Server-side approaches to clickjacking detection

Brad Hill, PayPal

WebAppSec WG F2F, 3 April 2012

Drawbacks of X-Frame-Options

- IFRAMES desirable for many key clickjacking attack cases. (Like, Pay, Follow, +1) Users want in-context information without disclosure to embedding origin

- Allow-From doesn't help – adversary is potentially the same as the "legitimate" origin

- Also doesn't stop pop-under-and-close attacks

Drawbacks of client-enforced screenshot approach

- Incomplete coverage of attack scenarios
  - Fake mouse cursor, attention stealing attacks
- False positives
- User-interaction to resolve false positives
- Low deployment rates

Server side approaches?

- What can we do today without user-agent support?

- Can we profitably combine these techniques with user-agent mechanisms?

Adaptive UI Randomization

- Clickjacking attacks are still subject to the read restrictions of the same-origin policy
- Attack setup relies on a consistent layout of the victim page
- What if we randomize the location of the button?

Naïve Randomization

- Attacker can send multiple clicks to possible locations
- Attacker can profit even at a small success rate
- Few interfaces allow randomization among a large number of locations without creating a very poor user experience

Refining Randomization

- Among a set of possible locations for a randomized placement:
  - Record missed clicks (to locations where the button is not)
  - Record just the first click, hit or miss
  - Group first-click statistics by the target of the action (“bucketize”)

Detect Clickjack	*Detect Clickjack*	Pay Now

“Bucketizing”

- Associate possible clickjacking targets with a beneficiary or beneficiaries
- Perform back-end fraud analysis based on these buckets

- Examples:
  - “pay” -> payee
  - “like, +1, etc.” -> social graph node

Look at first-click miss rates, bucket-by-bucket

- A given interface will have a discoverable natural rate of missed clicks, but it should be small

- If clickjacking attempts are made on that interface, miss rate will be (1 - 1/N) where N is the number of possible randomized placements

  (also works for pop-under-and-close attacks)

Campaign detection

- Can’t distinguish individual clickjacking attempts

- But a campaign of clickjacking will quickly show up – the missed click rate for that bucket will rise above the natural missed click rate

Sensitivity of Detection

$$100(M + 2\sigma) = M(100 - x) + (x \; * \; (1 - 1/N))$$

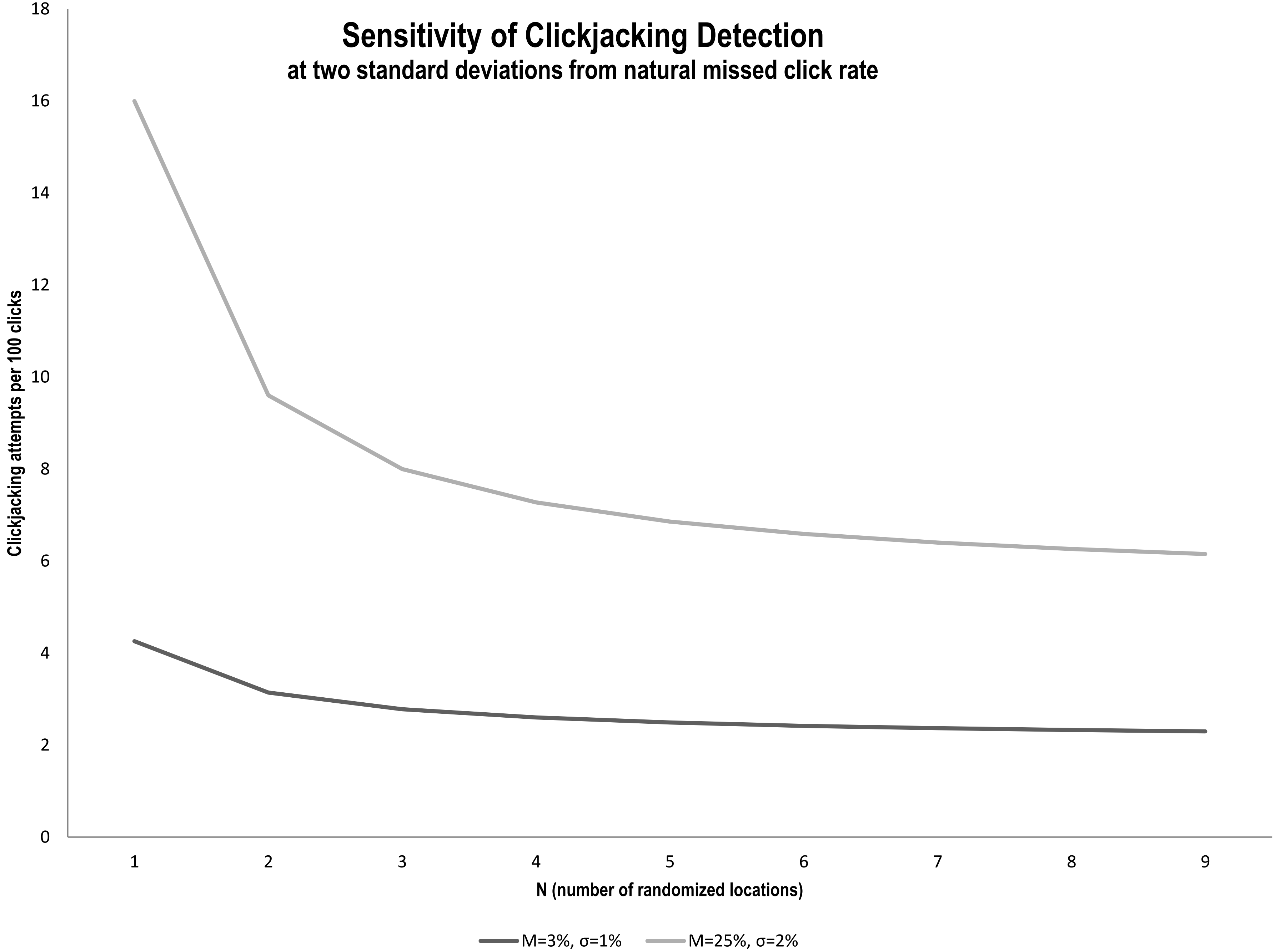
Where:

σ = standard deviation for natural missed click distribution

M = natural miss rate

N = number of randomized locations

x = clickjacking attempts per 100 clicks



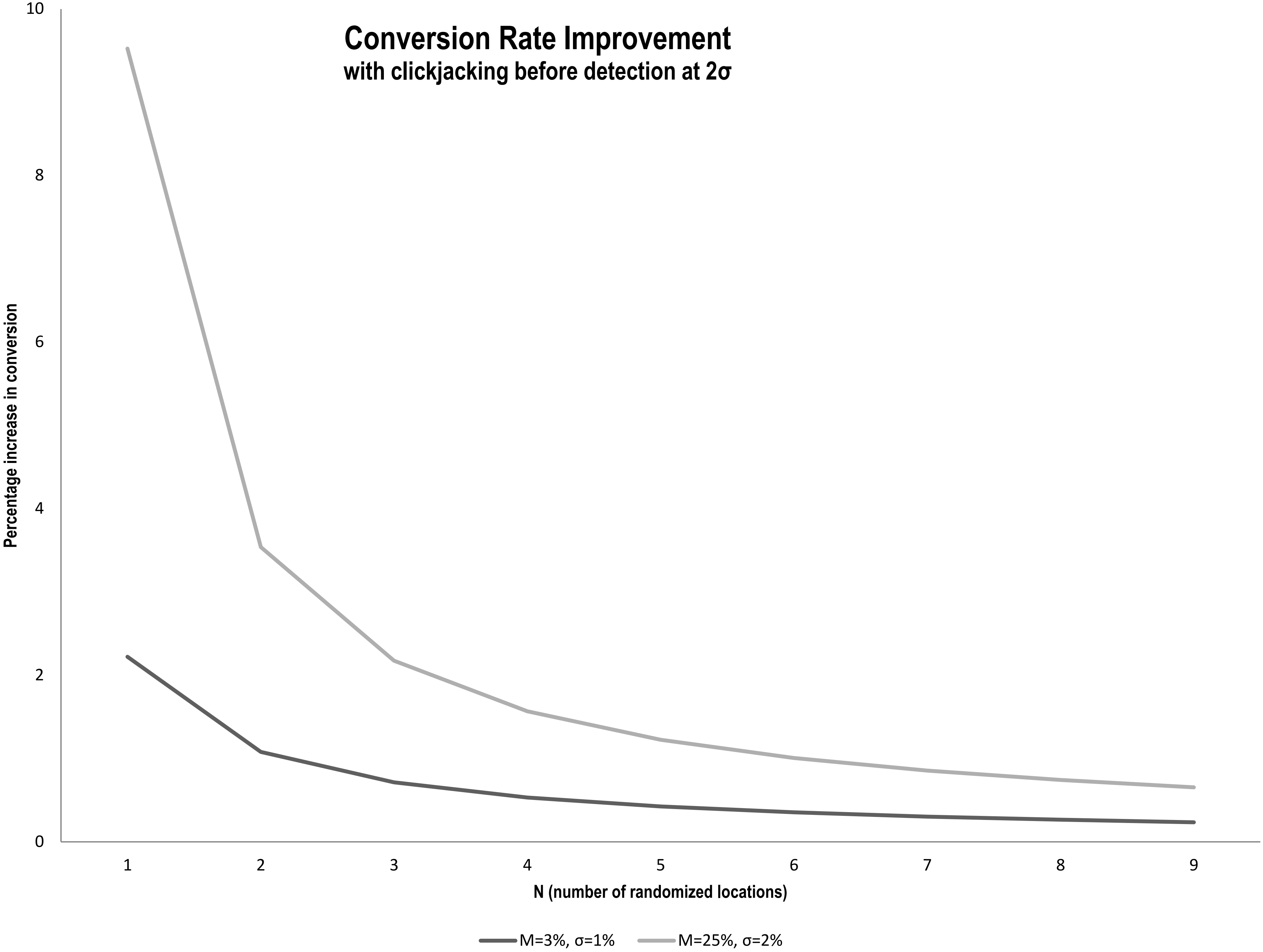
Sensitivity of Clickjacking Detection
at two standard deviations from natural missed click rate
Clickjacking attempts per 100 clicks
0
2
4
6
8
10
12
14
16
18
1
2
3
4
5
6
7
8
9
N (number of randomized locations)
M=3%, σ=1%
M=25%, σ=2%

Pretty good…

- And it's better than it looks.

- As N increases, the chances of the success of each attempt goes down.

- Increase in natural conversion rate possible before detection is even lower:



Conversion Rate Improvement
with clickjacking before detection at 2σ
Percentage increase in conversion
0
2
4
6
8
10
1
2
3
4
5
6
7
8
9
N (number of randomized locations)
M=3%, σ=1%
M=25%, σ=2%

Results

- Randomizing among as few as 3 locations, if the natural missed click rate is low, we can put the attacker at risk of detection if they attempt to increase their natural conversion rate as little as 1% through clickjacking.

Adaptive Response

- What if rivals mount clickjacking campaigns against their competition to cause a DoS
- Instead of turning off service, can trigger a switch to a functional, if less optimal, interface that is more clickjacking resistant
  - Popup in dedicated context with X-Frame-Options
  - Add a CAPTCHA or re-verify credentials
  - These responses can be completely automated, and combined with manual investigation according to standard anti-fraud practices

Weaknesses

- Doesn’t work for complex UIs with lots of buttons (webmail, etc) or no room for randomization (“NASCAR” interfaces)
- Doesn’t work where bucketization isn’t possible (privacy attacks like Flash camera settings)
- Needs sophisticated back-end analysis and fraud response processes
- Can’t stop targeted or small-scale attacks
- Attacker can try to pollute the natural missed click rate of their own or a large population of buckets at low cost

Attacks: The Sleepy Frog

+

=

Click the Sleepy Frog to WIN!

Combining with Client-Side Screenshot Approaches

- “Sleepy Frog” attack easily detected by screenshot approaches

- UI Randomization effective against attention stealing and phantom cursor attacks

Combining with Client-Side Screenshot Approaches

- Add a feedback loop to apply statistical approach to client-side enforcement
- Resource advertises a feedback URI for suspected clickjacking
- Front-end screenshot technology allows clicks to go through, but reports to the target server that it suspects a clickjacking attack

Advantages:

- False positive problem disappears
  - Each site can find its own rate of false positives and use back-end fraud response processes to deal with suspected clickjacking
  - No need to pop-up a confusing dialog to the user
- Small install base can help protect everyone
  - Suspected clickjacking from a small install base of user-agent support can add good evidence to buckets
  - Detecting and disabling attackers protects even users that can’t detect or prevent the attacks

Conclusions

- Randomization isn't for everyone
  - High cost, only usable in certain UIs
  - *But* the primary attack targets are in its "sweet spot"
- Combines well with client-side techniques
- A reporting loop + back-end fraud analysis approach can remove some weaknesses of heuristic client-side techniques, even if no randomization is applied