



Open to Attack

Vulnerabilities of the Linux Random Number Generator

Zvi Guterman

Chief Technology Officer

with

Benny Pinkas

Tzachy Reinman



Zvi Guterman

CTO, Safend

Previously a chief architect in the IP infrastructure group for ECTEL (NASDAQ:ECTX) and an officer in the Israeli Defense Forces (IDF) Elite Intelligence unit.

Master's and Bachelor's degrees in Computer Science from the Israeli Institute of Technology. Ph.D. candidate at the Hebrew University of Jerusalem, focusing on security, network protocols, and software engineering.

Safend

Safend is a leading provider of innovative endpoint security solutions that protect against corporate data leakage and penetration via physical and wireless ports.

Safend Auditor and Safend Protector deliver complete visibility and granular control over all enterprise endpoints. Safend's robust, ultra-secure solutions are intuitive to manage, almost impossible to circumvent, and guarantee connectivity and productivity, without sacrificing security.

For more information, visit www.safend.com.

Pseudo-Random-Number-Generator (PRNG)

- Elementary and critical component in many cryptographic protocols
- Usually:
 - "... Alice picks **key K** at random ..."
 - In practice looks like

```
random.nextBytes(bytes);  
session_id = digest.digest(bytes);
```
 - Which is equal to

```
session_id = md5(get next 16 random bytes)
```

- If the PRNG is predictable the cryptosystem is not secure

Demonstrated in -

- Netscape SSL [GoldbergWagner 96]
<http://www.cs.berkeley.edu/~daw/papers/ddj-netscape.html>
- Apache session-id's [GutermanMalkhi 05]
http://www.guterman.net/publications/2005/02/hold_your_sessions_an_attack_o.html

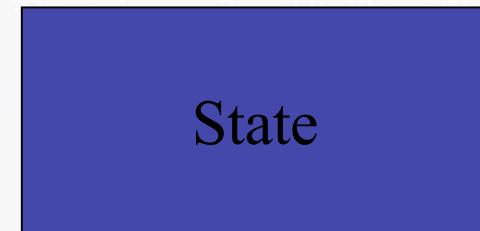
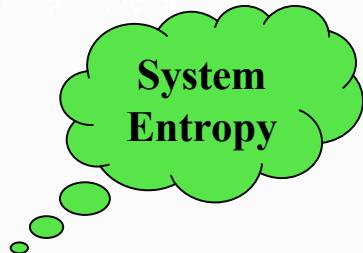
General PRNG Scheme



Properties:

1. Pseudo-randomness
Output bits are indistinguishable from uniform random stream
2. Forward security
Adversary revealing $\text{State}(t)$ does not learn anything about $\text{State}(t-1)$

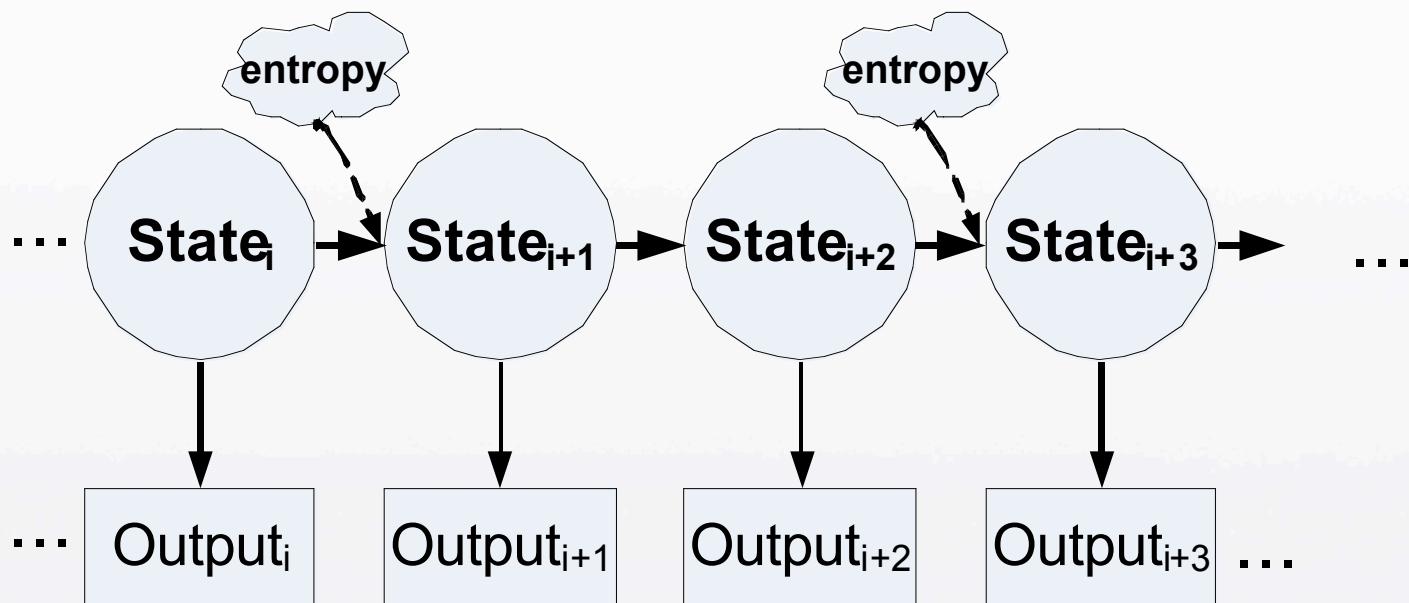
Entropy based PRNG (EPRNG)



Properties:

1. Pseudo-randomness
Output bits are indistinguishable from uniform random stream
2. Forward security
Adversary revealing $\text{State}(t)$ does not learn anything about $\text{State}(t-1)$
3. Backward security
Adversary revealing $\text{State}(t)$ does not learn anything about $\text{State}(t+k)$

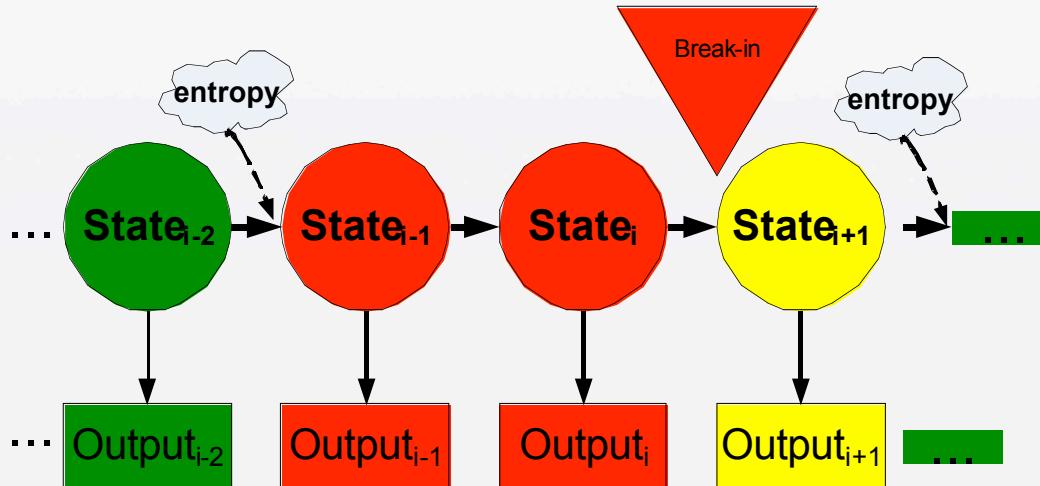
Entropy Based PRNG (2)



Our Research Results: Breaking the Linux PRNG

Outline of the process:

- Break-in and get the PRNG state
 - Buffer overflow
 - Physical access
 - ...
- Run our Forward security attack and learn past PRNG outputs
- Use the past outputs to study past PGP Keys, SSL keys, ...



Outline

- PRNG and Entropy PRNG
- Previous work
 - Entropy based PRNGs
 - OS based PRNGs
 - Barak and Halevi construction [CCS 05]
- Analysis of Linux PRNG
- Attacks
 - Denial of Service (DOS)
 - OpenWRT
 - Forward Security Cryptanalysis
- Entropy measurements
- Conclusions and recommendations



PRNGs: Previous Work



Entropy Based PRNGs

- Ad-hoc structures
- No proved theory

- Examples
 - Yarrow [Kelsey, Schneier, Ferguson 99]
 - CryptLib [Gutmann 98]
 - PGP [Zimmermann 95]
 - RSAREF [RSA Labs 94]
 - X9.17 [NIST 92]

OS Based PRNG's

-
- FreeBSD [Murray 02]
 - Yarrow with AES

OS Based PRNG's (2)

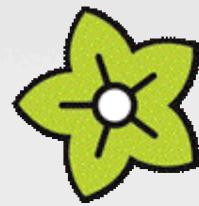


➤ Windows

- Kernel based CryptGenRandom
- Entropy based
- Proprietary unpublished algorithm
- US patent 5,778,069

Barak and Halevy CCS '05

- Provide a rigorous definition of entropy based PRNGs
- Describe a *generic* construction of entropy based PRNG from a *pseudo-random generator* and an *extractor*.
 - Structure:
 - Part of the PRNG output is used as the new state.
 - System entropy is input to the extractor.
 - The extractor's output is xored to the state.
 - Proved security as long as underlying blocks are secure
 - Forward + Backward + Pseudo-randomness
- Example:
 - Use AES as the PRNG, HMAC-SHA1 as the extractor.



safend
Securing Your Endpoints

Analysis of Linux PRNG

This is a report

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

This is a report

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

December

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

This is a report

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

This is a report

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

December

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

This is a report

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

This is a report

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

December

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

This is a report

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

December

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

This is a report

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222

December

12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222



Linux PRNG (LRNG)

- Implemented in the kernel
- Entropy based PRNG
- Started by Theodore Ts'o in 1994
- Engineering
 - Complex structure
 - Hundreds of patches to date
 - Changes on a weekly base
- Used by many applications
 - TCP, PGP, SSL, S/MIME, ...



Two Interfaces

- Kernel interface – *get_random_bytes (non-blocking)*
- User interfaces –
 - /dev/random (blocking)*
 - /dev/urandom (non-blocking)*

/dev/random - returns random bytes only when sufficient amount of entropy has been collected. If there is no entropy to produce the requested number of bytes, /dev/random blocks until more entropy can be obtained.

/dev/urandom - returns bytes regardless of the amount of entropy available. It does not block on a read request due to lack of entropy.

<http://bama.ua.edu/cgi-bin/man-cgi?urandom+7D>

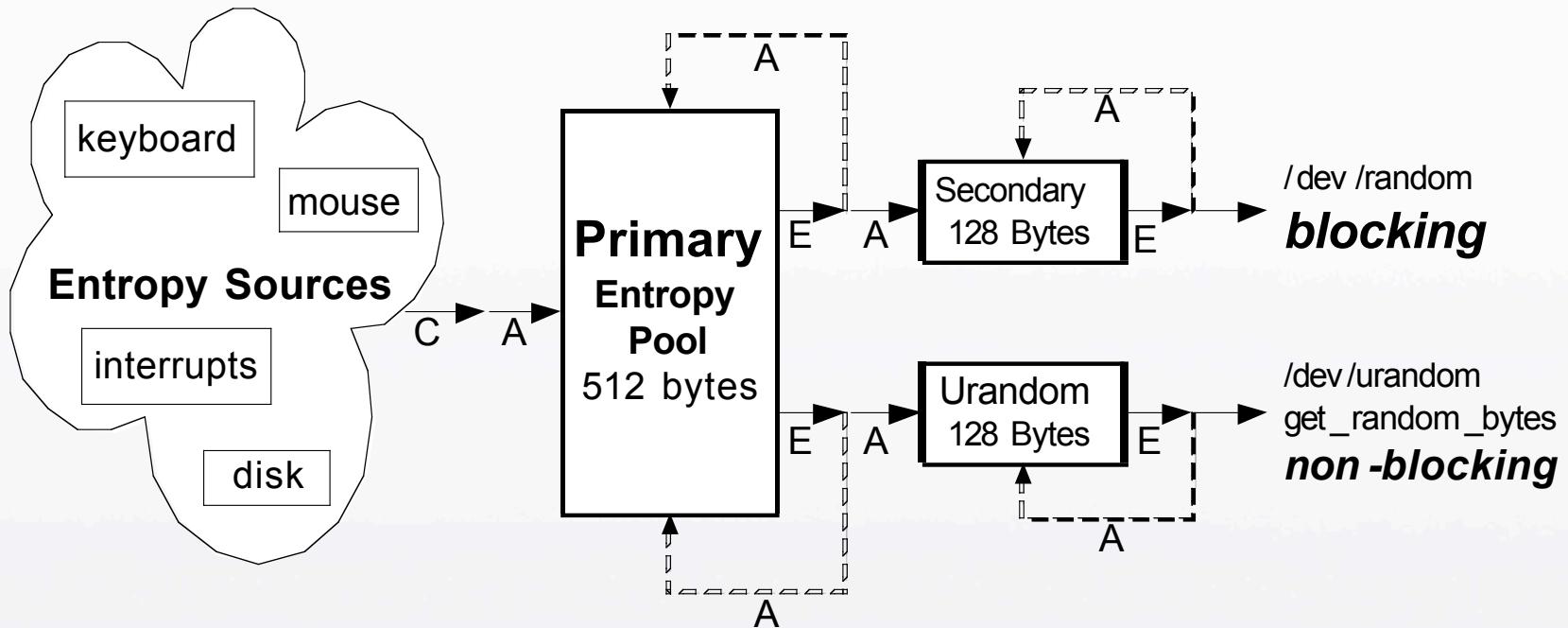
Entropy Estimation

- A counter estimates the physical entropy in the LRNG
- Increased on entropy addition
- Decreased on data extraction
- *blocking* and *non-blocking* interfaces
 - *Blocking* interface does not provide output when entropy estimation reaches zero
 - *Non-blocking* interface always provides output
 - Blocking interface is “considered more secure”
- No guaranteed relation between the estimation and output bits

On Reverse Engineering

- The Linux PRNG is part of the Linux kernel and hence an open source
- The entire code is 2500 lines written in C
- However
 - Kernel code – measurement interference
 - Hundreds of code patches over the years
 - Very unclear, complex structure
- Our tools
 - Static analysis
 - Kernel modification
- We implemented and confirmed our findings with a [user mode simulator](#) (<http://www.cs.huji.ac.il/~reinman/>)

LRNG Structure



C – entropy collection

A – entropy addition

E – data extraction

Entropy Collection

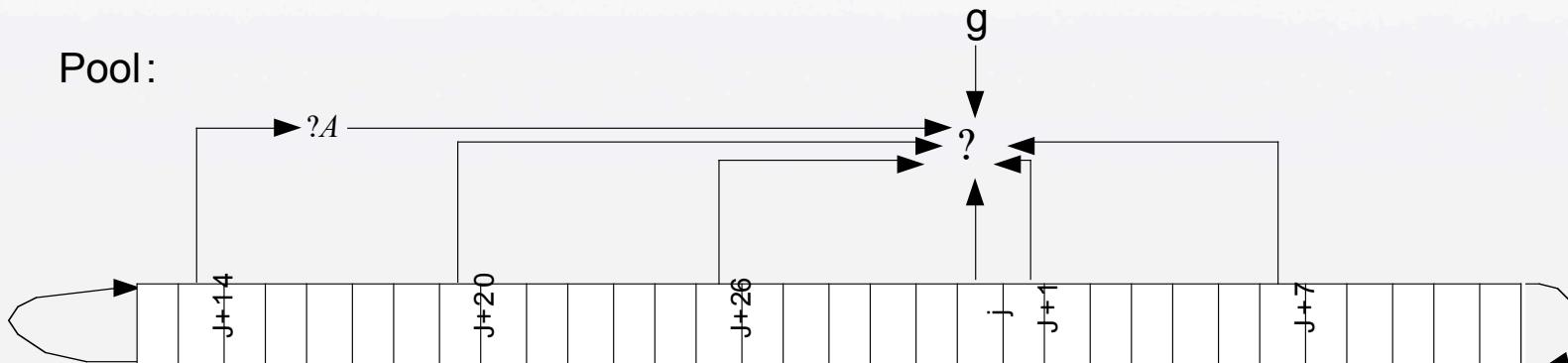
- Asynchronous
- Events are represented by two 32-bit words
 - Event type
 - E.g., mouse press, keyboard value
 - Event time in milliseconds from up time
- Bad news:
 - Actual entropy in every event is very limited
 - e.g., a common PC with single IDE drive has a fixed event type for I/O
- Good news:
 - There are many of these ...

Entropy Addition

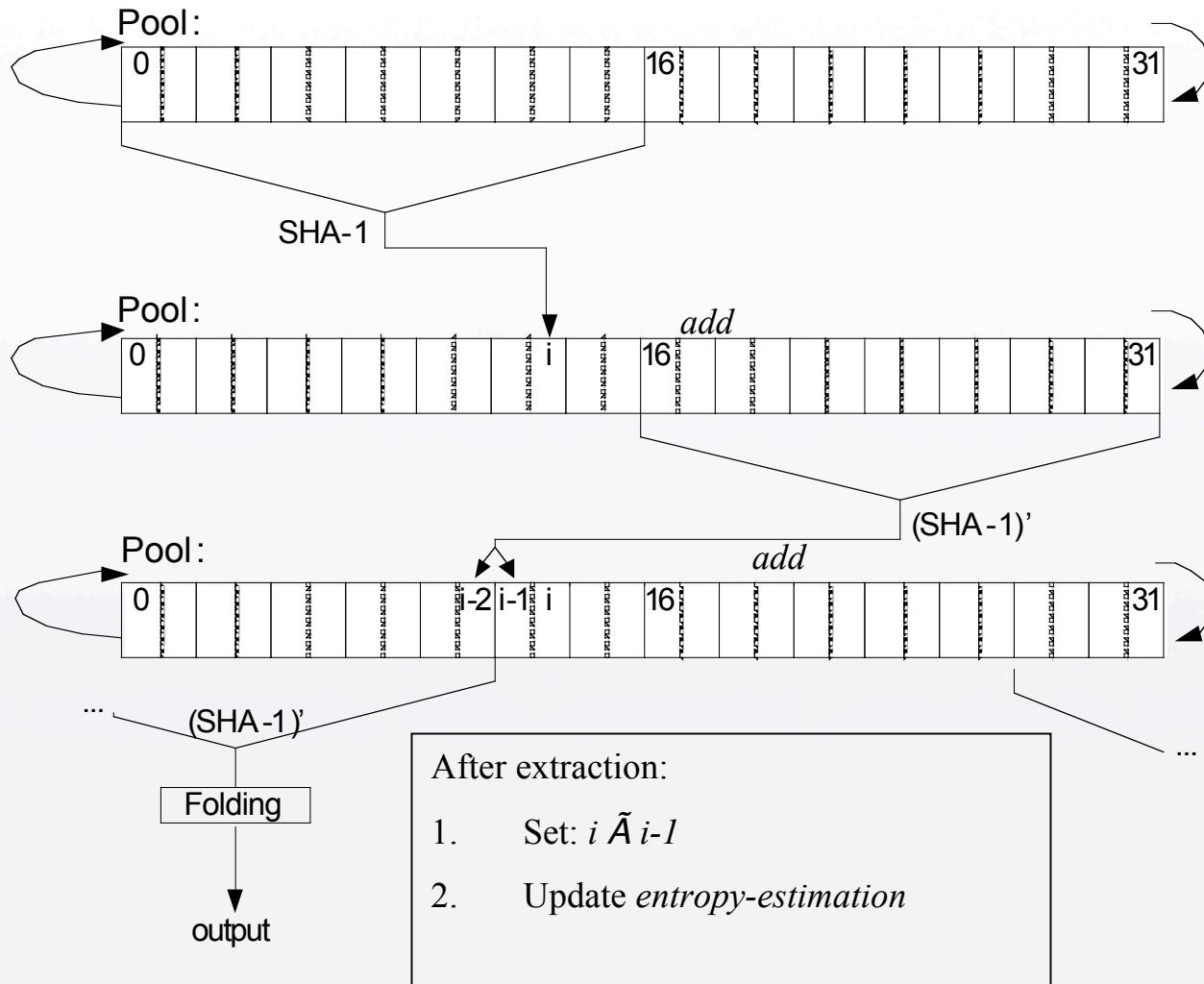
- Cyclic pool, generalization of LFSR
- Different polynomial for each pool size
- A is a known matrix
- Polynomial:
 $X^{32}+X^{26}+X^{20}+X^{14}+X^7+X+1$
- Addition algorithm:
 g – input, j – current pool position

```

Algorithm add(pool , j , g):
    pool [ j ]:= g
    ⊕ pool [ j mod m]
    ⊕ pool [ ( j +26) mod m]
    ⊕ pool [ ( j +20) mod m]
    ⊕ pool [ ( j +14) mod m]*A
    ⊕ pool [ ( j +7) mod m]
    ⊕ pool [ ( j +1) mod m]
```



Extraction





Attacks

- Denial of Service (DOS)
- OpenWRT
- Forward Security Cryptanalysis



This is a report
12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222
December
1255 km 125425366 N/y 252235 121
00000 0123554 223 25 255

This is a report
12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222
December
1255 km 125425366 N/y 252235 121
00000 0123554 223 25 255

This is a report
12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222
December
1255 km 125425366 N/y 252235 121
00000 0123554 223 25 255

This is a report
12355 9658 225225 665ykm 225253
6656 222 ykm 225253 6656 222
December
1255 km 125425366 N/y 252235 121
00000 0123554 223 25 255

Denial of Service Attack

- LRNG does not limit usage of /dev/random
- /dev/random blocks whenever entropy estimation reaches zero (~ consumption > entropy)
- Internal Attack
 - Read and drop bytes from /dev/random
 - All others consumers will starve
 - Because /dev/random and /dev/urandom share the primary pool reading from /dev/urandom will also cause starvation
- External Attack
 - /dev/urandom is also used for TCP sequence numbers (to avoid spoofing attacks)
 - But /dev/urandom also shares the primary pool with /dev/random
 - Hence, remote attack:
 - Generate as many TCP connections and empty the primary pool
 - Same result: Starvation
- Solution: Quota

OpenWRT Platform

- Linux distribution for wireless routers
- Implements SSL termination, SSH server, wireless encryption,
...
- No hard-drive entropy
- No state saving between reboots!
 - Initial state depends on boot time only
 - Very weak



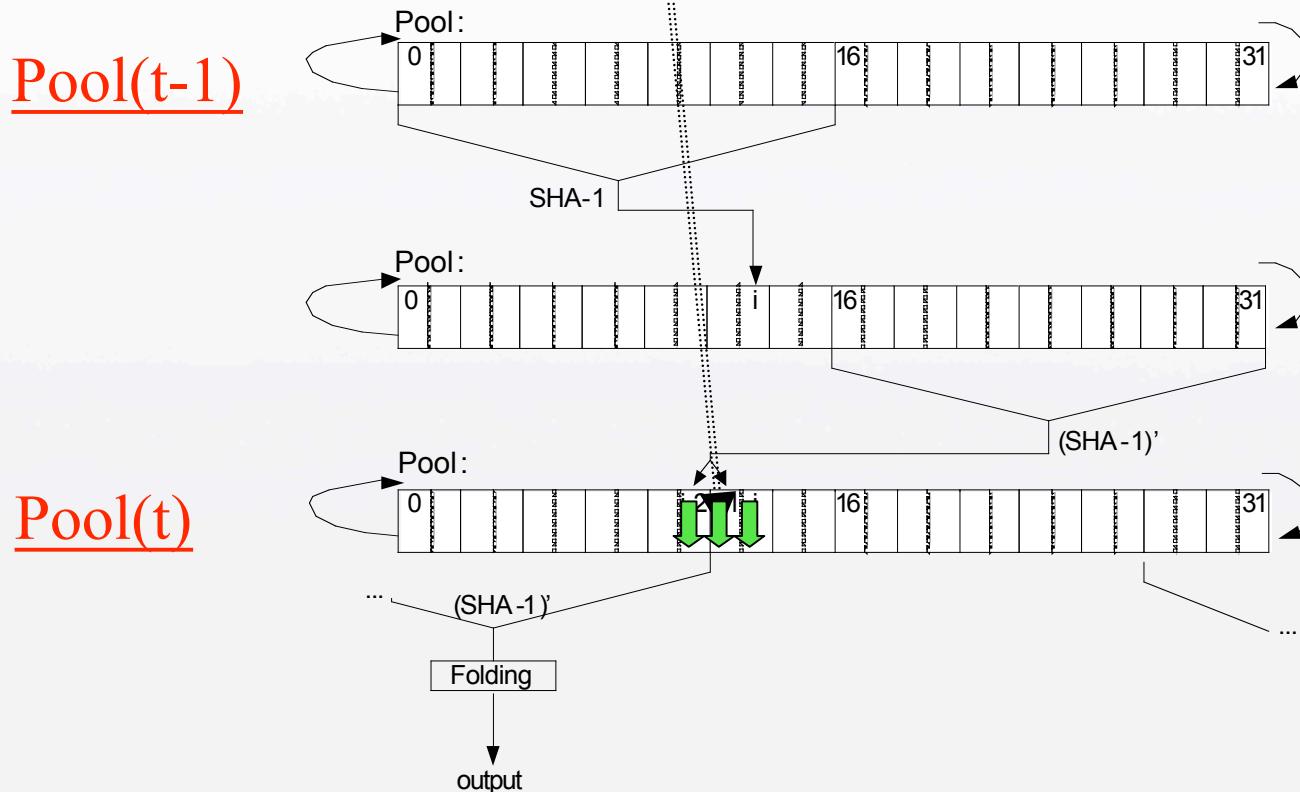
Cryptanalysis

- Based on the entropy extraction process we were able to mount an attack on the LRNG forward security
 - Input: state in stage n
 - Output: state in stage n-1
 output in stage n-1
 - Limitation
 - Assuming that no entropy was added during the extraction
- Brute Force
 - 2^{1024} computations per stage (32 x 4 x 8 bits)

Cryptanalysis (2)

➤ Generic Attack

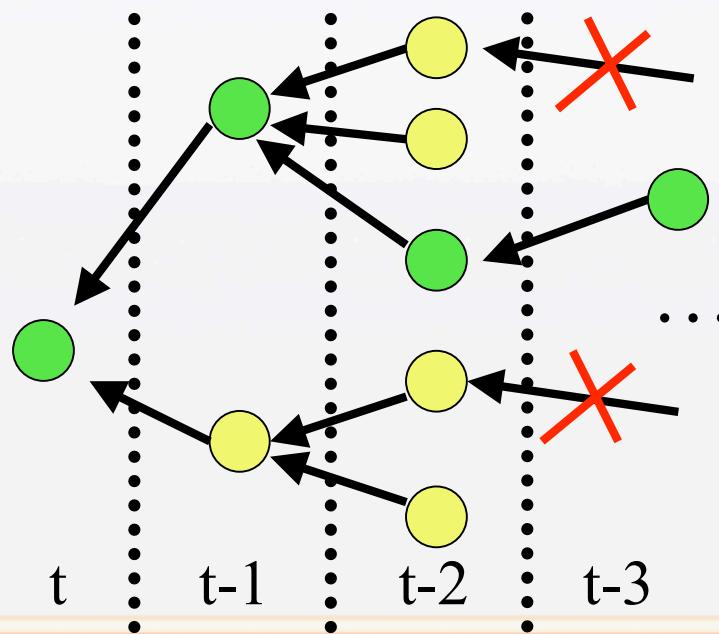
- Each stage changes three words ($i, i-1, i-2$)
 - Given $\text{Pool}(t)$, check all 2^{96} possibilities for $\text{Pool}(t-1)$



Assimilation problem

- Searching for Pool(t-1) we get the real result together with false positives
- Probability of having k false positive is:

$$\binom{n}{k} n^{-k} (1 - 1/n)^{n-k} \approx e^{-1}/k!$$



Assimilation problem (2)

- Let d_i be the number of false positives at time $t-i$ ($d_0=0$)
- Let's compute $E(d_1)$:

$$E(d_1) = \sum_{k=1}^n k e^{-1} / k! = e^{-1} \sum_{k=0}^{n-1} 1/k! = e^{-1} \cdot e = 1.$$

- And in the general case we get:

$$E(d_i | d_{i-1} = c) = (c + 1) E(d_i | d_{i-1} = 1) = c + 1$$

Assimilation problem (3)

- We define a new variable $z_i = d_i - i$
- A martingale is a sequence of random variables X_0, X_1, X_2, \dots which satisfies the relation:

$$E(X_i | X_{i-1}, \dots, X_0) = X_{i-1}.$$

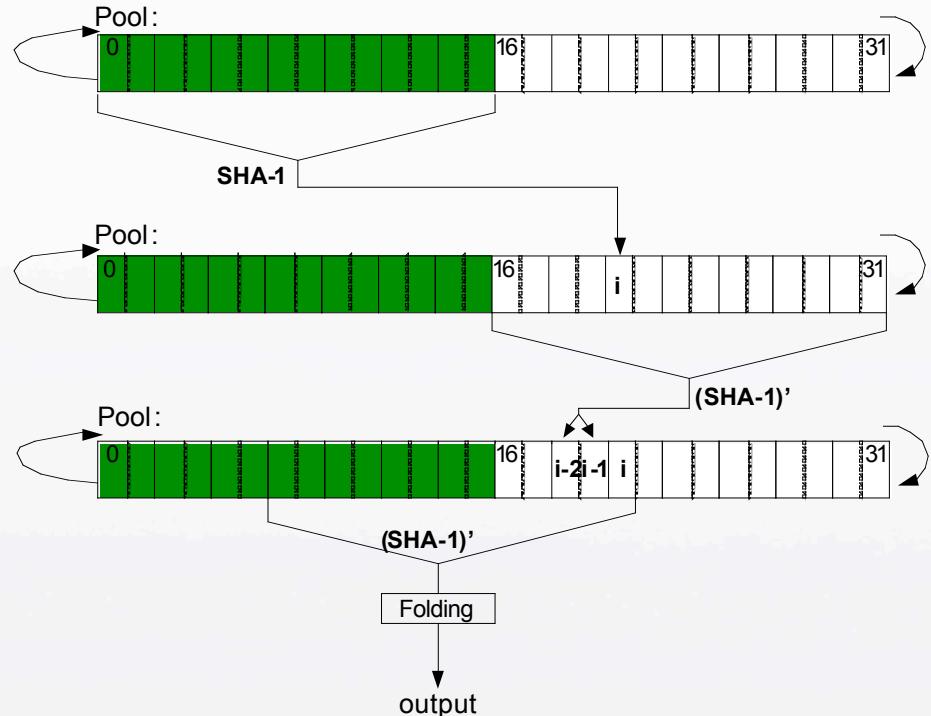
- Martingale property
 $E(X_i) = E(X_0)$
- Z_i is a martingale
- Hence, we get $E(Z_i) = E(Z_0) = 0$
- Therefore, $E(d_i) = i$
 - With very small variance

Assimilation problem (4)

- Not really an issue because the growth rate is linear with the number of reverse steps
- Final candidate validation should be done using higher protocol

Cryptanalysis (3)

- In certain cases (18 out of 32) we can mount a 2^{64} attack
- Depends on current pointer position
- Example demonstrates one such index where we only need 2^{64} computations
- We get the same assimilation problem



Entropy Measurements

- Each addition is made of two 32-bit words:
 - Event type (mouse, keyboard, HD, interrupts)
 - Event time
- Maximal unknown bits per event *type*

Keyboard	Mouse	Hard Drive	Interrupts
8	12	3	4

- Measuring entropy of HD event *times* over 140,000 events on an idle server resulted with entropy of $H:=1.028$ bits per event

Conclusions

- We presented an analysis of the Linux PRNG
- Limitations of our analysis
 - Unique hardware
 - e.g., SMP
 - Different distributions
 - It's an open source ...
- 2^{96} and 2^{64} Cryptanalysis of the LRNG forward security
- Entropy measurements
- OpenWRT use case

Recommendations

- Fixing the Linux PRNG
 - Solve the forward security problem
 - Replace the Entropy estimation with a different criteria (based on time or size)
- Implementing random-bits quota
 - Avoiding denial of service attacks
- Adopting the Barak-Halevi construction
 - Proved security
 - Simple to implement

Recommendations (2)

➤ Security engineering

- Open source is not a synonym for well documented or secured systems
- Better framework needed
- Security related code must be treated different
- Better documentation
- Better validation process

Questions?

Contact information: **Zvi Guterman**

zvi@safend.com

www.safend.com

Download recent research results at:
guterman.net/blog/