Hey You, Get Off of My Cloud

Application-Level Denial of Service Attacks and Defenses

Bryan Sullivan, Senior Security Researcher, Adobe Systems

DoS is the new EoP.

Exploitable privilege escalations are getting rare

"It's harder to find exploitable bugs now…it used to be that if you found ten bugs, nine of them would be exploitable."

- Charlie Miller

- NVD shows a 20% decrease in reported vulnerabilities last year

- Platform defenses (ASLR, NX, stack canaries) are working
- Secure development methodologies (SDL, SPLC) are working

Political motivation

VS

VS

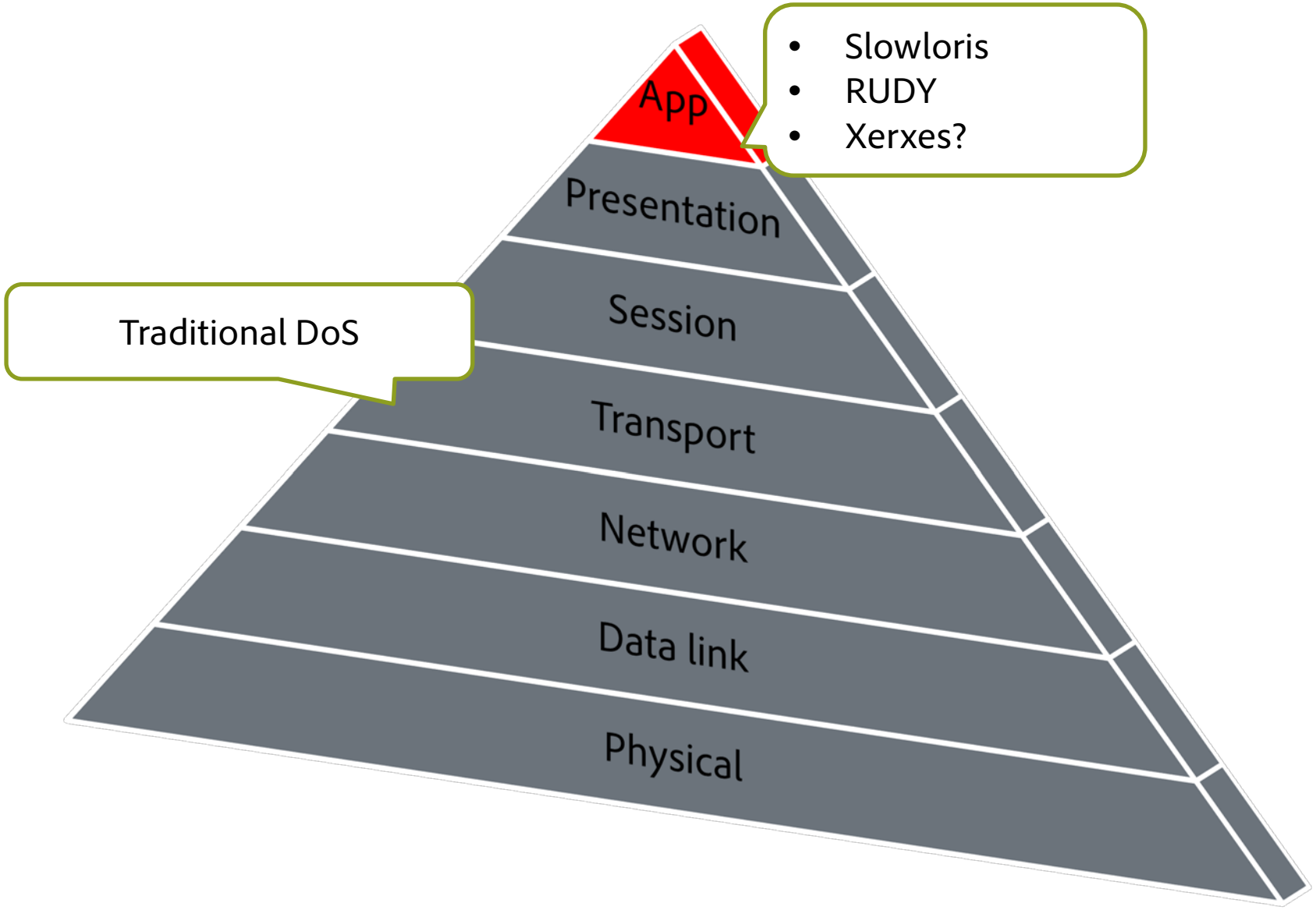
Economic motivation

- For *aaS targets, economically-motivated DoS is less about crashing systems, more about bankrupting service owners

- Typical cloud computing costs:
  - US $0.15/instance-hour for processing time
  - US $0.15/GB-month for storage
  - US $0.01/10K-transactions
  - US $0.10/GB incoming bandwidth
  - US $0.15/GB outgoing bandwidth

- Sounds small but adds up quickly…

Application-level (OSI Layer 7+) DoS



Characteristics of application-level DoS vulnerabilities

Traditional DoS/DDoS

- Very easy to find
- Fairly even in terms of attacker resources vs target impact
- Solve the problem with IPS/QoS firewall

Application-level DoS

- Difficult to find
- Extremely asymmetric in terms of attacker effort vs impact
- Solve the problem with code changes/redesign

Application DoS Example 1

Synchronization

Intentionally induced deadlocks and livelocks

- Pseudocode for a banking SaaS application:

```
AcquireLock(payee_account)
AcquireLock(payer_account)
DebitPayer
CreditPayee
ReleaseLock(payer_account)
ReleaseLock(payee_account)
```

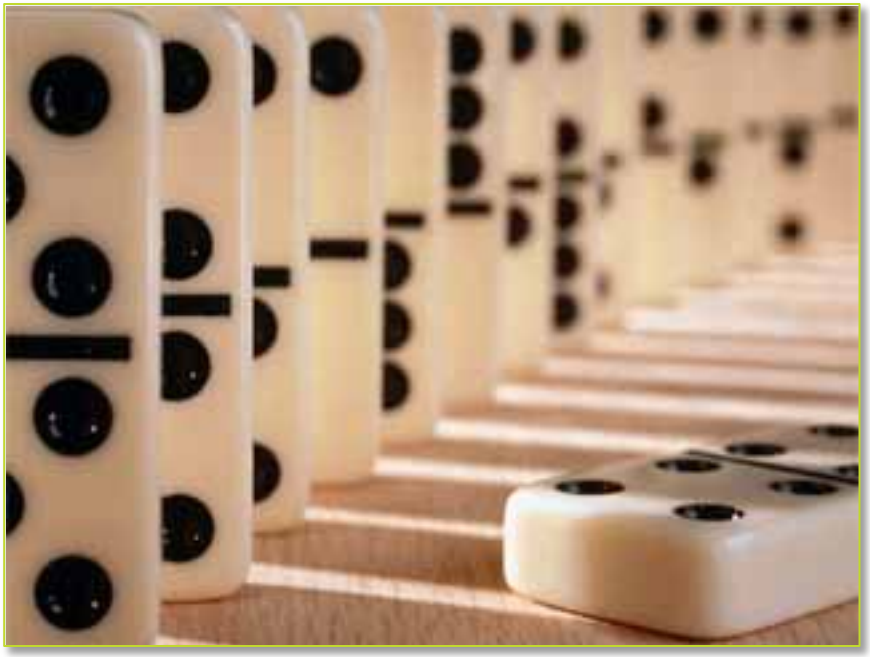
Alice and Bob collude to DoS the bank

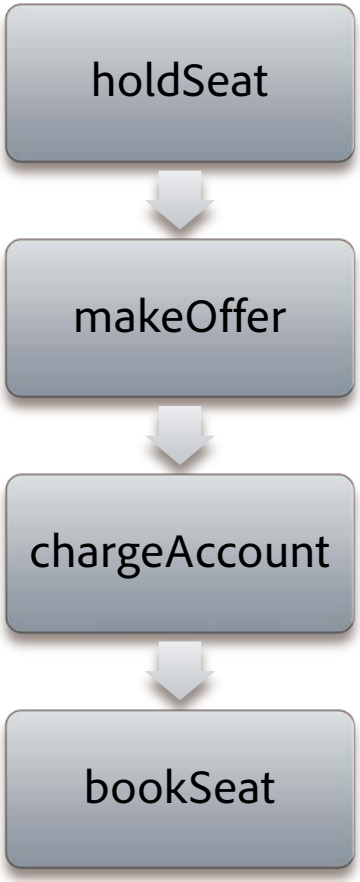
Alice pays Bob $50	Bob pays Alice $50
AcquireLock(Alice)	AcquireLock(Bob)
AcquireLock(Bob)	**AcquireLock(Alice)**
Debit Alice $50	Debit Bob $50
Credit Bob $50	Credit Alice $50
ReleaseLock(Bob)	ReleaseLock(Alice)
ReleaseLock(Alice)	ReleaseLock(Bob)

DoS'ing a plane

- Ajax Security talk @ BlackHat, RSA
- Hacker Vacations site

Mitigating the attack: Avoid the domino effect





Application DoS Example 2

PHP 2.22505 Infinite Loop

String-to-double conversion

- Vulnerability occurs when a string is converted to a floating point value:

```
<?php $number = (float) $_GET['number']; ?>
```

- Code execution path leads to the C function strtod()

Demonstration

PHP “Number of the Beast”

Mitigating the attack

- Upgrade to 5.2.17 or 5.3.5
- Or recompile with –ffloat-store compiler flag
- Or blacklist-validate* for the malicious string

```
if (strstr(str_replace('.','',serialize($_REQUEST)), '2225073858507201 1'))
{
  // request is malicious, abort processing
}
```

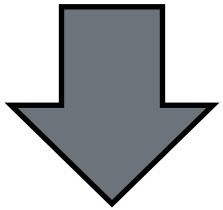
* Yes, I know you're not supposed to do this. But for now, it's the best alternative.

Application DoS Example 3

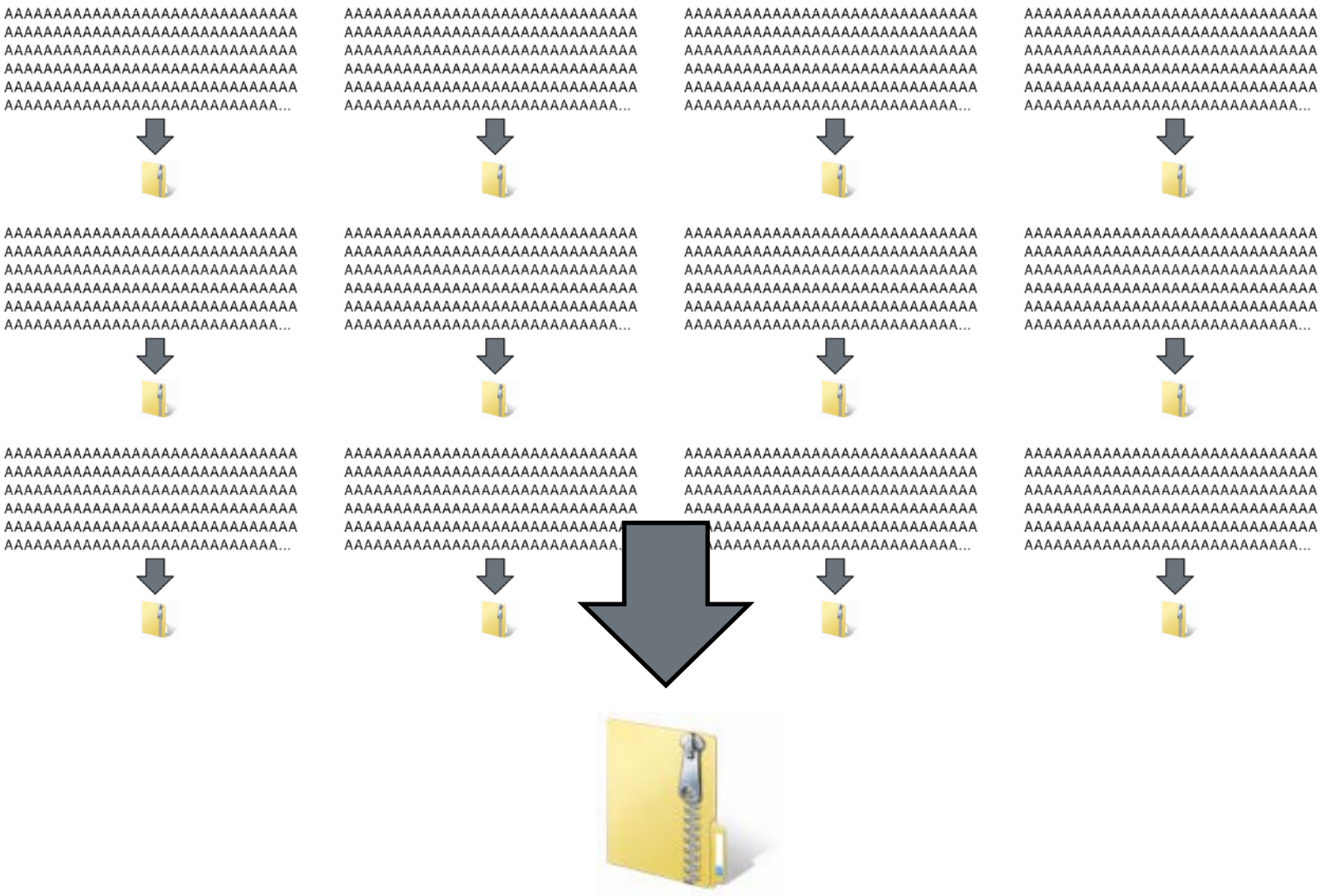
Decompression (Zip Bombs)

Zip compresses text very efficiently

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...

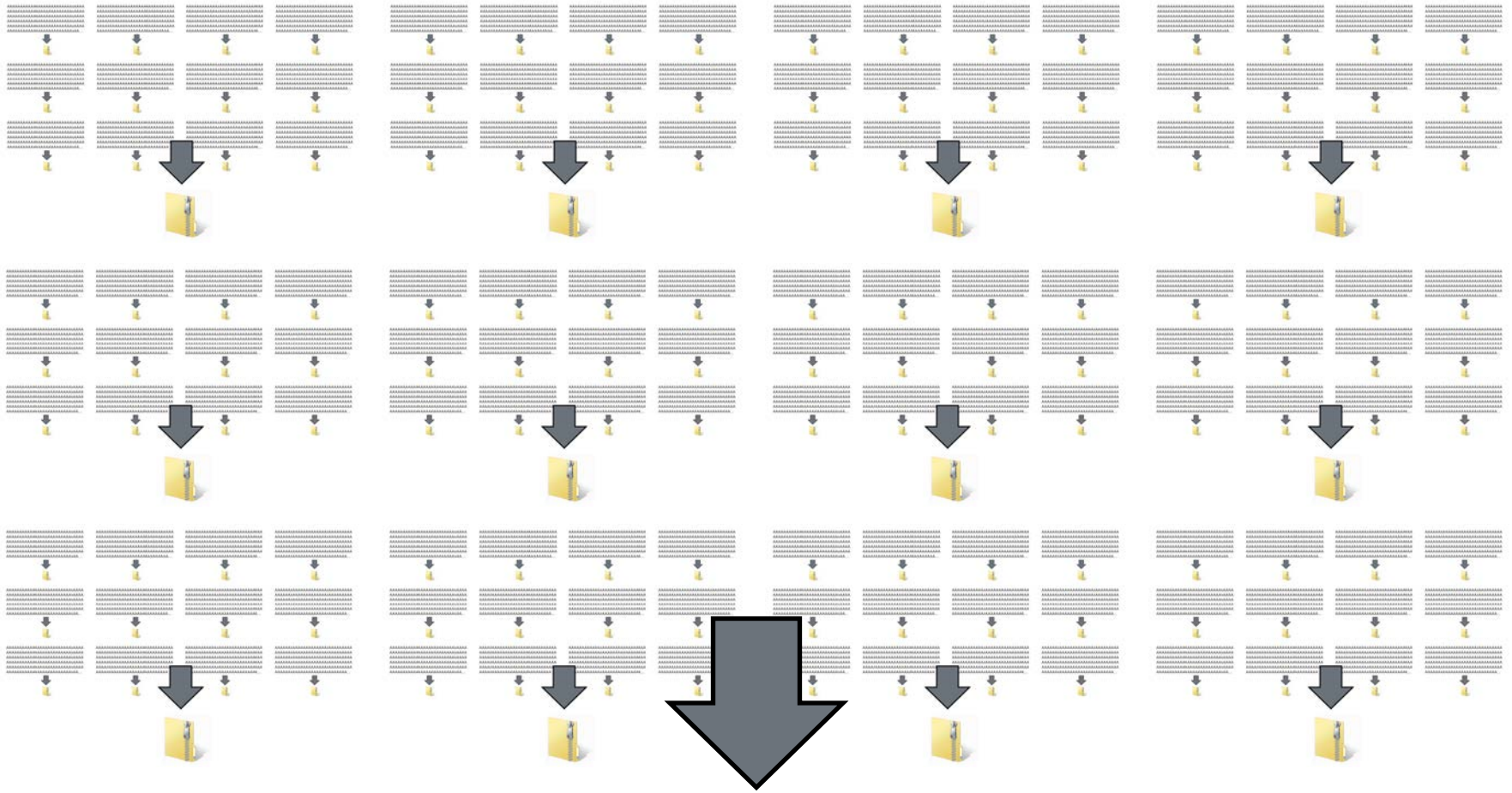


Nesting Zips



Adobe

42.zip

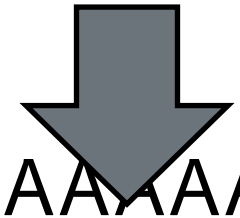


42.zip

Adobe

Decompression explosion

42.zip



AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA…

4.2 petabytes of A's!

Application DoS Example 4

XML Entity Attacks

XML entities

- Like macros for XML documents

```
<!DOCTYPE employees [
<!ENTITY companyname "Adobe Systems, Inc.">
]>
<employees>
  <employee>Bryan S, &companyname;</employee>
  <employee>Lucas N, &companyname;</employee>
  <employee>Peleus U, &companyname;</employee>
</employees>
```

Nesting entities

- You can nest entities, too

```
<!DOCTYPE employees [
<!ENTITY companyname "Adobe Systems, Inc.">
<!ENTITY divisionname "Adobe Secure Software Engineering Team,
                       &companyname;">
]>
<employees>
  <employee>Bryan S, &divisionname;</employee>
  <employee>Lucas N, &divisionname;</employee>
  <employee>Peleus U, &divisionname;</employee>
</employees>
```

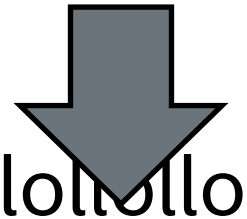
Exponential Entity Expansion attack

- Aka “The Billion Laughs Attack”

```
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
…
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

Expansion explosion

<lolz>&lol9;<lolz>



lollollollollollollollollollollollollollollollollollo
llollollollollollollollollollollollollollollollollol
ollollollollollollollollollollollollollollollollollol
lollollollollollollollollollollollollollollollollollo
llollollollollollollollollollollollollollollollollollo
llollollollollollollollollollollollollollollollollol...

3GB of LOLs

Demonstration

Billion Laughs Attack

Infinite entity recursion?

```
<!DOCTYPE lolz [
<!ENTITY lol1 “&lol2;">
<!ENTITY lol2 "&lol1;">
]>
<lolz>&lol1;</lolz>
```

- Fortunately, not legal!

Quadratic Entity Blowup

```
<!DOCTYPE kaboom [
<!ENTITY a "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...">
]>
<kaboom>&a;&a;&a;&a;&a;&a;&a;&a;&a;&a;&a;...</kaboom>
```

External entity resolution attacks

```
<!ENTITY stockprice SYSTEM  "http://www.mysite.cxx/stockticker.ashx">
```

- Some attack ideas:
  - Infinite delay
  - Infinite streaming data pipe
  - Very large file downloads
  - Intranet redirection

Application DoS Example 5

Regular Expression DoS

Expert advice?

- "…the developer should be able to define a very strong validation pattern, usually based on regular expressions, for validating [user] input."
  - OWASP SQL Injection Prevention Cheat Sheet
- "Regular expressions are a good way to validate text fields such as names, addresses, phone numbers, and other user information."
  - MSDN Patterns & Practices
- "Regex is a perfect tool for input validation."
  - Bryan Sullivan, *Ajax Security*

Backtracking (NFA) regular expression engines

- Example 1: ^\d+$
- Evaluate this pattern against test input: 123456X

123456	[no match, backtrack]
12345	[no match, backtrack]
1234	[no match, backtrack]
123	[no match, backtrack]
12	[no match, backtrack]
1	[no match]

- Fails in 13 steps (including backtracks)
- Operates in O(n) time

Backtracking (NFA) regular expression engines continued

- Example 2: ^(\d+)+$
- Evaluate this pattern against test input: 123456X

```
123456      [no match, backtrack]
12345
123456      [no match, backtrack]
12345
123456      [no match, backtrack]
...
```

- Fails in 223 steps
- Operates in $O(2^n)$ time

Demonstration

ReDoS

More irony…

- "Just as we perform whitelist input validation on the server for security purposes, developers must perform client-side validation to ensure security of their offline applications."
  - Ajax Security

Detecting vulnerable regexes

- Look for:
  - Grouping expressions containing repetition that is itself repeated
  - Groups containing alternation where the alternate subexpressions overlap each other
- This is harder than it sounds, and it doesn't sound easy

Regex testing strategies

- Test each regex clause with a large, valid sample
  - aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa@test.com
  - bryan@aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.com
- Add one invalid character to the end of the sample
  - aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa@test.com!
  - bryan@aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.com!
- Why does this work?

Demonstration

SDL Regex Fuzzer

A truth about regular expressions

"Some people, when confronted with a problem, think, 'I know, I'll use regular expressions.' Now they have two problems."

- Jamie Zawinski

Conclusions

Defend yourself

- Don’t focus solely on EoP or Confidentiality/Integrity issues
  - DoS is the next battleground
- Follow established patterns for synchronization
- Avoid discretely callable transaction state changes
  - I.e., *HoldSeat*, *ReleaseSeat*, *BookSeat*
- Deploy antivirus on systems processing user uploads
- Decompress asynchronously, kill the thread if necessary

Defend yourself

- For XML parsing code:
  - When possible, disable inline DTD processing entirely
  - If not possible, disable external entity resolution entirely
  - If not possible, throttle external entity resolution requests
- For regular expressions:
  - Avoid group expressions with repetition that are themselves repeated
  - Avoid alternation within groups where the alternate subexpressions overlap each other
  - Use dynamic testing tools along with manual code review

Special thanks

- Special thanks to the following people for their excellent original research in the areas discussed today:

- Steve Orrin
- Amit Klein
- Alex Roichman
- Adar Weidman
- Rick Regan

Q&A

- Adobe Secure Software Engineering Team (ASSET) blog
  - http://blogs.adobe.com/asset

- My alias
  - brsulliv


Adobe