Secure Design

Of Password Storage

-jOHN
Internal CTO, C
@m1spla

Secure Design

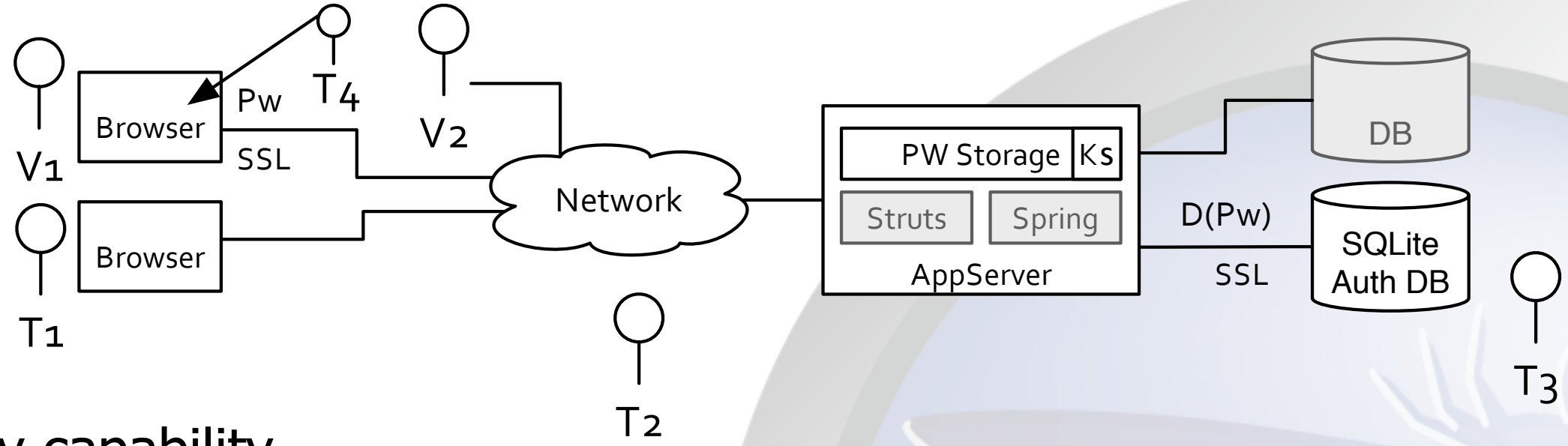
SHA-3 was just released.

So, we're done.

(haha)

Threat Model

1) Acquiring PW DB
2) Reversing PWs from stolen boo



y capability

- Script-kiddie
- **AppSec Professional**
- **Well-equipped Attacker**
- Nation-state

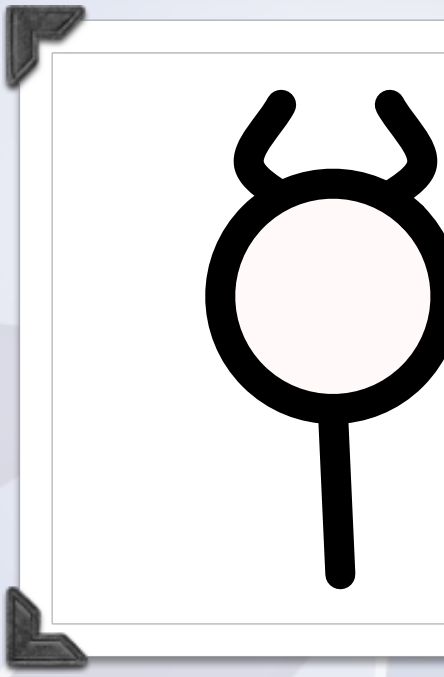
ttacks Specific to PW Storag

1. Dictionary attack
2. Brute-force attack
3. Rainbow Table attack

Well-equipped

4. Length-extension attack
5. Padding Oracle attack
6. Chosen plaintext attack
7. Crypt-analytic attack
8. Side-channel attack

Nation State



ext
oted
d (using SHA)
nd Hash
ve Hashes
BKDF
crypt
crypt

Breaking the Desig Down

Hash Properties

```
digest = hash(plaintext);
```

queness

terminism

lision resistance

n-reversibility

n-predictability

fusion

htning fast

Use a Better Hash?

- SHA-1
- SHA-2
  - SHA-224/256
  - SHA-384/SHA-512
- SHA-3

What property of hashes do these effect?
Collisions. – Was this the problem?
No

What Does the Salt Do?

```
alt || digest = hash(salt || plaintext);
```

duplicates digest texts

s entropy to input space*

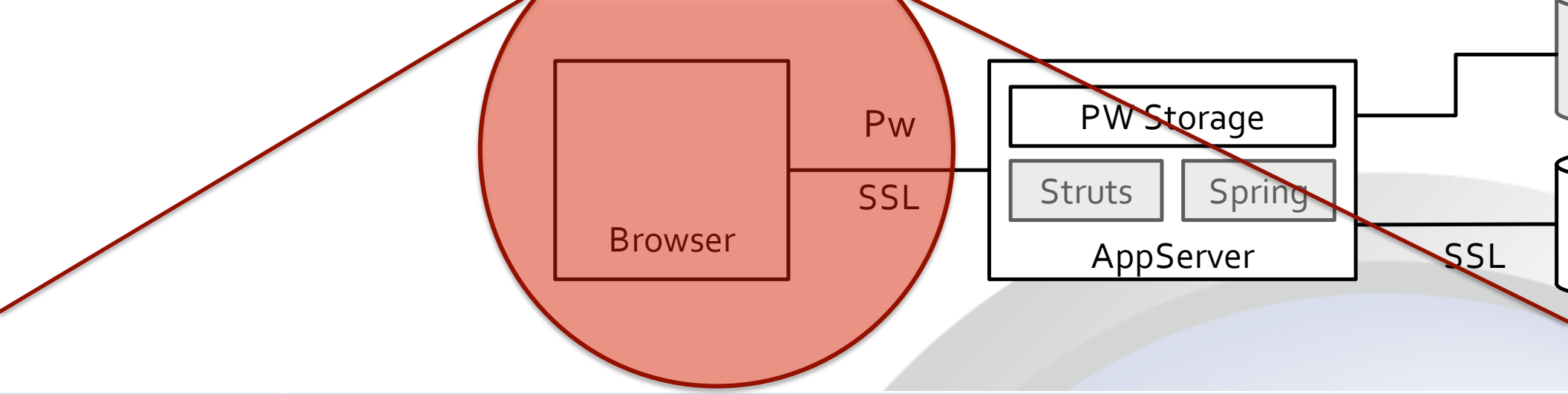
increases brute force time

requires a unique table per user

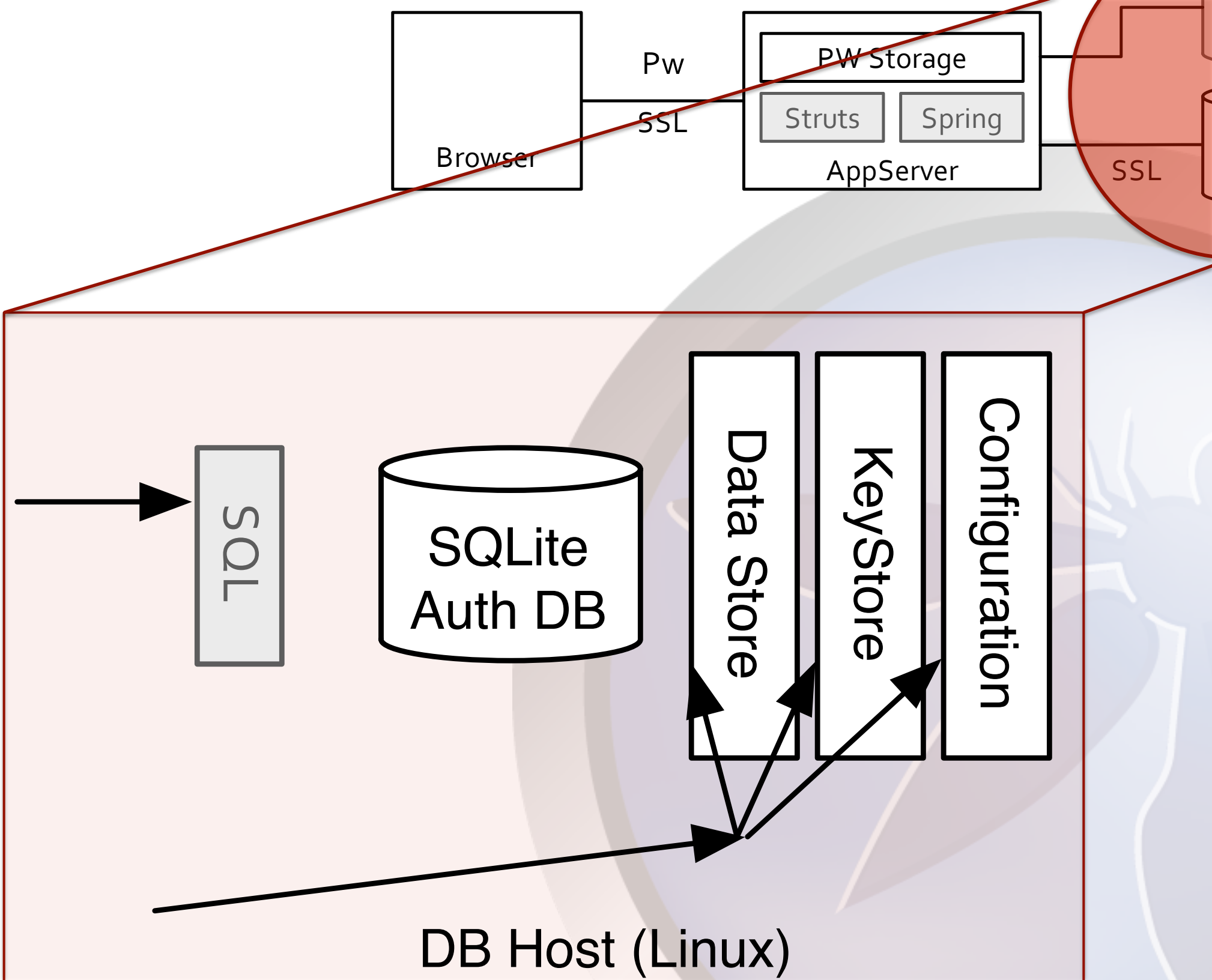
enting Acquisition

enting Reversing

Designing f
Security



	Attack Vector
ppSec	AVA00 - Attack code running in browser
	AVA01 – Inject database and lift (bulk) credentials
	AVR01 – Use API to brute force credentials
itB	AVA10,11 – Keylogger or other scripted attack on client data/entry
itM	AVA03,04 – Interposition, Proxy, or SSL-based attack
oncerted	AVA12 – Infrastructure Attack (Network operators, DNS, or CA compromise)


Browser
Pw
SSL
PW Storage
Struts
Spring
AppServer
SSL
SQL
SQLite
Auth DB
Data Store
KeyStore
Configuration
DB Host (Linux)

Preventing SQLi

est Practices”

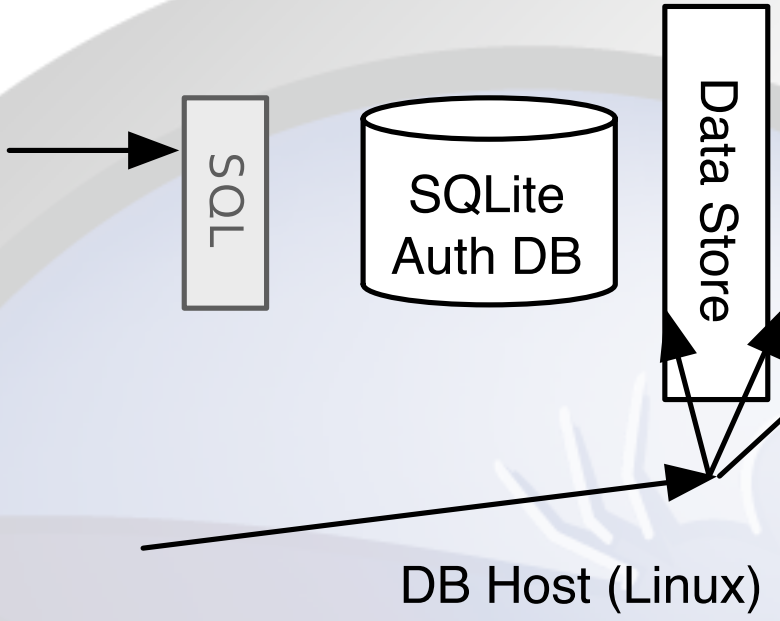
Separate cred./app stores

Parameterize SQL queries

Limit character sets

member hash properties?

Fixed output size, character-set



`ash(“password’); …”)` → `AF68B0E4…`

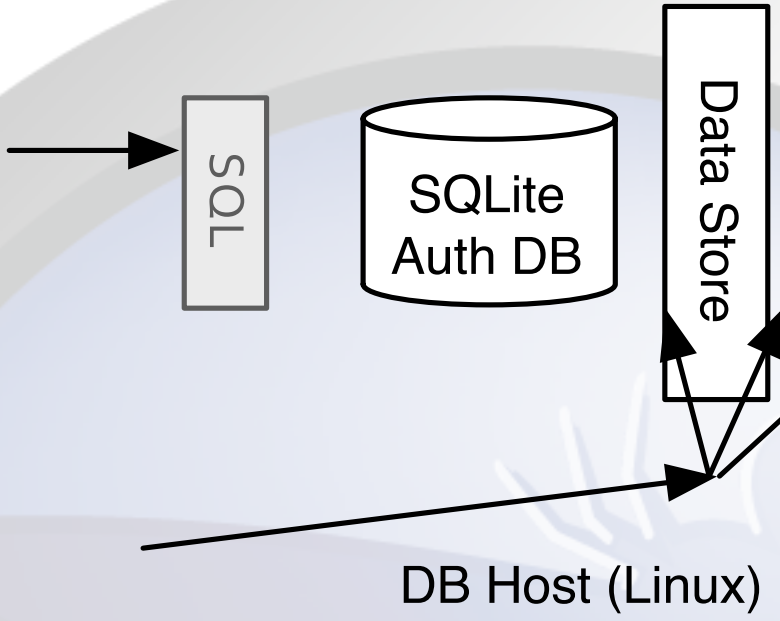
Attacks via Host

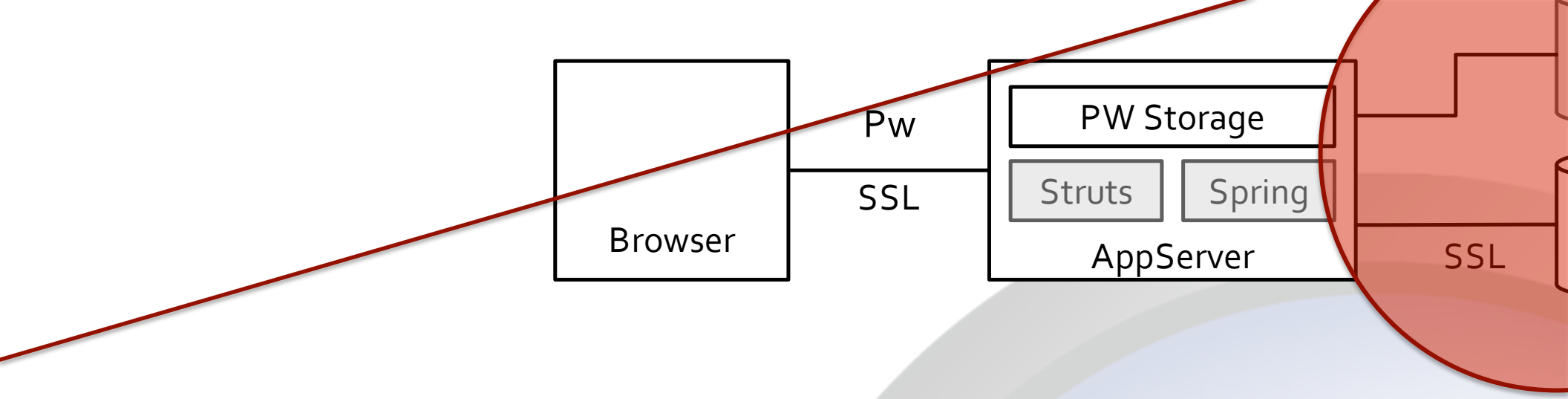
rreversible”

Treat DB as “untrusted”

Store secrets elsewhere

Validate protected form





	Attack Vector
dmin	AVA05 – Bulk credential export
	AVA06 – [T1]-style attack from LAN
	AVA07 – Direct interaction w/ database
itB	AVA08 – Interaction w/ database backups
	AVA09 – Access to logs (SEIM, etc.)
	AVR03– Stored data organization, sort, duplicate-detection
oncerted	Dictionary Attack
	Brute Force Attack
	Rainbow Table Attack
	Cryptanalytic attacks, as applicable

Current Industry Practices

ext
pted
d (using SHA)
nd Hash
ve Hashes
BKDF
crypt
crypt

Hash Properties

```
digest = hash(plaintext);
```

queness

terminism

lision resistance

n-reversibility

n-predictability

fusion

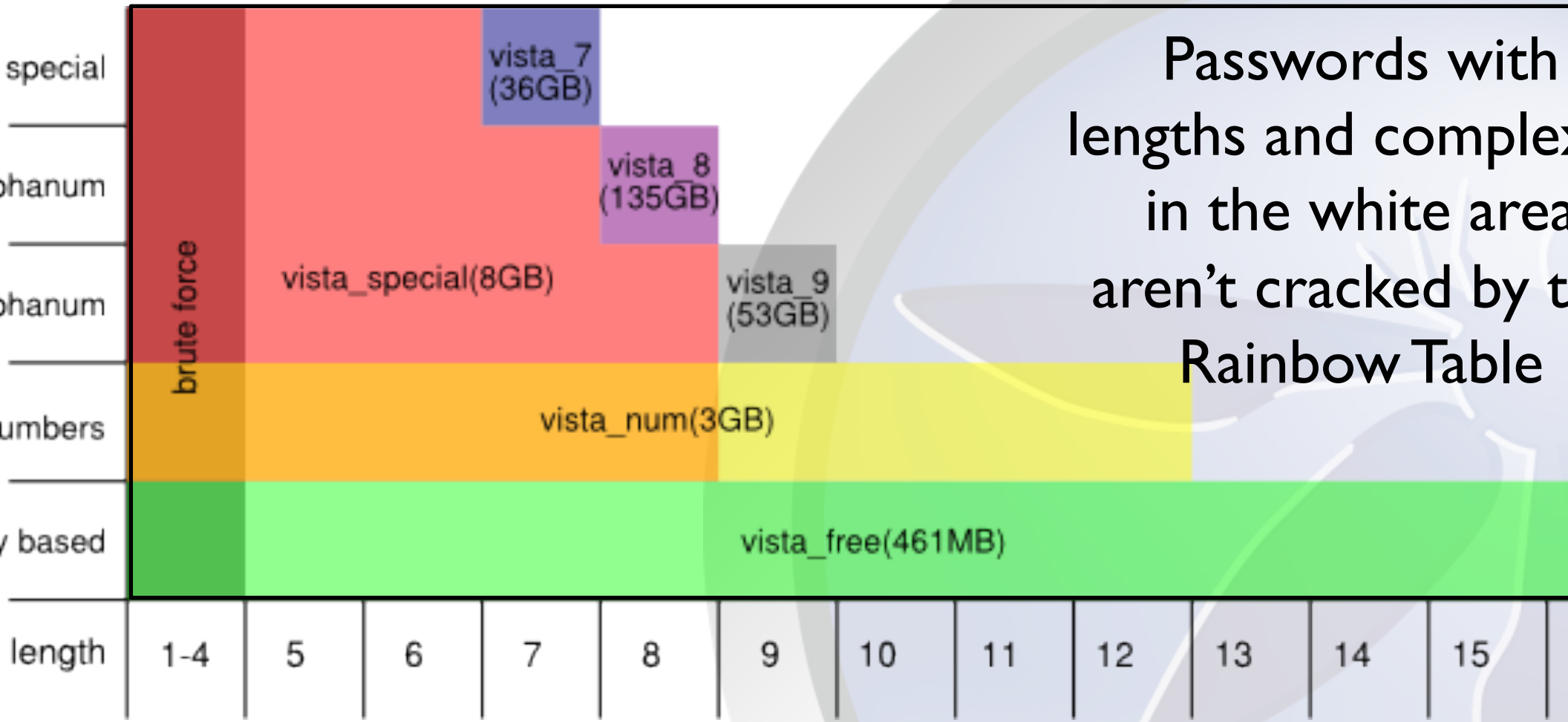
htning fast

Use a Better Hash?

- SHA-1
- SHA-2
  - SHA-224/256
  - SHA-384/SHA-512
- SHA-3

What property of hashes do these effect?
Collisions. – Was this the problem?
No

nbow Tables: Fast but Inher Limitations



Source: ophcrack

es are crafted for specific complexity and le

Per User Table Building

Brute Force Time for SHA-1 hashed, mixed-case-a alphanumeric password

		8 Characters	9 Character
cking a single n (32 M/sec)	NVS 4200M GPU (Dell Laptop)	80 days	13 years
cking a single n (85 M/sec)	$169 Nvidia GTS 250	30 days	5 years
cking a single n (2.3 B/sec)	$325 ATI Radeon HD 5970	1 day	68 days

What Does the Salt Do?

```
alt || digest = hash(salt || plaintext);
```

duplicates digest texts

s entropy to input space*

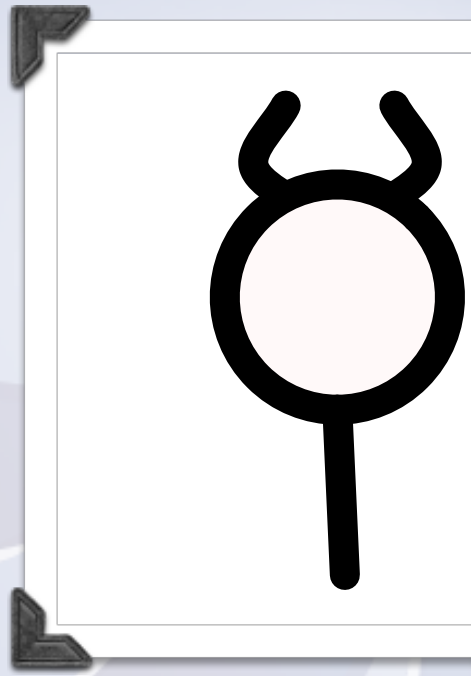
increases brute force time

requires a unique table per user

an salted hashes be Attacked

epends on the threat-actor...

- Script-kiddie
- Some guy
- Well-equipped Attacker
- Nation-state



ttacking a table of salted
hashes means building a
Rainbow Table per user

Adaptive Hashes

hms designed specifically
emove the "lightning-
t" property of hashes

protecting passwords
m Brute Force and
nbow Table attacks

ve Hashes increase the
ount of time each hash
es through iteration

-Based Key Derivation (PBK

t || digest = PBKDF(hmac, salt, pw, c=);

cation Code:

```
= random.getBytes(8)
0000000

 pbkdf2(salt, pw, c, )
cted_pw = concat(salt, key)
```

ring implementation:

```
salt, pw, c, b){
mputeNumOutputBlock(b)
= SHA1-HMAC(p, s || 1)
=2; i <= c; i++)
] = SHA1-HMAC(p, md[i-1])

=0; j < b; j++)
[j] = xor(md[1] || md[2]…md[j]

 concat(kp[1] || kp[2] … kp[r])
```

Well-supported & ve

HMAC key is passwo

Attacker has all entro

What is the right 'c'?

- NIST: 1000
- iOS4: 10000
- Modern CPU: 100000

bcrypt

|| salt || digest = bcrypt(salt, pw, c=)

cation Code:

```
= bcrypt.genSalt(12)
0000000

lt, key = bcrypt(salt, pw, c)
cted_pw = concat(c, salt, key)
```

ring implementation:

```
salt, pw, c){
OrpheanBeholderScryDoubt”
ate = EksBlowfishSetup(c, salt, pw)

int i=0, i < 64,i++){
 blowfish(keyState, d)


n c || salt || d
```

Not supported by JCE

2^{cost} iterations slows ha
operations

Is 2^{12} enough these da

What effect does chang
cost have on DB?

Outputting ‘c’ helps

Resists GPU paralleliza
but not FPGA

scrypt

|| digest = scrypt(salt, pw, N, r, p, dkL

cation Code:

```
6384


crypt(salt, pw, N, p, dkLen){
cted_pw = concat(salt, key)
```

ring implementation:

```
, salt, N, p, c){
0, i < p1, i++)
= PBKDF2(pw, salt, 1, p*Mflen)
, i < p1, i++)
= ROMmix(b[i], N)
PBKDF2(pw, b[1]||b[2]||…b[p-1], 1, dkLen)



0, i < N-1, i++)
* Chain BlockMix(x) over N*/
0, i < N-1, i++)
 Integrify(b) mod N */
* Chain BlockMix(x xor v[j]) */
x
```

Packages emerging, w
trodden than bcryp

Designed to defeat FP
attacks

Configurable

- N = CPU time/Mem
footprint
- r = block size
- P = defense agains
parallelism

***DRAMATICALLY SIMPLIFED Code:

Adaptive Hash Properties

Motivations

sts most Threats' attacks

oncerted (nation-state)
n succeed w/ HW & time

ble implementation

e CPU-difficulty w/
meter*

Limitations

1. Top priority is convincing Sec
   - C=10,000,000 == definition of insanit
   - May have problems w/ heterogeneous
2. API parameters (c=) != devc
   - Must have a scheme rotation plan
3. Attain asymmetric warfare
   - Attacker cost vs. Defender cost
4. No password update w/o use

Defender VS Attacker

Defender

er in w/out > 1sec delay

20M Users, 2M active / hr.

n:

lidation cost * users / (sec / hr.)

vare:

16 CPUs on App Server

64 servers

ss Gauge :

of machines required for AuthN

Attacker

Goal(s vary):

Crack a single password, or *particular*

Create media event by cracking n pas

Rate: Scales w/ Capability

Burden:

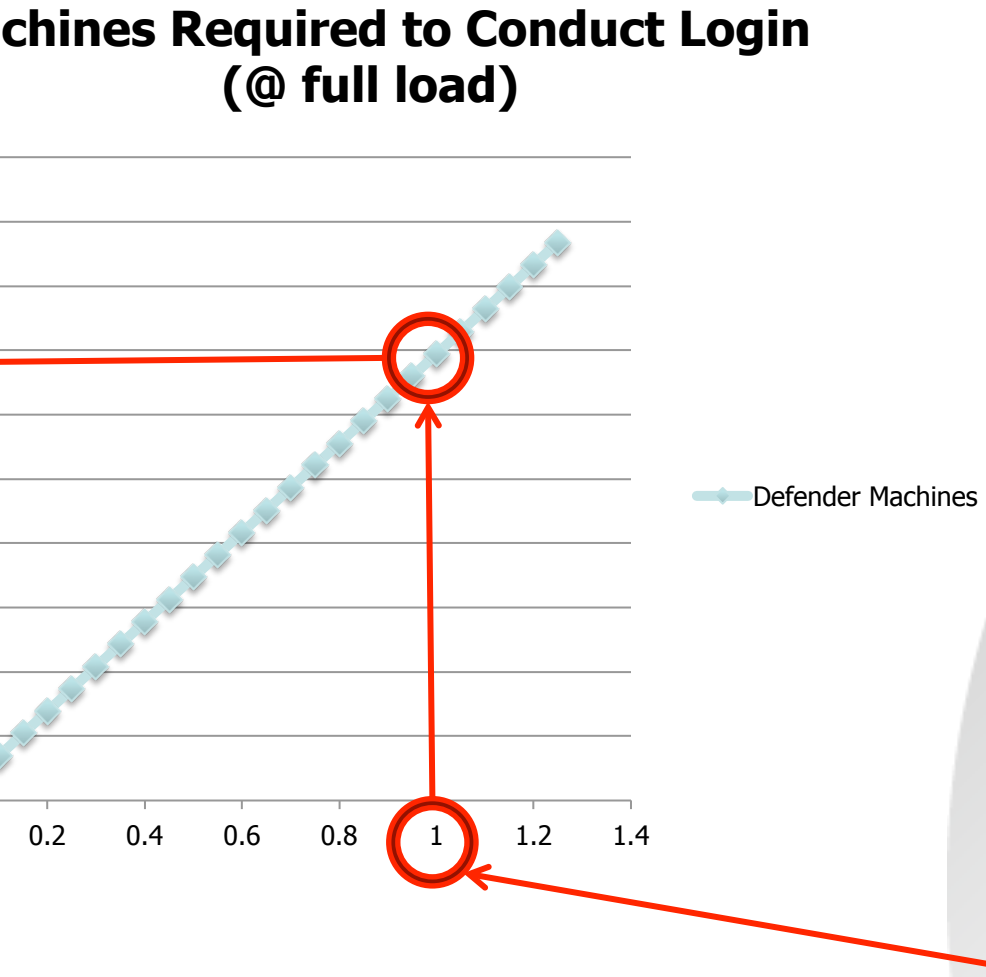
Bound by PW reset interval

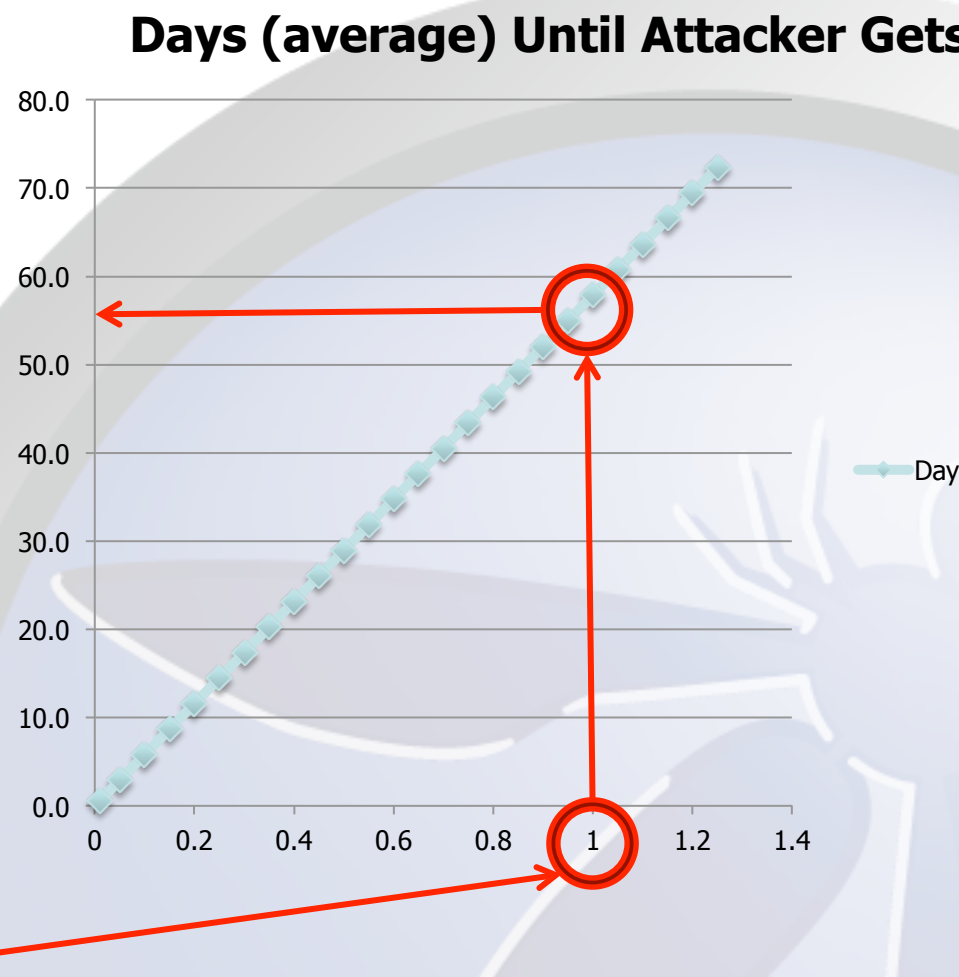
Population / 2 = average break = 10M

Hardware: Custom: 320+ GPUs / card, FP

Success Gauge: Days required to crack P

Tradeoff Threshold





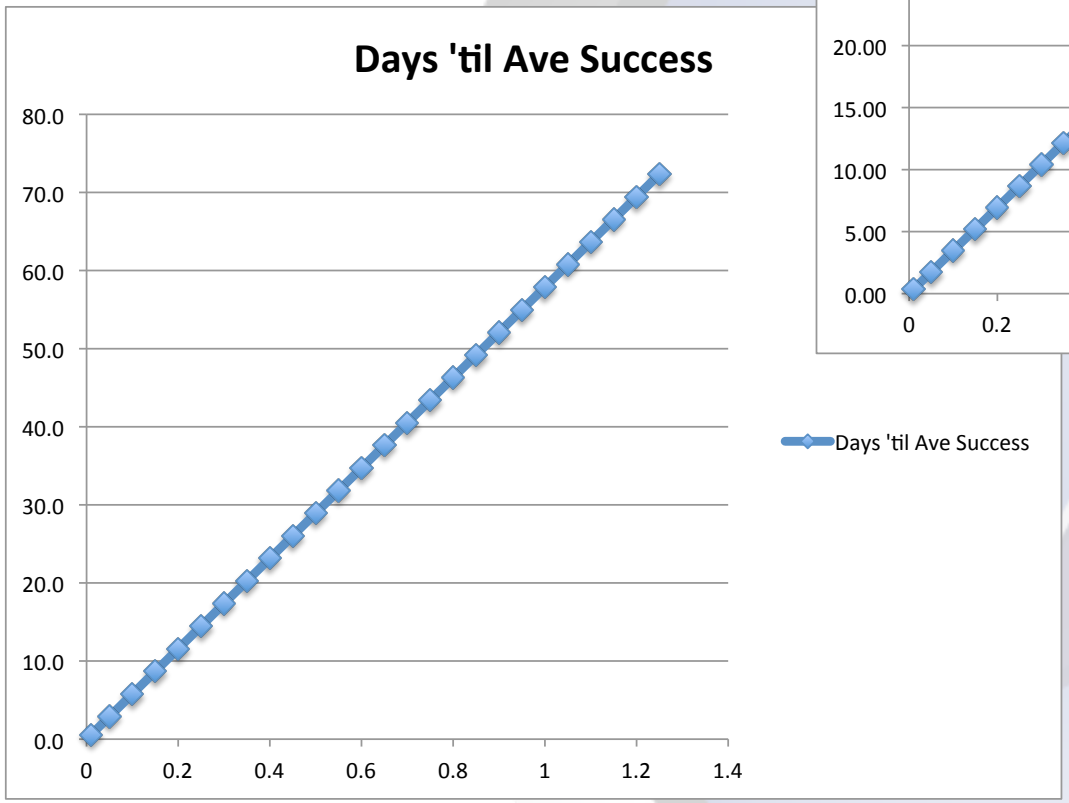
nore than 8 AuthN machines reasonable?

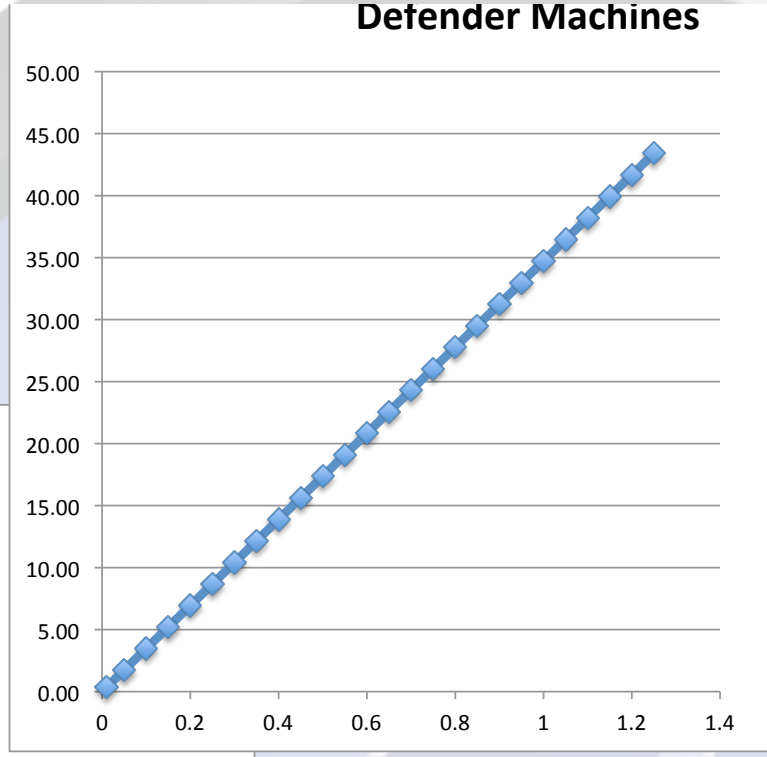
ess than 2 months to average crack good enough?

Attacker/Defender Worksheet

peedup	2
elding success	10000000
der CPU	16
der work (/ sec)	556

der Machines	Days 'til Ave Success
0.35	0.6
1.74	2.9
3.47	5.8
5.21	8.7
6.94	11.6
8.68	14.5
10.42	17.4
12.15	20.3
13.89	23.1
15.63	26.0
17.36	28.9
19.10	31.8
20.83	34.7
22.57	37.6
24.31	40.5
26.04	43.4
27.78	46.3
29.51	49.2
31.25	52.1
32.99	55.0
34.72	57.9
36.46	60.8
38.19	63.7
39.93	66.6
41.67	69.4
43.40	72.3





otive Hashes At Best
rengthen a Single
ontrol Point

Can Do Better with
efense In Depth

Requiring a H
Gains Defen
In Depth

Hmac Properties

```
digest = hash(key, plaintext);
```

Motivations

herits hash properties

This includes the lightning speed

sists all Threats' attacks

Brute force out of reach

- >= 2^{256} for SHA-2

Requires 2 kinds of attacks

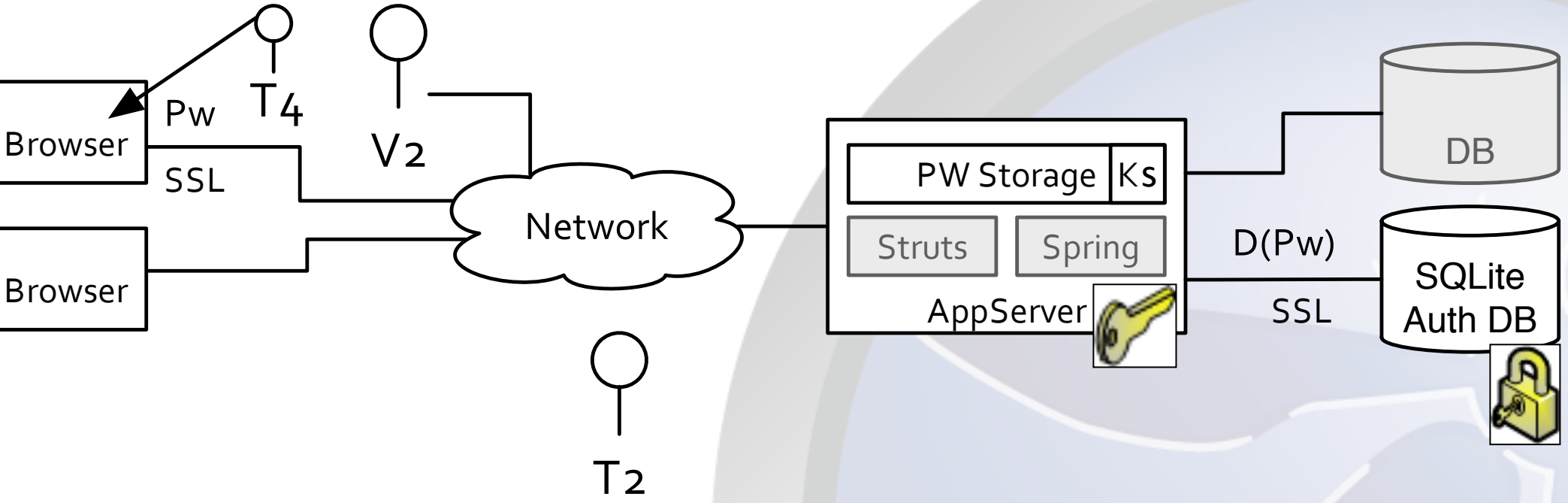
- AppServer: RMIi Host keystore

Limitations

1. Protecting key material challe
   developers
   - Must not allow key storage in DB!!!
2. Must enforce design to stop
   - compartmentalization and
   - privilege separation (app server & db
3. No password update w/o use
4. Stolen key & db allows brute
   force

COMPAT/FIPS Design

on||salt||digest = hmac(key, version||salt||passw




```
ac = hmac-sha-256
rsion per scheme
it per user
y per site
```

- Add a control requiring a key stored on the App Server
- Threats who exfiltrate passwo table also needs to get hmac

COMPAT/FIPS Solution

on$_{scheme}$>||<salt$_{user}$>||<digest> := HMAC(<key$_{site}$>, <mixed construc

construct> := <version$_{scheme}$>||<salt$_{user}$>||<pw$_{user}$>

te := hmac-sha256

ser := PSMKeyTool(SHA256()):32B;

:= SHA1PRNG():32B | FIPS186-2():32B;

:= <governed by password fitness>

:

d construct> := <version$_{scheme}$>||<salt$_{user}$>||':'||<GUID$_{user}$>||<p

er := ***NOT*** username or available to untrusted zone

Just Split the Digest?

They’re not the same.

acks key space (brute force expansion)

teal both pieces with the same technique

emember `000002e09ee4e5a8fcdae7e3082c9d8ec3d304a5`

```
anence:code jsteven$ python split_hash_test.py -v 07606374520 -h ../hashes.txt
und ['75AA8FF23C8846D1a79ae7f7452cfb272244b5ba3ce315401065d803'] verifying passw
total matching


anence:code jsteven$ python split_hash_test.py -h ../hashes_full.txt -v excal1be
und ['8FF8E2817E174C76b8597181a2ee028664aadff17a32980a5bad898c'] verifying passw
total matching
```

(More) Just Split the Digest

nparing 20B PBKDF2 chunks created no collisio

t

ase:
0/50 split

quely

s
oad
ER

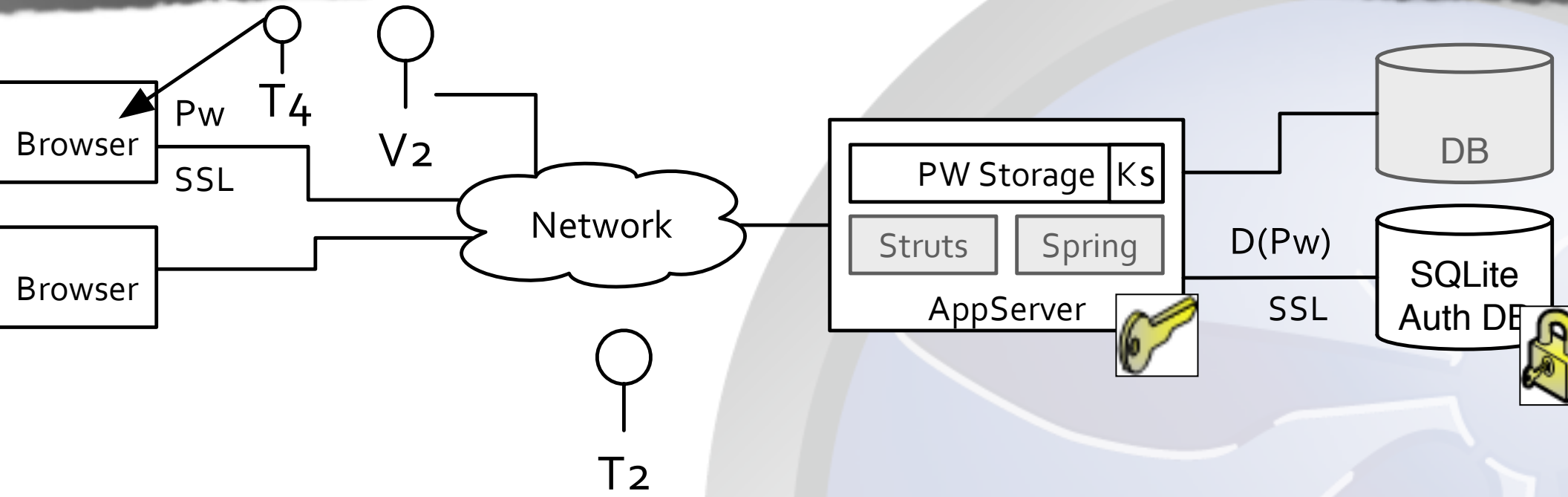
520

18

```
Permanence: jsteven$ grep passwords ../hashes.txt
Permanence: jsteven$ python split_hash_test.py -v passwords -h ../hashes.txt
+ Found [] matching passwords
Permanence: jsteven$ python split_hash_test.py -h ../hashes_full.txt -v excal1b
+ Found 1 ['8FF8E2817E174C76b8597181a2ee028664aadff17a32980a5bad898c'] matching
+ Found 1 ['4F10C870B4E94F814fd07046b8d3bea650073e564c39596b8990d74b'] matching
+ Found 1 ['EBD19B279CC64554f83f485706073fab5a112ea63143ec82a37e6d41'] matching
+ Found 1 ['A4575F1E7D4C41DEc0ae49c5ce48ce4a9dbe28b9e87635e7289eb7eb'] matching
+ Found 1 ['E1301662EC6349E5021c4cd8c158533aa9342ddee452f74f321ea0fa'] matching
+ Found 1 ['72532DBFBF954FA1d9a068690ed1c3fc09459932be96bad5af4e1453'] matching
+ Found 1 ['043EAF3FE8434630d9d513284835c0891f0fbfcbeaf1f6bb6f76bc06'] matching
+ Found 1 ['636BEF93F99449114785304641f419d450ce24ddfa03f4383e7593e6'] matching
+ Found 1 ['A66772BEAF7A47361f6929611cc24b92b86cb84403c7773996ac49bc'] matching
+ Found 1 ['8C8066C40C224A6700c50395afa1d3a87c9b76a1215193a29226e170'] matching
```

Reversible Design

||cipher = ENC(wrapper key_{site}, <pw digest>)
est> = version||salt|| digest = ADAPT(version||$salt_{user}$||passw



= AES-256
T = pbkdf2 | scrypt
ion per scheme
per user
per site

hmac Solution Properties

ttack	Resistance
esist chosen plain text attacks	**Yes**, Scheme complexity based on ($salt_{user}$ & pw_{user}) +
esist brute force attacks	**Yes**, $Key_{site} = 2^{256}$, $salt_{user} = 2^{256}$
esist D.o.S. of entropy/randomness exhaustion	**Yes**, 32B on password generation or rotation
event bulk exfiltration of credentials	Implementation detail: <various>
event identical <protected>(pw) creation	**Yes**, provided by salt
event <protected>(pw) w/ credentials	**Yes**, provided by Key_{site}
event exfiltration of ancillary secrets	Implementation detail: store Key_{site} on application se
event side-channel or timing attacks	**N/A**
event extension, similar	**Yes**, hmac() construction (i_pad, o_pad)
event multiple encryption problems	**N/A** (hmac() construction)
event common key problems	**N/A** (hmac() construction)
event key material leakage through primitives	**Yes**, hmac() construction (i_pad, o_pad)

Reversible Properties

||cipher = ENC(wrapper key_{site}, <pw digest>)
est> = version||salt|| digest = ADAPT(version||$salt_{user}$||passw

Motivations

rits

compat" solution benefits

daptive hashes' slowness

uires 2 kinds of attacks

pp Server & DB

rute forcing DB out of reach (>=2^{256})

tolen key can be rotated ***w/o*** user
teraction

tolen DB + key still requires reversing

Limitations

1. Protecting key material challeng
   developers
   1. Must not allow key storage in DB!!!
2. Must enforce design to stop T3
   1. compartmentalization and
   2. privilege separation (app server & db)
3. No password update w/o user
4. Stolen key & db allows brute fo
   1. Rate = underlying adaptive hash

ST IMPORTANT TOPIC

onding once attacked

Operation

Replacing legacy PW DB

rotect the user's account

- Invalidate authN 'shortcuts' allowing login w/o 2nd factors or secret questions
- Disallow changes to account (secret questions, OOB exchange, etc.)

ntegrate new scheme

- Hmac(), adaptive hash (scrypt), reversible, etc.
- Include stored with digest

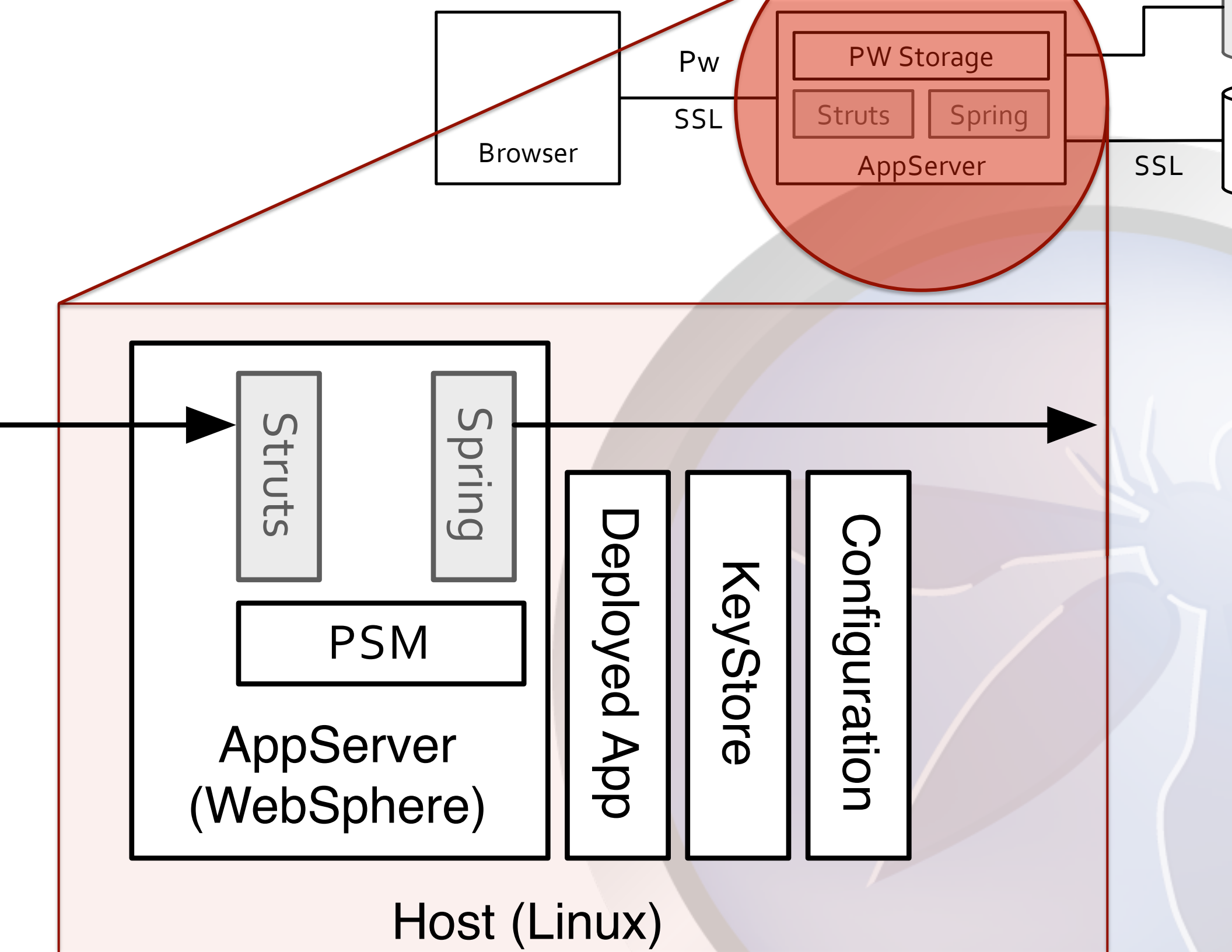
Vrap/replace legacy scheme: (incrementally when user logs in--#4)

$version||salt_{new}||protected = scheme_{new}(salt_{old}, digest_{existing})$ –or–

- For reversible scheme: rotate key, version number

Vhen user logs in:

- Validate credentials based on version (old, new); if old demand 2nd factor or secret ans
- Prompt user for PW change, apologize, & conduct OOB confirmation


Browser
Pw
SSL
PW Storage
Struts
Spring
AppServer
SSL
Struts
Spring
PSM
AppServer
(WebSphere)
Deployed App
KeyStore
Configuration
Host (Linux)

ank You for Your Time

Question

Conclusions

- Without considering specific threats, the solutions misses key properties
- Understanding operations drives a whole set of hidden requirements
- Many solutions resist attack equivalently
- Adaptive hashes impose on defenders, affecting scale
- Leveraging design principles balances solution
  - Defense in depth
  - Separation of Privilege
  - Compartmentalization

TODO

- Revamp password cheat sheet
- Build/donate implementation
  1. Protection schemes
  2. Database storage
  3. Key store ← Vital to preventing dev err
  4. Password validation
  5. Attack response

itional Material for longer-format presentations

Supportin Slides

Select Source Material

le material

word Storage Cheat Sheet

tographic Storage Cheat Sheet

S #5: RSA Password-Based Cryptography Standard

e to Cryptography

n Wall's Signs of broken auth (& related posts)

Steven's Securing password digests

am-Cumming 1-way to fix your rubbish PW DB

RFC2898

er work

g Security, Resin

ypt

Applicable Regulation, Audit, or Special Gui

- COBIT DS 5.18 - Cryptographic key manage
- Export Administration Regulations ("EAR") I
- NIST SP-800-90A

Future work:

- Recommendations for key derivation NIST S
- Authenticated encryption of sensitive mater NIST SP-800-38F (Draft)