Documents uncovered by the Electronic Frontier Foundation show that the FBI created a fake web page designed to look like a Seattle Times article, and used the page to spread tracking malware onto the suspect's computer. Creating dummy pages is a common way to spread malware — typically known as spoofing — but it's more common among criminals than law enforcement, and many are already interpreting the fake page as an attack on the press. "We are outraged that the FBI, with the apparent assistance of the U.S. Attorney's Office, misappropriated the name of The Seattle Times," a Times editor told the paper. "Not only does that cross a line, it erases it." The Associated Press echoed the concern, saying, "this ploy violated AP's name and undermined AP's credibility."
Crypto:
Passwords and RNGs

CS 642
Guest Lecturer: Adam Everspaugh
http://pages.cs.wisc.edu/~ace
Topics

Password-based Crypto

Random Number Generators
Symmetric Key Encryption

Correctness: $D_k( E_k(M,R) ) = M$
Password-based Symmetric Encryption

Correctness: \( D(pw, E(pw, M, R)) = M \)
Encrypt-then-MAC with CBC and HMAC

Ciphertext is: $(C, T_{K2})$

How do we use this with a password?
Password-based Key Derivation (PBKDF)

PBKDF(pw, salt):

\[
pw \| \text{salt} \| 1 \xrightarrow{H} H \xrightarrow{H} \ldots H \xrightarrow{H} K_1
\]

\[
pw \| \text{salt} \| 2 \xrightarrow{H} H \xrightarrow{H} \ldots H \xrightarrow{H} K_2
\]

Truncate if needed

repeat c times
PBKDF + Symmetric Encryption yields PW-Based Encryption

**Enc**(pw,M,R):
salt || R’ = R
K = PBKDF(pw,salt)
C = Enc’(K,M,R’)
Return (salt,C)

Here Enc’/Dec’ is a typical symmetric encryption scheme (CBC+HMAC)

**Dec**(pw,C):
salt || C’ = C
K = PBKDF(pw,salt)
M = Dec’(K,C’)
Return M

Attacks?
Password Distribution

From an Imperva study of released RockMe.com password database (2010)
Dictionary Attack

• Given a (message,ciphertext) pair:
• Enumerate a dictionary D of possible passwords, in order of likelihood
• Test each candidate password

```
DictionaryAttack(D,M,C):
R || C' = C
for pw* in D:
    C* = Enc(pw*,M,R)
    if C* == C':
        return pw*
```

```
R || C' = C
```

```
IV
M1

E_{K1}

C0

C1
```
PBKDF Slows Down Dictionary Attacks

Salts:
Different derived keys, even if same password
Slows down attacks against multiple users
Prevents precomputation attacks, if salts chosen randomly

Iterating c times should slow down attacks by factor of c
How Fast Are Dictionary Attacks?

- openssl speed sha1
- Assume: 4 cores @ 2.2M hashes per second

<table>
<thead>
<tr>
<th></th>
<th>Size of Dictionary</th>
<th>Computation time c=1</th>
<th>Computation time c=4096</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 digit PIN</td>
<td>10</td>
<td>0.11 seconds</td>
<td>7.8 minutes</td>
</tr>
<tr>
<td>6 alphanumerics (lowercase)</td>
<td>36</td>
<td>4.1 minutes</td>
<td>11.7 days</td>
</tr>
<tr>
<td>8 alphanumerics (mixed case)</td>
<td>62</td>
<td>287 days</td>
<td>3,222 years</td>
</tr>
</tbody>
</table>
802.11 WPA Authentication

Observe just one handshake by another party, and attacker can mount *offline* dictionary attack against the password

PMK = PBKDF( pw, ssid || ssidlength ) with c = 4096

PTK = H( PMK || ANonce || SNonce || AP
MAC address || STA MAC address )

MIC = HMAC-MD5(PTK, M2)
Attacking WPA Passwords

\[ \text{PMK} = \text{PBKDF}(\pw, \text{ssid} || \text{ssidlength}) \] 
\[ \text{with } c = 4096 \]

\[ \text{PTK} = H(\text{PMK} || \text{ANonce} || \text{SNonce} || \text{AP MAC address} || \text{STA MAC address}) \]

\[ \text{MIC} = \text{HMAC-MD5}(\text{PTK}, M2) \]

**DictionaryAttack**\((D, \text{MIC}, \text{ANonce}, \text{SNonce}, \text{SSID}, M2)\):

for \( \pw^* \) in \( D \):

\[ \text{PMK}^* = \text{PBKDF}(\pw^*, \text{ssid} || \text{ssidlength}) \]

\[ \text{PTK}^* = H(\text{PMK}^* || \text{ANonce} || \ldots) \]

\[ \text{MIC}^* = \text{HMAC-MD5}(\text{PTK}^*, M2) \]

If \( \text{MIC}^* = \text{MIC} \):

return \( \pw^* \)

return None
Recap: Password-based Crypto

- Allows use of passwords in existing crypto schemes

- Gain:
  - Increases attackers computations
  - Prevents precomputation

- Cost:
  - Increased computation

- Limitation:
  - Strength of key still limited to strength of password
  - Don’t make it easy for attacker to mount offline dictionary attacks

pw || salt || 1 → H → H → ... → H → K1
INTERMISSION
Uses for Secure Random Numbers

**Cryptography**
- Keys
- Nonces, initial values (IVs), salts

**System Security**
- TCP Initial Sequence Numbers (ISNs)
- ASLR
- Stack Canaries
Where can we get secure random numbers?

**OSX/Linux**
- `cat /dev/urandom`
- `xxd -l 1024 -p /dev/urandom`
- `openssl rand 256 -hex`

**Intel HW RNG**
- **OSX**: `sysctl -a | grep RDRAND`
- **Linux**: `cat /proc/cpuinfo | grep rdrand`
Operating System Random Number Generators

System Events
- Keyboard Clicks
- Mouse Movements
- Hard Disk Event
- Network Packets
- Other Interrupts

RNG

Random Numbers
- Statistically Uniform
- Hard to predict
Random Numbers

Linux RNG

System Events → RNG → Random Numbers

Linux /dev/(u)random:

interrupt events

Interrupt Pool

disk events
keyboard events
mouse events
hardware RNGs

Input Pool

Random Pool → /dev/random

URandom Pool → /dev/urandom

Cryptographic hash
Random Numbers

RNG Failures
Predictable Output
Repeated Output
Outputs from a small range (not-statistically uniform)

Broken Windows RNG: [DGP 2007]
Broken Linux RNG: [GPR 2008], [LRSV 2012], [DPRVW 2013], [EZJSR 2014]
Factorable RSA Keys: [HDWH 2012]
Taiwan National IDs: [BCCHLS 2013]
Virtual Machine Snapshots

disk

Snapshot

Resumption
Security Problems with VM Resets

VM Reset Vulnerabilities [Ristenpart, Yilek 2010]

Initialization

App starts /dev/urandom

Read

Derives key

Snapshot

Use key

Firefox and Apache reused random values for TLS
Attacker can read previous TLS sessions, recover private keys from Apache
Linux RNG after VM Reset

Not-So-Random Numbers in Virtualized Linux

Experiment:
- Boot VM in Xen or VMware
- Capture snapshot
- Resume from snapshot, read from /dev/urandom

Repeat: 8 distinct snapshots
20 resumptions/snapshot
Linux RNG is **not** reset secure: 7/8 snapshots produce mostly identical outputs

<table>
<thead>
<tr>
<th>Reset 1</th>
<th>Reset 2</th>
<th>Reset 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E6DD331</td>
<td>1E6DD331</td>
<td>1E6DD331</td>
</tr>
<tr>
<td>8CC97112</td>
<td>8CC97112</td>
<td>8CC97112</td>
</tr>
<tr>
<td>2A2FA7DB</td>
<td>2A2FA7DB</td>
<td>2A2FA7DB</td>
</tr>
<tr>
<td>DBBF058C</td>
<td>DBBF058C</td>
<td>DBBF058C</td>
</tr>
<tr>
<td>26C334E7</td>
<td>26C334E7</td>
<td>26C334E7</td>
</tr>
<tr>
<td><strong>F17D2D20</strong></td>
<td><strong>F17D2D20</strong></td>
<td><strong>45C78AE0</strong></td>
</tr>
<tr>
<td><strong>CC10232E</strong></td>
<td><strong>CC10232E</strong></td>
<td><strong>E678DBB2</strong></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Reset insecurity and applications

Generate RSA key on resumption:

```
openssl genrsa
```

30 snapshots; 2 resets/snapshot  (ASLR Off)
• 27 trials produced identical private keys
• 3 trials produced unique private keys
Why does this happen?

Interrupts → Interrupt Pool
  |              |
  | disk events  |
  ↓            ↓
Input Pool → if (entropy estimate >= 64)
  ↓          ↓
Random Pool → /dev/random
  ↓          ↓
URandom Pool → /dev/urandom

if (count > 64 or elapsed time > 1s )

Buffering and thresholds prevent new inputs from impacting outputs.

Linux /dev/(u)random
What about other platforms?

 FreeBSD
/dev/random produces identical output stream
Up to 100 seconds after resumption

Microsoft Windows 7
Produces repeated outputs indefinitely
rand_s (stdlib)
CryptGenRandom (Win32)
RngCryptoServices (.NET)
RNG Recap

• RNGs are critical for security
  • Keys, nonces, etc

• Building good RNGs is hard

• OS provides a strong RNG
  • e.g.: /dev/urandom

• Intel CPUs provide an RNG
  • RDRAND instructions