Name: Jun Wang 000791814

ITSC-302

Lab 3: Attacking Authentication

Web Application Security

Marks: \_\_\_\_\_

Bonus: \_\_\_\_\_

Table of Contents

[Lab Outcomes 2](#_Toc20678352)

[Background Reading 2](#_Toc20678353)

[Architecture Diagram 2](#_Toc20678354)

[Required Hardware/Software 2](#_Toc20678355)

[Introduction 3](#_Toc20678356)

[1.0 Sniffing Login 3](#_Toc20678357)

[2.0 Improper Error Handling 5](#_Toc20678358)

[3.0 Password Strength 6](#_Toc20678359)

[4.0 Forgot Password 7](#_Toc20678360)

[5.0 Multi-level Logins 8](#_Toc20678361)

[6.0 Authentication Techniques and Attack Tools 12](#_Toc20678362)

[7.0 Sign-Off – Lab 3: Attacking Authentication 15](#_Toc20678363)

Web Application Security

Lab 3: Attacking Authentication

# Lab Outcomes

* Exploit the login process.
* Exploit passwords and related functionalities.

Background Reading

Read the textbook sections listed in the Course Schedule.

Architecture Diagram

|  |
| --- |
|  |

Required Hardware/Software

* VM Ubuntu 18.04 – 2 CPU, 4GB Ram, 20 GB hard disk
  + Docker
    - WebGoat v7.1
    - DVWA (Dawn Vulnerable Web App)
* VM Kali
  + Burp or other Web proxy (scanner)

# Introduction

Authentication is essential to an application’s protection. It is the first line of defence against attackers seeking to gain unauthorized access and full control of a system.

# 1.0 Sniffing Login

In WebGoat, complete the **Insecure Communication > Insecure Login** lesson.

Intercept on zap, capture the form content, password=sniffy.

|  |
| --- |
|  |
|  |
|  |

# 2.0 Improper Error Handling

In WebGoat, complete the **Improper Error Handling > Fail Open Authentication Scheme** lesson.

|  |
| --- |
|  |
|  |
|  |

# 3.0 Password Strength

In WebGoat, complete the **Authentication Flaws > Password Strength** lesson.

|  |
| --- |
| Insert evidence here |

# 4.0 Forgot Password

In WebGoat, complete the **Authentication Flaws > Forgot Password** lesson.

The problem is users can try as many time as they want. There is no restriction of the try times on username or secret question.

|  |
| --- |
| Insert evidence here |
|  |

# 5.0 Multi-level Logins

1. In WebGoat, complete the **Authentication Flaws > Multi Level Login 1** lesson.

|  |
| --- |
| Insert evidence here |
|  |
|  |

1. In WebGoat, complete the **Authentication Flaws > Multi Level Login 2** lesson.

|  |
| --- |
|  |
|  |
|  |

# 6.0 Authentication Techniques and Attack Tools

1. Research and describe the tools and techniques used to attack HTTP Forms-based authentication (use screen shots and verbal description)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Form Based Authentication uses a form (usually in html) with input tags to allow users to enter their username and password. Once the user submits the information, it is passed over through either GET or POST methods via HTTP or HTTPs to the server. On the server side if the credentials are found to be correct, then the user is authenticated and some random token value or session id is given to the user for subsequent requests. One of the good features of Form Based authentication is that there is no standardized way of encoding or encrypting the user name/password, and hence it is highly customizable, which makes it immune to the common attacks which were successful against HTML Basic and Digest Authentication mechanisms. Form Based Authentication is by far the most popular authentication method used in Web applications. Some of the issues with Form Based Authentication is that credentials are passed over in plaintext unless steps such as employment of TLS (Transport Layer Security) are not taken.

\_\_\_\_\_\_we can carry out a brute force attack in Burpsuite.

|  |
| --- |
|  |

1. Research and describe the tools and techniques used to attack HTTP basic authentication (use screen shots and verbal description)

HTTP-Basic authentication uses a combination of a username and password to authenticate the user. Basic authentication sends a Base64-encoded string that contains a user name and password for the client. Base64 is not a form of encryption and should be considered the same as sending the user name and password in clear text. The process starts when a user sends a GET request for a resource without providing any authentication credentials.

The server responds back with a “Authorization Required” message in its header. We can see the packet in Wireshark. As we can see from the header, the authentication is of the type “Basic”. The browser is quick to recognize this and displays a popup to the user requesting for a Username and a Password. Note that the popup is displayed by the browser and not the web application. The text after Basic holds the key. These are basically the credentials in encoded form.The username and password are concatenated with a colon (:) in between and the whole thing is then encoded using the Base64 algorithm. For example, if the username is “infosec” and the password is “infosecinstitute” then the whole thing “infosec:infosecinstitute” is encoded using the Base 64 algorithm. The server then gets the header value, decodes it to get the credentials and grants access to the user if the credentials are correct. The point to note here is that it is very trivial to decode the encoded string to obtain the credentials, hence it is widely vulnerable to eavesdropping attacks.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_We can attack http basic authentication by wireshark.\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
|  |

1. Research and describe the tools and techniques used to attack HTTP digest authentication (use screen shots and verbal description)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Digest Authentication was designed as an improvement over the HTTP Basic Authentication. One of the major improvements is that the data is not passed over in cleartext but in encrypted format. The user first makes a request to the page without any credentials. The server replies back with a WWW-Authenticate header indicating that credentials are required to access the resource. The server also sends back a random value which is usually called a “nonce”. The browser then uses a cryptographic function to create a message digest of the username, password, nonce, the HTTP methods, and the URL of the page. The cryptographic function used in this case is a one way function, meaning that the message digest can be created in one direction but cannot be reversed back to reveal the values that created it. By default, Digest authentication uses MD5 cryptographic hashing algorithm.\_Digest Access authentication is less vulnerable to Eavesdropping attacks than Basic Authentication, but is still vulnerable to replay attacks, i.e., if a client can replay the message digest created by the encryption, the server will allow access to the client. However, to thwart this kind of attack, server nonce sometimes also contains timestamps. Once the server gets back the nonce, it checks its attributes and if the time duration is exceeded, it may reject the request from the client. One of the other good things about Digest access authentication is that the attacker will have to know all the other 4 values (username, nonce, url, http method) in order to carry out a Dictionary or a Brute force attack. This process is more computationally expensive than simple brute force attacks and also has a larger keyspace which makes brute force attack less likely to succeed.

The following is an example of cracking HTTP digest authentication using wireshark and a brute force python script.

1> capture the GET request from the client side on wireshark. In the TCP header, we can find valuable information including: the nonce value, algorithm, realm, nc, and hash value(response).

2>Write a python script to calculate hash values for every combination of all information we collected from the TCP header from wireshark, with a dictionary for usernames and a dictionary for passwords. Compare the hash (md5) values generated with the response value, if find a match, then output the username and password.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
|  |

1. Research and describe the tools and techniques used to attack HTTP windows-integrated authentication using NTLM (use screen shots and verbal description)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

NTLM is a challenge-response mechanism and uses a version of the

Windows NTLM protocol. Integrated Windows authentication enables users to log in with their Windows credentials, using Kerberos or NTLM. The client sends credentials in the Authorization header. Windows authentication is best suited for an intranet environment.

Advantages: Disadvantages:

|  |  |
| --- | --- |
| - Built into IIS. - Does not send the user credentials in the request. - If the client computer belongs to the domain (for example, intranet application), the user does not need to enter credentials. | - Not recommended for Internet applications. - Requires Kerberos or NTLM support in the client. - Client must be in the Active Directory domain. |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
|  |

<https://labs.f-secure.com/blog/pth-attacks-against-ntlm-authenticated-web-applications/>

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

1. Research and describe the tools and techniques used to attack HTTP windows-integrated authentication using kerberos (use screen shots and verbal description)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Kerberos is a distributed authentication service that allows a process (a client) running on behalf of a principal (a user) to prove its identity to a verifier (an application server, or just server) without sending data across the network that might allow an attacker or the verifier to subsequently impersonate the principal. Kerberos optionally provides integrity and confidentiality for data sent between the client and server. Kerberos is not effective against password guessing attacks; if a user chooses a poor password, then an attacker guessing that password can impersonate the user. Similarly, Kerberos requires a trusted path through which passwords are entered. If the user enters a password to a program that has already been modified by an attacker (a Trojan horse), or if the path between the user and the initial authentication program can be monitored, then an attacker may obtain sufficient information to impersonate the user.

Wagging the Dog: Abusing Resource-Based Constrained Delegation to Attack Active Directory <https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html>

Constructing Kerberos Attacks with Delegation Primitives

https://shenaniganslabs.io/media/Constructing%20Kerberos%20Attacks%20with%20Delegation%20Primitives.pdf

|  |
| --- |
|  |
|  |
|  |

1. Research and describe the tools and techniques used to attack HTTP of a multi factor authentication SMS (use screen shots and verbal description)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Multi-factor authentication** (**MFA**) is an [authentication](https://en.wikipedia.org/wiki/Authentication) method in which a [computer user](https://en.wikipedia.org/wiki/Computer_user) is granted access only after successfully presenting two or more pieces of evidence (or factors) to an [authentication](https://en.wikipedia.org/wiki/Authentication) mechanism: knowledge (something the user and only the user knows), possession (something the user and only the user has), and inherence (something the user and only the user is).

Many, especially early, MFA methods included sending additional authentication code via a user’s cell phone short message service (SMS)

A SIM swapping attack can steal/transfer the user’s cell phone operations to another phone, allowing the attacker to get the sent the SMS code

NIST (in SP 800-63) does not accept SMS codes as valid authentication because of how easy it is to hack

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.Mobile Number Transfer: The first rash of 2FA bypasses occurred in countries where phone number porting (moving a number from service to service) was relatively easy. Australia was a prime early hunting ground for attackers who, after collecting the credentials of a target, were able to research the victim’s phone number, too. A quick call to the mobile carrier could get the phone number assigned to a phone that the attacker controlled. From then on, all 2FA codes were intercepted by the attackers, and the victims often had no idea—they woke up the next day and their phones didn’t work. It could take them a week to get their number back, and only then did they realize their bank accounts had been drained.

2. Interception at Mobile Operator: Here’s a novel one that’s seen a lot of use in the last year. Attackers get access to 2FA codes through the mobile operator’s customer portal. Where a lazy person reuses the same password for their email and mobile accounts, all the attacker needs to intercept the 2FA code is to log into the user’s mobile account and see the code among the stored text messages. From there they can reset the bank password (if they didn’t already have it), and theft ahoy.

3. Malware Intercept: Since at least 2014, custom malware has infected mobile phones and intercepted the SMS-based 2FA codes as they arrived. Sometimes this malware was part of a banking trojan package. Other times, the malware would just forward the 2FA codes to the attacker, and voila, game over. This problem was particularly widespread in the Android ecosystem, but rarely, if ever, seen with Apple.

4. Lost Phone Reset Password Bypass. People lose phones and change phone numbers. It happens, like diabetes. So all services that use SMS-based authentication systems must have recovery services where people can reset their account or update their phone number. If the attacker has already compromised the email account (perhaps because of re-used passwords), they can reset, update, or otherwise bypass the 2FA system. Lost phone and password-reset pages are the most common targets of unwanted automation today. Don’t believe me? Check your access log for lost-password.html.

5. Social Engineering. Attackers targeting a specific organization or person have been known to use social engineering to bypass 2FA. For example, the attacker calls you, the victim, on your phone, and claims to be a representative of your bank. He says they are checking accounts for fraud, and he’s going to send you a code to verify your identity. He asks you to read it back to him, then logs into a site with your credentials while you wait. The 2FA code is sent to you, and you give it to him over the phone. He thanks for you for your help, then robs you of your money and residual dignity.

6. Man-in-the-Middle Website Proxies—Modlishka. A group of researchers created the Modlishka phishing proxy framework [[github link](https://github.com/drk1wi/Modlishka)] to show how easy it is to trick a user into entering their SMS 2FA code. If you haven’t seen the [video](https://vimeo.com/308709275), it’s a total forehead slapper for the security community. How did we not see this coming?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
|  |

1. Research and describe the tools and techniques used to attack HTTP of a multi factor authentication Email (use screen shots and verbal description)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Hackers beat 2-factor protection with automated phishing attacks

The goal behind the attacks has been to trick victims into handing over access to their Google and Yahoo accounts, even when two-factor authentication is in place. "What makes these campaigns especially troubling is the lengths to which they go to subvert the digital security strategies of their targets," Amnesty International said in its report.

For the uninitiated, two factor-authentication is a safeguard designed to protect your online account in the event your [password](https://www.pcmag.com/article2/0,2817,2407168,00.asp) is stolen. It works like this: when you try to access the account, you not only have to enter your login credentials, but also a special one-time passcode that's been generated over your phone.

Unfortunately, the special passcodes generated by two-factor authentication systems are usually just a string of a random numbers, which can make them easy to [phish](https://www.pcmag.com/news/362871/reddit-hacked-despite-sms-two-factor-authentication); all the hacker has to do is to trick you into giving up the special codes.

Amnesty International said the group of hackers they've been tracking pulls this off by sending out fake but convincing security alerts that look like they came from Google or Yahoo. The alerts will claim the victim's account may have been breached and provide a link to an official-looking login page to initiate a password reset.

"To most users a prompt from Google to change passwords would seem a legitimate reason to be contacted by the company, which in fact it is," Amnesty International said. But in reality, the login pages are fake.

The hackers created the phony process to both phish the victim's password and the special two-factor authentication code. Amnesty International has been investigating the scheme based on suspicious emails the group has been receiving from human rights activists and journalists. To test out the attacks, the group created a disposable Google account and then clicked through one of the phishing emails.

"Sure enough, our configured phone number did receive an SMS message containing a valid Google verification code," Amnesty said in its report.

The group also investigated how the hackers were creating their phishing schemes and noticed that the mysterious group accidentally made public an online directory they were using to host their attacks. The information revealed the hackers were using web application testing tools to automate the phishing process.

"Essentially, they built an 'auto-pilot' system that would launch Chrome and use it [to] automatically submit the login details phished from the user to the targeted service, including two-step verification codes sent for example via SMS," said Claudio Guarnieri, a technologist at Amnesty, in a [tweet](https://twitter.com/botherder/status/1075395568652828672).

The hackers' automated process is important because it lets them input the special one-time passcode into the real Google or Yahoo login page, before the time limit on the passcode runs out.

Typically those concerned about getting 2FA codes via SMS can also do so via an authenticator app, which serves up codes that change every few seconds. Amnesty did not immediately respond to PCMag's request for comment about whether this affects such apps, but a technologist there [told Motherboard](https://motherboard.vice.com/en_us/article/bje3kw/how-hackers-bypass-gmail-two-factor-authentication-2fa-yahoo) that "the same approach could potentially be used to phish codes from a 2FA app such as Google Authenticator."

The human rights group still recommends people adopt two-factor authentication, but to be aware that the system does have limitations. So don't be fooled into thinking you're completely safe. For example, government-sponsored hackers have the resources to create elaborate phishing schemes to crack the safeguard. They can also attempt to infect your PC with [malware](https://www.pcmag.com/roundup/354226/the-best-malware-removal-and-protection-tools).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
|  |

1. Research and describe the tools and techniques used to attack HTTP of a multi factor authentication token (use screen shots and verbal description)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Most MFA code-generating tokens start with a (randomly) generated (permanently) stored “seed” or “shared secret” value, which is then incremented by some sort of counter/algorithm which generates all subsequent values.

• Known as one-time passwords (OTP)

• “Will never be repeated again”

• Unique user/device identifier usually involved

• May also use current time/date to “randomly” generated code good

only for a particular time interval

Known as time-based one-time passwords (TOTP)

Shared secret will always be present in at least two places (e.g.

source database/verifier and device itself)

Attackers that learn seed/shared secret and algorithm can

generate duplicate/identical code generators that match the

victim’s code generator

Any system which allows users to authenticate via an untrusted network (such as the Internet) is vulnerable to man-in-the-middle attacks. In this type of attack, a fraudster acts as the "go-between" of the user and the legitimate system, soliciting the token output from the legitimate user and then supplying it to the authentication system themselves. Since the token value is mathematically correct, the authentication succeeds and the fraudster is granted access.

This is the most common failure of tokens, often allowing attackers to hijack another user's session by manipulating a token. Most token schemes have no binding to a particular user, relying on the assumption that possession of the token is proof of ownership.

#### Does the Server Transmit the Token to the Client in a Secure Manner?

Essentially, this means using SSL or IPSec to encrypt the traffic when sending the token. If the traffic is not encrypted, the token could be visible to others. When you use a cookie, it is important to mark it as secure so that the client's browser handles it properly.

#### Does the Application Use a Sufficiently Large Keyspace?

To prevent brute-force attacks on the session token, you should use a large enough token to ensure that an attacker cannot easily come across a valid token. For example, a 128-bit number allows for 340,282,366,920,938,463,463,374,607,431,768,211,456 (or 2128) possible unique session tokens.

### Example: Hacking the iButton Authentication Token

The Dallas Semiconductor DS1991 MultiKey iButton (www.ibutton.com) is a hardware [authentication](https://www.sciencedirect.com/topics/computer-science/authentication) token that has three internal secure data areas, each protected by a distinct password. Depending on the application, the iButton can be used for cashless transactions, [user authentication](https://www.sciencedirect.com/topics/computer-science/user-authentication), or access control; and the secure data could include financial information, monetary units, or user registration/identification information.

The goal of this example is to attempt to recover either the passwords or the secure data within the device without having legitimate credentials. By communicating with the device via a PC serial port and using some basic [cryptanalysis](https://www.sciencedirect.com/topics/computer-science/cryptanalysis) techniques (similar to that discussed in the “Cryptanalysis and Obfuscation Methods” section), we discover a vulnerability that potentially allows an attacker to determine the passwords used to protect these secure areas, thus gaining access to the protected data. This example is based on Kingpin's DS1991 MultiKey iButton Dictionary Attack Vulnerability advisory (www.atstake.com/research/advisories/2001/a011801-1.txt).

#### Experimenting with the Device

The DS1991 contains 1,152 bits of [non-volatile memory](https://www.sciencedirect.com/topics/computer-science/non-volatile-memory) split into three 384-bit (48-byte) containers known as subkeys. Each subkey is protected by an independent 8-byte password. Only the correct password will grant access to the data stored within a subkey area and return the data. If an incorrect password is given, the DS1991 will return 48-bytes of random data intended to prevent an attacker from comparing it against a known constant value. Dallas Semiconductor marketing literature (www.ibutton.com/software/softauth/feature.html) states that “false passwords written to the DS1991 will automatically invoke a [random number generator](https://www.sciencedirect.com/topics/computer-science/random-number-generator) (contained in the iButton) that replies with false responses. This eliminates attempts to break security by pattern association. Conventional protection devices do not support this feature.”

By using the iButton-TMEX software (www.ibutton.com/software/tmex/index.html), which includes an iButton Viewer to explore and connect to iButton devices, it was determined that the data returned on an incorrect password attempt is not random at all and is calculated based on the input password and a constant block of data stored within the DS1991 device. Figure 14.12 shows the data contents of a DS1991 device. Note the identical values returned for Subkey IDs 1 and 2 when an incorrect password of “hello” is entered.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
|  |

The returned data has no correlation to the actual valid password, which is stored in the DS1991's internal memory. The constant block of data, which is a 12k array containing 256 entries of 48-bytes each, is constant across all DS1991 devices and has no relation to the actual contents of the subkey memory areas. This means that for any given character (1 byte = 256 possibilities), there is a unique 48-byte response sent back from the iButton device. To determine what comprised that constant block, Dallas Semiconductor wrote a test program (based on the TDS1991.C sample code, ftp://ftp.dalsemi.com/pub/auto\_id/softdev/tds1991.zip) to simply set the password 256 times, ranging from 0x00 to 0xFF, and record the response. The serial port was monitored to view the responses from the iButton device. It was then a matter of puzzle-solving to determine what the responses would be for longer passwords. By pre-computing the return value expected for an incorrect password attempt, it is possible to determine if a correct password was entered. This is due to the fact that, if the password is correct, the data returned by the DS1991 will be the actual data stored in the subkey, not the “incorrect password” response.

The transaction time is limited to 0.116 seconds for each password attempt by the computational speed of the DS1991 and the bus speed of its 1-Wire interface. Because of this, it is not possible to perform an exhaustive brute-force search of the entire 64-bit password keyspace, or that of only ASCII-printable characters (which would require approximately 22,406,645 years). However, it is still possible to perform a dictionary attack against the device using a list of commonly used passwords.

#### Reverse-engineering the “Random” Response

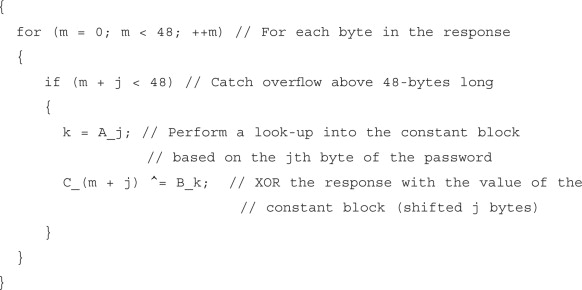
By comparing the 48-byte “random” device responses of various known incorrect passwords, it was determined that they were computed in a simple loop, as shown below. Although the code may appear complex, we are essentially just XORing a number of [constant strings](https://www.sciencedirect.com/topics/computer-science/constant-string) together.

Let A\_j be the jth byte of A, the 8-byte password (padded with 0x20 if less than 8-bytes)

Let B\_k be the kth entry of B, the 12kB constant block (256 entries each 48-bytes in length)

Let C\_m be the mth byte of C, the 48-byte response (initialized to 0x00)

for (j = 0; j < 8; ++j) // For each remaining character in p/w.



There is an additional step taken if the last character of the password (A\_7) is signed (greater than 0x7F). If this is the case, the pre-computed subkey value is XORed against another constant block containing 128 entries of 48-bytes each. It is unclear why iButton performs this step, but it is possibly to add an additional level of obscurity to the “random” response.

As shown in the code above, the constant block is used to retrieve a 48-byte string for each byte of the entered password. Each string is XORed together to produce the final response that the iButton device returns if the password is incorrect. For the example shown below, let's use a password of “hello” (padded up to 8 characters with 0x20, which is a blank space) and compute the 48-byte “incorrect password” string. In the interest of space, we will only look at the first 16-bytes of the resultant 48-byte response.

Let A = “hello” = 68 65 6C 6C 6F 20 20 20

B\_68 (‘h’) = D8 F6 57 6C AD DD CF 47 CC 05 0B 5B 9C FC 37 93 …

B\_65 (‘e’) = 03 08 DD C1 18 26 36 CF 75 65 6A D0 0F 03 51 81 …

B\_6C (‘l’) = A4 33 51 D2 20 55 32 34 D8 BF B1 29 40 03 5C 9C …

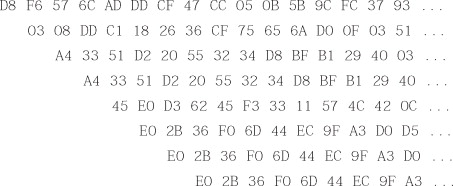
B\_6C (‘l’) = A4 33 51 D2 20 55 32 34 D8 BF B1 29 40 03 5C 9C …

B\_6F (‘o’) = 45 E0 D3 62 45 F3 33 11 57 4C 42 0C 59 03 33 98 …

B\_20 (‘’) = E0 2B 36 F0 6D 44 EC 9F A3 D0 D5 95 E3 FE 5F 7B …

B\_20 (‘’) = E0 2B 36 F0 6D 44 EC 9F A3 D0 D5 95 E3 FE 5F 7B …

B\_20 (‘’) = E0 2B 36 F0 6D 44 EC 9F A3 D0 D5 95 E3 FE 5F 7B …



The final pre-computed “random” response is calculated by XORing all of the above lines together, keeping the most significant 48 bytes. Note that this string is the [hexadecimal representation](https://www.sciencedirect.com/topics/computer-science/hexadecimal-representation) of the “garbage” in Figure 14.12 that was returned when “hello” was entered as an incorrect password:

D8 F5 FB 26 4B 46 03 9B CC 2E 68 82 22 F7 F3 2B …

The DS1991 device will return the 48-byte “incorrect password” string if the given password is incorrect (as demonstrated by our example). The pre-computed value will always be the same for any device that is given the same password. Because of this, if the pre-computed value matches the response returned from the DS1991, we know the guessed password is incorrect. If the responses are different, the guessed password is the correct password. This is because the device is returning the actual subkey data rather than the “random” data normally returned for a given incorrect password.

A proof-of-concept tool with source code (showing the 12kB constant block) is available (www.atstake.com/research/advisories/2001/ds1991.zip) to demonstrate dictionary attacks against the DS1991 iButton. The demonstration performs the following actions:

1.Finds a DS1991 iButton on the default COM port.

2.Given a dictionary/word file as input, calculates the expected 48-byte response returned on an incorrect password attempt.

3.Attempts to read subkey area #1 using a password. If correct, the protected subkey data is displayed. Otherwise, Step 2 is repeated with the next password in the word file.

# 7.0 Sign-Off – Lab 3: Attacking Authentication

Detach this page and submit it to your instructor to indicate you have completed each section.

Name:

Student ID:

|  |  |
| --- | --- |
| **Section** | **Instructor Initials** |
| 1.0 Sniffing Login |  |
| 2.0 Improper Error Handling |  |
| 3.0 Password Strength |  |
| 4.0 Forgot Password |  |
| 5.0 Multi-level Login – 1 |  |
| 5.0 Multi-level Login – 2 |  |
| 6.0 Authentication Techniques and Attack Tools |  |