due today
- The next assignment: cracking
- Preparation for the integrated session
  - worksheet
  - survey (https://uarizona.co1.qualtrics.com/jfe/form/SV_3UJ9mwhMhApONPU)

Today

- Thompson on “trusting trust”
- But first...

Privacy and transparency: 9/11 edition

- 9/11 is coming up, but we will be doing the integrated session next time
- (audio clip)

  Question: Evaluate the claim that this case is about privacy as control for bin Laden.

Decreasing trust in third parties

- “A different family of privacy technologies considers however that placing such high levels of trust in organizations should be avoided whenever possible, as they leave individuals vulnerable to incompetent or malicious organizations.” (Diaz et al. 2)
- What do the details look like on this view? Danezis on “soft vs. hard privacy technologies”

Two families of privacy technologies

Soft Privacy Technologies

- Focus on compliance.
- Focus on “internal controls”.
- **Assumption: a third party is entrusted with the user data.**
- Threat model: third party is trusted to process user data according to user wishes.
- Examples technologies:
  - Access control, tunnel encryption (SSL/TLS)
- “Keeping honest services safe from insiders / employees”.

Hard Privacy Technologies

- Stronger focus on data minimization.
- **Assumption: there exists no single third party that may be trusted with user data.**
- Threat model: a service is in the hands of the adversary; may be coerced; may be hacked.
- Common assumption: k-out-of-n honest third parties.
- May relay on service integrity if auditing is possible.
- **Challenge: achieve functionality without revealing data!**

Slide credit: George Danezis

Two families of privacy technologies

Soft Privacy Technologies

- Focus on compliance.
- Focus on “internal controls”.
- **Assumption: a third party is entrusted with the user data.**
- Threat model: third party is trusted to process user data according to user wishes.
- Examples technologies:
  - Access control, tunnel encryption (SSL/TLS)
- “Keeping honest services safe from insiders / employees”.

Hard Privacy Technologies

- Stronger focus on data minimization.
- **Assumption: there exists no single third party that may be trusted with user data.**
- Threat model: a service is in the hands of the adversary; may be coerced; may be hacked.
- Common assumption: k-out-of-n honest third parties.
- May relay on service integrity if auditing is possible.
- **Challenge: achieve functionality without revealing data!**

Slide credit: George Danezis

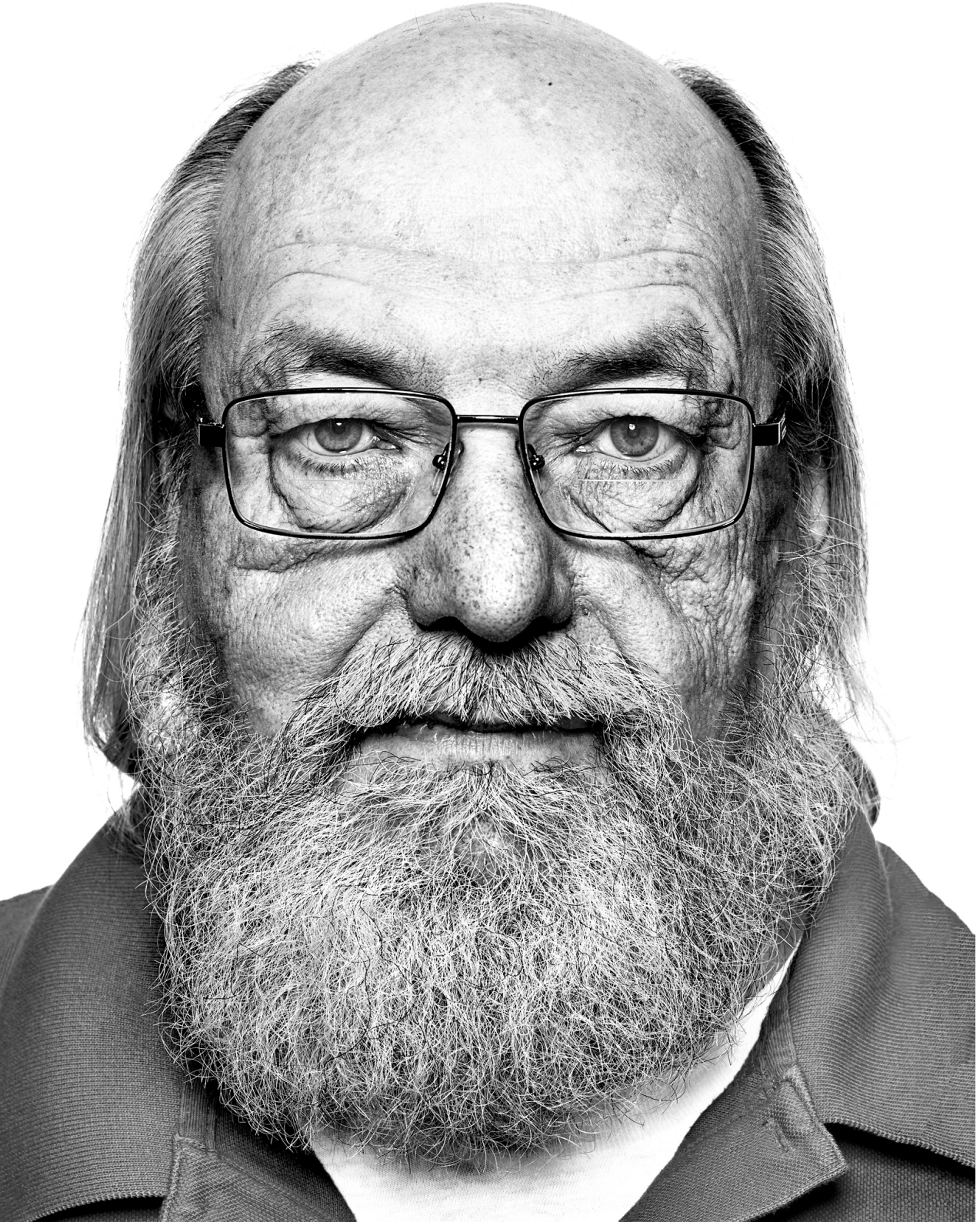
Philip Zimmerman on trust

- “When examining a cryptographic software package, the question always remains, why should you trust this product? Even if you examined the source code yourself, not everyone has the cryptographic experience to judge the security. Even if you are an experienced cryptographer, subtle weaknesses in the algorithms could still elude you.”

“In some ways, cryptography is like pharmaceuticals. Its integrity may be absolutely crucial. Bad penicillin looks the same as good penicillin.”

You can tell if your spreadsheet software is wrong, but how do you tell if your cryptography package is weak? The ciphertext produced by a weak encryption algorithm looks as good as ciphertext produced by a strong encryption algorithm. There's a lot of snake oil out there. A lot of quack cures.”

Trusting software: The trusting trust attack



The actual bug I planted in the compiler would match code in the UNIX "login" command. The replacement code would miscompile the login command so that it would accept either the intended encrypted password or a particular known password. Thus if this code were installed in binary and the binary were used to compile the login command, I could log into that system as any user. ...

the simplest quine

...write one yourself!

the simplest quine

careful! The empty program is copyrighted

/bin/true

```
#
#     Copyright (c) 1984 AT&T
#       All Rights Reserved
#
#     THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T
#     The copyright notice above does not evidence any
#     actual or intended publication of such source code.
#
#ident        "@(#)cmd/true.sh        50.1"
```

A simple Quine

```
char s[ ] = {
        '\t',
        '0',
        '\n',
        '}',
        ';',
        '\n',
        '\n',
        '/',
        '*',
        '\n',
        (213 lines deleted)
        0
};

/*
 * The string s is a
 * representation of the body
 * of this program from '0'
 * to the end.
 */

main( )
{
        int i;

        printf("char\ts[ ] = {\n");
        for(i=0; s[i]; i++)
                printf("\t%d, \n", s[i]);
        printf("%s", s);
}
```

Here are some simple transliterations to allow a non-C programmer to read this code.

=	assignment
==	equal to .EQ.
!=	not equal to .NE.
++	increment
'x'	single character constant
"xxx"	multiple character string
%d	format to convert to decimal
%s	format to convert to string
\t	tab character
\n	newline character

FIGURE 1.

A simple quine

①

```
char s[ ] = {
        '\t',
        '0',
        '\n',
        '}',
        ';',
        '\n',
        '\n',
        '/',
        '*',
        ' ',
        '\n',
        (213 lines deleted)
        0
};

②

/*
 * The string s is a
 * representation of the body
 * of this program from '0'
 * to the end.
 */

main( )
{
        int i;

        printf("char\ts[ ] = {\n");
        for(i=0; s[i]; i++)
                printf("\t%d, \n", s[i]);
        printf("%s", s);
}
```

Here are some simple transliterations to allow a non-C programmer to read this code.

=	assignment
==	equal to .EQ.
!=	not equal to .NE.
++	increment
'x'	single character constant
"xxx"	multiple character string
%d	format to convert to decimal
%s	format to convert to string
\t	tab character
\n	newline character

FIGURE 1.

A quine in Python

```
from string import Template

second = Template('print(${sq}from string import Template${sq})\nprint("second =
Template(${sq}" + second.template.encode(${sq}unicode_escape${sq}).decode() + "$
{sq}")\nprint(second.substitute(sq="${sq}"))')

print('from string import Template')

print("second = Template('" + second.template.encode('unicode_escape').decode()
+ "'")

print(second.substitute(sq="'"))
```

This a quick one I hacked together. There are (much) simpler ones, more boring ones, more interesting ones. Try it!

```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
if(c == 'v')
        return('\v');
...
```

FIGURE 2.1.

```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\ n');
if(c == 'v')
        return(11);
...
```

FIGURE 2.3.

quines in Compilers

targeting login command

```
compile(s)
char *s;
{
        ...
}
```

FIGURE 3.1.

```
compile(s)
char *s;
{
        if(match(s, "pattern")) {
                compile("bug");
                return;
        }
        ...
}
```

FIGURE 3.2.

```
compile(s)
char *s;
{
        if(match(s, "pattern1")) {
                compile ("bug1");
                return;
        }
        if(match(s, "pattern 2")) {
                compile ("bug 2");
                return;
        }
        ...
}
```

FIGURE 3.3.

emit a backdoor!

“that’s worrying... let’s audit the source code for both the login command and for the compiler we use, and recompile everything cleanly”

“that’s worrying... let’s audit the source code for both the login command and for the compiler we use, and recompile everything cleanly”

nope

quines in Compilers

```
compile(s)
char *s;
{
        ...
}
```

FIGURE 3.1.

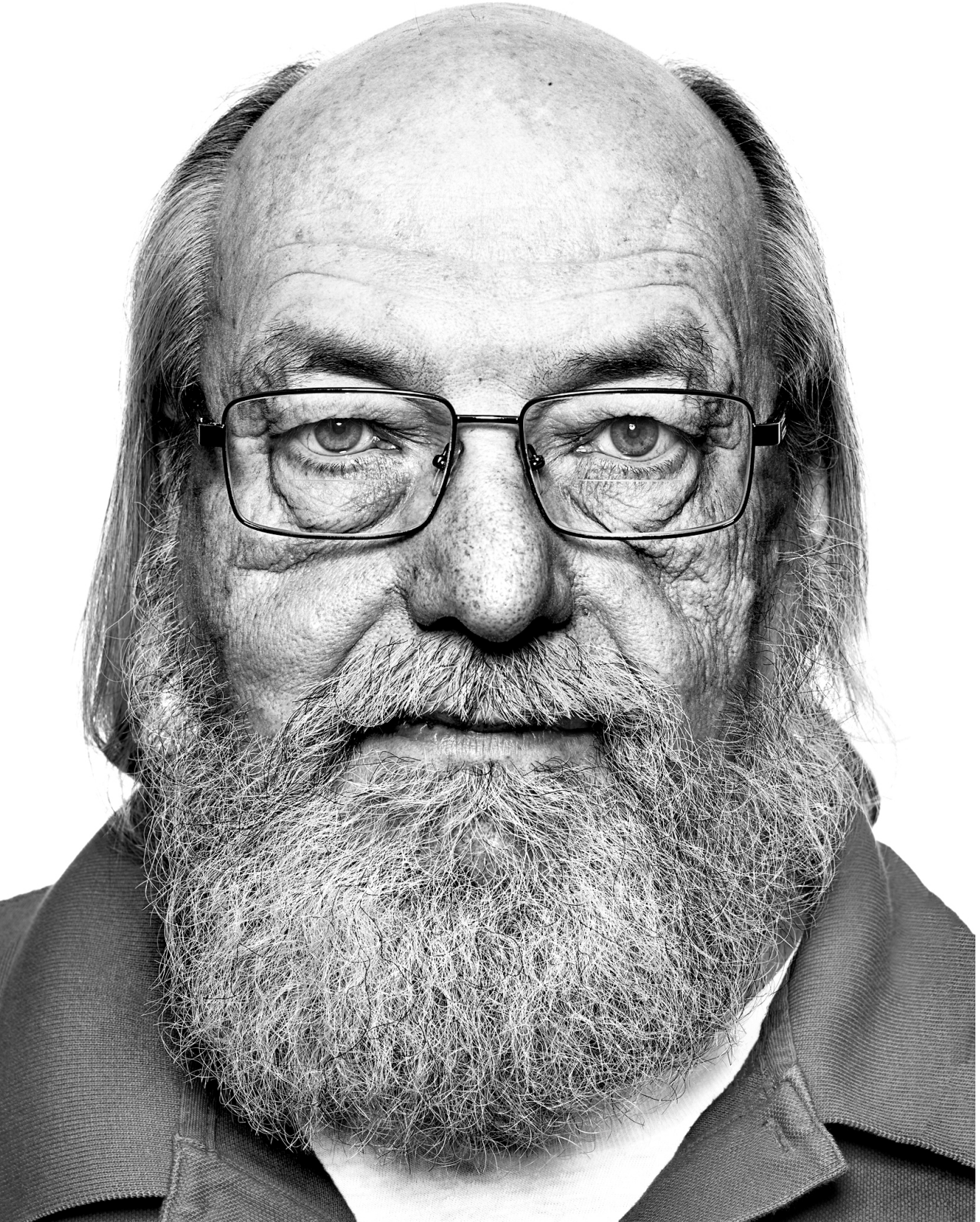
```
compile(s)
char *s;
{
        if(match(s, "pattern")) {
                compile("bug");
                return;
        }
        ...
}
```

FIGURE 3.2.

```
compile(s)
char *s;
{
        if(match(s, "pattern1")) {
                compile ("bug1");
                return;
        }
        if(match(s, "pattern 2")) {
                compile ("bug 2");
                return;
        }
        ...
}
```

FIGURE 3.3.

targeting compiler



The moral is obvious. You can't trust code that you did not totally create yourself. (Especially code from companies that employ people like me). No amount of source-level verification or scrutiny will protect you from using untrusted code.

Ken Thompson, ACM Turing Award Speech, “Reflections on Trusting Trust”

Q: **What is ‘trust’ here? What does being stuck in this situation tell us about it?**