

DOTA2024:7

Defense of the Ancients

Seventh topic - Web applications

Hugh Anderson

National University of Singapore
School of Computing

July, 2024



Look what is above the pretty table...



: Administrator Login :

Username :

Password :



Outline

- 1 **Attacking web apps**
 - Command injections
 - Web applications and SQL injections
- 2 **Exploiting trust between browsers and sites**
 - XSS, CSRF - Cross site scripting and request forgery
 - Attacking the problem
- 3 **Hash functions**
 - Non-cryptographic, and cryptographic
 - Time memory tradeoff, to find preimages
 - Birthday attacks, to find collisions

Where are we?

The topics: Attack surfaces so far...

Introduction (in May): Warfare, with an indistinct enemy.

Social Engineering (in May): Examples - phishing, vishing, tricking.
Human weaknesses.

System Complexity: Our difficulty in designing and building complex systems. Fight back with design principles, standards and formal methods.

Crypto: Traditional systems - substitution and permutation.
Asymmetric systems based on OWF and computational complexity. Not just confidentiality - SS, MPC, IBE...

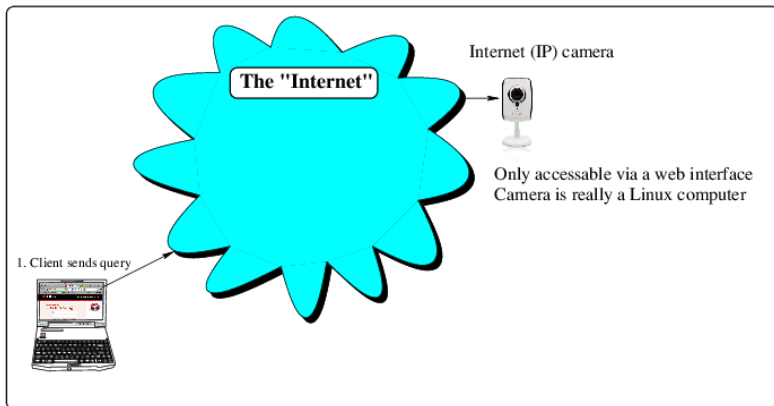
IP Networks: Complex layered structure designed for resilience, not security. Tools and work on hardening the Internet.

OtherNets: Samples of other communication worlds in the electromagnetic spectrum that largely rely on security-by-obscurity.

Web Applications: Today we will look at the applications that sit on the Internet - forms of attacks, defenses. We will also catch up on hash functions - missed from the crypto topic.

Accessing an embedded system

Internet cameras are everywhere



Accessing an embedded system...

...from a web page...

Normally you get only **limited access** to the system:

Product: DCS-2121Firmware Version: 1.00

D-Link

DCS-2121

LIVE VIDEO

SETUP

MAINTENANCE

STATUS

HELP

Wizard

Network Setup

Wireless Setup

Dynamic DNS

Image Setup

Audio and Video

Motion Detection

Time and Date

Recording

Snapshot

Digital Output

Logout

MOTION DETECTION

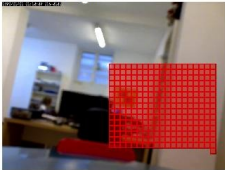
In order to use motion detection, you must first check the checkboxes, then draw the areas you want to monitor for motion.
Motion detection is disabled while camera is set to SXGA mode.
To enable motion detection, select VGA or XGA mode in [Audio and Video](#).

Save Settings

Don't Save Settings

LIVE VIDEO

☒ Enable Video Motion



Sensitivity

100

0~100%

Drawing Mode

☒ Draw motion area

☐ Erase motion area

Clear

Save Settings

Don't Save Settings

Helpful Hints..

Sensitivity - Set the sensitivity for motion detection.

High sensitivity makes the motions easier to be detected.

SECURITY

Copyright © 2007-2008 D-Link Corporation.

Accessing an embedded system...

But...

- ...on one page you can **enter usernames and passwords** for remote mounting a file system:

Product: DCS-2121 Firmware Version: 1.04

D-Link

DCS-2121 //	LIVE VIDEO	SETUP	MAINTENANCE	STATUS	HELP
Setup Wizard Network Setup Wireless Setup Dynamic DNS Image Setup Audio and Video Motion Detection Time and Date Recording Snapshot Digital Output SD Card Logout	<div>RECORDING</div> <p>Here you may configure and schedule the recording of you camera. You must select the checkbox of 'Enable Recording' to turn on the feature.</p> <p><input type="button" value="Save Settings"/> <input type="button" value="Don't Save Settings"/></p>			<div>Helpful Hints...</div> <p>You can record the video to a 'SD Card' or a 'Samba network drive' based on the selected events. You may also configure the 'Recording Options' and select a scheduling method to specify when the camera will record video.</p>	
	<div>RECORDING</div> <p><input checked="" type="checkbox"/> Enable recording</p> <p>Record to :</p> <p><input type="radio"/> SD Card</p> <p>SD Card status : Disable <input type="button" value="Get status"/></p> <p><input checked="" type="radio"/> Samba network drive</p> <p>Samba Auth Account <input type="button" value="v"/></p> <p>User name <input type="text" value="toto"/></p> <p>Password <input type="password" value="*****"/></p> <p>Password confirm <input type="password" value="*****"/></p> <p>Server <input type="text" value="192.168.0.5"/></p> <p>Shared folder <input type="text" value="toto"/></p> <p><input type="button" value="Test"/> <input type="button" value="ok"/></p> <p>Samba status : Disable <input type="button" value="Get status"/></p>				

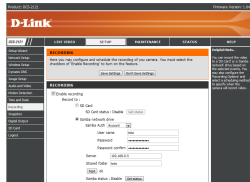
Accessing an embedded system...

Page results in execution of a command in the system

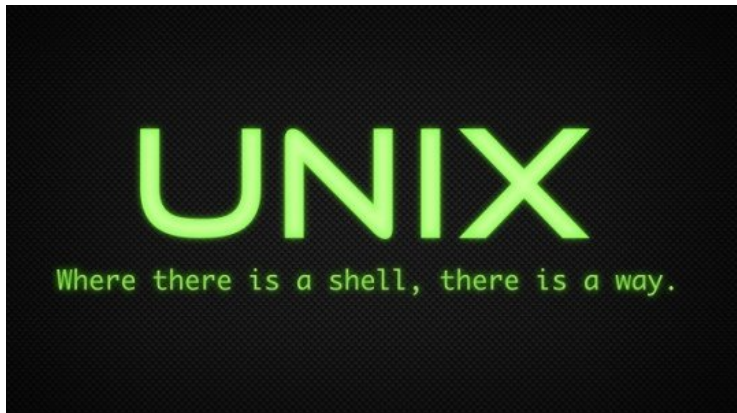
The actual command executed inside the camera will be:

```
smbmount //192.168.0.5/toto /mnt -o  
username=toto,password=PASS
```

- The trick here is to replace the password **PASS** with something like "**PASS;/bin/command**".
- After mounting the drive, your **command** will be executed (it can be anything, but should be something to get you extra access).



Bash, the Bourne (again) shell is everywhere...



And in 1989, an “improvement” to bash was made, which in September 2014, was found to be an awesome mistake.

Bash variables

...are called "environment" variables...

And we have lots of them - for example \$PATH, \$SHELL and so on. Sometimes we use them as function definitions:

```
hugh$ export foo='() { echo "Inside"; };'  
hugh$ bash -c foo  
Inside  
hugh$ export foo='() { echo "Inside"; }; echo "###"'  
hugh$ bash -c foo  
###  
Inside  
hugh$
```

Note that the extra code after the end of the function definition got executed. It was done when bash started up, and processed the environment variable foo.

What is the issue?

This is considered insecure because variables are not typically allowed or expected, by themselves, to directly cause the invocation of arbitrary code contained in them.

Bash variables

More chaos

Lets see some difficult effects of this:

```
hugh$ env foo='() { echo "Inside" ; };' bash -c "echo Hello"
Hello
hugh$ env foo='() { echo "Inside" ; }; echo "###"' bash -c "echo Hello"
###
Hello
hugh$
```

In this case, the extra injected code has run...

Even when we do not call it, or foo.

It doesnt sound too bad...

But unfortunately, lots of things on UNIX and MAC systems use bash. People have identified ways of attacking the openSSH login, CGI scripts on web servers, DHCP and so on.

In each case by manipulating environment variables passed to the remote server.

Bash variables

CHECK THIS OUT...

```
localhost:BASH hugh$ ls
localhost:BASH hugh$ bash -c "echo date"
date
localhost:BASH hugh$ ls -l
localhost:BASH hugh$ env X='() { (a)=>\'} bash -c "echo date"
sh: X: line 1: syntax error near unexpected token `='
sh: X: line 1: `sh: error importing function definition for `X'
localhost:BASH hugh$ ls -l
total 8 -rw-r--r--  1 hugh  staff   29 Sep 27 16:19 echo
localhost:BASH hugh$ cat echo
Sat Sep 27 16:19:32 SGT 2014
localhost:BASH hugh$
```

Thats pretty weird. echo date became date>echo - i.e. we wrote a file - oooooohhhh.

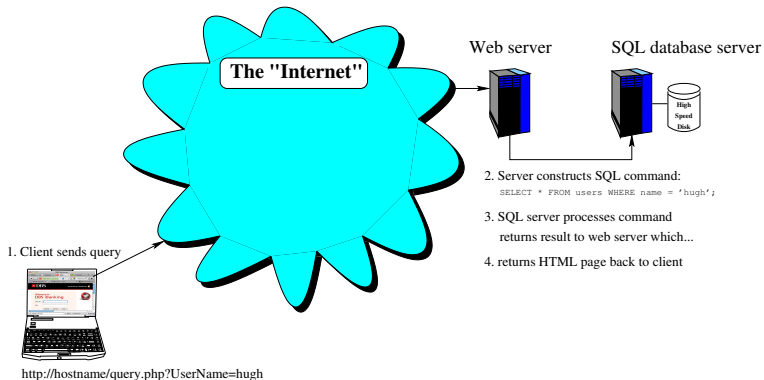
I think I am shellshocked...



Better get bash fixed ASAP methinks.

A web application injection...

It is common to link pages to database servers



The system architecture seems innocuous, but...

You access a remote web site, and fill in a form asking for particular information from a database. For example - getting information on a particular user, or a particular product.

The way this normally works is that you type in a substring in a web form, and this substring is used to build a new string which is a (SQL) query to a database.



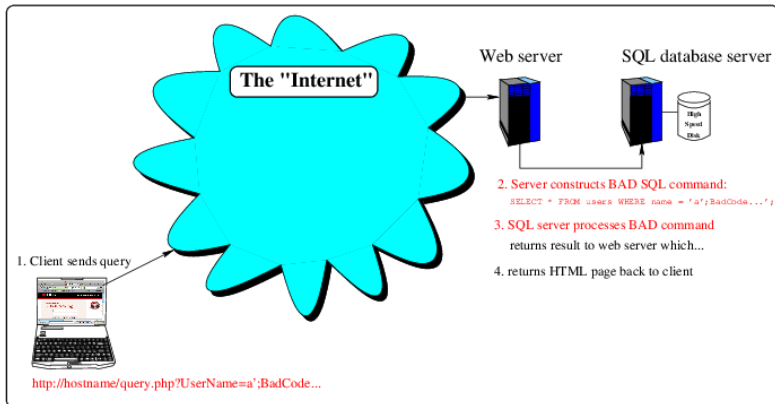
Unfortunately, sometimes it is possible to craft a substring containing SQL commands that can do different things from what the developer of the web site intended. As a result... The attacker may **gain access** to (supposedly) secret data and may be able to **change/manipulate** the secret data.



Technique of crafting special substrings with SQL commands is called the "**SQL injection attack**".

The result is BAD...

We end up with this:



Detail on the injection...

userName web form input is given to a SQL command

```
SELECT * FROM users WHERE name = '"' + userName + '');
```

If the user types in (say) hugh, then you get

```
SELECT * FROM users WHERE name = 'hugh';
```

A malicious user could type in (say)

```
a';DROP TABLE users;SELECT * FROM info WHERE 't' = 't
```

And then you get:

```
SELECT * FROM users WHERE name = 'a';  
DROP TABLE users;  
SELECT * FROM info WHERE 't' = 't';
```

Erk...

News in 2012...

[ZDNet](#) / [News](#) / [Security](#)


Singapore's NUS confirms security breach

By [Liau Yun Qing](#), ZDNet Asia on January 5, 2012 (4 days ago)

Summary

update Hacker group Team Intra breaches local institution's database and publishes information such as staff username and hashed passwords. NUS confirms attack but says data on server not confidential.

 Recommend

 17 people recommend this. Be the first of your friends.

 Tweet 8

 +1 0

 Share 2

 Submit

update SINGAPORE--Hackers have infiltrated the National University of Singapore's (NUS) backend systems and made away with a trove of information, including staff usernames, domain information and hashed passwords. University has confirmed the incident but noted that the affected data is not confidential.

According to a report Thursday on IT security Web site [Secure Computing Magazine's \(SC Magazine\)](#), hacker group Team Intra had infiltrated the tertiary institution's database by exploiting a SQL vulnerability. The hackers apparently felt the urge to do so after the NUS Web site, upon receiving probes, generated an error message which stated: "If you're trying to use the SQL error message to dig for juicy information, get lost."

NUS attacks

What was done?

Firstly - it was [not NUS](#), but a departmental web server at NUS that was hacked. The [hackers](#) got [irritated](#) by a message on the web site, and made it a mission to hack it. They reported that the web site had [minimal security](#).

The attack was a [SQL injection](#) attack, which allowed them to download [usercode/password hash](#) entries stored in the SQL database attached to the web server. The [passwords](#) were [not NUSNET](#) ones, but ones specifically for the application on the departmental server.

Somewhere, today, someone is probably trying to reverse some of those hashes.

What can be done?

Kristy Swanson explains about SQL injections.



PHP: control over inputs

For example semicolon and SQL-injection

PHP/HTML Code for a web page, for accessing SQL database:

```
<html> <body>
...
<?php
$matric = $_GET["matric"];
$sql     = "SELECT `marks` FROM `test`.`mrks`
           WHERE userName=' $matric' ";
$result  = mysql_query($sql)
           or die('Error: ' . mysql_error());
?>
...
<form action="grades.php?method=1" action="get">
    Student ID: <input type="text" name="matric"/>
    <input type="hidden" value=1 name="method"/>
    <input type="submit"/>
</form>
</body> </html>
```

PHP: control over inputs

mysql_real_escape_string()

... puts backslashes before critical characters:

```
<?php
$matric = mysql_real_escape_string($_GET["matric"]);
$sql    = "SELECT `marks` FROM `test`.`mrks`
          WHERE userName='$matric'";
$result = mysql_query($sql)
          or die('Error: ' . mysql_error());
?>
```

...which turns...

A' OR '1' = '1'
into
A\' OR \'1\' = \'1' (with the quotes escaped)

PHP: control over inputs

Other techniques: filters

- PHP also has **sanitizing filters**, which can ensure (for example)
 - That input is a **number**, or a float
 - That input is an **email** address
 - That input is a **URL**

These might be used like this:

```
...  
<?php  
...  
filter_var($somevar, FILTER_SANITIZE_NUMBER_FLOAT,  
           FILTER_FLAG_ALLOW_FRACTION);  
...  
?>
```

PHP: control over inputs

Other techniques: Typing

- If the variable was a number, we might have:

```
$pid = $_GET["productid"];  
$sql = "SELECT * FROM prods WHERE id='$pid'";
```

- Much better to do change it to:

```
settype($pid, 'integer');  
$pid = $_GET["productid"];  
$sql = "SELECT * FROM prods WHERE id='$pid'";
```

PHP: control over inputs

Other techniques: hashing

How about this?

```
$sql = "SELECT * FROM prods WHERE MD5(id)=".md5($pid);
```

...which results in queries like this:

```
SELECT * FROM prods WHERE
      MD5(id)='c897eeae801ef096c0918ad338690917'
```

- Use `md5` function in PHP, and `MD5` in the SQL server.
- Makes it extremely difficult for someone to do an injection.

Java: control over inputs

For example semicolon and SQL-injection

Java Code in a web page for accessing SQL database:

```
String Matric = request.getParameter("matric");  
...  
String sel = "SELECT marks FROM test.mrks  
            WHERE userName= '" + Matric + "'";  
Statement stmt = connection.createStatement ();  
ResultSet resultSet = stmt.executeQuery(sel);  
...
```

Java: control over inputs

We can use similar ideas from the PHP ones:

- Use Java methods to **sanitize input** (like `mysql_real_escape_string()`)
- Java has a **PreparedStatement** class (instead of the `Statement` class above).
 - Parameters passed in have **TYPE** (i.e. integer, float..).
 - A `PreparedStatement` does not automatically stop SQL injections, but it **allows** you to **control** the constructed SQL strings more precisely.
 - You may **still need to filter** the parameters (if for example the parameter was a string).
- **Use the MD5 trick**

C# .NET: control over inputs

SQL-injection

.NET code in a web page for accessing SQL database:

```
private bool DoesUserExist(string name) {  
    using (SqlConnection conn = new SqlConnection(@"server=SrvrName; ...  
    {  
        string sql = "SELECT * FROM Users WHERE Name = '" + name + "'";  
        SqlCommand cmd = conn.CreateCommand();  
        cmd.CommandText = sql;  
        ...  
        DataSet dsResult = new DataSet();  
        ...  
        return...  
    }  
}
```

C# .NET: control over inputs

Fixing using "adding parameters"

Note that a string added in this way will still just be a string.

```
private bool DoesUserExist(string name) {  
    using (SqlConnection conn = new SqlConnection(@"server=SrvrName; ...  
    {  
        string sql = "SELECT * FROM Users WHERE Name = @name";  
        SqlCommand cmd = conn.CreateCommand();  
        cmd.CommandText = sql;  
        cmd.CommandType = CommandType.Text;  
        SqlParameter usernameParameter = new SqlParameter();  
        usernameParameter.ParameterName = "@name";  
        usernameParameter.DbType = DbType.String;  
        usernameParameter.Value = name;  
        cmd.Parameters.Add(usernameParameter);  
        cmd.Connection.Open();  
        ...  
        DataSet dsResult = new DataSet();          /*  
        ...  
        return...  
    }  
}
```

Testing for (SQL) injections

Various tools exist: sqlmap at sourceforge:

```
r-235-188-28-172:sqlmap hugh$ python sqlmap.py -u http://...
    sqlmap/0.9 - automatic SQL injection and database takeover tool
[*] starting at: 12:26:09
[12:26:09] [INFO] testing connection to the target url
sqlmap identified the following injection points:
---
Place: GET
Parameter: matric
    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE or HAVING clause
    Payload: matric=A12345678' AND (...)

    Type: UNION query
    Title: MySQL UNION query (NULL) - 1 to 10 columns
    Payload: matric=A12345678' UNION ...

    Type: stacked queries
    Title: MySQL > 5.0.11 stacked queries
    Payload: matric=A12345678'; SELECT SLEEP(5);-- AND ...

    Type: AND/OR time-based blind
    Title: MySQL > 5.0.11 AND time-based blind
    Payload: matric=A12345678' AND SLEEP(5) AND 'WtrO'='WtrO&method=1 ---

[12:26:10] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.2.12, PHP 5.3.0
back-end DBMS: MySQL 5.0
r-235-188-28-172:sqlmap hugh$
```

Too many privileges...

Base scenario - as before - a web server with SQL access:

PHP code in a web page for accessing SQL database:

```
<?php>
...
$con = mysql_connect('...', 'root', 'rootpasswd')
        or die('Could not connect to the server!');
...
?>
```

Instead should be:

```
<?php>
...
$con = mysql_connect('...', 'nobody', 'nobodypasswd')
        or die('Could not connect to the server!');
...
?>
```

Too many privileges...

Why?

Consider the situation where someone succeeds with an injection, or with XSS attack, or with some combination. In the two cases:

- With root access, the attacker could cause significant damage to the database - perhaps removing the database entirely (using the SQL “DROP TABLE” command).
- Without root access, i.e. only enough access to perform the required task, a successful attacker is limited in the damage that could be done.

Note that the nobody user might only be allowed to read and write entries, not create new ones, or delete ones, or delete tables, and so on.

Cross site scripting (XSS)

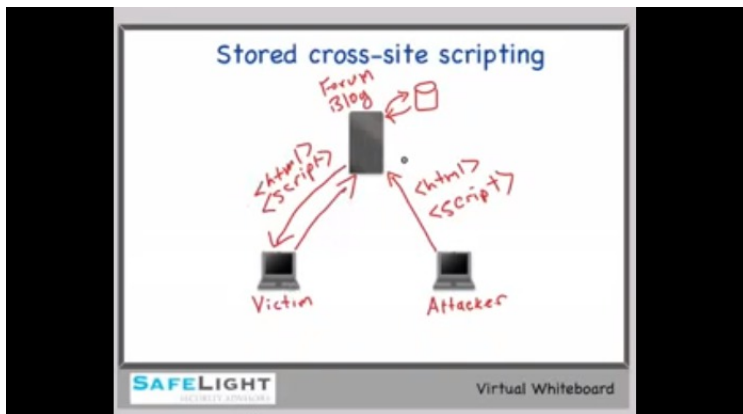
Description:

Many web-based applications are open to XSS attacks, through **not checking** for scripts/HTML in inputs.

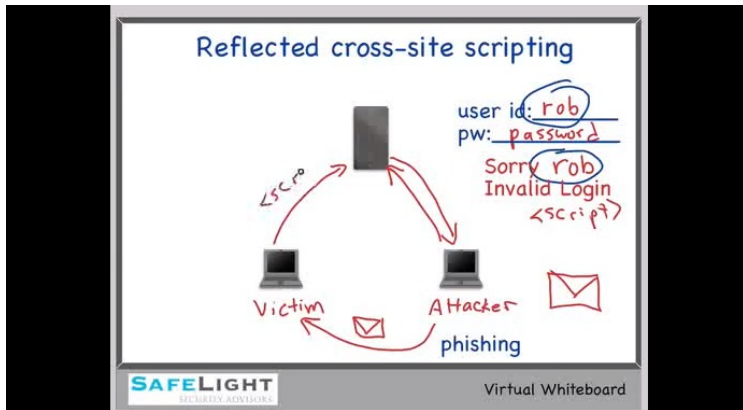
- Attacks can **steal cookies**, or other information
- **Two attacks** Non-persistent (reflected) or persistent (stored).

Now the most common reported vulnerability (even more than buffer overflow). About **70% of websites** in a survey appear to be vulnerable. Examples of affected websites include Google, Yahoo!, MySpace, Twitter...

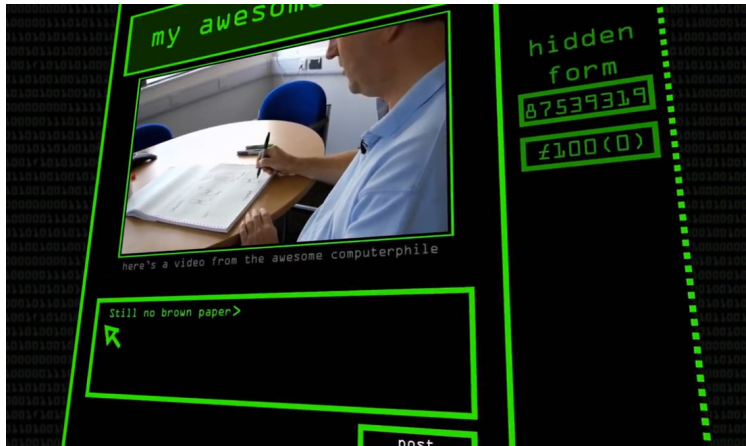
XSS (Stored/persistent)



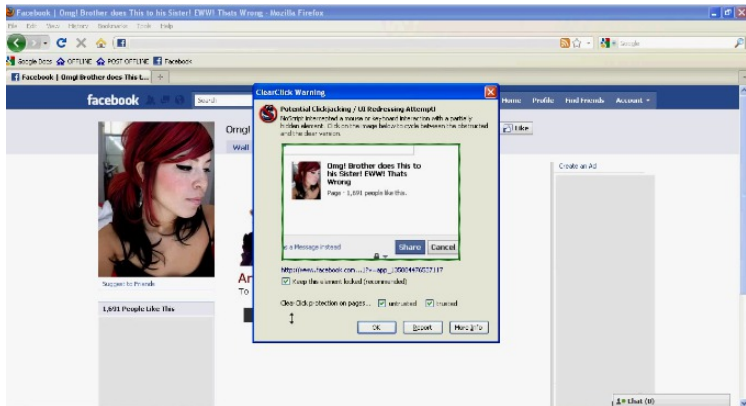
XSS (Reflected/non-persistent)



CSRF - Cross-site request forgery



Clickjacking



What can be done?

Kristy explains more about SQL and XSS injections.



Control over outputs

Why is it needed? In short, persistent XSS

- Consider the situation where someone has **uploaded** some HTML:
 - You may want to **display** this HTML with **rich text** (i.e. formatting).
 - However, they may have included some **malicious** (javascript or other) **code** in their HTML
- Your task then is to take the HTML provided, and output HTML without extra stuff. You are to **sanitize** the HTML.
- To do this effectively, no real shortcuts:
 - it is best to **use a library** intended for this purpose.
 - For PHP for example: <http://htmlpurifier.org/>

PHP: control over outputs

Arbitrary html from users could be badhtml!

```
<?php
...
... get html from storage (possibly $badhtml)
...
... print out $badhtml ...
...
?>
```

Turn badhtml into goodhtml!

```
<?php
...
require_once '/path/to/library/HTMLPurifier.auto.php';
require_once '/path/to/library/HTMLPurifier.func.php';
$goodhtml = HTMLPurifier($badhtml);
...
... print out $goodhtml...
...
?>
```

CSRF - tokens/challenges for forms

Generate one-use tokens...

```
<?php
# ... generating a token with reasonable entropy...
session_start();
if (empty($_SESSION['token'])) {
    $_SESSION['token'] = bin2hex(random_bytes(32));
}
$token = $_SESSION['token'];
?>
```

And in the form:

```
<form>
//...
<input type="hidden" name="token" value="<?php echo $token; ?>" />
//...
</form>
```


Non-cryptographic hashes are checksums...

Protecting data (I and A)...

A **hash** function maps a **long** message, or a large amount of data, to a (**shorter**) check message of some sort. We attach the **hashed** value to the original message, as a **check**.

However, here we consider the use of efficient functions that are not cryptographically secure. These functions are commonly called checksums, and they provide protection against noise.

Checking for errors...

Transmit data:

1	65	3	22	47	2
---	----	---	----	----	---

Transmit data+checksum:

1	65	3	22	47	2	140
---	----	---	----	----	---	-----

Cryptographic hashes... hash \neq encrypt



Maps data to a value, the reverse may be difficult...

```
hugh@comp:~[508]$ md5sum ss.c
550114bc3cc3359e55ba33abe8983a85  ss.c
hugh@comp:~[511]$ md5sum TXT/cybercom.txt
9ec4c12949a4f31474f299058ce2b22a  TXT/cybercom.txt
```

MD5 and SHA

MD5...

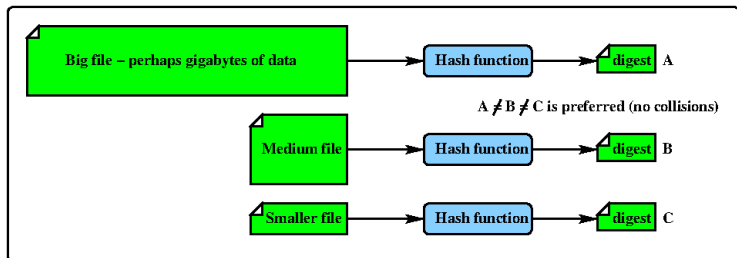
Was designed in 1991 by Ron Rivest (of RSA fame), as a cryptographic hash function. However it has been found to **not** be *collision-resistant* (defined later), and so, despite extensive use in industry today, it should no longer be used as a cryptographic hash.

The SHA family...

Originally designed by NIST in 1993 (SHA-0) but revised in 1995 as SHA-1. Revisions in 2002 led to SHA-2 256/384/512: higher security levels, and with no known weaknesses. SHA-3 was released in 2012 - it is the newest one, with some different design elements.

	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Digest (hash) size (bits)	160	224	256	384	512
Message size	$< 2^{64}$	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
Block size (bits)	512	512	512	1024	1024
Number of steps	80	64	64	80	80

Cryptographic hashes



A *hash* is not just a checksum

A cryptographic hash meets extra security requirements. In particular, we want them to be collision resistant (defined on next slide).

A cryptographic hash is an important crypto primitive. It is used in digital signatures : hash-and-sign, protecting password files (salted hashes), Commitment schemes, Proof-of-Work in Bitcoin, and so on.

Properties of cryptographic hash functions

Desirable properties: It should be efficient, public, and...

- ① It should be computationally infeasible to find two items x, x' where $x \neq x'$, but both map to $y = \mathcal{H}(x) = \mathcal{H}(x')$ (collision-resistant).
- ② Given x , it should be computationally infeasible to find an $x' \neq x$ such that $\mathcal{H}(x') = \mathcal{H}(x)$ (target-collision, or second pre-image resistant).
- ③ Given y , it should be computationally infeasible to find item x mapping to it: $y = \mathcal{H}(x)$ (one-way property, or pre-image resistant)

Note that (1) implies (2), and also a (2) attack implies a (1) attack.

When two messages map to same hash...

- ① Can there be no collisions at all? If the number of messages $\#m$ is greater than the number of hashes $\#\mathcal{H}(\dots)$, then consider the pigeonhole principle - if there are n roosts for $n+1$ pigeons...
- ② How likely are collisions? Consider the birthday paradox^a. What is the probability that at least two of N randomly selected people have the same birthday? It is much more likely than you would suspect...

^aBTW - It is not really a paradox, just counter-intuitive.

Reversing a hash of a password - by brute force?

To try checking the hashes for all the 9 character passwords, you might have to try (say) $78^9 = 106,868,920,913,284,608$ hashes.

That would take a long time. At 1uS for trying each hash, this would take over 3000 years to calculate.

Hugh will look like this when he reverses your password:



Password	(MD5) Hash
aaaaaaaaa	3dbe00a1676..
aaaaaaaaab	2125ea8b81b..
aaaaaaaaac	ea67f32d4e6..
aaaaaaaaad	746a8ab05d6..
aaaaaaaae	c554d695eb0..
aaaaaaaaf	09eb61fd25b..
aaaaaaaag	68b5af18408..
...	...

Reversing a hash of a password - by a dictionary?

A precomputed table for 9 character passwords, might have (say) $78^9 = 106,868,920,913,284,608$ entries, each containing a 16 byte value.

Thats a big disk (about 1,500,000 TB)! My 2TB disk cost me about \$100/TB, so I would need \$150,000,000 to buy this disk. Who has this sort of money? (... well ... actually ...)

Indexing by hash is even worse. We do not really have names for disks that big.

Password	(MD5) Hash
aaaaaaaaa	3dbe00a1676..
aaaaaaaaab	2125ea8b81b..
aaaaaaaaac	ea67f32d4e6..
aaaaaaaaad	746a8ab05d6..
aaaaaaaae	c554d695eb0..
aaaaaaaaf	09eb61fd25b..
aaaaaaaag	68b5af18408..
...	...

Time-memory tradeoff: chains

Idea #1: define a reduce function $R()$

The reduce function $R(y)$ maps a digest y to a word w in the dictionary D .

For illustration, if D consists of all 48-bit messages, and each digest is (say) 320-bits, then a possible reduce function simply keeps the first 48-bits of the digest. This is then used to generate the next word in D . Let's look at this chain, starting from a randomly chosen word w_0 in D :

$$w_0 \xrightarrow{\mathcal{H}} y_0 \xrightarrow{R} w_1 \xrightarrow{\mathcal{H}} y_1 \xrightarrow{R} w_2 \xrightarrow{\mathcal{H}} y_2$$

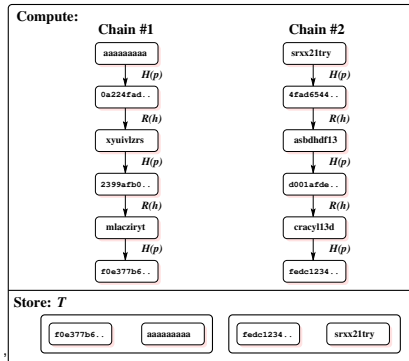
w_0 is the starting-point, and y_2 the ending-point. The pair $\langle w_0, y_2 \rangle$ is stored in the data-structure T , with other chains from different starting points.

- 1 If y is in T (an ending-point; in the above $y = y_2$), then the corresponding starting point w_0 is traced to find the preimage of y_2 . We discover w_2 .
- 2 Otherwise, from y we recalculate a chain until we reach an endpoint. In the above if $y = y_1$, then the chain is re-computed until y_2 , and then the corresponding starting point w_0 is traced to find the preimage of y_1 .

Using these tables for password reversing...

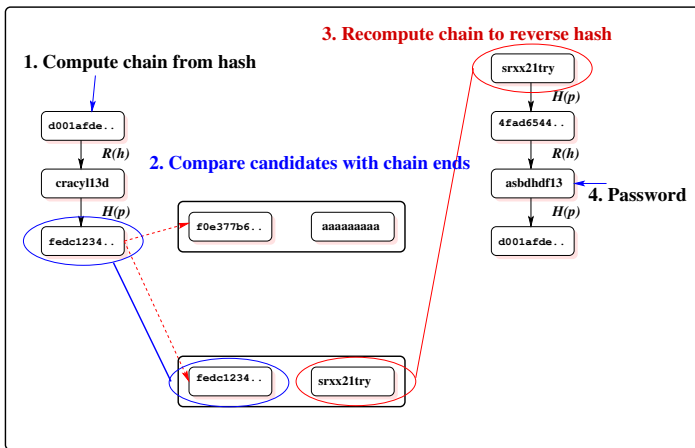
Precompute chains of values starting from a password guess, and using alternately, a hash function $\mathcal{H}(p)$, and a reducing function $R(h)$, which generates a predictable plausible guess from the hash.

Only store the first and last entries from the chain. It is space efficient, and you can recompute the intermediate values (a space-time tradeoff).



Reversing the hash

Compute, compare with "last" values, recompute...



Analysis of time and space required

(Compared with the full table)

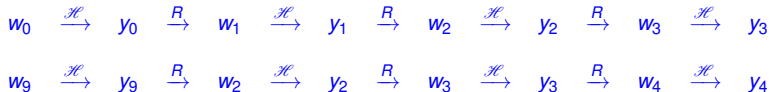
Space: If a chain had n pairs in it, then we only need to store only 1 pair, the beginning and the end pair. if (for example) we had n such chains, we would reduce the space from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.

Time/query: The number of hash and reduce operations now needed is now also $\mathcal{O}(n)$.

Accuracy: The chains contain repetitions. (why?)

In general, we can choose the length of the chain so that the reduction of space is a factor of k , with the penalty of increase the search time by a factor of k , where k is a parameter. In our 48-bits example, if we choose $k = 2^{16}$, and the number of entries in the table to 2^{32} , then the total number of entries covered in the virtual table is 2^{48} (as we shall see later, these 2^{48} entries are not unique).

Collisions due to the reduce function



Collisions of the reduce function may happen frequently

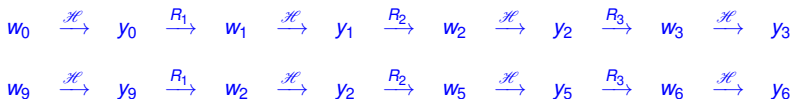
The pairs $\langle w_0, y_3 \rangle$ and $\langle w_9, y_4 \rangle$ will be stored. This causes two issues:

- 1 Efficiency in storage: Part of the chain is duplicated. The words w_2 and w_3 appear in the table twice.
- 2 Efficiency in search: Lead to searches in the wrong chains, before hitting the right chain. For the query y_9 , the lookup process would transverse both chains.

Solution to the problem: the “Rainbow” table

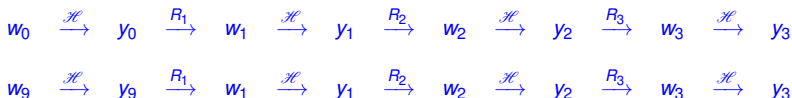
Suppose there are k reduce operations and k hash operations in the chain, instead of using the same function $R()$ in all the reduce operations, the Rainbow table uses k different functions. The first reduce is $R_1()$, followed by $R_2()$ and so on.

If collisions happen



If w_2 appears twice as above

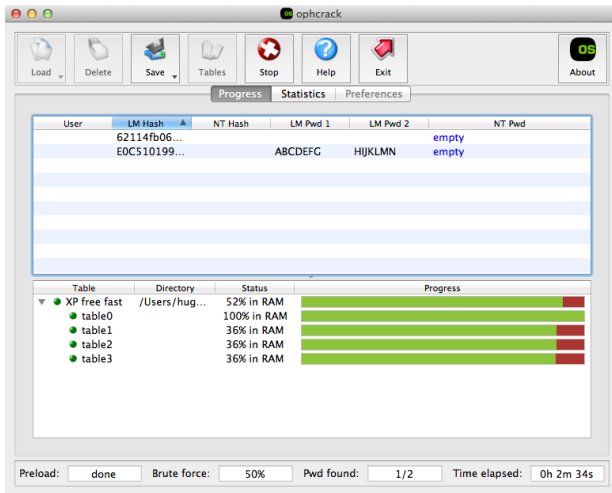
Fortunately, both $\langle w_0, y_3 \rangle$ and $\langle w_9, y_6 \rangle$ will be stored in the data-structure. Now, for the query y_9 , its preimage w_9 can be found.



Or if collision was on the same R_x ? There will be two y_3 !

Both chains may have to be traversed. Rainbow tables lower the number of duplications, and increase the number of words that can be found.

Reversing an old Windows password hash



Not only hashes!

This technique can be applied to a known plaintext attack on an encryption scheme where the keyspace is small. (how?)

The Birthday “Paradox” explained

It is NOT the likelihood that someone in a room full of people shares a specific person’s birthday...

It is the likelihood that amongst every pair of candidates in a room there will be (at least) one matching pair. It turns out that it is easiest to calculate the likelihood that the people will not share birthdays^a:

- If $N = 2$, then the likelihood the two do not share a birthday is $\frac{364}{365}$, because the first person can have any of the 365 days, leaving 364 days available for the second person.
- If $N = 3$, then the likelihood all three do not share is $\frac{364}{365} \times \frac{363}{365}$.
- For N , the likelihood is $\frac{364}{365} \times \frac{363}{365} \times \dots \times \frac{365-(N-1)}{365}$.

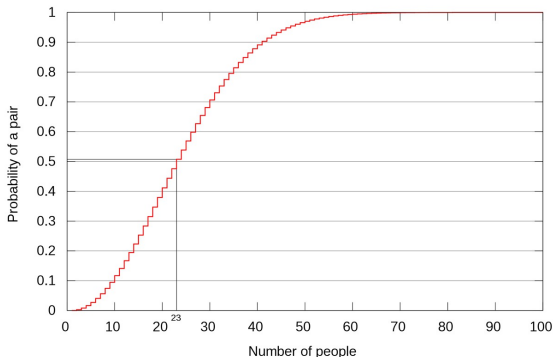
For $N = 23$, this likelihood is about 0.5.

We can attack some *digital signatures* using an attack based on the birthday paradox.

^aNote that the likelihood that the people *will* share birthdays is 1 (one) minus the likelihood that the people will *not* share birthdays.

The Birthday “Paradox”

Likelihood versus number of random choices



In general, in a birthday attack over n items, to achieve a probability of p , we should choose about $\sqrt{2n \ln \frac{1}{1-p}}$ items. In the above, for $n = 365, p = 0.5$ we have about $1.177\sqrt{365} \approx 22.5$. This value (1.177) gives rise to the common approximation we use: *take the square root*^a.

^aFor n -bit attack, only need $2^{\frac{n}{2}}$ values to get 50% chance of success.

The birthday attack - straightforward method

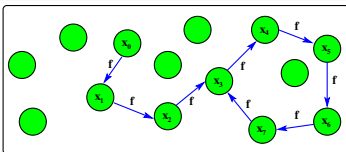
Find x, x' s.t. $\mathcal{H}(x) = \mathcal{H}(x')$; time $\mathcal{O}(2^n)$, memory $\mathcal{O}(2^n)$...

The birthday attack can be used to speedup the search for a hash collision with a “square-root” reduction.

Suppose a digest is n bits, and we have a random set of around $2^{\frac{n}{2}}$ messages (by the birthday attack, the chance of having a collision is around 0.5 probability). A straightforward implementation of a birthday attack stores this set in the memory and then searches for a collision. However, in practice, the memory requirement is more difficult to meet compared to the running time requirement. Imagine if $n = 80$ bits, then there are 2^{40} pairs of messages and digests. Assuming each pair take up 32 bytes, then the total storage is 2^{45} bytes, i.e. 32TB. Nevertheless, 2^{40} hash operations can be done in reasonable time.

The following *improved* birthday attack takes about twice the time, compared to the straightforward implementation, but uses constant memory.

Floyd - for chains like $x_0 \rightarrow f(x_0) \rightarrow f(f(x_0)) \rightarrow \dots$



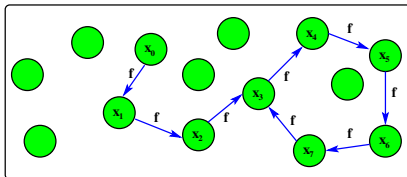
Find loop with $f(x) \stackrel{\text{def}}{=} \mathcal{H}(x)$; time $\mathcal{O}(2^n)$, and memory $\mathcal{O}(1)$

(Robert W.) Floyd's cycle-detection algorithm: the tortoise t and the hare h . Variables hop through a graph at different rates, stopping when they collide.

Algorithm 5.9: part 1 finds the loop in a (digest/hash) chain

```
def H(x):  
    return md5(x.encode('utf-8')).hexdigest()  
  
def f(x):  
    return H(x)[:10]  
  
def floyd(x0):  
    t = f(x0)  
    h = f(f(x0))  
    print('Find the loop')  
    while (t != h):  
        t = f(t)  
        h = f(f(h))
```

Floyd - for chains like $x_0 \rightarrow f(x_0) \rightarrow f(f(x_0)) \rightarrow \dots$

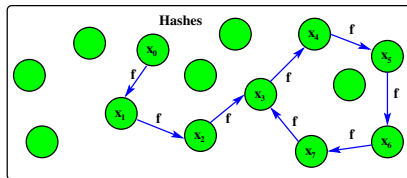


Algorithm 5.9: part 2 finds the collision

```
t = x0
print('Find the collision')
while (t != h):
    if (f(t) == f(h)):
        break
    t = f(t)
    h = f(h)
    if (t != h):
        temp_t = t
        temp_h = h
        pass
return temp_t, temp_h
```

This reruns the *tortoise* and the *hare*, but both just doing one step at a time. The *tortoise* restarts at the beginning, and the *hare* at the result of part 1. They meet at the beginning of the loop (where the collision is).

Finding a collision with floyd.py

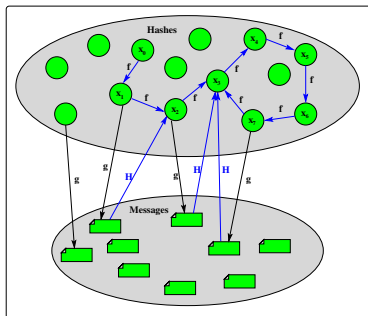


Run it with a starting point...

```
hugh@comp % python3 floyd.py 42
Find the loop
Find the collision
Tortoise f375526901 > cfd1d92d5b1d0fcae405b61165d261ed
Hare      0ecccc477 > cfd1d92d5bf0b71d1655bc12981bd608
hugh@comp % md5 -s f375526901
MD5 ("f375526901") = cfd1d92d5b1d0fcae405b61165d261ed
hugh@comp % md5 -s 0ecccc477
MD5 ("0ecccc477") = cfd1d92d5bf0b71d1655bc12981bd608
hugh@comp %
```

Note that we are only matching the first 10 hex digits of the digest.

Practical use in a message/digest collision attack

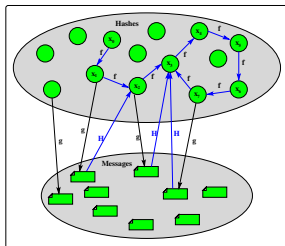


Apply Floyd by using $f(x) \stackrel{\text{def}}{=} \mathcal{H}(g(x))$

Consider messages $m \in M$, digests $h \in H$, the functions $g : M \rightarrow H$, and $f : H \rightarrow H$ defined as $f(x) \stackrel{\text{def}}{=} \mathcal{H}(g(x))$. The elements in the *chain/trail* of digests are mapped to the messages. Here is an example with *good*, and *bad* messages about Bob:

- $g(0000)$ = Bob is a good and honest worker.
- $g(0001)$ = Bob is a difficult and....

Code changes for the collision attack



Algorithm 5.9 for 2^{16} messages in G . Few code changes...

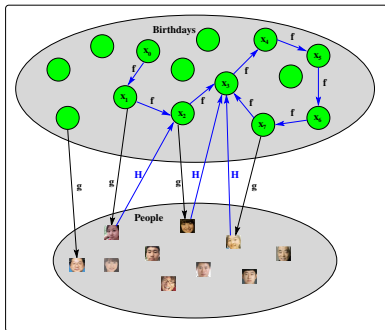
```
G = [m1,m2,m3...]
def g(x):
    return G[int(x[:4],base=16)]
def f(x):
    return H(g(x))[:4]
```

This code only matches the first 4 hex digits of the digest:

```
hugh@comp % python3 floydfile4.py 423485
Tortoise Lord Baltinglas... > e09c3f2d63f7841d33fdbe0bfb4fd444
Hare      Tell me more ab... > e09c59046d3b1201931ff649d8b287e8
```

Birthdays and hashes? All are $f(x) \stackrel{\text{def}}{=} \mathcal{H}(g(x))$

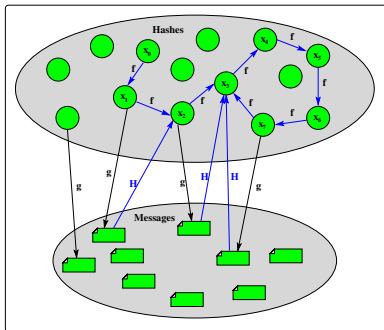
People sharing birthdays



B is the set of birthdays, P the set of people. and we have:

- $g: B \rightarrow P$ (random map)
- $H: P \rightarrow B$ (birthday)
- $f: B \rightarrow B$

Message digests colliding



H is the set of digests/hashes, M the set of message, and we have:

- $g: H \rightarrow M$ (random map)
- $H: M \rightarrow H$ (here MD5)
- $f: H \rightarrow H$